Miodrag Trajanović

Osnove Nodejs-React-Webpack-Babel-Bootstrap-PHP-Mysqli

Objavljeno 2019-tog leta

Impresum

Osnove Nodejs-React-Webpack-Babel-Bootstrap-PHP-Mysqli

Deo idejnog projekta “Srbi preko”

Objavio
Miodrag Trajanović, Ćićevac

Autor
Miodrag Trajanović

Tehnička obrada i priprema za štampu
Miodrag Trajanovic

Autor korice
Miodrag Trajanović

Lektor i korektor
Biljana Stanojević

Recenzija
mr Miloš Vujić, dipl.ing,
magistar pedagoško tehničkih nauka

Fromat
B5 (18,2 X 25,7cm)

Broj stranica
477 stranica

Tiraž
100 knjiga

Štampao
Dobroljublje Beograd

Godina Objavljivanja: 2019

Objavljivanje: Prvo

Predogovor

Programiranje je kao i sve drugo jedna igra, gde ljubav i znanje rađajući razumevanje stvaraju jednostavnost, koja je samo na prvi pogled složena. Upotrebom ne samo ove knjige, već i svega što je oko vas stvorite nešto što je vaše. Ne bitno da li to nazivate procedurom, funkcijom, klasom objekata, jedino je važno da to što izrađujete napravite tako da ga možete uvek upotrebiti u svakoj prilici. Jer, svaka prepravka starog teža je no da izgradite hiljadu novih mostova, zato budite pažl jivi, kako vas vaša dela ne bi stalno opominjala a vi bili izgubljeni u vremenu i prostoru.

Knjiga prestavlja osnovu upotrebe Nodejs, Reactjs, Bootstrap, PHP i mysqli, sa dodatnim osnovama Webpack-a i Babela, čija je namena da povežu i pokrenu i prevedu ES6 java skript kod u niži oblik kako bi ga brovseri mogli razumeti i prikazati njegov rad. Uz knjigu se dobija dvd disk sa primerima, koji se vrlo jednostavno uključuju u već izrađenu aplikaciju, dok ono što je vezano za upotrebu PHP-a je odvojeno u posebnom folderu. Ono što je najbitnije kompetan softver o kome će biti reči u ovoj knjizi je besplatan, osim nekih editora koda. Možda na prvi pogled tekst svega u ovoj knjizi vam izgleda malo razbacan i ne složen po nekim pravilima, ali to je samo na prvi pogled. Kao i kodovi failova na dvd disku. Cilj je da na taj način povećam vašu pažnju na ono što čitate ili isporobavate na vašim računarima. Čime isto tako želim da podstakem da programirate i da od toga pravite potpuno slobodno svoje ideje i dela, a da ovo vam posluži kao kraće upustvo kako treba i kako ne treba da radite. Drugačiji pristup nije povoljan jer u tom drugom pristupu neće moći da razumete smisao kako šta radi i kako je moguće da to isto radi drugačije, kao i čega se trebate čuvati prilikom izrade vaših aplikacija.

U pogledu upotrebe react-a u ovoj knjizi je opisana osnovna upotreba naprednije verzije java skript koda koja ima oznaku ES6, kao i automatizovani način izrade react aplikacija, sa ciljem da vas upozna sa njim samim i nepohodnim dodacima koji omogućavaju njegov rad.

NODE.JS

Node

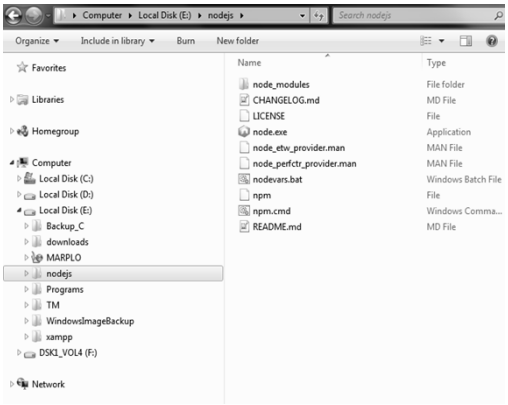
Node.js je otvoreni besplatni framework server, kog možete pokretati na različitim paltfomama (Windows, Linux, Unix, Mac OS Xi drugim.), koji omogućava pokretanje vašeg java skripta na server.
Node se upotrebljava za asihrono programiranje.
Njegovom pojavom omogućeno je da se java skript upotrebljava na serverskoj strani. Ova mogućnost je upotrebljena za dobijanje dinammičkih veb stranica. Što znači da njegov java skript kod generiše sadržaj stranica pre nego što se to i putem interneta prosledi korisniku. Tako da se tip programiranja može opisati , kao programiranje vođeno događajima. Takav pristup omogućava optimalizaciju propusnosti u veb aplikacijama, koja dopušta veliki broj ulazno izlaznih operacija. Ovim je omogućena izrada veb aplikacija koje rade u realnom vremenu, kao što su programi za komunikaciju ili igrice koje vidite na fejsbuku i slicnim mestima.

Šta se može uraditi upotrebom Node.js

- može se izraditi sadržaj dinamičkih stranica
- može se izraditi, otvarati, čitati, pisati, brisati i zatvoriti po failovima na serveru
- može prikupiti podatke iz formi.
- dodati, obrisati, promeniti podatke u bazi podataka.

Instalisanje programske aplikacije Node.js

Za instalaciju Node.js, može se upotrebiti instalater ili paket u binarnom obliku, koje se preuzima sa službenog veb sajta za node.js : Node JS Downloads.
https://nodejs.org/en/download/
Po preuzimanju faila ako upotrebljavate vindovs možete preuzeti i kompresovani fail, koji raspakujete u folder kog ste napravili predhodno sa imenom"nodejs", na particiji koju želite.
Slika dole to prikazuje



Ali u pocetku najbolje je preuzeti kada je vindovs u pitanju verziju koja se sama instalira.

Izrada jednostavnog server skripta sa Node.js

Kada je instalisan Node.js na vašem računaru, možete praviti js skriptove kojima možete izraditi sopstveni server. Node.js upotrebljava module za izradu skriptova. Moduli su kao i svaka druga java skript biblioteka koju uključujete u projekte upotrebom require() funkcijom.

Pa, evo kako možemo da dobijemo jedan server. Prvo napravimo folder projekta, onda mu pristupimo iz editora koda i tamo napravimo fail pod nazivom server.js sa sa sadržajem koda ispod. Sačuvamo ga u projektnom folderu zajedno sa ovim kodom.

```
//uključivanje http modula
const http = require('http');

//dodeljivanje konstante sa brojem porta na kome će biti pokrenut
// server
const port = 3000;

//izrada servera
const server = http.createServer((req, res)=> {
  res.writeHead(200, {'Content-Type':'text/plain'});
  res.write(Dobar dan, kako ste?.');
  res.end();
});

//pokretanje  servera sa postavljenim portom i prikaz porta u konzoli
server.listen(port, ()=> {
  console.log('Server running at //localhost:'+ port +'/');
});
```

Da bi ste videli kako radi server i proverili ispravnost koda dovoljno je iz konzole ili terminala pristupiti projekt folderu i uneti tamo

Kostanta port moze imati i ovakav kod
const port = process.env.PORT || 3000;
Čime se postavlja mogućnost upotrebe nekog drugog porta za izvršavanje aplikacije, jer env je skraćenica od environment u prevodu okruženje. Tačnije, ako nije drugačije određeno aplikacija se izvršava na portu 3000 .

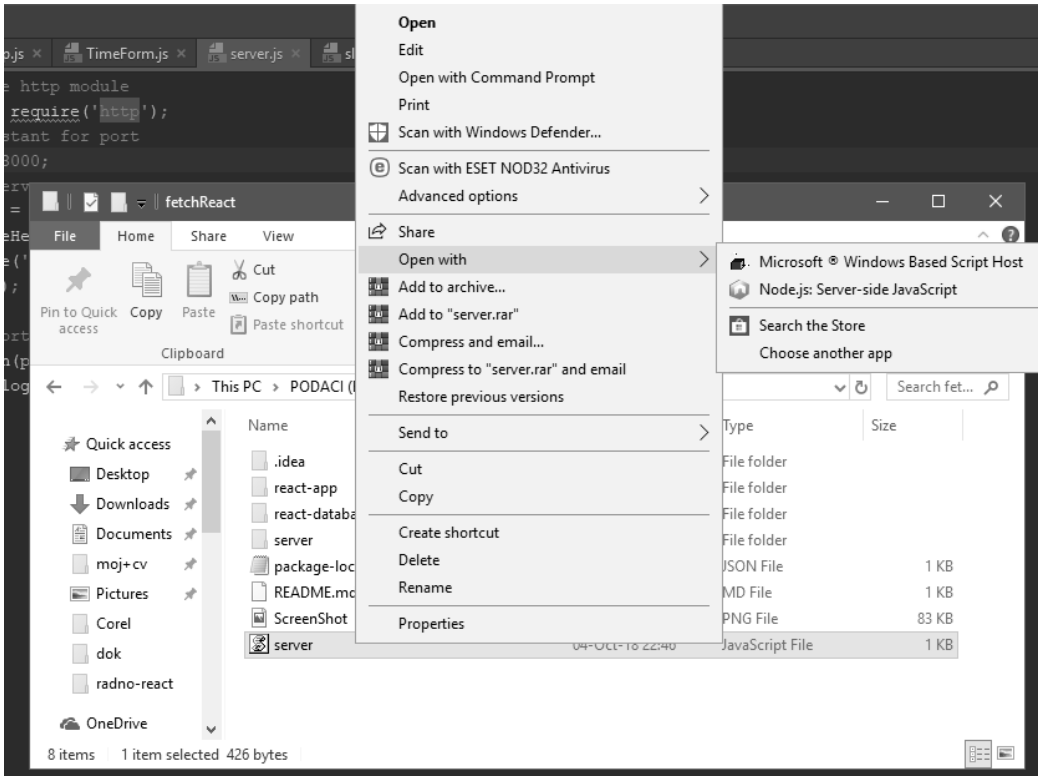
node server.js

U terminalu dobićete odgovor

```
E:\wamp64\www\fetchReact>node server
Server running at //localhost:3000/
```

U brovseru

Ili otvoriti eksplorer i u njemu pronaći projekt folder



kliknuti na fail server.js i otvoriti ga pomoću node.js.

Za prekid rada servera pritisnite Ctrl+C u CMD-u ili samo jendostavno zatvorite taj CMD prozor.

Node.js Moduli

Node.js se i sam sastoji od ugradjenih modula koji se nazivaju "core modules", koje možete upotrebiti bez ikakve dodatne instalacije, pa zato pogledajte tu sajtu modula na linku

https://coursesweb.net/nodejs/core-modules

Uključivanje modula

Da bi se upotrebili moduli Node.js, potrebno ih je uključiti require() funkcijom.

var module = require('module_name');

Na primer želite upotrebiti HTTP modul za izradu Node.js servera:

const http = require('http');

const server = http.createServer((req, res)=> {

```
 res.writeHead(200, {'Content-Type':'text/plain'});
 res.end('Hello to me.');
});

server.listen(8080, ()=> {
 console.log('Server running at //localhost:8080/');
});
```

Izrada Modula

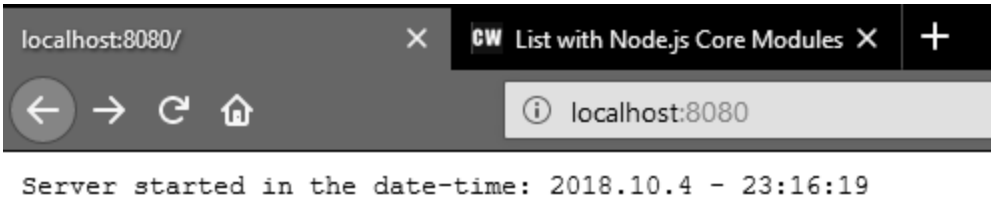
U node-u moduli su smešteni u posebnim JavaScript failovima. Da bi malo pojasnio poslužiću se primerom gde će biti izrađen modul sa objektom koji pokazuje trenutni datum i vreme:

```
//Modul koji daje i upotrebljava datum i vreme
class mDateTime {

 //postavljanje osobina za datum i vreme
 constructor() {
 this.dt = new Date();
 this.year = this.dt.getFullYear();
 this.month = this.dt.getMonth()+1;
 this.day = this.dt.getDate();
 this.hour = this.dt.getHours();
 this.minute = this.dt.getMinutes();
 this.seconds = this.dt.getSeconds();
 }
 //vraća string sa: godina.mesec.dan, ali možete promeniti redosled
 get date(){
 return this.year+'.'+this.month+'.'+this.day;
 }
 // vraća string sa: hour:minute:seconds
 get time(){
 return this.hour+':'+this.minute+':'+this.seconds;
 }
}
//Pronalaženje objekta mDateTime class sa module.exports
module.exports = new mDateTime();
```

Kada je pronađen opisani objekt kao instanca new mDateTime() sa **module.exports** biće mu napravljlene osobine i metode koje će biti primenjive i van modul faila. Tako da će biti omogućeno da objekt bude u upotrebi u skriptu kada je modul u njega priključen upotrebom sa require( ).
Sačuvajmo predhodni kod sa nekim imenom na primer vreme.js. Da bi smo proverili rad koda priključićemo ga kodu ispod koji je predhodno sačuvan u istom

folderu gde smo sačuvali i ovaj fail vreme.js.

```
//uključivanje http modula
const http = require('http');

//uključivanje modula vreme
var dt = require('./vreme');

// postavljanje u string datum i vreme, čime server bude startovan, upotrebom
// određenih osobina u modulu vreme
var current_dt = dt.date +' - '+ dt.time;

//definisanje kostante koja sadrži broj porta koji će se upotrebljavati
const port =8080;

//izrada koda servera
const server = http.createServer((req, res)=> {
 res.writeHead(200, {'Content-Type':'text/plain'});
 res.write('Server started in the date-time: '+ current_dt);
 res.end();
});

// prosleđvanje porta serveru
server.listen(port, ()=> {
 console.log('Server running at //localhost:'+ port +'/');
});
```



Node.js HTTP Modul

Node.js HTTP modul, dopušta da Node.js može da prihvata i šalje podatke putem upotrebe Hyper Text Transfer Protocol (HTTP).
Kao i za sve drugo pa i za HTTP modul, važi pravilu upotreba require() metoda:
const http = require('http');

Izrada Node.js veb servera

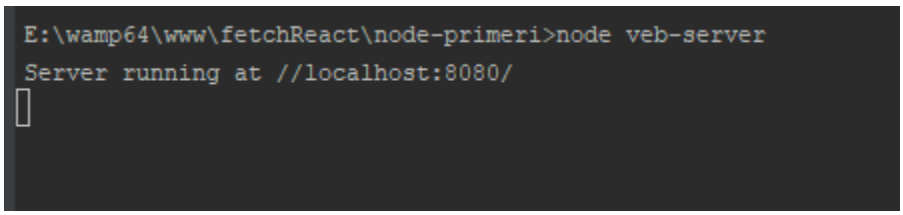
Sa HTTP modulom možemo izraditi HTTP server, tako da on sluša određeni port servera i šalje response nazad klijentu. Za izradu HTTP servera upotrebljava se createServer() metod:

```
//uključivanje http modula
const http = require('http');

//definisanje porta
const port =8080;

//izrada servera
const server = http.createServer((req, res)=> {
   res.writeHead(200, {'Content-Type':'text/plain'});
   res.write('Dobar dan meni sa veb servera na portu 8080 .');
   res.end();
});

// prosleđvanje porta serveru
server.listen(port, ()=> {
   console.log('Server running at //localhost:'+ port +'/');
});
```

Kao što se vidi funkcija biće prosleđena metodu **http.createServer()** kada neko pristupi localhostu na portu 8080.
Metod **res.writeHead()** uključuje HTTP headere u response.

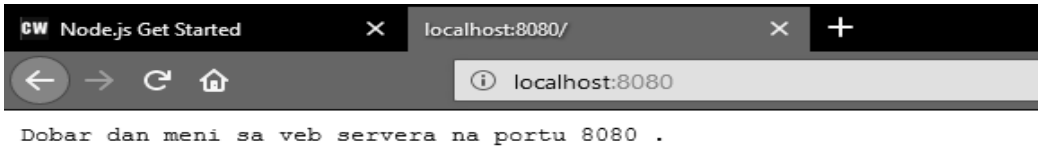
Prvi argument metode res.writeHead() je status kod, 200, što znači da je sve u redu (OK), drugi argument je objekt koji sadrži odgovore (response) headera.
- Tako na primer ako želite da kao izlazne informacije sadrže html tagove, upotrebite za header: {'Content-Type':'text/html'}.

Sačuvajmo ovaj kod pod nazivom "HTTPserver.js"
i pokrenimo ga sa node HTTPserver.js



- Kao što se vidi u odgovoru na ekranu terminla server radi.

Slika dole pokazuje rad servera u borvseru na http://localhost:8080



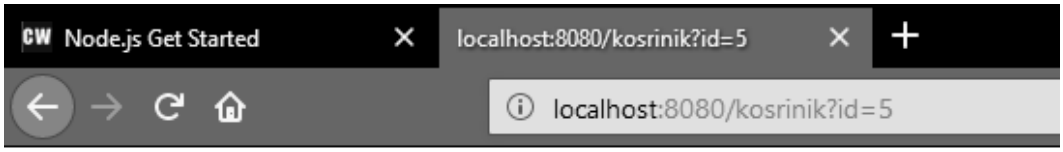
Čitanje Query Stringa iz URL-a

Funkcija propuštena u http.createServer() kao zahtev "req" argument prestavljen kao objekt, koji se zhateva od strane korisnika (http.IncomingMessage object). Ovaj objekat ima osobinu pozivnog "url", koji sarži delove urla koji dolazi posle imena domena.
Sačuvajte fail sa serverUrl.js

```
const http = require('http');

const server = http.createServer((req, res)=> {
  res.writeHead(200, {'Content-Type':'text/plain'});
  res.write(req.url);
  res.end();
});

server.listen(8080, ()=> {
  console.log('Server running at //localhost:8080/');
});
```

Po startovanju faila sa node serverUrl.js

u brovzeru dobijate jednu kosu crtu (znak za podeljeno), ali kada uneste posle http://localhost:8080 kosu crtu (znak za podeljeno) iza čega upisete nešto to će se i pojaviti na ekranu brovzera iza kose crte




```
/kosrinik?id=5
```

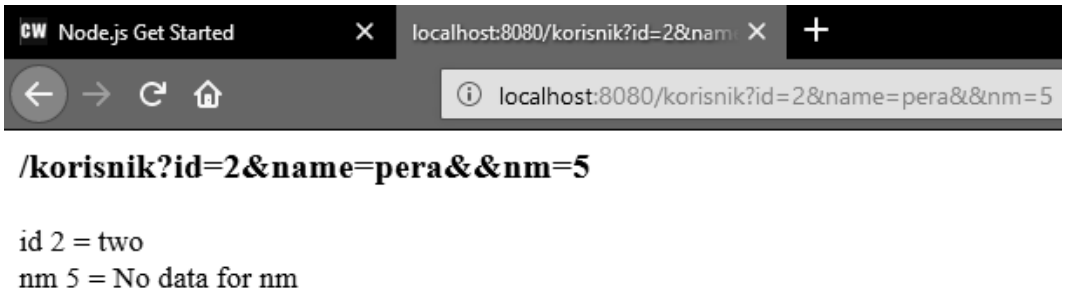
Podela Query Stringa u URL-u

U ovom primeru napravićemo modul koji će pročitati sadržaj url-a i podeliti ga na delove koje će prikazati u brovseru.

```
//ukljucivanje http i url modula
const http = require('http');
const url = require('url');

//objekt koji sadrži podatke, koje u cilju vežbe možete menjati
var cnt ={
   id:{1:'one', 2:'two', 3:'three'},
   nm:{'a':'Atom', 'e':'Electron', 'n':'Nothing'}
};

//izrada servera
const server = http.createServer((req, res)=> {
   res.writeHead(200, {'Content-Type':'text/html'});

   //dodavanje headera za prikaz html sadržaja
   //podela urla i dobijanje objekta sa {name:value} u upotu urla (query)
   var q = url.parse(req.url, true).query;

   //dobijanje i pisanje sadržaja u $cnt u url kao podatak
   var cnt_id = cnt.id[q.id] ? cnt.id[q.id] :'No data for id';
   var cnt_nm = cnt.nm[q.nm] ? cnt.nm[q.nm] :'No data for nm';
   var add_cnt ='id '+ q.id +' = '+ cnt_id +'<br>nm '+ q.nm +' = '+ cnt_nm;
   res.write('<h3>'+ req.url +'</h3>'+ add_cnt);
   res.end();
});

server.listen(8080, ()=> {
```

```
    console.log('Server running at //localhost:8080/');
});
```

Kod može da odgovori sa unosom kao što vidite



ili ako uneste /?id=1&nm=a

Na ekranu se dobija tekst ispod:
id 1 = one
nm a = Atom

Node.js rad sa failovima

Node.js fail sistem modul (*fs*), dopušta rad sa failovima i folderima na vašem serveru. Oznaka za upotrebu ovog modula je "fs":
const fs = require('fs');

Čitanje Failova

Upotrebom ovog metoda fs.readFile() možete pročitati neki fail na vašem serveru. U sledećem primeru pročitaćemo sadržaj html faila koji se zove failzacitanje.html i node faila citanje failova.js kojim vršimo čitanje.

fail failzacitanje.html

```
<html>
<body>
<h1>Fail koji će se pročitati</h1>
<p>I prikazati njegov sadržaj.</p>
</body>
</html>
```

Node fail citanjefailova.js

```
//uključivanje http-a i fs modula
const http = require('http');
const fs = require('fs');
```

```
//putanja i ime faila koji se čita
let file ='./failzacitanje.html';

//postavljanje servera
const server = http.createServer((req, res)=>{

   //čitanje faila; i ako nema grešaka biće prikazan njegov sadržaj
   fs.readFile(file, (err, data)=>{
      if(err) throw err;
      else {
         res.writeHead(200, {'Content-Type': 'text/html'});
         res.write(data);
      }
      res.end();
   });
});

server.listen(8080, ()=> {
   console.log('Server running at //localhost:8080/');
});
```

Ovim kodom u primeru možete pročitati i druge vrste failova, samo što će oni biti prikazani kao klasičan tekst, jer primer je formatizovan da prikazuje HTML sadržaj.

Dobijanje informacija o failovima

Za dobijanje informacija o failovima u node-u se upotrebljava metod **fs.stat()**.
U kodu niže dobićemo informacije o failu "ns.js":

```
const fs = require('fs');
var file ='./ns.js';

fs.stat(file, (err, stats)=>{
   if(err) throw err;
   console.log(stats);

   // Check file type
   console.log('isFile ? '+ stats.isFile());
});
```

```
E:\wamp64\www\fetchReact\node-primeri>node getinfo

Stats {
  dev: 3056659526,
```

```
  mode: 33206,
  nlink: 1,
  uid: 0,
  gid: 0,
  rdev: 0,
  blksize: undefined,
  ino: 3096224743949206,
  size: 700,
  blocks: undefined,
  atimeMs: 1538687764391.6677,
  mtimeMs: 1538687764403.6475,
  ctimeMs: 1538688105318.0386,
  birthtimeMs: 1538687764391.6677,
  atime: 2018-10-04T21:16:04.392Z,
  mtime: 2018-10-04T21:16:04.404Z,
  ctime: 2018-10-04T21:21:45.318Z,
  birthtime: 2018-10-04T21:16:04.392Z }
isFile ? true
```

- Status parameter je callback funkcija u objektu sa fail informacijom (mode, size, etc.); naravno možete upotrebiti i druge metode da bi ste proverili tipove failova.

stats.isFile() - vraća true ako fail tipa fail
stats.isDirectory() - vraća true ako je tip faila direktorijum.
stats.isBlockDevice() – vraća istinu ako je fail tipa blok uređaja.

Izrada i dodavanje podataka u fail

Sam modul "fs" ima metode za izradu novih failova, ili dodavanje podataka u već postojeći fail.

- fs.appendFile() za dodavanje podataka
- fs.open() otvaranje faila
- fs.writeFile() upis podataka u fail

1. Ako zahtevani fail u kome se dodaju podaci postoji, metod **fs.appendFile()** će dodati podatke i zahtevan sadržaj, u suprotnom napraviće prvo fail pa mu onda dodati sadržaj.
- u ovom primeru biće napravljen fail newfile.htm upotrebom metode appendFile(), pošto ne postoji:

```
const fs = require('fs');
var file ='./test/newfile.htm';
var content ='<div>Html content</div>';
```

```
fs.appendFile(file, content, (err)=>{
  if(err) throw err;
  console.log('Saved.');
});
```

I putem konzole u terminalu dobićete poruku console.log('Saved.');

2. Metod **fs.open()** u svom radu uzima "flag" kao drugi argument, pa ako je flag "w" za "writing", biće otvoren fail za upis podataka. U slučaju da traženi fail ne postoji biće kreiran prvo prazan fail, pa onda omogućenje za upis.

- Metoda fs.open() može biti upotrebljena za rad sa failovima kao što su: izrada novih failova, čitanje, pisanje u fail, u zavisnosti od upotrebljenog flaga (r, r+, a, w, w+, itd). pogledajte na linku dokumentaciju o tome https://nodejs.org/api/fs.html#fs_fs_open_path_flags_mode_callback .

  - Primer: izrada html faila upotrebom fs.open() metoda, i dodavanje sadržaja otvorenog faila upotrebom fs.write() metoda:

```
const fs = require('fs');
var file ='./nekifail.htm';
var content ='<div>Novi sadržaj faila</div>';

fs.open(file, 'w', (err, fd)=>{
  if(err) throw err;
  else {
    console.log('File created');
    fs.write(fd, content, (err)=>{
      if(err) throw err;
      console.log('Data added in file');
      fs.close(fd, (err)=>{ if(err) throw err;});
    });
  }
});
```

3. Metod **fs.writeFile()** zamenjuje navedeni fail i sadržaj ako postoji. Ako fail ne postoji biće izrađen i dodat mu novi sadržaj
Izrada novo faila upotrebom metoda writeFile():

```
const fs = require('fs');
var file ='./test/newfile.htm';
var content ='<div>Html content</div>';

fs.writeFile(file, content, (err)=>{
```

```
  if(err) throw err;
  console.log('Saved.');
});
```

Promena imena failovima

File System modulom moguća je promena imena failova upotrebom metoda **fs.rename()** .
Promena imena failova je ovakva "staroime.htm" to "novoime.htm":

```
const fs = require('fs');
var file ='./ staroime.htm ';
var newname ='./ novoime.htm';

fs.rename(file, newname, (err)=>{
  if(err) throw err;
  else console.log('Renamed.');
});
```

Brisanje failova

Ono što je primetno u radu FS modula je to da je jako sličan, sličnim metodama koje se upotrebljavaju u PHP-u za iste namene. Za brisanje failova upotrebljava se metod **fs.unlink()**.

Način za brisanje failova je prikazan u kodu dole

```
const fs = require('fs');
var file ='./putanja/failkojsebriše.njegovaekstenzija';
fs.unlink(file, (err)=>{
  if(err) throw err;
  else console.log('Deleted.');
});
```

Node.js: fs-extra

Drugi način za lakši rad sa folderima i failovima je upotreba **fs-extra modula,** koji praktično dodaje nove metode koje nisu uključeni u izvorni fs modul, pa tako ga može zameniti.
li pre upotrebe ovog modula potrebno ga je prvo instalisati

```
npm install --save fs-extra
```

Dobro je i instalisati nodemon, kako ne bi se stalno vršilo prekidanje sa ctrl+c i ponovni start sa node ime_faila.

```
npm i –g nodemon
```

iza čega se upotrebljava nodemon ime_faila umesto node ime_faila, ali nemorate dok razvijate aplikaciju da neprestalno zaustavljate server i ponovo pokrenete posle sačuvane promene.
Nakon čega je u node projektima praktično zamenjen izvorni fs modul.

```
const fs = require('fs-extra');
```

Sve metode iz fs modula sadžane su u fs-extra modulu. Sve fs metode vraćaju promises, ako je callback prošao.

```
const fs = require('fs-extra')

// Async with promises:
fs.copy('/tmp/file.htm', '/tmp/dirx/file.htm')
.then(() => console.log('success!'))
.catch(err => console.error(err));

// Async with callbacks:
fs.copy('/tmp/file.htm', '/tmp/dirx/file.htm', err =>{
  if (err) return console.error(err);
  console.log('success!');
});

// Sync:
try {
  fs.copySync('/tmp/file.htm', '/tmp/dirx/file.htm');
  console.log('success!');
} catch (err){
  console.error(err);
}
```

- Za više informacija na linku https://www.npmjs.com/package/fs-extra

Sync i Async i Async/Await

Metod async je default postavljen. Sve async metode vratiće promise, ako je callback stvarno propušten. Metodi sihronizacije sa druge strane će prijaviti moguće greške. Naravno Async/Await prijavljuju greške.

```
const fs = require('fs-extra')

// Async sa promises:
fs.copy('/tmp/myfile', '/tmp/mynewfile')
```

```
  .then(() => console.log('success!'))
  .catch(err => console.error(err))

// Async sa callbacks:
fs.copy('/tmp/myfile', '/tmp/mynewfile', err => {
  if (err) return console.error(err)
  console.log('Uspešno!')
})

// Sync:
try {
  fs.copySync('/tmp/myfile', '/tmp/mynewfile')
  console.log(' Uspešno!')
} catch (err) {
  console.error(err)
}

// Async/Await:
async function copyFiles () {
  try {
    await fs.copy('/tmp/myfile', '/tmp/mynewfile')
    console.log(' Uspešno!')
  } catch (err) {
    console.error(err)
  }
}

copyFiles()
```

Sadržaj metoda- ispod je sve ovo što svaka metoda podržava

Async

- copy
- emptyDir
- ensureFile
- ensureDir
- ensureLink
- ensureSymlink
- mkdirp
- mkdirs
- move
- outputFile
- outputJson
- pathExists

- readJson
- remove
- writeJson

Sync

- copySync
- emptyDirSync
- ensureFileSync
- ensureDirSync
- ensureLinkSync
- ensureSymlinkSync
- mkdirpSync
- mkdirsSync
- moveSync
- outputFileSync
- outputJsonSync
- pathExistsSync
- readJsonSync
- removeSync
- writeJsonSync

REACT.JS

React

React je praktično jedna velika biblioteka java skript otvorenog koda, koja za svoj rad upotrebljava u osnovi dva dela: react i react-dom. Prvi omogućava rad u virtuelnom svetu, stvaranje svih elemenata veb stranica. Dok, react-dom omogućava da sve to napravljeno reactom se poveže i prikaže u stvarnom DOM-u, pri čemu je obezbeđen jednosmeran tok podataka. Odnosno jedne komponente imaju podatke (state) i one ih prosleđuju drugima koje ih prihvataju (props) i prikazuju u stvarnom DOM-u. Sve je to omogućeno upotrebom proširenog java skript koda(JSX). Ovakva filozofija pristupa rešavanju problema preuzeta je iz fejsbukovog proširenja za PHP, XMP. Čime je omogućen lakši način pristupa i izradi aplikacija, jer je izrađen veliki broj dodatatnih paketa, koji omogućavaju lakšu upotrebu svih dosadašnjih tehnologija. U praksi postiže se dobijanje iz mnoštva failova samo jednog faila, kog možete nazvati kako god želite, ali uobičajeni naziv je bundle.js, kog se ugrađuje u index.html fail. Dobijene u kodu beline u redovima ne postoje. Za razliku od svih dosadašnjih tehnologija, kod je kompresovan.
Pored toga, na veoma jednostavan nacin omogućava povezivanje između komponenti i razmenu podataka. Pri čemu se veličina napisanog koda i pojava grešaka se drastično smanjuju. Tako da nije potrebno imati glomazan kod veb aplikacije, gde morate da praktično promene pravite samo u srednjem delu stranice, dok gornji i donji deo ostaju u svakoj stranici bez promena kao što je to bilo u upotrebi sa klasičnim html stranicama. Jednom rečiju možete da napravite sopstvene komponente i njihove delove i da ih prilikom svakog poziva na bilo kom mestu u aplikaciji dopunite podacima upotrebom koda

this.props.children

s time što će te ostaviti tagove otvorenima

<Podkomponenta> Ovaj deo je Child </Podkomponenta>

Dodajući u kod faila Podkomponete između div tagova u njenom return delu {this.props.children}

```
render(){
      return(
              <div>
                // kod podkomponente
                {this.props.children}
               </div>
           );
       }
```

U react aplikacijama je omogućeno miksovanje koda : HTM, XTML, java skripta, s time što je u kodu faila komponente određeno gde šta može da se napiše. Mislim na konstante, funkcije, konstruktore, html i drugi kod. Samim time je omogućeno da se po ugledu na C, php i slične mogu uputiti pozivi ka globalnim ili lokalnim promenjivama, konstantama, funkcijama. Ali, se može slobodno upotrebiti u istom failu komponente mogu naći više klasa, konstanti funkcija čiji je kod jedan ispod drugog gde je samo jedna izlazna. To jest ima ovaj kod

export default Ime_komonente;

Što će te videti u delovima knjige koji obrađuju sintaksu koda. U njemu su ugrađene i specijalne komponente, tako zvane HOC komponente, kolinranje elemenata, povezivanje komponenti ili ponistavanje postojanja, ili ograničenja rada komponente, kao i update o dr.
Povrh svega instalisanjem programske aplikacije NODE i jednog editora koda nije vam potrebno u izradi aplikacija ništa drugo, osim ako ne upotrebljavate php i mysql. Jer instalcijom Node dobijate upravljač instalacije dodatnih paketa. Možete automacki da izrađujete aplikacije. Prilikom čega upravljač instalacija, u zavisnosti koji upotrebite npm ili yarn dobijate po defoltu nakon kreiranja aplikacije sve što vam je potrebno za neometan rad vezan za razvoj i produkciju aplikacija. Dovoljno je da uneste samo

npm create-react-app –g

pređete u projekt folder

creat-react-app ime-aplikacije

I za par minuta imaćete već spremnu aplikaciju za razvoj, dovoljno da u projektnom folderu unesete nakon toga

cd ime-aplikacije
npm start

I aplikacija biće pokrenuta na http://localhost:3000 u brovseru.

Ono što je još bitno vezano za ovo programersko okruženje je da u slučaju da nešto menjate na budućoj veb stranici je da se samo to i menja dok ostali deo stranice je ne promenjen. Ovo je za razliku od mnogih prisutnih tehnologija u ovom području veliki napredak. Pored toga dužina samog koda failova je jako mala upoređujući je sa dužinom failova u php, html, xtml, xml.
React svojom koncepcijom omogućava i da pridružimo funkciju promenjivama, konstantama i dr, samim time je omogućeno slanje vrednosti za props, događaje i sve ono što se može obraditi u funkcijama.

Prilikom izrade react aplikacija u okruženju node-react, potrebno je u projektnom folderu kada se izrađuje sa serverskim delom, u razvojnom delu je napraviti još jedan folder za korisnički deo aplikacije. Dok u projektnom folderu se radi serverski deo aplikacije.

U slučaju izrade react aplikacije upotrebe reacta i phpa u razvojnom delu izrađuju se dva foldera jedan za react aplikaciju korisnički deo i drugi za php. I na kraju izrade aplikacije se sadržaj iz dobijenog build fodera prekopira na apache server zajedno sa php folderom

Početni koraci

Svaki početak je težak, ali kada se prvi korak načini onda sve izgleda kao dečija igra. Tako je i sa reactom i svim ostalim u životu. Zato, neka vas prvi neuspeh ne obeshrabri da krene trnovitom stazom znanja dalje u veselu budućnost. Jer, se svi jednako isti rađamo nebitno da li smo beli, žuti, crni isti nas put čeka i ista igra sa nevidljivim protivnikom sa one druge strane šahovske table, koja prestavlja našu pozornicu života. Kako ćemo igrati šah života zavisi od našeg znanja koje počiva na temeljima ljubavi. Čuvanjem onoga što jesmo uz pomoć ljubavi i znanja možemo da razumemo svet oko nas bez trunke straha onakvim kakav je on stvarno. Samo tako shvatićemo i jednostavnost i programiranje. Tako, da ova knjiga osnova postaće samo jedan mali kamenčić u vašem mozajku koji će dozvoliti meni da živim kroz vas, a vama da živite kroz druge kojima prenosite znanje. Znanje nije znati već umeti da ga na prilagodan način prenesete dalje, pri tome da ga razmenjujete sa drugima. Samo ako umete da razmenjujete nepristrasno onda će te imati sve više znanja koje će biti i ostati deo vas, a vaše aplikacije imaće najveću složenost. Biće veoma jednostavne, kratke i neće imati nikavih grešaka u sebi. Za neke namene može se upotrebiti i ovakav način upotrebe pisanja koda u reactu bez upotrebe Nodejs:

```
<!DOCTYPE html>
<html>
<head>
<meta charset= “utf-8>
<title> Neki naslov</title>
</head>
<body>
<div id=“ root“> </div>
<script src="https://unpkg.com/@babel/standalone/babel.js"></script>
<script src="https://unpkg.com/react/umd/react.development.js"></script>
<script src="https://unpkg.com/react-dom/umd/react-
dom.development.js"></script>
<script type="text/babel">

const Komponenta = () => <h1>Hello World today!</h1>;
```

```
ReactDOM.render(
< Komponenta />,
document.getElementById('root')
);

</script>
</body>
</html>
```

Primenom ovog koda možete eksperimentisati, no kod velikih aplikacija mogu nastati problemi u radu. Ono što može olakšati ovaj rad je preuzimanje skriptova sa neta i postavljanje na svoj lokalni računar ili na neki server, kako bi aplikacija radila sigurnije i brže.

Jednostranične React aplikacije

Jednostranične React aplikacije ili SPA, podsećaju na klasične desktop aplikacije. SPA sadrži aplikaciju koja se učitava samo jednom prilikom inicijalnog pokretanja aplikacije, dok sve ostale promene koje nastanu kao odgovor na korisnikove zahteve su parcijalne.Tako da je sva logika uglavnom prebačena na korisnikovu stranu, što omogućava brze promene stanja, budući da svaka promena ne zahteva podatke sa servera.
Kod standarnih HTML veb aplikacija uvek je zahtevano da se vrši osvežavanje stranica, dok kod SPA nakon učitavanja aplikacije sva komunikacija se odvija kroz AJAX ili XML pozive, na koje server odgovara uglavnom u JSON formatu.Tako dobijeni podaci aplikacija koristi za sopstveno dinamičko ažuriranje stranice bez njenog ponovog učitavanja. Dok, server se isključivo koristi kao uslužni sloj.

Arhitektura servera

Obzirom da komunikacija sa serverom se odvija u pozadini i na samu ulogu servera u aplikaciji, možemo razlikovati tri vrste SPA:

1. Arhitektura tankog servera, gde je sva logika prebačena na korisnika, čime je uloga servera svedena na jednostavnu vezu za prihvatanje podataka. Ovakvim pristupom se smanjuje kompleksnost u celini, a ove aplikacije se smatraju čistim jednostraničnim aplikacijama.
2. One su suprotne od predhodnih, znači debeli server u prevodu, ali podeljen u dve mogućnosti:

a) server čuva potrebna stanja u korisničkoj memoriji i kada dobije zahtev, pošalje odgovarajući odgovor koji ažurira stanje korisnika, ali u isto vreme i ažurira svoje stanje. Ovim načinom obzirom da se većina događaja i stalnijih poziva prema serveru zahteva i veći utrošak njegove memeorije.Tako da on zahteva da server ima veću količinu memorije, dok aplikacija je pojednostavljena jer su svi potrebni podaci već na serveru.

b) u ovom slučaju server ne čuva korisničke podatke, već ih korisnik šalje svoje stanje uglavnom AJAX zahtevom. Zatim server, rekonstruiše stanje dela stranice koji se obnavlja, prilikom čega vraća kod korisniku koji ga dovodi u novo stanje, te je za ovaj postupak izrade aplikacija potrebno poslati nešto više podataka nego u predhodnom slučaju, jer je potrebno i više resursa u radu. Ovim postupkom obzirom da server ne čuva specifične podatke za svakog korisnika, AJAX zahtevi mogu biti poslati na više čvorova bez potrebe za sesijama.

Komponente koje imaju već svoja stanja (stateless)

U reactu je dozvoljeno da svako ima svoje podatke i da bude njihov vlasnik, kao i u primeru dole gde imamo kod tri komponente u jednom failu App.jsx. Te je ova komponenta je vlasnik komponenti Header i Content. Gde smo posebno izradili Header i Content i dodali unutar App komponente, ali samo je App komponenti omogućeno eksportovanje. Kod komponenti Header i Content može biti napisan i van koda klase App i sve će raditi isto. Ali kada bi smo svaku komponentu napisali u poseban fail što je isto moguće tada bi svaka od njih imala svoje eksportovanje, te u tom slučaju bi svaku od njih morali da importujemo u App.

App.jsx

```
import React from 'react';

class App extends React.Component {
  render() {
    return (
      <div>
        <Header/>
        <Content/>
      </div>
    );
  }
}

class Header extends React.Component {
  render() {
    return (
      <div>
        <h1>Header</h1>
      </div>
    );
  }
}
```

```
class Content extends React.Component {
  render() {
    return (
      <div>
        <h2>Content</h2>
        <p>The content text!!!</p>
      </div>
    );
  }
}

export default App;
```

Ovako napisan kod je nemoguće videti bez donjeg koda faila main.js. Ovaj fail može se zvati i index.js. Njegova namena je da poveže virtualni DOM koji prestavlja kod faila u ovom slučaju App.jsx i da ga poveže sa realnim DOM-om uz upotrebu ReactDOM.render(). ReactDOM je taj najbitniji dodatak reactu, jer on kao što se vidi u kodu dole pronalazi java skript kodom id stvarnog DOM elementa html stranice

```
document.getElementById('app')
```

Koji od njega dobija podatke koje treba da prikaže. Prednost samog ovakvog pristupa omogućava stalne promene i to samo jednog dela stranice koji se menja. Dok ostali delovi ostaju isti dok se i za njih ne pošalje nalog za promenom. Te tako je omogućeno da se svaki deo stranice radi zasebno i to se može učiniti da taj svaki deo ima svoje pod delove.

main.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';

ReactDOM.render(<App />, document.getElementById('app'));
```

Slika prikazuje rezultat rada predhodnog koda

Slika React Stateless komponente

Komponente kojim se prosleđuju stanja (Stateful)

U ovom slučaju vlasnik vrednosti stanja (state) je isto komponenta (App). Komponenta Header je dodata iz predhodnog primera. Ali dodaću umesto komponente Content, html tag tabele, gde ću u telo tabele pozvati komponentu TableRow, koja će dinamički prikazati svaki objekt podataka koji joj prosleđuje komponenta(App) u obliku vrednosti stanja.
Ono što još je u ovom primeru, je upotreba koda EcmaScript 2015 arrow syntax (=>) kojim se dobija mnogo sintaksno čistiji java skript kod.
To će nam pomoći u stvaranju naših elemenata novim linijama koda. To je posebno korisno kada je potrebno stvoriti puno stavki.

App.jsx

```
import React from 'react';

class App extends React.Component {
  constructor() {
    super();

    this.state = {
      data:
      [
        {
          "id":1,
          "name":"Foo",
          "age":"20"
        },

        {
          "id":2,
          "name":"Bar",
          "age":"30"
```

```
        },

        {
          "id":3,
          "name":"Baz",
          "age":"40"
        }
      ]
    }
  }

  render() {
    return (
      <div>
        <Header/>
        <table>
          <tbody>
            {this.state.data.map((person, i) => <TableRow key = {i}
              data = {person} />)}
          </tbody>
        </table>
      </div>
    );
  }
}

class Header extends React.Component {
  render() {
    return (
      <div>
        <h1>Header</h1>
      </div>
    );
  }
}

class TableRow extends React.Component {
  render() {
    return (
      <tr>
        <td>{this.props.data.id}</td>
        <td>{this.props.data.name}</td>
        <td>{this.props.data.age}</td>
      </tr>
```

```
      );
    }
}

export default App;
```

Napomena:
Primetna je upotreba key = {i} unutar funkcije map(). To će vam pomoći da reagujete na ažuriranje samo potrebnih elemenata umesto ponovnog prikazivanja cijelog popisa kada se nešto promeni. To je ogroman poticaj performansi za dinamički stvorene elemente. Rezultat rada koda prikazuje slika dole.

React Komponente

State je mesto na kom dolaze podaci. Uvek biste trebali pokušati učiniti vašu vrednost State jednostavnijom mogućnošću i smanjiti broj komponenata (stateful) koje se mogu držati. Ako imate, na primer, deset komponenti kojima su potrebni podaci vrednosti stanja (State), trebali biste napraviti jednu komponentu koja će sadržati sve te vrednosti za sve njih. Upotreba Props, koju kod u nastavku pokazuje kako stvoriti stateful komponentu pomoću EcmaScript2016 sintakse.

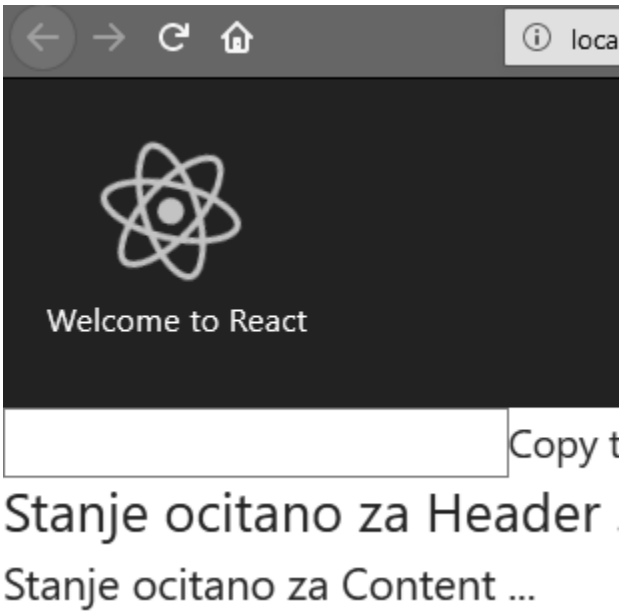
App.jsx

```
import React from 'react';

class App extends React.Component {
  constructor(props) {
    super(props);
```

```
    this.state = {
        “header”: "Header from state...",
      "content": "Content from state..."
    }
  }

  render() {
    return (
      <div>
        <h1>{this.state.header}</h1>
        <h2>{this.state.content}</h2>
      </div>
    );
  }
}

export default App;
```

main.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';

ReactDOM.render(<App />, document.getElementById('app'));
```



Uopšteno o state i props

Vrednostima za stanja (state) i osobine (props) react vrši sva upravljanja u svojim aplikacijama. Stanja su promenjive vrednosti koje mogu biti postavljene globalno: kao sadržaji kosntanti ili promenjivih ili kao pročitani sadržaji iz baze podataka, ili lokalno: kao sadržaji kosntaniti ili promenjivih. Njih je moguće menjati putem slanja između komponenti diretkno ili upotrebom posebnog postavljenja sa ovim kodom:

this.setState({ })

Gde se vrednosti mogu prema trenutnoj situaciji ili zahtevu korisnika promeniti, znači postavljena kao početna u neka potrebna unutar veliki zagrada se unose u JSON formatu

nekoStanje : this.state.novoStanje

Stanja možete menjati očitavanjem vrednosti iz unosnih polja kod formi za unos emaila i slično, i isto tako te očitane vrednosti možete proslediti na porebnim mestima u obliku JSON parova podataka.

Stanja i osobine

U ove namene se upotrebljavaju reči koje označavaju stanja (**state** za početna stanja i **setSate** za postavljanja novih stanja vrednosti promenjivih, konstanti i slično), dok za osobine se upotrebljava (**props),** obe reči se pišu sve malim slovima. Stanja šalju osobinama vrednosti. Osobine - props su uvek samo za očitavanje vrednosti. Stanja mogu imati klase i postavljanje osnovne vrednosti stanja postavljaju se obično u delu konstruktora za te namene.

```
class App extends React.Comoponent{
        constructor(){
        super();
        this.state= {
        element: njegova vrednost
}
```

Ako se prilikom izrade upotrebljavaju više od jednog elementa kome se dodeljuju osnovne vrednosti stanja u tome slučaju se na kraju prvog reda upisuje znak zareza.

```
this.state= {
        element: njegova vrednost,
        element1: njegova vrednost,
        .....
}
```

Ovo ovakvo napisano govori o formatu za razmenu podataka koji se naziva JSON, u prevodu java skript objekt notacija. JSON je nešto što bi smo nazvali prečica za razmenu podataka nebitno da li se upotrebljava u razmeni react-react ili php-php, jer je to univerzalni format koji se primenjuje svuda gde su potrebe za njime. Sastoji se od uparenih podataka. Ključa kog odvaja od njegove vrednosti dvotačka. Što se može i nazvati ako se u celini posmatra Objektom i načinom programiranja koje se naziva objektono programiranje. Gde po potrebi može pristupiti samo jedno podatku preko upotrebe njegovog kljuca i pročitati njegovu vrednost.

Napomena: upotreba JSON podataka nije ograničena samo na failove, koji su jedna vrsta malih baza podataka, čiji sadržaj možete veoma lako da promenite. Njih možete da postavite pridužujući ih konstantama. Ovaj format zapisa se može primeniti ako imate veb sajt na više govora što će biti kasnije objašnjeno malo detaljnije kada i obradimo dogadjaje.
Ovakvim pristupom problemu izrade i rada u Reactu postignuto je to da se zna koja komponenta je vlasnik podataka odnosno roditeljska komponata, a koja komponenta je praktično njena pod komonenta ili kako se to u svetu računara kaže Child (dete svog roditelja). Omogućeno je i kloniranje elemenata

```
import React from 'react';

export default class Pomocna extends React.Component{
   constructor(props){
      super();
      this.state={
         age: 10,
      };
   }
   napraviStarije(){
      console.log(this.props);
      this.setState({
        age:this.state.age +3
      });
   }
   render(){
      return(
         <div>
            <h2>Primena promene stanja sa state i setstate</h2>
               <p>Ime je---- {this.props.user.name},
                Godine koje se menaju=={this.state.age}
               </p>
               <hr/>
               <button onClick={()=> this.napraviStarije()}
                className="btn btn-danger">
```

```
                Povećaj godine
            </button>
        </div>
    );
  }
}
```

U ovom primeru imate ilustrovano promene vrednosti stanja klikom na dugme.

Zadavanje novog stanja

Ova mogućnost omogućava dinamička - čitaj stalna potrebna nova stanja vrednosti neke promenjive u komponentama. U ovu svrhu se može upotrebiti ovakav kod, gde imamo neku promenjivu kojoj smo prvo odredili početnu vrednost stanja i kojoj kasnije dodajemo po potrebi neku novu vrednost događajem onClick i u delu rendera se posle njenog renderovanja (obrade za slanje stvarnom DOM elementu koji je deo faila public/index.html. Najbolji primer za ovu svrhu je upotreba nekog brojača, gde ćemo zadati početnu vrednost nula i onda u koracima na primer vrednosti broja jedan povećavati tu vrednost. Komponente i još dosta drugih pogodnosti.

Fail Promena.js

```
import React,{Component} from 'react';

export default class Promena extends Component{
        constructor(props){
                super(props);
                this.state={
                        data: 0 // početno stanje vrednosti promenjive
                };

                // omogućavanje rada funkcija za promenu stanja
                this.setPlusBroj= this.setPlusBroj.bind(this);
                this.setMinusBroj= this.setMinusBroj.bind(this)

        };

        // povećanje trenutno vrednosti
        this.setPlusBroj(){
                this.setState({
                        data: this.state.data +1
                    // povećavanje trenutne vrednosti za korak jedan
                });
        }
```

```
        // smanjenje trenutne vrednosti
            this.setMinusBroj(){
                this.setState({
                        data: this.state.data -1
                });
        }

        render(){
                return(
                <div>
                 <h2> Smanjivanje i povećavanje</h2>
                  <p>
                     Ovim primerom se prikazuje dinamička promena stanja
                     klikom na dugmad za korak u vrednosti boja jedan
                  </p>
                {/* Dugmad koja pozivaju funkcije za povećanje ili smanjenje
                trenutne ili počene
                     vrednosti promenjive
                */}
                <button onClick={this.this.setPlusBroj}>Povecanje</button>
                <button onClick={this.this.setMinusBroj}>Smanjenja</button>
                {/*
                  pozivanje pod kompnente(child) kojoj se dinamički dodaje nova
                vrednost stanja
                  koju ona očitava kao svoju osbinu (props) i prikazuje je kao takvu
                  Sardžaj ime pod komponente mojBroj je promenjiva u
                  podkomponeti koja prima
                  promenu stanja.
                */}
                <Sadrzaj mojBroj={this.state.data}/>
                </div>
                );
        }
}

// Pod komponenta Sadrzaj
{/* Koja prima promenu stanja i prikazuje istovremeno kao svoju osobinu (props)
skraćeno od reči properties u prevodu svojstva ili osobine*/}

class Sadrzaj extends Component{
        render(){
                return(
                  <div>
```

```
                {/*
                Pod komponenta Sardržaj ispisuje dobijenu vrednost promenjive
                mojBroj, osobine,tj. prikazuje svoja svojstva upotrebom koda
                this.props.mojBroj uokvirenog velikim zagradama, obzirom da je to
                način za prikazivanje java skripta u delu render postavljenog između
                html tagova
                */}
                <h2> {this.props.mojBroj}</h2>
                </div>
                );
        }
}
```

Objašnjenje: Primer ovog faila Promena.js sadrži dve klase gde je prva roditelj, a druga dete klasa. Roditelj određuje osobinu detetu time što mu šalje postignute vrednost stanja. Ono što je još potrebno reći je sledeće: da se u reactu može pisati u jednom failu više više klasa, konstanti promenjivih, ali je najbolji način da svaka ima svoj poseban fail. Za taj slučaj kada postoje posebni failovi potrebno je u delu import klase roditelj importovati klasu dete pre njenog pozivanja u klasi roditelj. Ovako:

```
import React,{Component} from ‘react’;
// posto su u istom folderu putanja do faila sadržaj se ovako piše
import Sadrzaj from ‘./sadrzaj’;

export default class Promena extends Component{
```

Iz čega se postavlja

```
<Sadrzaj mojBroj={this.state.data}/>
```

onamo gde je to potrebno u kodu roditelj klase

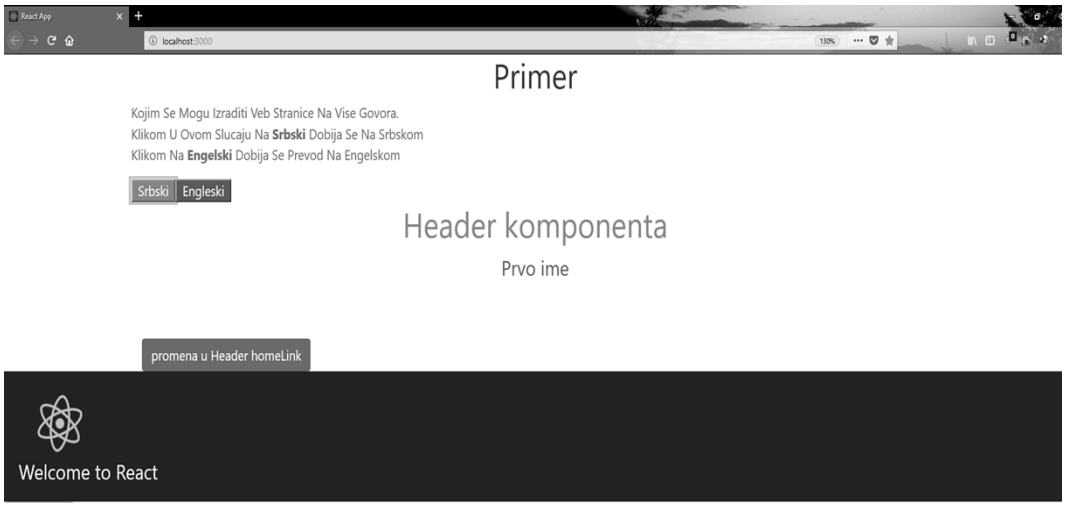
Ovakvim pristupom mogu se razmeniti podaci kroz više komponenti po dubini. U sledećim primeru može se videti takođe rad razmene vrednosti stanja i props-a.

```
import React,{Component} from 'react';

export default class Primer extends React.Component{
   constructor(props){
      super(props);
      this.state={
         homeLink : "Prvo ime ",
         data1 :"Prvo ime",
         data2 : "First name"
```

```
        };
    }
    vracanjeNaStaro() {
        this.setState({
            homeLink: this.state.data1
        });
    }
    vracanjeNaEngleski(){
        this.setState({
            homeLink: this.state.data2
        });
    }
    onChangeLinkName (newName){
        this.setState({
            homeLink : newName
        });
    }
    render(){

        return(
            <div className="container">
                <h1 className="text-center" style={{color:"red"}}> Primer</h1>
                <p className="text-justify text-capitalize text-primary">
                    Kojim se mogu izraditi veb stranice na vise govora.<br/>
                    Klikom u ovom slucaju na <strong>Srbski</strong>
                    dobija se na Srbskom<br/>
                    Klikom na <strong>Engelski</strong> dobija se prevod na
                    Engelskom <br/>
                </p>
                <button className="btn-success"
                onClick={this.vracanjeNaStaro.bind(this)}>
                Srbski</button>
                <button className="btn-danger"
                onClick={this.vracanjeNaEngleski.bind(this)}>
                 Engleski</button>
                <Header homeLink={this.state.homeLink} />
                <Home changeLink={this.onChangeLinkName.bind(this)} />
            </div>
        );
    }
}

class Home extends React.Component{
    constructor(props){
```

```
        super(props);
        this.state={
           homeLink : "Promenjeno ime "
        };
    }
    onChangeLink (){
        this.props.changeLink( this.state.homeLink );
    };
    render(){
        return(
            <div className="container" style={{left:"100px"}}><br/><br/>
                <button className="btn btn-primary"
                    onClick ={ this.onChangeLink.bind(this)}>
                    promena u Header homeLink
                </button>
            </div>
        );
    }
}
const Header = (props) => {
    return(
        <div>
            <h1 className="text-center text-info"> Header komponenta</h1>
            <p className="text-danger text-center" style={{fontSize:"24px"}}>
{props.homeLink}</p>
        </div>
    );
};
```

Na ekranu daje ovu sliku



Glavna razlika između state i props je da su props samo za čitanje. Zato je komponentu kontejnera potrebno ažurirati i mijenjati, dok bi komponente deteta trebale samo pročitati podatke iz state pomoću props.

Upotreba Props

Kada su potrebni nepromjenjivi podaci u vašoj komponenti, možete samo dodati props za reactDOM.render() funkciju u main.js i koristiti je unutar vaše komponente.

App.jsx

```
import React from 'react';
class App extends React.Component {
  render() {
    return (
      <div>
        <h1>{this.props.headerProp}</h1>
        <h2>{this.props.contentProp}</h2>
      </div>
    );
  }
}

export default App;
```

main.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';

ReactDOM.render(<App headerProp = "Header from props..." contentProp =
"Content
  from props..."/>, document.getElementById('app'));
```

Napomena: još jedan način prosleđivanja podataka između komponenti. Prosleđivanje vrednosti props-a u slučajavima upotrebe rutera i params objekta može biti i ovakvo. Početni kod

```
import React from ‘react’;

export default class Neka klasa extends React.Component{

render(){
      return(
```

```
            <h2> {this.props.params.deo}</h2>
        );
    }
}
```

može se napisati i ovako i kod ce raditi potpuno ispravno>

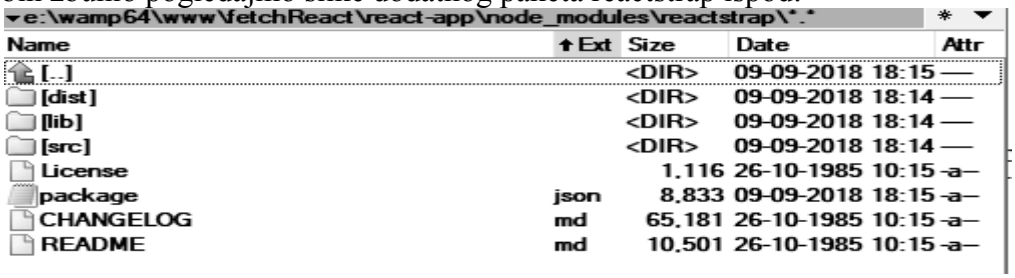
```
import React from 'react';

export default class Neka klasa extends React.Component{

render(){
            let {params}= this.props;
            let {deo} = params;
       return(
            <h2> ({deo})</h2>
        );
    }
}
```

Što znači da se dobija jasniji i čitkiji kod, koji daje jednostavniju primenu u praksi.

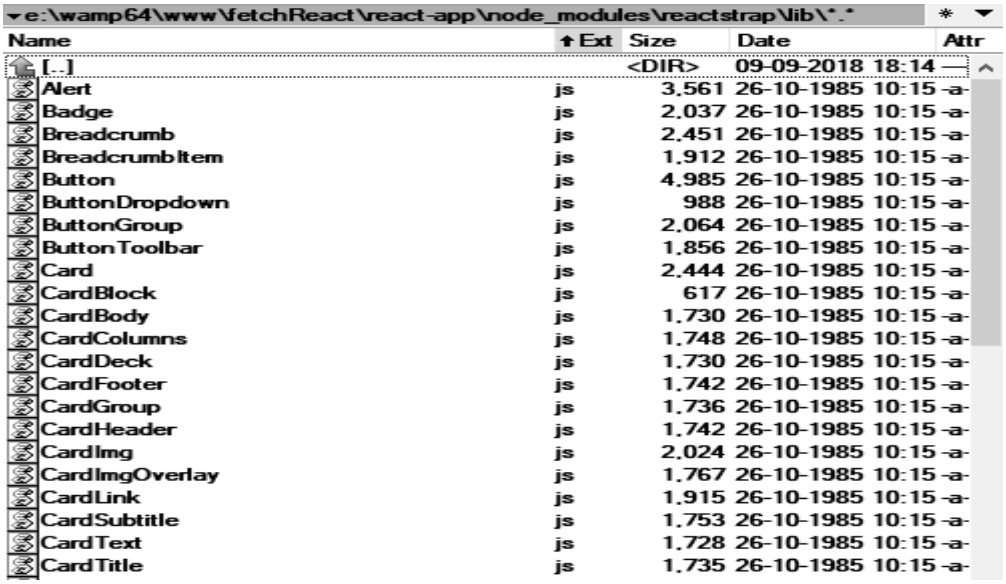
React Props primer

Također možete postaviti zadane vrednosti svojstava u konstruktoru komponente umesto da ga dodate ReactDom.render() element.

Napomena: Da bi ste videli šta koji dodati paket reacta sadrži, odnosno šta bi ste iz njega mogli da primenite u aplikaciji potrebno je pristupiti folderu sa tim instalisanim paketom koji se nalazi u folderu node_modiles. Pročtati njegov readme.md fail i pogledati u foldere koji su u folderu dodatnog paketa. Da vas nebih zbunio pogledajmo slike dodatnog paketa reactstrap ispod:



Surs (src) i lib folderi sadrže ovakve failove u sebi



kojima pristupate destruktivnom metodom izdvajajući ih kao pojedinačne prilikom importa iza čega upotrebljavate velike zagrade između kojih stoji ime onog faila koji vam je potreban u budućoj aplikaciji. Iza čega stoji reč from i u jednostrukim navodnicima ime dodatog paketa iz koga se uzima ovo u velikim zagradama. Ovako:

import { Button} from ‘reactstrap’;

Naravno možete ih uzeti više ako vam je to potrebno, s time što onda u velikim zagradama ih odvajate znakom zareza:

import { Button, Card, Navbar, ...} from ‘reactstrap’;

Fail App.jsx kojim je prikazan rad props-a

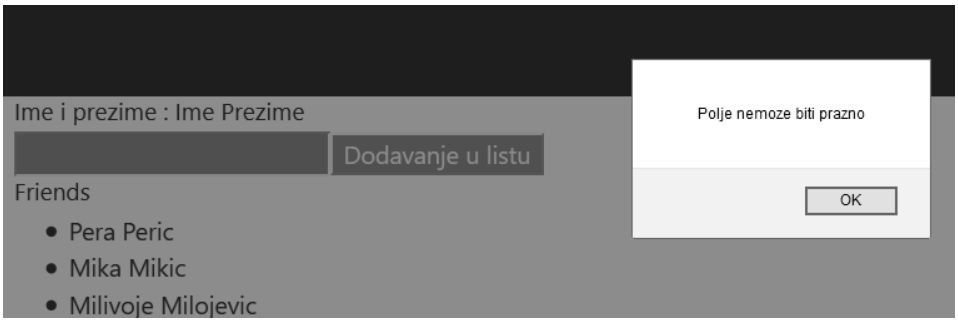
```
import React from 'react';

class App extends React.Component {
  render() {
    return (
      <div>
        <h1>{this.props.headerProp}</h1>
        <h2>{this.props.contentProp}</h2>
      </div>
    );
  }
}
```

```
App.defaultProps = {
  headerProp: "Header from props...",
  contentProp:"Content from props..."
}

export default App;
```

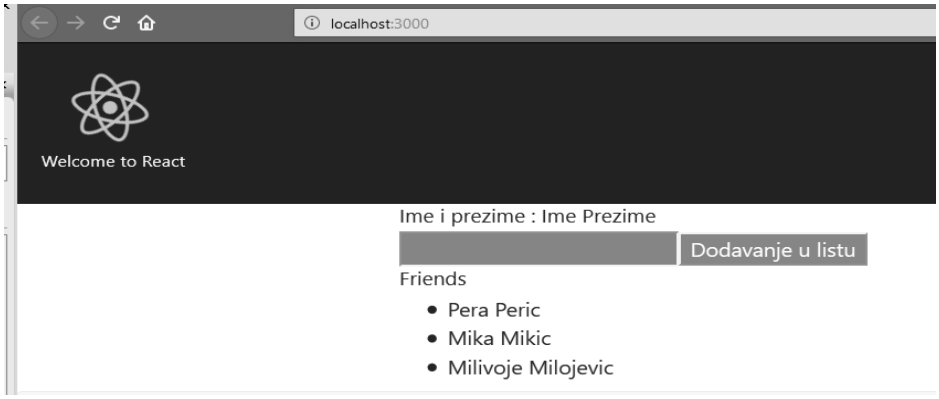
Slika prikazuje rad koda

Dodavanje podataka u listu

Ponekada je potrebno naraviti listu mogućih korisnika ili neku drugu listu u kojoj se upisuju podaci. Ali je potrebno i da postoji verifikacija unetih podataka, odnosno provera da li je pokušano da se unese prazan podatak. Provera unosa podataka vrši se prvo kod korisnika, a kasnije se još jednom vrši na serveru pre upisa unetih podataka u bazu podataka. Pored ove manje provere u primeru moguće je upotrebiti regularne izraze (regexp) sa if petljom ili switch case, gde se i postavljaju poruke ako ima nekih problema. U ovom slučaju postavljeno je da ako je unosno polje prazno i desio se događaj onClick pojaviće se Alert



Kojim ćemo upozoriti korisnika da je unosno polje prazno, u suprotnom unos biće dodat u niz podataka koji će biti prikazani na ekranu.

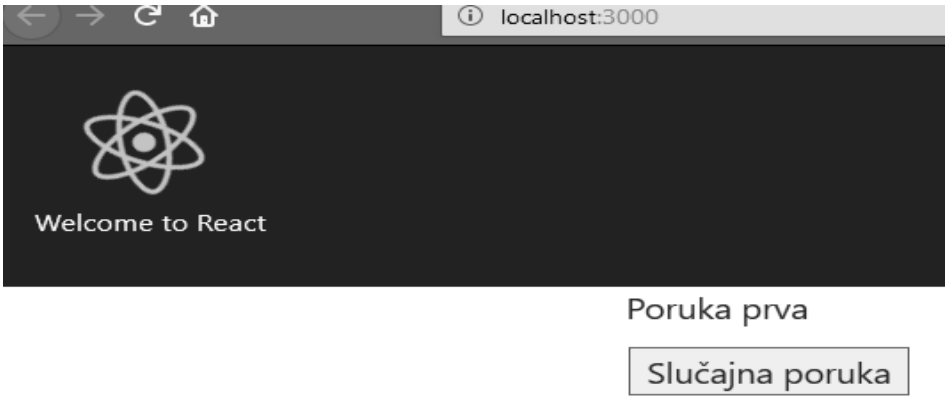



```
import React from 'react';

export default class FriendsContainer  extends React.Component{
   constructor(){
      super();
      this.state= {
         name: 'Ime Prezime',
         friends: [
            'Pera Peric',
            'Mika Mikic',
            'Milivoje Milojevic'
         ]
      };
      this.addFriend = this.addFriend.bind(this);
   }
   addFriend(friend){
      this.setState({
        // concat vrši pridruživanje novog podatka u niz predhodnih
         friends: this.state.friends.concat([friend])
      });
   }
   render(){
      return (
         <div className="container">
            <h3> Ime i prezime : {this.state.name} </h3>
            <AddFriend addNew={this.addFriend} />
            <ShowList names={this.state.friends} />
         </div>
      )
   }
};
```

```
class AddFriend extends React.Component{
  constructor(){
    super();
    this.state= {
      newFriend: ''
    };
    this.handleAddNew = this.handleAddNew.bind(this);
    this.updateNewFriend = this.updateNewFriend.bind(this);
  }

  updateNewFriend (e){
    this.setState({
      newFriend: e.target.value
    });
  }
  handleAddNew(){
    if( this.state.newFriend !==''){
      this.props.addNew(this.state.newFriend);
      this.setState({
        newFriend: ''
      });
    }else {
      alert("Polje nemože biti prazno");
    }
  }
  render(){
    return (
      <div>
        <input
          type="text"
          value={this.state.newFriend}
          onChange={this.updateNewFriend}
          className="badge-success"
        />
        <button onClick={this.handleAddNew} className="badge-info">
         Dodavanje u listu
        </button>
      </div>
    );
  }
}

class ShowList extends React.Component{
  render(){
```

```
        let listItems = this.props.names.map(function(friend){
            return <li key={friend}> {friend} </li>;
        });
        return (
            <div>
                <h3> Friends </h3>
                <ul>
                    {listItems}
                </ul>
            </div>
        )
    }
}
```

Generisanje poruka



Poruke se mogu korisnicima pojavljivati u različitim situacijama, kojima ih obaveštavamo o ne samo nekom problemu već i da im je neko posalo poruku, neka čestitka i drugo. Kod ispod generiše poruke iz niza poruka. Radi veoma jendostavno po pristupanju aplikaciji pojavlju je se poruka iz stanja u konstruktoru. Ali, ona može i da ne postoji, to jest da je upotrebljeno prazno stanje, i da se pojavi tek kada kliknemo na dugme ili bilo koji drugi deo na stranici kojim bi bila pozvana događajem onClick. A naziv slučajan poruka je proizašao iz razloga što je za aktiviranje poruka upotrebljen računski izraz Math.floor((Math.random() * 3)) kojim se vrše izračunavanja koja će poruka kada se prikazati, tako da je ovo primer pogodan za izradu nekih delova video igrica.

```
import React from 'react';

export default class RandomMessage extends React.Component{
    constructor(){
        super();
```

```
        this.state={
            message: 'Nulta poruka iz stanja'
        };
        this.onClick = this.onClick.bind(this);
    }

    onClick() {
        let messages = ['Poruka prva', 'Poruka druga ', 'Poruka treca'];
        let randomMessage = messages[Math.floor((Math.random() * 3))];

        this.setState({ message: randomMessage });
    }
    render() {
        return (
            <div>
                <MessageView message={ this.state.message }/>
                <p><input type="button" onClick={ this.onClick }
                  value="Slučajna poruka"/>
                </p>
            </div>
        );
    }
}

class MessageView extends React.Component{
    render() {
        return (
            <p>{ this.props.message }</p>
        );
    }
};
```

Kako napraviti aplikaciju u React okruženju

U ovu svrhu postoje dva načina: sve ručno i automatski. Oba načina dovode do istog rezultata. Možda nešto poput ovog dole niže na slici. Sam rad se svodi na izradu projektnog foldera u kome radite aplikaciju tako što prvu inicijalizaciju svega pokrećete u zavisnosti da li upotrebljavate editor kao što je Storm ili InteliJ idea ili upotrebljavate Notepad++. Kod prva dva potrebno je izvršiti podešavanja rada, ali je jednostavnije izraditi i projekt i aplikaciju, dok u drugom slučaju (NotePad++) biće vam potrebno da pokrente u vidovsu konzolu (CMD) i da preko nje dođete do lokacije projekta. Dok kod ispred pomenutih editora sve to radite u njima i nije vam potrebno da pokrećete terminal ili konzolu. Ono što je najbitnije i jednako za sve te varijante je da je potrebno u tom projektnom folderu uneti kod:

npm install create-react-app –g

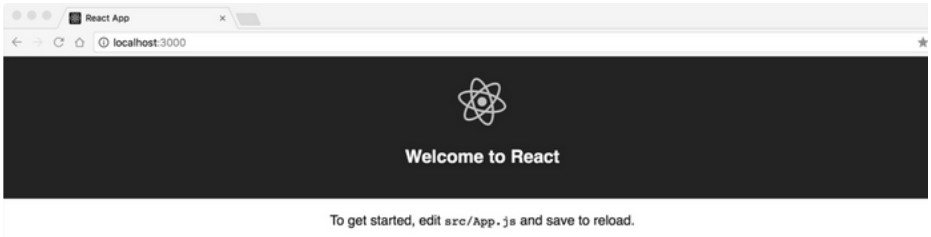
Nakon cega unosite

Create-react-app naziv-vase-aplikacije

Npm install create-react-app –g vrši instalaciju dodatka Nodu koja omogućava nešto poput hladne inicijalizacije jer se vrši njeno globalno instalisaranje u sistemu. Dok, create-react-app naziv-vaše-aplikacije vrši celokupnu pripremu i izradu osnovne React aplikacije. Tako da na kraju svega na ekranu ugledaćete nekoliko redova sa komandama:

Prelazak u folder aplikacije cd ime-vaše-aplikacije

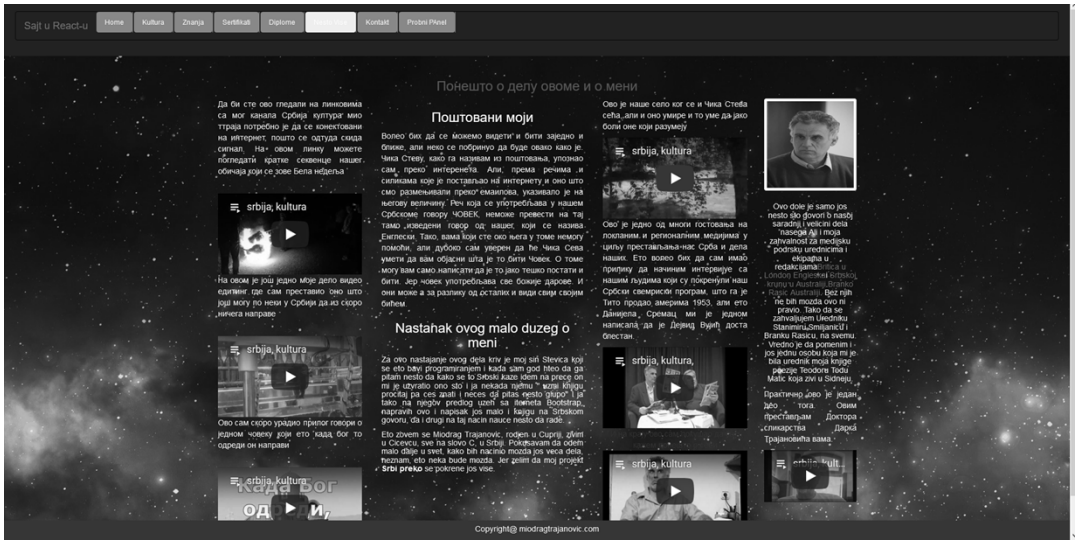
pokretanje aplikacije i njen prikaz u nekom od pretraživača kojei imate instalisanog u vašem računaru npm start (slika dole prikazuje izgled ovako dobijene aplikacije u brovseru)



izradu – bildovanje gotove aplikacije npm run build, nakon ovog ako nemate instalisan Wamp, Xamp i slično biće vam ponuđena instalacija paketa koji se zove serve sa komandom serve build, koju možete dodati i odredišno port na serveru sa ovim dodatkom koda -p80, koji je određen brojem iza minus slova p

Napomena: standardno je podešavanje prikazivanje aplikacije u pretraživaču u toku njenog razvoja na portu 3000 (http://localhost:3000/), ali se to može podesiti proizvoljno.

Dole na slici je jedan od već urađenih veb sajtova u pomenutim tehnologijama kao SPA(Single Page Application)



Kako to izgleda sa druge strane

Veliki broj koraka prilikom izrade aplikacija React-om zahteva prisustvo JSON failova, tako da praktično prilikom same izrade se pored svih neophodnih failova izrađuje fail package.json, dok sama struktura foldera i failova aplikacije ima ovakav izgled:

```
Project folder
        React-app folder
                Public folder
                    favicon.ico // mala ikonica na tabu
                    index.html // indeksni fail koji dobija podatke koje nam
                                // prikazuje u brovseru
                    manifest.json
                 src folder
                    App.js //  osnovni fail u kome vršimo povezivanje
                            //  ostalih failova
                    Index.js // fail koji šalje podatke u public/index.html
                    App.css // failovi za podešavanje izgleda i pozicije i sl.
                             // elemenata stranice
                    Index.css
            package.json       // fail za smeštanje osnovnih podataka i zapis
                               // instalisanih paketa
            package-lock.json // podaci o instalsainim paketima i sl.
```

Napomena: bez prisustva faila package.json nemoguće je pokrenuti aplikaciju u razvojnom delu.

Package.json
Fail u kome se nalaze različiti podaci o autoru, aplikaciji, instalisanim paketima. Jedan od mogućeg sadržaja koji ću objasniti radi lakšeg razumevanja namene i potrebe ovog faila može biti ovakav

```
{
 "name": "reactbs", // ime aplikacije
 "version": "0.1.0",   // verzija
 "dependencies": {     // zavisni failovi koji ulaze u sklop gotove aplikacije
  "axios": "^0.18.0",
  "babel-preset-php": "^1.2.0",
  "express": "^4.16.3",
  "fs": "0.0.1-security",
  "path": "^0.12.7",
  "qs": "^6.5.2",
  "react": "^16.2.0",
  "react-dom": "^16.2.0",
  "react-responsive-carousel": "^3.1.30",
  "react-scripts": "1.0.17",
  "reactstrap": "^5.0.0-alpha.4",
  "whatwg-fetch": "^2.0.4"
 },
 "scripts": {   // skriptovi koji su namenjni pokretanju u razvojnom načinu,
                //  bildovanju, test i odbacianja
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test --env=jsdom",
  "eject": "react-scripts eject"
 },
 "devDependencies": { // paketi koji su namenjeni za rad u razvojnom delu
                      // aplikacije
  "bootstrap": "^4.0.0-beta",
  "jquery": "^3.2.1",
  "react-bootstrap": "^0.31.5"
 },
 "proxy": "http://localhost:80" // dozvola pristupa i preko odredjenih portova
                                // aplikaciji
}
```

Kao sve drugo u React-u izrada ovog faila je moguća ručno i automacki. Prvi nacin je kada pristupite folderu gde nameravate izrađivati aplikaciju i uneste komandu:

npm init

Koja praktično kroz niz pitanja formira osnovni sardžaj ovog faila i na kraju vam ga na prozoru terminal prikaže i upita vas za saglasnost da to snimi ili ne. Ili malo automatizacije sa npm init –y. Gde se unosi osnovni sadržaj package.json faila i gde posle morate da ručno unesete potrebne parametre.
Drugi način je već pomenut upotrebom :

create-react-app naziv-vaše-aplikacije

Koja izrađuje sve sto je po defoltu potrebno za siguran rad aplikacije, pa i ovaj fail. Ono što je važno za ovaj fail je gde se šta dodaje i kako za rad vaše aplikacije. Kao što ste videli package.json se sastoji od objekata koji imaju ključeva i vrednosti, gde su objekti podeljeni u grupe. Taj osnovni sadržaj prikazuje kod niže:

```
{
  "name": "reactbs", // ime aplikacije
  "version": "0.1.0",   // verzija
  "private": "true",
  "dependencies": {     // zavisni failovi koji ulaze u sklop gotove aplikacije
    "react": "^16.2.0",
    "react-dom": "^16.2.0",
    "react-scripts": "1.0.17",
   },
  "scripts": {   // skriptovi koji su namenjni pokretanju u razvojnom načinu,
                 // bildovanju, test i odbacianja
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test --env=jsdom",
    "eject": "react-scripts eject"
  }
}
```

Dodavanje potrebnih paketa i njihov upis u ovaj fail vrši se automacki ovim naredbom:

npm install - -save ime-potrebnog-paketa // ovo dodaje podatke o paketu u deo dependencies (može se

napisati i skraćeno

npm i -S ime-potrebnog-paketa

Dodavanje paketa potrebnih za razvojni deo ili dok se aplikacija izrađuje i testira Devdependencies vrši se ovom komandom

npm install - -save-dev ime-potrebnog-paketa. Skraćeno npm i S D ime-potrebnog-paketa. Ove skraćenice komandi upravljača peketima možete videti na: https://nodejs.org/api/cli.html
ili ako upotrebite node –h u terminalu. Ali, ovom failu možete dodati sve ono što smatrate da je potrebno da on sadrži, ali u JSON formatu. Kod većih aplikacija koje su predviđene da rade ne kao jednostranicne aplikacije (SPA) već kao serverske i korisničke (SSP), potrebno je dodati Babel, webpack i druge. Razlog ovome je taj što mora da se sačini jedan fail koji se prosleđuje index.html failu i pored toga da se prevede u kod java skripta koji razumeju internet pretraživači, obzirom da je sav kod u Reactu mešavina kodova koja se naziva JSX ili ES6. ECMAScript 6, mada je u objavi i vezija ES7.

Src/App.js fail

Ovaj fail je namenjen da se u njega slivaju svi linkovi od ostalih failova komponenti koje izrađujete u aplikaciji. Filozofija Reacta je da se umesto dugačkih glomaznih kodova pisanih u mnogim programskim govorima, nebitno interpreterima ili kompajlerskim, koji izrađuju izvršne aplikacije koje rade samostalno, upotrebi rasčlanjivanje na sitne detalje, čime praktično pojava grešaka je nemoguća.
Da bih ovo malo pojasnio pre nego što počnem sa prikazom i objašnjenjem App.js faila, objasniću jedan mali primer ispred.
Tabela ima svoje delove: redove, kolone, zaglavlja, telo, gde su svi oni delovi tabele. Znači možemo sve to da sačinimo kao posebne komponete jedne komponente koja se zove tabela i koju kasnije samo pozovemo gde nam je potrebno u aplikaciji i tamo je popunimo potrebnim podacima. Ovaj pristup na omogućava da jedan isti kod pozovemo svuda gde nam je to potrebno u aplikaciji. Duzina koda svakog dela je kratka tako da je pojava grešaka svedena na beznačajnu meru. Čitaj nemoguće.

Napomena: Za ovakav rad potrebno je napraviti organizaciju foldera, tako da u folderu komponenti svaki od njih ima svoj folder i u njemu podfoldere za njene delove. Razlog ovome je očigledan jer svakom delu komponete u njenom podfolderu možete napravi njen css fail čime ste i te failove smanjili na manje redova i olakšali rad i sa njima.

Kod faila App.js

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  render() {
```

```
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h1 className="App-title">Welcome to React</h1>
        </header>
        <div className="App-intro">
      </div>
      </div>
    );
  }
}

export default App;
```

Sadržaj koda ovog fail može biti forma za izradu svih ostalih komponenti, mada, u daljem delu knjige biće prikazani i objašnjeni i drugi modeli koji mogu da posluže u istu svrhu.
Da bi radili sa komponentama u ovakvim slučajevima gde možemo da postavimo i početne vrednosti stanja komponenti uvek započinjemo sa importovanjem React-a. Tačnije u ovom slučaju putem importovanja Reacta mi vršimo proširenje klase App sa jednom ekspotrovanom funkcijom Component koja je u sastavu React bibiloteke. Sa kojom praktično izrađujemo virtualni dokument, kojim je mnogo lakše manipulisati.
Sam pojam klase je nesto što se može sresti u mnogim programskim govorima, gde praktično se opisuje neki izgled nečega, što je i namena klasa.
U ovom kodu kao i svuda u programiranju mogu se vrednosti konstanti i drugog dodati globalno za nju ili direktno samo za nju (class App). Ovo zato što možemo u ovom failu dodati i druge klase, constante, promenjive van koda ove klase, ali samo je ona izlazna iz ovog faila, jer ima proširenje na svom kraju

```
export default App;
```

Ovaj deo koda može da se nađe i ovako napisan

```
export default  class App extends Component {

  render() {
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h1 className="App-title">Welcome to React</h1>
        </header>
```

```
      <div className="App-intro">
    </div>
    </div>
  );
 }
}
```

Što je potpuno jednako predhodnom načinu pisanja, ali daje malo kraći kod. U delu za importovanje možemo importovati sve što je potrebno da ta komponenta ima. Sliku, tekst, proširenja drugim klasama, funkcijama, konstantama. Ono što je važno u tome da vodimo računa gde se koji fail kog importujemo nalazi u razvojnom delu aplikacije u odnosu na ono gde ga pozivamo. Na primer

```
import logo from './logo.svg';
```

Objašnjenje koda bi bilo importujem – unosim ovo što ću u ovom failu nazvati logo sa mesta koje je u istom folderu gde i kod komponente koja ga poziva i to se zove logo.svg. Drigim rečima ovo što smo importovali nalazi se u istom folderu gde i sama komponenta App.js, to je slika i zove se logo.svg. Dok, mi u samoj komponenti ga nazivamo samo logo. Pošto se radi o slici reč logo smeštamo u izvor podataka za sliku

```
<img src={logo} className="App-logo" alt="logo" />
```

Obzirom da se radi o java skript kodu umetnutom u html kod ovu reč logo moramo postaviti između velikih zagrada. Ova sintaksa se isto upotrebljava i za pozivanje kostanti i promenjivih i slično, kada ih pozivamo kao i ovu sliku u html kod komponente.

Ono što još možemo videti ovde je pozivanje css klase, sa className, što je obavezni način upotrebe. U suprotnom sam React javiće nam grešku, jer on upotrebljava class za oznaku java script klase, pa će shvatiti da se radi o tome. Sledeće je to da je obavezno da svaka slika sadrži alt atribut, kao to da se pre kraja taga img uvek mora naći kosa crta (znak podeljeno). Ovaj znak je takođe obavezan kod taga input, ali i kod svih onih html tagova gde nema dodele nekih vrednosti između otvorenog taga i zatvorenog istog taga. Jer se između otvorenog i zatvorenog taga računa za child deo. Što se u reactu naziva sadržajem child, jer za pokretanje tog sadržaja u funkciju se upotrebljava prefiks children (this.props.children).
Za više o primeni children na vebsajtu https://mxstbr.blog/2017/02/react-children-deepdive/

Primer:
napisano u Html dokumentu bilo bi ovako
```
<span atribut=”vrednost” >  <i class="fas fa-address-book"></i> </span>
```

Dok u Reactu bi ovo prijavilo grešku, te bi trebalo da se napiše ovakav kod

 <i class="fas fa-address-book"/>

ili

Ovo pravilo važi za sve html tagove gde se između njih nema potrebe da se dodaju neke vrednosti. Pored toga potrebno je voditi računa prilikom pisanja koda o tome koji je html tag sa većom vrednošću a koji je sa nizom, te one sa nizom zatvarati onima sa višom vrednošću.

```
<div className="App">
  <header className="App-header">
    <img src={logo} className="App-logo" alt="logo" />
    <h1 className="App-title">Welcome to React</h1>
  </header>
  <div className="App-intro">
</div>
</div>
```

Html tag ima višu vrednost te on može zatvoriti sve ostale,osim html, header, title, body, kao i tag <form>, što zatvara sve svoje niže tagove, <nav>, <ul> i tome slično. Što se vidi u gornjem kodu. te je time stavljeno do znanja gde šta može da se nađe u napisanom kodu.

Raspored koda u klasi

Donji kod pokazuje kako treba da izgleda sam kod klase komponente, ali i u njemu je određeno gde šta može da se od ostalih delova koda klase komponente može napisati.

```
export default  class App extends Component {

  render() {
    return (
      <div className="App">

      </div>
    );
  }
}
```

U delu iza velike zagrade posle reči Component pa dole prema reči render se upisuje kod za konstruktor ako su nam potrebani na primer početne vrednosti stanja u delovima komponente.

```
 constrctor(props){
   super(props);
   this.state={
    stanje-kom1: ‘njegova pocetna vrednost’,
    stanje-kom2: ‘’,
    stanje-kom3: {},
    stanje-kom3: [],
   }

   // Na ovom mestu upisujemo pozivanje funkcija koje upotrebljavamo u izradi
   // komponente
    this.NekaFunkcija = this.NekaFunkcija.bind(this);
}
```

U ovom delu iza zatvorene velike zagrade konstruktora pišu se tela funkcija i komponenti koje pozivamo u različitim namenama (na primer poziv nekog događaja).
Napomena: Funkcije koje pozivate i registrujete u delu konstruktora su funkcije koje imate u istoj komponenti gde su one u telu klase. Sve druge funkcije koje nisu u delu ispod do dela poziva povratnih informacija, obicno su smeštene promenjive, constante

```
 render(){
          konstante /promnenjive
          return(

                    );
        }
```

U delu između malih zagrada iza return-a piše kod java skripta, html, css-a između najčešće html

```
     <div >

     </div>
```

tagova. U cilju pisanja stila nekog dela primenjuje se ovakav kod:

```
 style={{ color: “ red”, boder:” 1px solid yellow, width:”100%”}}
```

Ili, ako je potrebno prikazati rezultat rada neke konstante ili promenjive, čija je vrednost je novi dobijeni niz, njeno ime se upisuje između velikih zagrada

```
<ul>
 {
        let ime-varijabile ili const ime-konstante = podaci.map( deo =>{
        return <li key={deo.id}> {deo.name} </li>});
 }
 </ul>
```

Što prestavlja način uključivanja java skripa u html ili xtml, xml. Kao što vidite <ul> tag je stariji od taga <li>, ono što je već napomenuto ispred. Ovo isto može biti primenjeno i na ovaj način da ceo kod koji je između velikih zagrada zatvoren <ul> tagovima, može da se napiše u delu ispred render i return, a da se samo ime-varijabile ili ime-konstante unese između velikih zagrada koje su zatvorene <ul> tagovima. Ovako:

```
<ul>
 {
        ime-varijabile ili const ime-konstante

 }
 </ul>
```

Rezultat bi bio potpuno jednak. Što čini još jednu prednost koju pruža okruženje React-Node.

Fail index.js

Uobičajeno da ovaj fail prihvata ono što šalje fail App.js o vrši renderovanje tog sadržaja kog prosleđuje elementu koji ima za svoj ID ono što navedemo u njemu. U ovom slučaju kao što se vidi u poslednjem redu koda fail index.js to je element sa ID-om root. Što znači da je on jedina veza sa stvarnim Dom elementom.

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';

ReactDOM.render(<App />, document.getElementById('root'));
```

Ovaj element se obavezno nalazi u failu public/index.html i tu se praktično prikazuje sve što smo napravili u celoj aplikaciji. Sam React ume da radi sa virtuelnim DOM elementima, te mu ovaj fail prestavlja neku vrstu mosta izmedju njega i stvarnosti.

Ono što još je bitno za ovaj fail, je to da se u njemu postavlja i veza prema css failu instalisanog Bootstrapa u Reactu.

import 'bootstrap/dist/css/bootstrap.min.css';

S time što je potrebno predhodno instalisati Bootstrap npm install - - save bootstrap, čime dobijate Bootstrap u celoj React aplikaciji što se tiče css podataka. Da bi radio i jquery.js i bootstrap.js potrebno je dodati CDN linkove u failu public/index.html. Drugi nacin je preuzeti sa službenog sajta Bootstrapa kompresovanu razvojnu verziju Bootstrapa i prekopirati u public folder, dist folder iz preuzetog paketa i onda postaviti linkove prema potrebnim failovima Bootstrapa u failu public/index.html. Ali o tome će biti reči više u delu Povezivanje React aplikacije sa Bootstrapom.

Ono što je još bitno za failove App.js i index.js je to da se u njima postavlja kod za dodatak Reacta koji radi posao rutiniranja stranicama sa svojim delovima: react-router i react-router-dom. Instalcija ovih delova paketa vrši se unosom ovog koda u konzolu ili dr. u zavisnosti od vaših mogućnosti:

npm i - - save   react-router react-router-dom

Ono što se još postavlja automackim izradom React aplikacije, je ovo

import registerServiceWorker from './registerServiceWorker';

registerServiceWorker();

Više o srvis workeru

Service Workers, MDN
https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API

Service Workers, Google Web Fundamentals
https://developers.google.com/web/fundamentals/primers/service-workers/

Simple offline site using service worker, CSS Tricks
https://css-tricks.com/serviceworker-for-offline/

Server worker: A case study, Smashing Magazine
https://www.smashingmagazine.com/2016/02/making-a-service-worker/
Index.html

<!DOCTYPE html>
<html lang="en">

```
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-
to-fit=no">
    <meta name="theme-color" content="#000000">
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json">
    <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico">
    <title>React App</title>
  </head>
  <body>
    <noscript>
      You need to enable JavaScript to run this app.
    </noscript>
    <div id="root"></div>
    <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" >
    </script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js" >
    </script>
    <script
      src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js" >
    </script>
  </body>
</html>
```

Inicijalno stanje komponenti

Da bi se moglo menjati neko stanje postavljenih promenjivih u react komponentama, potrebno je odrediti prvo njeno počento stanje. To inicajalno snanje koristi se kao početno, dok akcijama koje su zahtevom postavljene kroz upotrebu AJAX, AXIOS i sličnih paketa koji prave određene događaje na veb stranicama react komponenta može primiti novo stanje koje bi tim akcijama ili događajima promenilo prikazano stanje na veb stranici.
U sledećih nekoliko koraka navedeno biće objašnjeno

Korak 1: Naravno potrebno je prvo izraditi aplikaciju, primenjujući zbog jednostavnosti automatizovan način izrade react aplikacija unosom donjih kodova u kozolu:

npm install -g create-react-app // globalno pozivanje procesa za izradu react aplikacija – hladana inicijalizacija

create-react-app my-app // pokretanje izrade react apliakcije. Umesto my-app unesite ime svoje aplikacije

cd my-app // pošto se završio predhodni korak ovo je prelazak u folder aplikacije
npm start // pošto smo u folderu aplikacije ovo unosimo da bi je pokrenuli

Ovako je uvek bolje raditi u početku učenja, jer je na ovaj način svešto je u osnovi potrebno za rada aplikacije u njoj i instalisano. Naravno, ako su vam potrebni drugi paketi njih instalište posebno. Nakon svih provera startni skriptovi koji su napravil iza vas react aplikaciju pokrenuće i brovzer, gde će na localhostu i portu 3000 prikazti vašu osnovnu aplikaicju.
Ono što je još bitno u ovom slučaju nije potrebno osvežavati stranicu u brovzeru, jer je i to automatizovano nakon svake promene koda, pod uslovim da imate neki ozbiljniji editor koda, jer u suprotnom moraće te promene prvo u njemu sačuvati. Čim ih sačuvate, istovremeno prikaz u brovzeru te će se promene i prikazati.

Korak 2: Pristupite putem editora koda aplikaciji u folderu src gde se nalazi fail App.js i promenite kod koji tamo pronađete u ova dole niže, kako bi ste i postavili prvo inicijalno stanje vrednosti:

```
// App.js

import React, { Component } from 'react';

class App extends Component {

// postavljanje inicijalnog stanja
  state = {
    name: 'Inicijalno stanje komponente'
  };

  render() {
    return (
      <div className="App">

 {/* prikaz postavljenog stanja*/}
        {this.state.name}

      </div>
    );
  }
}

export default App;
```

Ovako postavljeno inicijalno stanje vrednosti se može menjati dodavanje nekog dugmeta kojim bi se putem HTTP zahteva ono moglo i menjati onim koje se

pročita iz baze podataka ili dodavanjem događaja koji bi pročitao novo stanje iz koda dole, gde je upotrebljen kod za postavljanje tog novog stanja

```
promena(){
          this.setState({
            name: 'React Tutorial'
          })
  }

     <button  onClick={this.promena.bind(this)}>
       Pozivanje promene
     </button>
```

Ovo je primer gde se ne upotrebljava konstruktor, dok u delu knjige koji opisuje konstruktor videćete slučaj kako izgleda kod kada se upotrebljava konstruktor

```
// App.js

import React, { Component } from 'react';

class App extends Component {
  constructor(props){
    super(props);
    this.state = {
      name: 'Neko početno stanje'
    };
  }
  render() {
    return (
      <div className="App">
        {this.state.name}
      </div>
    );
  }
}

export default App;
```

Najbolje je vršiti postavljanje početnog stanja kroz upotrebu konstruktora, a njegovu promenu kroz upotrebu postavljenja nove vrednosti sa thus.setState metodom.

Konstruktor

Konstruktor je deo komponente čiji se kod nalazi uobičajeno odmah ispod koda kojim započinje kod klase.
Ovo je deo koda u reakt aplikaciji koji ima dosta namena, osim za postavljanje početnog stanja u njemu se vrši i prijava, to jest, možemo vezati bilo koji događaj koji se pojavljuje u našoj komponenti u konstruktoru, omogućavanje poktretanja funkcija i sl, kao što je sljedeći.

```
 constructor(props){
    super(props);
    this.handleEvent = this.handleEvent.bind(this);
  }
```

U gornjem primeru događaj će se pokrenuti nakon što korisnik klikne dugme, ali samo povezivanje događaja se ne mora vršiti samo u konstruktoru već i direktno onamo gde je to potrebno.
Potrebno je postaviti kontekst roditelju (perent), prema kontekstu deteta (child).
Ovo je primer koji pojednostavljuje ovo objašnjenje.

```
// App.js

import React, { Component } from 'react';

class App extends Component {
  constructor(props){
    super(props);
  }
  handleEvent(){
    console.log(this.props);
  }
  render() {
    return (
      <div className="App">
        <button onClick={this.handleEvent}>Please Click</button>
      </div>
    );
  }
}

export default App;
```

U ovom primeru, kada se klikne na dugme događaj poziva funkciju

handleEvent ().
Međutim, kada vidite u preretraživaču, aplikacija prijavljuje ponekada grešku, kao ova dole niže

Dakle, ovde je problem da to više ne funkcioniše kao globalni objekt klase, to upućuje sve na taj način unutar te funkcije. Znači moramo ovo napraviti globalni pokazivač aplikacija na kojem možemo nazvati props i state vrednosti, te u cilju prevazilaženja ovog problema, napišite sljedeću i liniju u konstruktoru.

```
// App.js

import React, { Component } from 'react';

class App extends Component {
  constructor(props){
    super(props);
    this.handleEvent = this.handleEvent.bind(this); // prijava upotrebe
  }
  handleEvent(){
    console.log(this.props);
  }
  render() {
    return (
      <div className="App">
        <button onClick={this.handleEvent}>Please Click</button>
      </div>
    );
  }
}

export default App;
```

Bind () funkcija će uraditi ovaj posao, jer na polazištu naše komponente obvezujemo ovaj objekt globalnom elementu i ima sve vrednosti za props i state. Ali u reactu postoji još jedan način kojim se postiže isto uz vidno smanjenje dužine upotrebljenmog koda. To je upotreba strelaste funkcije(arrow).

```
// App.js

import React, { Component } from 'react';

class App extends Component {
  constructor(props){
    super(props);
  }
  handleEvent = () => {
    console.log(this.props);
  }
  render() {
    return (
      <div className="App">
        <button onClick={this.handleEvent}>Please Click</button>
      </div>
    );
  }
}

export default App;
```

Ova funkcija je nešto što je došlo sa pojavom standarada ES6, koji je postavio ispravno kontekst roditelja,

Destruktori

Pošto je React i celopkupno njegovo okruženje koda i sve ga ostalog vezanog za njega java skript, koji je potekao od programskih govora kao što je C, pacal i slični, normalno je da je nasledio pored konstruktora i destruktore. Nisu baš kao u pomenutim programskim govorima ali rade sličan posao. Tačnije oni razdvajaju podatke ili ih izdvajaju bolje rečeno iz mnogih. Jedan oblik upotrebe destruktora je i kod importovanja osnovnih i dodatih paketa u react aplikaciji.
import React, { Component } from 'react';

U ovom slučaju iz celokupnog paketa Reacta uzimamo samo skup koda koji sadržava kod koji opisuje komponentu.

Osim toga može se primeniti i u slučajevima pre nego što pokrenemo map metod

```
 render() {
  const { prop1, prop2 } = this.props
  const { state1, state2 } = this.state
  ...
}
```

U osnovi sama sintaksa destruktora izgleda ovako

```
 // Dodavanje props objektu izgleda ovako:
  // { foo: ‘foo’, bar: ‘bar’ }

// dok izdvajanje ili destrukcija
  const{ foo, bar  } = this.props;
```

Ako pogledamo ovaj kod i onaj pre njega videćemo da se ovo može primeniti i za vrednost stanja. Props i State To je sada, sada smo izdvojili dva svojstva unutar objekta i sada možemo koristiti varijable `foo` i` bar` gdje god je to potrebno. u react aplikacijama. Možda se ovo čini preterano složeno pristupanjem našim varijablama, kada bismo mogli samo upotrebiti "this.props.foo", ali, zapamtite, ovo je samo jednostavan primer. To je složeniji slučajevi korištenja u kojima destrukturiranje dolazi na svoje. Gde imamo pored ove dve i varijabile i jedan niz sa svojim elementima koji je u sastavu objekta.

```
//  sastav objekta { foo: ‘foo’, bar: { baz: ‘baz’ }, baz: ‘baz’ }

  const { foo, bar: {  baza }, baz: baz2 } = this.props;
```

To će dodeliti vrednost baze prvog nivoa ` baza` na novu konstatu nazvanu `baz2`. Budite pažljivi da biste dobili naziv varijable i preraspodelu na pravi način, premda - na prvi pogled, ako uporedite naš destrukturirani objekt s našim izvornim objektom, gornji primer može se činiti ispravan, ali to će dati tri konstante pod nazivom `foo`,` baza` i "baz2".

Na kraju pogledajmo jedan karakterističan primer koji se može dosta primeniti u budućem razvoju react aplikacija.

```
  function LinkComponent(props) {
    const {
      children,
      disabled
    } = this.props;

// ovaj red koda  ispod ima istu funkciju kao if else
    const statusClass = disabled ? ‘disabled’ : ‘active’;
```

```
  return (
    <a className={ statusClass }>
      { children}
    </a>
  );
}
```

Sada, budući da ovo samo izlaže vezu, verovatno ćete imati neke standardne props, npr. "Href", "onClick" ili " target" itd. Možete ih preneti tako da ih preuzmete u svoju komponentu i prenoseći ih do elementa veze, ali ovo je prilično dosadan način za to. To znači da trebate identifikovati sve moguće props koji se mogu preneti na određeni element koji šaljete.
Srećom, pomoću destrukturiranja, još jedan ES6 širenje sintakse, to možete učiniti na mnogo učinkovitiji i pouzdaniji način. Dakle, s istom komponentom kao i prije

```
function LinkComponent(props) {
  const {
    children,
    disabled,
    …drugo
  } = this.props;

  const statusClass = disabled ? ‘disabled’ : ‘active’;

  return (
    <a className={ statusClass } { …drugo }>
      { children}
    </a>
  );
}
```

Dodavanjem ovog koda sa tri tačke ispred omogućeno je da izvučemo i sve vrednosti odjednom koje su vezane za njega u objektu. Nakon toga možemo izvršiti bilo kakve operacije u našoj komponenti pomoću props koje smo definisali i preneti bilo koja druga svojstva, s vlastitim svojstvima koja se ručno uklanjaju u naš izvezni element. Na taj način, upotreba naše komponente u ovom slučaju, anchor element, a komponenta biblioteke ne treba da vodi računa o tome što oni mogu biti. Predstavlja fleksibilnost. U našem primeru komponente preporučljiva je praksa da koristimo ovu metodu. I zapamtite, sve props koje ne želite da pređu na element, ili da najpre sa njima želite manipulisati, jednostavno možete ekstrahovati (uzdvojiti ili upiti) pomoću primene destruktora u svojim react aplikacijama.
Sa druge strane gledano samim destruktorima se može poslužiti i na ovaj način:

Kod prikazuje upotrebu destruktora

```
class NekaKlasa extends Component {
  render() {
    const { modalList } = this.props;
    const { currIndex, showModal } = this.state;
    // jos neka dodatna radnja
  }
}
```

Kod gde nije primenjen destruktor:

```
class ChainedModals extends Component {
  render() {
    const modalList = this.props.modalList;
    const currIndex = this.state.currIndex;
    const showModal = this.state.showModal;
    // ..
  }
}
```

U oba navedena slučaja kod radi isti posao, omogućava pojavu modala na ekranu.

Napomena: da bi se nešto moglo priključiti u obliku detsruktor metoda iz neke druge komponente potrebno je da u svom sastavu koda ima službenu reč export:

export function ime() ili export.module ili export const ime

Jedan od primera objašnjenja i primene o destruktorima možete pogledati na veb sajtu

https://amido.com/blog/using-es6-destructuring-in-your-react-components/

Upotreba funkcija za razmenu podataka

Upotreba funkcija u reactu je potpuno ravnopravana sa svim drugim što uporebljavamo. Znači od jednostavnog poziva jedne funkcije destruktivnom metodom, preko kompleksnijih poziva odgovora iz funkcije u funkciji kao u ovom primeru ispod. Gde fail DrugaKomponenta ima u svom delu ne samo da prikaže putem poziva neki svoj rad već i da vrati props funkcije sendFunction.

```
import React from 'react';

export default class App extends React.Component {
  constructor(props) {
    super(props);
    this.functionToPass = this.functionToPass.bind(this);
  }
```

```
  functionToPass () {
    // Opis šta funkcija treba da uradi
  }
  render () {
    < DrugaKomponenta sendFunction = {this.functionToPass} />
    return (
    //something
    );
  }
}

class DrugaKomponenta extends React.Component {
  constructor(props) {
    super(props);
    this.functionPassed = this.functionPassed.bind(this);
  }
  functionPassed () {
    // Opis šta funkcija treba da uradi
    this.props.sendFunction();
  }
  render () {
    return (
      //šta je potrebno da se prikaže radom funkcije
    );
  }
}
```

Funkcija poziva funkciju

Često puta je potrebno pozvati neku funkciju u drugoj funkciji, jer jedna funkcija radi jedan deo posla druga drugi, nameće i potreba da rade zajedno. U tome slučaju bi ovaj primer pojasnio kako se to radi u slučaju kada su obe funkcije u istom failu. U konstruktoru se normalno prijavljuju obe

```
import React, { Component } from 'react';

class App extends Component {
   constructor(){
      super();
      this.state={
         podatakPrvi: "Popcetno satnje"
      };
      this.mouseClick = this.mouseClick.bind(this);
      this.prvaFun = this.prvaFun.bind(this);
      this.drugaFun = this.drugaFun.bind(this);
```

```
  }

  prvaFun(){
    console.log(' prvaFun radi');
    this.setState({podatakPrvi: " prvi podatak"})
  }

  drugaFun(){
    console.log(' drugaFun radi');
    this.prvaFun()
  }

  mouseClick(){
    this.drugaFun()
  }

  render() {
    return (
      <div >
        <h3>Funkcija poziva funkciju </h3>
        <h4>{this.state.podatakPrvi}</h4>
        <p>
          Da bi ste videli kako radi ovaj primer kliknite na dugme i u
          pretraživaču predjite u inspekt nacin rada a rezultat bice prikazan u
          konzoli
        </p>
        <button className="btn-primary" onClick={this.mouseClick}>
         Klikni
        </button>
      </div>
    );
  }
}
export default App;
```

Da bi se video neki rad prve funkcije dodao sam i promenu stanja promenjive podatakPrvi koja se dešava nakom klika na dugme koje je pokrenulo svojim pozivom drugu funkciju, a koja je pozvala prvu koja je unela svojim radom promenu pocetnog stanja promenjive podatakPrvi.

Mala modifikacija kojom se smanjuje dužina koda gde se direktno primenjuje bind(this) čime se nemora isto pisati u delu konstruktora

```
mouseClick = {  this.drugaFun.bind(this) }
```

ili uz primenu strelaste funkcije

```
mouseClick = { ()=> this.drugaFun () }
```

Dok ako bi bio slučaj da funkcija u jednom failu poziva nekim događajem funkciju iz drugog faila, onda je potrebno da ta koja se poziva u sebi sadrži export, a onda preko destruktivne metode iz tog drugog faila preuzeti samo tu funkciju koja potrebna. Na slične radnje u praksi naićićete kod rada u wordpressu, gde postoji jedan fail sa dosta funkcija pisan u php-u koji se importuje i onda se iz njega mogu povući funkcije bilo gde u aplikaciji. Drugo zašto je to bitno znati, je razlog da isto možete načiniti u svojim aplikacijama kako bi olakšali sebi posao.
Uzeću ovaj jednostavni primer gde se jedan broj uvećava za neki iznos

fail iz kog se eksportuje funkcija export.js

```
export const povećanje = (n)=>{
   Prikazi(0 += n);
}
```

fail gde vam je potrebna funkcija import.js

```
import { povećanje } from './export '

handleClick (event) {
      povećanje (1);
   }
```

osnovni model koda export.js faila je

```
export A(){
Neki rad funkcije
}

export B(){
Neki rad funkcije
}
```

Za ove namene može se primeniti i ovaj način rada, kada upotrebljavamo i module.exports
fail iz kog se eksportuje funkcija export.js

```
module.exports = {
  A: funtion(){
  },
```

```
  B: funtion(){
  }
 .......
}
```

fail gde vam je potrebna funkcija import.js

```
import {A} from ' ./export ';
```

Za pozivanje exportovanih funkcija može se upotrebiti zatev sa require, kao što je ovaj primer poziva

```
const functionName = require('./path_to_your_file');
```

na sajtu https://stackoverflow.com/questions/43046674/how-to-use-a-function-from-another-js-file-in-reactjs/43046752

Čitanje podataka iz JSON faila u reakt aplikaciji

Ovo je osnovni način pristupa podacima u failu JSON formata koji je u istom folderu kao i komponenta koja ga poziva.
Podaci koji su napisani u posebnom failu data.json su u obliku jednostavnog niza uređenih podataka ključeva i njihovih vrednosti. Ono što je bitno znati kod formiranja sadržaja ovih failova je to da se i ključ i njegova vrednosti pišu uokvireni znakom navoda, osim brojeva, ako se upoterbljavaju kao brojevi bez drugih slovnih znakova.

data.json

```
[{"id": 1,"title": "Child Bride"},
{"id": 2, "title": "Last Time I Committed Suicide, The"},
{"id": 3, "title": "Jerry Seinfeld: 'I'm Telling You for the Last Time'"},
{"id": 4, "title": "Youth Without Youth"},
{"id": 5, "title": "Happy Here and Now"},
{"id": 6, "title": "Wedding in Blood (Noces rouges, Les)"},
{"id": 7, "title": "Vampire in Venice (Nosferatu a Venezia) (Nosferatu in Venice)"},
{"id": 8, "title": "Monty Python's The Meaning of Life"},
{"id": 9, "title": "Awakening, The"},
{"id": 10, "title": "Trip, The"}]
```

React komponenta radi vrlo jednostavno, pristup se json podacima obavlja tako što se fail importuje u kod komponente i posle se putem primene map očitava deo po deo. U zavisnosti od potreba može se uraditi ovako kao u kodu gde su pročitane vrednosti smeštene između tagova neuređene liste <ul> i njenih delova <li>, ali se isto tako mogu smestiti u okvir tabele

fail dataApp.js

```
import React, { Component } from 'react';
import data from './data.json';

class App extends Component {
  render() {
    return (
        <ul>
        {
          data.map(function(movie){
            return <li key={movie.id} >{movie.id} - {movie.title}</li>;
          })
        }
        </ul>
    );
  }
}

export default App;
```

Podaci izgledaju ovako ispisani na ekranu

- 1 - Child Bride
- 2 - Last Time I Committed Suicide, The
- 3 - Jerry Seinfeld: 'I'm Telling You for the Last Time'
- 4 - Youth Without Youth
- 5 - Happy Here and Now
- 6 - Wedding in Blood (Noces rouges, Les)
- 7 - Vampire in Venice (Nosferatu a Venezia) (Nosferatu in Venice)
- 8 - Monty Python's The Meaning of Life
- 9 - Awakening, The
- 10 - Trip, The

Razlog je to što bi trebalo da se u list item

<li key={movie.id} >{movie.id} - {movie.title}

da se uz pomoć stila unese da ne prikazuje tacke ispred broja id vrednosti ili da se u css failu napiše kod dole.

li {

```
/*stil primenjen za sve li elemente u ovom slučaju ne prikazuje nikakav znak
umesto tačke*/
   list-style-type: none;
}

// dole kodovi omogućavaju prikazivanje tačke, što je standarno, ali umesto bullet
možete postaviti druge
// simbole
#ParentListID li { /* stil primenjen za  li elemente koje imaju id="ParentListID" */
   list-style-type: bullet;
}

li.className { /* stil primenjen za  li klase class="className" */
   list-style-type: bullet;
}
```

U našem primeru možemo izbrisati tačke ispred vrdnosti za Id ovako

```
return <li style={{listStyleType: "none"}} key={movie.id} >
{movie.id} - {movie.title}</li>
```

Napomena: delove css koda ako unosimo za stilove, a one se sastoje od vise reči potrebno je napisti kao i primeru koda iznad. Na primer marginLeft i dr. znači umesto znaka minus između spojiti ih i početno slovo sledeće reči je veliko.

Čime dobijamo ovu sliku prikazanu na ekranu

1 - Child Bride
2 - Last Time I Committed Suicide, The
3 - Jerry Seinfeld: 'I'm Telling You for the Last Time'
4 - Youth Without Youth
5 - Happy Here and Now
6 - Wedding in Blood (Noces rouges, Les)
7 - Vampire in Venice (Nosferatu a Venezia) (Nosferatu in Venice)
8 - Monty Python's The Meaning of Life
9 - Awakening, The
10 - Trip, The

Da bi smo dobili isti prikaz u tabeli možete primeniti ovaj kod,

```
<table>
      {
         data.map(function(movie){
            return(
               <tr>
               <td key={movie.id} >{movie.id}</td>
               <td key={movie.id} >  </td>
               <td key={movie.id} >{movie.title}</td>
               </tr>
            );
         })
      }
</table>
```

umesto koda sa ul i li tagovima, ali možemo ih i napisati i ovako kod

```
class App extends Component {
   render() {

      const Jpodak = data.map(function(movie){
         return(
            <tr>
               <td key={movie.id} >{movie.id}</td>
               <td key={movie.id} >  </td>
               <td key={movie.id} >{movie.title}</td>
            </tr>
         );
      });
      return (
```

i onda ga pozvati

```
<table>
   {Jpodak}
</table>
```

ili samo {Jpodak} između div tagova prikaz je potpuno jednak kao na poslednjoj slici.
Napomena: React će pojaviti grešku ako ostavimo samo tag table bez tagova tbody. Pa je savet u ovim prilikama upotrebljavate ovako napisan kod preventivno jer je sigurnije:
<table>

```
  <tbody>
   {Jpodak}
  </tbody>
</table>
```

Prikupljanje i prosleđivanje podataka

Vrednosti koje preuzimate ili postavljate u radu sa formama takođe mogu se upotrebom this.setState prevesti u vrednosti stanja jer su to podaci formirani u JSON formatu. Takve podatke je potrebno još samo ili prikazati ili proslediti.
Da bi bilo malo jasnije izradiću jednu komponentu koja ima formu za prikpuljanje podataka i iste podatke proslediću kao tekstualne podatke u istoj komponenti u obliku novog stanja dela postojeće komponente.
Kod će sadržavati dosta elemenata koje forma može da sadrži: inpute, textarea, ček bokseve, selekt liste. Pogledajte kod

```
import React from 'react';

class DemoComponent extends React.Component {

// postavljanja osnovnih vrednosti u konstruktoru
   constructor() {
      super();
      this.state = {
         username: '',
         password: '',
         description: '',
         status: true,
         gender: 'male',
         role: 'r2',
         languages: ['lang1', 'lang3']
      };

// može se reci najava dozvoljenih radnji ova dva reda
      this.handleChange = this.handleChange.bind(this);
      this.handleSubmit = this.handleSubmit.bind(this);
   }

// postavljenja za očitavanje stanja
   handleChange(event) {
      const target = event.target;
      let value = target.value;
      if (target.type === 'checkbox') {
         value = target.checked;
      }
```

```
        if (target.type === 'select-multiple') {
            let options = event.target.options;
            value = [];
            for (let i = 0; i < options.length; i++) {
                if (options[i].selected) {
                    value.push(options[i].value);
                }
            }
        }

// u ovom delu se konstanti name predaju prikupljene vrednosti putem atributa
// name koje je prisutno u svakom delu formi i to se formira u obliku niza vrednosti
        const name = target.name;

// jedan od načina upotrebe dinamičkog postavljanja novih vrednosti
        this.setState({
            [name]: value
        });
    }

// pozivanje događaja koji će se desiti posle klika na dugme Save i koje će se sve
opracije dogoditi
  handleSubmit(event) {
     event.preventDefault();

// smeštanje prikupljenih vrednosti stanja u promenjivu result
     let result = 'Account Info<br/>';
     result += 'Username: ' + this.state.username + '<br/>';
     result += 'Password: ' + this.state.password + '<br/>';
     result += 'Description: ' + this.state.description + '<br/>';
     result += 'Status: ' + this.state.status + '<br/>';
     result += 'Gender: ' + this.state.gender + '<br/>';
     result += 'Role: ' + this.state.role + '<br/>';
     result += 'Languages: ' + this.state.languages + '<br/>';

// prosleđivanje novih stanja promenjive result refrencom u html tag, kako bi bile
prikazane na ekranu
     this.refs.result.innerHTML = result;
     }

    render() {
        return (
            <div style={{margin:"100px"}}>
                <h2> Korisnicki podaci prijemna forma</h2>
```

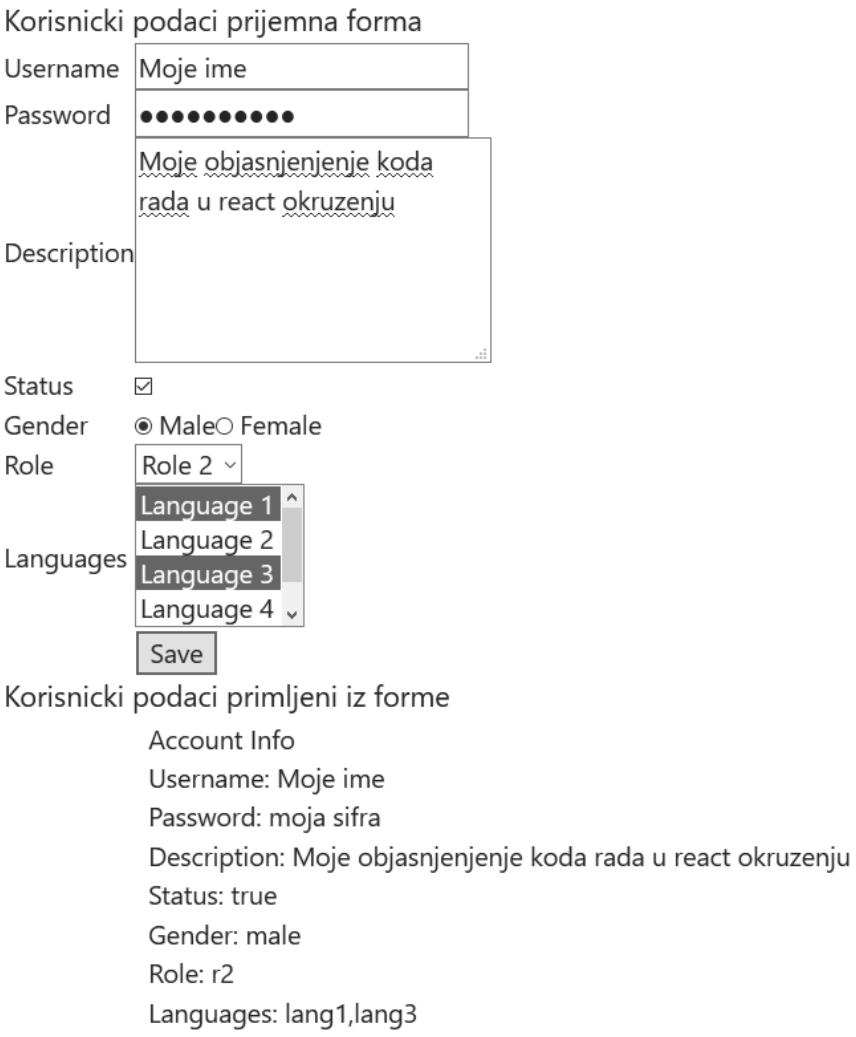
```
  <hr />
<form onSubmit={this.handleSubmit.bind(this)}>
   <table cellPadding="2" cellSpacing="2">
      <tr>
         <td>Username</td>
         <td><input type="text" name="username"
                  value={this.state.username}
                  onChange={this.handleChange} /></td>
      </tr>
      <tr>
         <td>Password</td>
         <td><input type="password" name="password"
                  value={this.state.password}
                  onChange={this.handleChange} /></td>
      </tr>
      <tr>
         <td>Description</td>
         <td>
            <textarea name="description" cols="20" rows="5"
                  value={this.state.description}
                  onChange={this.handleChange}/>
         </td>
      </tr>
      <tr>
         <td>Status</td>
         <td>
            <input type="checkbox" name="status"
              checked={this.state.status}
             onChange={this.handleChange} />
         </td>
      </tr>
      <tr>
         <td>Gender</td>
         <td>
            <input type="radio" name="gender" value="male"
        checked={this.state.gender === 'male'}
          onChange={this.handleChange} /> Male
            <input type="radio" name="gender" value="female"
        checked={this.state.gender === 'female'}
         onChange={this.handleChange} /> Female
         </td>
      </tr>
      <tr>
         <td>Role</td>
```

```
                    <td>
                      <select name="role" value={this.state.role}
                        onChange={this.handleChange}>
                        <option value="r1">Role 1</option>
                        <option value="r2">Role 2</option>
                        <option value="r3">Role 3</option>
                        <option value="r4">Role 4</option>
                        <option value="r5">Role 5</option>
                      </select>
                    </td>
                  </tr>
                  <tr>
                    <td>Languages</td>
                    <td>
                      <select name="languages" multiple={true}
                            value={this.state.languages}
                        onChange={this.handleChange}>
                        <option value="lang1">Language 1</option>
                        <option value="lang2">Language 2</option>
                        <option value="lang3">Language 3</option>
                        <option value="lang4">Language 4</option>
                        <option value="lang5">Language 5</option>
                      </select>
                    </td>
                  </tr>
                  <tr>
                    <td> </td>
                    <td><input type="submit" value="Save" /></td>
                  </tr>
                </table>
              </form>
               <hr />
               <h2> Korisnički podaci primljeni iz forme</h2>
               <hr />
              {/* u ovom slučaju se prikupljeni podaci prikazuju u
                 ovom div tagu koji ima id=“result“ */}
              <div ref="result" style={{marginLeft:"100px"}}> </div>
            </div>
        );
    }
}

export default DemoComponent;
```

Napomena : ovim kodom se uvek bezuslovno postavllja novo stanje, onamo gde je to potrebno, uvek u JSON formi

```
this.setState({
   [name]: value
});
```

This.setState nemojte postavljati posle rendera ili return delu jer moguće da dođe grešaka u radu kako savetuju mnogi u tutorijalima na netu.



Naravno ovu formu možete ulepšati upotrebom Bootstrapa ili sopstvenim stilovima sa dodatkom onFocus, ali u svakom slučaju mora se dodati prilikom upotrebe verifikacija odgovarajućih zahteva. O tome više u delu o komponentama u reactu.

Komponente u react aplikacijama

Komponente su skup koda kojim ih formiramo da bi se prikazale na veb stranicama. Samo uobličavanje koda njihovih delova ili njih samih može biti smešteno u klase, funkcije ili konstante. Gde, klase mogu da imaju svoje sopstvene vrednost i za stanja (state) ili mogu da ih prihvataju od drugih klasa direktno ili posredno. Konstante mogu samo da prihvataju poslate vrednosti stanja kao props. Dok, funkcije mogu da prihvataju i da vrate rezultat svoga rada. U svakom slučaju organizacija je sledeća:

1. svi delovi moraju biti povezani upotrebom komande ili službene reči import
2. svi delovi moraju sadržati na svom kraju ili prilikom svog počinjanja export default
3. imena klasa, constanti, promenjivih, funkcija uvek počinju velikim slovom
4. Prvi red kod klasa mora sadržati import React from 'react'; iza čega slede onda css-a i drugo
5. Prvom redu kod klasa dodaju se importi potrebnih dodataka za rad
6. Po potrebi kod klasa dodaju se redovi koda za konstruktor i ostale potrebne elemente klase u zavisnosti šta vam je potrebno da napravite.

To je nešto ukratko kako i od čega se sastoji jedan fail buduće komponente.

Napomena: najbolji po meni način u radu sa reactom je u src folderu napraviti folder za komponente u okviru koga za svaku komponentu po jedan folder u kome smeštamo jedan glavni fail. Moj savet je da bi ste se snalazili u tome da za njegovo ime upotrebite vasNazivApp.js ili ako upotrebljavate jsx kod vasNazivApp.jsx, a za njegove delove vasNazivImeDelaKojiJeUSastavu.js ili sa ekstenziom jsx. I u tome istom folderu izradite css folder za njegove css failove vasNazivApp.css, kao i za sve ostale delove koji čine vasNazivApp.js.
Ovakav pristup u radu još više smanjuje pojavu eventualnih grešaka, jer je kod male dužine i lak za snalaženje.
Komponente ili delovi iz kojih se dobijaju elementi budućih veb aplikacija, tj. njihovi kodovi u svom radu upotrebljavaju pored već navedenog i vrednosti koje putem stanja (**state**) šalju kao osobine (**props)**. Gde vrednosti stanja mogu menjati, postavljati, dok osobine su samo za čitanje. A sam rad se može opisati kao gradnja kuce. Prvo se planira vizija kako sta izgleda, posle toga se sagledavaju potrebe da bi se utvrdile funkcije i na kraju spojilo sve u jednu celinu.

U svim verzijama **React-a** do v16 postojala je upotrebe može se reći službene reči **var**, za promenjive, koja je u V16 zamenjena rečiju **let,** jer je uveden način pisanja koda ES6 i viši**.** Tako ako pogledate tutorijale na netu obratite pažnju, jer prilikom izrade aplikacija uvek će vam se dogoditi da se instališu paketi i react sa njima u najvišoj verziji, pa aplikacija rađena za nižu verziju može da se dogodi da u višoj neće da radi.

Komponente su delovi veb stranica, ali i kao što je napisano ispred i one same mogu imati svoje komponete (child). Pored toga, komponente mogu biti kontrolisane i ne kontrolisane. Takođe sadržati u sebi kodove različitih tehnologija i pristupa određenim delovima aplikacije ili sopstvenim delovima. Tako da mogu pristupati podacima po dubini aplikacije u više nivoa, pri čemu mogu da jedan preko druge da salju podatke o vrednostima stanja. Ali bilo kako bilo treba uvek imati na umu da je komponenta višeg reda (parent – roditelj) vlasnik podataka i ona uvek šalje vrednost stanja (this.state.name) koje je smešteno onamo gde je ona pozvala pod komponetu (child).
<PodKomponenta name={this.state.name} />
A, da pod komponeta prima to stanje kao props

```
const PodKomponenta=(props)=>{

      return <h2> {this.props.name)</h2>
}
```

Naravno, ovo ne mora da bude konstanta, može da bude i klasa, vrednost za props nemora da prihvati tag h2 može da ga prihvati i naziv dugmeta ili bilo kako drugačije u zavisnosti od vaših potreba.
U svakom slučaju one mogu da pristupaju po potrebi i zahtevu korisnika i naravno dozvolama nas koji ih pravimo pišući njihov kod, pristupu serveru, redirekciju na druge servere. Što react čini jako upotrebljivim sa svojim dodacima, obzirom da je on samo tu da lakše napravimo potrebnu aplikaciju.
Forme mogu biti izrađivanje kao kontrolisane i ne kontrolisane. Primer ne kontrolisanih formi su forme za pretragu po aplikaciji i serveru, ali i po internetu.

Forme u react aplikaciji

Osnovni kod formi je u HTML-u je

```
 <form>
  <label>
    Name:
    <input type="text" name="name" />
  </label>
  <input type="submit" value="Submit" />
</form>
```

Kao takav se dodaje i prilikom izrade komponente forme u react-u ovako

```
import React, { Component  } from 'react';

export default class NameForm extends React.Component {
```

```
  constructor(props) {
    super(props);
    this.state = {value: ''};

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value});
  }

  handleSubmit(event) {
    alert('A name was submitted: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
       <input type="text" value={this.state.value} onChange={this.handleChange} />
        </label>
           < br/>
         <label>
          Name:
          <textarea value={this.state.value} onChange={this.handleChange} />
        </label>
       < br/>
        <label>
           Možete izabrati iz naše ponude:
          <select value={this.state.value} onChange={this.handleChange}>
            <option value="jabuka">Jabuka</option>
            <option value="kruška">Kruška</option>
            <option value="jagoda">Jagoda</option>
            <option value="malina">Malin</option>
          </select>
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

Kao što se vidi uobičajeno je da se u formama upotrebljavaju dva događaja: onChange za praćenja promena vrednosti polja input i onSubmit za slanje unetih podataka na dalju obradu. Ovakav pristup i sam kod se naziva u reactu kontrolisana komponenta. Deo forme su numerička polja i ček boksevi, što možete videti u primeru dole

```
import React ,{Component} from 'react';

export default class Reservation extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      isGoing: true,
      numberOfGuests: 2
    };

    this.handleInputChange = this.handleInputChange.bind(this);
  }

  handleInputChange(event) {
    const target = event.target;
    const value = target.type === 'checkbox' ? target.checked : target.value;
    const name = target.name;

    this.setState({
      [name]: value
    });
  }

  render() {
    return (
      <form>
        <label>
          Is going:
          <input
            name="isGoing"
            type="checkbox"
            checked={this.state.isGoing}
            onChange={this.handleInputChange} />
        </label>
        <br />
        <label>
          Number of guests:
```

```
            <input
              name="numberOfGuests"
              type="number"
              value={this.state.numberOfGuests}
              onChange={this.handleInputChange} />
          </label>
        </form>
      );
    }
}
```

Ali, forme mogu biti i nekontrolisane kao ova ispred, niže je kod jedne takve forme

```
import React ,{Component} from 'react';

class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleSubmit(event) {
    alert('A name was submitted: ' + this.input.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input type="text" ref={(input) => this.input = input} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

One su ne kontrolisane jer prema mnogima upotrebljavaju u svom radu refrencu na input

```
<input type="text" ref={(input) => this.input = input} />
```

Primer kod forme koji poziva drugu komponentu za prikaz podataka

Ovaj primer forme zanimljiv je jer poziva drugu komponetu u kojoj se vrši prikaz unetih podataka, koji su prosledjeni i sacuvani u localStorage u obliku objekta.

```
import React from "react";
import Stranica from'./stranica';

export default class Singup extends React.Component {
   constructor() {
      super();
      this.state = {
         email:'',
         password:'',
         name:'',
         hasAgreed: false,
         objekt:{}
      };
      this.handleChange=this.handleChange.bind(this);
      this.handleSubmit=this.handleSubmit.bind(this);
   }

   handleChange(e){
      let target = e.target;
      let value = target.type === 'checkbox'? target.checked : target.value;
      let name= target.name;
      this.setState({[name]: value})
   }
   handleSubmit(e){
      e.preventDefault();
      let poslasto= this.state;
      localStorage.setItem('proba', JSON.stringify(poslasto));
     }

   render(){

      return (
      <div className="container">
        <h2>xxcvx</h2>
         <Stranica/>
```

```
    <hr/>
    <form onSubmit={this.handleSubmit}
      className="table-bordered col-md-5"
          style={{backgroundColor:"#F7F7F7", marginBottom:"20px",
          paddingBottom:"20px",
          borderRadius: "2px"}}>
      <div className="form-group">
        <label htmlFor="ime"> Ime :</label>
        <input type="text" id="name"  className="form-control"
               name="name" placeholder="Unesi cvoje ime"
              onChange={this.handleChange} value={this.state.name} />
      </div>
      <div className="form-group">
        <label htmlFor="email"> Email :</label>
        <input type="text" id="email"
              className="form-control"  name="email"
              placeholder="Unesi svoj email" onChange={this.handleChange}
              value={this.state.email}/>
      </div>
      <div className="form-group">
        <label htmlFor="password">Password :</label>
        <input type="text" id="password"
             className="form-control" name="password"
             placeholder="Unesi password" onChange={this.handleChange}
         value={this.state.password}/>
      </div>
      <div className="form-group">
        <label htmlFor="hasAgreed">
          <input type="checkbox" id="hasAgreed"
                    name="hasAgreed"
                onChange={this.handleChange}
                 value={this.state.hasAgreed}/>
          <a href=" " > Prihvatanje uslovi i servisi</a>
        </label>
      </div>
      <div>
        <button type="submit" name="submit"
            className="form-control btn btn-primary" >
          Sing Up
         </button>
      </div>
    </form>
  </div>
  );
```

```
    }
}
```

Kod faila stranica koji preuzima podatke iz localstorage i prikazuje ih

```
import React from "react";
//import { BrowserRouter, Route, NavLink,Switch, Redirect} from "react-router-dom";
export default class Stranica extends React.Component{
    componentWillMount(){
        this.setState({
            objekat: JSON.parse(localStorage.getItem('proba'))
        });
    }

    render(){
        Object.keys(this.state.objekat);
        console.log(this.state);
        return(
            <div>
                <h1> heder</h1>
                <h1>Moje ime :<span>  {this.state.objekat.name }</span></h1>
                <h2>Moj email : <span> {this.state.objekat.email }</span></h2>
                <h2>Moja sifra :<span> {this.state.objekat.password}</span></h2>
                <h2>Ovo je pocetna index stranica korisnika:
                  {this.state.objekat.name }
                </h2>
            </div>
        );
    }
}
```

Osim toga u ovom failu prikazana je i upotreba dela reacta componentWillMount koja se uvek formira pre komponente componentDidMount, koja je vezana za komunikaciju putem API i ona se pokreće uvek prva posle izvršenog rendera komponente.
Sistem prikazivanja i pokretanja je sledeći: Prvo se pokreće ono u konstruktoru, pa componentWillMount, pa nju sledi componentDidMount pa tek onda render.
Ali ono što se može biti u ovom primeru koda je način prikazivanja delova podataka objekta koji su u njemu.
Napomena: Obzirom da u ovom delu pišem o formama u react aplikaciji, nakon unosa podataka i klika na dugme Sing Up, obzirom da komponente treba još povezati upotrebom delova dodatnog paketa react-router-dom ručno refrešovati u brovseru da bi se prikazali podaci u komponenti stranica.

Kloniranje elemenata React.cloneElement

Komponente možete u react izrađivati i kloniranjem postojeće, to je jednostavan postupak kojim se dobijaju novi delovi na istom ili drugom mestu.
Kloniranje ili dobijanje elemenata sa istim osobinama na nekom drugom mestu na veb stranici react omogućava ovim kodom niže. Ono što je važno napomenuti da ovaj isti primer može da se upotrebi i kod drugih elemenata, kao što su inputi u formama i slično.
U ovom slučaju potrebne su nam dve klase, gde u prvoj formiramo izgled galerije slika

```
import React from 'react';
class App extends React.Component{
   render(){
      return(
         <div className="container ">
            <Galerija>// prikaz klonirane slike
               <hr/>
               <img className="btn btn-primary" src={require('./slike/jumbo.jpg')}
                  width={150} height={150} alt="prvo"/>
               <img className="btn btn-primary" src={require('./slike/motiv1.jpg')}
                  width={150} height={150} alt="prvo"/>
               <img className="btn btn-primary" src={require('./slike/motiv2.jpg')}
                  width={150} height={150} alt="prvo"/>
               <img className="btn btn-primary" src={require('./slike/karmen.jpg')}
                   width={150} height={150} alt="prvo"/>
            </Galerija>
         </div>
      );
   }
}
export default App;
```

U gornjem kodu prostor između tagova Galerija prestavlja ono što se u reactu naziva child deo sa kojm operira donji kod faila Galerija putem map metoda kroz upotrebu React.Children-a.
I druge koja obradjuje podatke o atributu taga img src, tačnije u njoj se ujedno kloniranjem odabira slika događajem onClick koja se prikazuje u gornjem delu galerije ili onamo gde to odredite upoterbom css-a.

fail Galerija.js

```
class Galerija extends React.Component{
   constructor(){
      super();
```

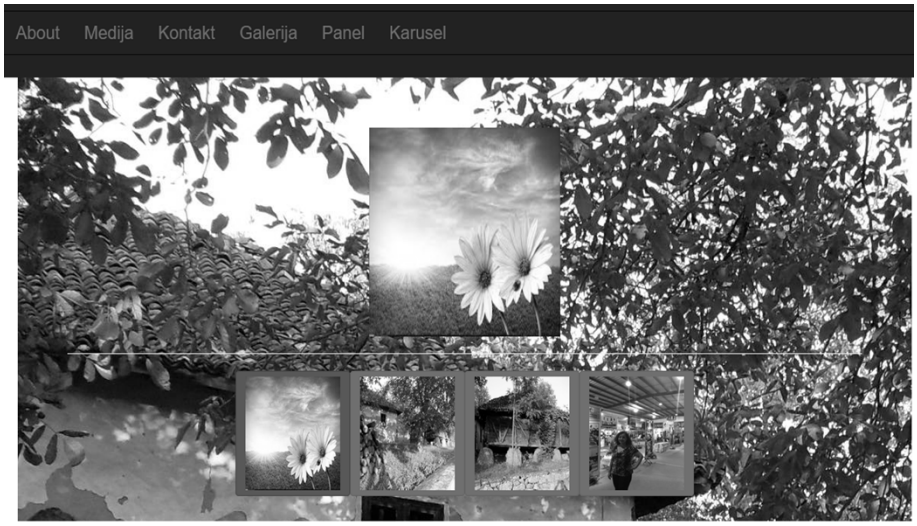
```
        this.state={
            selected: require('./slike/jumbo.jpg'),// postavljanje početnog stanja
        };
    }
    selectItem(selected) {
        this.setState({
            selected
        });
    }
    render(){
        let fn = child =>
            React.cloneElement(child,{
                onClick: this.selectItem.bind(this, child.props.src)
            });
        let items = React.Children.map(this.props.children, fn);
        return(
            <div className="odstojanje">
                <img src={this.state.selected} width={250} height={250}
                  alt="Slika u gornjem prikaznom delu "/>
                {items}
            </div>
        );
    }
}
```

Princip rada
Da bi sve moglo da radi potrebno je da postoji pocetno stanje selected, koje se događajem onClick moze menjati u neko novo stanje, pa je potrebno u tom slucaju napraviti konstruktor

```
    constructor(){
        super();
        this.state={
            selected: require('./slike/jumbo.jpg'),
        };
    }
```

gde je odmah uneta vrednost za stanje selected, koja je ujedno pocetna vrednost

Izgled u radom delu na prikazu u brovseru Galerije slika



Iza toga metod ili funkciju koja ce omogućavati da se to stanje menja

```
selectItem(selected) {
    this.setState({
        selected
    });
}
```

Što znaci da je potrebno i napraviti jednu ili vise promenjivih u zavisnosti od potrebe, kako bi se moglo upotrebiti u ovom slucaju koloniranje elementa, slike na koju se bude kliknulo

```
    let fn = child =>
        React.cloneElement(child,{
            onClick: this.selectItem.bind(this, child.props.src)
        });
    let items = React.Children.map(this.props.children, fn);
```

Rad predhodnog dela koda

Promenjiva fn, je kao sto se vidi strelasta funkcija kojom vršimo kloniranje elementa upotrebom React.cloneElement, predajuci child delu osobinu src izvora na koji smo kliknuli pokretanjem dogadjaja onClick .
Time odobrili da se njegove osobine primene upotrebom selectItem, koju je taj klik na neku od slika pokrenuo. Obzirom da smo time promenili vrednost stanja u stanju selected, doslo je i do promene stanja

```
<div className="odstojanje">
  <img src={this.state.selected} width={250} height={250}
    alt="Slika u gornjem prikaznom delu "/>
```

u atributu src u tagu img koji opisuje putanju do slike i ime, ekstenziju slike. Čime smo prakticno dobili promenu slike u gornjem delu prikaza galerije.
Dok u ovom delu promenjiva items, koja prestavlja prakticno jedan niz dobijen upotrebom React.Children kroz map metod. Kao sto se vidi to u donjem kodu {items} prikazuje sve slike koje su u klasi App

Pozvane kao pod komponente, odnosno child.

```
<div className="odstojanje">
   <img src={this.state.selected} width={250} height={250}
    alt="Slika u gornjem prikaznom delu "/>
   {/* niz slika u donjem delu prikaza galerije  */}
   {items}
</div>
```

Napomena: U pisanju koda na ovaj način

```
<Galerija>
  <hr/>
  <img className="btn btn-primary" src={require('./slike/jumbo.jpg')}
     width={150} height={150} alt="prvo"/>
  <img className="btn btn-primary" src={require('./slike/motiv1.jpg')}
     width={150} height={150} alt="prvo"/>
  <img className="btn btn-primary" src={require('./slike/motiv2.jpg')}
     width={150} height={150} alt="prvo"/>
  <img className="btn btn-primary" src={require('./slike/karmen.jpg')}
      width={150} height={150} alt="prvo"/>
</Galerija>
```

sve što se nađe izmedju tagova je child deo u ovom slučaju komponente **<Galerija> </Galerija>.**
Skraćeni oblik pisanja je ovaj **<Galerija />.** U svakom slučaju možemo i dodatno opisati svaki element css-om.

Zakljucak

Ovo je jedan od načina upotrebe onog sto sam **React** omogućava, naravno možete umesto da primenite ovaj poslednji kod ovako napisan da upotrebite i za to **map** metod i podatke smestite vezane za atribut **src**-a u **data.json** pa da ga povežete u komponentu ili da ga pretvorite u neku konstantu, sto zavisi od vaših potreba i namera. Slično se moze napraviti i za unosna **input** polja u tabelama i formama, odnosno svuda gde po vašim potrebama pogodno upotrebiti **React.cloneElement.**
Ovaj deo teksta preuzet sa mog profila na sajtu akadmije na ovom linku:

https://www.academia.edu/36968397/Izrada_galerije_slika_u_Node-React_okruzenju_upotrebom_React.cloneElement

Izrada galerije slika 1

Ovo je jedan primer koji opisuje navedenu pojavu razmene podataka između komponenti u react aplikacijama sa malo drugačijim pristupom od predhodnog koji vrši kloniranje ili kopiranje jednog elementa u drugi na drugom mestu.
Jedan od mnogih načina izrade galerije slika, koji može poslužiti o za druge svrhe, kao što su podaci koji govore na primer koliko je emailova pristiglo u email sandučetu. U ovom primeru poći ću da napravim prvo malu bazu podataka upotrebljavajući JSON format podataka smeštenim u jednu promenjivu iz koje ću preko map metoda pozvati te vrednosti i smesti ih u njihova mesta povezujući više komponenti.
Za ovaj slučaj prvo je potrebno napraviti taj niz i smesti ga u promenjivu ili ako vam je lakše u kostantu. Nebitno koju reč upotrebite let ili const ispred imena primer će isto raditi. To se radi na sledeći način:
posle znaka jednako slede srednje zagrade koje označavaju početak i kraj niza a između njih u velikim zagradama pišu se parovi ključeva i podataka – njihove vrednosti. Iza zatvorene zagrade ili kraja koda tih parova podataka obavezno se postavlja znak zareza, nakon čega ide sledeća grupa podataka i tako do količine podataka koji su nam potrebni.

```
let urls = [
    {
      title_button: 'Butstrap',
      number_button: 12,
      src: require('./slike/oblak.jpg'),
      title: 'Ime slike prve'
    },
    {
      title_button: 'Butstrap',
      number_button: 12,
      src: require('./slike/jumbo.jpg'),
      title: 'Ime slike druge'
    }
];
```

Napomena: Posle dužeg eksperimetisanja ono što sam utvrdio da je najbolje i najsigurnije upotrebiti u početku kod za src slika koje nameravate da upotrebite u aplikaciji je

```
src: require('./slike/oblak.jpg').
```

odnosno kada direktno upotrebljavate u img tagu napisati

```
<img  src={ require('./slike/oblak.jpg')}  alt=" slika" width={100}/>
```

Ovo je iz razloga upotrebe načina pristupa putanjama do slika, relative ili absolute path . Vezano za upotrebu slika u ovom i sličnim slučajevima je **alt=" "** je obavezan, nebitno da li će između navodnika imati neki tekst ili ne. U slučaju izostavljanja **alt=" "** prijavljuje se greška u inspekt modu brovsera, ali to ne ometa rad aplikacije. Oznake širine i visine piše se kao u gornjem kodu, ali najbolji način je pisanje u css fail ili upotrebiti kod za direktnu primenu u img tagu ili bilo kom gde vam je to potrebno:

```
style={{ width: "100px", height:"45%"}} // inline style
```

Možete instalisati dodatni paket **react-bootstrap** i njegova upotreba je kao u kodu dole niže. Njegova instalacija vrši se ovako :

```
npm i  - - save react-bootstrap
```

```
import React, {Component} from r'react';
import { Grid, Col } from 'react-bootstrap';
import'./style.css';

export default class Galerija extends Component{
  render(){
     return(
      <div>
   <img  src={ require('./slike/oblak.jpg')}  alt=" slika" width={100}
           className=" neka klasa"/>
      </div>
     );
  }
}
```

Predhodno delom opsani kod, u kome vidimo da ima niz od dva objekta, koji imaju istu vrednosti za ključeve, ali sa različitim vrednostima

```
 let urls = [
    {
       title_button: 'Butstrap',
       number_button: 12,
       src: require('./slike/oblak.jpg'),
       title: 'Ime slike prve'
    },
    {       title_button: 'Butstrap',
       number_button: 12,
```

```
        src: require('./slike/jumbo.jpg'),
        title: 'Ime slike druge'
    }
];
```

Konstanta List upotrebom map metoda pravi od predhodnog koda niz podataka gde preko spread-a (tri tačke – skraćeni način pisanja koda) priključujući sve osobine – props pod komponenti (child) . Predhodni JSON format promenjive **urls** može upotrebiti i JSON fail koji je van ovog faila gde se pozivaju njegovi podaci. U tom slučaju bi trebalo prvo importovati ga pa onda njegove podatke preko inicijalnog stanja dodeliti nekoj promenjivoj i onda je pozvati da se nad time izvrši map metod kao dole niže u ovom trenutnom slučaju na kome radimo sada.

```
const list = urls.map(function (thumbnailProps) {
   return (
<div key={thumbnailProps.src}>
        <Thumbnail {...thumbnailProps} />
   </div>
     );
});
```

Napomena: U slučaju da se izostavi **key={ },** react će vam prijaviti grešku te ih uvek morate gde je to potrebno izvršiti map metod imati u tagu koji je višeg nivoa. Na primer div je najvišeg nivoa jer on može da sadrži sve ostale tagove koji su nižeg nivoa od njega, ul je višeg nivoa od li i td.

U ovom primeru **key={}** se može primeniti

** **

или

<Thumbnail key={thumbnailProps.src} {...thumbnailProps} />

ili moguća primena može biti ovako

**<li key={thumbnailProps.src} > **

Uobičajeno je da **key={}** sadrži **id** polje u povratku, u nekim slučajevima može saržati index, ali može da sadrži bilo koji naziv polja ključa koji sadrži objekt u data.json failu ili u promenjivoj koja sadrži format json objekta.
Donji kod opisuje još jedan način pisanja koda u reactu, to jest upotrebu strelaste funkcije (arrow), koji radi potpuno isti posao kao gornji kod.

```
const list = urls.map((thumbnailProps)=> {
   return(
      <div >
      <Thumbnail key={thumbnailProps.src}{...thumbnailProps} />
         <Thumbnai key={thumbnailProps.src}{...thumbnailProps} />
      </div>
   );
});
```

Ali zbog pomenute pojave greške bolje je napisati ključ ovako

```
<div key={thumbnailProps.src}>
```

jer se ovaj deo koda postavlja izmedju

```
    <Thumbnail {...thumbnailProps} />
</div>
```

Napomena: ovako napisan kod

```
  <Thumbnail {...thumbnailProps} />
```

označava da se vraćaju sve osobine koje ima

```
 <Thumbnail  />
```

čiji se podaci šalju pod klasi sa istim imenom u ovom slučaju, za slučaj da je ona van ovog faila potrebno je prvo importovati, i konstanta **list** se mora naći unutar tamošnje klase

```
import Thumbnail from ‘./Klasa_faila’;
```

Kod klase posebnog faila

```
class Klasa_faila extends Component {

   render(){
let urls = [....
const list= ......
      return(
<div>
    {list}
</div>
```

Napomena: Reči koje upotrebite za nazive klasa, komponenti, varijabile, konstante, funkcija možete napisati šta želite na Srbskom ili nekom drugom govoru. Dok, službene reči moraju biti na engleskom. U ovom slučaju:

```
    class Klasa_faila extends Component {

   render(){
let urls = [....  // može biti let moji_linkovi = [....
const list= ......// const moja_lista= ......
      return(
<div>
    {list} //      {lista}
</div>
```

Kod pod klase dole niže formira kompletan izgled prikaza slika na ekranu.

```
class Thumbnail extends Component {
   render(){
      return(
         <div className="col-sm-6 col-md-2">
            <div className="thumbnail">
               <img src={this.props.src} alt="..." width={150} height={150}/>
               <div className="caption">
                  <h3>{this.props.title}</h3>
                  <p>{this.props.description}</p>
                  {this.props.child}
               </div>
            </div>
         </div>
      );
   }
}
```

U slučaju da se upotrebljava i modal sa klikom na dugme ili sliku polja u polaznoj varijabili **urls** mogu se dodati pošto u okviru modala mogu se dodati detaljniji opisi, čime bi se dobilo bolje objašnjenje. Kao primer može poslužiti online prodavnica racunara, gde pored slike računara imate i druge podatke, linkove i dr. Da se omogućila upotreba – prikaz gde je to potrebno upotrebljava se ovaj kod, za slučaj da se želi odvojiti ovaj sardžaj od ostalog sadržaja na stranici

```
<div>
    {list}
</div>
```

u suprotnom može se upotrebiti samo **{list}** zatvoren odgovarajućim tagovima. Dodatkom ovog koda moguće je postaviti i objaštenja na dugme u klasi Thumbnail ispod dela za objašnjenja

```
<p>{this.props.description}</p>

<p>
   <Badge title={this.props.title_button} number={this.props.number_button }
/>
</p>
```

Kao i dodavanjem klase za ovo dugme

```
class Badge extends Component{
   render() {
      return(
         <button className="btn btn-primary" type="button">
            {this.props.title} <span
className="badge">{this.props.number}</span>
         </button>
      );
   }
}
```

U ovom poslednjem primeru vidljivo je da je upoterbljena klasa iz Bootstrapa **badge –** bedž.

Događaji (event)

Događaji su sve ono čime možemo da utičemo na promene sadržaja koji su dozvoljeni kodom aplikacije na stranici gde smo trenutno. U reactu su podržani događaji: Klipboarda(Clipboard events), Kompozicioni(Composition events), tastature (keyborad events), Fokus(focus events), Miš (mouse events), Slekcioni(Selection events), dodir vezano za mobilne uređaje i react native(touch events), UI, točkić miša (Whell events), Medija, Slika, animacije, tranzicije i drugi. Obzirom da je ovo knjiga osnove reacta-bootstrapa-phpa navešću njihove refrence i u primerima kako oni rade i kako se pozivaju u većem delu. Ono što bih napomenuo da je za neke specifične događaje potrebno izvršiti instalcije dodatnih paketa sa https://www.npmjs.com/. da bi ovo bilo razumljivije objasniću događaj onCopy.

Za rad ovog događaja(event) potrebno je u aplikaciji instalisati dodatni paket react-copy-to-clipboard

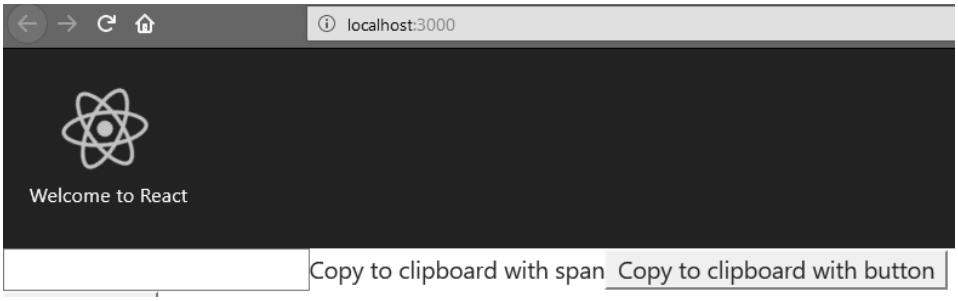
upotrebom

npm install --save react react-copy-to-clipboard

Potom primeniti ovaj kod kako bi se video učinak rada

```
import React from 'react';
import ReactDOM from 'react-dom';
import {CopyToClipboard} from 'react-copy-to-clipboard';

export default class App extends React.Component {
   state = {
      value: '',
      copied: false,
   };

   render() {
      return (
         <div>
            <input value={this.state.value}
                  onChange={({target: {value}}) =>
                  this.setState({value, copied: false})} />
            <CopyToClipboard text={this.state.value}
                        onCopy={() => this.setState({copied: true})}>
               <span>Copy to clipboard with span</span>
            </CopyToClipboard>

            <CopyToClipboard text={this.state.value}
                        onCopy={() => this.setState({copied: true})}>
               <button>Copy to clipboard with button</button>
            </CopyToClipboard>
            {this.state.copied ? <span style={{color: 'red'}}>Copied.</span> : null}
         </div>
      );
   }
}

const appRoot = document.createElement('div');
document.body.appendChild(appRoot);
ReactDOM.render(<App />, appRoot);
```

Koji daje ovakav prizor



Gde su vam ponuđene mogućnosti da kopirate upotrebom:

1. iz levog unosnog prozora
2. ili klikom na dugme

Naravno da se ovo kopiranje može primeniti i na druge podatke, te sa ovim događajima onCopy, onCut, onPaste možete izraditi jednostavan tekst editor.

Refrence

Klipboarda(Clipboard events)
Nazivi dogadjaja
onCopy, onCut, onPaste
namena(Proprietes)
za DOMDataTransfer u klipboardu

Kompozicioni(Composition events)
Nazivi dogadjaja
onCompositionEnd, onCompositionStart, onCompositionUpdate
namena(Proprietes)
string data

Tastature (keyborad events)
Nazivi dogadjaja
onKeyDown, onKeyPress, onKeyUp
namena(Proprietes)
boolean altKey
number charCode
boolean ctrlKey
booblean getModifierState(key)
string key
number keyCode
string locale
number location
boolean metaKey
boolean repeat

boolean shiftKey
number wich

Fokus(Focus events)
Naziv
onFocus, onBlur
Napomena: ovaj event radi sa svim elementima u reactu
DOMEventTarget relatedTarget

Form events
Naziv:
onChange, onInput, onSubmit

Mouse Events
Nazivi:
onClick onContextMenu onDoubleClick onDrag onDragEnd onDragEnter onDragExit
onDragLeave onDragOver onDragStart onDrop onMouseDown onMouseEnter onMouseLeave
onMouseMove onMouseOut onMouseOver onMouseUp
onMouseEnter i onMouseLeave su događaji koji se propagiraju od elementa koji je ostavljenome koji se unosi umesto običnog bubbling-a i nema fazu zauzimanja.
Properties:
boolean altKey
number button
number buttons
number clientX
number clientY
boolean ctrlKey
boolean getModifierState(key)
boolean metaKey
number pageX
number pageY
DOMEventTarget relatedTarget
number screenX
number screenY
boolean shiftKey

Selection Events
naziv:
onSelect

Touch Events

naziv:
onTouchCancel onTouchEnd onTouchMove onTouchStart
Properties:
boolean altKey
DOMTouchList changedTouches
boolean ctrlKey
boolean getModifierState(key)
boolean metaKey
boolean shiftKey
DOMTouchList targetTouches
DOMTouchList touches

UI Events
naziv:
onScroll
Properties:
number detail
DOMAbstractView view

Wheel Events
naziv:
onWheel
Properties:
number deltaMode
number deltaX
number deltaY
number deltaZ

Media Events
naziv:
onAbort onCanPlay onCanPlayThrough onDurationChange onEmptied
onEncrypted
onEnded onError onLoadedData onLoadedMetadata onLoadStart onPause onPlay
onPlaying onProgress onRateChange onSeeked onSeeking onStalled onSuspend
onTimeUpdate onVolumeChange onWaiting

Image Events
naziv:
onLoad onError
Animation Events
naziv:
onAnimationStart onAnimationEnd onAnimationIteration
Properties:
string animationName

string pseudoElement
float elapsedTime

Transition Events
naziv:
onTransitionEnd
Properties:
string propertyName
string pseudoElement
float elapsedTime

Ostali dogadjaji
naziv:
onToggle

Kao što se vidi iz prikazanog react podržava mnogo događaja kojima se mogu obraditi sve situacije koje vam se nađu u projektima. Ono što se može još napomenuti jeste to da postoje i sintetički događaji koje isto react podržava. Ali. ono što ćemo najčešće upotrebljavati su događaji vezani za onClick, onSubmit i td. Uobičajeno je upotrebiti ovakav kod za pozivanje događaja

```
nazivDogadjaja( e ili event){
```

U ovom delu se postavljaju uslovi koji su potrebi da bi određeni događaj se mogao izvršavati. Na primer povećavanje u koracima ja vrednost broja 2

```
        this.setState({
          brojac: this.state.brojac +2
        });
}
```

Ovako napisan kod označava da se svakim pozivom događaja postojećoj vrednosti promenjive brojač dodaje povećanje za vrednost broja dva ili čega odlučite.

Zanimljiv događaj dugmetom

```
import React from 'react';

export default class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};

    // Neophodno povezivanje da bi omogućilo rad događaja, funkcija i sl.
```

```
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState(prevState => ({
      isToggleOn: !prevState.isToggleOn
    }));
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}
```

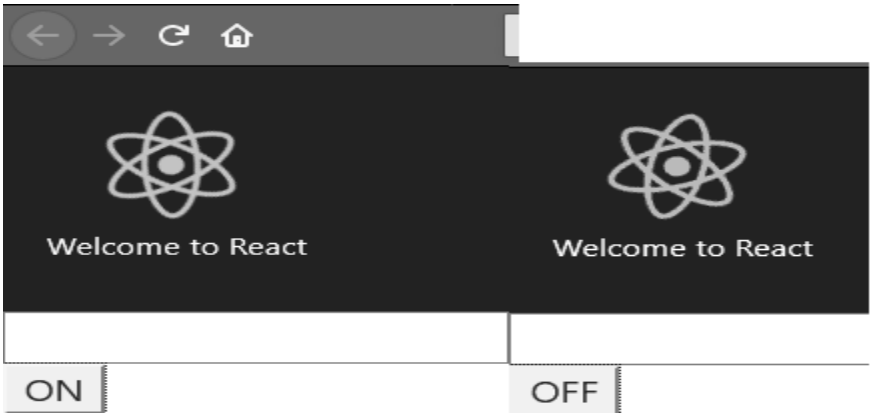
Šta je ovde posebnost? Posebnost ovog koda je to što se menja klikom na dugme njegov naziv. Pogledajmo malo ovaj deo koda

```
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
        // ovaj deo radi kao if else princip, ako nije On onda je OFF
      </button>
```

```
this.state.isToggleOn?  'ON' : 'OFF'
```

Deo koda iza isToggleOn susretaćemo kod slanja podataka upotrebom axiosa (dodatak reactu) i na drugim mestima i on bi praktično bio kao ispitivanje if petlje. Ako nije ono što jeste onda je ono drugo istina.

Znak pitanja je if ON je prvi uslov posle if-a , a dvotačka je else i OFF je ono iza else. Ovo sam namerno dodao jer je situacija slična slanju queriy-a Get metodom gde u prvom slučaju proveravamo postojanje prvog uslova i iza toga ako je on istinit onda se linkom šalju parametri sa vrednostima. Pa u ovom primeru se dobija dugme na koje kada kliknete menja mu se ispis ON ili OFF



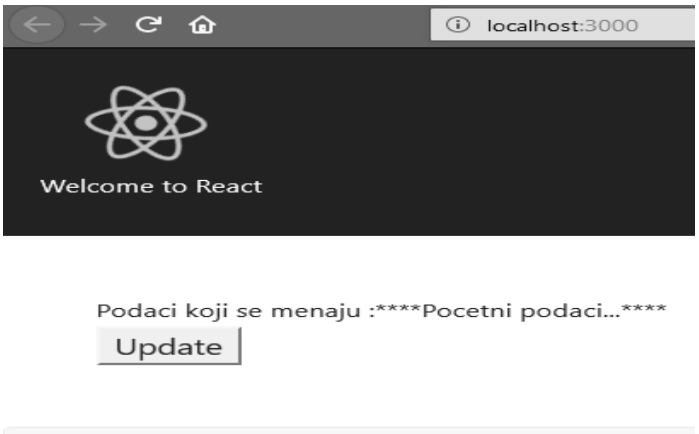
Jednostavni događaj(event)

U ovom delu prikazaću upotrebu događaja (events) u react aplikaciji.
U ovom primeru biće upotrebljena samo jedna komponenta gde ću dodati onClick događaj koji će poslužiti kao okidač za pozivanje funkcije updateState function jednostavnim klikom na dugme.

App.jsx

```
import React from 'react';

class App extends React.Component {
   constructor(props) {
      super(props);
      this.state = {
         data: 'Pocetni podaci...'
      };
      this.updateState = this.updateState.bind(this);
   };

   updateState() {
      this.setState({data: 'Pridodati podaci( updated)...'})
   }

   render() {
      return (
         <div style={{margin:"50px"}}>
            <h4>Podaci koji se menaju :****{this.state.data}****</h4>
            <button onClick = {this.updateState}>Update</button>
         </div>
      );
   }
}
```

export default App;



Upotreba fontawesome ikona

Potreba za dodacima i boljeg prikaza informacija na veb stranicama iziskuje potrebnu za malim sličicama, kojima se može prikazati više nego sa mnogo reči. Ali, sama njihova izrada nije jednostavna i laka, te su programeri osmislili olakšanja koja i u ovom slučaju nalaze se na sajtu https://www.npmjs.com/ gde postoji dodatak za ove ikonice koje se zovu font-awesome. Instalacija je ista kao za sve druge pakete sa ovog prostora sa:

npm install --save font-awesome

Uključivanje font-awesome može biti u index.js ili ako želite u fail gde ga upotrebljavate dodajući jednu od ove de linije koda ispod:

import '../node_modules/font-awesome/css/font-awesome.min.css';

ili

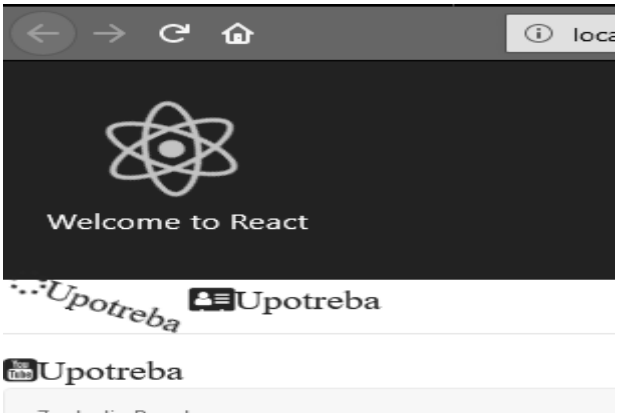
import 'font-awesome/css/font-awesome.min.css';

Kod komponente koja upotrebljava ove ikonice može biti ovakav, ali ako dodate u index.js fail onda ovaj red za njihovo uključivanje u drugim failovima nema potrebe da pisete. Razlog ovome je taj, što svi failovi komponenata koji se upotrebljavaju za izradu aplikacija povezuju se preko index.js. Standardno se povezuju na realni DOM element osim u slučajevima kada upotrebljavate neki drugi fail koji ima istu namenu kao i index.js.

```
import React from 'react';
import 'font-awesome/css/font-awesome.min.css';
```

```
export default class App extends React.Component{
   render() {
      return (
         <div>
          <i className="fa fa-spinner fa-spin">Upotreba </i>
          <i className="fa fa-address-card">Upotreba </i>
         <hr/>
         <i className="fa fa-youtube-square">Upotreba </i>
            </div>
       );
   }

}
```

Gornji kod daje ovakvu sliku na ekranu



Dodatno o ovim ikonicama

O instalcijama i upotrebama ovih ikonica možete videti na linku sajta dole niže.

https://github.com/FortAwesome/react-fontawesome

Pošto i postoje proširene mogućnosti za upotrebu postoje pored fontawesome i react-fontawesome ikone. Njih možete instalisati upotrebom donjih kodova :

```
npm i --save @fortawesome/fontawesome
npm i --save @fortawesome/react-fontawesome
npm i --save @fortawesome/fontawesome-free-solid
npm i --save @fortawesome/fontawesome-free-regular
```

Instalisanje može biti kao što prikazuje kod gore ali može se napisati i ovako
npm i --save @fortawesome/fontawesome @fortawesome/react-fontawesome @fortawesome/fontawesome-free-solid @fortawesome/fontawesome-free-regular

Znači, u jednom redu. Ovaj način pisanja koda za instalaciju paketa biće van često potreban u kasnijim vremenima kada više ovladate radom u reactu kada su vam potrebne u jednom dahu instalacije više dodatnih paketa.
Za deinstaliranje nekog paketa upotrebljava se npm uninstall kome se dodaje ime dodatnog paketa koje bi da se deinstalira iz aplikacije. Kako bi ste ih mogli upotrebljavati u vašim aplikacijama i ovaj primer koda vam može pomoći u tom nadstojanju.

```
import React, { Component } from 'react';
import FontAwesomeIcon from '@fortawesome/react-fontawesome'
import { faCheckSquare, faCoffee } from '@fortawesome/fontawesome-free-solid'
import './App.css';

class App extends Component {
  render() {
    return (
      <div className="App">
        <h1>
          <FontAwesomeIcon icon={faCoffee} />
        </h1>
      </div>
    );
  }
}

export default App;
```

Razmena vrednosti Props i State u react aplikacijama

Primer u nastavku pokazuje kako kombinovati stanje i props u aplikaciji. Funkcija unutarnjeg renderovanja postavljamo headerProp i contentProp koji se koriste u komponenti (child).

App.jsx

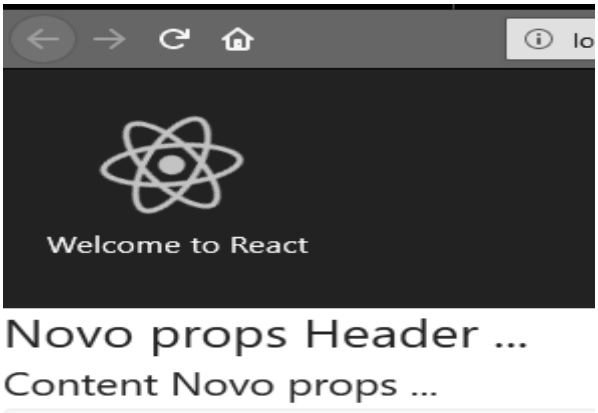
```
import React from 'react';

class App extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      header: "Header from props...",
      "content": "Content from props..."
    }
  }
```

```
  render() {
    return (
      <div>
        <Header headerProp = {this.state.header}/>
        <Content contentProp = {this.state.content}/>
      </div>
    );
  }
}

class Header extends React.Component {
  render() {
    return (
      <div>
        <h1>{this.props.headerProp}</h1>
      </div>
    );
  }
}

class Content extends React.Component {
  render() {
    return (
      <div>
        <h2>{this.props.contentProp}</h2>
      </div>
    );
  }
}

export default App;
```

main.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';

ReactDOM.render(<App/>, document.getElementById('app'));
```

Rezultat će opet biti isti kao u prethodna dva primera.



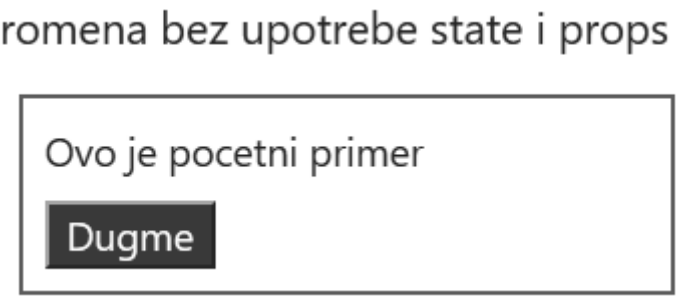
Statička primena postavljanja vrednosti za state i props

Ovo je jedna od specifičnosti u reactu gde se mogu dodavati vrednosti osobina iz faila komponente gde je pozvana pod komponenta. Pogledajmo jednostavan primer, gde u podkomponenti se izrađuje dugme sa jednim tekstom, koji se u daljem razvoja primera može zameniti prema potrebama aplikacije drugim tekstom. Da bi se video rad u ovu podkomponentu dodaću i događaj onClick koji će pokretati alert sa izvetštajem da je došlo do radnje, to jest da se mišem kliknuli na dugme.

```
import React from 'react';

// glavna ili parent komponenta polazno.js
export default class AppButton extends React.Component{

 render(){
    return(
       <div>
          <Dugme />
       </div>
    );
 }
}

// kod podkomponente

class Dugme extends React.Component{
   btnClik(){
      alert('Dugme je kliknuto');
   }
   render(){
      return(
         <div style={{border:" 2px solid green",width:"300px", margin:"20px",
```

```
                padding:" 10px" }}>
              <p> Ovo je pocetni primer</p>
              <button onClick={this.btnClik} className="btn-dark">
               Dugme
              </button>
            </div>
        );
    }
}
```

Ugledaćemo na ekranu



I kada kliknemo na njega dobijamo prikazan alert sa tekstom koji smo uneli. Međutim, element dugme nam je potreban u različitim delovima aplikacije, gde su nam potrebne neke izmene, ali ne i potreba da pišemo stalno novi kod za te namene. Pa ću zato napraviti još dva faila, gde će fail dugmeApp.js biti taj kojim ću pokupiti predhodni sa failom podkomponentaDugme.js. I sva tri postaviti u jedan folder kog sam nazvao dugme Obzirom da u konfiguraciji react aplikacije najbolje je da ostane sve što se radi u folderu src, tu ću prvo napraviti folder dugme i u njemu navedene failove i na kraju povezati fail dugmeApp.js importovanjem u fail App.js u folderu src.

Kod faila podkomponentaDugme.js

```
import React from 'react';

export default class Button extends React.Component {
      state={
         klasa1:this.props.klasa
      };

   btnClick() {
      alert(' je tekst koji');
   }

   render() {
```

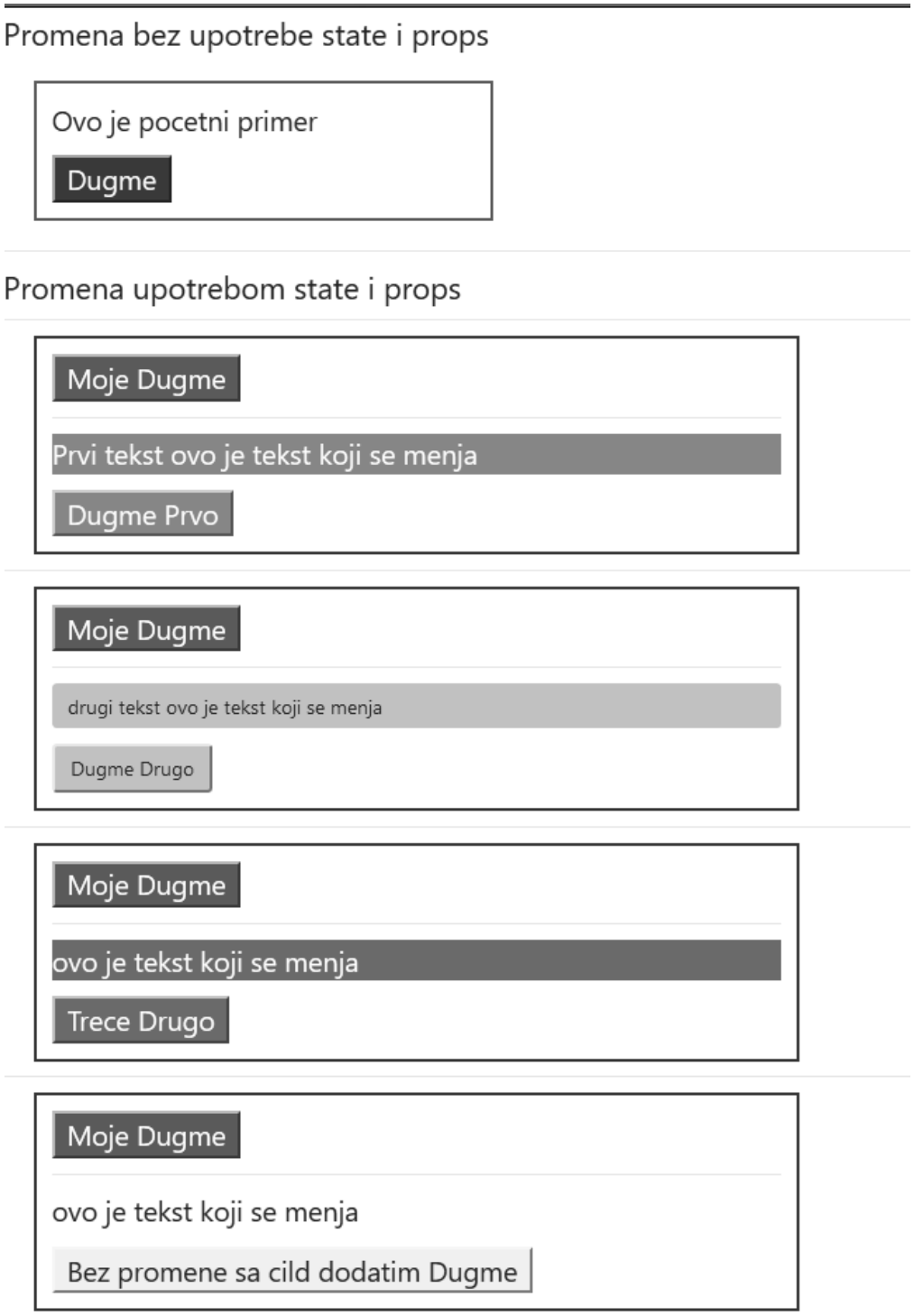
```
      return (
         <div style={{border:" 2px solid red",width:"500px", marginLeft:"20px",
               padding:" 10px" }}>
            <button onClick={this.btnClick} className="btn-danger" >
             Moje Dugme
            </button>
            <hr />
            <p className={this.state.klasa1} > {this.props.tekst}
                Ovo je tekst koji se menja
            </p>
            <button onClick={this.btnClick} className={this.props.klasa} >
             {this.props.naziv}
            </button>
         </div>
       );
    }
}
```

Kod faila dugmeApp.js

```
import React from 'react';
import Dugme from'./podkomponentaDugme';
import Dugme1 from'./polazno';

export default class AppButton extends React.Component {
      state={
         prvi: "Prvi tekst",
         drugi:" drugi tekst",
         klasa: "btn-info",
         klasa21: "btn-primary",
         klasa2: "btn-warning btn-lg",
         imeDugmeta1: " Dugme Prvo",
         imeDugmeta2: " Dugme Drugo",
         imeDugmeta3: "Trece Drugo",
         imeDugmeta: "Bez promene sa cild dodatim Dugme",

      };
   render() {
      console.log(this.state,"stanja iz dugme" );
      return (
         <div>
            <h2> Promena bez upotrebe state i props</h2>
            <Dugme1/>
            <hr />
```

```
            <h2> Promena upotrebom state i props</h2>
            <hr />
            <Dugme className="btn-success"  tekst={this.state.prvi}
               klasa={this.state.klasa} naziv={this.state.imeDugmeta1} />
             <hr />
            <Dugme tekst={this.state.drugi} style={{color:" dark-blue"}}
              klasa={this.state.klasa2} naziv={this.state.imeDugmeta2} />
             <hr />
            <Dugme klasa={this.state.klasa21} naziv={this.state.imeDugmeta3}/>
            <hr />
            <Dugme naziv={this.state.imeDugmeta}/>
            <hr />
          </div>
        );
    }
}
```

Ovaj kod daje ovakav prikaz na ekranu



Gde u prvom delu imate rad koda faila polazno.js a ispod njega rad koda failova dugmeApp.js i podkomponentaDugme.js.

Objašnjenje rada koda

Parent komponenta dugmeApp.js znači okuplja sve importovane failove i šalje ih u folder src failu App.js na dalju obradu. Pored toga u ovom failu su postavljena stanja kojima se utiče na props vrednosti u failu podkomponentaDugme.js. Početna stanja od vrezije v 16 se postavljaju u parent komponenti uz upotrebu reči state ili this.state odmah ispod koda class AppButton posle otvorene velike zagrade kada se radi o postavljanju inicijalnih vrednosti za stanja. Mada se ovaj kod može naći i u drugim delovima teksta koda izađenih komponenti.

```
export default class AppButton extends React.Component {
  state={
        prvi: "Prvi tekst",
        drugi:" drugi tekst",
        klasa: "btn-info",
        klasa21: "btn-primary",
        klasa2: "btn-warning btn-lg",
        imeDugmeta1: " Dugme Prvo",
        imeDugmeta2: " Dugme Drugo",
        imeDugmeta3: "Trece Drugo",
        imeDugmeta: "Bez promene sa cild dodatim Dugme",

      };
```

Ali, ako se namerava dinamička upotreba promene ovih stanja, potrebno je to ugraditi u konstruktor klase i obavezno pozvati funkciju super(); što će te videti u drugim primerima koda.
Osim toga ako namerava da kao što sam u ovom slučaju primenio i css klase, kao za sve drugo što primenjujete potrebno je da ima određen format, kako bi sam kod radio svoj posao. Te sam u ovom slučaju to postavio u obliku stringa, obzirom da se radi o statičkom pristupu dodeljivanja vrednosti.
Da bi bilo malo jasnije, vratimo se na html tagove i css klase, sa osvrtom na uključivanje toga u react kod aplikacije. Klasičan kod html-a i css-a je ovo:

```
<div class=" btn-danger btn-lg"> Neki sadržaj</div>
```

Znači ono što je potrebno za css klasu je tekst zatvoren navodnicima. Znači da mora biti u obliku stringa , ili skupa slovnih znakova. Kod upotrebe reacta može se upotrebiti potpuno isti kod sa malom izmenom

```
<div className=" btn-danger btn-lg"> Neki sadržaj</div>
```

Pošto on u svom kodu obzirom da je to u osnovi java skript ima reč class koja je namenjena deklaraciji klasa kojma se opisuju elementi, radnje i drugo u njoj, što rezultujući daje prikaz komponente koju pravimo. Da bi smo prilagodili te

tehnologije potrebno je prvo u html tag dodati java skript kod na potrebnim mestima, kojim ćemo pročitati određene vrednosti za stanja i proslediti ih podkomponentama ili child komponentama da ih samo one mogu pročitati, a ne i menjati. Java skript kod se unosi u html zatvoren velikim zagradama, dok se posebni stilovi za pojedinačne delove unose u JSON obliku zatvoreni kao što ste već videli dvostrukim velikim zagradama. Stanja (state) mogu slati perent ili roditeljske komponete upotrebom u formi this.state.naziv_stanja. Kao što se i vidi u ovom kodu:

```
<Dugme className="btn-success"
  tekst= {this.state.prvi}
  klasa={this.state.klasa}
  naziv={this.state.imeDugmeta1}
 />
```

Napomena: često ćete u tutorijalima susretati ovako napisan kod i to se čini zbog same preglednosti

Razlog da parent komonente mogu da šalju vrednosti stanja je taj što su one i vlasnici tih vrednosti. U njima je kao što će te videti moguće vršiti promene početnih vrednosti nekim drugim vrednostima. Dok su child samo namenjene da primaju poslate vrednosti stanja u obliku props (properties), što bi se moglo prevesti kao osobine. Što se i vidi u donjem kodu:

```
<hr />
<p className={this.state.klasa1} > {this.props.tekst} Ovo ispred je tekst
koji se menja </p>
<button onClick={this.btnClick} className={this.props.klasa} >
{this.props.naziv} </button>
```

Pošto je i to isto java skript kod i on je zatvoren u velike zagrade. Kao što vidite, vrednosti stanja koje su vezane za lokalnu upotrebu mogu se isto tako dodati podkomponenti, kao njena licna ili kao poslata od strane same komponete. Jer, html tag parametra upotrebljava za svoj opis lokalno stanje svoje klase koje je props od faila AppDugme.js

```
export default class Button extends React.Component {
    state={
       klasa1:this.props.klasa
    };
```

Taj red koda html tag pargraf ima između svog otvorenog i zatvorenog taga dodati java skript i tekst koji se klasično dodaje u praragraf tagu. Ako pogledamo taj deo koda

{this.props.tekst}

i iznad njega nevedeni kod

```
<Dugme className="btn-success"
  tekst= {this.state.prvi}
  klasa={this.state.klasa}
  naziv={this.state.imeDugmeta1}
 />
```

Primetno je ponavljanje reči tekst. Mehanizam rada je sledeći:

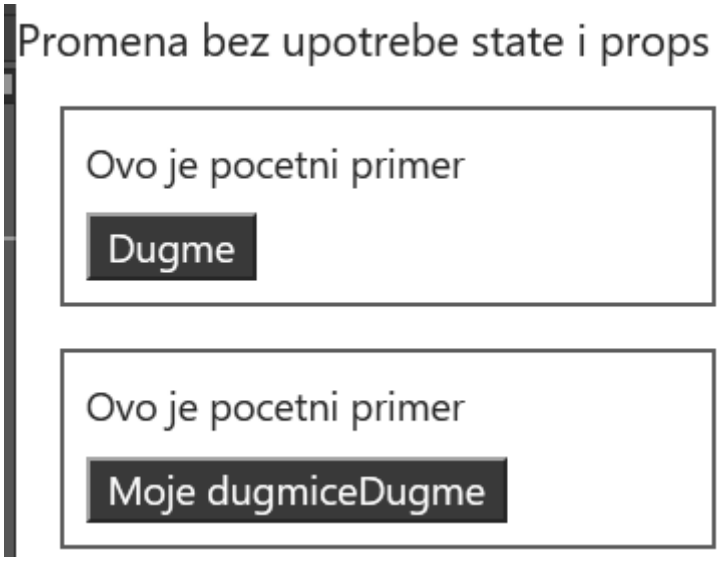
vrednost za stanje se pruzima sa this.state.prvi iz vrednosti stanja parenta on se izjednačava sa nazivom kojeg upotrebljavmo za vrednost props pod komponente. Znak tačke između ovih da kažemo službenih reči određuje šta u nastavku sledi, odnosno poslednji u nizu je praktično onaj koji odlučuje. Slično slučaju kod putanje foldera.

Napomena: upotreba znaka tačke ima identično značenje kao kod php-a

Ali ono što je omogućeno kod reacta da jedan isti kod može da se upotrebi koliko je to potrebno i da se izrađenim elementima stranica svakim pozivom mogu dodati novi izgledi. Što i prikazje kod dole niže :

```
<div>
    <h2> Promena bez upotrebe state i props</h2>
    <Dugme1/>
    <hr />
    <h2> Promena upotrebom state i props</h2>
    <hr />
    <Dugme className="btn-success"  tekst={this.state.prvi}
       klasa={this.state.klasa} naziv={this.state.imeDugmeta1} />
     <hr />
    <Dugme tekst={this.state.drugi} style={{color:" dark-blue"}}
      klasa={this.state.klasa2} naziv={this.state.imeDugmeta2} />
     <hr />
    <Dugme klasa={this.state.klasa21} naziv={this.state.imeDugmeta3}/>
    <hr />
    <Dugme naziv={this.state.imeDugmeta}/>
    <hr />
 </div>
```

U prvom slučaju je prikazano dugme gde je sve već određeno u podkomponenti. Ali i takvom izgledu se mogu promene vršiti da dođe do ovog slučaja na slici




```
import React from 'react';

export default class AppButton extends React.Component{

 render(){
    return(
       <div>
          <Dugme />
          <Dugme > Moje dugmice</Dugme>
       </div>
    );
 }
}

class Dugme extends React.Component{

   btnClik(){

      alert('Dugme je kliknuto');
   }
   render(){
      return(
         <div
              style={{border:" 2px solid green",width:"300px",
                      margin:"20px", padding:" 10px"   }}>
            <p > Ovo je pocetni primer</p>
            <button onClick={this.btnClik}  className="btn-dark">
            {this.props.children}Dugme
```

```
            </button>
          </div>
        );
    }
}
```

A, bilo je samo potrebno upotrebiti ponovno pozivanje iste pod komponete u rastavljenom obliku.

```
<Dugme > Moje dugmice</Dugme>
```

Gde je tekst ” Moje dugmence”, praktično prestavlja njegovu decu, to jest vrednost za njihovu osobinu

```
(this.props.children)
```

Što je i uneseno u kod podkomponete Dugme

```
<button onClick={this.btnClik}  className="btn-dark">
{this.props.children}Dugme</button>
```

Što je i prirodno da deca naleđuju svoje roditelje- sve njihove osobine.

React HOC-Primer prvi

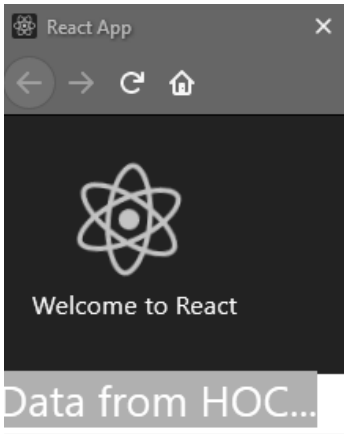
Komponete višeg reda (HOC) su java skript funkcije koje se upotrebljavaju za dodavanje dodatnih funkcionalnosti postojećoj komponenti. Ove funkcije su čiste, što znači da primaju podatke i vraćaju vrednosti prema tim podacima. Ako se podaci promene, funkcije višeg reda se ponovo pokreću sa drugačijim unetim podatkom. Ako, želimo da ažuriramo našu povratnu komponentu, nemoramo promeniti naš HOC, sve što je potrebno u tom slučaju je da se promene podaci koje naša funkcija upotrebljava.
Komponete višeg reda (HOC) praktično okružuje „normalne“ komponente i pruže joj dodatne ulazne podatke. To je zapravo funkcija koja uzima jednu komponentu i vraća drugu komponenti koja obrađuje prvobitnu.
Dole ispod sledi kod jednostavnog primera, koji će omogućiti razumevanje osnovnog koncepta.

MyHOC je funkcija višeg reda, koja se upotrebljava samo za prenošenje podataka u MyComponent. Ova funkcija uzima MyComponent menjajući je sa newData i vraćajući je kao takvu prikazanu na ekranu

```
import React from 'react';

let newData = {
```

```
    data: 'Podaci iz HOC...'
};

let MyHOC = (ComposedComponent)=> class extends React.Component {
    componentDidMount() {
        this.setState({
            data: newData.data
        });
    }
    render() {
        return <ComposedComponent {...this.props} {...this.state} />;
    }
};

class MyComponent extends React.Component {
    render() {
        return (
            <div>
                <h1>{this.props.data}</h1>
            </div>
        )
    }
}

export default MyHOC(MyComponent);
```

Napomena: sličan kod kao ovaj najdonji red piše se kada upotrebljavamo withRouter komponentu dodatnog react paketa react-router-dom. Prilikom pokretanja koda videćemo da su podaci prošli u MyComponent.



Napomena: Komponete višeg reda (HOC) mogu se upotrebiti za različite funkcionalnosti u aplikacijama, jer su to funkcije koje prestavljaju suštinu funkcionalnog programiranja. Čestom primenom ovih funkcija uvidećete da je lakše održavati i nadograđivati aplikacije.

Malo složeniji primer HOC

Komponenta višeg reda je funkcija koja preuzima komponentu i vraća novu komponentu. Komponenta višeg reda (HOC) je napredna tehnika u React.js za ponovnu upotrebu komponentne logike. HOC nisu dielovi React API . Oni su uzorci koji proizlaze iz Reactove kompozicijske prirode. Komponenta se pretvara u korisničko okruženje UI, a komponenta višeg reda pretvara se u drugu komponentu. Primeri HOC-a su Redux spojen sa Relay koji izrađuje konteiner (Relay.createContainer). Čiji kod izgleda ovako

```
Relay.createContainer(
  connect(mapStateToProps)(BaseComponent),
  {
    initialVariables: { ... },
    fragments: { ... },
  }
)
```

No u ovom primeru nećemo to obraditi, već samo HOC, pa ćemo izraditi novu aplikaciju.

1: Izradićemo React.js projekt kao i u primerima ranije

```
npm install -g create-react-app

create-react-app my-app

cd my-app
npm start
```

Ali možemo i u predhodnoj automacki izrađenoj aplikaciji u src folderu napraviti jedan folder i u njemu smestiti sve ove failove koji su nam potrebni u ovom primeru i tako ih importovati u već napravljenu aplikaciju u App.js fail te već napravljene aplikacije. Znači ovako

```
import Ciklusi from './dodaci/HOC/drugi-primer/App';
```

i u delu render/return napisati jednostavno ovaj XTML tag

```
<Ciklusi/>
```

2: Napravićemo u src/dodaci/HOC/drugi-primer fail HOC.js.

```
// HOC.js

import React, {Component} from 'react';

export default function Hoc(HocComponent){
   return class extends Component{
      render(){
         return (
            <div>
               <HocComponent></HocComponent>
            </div>
         );
      }
   }
}
```

Donji kod prikazuje njeno uključivanje u koji je u istom folderu kao i HOC.js u App.js fail.

```
// App.js

import React, { Component } from 'react';
import Hoc from './HOC';

class App extends Component {

  render() {
    return (
      <div>
        Higher-Order Component
      </div>
    )
  }
}
App = Hoc(App);
export default App;
```

Da pojasnim kod ispred. Prvo smo napravili HOC funkciju u failu HOC.js, tako da ona prihvata jedan argument kao komponentu. To je u našem slučaju komponenta App

```
App = Hoc(App);
```

Tako da je komponenta omotana unutar druge react komponente, tako da je možemo mijenjati. Zbog toga je primarna primena komponenti Višeg reda (HOC)

u react aplikacijama. Pretpostavimo da imamo jednu komponentu koja samo navodi ID i njeno ime. Dakle, prvo ćemo stvoriti jednu komponentu u obliku faila StockList.js

```
// StockList.js

import React, { Component } from 'react';
import TableRow from './TableRow';

class StockList extends Component {
   constructor(props) {
      super(props);
      this.state = {
        stocks: [
         { id: 1, name: 'Windows' }, { id: 2, name: 'Linux' },
         { id: 3, name: 'SrbinTms' }
        ]
      };
    }
    tabRow(){
     if(this.state.stocks instanceof Array){
      return this.state.stocks.map(function(object, i){
          return <TableRow obj={object} key={i} />;
      })
     }
    }
    render() {
     return (
        <div className="container">
        <table className="table table-striped">
          <thead>
            <tr>
              <td>Stock Name</td>
              <td>Stock Price</td>
            </tr>
          </thead>
          <tbody>
            {this.tabRow()}
          </tbody>
        </table>
     </div>
     );
    }
}
```

```
export default StockList;
```

Kojoj smo dodali tabelu kojoj za sada nedostaju redovi koje ćemo napraviti sledećim failom TableRow.js.

```
// TableRow.js

import React, { Component } from 'react';

class TableRow extends Component {
   render() {
      return (
         <tr>
            <td className="border-danger" >
               {this.props.obj.id}
            </td>
            <td>
               {this.props.obj.name}
            </td>
         </tr>
      );
   }
}

export default TableRow;
```

Gde, će ova komponenta prihvatati vrednosti stanja i prikazivati ih kao props. Da bi smo prikazali ono što prikazje komponenta StockList.js potrebno je i nju povezati sa App.js failom.

```
// App.js

import React, { Component } from 'react';
import StockList from './StockList';

class App extends Component {

  render() {
    return (
      <div>
        <StockList/>
      </div>
    )
  }
```

```
}

export default App;
```

Nakon što smo sačuvali promene na lokalhostu u pretraživaču na http://localhost:3000/ ugledaćemo tabelu sa podacima koje je poslala komponenta iz faila StockList.js. Ponovićemo još jednom isti postupak i napraviti fail UserList.js u kome ćemo pozvati opet fail TableRow.js i dodeliti nove podatke.

```
// UserList.js

import React, { Component } from 'react';
import TableRow from './TableRow';

class UserList extends Component {
   constructor(props) {
      super(props);
      this.state = {
        users: [
          {
            id: 1,
            name: 'Moje Ime'
          },
          {
            id: 2,
            name: 'TudjeIme'
          },
          {
            id: 3,
            name: 'Stranica'
          }]
      };
    }

    tabRow(){
      if(this.state.users instanceof Array){
        return this.state.users.map(function(object, i){
            return <TableRow obj={object} key={i} />;
        })
      }
    }
    render() {
      return (
          <div className="container">
```

```
          <table className="table table-striped">
            <thead>
              <tr>
                <td>ID</td>
                <td>Name</td>
              </tr>
            </thead>
            <tbody>
              {this.tabRow()}
            </tbody>
          </table>
        </div>
        );
      }
}
export default UserList;
```

Na ovaj način dobili smo dve tabele sa podacima, koji su prikazani sa po dva polja Id i name. Ono šta možemo učiniti je da napravimo jednu komponentu višeg reda, gde prolazimo sa obe komponente kao argumentima kada je to potrebno kroz HOC.
Napomena: ovaj red koda može se napisati

```
   return this.state.users.map(function(object, i){
            return <TableRow obj={object} key={i} />;
         })
```

i na ovaj način

```
         return this.state.users.map( (object, i) =>{
            return <TableRow obj={object} key={i} />;
         })
```

Kod faila HOC.js

```
// HOC.js

import React, {Component} from 'react';

export default function Hoc(HocComponent, data){
   return class extends Component{
      constructor(props) {
         super(props);
         this.state = {
            data: data
```

```
        };
    }
    render(){
        return (
            <HocComponent data={this.state.data} {...this.props} />
        );
    }
  }
}
```

Kao što vidite ovaj fai eksportuje jednu defoltnu funkciju koja modifikuje komponentu, pri čemu uzima dva argumenta za podatke

Komponete sa podacima

```
// StockList.js

import React, { Component } from 'react';
import TableRow from './TableRow';

class StockList extends Component {

  tabRow(){
    if(this.props.data instanceof Array){
      return this.props.data.map(function(object, i){
        return <TableRow obj={object} key={i} />;
      })
    }
  }
  render() {
    return (
      <div className="container">
        <h2>Iz liste stanja</h2>
        <table className="table table-striped">
          <thead>
          <tr>
            <td>ID</td>
            <td>Name</td>
          </tr>
          </thead>
          <tbody className="btn-success">
          {this.tabRow()}
          </tbody>
        </table>
```

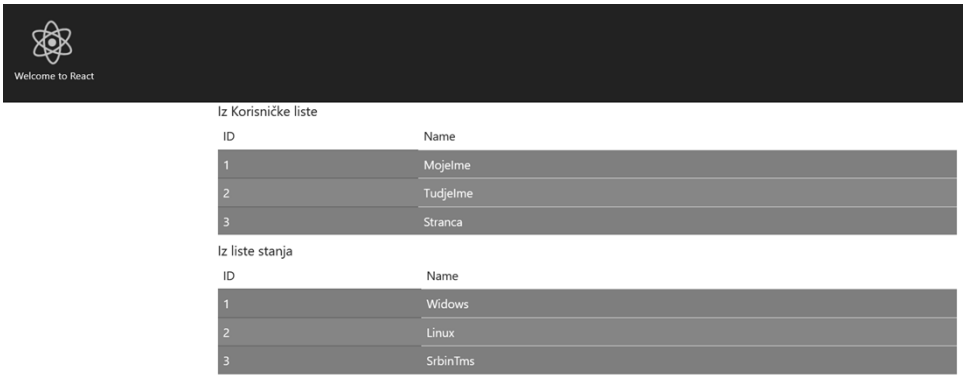
```
            </div>
        );
    }
}
export default StockList;

// UserList.js

import React, { Component } from 'react';
import TableRow from './TableRow';

class UserList extends Component {

    tabRow(){
        if(this.props.data instanceof Array){
            return this.props.data.map(function(object, i){
                return <TableRow obj={object} key={i} />;
            })
        }
    }
    render() {
        return (
            <div className="container">
                <h2>Iz Korisničke liste</h2>
                <table className="table table-striped">
                    <thead>
                    <tr>
                        <td>ID</td>
                        <td>Name</td>
                    </tr>
                    </thead>
                    <tbody className="btn-info">
                    {this.tabRow()}
                    </tbody>
                </table>
            </div>
        );
    }
}
export default UserList;
```

Sve ove komponente su pozvane u App.js fail, pre nego što je pozvana funkcija higher-order-component. Pored ovog, App.js je vlasnik i podataka koji se

upotrebom svega prikazuju u failovima StockList.js i UserList.js koji su smešteni u JSON formatu i dodati kao vrednost konstanti const StocksData i const UsersData.

```
// App.js

import React, { Component } from 'react';
import StockList from './StockList';
import UserList from './UserList';
import Hoc from './HOC';

const StocksData = [
    {
      id: 1,
      name: 'Widows'
    },
    {
      id: 2,
      name: 'Linux'
    },
    {
      id: 3,
      name: 'SrbinTms'
    }];
const UsersData = [
    {
      id: 1,
      name: 'MojeIme'
    },
    {
      id: 2,
      name: 'TudjeIme'
    },
    {
      id: 3,
      name: 'Stranca'
    }];

const Stocks = Hoc(
    StockList,
    StocksData
);
const Users = Hoc(
    UserList,
    UsersData
```

```
);
class App extends Component {
   render() {
      return (
         <div>
            <Users/>
            <Stocks/>
         </div>
      )
   }
}

export default App;
```



Kao što vidite na slici, na ekranu vide se podaci iz < Users /> i <Stocks/> prikazani render() funkcijom.

Zaključak: Primarna upotreba komponente višeg reda je poboljšanje ponovljivosti pojedinih komponenti u više modula ili komponenti. Stoga možemo koristiti različite komponente kako bismo poboljšali komponente.

Životni ciklusi komponenti

Pošto sve ima svoje vreme i mesto, tako isto i komponente u react aplikacijama imaju pored tih osobina i mogućnost da budu priključene, isključene, da im se dodaju novi podaci i drugo što nam potrebno zahtevom izrade aplikacije. Drugo o čemu se vodilo računa u reactu je to da sa jednostavnijim kodom mogu obaviti što sigurnije potrebne zadatke.
Komponente se mogu formirati da rade dok je to potrebno u aplikaciji, i da posle se isključe kada nisu više potrebne. Pri čemu mogu da nam urade potrebne zadatke. Na primer da izvrše prijem podataka i kada nam to nije potrebno da se u drugom delu koda isključe. Ili da prime neko novo stanje. Takav pristup omogućava lakši rad sa serverskim podacima i operacijama (GET,POST,PUT, DELETE i sl.). Ono što treba imati na umu je da reč Will u sastavu označava da se izvršava pre i nego

što se prikaže(render-uje) i to se izvršava na obe strane (klijentovoj i serverskoj), dok Did označava nešto što se izvršava samo kod klijenta i to samo posle izvršenja rendera. Navedene su u ovom delu sa kraćim objašnjenjem ono što se najčešće upotrebljava, mada postoje i one za brzo dodavanje novih vrednosti što možete videti u kodu dole niže

```
import React from 'react';
class App extends React.Component{
  constructor(){
    super();
    this.forceUpdateHandler = this.forceUpdateHandler.bind(this);
  };

  forceUpdateHandler(){
    this.forceUpdate();
  };

  render(){
    return(
      <div>
        <button onClick= {this.forceUpdateHandler} >FORCE UPDATE</button>
        <h4>Random Number : { Math.random() }</h4>
      </div>
    );
  }
}

ReactDOM.render(<App />, document.getElementById('app'));
```

Samo izvršavanje tokom rada komponente u reactu ide ovim redom: konstruktor, componentWillMount, componentDidMOunt, render.
Ali, za sada biće prikazane osnovne karakteritike, sa manjim objašnjenjma dole navedene.

componentWillMount(){}
Ova metoda se upotrebljava kako bi uradila promena na komponenti, ali će biti uvek prvo dodata DOM-u pre upotrebe. Izvršava se pre renderovanja i na serverskoj i na klijentskoj strani, pa kao takva je potrebno da se u nju smeštaju podaci o vrednostima stanja

componentDidMount(){}
Izvršava se pre prvog renderovanja samo na klijentskoj strani. Pa se ovde AJAX zahtevi i DOM ili stanja ažuriraju, ako se dogode. Ovaj metod se takođe koristi za integraciju sa drugim JavaScript frejmvorcima i bilo kojim drugim sa odloženim izvršavanjem kao što su setTimeout ili setInterval. Koristimo ga da bi ažurirali

stanje kako bi mogli da pokrenemo druge metode životnog ciklusa. Jako je upotrebljiva za očitavnje podataka. Pogodna za rad sa API

componentWillUnMount(){}
Suprotna od predhonih jer ona upavo okončava pozvane react funkcije od ovih funkcija u ovom delu, tačnije poziva se nakon što se komponena odvoji od dom-a. Pored toga ona briše listu događaja (event listner), tajmere, redux stanja i stopira sokete.

shouldComponentUpdate(){
/*treba da vrati true ili false vrednosti. Ovim će se odrediti da li će se komponenta ažurirati ili ne. Po default-u je postavljen na true. Ako ste sigurni da koponenta ne treba da renderuje nakon što su stanje ili props ažurirani, možete vratiti false vrednosti. */
console.log('shouldComponentUpdate');
return true;
}

componentWillRecesiveProps(){}
Metoda se poziva čim su props ažurirani prije nego što je bilo koji drugi render pozvan. Pozvan je iz setNewNumber čim je ažurirano stanje.

componentWillUpdate(nextProps,nextState){
/* metoda koja će biti pozvana pre nego što joj izmenimo props novim stanjem. tj. poziva se pre renderovanja*/
return nextProps.id !== this.props.id;
}

componentDidUpdate(prevProps, prevState){
/* metoda koja će biti pozvana u trenutku kada joj predhodnom props-u dodamo predhodno stanje, znači da je uvek pozvana odmah posle renderovanja*/
console.log('PrevProps', prevProps);
console.log('PrevState', prevState);
console.log('**componentDidUpdate**');
}

Bilo koju navedenu metodu da primenite dok učite da bi ste shvatili kako to radi, najbolji je način upotrebiti jedan dogadjaj (event) i postavljanje nove vrednosti stanja (this.setState), što se može i propratiti u inspekt načinu rada brovsera koji imate na računaru. Mada, se umesto konzole može upotrebiti i alert. Da bi to mogli da testirate, možete upotrebiti donji kod, ali pre toga pogledajmo šta je potrebno da imamo da bi smo mogli da proverimo i utvrdimo kako ovi metodi rade.

1. potrebno je napraviti klasu sa standardnim kodom
2. postaviti u delu konstruktora polazno stanje this.state={ data: 0}
3. napraviti funkciju za dogadjaj (event). Uobičajeno je za ovo upotrebiti onClick
4. u delu gde pokrećemo neki od metoda životnih ciklusa komonenti padi provere da li je događaj pozvan upotrebiti kod console.log('Dogadjaj je pozvan'); ili ako je lakse alert('Dogadjaj je pozvan'); samo u ovom slučaju treba posle pojave alerta klikuti na dugme ok u prozoru koji se pojavljuje
5. u return delu klase pošto se ovim menjaju vrednosti stanja u vekim zagradama između div tagova uneti ovaj kod {this.state.data}
6. postaviti dugme i upotrebiti tag dugme (button) za pozivanje događaja

To je ono najmanje što je potrebno prilikom učenja o životnim ciklusima komponenti

componentWillUnMount(){
/*u ovom delu može se upotrebiti da se vrati recimo prazan div tag return kao odgovor na događaj onClick*/
}

Uobičajen kod kojim bi se moglo prikazati delovanje i rad navedenih je ovaj

Fail ZivotniCiklus.js

```
import React from 'react';

export default class App extends React.Component {
   constructor(props) {
      super(props);
      this.state = {
         data: 0
      };
      this.setNewNumber = this.setNewNumber.bind(this)
   };

   setNewNumber() {
      this.setState({data: this.state.data + 1})
   }
   render() {
      return (
         <div>
            <button onClick = {this.setNewNumber}>INCREMENT</button>
            <Content myNumber = {this.state.data}/>
         </div>
```

```
        );
    }
}

class Content extends React.Component {

    componentWillMount() {
        console.log('Component ce biti prikljucena pre upotrebe MOUNT!')
    }
    componentDidMount() {
        console.log('Component bice prikazana Posle prikljucenja DID MOUNT!')
    }
    componentWillReceiveProps(newProps) {
        console.log('Component WILL RECIEVE PROPS!')
    }
    shouldComponentUpdate(newProps, newState) {
        return true;
    }
    componentWillUpdate(nextProps, nextState) {
        console.log('Component WILL UPDATE!');
    }
    componentDidUpdate(prevProps, prevState) {
        console.log('Component DID UPDATE!')
    }
    componentWillUnmount() {
        console.log('Component WILL UNMOUNT!')
    }
    render() {
        return (
            <div>
                <h3>{this.props.myNumber}</h3>
                <button onClick={this.componentWillUnmount.bind(this)}>
Iskljucenje</button>
            </div>
        );
    }
}
```

Kod sa dugmetom prikazuje direktno pozivanje određene komponente, ali ako dodate u failu koji poziva ovaj kod faila ZivotniCiklus.js možete u vremenski ograničiti vreme upotrebe ovih i drugih komponenti.

a index.js fajl izgleda ovako:

```
import React from 'react';
import ReactDOM from 'react-dom';

import App from './App.jsx';

ReactDOM.render(<App/>, document.getElementById('app'));
setTimeout(() => {

  ReactDOM.unmountComponentAtNode(document.getElementById('app'));}, 
10000);
```

Rezultat rada se može videti u konzoli, slika dole, internet pretraživača Inspect modu (izborom u iskačućem meniju prilikom pritiska na desni taster miša ili na F-12).

```
Component WILL RECIEVE PROPS!
Component WILL UPDATE!
Component DID UPDATE!
```

ili

```
Component WILL MOUNT!
Component DID MOUNT!
```

u zavisnosti koja je od komponenti trenutno aktivna. U ovom primeru koda metoda componentDidMount upotrebljena je za prikaz podataka korisnika u obliku liste:

```
import React, { Component } from "react";

export default class Posts extends Component {
  constructor() {
    super();
    this.state = {
      users: []
    };
  }
  componentDidMount() {
    let dataURL = "http://localhost:80/server/test.php";
/* umesto ovog linka može se upotrebiti JSON data  ovako
   let dataURL = "putanja do faila/data.json";
*/
      fetch(dataURL)
          .then(response => response.json())
          .then(response => {
              console.log(response);
```

```
            this.setState({ users: response.users});
        });
  }

  render() {
          console.log(this.state);
    let users = this.state.users.map((user, index) => {
      return (
        <li key={index} style={{border:"2px solid red"}}>
          <h2> iD: {user.id}</h2>
          <h2>Name: {user.name}</h2>
          <h2>Email: {user.email}</h2>
          <h2>Title: {user.title}</h2>
          <h2>Poruka: {user.body}</h2>
           <hr />
           <br />
        </li>
      );
    });

    return (
      <div>
        <h2>Lista korisnika</h2>
        <ul style={{left:"150px"}}>{users}</ul>
      </div>
    );
  }
}
```

Primer koda

U ovom primeru koda možete videti kada se koja komponeta životnog ciklusa u react aplikacijama izvršava i kako radi.

```
import React from "react";

class Header extends React.Component{
    componentWillReceiveProps=()=>{
        console.log('Komponenta iz Hedera za prijemnovog Propsa');
    };

    render(){
        return(
          <div>
             <h1> heder</h1>
```

```
            <p>Ime : {this.props.username}</p>
          </div>
        );
    }
}


export default class Home extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            username: 'moje ime'
        };
    }
     componentWillUnmount(){
        console.log('Komponneta Will je iskljucena');
    };
    componentWillReceiveProps=()=>{
        // pozivi API, fetch i sl.
        console.log('Komponneta za prijemPropsa');
    };
    componentDidMount=()=>{
        // pozivi API, fetch i sl.
        console.log('Komponneta je Did prikacena');
    };
    componentWillMount=()=>{
        console.log('Komponneta Will je prikacena');
    };
    componentWillUpdate=()=>{
        // pozivi API, fetch i sl.
        console.log('Komponneta je Willupdate prikacena');
    };
    componentDidUpdate=()=>{
        // pozivi API, fetch i sl.
        console.log('Komponneta je Didupdate prikacena');
    };

   /* upotreba shuldComponetnUpdate moze se upotrebiti za proveru ako je ponovo
      uneto jednako unesenom onda ce se onemoguciti unos ili akcija*/
    shouldComponentUpdate(nextProps, nextState){
        console.log('Props:', nextProps);
        console.log('State:', nextState);
        // uslov
        if (this.state.username !== nextState.username ){
```

```
            console.log('rezultat je true');
            this.setState({ username: nextState.username});
            return true;
        }else{
            console.log('rezultat je false');
        }
        return false;
    }

    updateKorisnika=()=>{
        this.setState({username:'xcvxcvx ime'});
        console.log('Komponenta je updateKorisnika
                        prikacena'+this.state.username);
    };

    render() {
        console.log(this.props,"vraceni props" );
        return(
            <div className="container">
                <h2>Ovo je pocetna index stranica </h2>
                <h1>Ime: {this.state.username} </h1>
                <hr/>
                <Header username={this.state.username}/>
                <button onClick={this.updateKorisnika.bind(this)}> Promena</button>
            </div>
            )
    }
}
```

Ono što je primetno je to da se prvo izvršavaju komponente sa oznakom Will, a kasnije sa Did.

Izrada tabela u react aplikacijma

Tabele su komponente kojima je moguće prikazati različite podatke smeštene u bazama podataka. Ali ih uz manju prepravku možemo upotrebiti dosta korisno za formiranje različitih formi, kao i dodatkom dugmadi za brisanje, uredjivanje i dodavanje novih podataka u već postojećom. Upotrebom css-a u njima možemo pozicionirati i ostalo što nam potrebno.
U ovom slučaju imamo komponentu tabela koja se sastoji iz dva faila. U prvom failu se formiraju broj kolona, čiji sadržaj biće ispisan u zgalavlju tabele (th) u JSON formatu, upotrebom ključa i labele. U sledećem delu koda formiraćemo podatke koji se prikazuju u ćelijama tabele (td).
Ovi podaci mogu biti u sastavu nekog spoljnjeg JSON faila ili preuzeti putem php-a iz baze, o čemu će biti reči u poslednjem delu knjige.

fail tabelaApp.js

```
import React from 'react';
import Table from"./tabela";

// formiranje broja kolona u tabeli
let cols = [
    { key: 'firstName', label: 'First Name' },
    { key: 'lastName', label: 'Last Name' },
    { key: 'email', label: 'Email' },
    { key: 'adresa', label: 'Adresa' }
];

// padaci koji će se ispisati u tabeli
let data = [
    { id: 1, firstName: 'John', lastName: 'Doe' },
    { id: 2, firstName: 'Clark', lastName: 'Kent' },
    { id: 3, firstName: 'MIka', lastName: 'Perić' },
    { id: 4, firstName: 'Laza', lastName: 'Mikic' },
    { id: 5, firstName: 'Lazar', lastName: 'Lazic' },
    { id: 6, firstName: 'Pera', lastName: 'Lazarevic' }
];

export default class AppTabela extends React.Component{

   render(){
      return(
         <div>
            <Table cols={cols} data={data} />
         </div>
      );
   }
}
```

Pošto je fail AppTabela.js roditeljski fail u odnosu na tabela.js on je vlasnik odataka koje prosledjuje porebne podatke ovim kodom

```
         <div>
            <Table cols={cols} data={data} />
         </div>
```

failu tabella..js u obliku java skript koda koji predaje vrednosti u promenjive cols i data, koje se upotrebljavaju u podkomponenti tabela.js da bi se odredio broj kolona i redova sa podacima, kao i ispis tih podataka u novo stvorenoj tabeli.

Fail tabela.js

```
import React from 'react';

export default class Table extends  React.Component{

//generisanje zaglavlja tabele
   generateHeaders(){
      let cols = this.props.cols;  // [{key, label}]

      // Generisanje (th) ćelija
      return cols.map((colData)=> {
         return <th key={colData.key}> {colData.label} </th>;
      });
   }

// generisanje redova tabele
   generateRows(){
      let cols = this.props.cols,  // [{key, label}]
         data = this.props.data;
      return data.map((item) =>{

         // postavljanje kolona upotrebom map
         let cells = cols.map(function(colData) {

            // colData.key može biti i  "firstName"
            return <td> {item[colData.key]} </td>;
         });
         return <tr key={item.id}> {cells} </tr>;
      });
   }

   render(){
      let headerComponents = this.generateHeaders(),
         rowComponents = this.generateRows();
      return (
         <table className="table-bordered" style={{marginLeft:"50px"}}>
            <thead className="alert-secondary"> {headerComponents} </thead>
            <tbody className="alert-info"> {rowComponents} </tbody>
         </table>
      );
   }
}
```

Ovaj kod proizvodi ovakav prikaz

First Name	Last Name	Email	Adresa
John	Doe		
Clark	Kent		
MIka	Peric		
Laza	Mikic		
Lazar	Lazic		
Pera	Lazarevic		

Samim ovakvim pristupom kod fail tabela.js može se upotrebiti na više mesta sa različitim sadžajem, brojem kolona i redova, bez pravljenja novog koda za njega.

Ucitavanje podataka iz data.Json

Da bi se mogli postavljati podaci iz ovog faila u reactu se upotrebljavaju dva faila. Ali može se upotrebiti i spoljni fail sa podacima koji se može smesiti u istom direktoriju ili u nekom od direktorijuma projekta.

Kod faila App.js kod je dole nize :

```
import React, {Component} from 'react';
import Proba from './pomocna'

class App extends Component{

render(){
   const user = {
   name: "MIka",
   age: "25",
   hobbi: ["sport", "riba"]
   };
   return (
      <div className="App-podaci">
         <h4>
            Prikaz podataka iz ovog puta konstante , ali
            moze biti i iz data.json posebnog faila
         </h4>
         <p>
            Citanje samo polja ime iz json objekta
         </p>
         <Proba name={this.props.name} user={user}/>
         <hr/>
```

```
          <p>
             Citanje kompletnog json objekta
          </p>
          <Proba user={user} />
       </div>
       );
   }
}

export default App;
```

<Proba user={user} /> način povezivanja JSON podataka , gde se podaci iz konstante prosleđuju u promenjivu user. Ovi podaci se

Drugi se zove pomocna.js

```
import React from 'react';

export default class Pomocna extends React.Component{
   render(){
      console.log(this.props);
      const text ="neki tekst";
      return(
         <div style={{width:"300px"}}>
            <h5> Ocitavanje na ovaj nacin</h5>
            <p style={{color:"yellow" }} >{text}
              odavde pocine citanje podataka
            </p>
            <h3>Ime korisnika je   <br/>{this.props.user.name} ,--- Godine
               <br/>{this.props.user.age}
            </h3>
            <p style={{color:"white" }}>
               <br/>- Citanje samo podataka korisnickog imena-
               {this.props.user.name}
            </p>
            <p style={{color:"blue" }}>
              - Citanje samo podataka starosti korisnickog --
               {this.props.user.age}
            </p>
            <div>
               <h3>Citanje podataka iz niza Hobi</h3> <br/>
               <ul style={{color:"green" }}>
                  {this.props.user.hobbi.map((hobbi,i)=><li key={i}>{hobbi}</li>)}
               </ul>
```

```
            </div>
            <hr/>
          </div>
      );
   }
}
```

Prvi fail App.js poziva fail pomocna.js, kome prosledjuje podatake koji su smeštenu u primeru u obliku kontante koja sadrži podatke u obliku JSON. Dok, u failu pomocna.js se pored prijema podataka vrši i njihovo formatiranje . Da bi se uputili podaci osim importovanja potrebnog faila kome se oni upućuju potrebno je i povezati izvor JSON podataka sa podkomponentom koja ih prima i po potrebi obrađuje, pa ih tako obrađene vraća nazad. Format koda je:

<Ime koje smo odredili u polaznoj komponenti koja šalje podatke naziv polja u podkomponenti ={this.props.imena polja čiji se podatak uzima u obradu}

Naziv JSON faila ili u našem slučaju konstante koji se upotrebljava u podkomponenti da bi se obradili podaci iz ={naziv izvornog JSON faila ili konstante iz koje se čitaju potrebni podaci}/>
Što u našem slučaju izgleda ovako
Deo koda App.js

```
        <Proba name={this.props.name} user={user}/>
```

Znači da smo JSX tagu odredili vrednost za name da je jednaka vrednosti props polja name iz izvora podataka user .
Što se vidi u kodu podkomponente

```
            <p style={{color:"white" }}>
               <br/>- Citanje samo podataka korisnickog imena-
               {this.props.user.name}
            </p>
```

Gde je kodom {this.props.user.name}

Slika iznad koju je dao predhodni kod. U prvom fail u imamo konstantu koja sadrzi podatake koje upotrebljavamo za prikazivanje.

```
const user={
        name: “Mika”,
        age:    “25”,
        hobbi : [“sport”,”Posao” ];
};
```

Ono sto povezuje ove podatke sa prosledjenim podacima sa drugog faila je ovaj red

```
<Proba user={user}/>
```

Drugi fail je podfail ili child a ovaj fail roditelj ili Parent. U tom drugom failu prakticno odlučujemo kako će se podaci i u ovom slučaju konstante prvog pojavljivati u prikazani u glavnom prozoru aplikacije.

Oni se pozivaju na ovaj nacin kao posebni delovi komponente

{this.props.user.name} odnosno u ovom slučaju za prikaz godina {this.props.user.age}

A za slučaj da ih ima vise onda se upotrebljava ovakav kod:

```
<ul>
{this.props.user.hobbi.map((hobbi,i) => <li key={i}>{hobbi}</li>)}
</ul>
```

Ovo znaci da se poziva ova osobina—this.props koju sadrzi u ovom slucaju konstata const user u kojoj se poziva njeno polje name. ili bilo koje drugo. Postavljanje ovih podataka je moguce svuda gde vam je to potrebno u komponenti koju izradjujete, prakticno u ovom child failu.
U drugom slucaju kada se ucitava niz podataka upotrebljava se metod map koji se prakticno pristupa celom nizu podataka. Niz podataka se postavlja u srednjim zagradama, kao sto je napisano.

```
<ul>
{this.props.user.hobbi.map((hobbi,i) => <li key={i}>{hobbi}</li>)}
</ul>
```

Napomena: u ovom slučju smo pristupili podacima pod niza hobby, pozivanjem user koji u sebi ima pod niz podataka hobby. Kod ovog slučaja izgleda ovako

niz podataka user{ name,age, array hobbi{ delovi pod niza hibbi}}

Ovakve situacije su čete u pisanju koda, pa ako se ima i još neki pod niz podataka po dubini kod za pristup njima možete uporebom ovog možete i sami napisati bez ikavih problema jer je ovo filozofija kako to radi.
Ono sto je bitno u ovom slucaju je to da se svi ovi podaci moraju postaviti u odgovarajuće html tagove inače react prijavljuje greske. Verovatno je to tako njegov projektant predvideo pravopisom. Praktično pozivaju se osobine konstante user gde se moze reći mapira ceo niz- Array, a povratni podaci se vraćaju u obliku pod elemeneta –taga

<li key={i}>{hobbi}.

Sam react ima i pravilo da se uvek mora vratiti neki kljuc koji je povezan sa podacima, ako ga nema javlja gresku o nepostojanju jedinstvenog kljuca. Ovom kodu se može promeniti izgled css klasom :

```
.App-podaci{
  background-color: #233;
  width: 300px;
  margin: 20px;
  padding: 10px;
  color: red;
}
```

Prikaz podataka iz spoljnjeg JSON faila

Da bi smo mogli da prikazujemo podatke koji su u json failu u nekoj komponenti react aplikacije, pored izrađenog json faila, potrebno je i instalisati dva spolja paketa 'isomorphic-fetch' i 'es6-promise'. JSON fail ima ulogu baze podataka i njega u slučaju razvoja treba postaviti u public folder aplikacije. Razlog tome je taj što je taj folder kada se pokrene aplikacija u razvojnom načinu rada, ako je aplikacija izradjena u automackom režimu nalazi se na http://localhost:3000.

```
import React from 'react';
import 'isomorphic-fetch';
import 'es6-promise';

export default class Person extends React.Component {

   constructor(props) {
      super(props);
      this.state = {
         korisnik: [],
       };
   }

   componentDidMount(){

      fetch('http://localhost:3000/people.json',{
         method:'get',
         headers:{
            "Content-Type":' application/json',
             'Accept':' application/json'
            }
      })
         .then(response => response.json())
         .then(json => {
            console.log(json);
            this.setState({
               korisnik: json.korisnik
            })
```

```
        }).catch(error => this.setState({ error}));
    }

    render() {
       return(
          <div>
             <h2> Procitani podaci iz post.json faila </h2>
             <ul>
             {
                this.state.korisnik.map(kor=>{
                   return(
                      <div  key={kor.id} >
                         <li>ID Korisnika {kor.id} </li>
                         <li>IME Korisnika {kor.name} </li>
                         <li>EMAIL Korisnika {kor.email} </li>
                         <li>NASLOV Korisnika {kor.title} </li>
                         <li>TEKST Poruke {kor.body} </li>
                      </div>
                      )
                })
             }
             </ul>
          </div>
       );
    }
}

/* people.json */
{
  "korisnik": [
    {
      "id": 0,
      "name": "Milka",
      "email": "nean@gmail.com",
      "title": "Naslov treci ",
      "body": "neki tekst treci"
    },
    {
      "id": 1,
      "name": "Negovan",
      "email": "negan@gmail.com",
      "title": "Naslov treci ",
      "body": "neki tekst treci"
    },
```

```
    {
      "id": 2,
      "name": "Nega",
      "email": "negan@gmail.com",
      "title": "Naslov treci ",
      "body": "neki tekst treci"
    },
    {
      "id": 3,
      "name": "Ne",
      "email": "negan@gmail.com",
      "title": "Naslov treci ",
      "body": "neki tekst treci"
    }
  ]
}
```

Refs atribut u react aplikacijama

Ovaj atribut se se prilikom izrade komponenti react aplikacije može se upotrebiti na više mesta.

Refs i DOM

U tipičnom prenosu podataka React jedini način na koji roditeljske komponente komuniciraju s decom. Da biste promenili osobine za dete, ponovo ćete ga prikazati novim props. Međutim, postoji nekoliko slučajeva u kojima morate imperativno modifikovati osobine deteta van tipičnog toka podataka.
Za slučaj da podkomponentu je potrebno promeniti to može biti na primer react komponenta ili bi to mogao biti DOM element. Za oba slučaja, React pruža mogućnosti za rešavanje problema.

Kada koristiti Refs

Postoji nekoliko dobrih primera za preporuke:
1. Upravljanje fokusom, odabir teksta ili reprodukcija medija.
2. Pokretanje animacija.
3. Integracija s DOM bibliotekom treće strane.

Izbegavajte koristiti refs za bilo šta, što se može učiniti deklarativno. Na primer, umesto izlaganja metoda `open ()` i `close ()` na `Dialog` komponentu, proći ga 'isOpen'. Upotreba refsa može biti za "stvaranje stvari" u vašoj aplikaciji. Ako je to slučaj, odvojite trenutak i mislite kritički više o tome gde bi state vrednost trebala biti u vlasništvu u hijerarhiji komponente. Često postaje jasno da je pravo mesto za "posedovanje" tog stanja na višem nivou u hijerarhiji.

Dodavanje Ref u DOM element

React podržava poseban atribut koji možete pridružiti bilo kojoj komponenti. Atribut `ref` uzima funkciju povratnog poziva, a povratni poziv će se izvršiti odmah nakon što je komponenta montirana ili demontirana. Kada se atribut `ref` koristi na HTML elementu, povratni poziv` ref` prima element DOM kao njegov argument. Na primer, ovaj kod koristi povratni poziv "ref" kako bi sačuvao referencu na DOM čvor:

```
class CustomTextInput proširuje React.Component {
  konstruktor (props) {
    super (props);
    this.focusTextInput = this.focusTextInput.bind (this);
  }

  focusTextInput () {

    this.textInput.focus ();
  }

  render () {
    // Koristite povratni poziv `ref` da biste sačuvali referencu na unos teksta DOM
    // element u polju instance (na primer, this.textInput).
    povratak (
      <Div>
        <input
          type = "text"
          ref = {(ulaz) => {this.textInput = ulaz; }} />
        <input
          type = "gumb"
          value = "Fokusiranje unosa teksta"
          onClick = {this.focusTextInput}
        />
      </ Div>
    );
  }
}
`` `
```

React će nazvati povratni poziv "ref" s elementom DOM kada se komponenta montira i nazvati ga s "null" kada ga ukloni. `callbacks` ref` se pozivaju pre `hooks lifecycle` componentDidMount` ili `componentDidUpdate`.Upotreba povratnog poziva "ref" samo za postavljanje entiteta na klasi zajednički je obrazac za pristup DOM elementima. Poželjan način je postavljanje entiteta u povratni poziv `ref` kao u gore navedenom primeru. Postoji čak i kraći način da ga napišete:

`ref = {input => this.textInput = input}`.

Dodavanje referentne oznake na komponentu klase

Kada se atribut `ref` koristi na prilagođenu komponentu koja se deklariše kao klasa, povratni poziv" ref "dobiva montiranu instancu komponente kao argument. Na primer, ako bismo hteli zamotati gore navedeni "CustomTextInput" kako bismo simulirali da se klikne odmah nakon montaže:

Napomena: reč zamotati odnosi se na to da se ono što je zamotano postavi da bude child <Custom> ono što je zamotano</Custom>

```
klasa AutoFocusTextInput proširuje React.Component {
  componentDidMount () {
    this.textInput.focusTextInput ();
  }

  render () {
    povratak (
      <CustomTextInput
        ref = {(ulaz) => {this.textInput = ulaz; }} />
    );
  }
}
```

Imajte na umu da to funkcioniše samo ako je "CustomTextInput" naglašen kao klasa:

```
class CustomTextInput proširuje React.Component {
  // ...
}
```

Refs i funkcionalne komponente

Ne možete koristiti atribut `ref` na funkcionalnim komponentama jer nemaju slučajeve:

```
funkcija MyFunctionalComponent () {
  povratak <input />;
}

klasa Roditelj proširuje React.Component {
  render () {
    // ovo neće * raditi!
    povratak (
      <MyFunctionalComponent
```

```
      ref = {(ulaz) => {this.textInput = ulaz; }} />
    );
  }
}
```

Trebali biste pretvoriti komponentu u klase ako vam je potreban ref, kao i kada su vam potrebne metode ili stanje u životnom ciklusu. Međutim, možete upotrebljavati atribut "ref" unutar funkcionalne komponente sve dok se odnosi na DOM element ili klasičnu komponentu:

```
funkcija CustomTextInput (props) {

  // textInput mora biti ovde prijavljen tako da se povratni povratni poziv može odnositi na njega
  neka textInput = null;

  funkcija handleClick () {
    textInput.focus ();
  }

  povratak (
    <Div>
      <input
        type = "text"
        ref = {(ulaz) => {textInput = ulaz; }} />
      <input
        type = "button"
        value = "Fokusiranje unosa teksta"
        onClick = {handleClick}
      />
    </ Div>
  );
}
```

Izlaganje DOM Refs komponenti

U retkim slučajevima možda biste hteli imati pristup DOM čvoru djeteta iz roditeljske komponente. To se obično ne preporučuje jer razbija komponentnu enkapsulaciju, ali može povremeno biti korisno za pokretanje fokusa ili merenje veličine ili položaja DOM čvora za dijete. Dok možete dodati referentnu komponentu za dete, ovo nije idealno rešenje jer biste dobili samo komponentnu instancu umesto DOM čvora. Ovo radi i za klase i za funkcionalne komponente.

```
function CustomTextInput(props) {
  return (
```

```
    <div>
      <input ref={props.inputRef} />
    </div>
  );
}

class Parent extends React.Component {
  render() {
    return (
      <CustomTextInput
        inputRef={el => this.inputElement = el}
      />
    );
  }
}
```

U gornjem primeru, 'Roditelj' prosleđuje povratni poziv kao `inputRef` prop na` CustomTextInput` i `CustomTextInput` prenosi istu funkciju kao poseban atribut` ref` na `<input>`. Kao rezultat toga, `this.inputElement` u" Roditelj "će biti postavljen na DOM čvor koji odgovara elementu` <input> `u 'CustomTextInputu'. Imajte na umu da naziv `inputRef` props u gore navedenom primeru nema posebnog značenja, jer je to regularna komponenta prop. Međutim, upotreba atributa `ref` na samom` <inputu> ' važna je, jer govori da React će poslati referencu na svoj DOM čvor. To funkcioniše i ako je "CustomTextInput" funkcionalna komponenta. Za razliku od posebnog atributa `ref` koji se može biti naveden samo za DOM elemente i za komponente klase, nema ograničenja za redovite komponentne props poput` inputRef`.
Još jedna prednost ovog obrasca je ta da deluje dublje u nekoliko komponenti. Na primer, zamislite da roditelj nije potreban za taj DOM čvor, ali komponenta koja je prikazala roditelj (nazovimo ga "babe i dede") trebala mu je pristupiti. Onda bismo dopustili da se "baba i deda" navede "inputRef" prop za "roditelja" i neka roditelj "prosledi" na "CustomTextInput":

```
function CustomTextInput(props) {
  return (
    <div>
      <input ref={props.inputRef} />
    </div>
  );
}

function Parent(props) {
  return (
    <div>
```

```
    My input: <CustomTextInput inputRef={props.inputRef} />
   </div>
  );
}

class Grandparent extends React.Component {
  render() {
   return (
     <Parent
       inputRef={el => this.inputElement = el}
     />
   );
  }
}
```

Evo, povratni poziv prvo navodi "baba i baba". Prebacuje se na "Roditelj" kao redovni podroditelj pod nazivom "inputRef", a "Roditelj" prenosi i na "CustomTextInput" kao prednost. Konačno, "CustomTextInput" čita "inputRef" propoveda i pridaje propuštenu funkciju kao atribut `ref` na` <input> '. Kao rezultat toga, `this.inputElement` u" baba " bi će postavljen na DOM čvor koji odgovara elementu` <input> `u 'CustomTextInputu'.
Sve što se uzima u obzir, savetujem da ne izlažete DOM čvorove kad god je to moguće, ali to može biti koristan izlaz za beg. Imajte na umu da ovaj pristup zahteva dodavanje nekog koda u komponentu za dete. Ako nemate apsolutno nikakvu kontrolu nad implementacijom dečijih komponenti, vaša poslednja mogućnost je upotreba `findDOMNode ()`.

```
import ReactDOM,{findDOMNode } from 'react-dom';
```

Uslovi
Ako je povratni poziv "ref" definisan kao inline funkcija, biti će pozvan dvaput tokom ažuriranja, prvo s nullom, a zatim ponovno s elementom DOM. To je zato što je nova instanca funkcije izrađena sa svakim ponovnim prikazom, pa React treba uklonit stari ref i postaviti novi. To možete izbeći definisanjem povratnog poziva "ref" kao obvezatnom metodom.

Brisanje inputa upotrebom ref

Upotrebom refrenci na određene unosne elemente možemo im obrisati sadržaj. U ovom primeru je i dodati deo iz react-dom obaveznog paketa u react aplikacijama njen deo findDOMNode, koji prema refrenci pronalazi u ovom slučaju input u formi i omogućava da se izvrši brisanje njegovog sadržaja.
Kada komponenta prikaže null ili false, findDOMNod vraća null. U slučaju da komponenta prikaže niz, tada findDOMNod vraća tekstualni DOM čvor koji sadrži tu vrednost. Ono što je još bitno za ovaj deo react-dom paketa je da on radi samo sa

montiranim (mount) komponentama i nemože biti primenjen u funkcionalnim komponentama. Možda će te pronaći u bibilotekama koda za react kod koji traži njegovo prisustvo te prilikom toga možete primeniti delove ovog koda koji se odnosi na te potrebe kako bi taj kod radio.

fail App.js

```
import React from 'react';
import ReactDOM from 'react-dom';

class App extends React.Component {

   constructor(props) {
      super(props);

      this.state = {
         data: ''
   };

      this.updateState = this.updateState.bind(this);
      this.clearInput = this.clearInput.bind(this);
   };
   updateState(e) {
      this.setState({data: e.target.value});
   }
   clearInput() {
      this.setState({data: ''});
      ReactDOM.findDOMNode(this.refs.myInput).focus();
   }
   render() {
      return (
         <div>
            <input value = {this.state.data} onChange = {this.updateState}
                 ref = "myInput"/>
      <button onClick = {this.clearInput}>CLEAR</button>
      <h4>{this.state.data}</h4>
   </div>
   );
   }
}

export default App;
```

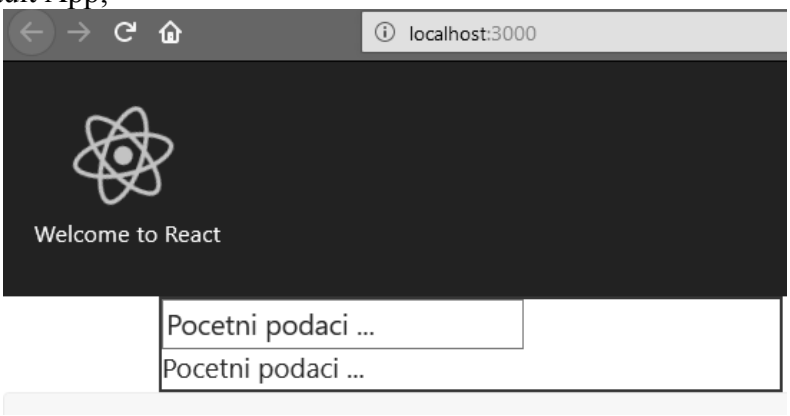
Kompleksni Primer

U ovom primeru biće prikazano kako se upotrebljava child komponenta. onChange metod će postaviti novo stanje koje će proći kroz input child komponente,što će biti prikazano na ekranu. Kada god vam je potrebno da ažurirate vrednosti stanja iz podkomponente, potrebno vam je da prenesete to funkcijom koja će to stanje promeniti (updateState) kao props (updateStateProp).

App.jsx

```
import React from 'react';

class App extends React.Component {
   constructor(props) {
      super(props);
      this.state = {
         data: 'Pocetni podaci ...'
      };
      this.updateState = this.updateState.bind(this);
   };

   updateState(e) {
      this.setState({data: e.target.value});
   }

   render() {
      return (
         <div style={{marginLeft: "100px", border:"2px solid red",
                        width: "400px"}}>
            <Content myDataProp = {this.state.data}
                  updateStateProp = {this.updateState}/>
         </div>
      );
   }
}

class Content extends React.Component {
   render() {
      return (
         <div>
            <input type = "text" value = {this.props.myDataProp}
                  onChange = {this.props.updateStateProp} />
            <h3>{this.props.myDataProp}</h3>
         </div>
      );
```

```
    }
}

export default App;
```



React-Ruter

Ovaj dodatak u react aplikaciji omogućava u najčešćem slučaju rad menija, ali i drugih delova aplikacije koje zahtevaju takve radnje koje se nazivaju rutiniranje, a kojim se omogućava kretanje kroz aplikaciju.
Da bi mogli da upotrebimo ove mogućnosti u aplikaciji potrebno je instalisti ovaj dodatak, i kasnije napraviti više failova koje ćemo da njime pozivamo. Za te namene upotrebićemo za sada automatizovan način izrade react aplikacija kao sto je u predhodnim delovima knjige i opisano i tome ćemo u terminalu dodati ovaj kod

```
npm install react-router-dom –save
```

React ruter je moglo bi se reći nešto kao HOC, jer radi približno isto, tačnije između njegovih tagova <Router> se postavljaju komponente i sve drugo našta se on primenjuje i ubačeni kao objekti sa svojim props.

Napomena: <Router > </Router> između ovih tagova moguće je postaviti samo jednu child Route, te se za upotrebu više pitanja ili Route sa dodatim child komponentama mora pribeći upotrebi div taga i dela react-router-doma Switch.

Ali, malo bih da se zadržimo na tri dela ovog dodatnog paketa: match, location i history.

Match

To je objekt, koji sadrži informacije o tome kako <Router path> pronalazi određen URL. U svom sastavu ima: params, isExact, path i url.

Params

Pramas je objekt koji sadrži ključeve i njihove vrednosti (key/value), URL koji odgovara dinamičkim segmentima staze.

isExact

Sadrži bulean (boolean) vrednosti, istina (true) je ako je URL pronađen

Path

Path je tipa string i to obrazac (patern) puta koji se koristi za podudaranje. Korisnje ga je upotrebiti za izradu ugnježdenih putanja

URL

Je tipa string i namenjen za potvrđivanje podudarnog dela URL-a. Korisno za izgradnju ugniježđenih veza (linkova).

Location

Lokacija je objekt koji prestavlja gde je trenutno postavljena aplikacija (na kojoj je veb stranici), ali i ukazuje gde možemo da se krećemo po aplikaciji. Praktično ovaj objekt se nikada ne menja, te se upotrebljava za utvrđivanje stanja navigacije, pri čemu je koristan za prihvatanje podataka(fetchign data) ili onog što dolazi od strane DOM-a. možete očitati njegove vrednosti ako u kod gde ga primenjujete unesete

console.log(location)

Neke od mogućnosti za primenu:

history.location, history.push ili history.replace.

```
componentDidUpdate(prevProps, prevState, snapshot){
    if (prevProps.location.pathname !== this.props.location,pathname){
      // neke radnje, na primer fetch API i slicno
    }
}
```

ili kod konstante:

```
const locationN ={
       pathname: '/korisnik/Korisnikxx',
       state:{ fromKorisniklista: true}l
    }
```

Koju možemo da primenimo

<Redirect to={locationN} />
<Link to={locationN} />
history.push(locationN)

Postavljajući, izuzev poslednje, ih između <Route > i <Switch > komponenti.

History

Objekt koji omogućava upravljanje istorijom u vašem pretraživaču unutar vaših pregleda ili komponenti pružajući vam time dosta pogodnosti.
Napomena: u višim varijantama Reacta potrebno ga je instalisati kao dodatni paket sa npm i - -s

Length

Je brojna vrednost koja vam pokazuje koliko imate, na primer posećenih veb stranica

Action

Je string i odnosi se na trenutnu akciju (PUSH, REPLACE ili POP)

Location

Je objekt i odnosi se na trenutnu lokaciju

Push

Je funkcija koja postavlja novi unos u istoriju pregleda (History Stack) .
push(path,[state])

Replace

Slična predhodnoj samo što ona zamenje trenutni unos u istoriji
replace(path,[state])

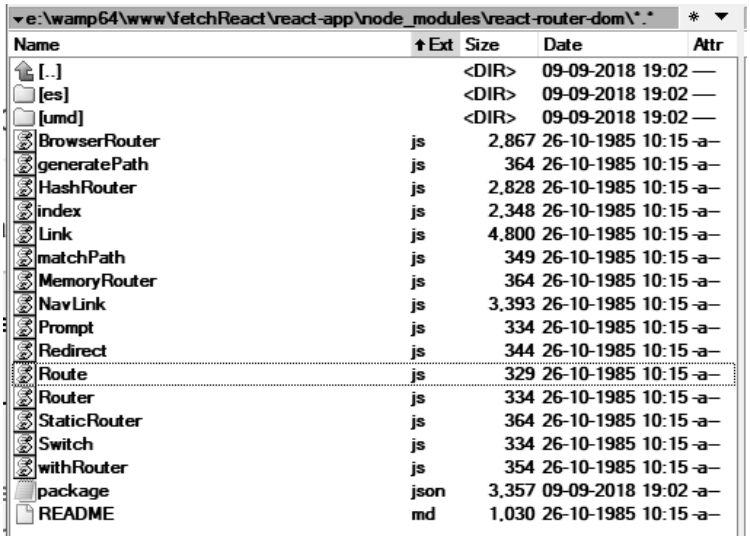
Go(n), goBack(), goForward()

su funkcije za pomeranje poentera po istoriji pregleda (History Stack), u prvom slučaju se pomeranje vrši za vrednost unetog broja, dok preostale dve su namenjene za kretanje u minusu/plus za jedan korak

Block

Funkcija koja upotrebljava Prompt, namenjena blokiranju nepoželjenih pregleda(veb sajtova)

Ako pogledamo u sadržaj instalisanog paketa u folderu node_modules videćemo ovo:



Slika nam prikazuje sve ono što ima u ovom dodatnom paketu react aplikacije, a što možemo da pozivamo pijedinačno ili sve odjednom u jednom dahu, a da to posle u aplikaciji samo nagasimo da će biti upotrebljeno.
Primer koda dole prikazje način upotrebe vrednosti stanja koje se mogu upotrebiti u delovima za rutiniranje između komponenti react aplikacije.

```
import React from "react";
import { Route, NavLink, BrowserRouter, Redirect} from "react-router-dom";

const User = ({ match }) => {
   return(<div>
         <h1>Korisnik:::{match.params.id}</h1>
         <p>{match.params.id}</p>
      </div>
   )
};
const Laki = ({ match }) => {
   return(<div className="btn-light">
         <h1>Korisnik:::{match.params.id}</h1>
         <p>{match.params.id}</p>

      </div>
   )
};

class Users extends React.Component {
   constructor(){
      super();
      this.state={
         pera:'laza',
```

```
            mika: "Milisav"
        }
    }
    render() {
        // const {url } = this.props.match;
        // let pera = localStorage.getItem('pera');
        let {pera, mika} = this.state;
        return (
            <BrowserRouter>
            <div className="container">
                <h1>Users</h1>
                <Route path="/users/:id" component={User} />
                <Route path={`/users/:${mika}`} component={Laki} />
                <strong>select a user</strong><br/>
                        <NavLink className="btn btn-dark btn-lg"
                          to={`/users/${pera}`}>{pera}
                        </NavLink>
                        <NavLink className="btn btn-dark btn-lg"
                          to={`/Laki/${mika}`}>{mika}
                        </NavLink>
                        <NavLink className="btn btn-dark btn-lg" to="/users/2">
                          User 8
                        </NavLink>
                        <NavLink className="btn btn-dark btn-lg" to="/users/3">
                         User 3
                        </NavLink>
            </div>
            </BrowserRouter>
        );
    }
}

export default Users;
```

U klasi Users postavljene su vrednosti za stanja dve promenjive, koje da bi se upotrebile pristupljeno je destruktivnom metodom da se one proglase za standardnim pojedinačnim vrednostima stanja, kako bi se mogle posebno potom upotrebljavati gde vam je to postrebno u komponenti,

```
        let {pera, mika} = this.state;
```

a kasnije su dodate u deo komponete NavLink ,react-router-dom-a dodale

```
<NavLink className="btn btn-dark btn-lg"
to={`/Laki/${mika}`}>{mika}</NavLink>
```

Ali se mogu dodati i na ovaj način :

```
<NavLink className="btn btn-dark btn-lg"
to={`/users/${this.state.pera}`}>{pera}</NavLink>
```

Na ovaj način moguće ih je dinamički menjati dodelom nove vrednosti stanja uporebom

```
this.setState({ vrednost : this.state.novavrednost})
```

Šta i kako upotrebiti

Iz svih dodataka se mogu upotrebljavati java skript failovi kao dodatni za naše potrebe. U ovom slučaju svi sem JSON i md, kao i failovi iz ova dva direktorijuma. Njihovo pozivanje i postavljanje u budućoj aplikaciji može se vršiti kao što je pomenuto pojedinačno upotrebom destruktivne metode u delu za imoport koda komponente

```
import { BrowserRouter as Router, Switch, Route, Link, NavLink } from 'react-
router-dom';
```

Čime pozivamo pojedinačne delove paketa u komponentama gde nam je to potrebno, ali se mogu pozivanja obaviti i na ovaj način:

```
import 'react-router-dom';
```

S tome da kasnije upotrebimo kao kod upotrebe delova paketa ReactDOM

```
import ReactDOM from 'react-dom';

ReactDOM.render(<Router history={browserHistory}>,
document.getElementById('react-dom');
```

Za kretanje po vebsajtu možemo upotrebiti i Redirect ili pomeranjem po istoriji pristupa, kao što se vidi u donjem kodu

```
onSubmit = () => {
   //this.props.history.push('/contact')
   return <Redirect to="/contact" />;
};
```

```
<NavLink className="btn-danger btn-lg" to="/contact" onClick={LoggOut}
>Kontakt</NavLink>
```

Da bi ovo pozivanje odredjene lokacije radilo potrebno je da u nekoj komponenti postoji kod

```
<BrowserRouter>
   <div>
      <header style={stil}>
        <NavLink className={!this.state.prijavaL?'btn-danger btn-lg':''} to="/login"
              onClick={this.Prijava} >{!this.state.prijavaL?'PRIJAVA':''}
        </NavLink>
      </header>
      <Switch>
         <Route path="/home" component={Home} />
         <Route exact={true} path="/login" component={LogIn} />
         <Route exact={true} path="/users" component={Users} />
         <Route path="/contact " component={Kontakt} />
      </Switch>
   </div>
</BrowserRouter>
```

Dok gornji kod za pozivanje može biti bilo gde u aplikaciji, samo što je u toj komponenti gde se on postavlja porebno imati importovane delove react-pouter-dom paketa.

Napomena: Najbolje je i malo više pročitati na internetu o tim mogućnostima, to jest kada koju upotrebiti i u kom paketu. Razlog za opomenu je taj što programeri u cilju olakšanja rada prave novije verzije. Pa je moj savet upotrebiti kod kao u knjizi, jer svaki kod koji sam napisao proveren je u praksi da li radi. Deo koda koji je vezan za istoriju u ruteru poželjno je instalisati sa već pomenutog sajta za preuzimanje react dodatnih paketa.

Izrada potrebnih failova

Da bi ste videli kako ovaj deo reacta radi potrebno je napraviti fail koji bi bio praktično meni ili izbornik za kretanje između komponenti, dodati kod u App.js i index.js u već urađenoj aplikaciji, kao i nekoliko falova za prikaz kada se klikne na deo menija koji ih poziva. Kod failova koji se pozivaju može biti i ovakav za sve, samo im možete promeniti tekst između taga H2

```
import React from 'react';

class Fail1ZaPozivanje extends React.Component {
```

```
   render() {
      return (
        <div>
          <h2>Prvi fali za pozivanje</h2>
        </div>
      );
   }
}

export default Fail1ZaPozivanje;
```

Fail Meni.js

```
import React from 'react';
import {Link} from 'react-router-dom';

const Meni = () => (
   <div>
      <nav className="navbar navbar-inverse" style={{zIndex:500}}>
         <div className="container-fluid">
            <div className="navbar-header">
               <button type="button" className="navbar-toggle collapsed"
                     data-toggle="collapse" data-target="#navbar"
                     aria-expanded="false" aria-controls="navbar">
                  <span className="sr-only">Toggle navigation</span>
                  <span className="icon-bar"></span>
                  <span className="icon-bar"></span>
                  <span className="icon-bar"></span>
               </button>
               <div className="navbar-brand">
                  <Link to="/">
                    Sajt u React-u
                   </Link>
                </div>
            </div>
            <div id="navbar" className="navbar-collapse collapse">
               <ul className="nav nav-pills">
                  <li> <Link to="/"> Fail1ZaPozivanje </Link> </li>
                  <li> <Link to="/ Fail1ZaPozivanje ">
                    Fail1ZaPozivanje </Link> </li>
                  <li> <Link to="/ Fail2ZaPozivanje ">
                     Fail2ZaPozivanje </Link> </li>
                  <li> <Link to="/ Fail3ZaPozivanje ">
                     Fail3ZaPozivanje </Link> </li>
```

```
                </ul>
              </div>
            </div>
          </nav>
      </div>
);

export default Meni;
```

App.js fail

```
import React, { Component } from 'react';

import Fail1ZaPozivanje from './ Fail1ZaPozivanje ';
import Fail2ZaPozivanje from './ Fail2ZaPozivanje ';
import Fail3ZaPozivanje from'./ Fail3ZaPozivanje ';

import './App.css';
import {  Route,  Switch,  Redirect} from 'react-router-dom';

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <Meni/>
        </header>
        <Switch>
          <Route path="/ Fail1ZaPozivanje " component={ Fail1ZaPozivanje } />
          <Route path="/ Fail2ZaPozivanje " component={ Fail2ZaPozivanje }/>
          <Route exact path="/"  component={ Fail1ZaPozivanje } />
          <Route path="/ Fail3ZaPozivanje " component={ Fail3ZaPozivanje }/>
          <Redirect to="/" />
        </Switch>
        <Futer/>
      </div>
    );
  }
}

export default App;
```

I na kraju to ćemo povezati sa realnim svetom ili ti stvarnim DOM elementom putem koda u failu index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import { BrowserRouter } from 'react-router-dom';

ReactDOM.render(
   <BrowserRouter>
      <App />
   </BrowserRouter>
   , document.getElementById('root')
);
```

U ovom kod u je upotrebljeno ponešto iz Bootstrapa, a uz još malo dodatog css koda izgleda ovako u orginalu na mom veb prostoru.

Drugi primer React rutera

Sasvim je svejedno upotrebili NavLink ili Link u aplikaciji, ali ako na više mesta upotrebljavate react router onda je moguće da se pojave problemi ako se na više mesta u aplikaciji koristi Link, te je u tom slučaju bolje upotrebiti NavLink. Ovo zato što mi se dogodilo navedeno tokom testiranja i učenja.

Fail App.js

```
import React, { Component } from 'react';
import { BrowserRouter as Router, Switch, Route, Link } from 'react-router-dom';
import Home from './Home';
import Login from './Login';

class App extends Component {

   render() {
      return (
         <Router>
            <div style={{ marginLeft:"100px"}}>
               <h2>Jedan od nacina upotrebe React Routera </h2>
               <ul className="navbar navbar-nav">
                  <li style={{color:" red", border:" 2px solid yellow" }}>
```

```
                    <Link to={'/'}>Home</Link></li>
                    <li style={{color:" red", border:" 2px solid green" }}>
                    <Link to={'/Login'}>Login</Link></li>
                </ul>
                <hr />
                < br />
                <Switch>
                    <Route exact path='/' component={Home} />
                    <Route exact path='/Login' component={Login} />
                </Switch>
            </div>
        </Router>
    );
  }
}
export default App;
```

Fail Home.js

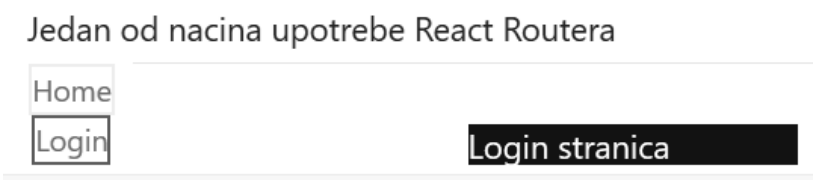
```
import React, { Component } from 'react';

class Home extends Component {
  render() {
    return (
      <div className="container">
        <h2 style={{ width:" 200px", background:"orange "}}>
          Home stranica
        </h2>
      </div>
    );
  }
}
export default Home;
```

Fail Login.js

```
import React, { Component } from 'react';

class Login extends Component {
  render() {
    return (
      <div className="container">
        <h2 style={{ width:" 200px", background:"blue",
                     color:" white" }}>
```

```
                Login stranica
            </h2>
          </div>
        );
    }
}
export default Login;
```

Daju sliku dole



Prolazak kroz više nivoa

Kao što imamo menije u više nivoa, tako i ruter ima iste mogucnosti. Ovaj primer omogućava pristup podacima putem urla podacima u kosntanti koja ima više nivoa podataka. Dole je kompletan kod komponente.

Fail ruterSaViseNivoa.js

```
import React, { Component } from 'react'
import { BrowserRouter as Router, Route, Link,NavLink} from 'react-router-dom'

const baza = [
    {
        name: 'O React Routeru',
        id: 'react-router',
        objasnjenje: 'Osnove react rutera kroz objasnjenja',
        resources: [
            {
                name: 'URL Parameteri',
                id: 'url-parameters',
                objasnjenje: "URL parameterima se omogućava prelazak sa
                 stranice na stranicu.",
                url: 'http://miodragtrajanovic.com/'
            },
            {
                name: 'Programsko kretanje kroz aplikaciju',
                id: 'Programirana Navigacija',
                objasnjenje: " React ruter omogucava da mozete da se
```

```
            krecete dubinom aplikacije .",
            url: 'http://miodragtrajanovic.com//'
        },
        {
            name: 'Mogucnosti',
            id: 'Uporteba',
            objasnjenje: " React ruter je vrlo jednostavan za upotrebu .",
            url: 'http://miodragtrajanovic.com//'
        }
    ]
},
{
    name: 'React.js',
    id: 'reactjs',
    objasnjenje: 'JS biblioteka koda ',
    resources: [
        {
            name: 'React Dogadjaji',
            id: 'react-lifecycle',
            objasnjenje: "React moze da upravlja razlicitim dogadjajima",
            url: 'http://miodragtrajanovic.com/'
        },
        {
            name: 'React na trenutak',
            id: 'reacta',
            objasnjenje: "Trenuci ucenja React.",
            url: 'http://miodragtrajanovic.com/'
        }
    ]
},
{
    name: 'Functionalno Programmiranje',
    id: 'functionalno-Programmiranje',
    objasnjenje: 'Kao i svuda i ovde mozete načiniti svoja dela',
    resources: [
        {
            name: 'Mozda neki trik  ',
            id: 'deklarativno',
            objasnjenje: 'Nije tesko samo nastavi sa radom',
            url: 'http://miodragtrajanovic.com/'
        },
        {
            name: 'Izrada stranica',
            id: 'Lako',
```

```
            objasnjenje: 'U reactu se radi iz malih delova',
            url: 'http://miodragtrajanovic.com/'
          }
        ]
    }
];

function Treci ({ match }) {
    const podatak = baza.find(({ id }) => id === match.params.pocetniId)
      .resources.find(({ id }) => id === match.params.drugiId);
    return (
      <div>
        <h1>Treci nivo</h1>
         <h2>{podatak.name}</h2>
         <p>{podatak.objasnjenje}</p>
         <hr style={{border:"2px red solid"}}/>
         <a href={podatak.url}
             className="btn-info btn-lg" style={{paddingBottom:"10px" }}>
           Dodatne Infromacije
         </a>
      </div>
    )
}

function Druga ({ match }) {
    const podatak = baza.find(({ id }) => id === match.params.pocetniId);
    return (
      <div>
         <h1> Drugi nivo</h1>
         <h2>{podatak.name}</h2>
         <p>  {podatak.objasnjenje}</p>
         <ul>
            {podatak.resources.map((sub) => (
               <li key={sub.id} style={{paddingBottom:"5px",
                     listStyleType: "none"  }}>
                  <Link className="btn-dark btn-lg"
                  to={`${match.url}/${sub.id}`}>{sub.name}</Link>
               </li>
           ))}
         </ul>
         <hr style={{border:"2px red solid"}}/>
         <Route path={`${match.path}/:drugiId`} component={Treci} />
      </div>
    )
```

```
}

function Prva ({ match }) {
   return (
      <div>
         <h1>Prvi nivo</h1>
         <p> Prvi nivo pristupa react ruterom</p>
         <ul>
            {baza.map(({ name, id }) => (
               <li key={id} style={{paddingBottom:"5px",listStyleType: "none"  }}>
                  <Link className="btn-success btn-lg"
                    to={`${match.url}/${id}`}>{name}</Link>
               </li>
            ))}
         </ul>
         <hr style={{border:"2px red solid"}}/>
         <Route path={`${match.path}/:pocetniId`} component={Druga}/>
      </div>
   )
}

const Pocetna =()=> {
   return (
      <div>
      <h1>Pocetna Stranica </h1>
      <p>
        Prikaz rada dodatnog paketa React rutera u vise nivoa pristupa, podacima
      </p>
      </div>
   )
};

class App extends Component {

   render() {
      return (
         <Router >
            <div style={{backgroundColor:"pink", margin: '0 auto',
                    border:"4px solid grey"}}
                  className="container" >
               <NavLink className="btn-primary btn-lg" to='/'
                    style={{marginRight:"5px"}}>Glavna</NavLink>
               <NavLink className="btn-primary btn-lg" to='/prvi'>
                Prikaz Rutera
```

```
                </NavLink>
                <hr style={{border:"2px red solid"}}/>
                <Route exact path='/' component={Pocetna} />
                <Route path='/prvi' component={Prva } />
            </div>
        </Router>
    )
  }
}

export default App
```

Kao što se vidi u tekstu koda, za ovaj primer je uzeta konstanta koja sadrži potrebne podatke u obliku JSON koda, gde im pristupa upotrebom react rutera, pri čemu se taj pristup vidi u obliku url koda u brovzeru.
Klasa App koja je ujedno i jedina sa eksportom ili mogućnošću povezivanja u aplikaciji dalje je jedan meni koji je zatvoren ruterom

```
<Router>
    Children
</Router>
```

između kojih je postavljeno između div tagova NavLink i Route. Gde klikom na NavLink se poziva Route kojom se vrši pozivanje upotrebljenih komponenti koje su pozvane i bivaju prikazane na ekranu, ali ujedno se i vrši prikaz urla odabrane veb stranice. Kosa crta podeljeno u delu path označava osnovnu polaznu stranicu, dok prvi ispis iza nje se pojavljuje u urlu iza domena iza kose crte koja se pojavljuje odmah posle imena domena, iza koje se ispisuje taj tekst koji smo uneli iza kose crte. Istovremeno dolazi do pozivanja komponente ciji je naziv ispisan iza reči component u velikim zgradama.

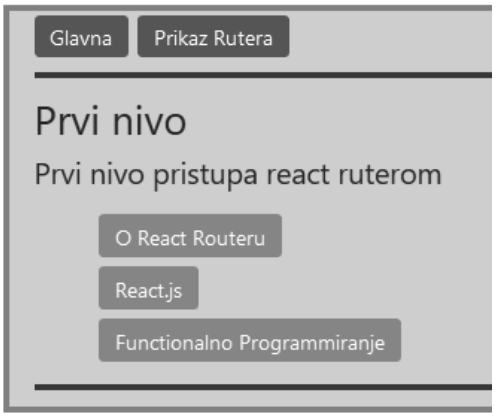
```
<Route path='/prvi' component={Prva} />
```

Slično se ponavlja i za preostala pozivanja u primeru. Obzirom da je ponekada potrebno upotrebiti podatke iz baze da se isti prikažu potrebno je to i povezati sa komponentom, što je i urađeno u komponenti

```
function Prva ({ match }) {
  return (
    <div>
      <h1>Prvi nivo</h1>
      <p> Prvi nivo pristupa react ruterom</p>
      <ul>
        {baza.map(({ name, id }) => (
```

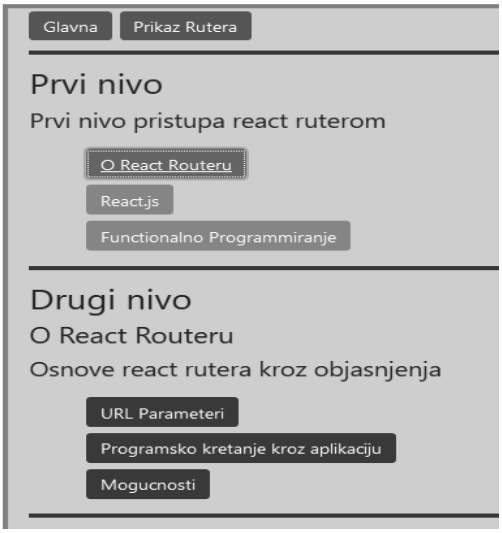
```
                <li key={id} style={{paddingBottom:"5px",listStyleType: "none"  }}>
                   <Link className="btn-success btn-lg"
                 to={`${match.url}/${id}`}>{name}</Link>
                </li>
              ))}
          </ul>
          <hr style={{border:"2px red solid"}}/>
          <Route path={`${match.path}/:pocetniId`} component={Druga}/>
        </div>
    )
}
```

Koja vraća match objekt, a koji ukazuje na url kog sledi dopuna sa Id-eom. U ovom slučaju se map metodom izrađuje novi niz elemenata od preuzetog iz baze podataka ili JSON faila ili konstante kao u ovom slučaju. Što daje prikaz na slici



Dobili smo prikaz dela name iz objekta koji je van objekta resources. Pošto je potrebno da prikažemo još podataka osim podatka koji je vezan za name, izrađujemo funkciju Druga i povezujemo je za ovu upotrebom Route(ruta ili putanja ili put). Na ovaj način se samo vrši prikaz komponente, ali ne i njenih podataka, da bi smo prikazali njen sadržaj potrebno je napraviti jednu konstatnu kojoj bi se pronađeni u bazi Id, a koji je jednak Id-eu ekvivalentnom koji već postoji u Id-eu početniId, kako bi smo mogli da pročitamo i preostale podatke ovog nivoa. Čime smo dobili prikaz vrednosti name i objašnjenje. Obzirom da u ovom slučaju koji može biti sličan i nekim vašim budućim, sadrži više podataka po dubini ponovićemo postupak iz predhodnog dela koji se odnosi na izradu neuređene liste koja pored li taga sadrži i deo react rutera Link. Umesto ovog dela react rutera možete upotrebiti i NavLink, samo mora te ga pozvati u import delu za react-router-dom. Jer rade skoro isti posao. Da bi smo dobili podatke iz podobjekta resoureces potrebno ga je nadovezati na ime konstante, kao njenu vrednost i od toga napraviti novi niz podataka. Ono što je važno i to nemojte zaboraviti na dodelu key vrednosti, jer će vam se javiti upozorenje, te se on uvek dodaje u tag

koji okružuje onaj drugi koji se upotrebljava za ispis preostalih podataka. U našem primeru je to tag li. Vrednost za key obično je id, ali može biti i index u li ono što vam je jedinstveno u svemu.



Kako bi se mogao prikazati sadržaj u ovom delu predhodnoj putanji dodat je deo koji se odnosi na id dobijen upotrebom map metoda sub.id ,

```
<Link className="btn-dark btn-lg"
   to={`${match.url}/${sub.id}`}>{sub.name}
</Link>
```

i ovom prilikom čitamo samo podatak name {sub.name}, mada možete čitati i više podataka ako ih imate u vašim primerima. Ako pogledamo u brovzer videćemo ovu vrednost za url

http://localhost:3000/prvi/reactjs

Što i prikazje slika iznad. Dodavanje mogućnosti za čitanje dalje podataka po dubini na ovaj način povezujemo i treću komponentu upotrebom koda :

```
<Route path={`${match.path}/:drugiId`} component={Treci} />
```

Ovo bi značilo da se predhodnom url-u dodaje sledeći /:drugiId , koji kao i predhodni /:pocetniId su nam potrebni u komponenti Treci da bi smo mogli da pročitamo sledeće podatke koji se trebaju prikazati kada se ona pozove da ih prikaže. Obzirom da smo prikazali dva nivoa pristupa po dubini za ovaj treci nivo je potrebno povezati predhodna dva sa pod objektom resources u vrednost kosnstante gde ćemo da dodatku imena konstante iza tačke dodati imena onih polja koje trebamo da očitamo i prikažemo.

```
const podatak = baza.find(({ id }) => id === match.params.pocetniId)
    .resources.find(({ id }) => id === match.params.drugiId);
```

Znači da ekvivalentom pronađenom id, koji je ekvivalentan početniId nadovezujemo pronađen id iz resources podobjekta za svaki slučaj ako je samo kao je on ekvivalentan id-u parametara vrednosti drugiId i to smeštamo u konstantu podatak. Dok sama funkcija će opet kroz match objekt vratiti prikazane vrednosti koje još sadrži podobjekt resources

```
return (
    <div>
      <h1>Treci nivo</h1>
       <h2>{podatak.name}</h2>
       <p>{podatak.objasnjenje}</p>
       <hr style={{border:"2px red solid"}}/>
       <a href={podatak.url} className="btn-info btn-lg"
          style={{paddingBottom:"10px" }}>
         Dodatne Infromacije
       </a>
    </div>
)
```

Napomena: Prilikom primene ovog primera sa ovim kodom dolazi do povećanja visine prikaza i pomeranja sadržaja kako visina prikaza narasta, pa je potrebno u svakom povratnom delu funkcija to regulisati css-om ili upotrebljavati modale.
Kada su pokrenute sve tri kompnente url je ovaj
http://localhost:3000/prvi/reactjs/reacta
Ovim načinom je pojašnjen način pristupanja kada u jednom nizu podataka imamo još drugih nizova koji se upoterbljavaju kao izvor podataka. Ta slika prikazuje da je na ovaj načim moguće pristupati po dubini podataka nebitno na kojoj su oni dubini u bazi podataka. Znači baza je ime osnovnog niza prvog nivoa, koji u svom satavu ima još jedan niz resources. U računskim radnjama se ovakva pojava naziva na Srbskom matrica, jer ima više nizova u svom sastavu. Koji imaju svoje pod nizove. Znači baza osnovni naziv niza [pod niz resources][pod niz drugi][pod niz treći], ali u praksi možemo da naiđemo i na slučajeve da pod niz ima svoje pod nizove.

Zaključak

Ruter može se upotrebiti bilo gde u aplikaciji bez ikavih problema. Tag <Router> može imati samo jedan element unutar sebe, kao child, u kome se mogu smestiti drugi potrebni delovi.

```
<Router>
        <div>
        ..... preostali potrebni elementi
        </div>
</Router>
```

U suprotnom doćiće do pojave greške. Za više unosa uporebljava se Switch koji zatvara preostale tagove,tj. komponete react-router-som paketa. Tako da je korisno poštovati hijerarhiju elemenata i posle unosa mogućeg potrebnog dodatnog react paketa pronaći o njemu orginalne podatke kako bi se izbelgo gubljenje vremena. Ali, ako upotrebimo kod koji je dole ispod, možemo da još lakše vršimo preusmeravanje na neku stranicu u aplikaciji. Deo paketa react-router-dom <BrowserRouter>, kao <Router> mogu imati samo po jedan element unutar sebe zatvorenim, da bi se moglo unutar njih postaviti više elemenata, potrebno je unutar njih postaviti html tag , koji bi sadržao sve ostale potrebne elemente. <BrowserRouter> ili <Router> su oni koji imaju najviši nivo u hijerarhiji paketa react-router-dom koliko se da zaključiti. Dok sama upotreba putanja po istoriji

bovrsera ili aplikacije je u ovom slučaju zatvorena sa delom paketa react-router-dom <Switch> i sve putanje koje su smeštene između nje mogu se upotrebiti kao istorija dozvoljenih putanja u aplikaciji. U ovom primeru koda ispod možemo da se krećemo na ovaj način koji daje funkcija Appp na četri putanje.

```
export function Appp() {
   return (
      <BrowserRouter>
         <div>
            <header style={stil}>
               <NavLink className="btn-dark btn-lg"
                to="/home">Home</NavLink>
               <NavLink className="btn-dark btn-lg"
                 to="/login">LogIn</NavLink>
            </header>
            <Switch>
               <Route path="/home" component={Home} />
               <Route path="/login" component={LogIn} />
               <Route path="/nekatrecakomponenta" component={
                     nekatrecakomponenta } />
               <Route path="/nekakomponenta" component={ nekakomponenta } />
            </Switch>
         </div>
      </BrowserRouter>
   );
}
```

Konstanta KretanjeButton upotrebom withRouter, koji je isto tako sastavni deo paketa react-router-dom omogućava putem objekta history kretanje po dozovljenim putanjama u aplikaciji kroz upotrebu strelaste funkcije koju sadržava događaj onClick.

```
const KretanjeButton = withRouter(({ history }) => {
   return (
      <div>
         <h1>Klikni na dugme</h1>
         <p>
            <button className="btn-dark btn-lg" onClick={() =>
                  history.push("/nekakomponenta ")}>
               Neka Komponenta
            </button>
            <button className="btn-dark btn-lg"
               onClick={() => history.push("/nekatrecakomponenta ")}>
               Sign Out
```

```
            </button>
          </p>
        </div>
    );
});

class LogIn extends React.Component {
    render() {
        return (
            <div>

            <p>
                 LogIn
               < KretanjeButton />
            </p>
            </div>
        );
    }
}
```

Upotreba css stilova

Css podešavanja osim kao u slučaju upotrebe bootstrapa css opisi mogu da se primene u react aplikacijama u obliku promenjive, kao dole u kodu

```
import React from 'react';

class App extends React.Component {

    render() {

        let myStyle = {
            fontSize: 100,
            color: '#FF0000',
            border: '2px solid blue',
            width: 350
    };

        return (
            <div>
                <h1 style = {myStyle}>Header</h1>
            </div>
        );
    }
```

```
}
export default App;
```

Kod daje ovu sliku. Pošto je u ovom slučaju promenjiva već java skript i u njoj automaki postoje jedan par velikih zagrada u tom slučaju se ne primenjuju dvostruke velike zagrade gde se taj stil poziva.

Ali, se css stilovi mogu primenti direktno u kodu gde je to potrebno (inline)

```
<h1 style = {{color: “red”, border: “2px solid blue” } }>Header</h1>
```

Css opise možete da učitate i iz baze podataka ili JSON posebnog faila. Ili

```
render() {
  return <span className="menu navigation-menu">Menu</span>
}

render() {
  let className = 'menu';
  if (this.props.isActive) {
    className += ' menu-active';
  }
  return <span className={className}>Menu</span>
}
```

Ako, pogledamo inline stile primetno je da se opet time povećava dužina koda i mogućnost pojava grešaka, te je potrebno ga izbegavati. To jest najbolji način za css je poseban fail za svaku komponentu.
Drugi način upotrebe css-a u react aplikacijama prikazuje kod dole ispod. Kod css-a smešten je u obliku konstante, gde imamo praktično dva objekta sa css-om jedan je za heder drugi je za naslov. A samo pozivanje u ovakvim slučajevima je uz upotrebu velikih zagrada i ispisa prvo imena konstante sa stilovima, a posle tacke u nastavku ime objekta ili klase sa css kodom.

```
<header className={styles.header}>
```

i drugi slučaj je ovakav

```
<h1 className={`${styles.title} text-center`}>{title}</h1>
```

gde dodajemo css iz klase stilova postojećoj css klasi, gde je isto sve postavljeno u velike zagrade ali uz upotrebu izvrnutog apostrofa koji zaokružuje ceo izraz s time što ispred css-a iz java scripta se nalazi oznaka dolara, dok je on isto napisan uokviren velikim zagradama koa u gornjem slučaju.

```
export function Header({ title }) {
return (
<row>
<columns className={styles.header}>
<h1 className={`${styles.title} text-center`}>{title}</h1>
</columns>
</row>
);
}

const styles = {
header: css({
background: '#e54226',
background: `linear-gradient(to left, #ffa81f, #e54226)`,
paddingTop: 16
}),
title: css({
color: 'white',
fontWeight: 'bold',
fontSize: 24,
margin: 0,
padding: 0,
textTransform: 'uppercase'
})
};
```

API

Api (aplikaciono programbilno prilagođenje) je praktično jedan folder na serveru gde se nalaze failovi kojima se vrši prenos podataka ka serveru i od servera dalje gde je to potrebno, a predviđeno zahtevima aplikacije koja ga upotrebljava. Čime kod tih failova vrši prilagođavanje između različitih platformi. Primera radi node-react je različit od php-a, tako da kod failova u ovom folderu prilagođava korisničke i serverske zahteve uzmeđu njih. Moglo bi se reći da su ovde smešteni kontroleri. Ako pogledamo tu komunikaciju primetićemo jedan fail koji se uvek cilja kao polazni. Kod tog faila na serveru prema upućenim zahtevima poziva druge failove koji obrađuju te zahteve. Ti failovima koje on poziva nazivaju se u nekim sutiacijama kontroleri prema poslu koji obavljaju. Naravno prema svemoj veličini

aplikacije i oni mogu da za svoj rad pozivaju druge failove na serveru u tom pozadinskom delu aplikacije.
Kod u failu je u zavisnosti od zahteva aplikacije šalje, prima, piše, briše podatke u bazi podataka na serveru. Što daje sliku jednog mosta koji povezuje klijenta i server. Te radnje obavljaju biblioteke dodatnih paketa reacta: AJAX, Axios ili Fetch u najćešćem slučaju.

Pristup konstantama u reactu

Da bi smo mogli da pristupamo delovima failova od kojih gradimo komponente u reactu potrebni je :

1. da u komonentu gde pristupamo da taj deo sadrži u slučaju rada sa serverom ključnu reč export ispred svog imena, u donjem slučaju to je pristup konstanti, mada se može pristupati i funkcijama i dr.
2. potrebna je istovetnost dela koji je zajednički za sve one koje na ovaj način pristupaju deljenju u našem slučaju je email
3. failu ili komponenti kojoj je potreban deo neke za njen rad prijava se vrši destruktivnim načinom prilikom importa, mada se nije obavezno, ali je sigurnije u radu i preporučljivije
4. Primena potrebnog dela jednog faila u fail koji ga poziva najčešće je u delu ispod dela početka klase, deo za konstruktor. Ovaj način je najpreporučljiviji, mada, se može primeniti i na drugim mestima u načinu pisanja java skript koda u react aplikacijama.
5. Ako se ne primenjuju u failu koji poziva deo nekog drugog u ove namene ne postoje komponente životnog ciklusa onda se i failovi koji ih imaju i potrebuju ove koje pozivaju druge potrebno je ih importovati u ove koje pozivaju i omotati ih oko ovih što pozivaju.

```
import TreciFail from './ TreciFail ';

.....
export default TreciFail(SendEmail);
```

Pojedinačni pristup konstantama i funkcijama u react aplikacijama koje su smeštene u nekom failu react aplikacije, kao što vidite u kodu dole vrlo je jednostavan. Znači to što se poziva od mesta iz koga se vrši pozivanjem potrebno je dodati naredbu export i u ovom našem slučaju ono što je istovetno za oba faila,, prestavljeno u ovom slučaju kao objekt email. Na ovom mestu se mogu naći i props i drugo što je nam nepohodno u aplikaciji.

Fail kome-se-pristupa.js

```
// deo faila kome se pristupa iz drugih failova

export const sendRequestToServer = ({ email }) =>
```

```
  sendRequest('/api/v1/public/send-email', {
    body: JSON.stringify({ email }),
  });
```

U ovom drugom failu koji poziva u ovom slučaju konstantu iz prvog u svom radu potrebno je importovati ceo predhodni fail iz koga se traži jedan ili više njegovih delova se upotrebili, potrebno je pristupiti destruktorom, čime se izdvaja samo ono što je nama potrebno. U ovom slučaju to je

```
import { sendRequestToServer } from './ kome-se-pristupa ';
```

Čime smo praktično odredili šta nam je potrebno iz nekog faila u aplikaciji u nekom drugom, u ovom slučaju u failu slanje-emaila.js.

fail slanje-emaila.js

```
import React from 'react';
// import TreciFail from './ TreciFail ';

// destruktivni pristup impoertovanom failu
import { sendRequestToServer } from './ kome-se-pristupa ';

class SendEmail extends React.Component {

  onSubmit = async (e) => {
    e.preventDefault();
    const email = (this.emailInput && this.emailInput.value) || null;
    if (this.emailInput && !email) {
      return;
    }
    try {
     // poziv konstane uz drugog faila
        await sendRequestToServer({ email });
      if (this.emailInput) {
        this.emailInput.value = ' ';
      }
      console.log('Email was sent');
    } catch (err) {
      console.log(err);
    }
  };

  render() {
    return (
```

```
      <div style={{ margin: '20px ',  padding: '10px 45px' }}>
        <Head>
          <title>Neki naslov</title>
          <meta name="description" content="Objasnjenje " />
        </Head>
        <br />
        <form onSubmit={this.onSubmit}>
          <p>Dodavnje email adrese  </p>
          <input
            inputRef={(elm) => { this.emailInput = elm; }}
            type="email"
            label="Vaš email"
            required
          />
          <p />
          <button className="btn-primary" type="submit">
            Upis emaila u bazu servera
          </button>
        </form>
      </div>
    );
  }
}

export default SendEmail;
```

Mesto na kome se poziva konstanta pošto upotrebljavamo formu za to je najbolji način primene u okviru događaja onSubmit. Karakteristično je u ovom primeru upotreba async i await. Gde async šalje zahtev API-ju, dok await čeka na response od servera. Prema mnogim tutroijalima najbolje upotreba async/await je kada se upotrebljavaju u kombinaciji sa true/cach.

```
onSubmit = async (e) => {
    e.preventDefault();
    const email = (this.emailInput && this.emailInput.value) || null;
    if (this.emailInput && !email) {
      return;
    }
    try {
     // poziv konstane uz drugog faila
        await sendRequestToServer({ email });
      if (this.emailInput) {
        this.emailInput.value = ' ';
      }
```

```
    console.log('Email was sent');
  } catch (err) {
    console.log(err);
  }
};
```

Ako u nekom trećem failu napravljenja vremenska komponenta kao što je

```
  componentDidMount() {
      ........
  }
```

potrebno je samo zameniti kraj fail slanje-emaila.js

```
export default TreciFail(SendEmail);
```

Inače bi u onSubmit bio smešten u kodu priključene vremenske komponente

```
  componentDidMount() {
      onSubmit ........
  }
```

Da bi ovo bilo još jasnije pogledajte ovaj kod vezan za proizvod reacta za mobilne uređaje React native

```
const URL = "https://api.github.com/repos/facebook/react-native";

class StarCount extends Component {
 constructor() {
  super();

  this.state = {stars: "?"};
 }

 componentDidMount() {
  this.fetchData().done();
 }

 async fetchData() {
  const response = await fetch(URL);
  const json = await response.json();
  const stars = json.stargazers_count;

  this.setState({stars});
 }
```

```
  render() {
    return (
      <View style={styles.container}>
        <Text style={styles.text}>
          React Native has {this.state.stars} stars
        </Text>
      </View>
    );
  }
}
```

Preuzetog da sajta https://www.dalejefferson.com/es7-async-await-with-react-native/ . Da bi se ovaj primer primenio u čistom reactu potrebno bi bilo sem izmene u delu importa, zameniti View sa div i Text sa recimo html tagom p, obzirom da su to neke ekvivalencije sa recatom.
Primeri koda su tu da vam pojasne sam značaj u moguće upotrebe kada radite sa async/await severske react aplikacije, pošto je ovo knjiga osnove.

Kraća upustva AJAX, Axios, fetch ?

U radu sa reactom možete upotrebljavati za rad sa API-em, biblioteke kao što su AJAX, Axios, o kojim možete pronaći bliže imformacije na linkovima:

1. Axios na https://github.com/axios/axios
2. jQuery AJAX na https://api.jquery.com/jQuery.ajax/
3. U brovserima postoji i ugrađen window.fetch, https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

Nebitno koje od ovih navedenih ili nekih drugih upotrebite, potrebno je upotrebiti komponente koje imaju ograničeno vreme trajanja za svoje zahteve serveru. Na primer componentDidMount za get zahteve, kada primate podatke sa servera.

Upotreba AJAX-a

U ovom primeru biće prikazana upotreba AJAX-a prilikom koje ćemo postaviti vrednosti lokalnog stanja komponente upotrebom komponente “componentDidMount”. Ovaj primer vratiće nam JSON objekt.

```
{
  items: [
    { id: 1, name: 'Apples', price: '$2' },
    { id: 2, name: 'Peaches', price: '$5' }
  ]
}
```

```
import React from 'react';

class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      error: null,
      isLoaded: false,
      items: []
    };
  }

  componentDidMount() {
    fetch("https://api.example.com/items")
      .then(res => res.json())
      .then(
        (result) => {
          this.setState({
            isLoaded: true,
            items: result.items
          });
        },

        // Napomena: U ovom delu rešićemo važan problem obradu
        // i nastanak mogućih grešaka, ali umesto klasičnog catch()
        // bloka upotrebićemo malo drugačiji pristup rešavanja ovog
        //problema kodom ispod
        (error) => {
          this.setState({
            isLoaded: true,
            error
          });
        }
      )
  }

  render() {
    const { error, isLoaded, items } = this.state;
    if (error) {
      return <div>Error: {error.message}</div>;
    } else if (!isLoaded) {
      return <div>Loading...</div>;
    } else {
      return (
```

```
      <ul>
        {items.map(item => (
          <li key={item.name}>
            {item.name} {item.price}
          </li>
        ))}
      </ul>
    );
  }
 }
}
```

Nakon toga je potrebno ukloniti ovu komponentu, ali ne pre nego što bude završen AJAX poziv. Razlog ovome je pojava moguće greške. Na primer nemogu pročitati vrednosti za stanje(state), jer je setState vratio nedefinisan rezultat. Pa, ako će vam ovo možda praviti problem u radu aplikacije prilikom AJAX zahteva ne upotrebljavajte `componentWillUnmount` metod.

Upotreba Axios-a i Fetch-a
Za upotrebu ovog dodatnog paketa react aplikacijama prvo ga je potrebno instalisati, a kasnije importovati u kod komponente koja ga upotrebljava. Instalacija sa npm izgleda ovako

```
npm i axios -- save
```

ili ako upotrebljavate yarn

```
yarn add axios
```

a bliže informacije na https://www.npmjs.com/package/axios veb sajtu.

Dok sama upotreba je jednostavna.

1. putem CDN linkova u index.html failu
   <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
   gde vam nije potrebno da ga imate instalisanog
2. kada je instalisan sa import axios from 'axios';

Dok sama upotreba u zavisnosti od HTTP zahteva koje ima apliakcija : PUT, GET, POST, DELETE i drugo upotrebljava se ovaj kod

```
axios.put()
axios.delete()
axios.patch()
axios.options()
```

```
        axios.head()
```

Ono što čini axios možda moćnim je to da može da obradi više zahteva odjednom // zahtev u ovom slučaju je paralelan, ali povratne informacije biće izvršene samo ako su oba zahteva kompletna

```
axios.all([ axios.get('https://api.neki.com/users/post');

axios.get('https://api.neki.com/users/repos') ])

.then(axios.spread(function (userResponse, reposResponse)
        console.log('User', userResponse.data);
        console.log('Repositories', reposResponse.data);
}));
```

Kod komponente je istovetan sa kodom gde se primenjuje fetch, pa ću ga prikazati za oba slučaja u jednom kodu gde ću samo dodati red za axios zatvoren komentarom. Ali ipak evo jednog primera primene axiosa u obliku funkcije

```
function getGithubData() {
       axios.get('https://api.neki.com/users/ime_korisnika')
      .then(res => { console.log(res.data.login);
      })
     .catch(err => { console.log(err);
      });
 }

      // način pozivanja funkcije
     onClick= {this.getGithubData }
```

Samim ovim možete napraviti slično funkciju i za slanje podataka u istom kodu komponente.
Za axios i fetch ako je bitno u upotrebiti hedere kojima serveru dajemo do znaja koji tip podataka mu se šalje ili od njega traži, a failu na API-ju u tom slučaju postaviti dozvole za te tipove podataka. Ovo poslednje videćete u delu knjige koji obrađuje relaciju react-php. Sam kod može biti i ovakav kada se radi o konfigurisanju koje se kasnjie upotrebljava u komponenti koja upotrebljava oba Axios i Fetch

```
let config = {
        Osnovniurl: 'https://api.neki.com/users/ime_korisnika'
        headers: { 'Content-Type': 'application/x-www-form-urlencoded' }
};
```

```
axios.post(Osnovniurl, config)
.then((response) => {
 console.log(response.data);
});

ili kada se radi parametrima koji se primaju
axios.get('https://site.com/', {
  params: {
    parametar: 'vrednost'
  }
})
```

U svakom slučaju sam rad izgleda ovako axios ili fetch šalju zahtev (request) serveru, server odgovara slanjem svog odgovora upotrebom response, response objektom koji ima sledeći sastav:

1. **data** podatke koje vraća server. Normalno je za axios ili fetch da očekuju JSON oblik podataka, koji da bi se dalje upotrebljavali u aplikaciji moraju se rastaviti na delove
2. **status** HTTP kod koji se vraća sa servera statusnom porukom u statusText
3. **statusText** HTTP statusna poruka koju je vratio server
4. **headers:** headers vraćen sa servera
5. **config** orginalna konfiguracija zahteva
6. **request** stvarni XMLHttpRequest objekat kada se pokrene u pretaživaču.

Drugi objekat koji je takođe bitan u primeni oba je objekat grešaka koji ima sastav:

1. **message** tekst poruke koji sadrži opis greške koja se pojavila
2. **response** objekt sa opisom grešeke koja se pojavila u radu ili povratni podaci
3. **request** stvarni XMLHttpRequest objekat kada se pokrene u pretaživaču
4. **config** orginalna konfiguracija zahteva

Napomena: pogledajte link https://www.sitepoint.com/axios-beginner-guide/ . Ostali tipovi formata podataka koje možete slati upotrebom axiosa osim JSON formata su arraybuffer, blob, document, text, ili stream.

Fetch

Od mnogih ovih dodataka upotrebio sam whatwg-fetch, tako da ću u kraćim crtama njega i opisati. Instališe se kao i sve drugo u reactu

npm i @xg-wang/whatwg-fetch

O nejmu kao i o mnogim potrebnim paketima na oficijelnom sajtu

https://www.npmjs.com/package/@xg-wang/whatwg-fetch

Upotreba u komponentama gde vam je on potreban, posle importovanja reacta

```
import 'whatwg-fetch'; // kada ne upotrebljavate destruktor i možete u kodu
                       //pristupiti bilo kom delu whatwg-fetch paketu

import {fetch} from 'whatwg-fetch';// prilikom upotrebe pojedinacnog dela –
                                   // upotreba destruktora
```

Primena kada su u pitanju JSON podaci, osnovni kod je ovaj

```
fetch('/posts.json')
  .then(function(response) {
    return response.json()
  })
.then(function(json) {
    console.log('parsed json', json)
  })
.catch(function(ex) {
    console.log('parsing failed', ex)
  })
```

Ovo je jedan od načina, drugi je upotreba arrow funkcije

```
fetch('/users', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    name: 'neko_ime',
    login: 'sifra',
  })
})
```

Drugi primer upotrebe

url je link ka failu koji prima podatke i salje ih tamo gde ste to odredili.

```
fetch(url, {
  method: "POST",
    headers: {
    "Content-Type": "application/json"
```

```
  },
  body: JSON.stringify(data),
  credentials: "same-origin"
}).then(function(response) {
  response.status     //=> number 100–599
  response.statusText //=> String
  response.headers    //=> Headers
  response.url        //=> String
  return response.text()
}, function(error) {
  error.message //=> String
})
```

Možete upotrebiti ovaj kod zatvoren sa kodom komponente gde pre njega dodajete uslove validacije upotrebom regulativnih izraza (regexp).

```
import React from 'react';
import 'whatwg-fetch';

class FetchComponent extends React.Component {
    constructor() {
        super();
        this.state = {
            users: {},
            errors: {},
        };
        this.handleChange = this.handleChange.bind(this);
        this.handleSubmit = this.handleSubmit.bind(this);
    }

    handleChange(event) {
        let users = this.state.users;
        let field = event.target.name;
        users[field] = event.target.value;
        this.setState({ users });
    }
    handleSubmit(event) {
        event.preventDefault();
        let dataURL = "http://localhost/fetchReact/server/test.php";
        let h = new Headers();
        let req = new Request( dataURL,{
            headers: h,
            method: 'POST',
            body: JSON.stringify({users: this.state.users})
```

```
    });
    console.log(this.state.users,"users");
    fetch(req)
       .then((response)=>{
       if(response.ok){
          return response.json();

       }else{
          throw new Error("losa konekcija");
       }
       })
       .then((users) =>{
          alert("forma je poslata");
       })
       .catch( (err) =>{
          console.log('Error', err.message);
       });
}
render() {
   return (
      <div>
         <form onSubmit={this.handleSubmit.bind(this)}>
           <label htmlFor="name">Username </label>
              <input name="name" type="text" onChange={this.handleChange}
                value={this.state.users["name"]} />
           <br/>
           <label htmlFor="password">Password</label>
              <input name="password" type="password"
               onChange={this.handleChange}
                value={this.state.users["password"]} />

           <br/>
           <label  htmlFor="age"> Age </label>
              <input name="age" type="text" onChange={this.handleChange}
                 value={this.state.users["age"]} />

           <br/>
           <label  htmlFor="email">Email </label>
              <input name="email" type="text" onChange={this.handleChange}
                value={this.state.users["email"]} />

           <br/>
           <label  htmlFor="website">Website</label>
              <input name="website" type="text"
```

```
                    onChange={this.handleChange}
                    value={this.state.users["website"]} />
                <br/>
                <input type="submit" value="Slanje" />
            </form>
        </div>
    );
  }
}

export default FetchComponent;
```

Mada se može upotrebiti i ovaj kod

```
        let defaultOptions = {
            url:'',
            method:'POST',
          mode: 'cors',
          headers:{
                  'Access-Control-Allow-Origin':'*'
                  },
         body:null,
        };

let UploadFile=function(options){
let header = new Headers({
  'Access-Control-Allow-Origin':'*',
  'Content-Type': 'multipart/form-data'
});

let opt = Object.assign({}, defaultOptions, options);
 let sentData={
  method:opt.method,
  mode: 'cors',
  header: header,
  body:opt.body || ''
};
return new Promise((reslove,reject)=>{
  fetch(opt.url, sentData)
    .then(response=> response.json())
    .then(responseText=>{
      let resp = typeof responseText === 'string' ? JSON.parse(responseText) :
                responseText;
      //console.log(resp);
```

```
            reslove(resp);
        }).catch(err=>{
        //console.log(err);
        reject(err);
    });
}).catch(err => {
        console.log('error');
    });
}
export default UploadFile;
```

Kada je reč o slanju failova, slika i sličnog binarnog materijala, gde je potrebno upotrebiti multipart-form-data.

Više o upotrebi fetch-a na https://github.github.io/fetch/ .

Razmena podataka kroz dubinu aplikacije (React.Context)

Ponekada je potrebno vršti razmenu podataka nebitno na koju dubinu se ti podaci razmenjivali između aplikacije i servera ili po njenoj dubini između više korisnika. Princip je kao i svuda u react aplikacijama, jedna komponenta je vlasnik podataka u obliku vrednosti stanja koji se prosleđuju podkompnentama koje ih prikazuju vrednostima za props. Ali tako se može pristupiti jedan ili dva nivoa od počenog. Da bi se pristupalo na dubljim nivoima u aplikaciji često je u upotrebi redux, flux no kako ova knjiga je nešto što je osnova možemo primeniti kod dole. Gde se upotrebljava react context i fragment, kako bi smo mogli proslediti podatke preko vrednosti za props constante family prosleđujemo klasi person. Pođimo redom. Da bi bilo malo jasnije kompletan kod je smešten u jedan fail koji posle

importujemo u fail src/App.js.

```
import React, { Component } from 'react';
import './stul.css';
// izrada instance new context
const MyContext = React.createContext();

// izrada provider komponente
class MyProvider extends Component {

    state = {
        name: 'Prvo ime',
        age: 20,
        cool: true
    };
    render() {
        return (
```

```
// akcija ili radnje koje se izvršavaju  na poziv klikom
          <MyContext.Provider value={{
             state: this.state,
             growAYearOlder: () => this.setState({
                age: this.state.age + 10,
                name:  "Drugo ime"
             })
          }}>
             {this.props.children}
          </MyContext.Provider>
       )
    }
}

const Family = (props) => (
    <div className="commentBox">
       <Person />
    </div>
);

class Person extends Component {

    render() {
       return (
          <div className="person">
             <MyContext.Consumer>
                {(context) => (
                   <React.Fragment>
                      <p>Age: {context.state.age}</p>
                      <p>Name: {context.state.name}</p>
                      <button onClick={context.growAYearOlder}>Dugme</button>
                   </React.Fragment>
                )}
             </MyContext.Consumer>
          </div>
       )
    }
}
class App extends Component {
    render() {
       return (
          <MyProvider>
             <div className="container" style={{background: "#fab"}}>
```

```
                <p className="h2">Ovo je iz App komponente</p>
                <Family />
            </div>
        </MyProvider>
    );
  }
}

export default App;
```

Ovaj fail možete nazvati kako želite. Na početku napravićemo prvo klasu App kojoj ćemo dodati vrednosti za stanja.

```
class App extends Component {

  state = {
    name: 'Prvo ime',
    age: 20,
    cool: true
  };
  render() {
    return (
          <div className="container" style={{background: "#fab"}}>
          <p className="h2">Ovo je iz App komponente</p>
              <Person />
          </div>
    );
  }
}

export default App;
```

Zatim napraviti klasu Person koju ćemo pošto se radi o jednom failu samo prijaviti html tagom u okviru rendera clase App

```
class Person extends Component {

  render() {
    return (
      <div className="person">
          <p>Pojava iz  klase Person</p>
      </div>
    ):
```

```
    }
}
```

I na ekranu ugledaćete samo tekst koji je uokviren html tagovima paragrafa jedan ispod drugog. Da bi vlasnik vrednosti stanja mogao da pošalje te vrednosti svom korisniku potrebno je prvo omogućiti dodavanjem u pod komponentu <Person /> . Uzeću da je potrebno prikazati ime osobe, znači:

```
<Person  name={this.state.name}/>
```

Ovim je vrednost stanja name pridodata podkomponenti ili svom child-u koji upotrebom u svom delu props-a može je pročitati i pošto je pozvan u glavnu komopnentu App i prikazati taj pročitan sadržaj, tako što naš primer u uokviru p taga Person klase dodajemo kod

```
<p>Pojava iz  klase Person { this.props.name}</p>
```

Kao što vidite princip je isti kao i u predhodnim delovima knjige. U klasi ili komponenti koja je vlasnik podataka se prvo pozove njeno dete ili child onda mu se u velikim zagradama doda mogućnost da može da prihvata promene vrednosti stanja i to da je to jednako nečemu što će dobiti props u childu. Na ovaj način komuniciraju parent i child. No pođimo dalje. Prepostavimo da Person treba da dobije indirektno vrednost iza svoje props preko još jedne klase ili komponente koja bi prosledila njoj te vrednosti. Pošto je jedna osoba deo nekog društva, porodice i sl. Napravićemo jednu konstantu preko koje ćemo prihvatiti vrednosti stanja iz App i proslediti to Person klasi. Da se nebi zbunili klase i funkcije mogu da sadrže povratne informacije u kojima je moguće smestiti html tagove a sa njima i druge informacije. Zbog same vežbe i raznolikosti koju pruža sam react primeniću strelastu funkciju kojom ću dodeliti vrednost kostanti Family

```
const Family = (props) => (
   <div className="commentBox">
      <Person />
   </div>
);
```

Gde sam sada pozivao klasu Person da joj dodelim vrednosti za njen props. Sama dodela je označena u malim zagradama. I naravno potrebno je prijaviti Family umesto Person u App klasi. To se postiže

```
<Family  name={ this.state.name}/>
```

Na ovaj način smo omogućili prolaz podataka kroz dve komponente, pošto i Person i Family mogu biti odvojeni failovi, pa tako i posebni delovi iste komponente. Obzirom da će doći do greške ako se na ovaj način radi iz razloga što nemože da se

vrši direktno ovako već se mora izvršiti dodela podataka u konstanti Family, jer komponenta Person mora da ima svoju vrednosti za props

```
const Family = (props) => (
   <div className="osobine">
      <Person name={ props.name}/>
   </div>
);
```

Da bi mogla da pročita i vrati to nazad na ekran. Tako smo uspešno prošli kroz dve komponente ili tri nivoa razmene podataka u react aplikaciji. Izgleda malo zamršeno na prvi pogled, ali react okruženje nudi još jednostavniji pristup rešavanju ovakvih problema upotrebom redux-a. Gde imamo jedno mesto sa podacima čije vrednosti prosleđujemo po potrebi onamo gde je to potrebno u aplikaciji. Zato ćemo upotrebiti Provajder i Consumer. Predhodno potrebno je upotrebiti React.createContext i pridružiti ga nekoj konstanti da bi ga mogli upotrebljavati u aplikaciji.

```
const myContext = React.createContext();
```

Iza čega pravimo komoponentu provajder koja je i vlasnik podataka vrednosti stanja koje prosleđujemo ostalim komponentama. Znači vrednosti stanja koja su bila u klasi App pomerićemo u komponentu provajder.

```
class myProvider extends Component {

   state = {
      name: 'Prvo ime',
      age: 20,
      cool: true
   };
   render() {
      return (
            <myContext.Provider value=" Neka vrednost iz provajdera">
                {this.props.children}
            </ myContext.Provider >
      );
   }
}
```

Pošto smo napravili komponentu Provajder, potrebno ju je uključiti u App komponentu. U reactu je pravilo ako upotrebimo da je nešto kao osobina children, označava da se u pisanju koda upotrebljavaju oznake kao i kod običnog html-a. Znači otvoreni tag, ono što se postavlja između je child i zatvoren tag na kraju. U

našem slučaju potrebno je između tagova myProvider postaviti sve ono što se nalazi u return delu klase App, kako bi se omogućio učinak komponente myProvider.

```
        return (

           <myProvider>
                  // ovo između div tagova tretira se kao child componente
                  // myProvider, jer se nalazi
                 // između tagova <myProvider> child</ myProvider >
              <div className="container" style={{background: "#fab"}}>
                  <p className="h2">Ovo je iz App komponente</p>
                 <Person />
               </div>
           </ myProvider >
      );
```

Što znači da se može i Person tag napisati isto kao i za slučaj provajdera u nekim sličnim prilikama. Ali vratimo se našem problemu. Kako bi izbegli moguće pojave grešaka potrebno je uneti i u komponentu Family promene, tačnije izbrisati deo koda name={ props.name}

```
const Family = (props) => (
   <div className="commentBox">
      <Person />
   </div>
);
```

Ono što je potrebno u ovom slučaju je dodavanje neke akcije dešavanja, to jest pozivanja promena vrednosti u stanjima komponenti. Pošto je nama u ovom slučaju potrebno da se promene vrše u komponenti Person tu ćemo i omogućiti promenu dodavanjem konteksta i kosumera tagom <myContext> u otvorenoj varijanti. Obzirom da je context uvek funkcija tako ćemo je i upotrebiti, samo u skraćenom obliku kao strelastu funkciju

```
<myContext.Consumer>
{(contex) => (
  <p> Za probu neki tekst</p>
 )}
</myContext.Consumer >
```

Ono što će se pojaviti na ekranu je pored svega ovaj tekst između p tagova, ali ako tu između p tagova iz teksta dodamo

```
{(contex) => (
  <p> Za probu neki tekst{ contex)}</p>
 )}
```

ugledaćemo i vrednosti iz provajdera.

```
<myContext.Provider value=" Neka vrednost iz provajdera">
```

Što znači da smo prošli kroz mnogo nivoa komponenti, nameće se pitanje kako omogućiti promene vrednosti pozvane nekim događajem u komponenti koja ih poziva. Pošto je vlasnik vrednosti stanja koja se mogu menjati komponenta provajder tamo ćemo i omogućiti čitanje i slanje tih podataka komponenti koja ih zahteva nekim svojim događajem. Pošto je ovo rad sa objektima i pristupom pojedinačnim delovima objekta umesto statične vrednosti za value u provajderu

```
<myContext.Provider value={{state: this.state }}>
```

omogućiti dinamičke promene vrednosti tako što ćemo omogućiti komponenti Person da pristupi jednoj ili više vrednosti članova objekta koje šalje provajder contextom

```
{(contex) => (
  <p> Ime:{ contex.state.name )}</p>
  <p> Godine:{ contex.state.age )}</p>

 )}
```

Ovako napisano proizvešće pojavu greške :
Syntax error : Adjacent elements must be wraped in an encolsing tag.
Što nas upozorava da je potrebno upotrebiti React.Fragment jer pristupamo pojedinačnim delovima.

```
{(contex) => (
   <React.Fragment>
     <p> Ime:{ contex.state.name )}</p>
      <p> Godine:{ contex.state.age )}</p>
  </React.Fragment>
 )}
```

Što će dati na ekranu ispravan sadržaj vrednosti koje su upisane u vrednostima stanja komponente Provajder. Znači uspeli smo poslati vrednosti stanja iz provajdera, preko kosumera, familije o jednoj osobi i to prikazati u glavnoj komponenti App. No ono što je bit ovog dela knjige je kako te podatke možemo promeniti recimo klikom na neko dugme u neke druge vrednosti? Pošto je

provajder vlasnik podataka tamo ćemo i omogućiti te promene. Da bi takve promene bile moguće potrebno je upotrebiti this.setState i povezati to sa nekim imenom stanja koje se menja. Sa druge strane ovo zahteva i upotrebu funkcije koja će pozivati te promene. Što bi dalo ovaj kod dole niže

```
<MyContext.Provider value={{
   state: this.state,
   growAYearOlder: () => this.setState({
      age: this.state.age + 10,
      name:  "Drugo ime"
   })
}}>
```

Ako bi se u state za name u provajdera uneo neki niz onda bi se svakim klikom koji ćemo dodati u drugoj komponenti menja, u tom slučaju bi se svakim klikom menjale godine starosti sa imenom. U ovom slučaju godine će rasti sa svakim klikom na dugme u komponenti Person izmedju consumer tagova a ime imaće defoltnu vrednost i koliko promeniće se na stanje u ovom delu dodato u Drugo ime.

```
return (
   <div className="person">
      <MyContext.Consumer>
         {(context) => (
            <React.Fragment>
               <p>Age: {context.state.age}</p>
               <p>Name: {context.state.name}</p>
               <button onClick={context.growAYearOlder}>Dugme</button>
            </React.Fragment>
         )}
      </MyContext.Consumer>
   </div>
```

Jer, sam u istu funkciju povezao promene i imena i godina.

Primena funkcija u komponentama

Funkcije u reactu se normalno upotrebljavaju kao i u svim progamskim govorima potpuno ravnopravno prilikom izrade aplikacija. Ali pošto je react okruženje takvo, kakvo je pa nije moguće primeniti funkciju bez službene reči this, prilikom njenog pozivanja (to važi za slučaj kada ta funkcija je u okviru klase koja je poziva).
Te u slučajevima upotrebe događaja klik pozivnaje funkcija kao props child komponente se vrši ovako

```
<button onClick={this.handleClick}>
```

Ali, kada se želi napraviti pristup roditeljskoj komponenti potrebno je povezati funkciju sa istancom komponente. Što se postiže na više načina, čime se omogućava da funkcije imaju pristup atributima komponenti kao što su `this.props` i `this.state`. U zavisnosti koja je sintaksa u upotrebi možete primeniti u radu:

Povezivanje u konstruktoru (ES2015)

```
class Neka extends Component {

  constructor(props) {
    super(props);
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() {
    console.log('Click happened');
  }
  render() {
    return <button onClick={this.handleClick}>Click Me</button>;
  }
}
```

Kao osobinu klase

```
class Neka extends Component {

  handleClick = () => {
    console.log('Click happened');
  }
  render() {
    return <button onClick={this.handleClick}>Click Me</button>;
  }
}
```

Povezivanje u delu rendera

```
class Nesto extends Component {

  handleClick() {
    console.log('Click happened');
  }
  render() {

    return <button onClick={this.handleClick.bind(this)}>Click Me</button>;
  }
}
```

Kao Arrow funkciju u renderu

```
class Neko extends Component {

  handleClick() {
    console.log(Pokrenuti događaj');
  }
  render() {
    return <button onClick={() => this.handleClick()}>Klikni!!</button>;
  }
}
```

Ova primena poziva funkcije u render delu komponente prilikom svakog pokretanja, stvara novu funkciju prilikom svakog pokretanja komponente što ima odraz na performanse.

Zašto je neophodno povezivanje svega ?

Ako pogledamo ova dva na izgled koda u Java Skriptu, primetićemo da nisu ekvivalentna:

```
obj.method();

var method = obj.method;
method();
```

Ali metodom povezivanja omogućavamo da drugi kod radi na isti način kao i prvi. Za react je tipično da dođe do povezivanja kako bi došlo do prolaska jedne komponete kroz drugu. Ako uzmemo za primer

```
 <button onClick={this.handleClick}>
```

se omogućava poziv `this.handleClick`, ali ponekada je nepotrebno to primenjivati u render metodu ili lifecycle metodu: jer ne prelazim na druge komponete. Da bi ste detaljnije razumeli kompletan problem pogledajte na linku http://yehudakatz.com/2011/08/11/understanding-javascript-function-invocation-and-this/

Zašto upotrebljena funkcija se poziva svaki put kada se komponenta renderuje ?

Odgovor je jednostavan komponenta uvek poziva sve svoje što joj pripada, da bi mogla da radi kako je to predviđeno. Ali ono što nije ispravno u reactu je ovakav način pozivanja, gde je pozvani događaj prosleđen kao refrenca kod dole:

```
render() {
  return <button onClick={this.handleClick()}>Click Me</button>
}
```

Te je potrebno uvek prosleđivati funkcije bez malih zagrada, kao što je prikazano u kodu dole

```
render() {
 return <button onClick={this.handleClick}>Click Me</button>
}
```

U svakom slučaju nameće se pitanje kako proslediti parametre kroz upotrebu poziva događaja ili nekim povratnim pozivom? U tom slučaju nam može pomoći strelasta funkcija koja okružuje pozvani događaj kojim se poziva i parametar id.

```
<button onClick={() => this.handleClick(id)} />
```

Što je ekvivalentno upotrebi službene reči `.bind`, kao u primeru dole, samo što je ovo kraći način pisanja koda :

```
<button onClick={this.handleClick.bind(this, id)} />
```

Kroz samu strelastu funkciju je tako i moguće propustati parametre. Što pokazuje i kod komponente dole koja nam pokazuje na koje smo slovo kliknuli.

```
import React  from 'react';

const A = 65;

export default class Alphabet extends React.Component {
   constructor(props) {
      super(props);
      this.handleClick = this.handleClick.bind(this);
      this.state = {
         justClicked: null,
         letters: Array.from({length: 26}, (_, i) => String.fromCharCode(A + i))
      };
   }
   handleClick(letter) {
      this.setState({ justClicked: letter });
   }
   render() {
      return (
         <div>
            Just clicked: {this.state.justClicked}
            <ul>
               {this.state.letters.map(letter =>
                  <li key={letter} onClick={() => this.handleClick(letter)}>
                     {letter}
```

```
                </li>
              )}
            </ul>
          </div>
        );
    }
}
```

Prolazak parametara kao data atributa

Osim toga, možemo upotrebiti DOM API za čuvanje podataka potrebnih za rukovanje događajima. Pa je preporučljivo pogledati ovaj primer pristupa za slučaj kada morate da optimalizujete veliki broj elemenata ili imate render koji se oslanja na React.PureComponent.

```
import React from 'react'

const A = 65; // ASCII character code

class Alphabet extends React.Component {
  constructor(props) {
    super(props);
    this.handleClick = this.handleClick.bind(this);
    this.state = {
      justClicked: null,
      letters: Array.from({length: 26}, (_, i) => String.fromCharCode(A + i))
    };
  }

  handleClick(e) {
    this.setState({
      justClicked: e.target.dataset.letter
    });
  }

  render() {
    return (
      <div>
        Just clicked: {this.state.justClicked}
        <ul>
          {this.state.letters.map(letter =>
            <li key={letter} data-letter={letter} onClick={this.handleClick}>
              {letter}
            </li>
          )}
```

```
            </ul>
          </div>
        )
    }
}
export default Alphabet;
```

Funkcija Toogle

Ova funkcija je karakteristična za jquery, ali je moguća i u reactu jer je i sam react i njegovo okruženje java skript kao i jquery.
Šta radi ova funkcija? Ona otvara i zatvara prostor u kome se mogu naći dodatni podaci nekog elementa na veb stranici jednostavno jednim klikom na ono kome smo dodelili da pokrene tu funkciju. Uobičajena praksa je da se to povezuje na neko dugme kao u primeru:

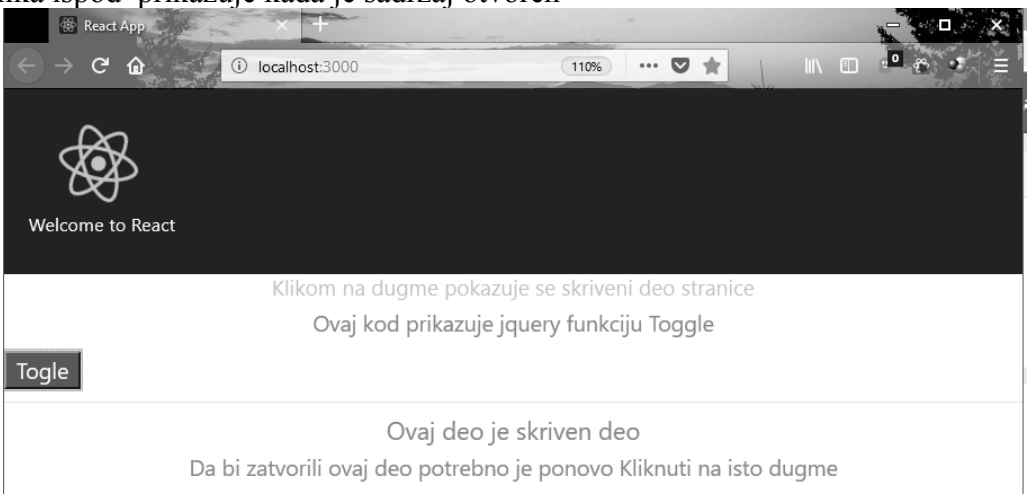
```
import React from 'react';

export default class App extends React.Component {
   constructor() {
      super();
      this.state = {
         show: false
      };
      this.toggleDiv = this.toggleDiv.bind(this);
   }

   toggleDiv(){
      const { show} = this.state;
      this.setState({ show: !show })
   }

render(){
   return (
     <div>
        <h3 className="text-center text-warning">
             Klikom na dugme pokazuje se skriveni deo stranice
        </h3>
        <p className="text-center text-info">
        Ovaj kod prikazuje jquery funkciju Toggle
        </p>
      <button className="btn-primary" onClick={ this.toggleDiv}>
Togle</button>
       <hr/>
    {this.state.show && <Box/>}
```

```
      </div>
    );
  }
}
class Box extends React.Component{
   render() {
      return (
         <div>
            <h2 className="text-center text-success">
               Ovaj deo je skriven deo </h2>
             <p className="text-center text-info">
                Da bi zatvorili ovaj deo  potrebno je ponovo Kliknuti na isto dugme
             </p>
         </div>
      );
   }
}
```

Ukoliko vam je potrebno da sadržaj bude prvo prikazan a potom da se klikom na dugme zatvori potrebno je u konstruktoru zameniti false sa true.

```
   constructor() {
      super();
      this.state = {
         show: false
      };
```

Slika ispod prikazuje kada je sadržaj otvoren



Paginacija

Paginacija je moglo bi se nazvati proces kojim veliku količinu podataka prikazujemo na jednom mestu u virtualnim veb stranicama, s time što programski možemo da odredimo broj prikazanih stavki ili redova po jednoj virtualnoj stranici. Pri čemu možemo da pristupamo određenom broju virtualnih stranica, ili pak početnoj ili krajnjoj stranici sa podacima. Princip rada paginacije je da prvo se pročita koliko redova ima određena tabela u bazi kojoj se pristupa da bi se ti podaci prikazali. To je jedan parametar koji učestvuje u izradi broja virtualnih stranica. Broj redova u ciljnoj tabeli iz koje se žele prikazati podaci se deli sa brojem redova koji se žele u jednom momentu prikazati na jednoj virtualnoj stranici. Tako se dobija ukupni broj virtalnih stranica. Ovaj dobijeni broj virtualih stranica mora se prevesti u celobrojnu vrednost.
U kodu paginacije potrebno je postaviti uslove ako se dostigne maksimalna vrednost ili vrednost broja nule, kao minilana vrednost broja virtualih stranica. Tačnije kada je postignuta minimalna vrednost onda se to dugme deaktivira (isključi i sl.) i onda je moguće ići samo prema maksimumu i obratno. Dok između ova dva dugmeta mogu se postaviti nekoliko između na kojima stoji napisano za koju su oni stranicu namenjni.

```
import React from 'react';
import './pag.css';

class Pagination extends React.Component {
   constructor(props, context) {
      super(props, context);
      this.state = {
         currentPage: null, // trenutna stranica
         pageCount: null, // vrednost brojača je null
      }
   }

   componentWillMount() {
      const startingPage = this.props.startingPage
         ? this.props.startingPage
         : 1;
      const data = this.props.data;
      const pageSize = this.props.pageSize;
       // brojanje ukupnog broja stranica
      let pageCount = parseInt(data.length / pageSize);
      if (data.length % pageSize > 0) {
         pageCount++;
      }
      this.setState({
         currentPage: startingPage,
```

```
      pageCount: pageCount
    });
  }

  setCurrentPage(num) {
    this.setState({currentPage: num});
  }

  createControls() {
    let controls = [];
    const pageCount = this.state.pageCount;
    for (let i = 1; i <= pageCount; i++) {
      const baseClassName = "pagination-controls__button";
      const activeClassName = i === this.state.currentPage ?
            `${baseClassName}--active` : '';
      controls.push(
        <div key={i}
          className={`${baseClassName} ${activeClassName}`}
          onClick={() => this.setCurrentPage(i)}>
          {i}
        </div>
      );
    }
    return controls;
  }

  createPaginatedData() {
    const data = this.props.data;
    const pageSize = this.props.pageSize;
    const currentPage = this.state.currentPage;
    const upperLimit = currentPage * pageSize;
    const dataSlice = data.slice((upperLimit - pageSize), upperLimit);
    return dataSlice;
  }

  render() {
    return (
      <div className="pagination">
        <div className='pagination-controls'>
          {this.createControls()}
        </div>

        <div className="pagination-results">
```

```
                <hr/>
                {React.cloneElement(this.props.children,
                    {data: this.createPaginatedData()})}
            </div>
        </div>
    );
  }
}

Pagination.defaultProps = {
  pageSize: 25, // broj prikazanih redova po stranici
  startingPage: 1  // početna stranica
};

class Example extends React.Component {
  render() {
    const data = this.props.data;
    return (
          <div className="example " >
            {data.map((item) => {
              return (
                <div className="example__item  btn-info " key={item.id}>
                  {item.id} {item.first_name} {item.last_name}
                </div>
              );
            })}
          </div>
    );
  }
}

export default class App extends React.Component {

  render() {
    return (
          // pridruživanje podataka upotrebom funkcije
        <Pagination  data={testData()}  >
          <Example />
        </Pagination>
    );
  }
}
```

```
// U ovom  delu možete navesti koliko želite podataka za vežbu, ali kada se primer poveže na bazu, onda -// će podatke uzimati iz baze
function testData() {
   return (
      [{"id":1,"first_name":"Mika","last_name":"Moloc"},
         {"id":2,"first_name":"Mila","last_name":"Gordic"},
         {"id":3,"first_name":"Milivoje","last_name":"Goranic"},
         {"id":4,"first_name":"Milosava","last_name":"Rebic"},
         {"id":5,"first_name":"Koja","last_name":"Poji"},
         ]
   );
}
```

Sam izgled možete primeniti ovaj primer css koda, ali i promeniti ga nekim vlastitim kodom, shodno situaciji.

fail pag.css

```
body {
   background-image: linear-gradient(to bottom, #ffdfa2, #7ebddd);
   display: flex;
   justify-content: center;
   font-family: 'Roboto Condensed', Helvetica, Arial, sans-serif;
   padding: 1em;
   min-height: 100vh;
}
.pagination-results{
   position: fixed;
   margin-left: 0px;
   width: 40%;
   margin-top: 50px;
}
.pagination-controls {
   display: flex;
}

.pagination-controls__button {
   background-color: white;
   border-radius: 0.15em;
   box-shadow: 0 1px 1px rgba(0, 0, 0, 0.1);
   cursor: default;
   margin: 0 0.25em;
   padding: 0.5em 0.75em;
```

```
    height: 40px;
}

.pagination-controls__button:first-child {
    margin-left: 0;
 }

.pagination-controls__button:last-child {
    margin-right: 0;
 }

.pagination-controls__button:hover:not(:active) {
    transition: transform 0.15s;
    transform: translateY(-1px);
 }


.pagination-controls__button--active {
    background-color: #4D7EA8;
    color: white;
}
.pagination-controls__button--active :hover, .pagination-controls__button--
active:hover:not(:active) {
    transform: none;
 }
.example {

    background-color: rgba(255, 255, 255, 0.5);
    border-radius: 0.15em;
    box-shadow: 0 1px 1px rgba(0, 0, 0, 0.1);
    margin: 1em 0;
    padding: 0.75em 1em;
}

.example__item {
     margin: 0.5em 0;
}
.example__item:first-child {
    margin-top: 0;
 }

.example__item:last-child {
    margin-bottom: 0;
 }
```

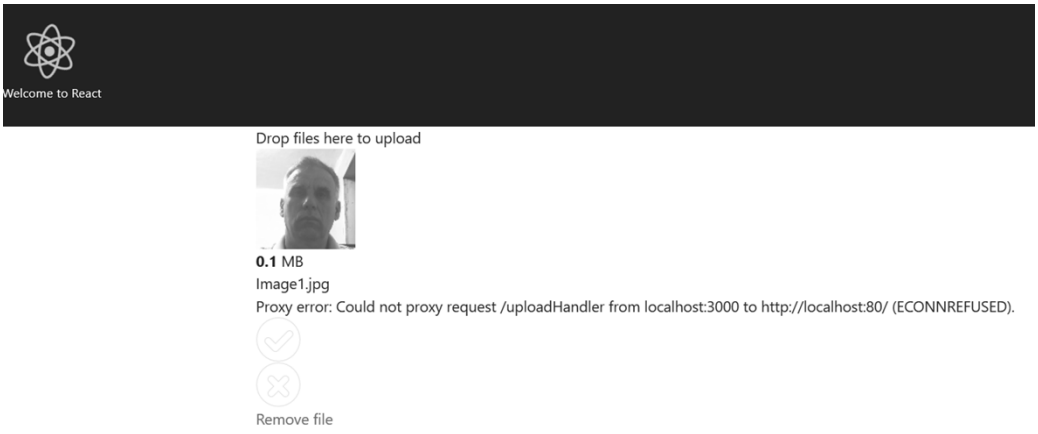
Učitavanje failova reactom

Različitim dodatnim paketima možemo omogućiti da react postavi neki fail na server, nebinto da li je to slika, ili već neki drugi tip podataka. U primerima ispod biće prikazano učitavanje slika i pdf dokumenta, koje ujedno možet i videti na ekranskom prikazu. Ovo ne znači da će se upoterbljen fail postaviti svojom fizičkom veličinom u bazu podataka, već moramo da bazi podatak samo prosledimo tekst linka gde će biti taj fail smesten na serveru. Razlog tome je izbegavanje veličine i opterećenja baze podataka, a smim time se to odražva i na količinu potrebnog vremena za prostup i ostalo umanjujući brzinu rada .

Učitavanje slika

U ovom primeru daću osnovno učitavanje slika react aplikacijom. Kao što slika prikazuje, imamo ispod slike informacije, koje se mogu konfiguracionim failom otkloniti. Da bi ovaj primer mogao da radi potrebno je instalistati dodatni reactov paket 'react-dropzone-component':

npm i react-dropzone-component




```
import React from 'react';
import DropzoneComponent from 'react-dropzone-component';

export default class Slika extends React.Component{

   render(){

      let componentConfig = {
        iconFiletypes: ['.jpg', '.png', '.gif'],
        showFiletypeIcon: true,

        // folder na serveru gde se postavlja slika
        postUrl: '/image'
```

```
        };

        let djsConfig = {
            // addRemoveLinks uključuje ili isključije pojavu linka za brisanje slike sa
            // ekrana. Standardno postavljeno na uključena pojava (true)
            addRemoveLinks: true,
            params: {
                myParameter: "Ja sam parametar!"
            }
        };

        let eventHandlers = { addedfile: (file) => console.log(file) };
        return(

            <DropzoneComponent config={componentConfig}
                               eventHandlers={eventHandlers}
                               djsConfig={djsConfig}
              className=”container “
             />
        );
    }
}
```

Oba primera samo pokazuju mogućnost za postavljanje slika na server, ali ne i direktno postavljanje, jer za tako nešto potrebno je i podesiti proxy i ostale parametre kako bi slika bila postavljena na server i upisan link na serveru do nje u bazu podataka. Odakle bi se slika pozvala na primer kada se korisnik loguje i onda se pojavi njegova profilna slika. Nešto više o primeni i radu na linku https://techhandle.net/libraries-code/React-Dropzone.html
Ali isto tako možemo i ovim kodom dole da učinimo istu stvar, samo što nema ikona prihvatanje ili odbacivanje učitanih slika koje možemo dodati.
npm i react-dropzone

```
import React from 'react';
import Dropzone from  'react-dropzone';
import’./stil.css’;

const handleDropRejected = (...args) => console.log('reject', args);

export default class ImageUpload extends React.Component {
    constructor(props) {
        super(props);

        this.state = { preview: null };
```

```
    this.handleDrop = this.handleDrop.bind(this)
  }

  handleDrop([{ preview }]) {
    this.setState({ preview })
  }

  render() {
    const { preview } = this.state;

    return (
      <section className="container">
        <Dropzone onDrop={ this.handleDrop }
            accept="image/jpeg,image/jpg,image/tiff,image/gif"
            multiple={ false } onDropRejected={ handleDropRejected
           Drag a file here or click to upload.
        </Dropzone>
        <div style={{posotion:"relative", left: "200px" }}>
        { preview &&
        <img src={ preview } style={{width:"30%",height:" 30%"}}
           className="img-thumbnail img-responsive " alt="prikaz slike" />
        }
        </div>
      </section>
    );
  }
}
```

fail stil.css

```
img {
  max-width: 200px;
  max-height: 100%;
}

img, div {
  float: left;
  margin: 10px 0 0 10px;
}
```

multiple={false} dopušta učitavanje samo jedne slike
accept="image/*" ovako postavljeno omogućava učitavanje bilo kog formata slike osim ako to ne postavite kao accept="image/jpg,image/png".
onDrop metod koji učitava sliku

Za više o ovome na linku https://css-tricks.com/image-upload-manipulation-react/

Napomena: Ako slike koje koristite postavite u public folder, znači one su na portu 3000 (standarni port za react aplikacije koji pravi Nodejs), u tom slučaju možete upotrebljavati i kod

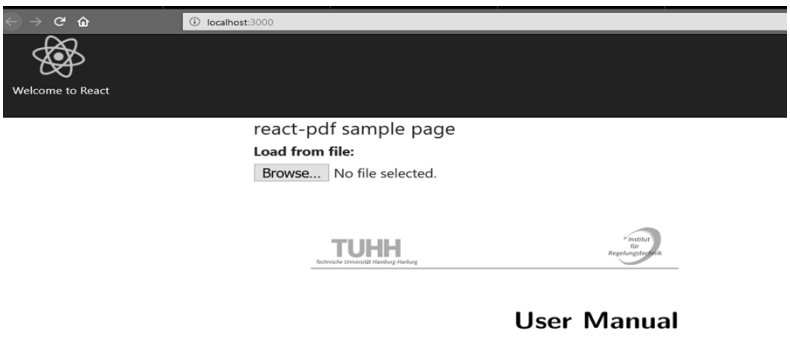
<img src={img/nazivslike.ekstenzija} alt=" "}
Mada je bolje rešenje upotrebiti za tu opreaciju sa require('./putanja do slike/nazivslike.ekstenzijaslike'}

Različiti dodati paketi reacta na sajtu https://www.npmjs.com omogućavaju velike mogućnosti od kako što ste videli učitavanje slika do neslućenih mogućnosti samo je potrebno o njima dobro pročitati upustva. Jer, od toga zavisi koliko ćete brzo pokrenuti bilo koji od njih. Zanimljiv je dodatak koji omogućava čitanje pdf failova react-pdf.

Učitavanje i prikaz pdf failova

```
import React, { Component } from 'react';
import { Document, Page } from 'react-pdf/dist/entry.webpack';

 const options = {
   cMapUrl: 'cmaps/',
   cMapPacked: true,
};
export default class Sample extends Component {
   state = {
      file: './FAST.pdf',
      numPages: null,
   };

   onFileChange = (event) => {
      this.setState({
         file: event.target.files[0],
      });
   };

   onDocumentLoadSuccess = ({ numPages }) => {
      this.setState({ numPages });
   };

   render() {
```

```
    const { file, numPages } = this.state;

    return (
      <div className="container">
        <header>
          <h1>react-pdf Primena u praksi</h1>
        </header>
        <div className="Example__container">

          <div className="Example__container__load">
            {/* deo za učitavanje faila sa racunara*/}
            <label htmlFor="file">Učitaj fail iz:</label>
            {' '}
            <input
              type="file"
              onChange={this.onFileChange}
            />

          </div>
          <div className="Example__container__document">
            <Document
              file={file}
              onLoadSuccess={this.onDocumentLoadSuccess}
              options={options}
            >
              {
                Array.from(
                  new Array(numPages),
                  (el, index) => (
                    <Page
                      key={`page_${index + 1}`}
                      pageNumber={index + 1}
                    />
                  ),
                )
              }
            </Document>
          </div>
        </div>
      </div>
    );
  }
}
```



Klikom na dugme Browse možete i otvoriti neki svoj fail sa računara inače u osnovnom prikazu otvara se fail koji je postavljen u stanju komponente

```
  state = {
        file: './FAST.pdf',
        numPages: null,
    };
```

Napomena: Da bi se promenio broj porta prilikom upotrebe noda i reacta u toku razvoja aplikacije potrebno je u failu start-js koji se nalazi u folderu na putanji:

node_modules/react-scripts/scripts/start.js

Pronaći ovu liniju koda dole niže i promeniti broj porta u željeni

var DEFAULT_PORT = process.env.PORT || *4000*;

Taj kod u mom slučaju bio je ovaj

```
// Tools like Cloud9 rely on this.
const DEFAULT_PORT = parseInt(process.env.PORT, 10) || 3000;
const HOST = process.env.HOST || '0.0.0.0';
```

Tako će u radu dok izrađujete react aplikaciju nod pokretati svoj server na tom portu gde će prikazati vaš rad umesto na njegovom standardnom portu 3000. Da ne bi stalno prekidali rad node programa i time rad njegovog servera prilikom svake promene koje napravite u kodu aplikacije prilikom njene izrade za slučaj da upotrebljavate neki od ne automatizovanih editora koda aplikaciju u razvojnom okruženju, potrebno je prvo instalisati dodatni paket nodemon

npm i nodemon

i posle toga pokretanje node servera i aplikacije u njem use čini sa

nodemon start

Na ovaj način će node monitor pratiti sve promene koje nastanu posle klika na save u vašem editoru, u suprotnom morali bi da prekidate rad servera istovremenim pritiskom na CTRL i c i da onda odogovorite znakom y, a zatim da ponovo pokrenete rad sa npm start, jer je potrebno server resetovati da bi prikazao nastale promene i kodu. Drugi način za automatizovano osvežavanje aplikacije dok je izrađujete upotreba webpack, webpack-dev-server i babela. Obzirom da je napomenuto da je ova knjiga nešto najosnovnije u cilju upoznavanja i počinjanja rad u ovom okruženju, kao i sama obimnost materije pa je nemoguće sve odjednom opisati na ovu temu možete pogledati i primer na ovom linku:
https://ipenywis.com/tutorials/Let's-Create-a-Drag-&-Drop-Image-Uploader-on-React-01
Gde objašnjena upotreba dodatnog paketa blueprintjs o kome se može videti dokumentacija na
https://blueprintjs.com/
Instalacija ovog paketa vrši se ovako kada je u pitanju yarn

yarn add @blueprintjs/core

ili kada se upotrebljava npm
npm i --save @blueprintjs/core

Paket sadržava dosta dodataka, koji pomalo podsećaju na bootstrap, dok upotreba se zasniva na dodavanju potrebnog dela koda

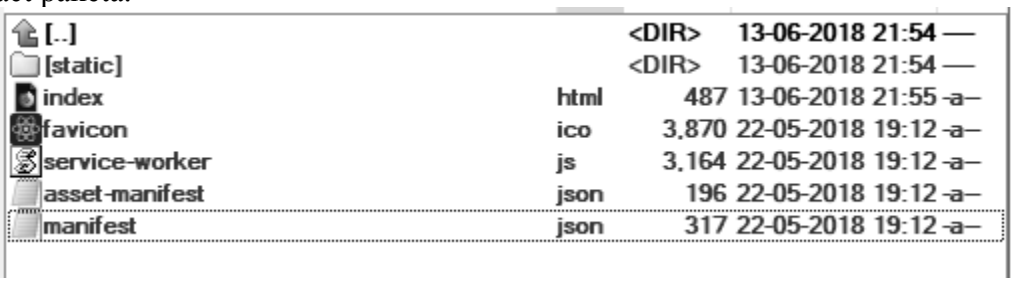
import { Button } from "@blueprintjs/core";

<Button intent="success" text="button content" onClick={incrementCounter} />

Napomena: Slike možete konvertovati u format data:image/neki format; base64

Webpack

Webpack je dodati paket u reactu koji omogućava da se od svih failovima od kojih ste napravili aplikaciju napravite jedan u principu fail u kome se nalaze podaci od svih failova iz vaše aplikacije. Što se postiže izradom faila webpack.config.js. Ali, taj produkcioni fail se dodaje u kao što vidite u jednom folderu i povezuje se preko koda index.html faila koji je u public folderu, a kopira se u takozvani producioni folder sa svim drugim failovima koje je u neku ruku i obradio jedan dodatni paket reacta koji se zove Babel (Biblija). Da vas ne bih zbunio pogledajte dole slika prikazuje minimalni sastav dobijenog direktorija posle primene ova dva dodatna react paketa.



Koji se posle toga može postaviti na server.

Instalacija

Instalcija **webpack** bundlera se obavlja instalisanjem paketa webpack, webpack-cli i webpack-dev-server.

npm install webpack –save-dev
npm install webpack-dev-server - -save-dev
npm i webpack-cli

Napomena: instalisanje više dodatnih paketa reacta možete i instalisati u jednom koraku tako što napišete ovako
npm install webpack webpack-cli webpack-dev-server - -save-dev
Webpack.config.js fail se smešta u osnovni folder aplikacije kao i babel.rc. On se može izraditi ručno prema potrebi zahteva za izradu aplikacije.

Konfigurisanje Webpack

Izradom konfiguracionog faila sapoštavamo webpacku gde se nalaze failovi od kojih se pravi sadržaj u direktorijumu koji se posle samo kopira na server.
U **webpack.config.js** failu kod može biti kao i dole u tekstu. Gde je kao polazni fail uzet fail **main.js**. Drugo što je određeno kao objekt je izlazni fail i folder. U ovom primeru koda je postavljen i port servera za razvoj na portu 8080, ali to možete izmenimo u ovom kodu i sami promeniti u koji god port želite, pri čemu se mora voditi računa da to ne budu portovi koji su već nekim stadradnim određeni za nešto drugo. Na primer port 80, 21 ili 3600, i sl., kako ne bi došlo do problema u

izvršenju koda ili da vaša aplikacija koja pokreće localhost obavesti da je taj port zauzet i da vam predloži neki drugi.
Ono što je još u ovom kodu webpack.config.js faila dato je konfiguracija za Babel loader u ovom slučaju za java skript failove i upotreba njegovih delova **es2015** i **react** presets o čemu će biti reči u delu knjige za Babel.

webpack.config.js

```
let config = {
  entry: './main.js',
  output: {
    path:'/',
    filename: 'index.js',
  },
  devServer: {
    inline: true,
    port: 8080
  },
  module: {
    loaders: [
      {
        test: /\.jsx?$/,
        exclude: /node_modules/,
        loader: 'babel-loader',
        query: {
          presets: ['es2015', 'react']
        }
      }
    ]
  }
}
module.exports = config;
```

Mada webpack.config.js može imati i ovakav kod

```
// zahtevani delovi da bi se moglo raditi
let webpack = require('webpack');
let path = require('path');

// konfigurisanje izlaznih failova i direktorija, u nekim slučajevima piše build za ime foldera
let BUILD_DIR = path.resolve(__dirname, 'src/client/public');

// konfigurisanje putanja do polaznih failova i foldera
let APP_DIR = path.resolve(__dirname, 'src/client/app');
```

```
let config = {

//polazni fail iz pozanog foldera
  entry: APP_DIR + '/index.jsx',

// izlazni folder i izrada pomenutog faila koji u ovom slučaju izrađuje babel a
webpack ga smešta u
// izlazni folder
  output: {
    path: BUILD_DIR,
    filename: 'bundle.js'
  }
};

// eksportovanje modula
module.exports = config;
```

Podešavanja u failu package.json

Da bi ste omogućili upotrebu ovako ručno napravljenog faila webpack.config.js potrebno je da u editor koda otvorite fail package.json i u njemu obrište red koda koji sadrži ovaj kod

```
 "test" "echo \"Error: no test specified\" && exit 1" .
```

Njega ćete pronaći unutar objekta koji se zove "scripts". Ovo činimo jer nam u radu neće biti potreban taj red koda za testiranje. U vašem radu možete upotrebiti i

```
webpack --watch
```

(za webpack watch pogledajte na njihovom oficijelnom sajtu https://webpack.js.org/configuration/watch)

i

```
webpack-dev-server --hot
```

koje dodajete ručno u fail package.json, a oni imaju namenu da prilikom razvoja react aplikacije osvežavaju localhost kada napraviti i sačuvate napravljenu promenu koda aplikacije. Jednostavno u script objekt kod uneste ovaj kod ispod:

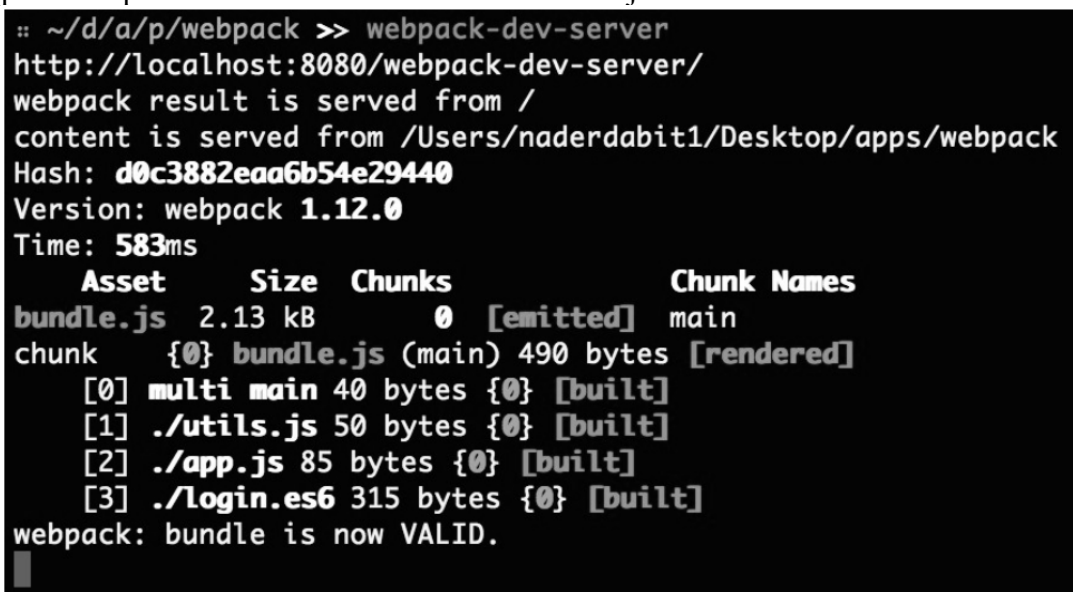
```
"start": "webpack-dev-server --hot"
```

Napomena: u ovom delu možete upotrebiti i komande webpack-dev-server --open --mode development, mada postoje i druge. Standardno webpack-dev-server koristi na lokalhostu port 8080

Ali, pre nego što pokrente rad upotrebom komande npm start webpack-dev-server. potrebno ga je i instalisati upotrebom ove komande

C:\Users\username\Desktop\reactApp>npm install webpack-dev-server –g

Znači da se on instališe globalno. Prekidači kao što je - -hot, ili - - inline, su komande za webpack-dev-server. A nakon pokretanja webpack-dev-server i po proveri ispavnosti koda na ekranu terminala dobija se ovakava slika



Slika preuzeta sa prostora
https://medium.com/javascript-training/beginner-s-guide-to-webpack-b1f1a3638460
Gde nas on obaveštava o svome radu. Webpack se može pokrenuti i zadavanjem ove komande u terminalu

npm run webpack ./app.js bundle.js

Gde webpack-u kažemo da je potrebno da uzme fail app.js i da od njega napravi fail bundle.js.

Malo detaljnije o webpack config failu

Ovo je fail kojim određujemo mnogo toga u samoj aplikaciji. Koje ćemo module uzeti za rad aplikacije, uključivanjenje Babela i dr. Ovih failova može biti više u zavisnosti od potrebe, koji mogu biti smešteni u poseban folder i onda se pozivati od strane osnovnog u root folderu aplikacije. U prvim redovima ovog faila navode se zahtevi za u prvom redu webpack, a posle i za drugo što vam bude trebalo u radu, iza toga su moduli za ekspotrovanje, plaginovi i slično. Pođimo redom.

```
module.exports = {
  entry: "./app.js",
  output: {
    filename: "bundle.js"
  }
}
```

entery je ime najvišeg nivoa failova i u njemu se postavljaju imena failova koje uključujemo u izgradnju izlaznog faila. Falovi mogu biti pojedinačni ili smešteni u neki niz failova. U gronjem kodu je samo uključen app.js fail. Kod dole pokazuje način unošenja više ulaznih failova u konfiguracioni fail

```
module.exports = {
  entry: ["./global.js", "./app.js"],
  output: {
    filename: "bundle.js"
  }
}
```

output je objekt kojim se konfiguriše ono što dobijamo na kraju rada proizvodnje. U ovom slučaju fail bundle.js, ali pored toga možemo odrediti i ime foldera gde će biti smešteni izlazni failovi zajedno sa bundle.js failom, praktično finalni folder je ono što postavljamo na server u praksi.
Nadzorni mod rada se može omogućiti unošenjem komande u terminalu

```
npm run webpack - - watch
```

ili unošenjem u konfiguracioni fail

```
module.exports = {
  entry: "./app.js",
  output: {
    filename: "bundle.js"
  },
  watch: true
}
```

Na ovaj način će webpack prilikom svake sačuvane promene vašeg koda praviti novi bundle.js fail. Ono što još se može uključiti su loaderi i preloaderi koji mogu izraditi neke delove poslova u aplikaciji. To mogu biti: file-loader, css-loader, babel-loader i sl. Naravno pre nego što ih upotrebimo moramo ih i instalisati.

```
npm install babel-core babel-loader jshint jshint-loader node-libs-browser babel-preset-es2015 babel-preset-react webpack  --save-dev
```

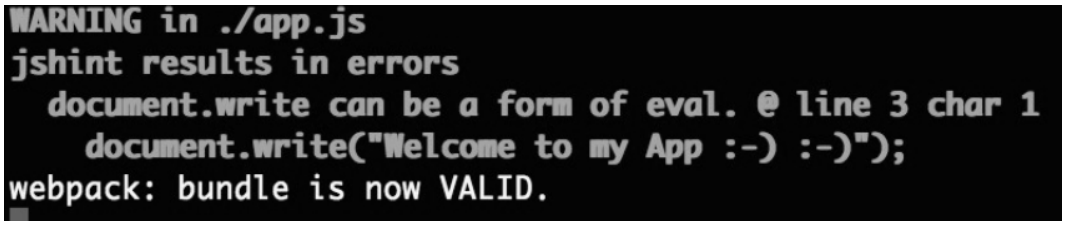
babel-core je babel npm paketa
babel-loader je modu loader za webpack
jshint je alat koji pomaže prilikom otkrivanja grešaka o potencijalnih problema u aplikaciji
node-lobs-browser je u zavisnosti od webpack-a i obezbeđuje određene biblioteke noda za korišćenje veb pretraživača
babel-preset-es2015 je preset za sve es2015 plaginove, u nekim primerima tutorijala možete naići i za novije vreme upotrebljen babel-preset-env. Samo uključivanje pomenutih može biti i ovako kao u kodu dole

```
module.exports = {
 entry: ["./global.js" , "./app.js"],
 output: {
   filename: "bundle.js"
 },
 module: {
   loaders: [
     {
       test: /\.es6$/,
       exclude: /node_modules/,
       loader: 'babel-loader',
       query: {
         presets: ['react', 'es2015']
       }
     }
   ]
 },
 resolve: {
   extensions: ['', '.js', '.es6']
 },
}
```

Gde su dodata tri ključa u loaderu, čime smo o postavili regualni test izraz kojim smo odredili da će se uzimati failovi koji imaju ekstenziju .es6 ključem test.
exclude je tu da kaže loaderu šta će isključiti ili odbaciti. Mi smo tu dodali node_modules folder
query određuje mogućnosti za prosleđivanje loaderu, tako što ih upošte kao niz upita na način koji smo već uradili u predhodnom kodu.
cacheDirectory Ovo je standarno isključeno, ali ako je uključen onda se u tom folderu smeštaju-keširaju podaci rezultata rada loadera. Tako da će webpack uvek pokušati da te rezultete pročita iz keša kako bi se izbegla ponovna rekompilacija Babela na svakom koraku.
Preloaderi su paketi koji se upotrebljavaju ispred pokretanja loadera. U našem slučaju prikazaću uključenje JSHint preloadera u webpack.config.js fail

```
module.exports = {
 entry: ["./global.js" , "./app.js"],
 output: {
  filename: "bundle.js"
 },
 module: {
  preLoaders: [
    {
     test: /\.js$/,
     exclude: /node_modules/,
     loader: 'jshint-loader'
    }
  ],
  loaders: [
   {
    test: /\.es6$/,
    exclude: /node_modules/,
    loader: 'babel-loader',
    query: {
      cacheDirectory: true,
      presets: ['react', 'es2015']
    }
   }
  ]
 },
 resolve: {
  extensions: ['', '.js', '.es6']
 }
}
```

Na ekranu terminala dobija se ovakav rezultat ako nema grešaka



Flag –p u webpack-u omogućava producioni fail koji daje sabiveniji- kraći kod producionog faila

Rad sa različitim konfiguracionim failovima

U principu doboljan je jedan webpack.config.js fail, ali zbog različitih potreba, zahteva uslova rada i razvoja same react aplikacije poželjno je napraviti posebne failove za produkciju i razvoj same aplikacije.
Te, će u ovom delu biti kraće objašnjena instalacija i upotreba nekih loadera koji se dodaju react aplikacijama. Kako je ovo primer kojim želim da pojasnim upotrebu nekih želim da vas navedem da malo više obratite pažnju i da počnente sa upotrebom ovih dodataka koji mogu da vam ubrzaju izradu i rad same aplikacije.

StripLoader

Namena mu je da uklonite prilagođene funkcije iz vašeg koda, pa je on jako koristan ako želite tokom razvoja aplikacije da upotrebljavate izveštaje o debagiranju, ali ne želite da to bude prikazano u produkcionom kodu.

Instalisanje strip-loader se vrši u delu znači samo za razvoj:

npm install strip-loader --save-dev

Izrada faila webpack-production.config.js

Kod webpack-production.config.js faila je naisan ispod:

```
var WebpackStripLoader = require('strip-loader');
var devConfig = require('./webpack.config.js');
var stripLoader = {
 test: [/\.js$/, /\.es6$/],
 exclude: /node_modules/,
 loader: WebpackStripLoader.loader('console.log')
}

devConfig.module.loaders.push(stripLoader);
module.exports = devConfig;
```

Kao što se vidi i kodu ono što je potrebno da bi ovaj fail radio kako treba je :

1. instalacija loadera
2. devConfig zahteva prisutnost orginalnog webpack.config.js faila
3. da bi varijabila var stripLoader mogla da napravi novi objekt i prosledi test i isključi ključeve

Loader: WebpackStripLoader.loader('console.log')
ova linija koda govori loaderu da ukloni sve iz konzole
WebpackStripLoader.loader() može da uzme u svom radu bilo koji broj argumanata

4. devConfig.module.loaders.push(stripLoader);—ovim kodom možemo da dodamo u naš loader niz iz naše orginalne webpack.config.js
5. module.exports = devConfig;—eksportovanje našeg novog objekta

Da bi smo videli učinak unetog koda u terminalu pišemo

webpack --config webpack-production.config.js –p

Ovim smo pokrenuli webpack sa konfiguracijskim flagom, kojim smo dozvolili upotrebu prilagođene konfiguracione datoteke .
-p ovaj flag minimalizuje produkcioni kod.
Da bi smo videli učinak predhodnih koraka otvorićemo bundle.js fail u editoru koda i ono što ćete videti tamo je da nepostoji kod kao što je console.log().

Primena Webpack-a u React aplikaciji

U daljem tekstu objasniću na manjem primeru react aplikacije kroz njenu izradu i primenu webpacka, gde će biti omogućena upotreba Babel-a, o kome će biti reč kasnije. Izradićemo App.js

```
import React from ‘react’;
export default class App extrends React.Component{
 render() {
   return (
     <div>
        Hello, {this.props.name}!
     </div>
   );
 }
}
```

Dododati u failu index.js

```
import React from ‘react’;
import ReactDOM from ‘react-dom’;
import App from ‘./app";
ReactDOM.render(
  <App name=" Necije ime" />,
  document.body
);
```

U failu webpack.config.js, napravićemo izmene da bi mogao babel-loader da radi svoj deo posla. Ovaj kod je u obliku niza, koji govori loaderu koje tipove faila da uzima u svom radu, da bi ih preveo u oblik java skripta koji razumeju današnji brovseri:

```
loaders: [
  {
    test: [/\.js$/, /\.es6$/],
    exclude: 'node_modules',
    loader: 'babel-loader'
  }
 ]
```

U ovom slučaju sam uzeo failove sa nastavkom .js and .es6 . Loaderi postoje za mnoge druge namene, s jednim ciljem da olakšaju i ubrzaju izradu react aplikacija pružajući njen siguran rad u eksploataciji. Dole su prikazani primeri koda gde se upotrebljava file loader i image webpack loader kako bi se slike postavile u prvom slučaju se podešava putanja gde se slike u produkciji direktorijuma određuje sa outputPath korisnički drugi slučaj se određuje bez toga

```
{
          test: /\.(jpe?g|png|gif|svg)$/i,
          use: [
               'file-loader?name=[name].[ext]&outputPath=omages/',
        'image-webpack-loader'

          ]
}
```

/* na sjtu https://www.npmjs.com/package/image-webpack-loader instalacija

```
 npm install image-webpack-loader --save-dev
```

regulisanje dužine naziva slike vrši se preko komande hash: kolika je dužina potrebna brojčana oznaka

```
 'file-loader?name=[hash:6].[ext]&outputPath=omages/',
```

optimalizacija slika koje se postavljaju na server aplikacija react se vrši ovim dodatkom image-webpack-loader

```
{
          test: /\.(jpe?g|png|gif|svg)$/i,
          use: [
               'file-loader?name=images/[name].[ext],
        'image-webpack-loader'
             ]
}
{
          test: /\.(jpe?g|png|gif|svg)$/i,
          use: [
```

```
            {
              loader: 'file-loader',
              options:{
                      name:[name].{ext},
                      oputpuPath: /img,
                      publicPath: /img
               }
            }
          ]
}
```

Primer upotrebe i instalacije plagina

Instalisanje plagina html-webpack-pligin-a vrši se i njegovo postavljanje u Devdependencies objekt faila package.json:

npm i html-webpack-plugin html-loader --save-dev ili sa npm i -D html-loader

Pozivnaje dodatog palgina u webpack konfiguracionom failu:

```
const HtmlWebPackPlugin = require("html-webpack-plugin");

module.exports = {
module: {
rules: [
         {
          test: /\.js$/,
          exclude: /node_modules/,
          use: {
          loader: "babel-loader"
              }
        },
        {
          test: /\.html$/,
          use: [
              {
                loader: "html-loader"
            }
        ]
      }
  ]
},

plugins: [
                new HtmlWebPackPlugin({
```

```
            template: "./src/index.html",
            filename: "./index.html"
            })
        ]
};
```

Babel

Vavilon je generički višenamenski kompajler za java skript failove. Tačnije on java skript višeg nivoa prevodi u java skript nižeg nivoa. U reactu on prevodi jsx kod u vanila java skript, pored toga uklanjaju se beline iz koda, pa na taj način i smanjuje veličinu faila koji se dobija posle njegovog rada. Što se naziva i objašnjava rečiju transpajliranje. Da bi ovo bilo jasnije prikazaću jednu strelastu funkciju, koja je u react okruženju česta pojava

```
let proizvod = b=> b*b;
```

Takav kod je nerazumljiv za današnje brovsere pa ga je potrebno prevesti u ono poznato njima, Vavilon ovaj kod prevodi u sledeći

```
var proizvod = function proizvod(b){
      return b*b;
};
```

No ovo je samo jedan deo onog što može da uradi.

Instalacija i postavljanje

Da bi upotrebljavali babel u react aplikacijama, potrebno ga je instalirati, ali pored njega potrebno je dodati i nejgov paket babel-cli (command line interpreter)

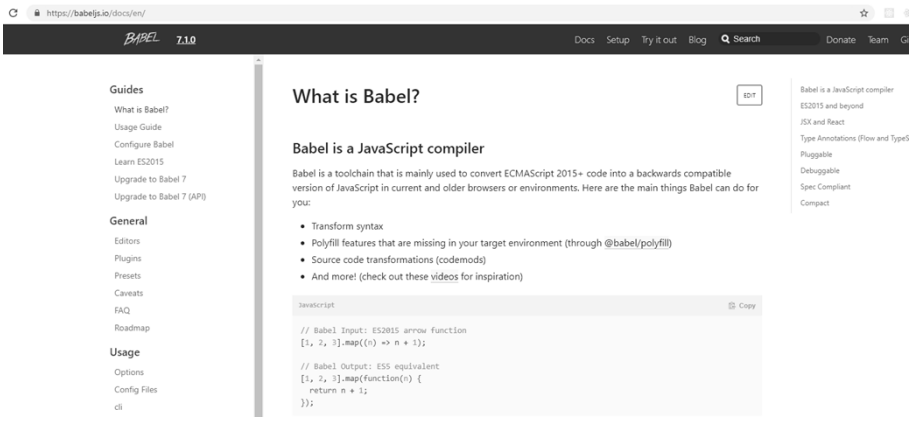
```
npm install - - save-dev babel babel-cli
```
ili kako u mnogim turijalima se može naći

```
npm install –g babel-cli
```

Sa globalnom instalacijom možemo i zadavati komande iz konzole kako bi smo ga pokrenuli. Jednostavno unošenjem komande u terminalu:

```
babel neki_fail.js
```

Ali ako pogledamo njegov oficelni veb sajt na https://babeljs.io/docs/en/



videćemo u dokumentaciji da on može da kompajlira ne samo pojedinacni fail već i citave direktorijume. Ako primenimo njegove prekidače - - out ili o, za failove

babel neki_fail.js – out kompajlirani_fail.js

ili kod primene za dorektorijume – - - out-dir -o-dir ili –o-d

babel neki_direktorijum -o-dir nekiDir

Ono što je još potrebno osim njegovog uključivanja u webpack.config.js fail potrebno je i napraviti njegov fail koji se zove .babelrc ali i instalisati i ostale njegove delove
npm i babel-core babel-loader babel-preset-es2015 babel-preset-react -S

Fail .babelrc mora biti u root folderu react aplikacije. Kada je u pitanju SPA onda u projektnom folderu.

.babelrc file

```
{
  "presets" : ["es2015", "react"]
}
```

Sam babel može se pokrenuti i kada se poziv njemu doda kao deo skript objekta u failu package.json

```
{
   "scripts": {
     "start": "node server.js",
    "script-babel": "babel-node script.js"
   }
 }
```

Ako svoju aplikaciju radite sve ručno biće vam potrebnije veće znanje i iskustvo, zato je najbolje u početku primenjivati takozvano automacko izradjivanje react aplikacija gde je sve postvaljeno na optimum.

Fail .babelrc

Ovaj fail sadrži podatke o instalisanim dodacima **plugins** ili podešavanjima **presets** (grupa pluginova), koji babelu daju instrukcije kako i šta da radi. Tako da osnovni kod ovog faila je ovakav

```
{
  "presets": [ u ovom delu se kao i u donjem delu dodati delovi pišu zatvoreni
                navodnicima i međusobno odvojeni zarezom ],
  "plugins": []
}
```

Tako da uneti dodaci i podešavanja podsećaju na pisanje niza podataka. U slučaju da instališemo:

```
npm install --save-dev babel-preset-es2015
npm install --save-dev babel-preset-react
npm install --save-dev babel-plugin-transform-runtime
npm install --save babel-runtime
```

```
  {
    "presets": [
     "es2015",
      "react"
    ],
    "plugins": [
      "transform-runtime",
      "transform-es2015-classes"
    ]
  }
```

Pa u zavisnosti od potreba aplikacije ovaj fail može imati i ovakav izgled koda

```
  {
    "presets": ["es2015", "react"],
    "plugins": [],
     "env": {
     "development": {
     "plugins": [...]
     },
     "production": {
```

```
    "plugins": [...]
  }
 }
}
```

Prema svemu ako podešavate sami vezano za upotrebu babel-a morate instalisati više njegovih delova ovako:

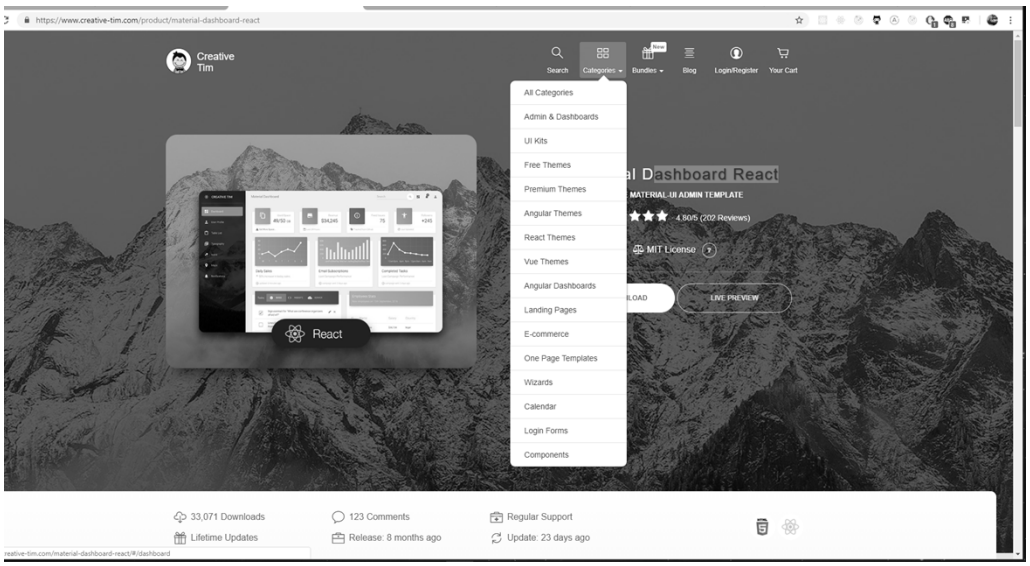
npm i @babel/core babel-loader @babel/preset-env @babel/preset-react --save-dev

U njegovom konfiguracionom failu .babelrc napisati tada

```
{
  "presets": ["@babel/preset-env", "@babel/preset-react"]
}
```

Izrada react aplikacije sa webpackom i babelom

Do sada smo imali izrade komponenata, objašenjenja automastkog izrađivanja react aplikacija, gde nismo imali potrebe da išta podešavamo, jer je za nas to radila skripta koja se pokretala globalnom instalcijom za izradu react aplikacija. U ovom delu biće izrađena aplikacija potpuno ručnim radom, gde ću prikazati povezivanje react aplikacije sa webpackom i babelom, ako vam je potrebno možete dodati i materijal za dizajniranje, kog možemo preuzeti na web adresi https://www.creative-tim.com/product/material-dashboard-react .



Na ovom prostoru ima dosta toga što je već podešeno i spremno za upotrebu, tako da će vam biti lako da preuzeti kod i ostalo prilagiđavate vašim potrebama.

Izrada projekt foldera i package.json faila

Izrada projekt foldera može biti na više načina u zavisnosti od toga koji OS imate, da li upotrebljavate ozbiljniji editor koda ili jednostavniji ili upotrebljavate shell, terminal i sl. Prepostavimo da upotrebljavate vindovsov terminal ili konzolu upotrebiću najjednostavniji način. U njemu ili njoj upisete :

mkdir react-webpack-babel-primer
cd react-webpack-babel-primer

Sledeći korak je izrada package.json faila. Za ovaj korak upotrebićemo pomenuto -y u komandnom redu:

npm init –y

Na ovaj način smo automacki dobili osnovni kod package.json faila, koji možemo da kasnije dopunjavamo po potrebi drugim delovima koji su nam potrebni prilikom izrade react aplikacija. U ovom koraku izradićemo i src folder u projektnom folderu

mkdir src

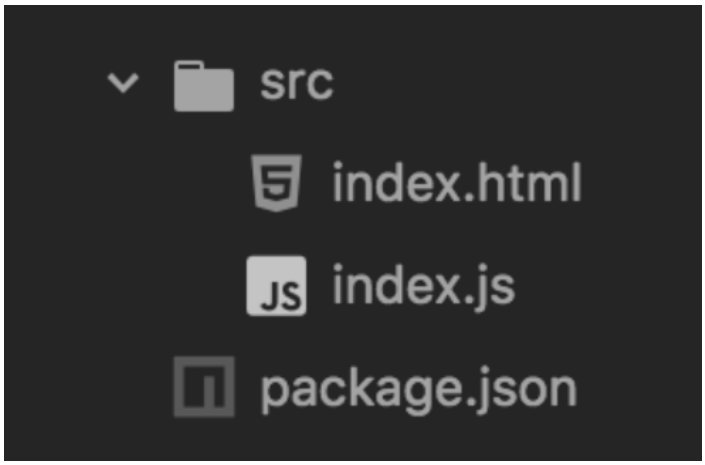
i iz editora koda napraviti u njemu za sada dva faila index.html i index.js. Gde će kod u index.html failu biti ovakav:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport"
          content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <meta name="theme-color" content="#000000">
    <title>React Webpack Bable primer</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

Dok u index.js za sada postavićemo samo ovo, zatvorenog u tagu script, pošto je ovo java skript:

```
(function () {
  console.log("Dobar dan");
}());
```

I dobili smo projekt folder sa ovim sadržajem:



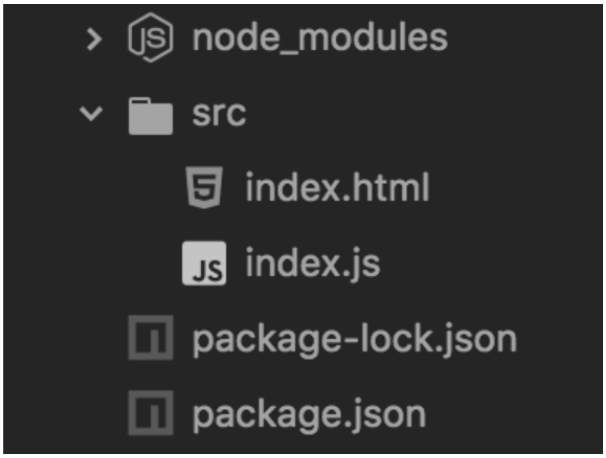
Dodavanje webpacka u projekt

Obzirom da smo završili deo posla u predhodnom koraku u ovom delu dodajemo Webpack da pratećim delovima njegovog paketa. Da vas posetim na sadržaj package.json fail. U njemu postoje više celina. Obzirom da ćemo u ovom slučaju upotrebljavati webpack i njegove pakete u razvojnom delu mi ćemo ih dodati u delu za razvoj, tačnije u devDepencies-u. Ono što ćemo instalisati da bi webpack radio svoj posao je pored njega webpack-cli (njegov komandni liniski interpreter) i webpack-dev-server na sledeći način:

npm i - - save-dev webpack webpack-cli webpack-dev-server
Na ovaj način smo omogućili da možemo pristupati webpacku iz komandne linije i dobili server za razvoj aplikacija. Ako pogledamo iz editora koda sadržaj package.json fail videćemo da je prilikom instalacije u njemu promenjen sadržaj. Tačnije dobili smo nekoliko redova:

```
"devDependencies": {
  "webpack": "^4.19.0",
  "webpack-cli": "^3.1.0",
  "webpack-dev-server": "^3.1.8"
}
```

Ali ovim smo dobili i promenu u projektnom folderu



Upotreba webpacka zasniva se na upotrebi njegovog konfiguracionog faila u kome podešavamo sve potrebne parametere, plaginove i ostalo što nam je bitno u aplikaciji. Izrada configracionog faila webpacka obavlja se u editoru koda otvaranjem novog faila sa imenom webpack.config.js . Ovaj fail može se napraviti jedan, ali i mogu se napraviti više njih u zavisnosi od potrebe aplikacije, tako da iz jednog se mogu pozvati ostali prema potrebi. U samom početku ovog faila webpack zahteva i upotrebu dodatnog paketa path koji se mora instalisati pre upotrebe:
npm i - -save-dev path

Pošto ćemo u ovom primeru upotrebljavati i plaginove za sada ćemo instalisati html-webpack-plugin:

npm i - -save-dev html-webpack-plugin

Ovaj plagin praktično generiše html fail za naše potrebe, čemu je i namenjen. Tako da primenom ovog plagina se index.html dinamički generiše sam. Za više o podešavanjima ovog webpack plagina na sajtu:
https://github.com/jantimon/html-webpack-plugin
ili oficijelnom vebsajtu https://webpack.js.org/plugins/html-webpack-plugin/
Obzirom da ovaj plagin daje kod index.html faila sa nekim svojim osnovnim podešavanjima sam izgled tog koda je ovaj.

```
 <!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Webpack App</title>
  </head>
  <body>
    <script src="index_bundle.js"></script>
  </body>
```

```
</html>
```

Što daje webpack.config.js fail koji sadrži ovaj kod

```
const HtmlWebpackPlugin = require('html-webpack-plugin')

module.exports = {
  entry: 'index.js',
  output: {
    path: __dirname + '/dist',
    filename: 'index_bundle.js'
  },
  plugins: [
    new HtmlWebpackPlugin()
  ]
}
```

Obzirom da je nam potrebno da unosimo promene u praktično tako dobijen templejt index.html fail, znači da je potrebno dodavati te promene u konfiguracionom failu.

Izrada webpack.config.js faila

Ovaj fail je praktično srce svega u ovakvim slučajevima, jer on omogućava isključivanje, uključivanje potrebnih loadera, plaginova, izradu foldera, i dr. U našem slučaju on ima sledeći kod:

```
const path = require('path');
const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
  entry: path.join(__dirname,'src','index.js'),
  output: {
    path: path.join(__dirname,'build'),
    filename: 'index.bundle.js'
  },
  mode: process.env.NODE_ENV || 'development',
  resolve: {
    modules: [path.resolve(__dirname, 'src'), 'node_modules']
  },
  devServer: {
    contentBase: path.join(__dirname,'src')
  },
  plugins: [
    new HtmlWebpackPlugin({
```

```
    template: path.join(__dirname,'src','index.html')
   })
 ]
};
```

Gde smo u prvom redu uključili path a posle njega i html-webpack-plugin. U toku rada videćete da se u nekim slučajevima u ovo delu webpack.config.js faila dodaje i sam webpack na ovaj način:

```
const webpack = require(' webpack ');
```

Kao što vidite ovaj fail ima dosta svojih delova koji su smešteni kao moduli koji se eksportuju. U prvom delu module.export imamo ulazne failove ili fail:

entry gde smo pripojili direktorijum koji se zove src i uzeli iz njega fail index.js

```
entry: path.join(__dirname,'src','index.js'),
```

Ono što je webpack uzeo na ovaj način on treba to i da negde pošalje kako bi smo to mogli da vidimo na ekranu. Taj deo je izlazni deo, koji je praktično jedan objekt:

```
 output: {
   path: path.join(__dirname,'build'), // dodavanje foldera za smestanje produkcione
                                          aplikacije
   filename: 'index.bundle.js'// fail koji se povezuje za index.html fail
                                 omogućavajući da se naša aplikacija vidi u
                                 brovseru, jer je on sve ono što smo napravili u reactu
 },
```

Ovaj deo, ako nepostoji direktorijum bulid, određuje da webpack ga pravi i u njemu izradi fail 'index.bundle.js'. Sledeći deo je mode. U ovom slučaju smo podesili node-env (environment) varijabile, gde smo postavili uslove ili će biti u upotrebi: process.env.NODE_ENV ili 'development'.

```
 mode: process.env.NODE_ENV || 'development'
```

Deo resolve omogućava uvoz bilo čega upotrebljavajući umesto apsolutnih putanja upotrebu relativnih putanja(relative path). Pa je u našem primeru omogućeno da se upotrebi - uveze bilo šta iz src i node-modules foldera po potrebama aplikacije i webpacka.

```
 resolve: {
   modules: [path.resolve(__dirname, 'src'), 'node_modules']
 },
```

Obzirom da je react i njegovo okruženje serverskog tipa potrebno je za njegov prikaz upotrebiti i server, što je omogućeno kodom koji ga poziva:

```
 devServer: {
   contentBase: path.join(__dirname,'src')
 },
```

Naravno ovo je samo za vreme dok se bavimo samim razvojem i izradom react aplikacije. Tako da samo ovim uključili u rad i webpack-dev-server, koji će sve što se nalazi u src folderu prikazati na ekranu brovzera.
Deo za dodatke - plaginove omogućava njihovo konfigurisanje i prijavu i primenu. U ovom trenutku smo samo primenili html-webpack-plugin, kojm smo praktično izradili templejt, drugim rečima dinamički izrađujemo index.html fail i gde smo naredili serveru da unese u taj templejt link ka failu index.bundle.js.

Pokretanje i upotreba konfiguracionog faila

Da bi smo mogli da pokrenemo u tokom izrade react aplikacije bilo šta potrebno je da to i omogućimo da node razume i uradi to za nas. Tako nešto se radi unošenjem promena u delu za skriptove package.json faila. Koje kasnije kada unesemo naziv ključa node.js pokrene. Pa je za upotrebu webpack.config.js fail i webpack-dev-server potrebno napisati u ovom delu sledeće:

```
"webpack": "webpack",
"start": "webpack-dev-server --open"
```

Pa je kod package.json fila dobio ovakav sadržaj:

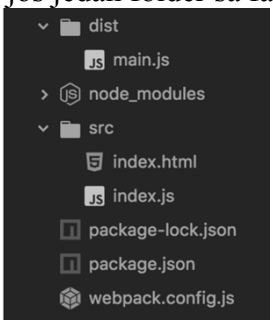
```
{
 "name": "react-webpack-babel ",
 "version": "1.0.0",
 "description": "Upotreba u React apliakciji  Webpack i Babel",
 "main": "index.js",
 "scripts": {
   "test": "echo \"Error: no test specified\" && exit 1",
   "webpack": "webpack",
   "start": "webpack-dev-server --open"
 },

 "keywords": [
   "react",
   "webpack",
   "babel",
   "creative-tim",
   "material-design"
```

```
  ],
  "author": "autor",
  "license": "MIT",

  "homepage": "https://nekivebsajt.org",
  "devDependencies": {
    "html-webpack-plugin": "3.2.0",
    "path": "0.12.7",
    "webpack": "4.19.0",
    "webpack-cli": "3.1.0",
    "webpack-dev-server": "3.1.8"
  }
}
```

Ako pokrenemo webpack sa npm run webpack u terminalu dobijamo :

```
Hash: b9e0f6b6745a84f65f8e
Version: webpack 4.18.0
Time: 90ms
Built at: 2018-09-12 10:03:15
  Asset       Size  Chunks             Chunk Names
main.js  930 bytes       0  [emitted]  main
Entrypoint main = main.js
[0] ./src/index.js 94 bytes {0} [built]

WARNING in configuration
The 'mode' option has not been set, webpack will fallback to 'production' for this value. Set 'mode'
option to 'development' or 'production' to enable defaults for each environment.
You can also set it to 'none' to disable any default behavior. Learn more: https://webpack.js.org/cor
cepts/mode/
```

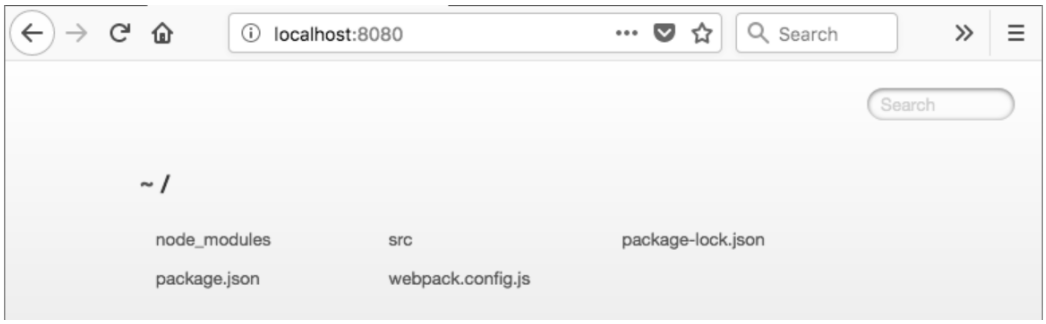
dok u projekt folderu dobijamo još jedan folder sa failovima



Ako upotrebimo npm start

```
ℹ ｢wds｣: Project is running at http://localhost:8080/
ℹ ｢wds｣: webpack output is served from /
ℹ ｢wdm｣: wait until bundle finished: /
⚠ ｢wdm｣: Hash: 7f898a38aca445293ff1
Version: webpack 4.18.0
Time: 3501ms
Built at: 2018-09-12 10:18:15
  Asset     Size  Chunks             Chunk Names
main.js  139 KiB       0  [emitted]  main
Entrypoint main = main.js
 [2] multi (webpack)-dev-server/client?http://localhost:8080 ./src 40 bytes {0} [built]
 [3] (webpack)-dev-server/client?http://localhost:8080 7.78 KiB {0} [built]
 [4] ./node_modules/url/url.js 22.8 KiB {0} [built]
 [7] ./node_modules/url/util.js 314 bytes {0} [built]
 [8] ./node_modules/querystring-es3/index.js 127 bytes {0} [built]
[11] (webpack)-dev-server/node_modules/strip-ansi/index.js 161 bytes {0} [built]
[12] (webpack)-dev-server/node_modules/ansi-regex/index.js 135 bytes {0} [built]
[13] ./node_modules/loglevel/lib/loglevel.js 7.68 KiB {0} [built]
[14] (webpack)-dev-server/client/socket.js 1.05 KiB {0} [built]
[15] ./node_modules/sockjs-client/dist/sockjs.js 177 KiB {0} [built]
[16] (webpack)-dev-server/client/overlay.js 3.58 KiB {0} [built]
[18] ./node_modules/html-entities/index.js 231 bytes {0} [built]
[21] (webpack)/hot sync nonrecursive ^\.\/log$ 170 bytes {0} [built]
[23] (webpack)/hot/emitter.js 75 bytes {0} [built]
[25] ./src/index.js 94 bytes {0} [built]
    + 11 hidden modules

WARNING in configuration
The 'mode' option has not been set, webpack will fallback to 'production' for this value. Set 'mode'
option to 'development' or 'production' to enable defaults for each environment.
You can also set it to 'none' to disable any default behavior. Learn more: https://webpack.js.org/con
cepts/mode/
ℹ ｢wdm｣: Compiled with warnings.
```

Gde smo dobili obaveštenje da se je projekt pokrenut na localhostu 8080 i šta je on uradio. Dok u brovzeru samo ovo



Razlog prikaza ovog sadržaja u brovzeru je to što još nismo intalisali react i react dom brz kojih se sadržaj koda reacta nemogu prikazivati u brovzeru. Ovako napisan kod u konfiguracionom failu webpacka dodeljuje titl failu admin.html

```
plugins: [
   new HtmlWebpackPlugin({
     title: 'My App',
     filename: 'assets/admin.html'
   })
 ]
```

```
}
```

Dok ovaj primer koda uključuje generisanje više html failova

```
plugins: [
   new HtmlWebpackPlugin(), // Generiše osnovni index.html
   new HtmlWebpackPlugin({  // ovim se generiše i test.html
     filename: 'test.html',
     template: 'src/assets/test.html'
   })
 ]
```

Više o html-webpack-plugin na https://github.com/jantimon/html-webpack-plugin

Instalacija Reacta i React-doma

Obzirom da je react i react dom potreba da budu uvek tamo gde je i aplikacija onda njih instališemo kao i druge zavisne pakete upotrebom:
npm i - - save react react-dom

Kao što je dva znaka minus bilo ispred save-dev određivalo da se to smesti kao deo devdependecies (deo za razvoj) , tako bez nastavka –dev se smešta u deo dependencies (stalno potreban ili zavistan) postavljaju ključevi i njihove vrednosti u deo zavisnih-potrebnih delova package.json faila. Pa je u ovom slučaju potrebno malo promeniti kod index.js faila:

```
import React from "react";
import ReactDOM from "react-dom";

let Čestitka = () => {
  return <h1>Postao si srećan dobitnik!</h1>
}

ReactDOM.render(
  <Čestitka/>,
  document.getElementById("root")
);
```

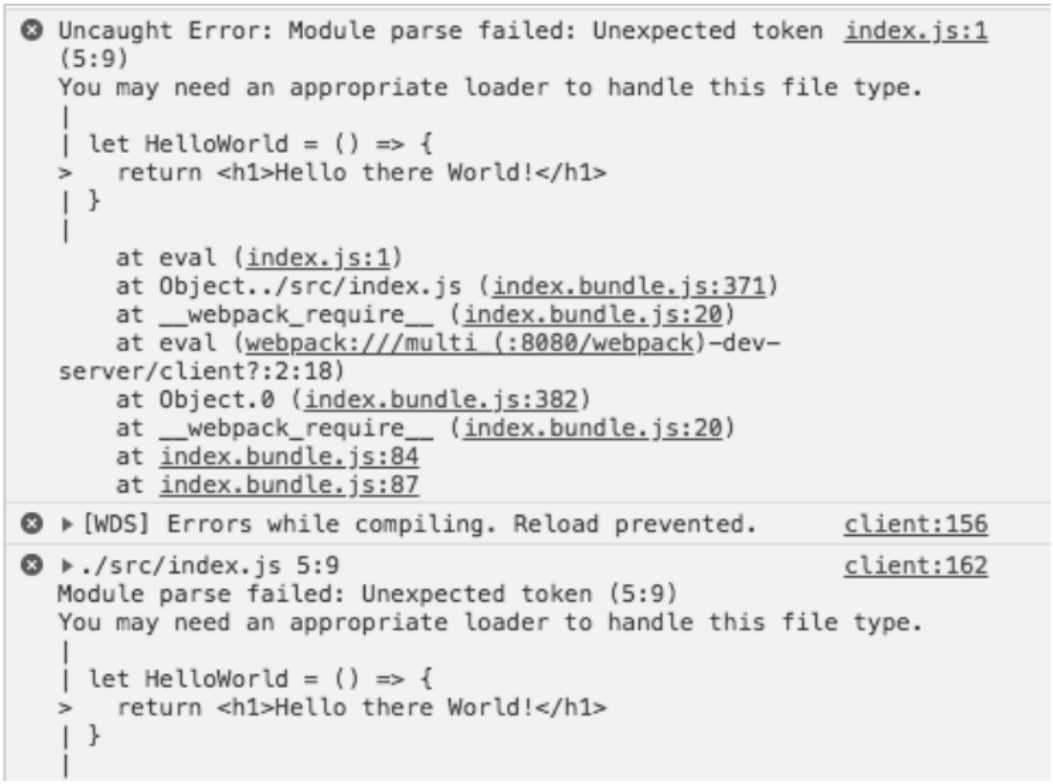
React-dom je paket koji omogućava povezivanje virtualnog dom-a u kome se nalazi naša aplikacija, sa stvarnim putem svog drugog parametra:

```
document.getElementById("root")
```

Gde je root id div taga u html failu dobijene produkcione aplikacije.Standardno je to index.html fail.

Uključivanje Babela

Ovaj kod gore index.js faila brovzeri nemogu da razumeju jer je to kod java skripta višeg nivoa, te je potrebno ga prevesti na onaj razumljivi kod. U te svrhe se upotrebljava babel. Pa brovzer prjavljuje grešku



pošto babel još nije instalisan u aplikaciju. Babel se isto instališe u deo package.json faila kao potreban deo samo za razvoj na ovaj način:

npm install --save-dev @babel/core @babel/node @babel/preset-env @babel/preset-react babel-loader

Delovi paketa babel

Kao i webpack tako i babel ima svoje delove koji mu omogućavaju neometan rad, to jest prevođenje i ostalo iz čega se i na kraju pravi index-build.js fail. Dodavanje znaka @ ispred imena paketa zadaje upravljaču paketa za instalaciju da instališe najvišu verziju paketa.

babel/core prevodi iz ES6 i drugih u ES5 verziju java skripta koja je razumljiva svim brozerima

babel/node omogućava uvoz naših plaginova i paketa unutar webpack.config-a umesto da ih tražimo na nakom drugom mestu

babel/preset-env ovim se utvrđuje koje će transformacije ili dodatke ili polifile upotrebljavati na osnovu matrice pretraživača koji želite da podržite, čime se obezbeđuje funkcionalnost kod starijih pretraživača.

babel/preset-react namena mu je da kompajluje React kod ES6 u ES5 kod

babel-loader ovaj deo paketa babel je moglo bi se reći da je pomoćnik Webpacku prilikom transformacija vaših java skriptova, na primer transformacija ulaznih naredbi u one koje se zahtevaju.

Konfiguracija babela može biti direktno u webpack.config.js failu ili u posebnom failu koji se zove .babelrc čime se omgućava babel-loaderu da zna kako da kompajluje kod aplikacije. Jedan primer koda ovog faila je ispod:

```
{
  "presets": [
    "@babel/env",
    "@babel/react"
  ]
}
```

Instalacija drugih loadera

Loaderi ili oni koji omogućavaju da se nešto učita što je specifičnog tipa podataka, su nam nepohodni kada na primer učitavamo neku sliku, css stil i slično za potrebe naše aplikacije ili na server. Naravno kao i sve drugo nisu obavezni ali možda vam zatrebaju. Te ću objasniti nekoliko koji se upotrebljavaju češće.

Loaderi koji se odnose na stilove

Ovi loaderi obuhvataju stari css kod i noviji sass. Instalacija

npm install --save-dev style-loader css-loader sass-loader node-sass

style-loader
—Namena za dodavanje opisa izgleda DOMu (primenjuje se ubacivanjem <style> taga unitar HTML fila)

css-loader
—dozvoljava nam da uvozimo naše CSS filove u naš projekt

sass-loader
—radi isti posao kao predhodni samo za SCSS filove

node-sass
—kompajlira SCSS filove u normalne CSS filove

Ako posle ovog pogleamo naš package.json fail u delu dodataka za razvoj on izgleda ovako:

```
 "devDependencies": {
   "@babel/core": "7.0.1",
   "@babel/node": "7.0.0",
   "@babel/preset-env": "7.0.0",
   "@babel/preset-react": "7.0.0",
   "babel-loader": "8.0.2",
   "css-loader": "1.0.0",
   "html-webpack-plugin": "3.2.0",
   "node-sass": "4.9.3",
   "path": "0.12.7",
   "sass-loader": "7.1.0",
   "style-loader": "0.23.0",
   "webpack": "4.19.0",
   "webpack-cli": "3.1.0",
   "webpack-dev-server": "3.1.8"
 }
```

Napomena: neki preporučuju brisanje znakova carets (^) u package.json

Ostali loaderi

Od ostalih mogu napomenuti za učitavanje failova – file-loader

npm install --save-dev file-loader

Ako primenimo pomenute sve dosadašnje delove koji smo instalisali webpack.config.js fail bi imao ovaj kod

```
// stariji način pisanja
// const path = require('path');
// const HtmlWebpackPlugin = require('html-webpack-plugin');

// noviji naččin pisanja
import path from 'path';

import HtmlWebpackPlugin from 'html-webpack-plugin';

module.exports = {
  entry: path.join(__dirname,'src','index.js'),
  output: {
    path: path.join(__dirname,'build'),
```

```
  filename: 'index.bundle.js'
 },
 mode: process.env.NODE_ENV || 'development',
 resolve: {
  modules: [path.resolve(__dirname, 'src'), 'node_modules']
 },
 devServer: {
  contentBase: path.join(__dirname,'src')
 },
 module: {
  rules: [
   {
    // dole je napisan kod koji omogućava kompajlovanje bilo kog  React koda,
    // ES6 i slične u normalni ES5
    test: /\.(js|jsx)$/,

    // ovim se isključuje kompajlovanje node_modules
    exclude: /node_modules/,
    use: ['babel-loader']
   },
   {
    test: /\.(css|scss)$/,
    use: [
     "style-loader", // izrađuje style nod u JS string
     "css-loader", // prevodi CSS u CommonJS
     "sass-loader" // kompajlira Sass u CSS, upotrebljva Node Sass kao osnovni
]
   },
   {
    test: /\.(jpg|jpeg|png|gif|mp3|svg)$/, //učitava slike ovih formata
    loaders: ['file-loader']
   }
  ]
 },
 plugins: [
  new HtmlWebpackPlugin({
   template: path.join(__dirname,'src','index.html')
  })
 ]
};
```

Na kraju potrebno je uneti i izmene u package.json kako bi sve radilo kako treba:

```
"scripts": {
   "test": "echo \"Error: no test specified\" && exit 1",
   "webpack": "babel-node ./node_modules/webpack/bin/webpack",
   "start":
"babel-node ./node_modules/webpack-dev-server/bin/webpack-dev-server --open"
  }
```

Finalizacija izrade react aplikacije

U osnovi SPA tip aplikacije iz razvojnog u finalni oblik se pretvara upotrebom

npm run build

iza čega se dobija folder build sa ovim sastavom

Iza čega se pojavljuje upit da li želite da vidite svoju produkcionu verziju sa unosom u terminal

serve bulid

Napomena: pre pokretanja serve build biće vam prikazano u terminalu da morate da ga instališete globalno upotrebom prekidača –g, što će te u ovom slučaju biti obavešteni na istom prikazu kako da postupite.

Iza build možete dodati na kom će portu biti prikazana aplikacija upotrebom –p broj porta. Na primer:
serve build –p80 za prikaz na portu http://localhost:80
Ali ako upotrebljavate dodatne module potrebno je podesiti i webpack.config.js fail kako je to potrebno zbog moguće upotrebe dodatnih react paketa, kako bi i oni ispravno radili.

Mogući problemi

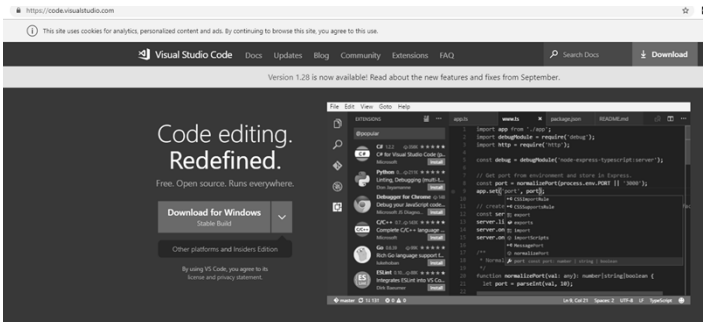
Najćešći problem koji može da vas zbuni je kada ste još u ovo fazi učenja napravili krajnju aplikaciju koja je spremna da se postavi na udaljeni server i vi je postavite na svoj wamp ili xamp lokalni server a na ekranu dobjate delimične sadržaje vaših

veb stranica, što se može odnositi u najvećem broju na slike. Tačnije slike koje su formata base64 normalno se vide, dok ostalih formata ne. No nemojte da vas to zbuni. Jer u razvojnom delu je već sve radilo, znači da je vaša aplikacija ispravna, ali je problem je u vašem podešavanju za apche server koji se upotrebljava u pomenutim aplikacijama gde je on sastavni deo.
Da se nebi mučili previše oko tih podešavanja napravite jedan virtualni host u wamp ili xamp aplikaciji, tamo prekopirajte sadržaj foldera buld iz vašeg projektnog foldera, onda pokrenite wamp ili xamp, i na njegovim lokal hostu kliknite na naziv vašeg virtualnog hosta i aplikacija će proraditi bez ikavih problema.

Editori koda

Najjendostavniji je NotePad ++ ali evo jednog koji je isto tako besplatan ali je dosta dobar za rad. Proizvod je firme MicroSoft zove se VC editor. O njemu na ovom linku, gde postoji i link za preuzimanje:

https://code.visualstudio.com/



Gde u delu

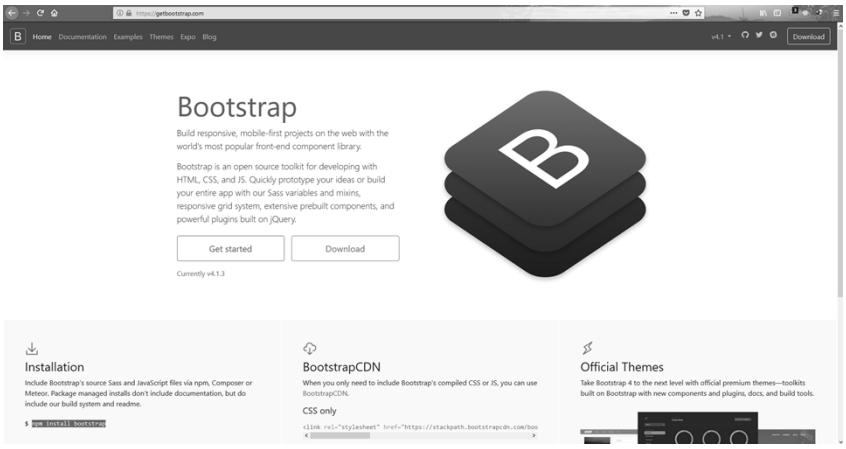
https://code.visualstudio.com/docs/nodejs/reactjs-tutorial
imate i ovaj turorijal, kao i podešavanja editora za rad sa node react okruženjem, čime možete dosta lakše pisati svoje buduće react aplikacije. Editor koda je jednostavan za rad, mada a na istom sajtu postoje i druge online pomoći za brže učenje rada i podešavanja editora. Problem je sa njime ka i drugim prozvodima ove kompanije potrošnja radne memorije.

REACT-BOOTSTRAP

React- Bootstrap

Bootstrap je frame work gde praktično postoje tri faila kojima je opisano mnogo elemenata i radnji, putem css, java skripta i posebne vrste java skripta pod nazivom jquery. Ono što je još dobro regulisano u samom Bootstrapu je podjednako prestavljanje sadržaja ekrana na bilo kojoj veličini ekranskog prikaza (responsive view). O njemu sam napisao u knjizi " Osnove Butstrapa –Bootstrap" , koju možete i pogledati na sajtu akademije na ovom linku : https://www.academia.edu/36443823/%D0%9C%D0%B8%D0%BE%D0%B4%D1%80%D0%B0%D0%B3_%D0%A2%D1%80%D0%B0%D1%98%D0%B0%D0%BD%D0%BE%D0%B2%D0%B8%D1%9B_%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D0%B5_%D0%B1%D1%83%D1%82%D1%81%D1%82%D1%80%D0%B0%D0%BF%D0%B0_Bootstrap

Autori Bootstrapa su uneli novi prostup u rešavanju problema prilagodjavanja sadržaja veličini ekrana gde se on prikazuje tako što su uzeli umesto fiksnih vrednosti procentualne. Posmatrajući događanja izvršili su i reorganizaciju širine ekranskog prikaza na 12 kolona ne bitno o kojoj se širini ekrana radi, s time što je broj redova ne ograničen. Tako da ta podela podseća na vituelnu tabelu koja je potpuno prilagodljiva potrebama i koja usput prilagođava i svoj sadržaj sebi. Pri čemu treba obrati pažnju na parametre bordera, padinga i margina, jer u tom slučaju ako se pokaže da je veća vrednost od moguće sadržaj iz gornjeg reda moguće da bude prebačen u donji red.
O samom Bootstrapu na njegovom oficijelnom sajtu više na ovom linku: https://getbootstrap.com/



Ono što je po mnogima u tutorijalima je da je poželjno izbegavati instaliranje Jquery-a u react aplikaciju, ali to se može izbeći na drugi način, koji ću kasnije opisati.

Način upotrebe Bootstrapa u react aplikacijama

Upotreba Bootstrapa u reactu se može uraditi putem instalacije sa node paket upravljačem (npm), ali time se instališe samo css, dok bootstrap.js i jquery.js je potrebno upotrebiti CDN linkove u public/index.html failu. Drugi način je da se preuzme razvojna verzija bootstrapa i onda iz faila dist prekopirati failove u odgovarajući u folder public u react aplikaciji i onda ih linkovati u public/index.html fail. Efekt rada je skoro istovetan, sa jednom razlikom što prilikom upotrebe CDN linkova prema Bootstrap failovima svaki put kada to treba aplikacija te failove poziva sa njihovog sajta. Što i nije baš dobro. Zato je bolje preuzeti bootstrap i primeniti drugi način, jer je onda sa vašom aplikacijiom sve na istom serveru. Za isti posao možete preuzeti i dodatni paket reactstrap ili react-bootstrap kog instališete direktno u react aplikaciju upotrebom koda

npm install --save reactstrap // umesto ovog koda može i ovaj npm i S reactstrap

Bliže infromacije o reactstrapu na vebsajtu

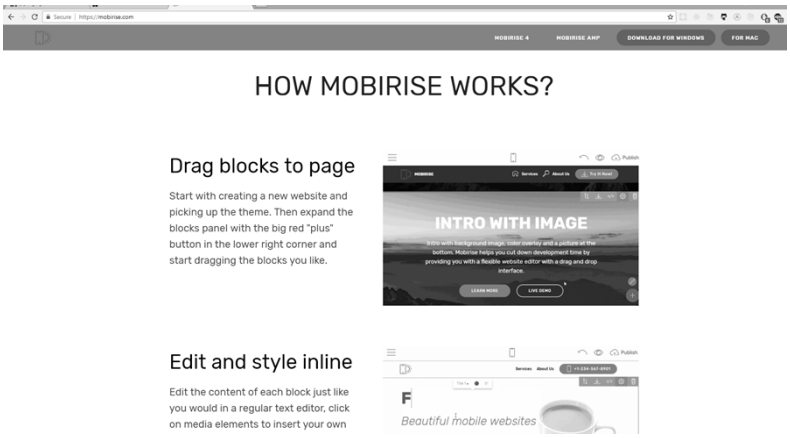
https://reactstrap.github.io/

Ali, dok ne ovladate malo bolje radom u reactu i njegovim dodatnim paketima bolje je da uporebite čist Bootstrap kod. Čime ćemo se pozabaviti u daljem radu.

Šta to nudi Bootstrap

Bootstrap kao što je napomenuto praktično ima tri obavezna faila koje omogućuju svojim sadržajem već unapred definisane oblike, boje i druge osobine koje se odnose na sam izgled delova veb stranica. Pored toga upotrebom java skripta u preostala dva faila omogućava rad menija, slajdera, i sličnih funkcija radnji koje susrećemo prilikom izrade veb sajtova. Sam bootstrap je prvi put upotrebljen kod izrade Twitera, gde su i bili zapošljeni njegovi tvorci.

Ono što je još bitno znati, da je nebitno na to unaped određeno i postavljeno, postoji mogućnost da i vi sami to prilagodjavate sebi i svojim potrebama kombinujući postavljeno sa vašim potrebama i nivom znanja.
Sam bootstrap nudi dosta toga za izradu elemenata veb stranica, čime im daje svoju posebnu dinamičnost, pri čemu nam ostavlja veliku slobodu da to što je osnovno podešeno da možemo da menjamo i prilagođavamo našim potrebama. Uz to on je idelan za početne radove i učenja u kombinaciji sa reactom.
Pa ću objasniti jedan po jedan deo boostrapa kroz upotrebu primera sa reactom, kako bi ste stekli uporednu sliku primene u html-u i react-u. Princip rada je zadržan i u reactovom dodatom paketu reactstrap, tako da uz ovaj deo knjige moći ćete da razumete i njegov rad. Ono što je prvo omogućeno je podjednaki prikaz sadržaja veb stranica na bilo kom uređaju, upotrebom responsive tehnike. Pored toga ponuđen je veliki broj već gotovih elemenata u više varijanti pri čemu je zadržano i pravo da korisnik sam podešava kako izgled gotovih elemenata, tako i java skriptova koji su sa namenom određenih akcija: od skrivanja sadržaja, pa do sklopivih menija, slajdera, toglera i dr. U nekim situacijama je omogućen i feijd elemenata. U tome nalazi primenu i jquery posebna vrsta java skripta, koja omogućava veliki broj efekata i akcija uz malo pisanja koda.
Primetno je da su mnogi krenuli da prave svoje varijante Bootstrapa, obzirom na njegovu lakoću primene i rada. Tako je jedan od značajnijih u Mobrise, što možete videti na internetu na https://mobirise.com/



Koji je ponudio za rad sa bootstrapom njegovu verziju zajedno sa editorom koda, kog možete preuzeti sa njegovog oficijelnog sajta. Na istom prostoru prilikom preuzimanja editora dobijate i tutorijale.
Sam rad Bootstrapa se zasniva na radu java scripta koji poziva određene ili one koje mi to odredimo css klase ako ih dodaje html tagovima, ili upotrebi njhove nazive za izvršenje sopstvenih radnji nad html elementima koje u svom sadržaju koda sadrže naziv neke klase, id , naziv određenog taga i sl.
Pojedini elementi koji su izgradjeni u bootstrapu podsećaju na školsku tablu (Paneli), kojima možete dodati bilo koji element stranice. Nebitno što su to pod elementi, oni mogu da sve sadrže što i sama stranica, pa čak i druge panenle. Pored

toga, postoji još jedan element koji je identičan po osobinama panelu, samo on se pojavljuje privremeno pod nazivom Modal.
Da malo pojasnim reč privremno. Praktično i on se pojavljuje događajem klik, ali kada se klikne mišem van njega on nestaje. Zove se Modal, pogodan je za mnoge forme, logovanje ili ako ste pokrenuli neku stranicu na racunaru i kliknete na neku sliku onda se poveže za događaj pojava modala i onda u njemu podesite da se slika vidi u većoj rezoluciji i dodate još opisa, tekstova i sl.

Napomena: Zbog veličine ekrana na mobilim uređajima u css-u potrebno je isključiti njegovu pojavu, kao i menije koji iskaču, ili menije koji se na desktop racunarima pojavljuju sa strane na ekranskom prikazu.

Pored toga, kod organizacije sadržaja ekranskog prikaza Bootstrap nudi još jednu mogućnost Collapse. Ovaj efekt može primeniti od onog popularnog „Kliknite za više „ , pa do primene u menijima, panelima i td. Primenom Collapse možete u Bootstrapu napraviti panele u tri i više niova.

Paneli

Primena panela u react – bootstrap aplikaciji vrlo je jednostavna. Paneli mogu imati kao u donjem kodu staticne podatke.

```
// obavezan deo importa
import React from 'react';

// pocetak kod panela
const Svedocanstva = () => (

{/* sadržaj koda kojim se pozivaju delovi panela. Slike, tekst i slično.*/}
<div>

{/* Postavljenje slike u jednom redu i kolonama u okruženju bootstrap.*/}
   <div className="row">
      <div className="col-xs-12 col-sm-2 col-md-2 col-lg-2">

{/* postavljanje slike .*/}
         <img src= {require('../../img/sved1razr.jpg')}
           className="img-thumbnail img-responsive" alt=".."/>
      </div>
      <div className="col-xs-12 col-sm-2 col-md-2 col-lg-2">
         <img src= {require('../../img/sved2razr.jpg')}
          className="img-thumbnail img-responsive" alt=".." />
      </div>
      <div className="col-xs-12 col-sm-2 col-md-2 col-lg-2">
         <img src= {require('../../img/sved3razr.jpg')}
```

```
            className="img-thumbnail img-responsive" alt=".."/>
        </div>
        <div className="col-xs-12 col-sm-2 col-md-2 col-lg-2">
          <img src= {require('../../img/sved4razr.jpg')}
          className="img-thumbnail img-responsive" alt=".."/>
        </div>
        <div className="col-xs-12 col-sm-4 col-md-4 col-lg-4">

{/* sadržaj koda kojim se postavlja tekst.*/}
          <h5 className="text-justify">
             Ovo su svedočanstva iz srednje škole, koja u svetu se računa kao skola sa
              nekim stepenom. Sta reći na kraju, Imao sam učitelje raznolike, učili su
              me kako su mogli znali umeli i oni sada u meni žive, i to njihovo znanje
              je samnom. Bez obzira na sve razumem ih i dosta njih je živo i eto
              srećemo se i govorimo bez ljutnje ili neceg loseg. Jer svako od njih je iz
              nekih razloga bio takav. Eto odem i sada ko ponekih kada me pozovu
              kuci, kada im se ponešto od aprata pokvari, ili nekom nesto na računaru
              nije jasno ili neće da im radi. Odem pogledam popravim i to je zivot.
             Ponekada se zato nasmejemo uz kafu.
          </h5>
        </div>
    </div>
</div>
);

// kraj koda kojim se vrši eksportovanje koda panela
export default Svedocanstva;
```

Kao što vidite ovaj kod prikazuje Staticni panel. Da bi se dobio dinamički panel, potrebno je u ovom kodu dodati takvu mogućnost. Što se postiže dodavanjem koda

```
this.props.src
```

ili ako se direktno vrše promene u klasi sa :

```
this.state.src  ili
this.setState({ src: this.state.src}) – kada je u pitanju deo taga slike src.
```

Ako se promene odnose na nove vrednosti stanja koje se zadaju van render dela klasa u reactu. Ovi kodovi se umeću u sastav tagova gde je potrebno vršiti promene. Donji kod prikazuje da img src prima osobinu, tj. props i primenjuje se u pod komponenti – child. Tako, da ovaj kod se može primeniti kao u gornjem kodu u obliku konstante, ali se može primeniti i kao klasa koja prima podatke.

```
<div className="col-xs-12 col-sm-2 col-md-2 col-lg-2">
   <img src= {this.props.src}
   className="img-thumbnail img-responsive" alt=".."/>
</div>
```

Upotreba this.state se upotrebljava kod klasa gde se pozivom događaja menjaju vrednosti stanja. Vrednosti stanja se dobijaju iz baze podataka ili json failova koji imaju istu namenu.

```
<div className={this.state.className}>
   <img src= {this.state.src}
   className="img-thumbnail img-responsive" alt=".."/>
</div>
```

Da bi se ovo moglo postići upotrebom this.state potrebno je prvo napraviti početno stanje, kome se onda vrše promene stanja. Početno stanje se postavlja u kontruktoru:

```
constructor(){
  super();
  this.state= {src : require('../../img/slika0.jpg')}
}
```

Dok sama promena se može vršiti događejem onClick ovako:

```
<button= this.setState({ src: require('../../img/slika1.jpg')})> Promena Slike
</button>
```

Ovakav pristup se primenjuje svuda gde su potrebne dinamičke promene. Same promene se mogu vršiti direktno u samoj klasi komponente ili se mogu vršiti iz klase roditeljske komponente u podkomponenti, ili se mogu vršiti promene iz drugih klasa preko roditeljske klase u neku ciljanu klasu. Ali vratimo se panelima. Da bi se omogućilo pozvanje više panela u jednom roditeljskom panelu potrebno je upotrebiti kod ispod:

```
// obavezni import deo
import React from 'react';
//importovanje panela
import Prvipanel from'./panel'
import Drugipanel from'./panel1'
import Trecipanel from'./panel2'
import Cetvrtipanel from'./panel3'

// početak koda sadržaja panela koji poziva druge panele u sebi
```

```
const Paneli = () => (

   <div className="container">
      <div className="tab-body">
         <ul className="nav nav-pills ">
            <li className="active"><a href="#galerija5_1" data-toggle="pill">
                Diplome </a></li>
            <li><a href="#galerija5_2" data-toggle="pill">Svedocanstva</a></li>
            <li><a href="#galerija5_3" data-toggle="pill"> Preporuke</a></li>
            <li><a href="#galerija5_4" data-toggle="pill"> Ostalo</a></li>
         </ul>
         <div className="tab-content">
            <h3 className="text-center naslovh2">
              Diplome,svedocanstva i preporuke</h3>
            <div className="tab-pane fade-in active " id="galerija5_1">
               <Drugipanel/>
            </div>
            <div className="tab-pane fade-in" id="galerija5_2">
                <Prvipanel/>
            </div>
            <div className="tab-pane fade-in " id="galerija5_3">
            <Trecipanel/>
            </div>
            <div className="tab-pane fade-in" id="galerija5_4">
            <Cetvrtipanel/>
            </div>
         </div>
      </div>
   </div>
);

export default Paneli;
```

Kod posle otvorene velike zagrade počinje sa tagom div kojim se uokviruje ceo kod. U ovom tagu može se upotrebiti klasa, ali ne mora, dok početak koda panela u kome se smešta preostali kod panela klasom tab-body.

```
   <div className="container">
      <div className="tab-body">

         u ovom delu se upisuje sadržaj panela

      </div>
   </div>
```

Manji meni se pravi običnom neuređenom listom poređanom u jednom horizontalnom redu

```
<ul className="nav nav-pills ">
    <li className="active"><a href="#galerija5_1" data-toggle="pill">
      Diplome </a></li>
    <li><a href="#galerija5_2" data-toggle="pill">Svedocanstva</a></li>
    <li><a href="#galerija5_3" data-toggle="pill"> Preporuke</a></li>
    <li><a href="#galerija5_4" data-toggle="pill"> Ostalo</a></li>
 </ul>
```

Za pozivanje ili selektovanje panela upotrebljava se standarni način bootstrapa,

uporebom taga
id="naziv-id-panela"

u telu panela, što se poziva upotrebom hiper linka

href="#galerija5_4"

i

data-toggle="pill"

Upotrebom clase active u listitem i iste u tagu div klase tab-pane se određuje prvo aktivno dugme menija i prvi aktivan panel. Ako se ne odredi klasa active u tagu div klase tab-pane neće moći da se pojave paneli.

```
<li className="active"><a href="#galerija5_1" data-toggle="pill"> Diplome
</a></li>
```

```
<div className="tab-content">
        <div className="tab-pane fade in active " id="galerija5_1">
                <Drugipanel />
         </div>
</div>
```

Između div tagova sa klasom tab-content se postavljaju tagovi div sa klasom tab-pane između kojih se pozivaju kodovi panela u obliku koda

```
import Prvipanel from'./panel'; // poziv koda iz faila

< Prvipanel />
{/* Poziv u return delu klase ili u konstanti u skraćenom obliku pisanja*/}
```

Napomena: < Prvipanel /> ovo je skraćeni oblik pisanja za slučaj da se ne dodaju potrebni elementi kojim se bliže opisuje bliže panel.

Promena vrednosti podataka u pod komponenti

Da bi se mogla vršiti promena vrednosti stanja potrebno je imati neki izvor podataka. U reactu se najčešće za manje poslove upotrebljava JSON baza podataka i jednom od više oblika.

1. kao konstanta
2. kao poseban fail
3. kao stvarna baza podataka

U prvom slučaju imamo jednu kostantu koja sadržava više podataka koji se upotrebljavaju kao podaci kojim se menjaju vrednosti stanja u podkomponenti Prvipanel i njen kod izgleda ovako:

```
const src = {
              require('../../img/slika1.jpg',
              require('../../img/slika2.jpg',
              require('../../img/slika3.jpg'
             }
```

fail src.json

```
           {
             src: require('../../img/slika1.jpg',
             src: require('../../img/slika1.jpg',
             src: require('../../img/slika1.jpg'
             }
```

Da bi se mogla jednostavnije načiniti dinamička promena slika klikom na dugme, sliku ili bilo šta se odredi potrebno je dodati još od podataka u JSON kod polje za id i alt. Jer, se id upotrebljava za orjentaciju koja je slika prikazana i koja će biti prikazana, a atribut alt je nepohodan za react, jer u slučaju izostanka sadržaja atributa alt react javlja grešku.

```
const slika = [
              {
               id: 0,
               src:require('../../img/slika1.jpg',
               alt: “ neki tekst”
              },
              {
               id: 1,
```

```
                src:require('../../img/slika1.jpg',
                alt: " neki tekst"
               },
              {
                id: 2,
                src:require('../../img/slika1.jpg',
                alt: " neki tekst"
               },
               {
                id: 3,
                src:require('../../img/slika1.jpg',
                alt: " neki tekst"
               }
];
```

U ovom slučaju potrebno je prvo upotrebiti filter, pa onda map. I filter i map prave od polaznog niza drugi niz podataka gde su dobijeni podaci u novom nizu poređani onako kako je to nama potrebno u aplikaciji. Filter vraća niz objekata ili podataka koji su zadovoljili određen uslov.

```
let OdređeniUslov = slika.filter(function(el){
        return === 1;
});

let noviNiz = OdređeniUslov.map(function(element){
            return(
                    <div key={element.id>
                      <img src={element.src}   alt={element.alt} />
                    </div>
                   );
});
```

U delu za render- return dovoljno je pozvati promenjivu noviNiz ovako

```
            return(
                    <div>
                      { noviNiz }
                    </div>
                   );
```

U delu koda za filter pozvali smo konstantu koja je sadržala podatke o slici i iz toga samo izdvojili samo one podatke koji su jednaki broju jedan. Onda je dobijeni niz podataka upotrebom map pretvoren u pojedinačne članove istog niza koji su postavljeni na potrebna mesta u html tagu img. Obzirom da nam je potrebno da

događajem onClick menjamo u ovom slučaju podatke u html tagu img potrebno je da to i kažemo react aplikaciji. To se može uraditi ovako

<button onClick={()=> this.setState({Promenjiva: this.state.Promenjiva +1 })}> Povećanje</button>

<button onClick={()=> this.setState({Promenjiva: this.state.Promenjiva -1 })}> Smanjenje</button>

U ovom slučaju sam upotrebio strelastu funkciju (Arrow), što je još nešto što stiže sa reactom. Ona omogućava karaći kod. Ovim kodom smo omogućili stalnu promenu vrednosti trenutnog stanja ovom što je nazvano Promenjiva u koracima za broj jedan u smislu povećanja ili smanjenja trenutne vrednosti. Mada, kada nam je potrebno možemo i da direktno odemo na neku ciljanu vrednost ako upotrebimo ovaj kod

<button onClick={()=> this.setState({Promenjiva: 4 })}> Direktan prelaz na podatak broj 4 </button>

Ali, bezuslovni prelazak može se i ovako izvršiti

```
let noviNiz = slika.map((element,i)=>{

   if(i===5) {
      return (
         <div key={ element.id}    >
                <img src={element.src}   alt={element.alt} />
         </div>
      );
   }
});
```

Objašnjenje koda: Ako je i identično broju 5 onda se vraća sve iza return Kompletan kod izgleda ovako

Fail DinamickaGalerija.js

```
import React,{Component} from 'react';
import Manji from './manjiprozor';

class Brojac extends Component {
   constructor() {
      super();
      this.state = {
         claps: 0,
```

```
        nova:[],
        broj: '',
        nova_delovi:[],
        disable:false

    };
    //this.promena=this.promena.bind(this);

}

/* promenax = () => {
 this.setState({claps: this.state.claps +1})
 };

 promena() {
 this.setState({claps: 2})
 }
 */
render(){
    const broj = this.state.claps;
    // Filtriranje vraca: niz objekata koji zadovoljavaju uslov
    let nova1 = nova.filter(function(el){
        return el.id === broj ;
    });
    const nova_delovi = nova1.map((deo)=>{
        return(
            <div key={deo.id}>
                <Manji   {...deo}/>
                <Modal  {...deo}/>
            </div>
        );
    });
    let nova2  = nova.length;

    if( this.state.claps === nova2+1){
        this.setState({
            claps: this.state.claps -1
        });
    }

    return(
        <div>
            <div>
                <h3> Paginacija- može bilo koji naslov</h3>
```

```
    <ul className="pagination">
      <li>   <button
        type="button" className="btn btn-primary"
        onClick={() =>
          this.setState({claps: this.state.claps -1})
        }>
        {this.state.claps} Stalno Smanjenje
      </button>
      </li>
      <li> <button  onClick={() =>
        this.setState({claps:2})
      }
       >2</button></li>
      <li> <button onClick={() =>
        this.setState({claps:3})
      }>3</button></li>
      <li> <button  onClick={() =>
        this.setState({claps:4})
      }>4</button></li>
      <li>  <button
        type="button" className="btn btn-danger"
        onClick={() =>
          this.setState({claps: this.state.claps +1})
        }>
        {this.state.claps} Stalno Povecanje
        disable={this.state.disable}
      </button>
      </li>
    </ul>
  </div>
  <div>
    <h3>Nova delovi</h3>
    {nova_delovi}
  </div>
  <button
    type="button" className="btn btn-warning"
    onClick={() =>
      this.setState({claps: this.state.claps +1})
    }>
    {this.state.claps} Stalno povecanje disable=
    {this.state.disable}
  </button>
  <button
    type="button" className="btn btn-info"
```

```
                onClick={() =>
                   this.setState({claps: this.state.claps -1})
                }>
                {this.state.claps} Stalno smanjenje
             </button>
          </div>
       );
    }
}

export default Brojac;

class Modal extends Component{
   constructor() {
      super();
      this.state = {
         claps: 0,
         nova: [],
      };
   }
   render(){
      return (
         <div className="modal fade in" id="mojModal" role="dialog"
           aria-labelledby="mojModal">
            <div className="modal-dialog" role="document">
               <div className="modal-content">
                  <div className="modal-header">
                     <button type="button" className="close"
                       data-dismiss="modal" aria-label="Close">
                         <span aria-hidden="true">&times;</span>
                      </button>
                      <h4 className="modal-title">
                      {this.props.title_button}</h4>
                  </div>
                  <div className="modal-body">
                     <img src={this.props.src} alt="..." width={250}
                       height={200}/>
                     <p>
                        {this.props.title_deskript}
                     </p>
                  </div>{this.props.child}
                  <div className="modal-footer">
                     <button type="button" className="btn btn-default"
```

```
                    data-dismiss="modal">Close</button>
                  <button
                    type="button" className="btn btn-warning"
                    onClick={this.promenax}> {this.state.claps}
                    Promene u modalu
                  </button>
                </div>
              </div>
            </div>
          </div>
        );
    }
}

let nova= [
    {"id": 1,
      title_button: 'Reakt',
      number: 25,
      src: require('./slike/jumbo.jpg'),
      title: 'Ime slike druge',
      title_deskript: 'ovo je slika prizora drugog'
    },
    {
      "id": 2,
      title_button: 'Reakt',
      number: 25,
      src: require('./slike/jumbo.jpg'),
      title: 'Ime slike druge',
      title_deskript: 'ovo je slika prizora drugog'
    },
    {
      "id": 3,
      title_button: 'Reakt',
      number: 25,
      src: require('./slike/nikola l.jpg'),
      title: 'Ime slike druge',
      title_deskript: 'ovo je slika prizora drugog'
    },
    {
      "id": 4,
      title_button: 'HTML',
      number: 25,
      src: require('./slike/jumbo.jpg'),
      title: 'Ime slike druge',
```

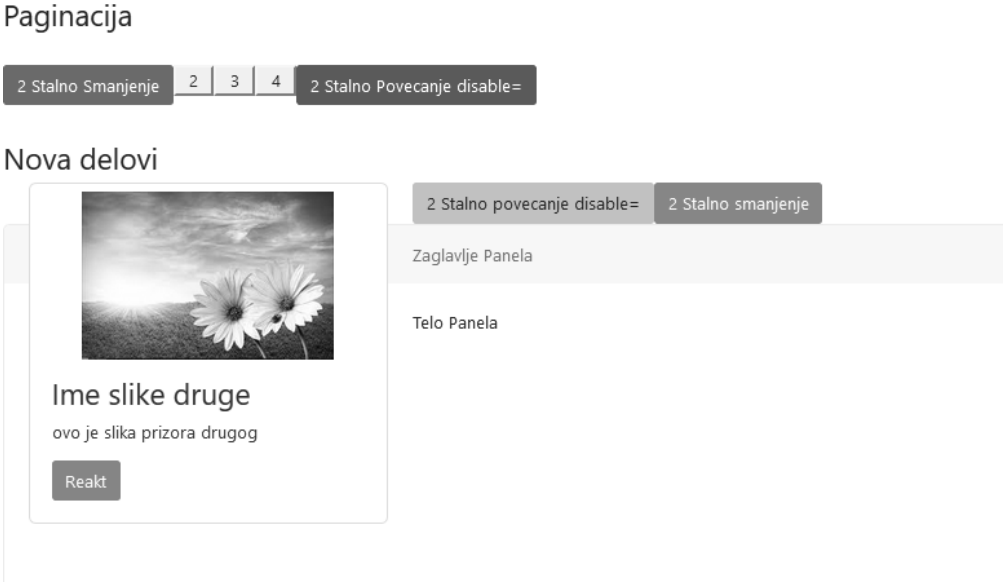
```
    title_deskript: 'ovo je slika prizora drugog'},
  {"id": 5, "title": "Peti"},
  {"id": 6, "title": "Sesti"},
  {"id": 7, "title": "Sedmi"},
  {
    "id": 8,
    title_button: 'BASIC',
    number: 25,
    src: require('./slike/motiv2.jpg'),
    title: 'Ime slike druge',
    title_deskript: 'ovo je slika prizora drugog'
  },
  {"id": 9, "title": "Deveti"},
  {
    "id": 10,
    title_button: 'TRIKA',
    number: 25,
    src: require('./slike/motiv1.jpg'),
    title: 'Ime slike druge',
    title_deskript: 'ovo je slika prizora drugog'
  }
];
```

Fail manjiProzor.js

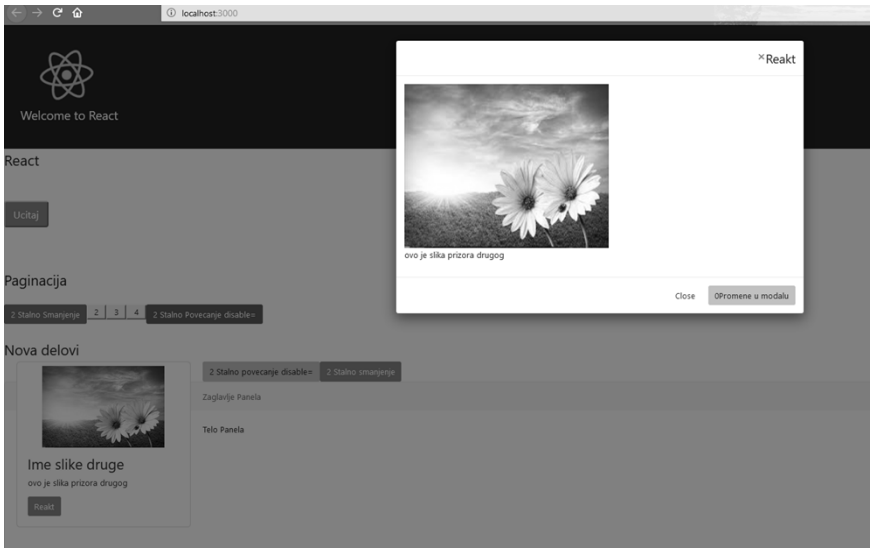
```
import React,{Component} from 'react';

export default class Manji_prozor extends Component {
  render(){
    return(
      <div className="col-sm-6 col-md-2">
        <div className="thumbnail">
          <img src={this.props.src} alt="..." width={150} height={150}
              data-toggle="modal" data-target="#mojModal"
          />
          <div className="caption">
            <h3>{this.props.title}</h3>
            <p>{this.props.title_deskript}</p>
            <button className="btn btn-success">
             {this.props.title_button}</button>
            {this.props.child}
          </div>
        </div>
      </div>
```

```
        );
    }
}
```

U ovom kodu je dodata mogućnost da kada se klikne na prikaz



onda se otvara i modal



Gde opet možete dodati još potrebnih podataka koje zahtevaju vaše aplikacije.

Kako upotrebiti delove Bootstrapa u react aplikaciji

Delovi koda bootstrapa se mogu upotrebljavati u return delu svake komponente, nakon instalacije bootstrap paketa za react okruženje. Delovi koda bootstrapa mogu se smeštati ne samo u return delu klasa već i konstanti i funkcija. Mogu se

pozivati na isti način kao i u bootstrapu ukoliko se upotrebljavaju orginalni ili ako se upotrebljavaju paketi kao reactstrap prema upustvu za njihovo postavljanje u kodu react komponenete. Ako je potrebno za njihovu upotrebu koristiti findDomNode onda se mora u tom kodu gde se upotrebljava u komponenti uključiti i react-dom, pošto je ovo sastavni deo njega. Sintaksa njegove primene je na primer ovakva:

```
componentDidUpdate() {
    let node = ReactDOM.findDOMNode(this);

    this.elementBox = node.getBoundingClientRect();
    this.elementHeight = node.clientHeight;
    }
```

Mada se on može i odnosti na povezivanje refrence. Da ne bi smo gubili vreme u sledećem delu opisaću kod svog sajta gde su u upotrebi paneli, forma za kontakt, geografska mapa, meni što je uobičajena postavka svakog sajta.

Primer veb sajta React-Bootsrap

U ovom primeru u upotrebi je zbog omogućavanja rada menija deo dodatnog paketa reacta – react-router-dom i react-router. Imaćemo delove gde jedna komponenta poziva više panela , a i sam je kao jedna grupni panel. Pored toga biće povezivanja sa video zapisima na youtube prostoru, kao i upotreba teksta i slika, uključivanje modala i slajdera, kao i deo koji nije sastavni do ovog primera sajta a odnosi se na jumbotron klasu, deo menija koji se rasteže na dole povlačaći dugme za klik na dole s time da se iznad njega otvara padajući meni.

Krenimo redom

Pošto smo ovladali reactom neću ponavljati neke detalje u vezi automatske izrade react aplikacije, Znači već u izrađenom projektnom folderu gde je izrađena react aplikacija otvorićemo u editoru koda fail src/App.js i u njemu ostaviti samo ovo:

```
import React, { Component } from 'react';

import './App.css';

class App extends Component {

  render() {

    return (
      <div >

      </div>
```

```
    );
  }
}

export default App;
```

Što bi prestavljalo kostur prilikom izrade ostalih komponenti i stranica veb sajta. Osim toga potrebno je napraviti folder u src folderu komponente, slike i ostalo potrebno za izradu aplikacije.

Index.js

unećemo i promene u failu zbog upotrebe rutera. Razlog je taj što ovaj fail povezuje react aplikaciju sa stvarnim dom elementima. Pored toga baš ovde i postavljamo link prema bootstrap css-u, kako bi svim ostalim komponentama bio dostupan.

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import '../node_modules/bootstrap/dist/css/bootstrap.min.css';
// import 'bootstrap/dist/css/bootstrap.min.css';

import { BrowserRouter } from 'react-router-dom';

ReactDOM.render(
   <BrowserRouter>
      <App />
   </BrowserRouter>
   , document.getElementById('root')
);
```

U upotrebi u failu je deo react-router-dom-a, BorwserRouter. Za više detalja o upotrebi ovog dodatnog react paketa pogledajte na https://reacttraining.com/react-router/web/api/Route . Ovo su dole delovi BorwserRouter-a. Nisu svi obavezni u ovom slučaju, ali ako vam nekada zatrebaju na navedenom prostoru su detaljno opisani.

```
<BrowserRouter
  basename={optionalString}
  forceRefresh={optionalBool}
  getUserConfirmation={optionalFunc}
  keyLength={optionalNumber}
>
  <App/>
```

```
</BrowserRouter>
```

Komponenta meni

```
import React from 'react';
import {Link} from 'react-router-dom';

const Meni = () => (
   <div>
      <nav className="navbar navbar-inverse" style={{zIndex:500}}>
         <div className="container-fluid">
            <div className="navbar-header">
               <button type="button" className="navbar-toggle collapsed"
                     data-toggle="collapse" data-target="#navbar"
                     aria-expanded="false" aria-controls="navbar">
                  <span className="sr-only">Toggle navigation</span>
                  <span className="icon-bar"></span>
                  <span className="icon-bar"></span>
                  <span className="icon-bar"></span>
{/* prazni tagovi kao ovi gore span tagovi trebalo bi da se napišu ovako
  <span className="icon-bar"/> na što će vas upozoriti malo kvalitetniji editori.
       Ali, ne plašite se u oba slučaja kod će raditi */}
               </button>
               <div className="navbar-brand">
                    <Link to="/">Sajt u React-u</Link>
               </div>
            </div>
            <div id="navbar" className="navbar-collapse collapse">
               <ul className="nav nav-pills">
                  <li> <Link to="/">Home</Link> </li>
                  <li> <Link to="/messages">Kultura</Link> </li>
                  <li> <Link to="/paneli">Znanja</Link> </li>
                  <li> <Link to="/about">Sertifikati</Link> </li>
                  <li> <Link to="/diplome">Diplome</Link> </li>
                  <li> <Link to="/nestovise">Nesto Vise</Link> </li>
                  <li> <Link to="/kontakt">Kontakt</Link> </li>
                  <li> <Link to="/pane">Probni Panel</Link> </li>
               </ul>
            </div>
         </div>
      </nav>
   </div>
);

export default Meni;
```

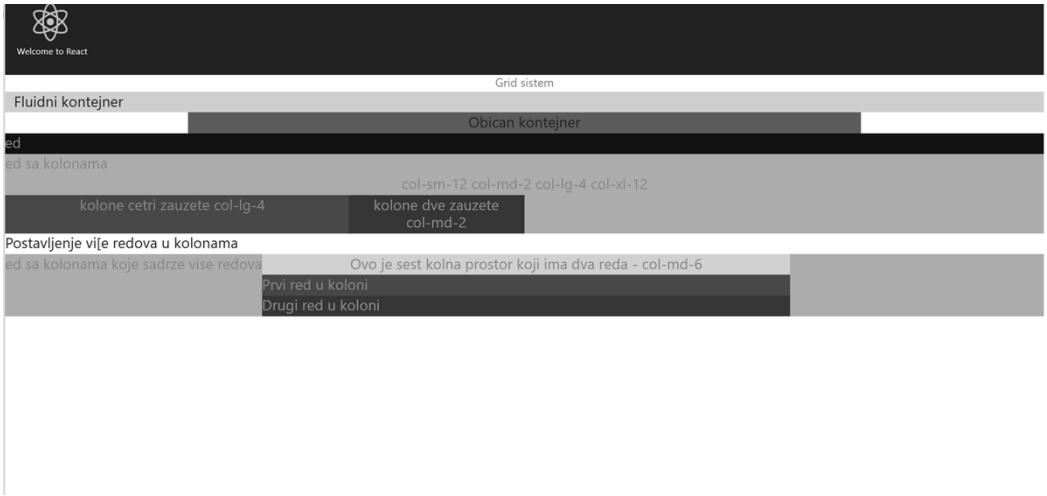
Ovaj kod pokazuje kako se može smestiti i bootstrap kod u konstantu. Ali bilo sta da upotrebljavate za smeštanje html koda bootstrapa ili nekog vašeg html koda potrebno je da bude zatvoren najćešće div tagovima ili html kod nižeg reda sa pripadajućim od višeg reda. U ovom slučaju Div je najvišeg nivoa, pa se onda nav prihvata kao njegovo dete koje ima svoje dete tag ul koje ima svoje dete tag li. Što je i sama hijerarhija nivoa. Tako da to treba uvek imati u vidu. U daljem delu ostaviću samo kod sa malo teksta sa svog veb sajta koji je urađen u react bootstrap okruženju i nalazi se na veb adresi https://miodragtrajanovic.com/

Grid sistem bootstrapa

Namena mu je da omogući lakše pozioniranje elemenata na veb stranicama, podelom na kontejnere, fluidne kontejnere, redove i kolone. S time što kolona ima maksimalno u svim ekranskim rezolucijama 12, a redova može biti neogranicen broj, pri čemu se svaki red može podeliti nezavisno na delove kolona. Kontejner je prostor na stranici čije su leva i desna ivica podjednako udaljene sa obe strance ekrana, dok je visina zavisna o visini upotrebljenih elemenata po visini. On može i da sadrži i redove i kolone, mada ga možete i smestiti u sadržaj reda.
Fluidni kontejner je sličan običnom kontejneru samo on zauzima kompletnu širinu stranice, dok je visina određena kao i kod kontejnera. Uobičajena praksa kod malih veličina ekranskih rezolucija da se fluidne klase isključuju.

```
import React from'react';

export default class extends React.Component{
   render(){
      return(
         <div>
         <h2 className="text-center text-success"> Grid sistem</h2>
            <div className="container-fluid" style={{backgroundColor:"pink"}}>
               <h1>Fluidni kontejner</h1>
            </div>
            <div className="container" style={{backgroundColor:"green"}}>
               <h1 className="text-center">Obican kontejner</h1>
            </div>
            <div className="row" style={{backgroundColor:"blue"}}>
               <h1 className="text-center text-info">Red </h1>
            </div>
            <div className="row" style={{backgroundColor:"orange"}}>
               <h1 className="text-center text-info">Red sa kolonama </h1>
```

```
            <div className="col-sm-12 col-md-2 col-lg-4 col-xl-12">
              <h1 className="text-center text-info">
              col-sm-12 col-md-2 col-lg-4 col-xl-12 </h1>
            </div>
             <div className=" col-lg-4 "
                style={{backgroundColor:"darkgreen"}} >
              <h1 className="text-center text-info">
                 kolone cetri zauzete col-lg-4  </h1>
            </div>
            <div className="col-md-2" style={{backgroundColor:"red"}}>
              <h1 className="text-center text-info">
               kolone dve  zauzete <br />col-md-2 </h1>
            </div>
          </div>
          <h1> Postavljenje vi[e redova u kolonama</h1>
          <div className="row" style={{backgroundColor:"orange"}}>
            <h1 className="text-center text-info">
                 Red sa kolonama koje sadrze vise redova </h1>
            <div className=" col-md-6 " style={{backgroundColor:"gold"}}>
              <h1 className="text-center text-info">
               Ovo je sest kolna prostor koji ima dva reda - col-md-6
              </h1>
               <div className=" row " style={{backgroundColor:"darkgreen"}} >
                  <h1 className="text-center text-info">Prvi red u koloni  </h1>
               </div>
               <div className=" row " style={{backgroundColor:"red"}} >
                  <h1 className="text-center text-info">Drugi red u koloni  </h1>
               </div>
            </div>
          </div>
        </div>
    );
  }
}
```



Uzimajući u obzir da je potrebno prikazivati sadržaje na svim ekranskim rezolucijama u div tagovima kolona mogu se navesti više klasa koje to određuju. Ispod je lista ekranskih rezolucija:

lg 1200px
md 992 px
sm 768 px
xs >=768 px

Kombinacije koje će te primenjivati zavise od vaših potreba, ali treba imati na umu to da je svako prekoračenje sirine ekranskog prikaza ima posledicu da sadržaj prebaca u sledeći red. Ovo zato što se u obračun širine uzimaju veličine zbira širine kolona sa margin-ama, padding-om i border-om. Tako da je moj savet da budete pažljivi.

Pomeranje sadržaja

Upotrebom klasa pull i push možemo pomerati sadržaj u koloni, dok upotrebom offset dolazi do pomeranja kolone u desnu stranu sa povećanjem leve margine.

.col-*-offset-*
.col-*-pull-*
.col-*-push-*
** **

```
          <div class="col-sm-4 col-sm-push-8"
             style="background-color:lavender;">
               col-sm-4 .col-sm-push-8
          </div>
           <div class="col-sm-8 col-sm-pull-4" style="background-color: red ;
        ">
             .col-sm-8 .col-sm-pull-4
          </div>
 </div>
```

Osim primene navedenih u bootstrapu postoje mogućnosti ugneždenja kolona


```
<div class="col-sm-9">
   Level 1: .col-sm-9
   <div class="row">
     <div class="col-xs-8 col-sm-6">
       Level 2: .col-xs-8 .col-sm-6
     </div>
     <div class="col-xs-4 col-sm-6">
       Level 2: .col-xs-4 .col-sm-6
     </div>
   </div>
 </div>
</div>
```

Kod izrade menija postoje takođe klase kojima se pomera sadržaj u izrađenom meniju

```
 <ul class="nav navbar-nav push-right">
       <li class="active"><a href="#">Home</a></li>
       <li><a href="#">Страница  1</a></li>
       <li><a href="#"> Страница 2</a></li>
       <li><a href="#"> Страница </a></li>
  </ul>

<ul class="nav navbar-nav pull-left">
.....
</ul>
```

Ali naravno možete i sami da napravite prema svojim potrebama klase kojima će te opisati potrebne elemente na stranici i to dodati.

Meni

Meniji ili mesta na kojima događajem klik pozivamo prikazivanje određenih stranica ili nečeg što bi trebalo time da se pojavi na ekranu brovzera, kao i sve drugo i u bootstarpu ima svoje delove koje su opisane klasama. Koje možemo podeliti na one koje vrše pozicioniranje:

```
<nav class="navbar navbar-inverse navbar-fixed-bottom ">

</nav>

<nav class="navbar navbar-inverse navbar-fixed-top">

</nav>
```

.navbar-left postavljanje nav linkova, forme, dugmadi, teksta u navigacionom baru na levu stranu
.navbar-right postavljanje nav linkova, forme, dugadi, teksta u navigacionom baru na desnu stranu

.navbar-nav upotrebljava se u <ul> tagu za kontejner <li> tagove

One koje vrše izradu određenih delova menija

.navbar-form

```
<div id="navbarCollapse" class="collapse navbar-collapse">
      <form class="navbar-form navbar-left">
         <div class="form-group">
            <input type="text" placeholder="Search"
             class="form-control">
         </div>
         <button type="submit" class="btn btn-default">
          Submit
         </button>
      </form>
</div>
```

.navbar-header za izradu kontejnera koji sadrži link sa nekim tekstom ili logom

```
<div class="navbar-header">
    <a class="navbar-brand" href="#">Име сајта </a>
</div>
```

.navbar-static-top namena mu je da pomeri bodere levo, desno i gore, kao i da prikaže malo zaokružene uglove u navigacionom baru. Standardni navigacioni bar je sa sivim borderom i ima vrednosti

4px boorder-radius.

.icon-bar sa klasom sr-only je deo koji se upotrebljava da bi se prikazao hamburger meni kod manjih ekranskih rezolucija u obliku izreckanog dugmeta (hamburger/bars), koji se viđa kod mobilnih uređaja.

navbar-inverse boji ceo navigacioni bar u crnu boju, dok je default obojen sivom

<nav class="navbar navbar-inverse ili deafult">

</nav>

navbar izrađuje navigacioni bar
navbar-brand mesto gde se postavlja logo ili naziv sajta i tome slicno i klase koje se upotrebljavaju kao oznake koje kada se upotrebe za pozive iz java skripta omogućile da se time urade određene radnje nad tim elementima

navbar-toggle i navbar-collpase otvaraju skrivene delove menija

Primer koda navigacionog bara

```
<nav class="navbar navbar-inverse">
  <div class="container-fluid">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-toggle="collapse"
              data-target="#myNavbar">
          <span class="icon-bar"/>
          <span class="icon-bar"/>
        <span class="icon-bar"/>
      </button>
      <a class="navbar-brand" href="#">WebSiteName</a>
    </div>
    <div class="collapse navbar-collapse" id="myNavbar">
      <ul class="nav navbar-nav">
        <li class="active"><a href="#">Home</a></li>
        <li><a href="#">Page 1</a></li>
        <li><a href="#">Page 2</a></li>
        <li><a href="#">Page 3</a></li>
      </ul>
      <ul class="nav navbar-nav navbar-right">
```

```
      <li><a href="#"><span class="glyphicon glyphicon-user"/> Sign Up</a></li>
      <li><a href="#"><span class="glyphicon glyphicon-log-in"/> Login</a></li>
     </ul>
   </div>
  </div>
</nav>
```

Napomena: umesto reči class u primerima bootstrapa kada ga upotrebljavate za react aplikaciju upotrebljavajte className, jer će se javiti greška prilikom izvršenja takvog koda, obzirom da java skript upotrebljava class kao službenu reč.

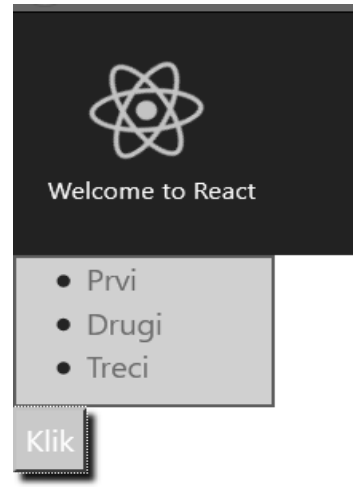
Meni koji se otvara na dole

```
import React from'react';

export default class extends React.Component{
   render(){
      return(
        <div>
          <div style={{ position:"absolute", boder:"#cf422b 2px solid",
                        width:"150px", zIndex:"10"}}>
            <div id="demo" className="collapse"
             style={{height:"auto", width:"100%", backgroundColor: "gold",
                   border:"solid 2px #cf422b" }}>
              <ul>
                <li>
                  <a href="nesto.html" data-toggle="tab"> Prvi</a>
                </li>
                <li>
                  <a href="nesto.html" data-toggle="tab">Drugi</a>
                </li>
                <li>
                  <a href="nesto.html" data-toggle="tab"> Treci</a>
                </li>
              </ul>
            </div>
            <div data-toggle="collapse" data-target="#demo" >
              <button
                style={{ backgroundColor:"aqua", padding:"5px",
                         boxShadow:" blue 5px 5px 3px"}}
                className="btn-info">
                 Klik
              </button>
            </div>
          </div>
```

```
        </div>
      );
   }
}
```

Zatvoren meni

Otvoren meni posle klika na dugme Klik



Karakteristika ovog menija je to što se dugme pomera na dole i pada ispod liste, što nije slučaj kod Dropdown menija, gde lista pada ispod onoga što je poziva. Što je postignuto upotrebom dva div taga, gde je prvi pozicioniran apsolutno, a u drugom je dodato da ima automacku visinu. Tako da collapse klasa poziva istoimeni bootstrapov java skript koji putem poziva događajem onClick na dugme Klik poziva prvi kolapsirajući div putem id taga što je određeno atributom data-target="#demo", dugme pada naniže što ukazije data-toggle="collapse" svojim atributom. Ovaj primer pomalo podseća na efekt koji proizvodi toggle.
Umesto menija mogu se postaviti i drugi elementi veb stranica, tako da uz male promene css-a ovaj kod može da pronađe veliku primenu.

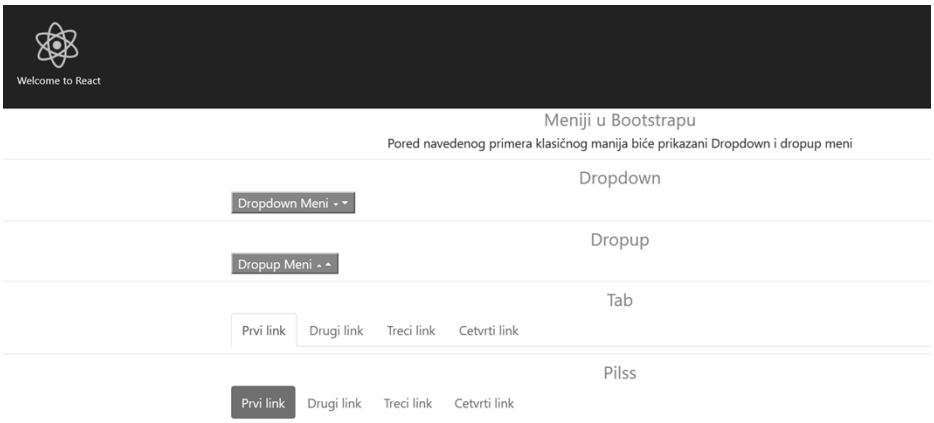
Ostale vrste menija

U boostrapu postoje i ove vrste menija: koji se otvara na dole, na više, u obliku tabova i pils. Za prva dva prilikom upotrebe obrati pažnju na njihovo postavljanje i prostor koji treba da zauzmu u smeru otvaranja. Kod za sve ove vrste menija je

```
import React from'react';
import './stil.css';

export default class extends React.Component{
   render(){
      return(
         <div>
            <h1 className="text-center text-success"> Meniji u Bootstrapu</h1>
```

```
 <p className="text-center">
Pored navedenog primera klasičnog manija
biće prikazani Dropdown i dropup meni
 </p>
 <hr />
<div className="container">
   <h1 className="text-center text-success"> Dropdown</h1>
   <div className="btn-group">
      <button className="btn-success dropdown-toggle"
        data-toggle="dropdown">
         Dropdown Meni <span className="caret"/>
      </button>
      <ul className="dropdown-menu">
         <li> <a href="neki.href"> Prvi link</a></li>
         <li> <a href="neki.href"> Drugi link</a></li>
         <li> <a href="neki.href"> Treci link</a></li>
         <li> <a href="neki.href"> Cetvrti link</a></li>
      </ul>
   </div>
</div>
 <hr />
<div className="container">
   <h1 className="text-center text-success"> Dropup</h1>
   <div className="dropup">
      <button className="btn-success dropdown-toggle"
           data-toggle="dropdown">
         Dropup Meni <span className="caret"/>
      </button>
      <ul className="dropdown-menu">
         <li> <a href="neki.href"> Prvi link</a></li>
         <li> <a href="neki.href"> Drugi link</a></li>
         <li> <a href="neki.href"> Treci link</a></li>
         <li> <a href="neki.href"> Cetvrti link</a></li>
      </ul>
   </div>
</div>
 <hr />
 <div className="container">
   <h1 className="text-center text-success">Tab</h1>
      <ul className="nav nav-tabs">
         <li className="active"> <a href="neki.href" > Prvi link</a></li>
         <li> <a href="neki.href"> Drugi link</a></li>
         <li> <a href="neki.href"> Treci link</a></li>
         <li> <a href="neki.href"> Cetvrti link</a></li>
```

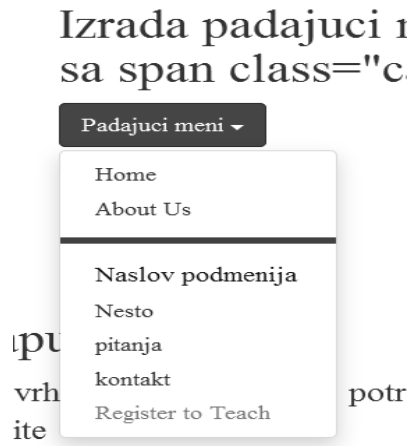
```
                </ul>
            </div>
             <hr />
             <div className="container">
               <h1 className="text-center text-success">Pilss</h1>
                  <ul className="nav nav-pills">
                     <li className="active"> <a href="#" data-toggle="pill" >
                      Prvi link</a></li>
                     <li> <a href="#"  data-toggle="pill"> Drugi link</a></li>
                     <li> <a href="#"  data-toggle="pill"> Treci link</a></li>
                     <li> <a href="#"  data-toggle="pill"> Cetvrti link</a></li>
                  </ul>
               </div>
          </div>
       );
    }
}
```



Kod menija koji se rasklapaju naviše i naniže postoji upotrebljena klasa caret koja pokazuje smer otvaranja menija, i može biti kao u ovom slučaju sa desne strane ispisa:

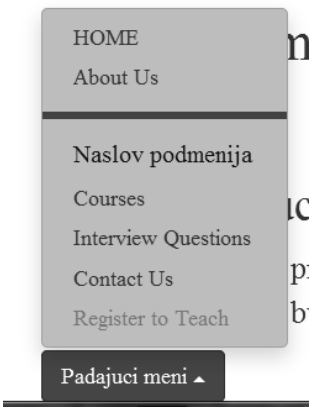
```
<button className="btn-success dropdown-toggle"
        data-toggle="dropdown">
        Dropdown Meni <span className="caret"/>
    </button>
```

Ali se može i pomeriti sa leve, tako da se može videti i sa leve strane ispisa naziva menija koji se vidi u svakom slučaju. Ostala dva menija mogu biti kao na slici u horizontalnom položaju, ali isto tako mogu biti u vertikalnom položaju, takođe se može dodati kombinacija u tab meniju sa padajućim podmenijem




```
<ul className="nav nav-tabs">
   <li className="active"><a href="#">Home</a></li>
   <li className="dropdown">
      <a className="dropdown-toggle" data-toggle="dropdown" href="#">Menu 1
         <span className="caret"/></a>
      <ul className="dropdown-menu">
         <li><a href="#">Submenu 1-1</a></li>
         <li><a href="#">Submenu 1-2</a></li>
         <li><a href="#">Submenu 1-3</a></li>
      </ul>
   </li>
   <li><a href="#">Menu 2</a></li>
   <li><a href="#">Menu 3</a></li>
</ul>
```

Tab meni daje sliku menija u obliku tabova, dok pilss omogućava automatizaciju aktivnosti dugmeta u meniju. Ta mogućnost kod pills se postiže ako se isključi iz listitems (li) klasa active.

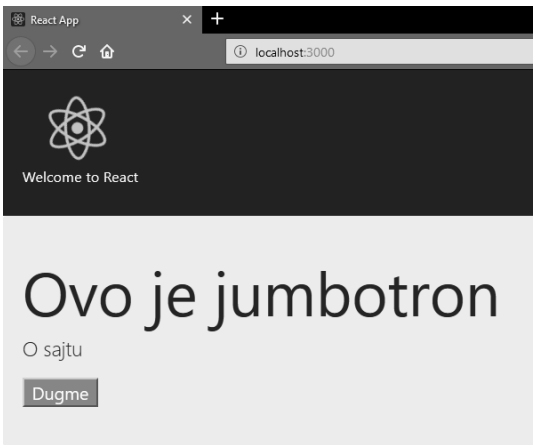


slika dropup menija

Jumbotron

Prestavlja jedno veće područje sivkaste boje u kone možete da smestite dosta toga. Namena mu je da se tu postavi naslov sajta, slajder(carousel), jedno dugme, možda meni. Izgled Jumbotrona, možete da promenite ako mu dodate sliku kao pozadinu iz css-a. Dugmetu se može dodati link ili neka druga akcija kao i svemu na njemu.

```
<div className="jumbotron">
   <h1>Ovo je jumbotron</h1>
   <p> O sajtu</p>
   <button className="btn-info"> Dugme</button>
</div>
```



Slajder(carousel)

U njemu možete da prikazujete slajd po slajd, koji sadrži zaglavlje, telo i dno. Na njemu se mogu pored slika pronači i linkovi prema video zapisima, dugmad, naslovi i manji tekstovi. U njemu je omogućeno da direktno idete na određen slajd ili da prelazite na sledeći ili se vraćate na predhodni slajd. Dok, oblici znakova, koji pokazuju na kom ste trenutno slajdu, možete da im menjati oblike od krugova standardno u druge oblike. Slajder je smešten u jedan div tag sa klasom carousel u kome se nalaze njegovi delovi:

1. slajdovi
2. i ne obavezni delovi: kontrole levo/desno i indikatori

Slajdovi

Ovaj deo je obavezan jer se u njemu prikazuje sadržaj koji prikazuje Karusel, zatvoren je u div tag sa klasom carousel-inner u kojoj svaki njegov deo počinje div tagom sa klasom carousel-item, i gde aktivan ili onaj slajd od kog se uvek počinje ima klasu carousel-item active. Klasa active je karakteristična i za druge njegove delove, pa tako ako je kao u našem slučaju prvi slajd aktivan onda je klasa active i u delu indikatora ako se upotrebljavaju na istom mestu. Ali nemojte da vas zbuni to što je ovde u pitanju broj nula označen kao prvi, to je zato što sva brojanja uvek

počinju od broja nule nebitno da li su u plus ili minus beskonačno. Ta nula vezan je i za brojnu vrednost našeg znaka “A”. U ovom delu možete imati koliko vam je potrebno ovih delova:

```
<div className="carousel-item active">
  <img src="karmen.jpg" className="img img-responsive" alt=""/>
    <div className="carousel-caption">
      <h1>Naslov u karuselu</h1>
      <p>Neki tekst objasnjenja.</p>
      <button className="btn-success">Dugme u karuselu</button>
    </div>
</div>
```

Čime svaki slajd može imati svoje sve ove delove ili može imati samo sliku. Ali, ono što je još intersantno umesto slike možete postaviti link prema vašim video zapisima, tako da se umesto slike pojavljuje statična slika video zapisa ali kada se klikne na nju on se pokreće. Pozicioniranje tog srednjeg dela slajdera gde se vide slike i slično vrši se iz css-a, gde s vrši i određivanje parametara veličine te slike, ali kao i što vidite i brzina rada prikaza slika u karuselu.

```
.carousel-inner img {
  margin: auto;
// ovim delom se postavljaju slajdovi uvek u sredinu prikaza na ekranu
  width: 400px;
  height: 250px;
}

.carousel .carousel-item {
  transition-duration: 1.5s;// brzina prolaza slajdova
}
```

Indikatori

```
{/* Indikatori */}
<ol className="carousel-indicators">
  <li data-target="#carousel-example-generic" data-slide-to="0"
      className="active"/>
  <li data-target="#carousel-example-generic" data-slide-to="1"/>
  <li data-target="#carousel-example-generic" data-slide-to="2"/>
</ol>
```

Su namenjeni da se klikom na jedan od njih direktno ode na ciljni slajd u slajderu. Napravljeni su u obliku uređene liste sa listItem delom, tako da promenama u css-u možete da podesite izgled indikatora upotrebom css-a list-style-type: circle; . Možete ih postaviti da budu iznad dela u kom se prikazuju slike ili ispod.

Kontrole levo/desno

Kontrole su tu da se krećete po prezentaciji u slajderu po principu jedan slajd ispred ili iza onog na kome ste trenutno.

```
    {/* Kontrola */}
    <a className="carousel-control-prev" href="#carousel-example-generic"
          role="button" data-slide="prev">
       <span className="carousel-control-prev-icon" aria-hidden="true"/>
       <span className="sr-only">Previous</span>
    </a>
    <a className="carousel-control-next" href="#carousel-example-generic"
            role="button" data-slide="next">
       <span className="carousel-control-next-icon"
style={{backgroundColor:"red"}}
         aria-hidden="true"/>
       <span className="sr-only">Next</span>
    </a>
```

Promena njihovog izgleda pored css-a može se uraditi izmenom

```
       <span className="carousel-control-prev-icon" aria-hidden="true"/>
```

u

```
<span class="glyphicon glyphicon-chevron-left" " aria-hidden="true" />
```

Upotrebom boostrapovih ikonica ili glifikona ili upotrebom awesome fontova, s time što se awesom fontovi moraju naknadno instalisati, kako piše u delu knjige o reactu.

Kompletan primer kod za slajder ili karusel:

```
<div id="carousel-example-generic" className="carousel slide" data-
ride="carousel" style={{backgroundColor:" pink"}}>
    {/* Indikatori */}
    <ol className="carousel-indicators">
       <li data-target="#carousel-example-generic" data-slide-to="0"
className="active"/>
       <li data-target="#carousel-example-generic" data-slide-to="1"/>
       <li data-target="#carousel-example-generic" data-slide-to="2"/>
    </ol>

{/* sladovi */}
    <div className="carousel-inner" role="listbox">
       <div className="carousel-item active">
          <img src="karmen.jpg" className="img img-responsive" alt=""/>
             <div className="carousel-caption">
                <h1>Naslov u karuselu</h1>
```

```
                <p>Neki tekst objasnjenja.</p>
                <button className="btn-success">Dugme u karuselu</button>
              </div>
        </div>
        <div className="carousel-item">
          <img src="motiv1.jpg" className="img img-responsive" alt="slika"/>
        </div>
        <div className="carousel-item">
          <img src="motiv2.jpg" className="img img-responsive" alt="slika3"/>
        </div>
    </div>

    {/* Controls */}
    <a className="carousel-control-prev" href="#carousel-example-generic"
            role="button" data-slide="prev">
      <span className="carousel-control-prev-icon" aria-hidden="true"/>
      <span className="sr-only">Previous</span>
    </a>
    <a className="carousel-control-next" href="#carousel-example-generic"
           role="button" data-slide="next">
      <span className="carousel-control-next-icon"
       style={{backgroundColor:"red"}} aria-hidden="true"/>
      <span className="sr-only">Next</span>
    </a>
 </div>
```

Princip rada slajdera

Sam slajder, kao što mu i ime govori nemanjen je da svojim sadržajem bliže informiše o na primer ponudama proizvoda neke firme, te zato se uobičajeno postavlja na vrh buduće stranice, gde se nalazi i izbornik kojim se krećemo po veb sajtu. U nekim slučajevima se može postaviti ispod Jumbotron-a, ali i u njemu. Radi vrlo jednostavno, jer ima osnovni div koji je praktično ono nevidljivo oko slajda i koji je veće veličine u kome se nalazi jedan manji deo koji je tu providan u kome se prikazuju slajdovi sa svojim sadržajem. Slajdovi se kreću zahvaljujući css-u, standardno sa desna na levo, ali se može podesiti upotrebom css-a da se kreću i u ostalim pravcima.
Veličina slajda može da se menja css-om, što je pogodno ako su upotrebljene slike različite rezolucije, čime bi se sprečilo podrhtavanje stranice. Dok, ovim delom u css-u postavljaju se slajdovi uvek u sredinu prikaza na ekranu

```
.carousel-inner img {
   margin: auto;
```

Napomena: Veličina slika ili video zapisa u slajderima najbolje je da se podese u css-u

Modal

Deo može se reći arsenala gotovih elemenata budućih starnica koji ima veliku primenu u dosta slučajeva. Na primer login, singup i registar forme, bliže pojašnjenje nekog dela proizvoda ili usluga. To je jedan manji uokviren prostor, koji se pojavljuje pozivom događaja ostavljajući iz sebe malo zatamljenu preostalu površinu.
On može da sadrži sve elemente jedne veb stranice. Ima heder, telo i dno. A nestaje klikom na određena dugmad ako ih ima na njemu ili bilo gde van njega u prostoru.
U klasičnom html kodu ili u reactu možete ga pozvati dodavanjem koda

data-toggle ="modal" data-target ="#Mojmodal"

data-toggle otvara modal prozor
data-target poziva upotrebom id-a odreređeni modal koji ima isti taj id
Klasa modal identifikuje sadržaj div taga
klasa fade omogućava tranzicioni efekt pojave modala
modal-header zaglavlje modala
modal-title naslov modala
modal-body telo modala –srednji deo
modal-footer dno modala
modal-sm i modal-lg određuju veličinu modala

U bilo koji element na stranici. U react kodu za početak najbolje ga je upotrebiti direktno u failu koji ga i upotrebljava.

```
import React from'react';

export default class extends React.Component{
   render(){
      return(
         <div>
            <h2 className="text-center text-success"> Modal</h2>
            <div className="container">
               <h2>Modal Example</h2>
               <button type="button" className="btn btn-info btn-lg"
                  data-toggle="modal"
                  data-target="#myModal">
                   Open Modal
               </button>
               <div className="modal " id="myModal" role="dialog">
                  <div className="modal-dialog">
                     <div className="modal-content">
                        <div className="modal-header">
                           <button type="button" className="close" data-
```

```
                        dismiss="modal">
                      &times;
                    </button>
                    <h4 className="modal-title">Zaglavlje modala</h4>
                  </div>
                  <div className="modal-body">
                    <h4 className="text-left">Telo modala</h4>
                    <p>
                      Ovde možete dodati sliku tekst dugme ili video zapis link.
                    </p>
                  </div>
                  <div className="modal-footer">
                    <h4 className="text-left">Dno modala</h4>
                    <button type="button" className="btn btn-default"
                      data-dismiss="modal">
                        Close
                    </button>
                  </div>
                </div>
              </div>
            </div>
          </div>
        );
    }
}
```

Modalu je moguće dodati klase poput text-warning i slične za upotpunjenje ako se upotrebljava za neka obaveštenja ili upotrebiti kod

```
<div className="modal-header " style={{backgroundColor:"red"}}>
```

Čime smo obojili heder, mada se može primeniti i na ostale delove slično.

Reactstrap modal

Pored Bootstrap koda za modal se može upotrebiti i reactstrap za njihovu izradu, kao što se može videti u kodu dole.

```
import {Modal, ModalHeader, ModalBody, ModalFooter, Button} from
'reactstrap';

class Parent extends React.Component {

   constructor(props) {
```

```
    super(props);
    this.state = {
      modal: false
    };
    this.toggle = this.toggle.bind(this);
  };

  toggle(){
   this.setState({
    modal: !this.state.modal
   });
  }

  render() {
    return(
      <div>
        <button onClick={ this.toggle}>Contact Us</button>
        <Modal isOpen={this.state.modal} fade={false}
            toggle={this.toggle} style={{width:"200px", display:"block"}}>
          <ModalHeader toggle={this.toggle}>Modal title</ModalHeader>
          <ModalBody>
           Ovo je telo modala, gde možete postaviti bilo koji element
           koji se može postaviti na veb stranici
         </ModalBody>
          <ModalFooter>
            <Button onClick={this.toggle}>Do Something</Button>{' '}
            <Button onClick={this.toggle}>Cancel</Button>
          </ModalFooter>
        </Modal>
      </div>
    );
  }
}
```

Napomena: Kod upotrebe ovog dodatnog paketa delovi koji se upotrebljavaju naznačavaju se u delu importa u obliku destruktora, a kada se upotrebljavaju u obliku tagova počinju velikim slovom za razliku od html tagova.
Problem koji se može pojaviti u slučaju upotrebe reactstrap modala je da se ceo ekran zatamni, te je u tom slučaju potrebno je dodati da je fejd false, kao što sam i ja uradio

```
        <Modal isOpen={this.state.modal}
            fade={false}
            toggle={this.toggle}
```

```
                style={{width:"200px", display:"block"}}>
```
Ali ako se dodaju forme u ovaj modal porebno je da form tag počne pre taga Modal.Header i da se završi ispred taga Modal kojim se on zatvara. Ovo zato da bi se iskoristili delovi modala za delove tabele. Tačnije da bi Modal.Title preuzeo ulogu naziva tabele, a Modal.Footer da bi se postavila potrebna dugmad kako za formu, tako i za modal.
U slučaju da upotrebljavate modal za prijavu ili registraciju potrebno je dodati izmene. Dati još jednu komponentu koja se naziva Tab, čiji je kod ovakav:

```
import React from 'react';
import { TabContent, TabPane, Nav, NavItem, NavLink, Card, Button,
            CardTitle, CardText, Row, Col } from 'reactstrap';
import classnames from 'classnames';

class Tab extends React.Component {
   constructor(props) {
      super(props);
      this.toggle = this.toggle.bind(this);
      this.state = {
         activeTab: '1'
      };
   }

   toggle(tab) {
      if (this.state.activeTab !== tab) {
         this.setState({
            activeTab: tab
         });
      }
   }
   render() {
      return (
         <div>
            <Nav tabs>
               <NavItem>
                  <NavLink
                     className={classnames({ active: this.state.activeTab === '1' })}
                     onClick={() => { this.toggle('1'); }}
                  >
                     Tab1
                  </NavLink>
               </NavItem>
               <NavItem>
                  <NavLink
```

```
                  className={classnames({ active: this.state.activeTab === '2' })}
                    onClick={() => { this.toggle('2'); }}
                  >
                    Moar Tabs
                  </NavLink>
                </NavItem>
              </Nav>
              <TabContent activeTab={this.state.activeTab}>
                <TabPane tabId="1">
                  <Row>
                    <Col sm="12">
                      <h4>Tab 1 ukupan naslov svega</h4>
                    </Col>
                  </Row>
                </TabPane>
                <TabPane tabId="2">
                  <Row>
                    <Col sm="6">
                      <Card body>
                        <CardTitle>Prijava na nalog</CardTitle>
                        <CardText>
                            Prostor za dodavanje sadržaja.
                        </CardText>
                        <Button>Odustajanje </Button>
                        <Button>Prijava </Button>
                      </Card>
                    </Col>
                    <Col sm="6">
                      <Card body>
                        <CardTitle>Registracija naloga </CardTitle>
                        <CardText>
                        Forma za registraciju
                        </CardText>
                        <Button>Odustajanje </Button>
                        <Button>Reagistracija </Button>
                      </Card>
                    </Col>
                  </Row>
                </TabPane>
              </TabContent>
            </div>
        );
    }
}
```

Jednostavno na primer ovakvom izmenom dela koda u tabpanelu broj id=2

```
<TabPane tabId="2">
   <Row>
      <Col sm="6">
         <form>
            <input type="text"/><br/>
            <input type="text"/><br/>
            <input type="text"/><br/>
         </form>
         <hr height={5}/>
         <Button className="btn-danger pull-left btn-sm">Odbijanje</Button>
         <Button className="btn-success pull-right">Prihvatanje</Button>
      </Col>
```

Pošto je ovo primer, kao i drugi u knjizi potrebno je samo dodati potreban css za bolji izgled.

Slike i video zapisi

Slike, kao i video zapisi i sve drugo u bootstrapu je responsivno. Slike pored osnovne klase img imaju i:

1. img thumbnail koja daje sluku u jendom manjem okviru sivkasete boje
2. img-rounded prikazje sliku sa malo zaobljenim uglovima
3. img-cicrle prikazuje zaokruženu sliku ili sliku čiji izgled podseća na krug
4. img-responsive klasa za prilagođavanje različitin rezolucijama

Sa napomenom da je u reactu obavezan atribut alt=" " i da tag slike ima ovakav kod

```
<div className=" img-cicrle " src=   alt=" " />
```

Gde u zavisnosti od pristupa failovima slika u src –u se menja kod prema tome da li se upotrebljavaju apsolutne ili relativne putanje do njih. Moguće je upotrebiti više klasa odjednom.
Video sadržaju se kao i sve drugo potpuno ravnopravno mogu prikazivati, ali ono što je preporučljivo da oni budu zbog svoje fizičke veličine na nekom drugom serveru. Video zapisi na stranicama se uključuju ovako:

```
<div className="embed-responsive embed-responsive-16by9">
          <iframe src="https://www.youtube.com/embed/Z-
         43r0rSI4s?list=PLPQBfZaQFu4aXwFRcugpPh7G7vyDCoX7s"
```

```
                    title="video5" frameborder="0" allow="autoplay;
    encrypted-media" webkitallowfullscreen mozillowfullscreen
    allowfullscreen>
    </iframe>
</div>
```

React zahteva da imaju svoj title atribut ispunjen. Klase:

1. embed-responsive prikazuje na svim rezolucijama podjednako
2. embed-responsive-16by9 isto kao prva samo sto je usmerna na prikaz u odnosu 16 X 9
3. embed-responsive-4by3 ovaj za prikaz 4 X 3

Prikaz google mapa

Google mape su dosta karakteristične za uključivanje i prikaz na veb stranicama zbog nekih njihovih posebnih osobina, te zahtevaju pored preuzimanja linka sa koodrinatama:

```
import React from 'react';

const Kontakt = () => (
  <div className="container odozgo">
    <div className="row ">
      <div className="col-xs-12 col-sm-6 col-md-6 col-lg-6">
        <a href=" https://www.sololearn.com/Profile/509144/">
          <img src=   {require('../img/solo.jpg')} width={100}
          className="desno img-thumbnail img-responsive pull-left" alt=".."/>
        </a>
        <a href="https://www.linkedin.com/in/miodrag-trajanovic-4559a161/">
          <img src={require('../img/linked.jpg')} width={100}
           className=" img-thumbnail img-responsive pull-left" alt=".."/>
        </a>
      </div>
      <div className="col-xs-12 col-sm-6 col-md-6 col-lg-6">
        <div className="row">
          <div className="col-xs-12 col-sm-3 col-md-3 col-lg-3">
            <img src={require('../img/vats.jpg')} width={80}
             className=" img-thumbnail img-responsive pull-left"
                    alt=".."/><br/>
          </div>
          <div className="col-xs-12 col-sm-3 col-md-3 col-lg-3">
            <a>+38166402583</a>
          </div>
        </div>
        <div className="row">
```

```
          <div className="col-xs-12 col-sm-3 col-md-3 col-lg-3">
             <img src={require('../img/viber.jpg')} width={80}
             className=" img-thumbnail img-responsive pull-left"
               alt=".."/><br/>
          </div>
          <div className="col-xs-12 col-sm-3 col-md-3 col-lg-3">
             <a>+38166402583</a>
          </div>
        </div>
    </div>
</div>
<div className="row">
    <div className="alert collapse alert-success" id="uspesno">
       <strong>Postovani!</strong> Vasa poruka je poslata .
       <a href="/" className="close" data-dismiss="alert"
          aria-label="close">&times;</a>
    </div>
    <div className="col-xs-12 col-sm-6 col-md-6 col-lg-6">
       <h3>Mozete putem ove forme poslati email</h3>
       <form >
          <div className="input-group">
             <span className="input-group-addon">Tema poruke </span>
             <input id="subjekt" type="text" className="form-control"
                 name="subjekt" placeholder="Tema poruke "/>
          </div>
          <div className="input-group">
             <span className="input-group-addon">
                <i className="glyphicon glyphicon-user"/></span>
             <input id="ime" type="text" className="form-control"
              name="ime" placeholder="Svoje ime "/>
          </div>
          <div className="input-group">
             <span className="input-group-addon">
               <i className="glyphicon glyphicon-inbox"/></span>
             <input id="email" type="text" className="form-control"
                name="email" placeholder="Email"/>
          </div>
          <div className="input-group">
             <span className="input-group-addon">
               <i className="glyphicon glyphicon-lock" /></span>
             <input id="password" type="password" className="form-control"
                 name="password"  placeholder="Password"/>
          </div>
          <div className="input-group">
```

```
                    <span className="input-group-addon">
                      <i className="glyphicon glyphicon-envelope"/>
                     </span>
                    <textarea id="msg"  className="form-control" name="msg"
                            placeholder="Prostor za ostavljanje pooruke"/>
                  </div>
                  <button className="btn-danger pull-left" type="reset" >
                    Resetuj</button>
                     <button className="btn-success pull-right" type="submit"
                      name="posalji" data-toggle="collapse"
                       data-target="#uspesno">Posalji</button>
                </form>
              </div>
              <div className="col-xs-12 col-sm-6 col-md-6 col-lg-6">
                <h3>
                   Dodatne infromacije
                </h3>
              </div>
            </div>
            <br/>
            <div className="google-maps ">
              <iframe
src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d23074.53743
2248846!2d21.441146427288807!3d43.703957143357805!2m3!1f0!2f0!3f0!3m2!
1i1024!2i768!4f13.1!3m3!1m2!1s0x47569ab97687560b%3A0x6adb83accf31be2a
!2zVG9tacSHYSBNaWxpxIdhLCDEhmnEh2V2YWM!5e0!3m2!1sen!2srs!4v152
6032607086"
                 title="mapa"  width={600} height={450}  style={{border:"0"}}
allowFullScreen></iframe>
            </div>
            <br/><br/><br/>
          </div>
);

export default Kontakt;
```

svoj title, ali i css kod:

```
.google-maps {
  position: relative;
  padding-bottom: 50%; // This is the aspect ratio
  height: 0;
  overflow: hidden;
}
```

```
.google-maps iframe {
  position: absolute;
  top: 0;
  left: 0;
  width: 100% !important;
  height: 100% !important;
}
```

Bez čega se teško pozicioniraju na veb stranici. Ovaj primer je na mom veb sajtu i to je stranica kontakt (http://miodragtrajanovic.com/kontakt).

Forme

Elementi za unos i prosleđivanje podataka, gde je u bootstrapu uneta i kontrola polja za unos specfičnim klasama i glifikon ikonicama.

```
import React from'react';

export default class Froma extends React.Component {
   render() {
      return (
         <div>
            <h2>Inline forma: kontrolom stanja unosnih polja</h2>
            <form className="form-inline">
               <div className="form-group">
                  <label htmlFor="focusedInput">Focused</label>
                  <input className="form-control" id="focusedInput" type="text"/>
               </div>
               <div className="form-group">
                  <label htmlFor="inputPassword">Disabled</label>
                  <input className="form-control" id="disabledInput" type="text"
                     disabled/>
               </div>
               <div className="form-group has-success has-feedback">
                  <label htmlFor="inputSuccess2">Input with success</label>
                  <input type="text" className="form-control" id="inputSuccess2"/>
                     <span className="glyphicon glyphicon-ok form-control-
                        feedback"/>
               </div>
               <div className="form-group has-warning has-feedback">
                  <label htmlFor="inputWarning2">Input with warning</label>
                  <input type="text" className="form-control"
                        id="inputWarning2"/>
                     <span className="glyphicon glyphicon-warning-sign
```

```
                    form-control-feedback"/>
                </div>
                <div className="form-group has-error has-feedback">
                  <label htmlFor="inputError2">Input with error</label>
                  <input type="text" className="form-control" id="inputError2"/>
                  <span className="glyphicon glyphicon-remove
                    form-control-feedback"/>
                </div>
              </form>
            </div>
        );
    }
}
```

Mada ova obaveštenja mogu se putem if else uslova nadograditi, tj. dodati uslovi validacije unetih podataka na strani korisnika zajedno sa klasom has-error.

1. Form-control omogućava kontrolu unosa dok form-control-feedback se može upotrebiti kao što se vidi na slici za povratnu informaciju prilikom validacije unešenog podatka u unosno polje.

2. Klasa from-group je praktično kontejner koji grupiše unosna polja i njhove labele.
3. form-horizontal poravnava labele i from-groups-control u horizontalnom smeru
4. from-inline izrađuje forme, gde su poravnanja na levo sa inline block kontrolama. Prema dokumentima o bootstapu namena ove klase je za prikaz na ekranima 767px i većim, a forma je prikazana u jednoj liniji kao na slici iznad. Te je pogodna za izradu formi koje se upotrebljavaju za pretrage interneta ili prema nazivu nekog proizvoda.

Primena klasa from-inline i form-horizontal izgleda ovako:

```
<form class="form-horizontal">
*********
</form>
```

Dok ostalih pomeutih vidite u gornjem primeru koda kompletne forme.

Dugmad u bootstrapu

Dugmad ima standardna obojenja, koa što vidite u kodu ispod

```
<p><h2>Klasa  btn-group u bootstrapu namenjena je da grupiše dugmad</h2></p>

<div class="btn-group">
   <button type="button" class="btn btn-default">
       standardno belodugme sa sivim okvirom</button>
   <button type="button" class="btn btn-primary">
       primarno ima plavu tamniju boju</button>
   <button type="button" class="btn btn-info">
        informaciono ima svetliju plavu boju</button>
   <button type="button" class="btn btn-danger">
        opasnost obojeno je crvenom jarkom bojom</button>
   <button type="button" class="btn btn-warning">
         upozorenje obojeno je narandzstom bojom</button>
   <button type="button" class="btn btn-success">uspesno Zelenom bojom
    </button>
</div>
```

.btn Osnovna klasa koja daje sivu boju dugmetu.
.btn-block postavlja dugmad u jedan vertikalni niz sa redoslednom pozicijom (parent element)
.btn-group btn-group u bootstrapu namenjena je da grupiše dugmad u jednoj horizontalnoj liniji
.btn-group-justified proširena btn-group samo koja grupisanu dugmad prikazuje raširenо preko celokupne širine ekranskog prikaza u jednom redu
.btn-group-lg , .btn-group-sm, .btn-group-xs ove klase se upotrebljavaju shodno širini ekranskog prikaza, gde je lg najveća moguća širina(preko 1200px) xs je najmanja
.btn-group-vertical grupisanje dugmadi u vertikalnu grupu dugmadi
.btn-lg .btn-sm .btn-xs izrađuje dugmad prema veličini(lg najveće, xs najmanje dugme)
.btn-link daje boju dugmetu kao što je upotrebljene boja za linkove.

Tabele

Tabele imaju isti html kod ali imaju određene opise-klase koje im daju mogućnosti da budu prilagodljive veličini ekranskog prikaza. Klase kojima su tabele prestavljene u bootstrapu:

1. table je osnovna klasa
2. table-bordered za dodelu borvsera na svim stranama tabele
3. table-condensed upotrebljava se za izradu mnogo kompaktniji tabela, smanjujući padding na polovinu vrednosti. Ovim se tabelama daje celoviti

izgled jer se i smanjuje odstojanje od ispisa u ćelijama na polovinu rastojanja

4. table-hover Ovo je klasa koja daje efekt hover kao i kod ostalih elemenata veb stranica, samo što ova klasa kada se prođe pokazivačem miša preko tabele taj deo ispod pokazivača je obojen sivom bojom.
5. table-responsive upotrebljava se za izradu tabela koje se mogu prilagoditi različitim širinama ekrana, pri čemu na manjim širinama tabela dobija jedan horizontalni skrolbar, te je onda moguće kretanje po horizontalnom smeru koliko je potreban da se vide svi delovi tabele
6. table-striped upotrebom ove klase tabela dobija sivo-bele pruge u redovima

Umesto table-stiped možete upotrebiti sledeći kod da bi ste imali tabele određenih potreba. Dodavanje boje redovima

```
<tr class="active">...</tr>
 <tr class="success">...</tr>
 <tr class="warning">...</tr>
<tr class="danger">...</tr>
 <tr class="info">...</tr>
```

Dodavanje obojenja ćelijama i zaglavljima tabela (`td` or `th`)

```
 <tr>
<td class="active">...</td>
<td class="success">...</td>
<td class="warning">...</td>
<td class="danger">...</td>
<td class="info">...</td>
</tr>
```

Ali isto može se primeniti i na tag tbody. Da bi smo dodali neke vrednosti koje se mogu menjati prema učitanim podacima u react aplikaciji u delovima ćelija, zaglavlja možemo dodati u zavisnosti da li se tabela upotrebljava kao komponenta koja ima svoje vrednosti stanja

```
{this.state.naziv-polja-cija-se-vrednost-ocitava}
```

Ili ako se upotrebljava kao pod komponenta- child onda se njoj šalju stanja da bi ona prikazala props vrednost

```
{this.props.naziv-polja-cija-se-vrednost-ocitava}
```

Media

Upotreba ove klase je na primer da bi se mogla dobiti lista komentara između više korisnika, gde se mogu postavljati slike i tekstovi. Slike moraju biti manje veličine, na primer do 100 x 100 px, jer se dešavaju u slučaju upotrebe slika većih dimanzija greške u prikazu, te bi onda morali da izrađujete poseban css. Ovo je izgled koji se dobija primenom ove klase i njenih delova :

Slika rada klase preuzeta sa
https://www.w3schools.com/bootstrap/bootstrap_media_objects.asp

```
</div>
<div class="media-body">
 <h4 class="media-heading">Naslov</h4>
     Neki tekst
 <div class="media">
  <div class="media-left">
   <a href="#">
    <img src=" holder.js/64x64" class="media-object"
    style="width:45px">
   </a>
  </div>
  <div class="media-body">
   <h4 class="media-heading">Naslov</h4>
       Tekst poruke
  </div>
 </div>
</div>
</div>
<div class="media">
 <div class="media-left">
  <a href="#">
   <img src=" holder.js/64x64" class="media-object"
   style="width:45px">
  </a>
 </div>
```

```
      <div class="media-body">
        <h4 class="media-heading">Naslov poruke</h4>
            tekst poruke
      </div>
     </div>
    </div>
  </li>
</ul>
</div>
```

https://react-bootstrap-table.github.io/react-bootstrap-table2/storybook/index.html?selectedKind=Remote&selectedStory=Remote%20Filter&full=0&addons=1&stories=1&panelRight=0&addonPanel=storybook%2Factions%2Factions-panel

Primer upotrebe bootstrap koda

Ovaj primer react koda sa dodatim delom bootstrap koda, gde klikom na dugme se otvara modal za logovanje, ali istovremeno pokazuje da se može pokrenuti i drugi modal koji bi omogućio Singup ili neku drugu mogućnost. Ali, kod između div tagova se može primeniti i u čistom html-kodu gde se upotrebljava samo bootstrap. Pokretanje modala obalja se klikom na dugme

```
<button className="btn-primary" data-toggle="modal"
      data-target="#Mojmodal" >
            Login
</button>
```

To se obavlja upotrebom koda data-toggle="modal" data-target="#Mojmodal" koji obavlja pozivanje modala sa id-om Mojmodal, koji vrši prikazivanje modala, to jest upotrebom data-toggle="modal". Ovaj poslednji deo koda vrši prikazivanje ili sakrivanje div taga koji ima u sebi klasu modal.

```
import React from'react';

export default class Login extends React.Component {
  render() {
    return (
      <div>
        <button className="btn-primary" data-toggle="modal"
          data-target="#Mojmodal" >
          Login
        </button>
        <hr/>
```

```
<div className="modal fade" id="Mojmodal" >
  <div className="modal-dialog">
```

U ovom delu počinje sadržaj sadržaja modala

```
<div className="modal-content">
```

Obzirom da i modl podseća na veb stranicu on ima i delove kao što je heder, body i futer. U kojima se mogu postaviti potrebni elementi. U našem slučaju postavljeno dugme, koje ima ulogu da sakrije modal, dok deo modal-title je da se postavi ime modala gde je dodatna glifikon ikonica

```
<div className="modal-header" style={{padding:"35px 50px"}}>
  <button className="close" data-dismiss="modal">&times;
    </button>
  <h4 className="modal-title">
    <span className="glyphicon glyphicon-lock"/>
    Login
  </h4>
</div>
```

Telo modala, u ovom slučaju postavljena je forma, ali može biti postavljen bilo koji deo veb stranice

```
<div className="modal-body" style={{padding:"40px 50px"}}>
  <form>
    <div className="form-group">
      <label htmlFor="korisnickoime" >
        <span className="glyphicon glyphicon-user"/>
        Korisnicko ime
      </label>
      <input type="text" className="from-control"
      id="korsnickoime" placeholder="Korisnicko ime"/>
    </div>
    <div className="form-group">
      <label htmlFor="Korisnicka sifra" >
        <span className="glyphicon glyphicon-hand-up"/>
        Korisnicka sifra
      </label>
      <input type="text" className="from-control"
        id="korsnickasifra" placeholder="Korisnicka sifra"/>
    </div>
    <div className="checkbox">
      <label >
```

```
                        <input type="checkbox" value="" />
                        Poseti me
                      </label>
                    </div>
                    <button type="submit" className="btn-success btn-block">
                      <span className="glyphicon glyphicon-off"/>
                      Login
                    </button>
                  </form>
                </div>
                <div className="modal-footer">
                  <button type="submit" className="btn-danger pull-left"
                      data-dismiss="modal">
                    <span className="glyphicon glyphicon-remove"/>
                    Otkazi
                  </button>
                  <button type="submit" className="btn-info pull-right"
                        data-target="#singupmodal" >
                    <span className="glyphicon glyphicon-info-sign"/>
                    Singup
                  </button>
                </div>
              </div>
            </div>
          </div>
        </div>
      );
    }
}
```

React - PHP

PHP – React

PHP kao i React su interpreteri, sa dosta sličnosti i razlika. U php kodu se mogu odmah upisivati kodovi budućih veb stranica koje se automacki izvršavaju na apache i drugim serverima bez ikavih dodataka. Pored ovog php ima veliku biblioteku koda, koja pored svega može direktno da radi sa bazama podataka bez ikavih dodataka. Sa druge strane u php kod moguće je dodati i kodove drugih programskih govora, kao i asemblera. Što mu daje ogromnu snagu kada se primeni u veb apliakcijama. Te je jako pogodan za primenu u ovoj kombinaciji izrade veb aplikacija sa Reactom. Da bi ste upoznali mali deo njegovih mogućnosti pogledajte na linku :
https://www.learn-php.org/
ili upotreba php sa progamskim govorom C
http://www.php-cpp.com/

Rad servera

Serveri su isto računari kao i one što imamo kući, na poslu i slično, ali samo na njima postoje programske aplikacije koje su namenjene da osluškuju pozive na koje odgovaraju. Ti pozivi su vezani za određene memorisjke lokacije ili portove. Port je nešto što omogućava da se tu nešto priveže, prikači. Jednostavno ako pogledamo šta je čije videćemo da reč PORT u galija je mesto za vezivanje brodova, te odtuda i ova logika.
Kod njih postoje standardizovani portovi za aplikacije, veb servise i slično. Te portove ne bi trebali da upotrebljavamo van njihovih namena.
Ali zato postoje neki koji su van toga koji su pogodni za komunkacije. Sa druge strane u svakom slučaju potrebno je imati neke podatke promenjive shodno situaciji ili dozvoljenim akcijama korisnika. Osim toga broj korisnika može da se promeni, a svaki od njih ima neke svoje podatke, koje je potrebno promeniti. Pored svega korisnik može da pošalje drugom korisniku poruku, nebinto slika, tekst i dr. U svim ovim i mnogim drugim situacijama potrebno je da ti podaci budu smeštani, nadograđivani i dr. U tim situacajima potreban je server sa bazom podataka, koji će trenutno opslužiti zahtevane promene, koje su dozvoljene veb aplikacijom. Te je tako potrebna aplikacija koja će raditi taj posao na serveru i druga koja bi povezala našu aplikaciju pisanu u React-u. Kako je ract java skript a php je nešto drugo, za razmenu podataka između njih potreban je kod koji razumu oba. To je format podataka JSON. Za react postoje serveri koji su namenjeni java skript kodu: node, express, ngx, ali obzirom da je react nemenjen za izradu korisničkog dela i da je u osnovi java skript moguće ga je povezati i sa php-om i prikazati ga na Apache serverima bez ikavih problema. Jedino je ponekada potrebno uraditi podešavanja na Apache serveru. Što i nije nešto mnogo posla. Uz to na ovim serverima obavezno postoji i mogućnost upotrebe Mysql baze podataka, za koju takođe postoji dosta velika proverena podrška. Tako da sa manje muke u početku dok potpuno se ne ovlada novim tehnologijam najbolji način je upotrebljavati stare.

HTTP zahtevi

Rad servera i programa koji ga čine serverom ovim zahevima upravlja razmenom podatka. Sama razmena podataka se vrši pomoću metoda (koji se uvek pišu velikim slovima u kodu gde se primenjuju): GET, POST, HEAD, DELETE, CONNECT, OPSTIONS, TRACE i PATCH. Najčešće u upotrebi su GET i POST, mada se ređe upotrebljavaju PUT, PATCH i DELETE.

1. GET metod je namenjen zahtevu za preuzimanje podataka sa servera, praktično svaku put kada unesemo netu URL tu vršimo GET zahtev
2. POST metod namenjena za zahtev kojim se šallju podaci ma server
3. PUT metod je sličan POST-u upotrebljava dodavnje podataka u već postojeći objekt
4. DELET brisanje nenkog podatka
5. PATCH za izmenu(Update)

Prilikom slanja zahteva uvek dolazi do povratnih informacija u obliku njihovog statusa. Statusi su podeljeni u pet grupa i prestavljeni su sa trocifrenim brojem i objašnjenjem:

1. 100 – 199 informativni statusi, zbači da je zahtev primeljen i upućen na dalju obradu
2. 200 – 299 zahtev je primljen i uspešno okončan
3. 300 – 399 redirekcija. Praktično poslat zahtev nemože biti obrađen na ovom serveru pa se onda ovim zahtevom preusmerava na drugi server ili ULR
4. 400 – 499 povratno saopštavanje od pogrešnim upućenim zahtevom (najčešće ugledamo 404)
5. 500 – prikazuje pojavu greške na serveru jer on nije mogao iz nekih razloga da obradi zahtev

Pored toga tu je Content Type polje, kojim se saopštava koji se tip podataka šalje nazad i u kom formatu. Svakim poslatim HTTP GET dobijamo Content Type u obliku text/html. Pored ovog tipa podataka najčešće se upotrebljava i application/json.

CORS ili preuzimanje resursa sa različitih izvora(PRRI)

Obzirom da mnogi pokušavaju da čine loše kako bi izlečili svoju sujetu CROS je namenjena je da se zaštiti i osigura rad sajta na serveru. Tačnije, prilikom preuzimanja resusrsa sa različitih izvora (Cross-origin resource sharing - CORS) mehanizam omogućava da se resursi koji nisu dostupni (kao što su fontovi, java skriptovi i slično), a koji se nalaze na veb stranicama, a zahtevaju se od drugog domena koji je van onog iz kog je zahtev za resursom potekao, da ili obave ili zabrane. Princip rada je jednostavan. Znači možete pristupati bez ikavih problema kada je u pitanju ovakav pristup:

1. nekiDomen.com prema nekiDomen.com/podaci
2. nekiDomen.com prema nekiDomen.com/podaci?stranica=3
3. nekiDomen.com prema users:pass@nekiDomen.com/podaci

Ali se nemože pritupati

4. nekiDomen.com prema nekiDomen.com:5000 različitim portovima na istom domenu
5. nekiDomen.com prema https://nekiDomen.com različitim protokolima
6. nekiDomen.com prema nekiDrugiDomen.com različitim hostovima

Za rad sa githubom pogledati na ovom linku https://developer.github.com/v3/
To bi značilo da se samo ono što je dopušetno u aplikaciji može i ugraditi na veb sajt, odnosno kopirati na host. Međutim, da bi se mogli dodati podaci putem raznih formi i slično upotrebom na primer AJAX(XML.HttpRequest) i sl. su tradicionalno ograničeni na traženje dozvola za pristup iz razloga bezbednosti. Obzirom da time mogu da se upute i podaci kojim bi se ugrozio rad servera. Kako na istom domenu, tako i na pristup sa različitih domena, pogotovu ako su to zahtevi (POST, PUT i DELETE, kao i druge vrste HTTP zahteva, uključujući i specifična HTTP zaglavlja). Što se može desiti i priliko razvoja pojava greške CROS orgine not allowed.

Kako PRRI (CROS) funkcioniše

Njegova funkcija je da opiše način na koji će da se obavi kominikacija između pretraživača i servera, pri čemu određuje da li je bezbedno da se dozvole zahtevi sa nekim sadržajem sa drugih domena. Ovaj vid dozvoljava više slobode i funkcionalnosti nego zahtevi sa izvora istog porekla, ali je mnogo bezbedniji nego kad bi se dopustili svi vidovi razmene između domena.
PRRI (CROS) standard opisuje nova HTTP zaglavlja koja omogućuju pregledačima i serverima da zahtevaju udaljene veb adreese, samo kada imaju dozvolu pristupa. Iako neke od potvrda i autorizacija mogu biti izvedena od strane servera, uglavnom se od pretraživača očekuje da podržava ta zaglavlja i restrikcije koje oni donose. U zaglavljima se uglavnom govori kog su tipa podaci koji se razmenjuju. Za AJAX, fetch i HTTP metode koje mogu da izmene informacije, specifikacija nalaže da sami pretraživači „prelete" zahtev, zahtevajući podržane metode od servera sa HTTP OPTIONS zaglavljem za zahteve, a potom, po potvrdi od strane servera, šalju stvarne zahteve sa stvarnim HTTP metodom za zahteve. Serveri takođe mogu da obaveste klijenta da li bi "potvrde" (uključujući kolačiće i HTTP podatke za verifikaciju) trebalo da budu poslate zajedno sa zahtevima.

Jednostavan primer

Osnovni PRRI se sastoji iz dva koraka:

Pregledač šalje zahtev sa Origin HTTP zaglavljem. Dato zaglavlje sadrži domen koji je opslužio roditeljsku stranicu. Kada stranica sa http://www.neki.com pokuša da pristupi korisničkim podacima na bar.com, sledeći zahtev se šalje ka bar.com:

Origin: http://www.neki.com
Server može da odgovori sa :
Access-Control-Allow-Origin zaglavljem čime stavlja do znanja koje veb stranice su dostupne. Na primer:
Access-Control-Allow-Origin: http://www.neki.com
Stranicom za grešku
Access-Control-Allow-Origin zaglavljem sa naznakom da dozvoljava sve domene:
Access-Control-Allow-Origin: *

Napomena: Za više o CROS na veb sajtu: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
Ovo generalno nije odgovarajući način kada se koristi polisa zajedničkog porekla. Jedini slučaj kada jeste je kada se radi o nekom domenu koji se smatra da je svima dostupan kao što je na primer Google fonts.
Sa druge strane, navedeni primer komunikacije je prikladan ako se koristi model bezbednosti koji se na engleskom naziva object-capability model gde stranice imaju sakriven URL i jedino im mogu pristupiti oni koji imaju ključ.
Primetimo da u PRRI arhitekturi, ACAO zaglavlje postavlja spoljna veb strana (bar.com), a ne originalna strana (foo.com). PRRI omogućava spoljnim veb servisima da daju autorizaciju veb aplikacijama da koriste njihove servise i ne kontroliše spoljne servise kojima pristupaju veb aplikacije.

Primer drugi

Moderni pregledači koju podržavaju PRRI pri radu sa AJAX zahtevima ka različitim izvorima će dodati „preflight" zahtev da utvrde da li imaju potrebna ovlašćenja.

OPTIONS /
Host: primer.com
Origin: http://neki.com
Ako primer.com odobrava zahtev, onda će odgovoriti sa sledećim zaglavljima:
Access-Control-Allow-Origin: http://neki.com
Access-Control-Allow-Methods: PUT, DELETE

Zaglavlja

HTTP zaglavlja koju su vezana za PRRI (CROS) su:

Zaglavlja za zahteve

Origin
Access-Control-Request-Method
Access-Control-Request-Headers

Zaglavlja za odgovore

Access-Control-Allow-Origin

Access-Control-Allow-Credentials
Access-Control-Expose-Headers
Access-Control-Max-Age
Access-Control-Allow-Methods
Access-Control-Allow-Headers

U primerima koji su zastupljeni u ovoj knjizi, a namenjeni su kao osnova i ideja da bi se shvatio i razumeo princip rada kompletnog skolopa node-react-bootstrap-php, često će te ugledati kod koji korisniku omogućava komunikaciju sa serverom putem koda:

```
fetch('http://localhost/fetchReact/server/prijem.php',{
   method: 'POST',
   headers: {
      Accept: 'application/json',
      'Content-Type':'application/json'
   },
   body: JSON.stringify({
      content: 'upis'
   }),
})
```

Što označava da se prihvataju json aplikacije i način formatiranja podataka između pretraživača i servera. Dok u serverskom delu postavio sam:

```
<?php
header('Accept: application/json');
header('Access-Control-Allow-Methods: POST,GET,PUT,DELETE,OPTIONS');
header("Access-Control-Allow-Headers: eToken,X-Requested-With,
          Content-Type");
header('Content-type: text/json; charset=utf-8;
           application/x-www-form-urlencoded');
header ( 'Access-Control-Allow-Origin: *');
```

dosta toga, zato što je ovo učenje i sagledavanje kompletne slike, što naravno u praksi primenićete prema potrebama, jer ovako da sve postoji u failu na serveru je ne bezbedno. Te samim time na serverskom delu potreban je jedan fail koji bi prihvatao zahteve react aplikacije i onda shodno zahtevu upućivao na obradu skriptama koje bi obavljale obradu zahteva i vraćale rezultat rada react aplikaciji. Koja bi ih korisniku prikazivalau svojoj render metodi.

Zašto povezivanje sa php-om

Razloga ima dosta, najbitniji su:

1. velika biblioteka koda

2. direktan rad sa bazama podataka
3. stabilan sistem
4. radi sa JSON formatom datoteka

Ovo je nešto najkraće na odgovor zašto baš php, a ne nešto drugo, a osim toga php je serverski orjentisan. Za njegov rad potreban je server, dok react je uglavnom bez drugih dodataka korisničiki deo. Tako da je to idelan spoj. Bilo kako bilo da bi ste mogli da pošaljete email ili post na blogu bilo bi nemoguće bez podrške servera. Ako bi se upotrebilo okruženje Node-React za rad na serveru bio bi potreban dodatak ekspres i još po nešto od dodataka da bi aplikacija radila. Što je malo kompleksno za početak. Ovo nije napisano da bih umanjio veliki rad mnogih, već da je bolje krenuti sa nečim što je manje složeno i što je dugo u primeni. Verujem da je veliki broj vas koji će te pročitati ovu knjigu već imao susreta i rada sa PHP-om, mysqli, pyton, pear i sličnima u svetu veba, te će vam objašnje pristupa biti jedino potrebno i da će te veoma brzo i lako krenuti u svet stvaranja.

Kako povezati React i PHP

Pokušaću da upotrebljavam naše reči svuda gde je moguće pa i ovde. Princip rada je osluškivanje slanja i prijema signala na serverskom računaru, jer programi na njima tako su i pravljeni da rade. Te u ovom slučaju prečesto je u upotrebi strana reč API. Što bi se moglo prevesti kao programsko prilagodjavanje raznorodnih sistema. A radi se o jednom direktorijumu ili folderu gde bi se smestio jedan ili više failova koji bi omogućavali tu povezanost. U našem slučaju između React-a i PHP-a. Kodu u tom failu ima zadatak da omogući razmenu podataka i u našem slučaju biće napisan u PHP-u. Da li ćete ga nazvati index.php ili drugačije to je nebitno. Bitno je da u react aplikaciji postavite link prema njemu. A u njemu postoji kod koji će prihvatati zahteve i podatke iz React aplikacije i da će potrebne podatke sa serverske strane prosleđivati po zahtevu react aplikaciji. Da li će te taj folder gde se nazvati fail ili failovi nazvati API ili server i to je nebitno. U mom slučaju taj link bi za povezivanje izgledao ovako:

http://localhost/fetchReact/server/demo.php

Umesto htp://localhost može biti domen veb prostora sa kojim se radi
http://nekiDome.com/fetchReact/server/demo.php
U našem slučaju kompletna aplikacija i react i PHP je u foldedru fetchReact pošto smo na locahostu, ali kada smo na serveru gde je smeštena react aplikacija tada ćemo imati samo react u root folderu gde se nalazi folder server za serverski deo aplikacije.

http://nekiDomen.com/server/demo.php

Od strane PHP-a najbolje je za zahteve GET, POST upotrebiti, ali mogu biti po potrebi upotrebi i ostali. Ali, prihvatanje tih zahteva možemo obraditi na ovaj način koji prikazuje kod niže:

```
$ime-vraijabile-u-koju-smeštamo-prijemne-zahteve =
json_decode(file_get_contents('php://input'), true);
```

kod

```
$content = json_decode(file_get_contents('php://input'), true);
```

Ovako napisano ima značenje varijabili $content dajemo vrednosti niza podataka, koje je file_get_contents prihvatila kodom ('php://input') i odmah poslala istinu da je prihvatila te poslate podatke.
Iz koje kasnije obrađujemo prihvaćene podatke, jer json_decode od njih formira niz podataka. Ovim kodom možemo dodati poruku koju šalje Response da su podaci uspešno primljeni, poslati primljeni.

```
$response = array( "success"=> true, "message"=>"
                poruka je uspesno poslata");
$output = array_slice($response, 1,2);

echo json_encode($output);// kod za povratak odgovora react aplikaciji
```

Ili postavljenjem if uslova javiti da ima nekih problema. U svakom slučaju biće potrebno da react aplikaciji ovaj poslat kodiran niz podataka na neki način dekodiramo i izdvojimo onaj potreban deo iz niza da bi se on prikazao onamo gde nam je to potrebno, kako bi smo o tome obavestili korisnika aplikacije.

```
  fetch('http://localhost/fetchReact/server/demo.php', {
        .then( res=> res.json())
        .then( responseData => {
          console.log(responseData);
          alert(responseData.message); // vraćeni podaci sa servera
          });
  }
```

Ovim smo u klasičnom java skript alertu ispisali

Dok, povratne podatke iz mysqli baze podataka možemo dobiti na ovaj način

```
<?php
$connect =mysqli_connect("localhost", "root"," ", "react-baza");
$result = mysqli_query($connect, $sql);
$sql = "SELECT * FROM react_data";
$json_array();
if (mysqli_num_rows > 0) { // ispitivanje da li ima podataka u bazi
  while($row = mysqli_fetch_assoc($result)){
```

```
    $json_array[] = $row;

// $json_array['users'] []  = $row; pristup nizu users koji u osnovnom nizu poseban niz
//     { “users”:[{‘jedan’}.{‘dva’}],}
    }
}
/* za prikaz i proveru na istoj stranici upotrebljava se ovaj kod

echo'<pre>';
   print_r($json_array);
echo'</pre>';
*/
// Pošto je potrebno podatke vratiti react aplikaciji potrebno je upotrebiti donji kod
echo json_encode($json_array);

// na kraju rada sa bazom potrebno je zatvoriti
mysqli_close($connection);
```

Slanje podataka iz React aplikacije Php skriptu

Postavlja se pitanje „Zašto“?? Sam React je nesto što je namenjeno radu sa korisnikom, te nema mogućnosti da radi serverski deo poslova, gde se obavljaju razlicite potrebe. Na primer: slanje emaila, rad sa bazom podataka i slicno. Te, je u te i slične svrhe potrebno upotrebiti serverske radnje. Za rad iz Reacta potrebno je dodati još drugih dodataka, koji kao i kod svake nove tehnologije traže još dosta drugih i podešavanja i plaginova. Pa, je u tu svrhu dosta dobro upotrebiti već poznato : PHP i MYSQL kombinaciju.
Obzirom da PHP ima već veliku podršku i brojne sktiptove za rad sa podacima i ostalim potrebama veb dizajna koji se odvijaju na serveru.

Šta nam je potrebno

Da bi smo povezali react i php/mysql potrebno je naravno napraviti u jednom projektnom folderu tokom izrade i razvoja aplikacije dva flodera: jedan za React aplikaciju drugi za PHP. Obzirom da je za ovo neophodno instalirati React aplikaciju potrebno je prvo pristupiti Folderu gde će se ona instalirati nakon čega pokrenuti

```
create-react-app naziv_aplikacije
```

ili prekopirati tekst ovog dole koda u taj folder i pokrenuti
Package.json file

```
{
  "name": "fetch-test",
```

```
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "react": "^16.4.2",
    "react-dom": "^16.4.2",
    "react-scripts": "1.1.4",
    "whatwg-fetch": "^2.0.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test --env=jsdom",
    "eject": "react-scripts eject"
  },
  "proxy": http://localhost: }
"proxy": http://localhost:80  // dopustanje komunikacije react aplikacije sa phpom
                              // na portu 80
```

npm install

kako bi se proces malo automatizovao. Ali pre toga ako se odlučimo za drugi postupak u istom folderu potrebno je napraviti i foldere: public i src. U public folderu napraviti fail index.html sa ovim kodom:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,
        shrink-to-fit=no">
    <meta name="theme-color" content="#000000">
     <!—ova dva reda ispod nisu obavezna- →
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json">
    <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico">
    <title>Naziv kako vam to odgovara u aplikaciji</title>
  </head>
  <body>
    <noscript>
      You need to enable JavaScript to run this app.
    </noscript>
    <div id="root"></div>
  </body>
</html>
```

Napomena: ako upotrebljavate Bootstrap unesite i CDN linkove prema njegovim failovima u index.html

Dok, u src folderu su potrebna dva faila: App.js i index.js , sa ovim kodom:

Fail App.js

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

import Slanje from './react-php';

class App extends Component {

  render() {
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h1 className="App-title">Welcome to React</h1>
        </header>
        <div className="App-intro">
      </div>
          <Slanje/>
      </div>
    );
  }
}

export default App;
```

Fail index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';

ReactDOM.render(<App />, document.getElementById('root'));
```

Napomena: Da bi ste imali neki raspored na stranici potrebno je napraviti i css failove za sve failove.

Pošto smo napravili traženo i pokrenuli npm install i time prektično napravili u folderu za react sve potrebno da bi se na ekranu u brovseru video osnovni izgled react aplikacije, vratićemo se korak unazad. U folderu aplikacije napraviti folder za php pod nazivom server, kako bi smo imali oznaku da su tu sve serverske skripte. I u njemu možemo napraviti fail index.php ili kao što sam i nazvao u aplikaciji demo.php. sa ovim kodom:

```
<?php
/**
 * Created by IntelliJ IDEA. demo.php
 * User: trika
 * Date: 27-Aug-18
 * Time: 16:05
Kod sa hederima omogućava dozvolu za različite pristupe serveru i radu sa
podacima , ali ovo je ne bezbedno pa će te vi u vašim slučajevima postaviti samo
potrebno
 */
header('Accept: application/json');
header('Access-Control-Allow-Methods: POST,GET,PUT,DELETE,OPTIONS');
header("Access-Control-Allow-Headers: eToken,X-Requested-With,
        Content-Type");
header('Content-type: text/json; charset=utf-8;
         application/x-www-form-urlencoded');
header ( 'Access-Control-Allow-Origin: *');

// osluškivanje i prihvatanje podataka na serveru koji se dekodiraju i smeštaju u
// promenjivu
$content = json_decode(file_get_contents('php://input'), true);

// povratna poruka koja se šaljle ract aplikaciji  o uspešnosti prijema podataka
$response = array( "success"=> true, "message"=>"
              poruka je uspesno poslata");
$output = array_slice($response, 1,2);

// upotrebom donjeg koda
echo json_encode($output);
```

Veoma je pogodan ovaj deo koda čija je namena da izdvoji delove niza podataka iz nekog niza

```
$response = array( "success"=> true, "message"=>"
              poruka je uspesno poslata");
$output = array_slice($response, 1,2);
```

Varijabila se proglašava nizom podataka gde se putem array_slice vrši izdvajanje sadržaja iz drugog kljuca koji se predaje varijabili $output.
Ovu vrednost varijabile $output možete proslediti kroz json_encode nazad react aplikaciji, koja to dobija kroz response.message ako je u pitanju alert.
U istom folderu potrebno je napraviti još jedan fail, kako nam react ne bi javljao gresku CROS orgin i time upozoravao na probleme dozvole pristupa serveru.

Fail .htaccess

```
<ifModule mod_rewrite.c>
   Header add Access-Control-Allow-Origin: "*"
   Header add Access-Control-Allow-Methods:
            "POST,GET,PUT,DELETE,OPTIONS"
   Header add Content-type: "text/json; charset=utf-8;
            application/json;
            application/x-www-form-urlencoded"
   Header add Accept: "application/json"
   Header add Access-Control-Allow-Headers: "eToken,X-Requested-With,
            Content-Type"
   RewriteBase on
   RewriteBase/
</ifModule>
```

i dodati deo koda u htacess failu, prilikom izrade virtualnog hosta na localhost-u vašeg računara

```
<VirtualHost *:80>
        ServerName React
         ServerAlias React
        DocumentRoot "e:/wamp64/www/projekt-react"
        <Directory  "e:/wamp64/www/projekt-react/">
                Options +Indexes +Includes +FollowSymLinks +MultiViews
                AllowOverride All
                Require  all granted
        </Directory>
</VirtualHost>

# END WordPress
```

Pošto smo obavili pripremu za rad prelazimo na izradu react skripta koji će da radi za sada slanje jednog podataka koji će da php prihvati i u konzoli vrati razultat izvršenosti tog rada. To je fail kog sam nazvao react-php.js kod je dole niže:

Fail react-php.js

```
import React from 'react';
import 'whatwg-fetch';

export default class ReactPhp extends React.Component{

   constructor(props){
      super(props);

      this.getPHP = this.getPHP.bind(this);
   }

   getPHP ()
   {
      fetch('http://localhost/fetchReact/server/demo.php',{
         method: 'POST',
         headers: {
            Accept: 'application/json',
            'Content-Type':'application/json'
         },
         body: JSON.stringify({
            content: ' test1'
         }),

      })
         .then( res=> res.json())
         .then( response => {
           console.log(response);
         });
   }

   render(){

      return(
         <div>
            <p>
               React
            </p>
            <br/>
            <button
            onClick={this.getPHP}
            >
              Ucitaj
            </button>
         </div>
```

```
        );
    }
}
```

Napomena: da bi ste poslali više podataka u failu react-php.js porebno je dodati umesto ovog koda

```
fetch('http://localhost/fetchReact/server/demo.php',{
          method: 'POST',
          headers: {
             Accept: 'application/json',
             'Content-Type':'application/json'
          },
          body: JSON.stringify({
             content: ' test1'
          }),

       })
```

Ovaj kod

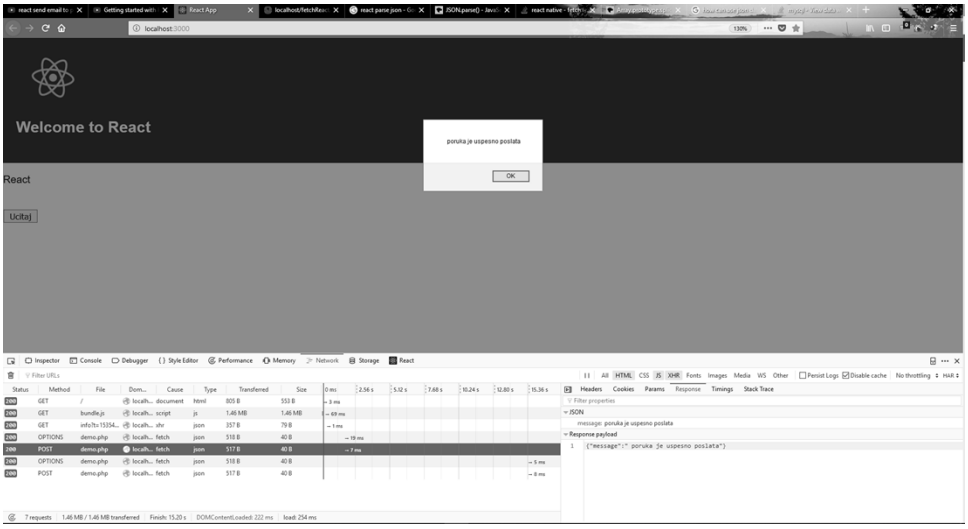
```
fetch('http://localhost/fetchReact/server/demo.php',{
    method: 'POST',
    headers: {
       Accept: 'application/json',
       'Content-Type':'application/json'
    },
    body: JSON.stringify({
       ime: 'prvi',
       tema: 'prvi',
       email: 'prvi',
       mesto: 'prvi',
       poruka: 'prvi',
    }),

})
```

time će biti dosta lakše da se snađete prilikom upisa u podataka PHP-om u mysqli bazu podataka. Dodavanjem ovog koda u delu faila react-php.js, dobija se poruka u alertu koja označava da je radnja uspešno izvršena

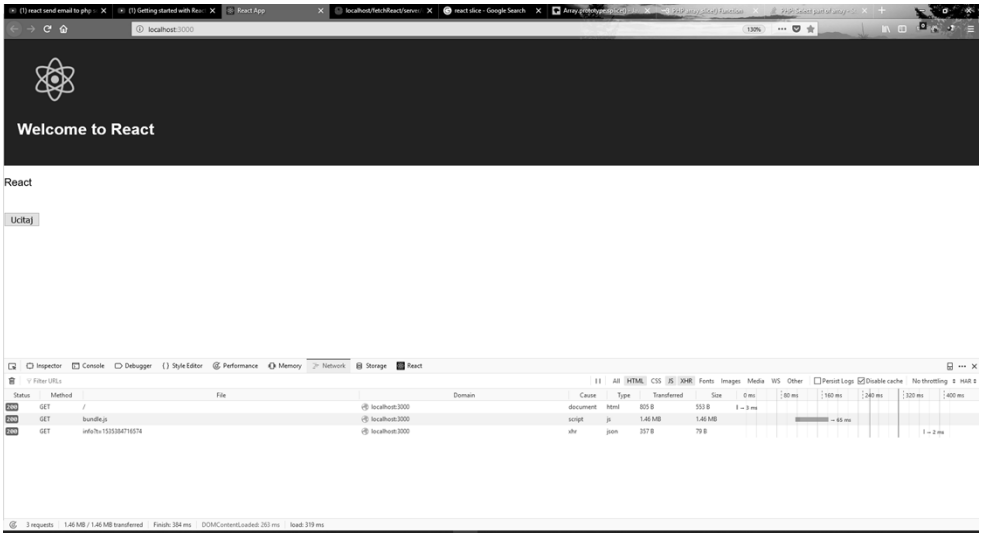
```
  })
       .then( res=> res.json())
       .then( responseData => {
```

```
        console.log(responseData);
        alert(responseData.message);
        });
}
```

Kao sto i pokazuje ova slika ispod.



Što je veoma upotrebljivo kod slanja podataka kao što je email poruka ili neka druga radnja koja zahteva potvrdu uspešnosti ili povratak opisa nastale greške. Slika posle instalacije, izrade potrebnih failova, prelaska u folder react aplikacije i starta upotrebom npm starat, gde se vidi u developer okruženju u Mozili



Kao što vidite, pošto sam upotrebio prvi način izrade aplikacije zadržan je i standardni izgled koji se postiže upotrebom naredbe creat-react-app, iz čega sam ručno izvršio instalaciju paketa

"whatwg-fetch": "^2.0.4"

uporebom naredbe npm install - - save whatwg-fetch, kako bi se izvršila instalacija u delu package.json dependencies.

Napomena: prilikom instalacije potrebnih paketa u reactu postoje dve potrebe: jedna je razvojni deo (devdependencies) npm install - - save-dev. Možete upotrebiti (i) skraćeno pisanje za npm install pisanjem npm i . Da bi radilo ovo sve potrebno ga je postaviti u folder WWW wampa i slično u zavisnosti koji program upotrebljave na svom računaru.
Da bi ste videli šta piše i kako izgleda php stranica potrebno je pokrenuti wamp ili slično, jer za rad php-a potrebno je imati instalisan wamp ili sl. gde ima Apache servera, mada se može upotrebiti i mikrosoftov IIL server.
U mom slučaju putanja do pomenitog faila je:
http://localhost/fetchReact/server/demo.php
Što se i vidi na slici dole ispod :

Redovi koda u php failu

$response = array("success"=> true, "message"=>" poruka je uspesno poslata");
$output = array_slice($response, 1,2);

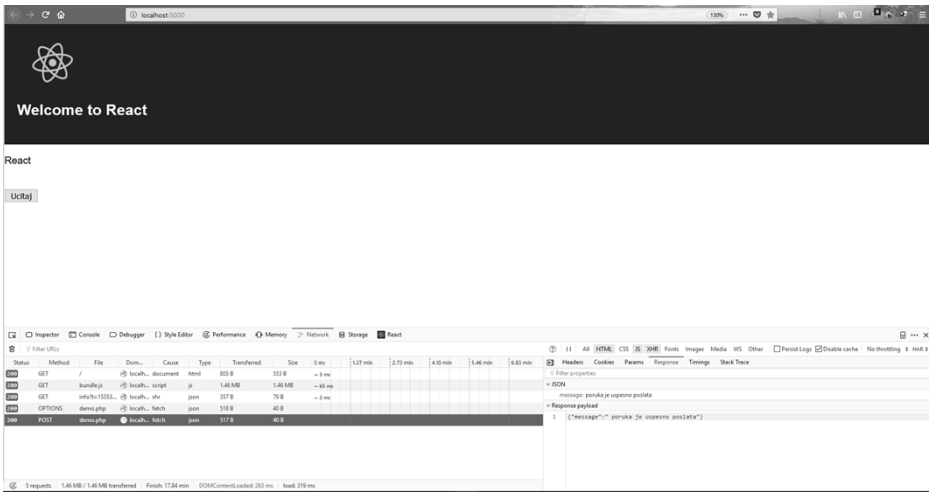
Postiže se izdvajanje dela promenjive $response u promenjivoj $output čiji se rezultat kasnije vraća upotrebom

echo json_encode($output);

u react aplikaciju koja je poslala podatke na obradu PHP skriptu. Što se prikazuje u alertu ovim kodom

alert(responseData.message);

Zašto se ovo uradilo? Promenjivoj $response dodali smo za njenu vrednost niz, koji sadrži dva elementa success i message. Nama je potrebno da znamo u ovom slučaju vrednost ključa message, te sam zato i upotrebio kod u alertu. responseData je vrednost celokupnog JSON objekta koji je php vratio, i kao takvog je veoma teško ga prikazati. Upotrebom responseData.message se postavilo da se prikaže samo sadržaj promenjive message. Cilj toga je da možete steći sliku istovetnosti u sličnim prilikama kako u upotrebi react-a tako i php. Tako da će u alertu biti prikazan sadržaj promenjive message iz php-a.
Posle klika na dugme u prozoru sa react aplikacijom sa pokrenutim razvojnim okuženjem za praćenje rada aplikacija, pritiskom na F-12 dobijamo sledeći prikaz:



Gde klikom na dugme Prameters možemo videti u ovom selektovanom slučaju ono što smo poslali iz react aplikacije.

content: test1

Kao i ono što smo zahtevali da se vrati u react o uspešnosti izvršenog zadatog posla kada kliknemo na dugme Console:

Object { message: " poruka je uspesno poslata" }

Slanje podataka iz React-a u php script Izrada Json faila php-om

Kao što će te videti prodhodni postupak se ponavalja, samo što u ovom slučaju umesto poslednjeg react faila izradjujemo fail Slanje.js, kojim ćemo poslati malo više podataka preko php skripta. Nakon prihvatanja i obrade podataka php skript će od primljenih podataka napraviti json fail, kao jednu manju bazu podataka. Čiji sadržaj će zavisiti od toga šta prosledimo react aplikacijom.

Slanje.js file

```
import React from 'react';
import 'whatwg-fetch';

class DemoComponent extends React.Component {
   constructor() {
      super();
      this.state = {
         users: {},
         errors: {},
      };
      this.handleChange = this.handleChange.bind(this);
      this.handleSubmit = this.handleSubmit.bind(this);
   }

   handleChange(event) {
      let users = this.state.users;
      let field = event.target.name;
      users[field] = event.target.value;
      this.setState({ users });
   }

   handleSubmit(event) {
      event.preventDefault();
      let dataURL = "http://localhost:80/fetchReact/server/test.php";
      let h = new Headers();
      let req = new Request( dataURL,{
         headers: h,
         method: 'POST',
         body: JSON.stringify({users: this.state.users})
      });
      console.log(this.state.users,"users");
      fetch(req)
         .then((response)=>{
         if(response.ok){
```

```
            return response.json();

        }else{
            throw new Error("losa konekcija");
        }
        })
        .then((users) =>{

            alert("forma je poslata");
        })
        .catch( (err) =>{
            console.log('Error', err.message);
        });
}
render() {
    return (
        <div>
            <form onSubmit={this.handleSubmit.bind(this)}>
              <label htmlFor="name">Username </label>
                 <input name="name" type="text"
                       onChange={this.handleChange}
                       value={this.state.users["name"]} />
              <br/>
              <label htmlFor="password">Password</label>
                 <input name="password" type="password"
                       onChange={this.handleChange}
                       value={this.state.users["password"]} />
              <br/>
              <label  htmlFor="age"> Age </label>
                 <input name="age" type="text"
                       onChange={this.handleChange}
                       value={this.state.users["age"]} />
              <br/>
              <label  htmlFor="email">Email </label>
                 <input name="email" type="text"
                       onChange={this.handleChange}
                       value={this.state.users["email"]} />
              <br/>
              <label  htmlFor="website">Website</label>
                 <input name="website" type="text"
                       onChange={this.handleChange}
                       value={this.state.users["website"]} />
              <br/>
              <input type="submit" value="Slanje" />
```

```
            </form>
          </div>
        );
    }
}

export default DemoComponent;
```

test.php file

```
<?php
header('Accept: application/json');
header('Access-Control-Allow-Methods: POST,GET,PUT,DELETE,OPTIONS');
header("Access-Control-Allow-Headers: eToken,X-Requested-With,
        Content-Type");
header('Content-type: text/json; charset=utf-8;
         application/x-www-form-urlencoded');
header ( 'Access-Control-Allow-Origin: *');
// ako zelimo procitati sadrzaj faila posts.json možemo upotrebiti php kod
// readfile('./posts.json');

$podaci = json_decode(file_get_contents('php://input'), true);
$podaci['primljeno'] = true;

// vrati resultat da se snimnjeno
echo json_encode($podaci);

// upis rezultata u json fail
$primljeno= json_encode($podaci);
file_put_contents('posts.json', $primljeno);
```

Poslednja dva reda koda php faila obavljaju sledeći posao, prvi red čita i priprema predhodne podatke koji su već na raspolaganju u odgovarajućem formatu, nakon čega se u sledećem redu vrši dodavanje istih u postojeći JSON fail. Tako što ga file_put_content funkcija dodaje kao novi podatak postojećima.

Organizacija failova

```
Project folder
        React-app folder
                Public folder
                  Index.html

                Src folder
                     App.js
                     Index.js
```

Package.json

Server folder

Test.php

Posts.json

Princip rada

Pripreme

Postaviti kompletnu aplikaciju (Project folder) sa svim delovima u www folder Wamp-a u windowsu ili u httpdocs u slucaju Xamp-a i sl. Pokrenuti konzolu ili ako imate neki bolji editor(InteliJ idea ili storm) da bi se uradilo sledece:

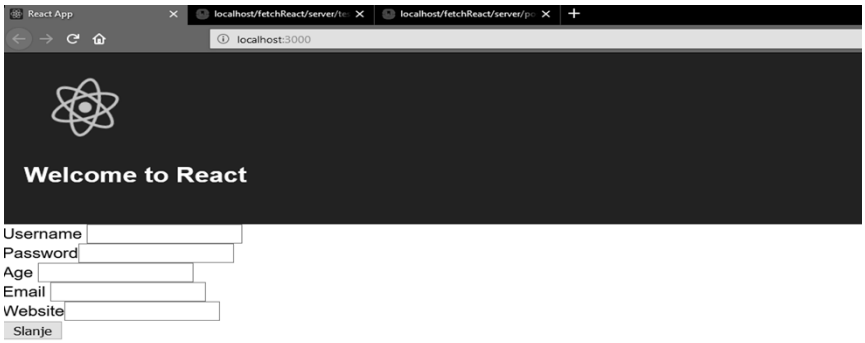
Preći iz Project foldera u folder React-app(cd react-app)

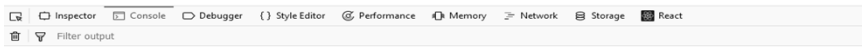
U konzoli ili editoru uneti

Npm i

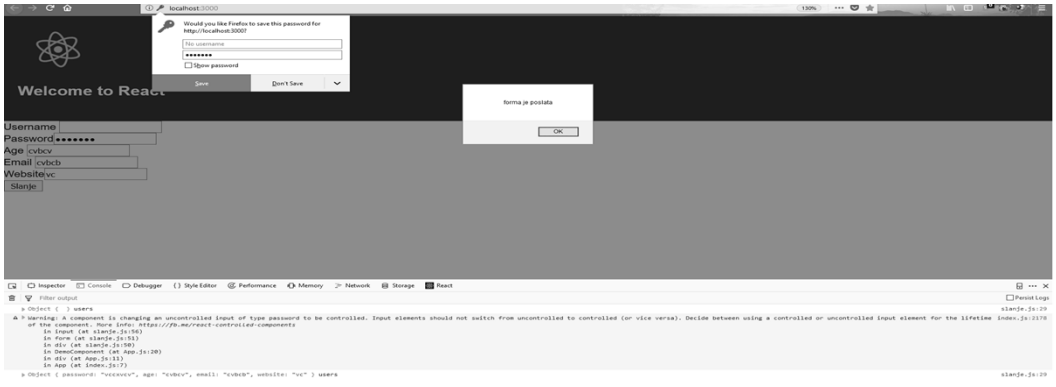
Da bi se izvršila instalacija neophodnih modula koje možete videti u failu package.json.

U medjuvremenu pokrenuti WAMP ili sta vec imate, jer je za rad php-a potreban Apache server. Po izvrsenoj instalaciji u konzoli ili editoru startovati aplikaciju sa Npm start





Izgled koji dobijate na ekranu brozvsera gde unosite sve ili ono sto želite jer nije dodata validacija



Upisano u donjem redu

Object { password: "vccxvcv", age: "cvbcv", email: "cvbcb", website: "vc" }users
Moze se videti u ciljnom posts.json failu

Dodavanje novih podataka

Ozirom da smo u predhodnom slučaju dodali poslate podatke iz react aplikacije php failu koji ih je upisao u JSON fail, takva operacija se ponavalja čim stignu novi podaci. Ali, u predhodnom kodu ti novi podaci će prepisti se preko postojecih, dok nama je potrebno da sačuvamo predhodne i njima dodamo nove. To se postiže na sledeći način:

1. prihvatamo podatke na način kao u kodu ispred
   $podaci = json_decode(file_get_contents('php://input'), true);
2. formiramo odgovor o uspešnosti prijema podataka
   $podaci['primlenjeno'] = true;
3. šaljemo odgovor react aplikaciji
   echo json_encode($podaci);
4. otvaramo potreban json fail kome dodajemo podatke
   $current_data= file_get_contents('posts.json');
5. dekodiramo podatke iz učitanog faila i smeštamo ih u promenjivu
   $array_data = json_decode($current_data, true);
6. Dobijenu promenjivu u kodu gore naglašavmo da je ona niz podataka i koja je jednaka vrednostima prihvaćenih podataka poslatih iz react aplikacije
   $array_data[]= $podaci;
7. Ovako dobijene podatke potrebno je kodirati i smestiti u promenjivu za sledeći korak
   $data_proccesed = json_encode($array_data, JSON_PRETTY_PRINT);
8. Krajnji korak je upis podataka u već otvoreni JSON fail
   file_put_contents('posts.json', $data_proccesed);

U svakom slučaju da bi se izbegle nepravilnosti u radu potrebno je na početku php faila dodati vrednosti za hedere i to na samom početku php faila, ovako

```
header('Accept: application/json');
header('Access-Control-Allow-Methods: POST,GET,PUT,DELETE,OPTIONS');
header("Access-Control-Allow-Headers: eToken,X-Requested-With,Content-
Type");
header('Content-type: text/json; charset=utf-8; application/x-www-form-
urlencoded');
header ( 'Access-Control-Allow-Origin: *');
```

Da li će vam biti potrebne sve ove vrednosti neznam, jer to zavisi od vaših namera, ali je bolje u početku ih dodati sve, a kasnije u toku učenja i eksperimentisanja i sami će te vremenom to utvrditi.
U slučaju izostanka ovih vrednosti javlja se greška o CROS orgin-u, to jest nećete imati dozvolu za upis podataka kao i pristup serveru. Tačnije, React biće onemogućen da šalje podatke php skriptu (error 'Access-Control-Allow-Origin') jer se traži da zahtev se obradu na drugom portu servera.

Fail test.php

```
<?php
        header('Accept: application/json');
        header('Access-Control-Allow-Methods:
        POST,GET,PUT,DELETE,OPTIONS');
        header("Access-Control-Allow-Headers: eToken,X-Requested-
        With,Content-Type");
        header('Content-type: text/json; charset=utf-8; application/x-www-form-
        urlencoded');
        header ( 'Access-Control-Allow-Origin: *');

                $podaci = json_decode(file_get_contents('php://input'), true);
                $podaci['primljeno'] = true;
                echo json_encode($podaci);
                $current_data= file_get_contents('posts.json');
                $array_data = json_decode($current_data, true);
                $array_data[]= $podaci;
                $data_proccesed = json_encode($array_data,
        JSON_PRETTY_PRINT);
                file_put_contents('posts.json', $data_proccesed );
? >
```

Čitanje podataka iz JSON faila

Najjednostavniji način da bi se podaci pročitali iz nekog faila upotrebom php-a je ovaj kod dole niže

Fail citajPodatke.php

```
<?php
        header('Accept: application/json');
        header('Access-Control-Allow-Methods:
        POST,GET,PUT,DELETE,OPTIONS');
        header("Access-Control-Allow-Headers: eToken,X-Requested-
        With,Content-Type");
        header('Content-type: text/json; charset=utf-8; application/x-www-form-
        urlencoded');
        header ( 'Access-Control-Allow-Origin: *');

        readfile('./test.json');
? >
```

Ovako se učitava kompletan JSON fail koji se kasnije obrađuje u react aplikaciji. Ali, umesto ovog koda

```
readfile('./test.json');
```

Čitanje podataka iz JSON faila i prikaz na php stranici

```
header('Accept: application/json');
header('Access-Control-Allow-Methods: POST,GET,PUT,DELETE,OPTIONS');
header("Access-Control-Allow-Headers: eToken,X-Requested-With,
        Content-Type");
header('Content-type: text/json; charset=utf-8;
       application/x-www-form-urlencoded');
header ( 'Access-Control-Allow-Origin: *');

$podaci = json_decode(file_get_contents('php://input'), true);
$podaci['primljeno'] = true;

// vrati resultat da se snimljeno
echo json_encode($podaci);

//otvaranje i čitanje postojećeg json faila
$current_data= file_get_contents('posts.json');

// dekodiranje –pretvaranje u niz podataka
$array_data =json_decode($current_data, true);

// priključivanje niza primljenih podataka iz react aplikacije
$array_data[]= $podaci;

// upis rezultata u json fail
$data_proccesed = json_encode($array_data, JSON_PRETTY_PRINT);
```

file_put_contents('posts.json', $data_proccesed);// dodavanje podataka u postojeći niz podataka

Dole niže je kod kojim se mogu prikazati podaci kada je slučaj upisanih podataka u json failu ovakvog izgleda:

```
{
    "1": {
         "users": {
        "name": "Miodrag",
        "password": "Trajanovic",
        "age": "55",
        "email": "trajanovic@gamil.vcom`",
        "website": "trajanovicmiodrag.com"
      },
      "primljeno": true
       },
  "2": {
      "users": {
        "name": "Miodrag Trajanovic",
        "password": "Trajanovicsifra",
        "age": "22",
        "email": "trajanovicmiodrag@yahoo.com",
        "website": "https:\/\/miodragtrajanovic.com\/"
      }
  }
}
```

```
echo '<h3> Prikaz u sa echo</h3>'."\n\n\n\n";
echo $priakz1= $array_data[2]['users']['name']."\n" ;
echo $priakz2= $array_data[1]['users']['age']."\n" ;
echo $priakz3= $array_data[2]['users']['password']."\n" ;
echo $priakz4= $array_data[1]['users']['email']."\n" ;
echo $priakz5= $array_data[2]['users']['website']."\n\n\n\n" ;
```

Na ovaj način se pristupa pojedinačnom nizu grupe podataka, koji se na ovaj način može prikazati kao pojedinačan podatak iz svega.

."\n" oznaka za novi red u php-u

Da bi ste videli podatke na stranici gde je php skript može se upotrebiti u toku eksperimentisanja i ovaj kod:

```
echo '<h3> Prikaz u print-r</h3>'."\n\n\n\n";
```

print_r($array_data['users']);

Dok ovaj kod niže u foreach petlji se upotrebljava kada imate ovakav izgleda koda u json failu i kada pristupate svim podacima users, koje ujedno i prikazujete u grupama podataka.

```
{
   "users": [
      {
         "name": "Mika ",
         "password": "Mika sifra",
         "age": "25",
         "email": "moj_Mika@email.com",
         "website": "Mika.com"
      },
      {
         "name": "Pera ",
         "password": "Pera sifra",
         "age": "35",
         "email": "moj_Pera@email.com",
         "website": "Pera.com"
      },
      {
         "name": "Moje ime Cetvrta",
         "password": "MOja sifra",
         "age": "1030",
         "email": "moj@email.com",
         "website": "moja sajt "
      },
      {
         "name": "MojeCetvrta ime ",
         "password": "Cetvrta sifra",
         "age": "120",
         "email": "moj@email.com",
         "website": "moja sajt "
      },
      {
         "name": "Miodrag",
         "password": "Trajanovic",
         "age": "55",
         "email": "trajanovic@gamil.vcom`",
         "website": "trajanovicmiodrag.com"
      }
   ]
```

```
}
/*
Ovaj kod dole u php-u vrši prikazivanje svih podataka jedan ispod drugog, čime se dobija nalik gornjem JSON failu na ekranu brovsera
*/
echo '<h3> Prikaz u foreach</h3>';
foreach( $array_data['users'] as $key => $value){
   echo  'ime: ' . $value['name']."\n";
   echo  'age: ' . $value['age']."\n";
   echo  'password: ' . $value['password']."\n";
   echo  'email: ' . $value['email']."\n";
   echo  'website: ' . $value['website']."\n\n\n\n";
}
```

Foreach petlja dati niz podataka razvrstavajući ih na ključeve i vrednosti tih ključeva. Pošto tih vrednosti za jedan ključ može da ih ima više, uzima se za njihov tip podataka niz.

Upotreba switch case

Zbog potreba univerzalnosti zahteva aplikacija kada je potrebno obraćanje serveru da uradi neke uslovne poslove pogodno je upotrebiti u php failu switch case u kombinaciji sa foreach petljom. Ako pogledamo react fail šalje zahtev php obrađuje zahtev i vraća rezultat.

React fail

React fail je osnovnog koda uz upotrebu dodatnog react paketa 'whatwg-fetch', koji se instališe kao i svi drugi dodatni paketi u reactu,

```
npm i - -save  whatwg-fetch

import React from 'react';
import 'whatwg-fetch';

export default class ReactPhp extends React.Component{

   constructor(props){
      super(props);

      this.getPHP = this.getPHP.bind(this);
   }

   getPHP ()
   {
      fetch('http://localhost/fetchReact/server/prijem.php',{
```

```
            method: 'POST',
            headers: {
               Accept: 'application/json',
               'Content-Type':'application/json'
            },
            body: JSON.stringify({
               content: 'upis'
            }),

         })
            .then( res=> res.json())
            .then( response => {
               console.log(response);
               alert(response.message);
            });
      }

      render(){
         return(
            <div className="container">
               <p>
                  React
               </p>
               <br/>
               <button className="btn-success"
                  onClick={this.getPHP}
                  >
                  Ucitaj
               </button>
            </div>
         );
      }
}
```

Kratak opis react faila

React fail ima jedno dugme koje poziva funkciju getPHP koja poziva metod fetch. Fetch prosleđuje u svom bodiju upotrebljeno this.setState objekt koji on prosleđuje izabranom php failu na serveru. Sever mu odgovara vraćajući response objekt. Obzirom da se radi o portu 80 na localhostu ili na vašem hostu, nije potrebno naglasiti to prilikom fetch-a jer php i server apache rade na tom portu zajedno. Ono što je bitno iz react aplikacije je da se njome proslede hederi, metod slanja i sadržaj koji se upućuje serveru. Za slučaj da šaljete više json podataka php skripti na serveru možete napraviti formu za unos. Ali ako pogledamo ovaj deo koda

```
body: JSON.stringify({
    content: 'upis'
}),
```

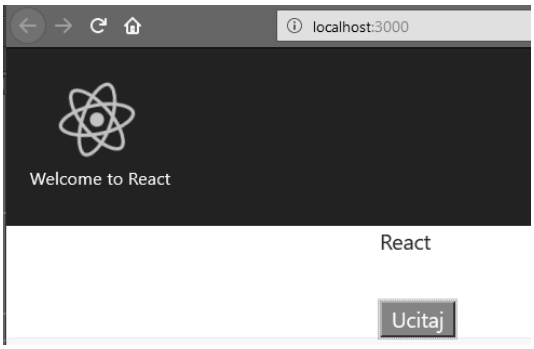
Možemo da tu dinamički menjamo sadržaj content ako mu dodamo

```
body: JSON.stringify({
    content: nekisadržaj
}),
```

Promenjiva vrednost (nekisadržaj) može da se izvrši prilikom događaja onClick ako se tu prema potrebama pozove postavljanje nove vrednosti stanja upotrebom

this.setState({nekisadržaj: “ slanje”}):

Umesto stringa u navodnicima možemo upotrebiti this.state.stanje. Na taj način omogućavamo stalne promene vrednosti.
Koji je namenjen u ovom slučaju kao što mu i u nazivu piše za Fetchovanje podataka. Na korisničkom ekranu se dobija ova slika.



Da bi smo videli da li i kako radi potrebno je u pretraživaču pritisnuti F12 za inspect mod. Gde kada kliknemo na ispis Network vidimo na slici

Vidimo prikazane zajedno i ključ i njegovu vrednost koju je vratila php skripta

```
<?php
header('Accept: application/json');
header('Access-Control-Allow-Methods: POST,GET,PUT,DELETE,OPTIONS');
header("Access-Control-Allow-Headers: eToken,X-Requested-With,
```

```
            Content-Type");
header('Content-type: text/json; charset=utf-8;
          application/x-www-form-urlencoded');
header ( 'Access-Control-Allow-Origin: *');

$results = json_decode( file_get_contents('php://input'), true);
```

Prisustvo hedera u php skripti je iz razloga da bi se znalo koje vrste podataka mogu se očekivati i koje opracije se mogu desiti. Za slučaj da vam je potrebno da proverite da li nešto uopšte šalje php u promenjivu $results unsite kod

```
print_r( file_get_contents('php://input'));
```

Sadržaj promenjive je dakle json objekt koji je potrebno da bi smo njegove delove upotrebili za dalji rad rastaviti. Najpogodniji način je upotrebe foreach petlje

```
foreach ($results as $key =>$val){
   echo "kljuc".$key."vrednost:".$val . PHP_EOL;
}
```

Kojom izdvajamo potrebno u ovom slučaju za pokretanje switch case-a. Ova može se reči petlja ili više vezanih if else uslova kako se to u praksi govori radi jednostavno u malim zagradama iz switch nalazi se uslov koji se ispituje, pa ako on odgovara nekom od istina u delu case-a onda se taj deo i pokreće, a ako ne izvršava se default slučaj i javlja grešku ili šta odredite.

```
switch($val){
    case "upis":
        echo "Ovim mogu da posljem poruku.";
        $response = array( "success"=> true, "message"=>" poruka je uspesno
poslata");
        $output = array_slice($response, 1,2);
        echo json_encode($output);
        break;
    case "NekiDrugiUslov":
        echo "Neka radanja.";
        break;
    default:
        echo "Za traženu radnju ne postoji odgovarajući slučaj.";
        break;
 }
```

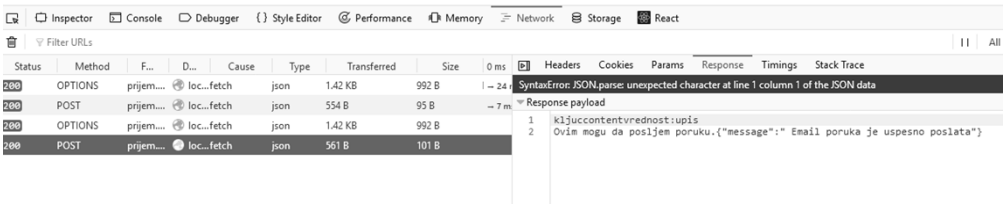
U okviru case možete pozvati funkcije i ono što vam je potrebno da uradi php skripta na serveru.

Povezivanje foreach i switch-case

To je vrllo jednostavna operacija, koja zahteva da se unutar foreach petlje usadi switch case

```
 foreach ($results as $key =>$val){
   echo "kljuc".$key."vrednost:".$val . PHP_EOL;

          switch($val){
              case "upis":
                 echo "Ovim mogu da posljem poruku.";
                 $response = array( "success"=> true, "message"=>"
                      Email poruka je uspesno poslata");
                 $output = array_slice($response, 1,2);
                 echo json_encode($output);
                 break;
              case "NekiDrugiUslov":
                 echo "Neka radanja.";
                 break;
              default:
                 echo "Za traženu radnju ne postoji odgovarajući slučaj.";
                 break;
           }

}
```

Na ovaj način smo povezali za sve one podatke koje budemo poslali zahtevom da se oni i primene.



sačuvati promene i pokrenuti ponovo apache server nakon cega pokrenuti localhost/node na ekranu bi ste brovsera trebali ugledati poruku is node okruženja
Na linku ispod nalazi se kraći tutorijal php-a koji prikazuje osnovne operacije php-a u radu sa podacima, kao što su dpdavnje novog zapisa, uređuvanje starih i brisanje podataka u bazi podataka. Odtuda se ovakav skup skriptova naziva CRUD. Skraćeno Create Read Update and Delete. Slično postoji i za React i druga okruženja. U prevodu :napravi, pročitaj , izmeni i obriši.
https://www.tutorialrepublic.com/php-tutorial/php-mysql-crud-application.php

Povezivanje react aplikacije sa php-om i mysql-om

Potrebno za rad

1. react aplikacija
2. php skripta
3. izrada baze podataka za podatke

Ali pre toga potrebno je upoznati kako se uključuju funkcije u php-u, da bi smo mogli da primenimo sve ono ispred.

Funkcije u php okruženju

Funkcije su delovi koda mnogih programskih govora i namenjene su da urade praktično jednim pozivom dosta kompleksne poslove na serveru ili kod korisnika, u zavisnosti gde su primenjene. One mogu biti u jednom failu odakle ih prema potrebi pozivamo onamo gde su nam potrebne. U slučaju php-a pozivanje drugih failova u onom koji ih poziva upotrebljavaju se : include, require, include_once i require_once. U formatu koda, nebitno koja se od njih upotrebljava:
include ('string.php');// pozivanje na upotrebu sadržaja celog php faila u drugom

Upotreba **require** i **include** je gotovo ista. Razlika je ta što u slučaju greške unutar fajla koji umeće **require** prekida izvršenje skripta, dok **include** daje samo upozorenje. A require_once/include_once, isto kao prethodno s tim to se datoteke ne mogu uključiti više puta, što je pogodno zbog praćenja redefinisanja funkcija, inicijalizacije promenljivih.
Ono što je bitno u php-u, je to da funkcije pre upotrebe moraju biti napisane u jednom ili više failova, ako neke promenjive imaju potrebu dostupnosti na više mesta onda moraju da imaju ispred svog imena reč global, kako bi bile vidljive.
Više o ovome na linku:
https://sr.wikipedia.org/sr-el/PHP#%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%98%D0%B5
Ali kao i u reactu, tako i u php-u funkcije mogu imati svoje parametre ili ih dobijati sa mesta odakle se one pozivaju. Znači odnos parent/child.

Rad funkcija

Osnovni kod funkcije izgleda ovako:

```
function ime_funkcije(promenjiva1, promenjiva 2, promenjiva..n)
{
   Blok naredbi;
}
```

U bloku naredbi ako je potrebno da funkcija nešto vrati i da se prekine njen rad upotrebljava se reč return iz koje se upisuju promenjive koje se se vraćaju nazad onamo gde je funkcija i pozvana. Uzeću primer gde se u jednom failu pozivaju funkcije iz drugog

fail probafun.php

```
<?php
/**
 * Created by IntelliJ IDEA.
 * User: trika
 * Date: 03-Nov-18
 * Time: 14:32
 */
include ('string.php');
$ime="<br><strong>Miodrag</strong><br>";
$prezime ="<br><b>Trajanovic</b><br> ";

echo"<form>".toString($ime,$prezime)."<form> <br><br><br>";
echo la($ime,$mika)."<br><br><br>";

povratna($ime,$prezime,$mika );
```

fail string.php

```
<?php
/**
 * Created by IntelliJ IDEA.
 * User: trika
 * Date: 03-Nov-18
 * Time: 18:38
 */
$mika="<br><h1><strong>Operacija globalnih vrednosti</strong></h1><br><p>
Praktican primer Funkcija u php-u</p>";
// proglašavanje varijabile za globalnom varijabilom
global $mika;

/* ako je potrebno da u neku promenjivu postavite više parametara možete na ovaj način ih spojiti u jedan niz podataka, s time što je potrebno prvo prijaviti praznu vrednost varijabile*/

function toString($ime,$prezime){
   $return_string='';
   $return_string .= "<textarea>";
```

```
   $return_string .= "<h1>".$ime."</h1>";
   $return_string .= "<h2> Podaci iz drugog faila prve funkcije>>>: '   '
$prezime</h2>";
   $return_string .= "</textarea>";
   return($return_string);
}

function la($ime,$mika){
   $return_str='';
   $return_str .= "<table>";
   $return_str .= "<tr><td>".$ime.$mika."</td></tr>";
   $return_str .= "<tr><td>Podaci iz drugog faila druge funkcije</td>";
   $return_str .= "</tr>";
   $return_str .= "</table>";
   return($return_str);
}

function povratna($mika,$ime,$prezime){
   echo "<br><h1><strong>Treca funkcija</strong></h1><br><p> Koja poziva
prve dve </p>";
   echo "<div><b> Povratna
funkcija</b><br>".la($ime,$mika)."<br>".toString($ime,$prezime)."<br>".$mika."
</div>";
}
```

Ovaj primer je pogodan jer se u njemu vidi pored primene i rada funkcija, i njihova upotreba kada se upotrebljavaju za prikazivanje rezultata koji vraćaju u obliku HTML-a. Kao što se vidi u prvom failu se pozivaju funkcije, obzirom da je ovo u pitanju rada sa html tagovima, funkcije su pozvane po tom pripadajućem mestu. Ali funkcija povratna, pozvana je samo kao

```
povratna($ime,$prezime, $mika);
```

Razlog je taj što ona ima već u sebi naredbu za prikazivanje echo,
povratna($ime,$prezime, $mika);
pa joj to nije potrebno. Ovo znači da funkcija može imati svoje vrednosti prametara, ali ih i može dobiti kao prosleđene vrednosti prametara od strane funkcija ili faila skripta koji je poziva. Može da ih vrati, ali i da ih po zahtevu i pre toga obradi. Što je bitno na primer kod postova, emaila i sličnih potreba, na koje će mo se bazirati u ovoj knjizi. Istovremeno primer pokazuje i način dodavanja globalnih varijabila koje se mogu prikazati na mestu gde je i pozvana funkcija, zajedno sa time da jedna funkcija može pozvati druge sa njenim vrednostima prametara. Ali ako prosleđujete html možete upotrebiti i ovaj način:

Dodatni deo koda u failu probafun.php
Heredoc_pr($ime,$prezime,$mika);

dodatni deo koda u failu string.php

```
function Heredoc_pr($ime,$prezime,$mika){
   $html = <<<HTML
  <div style="border: #5131ff 2px solid">
    <ul class='mylist'>
      <li>$ime</li>
      <li>$prezime</li>
      <li>$mika</li>
    </ul>
  </div>
HTML;

// na kraju prikaz
   echo $html;
}
```

Obzirom da nam je u ovom slučaju potrebno da php i mysql rade posao sa podacima na serveru ovo sam napomenuo kako bi se stekli sliku mogućih događanja i princip rada. Princip rad koji će biti zastupljen je da react aplikacija pošalje zahtev sa potrebnim podacima kako bi foreach i switch/sace mogla da to prosledi potrebnim funkcijama, koje bi onda uradile svoj deo posla vezan za rad na serveru i po potrebi posla pošalju obaveštenje o uspešnosti rada.

Uzjamni rad react i php

Da bi smo mogli da neke podatke unosimo na server ili da obavimo neku kupovinu i slično, potrebno je da na tom veb domenu koji je smešten na tom serveru, se prijavimo na nalog ako imamo nalog ili da ako nemamo nalog da ga napravimo. Ove radnje se rade putem unosnih formi. Za prijavu potrebno je u suštini imati dva polja za unos: korisničko ime i šifru, dok za izradu naloga zavisno od potreba potrebno je malo više unosnih polja. Pored toga, ove forme potrebno je da podržavaju proveru: da li su unosna polja prazna ili ne, da li sadržavaju potrebne znakove, da pre unosa u bazu se izbrišu praznine iz unosnog dela vrednosti i tome slično. Kod korisnika, znači react aplikaciji možemo da proverimo: dužinu unešenih vrednosti, da li su polja prazna, da li su popunjena zahtevana – obavezna polja i njhov saržaj da li ima odgovarajuće znakove i njhov raspored (da li je prvi znak broj ili slovo- veliko malo i sl). Dok u php-u pre upisa u bazu: može isto, ali i upotrebom trim, escape_string, mysqli_real_escape_string i sličnih poput preg_match("/^[0-9]+$/", $age) da li u unosnom polju ima nekog drugog znak osim dozvoljenog. Ovo je ujedno i jedan od načina zaštite od nedozvoljenih upada na server. Pogotovu gde se obvalja elektronsko plaćanje. Ali, je najbolji način da

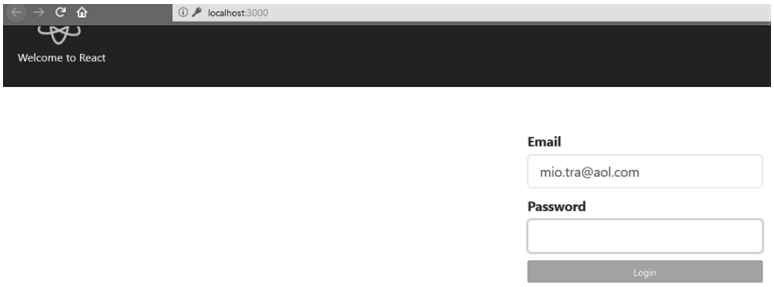
postoje provere unešenih podataka, mislim na sadržaj, i kod korisnika i na serveru. To je ujedno i način za najbolju sigurnost, te je to i moja preporuka svima.

React aplikacija

Primer koda dole ispod

```
import React, { Component } from "react";
import { Button, FormGroup, FormControl, ControlLabel } from "react-bootstrap";
import "./Login.css";

export default class Login extends Component {
  constructor(props) {
    super(props);

    this.state = {
      email: "",
      password: ""
    };
  }
/*validacija da li je uneta vrednost veća od prazno, odnosno da li je u unosno polje upisano nešto*/
  validateForm() {
    return this.state.email.length > 0 && this.state.password.length > 0;
  }

// čitanje promena unosenja vrednosti u unosnim poljima
  handleChange = event => {
    this.setState({
      [event.target.id]: event.target.value
    });
  };

/* rad posle klika na dugme za unos podataka dalje na server. Za sada se oni
  prikazuju na stranici. */
  handleSubmit = event => {
    event.preventDefault();
    let em = this.setState({em: this.state.email});
    let pasw = this.setState({pasw: this.state.password});
    console.log(this.state);
  };

  render() {
    return (
      <div className="Login">
```

```
          <form onSubmit={this.handleSubmit}>
            <FormGroup controlId="email" bsSize="large">
              <ControlLabel>Email</ControlLabel>
              <FormControl
                autoFocus
                type="email"
                value={this.state.email}
                onChange={this.handleChange}
              />
            </FormGroup>
            <FormGroup controlId="password" bsSize="large">
              <ControlLabel>Password</ControlLabel>
              <FormControl
                value={this.state.password}
                onChange={this.handleChange}
                type="password"
              />
            </FormGroup>
            <Button
              block
              bsSize="large"
              className="btn-primary"
              disabled={!this.validateForm()}
              type="submit"
               >
              Prijava
            </Button>
          </form>
          <h1>Unete vrednosti</h1>
           <h2>Unet email:{this.state.em}</h2>
           <h2>Unet sifra:{this.state.pasw}</h2>
        </div>
    );
  }
}
```



Unete vrednosti
Unet email:mio.tra@aol.com
Unet sifra:323435

Ova forma za prijavu na nalog ima još jednu namenu, a to je da prikaže kada se upotrebljava dodati paket react-bootstrap. Ima jednostavnu validaciju podataka, tačnije ako je neko od unosnih polja ili oba su prazna onda je dugme za prijavu isključeno.

Validacija

Validacija je provera podataka unetih u formu pre nego što je prihvati server i upiše u bazi podataka. Kod korisnika se proverava da li podaci odgovaraju postavljenim uslovima koji su postavljeni u formi ili ne. Za slučaj negativnog stanja korisnik dobija poruku upozorenja kako bi ispravio unete podatke i događaj na formi mogao da ih pošallje na server, gde se oni mogu još jednom proveriti. Kod ispod prikazuje moguću proveru podataka koji se unose u formu na strani korisnika:

```
import React from 'react';
// možete u ovom delu importovati vaš css kod

export default class Form extends React.Component {
   constructor (props) {
      super(props);
      this.state = {
         ime:'',
         prezime:'',
         email: '',
         password: '',
         formErrors: {ime:'', prezime:'', email: '', password: ''},
         imeValid: false,
         prezimedValid: false,
         emailValid: false,
         passwordValid: false,
         formValid: false
      }
   }
// omogućavanje da se promene mogu vršiti u unosnim poljima
   promenaUnosnihPolja = (e) => {
```

```
        const name = e.target.name;
        const value = e.target.value;
        this.setState({[name]: value},
            () => { this.validateField(name, value) });// arrow funkcija
    };

/* provera svakog pojedinačnog polja i njegove vrednosti(funkcija dobija vrednosti
za fieldName i value. Vrednost fieldName se upotrebljava u switch-u za case
uslove.*/
    validateField(fieldName, value) {
        let fieldValidationErrors = this.state.formErrors;
        let imeValid = this.state.imeValid;
        let prezimedValid = this.state.prezimedValid;
        let emailValid = this.state.emailValid;
        let passwordValid = this.state.passwordValid;

      // unosom vrednosti za fieldName se određuje šta će od unosnih polja da se
           testira uslovima koji su u delu između case
        switch(fieldName) {
            case 'ime':
            imeValid = value.length >= 10;
            fieldValidationErrors.ime = imeValid ? '': ' potrebno je da ima 10 i vise
                znakova ';
            break;
            case 'prezime':
                prezimedValid = value.length >= 15;
                fieldValidationErrors.prezime = prezimedValid ? '': ' potrebno je da ima
                 15 i vise znakova ';
                break;
            case 'email':
                emailValid = value.match(/^([\w.%+-]+)@([\w-]+\.)+([\w]{2,})$/i);
                fieldValidationErrors.email = emailValid ? '' : 'je neispravan ';
                break;
            case 'password':
                passwordValid = value.length = 15;
                fieldValidationErrors.password = passwordValid ? '': 'potrebno je da ima
                15 i vise znakova';
                break;

            default:
                break;
        }
      // postavljanje stanja u promenjive nakom izvršene provere
```

```
        this.setState({
            formErrors: fieldValidationErrors,
            imeValid: imeValid,
            prezimedValid: prezimedValid,
            emailValid: emailValid,
            passwordValid: passwordValid
        }, this.validateForm);// povezivanje sa validateForm funkcijom
    }

// validacija forme gde ako su svi unosi u poljima ispravni onda se te vrednosti
// ispravnosti predaju kao stanje promenjivoj fromValid

    validateForm() {
        let provera= this.state.imeValid + this.state.prezimedValid +
this.state.emailValid +
             this.state.passwordValid;
        this.setState({formValid: provera});
    }

// ako je dužina unosa nula uključujem has-error da bi se ukazao na neispravnost
    errorClass=(error)=> {
        return(error.length === 0 ? '' : 'has-error');
    };

// u ovom delu koda dodajemo fetch za komunikaciju sa serverom(prijem , slanje i
// sl. podataka od korisnika  prema serveru i bazi podataka)
    handleSubmit = event => {
        event.preventDefault();
        let ime1 = this.setState({ime1: this.state.ime});
        let prez = this.setState({prez: this.state.prezime});
        let em = this.setState({em: this.state.email});
        let pasw = this.setState({pasw: this.state.password});
        console.log(this.state); // prikaz svih stanja u konzoli pretraživača

// ovim postavljanjem stanja se vrši brisanje sadržaja unetih podataka iz forme
        this.setState({
            ime:'',
            prezime:'',
            email: '',
            password: ''
        });
    };
    render () {
        return (
```

```
<div className="container">
   <form className="demoForm" >
      <h2>Sign up</h2>
      <span >
         <FormErrors formErrors={this.state.formErrors} />
      </span>
      <div className={'form-group
            ${this.errorClass(this.state.formErrors.ime)}'}>
         <label htmlFor="ime">Ime </label>
         <input type="text" required className="form-control" name="ime"
               placeholder="Ime"
               value={this.state.ime}
               onChange={this.promenaUnosnihPolja}  />
      </div>
      <div className={`form-group
            ${this.errorClass(this.state.formErrors.prezime)}`}>
         <label htmlFor="prezime">Prezime</label>
         <input type="prezime" className="form-control" name="prezime"
               placeholder="prezime"
               value={this.state.prezime}
               onChange={this.promenaUnosnihPolja}  />
      </div>
      <div className={`form-group
            ${this.errorClass(this.state.formErrors.email)}`}>
         <label htmlFor="email">Email address</label>
         <input type="email" required className="form-control"
               name="email" placeholder="Email"
               value={this.state.email}
               onChange={this.promenaUnosnihPolja}  />
      </div>
      <div className={`form-group
         ${this.errorClass(this.state.formErrors.password)}`}>
         <label htmlFor="password">Password</label>
         <input type="password" className="form-control"
           name="password" placeholder="Password"
               value={this.state.password}
               onChange={this.promenaUnosnihPolja}  />
      </div>
      <button type="submit" className="btn btn-primary"
        disabled={!this.state.formValid} onClick={this.handleSubmit}>
        Reagistracija
      </button>
   </form>
   <hr/>
```

Ovaj deo (Unete vrednosti) je samo tu da bi mogli trenutno da vidite dok eksperimentišete

```
// ovaj deo ispod je namenjen da priakže umete podatke posle slanja
            <h1>Unete vrednosti</h1>
            <h2>Unet ime:{this.state.ime1}</h2>
            <h2>Unet prezime :{this.state.prez}</h2>
            <h2>Unet email:{this.state.em}</h2>
            <h2>Unet sifra:{this.state.pasw}</h2>
          </div>
      )
   }
}
```

Konstanta koja vraća tekst greške svakom polju u kom unosite podatak ukoliko nije zadovoljen postavljen uslov ispisan u p tagu redosledom: ime unosnog polja i koja se greška javila na kom polju.

```
const FormErrors = ({formErrors}) =>{
   return(
         <div style={{color:"red"}}>
            {Object.keys(formErrors).map((fieldName, i) => {
               if(formErrors[fieldName].length > 0){
                  return (
                     <p key={i}>{fieldName} {formErrors[fieldName]}</p>
                  )
               } else {
                  return '';
               }
            })}
         </div>
      );
};
```

Za unosna polja možete ovako upotrebljavati boostrapove ikonice:

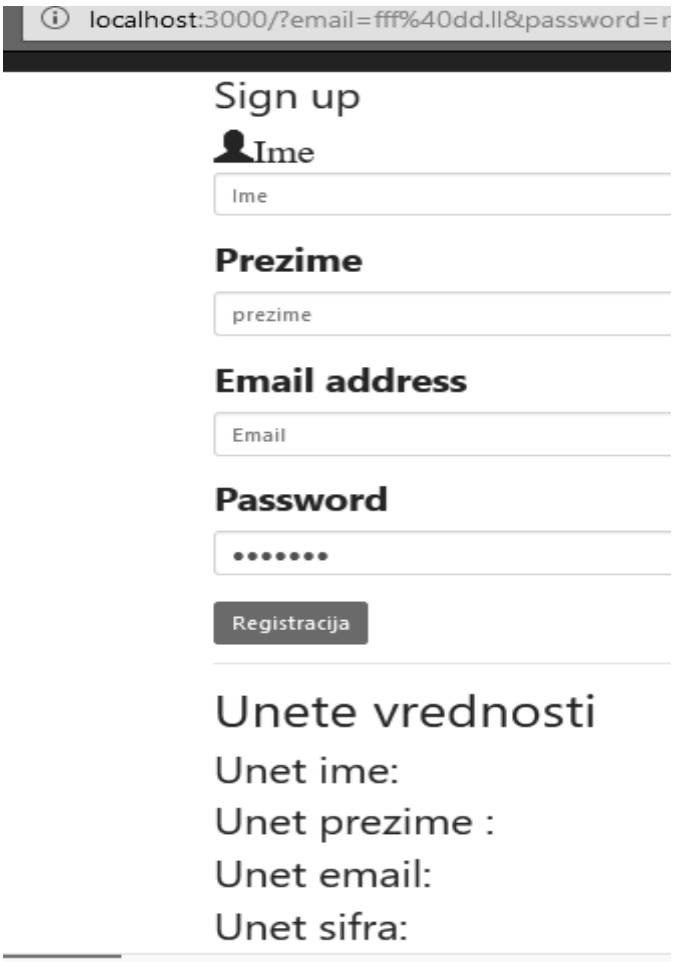
```
<div className={'form-group ${this.errorClass(this.state.formErrors.ime)}'}>
   <div className="input-group">
      <label htmlFor="ime" >Ime</label>

        kod za upotrebu ikonica je ovaj red ispod
     <i style={{color:"red"}} className="glyphicon glyphicon-user"/>

      <input id="email" type="text" className="form-control" name=" ime "
```

```
        placeholder=" Upisi svoje ime " style={{width:"300px",
                        fontSize:"16px", hegiht:"50px"}}/>
    </div>
</div>
```

Ali, ako upotrebite klasu za bootstrap inkone na ovaj način

<label htmlFor="ime" className="glyphicon glyphicon-user ">Ime</label>

u tom slučaju ikonica se prikazuje ispred ispisa labele



Ovo je jedan jednostavniji način provere ispravnosti unetih podataka na strani korisnika. Bliže o validacijama na linkovima:
https://learnetto.com/blog/how-to-do-simple-form-validation-in-reactjs
https://blog.cloudboost.io/how-i-did-validation-for-forms-in-react-js-5e2784803d7b

Kako dalje

Obzirom da imamo u predhodnim delovima knjige kako react može da pošalje podatke php-u, prećiću na objašnjenje šta bi trebalo da php skripta da uradi. Php na svojoj strani treba ne samo da omogući razmenu podataka sa bazom, nego i da uradi redirekciju korisnika na njegovu sekciju (session), kako bi korisnik mogao neometano da pristupi svom nalogu i svojoj osnovnoj stranici.
Prepostavimo da ste na nekoj socijalnoj mreži. Za pristup vam se traže obično dva parametra emali ili korisnicko ime i sifra (password). Ti podaci se upotrebljavaju da bi se proverila identičnost sa podacima koje ste upotrebili da napravilte nalog. Ali, se automacki pravi i jedan još podatak user_uid. Od svih tih podataka je moguće napraviti ime za sekciju, tako da su vaši podaci i privatnost zaštićeni. Ali, ovaj dodatni ček input može da omogući da se određeni postovi prikažu ili ne, kao i kod brisnja loših korisnika ili ako želite da obrišete svoj nalog, koji imaju vrednost ovog ček boksa jednako broju jedan, dok svi ostali ostaju skriveni. Tako da se time meže uraditi još jedna vrsta kontrole.

Napomena: uobičajeno je da na serveru se od passworda pravi u cilju povećane sigurnosti takozavani heš password tako da od unetog php svojom funkcijom:

```
$password = $_POST['password'];

$hashed_password = password_hash($password, PASSWORD_DEFAULT);
da bi videli kako to izgleda na ekranu unesite
var_dump($hashed_password);
```

Šta se postiže ovim? Otežava se da neko vam preuzme vaše podatke i naruši privatnost, jer ova funkcija pravi od vešeg unetog podatka niz znakova određene dužine, koji kada pogledate nema nekog nalika onom kog ste uneli. Provera ovog podatka :

```
if (password_verify($password_inputted_by_user, $password_encrypted)) {
   // Uspesno!
   echo 'Sifra je odgovarajuća ';
}else {
     // Neuspesno
       echo 'Sifra je neodgovrajuca';
```

Drugi zadatak php skripte je da upotrebljene funkcije obaveste korisnika kako o problemima, tako i o uspešnim akcijama ili da se pošalju podaci iz baze. Za takav odnos korisničkog dela i serverskog upotrebljava se JSON. Ono što se vrati u toj komunikaciji mora biti prevedeno u taj format zapisa, što se i čini upotrebom

```
$povratna_informacija_ili_podaci = vrednosti koje se vraćaju;
```

```
echo json_encode($povratna_informacija_ili_podaci);
```

Na ovaj način se vraća react aplikaciji objekt koji se treba u react aplikaciji rastaviti i formatizovati za prikaz. Ali, možete i od ovog objekta koji šalje php skripta u react aplikaciji napraviti niz podataka pa onda tamo izdvojiti potrebni podatak ili kao u opisu niže. Na primer želite da vratite neku poruku sa servera react aplikaciji

```
$poruka= „Dobar dan“;
echo json_encode($poruka);
```

U react aplikaciji bi smo ovaj objekt po prijemu rastavili i prikazali samo poruku vidljivo ako u prijemu

```
componentDidMount(){
.....
 then(response

        alert(response.poruka);
}
```

Naravno, i php može da proveri ispravnost podataka upotrebom svojih funkcija. Dole ispod je jedna

```
preg_match(„/^[a-zA-Z]*$/“, $varijavila)
```

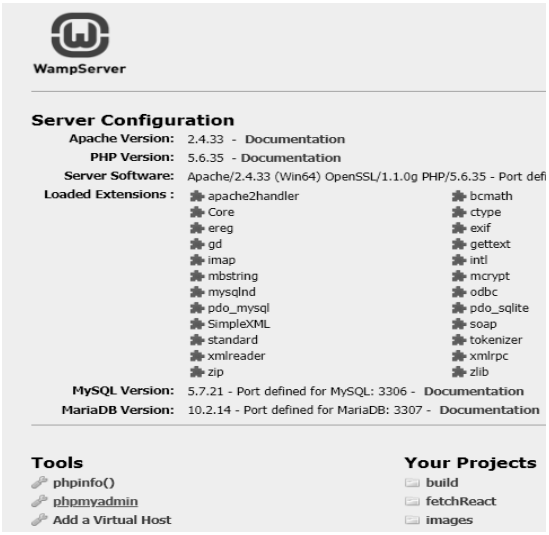
kojom php dozvoljava samo slova za unosna mesta gde se ona zahteva. Na sličan način se mogu postaviti i drugi uslovi u istoj funkciji. Pomenuo sam ovu karakterističnu jer postoji sličnosti u react (odnosno java skriptu i dr.).
Pa se taj deo koji povezuje korisnika sa serverom iz tih razloga naziva programbilno prilagođenje (API). Korisnički deo šalje zahteve (request), server šalje odgovore(response). Oba, zahteve i odgovore možete videti u brovseru kada sa pritiskom na F12 uđete u nadzorni način prikaza i kliknete na tab Network. A, potom kliknete na neki od natpisa u levom delu ekrana pored kog piše broj statusa, obično je to broj 200. U suštini ono što će biti u ovoj knjizi osnova je:

1. Izrada komonente putem koje ćemo se prijaviti, odjaviti ili registrovati
2. Prazna home stranica na koju ćemo biti upućeni posle uspešne prijave, a koju može se ostaviti da korisnik dodaje sadržaje ili šablonski odrediti sadržaj
3. Izrada baze podataka
4. Prihvatne php skripte

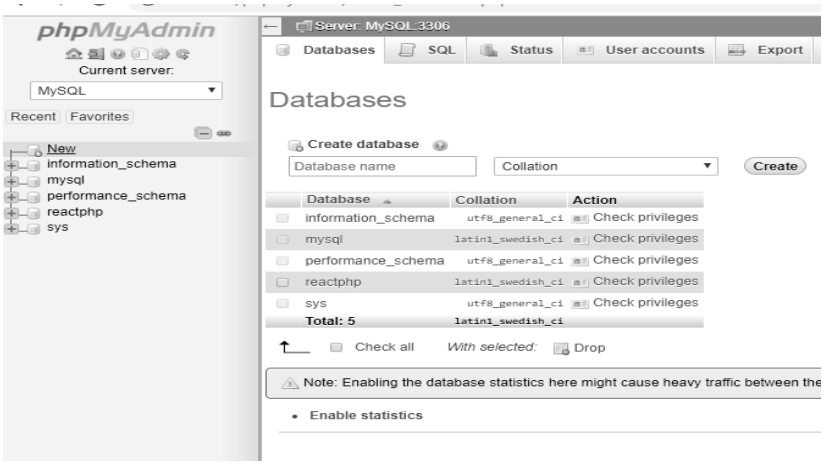
Izrada baze podataka

Izrada beze podataka vrši se u PHP my admin-u. To je najsigurniji način za početak. Ali, možete da napravite php skriptu sa if uslovom koji bi prvo ispitao da li postoji baza na serveru, iz toga da li postoji ciljna tabela u bazi i ako one ne

postoje da se automacki prave prema kodu. Što je malo više složeniji rad, te ću objasniti prvi postupak. Za njega potrebno je imati instalisanu neku od aplikacija koje sadrže u sebi Apache server, PHP i MySQLI. Wamp, Xamp i Mamp, zavisno od racunara i operativnog sistema koji imate.



Nakon klika na link phpmyadmin i prijave na lokalni server na racunaru

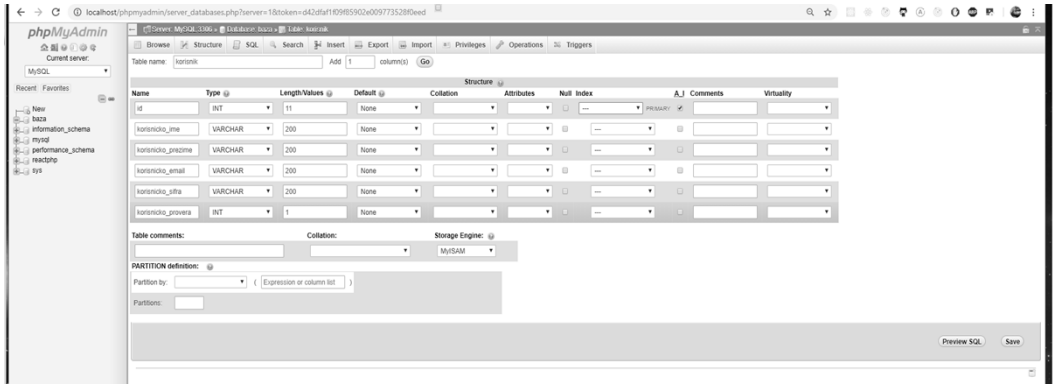


I klik na na New u polje sa desne strane Database_name unesite ime baze podatak koju izrađujete i kliknite na dugme Create pored. Nakon čega se pojavljuje upit za izradu tabele u bazi. Pored ispisa Name uneste naziv tabele do njega u polje Numbers of columns broj potrebnih kolona. Broj potrebnih kolona zavisi od broja polja koja su vam potrebna u poslu. Za potrebe prijave i registracije potrebna su polja:

1. id

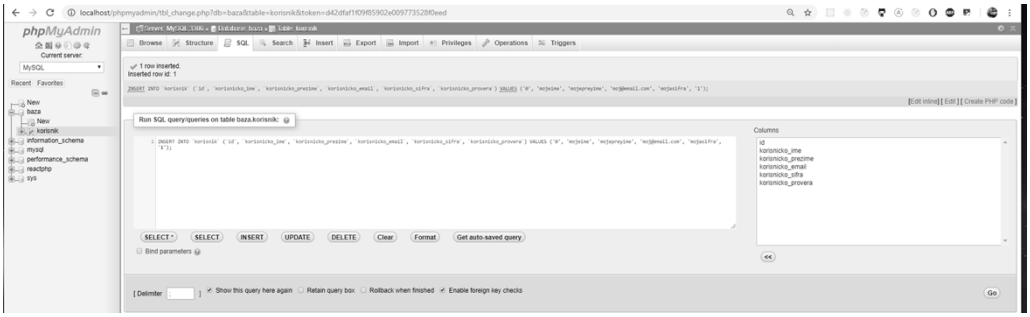
2. korisnicko_ime
3. korisničko_prezime
4. korisnički_email
5. korisnička_šifra
6. potvrda saglasnosti u obliku ček boksa.

Id polje je nešto kao matični broj u ličnoj karti, znači jedinstveno u celoj bazi za red u tabeli. Ovo polje je uvek u upotrebiti kao pokazivač kada se obraćamo za bilo koje drugo polje nad kojim se vrši neka operacija: brisanje, prepravka, izrada novog, ili dodavanje starom polju nekog novog podatka. Tako da je ono uvek brojnog tipa (numeric) koji nemože biti null i ima automacko povećanje vrednosti (AUTO_INCREMENT) i ono je uvek sa oznakom PRIMAY KEY. Korisnički preostali podaci mogu biti tipa VARCHAR, osim polja za saglasnost koje tipa int.



Kao što izgleda na ovoj slici. Nakon čega sačuvate unete promene sa klikom na dugme Save desno od ovog prikaza.

Napomena: U nekim slučajevima u zavisnosti od podešene konfiguracije potrebno je polje id označiti kao indeksno, Što se čini klikom levog tastera miša ispod natpisa Index i tu se izaberete INDEX. Id polje je standardno postavljeno da se povećava od početne vrednosti nule pa naviše ali ako vam je potrebno možete da to promenite klikom na tab operacije. Klikom na tab Import možete uneti ručno vrednosti za proveru ispravnosti urađenog ovog koraka, ako je sve u redu klikom na tab SQL dobijate ovu sliku



Ali da bi smo napravili tabelu u bazi podataka možemo napisati ovakav kod

```
CREATE TABLE `posts` (
 `id` int(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
 `user_id` int(11) DEFAULT NULL,
 `title` varchar(255) NOT NULL,
 `slug` varchar(255) NOT NULL UNIQUE,
 `views` int(11) NOT NULL DEFAULT '0',
 `image` varchar(255) NOT NULL,
 `body` text NOT NULL,
)
```

izbrisati kod koji je u prikazu i postaviti ovaj ispod u gornji prikaz i kliknuti desno na dugme Go.
Pre toga objasniću malo ovaj kod. Iza CREATE TABLE je u izvrnutim apostrofima naziv tabele. Donji redovi koda se odnose na redove u tabelama, koji čine semu njihovih opisa. Reč šema sam pomenuo jer i kod čistih react aplikacija sa node ili express se upotrebljava baza podataka koja je moglo bi se prevesti u oblacima, a koja upotrebljava jedan fail za svoj rad koji se naziva šema, ima istu namenu kao i prisutni kod kojim izrađujemo tabelu u mysqlu.

`id` int(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,

Ovaj red znači između apostrofa naziv reda int(11), da se radi o tipu podataka koji se tu postavljaju da je celobrojna vrednost i da ima dužinu od 11 brojnih mesta, iza čega da nemože biti prazno i da mu se početna vrednost automacki menja naviše, krajnje reči da je primarni ključ u tabeli.
Tako da ovaj kod možete da prepravite za vaše potrebe i da njime u munut dva napraviti tabelu sa onim što je vama potrebno.

Korak 2 izrada php skripti

U ovom delu izradićemo tri skripte:
1. index.php
2. konekcija.php
3. funkcije.php

Index.php je fail preko koga ćemo da pozivamo potrebne funkcije za izvršenje potrebnih poslova, drugim rečima on je taj kojim povezujemo react aplikaciju sa serverom i bazom podataka. I mogli bi smo za taj folder gde smešatamo ova tri faila nazvati tim čuvenim API.
U index.php failu potrebno je smestiti kod za dozvole pristupa i razmene podataka na serveru, shodno potrebama upis, čitanje, brisanje i sl. Te je u tom slučaju najbolje u ovom failu postaviti zbog jednostavnosti izrade koda foreach i switch/case. Jer, ova kombinacija sa malo koda može da uradi brojne poslove. Petljom foreach vršimo izdvajanja poslatih podataka iz objekta koji šalje react aplikacija JSON formatom podataka, switch prima određeni podatak da bi pokrenula odgovarajući case i time bila pokrenuta određena funkcija koja nakon svog izvršenja ili vraća reazultat ili samo uradi na primer slanje emaila. No kako nam je potrebna informacija o izvršenosti određjenog zadatka najbolje je da te informacije kao i druge korisne funkcija vrati preko index.php faila korisniku, kako bi on preuzeo potrebne korake. Razlog za naziv fail index je taj što svi programi na serverima uvek traže tu reč za ime faila koje se prvo uvek pogleda da li postoji a ako ga nema onda dobijamo prazan ekranski prikaz.
Fail konekcija.php namenjen je da inicijalizuje povezivanje na određenu bazu podataka, dok u failu funkcije.php biće smeštane funkcije koje ćemo da pozivamo.

Index.php

Kako bi vam bilo jasnije prvo ću napisati php kod sva tri faila. Kod index.php faila bi bio ovakav, pošto ću u kodu ispod ovih delova prvo napisati kako to izgleda sve u php-u u obliku OPP-a, a u delu posle njega prilagoditi samo deo koda koji je neophodan da bi se mogao o react uključiti. Cilj toga je da se može uočiti slika kakva je funkcija koda, pristupa i rada pojedinih delova:

```
<?php
/**
 * Created by IntelliJ IDEA.
 * User: trika
 * Date: 10-Nov-18
 * Time: 17:01
 */
// pozivanje faila sa funkcijama

include ('pocetna_funkcije.php');
/* ovo je petlja gde se proverava da li je primljen ($_POST['submit'], zahtev od forme koja ga događajem klik šalje serveru, koji osluškuje sve zahteve, ako je prosleđen ovaj zahtev onda se i vrši očitavanje sadržaja zahteva $_POST['prijavi'] , kojim se određuje akcija u switch/case-u, gde će u zavisnosti od sadržaja biti pozivane funkcije, koje rade svaka svoj zadatak */
```

```
if(isset($_POST['submit'])){
    switch($_POST['prijavi']){
      case "upisivanje":
         echo "Upisujem podatke u bazu podataka.";
         echo upis_podataka();
         header('refresh: 5; url = http://localhost/fetchReact/server/tabela.php');
         exit();
         break;
      case "procitaj":
         echo "Citam podatke iz baze podatak .";
         echo citanje_podataka();
         break;
      case "prijavi":
         echo "Prijava na nalog.";
         echo prijava_korisnika();
         break;
      case "odjavi":
         echo "Odjava sa naloga";
         echo odjava_korisnika();
         break;
      case "registracija":
         echo "registracija korisnika.";
         echo registracija_korisnika();
         break;
      case "preuredi":
         echo "Prepravka unetih podataka.";
         echo preuredjivanje_podataka();
         break;
      case "dodaj":
         echo "Dodajem novi podatak u bazu.";
         echo dodavanje_novog_podataka();
         break;
      default:
         echo "Ovo je nemoguci izbor.";
         break;
   }
}
```

Index fail je kao most izmedju ostalih failova. Fail tabela ga poziva, a on poziva fail sa funkcijama. Nakon poziva u njemu je i ovaj kod za redirekciju, koju vrši nakon 5 sekundi.

header('refresh: 5; url = http://localhost/fetchReact/server/tabela.php');

Ovaj jednostavni kod ima u sebi naredbe za prikaz teksta na ekran samo da bi ste stekli osećaj kako šta radi. Umesto toga će u daljem delu knjige biti samo pozivi funkcija koje rade određen posao i primaju ili vraćaju rezultat tog rada.

Napomena: u ovim php failovima je postavljeno da se rezultat rada vidi direktno na stranicama, kako bi se prikazao potreban rad svega, dok kada se kod primeni kao deo react aplikacije on neće to imati jer nije tada potrebno. Ovo napominjem kako se nebi kasnije zbunili, jer react aplikaciji je potrebno samo proslediti vrednosti promenjivih da bi ih onda on prikazao korisniku.

Tabela.php

Tabela kao što joj i ime govori je klasična forma koja šalje zahteve preko index.php faila u switch/case koji poziva funkcije iz trećeg faila. U ovom failu je i prikazano kako možete da uključite poseban css fail u php skriptu.
Forma je smeštena u funkciju, gde je promenjivoj $tabela pridodat sadržaj kompletnog koda forme, dok kodom tabela1(); pozivamo izvršenje funkcije u ovom failu. Kod react aplikacije ovaj kod iz forme šaljemo uz upotrebu fetch-a index-php failu.

```
<?php
?>
<!—ukljucivanje css-a u php fail -- >
<link href="tabela.css" rel="stylesheet" type="text/css"/>
<?php


function tabela1(){
  $tabela ="
     <form method=\"post\" action=\"pocetna.php\">
      <label for=\"prijava\">Za switch</label><br>
      <input type=\"text\" name=\"prijavi\" id=\"prijava\"><br><br>
      <button type=\"submit\" name=\"submit\">Posalji</button><br>
   </form>
   ";
  echo $tabela;
}
tabela1();
?>
```

fail tabela.css

```
form{
   margin-left: 200px;
```

```
    margin-top: 50px;
    background-color: #0f0f0f;
    border: #ff905b 2px solid;
    width: 300px;
    border-radius: 10px;
}
form input{
    width: 200px;
    margin-bottom: 10px;
    margin-left: 8px;

}
button{
    margin-top: -20px;
    margin-right: 10px;
    background-color: #2b81af;
    color: whitesmoke;
    float: right;

}
```

fail pocetna_funkcije.php

U ovom failu su samo php funkcije koje će obrađivati zahteve na poziv u index.php failu. U prvoj funkciji dodao sam ono što je potrebno i za react aplikaciju kod:

```
  echo (json_encode($povratno));
```

Koji vraća ono što se smestiti u promenjivu $povratno u obliku json objekta, to jest podatka vrednosti kljuca i njegove vrednosti. Što se kod react aplikacije može upotrebiti za čitanje i drugo što je vezano sa bazom podataka. Ovaj deo upisa možete videti u inspekt modu brovsera klikom na Network tab i klikom na POST fail u delu Response.Dok ne upisemo kompletan kod funkcija pored ovog što šalje

```
echo (json_encode($povratno));
```

videćemo i vrednost promenjive koju vraća naredba return.

```
<?php
/**
 * Created by IntelliJ IDEA.
 * User: trika
 * Date: 10-Nov-18
 * Time: 17:04
```

```
 */

function upis_podataka(){
   $Prijavi = "<br>Povratno iz fail pocetna_funkcije upisujem podatke u bazu ";
    $povratno = " vracam response";
    echo (json_encode($povratno));
    return $Prijavi;
}
function citanje_podataka(){
   $Prijavi = "<br>Povratno iz fail pocetna_funkcije citam podatke iz baze";
    return $Prijavi;
}
function prijava_korisnika(){
   $Prijavi = "<br>Povratno iz fail pocetna_funkcije prijavil iste se na nalog";

    return $Prijavi;
}
function odjava_korisnika(){
   $Prijavi = "<br>Povratno iz fail pocetna_funkcije Odjavili ste se sa naloga ";
    return $Prijavi;
}
```

Napomena: sve što funkcija vrati kao rezultat rada može prihvatiti druga funkcija kao svoj argument. Napominjem to jer u praksi je ponekada ugledati ideju.

Fail konekcija.php

Ovaj fail omogućava povezivanje na bazu podataka i ujedno ako ima nekih problema sa povezivanjem i slično iz nega se vraćaju opisi grešaka.

```
<?php
/**
 * Created by IntelliJ IDEA.
 * User: trika
 * Date: 27-Aug-18
 * Time: 23:16
 */

$locahost = 'localhost';
$admin = 'administrator';
$passw ='šifra';
$baza='test';

$mysqli = new mysqli( $locahost, $admin, $passw, $baza);
```

```
/* Provera konekcije */
if (mysqli_connect_errno()) {
   printf("Problemi sa konekcijom: %s\n", mysqli_connect_error());
   exit();
}
?>
```

Kao što se i vidi u kodu on ima četiri promenjive koje mu omogućavaju da proveri da li ima u ovom slučaju problema sa konekcijom na serveru gde se nalazi baza podataka. Umesto vrednosti localhost postavlja se naziv domen gde je smeštena baza podataka. Ovaj fail se poziva
include ('konekcija.php');

Mi ga možemo smestiti u fail sa funkcijama jer će ga one i pozivati za svoj rad. Ali on se najčešće poziva u index.php.

Kako primiti i poslati vrednosti php funkciji

U zavisnosti šta funkcija treba da uradi, njoj se mogu proslediti ili od nje dobiti podaci na mestu njenog poziva. Prosleđivanje podataka funkciji se vrši tako što se na mestu poziva funkcije u malim zagradama unose imena promenjivih, odvojene zarezom. Dok povratne upotrebom službene reči return u samom telu funkcije. Ali isto tako ako je potrebno neko obaveštenje da sama funkcija pošalje može se upotrebiti i json_encode. Mada, funkcije mogu da vrate i više od jedne promenjive.

```
function vise ($ako_je_istina = true) {
   $var1 = "Jedan";
   $var2 = "Dva";

   if($ako_je_istina === true) {
     return $var2;
   }

   if($ako_je_istina == "obe") {
     return array($var1, $var2);
   }

   return $var1;
}

echo vise ("obe")[0]
// echo: jedna
echo vise ("obe")[1]
// echo: druga
```

Ova funkcija ako nema istine vraća varijabilu jedan, ako ima istine vraća vrednost varijabile dva, ali ako je uneta vrednost jednaka obe onda vraća niz sa vrednostima promenjivih, nebitno koliko ih ima, a koje su međusobno razdvojene zarezom. Ovo je zato što kod računara se kao i kod brojanja uvek počinje brojem nula. Pa je prvi podatak u bazi ili nekom nizu uvek nuliti član [0] niza.
Dok za slučaj da iz funkcije šaljemo više promenjivih onda je potrebno ih prvo rastaviti na delove pa ih onda upotrebiti dalje. U kodu dole to čini napredba list

```
list($prva, $druga) = vise ("obe")
// vrednost za $prvu biće $var1, dok za $druga je $var2
```

Povratne vrednosti promenjivih koje funkcija šalje nazad upotrebom signala response, možemo upotrebiti da ih pretvorimo u stanja vrednosti u react aplikaciji kojima ćemo da izvršavamo redirekciju na stranicu na primer prijava, home korisnika ili registraciju upotrebom react dom rutera. Kao što funkcije vraćaju varijabilne vrednosti tako mogu i da ih prime kao argumente, ne bitno da li su argumenti funkcije obični stringovi, brojevi, znači pojedinačne vrednosti, nego to mogu biti i nizovi vrednosti ili neki logički izrazi. Što prikazje kod dole

```
function myFunction($needThis, $needThisToo, $optional=null) {

   Neka radnja sa dobijenim podacima
  }

function createList (array $args = array()) {

$args += $array;
 $args += $var;

}
```

Ovo sve napominjem jer u nekim slučajevima je potrebno poslati u bazu podataka na serveru više podataka odjednom. Tačnije u našem slučaju prvo se mora poslati podatak koji će pokrenuti neki od uslova u petlji switch/case , a preostali se prosleđuju kao argumenti funkcije. Ali, samo menjanje vrednosti može se postići i na ovaj način

```
<form method="post">
   <button name="test">test</button>
   <button name="test1">test1</button>
</form>
   <?php

   if(isset($_POST['name'])){
```

```
        i ovde  postaviti da switch/case prima potrebnu vrednost za promenjivu koja
upravlja uključivanjem
        određenog slučaja
    }
    ?>
ili
include (‘putanja do faila sa funkcijama koje se pozivaju/fail sa funkcijama’);

<button onclick=”<?php PozvnaFunkcija();?>”>Poziv funkcije</button>
```

Da bi smo pozvali određenu funkciju u našem slučaju i prosledili joj određene podatke, uzimajući da se radi o većem broju podataka, od kojih je jedan koji pokreće case, react aplikacijom potrebno je pre svega omogućiti pristup drugom portu serveru gde se nalaze php skripte. Napominjem sve moguće načine kako bi ste mogli da upotredite isto u reactu i drugim programskim govorima.

Kako radi povezivanje php i mysql

Sam način rada nebitno koliko izgledao komplikovano je jednostavan. Počinje sa uputom poziva provere da li postoje data baza, potrebna tabela u njoj, nakon čega se čeka akcija korisnika. Da bi smo upisali nešto u data bazu potrebno je u formi napraviti potrebna polja, dodeliti im imena (name) i na kraju potrebna dugmad kojima bi se pozvala akcija i primenio jedna od metoda (GET, POST,PUT, DELETE). Kada se radi o istom veb prostoru i istom portu (standardno za php je Apache port 80, mysql 3600), nije potrebno nista drugo. Međutim, react aplikacije rade, kao što se vidi na portu 3000. Mada, bez ikavih problema mogu da rade na bilo kom portu, to se može podesti. Ako pogledate u prvom delu knjige gde je bilo reči o Nodejs. Bitno je da na serveru gde se kompletna aplikacija nalazi, ako je to potrebno izvrše podešavanja. Ali, ono što je najbitnije je to da se u index.php failu nađu CROS dozvole upisane u hederima:

```
Dozvola za tazmenu podataka koje upotrebljavaju JSON tip podataka
header('Accept: application/json');
Dozvola za upotrebu metoda

header('Access-Control-Allow-Methods: POST,GET,PUT,DELETE,OPTIONS');

header("Access-Control-Allow-Headers: eToken,X-Requested-With,Content-
Type");

Dozvola za tip sadržaja json sa kalrakter setom utf-8

header('Content-type: text/json; charset=utf-8; application/x-www-form-
urlencoded');
```

Nesigurna jer dozvoljava pristup svemu i svim podacima i zahtevima
header ('Access-Control-Allow-Origin: *');

Ovo je deo onog što postoji, što sam upotrebio u ovoj knjizi. Više o njima na:
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Origin
https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
Napomena: Budite oprezni sa ovim dozvolama jer od toga zavisi sigurnost budućeg vebsajta koji izrađujete.

Fail konekcija.php

Povezivanjem sa bazom podataka je vrlo jednostavno

```
<?php
/**
 * Created by IntelliJ IDEA.
 * User: trika
 * Date: 27-Aug-18
 * Time: 23:16
 */
$locahost = 'localhost'; // host gde se nalazi veb sajt
$admin = 'root';     // ime administratora
$passw ='nekaSifra'; // njegova šfira
$baza='test';   // naziv upotrebljavane baze podataka

/* smeštanje i promenjivu podataka o mestu i nazivu baze podataka*/
$mysqli = new mysqli( $locahost, $admin, $passw, $baza);

/* provera konekcije */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

?>
```

Umesto localhost upotrebićete naziv domena gde se postavlja aplikacija, kao i za ostale druge podatke postavićete one koji vam budu u zahtevu za izradu aplikacije. Ovaj fail normalno uključujete u drugim failovima sa:

```
include ('konekcija.php');
```

Onamo gde vam je potrebno da ga pozovete, u ovom slučaju najbolje je da se pozove u fail funkcije.php jer će one imati često potrebu da kontaktiraju sa bazom.

U ovom failu možete da dodate upite o postojanju baze i tabela u njoj, ali ako ste ih već izradili to je izlišno.

Funkcija za čitanje podataka iz baze

Ova funkcija za svoj rad zahteva konekciju na bazu i tabelu iz koje čita podatke o ona te podatke vraća na mesto gde je pozvana u obliku JSON formata.

```
<?php
$result = mysqli_query($con,"SELECT * FROM Persons");
echo "<table border='1'>
<tr>
<th>Firstname</th>
<th>Lastname</th>
</tr>";

while($row = mysqli_fetch_array($result))
{
echo "<tr>";
echo "<td>" . $row['FirstName'] . "</td>";
echo "<td>" . $row['LastName'] . "</td>";
echo "</tr>";
}
echo "</table>";

mysqli_close($con);
?>
```

Ovo je jedan opšti pristup problemu, gde bi smo iz reacta poslali zahtev sa nazivom tabele, nizom podataka o imenima potrebnih polja koja se isčitavaju, kao i korisnički $uid koji je u stvarnosti ček boks, koji bi se proverio sadržajem promenjive $matchid, jer je ostavljeno da je on jednak ovoj promenjivoj.
U principu nebitno da li čitamo, dodajemo, brišemo prepravljamo neki od podataka u bazi.
Ako je ček boks selektovan onda je to vrednost boja jedan, ako nije onda se vraća nula. To je jedna od preventive, kojom se potvrđuje namera. Na primer kod brisanja ili registracije.
Prvo se uspostavlja konekcija, nakon toga se vrši upit kojim se izabire u toj bazi određena polja određene tabele.

```
   $sql=mysql_query("select $field from $table where $matchid=$uid");
```

nakon toga se u promenjivu $row smeštaju vrednosti

```
   $row=mysql_fetch_array($sql);
```

Ovim je preuzet podatak od prvog korisnika

```
$userinfo=$row[0]; // ako se izostavi ovde nula onda se preuzimaju svi podaci koji su zadovoljili kriterijum
```

Lakši pristup rada sa podacima iz PHP-a

Obzirom da je svuda zastupljeno objektno programiranje pogledajmo uporedno react i php-u .Tačnije, biće upotrebljene klase i klasični php skriptovi kojima će se pristupiti bazi podataka, uraditi obična validacija podataka, preuređivanje i dodavanje novih, kao brisanje starih podataka. Za ovo biće nam potreban jedan direktorijum sa sledećim sadržajem:

failovi: index.php, add.php, add.html, edit.php, delete.php i editaction

folder sa klasama classes o failovima klasa: Crud.php, Dbconfig.php i Validation.php.

Ako uporedim kod u reactu i ovaj ovde, mogu reći da isto postoje klase koje imaju konstruktor i njegove delove, kao i funkcije koje se pozivaju ili pozivaju druge funkcije. Kostruktor u php klasama izgleda ovako kao u kodu dole

```
public function __construct()
{
   parent::__construct();
}
```

umesto službene reči constructor() ima dvostruko podvučenu liniju iza koje sledu službena rec construct - __construct(). Razlika je upotreba ograničenja pristupa kako njemu, tako i funkcijama i promenjivama se vrši se upotrebom službenih reči: global, public i sl.
Uobičajeno ime za ovakav skup koda u failovima da se naziva CRUD, što bi bilo u prevodu izradi (create), pročitaj(read), preuredi(update) i obriši (delete). Pri ovakvom radu u php-u uobičajena je praksa da se karakteristični failovi po namenama postavljaju u svojim folderima, nalik folderima za kod komponenti u reactu. Zato su klase smeštene u svom folderu gde se on nalazi zajedno sa preostalim failovima. Ali najbolji način da se proučava ovaj deo o smeštanju failova u folderima može da se uoči kod oragizacije koda u wordpress-u.

Kod pomenutih failova

index.php

```
<?php
// pozivanje osnovne klase
include_once("classes/Crud.php");

// smestanje vrednosti u promenjivu cime se izrađuje novi objekat od podataka iz
```

```
klase kao njena istanca
$crud = new Crud();

//povezivanje sa data bazom fetching data in descending order (lastest entry first)
$query = "SELECT * FROM korisnik1 ORDER BY id DESC";

// povezivanje delova klase vezanim za preuzimanje podataka iz baze, kako bi se
// funkcijama u kalsi mogle izvršiti promene nad podacima
$result = $crud->getData($query);

//echo '<pre>'; print_r($result); exit; provera ispravnosti
?>
<! – html kod -- >
<html>
<head>
   <title>Glavna Strana</title>
</head>
<body>
<a href="add.html">Dodavanje novog korisnika</a><br/><br/>
<table width='80%' border=0>
   <tr bgcolor='#CCCCCC'>
      <td>Korisničko ime </td>
      <td>Korisničko perzime </td>
      <td>Dodaj</td>
   </tr>
   <?php

// prikaz svih podataka iz baze poređanih od poslednjeg unetog ka prvom unetom
// putem foreach petlje na ključeve i vrednosti unetih vrednost i polja u bazi
  foreach ($result as $key => $res) {
      //while($res = mysqli_fetch_array($result)) {
      echo "<tr>";

//  sitem je u celiji prikaži vrednost polja ['first_name'] kao niz ključa $res
      echo "<td>".$res['first_name']."</td>";
      echo "<td>".$res['last_name']."</td>";
      //echo "<td>".$res['email']."</td>";

// dodaj linkove prema failovima koji vrše promenu nad unetim poljima
         echo "<td><a  href=\"edit.php?id=$res[id]\">Preuredi</a> | <a
href=\"delete.php?id=$res[id]\"
                onClick=\"return confirm('Da li si sigruan?')\">Izbriši</a></td>";
   }
   ?>
```

```
</table>
</body>
</html>
```

dbconfig.php

```
<?php

/**
 * Created by IntelliJ IDEA.
 * User: trika
 * Date: 12-Nov-18
 * Time: 13:59
 */
class Dbconfig
{
/* povezivanje sa bazom u ovom slučaju na lokahostu, ali kada ide na server onda se unosi domain, obzirom da su ovi podaci bitni te tako moraju biti sugirni i osigurani, to za njihove promenjive u koje se smeštaju ove vrednosti je tipa private, a konekcija je zaštićena */

    private $_host = 'localhost';
    private $_username = 'root';
    private $_password = '';
    private $_database = 'reactphp';

    protected $connection;

    public function __construct()
    {
// ako je konekcija uspostavljena onda je prvi uslov
        if (!isset($this->connection)) {

// prvi uslov ispunjen
            $this->connection = new mysqli($this->_host, $this->_username,
                $this->_password,
                $this->_database);

// ako nije pojavljuje se prikazivanje greške o problemu konekcije
            if (!$this->connection) {
                echo 'Nemogu se povezati na databaze server';
                exit;
            }
        }
```

```
// povratak rezultata rada klase dbconfig.php
        return $this->connection;
    }
}
```

Crud.php

```
<?php
// uključivanje klase 'dbconfig.php'
include_once 'dbconfig.php';

// proširivanje klase Crud sa klasom 'dbconfig.php', čime se i njeni povratni
// rezultati dodaju ovoj klasi
class Crud extends DbConfig
{
    public function __construct()
    {
        parent::__construct();
    }

// funkcija kojoj mogu svi da pristupe namenjena za preuzimanje podataka, koja
// dobija argument ($query) i vraća niz podataka preko promenjive $rows
    public function getData($query)
    {

// povezivanje na bazu i upotreba upita query kroz argument funkcije ($query) i
// prolseđivanje dobijenih vrednosti promenjivoj $result
        $result = $this->connection->query($query);

// ispitivanje uslova ako je dobijena vrednost u promenjivoj $result prazna ili null
// onda se vraća grešla  (false)
        if ($result == false) {
            return false;
        }

// dodela vrednosti promenjivoj da je ona time postala tip podatak niz
        $rows = array();

// upotrebom while petlje se promenjivoj $row dodeljuju podaci u obliku
// asocijativnog niza, čije se vrednosti smetaju u niz $rows, koju ova funkcija vraća
// nazad
        while ($row = $result->fetch_assoc()) {
            $rows[] = $row;
```

```
        }
        return $rows;// povratna vrednost funkcije
    }

// funkcija koja vraća kao rezultat svog rada ako je sve bez greške argument
// predhodnoj
    public function execute($query)
    {
        $result = $this->connection->query($query);

        if ($result == false) {
            echo 'Error: cannot execute the command';
            return false;
        } else {
            return true;
        }
    }

// funkcija za brisanje podataka koja uz pomoć dve dobijene vrednost i argumenta
// briše ciljni podatak, uzimajući da je id jednistven za svaku kolonu u bazi
// podataka
    public function delete($id, $table)
    {
        $query = "DELETE FROM $table WHERE id = $id";

        $result = $this->connection->query($query);

        if ($result == false) {
            echo 'Greška: nemogu obrisati id ' . $id . ' u tabeli ' . $table;
            return false;
        } else {
            return true;
        }
    }

// ova funkcija uklanja specijalne znakove iz poslatog stringa u unosnim poljima
// zavisno od upotrebljenog karakter seta znakova (charset)
    public function escape_string($value)
    {
        return $this->connection->real_escape_string($value);
    }
}
?>
```

Validate.php

```
<?php
class Validation
{

// funkcija za proveru da li su unosna polja prazna ili ne, koja vraća porukom ako je
// neko pollje prazno
    public function check_empty($data, $fields)
    {
        $msg = null;
        foreach ($fields as $value) {
            if (empty($data[$value])) {
                $msg .= "$value polje nije popunjeno <br />";
            }
        }
        return $msg;
    }

   /* funkcija koja uslovljava samo upotrebu isključivo brojeve kao unosne
      vrednosti
public function is_age_valid($age)
    {
        //if (is_numeric($age)) {
        if (preg_match("/^[0-9]+$/", $age)) {
            return true;
        }
        return false;
    }*/

// funkcija koja prihvata kao unosne verdnosti slovne znakove
    public function imena_valid($age)
    {
        if (preg_match("/^[[a-zA-Z]+$/", $age)) {
            return true;
        }
        return false;
    }

// funkcija koja vrši proveru ispravnosti unetog emaila
    public function is_email_valid($email)
    {
if (preg_match
     ("/^[_a-z0-9-+]+(\.[_a-z0-9-+]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*(\.[a-z]{2,4})$/",
```

```
            $email)) {
        if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
            return true;
        }
        return false;
    }
}
```

add.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Dodavanje podataka</title>
</head>
<body>
<a href="index.php">Glavna</a>
<br/><br/>

<form action="add.php" method="post" name="form1">
    <table width="25%" border="0">
        <tr>
            <td>Ime korisnika</td>
            <td><input type="text" name="name"/></td>
        </tr>
        <tr>
            <td>Prezime</td>
            <td><input type="text" name="last_name"/></td>
        </tr>
        <tr>
            <td></td>
            <td><input type="submit" name="Submit" value="Add"></td>
        </tr>
    </table>
</form>
</body>
</html>
```

Za kod ovog faila možete upotrebiti fail sa funkcijama jer on je jedna forma gde je zbog lakšeg pozicioniranja u prostoru uzeta tabela. Namenjen za dodavnje novog podatka u bazi

add.php

```
<html>
<head>
   <title>Add Data</title>
</head>
<body>
<?php

//Uključivanje potrebnih klasa za rad ovog faila
include_once("classes/Crud.php");
include_once("classes/Validation.php");

// dodavanje vrednosti novih objekata u promenjive koje se upotrebljavaju u daljem
// radu
$crud = new Crud();
$validation = new Validation();

// ako je poslato submit onda se vrši prvo brisanje specijalnih znakova iz unosa,
// zatim provera ispravnosti unetih podataka
if(isset($_POST['Submit'])) {
    $name = $crud->escape_string($_POST['name']);
    $age = $crud->escape_string($_POST['last_name']);

   // $email = $crud->escape_string($_POST['email']);
// validacija
    //$msg = $validation->check_empty($_POST, array('name', 'age', 'email'));

    $msg = $validation->check_empty($_POST, array('name', 'last_name'));
    //$check_age = $validation->is_age_valid($_POST['age']);
    $check_age = $validation->imena_valid($_POST['last_name']);
   // $check_email = $validation->is_email_valid($_POST['email']);

    // provera da li su polja prazna
    if($msg != null) {
        echo $msg;

        //povezivanje sa perdhdnom starnicom
        echo "<br/><a href='javascript:self.history.back();'>Go Back</a>";
    } elseif (!$check_age) {
        echo 'Please provide proper age.';
    } /*elseif (!$check_email) {
        echo 'Please provide proper email.';
    }*/
    else {
```

```
        // ako su sva unosna polja ispravna onda se podaci unose u bazu podataka
        $result = $crud->execute("INSERT INTO korisnik1(first_name,last_name)
                VALUES('$name','$age')");

        //Prikaz uspešno unetih podataka kao i povratak na index.php stranicu
        echo "<h2 style='color:green'>Data added successfully.</h2>";
        echo "<br/><a href='index.php'>Pregled svih unetih rezultata</a>";
    }
}
?>
</body>
</html>
```

edit.php

```
<?php
include_once("classes/Crud.php");
$crud = new Crud();
//preuzimanje id iz url-a
$id = $crud->escape_string($_GET['id']);

/*izbor podataka sa ciljnom vrednošću id polja, nekada možda vam zatreba u
formama da izvrši podatak da li je neko saglasan sa uslovima prijave na budućem
nalogu na sajtu ili vam je potrebno brisanje nekog posta ili nekog korisnika tada se
može dodati rezultat koji se prihvata iz ček boksa. Selektovan jednak je vrednosti
boja jedan, suprotno ima vrednost nule. Onda se može dodati deo koda koji bi
brisao sve ono što ima vrednost nula */

$result = $crud->getData("SELECT * FROM korisnik1 WHERE id=$id");
// petlja kojom se iz niza podatak izdvajaju pojedinačno članovi niza
foreach ($result as $res) {
        $name = $res['first_name'];
        $age =    $res['last_name'];
    // $email = $res['email'];
}
?>
<html>
<head>
    <title>Prepravka podataka</title>
</head>
<body>
<a href="index.php">Glavna</a>
<br/><br/>
```

```
<form name="form1" method="post" action="editaction.php">
   <table border="0">
      <tr>
         <td>Name</td>
         <td><input type="text" name="first_name"
               value="<?php echo $name;?>"></td>
      </tr>
      <tr>
         <td>Age</td>
         <td><input type="text" name="last_name"
             value="<?php echo $age;?>"></td>
      </tr>
      <tr>
         <td><input type="hidden" name="id"
              value=<?php echo $_GET['id'];?>></td>
         <td><input type="submit" name="update" value="Update"></td>
      </tr>
   </table>
</form>
</body>
</html>
```

ovaj fail ispisuje vrednosti polja iz baze onog čiji id odgovara izabranom, tako što se omogućava njihova izmena o upis u bazu.

fail editaction.php

```
<?php
// including the database connection file
include_once("classes/Crud.php");
include_once("classes/Validation.php");

$crud = new Crud();
$validation = new Validation();

if(isset($_POST['update']))
{
   $id = $crud->escape_string($_POST['id']);
   $name = $crud->escape_string($_POST['first_name']);
   $age = $crud->escape_string($_POST['last_name']);
   //$email = $crud->escape_string($_POST['email']);

   $msg = $validation->check_empty($_POST, array('first_name', 'last_name'));
   $check_age = $validation->imena_valid($_POST['last_name']);
```

```
    //$check_age = $validation->is_age_valid($_POST['last_name']);
  //  $check_email = $validation->is_email_valid($_POST['email']);

    // Provera da li su unosna polja prazna
    if($msg) {
        echo $msg;
        //Link za vraćanje na predhodnu stranicu upotrebom istorije
        echo "<br/><a href='javascript:self.history.back();'>Korak nazad</a>";
    }/*  elseif (!$check_age) {
        echo 'Please provide proper age.';
    }elseif (!$check_email) {
        echo 'Proverite da li je unet email ispravan.';
    }*/ else {
        //updating the table
        $result = $crud->execute("UPDATE korisnik1 SET
                    first_name='$name',last_name='$age'
                     WHERE id=$id");
        //nakon izvršenog posla vraćamo se na index.php
        header("Location: index.php");
    }
}
?>
```

Principijelno ovaj fail ima skoro istu funkciju kao add.php, gde se vrši mysqli INSERT, s time što u ovom slučaju vrši UPDATE podataka u bazu.

fail delete.php

```
<?php
//including the database connection file
include_once("classes/Crud.php");

$crud = new Crud();

//getting id of the data from url
$id = $crud->escape_string($_GET['id']);

//deleting the row from table
//$result = $crud->execute("DELETE FROM users WHERE id=$id");
$result = $crud->delete($id, 'korisnik1');

if ($result) {
    //redirecting to the display page (index.php in our case)
    header("Location:index.php");
```

```
}
?>
```

Kao i edit.php kod i ovaj kod radi slično, s time što kada pronađe ciljni id podataka on pristupa brisanju svih podataka iz baze vezanih za taj id iz čega vrši redirekciju na index.php.

Zaključak

Kao što je napomenuto slično se dešava i u reactu i bilo kom drugom programskom govoru, jer da bi se nešto radilo potrebno mu je pristupiti na način koji ono razume, to jest na način koji je njegov tvorac to osmislio. Inače dolazi do problema. Zato je važno imati širinu prostupa i brojne puteve ka jednom cilju, to jest dobro napisanoj aplikaciji - programu koji će da radi sigurno, bezbedno i jednostavno. A njegove delove a možete da primenite svuda. Takav način rad donosi puno vremena za opuštanje i zadovoljstva, ali uvek treba biti na oprezu i svoje znanje razmenjivati sa okolinom po principu poštene razmene jer nema nikog sveznajućeg, već svi ponešto znamo.

Uzajamno react i php

Ono što je još potrebno obzirom da je React i sve oko njega nešto izmenjeni java skript, je da se sesije stvaraju kod korisnika na njegovom računaru, a da se podaci iz sesija koje se smeštaju u localStorage putem odjave ili u toku rada korisinka prenose na server ili sa servera prikazuju kod korisnika u zavisnosti od zahteva korisnika. Ovaj postup može biti sihronizovan ili ne sihornizovan.
Ali, posle prijave potrebno je preusmerenje na našu stranicu (kao što je to na primer kod facebook-a). Znači da je potrebno izraditi jedan fail sa ruterom, koji bi bio pozvan u slučaju da uneti podaci odgovaraju podacima u bazi podataka ili ako ne odgovaraju da nas vrati na stranicu za prijave ili registraciju.

Jednostavna React CRUD mogućnost

```
import React, {Component} from 'react';
import './app.css';

class App extends Component {

  constructor(props) {
    super(props);
    this.state = {
      Naziv: 'Jednostavna CRUD react app',
      act: 0,
      index: '',
      podaci: []
    }
```

```
}

componentDidMount() {
   this.refs.name.focus();
}

fSubmit = (e) => {
   e.preventDefault();
   let podaci = this.state.podaci;
   let name = this.refs.name.value;
   let prezime = this.refs.prezime.value;
   if (this.state.act === 0) {   //unos novog podatka
      let data = {
         name, prezime
      };
      podaci.push(data);
   } else {                    //preuredjivanje starog
      let index = this.state.index;
      podaci[index].name = name;
      podaci[index].prezime = prezime;
   }

   this.setState({
      podaci: podaci,
      act: 0
   });

   this.refs.myForm.reset();
   this.refs.name.focus();
};

Izbrisi = (i) => {
   let podaci = this.state.podaci;
   podaci.splice(i, 1);
   this.setState({
      podaci: podaci
   });

   this.refs.myForm.reset();
   this.refs.name.focus();
};

Preuredi = (i) => {
   let data = this.state.podaci[i];
```

```
    this.refs.name.value = data.name;
    this.refs.address.value = data.prezime;
    this.setState({
      act: 1,
      index: i
    });
    this.refs.name.focus();
};

render() {
    let podaci = this.state.podaci;
    return (
      <div className="App">
        <h2>{this.state.Naziv}</h2>
        <form ref="myForm" className="myForm">
          <input type="text" ref="name" placeholder="Unesi svoje ime"
            className="formField"/>
          <input type="text" ref="prezime" placeholder="Unesi svoje prezime"
            className="formField"/>
          <button onClick={(e) => this.fSubmit(e)}
            className="myButton">submit</button>
        </form>
        <table className="table-bordered myForm">
          <tr>
            <th>ID</th>
            <th>Ime</th>
            <th>Prezime</th>
          </tr>
          {podaci.map((data, i) =>
            <tr key={i} className="myList">
              {i + 1}.
              <td>{data.name}</td>
              <td> {data.prezime}</td>
              <td> </td>
              <td>
                <button onClick={() => this.Izbrisi(i)}
                  className=" btn-danger">Izbrisi</button>
              </td>
              <td>
                <button onClick={() => this.Preuredi(i)}
                  className=" btn-success">Preuredi</button>
              </td>
            </tr>
          )}
```

```
            </table>
          </div>
        );
    }
}
export default App;
```

Još jedan primer react forme

Ovaj primer formi je objedinjen za vršenje prijave i ragistracije korisnika na veb prostor. Šta je u ovom primeru karakteristično?
Prvo ovaj primer u sebi sadrži obe forme, od kojih je prva za prijavu vidljiva, dok druga sakrivena, ali klikom na njeno dugme pojavljuje se ona i menja se boja dugmadi, plava označava da je operacija aktivna, crvena da nije. Što je postignuto dodavanjem koda u delu za dugme

```
<button className={(this.state.isLoginOpen? "btn-primary" : "btn-danger")}
```

i promenom stanja od početnog u stanje koje se zadaje funkcijama showLoginBox i showRegisterBox.
Način na koji se menja boja dugmadi ili možete isto uraditi i za okvir forme je promena if else uslova u opisu klase za dugme koja se menja u zavisnosti od logičkog stanja bokseva.
Kao što se vidi početno stanje u konstruktoru je za Login(Prijavu) je istina, dok za Registraciju je neistina. U kodu klase za dugme doslovno piše stanje za ono što je otvoreno, postavi u prvu boju, kada je zatvoreno u drugu.

```
 if (this.state.isLoginOpen===true) {
   "btn-primary"
```

```
 }else{
   "btn-danger"
}
```

Ovo samo pokazuje koliko kod može biti kraći upotrebom react-a.

```
import React from 'react';

export default class App extends React.Component {

   constructor(props) {
      super(props);
      this.state = {
         isLoginOpen: true,
         isRegisterOpen: false
      };
   }

   showLoginBox(){
      this.setState({ isLoginOpen: true, isRegisterOpen: false })
   }

   showRegisterBox(){
     this.setState({ isRegisterOpen: true, isLoginOpen: false })
   }

   render() {
      return (
         <div className="container">
            <div className="box-controller">
               <button className={(this.state.isLoginOpen? "btn-primary"
                   : "btn-danger")}
                   onClick={this.showLoginBox.bind(this)}>
                  Prijava
               </button>
               <button className={(this.state.isRegisterOpen ? "btn-primary"
                       : "btn-danger")}
                    onClick={this.showRegisterBox.bind(this)}>
                   Registracija
               </button>
            </div>
            <div className="box-container" style={{height:"300px"}}>
               {this.state.isLoginOpen && <LoginBox/>}
               {this.state.isRegisterOpen && <RegisterBox/>}
```

```
            </div>
          </div>
      );
    }
}
```

Karakteristično je u ovom primeru upotreba posebnog načina primene praćenja promena vrednosti stanja. U ovom delu je taj način upotrebljen za promenu klasa koje ukazuju na aktivnosti, odnosno pokazuje koje je dugme aktivno

```
            <div className="box-controller">
              <button className={(this.state.isLoginOpen? "btn-primary"
                : "btn-danger")}
                 onClick={this.showLoginBox.bind(this)}>
                Prijava
              </button>
              <button className={(this.state.isRegisterOpen ? "btn-primary"
                : "btn-danger")}
                  onClick={this.showRegisterBox.bind(this)}>
                 Registracija
              </button>
            </div>
```

Ali se može u ovom primeru primeniti i dodavanje još jedne klase ispred koja može biti osnovna

```
className={“početna-klasa”+ (this.state.isRegisterOpen ? "btn-primary"
            : "btn-danger")}
```

Ovo se dešava, zato što je ovim kodom praktično prestavljena if else petlja, što znači da ovaj kod se može upotrebiti i u druge svrhe.

```
class LoginBox extends React.Component {

  constructor(props) {
    super(props);
    this.state = {

    };
  }

  submitLogin(e) {

  }
```

```
    render() {
        return (
            <div className="col-md-5" style={{border:"2px green solid"} } >
                <div className="header">
                    Prijava
                </div>
                <div className="col-lg-4">
                    <div className="input-group">
                        <label htmlFor="username">Kor.ime</label>
                        <input
                            type="text"
                            name="username"
                            className="login-input"
                            placeholder="Korisnicko ime"/>
                    </div>
                    <div className="input-group">
                        <label htmlFor="password">Kor.sifra</label>
                        <input
                            type="password"
                            name="password"
                            className="login-input"
                            placeholder="Korisnicka sifra"/>
                    </div>
                    <button style={{marginTop:"10px", marginBottom:"10px"}}
                        type="button"
                        className="btn-success"
                        onClick={this.submitLogin.bind(this)}>Login</button>
                </div>
            </div>
        );
    }
}

class RegisterBox extends React.Component {

    constructor(props) {
        super(props);
        this.state = {

        };
    }

    submitRegister(e) {
```

```
  }

  render() {
    return (
      <div className="col-md-5" style={{padding:"10px",
          border:"2px blue solid"}}>
        <div className="header">
          Registracija korisnika
        </div>
        <div className="col-lg-4">
          <div className="input-group">
            <label htmlFor="username">Kor.ime</label>
            <input
              type="text"
              name="username"
              className="login-input"
              placeholder="Korisnicko ime"/>
          </div>
          <div className="input-group">
            <label htmlFor="email">Kor.Email</label>
            <input type="text" name="email" className="login-input"
              placeholder="Korisnicki Email"/>
          </div>

          <div className="input-group">
            <label htmlFor="password">Kor.sifra</label>
            <input
              type="password"
              name="password"
              className="login-input"
              placeholder="Korisnicka sifral"/>
          </div>
          <button style={{marginTop:"10px"}}
            type="button"
            className="btn-success"
            onClick={this
              .submitRegister
              .bind(this)}>Registracija</button>
        </div>
      </div>
    );
  }
}
```

LocalStorage u react aplikacijama

Uporeba LocalStorage u veb aplikacijama može se ostvariti kroz pet metoda:
setItem(): dodavanje ključeva (key) i vrednosti (value) u LocalStorage
getItem(): Prijem podataka u obliku ključeva (key) i vrednosti (value) u LocalStorage
removeItem(): brisanje pojedinačnog podatka upotrebom vrednosti ključa (key) u LocalStorage
clear(): brisanje kompletnog LocalStorage
key(): praktično pozivanje elementa putem unosa broja njegovog ključa (key) u LocalStorage

setItem()

Ova metoda kao što naziv implicira omogućuje pohranjivanje vrednosti u LocalStorage objektu. Potrebna je dva parametra, ključ i vrijednost. Ključ se može upućivati kasnije za dohvaćanje vrednosti pridružene njoj.

```
window.localStorage.setItem('name', 'Moja vrednost');
```

Gde name je ključ i Mojavrednost njegova vrednost. Da bi smo mogli da sačuvamo nizove ili objekte moramo ih pretvoriti u JSON objekte, što se radi upotrebom JSON.stringify() metoda pre upotrebe setItem() metoda .

```
const person = {
   name: "Moja vrednost",
   location: "Neka lokacija"
}
window.localStorage.setItem('user', JSON.stringify(person));
```

getItem()

Metod getItem() omogućava pristup podacima u brovzeru localstorage objektima upotrebom vrednosti ključa da bi vratio se kao rezultat vrednosti kao string.

Prijem podataka sa user ključem (key):

```
window.localStorage.getItem('user');
```

Vraćanje string vrednosti kao value ;

```
“{name: "Moja vrednost",
   location: "Neka lokacija"}”
```

Da biste upotrebili ovu vrednost, pretvorili biste je u objekt. Kako bismo to učinili, koristimo metodu JSON.parse () koja pretvara JSON niz u Javascript objekt

```
JSON.parse(window.localStorage.getItem('user'));
```

removeItem()

Metoda removeItem () kada je prošla ključni naziv, uklonit će taj ključ iz localStorage ako postoji. Ako nema stavke povezane s datim ključem, ova metoda neće učiniti ništa.

```
window.localStorage.removeItem('name');
```

clear()

Ova metoda, kada se poziva, briše kompletno sve zapise za taj domen. Ne prima nikakve parametre.

```
window.localStorage.clear();
```

key()

Metoda key () (ključa) dolazi u situacijama u kojima morate prelaziti ključeve i omogućiti vam da unesete broj ili indeks u lokalu kako biste dohvatili naziv ključa.

```
var KeyName = window.localStorage.key(index);
```

LocalStorage podrška u borvzeru

LocalStorage kao vrsta web-pamćenja HTML5 specifikacija.Podržan je od strane glavnih pretraživača, uključujući IE8.To bi bili sigurni da pretraživač podržava LocalStorage, možete proveriti koristeći sledeći kod:

```
if (typeof(Storage) !== "undefined") {
   // Kod za localStorage
} else {
   // Nema podrške .
}
```

LocalStorage ograničenja

Kao što je lako kao što je to koristiti LocalStorage, to je također lako zloupotrebiti. Sledeća su ograničenja i načina da se ne koriste localStorage:

- Nemojte čuvati osetljive podatke o korisniku u localStorage
- Ne menja bazu podataka servera jer se podaci čuvaju samo u pretraživaču
- LocalStorage je ograničen na 5 MB na svim većim pretraživačima
- LocalStorage je prilično nesiguran jer nema oblik zaštite podataka i može se pristupiti bilo kojim kodom na vašoj web stranici.
- LocalStorage je sinhronizovan. Značenje da će svaka operacija biti izvršena jedna za drugom.

React aplikacija

Da bi smo upotrebili localstorage u aplikaciji potrebno je u konstruktoru to naglasiti u delu stanja.

```
constructor() {
   super();
   this.state = {
      radno: JSON.parse(localStorage.getItem('radno')) != null ?
               JSON.parse(localStorage.getItem('radno')) : ' '
   };
}
```

Dok, kod za dodavanje vrednosti u localStorage može se uraditi ovako:

```
Dodaj() {
   let naslov = this.refs.naslov.value;
   if (localStorage.getItem('radno') == null) {
      let radno = [];
      radno.push(naslov);
      localStorage.setItem('radno', JSON.stringify(radno));
   } else {
      let radno = JSON.parse(localStorage.getItem('radno'));
      radno.push(naslov);
      localStorage.setItem('radno', JSON.stringify(radno));
   }
   this.setState({
      radno: JSON.parse(localStorage.getItem('radno'))
   });
}
```

Kao i kod baza podataka dodavanje se može vršiti u već otvoreni postojeći fail ili bazu podataka uz predhodnu proveru da li taj izabrani fail ili baza podataka ili tabela postoje. Obzirom da se očekuje niz podataka varijabila u koju smeštamo buduće podatke je potrebno da bude odgovarajućeg tipa, kako bi smo mogli da te podatke smestimo u nju. U suprotnom dobijamo pojavu greške. Dalje u kodu vidimo dodavanje radno.push(naslov);, sa unosnog polja nakon čega sledećom redom

```
 localStorage.setItem('radno', JSON.stringify(radno));
```

praktično vršimo upis u formatizovanom JSON obliku. Samo postavljanje novog stanja vrši se upotrebom.Gde se doslovno kaže da radno ima sve vrednosti koji su zapisane u localStorage ključem radno:

```
   this.setState({
      radno: JSON.parse(localStorage.getItem('radno'))
   });
```

Uslovno da je ispunjen uslov koji je iza else. Samo brisanje podataka se vrši na sličan način s time što upotrebimo vrednost index ključa, koji ukazuje na ciljni zapis u lokalnom skladištu podataka koji bi smo da obrišemo.

Brisanje podataka u localstorage:

```
Obrisi(e) {
     let index = e.target.getAttribute('data-key');
     let lista = JSON.parse(localStorage.getItem('radno'));
     lista.splice(index, 1);
     this.setState({
        radno: lista
     });
     localStorage.setItem('radno', JSON.stringify(lista));
  }
```

U render delu se može upotrebiti neuređena lista da bi se isti prikazali.

```
render() {
   let RadnaLista = '';
   if (localStorage.getItem('radno') != null) {
      RadnaLista = (
         <ul>
            {this.state.radno.map(function (rad, index) {
               return (
                  <li key={index}>{rad} <input type="button" value="X"
                     onClick={this.Obrisi.bind(this)} data-key={index} /></li>
               );
            }, this)}
         </ul>
      );
   }
```

let RadnaLista = ' '; je nešto što se može susresti kod dosta programskih govora, ovo je način da odredimo neku promenjivu: njen tip i njenu vrednost, a da ne moramo da to činimo u delu konstruktora ili u delu state. U ovom slučaju smo odredili da promenjiva ima vrednost null ili prazno. Samo da podsetim, slično se radi kada upotrebimo određivanje promenjive ili više upotrebom destruktora.
U ovom delu se opet proverava postojanje u loklanom skladištu postojanje jednog njegovog dela, ako postoji onda se upotrebimo map metod sve unešeno se prikazuje u jednoj ne uređenoj listi, gde je i smešteno dugme kojim se poziva događaj Obrisi. Obzirom da input poljle ima tip dugmeta možete upotrebiti i tag button:

```
<button  className="btn-danger" onClick={this.Obrisi.bind(this)}
```

```
   data-key={index} >
   Obrisi
</button>
```

Pošto je ovo primer koji ćemo da u kasnijem delu knjige da upotrebimo, bez ovog dela koji će obaviti prikaz na ekranu, sada nam je to potrebno, kako bi smo videli rad koda. U delu returna izrađujemo dva unosna polja i vršimo prikaz unešenih vrednosti u lokalno skladište podataka (localStorage)

```
      return (
         <div>
            <h3>Upotreba localStorage u React app</h3>
            <input type="text" placeholder="Unosno polje..." ref="naslov" />
            <input type="button" value=" Dodaj" onClick={this.Dodaj.bind(this)} />
            <br /><br />
          Pozivanje konstante koja vrši prikaz saržaja u lokalnom skaldištu podataka
            {RadnaLista}
         </div>
      );

import React from 'react';

export default class LokalnoSkladiste extends React.Component {

U ovom delu smeštamo predhodni kompletan kod

   }
}
```

Slika prikazuje rad predhodnog koda

Za unos vrednosti više unosnih polja može upotrebiti ovaj kod

```
let naslov = this.refs.naslov.value;
let naslov1 = this.refs.naslov1.value;
let unos= naslov + naslov1;
```

Ili upotrebiti da krajnja promenjiva koja se postavlja umesto ispred promenjive naslov putem upotrebe this.setState dobije sve te vrednosti u obliku niza.

Zaključak

Lokalno skaldište (localStorage) je jedan od načina da gde se nakon prijave na nalog tu napravi korisnički deo gde korisnik može biti opslužen sa manjim angažovanjem servera. Ali nakon zatvaranja brovzera ili odjave sa naloga bilo bi potrebno u tom mehanizmu upotrebiti brisanje ovog skaldišta podataka

```
window.localStorage.clear();
```

u cilju sigurnosti. Ali pre toga sve podatke prebaciti u bazu podataka na serveru.

Prijava ili registracija na nalog

Prijava ili registracija na nalog u principu radi identično, ne binto koja je platforma u pitanju. Korisnik pronađe domen, u login formu ili registracionu formu unese potrebne podatke, aplikacija pokreće upit ka serveru, gde se pročitaju delovi baze podataka gde se nalaze podaci o registrovanim korisnicima, i ako podaci u bazi su jednaki unešenim otvara se korisnička stranica. U suprotnom javlja se poruka o problemu ili se traži ponovni unos podataka ili da se izvrši registracija novog korisnika. To je način kada samo direktno na serveru gde je domen i slučaj na primer direktnog rada sa serverom. Kada imamo slučaj korisnik server aplikacije, onda je taj put malo drugačiji. Prema predhodnim delovima ove knjige za ovaj slučaj imamo dve mogućnosti u react aplikacijama. Prva je da se podaci sa servera

prebace kod korisnika pa da onda korisnik na svom računaru radi, a da se po potrebi tih zahteva povremenom aplikacija obraća serveru. Druga je da sve ide na serveru. Koji će način biti u upotrebi, zavisi od mnogih činjenica, na primer namene aplikacije, veličine, broja mogućih korisnika i drugo.
Pomenuti deo na serveru ostao bi isti, dok kod korisnika bi imao neke manje izmene, jer bi u aplikaciju bilo potrebno uključiti lokalno skladište podataka (local Storage, možda local Session). U oba slučaja vezanih za skladište podataka pristup tim podacima je ovakav:

localStorage.setItem('jwtToken' , token); // za upis podataka

ili

localStorage.getItem('jwtToken' , token); // za čitanje podataka

Po mnogima ovo je nesigurno, ali je dobro znati da se i to može primeniti u aplikaciji. Ovaj prostor može biti keširan ili ne, ali i sihronizovan, što opet zavisi od potrebe same aplikacije. Ako se vratimo na predhodni deo “React aplikacija“, kao i deo za izradu forme prijava /registracija pre njega, možemo napraviti kombinaciju koda koja bi radila zajedno sa php skriptom i mysql bazom. Podaci iz baze preko php skripte bili bi vraćeni nazad korisniku i smešteni u lokalnu memoriju brovzera (localStorage). Ali vratimo se delu prijava na nalog. Prijava se može obaviti upotrebom ovog kostura koda

```
class LoginBox extends React.Component {

   constructor(props) {
      super(props);
      this.state = {
        korisnik:'',
        tacno:"moje"
      };
   }

   Promena= e=>{
    this.setState({[e.target.name]: e.target.value});
   };

   submitLogin(e) {
     e.preventDefault();

      if( this.state.korisnik !== this.state.tacno){
         alert("pogresno");
```

```
        }else{
            console.log(this.state.korisnik);
              this.setState({value:""});
            alert("kliknuto");
        }

        console.log(this.state.korisnik);

    }
    render() {
        return (
            <div className="col-md-5" style={{border:"2px green solid"} } >
                <div className="header">
                    Prijava
                </div>
                <div className="col-lg-4">
                    <div className="input-group">
                        <label htmlFor=" korisnik ">Kor.ime</label>
                        <input
                            type="text"
                            name="korisnik"
                            className="login-input"
                            placeholder="Korisnicko ime"
                            value={this.state.name}
                            onChange={this.Promena.bind(this)}/>
                    </div>
                    <button style={{marginTop:"10px", marginBottom:"10px"}}
                            type="button"
                            className="btn-success"
                            onClick={this.submitLogin.bind(this)}>Login</button>
                </div>
            </div>
        );
    }
}
```

Nakon klika na dugme Login kroz upotrebu if else uslova

```
submitLogin(e) {
      e.preventDefault();
```

/* ako je uneta vrednost za promenjivu korisnik različita od onog što smo postavili za vrednost stanja promenjive tačno u delu za stanja konstruktora izvršava sed prvi uslov, u suprotnom izvršava se drugi uslov iz else*/

```
    if( this.state.korisnik !== this.state.tacno){
       alert("pogresno");
    }else{
       console.log(this.state.korisnik, “Iz else”);
       alert("kliknuto");
    }
    console.log(this.state.korisnik, “Iza else”);
  }
```

U oba slučaja izvršava se i alert da bi nam ukazao šta je urađeno. Umesto alerta mogu se pozvati funkcije. U prvom redu poziv pre ove petlje pozivaju se funkcije koja pozivaju našem slučaju php skriptu za očitavanje i upoređivanje unetih podataka sa onima u bazi podataka ako su ispravno uneti podaci potrebno je pokrenuti uslov iza else, gde bi se dobili podaci za redirekciju na home stranicu korisnika i upisivanje njegovih podataka kod njega u memoriju brovsera (localStorage ili localSession) ili u sesiji na serveru čime bi bio omogućen rada korisnika. Znači ako je php vratio podatke onda se nastavlja dalje sa prijavom u suprotnom on šalje opis problema- grešku da traženi korisnik ne postoji, a react u tom slučaju vrši redirekciju na stranicu koju smo odredili za taj slučaj.

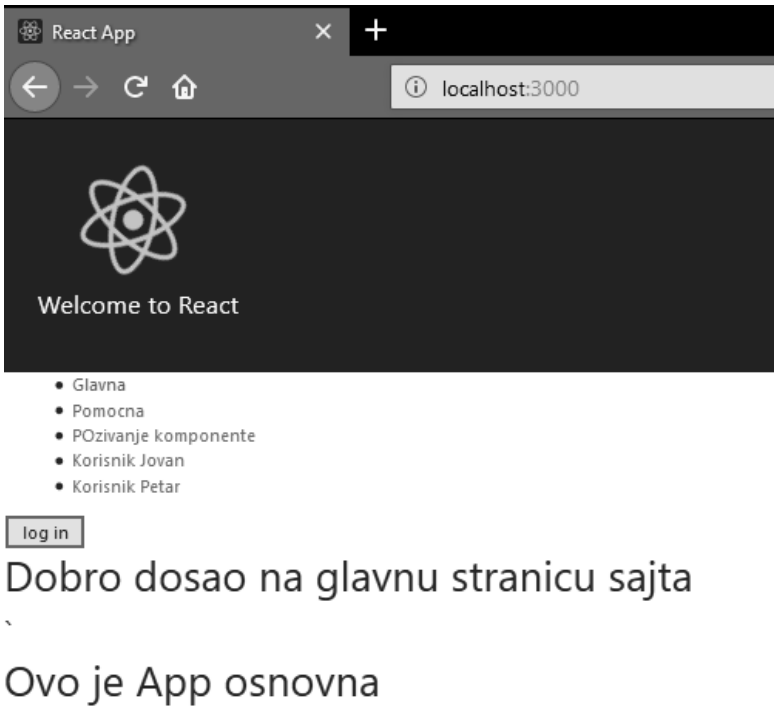
Malo drugačija upotreba rutera

Za kretanje po react aplikacijama uobičajena je upotreba dodatnog react paketa react-router-dom i react-router. Kojima se praktično pozivaju komponente da bi se prikazale. Ali, nemora uvek da se vrši povezivanje ovih paketa upotrebom u index.js failu i failu koji sadržava meni veb sajta, već je moguće napraviti jedan ovako fail bilo gde u aplikaciji.

```
mport React, { Component } from 'react';

import { BrowserRouter as Router, NavLink, Redirect, Prompt} from
 'react-router-dom';
import Route from 'react-router-dom/Route';
import Forma from'./forma';
const User = (params) => {
   return ( <h1> Dobar dan Prijatelju {params.username} </h1>)
};

class App extends Component {
   state = {
      loggedIn:false
   };
   loginHandle = () => {
      this.setState(prevState => ({
         loggedIn: !prevState.loggedIn
```

```
    }))
};
render() {
    return (
        <Router>
            <div className="App">
                <ul>
                    <li>
                        <NavLink to="/" exact activeStyle={
                            { color:'green' }
                        }>Glavna</NavLink>
                    </li>
                    <li>
                        <NavLink to="/about" exact activeStyle={
                            { color:'green' }
                        }>Pomocna</NavLink>
                    </li>
                    <li>
                        <NavLink to="/Forma" exact activeStyle={
                            { color:'green' }
                        }>Pozivanje komponente</NavLink>
                    </li>
                    <li>
                        <NavLink to="/user/jovan" exact activeStyle={
                            { color:'green' }
                        }>Korisnik Jovan</NavLink>
                    </li>
                    <li>
                        <NavLink to="/user/petar" exact activeStyle={
                            { color:'green' }
                        }>Korisnik Petar</NavLink>
                    </li>
                </ul>
                <Prompt
                    when={!this.state.loggedIn}
                    message={(location)=>{
                        return location.pathname.startsWith('/user') ?
                      'Da li si siguran da bas to želis?' : true
                    }}
                />

                <input type="button" value={this.state.loggedIn ? 'log out': 'log in'}
                  onClick={this.loginHandle.bind(this)}/>
```

```
                <Route path="/" exact strict render={
                    ( ) => {
                        return ( <h1>Ovo je glavna stranica</h1>);
                    }
                }/>
                <Route path="/about" exact strict render={
                    ( ) => {
                        return ( <h1>Ovo je pomocna stranica</h1>);
                    }
                }/>
                <Route path="/Forma" exact strict component={Forma}/>
                <Route path="/user/:username" exact strict render={({match})=>(
                    this.state.loggedIn ? ( <User
                   username={match.params.username}/>) : (<Redirect to='/' />)
                )}/>
            </div>
        </Router>
    );
  }
}

export default App;
```



Ovaj kod omogućava svojom primenom na bilo kojoj stranici da se krećete uz pomoć delova koda koji sadrže kod za Glavnu stranicu, Pomoćnu, ali ne i kada kliknete na imena korisnika, sve dok ne kliknete na dugme Login.

```
<input type="button" value={this.state.loggedIn ? 'log out': 'log in'}
  onClick={this.loginHandle.bind(this)}/>

<Route path="/user/:username" exact strict render={({match})=>(
    this.state.loggedIn ? (
 <User username={match.params.username}/>) : (<Redirect to='/' />)
)}/>
```

Ako pogledamo input polje primetno da je u delu za njegovu vrednost upotrebljen skraćen način if else petlje, koji je povezan sa vrednošću stanja za loggedIn. Čija je početna vrednost postavljena u osnovnoj vrednosti stanja

```
state = {
    loggedIn:false
};
```

kao neistina. Koja se menja pozivom koda ispod

```
loginHandle = () => {
    this.setState(prevState => ({
        loggedIn: !prevState.loggedIn
    }))
};
```

kojim menjamo trenutno stanje novim. Ako pogledamo kod u delu <Rute /> .

```
this.state.loggedIn ? ( <User username={match.params.username}/>)
                        : (<Redirect to='/' />)
```

Ovaj kod nam govori ako je stanje logedIn aktivno onda će se aktivirati komponenta User u suprotnom bićemo preusmereni na stranicu čije ime postavimo u delu iz kose crte.

```
(<Redirect to='/ime-stranice-na-koju-se-preusmeravamo' />)
```

U našem slučaju ako su uneti podaci u login formu tačni, nakon provere potrebno je da nas aplikacija preusmeri na našu osnovnu stranicu, pri tome da pokrene localStorage sa imenom korisnika. Ali da i na toj našoj stranici postoji mogućnost za redirekciju na osnovnu stranicu veb sajta gde imamo nalog. Te je potrebno

napraviti jednu funkciju koja bi pozvala funkciju za brisanje localStorage iza čega bi izvršila redirekciju. Redirekcija ili prusmeravanje ima ovakvu sintkasu:

```
<Redirect from =”/ime-stranice–sa-koje-se-preusmerava”
    to='/ime-stranice-na-koju-se-preusmeravamo' />
```

Ono što je još potrebno pre toga je to da mi neznamo koji je korisnik zahtevao prijavu, neznamo njegove podatke, te je to potrebno dodati kao novu vrednost stanja za te potrebe. Ove vrednosti bi smo dobili kao povratnu (response) informaciju sa servera iz baze podataka. Mada, se može izvršiti i redirekcija i jednostavnijim načinom, upotrebom skraćene if else petlje u return delu potrebnog faila koji se pokreće ako su uneti podaci istiniti u unosnoj formi.

```
import React from 'react';

export default class App extends React.Component {

   constructor(props) {
      super(props);
      this.state = {
         korisnik: '',
        };
   }

   LoginIn(korisnik){
      if (korisnik === this.state.korisnik){
         this.setState({ korisnik: this.state.korisnik});
      }else{
         this.setState({ korisnik: 'nnn'});
      }
      return korisnik
   }

   LogOut(korisnik){
      this.setState({ korisnik: ''});
      return korisnik;
   }
render(){
   console.log( this.state.korisnik," u renderu");
   return (
      <div>
         {!this.state.korisnik ? (<Home/>) : (<Login />)}
         <button onClick={this.LoginIn.bind(this)}>Login</button>
         <button onClick={this.LogOut.bind(this)}>LogOut</button>
```

```
      </div>
    );
  }
}

const Home=()=>{
  return(
     <div> <h2>Home</h2></div>
  )
};

const Login=()=>{
  return(
     <div> <h2>Login</h2></div>
  )
};
```

Princip rada ovog koda je krajnje jednostavan, klikom na dugmad menja se vrednost početnog stanja promenjive korisnik. Login prvo vrši proveru da li je i dalje vrednost stanja promenjive identicna početnoj ili je uneta neka vrednost. Ako je nepromenjena vrednost onda vraća početnu stranicu Home, ako je suprotno onda vraća Login stranicu. LogOut vraća Home stranicu vraćajući vrednost stanja promenjive korisnik na početno stanje, to jest na prazan string. Samo povezivanje na localStorage i rad sa njime vrši se dodavanjem koda u predhodni i delu za logIn funkciju

```
   this.setState({korisnik: 'Stanje korisnika broj dva '});
   localStorage.getItem('primer'); // otvaranje memorijskog prostora sa imenom
                                   // primer
   let primer  = [];  // proglašenje u ovom slučaju da je to niz
   primer.push(this.state.korisnik);  //dodavanje nove vrednosti u memorijskom
                                      //prostoru primer
    console.log(this.state.korisnik); // kontrola novog stanja
```

U delu LogOut funkcije

```
LogOut(korisnik){
   this.setState({ korisnik: ''});
   localStorage.clear(); // brisanje localStorage
   return korisnik;
}
```

U delu render/return

```
render(){
        localStorage.setItem('primer', JSON.stringify(this.state.korisnik)); // upis
novih podataka
   return (
      <div>
         {!this.state.korisnik ? (<Home/>) : (<Login />)}
              // prikaz upisanih podataka u localStorage
         <h2>{ JSON.parse(localStorage.getItem('primer'))}</h2>
         <button onClick={this.LoginIn.bind(this)}>Login</button>
         <button onClick={this.LogOut.bind(this)}>LogOut</button>
      </div>
   );
```

Primer upotrebe React context-a

U ovom primeru koda praktično je omogućeno da se izvrši Prijava i odjava sa naloga i upis podataka u localStorage upotrebom React Context-a, Fragment-a. Imamo dodatu komponentu koja se vidi posle klika na Login dugme pod nazivom Nova. Njeno povezivanje na mestu u komponenti Login je jedan deo posla, da bi se ona pojavila nakon klika na dugme, a drugi deo posla je njeno povezivanje sa podacima, sa jedne strane one koji se dobijaju putem Kontext-a i drugi koje možete dodati sami. Podaci koji su dobijeni posrestvom kontext-a postavljeni su izmedju tagova

<MyContext.Consumer>.....</MyContext.Consumer>

između kojih se prvo pozivaju podaci u obliku objekta value, koji se kasnije destrutivnom metodom rastavlja i vrši njegova primena prema potrebi. Putem njega mogu se proslediti više podatka, koji se primenom destruktora mogu prikazivati različito. Tačnije gde vam je potrebno prikazati koji podatak u kojoj komponenti, kasnije između tagova

<React.Fragment>.....</React.Fragment>

Između ovih tagova se mogu dodati i drugi podaci koji nisu prosleđeni iz Context-a. Ti drugi potrebni podaci mogu se proslediti putem ove komponente i posle prvog return-a i oni mogu biti ovako statični, ali mogu biti i dinamički. Što se postiže prosleđivanjem state/props koje je zastupljeno u reactu.

Kod fail loginfragment.js

```
import React from'react';

const Nova=()=>{
   return(
```

```
        <div className="col-md-2 btn-info">
          <MyContext.Consumer>
            {(value) => {
              const {korisnik, logOut} = value;

// čitanje podataka iz localStorage čije su očitane vrednosti postavljenje u konstantu
// i prikazuju se ispod
              const pera =JSON.parse(localStorage.getItem('pera'));
              return (
                <React.Fragment>
                  <h3> Prijavljen korisnik: {korisnik}</h3>
                  <h2 className="text-info"> nova komponenta u Loginu</h2>
                  <h2 className="text-info">
                    Pročitani podaci iz localStorage koji su smesteni u konstantu
                     {pera}
                  </h2>
                  <p className="text-warning">Paragraf iz nove komponente</p>
                  <button className="btn-success"
                    onClick={logOut}> Logout</button>
                </React.Fragment>
              )
            }
          }
          </MyContext.Consumer>
        </div>
    )
};

// izrada konteksta
const MyContext = React.createContext();

// izrada provajdera
class Provider extends React.Component{

// smeštanje onog što se prosleđuje provajderom ostalim komonentama
    state={
      korisnik: null
    };

    logIn = name =>{
      this.setState({ korisnik: name});

      // izrada localStorage baze podataka u koju se upisuju vrednosti
      localStorage.setItem('pera',JSON.stringify(name));
```

```
    };

    logOut=()=>{
        this.setState({korisnik: null});
        console.log(this.state.korisnik);
        localStorage.removeItem('pera');
    };

    render(){
        return(
            <MyContext.Provider
            value={{
              // objekt u kome se smeštaju vrednosti koje će biti deljene komponentama
               korisnik: this.state.korisnik,
               logIn: this.logIn,
               logOut: this.logOut
            }}
            >
               {this.props.children}
            </MyContext.Provider>
        );
    }
}
// komponenta koja služi za prenos drugih komponetni u App
const Prenosna=()=> <Login  />;

// klasa login
class Login extends React.Component{
    state={};
    render (){
        return(
            <MyContext.Consumer>
            {(value) =>{
               const{korisnik,logIn, logOut} = value;
              return korisnik ? (
                      <React.Fragment>
                         <h3> Prijavljen korisnik: {korisnik}</h3>
                         <Nova />
                      </React.Fragment>
                   ):(
                      <React.Fragment>
                         <input placeholder="Unesi ime"
                               ref ={r => (this.inputRef = r)}/><hr/>
                         <button type="submit"
```

```
                        onClick={()=>{logIn(this.inputRef.value)
                        }}
                    >
                        Log In
                    </button>
                </React.Fragment>
            );
        }}
    </MyContext.Consumer>
    );
  }
}

export default class App extends React.Component{

  render(){
    return(
      <Provider>
      <div>
        <div className="container">
          < Prenosna />
        </div>
      </div>
      </Provider>
    );
  }
};
```

Drugi način za istu namenu prijave na nalog

Ovaj primer koda zasnovan je na kodu faila upotrebe rutera u više nivoa. Prikazuje princip rada prijave na nalog na nekom budućem veb sasjtu. Gde je App klasa pocetna stranica koja prestavlja meni izbora da li ćemo se prijaviti ili ostati na glavnoj stranici. Funkcije koje su prisutne u kodu rade sledeće:

1. Početna ispisuje polazni tekst
2. Prva prikazuje meni, gde klikom na ponuđene mogućnosti prelazite u sledeću fazu
3. Druga prikazuje formu za unos podataka da bi se izvršila prijava na nalog
4. Treca prikazuje korisničku stranicu kada je uspešno korisnik prijavljen na svoj nalog

```
import React, { Component } from 'react'
import { BrowserRouter as Router, Route,NavLink} from 'react-router-dom'
```

```
class App extends Component {

   render() {
      return (
         <Router >
            <div style={{backgroundColor:"pink", margin: '0 auto',
                border:"4px solid grey"}}
                className="container" >
               <NavLink className="btn-primary btn-lg" to='/'
                  style={{marginRight:"5px"}}>
                 Glavna
               </NavLink>
               <NavLink className="btn-primary btn-lg"
               to='/prvi'>Prijava</NavLink>
               <hr style={{border:"2px red solid"}}/>
               <Route exact path='/' component={Pocetna} />
               <Route path='/prvi' component={Prva} />
            </div>
         </Router>
      )
   }
}
```

Izbor da li će se korisnik prijaviti ili vratiti na početnu stranicu. U ovom slučaju prikazano je da potpuno ravnopravno mogu se upotrebiti: delovi react dom rutera NavLink i Link, kao i klasičan pristup upotrebom anker html taga a sa href-om.

```
function Prva ({ match }) {
   const id = "id";
   return (
      <div>
         <h1>Login Forma</h1>
         <p> Niste prijavljeni</p>
         <NavLink to={`${match.url}/${id}`}  style={{marginRight:"5px"}}
          className="btn-info btn-lg">Prijava NavLink-om </NavLink>
         <Link to={`${match.url}/${id}`}  style={{marginRight:"5px"}}
          className="btn-info btn-lg">Prijava Link-om</Link>
         <a href ={`${match.url}/${id}`} className="btn-warning btn-lg">
           Prijava upotrebom "a href"</a>
         <hr style={{border:"2px red solid"}}/>
         <Route path={`${match.path}/:pocetniId`} component={Druga}/>
      </div>
   )
}
```

Uobičajena je praksa da se uvek uzima nešto što je jedinstveno za svakog korisnika kako bi se pristupalo korisničkim podacima. U bazama podataka to je polje koje se zove id, jer je ono spečifično za svakog korsnika za razliku od drugih polja podataka vezanih za korisnike. Te je u objektu match ono i dodato iza kose crte. Mada može za ovo biti upotrebljena bilo koja druga reč. Ako pokrenemo kod u razvojnom delu rada i postavimo pokazivač miša na dugme:

1. Glavna videćemo url ispod http://localhost:3000
2. Prijava http://localhost:3000/prvi
3. ispod natpisa Login forma na oba dugmeta pise http://localhost:3000/prvi/id

Ako zamenimo u function Prva vrednost konstate id sa nekom drugom vrednošću dobićemo to ispisano umesto Id kao u slučaju tri. Što znači da ova konstata može dobijati i vrednosti direktno iz baze podataka. Ovaj deo koda

```
<Route path={`${match.path}/:pocetniId`} component={Druga}/>
```

poziva sledeću komponentu koja se pojavljuje na ekranu, ali da bi se ona pojavila potrebno je da ima u svom sastavu koda vrednost za parametar vrednost ”pocetniId”. Pored toga sa odlaskom na ovaj link dolazi da se tamo renderuje komponeta koja se zove Druga.

```
// prijava korisnika na nalog unosom podataka u formu
function Druga ({ match }) {
   const podatak =(({ id }) => id === match.params.pocetniId);
   const id = "drugiId";
   return (
      <div>
         <form>
            <label htmlFor="ime">Ime</label>
            <input type="text" name="ime" placeholder="Unesi svoje ime"/>
            <hr style={{border:"1px green solid"}}/>
            <label htmlFor="prezime">Ime</label>
            <input type="text" name="prezime"
               placeholder="Unesi svoje prezime"/>
            <hr style={{border:"1px green solid"}}/>
            <a href ={`${match.url}/${id}`} type="button"
            style={{marginRight:"5px"}}
               className="btn-primary btn-lg">LogIn</a>
            <NavLink className="btn-success btn-lg" to="/"> LogOut</NavLink>
         </form>
         <hr style={{border:"2px red solid"}}/>
         <Route path={`${match.path}/:drugiId`} component={Treca} />
      </div>
```

```
    )
}
```

Kao i u predhodnom slučaju klikom na dugme LogIn prelazi se na prikaz podataka korisnika koji je predhodno uneo tražene podatke u formu

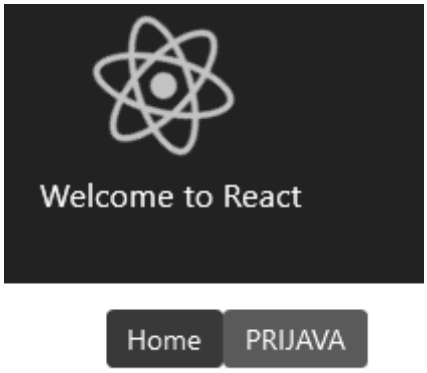
```
// login stranica korisnika sa njegovim podacima
function Treca ({ match }) {
    const podatak = (({ id }) => id === match.params.drugiId);
    return (
        <div>
            <h1>Prijavljen si na stranici</h1>
            <h2>Korisnik{podatak.name}</h2>
            <p>Dodatano Objasnjenje{podatak.objasnjenje}</p>
            <hr style={{border:"2px red solid"}}/>
            <a href={podatak.url} className="btn-info btn-lg"
              style={{paddingBottom:"10px" }} >Dodatne Infromacije</a>
            <hr style={{border:"1px red solid",marginBottom:" 5px"}}/>
            <NavLink className="btn-success btn-lg" to="/"> LogOut</NavLink>
        </div>
    )
}
// informativna da bi se znalo o čemu je rec na dobijenoj stranici
const Pocetna =()=> {
    return (
        <div>
            <h1>Pocetna Stranica </h1>
            <p>
             Prikaz rada dodatnog paketa React rutera u vise nivoa pristupa podacima u
             sistemu prijave na nalog
            </p>
        </div>
    )
};

export default App
```

Ovom kodu potrebno je dodati uslovne petlje i tako preusmeravati i omogućiti samo pojavu neophodnih delova u određenim koracima i dodati formu za registraciju korisnika.

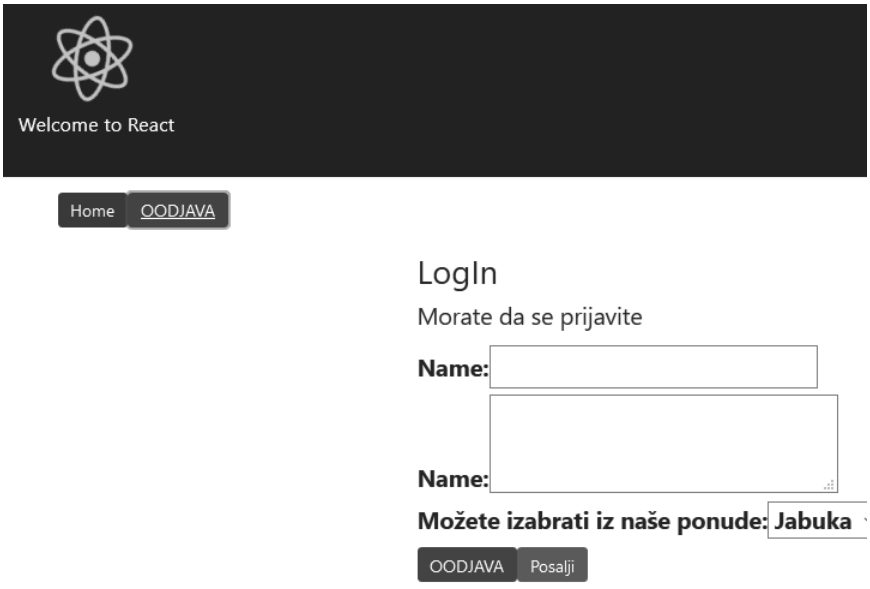
Kod react komponente za login/logout

Ovaj kod sadrži više komponenti koje se klikom pojavljuju na svojim lokacijama upotrebom delova react-router-dom paketa, pri čemu upotrebljavaju podatke iz

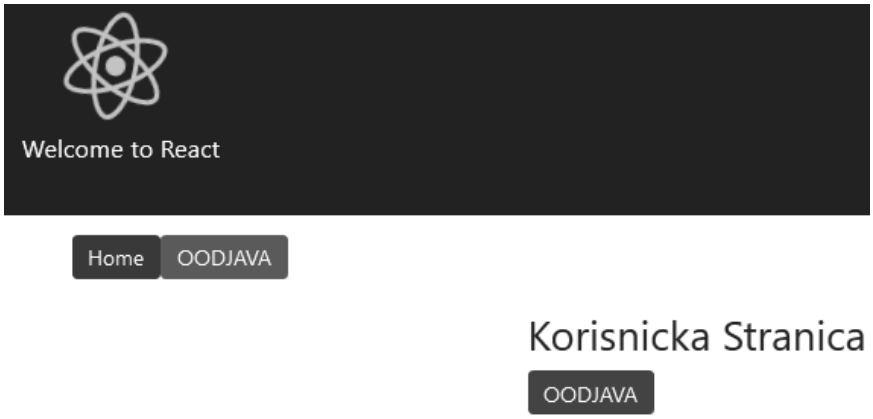
istorije kretanja, pri čemu je omogućeno kretanje po potrebi na home i index stranicu korisnika i povratak na polaznu stranicu.
U primeru pod nazivom ruterx.js, upotrebljen je nešto prepravljen kod u ovoj knjizi iz predhodne lekcije „Zanimljiv događaj dugmetom“, kao i delovi koda Dugme.js, forma.js o probnideo/App.js. U primeru nisu dodati potrebni uslovi jer je cilj videti kako ceo sklop prijave i odjave sa naloga radi. Na polaznom ekranu se dobija ovaj prikaz, gde klikom na dugme Home idete na home stranicu,



a na dugme prijava pojavljuje se forma



gde klikom na odjavu vraćamo se na polaznu stranicu dok klikom na dugme pošalji odlazimo na link i stranicu user



gde možemo postaviti korisničke podatke i slično i odjaviti se od korisničkog naloga.

```
import React from "react";
import { BrowserRouter, Route, NavLink,Switch, Redirect} from
        "react-router-dom";
```

Kod kojim objedinjujemo kod ovog faila

Klasa Appp je jedina izlazna i nju imoprtujemo onamo gde nam je to potrebno. Da bi smo izvršili prijavu na nalog potrebno nam je stanje neke promenjive koja će biti bulean tipa i uz pomoć nje ćemo održavati stanje prijava/odjava. Tačnije ako je kliknuto dugme za prijavu ono će nestati sa ekrana, a pojaviti se gde je to potrebno dugme Odjava. Ovo se postiže upotrebom postavljanja osnovnog stanja promenjive prijavaL koje se menja na događaj klik koji je povezan sa funkcijom Prijava, koja predhodno stanje menja za stanje vrednosti koje je od nje različito. Dok u kodu komponente

```
<NavLink className={!this.state.prijavaL?'btn-danger btn-lg':' '}
    to="/login"
    onClick={this.Prijava} >{!this.state.prijavaL?'PRIJAVA':''}
</NavLink>
```

ta je vrednost promenjive prijavL povezano u dve if else petlje. Prvi slučaj vezan je za klase

```
className={!this.state.prijavaL?'btn-danger btn-lg':''}
```

kojim je omogućeno da se pojavi prva klasa iza znaka upitnika i time pokaže dugme sa ispisom ili da se ne primeni ni jedna klasa te prikaz dugmeta nestaje. U drugom slučaju povezana je sa ispisom na dugmetu

```
{!this.state.prijavaL?'PRIJAVA':' '}
```

Što je preuzeto iz koda zanimljivo dugme. dok to="/login" nas automacki postavlja na link u ovom slučaju http://localhost:3000/login pošto smo trenutno na lokalhostu, gde se vrši prikaz komponete Login

```
<Route exact={true} path="/login" component={LogIn} />
```

Kao što to prikazuje kod iznad u putanji komponente react-router-dom-a Route. Ono što je karakteristično za BrowserRouter je to da on i komponenta react-router-dom-a Router mogu da sadržavaju samo jedan element kao svoj podelement. Pa se zato pribegava u slučaju potrebe za više elemenata-komponenti react-router-dom-a upotrebi prvo div html taga u kome zatvaramo ostale potrebne komponente. Dok za uključivanje više komponenti Route upotrebljava se komponenta react-router-dom-a Switch, koja ih sadrži kao svoje pod komponente. Za kretanje po putanjama koje navedete između mogu se upotrebljavati kao što će te i videti u daljem kodu komponenta react-router-dom-a Redirect i vrednost propsa istorije brovsera

```
this.props.history.push('/Na koju putanju se upućuje kretanje');
```

U ovom slučaju primenio sam čitanje iz localStorage podataka, koje se upisuje prilikom prijave u formi. Ovim kodom se očitava uneta vrednost u localStorage

```
let pera = localStorage.getItem('pera');
```

kako bi se dodao link u putanji gde se i vrši prikaz tog profila za koji je uneto ime u formi posle prijave

```
<Route path={`/users/:${pera}`} component={Profile}/>
```

Napomena: U ovom slučaju se upotrebljavaju inverzni apostrofi, koji okružuju java skript kod

```
path={`/users/:${pera}`}
```

Kod vindovsa taster iznad tastera tab kod engleske tatature.

Kod:

```
export default class Appp extends React.Component{
  constructor(props){
    super(props);
    this.state={
      prijavaL: this.props.prijavaL
    };
    this.Prijava=this.Prijava.bind(this);
```

```
    }

    Prijava(){
        this.setState(prevState=>({
            prijavaL: !prevState.prijavaL,
        }));
    }

    render(){
        console.log(this.props.prijavaL);
        let pera = localStorage.getItem('pera');
        return (
            <BrowserRouter>
                <div>
                    <header style={stil}>
                      <NavLink className={!this.state.prijavaL?'btn-danger btn-lg':''}
                               to="/login"
                              onClick={this.Prijava} >{!this.state.prijavaL?'PRIJAVA':''}
                      </NavLink>
                    </header>
                    <Switch>
                        <Route path="/home" component={Home} />
                        <Route exact={true} path="/login" component={LogIn} />
                        <Route path={`/users/:${pera}`} component={Profile}/>
                        <Route path="/toto" component={Toto} />
                    </Switch>
                </div>
            </BrowserRouter>
        );
    }
}
```

Način pisanja koda direktnog css stila

Ovo je još jedan od načina kada vam zatreba direktni način unosa css koda stilova u komponentu. Po svojoj prilici to je klasičan JSON format prisanja, koji se poziva gde je to potrebno zatvoren velikim zagradama, kao što možete i uostalom pozivati java skript kod u return delu react aplikacija.

```
<header style={stil}>
```

pošto je to isto java skript.

```
const stil={
    marginTop: '10px',
```

```
    marginLeft:'50px',
    marginBottom:'20px'
};
```

Klasa komponente profila

Prikazivanjem ove komponente prikazuje se stranica korisnika koji se upešno prijavio svojim podacima. Za sada klikom na dugme prijava. Gde se očitana vrednost za sada iz localStorage pored upotrebe za link u pretraživaču upotrebljava i za ime korisnika koji se prijavio. Ova komponenta u svom kodu sadrži i dugme za odjavu, kojim za sada se samo vrši povratak na osnovni link aplikacije klikom na dugme odjava.

```
class Profile extends React.Component {
    constructor() {
        super();
        this.state = {

        };
    }
    render() {
        let pera = localStorage.getItem('pera');
        return (
            <div className="container">
                <h2>Ovo je pocetna index stranica korisnika:  --{pera}</h2>;
                <p className="text-justify text-success">
                    The Designer will translate the vision of the creative director
                </p>
                <NavLink className="btn-danger btn-lg" to="/" onClick={LoggOut} >
                  ODJAVA
              </NavLink>
            </div>
        );
    }
}
```

Home stranica

Je samo jedan primer da bi omogućio raznolikost u primeru, mada može biti upotrebljen kao model kod za neke buduće komponente

```
class Home extends React.Component {
    constructor() {
        super();
        this.state = {
        };
```

```
    }
    render() {
        return <h2>Ovo je pocetna index stranica </h2>;
    }
}

// korisnička stranica
class Users  extends React.Component{
    constructor(props) {
        super(props);
        this.state = {

        };
    }

    render() {
        let pera = localStorage.getItem('pera');
        return (
            <div className="container">
                <h1>Korisnik</h1>
                <p>Prijavljen si kao korisnik {pera}</p>
                <NavLink className="btn-danger btn-lg" to="/" onClick={LoggOut} >
                  ODJAVA
                </NavLink>
            </div>
        );
    }
}
```

Klasa za prijavu

Namena ove klase je da prikaže formu za prijavu na nalog i da posle unetih podataka se izvrši provera unetih podataka, nakon čega sledi preusmerenje na korisničku stranicu koja odogovara unetim podacima ili ako podaci nisu ispravni da vrati korisnika na ponovni unos ili registraciju novih korisnika, sa mogućnošću da se može vratiti i na početnu stranicu u ovom primeru klikom na dugme odjava. No, u ovom početnom slučaju izrade forme i komponente za prijavu postavljeno je da korisnik prvo unese podatak u formu i klikne na dugmad Submit (pošalji), kako bi se unet podatak uneo u localStorage i kasnije klikom na jedno od poreostala dva dugmeta preusmerio na svoju stranicu. Jer, sam postavio malo različito napisan kod koji radi isti posao.

```
                <NavLink className="btn-danger btn-lg" type="submit"
                  to={`/users/${pera}`}>
                    Posalji
```

```
</NavLink>
<NavLink className="btn-primary btn-lg" type="submit"
    to={"/users/"+ pera}>
  Posalji
</NavLink>
```

uzimajući da reč pera može biti bilo koja druga primenjena u aplikaciji, uzimajući u obzir da gde je ona primenjnena, da se primeni svuda gde se ona primenjuje ta druga. Ono što je bitno u ovom slučaju da se promeni stanje za promenjivu prijavaL da je istina, kako bi dugme za prijavu ostalo skriveno,tj, ne prikazano. Obzirom da je početno stanje za unošenje vrednost i potrebno da bude prazno inicijalna vrednost mu je prazana vrednost.

```
this.state = {
   value:'',
   prijavaL: true
};
```

U kodu imamo i tri funkcije handleSubmit za slanje podataka iz forme, handleChange za omogućavanje vršenja promene unosa u unosno polje forme i onSubmit koja poziva redirekciju na drugu lokaciju, u našem slučaju na komponentu Kontakt.
handleSubmit povezana da deluje kao događaj koji će se izvršavati nakon klika na dugme koji je poziva. Ona poziva metod preventDeafult, nakon njega može se to u cilju vežbe postaviti alert. Uneta vrednost se prosleđuje konstanti koja se beleži u localStorage. Nakon čega se vraća vrednost komponete Profile i prikazuje stranicu korisnika

```
handleSubmit(e){
  e.preventDefault();
  //alert(this.state.value);
  let peraa= this.state.value; // postavljanje vrednosti u konstantu
  localStorage.setItem('pera',peraa ); // upis u localStorage
  console.log(this.state.value,'vraćena vrednost iz propsa ' ); // provera
   return < Profile vrednos={this.state.value} />;
}

class LogIn extends React.Component{
  constructor() {
    super();
    this.state = {
       value:'',
       prijavaL: true
    };
```

```
      this.handleChange = this.handleChange.bind(this);
    }

     handleSubmit(e){
       e.preventDefault();
       alert(this.state.value);
       let peraa= this.state.value; // postavljanje vrednosti u konstantu
       localStorage.setItem('pera',peraa ); // upis u localStorage
       console.log(this.state.value,'vraćena vrednost iz propsa ' ); // provera
        return < Profile vrednos={this.state.value} />;
     }

// omogućava da dođe do promene u unosnom polju frome
     handleChange(e){
        this.setState({value: e.target.value});
     }

// bezuslovni prelazak na određenu komponetu
   onSubmit = () => {
      //this.props.history.push('/contact')
      return <Redirect to="/contact" />;
   };
   render() {
            const pera= this.state.value;
      return (
         <div className="container">
            <h1>LogIn</h1>
            <p> Morate da se prijavite</p>
            <form onSubmit={this.handleSubmit.bind(this)}>
               <input type="text" value={this.state.value}
                 onChange={this.handleChange} />
               <button type="submit" className="btn-dark" > posalji</button>
               <input type="submit" value="Submit" onClick={this.onSubmit}/>
               <NavLink className="btn-danger btn-lg" to="/"
                  onClick={LoggOut} >
                   ODJAVA
                </NavLink>
               <NavLink className="btn-danger btn-lg" to="/contact"
               onClick={LoggOut} >
                 Kontakt</NavLink>
               <NavLink className="btn-danger btn-lg" type="submit"
                    to={`/users/${pera}`}>
                    Posalji
                </NavLink>
```

```
                <NavLink className="btn-primary btn-lg" type="submit"
                    to={"/users/"+ pera}>
                   Posalji
                </NavLink>
             </form>
          </div>
       );
    }
}
```

Pomocna funkcija, potrebna je da vrati stanje promenjive prijavaL u pojedinim komponetama koje su pozvane kako bi se zadržalo potrebno stanje vezano za pojavu dugmeta Prijava samo kada je i gde to potrebno da se ono pojavi.

```
function LoggOut(){
   this.setState({
      prijavaL: this.state.prijavaL
   });

 }
```

Isti primer koda na drugi način

Pošto sam u početku pisao o načinu razmišljanja u reactu pomenuo sam usitnjavanje glomaznog koda i dobijanje jasnijeg i kraćeg veličinski koda u više komponenti, koje po potrebi u aplikaciji pozivaju jedna drugu. Tako da ću gornji kompletan kod podeliti u više failova i dva foldera, da bih prestavio sliku svega. Za to je potrebno da u src direktorijumu napravim jedan folder kog ću nazvati prijava, a u kome napraviti još jedan koji ću nazvati komponete.

Sadržaj foldera prijava : Glavna.js i funkcije.js
Sadržaj foldera komponente: Profil.js, Login.js, Users.js, Toto.js i Home.js.

Ove buduće failove napraviću od predhodnog koda i oni će imati ovaj sadržaj:
fail Glavna.js

```
import React from "react";
import { BrowserRouter, Route, NavLink,Switch} from "react-router-dom";
import Profile from './komponente/Profil';
import LogIn from './komponente/Login';
import Home from './komponente/Home';
import Toto from './komponente/Toto';

export default class Appp extends React.Component{
   constructor(props){
```

```
        super(props);
        this.state={
            prijavaL: this.props.prijavaL
        };
        this.Prijava=this.Prijava.bind(this);
    }

    Prijava(){
        this.setState(prevState=>({
            prijavaL: !prevState.prijavaL,
        }));
    }

    render(){
        console.log(this.props.prijavaL);
        let pera = localStorage.getItem('pera');
        return (
            <BrowserRouter>
                <div>
                    <header style={stil}>
                        <NavLink className={!this.state.prijavaL?'btn-danger btn-lg':''}
                                to="/login"
                                onClick={this.Prijava} >{!this.state.prijavaL?'PRIJAVA':''}
                        </NavLink>
                    </header>
                    <Switch>
                        <Route path="/home" component={Home} />
                        <Route exact={true} path="/login" component={LogIn} />
                        <Route path={`/users/:${pera}`} component={Profile}/>
                        <Route path="/toto" component={Toto} />
                    </Switch>
                </div>
            </BrowserRouter>
        );
    }
}
const stil={
    marginTop: '10px',
    marginLeft:'50px',
    marginBottom:'20px'
};
```

fail funkcije.js

```
export function LoggOut(){
   this.setState({
      prijavaL: this.state.prijavaL
   });
}
```

fail Home.js

```
import React from "react";
//import { BrowserRouter, Route, NavLink,Switch, Redirect} from "react-router-
dom";

export default class Home extends React.Component {
   constructor() {
      super();
      this.state = {

      };
   }
   render() {
      return <h2>Ovo je pocetna index stranica </h2>;
   }
}
```

fail Login.js

```
import React from "react";
import {  NavLink, Redirect} from "react-router-dom";
import{LoggOut} from '../funkcije';
export default class LogIn extends React.Component{
   constructor() {
      super();
      this.state = {
         value:'',
         prijavaL: true
      };
      this.handleChange = this.handleChange.bind(this);
   }

   handleSubmit(e){
      e.preventDefault();
      alert(this.state.value);
      let peraa= this.state.value;
      localStorage.setItem('pera',peraa);
```

```
    // Za brisanje sadržaja unosnih polja forme
      this.setState({
            name:' ',
      });
      // za ponovni reload iste stranice
       window.location.reload();

   // console.log(this.state.value,'vracena vrednost iz propsa ' );
   // return <Profile vrednos={this.state.value} />;
   return <Redirect to={`/users/${peraa}`} vrednos={this.state.value} />;

}
handleChange(e){
   this.setState({value: e.target.value});
}
onSubmit = () => {
   //this.props.history.push('/contact')
   return <Redirect to="/contact" />;
};
render() {
   const pera= this.state.value;
   return (
      <div className="container">
         <h1>LogIn</h1>
         <p> Morate da se prijavite</p>
         <form onSubmit={this.handleSubmit.bind(this)}>
           <input type="text" name=”name” value={this.state.value}
                onChange={this.handleChange} />
            <button type="submit" className="btn-dark" > posalji</button>
            <input type="submit" value="Submit" onClick={this.onSubmit}/>
          <NavLink className="btn-danger btn-lg" to="/" onClick={LoggOut} >
            ODJAVA
            </NavLink>
           <NavLink className="btn-danger btn-lg" to="/contact"
           onClick={LoggOut} >
            KA drugom
            </NavLink>
           <NavLink className="btn-danger btn-lg" type="submit"
            to={`/users/${pera}`}>
             Posalji
           </NavLink>
           <NavLink className="btn-primary btn-lg" type="submit"
              to={"/users/"+ pera}>
               Posalji
```

```
                </NavLink>
            </form>
        </div>
    );
  }
}
```

Napomena: da bi smo obrisali sadržaj unosnih polja u formama najjednostavniji način je ponovo vrati inicijalne vrednosti stanja. To se postiže nakon submita forme dodavnjem koda

```
this.setState({
   email:' ',
   password:' ',
   name:' ',
});
```

Kojim postavljamo na prazne vrednosti unosna polja koja smo koristili.Uzimajući da je sam react u osnovi java skript, to znači da možemo upotrebiti i čistog i malo prilagođenog java skript koda svuda gde nam je to potrebno. Kako bi ova forma nakon unosa i klika na dugme Sing Up mogla da ne samo unose u polja da ubeleži u localstorage, već i da se to pojavi na određenim mestima na stranici možemo da upotrebimo ovaj kod :

```
        // za ponovni reload iste stranice
         window.location.reload();
```

Kojim praktično vršimo ponovno učitavnje iste stranice sa upotrebom refresh-a.

fail Toto.js

```
import React from "react";
//import { BrowserRouter, Route, NavLink,Switch, Redirect} from
"react-router-dom";

export default class Toto extends React.Component {
   componentDidMount() {

   }
   render() {

      return(
         <div className="container">
            <table className="table-bordered">
```

```
                <tbody>
                <tr>
                   <td> korisnik</td>
                   <td>objasnjenja</td>
                   <td>Postavke</td>
                </tr>
                <tr>
                   <td> korisnik</td>
                   <td>objasnjenja</td>
                   <td>Postavke</td>
                </tr>
                </tbody>
             </table>
          </div>
       );
    }
}

fail Users.js
import React from "react";
import { BrowserRouter, Route, NavLink,Switch, Redirect} from
   "react-router-dom";
export default class Users  extends React.Component{
    constructor(props) {
       super(props);
       this.state = {
          prijavaL: false,
       };
    }

    render() {
       let pera = localStorage.getItem('pera');
       // console.log(this.props.vrednos);
       return (
          <div className="container">
             <h1>Korisnik</h1>
             <p>Prijavljen si kao korisnik {pera}</p>
             <NavLink className="btn-danger btn-lg" to="/"
              onClick={LoggOut} >
              ODJAVA
            </NavLink>
          </div>
       );
```

```
    }
}
```

Fail Profil.js

```
import React from "react";
import {  NavLink} from "react-router-dom";
import{LoggOut} from '../funkcije';
import '../../App.css';// prijava css-a iz foldera src aplikacije

export default class Profile extends React.Component {
    constructor(){
      super();
       this.state = {

       };
    }

    componentWillMount(){
       localStorage.getItem('testObject')&&this.setState({
          objekat: JSON.parse(localStorage.getItem('testObject'))
       });
       localStorage.getItem('tObject')&&this.setState({
          objekati: JSON.parse(localStorage.getItem('tObject'))
       });
     }
    componentDidMount(){

    }

    render(){

       let pera = localStorage.getItem('pera');
       let testObject =
         {name:"test", time:"Date 2017-02-03T08:38:04.449Z", ime:"Neko"};
       localStorage.setItem('testObject', JSON.stringify(testObject));

       let tObject ={name:"Moje", prezime:"nekoprezime"};
       localStorage.setItem('tObject', JSON.stringify(tObject));
          return (
           <div className="container">
              <header className="App-header">
             <NavLink className="btn-danger btn-lg" to="/" onClick={LoggOut} >
                 ODJAVA
```

```
            </NavLink>
          </header>
          <h2>Ovo je pocetna index stranica korisnika:  --{pera}</h2>;
          <p className="text-justify text-success">
            The Designer will translate the vision of
            and the lead designer into in-game story {}događaj.
            Your role is to create a successful
          </p>
            {
              Object.keys(this.state.objekat) &&
              Object.keys(this.state.objekat).length >0
            }
          <h1>{this.state.objekat.name }</h1>
          <h3>{this.state.objekat.time }</h3>
          <h4>{this.state.objekat.ime}</h4>
          <hr />
          {
            Object.keys(this.state.objekati) &&
            Object.keys(this.state.objekati).length >0
          }
          <h3>{this.state.objekati.name }</h3>
          <h4>{this.state.objekati.prezime}</h4>
        </div>
    );
  }
}
```

Na ovom principu možete izraditi i druge failove potrebnih komponenti. U poslednjem failu Profil.js postoji i kod o kome će biti reči u dole niže prestavljenoj lekciji o unosu više vrednosti u localStorage u obliku java skript objekta.

Unos više vrednosti

Predhodni primer biće dopunjen sam u delovima za prijavu i delu koji prikazuje primeljene podatke, dok sve ostalo ostaće isto. Obzirom da radimo sa objektima potrebno je da ih takve sačuvamo u prvo vreme u localStorege-u, jer princip je istovetan i sa radom i sa bazom podataka na serveru sa razlikom dodatog koda. Prvo se pogleda da li nešto već ima upisano ili nema, ako nema upisuje se trenutno što ima, u suprotnom novo se dodaje starom i vrednost id polja koja raste za jedan korak u plusu. Kod brisanja podataka isto se upotrebljava ovo polje kako bi se tačno znalo koji podatak se briše. Iza čega se podaci prikazju prema nekom utvrđenom redu i rasporedu na stranici.
JSON je skraćenica za java skript objekt format, koji se sastoji od parova vrednosti koje predstavljaju ključeve i njhove vrednosti, ovakvog izgleda kada je napisan direktno u kodu faila komponente:

```
    let testObject ={name:"test", time:"Date 2017-02-03T08:38:04.449Z",
ime:"Neko"};
```

ali mogu to biti i failovi koji su van koda komponenti, što je prikazano u predhodnim delovima knjige. Što znači u ovakvim slučajevima nemožemo da primenimo map da bi smo dobili delove koje bi smo hteli da prikažemo kao delove komponenti. Razlog je jednostavan map radi sa nizovima, a ne sa objektima. I pojaviće se greška da map nije funkcija. U tim slučajevima potrebno je napraviti posebnu komponentu određenog životnog ciklusa koja će kasnije biti priključena (mountovana): componentDidMount. U kojoj ćemo promenjivama dodeliti pročitane vrednosti iz localStorage, gde je primenjen metod JSON.parse. Metoda JSON.parse () analizira JSON niz, praveći JavaScript vrednost ili objekt opisan nizom. Dajući mogućnosti revizije za obavljanje transformacije na rezultantnom objektu pre nego što se vrati.

```
componentWillMount(){
   this.setState({
      objekat: JSON.parse(localStorage.getItem('testObject')),
      objekati: JSON.parse(localStorage.getItem('tObject'))
   });
  }
```

Čime je promenjivama objekat i objekati dodeljena vrednost stanja. Da posetim, samo ono što ima vrednosti stanja može da menja svoje vrednosti, dok props su vrednosti samo namenjene da se mogu pročitati i prikazati, to jest nemogu se menjati ako se ne promene vrednosti stanja od kojih props dobija vrednost. Tako da ih možemo pozvati bilo gde u komponenti radi daljih radnji sa njima. U ovom primeru sam upotrebio dva java skrip objekta, čije sam vrednosti uneo u localStorage pod različitim nazivima.

```
let testObject ={name:"test", time:"Date 2017-02-03T08:38:04.449Z",
                 ime:"Neko"};
localStorage.setItem('testObject', JSON.stringify(testObject));

let Object ={name:"Moje", prezime:"nekoprezime"};
localStorage.setItem(' testObject ', JSON.stringify(Object));
```

localStorage.getItem čita određene podatke iz localStorage, a localStorage.setItem vrši upisivanje podataka u localStorage. JSON.stringify je suprotna od JSON.parse i ona prevodi java skript objekt u JSON objekt. Obzirom na složenost problema prvo je potrebno izdvojiti ključeve objekta, pa kasnije pročitati njihove vrednosti. U ovom slučaju upotrebljava se službena reč java skripta Object.key. Kojoj dajemo da ima vrednost stanja koja je dobijena iz componetDidMount metoda.

```
{
   Object.keys(this.state.objekat)
}
```

ili ovako gde proveravamo da li je dužina veća od prazne vrednosti ako nam je to potrebno.

```
{
Object.keys(this.state.objekati) &&
Object.keys(this.state.objekati).length >0
}
```

Napomena: Kada kao u ovom primeru očitavate vrednosti više objekata potrebno je svaki novi Object.key odvojiti posebno.

Prikazivanje ovih vednosti u komponenti vrši se tako što se dobijenom ključu u obliku vrednosti dodaje nastavak kojim on praktično očitava vrednost drugog para objekta vrednost (value)

```
<h1>{this.state.objekat.name }</h1>
<h3>{this.state.objekat.time }</h3>
<h4>{this.state.objekat.ime}</h4>
<hr />
<h3>{this.state.objekati.name }</h3>
<h4>{this.state.objekati.prezime}</h4>
```

Da se ne bi zbunili this.state.objekat je ključ (key) ,a name je vrednost (value). Unos više vrednosti u java skript objekt najčešće se objavlja unosnim formama: za logovanje, slanje emaila, unos u bazu podataka, s time što se na serveru ti podaci koji čine java skript objekt prevode u druge potrebne oblike kako bi se na serveru uradio potreban posao. Umesto imena i prezimena može biti link prema nekoj slici ili failu čiji sadržaj se želi prikazati.

Objekti

Kod rada sa objektima oni prvo moraju da postoje, kao što postoje i u našem realnom svetu opisani, kao u kodu dole niže:

```
// opis budućeg objekta
let Auto= (marka, model, godinaProizvodnje)=> {
   this.marka = marka;
   this.model = model;
   this.godProizvodnje = godinaProizvodnje;
};
```

```
// izrada instance postojećeg objekta
let MojAuto = new Auto("Citroen", "C5", 2012);

// prikaz sadžaja novog objekta u konzoli
console.log("Auto je: " + this.marka +" " + this.model + " proizveden " +
this.godProizvodnje + " godine");
```

Napomena: da bi se dobile stalno promenjive vrednosti koje unosimo putem formi kod koji prikazje konzola se može upoterebiti na seledeći način:

1. umesto let MojAuto = new Auto("Citroen", "C5", 2012); uneti umesto this.marka i ostalo uneti this.state.marka i sl

Na taj način smo obezbedili da funkcija Auto na novoj instanci stalno može dobijati nove vrednosti.

Bez postojanja definisanog objekta javljaće vam se greška jer ako u napisanom kodu postoji greška, ako nemože da se reši onda u brovseru se prikazje sta je problem ili u terminaliskom prikazu. Što je još jedna prednost rada u react okruženju. Čak šta više u velikom broju slučajeva prikazaće u kom failu je problem, koji je problem i u kom je redu koda nastao problem. Tako da su njegovi kreatori stvarno postavili temelje učenju dobrog programiranja. Ali, uz upotrebu boljih editora koda vam se pruža još više olakšanja, jer i oni u toku pisanja koda vam javljaju ako nastane problem. Objekte možete napraviti i na ovaj način:

Prvi korak: napravite fail sa funkcijama u kojima napravite opis objekta sa njegovim opisom u obliku osobina sa kojima bi ste da radite u aplikaciji:

```
export function Opis(name, maker) {
   this.name = name;
   this.maker = maker;
}
```

Mogu biti upotrebljena brojčano koliko vam je potrebno osobina objekta u aplikaciji, s time što putem njih vršite njihova pozivanja, odnosno isti broj se njih mora naći i u malim zagradama funkcije.
Što react komponeta poziva sa

```
import {Opis} from ‘./putanja do faila/funkcija’;
```

Za ime funkcije možete uzeti bilo koju reč koja vam odgovara. U malim zagradama je ono što funkcija vraća u obliku vrednosti stanja ili ono što joj pošaljemo kao potrebno za njen rad. Dok u velikim zagradama jo ono što nam je potrebno uradi sa podacima i kome dajemo u ovom primeru vrednost stanja.

Drugi korak: u komponenti koja će pozvati možemo napraviti između rendera i return dela u ES6 klasama promenjivu kojom ćemo izraditi novu instancu postojećeg objekta iz prvog koraka

```
let Auto1 = new Opis('Fiesta','Auto');
let Auto 2 = new Opis ('Santa Fe', 'Hyundai');
```

Umesto sadržaja u malim zagradama mogu se upotrebiti nove vrednosti stanja koje ćemo dodeljivati objektu, nastale upotrebom this.state.novavrednost ili na drugi način

Treći korak: prikaz stanja u novim objektima i njihovim delovima u delu return

```
return(
   <div>
      <h1> Automobil 1: { Auto1.name}|| { Auto2.maker}</h1>
   </div>
```

ili upis u localstorage

```
localStorage.setItem('kola',JSON.stringify(Auto1));
```

Kod iznad se postavlja obično gde i kod u drugom koraku, ali po potrebi i na drugim mestima u kodu komponenti u zavisnosti od potrebe.

Login/Logout react-php

Ova komponenta treba da izvrši sledeće zadatke:

1. omogući unos podataka iz forme
2. pošalje ih na server
3. sačeka odgovor sa servera
4. na kraju da izvrši u zavisnosti od odgovora sa servera preusmeravanje

Zato su potrebna početna prazna stanja unosnih polja, kojima su postavljene vrednosti stanja u konstruktoru, gde su i prijavljene tri funkcije:

1. za omogućavanje promene vrednosti stanja u unosnim poljima
   this.handleChange = this.handleChange.bind(this);
2. za slanje unetih vrednosti stanja na server
   this.handleSubmit = this.handleSubmit.bind(this);
3. za promenu vrednosti stanja promenjive login
   this.Prijava = this.Prijava.bind(this);

Namena prve funkcije handleChange:

Upotrebnom koda

this.setState({[event.target.name]: event.target.value, login: this.state.login});

postavljamo unete vrednosti uzimajući događajem vrednosti koje pridružujemo imenima unosnih polja. A u promenjivu login postavljamo osnovno stanje, to jest praznu vrednost. Čime je onemogućeno slanje podataka sve do klika na dugme Submit, čime se poziva funkcija handleSubmit.

Funkcija handleSubmit ima dosta složen zadatak. Ona treba da prvo prikupi podatke, to jest unete vrednosti stanja unosnih polja i postavi ih u jednu konstantu

```
let podaci = this.state;
```

Formira promenjivu za url gde treba da pošalje na server, ciljajući fail na serveru index.php koji će izvršiti obradu podtaka i vratiti rezultat

```
let dataURL = "http://localhost:80/fetchReact/prijava/index.php";
```

Izradi novi hendler upotrebom njegove instance let h = new Headers();

Formira zahtev

```
let req = new Request( dataURL,{
   headers: h,
   method: 'POST',
   body: JSON.stringify(podaci)
});
```

gde će podatke metodom POST poslati putem dela fetch metoda kao JOSN objekt, što će fetch metod prihvatiti i poslati, očekujući sa servera odgovor.

```
fetch(req)
   .then((response)=>{
      if(response.ok){
         return response.json();
      }else{
         throw new Error("losa konekcija");
      }
   })
   .then((response) =>{
```

/* očitavanje rezultata rada sa servera. Uzimajući da su poruke sa servera postavljene u nizu error koji ima u sebi drugi niz message, da bi se poruka pročitala potrebna je ovakva konstrukcija koda, kao u kodu ispod. Jer je Response u ovom

slučaju objekt koji sadrži ostale delove. Ovakav pristup podacima se upotrebljava i kod pristupa metodom map u react aplikacijama, što ste videli ispred. Ili u php kodu kada se upotrebljavaju usađene forech petlje. */

```
            console.log(response.error.message,"ispod" );
            alert(response.error.message);
        })

// očitavanje mogućih greški
        .catch( (error) =>{
            console.log('Error', error.response.error.message);
        });

// brisanje unetih vrednosti stanja u unosnim poljima forme
        this.setState({
            email:'',
            sifra:''
        });
```

React fail phpslanje.js

```
import React from 'react';
import 'whatwg-fetch';
class DemoComponent extends React.Component {
   constructor() {
      super();
      this.state = {
         email:'',
         $pass:'',
         login:false,
         first_name:'' ,
         last_name:'' ,
         podaci:[] ,
      };
      this.handleChange = this.handleChange.bind(this);
      this.handleSubmit = this.handleSubmit.bind(this);
      this.Prijava = this.Prijava.bind(this);
   }

   handleChange(event) {
      this.setState({[event.target.name]: event.target.value});
   }
   Prijava(){
      this.setState({ login: true, requestName: "login"})
```

```
}
handleSubmit(event) {
    event.preventDefault();
    console.log( this.state.login, "login");
    let dataURL = "http://localhost:80/fetchReact/prijava/idex1.php";
    let h = new Headers();
    let podaci = this.state;
    let req = new Request( dataURL,{
        headers: h,
        method: 'POST',
        body: JSON.stringify(podaci)
    });
    fetch(req)
        .then((response)=>{
            if(response.ok){
                return response.json();
            }else{
                throw new Error("losa konekcija");
            }
        })
        .then((response) =>{
            this.setState({podaci: response});
            console.log(response,"ispod" );
            alert(response.error.message);
        })
        .catch( (error) =>{
            console.log('Error', error.response.error.message);
        });
    this.setState({
        $pass:'',
        email:'',
    });
}
render() {

    const povratno=[{
        'first_name':this.state.podaci.first_name,
        'last_name': this.state.podaci.last_name,
        'email':  this.state.podaci.email,
        'password':  this.state.podaci.password,
         'login':'true'
    }];

    localStorage.setItem('baza',JSON.stringify(povratno));
```

```
        let promena= povratno.map((podatak,i)=>{
          return(
            <tr key={i}>
              <td>Ime: {podatak.first_name} <br/>
                Prezime: {podatak.last_name}<br/>
                Email: {podatak.email}<br/>
                Sifra: {podatak.password}<br/>
              </td>
            </tr>
          );
        });
    //console.log(promena,'vraceni podaci');
    return (
      <div>
        <form onSubmit={this.handleSubmit.bind(this)}>
          <label htmlFor="email">email</label>
          <input name="email" type="text" onChange={this.handleChange}
            value={this.state.email} />
          <br/>
          <label  htmlFor="$pass">$pass </label>
          <input name="$pass" type="text"  value={this.state.$pass}
            onChange={this.handleChange}/>
          <br/>
          <button type="submit" name="login" onClick={this.Prijava} >
            Login</button>
        </form>
        <hr/>
        <div className="container"><table ><tbody>
          { promena}</tbody></table> <hr/>
        </div>
      </div>
    );
  }
}

export default DemoComponent;
```

Ono što možete dodati prema predhodnim delovima knjige je validacija unosnih polja, delove bootstrap css-a da bi korisnik mogao da zna odgovore na svoju akciju

Povezivanje baze podataka

Obzirom da je potrebno da se uneti podaci provere i uporede sa podacima koji su već upisani u bazu podataka u predhodni kod dodaću samo dopune postojećem

kodu. Samo povezivanje koda koji poziva bazu podataka videli ste u index.php failu:
include_once 'konfig.php';

Dok kod ovog faila konfig.php , može imati i ovakav kod ako se upotrebljava poziv iz php klase

```
define('DB_HOST', 'localhost');
define('DB_UNAME', 'root');
define('DB_PASS', '');
define('DB_NAME', 'neka_baza');
define('DB_TBL_USERS', 'tbl_users');
```

gde se upotrebljava definisanje potrebnih podataka upotrebom php konstanti sa službenom rečiju define, ali pošto su ovo osnove svega upotrebiću malo jendostavniji kod

```
$host ='localhost';
$admin ='root';
$password= ' ';
$databasename= 'neka_baza';

/* umesto dodeljenih ovako vrednosti promenjivama postavite svoje */
$mysqli = new mysqli($host, $admin,$password, $databasename);

// provera konekcije

if (mysqli_connect_errno()){
   printf(“problemi sa povezivanjem:%s\n”, mysqli_connect_error());
exit();
}
```

Možete ovom kodu dodati i proveru da li postoji na serveru baza kao uslov. Tim kodom se u slučaju ne ispunjavanja uslova postojanja određene baze na serveru automacki izrađuje potrebna baza priloženom php skriptom.
Kod skripta za izradu baze podataka u ovom slučaj upod nazivom test_db

```
$dbhost = 'localhost';
$dbuser = 'root';
$dbpass = '';
$conn = mysqli_connect($dbhost, $dbuser, $dbpass);

if(! $conn ) {
die('Nemogu se povezati na host zbog : ' . mysqli_error($conn));
```

```
}

echo 'Povezan sam na host';

$sql = 'CREATE DATABASE test_db';
$retval = mysqli_query($conn,$sql );

if(! $retval ) {
   die('Nemogu napraviti bazu podataka zbog : ' . mysqli_error());
}

echo "Data baza test_db je napravljena\n";
mysqli_close($conn);
```

Izrada tabele iz php skripte

```
<?php
$dbhost = 'localhost';
$dbuser = 'root';
$dbpass = '';
$conn = mysqli_connect($dbhost, $dbuser, $dbpass);

if(! $conn ) {
   die('Nemogu se povezati na host zbog : ' . mysqli_error($conn));
}

echo 'Povezan sam na host';

$sql = 'CREATE TABLE korisnici( '.
   'emp_id INT NOT NULL AUTO_INCREMENT, '.
   'emp_name VARCHAR(20) NOT NULL, '.
   'emp_address  VARCHAR(20) NOT NULL, '.
   'emp_salary   INT NOT NULL, '.
    'primary key ( emp_id ))';

mysqli_select_db($conn,'test_db');
$retval = mysqli_query(  $conn, $sql);

if(! $retval ) {
   die('Nemogu napraviti tabelu zbog: ' . mysqli_error($conn));
}

echo "Tabela korisnici je napravljena\n";
mysqli_close($conn);
?>
```

Ovim kodovima možemo da izradimo i tabelu za ovaj slučaj na isti način.

Php skript za prikaz podataka iz baze

```
<?php
$dbhost = 'localhost';
$dbuser = 'root';
$dbpass = '';
$data= 'neka_baza';
$conn = mysqli_connect($dbhost, $dbuser, $dbpass,$data);

if(! $conn ) {
die('Could not connect: ' . mysqli_error());
}

echo 'Connected successfully';

// poziv tabele iz koje uzimamo podatake
$sql = "SELECT * FROM tbl_users";
$result = mysqli_query($conn, $sql);

// if petlja za izlistavlnje svih podatka u bazi podtaka
if (mysqli_num_rows($result) > 0) {

   // izlistavnje svakog reda u bazi
   while($row = mysqli_fetch_assoc($result)) {
      echo "id: " . $row["id"]. " - Name: " . $row["first_name"]. " " .
$row["last_name"]. "<br>";
   }
} else {
   echo "0 results";
}
mysqli_close($conn);
?>
```

Napomena: najčešće se javljaju greške ako se tačno ne upišu podaci koji se traže, pa skripta prijavi problem, jer je php osetljiv na upotrebu znakova(raspored velikih malih slova drugih znakova). Zato shodno upotrebi imenujete tabele sa tbl_paimetabele. Ili se greške najviše javljaju usled pisanja delova koda.
Ono što nam je sada potrebno je da pronađemo odgovarjući podatak u bazi. To se čini ovim kodom

```
<?php
$dbhost = 'localhost';
$dbuser = 'root';
```

```
$dbpass = '';
$data= 'neka_baza';
$conn = mysqli_connect($dbhost, $dbuser, $dbpass,$data);

if(! $conn ) {
die('Could not connect: ' . mysqli_error());
}

echo Povezani smo na server i bazu <br>';

$sql = "SELECT * FROM tbl_users";
$result = mysqli_query($conn, $sql);

if (mysqli_num_rows($result) > 0) {

    // prikazivanje svog sadržaja baze podatak i njene tabele tbl_users
    while($row = mysqli_fetch_assoc($result)) {
        echo "Prikaz svega sto ime u bazi<br>id: " . $row["id"]. " - Name: " .
                $row["first_name"]. "- Prezime: " . $row["last_name"]. "<br>";
    }
} else {
   // echo "0 results";
}
echo"<hr>";
$Ime="Miodrag";
$prezime="Trajanovic";

$sqli = "SELECT * FROM tbl_users WHERE id=2 ";
$result = mysqli_query($conn, $sqli);

$row = mysqli_fetch_array($result) ;
 echo "Pretraga po Id polju<br>id: " . $row["id"]. " - Name: " . $row["first_name"].
"- Prezime: " . $row["last_name"]. "<br>";

/* mysql komanda WHERE je ta koja vršenjem upoređnja unetog i onog u bazi vrši
proveru podataka*/
 $sqlx = "SELECT * FROM tbl_users WHERE first_name='$Ime'  ";
$result = mysqli_query($conn, $sqlx);

$row = mysqli_fetch_array($result) ;
 echo "Pretraga po imenu <br> id: " . $row["id"]. " - Name: " . $row["first_name"].
"-Prezime:" . $row["last_name"]. "<br>";

 $sqly = "SELECT * FROM tbl_users WHERE first_name='$Ime' AND
```

```
            last_name ='$prezime' ";
$result = mysqli_query($conn, $sqly);

$row = mysqli_fetch_array($result) ;
 echo "Pretraga po imenu i perzimenu <br>id: " . $row["id"]. " - Name: " .
$row["first_name"]. "-Prezime: " . $row["last_name"]. "<br>";

mysqli_close($conn);
?>
```

U gornjem kodu je prikazana pretraga ako ima uslov samo za jedno polje koje je brojna vrednost i za više polja kada je u pitanju pretraga prema podacima za više polja u bazi podataka. Ovi podaci mogu se i filtrirati kako bi se izbegli dulicirani podaci, dok za brisanje ili prepravke se upotrebljva id polje ciljnnog podatka.

Fninalni kod

Pošto je u predhodnom delu knjige Login/Logout react-php prikazan kod react comopnente u ovom delu biće prikazan i objašnjem kod php skripti i klase.
U ovaj kod omogućava da se prema unešenim podacima u react komponenti isti pošalju php skripti koja pokreće msql data bazu i u svom radu obaveštava ako ima problema, a ako su podaci postojeći onda ih on vraća nazad putem response react komponenti koja ih prikazje i upisuje u localStorage. Ono što još je potrebno dodati react komponenti preusmerenje na react komponentu korisnika, koja će početne podatke pročitati iz localStorage-a.
LocalStorage se upotrebljava za smeštanje kolačića u najćešćem broju slučajeva, ali sada biće upotrebljen za smeštanje primljenih podataka iz baze koji će biti prikazani u komponenti Korisnik.
Naziv ovog faila idex1.php možete promeniti u recimo login.php s time što te promene unosite i u delu za njegovo pozivanje u react kompnenti.

Php fail idex1.php

Ovaj fail omogućava vezu react aplikacije na serveru za posao logovanja (prijave) na nalog, pozivajući potrebne funkcije i vraćajući njihov odgovor nazad. Pošto fail funkcija ima i druge funkcije, možete isti fail preimenovati za upotrebu tih dodatnih funkcija ili uz upotrebu predhodnih delova ove knjige u ovom istom failu napraviti mogućnosti za pozivanje svih funkcija.

```
<?php
header('Accept: application/json');
header('Access-Control-Allow-Methods: POST,GET,PUT,DELETE,OPTIONS');
header("Access-Control-Allow-Headers: eToken,X-Requested-With,
        Content-Type");
header('Content-type: text/json; charset=utf-8;
```

```
application/x-www-form-urlencoded');
header('Access-Control-Allow-Origin: *');
ini_set('xdebug.overload_var_dump', 0);
include 'funkcije.php'; // pozivanje faila sa funkcijama

// postavljanje dekodiranih primeljenih podataka iz react komponente
$requestData = json_decode(file_get_contents('php://input'), true);

/* dodeljivanje vraćenih vrednosti iz php funkcije login i ujedno prosledjivanje
poslatih parametara php funkciji login za njen rad*/
$result = login($requestData['email'], $requestData['$pass']);

// dodeljivanje vrednosti promenjive $result php funkciji response
response($result);
```

Php fail konfig.php

Namena je da omogući povezivanje php skriptova sa bazom podataka. Data su dva primera potrebnog koda u zavisnosti kako će te upotrebiti u vašim aplikacijama. Umesto vrednosti sa desne strane koje su ovde upisujete vaše koje imate na serveru gde će se aplikacija naći.

```
<?php
/**
 * Created by IntelliJ IDEA.
 * User: trika
 * Date: 27-Aug-18
 * Time: 23:16
 */
define('DB_HOST', 'localhost');              // naziva domena
define('DB_UNAME', 'root');                  // administratorsko ime
define('DB_PASS', '');                       // šifra admine
define('DB_NAME', 'neka_baza');              // naziv data baze  koja se upotrebljava
define('DB_TBL_USERS', 'tbl_users');         // naziv tabele u bazi kojase poziva
/*
$dbhost = 'localhost';
$dbuser = 'root';
$dbpass = '';
$data= 'neka_baza';*/
```

Php fail DB.php

Ovo je php klasa, koju pozivaju funkcije da bi uradile svoje posao. Ona ujedno omogućava povezivanje sa bazom podataka, ali i ispituje da li postoji data baza sa unešenim imenom, kao i tabela u bazi, a ako to nepostoji onda ona vrši izradu istih

na ciljnom serveru. Tako da ona za svoj rad poziva samo konfiguracioni php fail konfig.php.

```
<?php
/**
 * Created by IntelliJ IDEA.
 * User: steewsc
 * Email: steewsc@gmail.com
 * Date: 25.11.2018
 * Time: 13:18
 */

include_once "konfig.php"; // uključivanje konfig.php faila

class DB extends mysqli {

    /** @var DB postavljanje null vrednosti u promenjivu */
    private static $instance = null;

    /**  funkcija koja proverava da li je predhodna vrednost promenjive null, i ako je to istina, onda pravi novu istancu prema unetim podacima u konfig.php failu, vraćajući ih nazad
*/
    public static function getInstance() {
        if(DB::$instance == null) {
            DB::$instance = new DB(DB_HOST, DB_UNAME, DB_PASS);
        }
        return DB::$instance;
    }

/* konstruktor*/
    public function __construct($host, $username, $passwd){
        parent::__construct($host, $username, $passwd);
        if(!$this->select_db(DB_NAME)) {
            if(!$this->installDB()) {
                response(null, "Could not create database.", true);
            }
        }
        if ($this->connect_error) {
            response(null, "Connect failed: code: " . $this->connect_errno . ': ' .
                    $this->connect_error, true);
        }
    }
```

```
    /**
     * @param $stmt mysqli_stmt
     * @return array – vraća podatke u obliku niza
     */
    public function getResult($stmt) {
        $stmt->execute();
        $result = $stmt->get_result();
        $arrRes = array();
        while ($row = $result->fetch_assoc()) {
            $arrRes[] = $row;
        }
        $stmt->close();
        return $arrRes;
    }

/* provera da li postoje baza podataka i tabela u njoj */
    private function installDB() {
        $cDB = $this->query('CREATE DATABASE IF NOT EXISTS `' .
DB_NAME . '` DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;');
        if(!$this->select_db(DB_NAME)) {
            response(null, "Could not create database.", true);
        }
        $cTbl = $this->query('CREATE TABLE IF NOT EXISTS `' .
          DB_TBL_USERS . '` (
           `id` bigint(20) NOT NULL AUTO_INCREMENT,
           `first_name` text NOT NULL,
           `last_name` text,
           `email` text NOT NULL,
           `password` text NOT NULL,
           PRIMARY KEY (`id`)
           ) ENGINE=InnoDB DEFAULT CHARSET=utf8;');
        return $cDB && $cTbl;
    }
}
```

Php fail funkcije.php

```
<?php
/**
 * Created by IntelliJ IDEA.
 * User: trika
 * Date: 21-Nov-18
 * Time: 13:33
 */
```

```
// Definisanje konstanti kojima se pozivaju funkcije o čemu ćemo kasnije
define('RN_LOGIN', 'login');
define('RN_REGISTRATION', 'registration');
define('RN_UPDATE', 'update');
define('RN_DELETE', 'delete');
define('RN_ALL_USERS', 'getusers');

// pozivanje potrebnih failova
include_once 'DB.php';
include_once 'konfig.php';

// funkcija login
function login($email, $pass) {
    if(!isset($email) || strlen(trim($email)) == 0) {
        response(null, 'Login failed. Email not set', true);
    }
    if(!isset($pass) || strlen(trim($pass)) == 0) {
        response(null, 'Login failed. Password not set', true);
    }
    $dbRows = getUser($email, $pass);
    if(count($dbRows) == 0) {
        response(null, 'Login failed. Wrong email or password.', true);
    }
    return $dbRows[0];
}

// funkcija za logout
function logout() {
// kod za raskidanje veze sa bazom , brisanje php sesija ako ih ima
}

// funkcija za brisanje u ovom slučaju korisnika
function deleteUser($id) {
    if(!isset($id) || (int)$id == 0) {
        response(null, 'Delete failed. Id not set or zero', true);
    }
    $db = DB::getInstance();
    $stmt = $db->prepare(
        "DELETE FROM " . DB_TBL_USERS . " WHERE id = ?");
    $stmt->bind_param('d', $id);
    $isDeleted = $stmt->execute();
    $stmt->close();
    if(!$isDeleted) {
        response(null, 'Delete failed.', true);
```

```
    }
    response(null, 'Delete success.');
}

// funkcija za preuređivanje unetih podataka u bazu
function updateData($data) {
    $id = $data['id'];
    $firstName = $data['ime'];
    $lastName = $data['prezime'];
    $email = $data['email'];
    $pass = $data['sifra'];

    if(!isset($firstName) || strlen(trim($firstName)) == 0) {
        response(null, 'Registration failed. First name not set', true);
    }
    if(!isset($lastName) || strlen(trim($lastName)) == 0) {
        response(null, 'Registration failed. Last name not set', true);
    }
    if(!isset($email) || strlen(trim($email)) == 0) {
        response(null, 'Registration failed. Email not set', true);
    }
    if(!isset($pass) || strlen(trim($pass)) == 0) {
        response(null, 'Registration failed. Password not set', true);
    }

    $db = DB::getInstance();
    $stmt = $db->prepare(
        "UPDATE " . DB_TBL_USERS . " SET first_name = ?, last_name = ?,
                email = ?, password =? WHERE id = ?");
    $stmt->bind_param('ssssd', $firstName, $lastName, $email, $pass, $id);
    $isSaved = $stmt->execute();
    $stmt->close();
    if(!$isSaved) {
        response(null, 'Update failed.', true);
    }
    // dohvati iz tabele promenjenog korisnika
    $result = getUser($email, $pass);
    if(count($result) == 0) {
        response(null, 'Update failed. Korisnik sa id: ' . $id . ' Ne postoji.', true);
    }
    return $result[0];
}
```

```
/* funkcija za dodavanje novog korisnika sa kontrolom ispunjenosti –validacijom
unosnih polja */
function createNewUser($firstName, $lastName, $email, $pass) {
   if(!isset($firstName) || strlen(trim($firstName)) == 0) {
      response(null, 'Registration failed. First name not set', true);
   }
   if(!isset($lastName) || strlen(trim($lastName)) == 0) {
      response(null, 'Registration failed. Last name not set', true);
   }
   if(!isset($email) || strlen(trim($email)) == 0) {
      response(null, 'Registration failed. Email not set', true);
   }
   if(!isset($pass) || strlen(trim($pass)) == 0) {
      response(null, 'Registration failed. Password not set', true);
   }
   $db = DB::getInstance();
   $stmt = $db->prepare(
      "INSERT INTO " . DB_TBL_USERS . " (first_name, last_name, email,
       password)
         VALUES (?, ?, ?, ?)");
   $stmt->bind_param('ssss', $firstName, $lastName, $email, $pass);
   $isSaved = $stmt->execute();
   $stmt->close();
   if(!$isSaved) {
      response(null, 'Registration failed.', true);
   }
   // dohvati iz tabele novog korisnika
   return getUser($email, $pass)[0];
}

// preuzimanje podataka samo jednog korisnika
function getUser($email, $pass) {
   $db = DB::getInstance();
   $stmt = $db->prepare("SELECT * FROM " . DB_TBL_USERS . "
            WHERE email = ? AND password = ?");
   $stmt->bind_param('ss', $email, $pass);
   return $db->getResult($stmt);
}

// izlistavanje podataka svih korisnika u bazi podataka
function getAllUsers() {
   $db = DB::getInstance();
   $stmt = $db->prepare("SELECT * FROM ". DB_TBL_USERS ." ORDER BY
id");
```

```
    return $db->getResult($stmt);
}

// izrada emaila
function createEmail() {
    $to = '';
    $subject = '';
    $message = '';
    $headers = '' . "\r\n" .
        'Reply-To: webmaster@example.com' . "\r\n" .
        'X-Mailer: PHP/';
    mail($to, $subject, $message, $headers);
}

/* Funkcija koja vraća rezultat rada svih funkcija u ovom failu react komponenti
koja je uputila zahtev u obliku JSON podataka */
function response($data, $message = null, $isError = false){
    if($isError) {
        echo json_encode(array('error' => array('message' => $message)));
    }else {
        if(isset($message)) {
            echo json_encode(array('message' => $message));
        }else{
            echo json_encode($data);
        }
    }
    exit(0);
}
```

Dodatno

Upotrebom koda fail idex1.php

```
<?php
header('Accept: application/json');
header('Access-Control-Allow-Methods: POST,GET,PUT,DELETE,OPTIONS');
header("Access-Control-Allow-Headers: eToken,X-Requested-With,
            Content-Type");
header('Content-type: text/json; charset=utf-8;
            application/x-www-form-urlencoded');
header('Access-Control-Allow-Origin: *');
ini_set('xdebug.overload_var_dump', 0);

include 'funkcije.php'; // pozivanje faila sa funkcijama
$requestData = json_decode(file_get_contents('php://input'), true);
```

```
/* pozivanje login funkcije umesto koda ova dva reda niže u idex1.php failu
postavite kod ispod naziva donjih php skriptova */
$result = login($requestData['email'], $requestData['$pass']);
response($result);
```

možemo napraviti više potrebnih failova sa kraćim kodom kopiranjem iznad reda komentara koda

```
/* pozivanje login funkcije i postavljenjem istog u novi otvoren fail dodajemo kod
ispod naziva koda php skriptova ispod i sačuvamo to sve u jednom folderu sa
imenom naziva koda php skriptova ispod.*/

fail logOut.php
   logout();

fail delete.php
   deleteUser($requestData['id']);

fail update.php:
   response(updateData($requestData));

fail sviKorisnici.php
   response(getAllUsers());

fail registracija.php
   response(createNewUser(
      $requestData['ime'], $requestData['prezime'], $requestData['email'],
      $requestData['sifra']));

fail slanjeEmaila.php
   createEmail();
```

Napomena: objasnio bih deo ovog koda kako bi vam bio jasnij rad.

```
$requestData = json_decode(file_get_contents('php://input'), true);
$result = login($requestData['email'], $requestData['$pass']);
response($result);
```

U ovom kodu imamo funkcije:

php://input -- vraća sve neobrađene podatke nakon HTTP zaglavlja zahteva, bez obzira na vrstu sadržaja. Te je na taj način i prihvaćen zahtev od react komponente upotrebom: fetch, axios i sl.

file_get_contents – namenjena da čita (failove i sve ono što se pozove i smesti unutar malih zagrada, između apostrofa ili navodnika) i da svoj rezultat smešta u string.

json_decode – dekodiranje stringa, ima dva parametra odvojena zarezom. Najkraće rečeno razdvaja na delove onogo što je unutar malih zagrada dodeljeno njoj, praveći jedan niz podataka. Pogledajte ovaj primer koda da bi vam bio jasniji rad:

```
<?php

$json = '{"a":1,"b":2,"c":3,"d":4,"e":5}';
print_r(json_decode($json));
echo "<br><br><br><br>";
print_r(json_decode($json, true));
?>
```

I dobićete ovaj prikaz

prvi red komande
print_r stdClass Object ([a] => 1 [b] => 2 [c] => 3 [d] => 4 [e] => 5)
drugi red komande
print_r Array ([a] => 1 [b] => 2 [c] => 3 [d] => 4 [e] => 5)

Znači dobli smo niz podataka kome možemo da pristupimo i da preuzmemo potrebne delove.
$requestData['$pass']); -- preuzimanje samo pojedinačnog niza podataka iz velikog niza čije podatke smo pridružili promenjivoj $requestData

response($result); -- funkcija koja vraća vrednost promenjive $result za slučaj da je to potrebno.

React komponente

Izrada react komponenti je slična izrada pomenutoj za php skripte. U ovom slučaju isti kod react komponente s time što u delovima koda se vrše izmene u ovom delu gde se ovaj kod postavlja kao nova vrednost stanja, kako bi se znalo koja se funkcija poziva

```
requestName: "login"
u funkciju
Prijava(){
   this.setState({ login: true, requestName: "login"})
}
```

ili za druge unsete jedan od onih u navodnicima koji prestavljaju vrednosti php konstanti:

RN_REGISTRATION
RN_ALL_USERS
RN_DELETE

i tako dobijate na primer :

```
Prijava(){
   this.setState({ login: true, requestName: "login"})
}
```

Druga izmena je u ovom delu koda

```
let dataURL = "http://localhost:80/fetchReact/prijava/idex1.php";
```

gde umesto index1.php uneste ime ciljnog faila sa kojm komponenta komunicira. Drugih izmena kada je u pitanju POST metod nema osim naziva unosnih polja u react komponentama, imena polja u bazi podataka. Dok za pregled podataka potrebno je upotrebiti predhodne delove knjige koji se odnose na pozivanje JSON podataka koji su na serveru. Kada prekopirate kod ove komponete u drugu izvršite izmene i sačuvaje je pod potrebnim vama imenom.
Obzirom da se uvek teži nečemu savršenijem i jednostavnijem, ne moramo da uvek izrađujemo gomilu failova, kao što sam ispred napisao, već možemo da umesto koda faila idex1.php upotrebimo jedan fail kao glavni kontroler za više namena. Jednostavnim pozivom određenih ključnih reči i upotrebom switch/case-a izradimo jedan ovakav php fail čiji je kod dole niže:

Fail index.php

```
<?php
header('Accept: application/json');
header('Access-Control-Allow-Methods: POST,GET,PUT,DELETE,OPTIONS');
header("Access-Control-Allow-Headers: eToken,X-Requested-With,
        Content-Type");
header('Content-type: text/json; charset=utf-8;
        application/x-www-form-urlencoded');
header('Access-Control-Allow-Origin: *');
ini_set('xdebug.overload_var_dump', 0);
include 'funkcije.php';

$requestData = json_decode(file_get_contents('php://input'), true);
```

```
// provera da li je nešto poslato ili ne
if(!isset($requestData)) {
    $requestData = $_POST;
    if(!isset($requestData) && count($requestData) == 0) {
        response(null, "No request data.", true);
        return;
    }
}

/* ako je nesto poslato serveru šta je poslato, nema nekog podatka u poslatom
zahtevu */
if(!isset($requestData['requestName'])) {
    response(null, "No requestName in request data.", true);
    return;
}

/* prema poslatim podacima aktivira se određeni slučaj koji poziva funkciju da
obradi podatke*/
switch ($requestData['requestName']) {
    case RN_LOGIN:
        $result = login($requestData['email'], $requestData['sifra']);
        response($result);
        break;
    case "logOut":
        logout();
        break;
    case RN_DELETE:
        deleteUser($requestData['id']);
        break;
    case RN_UPDATE:
        response(updateData($requestData));
        break;
    case RN_ALL_USERS:
        response(getAllUsers());
        break;
    case RN_REGISTRATION:
        response(createNewUser(
            $requestData['ime'], $requestData['prezime'], $requestData['email'],
            $requestData['sifra']));
        break;
    case "slanjeEmaila":
        createEmail();
        break;
    default:
```

```
        response(null,"Za traženu radnju ne postoji odgovarajući slučaj.", true);
        break;
}

return json_response(200,'poruka');
```

U prvom redu podaci se prihvataju kao i kod predhodnog faila idex.php

```
include 'funkcije.php';

$requestData = json_decode(file_get_contents('php://input'), true);
```

Nakon čega sledi provera da li je nešto pristiglo ili ne, te shodno tome funkcija response vraća rezultat

```
// provera da li je nešto poslato ili ne
if(!isset($requestData)) {
    $requestData = $_POST;
    if(!isset($requestData) && count($requestData) == 0) {
        response(null, "No request data.", true);
        return;
    }
}
```

Ali, ako je nesto poslato serveru i šta je poslato, u prvom redu gleda se koji je sadržaj pristigao putem promenjive iz react aplikacije requestName. Taj sadržaj promenjive requestName obzirom da nije samo jedan ne promenjiv sadržaj već je on promenjive prirode u zavisnosti od potreba i zahteva on se prihvata kao jedan niz. Te zato promenjiva u php-u $requestData ga kao takvog i preuzima za svoju vrednost. Ta vrednost se upoređuje sa onom vrednošću konstanti u failu funkcije.php

```
if(!isset($requestData['requestName'])) {
    response(null, "No requestName in request data.", true);
    return;
}

// konstante u failu funkcije.php
define('RN_LOGIN', 'login');
define('RN_REGISTRATION', 'registration');
define('RN_UPDATE', 'update');
define('RN_DELETE', 'delete');
define('RN_ALL_USERS', 'getusers');
```

U ovom slučaju ne samo da se kaže da je na primer 'RN_LOGIN' jednako 'login', a već se tako i naziva određen slučaj koji u svom sastavu ima sa imenom funkciju login koju poziva i ujedno joj prosleđuje potrebne argumete za njen rad.

```
switch ($requestData['requestName']) {
   case RN_LOGIN:
      $result = login($requestData['email'], $requestData['sifra']);
      response($result);
      break;
```

Uzimajući da je i pozvan celokupni sadržaj faila funkcije.php upotrebom

```
include 'funkcije.php';
```

u index.php failu, to se ujedno i poziva istoimena funkcija login, i kao njen rezultat rada vraća se preko funkcije response nazad nama u react aplikaciju.

Napomena: Sa manjim izmenama failova index.php i funkcije.php praktično možemo da imamo kontroler za mnoge primene za uključivanje rada servera u react aplikacijama. Jer, je manje više sve slično. Tako, funkciju createNewUser, možemo da unosom drugih imena polja primenimo za izradu postova.

Dodavanje delova react rutera

Ono što je potrebno u ovom slučaju je da se u slučaju da eu vraćeni podaci da se izvrši preusmerenje na korisnikovu stranicu sa njegovim zaštićenim url-om, kako ne bi i drugi neovlašćeno pristupali podacima korisnika. Za ovo je potrebno kod korisnika postaviti if else petlju gde bi postavili uslov ako su podaci traženi poslati da se vrši prusmerenje ili da se ostane na istoj stranici dok se podaci ne unesu kako treba. Drugi način je da se sa servera vrati nekoj promenjivoj logički uslov. Ali, bolji je način da se upotrebi kombinacija oba uslova. Tačnije ako su oni identični da se onda izvrši preusmerenje, u suprotnom da bude ponudjena mogućnost da se ili odustane od prijave ili pokuša opet.

```
import React from 'react';
import {Redirect} from 'react-router-dom';
import 'whatwg-fetch';
class DemoComponent extends React.Component {
.....
fetch(req)
   .then((response)=>{
      if(response.ok){
         return response.json();
      }else{
         throw new Error("losa konekcija");
```

```
        }
    })
    .then((response) =>{
        this.setState({podaci: response});

        // provera da li response ima neke vrednosti ili ne
        if(response !==''){
        console.log(response ," nije prazna" );// provera u konzoli da li ime vrednosti
            this.setState({login: true}); // postavljanje stanja koje se upotrebljava za
                                          // okidanje u returnu
        }
        console.log(response,"ispod" );
        alert(response.error.message);
    })
.....
return (
    <div>

        {this.state.login ? <Redirect to="/korisnik" />:" stanje drugo"}
        <form onSubmit={this.handleSubmit.bind(this)}>
.....
export deafult DemoComponent;
```

Umesto na reda sa this.state.login može se napisati

```
<Router>
<Route path="/korisnik/:username" exact strict render{({match}) =>(
  this.state.login ? (<Korisnik username={match.params.username}/>) :
  (<Redirect to='/ '/>)
)} />
</Router>
```

Ali, ovaj i slični kodovi trebali bi da se postave u drugoj komponenti koja poziva DemoComponent komponentu, a u njoj postaviti tagove sa Router komponentom. Razlog za ovim pristupom i načinom je taj što Router tag može imati samo jednu child komponentu izuzev ako se u njega kao child upotrebi div tag, a između div tagova postaviti sve drugo potrebno. Nakon čega ovu komponentu uključiti u App komponetu u src folderu.

```
import React from 'react';
import {BrowsereRouter as Router, Redirect} from 'react-router-dom';
import DemoComponent from './ DemoComponent ';
class LoginAppComponent extends React.Component {
render(){
    return(
```

```
<Router>
   <div>

    ..... ovde možete dodati još putanja
    <Route path="/korisnik/:username" exact strict render{({match}) =>(
   this.state.login ? (<Korisnik username={match.params.username}/>) :
   (<Redirect to='/ '/>)
   )} />
   <div>

  </Router>
```

.....

To zato što da bi se mogli kretati po putanjama potrebno je one i postoje, zato se one unapred odrede i onda je moguće i upotrebiti istoriju kretanja (history.location), znači upotrebiti klasičan java skript kod za te namene.
U te svrhe upotrebiću delove koda " Pod nazivom prolaz kroz više nivoa ", kao i druge delove ove knjige koje se odnose na dodatak react paket react-router-dom.

Kontekst(Context) i rutiniranje

Među promenama u Reactu 16.3 napravljena je nova stabilna verzija Context API-ja. Pogledajmo kako to radi u slučaju izrade zaštićenog dela putanja u react ruteru.

Što je kontekst?

Kontekst je o inkapsuliranju stanja. Omogućuje nam da podatke s komponente koja je vlasnik matičnih podataka (perent), prenesemo na bilo koju potkomponentu (child), u može se nazvati stablu komponenti, tako da možemo upravljanjem vrednostima stanja poslati svakoj od njih te vrednosti, koje one prihvataju kao vrednosti osobina (props).

Nije li to, ono zašto je namenjen Redux?

Da, Kontekst radi slično tome kako se komponente mogu povezati s globalnom vrednostima stanja promenjivih (state) Redux-a. Međutim, izvorni element poput konteksta često će biti bolje rešenje za male i srednje aplikacije kojima nije potreban složeniji prostup koji nudi Redux.

Kontekst sadrži tri elementa:

createContext – Njegovim pozivanjem vraćaju se nekoliko komponenti, Provider i Consumer. Praktično se stvara odnos roditelja i dece po pitanju vlasnistva na vrednostima stanja (state).
Prodavac (Provider) - komponenta koja omogućuje da se proslede vrednosti stanja promenjivih proslede jednom ili na više potrošača (child) kojima će biti omogućene promene props-a.

Potrošač (Consumer)-komponenta, prakticno child, je komponenta kojoj je omogućeno da prima promene svojih props-a putem prijema vrednosti state od strane provajdera.

Kako upotrebiti Kontekest za rutiniranje

Izradićemo aplikaciju sa dve rute ili putanje, gde će jedna biti odredišna sa globalnim pristupom. Dok, druga će biti administrativna (dashboard) sa ograničenjem prilikom prijave na nalog(login).
Upotrebom dodatne logike i komponenti u ovoj knjizi možemo da izradimo kompletnu prijavu na nalog, iz predhodnih primera.

Komponenta zaglavlje(header.js)

Namenjena je se pomoću nje povežu i ostale komponente aplikacije, pa će u njoj i biti izrađen React Context.

```
import React from 'react';
const AuthContext = React.createContext();
```

Gde smo dodelili konstanti AuthContext vrednost React.createContext(); , tako da smo odredili onamo gde je primenimo da ona napravi kontekst kao njenu instancu. Iza toga napravićemo jednu klasu pošto je potrebno da ovom prilikom upravljamo vrednostima stanja promnjivih dinamički.

```
class AuthProvider extends React.Component {
  state = { isAuth: false }
  render() {
    return (
      <AuthContext.Provider
        value={{ isAuth: this.state.isAuth }}
         >
        {this.props.children}
      </AuthContext.Provider>
    )
  }
}
const AuthConsumer = AuthContext.Consumer

export { AuthProvider, AuthConsumer }
```

Ali u ovoj klasi izradićemo i Provajdera i korisnika, što ćemo i exportovati u drugim komponentama gde nameravamo da primenimo. Ono što je karakteristično u ovom delu je način na koji je napisan deo koda eksporta i klasa i konstanti ili bilo čega drugog. Samim tim možemo da samo potvrdimo mogućnosti koje nudi ova

tehnologija u pogledu pisanja manje koda u aplikaciji. Jer da imamo eksportovati više toga, znači samo dodajemo zarez i iza njega ime toga što se šalje dalje. Nemoramo više puta pisati export.
Provajder se upotrebljava u glavnoj izlaznoj komponenti istovetno kao BrowserRouter-u ili Router-u. Znači on je taj koji omotava naziv komponente u glavnoj izlaznoj komponenti nad kojima ima uticaj, ne bitno gde se one budu našle u aplikaciji. Kao što se vidi u donjem kodu index komponente

Fail index.js

```
import React from 'react';
import { AuthProvider } from './AuthContext';
import Header from './Header';

const App = () => (
  <div>
    <AuthProvider>
      <Header />
    </AuthProvider>
  </div>
);
```

Preko koje ćemo u ovom primeru pozvati upotrebom delove react-router-dom-a i ostale u ovo slučaju potrebne. Ali, umesto Header komponente mogu se postaviti dosta komponeti koje pored svojih podataka dele podatke preko provajdera.

Deo korisnik koji kontekts ovom prilikom isto izrađuje postavlja se u komponenti koja ga upotrebljava, kako za svoj rad, tako i za upravljanje vrednostima svojh podataka.

```
import { AuthConsumer } from './AuthContext'
import { Link } from 'react-router-dom'
export default () => (
  <header>
    <AuthConsumer>
          .....
    </AuthConsumer>
  </header>
)
```

Ono što se da zaključiti je da kontekst korisnici moraju imati u sebi funkciju kao svoje direktno dete (child), kako bi u njega mogli prosleđivati vrednosti dobijene iz provajdera.

Što izgledalo ovako

```
<AuthConsumer>
   {({ isAuth, login, logout }) => (
      <div style={headerStyle}>
         <h3>
            <NavLink className="btn-dark btn-lg"  to="/">
               Glavna stanica
            </NavLink>
         </h3>

         {isAuth ? (
               <ul>
                  <NavLink className="btn-primary btn-lg" to="/dashboard">
                     Aministracija
                  </NavLink>
                  <button className="btn-danger btn-lg"
                   onClick={logout}>Odjava</button>
               </ul>
            ) : (
               <button className="btn-primary btn-lg"
                  onClick={login}>Prijava</button>
            )}
      </div>
   )}
```

Gde imamo ispitivanje uslova logičke promenjive isAuth, čija je početna vrednost false, što omogućava samo vidljivost dugmeta Prijava.

Fail AuthContext.js

```
import React from 'react'

const AuthContext = React.createContext();

class AuthProvider extends React.Component {
   state = { isAuth: false };
   constructor() {
      super();
      this.login = this.login.bind(this);
      this.logout = this.logout.bind(this);
   }

   login() {
      setTimeout(() => this.setState({ isAuth: true }), 1000)
```

```
  }

  logout() {
    this.setState({ isAuth: false })
  }

  render() {
    return (
      <AuthContext.Provider
        value={{
          isAuth: this.state.isAuth,
          login: this.login,
          logout: this.logout
        }}
      >
        {this.props.children}
      </AuthContext.Provider>
    )
  }
}

const AuthConsumer = AuthContext.Consumer;

export { AuthProvider, AuthConsumer }// ovo je još jedan način eksportovanja
                                    // više komponenti
```

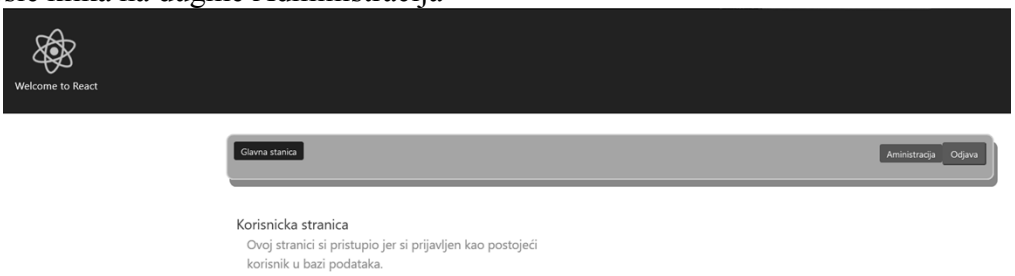
Ako u kodu state = { isAuth: false }; promenimo stanje u true videćemo efekte ručno, ali pošto smo to povezali za događaj onClick to će se i klikom na dugme Odjava promeniti u aplikaciji. Što prikazuje kod u Header komponenti.

Pristup zaštićenim putanjama(Route) React kontekstom

Uzimajući u obzir da na svim veb sajtovima, kao i svuda oko nas uvek ima delova koji su dostupni svima i delova kojima se može pristupiti uz određne unešene tekstulane i druge simbole, ili pak u novije vreme upotrebom različitih vrsta uređaja kojima se delovi tela prevode u električne inpulse, koje aplikacija proverava sa već unešenim prilikom izrade naloga. Ne bitno da li je to mrežnjača oka, otisak prsta ili cele sake leve ili desne ruke. Ta autoindetifikacija je jedan od uslova da bi smo negde mogli pristupiti. U našem slučaju nećemo pristupati ničemu već samo klikom na dugme Prijava otkriti taj zaštićen deo aplikacije. Naravno, upotrebom delova ove knjige možete i sami već napraviti potrebne ostale delove. Ako kako pokazuje slika ispod biramo između osnovne stranice

Posle klika na Prijava dugme

Posle klika na dugme Administracija



Možemo da biramo između dve stranice Polazne ili ove poslednje i da se odjavimo, što je moguće kada dopunimo fail index.js kodom da izgleda ovako

```
import React from 'react'
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom'
import { AuthProvider } from './AuthContext'
import Landing from './Landing'
import Dashboard from './Dashboard'
import Header from './header'
import ProtectedRoute from './ProtectedRoute'

const App = () => (
   <div className="container">
      <Router>
         <AuthProvider>
            <Header />
            <Switch>
               <ProtectedRoute path="/dashboard" component={Dashboard} />
```

```
                <Route path="/" component={Landing} />
              </Switch>
            </AuthProvider>
        </Router>
    </div>
);
export default App;
```

Gde između komponente react-router-dom-a <Switch> se postavljaju dve staze: jedna zastićena druga vidljiva svima, koja se odnosi na osnovnu stranicu u aplikaciji.

Fail ProtectRoute.js

```
import React from 'react'
import { Route, Redirect } from 'react-router-dom'
import { AuthConsumer } from './AuthContext'

const ProtectedRoute = ({ component: Component, ...rest }) => (
    <AuthConsumer>
        {({ isAuth }) => (
            <Route
                render={props =>
                    isAuth ? <Component {...props} /> : <Redirect to="/" />
                }
                {...rest}
            />
        )}
    </AuthConsumer>
);
export default ProtectedRoute
```

Ona u svom kodu između AuthConsumer na osnovu stanja vrednosti IsAuth uključuje jednu od dve mogućnosti upotrebom strelaste funkcije koja u svom sastavu ima if else petlju

```
  isAuth ? <Component {...props} /> : <Redirect to="/" />
```

Čime se proverava da li je korisnik kliknuo na dugme Prijava i time pozvao zaštićeni deo aplikacije da se prikaže.U ovom delu možemo da postavimo uslov, ako su vraćeni podaci iz baze, da aplikacija ide na dashboard, ako nema ničeg iz bazer da ostane naistom mestu ili da pojavi neku drugu mogućnost. Ali, klikom u ovom slučaju na dugme Prijava pokreće se i deo u index.js failu koji je pozvao zaštićenu komponentu Dashboard

<ProtectedRoute path="/dashboard" component={Dashboard} />

Čime ne samo da se pojavilo i ono na poslednjoj slici već i promenio se url u brovseru od

http://localhost:3000/

na

http://localhost:3000/dashboard

Dodavanjem ovog jednostavnog koda u komponentu Dashboard
Klasa Button izrađuje izgled dugmeta kojoj se pripisuju props osobine izrađenog dugmeta u klasi Message, kojoj te osobine šalje komponeta MessageList koja je vlasnik podataka (state).

```
class Button extends React.Component {
   render() {
      return (
         <button style={{background: this.props.color}}
               className="btn-danger btn-lg">
            {this.props.children}
         </button>
      );
   }
}

class Message extends React.Component {
   render() {
      return (
         <div>
            {this.props.text} <Button color={this.props.color}>Delete</Button>
         </div>
      );
   }
}

class MessageList extends React.Component {

   render() {
      const messages=[{'text':'prva'},{'text': 'druga'},{'text':'treca'}];
      const color = "purple";
      const children = messages.map((message) =>
```

```
            <Message text={message.text} color={color} />
        );
        return <div>{children}</div>;
    }
}
```

Uključivanjem <MessageList / > u dashboard.js ispod tagova paragrafa

```
</p>
<hr/>
<div className="row">
<MessageList />
</div>
```

Kod preostalih failova

Fail Landing.js

```
import React from 'react';

const Landing = () => (
    <div className="container" style={linkStyle}>
        <h2 className="text-danger"  >
            <strong>Osnovna stranica aplikacije</strong>
        </h2>
    </div>
);

export default Landing;

const linkStyle = {
    color: 'red',
    marginTop:'50px',
    borderRadius:'10px'
};
```

Fail Header.js

```
import React from 'react'
import { AuthConsumer } from './AuthContext'
import { NavLink } from 'react-router-dom'

const headerStyle = {
    display: 'flex',
    backgroundColor: '#26c6da',
```

```
    justifyContent: 'space-between',
    padding: 10,
    border: '2px solid pink',
    boxShadow:'5px 10px #888888 ',
    borderRadius:'10px',
    marginTop:'30px'
};

export default () => (
    <header>
        <AuthConsumer>
            {({ isAuth, login, logout }) => (
                <div style={headerStyle}>
                    <h3>
                        <NavLink className="btn-dark btn-lg"  to="/">
                            Glavna stanica
                        </NavLink>
                    </h3>
                    {isAuth ? (
                            <ul>
                                <NavLink className="btn-primary btn-lg" to="/dashboard">
                                    Aministracija
                                </NavLink>
                                <button className="btn-danger btn-lg"
                                onClick={logout}>Odjava</button>
                            </ul>
                        ) : (
                            <button className="btn-primary btn-lg"
                              onClick={login}>Prijava</button>
                        )}
                </div>
            )}
        </AuthConsumer>
    </header>
)
```

Fail Dashboard.js

```
import React from 'react'

const Dashboard = () => (
    <div className="container" style={linkStyle}>
        <h2>Korisnicka stranica </h2>
        <p className="col-md-5 text-success text-justify">
```

```
            Ovoj stranici si pristupio jer si
            prijavljen kao postojeći korisnik u bazi podataka.
        </p>
        <hr/>
        <div className="row">
        <MessageList />
        </div>
    </div>
);

export default Dashboard

const linkStyle = {
    color: 'red',
    marginTop:'50px',
    borderRadius:'10px'
};
```

Napomena: Pošto je osetljiv na upotrebljene znakove prilikom imenovanja i slicno potrebno je tome obratiti veliku pažnju, jer je to dosta čest problem nastajanja grešaka koje dovode da aplikacija ne radi.

Postavljanje aplikacije na server

Pošto završimo izradu react aplikacije, potrebno je od sveg koda upotrebom koda u donjem redu sklopiti aplikaciju (bildovati kako se to kaže). Za ovu namenu u konzoli unesite ovaj kod

```
E:\wamp64\www\fetchReact\react-app > npm run build
```

Napomena:
Ovo je putanja do moje aplikacije E:\wamp64\www\fetchReact\react-app> kod vas biće drugačije.
Iza čega u konzoli dobićete sledeće redove kao odgovor na uneto u konzoli npm run build

```
> fetch-test@0.1.0 build E:\wamp64\www\fetchReact\react-app
> react-scripts build

Creating an optimized production build...
Compiled successfully.

File sizes after gzip:

  37.55 KB  build\static\js\main.ada9d971.js
```

```
  21.02 KB  build\static\css\main.166d0f5c.css

The project was built assuming it is hosted at the server root.
You can control this with the homepage field in your package.json.
For example, add this to build it for GitHub Pages:

  "homepage" : "http://myname.github.io/myapp",

The build folder is ready to be deployed.
You may serve it with a static server:

  serve -s build

Find out more about deployment here:

  http://bit.ly/2vY88Kr

E:\wamp64\www\fetchReact\react-app>
```

Ovim je u projektnom folderu napravljen folder build u kome su svi izrađeni potrebni failovi budućeg veb sajta. Da bi ste proverili sam rad u serverskom okruženju sa statičkim serverom unesite kod dole ispod:

```
E:\wamp64\www\fetchReact\react-app>serve -s build -p 81
```

Ovaj kod kaže da će statički server raditi na portu lokal servera broj 81, ali kako se vidi u ovom dole prikazu, ako ste priključeni na internet možet uneti vaš IP iza čega i broj porta ili jednostavno copy/paste prekopirati potrebno u vas brovser.

```
Serving!

    - Local:            http://localhost:81
    - On Your Network:  http://192.168.1.105:81

    Copied local address to clipboard!
```

Napomena: Ovi uokvireni podaci dobijaju se automacki, jer ih formira sam uneta komanda ispred.

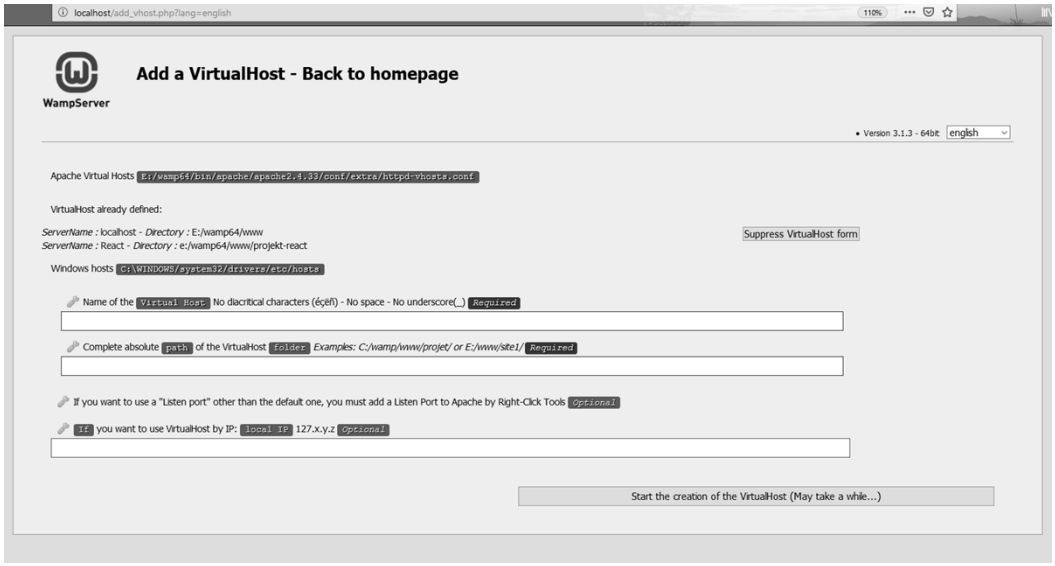
Za prenos ili postavljanje aplikacije na server samo je ostalo prekopirati sadržaj dobijenog foldera build na server gde će se nalazi budući veb sajt, ako radite SPA react aplikaciju, dok ako se kao u ovom slučaju upotrebljava react i php u tom slučaju osim sadržaja foldera build treba prekopirati i folder sa php skriptovima i

bazu podataka ili napraviti tamo direktno ili upotrebiti kod php za njenu izradu kao i pripadajućih tabela.
Za slučaj da želite da proverite na svom računaru i lokalhostu, moguće da će vam se desiti da aplikacija možda neće raditi korektno. Nije to neispravnost aplikacije, već je znak da morate izršiti potrebna podešavanja na svom računaru vezana za privilegije pristupa vašem lokalhostu. Tu imate dve mogućnosti:

1. izrada novog vitrualnog hosta
2. podešavanja koja se odnose dozvole u failu httpd-vhosts.config

U zavisnosti od toga na kom računaru radite i koji programski paket upotrebljavate i jedno i drugo morate pogledati u dokumentaciji za vaš slučaj. Pokazaću za slučaj da upotrebljavate Vindovs i Vamp paket.

Ovo je ono osnovno što se vidi nakon pokretanja wamp-a. Klikom na Add a Virtual Host pokreće se procedura kojom se vrši izrada virtualnog hosta.



Gde unosite tražene podatke, i po završetku toga kliknete na Start dugme, čime ste dobili vrtualni host sa unetim podacima. I kada se vratite ili ručno ili automacki na predhodnu sliku dobićete to vaše ime za host koje ste uneli ispod natpisa Your VirtualHost. Gde klikom idete direktno na vaš upravo napravljeni poseban serverski prostor. Dok sadržaj faila httpd-vhosts.config biće sličan ovome

```
# Virtual Hosts
#
<VirtualHost *:80>
  ServerName localhost
  ServerAlias localhost
  DocumentRoot "${INSTALL_DIR}/www"
  <Directory "${INSTALL_DIR}/www/">
    Options +Indexes +Includes +FollowSymLinks +MultiViews
    AllowOverride All
    Require all granted
  </Directory>
</VirtualHost>

#Ovo je novi kreiran virtualni host
<VirtualHost *:80>
        ServerName React
         ServerAlias React
        DocumentRoot "e:/wamp64/www/projekt-react"
        <Directory  "e:/wamp64/www/projekt-react/">
                Options +Indexes +Includes +FollowSymLinks +MultiViews
                AllowOverride All
                Require  all granted
```

```
        </Directory>
</VirtualHost>
```

Postoje i načini gde se vrše promene u failu .htacces.config, no za to je potrebno u zavisnosti od slučaja dosta toga, pa ta mogućnost neće biti sada objašnjena, jer obuhvata i proxy i sl.

Kraće o dodatnim paketima

React je praktično java skript namenjen da se napravi sam izgled budućih veb stranica koje bi se prikazivale kod korisnika, te tako namenjen neke druge poslove ne može da uradi bez dodataka. Oni su na internetu na ovom linku
https://www.npmjs.com
Gde možete pročitati o njima, koliko da se upoznate sa namenom i načinom instalacije. Neke od njih moraćete da prijavite putem webpack-a i ako ste ih instalisali, kako bi kod u aplikaciji, znao kako da ih upotrebi i gde da ih pronađe. Ali, bilo kako bilo u package.json failu su oni podeljeni prema vašim potrebama na one koji su namenjeni razvoju aplikacije i one koji moraju biti stalno sa kodom aplikacije za vreme izrade aplikacije. Pored toga neki dodaci zahtevaju i podešavanja u package.json.
Tako da je najbolje rešenje u početku raditi automatizovanim načinom izrade react aplikacija. Dole ispod ukratko o još nekim react-ovim dodacima u par reči.

body-pareser biblioteka koda namenjenom za promenu formata dolazećeg zahteva u JSON objekt, njena upotreba u serverskom delu react aplikacija, kada su primenjeni u njemu node i express
cors upotreba za konfigurisanje express paketa, kako bi se dodala zaglavlja koje prihvata vaš API, a koji dolaze od drugih izvora. Više o tome na vebsajtu mozile:
https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
Express vrsta paketa kojm se pravi server za react i node aplikacije
helmet ova biblioteka pomaže da se osigura Express aplikacija sa različitim HTPP zaglavljima
morgan : namena je da osigura dodatne mogućnosti prijava (logging) na Express aplikaciju
express-jwt: Middleware koji potvrđuje JSON Web Token (JWT) i postavlja req.user svojim atributima.
jwks-rsa: biblioteka za pronalaženje javnih ključeva RSA iz završne tačke JWKS (JSON Web Key Set).

Recenzija

(izvod iz recenzije)

Ovaj prirucnik vrlo jednostavno i bez ikakvih suvisnih cinjenica sa školskim pristupom vođenja tematike, omogućava čitaocima da u širokom prostoru nauke i tehnike pronađu sopstveni put i način, kojima će sami razvijati sopstvene principe. Možda, i u dalekoj budućnosti izgraditi svoje sopstvene programske aplikacije, operativne sisteme za svoje definisane ciljeve. U svojim nastojanjima da pored uvođenja u dubine programiranja, ipak ostavi i poneki zadatak budućnosti, da i ona sama na svoj način taj problem prevaziđe.

mr Miloš Vujić, dipl.ing,
magistar pedagoško tehničkih nauka

Literatura

Šta su funkcionalne, stateless components?
https://javascriptplayground.com/functional-stateless-components-react/

kursevi
https://egghead.io/lessons/react-stop-memory-leaks-with-componentwillunmount-lifecycle-method-in-react

https://egghead.io/courses/start-learning-react?fbclid=IwAR0MoE_l2nUuoWk1VbcQF0NeWm0vDteuQYfZ-1JUQwGvdrYJbWOJIipT9Zo

https://reactjs.org/community/videos.html?fbclid=IwAR33jNA_DI8Wz9jeXu4qqvA3MLy_FUsROAgCSxGzz_J3XegoA1FJxgvE0dg

cros
https://youtu.be/Bild2vT1l5U?list=PLdIz6gaqfRW2mursasiYf7x9fqtX0k-nE

link za React, Redux, ES6, and more
https://github.com/markerikson/react-redux-links?fbclid=IwAR1QZUkLvxBjMwyFBGK3l2AGhludPin9nfJ5mC2FRD0hExIs6s2nXFZSCmg

https://youtu.be/izz8l8rTSxY React Tutorial #01 - Build A Todo List
https://youtu.be/oRL-pttfNSc React Client Side Authentication

Tutorijali za React
https://appdividend.com/category/react-js/
https://www.scribd.com/document/337704114/React-Tutorial

Object.keys, values, entries
https://javascript.info/keys-values-entries
ddoavanje upotrebom concat this.setState(_.concat(this.getState(), { value }))
Data Structures: Objects and Arrays
https://eloquentjavascript.net/04_data.html

Heres how React's New Context API Works
https://github.com/wesbos/React-Context
https://youtu.be/XLJN4JfniH4

https://bitsrc.io/reactstrap

singup and login for php link https://youtu.be/xb8aad4MRx8
https://youtu.be/qVU3V0A05k8?list=PL0eyrZgxdwhwBToawjm9faF1ixePexft-
https://youtu.be/f85jvD_Y8Ck
https://www.codementor.io/blizzerand/building-forms-using-react-everything-you-need-to-know-iz3eyoq4y

How to create a complete login system in PHP (UPDATED 2018 VIDEO LINK IN DESC!) https://www.youtube.com/watch?v=xb8aad4MRx8&feature=youtu.be

php CRUD
https://youtu.be/mjVuBlwXASo

Introduction to PHP Programming | PHP Tutorial | PHP For Beginners | Learn PHP Programming
https://www.youtube.com/watch?v=qVU3V0A05k8&list=PL0eyrZgxdwhwBToawjm9faF1ixePexft-&ab_channel=mmtuts

pogledati i proučiti ovaj kod za react ruter https://codesandbox.io/s/k323w8nrn5

$ git clone https://github.com/austinroy/demo-router.git

https://scotch.io/bar-talk/getting-started-with-react-router-v4
https://scotch.io/tutorials/conditional-routing-with-react-router-v4
https://scotch.io/tutorials/routing-react-apps-the-complete-guide kompletno

Moje pritanje i odgovor korisnika grupe React Srbija na fejs buku Zarko Turicanin

Postovani kako da vratim podatke promenjive $data iz php faila gde je i react fail.
iz ovog niza mi vraca vrednost
'error' => array('message' => $message)
kada je ovako napisano response($data, $data, true);
https://codesandbox.io/s/0o3548143v

Zarko Turicanin Nije potreban tako specifičan tutorijal. Tebi u suštini treba da bolje razumeš tok događaja u tipičnoj react aplikaciji i onda će stvari same po sebi da budu jasne.

Ukratko:
- React aplikacija će se iscrtati samo jednokratno na bazi inicijalnog statea koji stavljaš u construktor. I nakon toga ce takodje samo jednom biti pozvana componentDidMount()
- Komponenta može ponovo da se izmeni samo promenom statea ali kako kad se sve izvršilo samo jednom?
Rešenje je da se u componentDidMount() stavi "listener" koji nastavlja da živi i nakon prvobitnog iscrtavanja. Listener zanima samo jedan jedini event "STATE_TREBA_DA_SE_PROMENI" i kada se taj event desi on poziva setState() sa novim stateom i to natera React da obavi promenu na ekranu.
- To je suština za početnike. Sve ostalo obavljaš van React aplikacije u nekom drugom kodu koristći neku drugu filozofiju, neke druge frameworke. React-u se obratiš event-om tek kada pripremiš novi state. Reactova uloga je samo to da kada dodje do izmena state-a da mu javiš i sve ostalo nije njegova briga. on je samo "View".

Eksperimentisi sa sledecim kodom: https://pastebin.com/995TN9VX
pa kad "provališ" onda ucis neki framework koji to radi elegantnije sa manje koda npr. Redux.
https://pastebin.com/995TN9VX?fbclid=IwAR1KNVPHBxZmLwEWLq1ZSRlIh3Wa4Sa_kGG1dPYW09FvpJQOZL_tOwgtjR0

https://youtu.be/6GaDgI741E8 ruter

https://www.digitalocean.com/community/tutorials/how-to-create-temporary-and-permanent-redirects-with-apache
https://codesandbox.io/s/rr00y9w2wm ruter
https://auth0.com/blog/react-router-4-practical-tutorial/

CIP - Каталогизација у публикацији - Народна библиотека Србије, Београд

004.42:004.738.12
004.438JAVA

ТРАЈАНОВИЋ, Миодраг, 1961-
Osnove Nodejs-React-Webpack-Babel-Bootstrap-PHP-Mysqli / Миодраг Трајановић. - 1. изд. - Ћићevac : M. Trajanović, 2019 (Beograd : Dobrotoljublje). - 477 str. : ilustr. ; 25 cm + 1 elektronski optički disk (DVD) ; 12 cm

Tiraž 100. - Bibliografija: str. 475-477.

ISBN 978-86-920321-2-7

a) Web презентације - Програмирање b) Програмски језик "Java"
COBISS.SR-ID 273107724