

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5299

Korisničko sučelje knjižnice za razmjenu multimedijskog sadržaja

Petar Kovačević

Zagreb, lipanj 2017.

Zagreb, 10. ožujka 2017.

ZAVRŠNI ZADATAK br. 5299

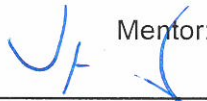
Pristupnik: **Petar Kovačević (0036484664)**
Studij: Računarstvo
Modul: Programsko inženjerstvo i informacijski sustavi

Zadatak: **Korisničko sučelje knjižnice za razmjenu multimedijских sadržaja**

Opis zadatka:

Načiniti korisničko sučelje knjižnice za razmjenu multimedijских sadržaja u kojoj će korisnici nuditi vlastite knjige, stripove, diskove i druge multimedijске sadržaje, a koja će omogućiti pretraživanje, rezervaciju i posudbu sadržaja.


Zadatak uručen pristupniku: 10. ožujka 2017.
Rok za predaju rada: 9. lipnja 2017.


Mentor:

Prof. dr. sc. Vedran Mornar


Djelovođa:

Doc. dr. sc. Mirjana Domazet-Lošo

Predsjednik odbora za
završni rad modula:


Izv. prof. dr. sc. Ivica Botički

SADRŽAJ

1. Uvod	1
2. Korištene tehnologije	3
2.1. ECMAScript 2015 JavaScript	3
2.1.1. Ključne riječi <i>let</i> i <i>const</i>	3
2.1.2. Klase i nasljeđivanje	5
2.1.3. <i>Promise</i> objekt	7
2.1.4. <i>Arrow</i> funkcije	8
2.2. React	10
2.2.1. Komponente	10
2.2.2. JSX	13
2.2.3. Podizanje stanja	15
2.3. Paket <i>create-react-app</i>	16
2.4. Paket <i>react-router</i>	17
2.5. Bootstrap i paket <i>react-bootstrap</i>	19
2.6. Fetch API	20
3. Specifikacija	22
3.1. Opis zadatka	22
3.2. Funkcionalni zahtjevi	23
3.2.1. Javni korisnik	23
3.2.2. Autorizirani korisnik	24
3.3. Nefunkcionalni zahtjevi	26
4. Implementacija	27
4.1. Arhitektura i dizajn	27
4.1.1. Raspodjela komponenti	28
4.1.2. Pomoćni moduli	31

4.2. Glavno sučelje	32
4.3. Upravljanje ponudama	34
4.4. Upravljanje zahtjevima	35
5. Zaključak	36
Literatura	38

1. Uvod

Potreba za razmjenom znanja, za razmjenom iskustava i ideja je jedna od ključnih karakteristika koja čini čovjeka čovjekom. Upravo je ta potreba, kao i sposobnost da je ispuni kvalitetnije od bilo kojeg drugog bića, snažno utjecala na evoluciju čovjeka i dovela nas na poziciji gdje se nalazimo danas. Na poziciji vrha hranidbenog lanca i neosporivo dominantnoj vrsti na Zemlji.

Nije iznenađenje kako mnoge drevne kulture i civilizacije imaju priče u kojima nekakvo božanstvo prenosi svoje znanje ljudima. Znanje uz pomoć kojeg ljudi postaju moćniji od svih ostalih smrtnih stvorenja. Jedna takva priča je mit o Prometeju, titanu iz grčke mitologije koji je ukrao vatru s Olimpa i darovao je ljudima kako bi omogućio razvoj civilizacije.

U modernom vremenu čovjek ima gotovo neiscrpan izvor znanja. Putem interneta može naći gotovo sve što poželi, no ipak nije sav sadržaj dospio u digitalan oblik. To je, između ostalog, slučaj s jako specifičnim sadržajima, kao što su fakultetske skripte čija je ideja da jednostavnim jezikom sažmu i prezentiraju akademsko znanje, te sadržajima koji su na jeziku s relativno malim brojem govornika, kao što je slučaj s hrvatskim jezikom.

Jeste li ikad pokušali naći podatke na hrvatskom jeziku o nekoj knjizi, a da ona nije dio popisa za lektiru? Jeste li ikad tražili funkcionalni primjerak videoigre izrađene u devedesetima? Jeste li se ikad mučili s nalaženjem neke knjige koju ste htjeli pročitati? Da je niste mogli naći u lokalnoj knjižnici ni u antikvarijatu, a u knjižarama je bila ili preskupa ili nije bila na policama, već bi se morala čekati više od mjesec dana da stigne putem narudžbe?

Vjerujem da smo se mnogi barem jednom našli u takvoj ili sličnoj situaciji.

Za sve te sadržaje, koje je teško naći na standardne načine, često postoji osoba koja im je vlasnik i voljna ga je posuditi. No problem je kako naći takvu osobu, odnosno kako osobi koja je voljna nešto posuditi omogućiti da to da do znanja drugima.

Aplikacija čije je korisničko sučelje tema ovog rada ima za cilj riješiti upravo taj problem.

Ideja aplikacije je omogućiti korisnicima u hrvatskom govornom području da ponude svoje knjige, skripte, stripove, CD-ove i sl. na posuđivanje. Odnosno, omogućiti drugim korisnicima da pronađu takav sadržaj. Isto kao što je Prometej nesebično ponudio vatru ljudima, tako korisnici aplikacije mogu drugim korisnicima ponuditi ono što imaju i time stvoriti *virtualnu knjižnicu* za razmjenu multimedijskog sadržaja.

U ovom radu je razmotren i riješen problem izgradnje jednostavnog i interaktivnog korisničkog sučelja takve knjižnice. Također su u radu istražene moderne tehnologije i alati za razvoj korisničkih sučelja. Detaljno o njima se može pročitati u drugom poglavlju. Ključna je biblioteka React opisana u potpoglavlju 2.2.

Poslije korištenih tehnologija, u trećem poglavlju, je detaljno opisan zadatak rada te su specificirani zahtjevi, dok se opis same implementacije zadatka nalazi se u četvrtom poglavlju. Nakon implementacije, u petom i zadnjem poglavlju, slijedi zaključak cjelokupnog rada.

2. Korištene tehnologije

2.1. ECMAScript 2015 JavaScript

Programsko rješenje je građeno primarno uz pomoć biblioteke React i pomoćnih paketa. Jezik pisanja je JavaScript po ECMAScript 2015 (izvornog naziva ECMAScript 6, tj. ES6) specifikaciji. Riječ je o standardu koji je uveo velike promjene u sam jezik. U svrhu razumijevanja koda programskog rješenja, u okviru rada će se prvo ukratko razmotriti neke ključne funkcionalnosti i dodatke koje ECMAScript 2015 uvodi u JavaScript.

2.1.1. Ključne riječi *let* i *const*

Standardna ključna riječ za definiranje varijabli u JavaScriptu - *var* ima neočekivano ponašanje unutar djelokruga bloka (engl. *block scope*). Promotrimo sljedeći primjer:

```
1 function varTest() {  
2   var x = 1;  
3   if (true) {  
4     var x = 2;  
5     console.log(x); // 2  
6   }  
7   console.log(x); // 2  
8 }
```

Isječak koda 2.1: Primjer djelokruga *var* varijable

Varijabla definirana u *if*-bloku je ista kao ona van njega, odnosno blok nije imao utjecaj na djelokrug varijable onako kako bismo očekivali u ostalim jezicima C-ovske sintakse.

Ovaj problem može se riješiti korištenjem ključne riječi *let* umjesto *var*[5].


```

1 function varTest() {
2   let x = 1;
3   if (true) {
4     let x = 2;
5     console.log(x); // 2
6   }
7   console.log(x); // 1
8 }

```

Isječak koda 2.2: Primjer djelokruga *let* varijable

Sada se dobije očekivano ponašanje. Varijabla unutar *if*-bloka je nova varijabla koja je unutar svog djelokruga zasjenila varijablu u vanjskom bloku.

Također, u slučaju da se u ovom kodu (isječak koda 2.2) pokušalo ponovno deklarirati varijablu *x* unutar istog bloka, bila bi bačena sintaksna greška. Varijabla deklarirana ključnom riječi *var* ne izaziva sintaksne greške, kod nje je dopušteno redefinicirati varijable u istom bloku. Takvo ponašanje može prouzročiti neprimjetne greške pri pisanju koda koje se teško pronalaze kasnije.

Još jedna razlika između *var* i *let* je da se u globalnom djelokrugu varijable definirane s *let* ne pridružuju globalnom objektu.

```

1 var x = 'global';
2 let y = 'global';
3 console.log(this.x); // "global"
4 console.log(this.y); // undefined

```

Isječak koda 2.3: Odnos *var* i *let* s globalnim objektom

I u ovom slučaju je ponašanje *let* varijable logičnije. Samo zato što se varijabla koristi unutar neke funkcije, odnosno objekta, ne znači da je cilj tu varijablu pridružiti kao svojstvo objekta u kojem se nalazi.

Budući da se *let* varijable ponašaju više u skladu s očekivanjem, pogotovo ako imamo podlogu u programskom jeziku koji spada u obitelj jezika C (engl. *C-family programming languages*), i da bacaju iznimke u slučaju grešaka, u novijem standardu se preporučuje izbjegavanje korištenja ključne riječi *var* u svim situacijama.

Kako u starim verzijama nije bilo drugog načina deklariranja varijable dalje od ključne riječi *var* i globalnih varijabli, programer je morao sam paziti da ne promjeni vrijednost varijabli koje si je odredio kao konstante. U novom standardu je taj problem riješen ključnom riječi *const*[3].

```
1 const PI = 3.14
2 PI = 3    // error
```

Isječak koda 2.4: Deklariranje konstante

Po pitanju djelokruga, ponašanje konstanti je identično ponašanju *let* varijable.

2.1.2. Klase i nasljeđivanje

Glavna novost u ECMAScript 2015 je uvođenje klasa. Prethodno se nasljeđivanje u JavaScriptu ostvarivalo kreiranjem objekata iz postojećih objekata te dodavanjem novih metoda i atributa novostvorenom objektu. Svaki objekt u JavaScriptu ima skriveno svojstvo prototip (engl. *prototype*) koje sadrži poveznicu do nekog drugog objekta, putem kojeg možemo ostvariti nasljeđivanje[6]. Takva mehanika je puno moćnija od korištenja klasa, odnosno nudi puno više fleksibilnosti, no istovremeno uzrokuje puno nejasniju strukturu koda.

Klase u novom standardu su ostvarene kao poseban oblik funkcija[2]. Oba objekta se deklariraju i njima se rukuje na identičan način, samo za različite svrhe. Pri prevođenju JavaScripta pisanog u novom standardu u stariji (zbog kompatibilnosti sa starijim preglednicima) klase se prevode u funkcije. (Primjer: isječak koda 2.6 - izrađen putem prevoditelja Babel¹).

```
1 class Rectangle {
2   constructor(height, width) {
3     this.height = height;
4     this.width = width;
5   }
6 }
```

Isječak koda 2.5: Primjer klase

¹<https://babeljs.io>

```

1 function _classCallCheck(instance, Constructor) {
2   if (!(instance instanceof Constructor)) {
3     throw new TypeError("Cannot call a class as a function");
4   }
5 }
6
7 var Rectangle = function Rectangle(height, width) {
8   _classCallCheck(this, Rectangle);
9
10  this.height = height;
11  this.width = width;
12 };

```

Isječak koda 2.6: Isječak koda 2.5 preveden u stari standard

Nasljeđivanje klasa postiže se sintaksom sličnom programskom jeziku Java. Ključna riječ kojom se definira nasljeđivanje je *extends*, dok se natklasa poziva s ključnom riječi *super*.

```

1 class Cat {
2   constructor(name) {
3     this.name = name;
4   }
5   speak() {
6     console.log(this.name + ' makes a noise. ');
7   }
8 }
9
10 class Lion extends Cat {
11   speak() {
12     super.speak();
13     console.log(this.name + ' roars. ');
14   }
15 }
16
17 let l = new Lion('Simba');
18 l.speak();
19 // Simba makes a noise.
20 // Simba roars.

```

Isječak koda 2.7: Primjer nasljeđivanja

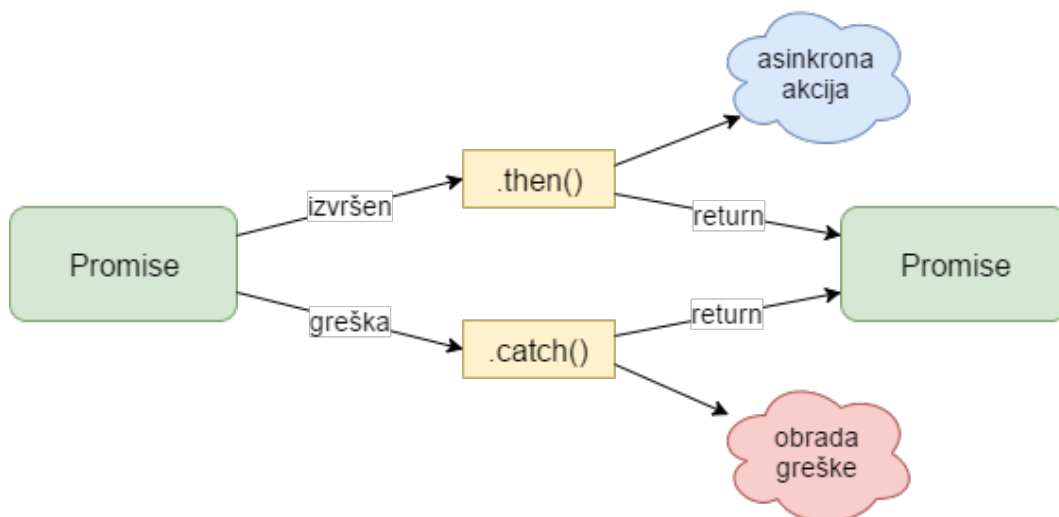
2.1.3. *Promise* objekt

Obećanje (engl. *Promise*) je poseban objekt koji predstavlja izvršavanje asinkrone operacije. Ideja je da se neka asinkrona operacija (npr. dohvat podataka s poslužitelja) omota u obećanje kako bi se definiralo što će se izvršiti u slučaju uspjeha ili neuspjeha. Operacija koja se izvršava nakon obećanja može ponovno biti obećanje. To nizanje obećanja (engl. *chaining*), koje se ostvaruje putem funkcije *then*, zgodno je rješenje kada logika zahtjeva višestruko ugnježđivanje funkcija povratnog poziva (engl. *callback function*)[8]. Primjer toga je u sljedećem isječku koda (2.8)

```
1 // callback pyramid of doom
2 doSomething(function(result) {
3   doSomethingElse(result, function(newResult) {
4     doThirdThing(newResult, function(finalResult) {
5       console.log('Got the final result: ' + finalResult);
6     }, failureCallback);
7   }, failureCallback);
8 }, failureCallback);
9
10 // using promises
11 doSomething()
12 .then(function(result) {
13   return doSomethingElse(result);
14 })
15 .then(function(newResult) {
16   return doThirdThing(newResult);
17 })
18 .then(function(finalResult) {
19   console.log('Got the final result: ' + finalResult);
20 })
21 .catch(failureCallback);
```

Isječak koda 2.8: Korištenje obećanja umjesto ugnježđivanja funkcija povratnog poziva

Životni ciklus objekta obećanja možemo pojednostavljeno opisati dijagramom na slici 2.1. Kada se objekt stvori, on je u stanju iščekivanja (engl. *pending*). Ovisno o uspješnosti operacije prelazi u stanja *ispunjen* (engl. *fulfilled*) ili *odbačen* (engl. *rejected*). Iz tih stanja može vratiti neku povratnu vrijednost koja se onda omata kao novo obećanje.



Slika 2.1: Životni ciklus *promise* objekta

2.1.4. *Arrow* funkcije

Novim standardom je u JavaScript dodano mnogo *sintaksnog šećera* (engl. *syntax sugars*), jedna od kojih su i funkcije napisane sa strelicom (engl. *arrow functions*). Riječ je o kraćem načinu za zapisivati anonimne funkcije po uzoru na lambda račun.

```

1 doSomething()
2 .then(result => doSomethingElse(result))
3 .then(newResult => doThirdThing(newResult))
4 .then(finalResult => {
5   console.log('Got the final result: ' + finalResult);
6 })
7 .catch(failureCallback);

```

Isječak koda 2.9: Kod iz isječka 2.8 napisan pomoću *arrow* funkcija

Na prvi pogled ništa novo, samo kraći zapis postojeće funkcionalnosti, no *arrow* funkcije nisu u potpunosti ekvivalentne već postojećim anonimnim funkcijama u JavaScriptu. Unutar tijela starih anonimnih funkcija, referenca *this* se odnosila na okolinu, tj. objekt kojem je funkcija pridružena, a ne gdje je definirana. To je zahtjevalo dodatnu pažnju programera pri baratanju tom ključnom riječi. *Arrow* funkcije ne pate od tog problema, one ne mijenjaju referencu *this* pri stvaranju novog objekta funkcije, već pozivaju *this* u kojem su definirane[1].

```

1 function Object1(){
2     this.log = function() {
3         console.log(this);
4     };
5 }
6 function Object2(message){
7     this.log = () => console.log(this);
8 }
9
10 var o1 = new Object1();
11 o1.log() // Object1
12 var method1 = o1.log;
13 method1(); // Window
14
15 var o2 = new Object2();
16 o2.log() // Object2
17 var method2 = o2.log;
18 method2(); // Object2

```

Isječak koda 2.10: Ponašanje ključne riječi *this* u obe verzije anonimnih funkcija

2.2. React

React je JavaScript biblioteka otvorenog koda namijenjena brzom i jednostavnoj izgradnji korisničkog sučelja. Razvijena je od strane Facebooka 2013. godine i zadnja stabilna verzija u trenutku pisanja ovog rada je 15.5.4.[20]. U izvorima se biblioteka također može naći pod nazivima *React.js* i *ReactJS*, dok je React Native radni okvir za razvoj mobilnih aplikacija u JavaScriptu na sličan način kako se web aplikacije grade u Reactu.

Bitno je za naglasiti da, iako je React biblioteka (engl. *library*), nije u potpunosti netočno zvati je radnim okvirom (engl. *framework*). Tijekom vremena nagomilalo se mnoštvo biblioteka i radnih okvira za korištenje sa samim Reactom, da se pod izrazom „React“ često podrazumijeva i njihovo korištenje, što daje dojam cjelokupnog radnog okvira. No, po svojoj definiciji je biblioteka[18].

Reactom se grade jednostranične (engl. *single page*) web aplikacije. Producerska verzija React stranice se često sastoji od jedne male HTML datoteke (reda veličine dvadesetak linija koda) i JavaScript datoteke od nekoliko desetaka tisuća linija koda². Razlog tomu je što je sav sadržaj stranice ugrađen u JavaScript kako bi se mogao dinamički mijenjati.

React koristi virtualni DOM (engl. *virtual DOM*³). Virtualni DOM je apstrakcija standardnog HTML DOM-a koja se koristi kako bi se izbjegle skupe operacije dodavanja i brisanja elemenata u DOM-u. Efikasnim upravljanjem virtualnim DOM-om, elementi se crtaju u stvarni DOM samo kada je to potrebno, odnosno crtaju se samo elementi koji su se promijenili. Primjerice, nepotrebno je da se svaki put iznova crta (odnosno iščitava iz HTML dokumenta) navigacija, ako se samo mijenja sadržaj ispod nje.

2.2.1. Komponente

Filozofija Reacta je rastaviti korisničko sučelje na komponente kao izolirane cjeline koje se mogu ponovno upotrijebiti[12]. Instance komponenti se, slično HTML DOM elementima, organiziraju u stablastu strukturu. Za razliku od DOM elemenata, komponente imaju vlastito stanje i logiku. React komponente nasljeđuju klasu *React.Component* te obavezno implementiraju *render* metodu koja vraća React element, odnosno stablo

²Tijekom razvoja je kod u više datoteka koje se zapakiraju u jednu pri gradnji producerske verzije.

³krat. *Document Object Model*

React elemenata[14]. React elementi su nepromjenjivi objekti čije je stvaranje jeftina operacija. Crtaju se kao elementi DOM-a po potrebi[17].

```
1 class HelloWorld extends React.Component {  
2   render() {  
3     return <h1>Hello world!</h1>;  
4   }  
5 }
```

Isječak koda 2.11: Primjer komponente

Osim metode *render*, komponenta može implementirati vlastite metode i nadjačati metode nadrazreda (*React.Component*), tj. metode životnog ciklusa (engl. *lifecycle methods*). Metode životnog ciklusa dijelimo na tri skupine:

1. *Mounting* – Metode koje se pozivaju kada se stvara instanca komponente i dodaje DOM-u.
 - *constructor()*
 - *componentWillMount()*
 - *render()*
 - *componentDidMount()*
2. *Updating* – Metode koje se pozivaju kada se dogodi promjena stanja komponente i mora se ponovno crtati u DOM-u.
 - *componentWillReceiveProps()*
 - *shouldComponentUpdate()*
 - *componentWillUpdate()*
 - *render()*
 - *componentDidUpdate()*
3. *Unmounting* – Metode koje se pozivaju kada se komponenta uklanja iz DOM-a.
 - *componentWillUnmount()*

Metode se pozivaju redosljedom kojim su napisane, a uz njih se još nasljeđuju metode *setState* i *forceUpdate()*. One nisu dio životnog ciklusa, već se pozivaju nad objektom.

Svaka od metoda životnog ciklusa ima različito ponašanje, stoga postoje dobre prakse za što se koja od njih koristi. Metoda *constructor* se koristi za inicijaliziranje stanja, dok se logika promjene stanja preporučuje pisati u *componentDidMount*⁴. Razlog tomu je što promjena stanja u *componentDidMount* izaziva ponovno crtanje komponente u DOM-u, odnosno poziv metode *render*. Stanje se mijenja pozivanjem metode *setState* nad objektom komponente („*this.setState()*“). Ta metoda je asinkrona, stoga promjena stanja ne blokira daljnje pozive metoda u životnom ciklusu. Ponovno crtanje može se izbjeći ako se metodu *shouldComponentUpdate* postavi da vraća *false*. U tom slučaju ponovno crtanje komponente može se postići samo pozivanjem metode *force update*.

Uz stanje postoji još jedno ugrađeno svojstvo komponente, a to je *props*[13]. Ono se koristi kao objekt putem kojeg se predaju argumenti iz više komponente u nižu. Primjer takvog korištenja je komponenta koja sadrži listu drugih komponenti, a treba odrediti nazive i ključeve potkomponenti. U tom slučaju se, pri stvaranju potkomponenti kojima se nadjačava konstruktor, mora paziti da se koristi oblik konstruktora kakav je prikazan u isječku koda 2.12. Ako se ne definira vlastiti konstruktor, taj oblik konstruktora nasljeđuje se iz natklase⁵.

```
1 class Welcome extends React.Component {  
2   constructor(props) {  
3     super(props)  
4   }  
5  
6   render() {  
7     return <h1>Hello, {this.props.name}</h1>;  
8   }  
9 }
```

Isječak koda 2.12: Primjer komponente s *props*

Svojstva se pridodaju objektu *props* iz nadkomponente na način kako je prikazano u isječku koda 2.13. Pojašnjenje sintakse može se pronaći u pododjeljku 2.2.2.

⁴Dohvat podataka sa servera se često vrši u *componentDidMount*. Ti se podaci potom spremaju u stanje kako bi se izazvalo ponovno pozivanje metode *render*.

⁵U isječku koda 2.12 je pisanje konstruktora nepotrebno.

```

1 class Welcome extends React.Component {
2   render() {
3     return <h1>Hello, {this.props.name}</h1>;
4   }
5 }
6
7 class App extends React.Component {
8   render () {
9     return (
10      <div>
11        <Welcome name="Ed" />
12        <Welcome name="Edd" />
13        <Welcome name="Eddy" />
14      </div>
15    );
16  }
17 }

```

Isječak koda 2.13: Primjer korištenja *props*

2.2.2. JSX

JSX je sintaksno proširenje JavaScripta koji omogućuje jednostavno stvaranje React elemenata u formatu sličnom HTML-u. Datoteke koje koriste ovo sintaksno proširenje se često pišu s ekstenzijom „.jsx“, no pisanje te ekstenzije nije nužno⁶.

U isječku koda 2.13 *render* metoda komponente *App* vraća tri instance komponente *Welcome* ugnježđene u *div*⁷. Argumenti koji se predaju kao svojstva objekta *props* pri stvaranju React elementa se pišu kao atributi DOM elemenata. Budući da se u ovoj sintaksi React elementi mogu kombinirati s HTML elementima, pravilo je da se React elementi pišu velikim početnim slovom[15].

Ovakav format daje čitku strukturu kako će se React elementi posložiti u DOM-u. Primjerice, poziv metode *render* iz klase *App* bi rezultirao stvaranjem HTML koda kao u isječku 2.14.

⁶Za TypeScript datoteke: „.tsx“

⁷U metodi *render* sve korištene komponente uvijek moraju biti ugnježđene u jednu vršnu komponentu.

```

1 <div>
2   <h1>Hello, Ed</h1>
3   <h1>Hello, Edd</h1>
4   <h1>Hello, Eddy</h1>
5 </div>

```

Isječak koda 2.14: Elementi isječka koda 2.13 prikazani kao renderirani DOM elementi

Kako su u JavaScriptu klase samo poseban oblik funkcija, JSX sintaksa omogućava da se jednostavne elemente stvori pomoću anonimnih funkcija. Funkcije mogu primiti proizvoljan broj argumenata, a vraćaju React element. Elementi se imenuju tako da te funkcije sprememo u varijable, odnosno konstante, koje se potom koriste isto kao i komponente.

```

1 class App extends React.Component {
2   render () {
3     const Welcome = (props) => <h1>Hello, {props.name}</h1>;
4
5     return (
6       <div>
7         <Welcome name="Ed" />
8         <Welcome name="Edd" />
9         <Welcome name="Eddy" />
10      </div>
11    );
12  }
13 }

```

Isječak koda 2.15: Korištenje anonimne funkcije za stvaranje elementa

U svim primjerima dosad se vrijednosti dohvaćalo tako da ih se zapisivalo unutar vitičastih zagrada. Ta sintaksa vitičastih zagrada se može iskoristiti i za istinske vrijednosti, pomoću kojih se može *short-circuit* evaluacijom logičkih izraza ostvariti crtanje elementa ovisno o nekom uvjetu.

```

1 class App extends React.Component {
2   render () {
3     const Message = (props) => <h1>Ja sam {props.msg}</h1>;
4     let f = true;
5

```

```

6      return (
7        <div>
8          {f && <Message msg="vidljiv" /> }
9          {!f && <Message msg="nevidljiv" /> }
10         {!f || <Message msg="vidljiv" /> }
11         {f || <Message msg="nevidljiv" /> }
12       </div>
13     );
14   }
15 }

```

Isječak koda 2.16: Uvjetno crtanje elementa

2.2.3. Podizanje stanja

U interaktivnoj aplikaciji često je potrebno na promjenu jednog elementa mijenjati drugi. Primjerice, na unos teksta u okvir za pretraživanje filtrirati retke tablice s obzirom na neku od vrijednosti. U Reactu se ta funkcionalnost postiže podizanjem stanja[16].

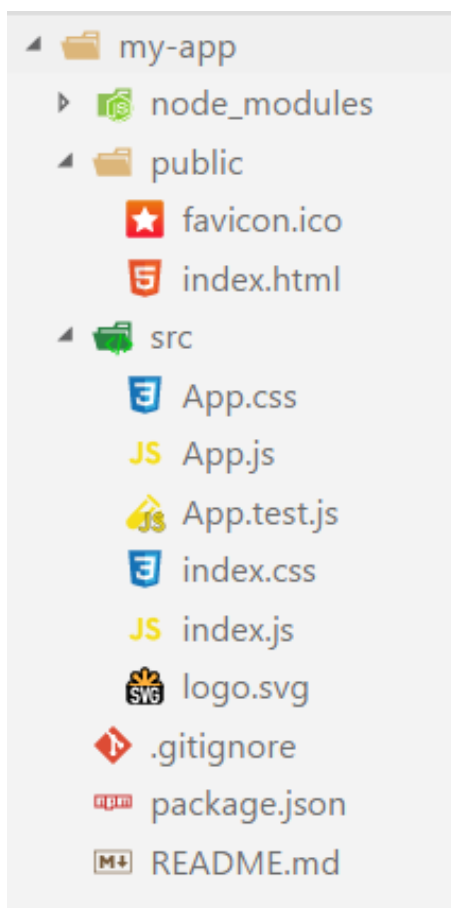
Na istom tom primjeru, ako su komponente okvira za unos teksta i tablice ugniježdene u zajedničkoj roditeljskoj komponenti, vrijednost teksta koji se unosi i lista s podacima za retke tablice bi se definirali kao atributi stanja te roditeljske komponente. Ugniježdene komponente bi putem *props* primile te atribute iz roditeljske komponente. Pri tome, komponenta koja uzrokuje promjenu stanja, što je u ovom slučaju komponenta u koju se unosi tekst, mora primiti i referencu na funkciju koju će pozvati kad se unos promijeni. Ta funkcija se nalazi u roditeljskoj komponenti i postavlja stanje na novu vrijednost, što uzrokuje ponovno crtanje ugniježđenih elemenata, odnosno komponenti. Na taj način se komponenta s tablicom može svaki put crtati s filtriranom listom redaka, s obzirom na uneseni tekst.

Reactova hijerarhija komponenti prati filozofiju „Podaci dolje, akcije gore“ (engl. *Data Down, Actions Up*). U opisanom primjeru su podaci vrijednosti koje se šalju elementima, dok je akcija poziv funkcije roditeljske komponente koja mijenja stanje.

2.3. Paket *create-react-app*

Npm⁸ paket *create-react-app* omogućuje stvaranje podešene React aplikacije, u svrhu bržeg i lakšeg razvoja nove aplikacije. Aplikacija stvorena ovim paketom ima podešen prevoditelj *Babel*⁹, upravitelj modulima *webpack*¹⁰ i mnoge druge dodatke koji su često potrebni u razvoju React aplikacija[11]. Podešene su također skripte za pokretanje, testiranje i izradu produkcijske verzije.

Sve te postavke su skrivene, a programer nakon stvaranja aplikacije vidi strukturu kao u slici 2.2¹¹. U slučaju potrebe za promjenom predefiniranih postavki pokreće se skripta „*npm run eject*“ koja nepovratno otkriva sve konfiguracijske datoteke.



Slika 2.2: Struktura aplikacije stvorene pomoću *create-react-app*

⁸Upravitelj paketa za JavaScript, primarno korišten uz Node.js.

⁹Prevodi novije JavaScript funkcionalnosti u stari standard, <http://babeljs.io>

¹⁰Sažima sve module i pakete o kojima ovise u statičke resurse, <https://webpack.github.io>

¹¹Mapa novostvorene aplikacije na disku zauzima prostor od otprilike 100 MB.

2.4. Paket *react-router*

Pri gradnji jednostraničnih web aplikacija, često nije u cilju da se korisniku doimaju kao takve. Korisnik želi imati mogućnosti da putem poveznice izravno dođe do nekog dijela aplikacije ili da se vrati na prethodno stanje aplikacije klikom *Nazad* u pregledniku, kao što to može s klasičnim web stranicama¹².

Takvo ponašanjem se unutar React aplikacije može postići koristeći API paketa *react-router*. Putem njega dobije se pristup promjenjivom objektu povijesti (engl. *history*), kojim se služi kao s povijesti preglednika, te gotovim navigacijskim komponentama za upravljanje s tom povijesti. Jedini uvjet za korištenje tih mogućnosti je da su komponente koje ih koriste ugniježdene u komponenti *Router*[19].

```
1 import React, { Component } from 'react';
2 import {
3   BrowserRouter as Router, Route, Link
4 } from 'react-router-dom';
5
6 class App extends Component {
7   render() {
8     const Home = () => ( <div> <h2>Home</h2> </div> );
9     const About = () => ( <div> <h2>About</h2> </div> );
10
11     return (
12       <Router>
13         <div>
14           <ul>
15             <li><Link to="/">Home</Link></li>
16             <li><Link to="/about">About</Link></li>
17           </ul>
18
19           <Route exact path="/" component={Home} />
20           <Route path="/about" component={About} />
21         </div>
22       </Router>
23     );
24   }
25 }
```

Isječak koda 2.17: Primjer korištenja *react-router*

¹²Ovdje se izraz *web stranica* odnosi na engl. *website*, odnosno kolekciju stranica (engl. *web page*).

U isječku koda 2.17 je primjer aplikacije koja bi se korisniku prikazala kao web stranica s dvije poveznice na čiji se klik se mijenja putanja aplikacije i tekst ispod poveznica.

BrowserRouter komponenta koristi HTML5 history API¹³ za upravljanje s povijesti preglednika. U tu komponentu su ugniježdene *Route* komponente koje služe za određivanje koje će se komponente crtati na kojim rutama. Bitno je imati na umu da je dovoljan uvjet za crtanje komponente u *component* atributu taj da početak putanje odgovara atributu *path*. Za postizanje ponašanja da se vrijednosti moraju u potpunosti podudarati koristi se ključna riječ *exact*. U slučaju da u prethodno navedenom primjeru nije korišten *exact*, na putanji „/about“ bi se crtala i komponenta *Home* uz komponentu *About*.

Route komponente se mogu dodatno ugnijezditi i u komponentu *Switch*. *Switch* komponenta definira logiku da se crta samo komponenta prvog *Route* koji se podudara s putanjom. Korištenjem ove logike može se definirati i posljednja *Route* komponenta za pokrivanje slučaja neočekivanih putanja, kao što je prikazano u isječku koda 2.18. U tom primjeru se također koristi *exact*, jer bi se u protivnom na svim rutama koje počinju s „/“ odabrala prva komponenta.

```
1 <Router>
2   <Switch>
3     <Route exact path="/" component={Home} />
4     <Route path="/about" component={About} />
5     <Route component={NoMatch} />
6   </Switch>
7 </Router>
```

Isječak koda 2.18: Primjer korištenja *Switch*

Od daljnjih komponenti paketa jako je bitna i komponenta *Redirect* koja pri svom crtanju nadjačava trenutnu lokaciju u objektu povijesti i preusmjerava korisnika na novu putanju (određena atributom *to*). Isto ponašanje se može postići pozivom metode *replace* nad objektom *history*.

Objektu *history* se može pristupiti u komponentama koje su „stvorene s *routerom*“. Odnosno u komponentama koje su predane kao vrijednosti *component* atributa od *Route* ili instancirane s metodom *withRouter*.

¹³https://developer.mozilla.org/en-US/docs/Web/API/History_API

2.5. Bootstrap i paket *react-bootstrap*

Bootstrap je jednostavan radni okvir za gradnju web stranica prilagodljivih za raznolike uređaje. Nudi velik skup predefiniраних HTML komponenti i standardiziraniх CSS klasa koje se mogu koristiti za brzu gradnju čestih elemenata stranice kao što su izbornici, obrasci (engl. *forms*), okviri za slike i sl. Također ima i definiran sustav mrežne raspodjele elemenata, koji osigurava prilagodljivu veličinu i raspodjelu sadržaja s obzirom na veličinu zaslona korisničkog uređaja.

Standardizirane CSS klase omogućavaju laganu izmjenu dizajna stranice ako se koristilo samo zadane Bootstrap klase za stiliziranje HTML elemenata. Dovoljno je samo zamijeniti CSS datoteku, odnosno temu s novom koja definira stil tih istih klasa. Također se time osobi koja gradi web stranicu pruža mogućnost da različite projekte gradi istim elementima, kojima zna ponašanje.

Budući da Bootstrap postoji od 2011.[10], na internetu postoji mnoštvo gotovih Bootstrap tema koje se lako mogu ugraditi u vlastitu aplikaciju.

Paket *react-bootstrap* nudi osnovne Bootstrap elemente, odnosno klase zapakirane kao React komponente. U trenutku pisanja ovog rada paket je još u razvoju i nema stabilnu verziju, no implementirani su neki od ključnih elementa koje pomažu u razvoju aplikacije (primjerice navigacijska traka)[9].

2.6. Fetch API

Fetch API je JavaScript sučelje za dohvaćanje resursa koji u odnosu na tradicionalni XMLHttpRequest¹⁴, nudi puno veći i moćniji skup mogućnosti.

Fetch omogućuje generičku definiciju objekata *Request* i *Response*, čime znatno proširuje područje svoje uporabe jer postiže kompatibilnost s drugim tehnologijama kao što su Service Worker API¹⁵, Cache¹⁶ i sl.[4]. Osim toga, nudi jedinstveno mjesto za definiciju koncepata kao što su CORS¹⁷ i HTTP ekstenzije.

Za dohvaćanje resursa i slanje zahtjeva koristi se metoda *GlobalFetch.fetch* koja je implementirana u mnogim sučeljima poput *Window* i *WorkerGlobalScope*, što ju čini raspoloživom u većini slučajeva (predstavlja globalnu metodu). Osim što je raspoloživa, nudi jednostavan i logičan način za asinkrono dohvaćanje resursa kroz mrežu.

Metoda *fetch* kao obavezni parametar prima putanju do resursa koji se treba dohvatiti, a vraća obećanje¹⁸ u obliku objekta tipa *Response*. Proizvoljni argument je *init* objekt koji predstavlja dodatne postavke za metodu *fetch*[7].

Ako se *fetch* usporedi sa svojim raširenim srodnikom *jQuery.ajax*, mogu se pronaći neke ključne razlike. Kao prvu bitnu razliku treba spomenuti to da će se *fetch* normalno izvršavati i za HTTP statuse koji inače predstavljaju grešku (npr. HTTP status 404 ili 500), samo će postaviti vrijednost svog OK statusa na false. Jedina situacija u kojoj će odbiti mrežu zbog neuspjeha je ako nešto spriječi zahtjev da se izvrši u potpunosti.

Osim toga, *fetch*, u svome pretpostavljenom ponašanju, ne šalje i ne prima kolačiće (engl. *cookies*)¹⁹ s poslužitelja što može prouzročiti zahtjeve bez autentifikacije u slučaju da se stranica oslanja na kolačiće. Za slanje kolačića, nužno je slati zaglavlje *credentials*.

¹⁴<https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest>

¹⁵https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API

¹⁶<https://developer.mozilla.org/en-US/docs/Web/API/Cache>

¹⁷engl. *Cross-Origin Resource Sharing*, https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS

¹⁸Vidi pododjeljak 2.1.3.

¹⁹<https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies>

```

1 fetch('slika.jpg').then(response => response.json())
2   .then(response => {
3     let objectURL = URL.createObjectURL(response);
4     myImage.src = objectURL;
5   });

```

Isječak koda 2.19: Primjer dohвата resursa s Fetch API

Isječak koda 2.19 pokazuje primjer korištenja metode *fetch* za dohvaćanje resursa, odnosno primjer HTTP GET zahtjeva (engl. *HTTP GET request*). Metoda prima putanju do željenog resursa i vraća obećanje koje sadrži objekt *Response* koji predstavlja HTTP odgovor (engl. *response*).

U primjeru se zatim primljeni objekt parsira u JSON objekt pozivanjem metode *json*. No, ovisno o prirodi resursa, mogla se primijeniti bilo koja od navedenih metoda: *json*, *text*, *blob*, *formData*, *arrayBuffer*, *redirect* ili *clone*[23]. Navedene metode mogu se izvršavati na objektima tipa *Request* i *Response*, što znači da se mogu koristiti i pri slanju zahtjeva. To čini korištenje ne-tekstualnih podataka puno jednostavnijim nego što je to bilo s XMLHttpRequest.

Na isječku koda 2.20 prikazan je HTTP POST zahtjev (engl. *HTTP POST request*) izveden korištenjem metode *fetch*. Kao drugi parametar šalje se *init* objekt u kojemu se tip metode postavlja na POST²⁰. Osim metode, po potrebi mogu se postaviti dodatni atributi, kao što su npr. *mode* i *headers* atributi.

```

1 fetch('/api/postUrl', {
2   method: 'POST',
3   mode: 'cors',
4   headers: new Headers({
5     'Content-Type': 'text/plain'
6   })
7 }).then(function() {
8   /* handle response */
9 });

```

Isječak koda 2.20: Primjer slanja resursa s Fetch API

²⁰Podrazumijevana metoda je GET pa zato nije naveden tip metode u primjeru u isječku koda 2.19.

3. Specifikacija

3.1. Opis zadatka

Knjižnica za razmjenu multimedijskog sadržaja je web aplikacija, odnosno sustav koji korisnicima omogućuje da s lakoćom pronalaze ili stavljaju na ponudu fizičke kopije multimedijskog sadržaja. Riječ je o predmetima poput knjiga, skripti za fakultete, CD-ova ili DVD-ova kojih se vlasnici ne žele riješiti, no rado bi na legalan i jednostavan način podijelili s drugima. Ovaj sustav tim osobama daje način da ponude svoje predmete na posuđivanje poput oglasa, te pronađu ljude koji su zainteresirani za taj sadržaj. Također nudi i osobama s druge strane jednostavan pronalazak sadržaja kojeg traže i način, odnosno informacije kako kontaktirati vlasnika.

Kako bi takva *knjižnica* bila uspješno ostvarena, odnosno upotrebljiva, nužno je da ima funkcionalno, jednostavno i interaktivno korisničko sučelje. Sučelje mora ponuditi funkcionalnosti pronalaska sadržaja, ponude sadržaja te način za rezervaciju sadržaja i upravljanje rezervacijom. Nužno je pritom moći pratiti stanje sadržaja u ponudi (ako je traženi sadržaj već posuđen od strane neke druge osobe) i održavati poveznicu s vlasnicima sadržaja.

Budući da je fokus na jednostavnosti aplikacije, sučelje je fokusirano na prikaz potrebnih informacija i omogućavanju glavne funkcionalnosti što je sistem rezervacije sadržaja. Način na koji korisnici obavljaju samu razmjenu predmeta nije u osnovnoj ideji aplikacije, tj. ostavljena je korisnicima na izbor. Sučelje samo nudi informacije da mogu stupiti u kontakt.

Implementacija sustava za komunikaciju unutar aplikacije, odnosno korisničkog sučelja je nadogradnja ostavljena za budući razvoj. Daljnje moguće nadogradnje su dijelovi sučelja koje bi prikazivale dodatne informacije. Primjer toga je prikaz povijesti posuđivanja korisnika ili sistem ocjenjivanja korisnika kako bi oni mogli ojačati svoju vjerodostojnost pred korisnicima od kojih žele posuditi sadržaj. Još jedna manja

nadogradnja, koja bi bila nužna s povećanjem broja korisnika aplikacije, je napredniji sustav pretraživanja, odnosno filtracije sadržaja.

Korisničko sučelje mora biti ostvareno modularno i po načelima dobrog oblikovanja kako bi sve buduće nadogradnje bile lako ostvarive.

3.2. Funkcionalni zahtjevi

Sustav u svojoj pozadini nema definiranu podjelu korisnika, odnosno nema hijerarhije ni različitih uloga, već je svaki korisnik zapisan u bazi samo *korisnik*. No, s perspektive funkcionalnih zahtjeva sučelja aplikacije moramo napraviti razliku između dvije vrste korisnika – *javni* i *autorizirani*.

Javni korisnik je bilo koja osoba koja je ušla u aplikaciju. Odnosno riječ je o korisniku koji nije prijavljen, tj. čije informacije nisu zapisane u pozadini aplikacije. Takav korisnik ima samo pristup javnim dijelovima aplikacije, odnosno može samo pregledavati ponudu sadržaja.

Autorizirani korisnik je korisnik koji je prijavljen u sustavu. Tijekom korištenja aplikacije, aplikacija prati tko je korisnik kako bi mogla osigurati uspješno izvođenje akcija za koje je potreban taj podatak. Ovaj korisnik je *pravi* korisnik aplikacije i ima pristup svim funkcionalnostima.

3.2.1. Javni korisnik

Zahtjevi javnog korisnika su sljedeći:

1. Pregled popisa ponuda

Korisnik može pregledati popis ponuda dostupnih u sustavu.

2. Pregled pojedinačne ponude

Korisnik može dohvatiti informacije o specifičnoj ponudi dostupnoj u sustavu.

3. Pretraživanje ponuda

Korisnik može pretraživati ponude koristeći tražilicu za filtrirat ponude s obzirom na zadane parametre.

4. Registracija

Korisnik se može registrirati (engl. *register*) u sustav.

5. Prijava

Korisnik se može prijaviti u sustav, tj. ulogirati (engl. *log in*) u aplikaciju.

3.2.2. Autorizirani korisnik

Prva tri zahtjeva autoriziranog korisnika su identična kao prva tri zahtjeva javnog korisnika, dok su ostali:

4. Pregledavanje profila

Korisnik može pregledati, odnosno dohvatiti informacije o vlastitom profilu i profilima drugih korisnika u sustavu.

5. Uređivanje profila

Korisnik može urediti informacije vlastitog profila. Ova opcija je proširenje nad pregledavanjem vlastitog profila, odnosno vidljiva je kad korisnik pregledava vlastiti profil.

6. Unos ponude

Korisnik može unijeti vlastitu ponudu u sustav.

7. Brisanje ponude

Korisnik može obrisati vlastitu ponudu iz sustava.

8. Rezervacija sadržaja

Korisnik može podnijeti zahtjev nad ponudom drugog korisnika da rezervira sadržaj koji je ponuđen.

9. Brisanje zahtjeva za rezervacijom

Korisnik koji je predao zahtjev za rezervacijom može obrisati taj zahtjev, ako korisnik kojem je zahtjev poslan nije prethodno potvrdio ili odbio zahtjev.

10. Potvrda rezervacije

Korisnik koji je zaprimio zahtjev za rezervacijom sadržaja od drugog korisnika ju može prihvatiti. Ova opcija uključuje brisanje zahtjeva za rezervacijom.

11. Odbijanje rezervacije

Korisnik koji je zaprimio zahtjev za rezervacijom sadržaja od drugog korisnika ju može odbiti. Ova opcija uključuje brisanje zahtjeva za rezervacijom.



Slika 3.1: Dijagram obrazaca uporabe za funkcionalne zahtjeve korisnika

3.3. Nefunkcionalni zahtjevi

Uz navedene funkcionalne zahtjeve korisnika, aplikacija mora također ispuniti sljedeće zahtjeve:

- nepredviđena akcija korisnika ne smije uzrokovati prekid rada aplikacije
- aplikacija mora raditi u stvarnom vremenu, pri čemu broj korisnika ne smije utjecati na performanse
- aplikacija mora biti funkcionalna na svim modernim web preglednicima sa značajnom zastupljenosti na tržištu
- funkcionalnosti aplikacije za čije je izvršavanje potrebna prijava ne smiju biti dostupne bez prijave
- navigacija do glavnih dijelova aplikacije mora biti dostupna u svim sučeljima
- do glavnih se dijelova aplikacije mora moći pristupiti putem povijesti preglednika
- sučelje aplikacije mora biti prilagodljivo za različite veličine ekrana
- sučelje ne smije sadržavati autorski sadržaj bez dopuštenja autora
- sučelje mora biti napisano na hrvatskom jeziku i podržavati hrvatske znakove
- komponente sučelja aplikacije moraju biti građeni prema načelima dobrog oblikovanja kako bi se smanjila međuovisnost i omogućila jednostavna nadogradnja u budućnosti

4. Implementacija

4.1. Arhitektura i dizajn

Arhitektura aplikacije temelji se na hijerarhijskoj raspodijeli komponenti¹. Komponente su odgovorne i znaju samo za one komponente izravno ispod njih u hijerarhiji. Dvije komponente na odvojenim granama hijerarhije mogu prenositi podatke jedna drugoj ili metodom podizanja stanja² ili korištenjem *history* objekta iz *react-router* paketa. Budući da komponente moraju biti ugniježdene u *Router* komponentu kako bi koristile *history* objekt, takvo prenošenje podataka je ustvari slično podizanju stanja jer također postavljaju stanje roditeljske komponente da bi izazvale promjene.

Najniže komponente u hijerarhiji enkapsuliraju elemente korisničkog sučelja te idejno ne implementiraju stanje i logiku dalje od toga kako će prikazati podatke. Glavna ideja iza njih je da budu male i lagane kako bi se mogle ponovno koristiti. Takve komponente se često nazivaju prezentacijskima i dio gotovih se već nalazi u paketu *react-bootstrap*.

Komponente više u hijerarhiji su zadužene za definiranje stanja i dohvat podataka za niže komponente. Budući da ove komponente sadrže i definiraju odnos prezentacijskih komponenti, često se za njih koristi izraz „sadržajne komponente“ (engl. *container components*)[21]. Ovakve komponente su uglavnom jedinstvene, odnosno teško da će doći do njihovog ponovnog upotrebljavanja.

¹Arhitektura temeljena na komponentama (engl. *Component-Based Architecture*) je tipičan način gradnje React aplikacija[22].

²Vidi pododjeljak 2.2.3.

4.1.1. Raspodjela komponenti

Komponente su logički raspodijeljene hijerarhijski s *App* kao najvišom, odnosno korijenskom komponentom. U nju je izravno ugniježđen *Router* u kojem su definirane sve putanje. Izvan definicija putanja, no unutar komponente *Router*, je jedino komponenta navigacijske trake (*Navigation*). Razlog tomu je što se navigacijska traka nalazi u svim dijelovima aplikacije te, budući da je gotovo nepromjenjiva³, želi se izbjeći njeno ponovno crtanje pri promjeni putanje.

Ispod navigacijske trake će se, ovisno o putanji, crtati jedna od sljedećih komponenti:

- *Main* [/home] – glavno sučelje aplikacije
- *OfferContainer* [/offer] – sučelje u kojem korisnik upravlja vlastitim ponudama
- *ReservationContainer* [/reservation] – sučelje u kojem korisnik upravlja zahtjevima (primljenim i poslanim)
- *ProfileContainer* [/profile/:id] – pregled profila korisnika
- *ItemDetailsContainer* [/item/:id] – pregled detalja ponude
- *Login* [/login] – obrazac za prijavu
- *Registration* [/registration] – obrazac za registraciju

Svaka od njih, kao i komponenta *Navigation*, definira vlastito stanje i ima pristup *history* objektu⁴.

Daljnje komponente koje definiraju vlastito stanje su *NewOfferForm* i *EditProfileForm*. One nemaju vlastitu putanju, već su potkomponente od *OfferContainer* i *ProfileContainer*.

Sve ostale komponente aplikacije ne definiraju vlastito stanje. Potrebne podatke im šalju roditeljske komponente, a sva definirana logika im služi isključivo na prikaz tih podataka. Primjeri takvih komponenti su *ItemThumbnailContainer*, *ItemThumbnail*, *ItemList*, *ReservationTable*, *ImageColumn*, *TextColumn*, *LoadingStatus*, *FieldGroup* i komponente paketa *react-bootstrap*.

³Navigacijska traka prikazuje drukčije poveznice ovisno je li korisnik prijavljen u sustav ili ne.

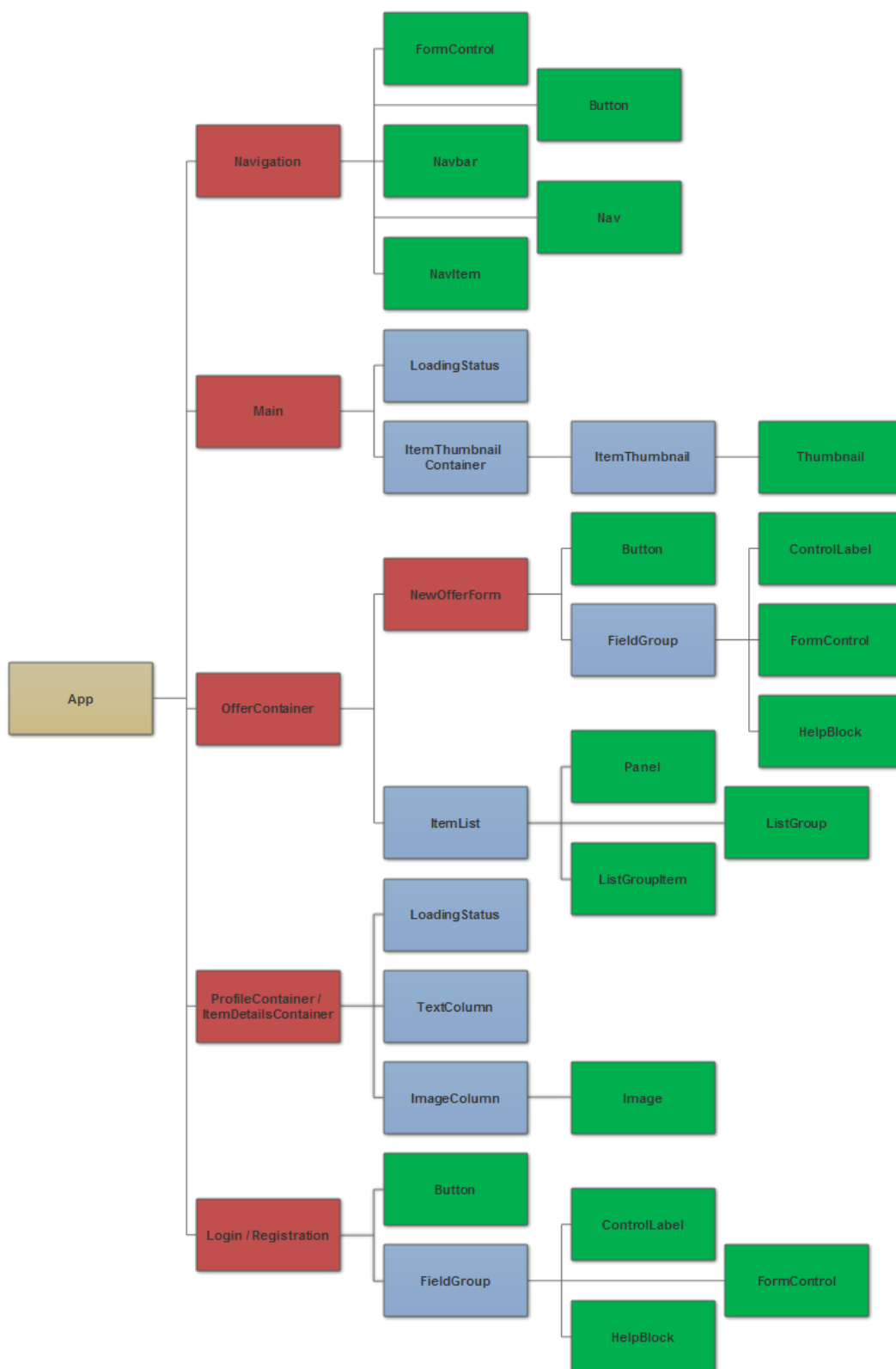
⁴Vidi odjeljak 2.4.

Fizički su komponente raspodijeljene svaka u svoju datoteku, koje su potom organizirane u jednu od sljedećih mapa, sukladno obrascima uporabe koje ispunjavaju:

- *auth* – autentifikacija korisnika
 - *Login*
 - *Registration*
- *item* – prikaz ponuda, tj. stvari u ponudama
 - *ItemDetailsContainer*
 - *ItemList*
 - *ItemThumbnail*
 - *ItemThumbnailContainer*
- *offer* – upravljanje ponudama
 - *NewOfferForm*
 - *OfferContainer*
- *profile* – prikaz i upravljanje korisničkim podacima
 - *EditProfileForm*
 - *ProfileContainer*
- *reservation* – upravljanje zahtjevima
 - *ReservationContainer*
 - *ReservationTable*
- *shared* – dijeljene prezentacijske komponente
 - *ImageColumn*
 - *LoadingStatus*
 - *TextColumn*

Izvan tih mapa su datoteke s ključnim komponentama aplikacije (*App*, *Navigation* i *Main*) te komponente definirane u vanjskim paketima.

⁵Nedostaje hijerarhija od komponenti *Reservation* i *EditProfileForm* te su komponente koje koriste iste potkomponente grupirane.



Slika 4.1: Pojednostavljen⁵ prikaz hijerarhije komponenti u aplikaciji – zeleno: komponente iz *react-bootstrap* ; crveno: komponente koje definiraju vlastito stanje

4.1.2. Pomoćni moduli

Osim vanjskih biblioteka, komponente koriste pomoćne module za ostvarivanje specifičnih operacija u različitim dijelovima aplikacije. Dva takva modula su datoteke *Auth.js* i *Constants.js*.

U datoteci *Auth.js* je definirana klasa *Auth* koja sadrži statičke metode koje se koriste za autentifikaciju i autorizaciju korisnika. Prijavom korisnika u aplikaciju, informacije o korisniku (id i token) spremaju se u *localStorage*⁶ kako bi se mogle dohvatiti iz različitih komponenti i u različitim instancama aplikacije u istom pregledniku. Klasa *Auth* definira metode za spremanje, dohvat i brisanje tih informacija. Ona enkapsulira logiku upravljanja podacima identiteta korisnika što omogućuje jednostavnu izmjenu tih funkcionalnosti. U slučaju da se u kasnijoj nadogradnji želi preći na neki drugi spremnik, kao što su primjerice kolačići (engl. *cookies*)⁷, potrebno je promijeniti samo kod u datoteci *Auth.js*.

Datoteka *Constants.js* isporučuje nepromjenjive objekte komponentama aplikacije, odnosno služi kao spremnik globalnih konstanti (bez da se stvarno deklariraju u globalnom dosegu). Jedna takva konstanta je lista kategorija sadržaja (*ITEM_TYPES*) koja se koristi u komponentama *Main* i *NewOfferForm*. U prvoj komponenti služi da stvoriti kartičnu navigaciju za filtriranje ponuda⁸, a u drugoj za definirati koje je kategorije sadržaj pri stvaranju ponude.

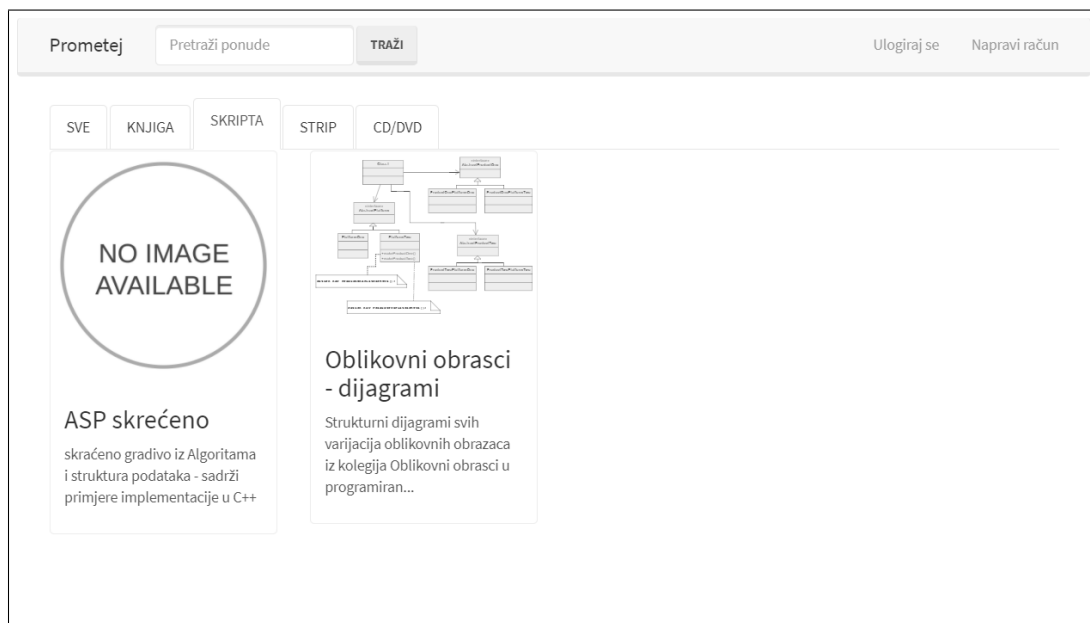
⁶Objekt za trajno spremanje podataka u pregledniku, svojstvo *Window* objekta. Više na: <https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage>

⁷<https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies>

⁸Vidi sljedeći odjeljak.

4.2. Glavno sučelje

Glavno sučelje (na slici 4.2) je sučelje koje svi korisnici prvo ugledaju kada otvore aplikaciju. Ono služi za pregled i pronalazak sadržaja, odnosno ponuda, te je jedino sučelje koje u potpunosti nudi iste mogućnosti i javnom i autoriziranom korisniku. Dapače, jedina razlika u onome što vide javni i autorizirani korisnici je navigacija, koja je ostvarena kao zasebna komponenta (*Navigation*) koja se prikazuje u svim sučeljima aplikacije i mijenja svoj izgled ovisno o tipu korisnika.



Slika 4.2: Glavno sučelje aplikacije

Izgled sučelja je modeliran pomoću komponente *Main*. U njoj je definirano stanje, dohvat podataka i navigacija po kategorijama sadržaja pomoću kartica (engl. *tabs*). Također je definirana i logika filtracije sadržaja kojeg šalje kao listu u komponentu *ItemThumbnailContainer*.

Sadržaj se filtrira navigacijom karticama po kategoriji i s obzirom na upit iz tražilice komponente *Navigation*. Budući da se tražilica nalazi u navigaciji, komunikacija s komponentom *Main* je ostvarena pomoću *history* objekta, odnosno preusmjeravanjem korisnika na novu rutu. Upisom teksta u tražilicu korisnika se preusmjerava rutu s upitom (npr. „/home?upit“) koja sadrži informaciju o filtraciji. Potom se filtrira sav sadržaj koji ne sadrži upisani tekst u svom imenu, niti u početku svog opisa. Filtracija se jednostavno resetira ponovnim klikom na ime aplikacije (u ovom slučaju je radni naslov *Prometej*) u navigaciji, odnosno odlaskom na rutu „/home“.

Slanjem liste sadržaja komponenti *ItemThumbnailContainer*, ona stvara potreban broj elemenata komponente *ItemThumbnail*, u kojima je definiran prikaz za elemente liste, te se brine o njihovom razmještanju. U svakom *ItemThumbnail* elementu je definirana poveznica na *ItemDetails* komponentu za sadržaj kojeg element prikazuje.

Dohvat podataka vrši se u metodi *componentDidMount*⁹ te se podaci spremaju u stanje komponente, što je preporučen obrazac korištenja te metode[14]. Dok traje ta operacija, na sučelju crta se komponenta *LoadingStatus* kako bi se korisnika obavijestilo da je dohvat podataka u tijeku. Ista komponenta crta se i u slučaju greške u dohvat podataka, samo se korisniku prikazuje poruka da je došlo do greške.

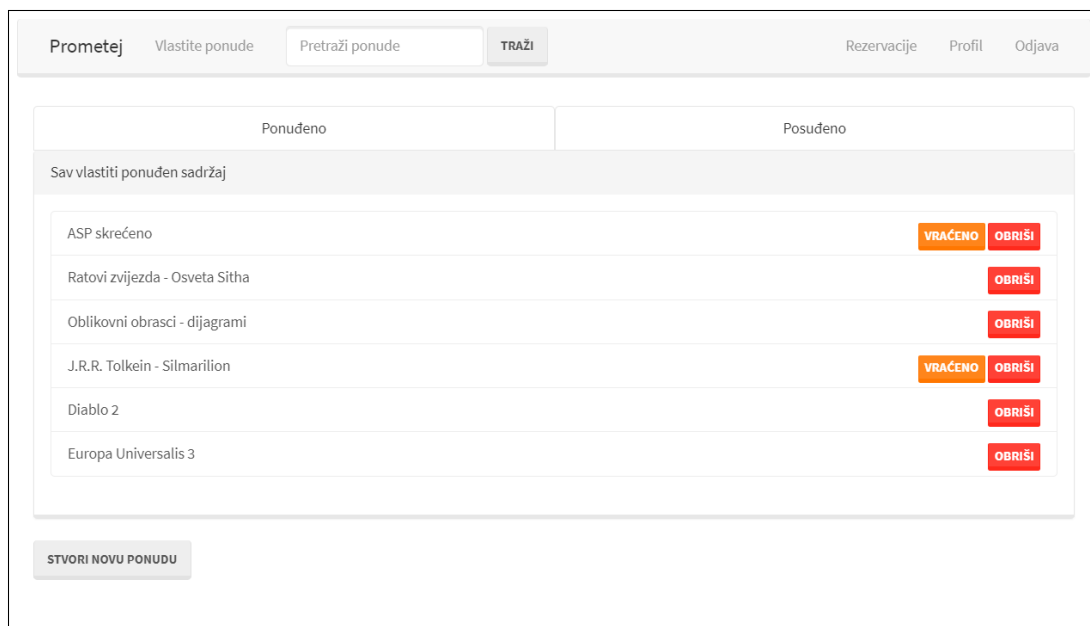
```
1 render() {
2   let tabs = ITEM_TYPES.map((type, index) => (
3     <Tab eventKey={index+2} title={type.toUpperCase()} />
4   ));
5   tabs.unshift(<Tab eventKey={1} title="SVE" />);
6
7   return (
8     <div>
9       <Tabs defaultActiveKey={1}
10         onSelect={this.handleSelectCategory}>
11         {tabs}
12       </Tabs>
13       <LoadingStatus isError={this.state.isError}
14         isLoading={this.state.isLoading} />
15       <ItemThumbnailContainer items={this.getFilteredItems()} />
16     </div>
17   );
18 }
```

Isječak koda 4.1: Metoda *render* komponente *Main*

⁹Metoda životnog ciklusa komponente. Vidi pododjeljak 2.2.1.

4.3. Upravljanje ponudama

Za upravljanje ponudama su primarno zadužene komponenta *OfferContainer* i njena potkomponenta *NewOfferForm*. Prva daje pregled svih ponuda koje je korisnik unio u sustav i tuđih sadržaja koje ima u vlasništvu, dok druga služi za dodavanje nove ponude u sustav. Pristupiti im može isključivo autorizirani korisnik klikom u navigaciji na „Vlastite ponude“, odnosno na ruti `./offer`.



Slika 4.3: Sučelje za upravljanje ponudama

Ulaskom u sučelje korisnik može pregledavati dvije liste ponuda. Klikom na element liste ga se preusmjerava na informacije te ponude. Prva lista je lista posuđenih sadržaja, gdje korisnik pregledava ponude koje je on dao. Nad elementima te liste korisnik ima dvije dodatne akcije, odnosno dva gumba. Prvi se pojavljuje samo ako je sadržaj već posuđen drugom korisniku i klikom na njega korisnik postavlja da mu je taj sadržaj vraćen, odnosno da je ponuda ponovno slobodna. Druga lista je lista sadržaja koje je korisnik posudio i nad elementima te liste nema nikakvih akcija, već ih može samo pregledavati. Liste se izmjenjuju pomoću kartične navigacije.

Klikom na gumb ispod liste se otvara forma kojom korisnik može unijeti nove ponude u sustav. Svojstva ponude su njen naziv, opis, poveznica na sliku ponuđenog sadržaja i kategorija koja se bira padajućim izbornikom¹⁰.

¹⁰Vrijednosti padajućeg izbornika su generirane pomoću *Constants.js*.

4.4. Upravljanje zahtjevima

Zahtjevi se, s perspektive korištenja aplikacije, odnose na rezerviranje sadržaja, odnosno obavješćavanje vlasnika ponude da je korisnik zainteresiran za nju. Zahtjev se stvara tako da korisnik¹¹ ode na rutu s detaljima ponude („/item/:id“) nekog drugog korisnika i klikne na gumb „Rezerviraj“. Gumb se prikazuje korisniku samo u slučaju ako je ponuda slobodna (sadržaj nije posuđen) i zahtjev za rezervacijom tog sadržaja nije već poslan, tj. ako taj korisnik nije već kliknuo gumb. Tom akcijom zahtjev se sprema u bazu. Podaci koje sadrži zahtjev su primarni ključevi ponude, vlasnika sadržaja i korisnika koji je poslao zahtjev.

Popis zahtjeva može se pregledati u sučelju za upravljanje rezervacijama (ruta „/reservations“, slika 4.4). U njoj se nalaze dvije liste zahtjeva, organizirane na način sličan listama ponuda u sučelju za upravljanje sadržajima¹². Prva je lista zahtjeva na ponude u vlasništvu korisnika. Njih korisnik može prihvatiti ili obrisati. Prihvatanjem ponude mijenja se trenutni vlasnik sadržaja, odnosno ponuda postaje zauzeta, te se zahtjev briše iz baze. Brisanje nema nikakvih dodatnih radnji uz brisanje zahtjeva iz baze. Druga lista je lista zahtjeva koje je korisnik koji trenutno koristi aplikaciju poslao. Nad njima se može izvršiti samo akcija brisanja.

Prometej	Vlastite ponude	Pretraži ponude	TRAŽI	Rezervacije	Profil	Odjava
Zahtjevi na vlastite ponude			Poslani zahtjevi			
Vrijeme	Predmet ponude	Poslao				
02. 07. 2017. 22:03:34	Diablo 2	novi			PRIHVATI	OBRIŠI
02. 07. 2017. 22:03:38	Europa Universalis 3	novi			PRIHVATI	OBRIŠI

Slika 4.4: Sučelje za upravljanje zahtjevima

¹¹Funkcionalnosti povezane uz zahtjeve su vidljive isključivo autoriziranim korisnicima, odnosno sve opisano u ovom odjeljku vrijedi isključivo za autoriziranog korisnika.

¹²Vidi prethodni odjeljak.

5. Zaključak

Cilj ovog rada, ostvarivanje jednostavnog korisničkog sučelja knjižnice za razmjenu multimedijskog sadržaja, je uspješno postignut. Rješenje je oblikovano na način da omogućuje nadodavanje brojnih nadogradnji u budućnosti, bez potrebe za velikim promjenama postojećeg koda.

Neke od predviđenih nadogradnji su slanje zahtjeva za dohvaćanje filtriranog sadržaja na serveru, bilježenje i prikaz ocjena korisnika i sadržaja, te implementacija dodatnih sučelja poput sučelja za prikaz povijesti posuđivanja ili sučelja za komunikaciju između korisnika. Budući da odabir bilo kojeg od kriterija filtriranja (kartična navigacija ili tražilica) izaziva promjenu stanja, na tu promjenu stanja može se na jednostavan način, putem objekta obećanja¹, nadovezati dohvat filtriranog sadržaja sa servera i time riješi potencijalan problem sporog učitavanja većeg broja ponuda. Za bilježenje i prikaz ocjena korisnika i sadržaja potrebno je samo izraditi interaktivnu² komponentu za prikaz ocjene. Potom u komponente *ProfileContainer* i *ItemDetails* definirati funkcije za dohvat i slanje ocjene, jer bi se bilježenje samih ocjena odvijalo na serveru, te ih poslati kao atribut novoizrađenoj komponenti. Implementacije dodatnih sučelja zahtijevaju najmanje promjena postojećeg koda, jer bi se ostvarile kao komponente s vlastitim stanjem, povezane izravno na *App* komponentu. Nužno je samo, za potrebe što kvalitetnijeg korisničkog iskustva, nadodati poveznice na rute tih sučelja u nekima od postojećih komponenti aplikacije.

React se pokazala kao izvrsna biblioteka za ostvarivanje lake nadogradivosti i kvalitetne organizacije korisničkog sučelja i njegovog koda. Uz to, produkcijska verzija aplikacije, generirane pomoću *create-react-app* paketa, bilježi izvrsne performanse na svim testiranim preglednicima³ i njihovim emulacijama za mobilne uređaje. Upotreba *Bootstrap* teme je učinila korisničko sučelje prilagodljivim svim rezolucijama,

¹Vidi pododjeljak 2.1.3.

²Korisnik mora moći unijeti vlastitu ocjenu, tj. kako on ocjenjuje drugog korisnika ili sadržaj.

³Google Chrome, Mozilla Firefox i Microsoft Edge.

pri čemu nema narušavanja korisničkog iskustva promjenom rezolucije.

Uz opisane prednosti korištenih alata, uporaba ECMAScript 2015 JavaScript standarda se pokazala zahvalnom za čitljivost i lakše održavanje koda. Takva karakteristika ukazuje na budućnost korištenja JavaScripta u sve kompleksnijim programskim rješenjima.

Aplikacija slična izgrađenoj u ovom radu (u trenutku pisanja rada) ne postoji na hrvatskom tržištu. Najbliže obrascu korištenja kojeg ispunjava su grupe s temom posuđivanja na društvenim mrežama, od kojih nijedna ne implementira mogućnost filtriranja sadržaja. Prostora na tržištu za ovakvu aplikaciju stoga svakako ima, no upitna je njena financijska isplativost. Budući da je riječ o aplikaciji čije se ispravno korištenje uvelike oslanja na dobroj volji samih korisnika, odnosno na premisi čovjekove potrebe za dijeljenjem i nalaženjem znanja, nejasno je koliki bi profit mogla generirati bez da odbija korisnike koji su ključni za njeno postojanje. Trebalo bi se pažljivo istražiti i odlučiti za model monetizacije koji bi aplikaciju učinio samoodrživom (isplativom za održavanjem) ili čak profitabilnom. Dodatan problem u tome stvara ograničenje aplikacije na korisnike koji su u hrvatskom govornom području, no takva analiza je izvan domene ovog rada.

LITERATURA

- [1] MDN – Mozilla Developer Network: Arrow functions. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises, . Datum zadnje izmjene: 31.05.2017., Datum pristupa: 01.06.2017.
- [2] MDN – Mozilla Developer Network: Classes. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes>, . Datum zadnje izmjene: 27.05.2017., Datum pristupa: 31.05.2017.
- [3] MDN – Mozilla Developer Network: Const. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const>, . Datum zadnje izmjene: 24.05.2017., Datum pristupa: 30.05.2017.
- [4] MDN – Mozilla Developer Network: Fetch API. https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API, . Datum pristupa: 04.06.2017.
- [5] MDN – Mozilla Developer Network: Let. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let>, . Datum zadnje izmjene: 30.05.2017., Datum pristupa: 30.05.2017.
- [6] MDN – Mozilla Developer Network: Inheritance and the prototype chain. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance_and_the_prototype_chain, . Datum zadnje izmjene: 01.05.2017., Datum pristupa: 31.05.2017.
- [7] MDN – Mozilla Developer Network: Using Fetch. https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch, . Datum pristupa: 04.06.2017.

- [8] MDN – Mozilla Developer Network: Using promises. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises, . Datum zadnje izmjene: 24.05.2017., Datum pristupa: 04.06.2017.
- [9] React-Bootstrap. <https://react-bootstrap.github.io/>, . Datum pristupa: 03.06.2017.
- [10] Wikipedia – The Free Encyclopedia: Bootstrap (front-end framework). [https://en.wikipedia.org/wiki/Bootstrap_\(front-end_framework\)](https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)), . Datum nastanka: 18.09.2012., Datum zadnje izmjene: 01.06.2017., Datum pristupa: 03.06.2017.
- [11] Github: facebookincubator/create-react-app. <https://github.com/facebookincubator/create-react-app>. Datum nastanka: 15.07.2016., Datum zadnje izmjene: 23.05.2017., Datum pristupa: 02.06.2017.
- [12] React. <https://facebook.github.io/react>, . Datum pristupa: 30.05.2017.
- [13] React – Docs: Components and Props. <https://facebook.github.io/react/docs/components-and-props.html>, . Datum pristupa: 31.05.2017.
- [14] React – Docs: React.Component. <https://facebook.github.io/react/docs/react-component.html>, . Datum pristupa: 31.05.2017.
- [15] React – Docs: JSX in Depth. <https://facebook.github.io/react/docs/jsx-in-depth.html>, . Datum pristupa: 31.05.2017.
- [16] React – Docs: Lifting State Up. <https://facebook.github.io/react/docs/lifting-state-up.html>, . Datum pristupa: 31.05.2017.
- [17] React – Docs: Rendering Elements. <https://facebook.github.io/react/docs/rendering-elements.html>, . Datum pristupa: 31.05.2017.
- [18] Github: facebook/react. <https://github.com/facebook/react>, . Datum nastanka: 29.05.2013., Datum zadnje izmjene: 29.05.2017., Datum pristupa: 30.05.2017.

- [19] React Training: React Router – Web. <https://reacttraining.com/react-router/web>, . Datum pristupa: 03.06.2017.
- [20] Wikipedia – The Free Encyclopedia: React (JavaScript library). [https://en.wikipedia.org/wiki/React_\(JavaScript_library\)](https://en.wikipedia.org/wiki/React_(JavaScript_library)), . Datum nastanka: 02.01.2015., Datum zadnje izmjene: 27.05.2017., Datum pristupa: 30.05.2017.
- [21] Dan Abramov. Presentational and Container Components. https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0. Datum nastanka: 23.03.2015., Datum pristupa: 05.06.2017.
- [22] Dan Shapiro. Understanding Component-Based Architecture. <https://medium.com/@dan.shapiro1210/understanding-component-based-architecture-3ff48ec0c238>. Datum nastanka: 16.06.2016., Datum pristupa: 05.06.2017.
- [23] David Walsh. Fetch API. <https://davidwalsh.name/fetch>. Datum nastanka: 15.04.2015., Datum pristupa: 05.06.2017.

Korisničko sučelje knjižnice za razmjenu multimedijskog sadržaja

Sažetak

Tema ovog rada je razvoj korisničke strane aplikacije za razmjenu multimedijskih sadržaja koristeći moderne alate, biblioteke i radna okruženja. Korisničkog sučelje je izgrađeno korištenjem JavaScript biblioteke React. Cjelokupna aplikacija korisniku nudi mogućnosti posuđivanja knjiga, stripova, CD-ova, DVD-ova i sl. od drugih korisnika. Implementira funkcionalnosti: prijave, registracije, dodavanja i upravljanja vlastitog multimedijskog sadržaja, slanje zahtjeva za posuđivanje tuđeg sadržaja, filtrirani pregled svog sadržaja u sustavu.

Cilj rada je ostvarenje jednostavnog i intuitivnog korisničkog sučelja programskog rješenja za razmjenu fizičkih kopija multimedijskog sadržaja te pregled korištenih tehnologija.

Ključne riječi: korisničko sučelje, JavaScript, React

Multimedia Exchange Library User Interface

Abstract

The subject of this thesis is the development of a client-side application for a multimedia exchange library using modern front-end tools, libraries and frameworks. The user interface is built using the React JavaScript library. The application as a whole enables the user to exchange books, comics, CDs, DVDs, etc. with other users. The application implements the following functionalities: login, registration, adding and managing owned multimedia items, requesting to borrow other users' items, a filtered overview of all items in the system.

The goal of this thesis is the realization of a simple and intuitive user interface for a software solution for exchanging physical copies of multimedia items, along with giving an overview of the technologies used in the process.

Keywords: UI, JavaScript, React