Razvoj web aplikacija pomoću React i Redux programskih biblioteka

Marković, Toni

SVEUČILIŠTE U SPLITU

PRIRODOSLOVNO MATEMATIČKI FAKULTET

ZAVRŠNI RAD

RAZVOJ WEB APLIKACIJA POMOĆU REACT I REDUX PROGRAMSKIH BIBLIOTEKA

Toni Marković

Split, rujan 2022.

SVEUČILIŠTE U SPLITU

PRIRODOSLOVNO MATEMATIČKI FAKULTET

ZAVRŠNI RAD

RAZVOJ WEB APLIKACIJA POMOĆU REACT I REDUX PROGRAMSKIH BIBLIOTEKA

Toni Marković

Split, rujan 2022.

Temeljna dokumentacijska kartica

Završni rad

Sveučilište u Splitu
Prirodoslovno-matematički fakultet
Odjel za informatiku
Ruđera Boškovića 33, 21000 Split, Hrvatska

RAZVOJ WEB APLIKACIJA POMOĆU REACT I REDUX PROGRAMSKIH BIBLIOTEKA

Toni Marković

SAŽETAK

Rad se bavi razvojem web aplikacija koristeći React i Redux tehnologije. React je jedna od najpopularnijih biblioteka za razvoj web aplikacija. Omogućava podjelu korisničkog sučelja na manje komponente koje se nakon toga povezuje u veće cjeline. Još jedna bitna karakteristika su React Hooks koji omogućuju funkcijskim komponentama pristup stanju i drugim React značajkama. Redux je spremnik predvidljivog stanja za JavaScript aplikacije. Pomaže u pisanju aplikacija koje se ponašaju dosljedno, izvode u različitim okruženjima i koje je lako testirati. Naglasak je na korištenju Redux Toolkit-a koji je preporučeni pristup pisanja Redux logike.

Ključne riječi: Web aplikacija, JavaScript, React, Hooks, Redux Toolkit

Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad sadrži: 28 stranica, 14 grafičkih prikaza i 11 literaturnih navoda.

Izvornik je na hrvatskom jeziku.

Mentor: **Dr.sc. Divna Krpan**, docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Ocjenjivači: **Dr.sc. Divna Krpan**, docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Dr.sc. Saša Mladenović, izvanredni profesor Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Dr.sc. Goran Zaharija, docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad prihvaćen: Rujan, 2022.

Basic documentation card

Thesis

University of Split
Faculty of Science
Department of Computer Science
Ruđera Boškovića 33, 21000 Split, Croatia

DEVELOPING WEB APPLICATIONS USING REACT AND REDUX LIBRARIES

Toni Marković

ABSTRACT

This paper is concentrated on developing web applications using React and Redux technologies. React is one of the most popular frameworks for developing web applications. It allows dividing the user interface into smaller components that are then connected to larger units. Another important characteristic is React Hooks which allow function components access to state and other React features. Redux is a predictable state container for JavaScript apps. It helps you write applications that behave consistently, run in different environments, and are easy to test. The emphasis is on the use of Redux Toolkit that is recommended approach for writing Redux logic.

Key Words: Web application, JavaScript, React, Hooks, Redux Toolkit

Thesis deposited in library of Faculty of science, University of Split.

Thesis consist of: 28 pages, 14 figures and 11 references

Original language: Croatian

Supervisor: **Divna Krpan**, **Ph.D.** Assistant Professor of Faculty of Science, University of Split

Reviewers: **Divna Krpan**, **Ph.D.** Assistant Professor of Faculty of Science, University of Split,
Saša Mladenović, Ph.D. Associate Professor of Faculty of Science, University of Split,
Goran Zaharija, Ph.D. Assistant Professor of Faculty of Science, University of Split

Thesis accepted: September, 2022.

ZAHVALA

Želio bih izraziti nekoliko riječi zahvale svim ljudima koji su bili uz mene tijekom ovih godina i bez kojih ne bih bio ovdje gdje jesam.

Veliko hvala profesorici i mentorici Divni Krpan na iskazanom povjerenju, vodstvu i korisnim savjetima tijekom izrade ovog rada. Hvala Vam na strpljivosti, razumijevanju i svemu što ste me naučili kroz brojne kolegije koje ste mi predavali u ove 3 godine.

Hvala i mojoj obitelji što su uvijek bili tu za mene i što su me podržavali za vrijeme studija. Bez vas ovo ne bi bilo moguće i pomogli ste mi da prijeđem sve prepreke koje su mi stajale na putu do cilja. Naravno, hvala i ostalim članovima uže i šire obitelji.

Također, zahvalio bih se svim prijateljima, i starim i novim koje sam upoznao tijekom studiranja. Pružili ste mi mnogo nezaboravnih trenutaka i nikad mi niste dali da odustanem kad je bilo najteže.

Na kraju bih se zahvalio svim profesorima koje sam sreo tijekom studija. Hvala vam na suradnji, ugodnom boravku i stečenim znanjima.

Sadržaj

1 Uvod

Razvojem tehnologije, internet je svakim danom sve dostupniji i web je postao dio svakodnevnog života. Web aplikacije su jedna od bitnih komponenti u današnjem svijetu. Korištenjem web aplikacija, organizacije se mogu jednostavnije razvijati i postizati ciljeve u kraćem vremenu. Dostupne su na velikom broju pametnih uređaja kao što su mobiteli, tableti, računala i pametni satovi, te ih je potrebno prilagoditi širokom rasponu uređaja. Zbog sve veće potrebe za digitalnim rješenjima putem interneta, nastali su mnogi programski jezici i različiti programski okviri koji su pridonijeli lakšoj i bržoj izradi web aplikacija i pružanju što boljeg korisničkog iskustva. Facebook je, kao vodeća društvena mreža, zbog povećanog broja korisnika i širenja platforme, odlučio izgraditi svoj vlastiti programski okvir poznatiji kao React.

React je besplatna *front-end* JavaScript biblioteka za izgradnju korisničkih sučelja temeljenih na UI (engl. *User Interface*) komponentama. Prva verzija *Reacta* izašla je u svibnju 2013. godine i od tada se svakim danom sve više razvija. Danas je jedna od najpopularnijih biblioteka za izradu web aplikacija. Omogućava jednostavno stvaranje dinamičkih aplikacija jer zahtjeva manje kodiranja i pruža veću funkcionalnost u odnosu na „čisti“ JavaScript, gdje kodiranje može postati kompleksno u kratkom vremenu. *React* koristi i virtualni DOM (engl. *Document Object Model*), čime brže stvara web aplikacije. Virtualni DOM uspoređuje prethodna stanja komponenti i ažurira samo one stavke koje su promijenjene u stvarnom DOM-u, umjesto ponovnog ažuriranja svih komponenti. Još jedna bitna karakteristika su *reusable* komponente. Komponente su građevni blokovi bilo koje React aplikacije, a jedna se aplikacija obično sastoji od više komponenti. Svaka komponenta ima svoju funkciju i može se koristiti u istoj aplikaciji na različitim mjestima, što značajno smanjuje vrijeme razvoja. Također, React se, osim web aplikacija, može koristiti i za razvoj mobilnih aplikacija. Radi se o okviru koji se zove React Native, izveden iz samog React-a, koji je iznimno popularan kod razvoja aplikacija za mobilne uređaje.

Redux je biblioteka namijenjena za čuvanje stanja aplikacije, odnosno pohranjuje vrijednosti svih varijabli u centralno skladište kojem svaka komponenta ima pristup. Koristi se u većim aplikacijama, gdje je tok podataka složeniji i teško ga je implementirati samo pomoću komponenata. Redux je napisan u JavaScript-u, kao i React. Također, Redux nije strogo povezan s React-om i moguće ga je koristiti sa bilo kojom drugom bibliotekom.

U ovom radu opisan je način stvaranja React aplikacije i korištenje Reduxa za upravljanje stanjem varijabli i prosljeđivanjem njihovih vrijednosti različitim komponentama. Uz to, opisan je i način korištenja različitih hooks koji su bitni za svaku aplikaciju, te izrada korisničkog sučelja koristeći okvir Material UI.

2 Web Aplikacije

Web aplikacija je program koji je pohranjen na udaljenom poslužitelju i isporučuje se putem interneta kroz sučelje preglednika. Mogu biti dizajnirane za različite namjene i namijenjene su svima za korištenje, od različitih organizacija pa sve do pojedinaca. Često korištene web aplikacije mogu uključivati web poštu, online kalkulatore ili e-trgovine. Nekim web aplikacijama se može pristupiti samo određenim preglednikom, no većina je dostupna bez obzira na preglednik. Web aplikacije se ne moraju preuzimati jer im se pristupa putem mreže. Korisnici im mogu pristupiti putem web preglednika kao što su *Google Chrome, Mozilla Firefox* ili *Safari*. Da bi web aplikacija radila, potreban joj je web poslužitelj, aplikacijski poslužitelj i baza podataka. Web poslužitelji upravljaju zahtjevima koji dolaze od klijenta, dok aplikacijski poslužitelj dovršava traženi zadatak. Baza podataka se može koristiti za pohranjivanje svih potrebnih informacija.

Većina web aplikacija je napisana u JavaScript-u, HTML5 (engl. *HyperTextMarkupLanguage)* ili CSS-u (engl. *Cascading Style Sheets)*. Programiranje na klijentskoj strani obično koristi navedeno, jer je korisno za izradu sučelja aplikacije. Programiranje na strani poslužitelja služi za stvaranje skripti koje će aplikacija koristiti. Tu su popularni jezici kao *Python, Java* i *Ruby* [1].

Kako bi se stvorilo dobro korisničko iskustvo, važno je odabrati dobru arhitekturu web aplikacija. Razlikujemo aplikacije s jednom stranicom (engl. *Single Page-Application,* skraćeno SPA) i aplikacije s više stranica (engl. *Multi Page-Application,* skraćeno MPA). SPA i MPA se razlikuju po tome što se MPA ponovno učitava svaki put kada korisnik otvori novu stranicu u pregledniku, stoga je potrebno više vremena i truda za održavanje sigurnosti. SPA se ponaša poput preglednika i eliminira potrebu za ponovnim učitavanjem stranice, ali je razina sigurnosti niža. Općenito se oslanjaju na *JavaScript* što ih čini ranjivima na napade. SPA se smatraju modernijim i prilagođenijim današnjim potrebama agilnog razvoja. Međutim, to ne znači da MPA nisu korisne i jedan od najpoznatijih primjera je najveća e-trgovina na svijetu, *Amazon*. Općenito su izvrsne za masivne aplikacije koje se mogu sastojati od velikog broja stranica koje se osvježavaju svaki put kada se podaci na njima promjene. Također, s razvojem AJAX-a (engl. *Asynchronus JavaScript And XML*), programeri su u mogućnosti stvoriti brze i dinamične MPA koje razmjenjuju malu količinu podataka s poslužiteljem kako bi stranica mogla ostati ažurirana bez ponovnog učitavanja svaki put kada se podaci promijene.

2.1 JavaScript

JavaScript je programski jezik web-a. Većina modernih web stranica koristi JavaScript, kao i svi moderni web preglednici na stolnim računalima, igraćim konzolama, tabletima i pametnim telefonima, što ga čini najprisutnijim programskim jezikom u povijesti. Svrha je omogućiti interaktivnost na različitim web stranicama i aplikacijama. Od jednostavnih značajki, poput vizualnih efekata ili prikaza trenutnog datuma i vremena, do sofisticiranijih i kompleksnijih zadataka kao što su dohvaćanje podataka ili izrada sučelja za različite aplikacije. Riječ je o jednoj od tri tehnologija koje svaki developer mora naučiti. Pomoću HTML-a se stvara sadržaj, CSS služi za prikladnu prezentaciju, a JavaScript pruža korisniku mogućnost interakciju sa različitim elementima [2].

2.1.1 Varijable

Varijable se koriste za pohranjivanje informacija koje se mogu referencirati i manipulirati u programu. Također omogućavaju pridruživanje različitih imena varijablama, kako bi program bio lakše razumljiv. Korisno je zamisliti varijable kao spremnike koji drže informacije. Njihova jedina svrha je imenovanje i pohrana podataka u memoriji i ti podaci se mogu koristiti kroz cijeli program.

Varijable možemo kreirati na više načina. U ranijim verzijama JavaScript-a, varijable su se deklarirale koristeći ključnu riječ `var` iza koje slijedi naziv varijable. Nakon ES6 (novije verzije JavaScript-a) postoje 2 nova načina za deklariranje varijabli, a to su ključne riječi `let` i `const`.

```
var variableName = '';
let variableName = '';
const variableName = '';
```

Kod 1 Različiti načini deklariranja varijabli

Tip varijable `let` dijeli mnogo sličnosti sa tipom `var`, ali za razliku od `var` ima određena ograničenja. Razlika je u tome što su varijable deklarirane pomoću `var` dostupne i izvan bloka u kojem se deklarirane. Blokovi se označavaju pomoću vitičastih zagrada. `Const` je još jedan tip varijable dodijeljen podacima čija se vrijednost ne može i neće promijeniti kroz skriptu.

2.1.2 Tipovi Podataka

Kao što je već spomenuto, vrijednosti se mogu pohranjivati u varijable i te vrijednosti moraju biti u obliku jednog od unaprijed definiranih tipova podataka. Postoje različiti tipovi podataka koje možemo podijeliti u dvije grupe, a to su primitivni tipovi podataka i ne primitivni tipovi podataka.

Primitivni tipovi podataka su:

- Broj (engl. *Number*)
- String
- Boolean
- Null
- Nedefiniran (engl. *Undefined*)
- Simbol (engl. *Symbol*)

U JavaScript-u nije potrebno definirati jeli broj decimalan ili ne jer se o tome brine sam programski jezik. Postoje različite operacije s brojevima koje uključuju zbrajanje, oduzimanje, množenje i dijeljenje.

String je niz znakova i koristi se za predstavljanje teksta. Jednostavno rečeno, to su polja teksta. Pri korištenju string-ova potrebno je koristiti navodnike, odnosno potrebno je okružiti riječ ili rečenicu navodnicima. Mogu se koristiti jednostruki i dvostruki navodnici, te *backticks*.

Boolean predstavlja logički entitet i može imati samo dvije vrijednosti: *true* ili *false*. To su jako bitne vrijednosti pomoću kojih se može stvoriti složeni sustav petlji i uvjeta.

Null je specijalna vrijednost koja predstavlja “ništa“, “prazno“ ili “vrijednost nepoznata“. Varijabla kojoj nije dodijeljena vrijednost je undefined.

Number	5, 6.5, 7 etc
String	“Hello everyone” etc
Boolean	true or false
Null	represents null i.e. no value at all
Undefined	A variable that has not been assigned a value is undefined.
Symbol	used as an identifier for object properties.

Slika 1 Primitivni tipovi podataka

Ne primitivni tipovi podataka su:

- Objekti
- Nizovi
- Regularni izrazi

Objekt je najvažniji tip podataka i čini građevni blok za moderni JavaScript.

```
const person = {
      name: 'John',
      age: 25;
}
```

Kod 2 Primjer jednostavnog objekta

Svi ostali tipovi se zovu primitivni zato što njihova vrijednost može biti samo jedna stvar, kao string ili broj. Jednostavno rečeno, objekt se koristi kako bi se varijable povezale u jednu grupu.

Object	instance through which we can access members
Array	group of similar values
RegExp	represents regular expression

Slika 2 Ostali tipovi podataka

2.1.3 Funkcije i Klase

Funkcije su jedan od najinteresantnijih i najvažnijih elemenata svakog programskoj jezika. JavaScript funkcija je blok koda dizajniran da izvrši određen zadatak. Također, funkcije su temeljni građevni blokovi programa. One omogućavaju da se dio koda pozove više puta, bez ponovnog pisanja istog dijela koda. Pozivanje funkcije govori računalu da izvrši kod koji se nalazi u njoj, te se vrati na mjesto gdje se funkcija nalazi u programu. Postoji više načina definiranja funkcija u JavaScriptu, s malim razlikama u onom što rade.

```
function square(number) {
      return number * number;
}
```

Kod 3 Definiranje funkcije

`Function` je rezervirana riječ u koja služi za stvaranje funkcije. Nakon toga slijedi naziv funkcije koja se može nazvati proizvoljno. Unutar zagrada se nalaze parametri. Parametri su vrijednosti koje se šalju funkciji kada se pozove. Vitičaste zagrade predstavljaju početak funkcijskog bloka, a sve što se nalazi unutar vitičastih zagrada predstavlja tijelo funkcije. Unutar tijela funkcije se mogu kreirati varijable, dodati *if/else* petlje i još mnogo toga. `Return` je isto vrlo bitan i svaka funkcija ga treba imati. `Return` precizira vrijednost koju će funkcija vratiti.

Postoji i još jedna vrsta funkcija koje se zovu *arrow* funkcije. Riječ je o modernom načinu pisanja funkcija koji je uveden 2015 godine s ES6 verzijom JavaScript-a. Kada se u funkciji nalazi samo jedan `return` izraz i ništa drugo, to se može napisati u jednoj liniji.

```
const square = (number) => number * number;
```

Kod 4 Arrow funkcija

Klase su dijelovi koda koji obuhvaćaju višestruke objekte, metode i dopuštaju manipulaciju svojim varijablama i funkcijama članicama. Unutar svakog jezika klasa ima različitu sintaksu, a isto vrijedi i za JavaScript. Za izradu klase u JavaScript-u potrebno je obratiti pažnju na nekoliko stvari. Klasa se stvara pomoću ključne riječi `class`.

Klasa ima zadani konstruktor koji je prazan, ali se također može definirati unutar metode klase. Za razliku od drugih jezika, za deklariranje konstruktora, JavaScript ima specifičnu ključnu riječ koja se zove `constructor`. Parametri koji su dani konstruktoru se zatim koriste za definiranje elemenata unutar klase.

```
class Student{
      constructor(name) {
            this.name = name;
}
```

Kod 5 Stvaranje klase

Kod klasa razlikujemo još neke ključne riječi, kao što su `new` i `this`. Ključna riječ `new` ima više aspekata povezanih s njom, ali jednostavno možemo reći da se pomoću nje stvara novi objekt. Ključna riječ `this` odnosi se na trenutnu instancu te klase.

```
const student = new Student('John');
```

Kod 6 Stvaranje instance klase

2.1.4 Asinkroni JavaScript

Kako bi razumjeli što je to asinkrono izvršavanje, prvo treba spomenuti sinkrono. Sinkroni JavaScript je onaj gdje se kod izvršava liniju po liniju i njihovi zadaci su odmah dovršeni, odnosno nema vremenskog kašnjenja kod obavljanja zadataka tih linija koda. Kod asinkronog izvršavanja neke linije koda trebaju vremena da se izvrše. Ti zadaci se izvode u pozadini dok JavaScript nastoji izvršiti ostale linije koda. Kada rezultat asinkronih operacija postane dostupan, onda se koristi u programu.

```
const functionOne = () => {
      console.log('Function One'); //1
      functionTwo();
      console.log('Function One, Part 2'); //2
};
const functionTwo = () => {
      setTimeout(() => console.log('Function Two'), 2000); //3
}
functionOne();
// Function One
// Function One, Part 2
// (after two second delay) Funtion Two
```

Kod 7 Primjer koda koji se izvršava asinkrono

Dakle, glavni koncept iza asinkronog JavaScripta se temelji na tome da nije potrebno čekati funkciju da se izvrši, obavi svoj zadatak i vrati rezultat. Jednostavno, asinkronoj funkciji se prepušta da obavi svoj posao u pozadini, a program nastavlja izvršavati sljedeće linije koda i zatim upotrijebi rezultat tog asinkronog zadatka kada je dostupan.

3 Material UI

Material UI je *front-end* biblioteka za React komponente. Izgrađen je koristeći Less. Less (*Leaner Style Sheet*) je jezično proširenje kompatibilno s prethodnim verzijama CSS-a. Material UI se temelji na Google-ovom materijalnom dizajnu kako bi pružio kvalitetno digitalno iskustvo dok razvija *front-end* grafiku. Usredotočen je na pružanje gotovih komponenti koje korisnici mogu prilagoditi svojim aplikacijama [3]. Neke od prednosti korištenja Material UI su:

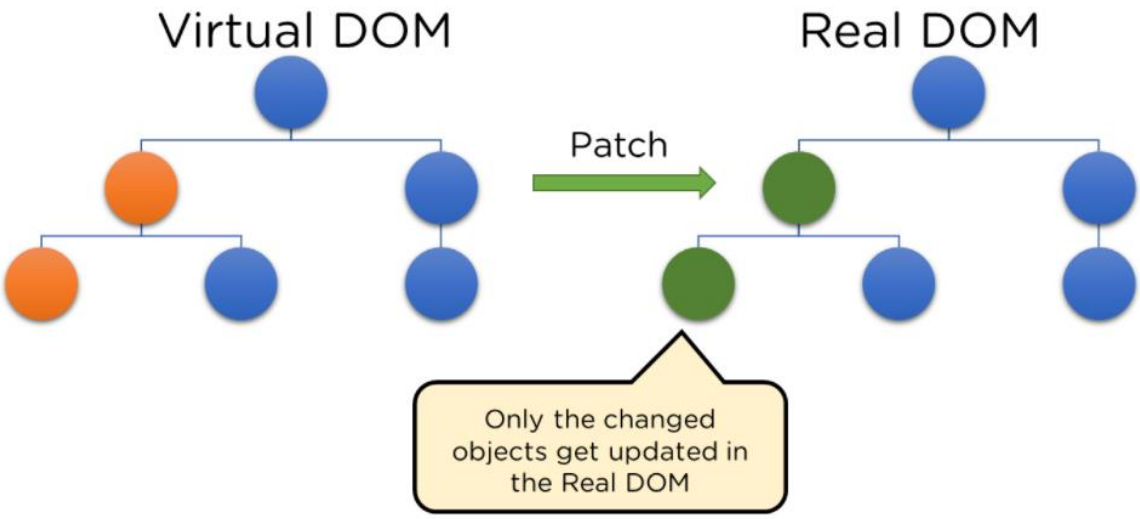
- Material UI ima detaljnu dokumentaciju koja olakšava razumijevanje elemenata, za razliku od nekih biblioteka koje nisu dobro dokumentirane, što otežava razvoj s njima
- Redovito se ažurira
- Komponente su dosljedne u dizajnu i bojama, što omogućuje aplikaciji da izgleda estetski privlačno

Slika 3 Primjer *button* komponente iz Material UI. Preuzeto iz https://mui.com/

4 React

React je JavaScript biblioteka koja služi za stvaranje deklarativnih korisničkih sučelja (engl. *user interface*) koristeći koncept temeljen na komponentama. Može se koristiti za web i mobilne aplikacije i glavni cilj React-a je biti brz, fleksibilan i jednostavan. Neke od glavnih karakteristika React-a su: virutalni DOM, JSX i React Native.

Virtualni DOM je React-ova verzija stvarnog DOM-a. Manipulacija stvarnim DOM-om je znatno sporija od manipulacije virtualnim DOM-om jer, kada se stanje objekta promijeni, virtualni DOM ažurira samo taj objekt u stvarnom DOM-u. DOM tretira XML ili HTML dokument kao strukturu stabla u kojoj je svaki čvor objekt koji predstavlja dio dokumenta. Međusobno djeluju na način da, kada se stanje objekta promijeni u React aplikaciji, virtualni DOM se ažurira. Zatim uspoređuje svoje prethodno stanje i ažurira samo one objekte u stvarnom DOM-u umjesto ažuriranja svih objekata. Zbog toga se stvari odvijaju brže, posebno u usporedbi s drugim *front-end* tehnologijama koje moraju ažurirati svaki objekt, čak i ako se promijeni samo jedan od njih u aplikaciji.



Slika 4 Funkcija virtualnog DOM-a. Preuzeto iz https://www.simplilearn.com/

React koristi JavaScript sintaktičko proširenje pod nazivom JSX za stvaranje elemenata. Odnosi se na kombinaciju HTML-a i JavaScript-a. JSX koristi *Babel* predprocesore za pretvaranje teksta nalik HTML-u u JavaScript datotekama u JavaScript objekte koji se analiziraju. React ne zahtijeva korištenje JSX-a, ali većina programera smatra da stvara bolje korisničko iskustvo unutar koda [4].

```
const element = <h1>Hello world</h1>;
```

Kod 8 Korištenje JSX-a u React-u

React Native je JavaScript okvir otvorenog koda (engl. *open source*) za izradu aplikacija na različitim platformama, kao što su iOS i Android. Temelji se na React-u i ima značajan utjecaj u razvoju mobilnih aplikacija. Koristi JavaScript za izradu korisničkog sučelja aplikacije ali također koristi izvorne prikaze operacijskog sustava.

4.1 Komponente

Komponenta je jedan od temeljnih građevnih blokova React-a. Svaka aplikacija koja se razvija u React-u sastojat će se od dijelova koji se nazivaju komponente, koje znatno olakšavaju zadatak izrade korisničkih sučelja. Tijekom razvoja, aplikacija se podijeli na više pojedinačnih dijelova koji se nazivaju komponente što omogućava rad na svakoj komponenti zasebno. Na kraju se sve komponente spoje u jednu nadređenu komponentu koja predstavlja konačno korisničko sučelje. Jednostavno, komponente u React-u vraćaju dio JSX koda koji govori što treba prikazati na zaslonu. U React-u se koriste dvije vrste komponenti:

- Funkcijske komponente
- Komponente klase

Funkcijske komponente su jednostavno JavaScript funkcije. Stvaraju se korištenjem ključne riječi `function` ili korištenjem *arrow* funkcija.

```
function Welcome() {
      return <h1>Hello World</h1>
}
```

Kod 9 Komponenta stvorena sa ključnom riječi function

```
const Welcome = () => <h1>Hello World</h1>
```

Kod 10 Komponenta stvorena pomoću *arrow* funkcije

Komponente klase su malo složenije od funkcijskih komponenti. Funkcijske komponente nisu svjesne drugih komponenti u programu, dok komponente klase mogu raditi jedna s drugom na način da omogućavaju prosljeđivanje podataka iz jedne komponente u drugu.

```
class Welcome extends React.Component {
render() {
return <h1>Hello World</h1>
}}
```

Kod 11 Komponenta stvorena pomoću klase

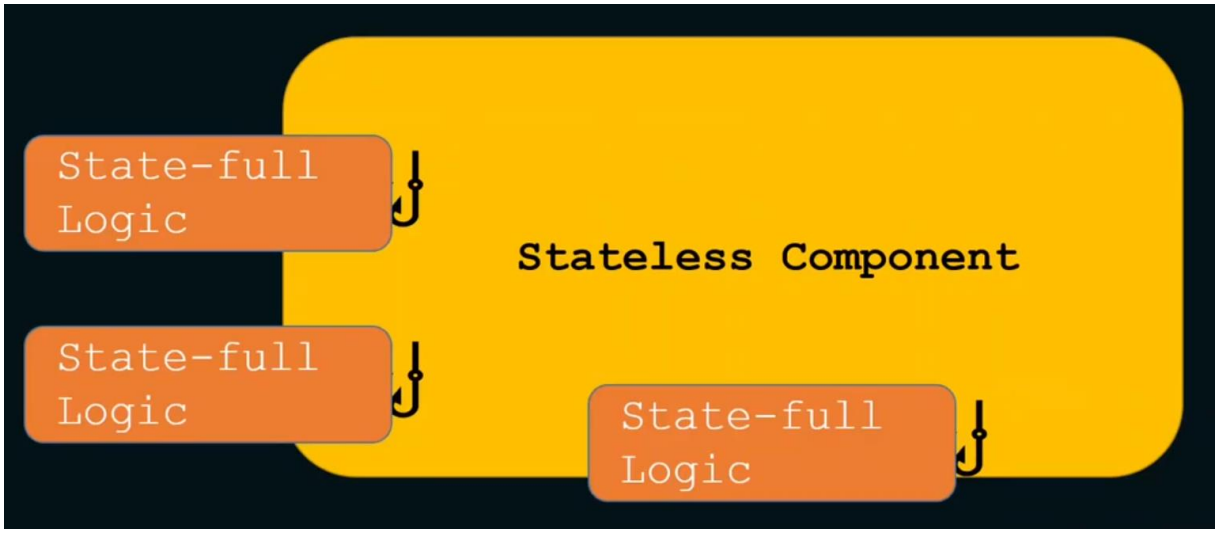
Također, bitno je spomenuti i način na koji se komponente renderiraju. Za renderiranje komponente u React-u potrebno je inicijalizirati element s korisnički definiranom komponentom i proslijediti taj element kao prvi parametar `React.DOM.render()` metode ili izravno proslijediti komponentu kao prvi argument metode.

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

const Welcome = () => {
      return <h1>Hello World</h1>
}
React.DOM.render(
      <Welcome />,
      Document.getElementById('root')
);
```

Kod 12 Renderiranje komponente

4.2 Hooks

Prije React verzije 16.8, programeri su mogli upravljati stanjem (engl. *state*) i drugim React značajkama koristeći samo komponente klase. Ali s verzijom 16.8, React je uveo novi obrazac pod imenom *hooks*. Uz React *hooks*, omogućeno je korištenje stanja i ostalih značajki u funkcijskim komponentama. *Hooks* su JavaScript funkcije koje upravaljaju stanjem i *side effects*-ima tako što ih izoliraju od komponente. Dakle, omogućuju *stateful* logiku u *hooks* koje se mogu koristiti u komponentama.



Slika 5 *Stateful* logika izoliranja u *hooks. Preuzeto iz* https://www.freecodecamp.org/news/react-hooks-fundamentals/

Stateful logika može biti bilo što što treba deklarirati i lokalno upravljati varijablom stanja. Na primjer, logika za dohvaćanje podataka i upravljanje podacima u lokalnoj varijabli je *stateful*. React *hooks* možemo jednostavno definirati kao JavaScript funkcije koje se mogu koristiti za izolaciju *reusable* dijela funkcijske komponente. *Hooks* mogu biti *stateful* i upravljati *side-effects*-ima [5]. React ima velik broj ugrađenih hooks, a neke od često korištenih su:

- useState
- useEffect
- useContext
- useReducer
- useCallback
- useMemo
- useRef

React također omogućava izradu prilagođenih (engl. *custom*) *hooks*. Prilagođene *hooks* su funkcije čiji naziv na početku mora sadržavati riječ use (kao i kod običnih *hooks*). Postavlja se pitanje zašto pored svih dostupnih *hooks* stvarati prilagođene. Na primjer, tijekom razvoja aplikacije potrebno je koristiti useState i useEffect u jednoj od komponenti. Nakon nekog vremena javi se potreba za korištenjem tih istih *hooks* u drugoj komponenti. Da ne dođe do dupliciranja koda, može se stvoriti prilagođena *hook* koja će se moći koristiti u svakoj komponenti. Dakle, jedna od prednosti je ta što se prilagođene *hooks* mogu koristiti na više mjesta, bez potrebe da se isti kod piše dva ili više puta. Također, stvara se čist kod jer se logika izdvoji iz komponente u *hook*. Lakše je i održavanje koda, jer ako je potrebno promijeniti logiku, promijeni se samo jednom unutar prilagođene *hook*, umjesto promjene unutar svake komponente.

```
import { useState, useEffect } from 'react';
const useFetch = (url) => {
      const [data, setData] = useState(null);
      useEffect(() => {
            fetch(url)
            .then((res) => res.json())
            .then((data) => setData(data));
      }, [url]);
      return [data]
};
```

Kod 13 Prilagođena *hook* za dohvaćanje podataka

4.3 Svojstva

Svojstva ili props se koriste u Reactu za prijenos podataka s jedne komponente na drugu (s roditeljske komponente na djecu komponente). Props je kratica za svojstva (engl. *properties*) i korisni su kada tijek podataka u aplikaciji treba biti dinamičan. Kako bi se props mogli koristiti, potrebno ih je poslati kao argument određenoj funkciji. Unutar roditeljske komponente se mogu postaviti već određene vrijednosti za svojstva, ali ti podaci mogu biti i dinamički. Nakon toga se svojstva s određenim nazivom mogu koristiti unutar komponente kojoj su proslijeđeni. Koriste se na način da se prvo navede parametar props i naziv određenog svojstva odvojen točkom. Također, postoji i drugi način. Riječ je o svojstvima s destrukturiranjem. Umjesto da komponenta prima parametar props, moguće je pomoću destrukturiranja, kao parametre, poslati nazive svojstava koje komponenta prima. To je korisno kako se ne bi uvijek trebala navoditi riječ props prije vrijednosti samog svojstva. Bitno je napomenuti da su props objekti nepromjenjivi. Njih se ne bi trebalo ažurirati unutar komponente kojoj su proslijeđeni. Oni su u vlasništvu komponente u kojoj su definirani.

```
import Tool from './Tool';
const App = () => {
      return(
            <div>
                  <Tool name='John' tool='React' />
            </div>
      )
}

const Toll = (props) => {
      return(
            <div>
                  <h1>My name is {props.name}</h1>
                  <p>My favorite development tool is {tool}.</p>
</div>
      )
}
```

Kod 14 Prosljeđivanje podataka komponenti koristeći props

4.4 Router

Na performanse web aplikacije izravno utječu stvari poput broja učitanih stranica na početku, količina dohvaćenih i prikazanih podataka i vrijeme potrebno za prebacivanje s jedne stranice na drugu. Renderiranje i usmjeravanje (engl. *routing*) na strani klijenta daju suptilni osjećaj performansi aplikacije i bitno je iskoristiti sve njihove značajke. Usmjeravanje je proces kroz koji korisnik dolazi do različitih stranica koje se nalaze u web aplikaciji, a renderiranje je proces prikazivanja tih stranica na korisničkom sučelju. Svaki put kada se zatraži ruta do određene stranice, ta stranica se renderira, ali nije svako renderiranje rezultat rute.

React renderira odgovarajuće informacije o DOM-u koristeći strukturu svoje komponente. Usmjeravanje na strani klijenta u React pomaže u održavanju besprijekornog korisničkog iskustva koje obećava tipična SPA aplikacija. To se postiže putem vanjske React biblioteke koja se zove React Router. React Router koristi dinamičko usmjeravanje kako bi osigurao da se usmjeravanje postiže onako kako korisnik to traži. To također znači da se sve potrebne komponente renderiraju bez ponovnog učitavanja stranice.

```
import React from 'react';
import {BrowserRouter, Switch, Route } from 'react-router-dom';
const App = () => {
      return(
            <BrowserRouter>
                  <Switch>
                  <Route exact path='/'>Početna</Route>
                  <Route exact path='/onama>O nama</Route>
                  <Route exact path='/kontakt>Kontakt</Route>
                  </Switch>
            </BrowserRouter>
      )
}
```

Kod 15 Primjer korištenja React *router*-a

Ovaj primjer prikazuje tri komponente između kojih React Router vrši usmjeravanje. `Switch` element prolazi kroz rute i traži prvi koji se podudara sa onim kojeg je korisnik odabrao. Na taj način korisnik daje routeru do znanja na koju komponentu treba usmjeriti korisnika. Postoji i `Link` element unutar kojeg se određuje ruta na koju će se router usmjeriti, dok se unutar `Route` elementa definiraju komponente koje će biti prikazane. Atribut `path` označava koji link vodi do određene komponente [6].

5 Redux

Redux je biblioteka za upravljanje i ažuriranje stanja aplikacije koristeći događaje koji se nazivaju akcije. Služi kao centralno spremište za stanje koje se koristi u cijeloj aplikaciji, s pravilima koja osiguravaju da se stanje može ažurirati samo na predvidljiv način. Redux je dobro koristiti jer pomaže kod upravljanja globalnim stanjem, odnosno stanjem koje je potrebno mnogim dijelovima aplikacije. Obrasci i alati koje nudi Redux olakšavaju razumijevanje kada, gdje, zašto i kako se stanje ažurira u aplikaciji te kako će se logika aplikacije ponašati kada dođe do tih promjena. Pomaže u pisanju koda koji je predvidljiv i koji se može testirati, što daje povjerenje da će aplikacija raditi prema očekivanjima.

Redux pomaže pri upravljanju dijeljenim stanjem, ali kao i svaki alat, ima kompromise. Potrebno je naučiti više koncepata i napisati više koda. Također dodaje neke upute kodu i traži da se slijede određena ograničenja. To je kompromis između kratkoročne i dugoročne produktivnosti. Redux je koristan kada:

- postoje velike količine stanja aplikacije koje su potrebne na mnogim mjestima u aplikaciji
- dolazi do čestog ažuriranja stanja aplikacije tijekom vremena
- logika ažuriranja tog stanja može biti složena
- aplikacija ima srednju ili veliku kodnu bazu na kojoj može raditi mnogo ljudi

Riječ je o maloj i samostalnoj JavaScript biblioteci, no obično se koristi s nekoliko drugih paketa. Redux se može integrirati s bilo kojim UI okvirom, a najčešće se koristi s React-om. React-Redux omogućava React komponentama interakciju s Redux skladištem kao što je čitanje dijelova stanja i slanje akcija za ažuriranje skladišta.

Redux Toolkit je preporučeni pristup za pisanje Redux logike. Sadrži pakete i funkcije koje su bitne za izradu Redux aplikacije. Također, ugrađuje najbolje prakse, pojednostavljuje većinu Redux zadataka, sprječava uobičajene pogreške i olakšava pisanje Redux aplikacija [7].

Redux je zasnovan na tri principa:

- skladište (engl. *state*)
- reduktori (engl. *reducers*)
- akcije (engl. *actions*)

5.1 Akcije

Akcija je običan JavaScript objekt koji ima `type` polje. Akcija se može zamisliti kao događaj koji opisuje nešto što se dogodilo u aplikaciji. Polje `type` bi trebalo biti string koji radnji daje opisni naziv. Taj string se obično piše kao "domain/eventName", gdje je prvi dio značajka ili kategorija kojoj ova akcija pripada, a drugi dio je specifična stvar koja se dogodila. Objekt akcije može imati druga polja s dodatnim informacijama o tome što se dogodilo. Prema konvenciji, te informacije se stavljaju u polje zvano `payload`.

```
import { useDispatch } from 'react-redux';
const incrementHandler = () => {
      dispatch({ type: 'increment' })
}

const increaseHandler = () => {
      dispatch({ type: 'increase', payload: 5 })
}
```

Kod 16 Redux ackija

Kreator akcije (engl. *action creator*) je funkcija koja stvara i vraća objekt akcije. Obično se koriste kako ne bi trebalo svaki put ručno pisati akcijski objekt [7].

5.2 Reduktor

Reduktor je funkcija koja prima trenutno stanje i akcijski objekt, odlučuje kako ažurirati stanje ako je potrebno i vraća novo stanje. Reduktor se može zamisliti kao slušatelj događaja koju rukuje događajima na temelju vrste primljene akcije. Naziv su dobile po tome jer su slične *callback* funkciji koja se prosljeđuje `Array.reduce()` metodi. Reduktori uvijek moraju slijediti određena pravila:

- Trebali bi odrediti novu vrijednost stanja samo na temelju argumenata stanja i akcije.
- Nije im dopušteno mijenjati postojeće stanje. Umjesto toga, moraju izvršiti nepromjenjiva ažuriranja kopiranjem postojećih stanja i izmjenama kopiranih vrijednosti.
- Ne smiju raditi nikakvu asinkronu logiku, izračunavati nasumične vrijednosti ili uzrokovati druge nuspojave

Logika unutar reduktora obično slijedi isti niz koraka. Prvo je potrebno provjeriti mari li reduktor za određenu radnju. Ako je tako, napravit će kopiju stanja, ažurirati je novim vrijednostima i vratiti je. U suprotnome će vratiti postojeće stanje nepromijenjeno.

U kodu ispod se nalazi primjer reduktora koji pokazuje korake koji bi svaki od njih trebao slijediti. Riječ je o reduktoru koji povećava stanje brojača za neki iznos, ovisno o vrsti akcije [7].

```
const counterReducer = (state = { counter: 0 }, action) => {
      if(action.type === 'increment') {
            return {
                  counter: state.counter + 1
            }
      }
      if(action.type === 'increase') {
            return {
                  counter: state.counter + action.payload
            }
      }
      return state
}
```

Kod 17 Redux reduktor

5.3 Skladište

Trenutno stanje Redux aplikacije nalazi se u objektu koji se zove skladište. Pohrana se stvara prosljeđivanjem reduktora unutar metode `createStore`. Nakon toga potrebno je okružiti temeljnu komponentu *provider*-om i kao parametar poslati skladište. Tada će svaka komponenta moći pristupiti skladištu.

```
import { createStore } from 'redux';
const store = createStore(counterReducer);

import { Provider } from 'react-redux';
import ReactDOM from 'react-dom/client';
const root =
ReactDom.createRoot(document.getElementById('root'))
root.render(<Provider store={store}><App /></Provider>)
```

Kod 18 Redux skladište

Redux skladište ima i metodu koja se zove `dispatch`. Jedini način ažuriranja stanja je pozivanje `dispatch` metode i prosljeđivanje akcijskog objekta. Skladište će pokrenuti svoju

reduktorsku funkciju i spremiti novu vrijednost stanja unutra, te se može pozvati `getState()` za dohvaćanje vrijednosti. Radnje otpremanja služe za pokretanje događaja u aplikaciji. Reduktori čekaju na događaje i kada vide radnju koja ih zanima, ažuriraju stanje kao odgovor [8].

```
store.disptach({ type: 'counter/increment' })

console.log(store.getState())
// {value: 1}
```

Kod 19 Dohvaćanje vrijednosti iz skladišta nakon akcije

5.4 Redux Toolkit

Paket Redux Toolkit je namijenjen da bude standardan način pisanja Redux logike. Izvorno je stvoren kako bi pomogao u rješavanju tri uobičajene brige o Reduxu:

- Konfiguracija Redux skladišta je prekomplicirana
- Potrebno je dodati mnogo paketa da bi Redux učinio nešto korisno
- Redux zahtijeva previše standardnog koda

Ne mogu se riješiti svi problemi, no Redux Toolkit pokušava pružiti neke alate koji apstrahiraju proces postavljanja i bave se najčešćim slučajevima upotrebe, kao i uključiti neke korisne uslužne programe koji će korisniku omogućiti da pojednostavi aplikacijski kod. Također uključuje mogućnost dohvaćanja podataka i predmemoriranja koja se zove Redux Toolkit (skraćeno RTK) *query*. Uključen je u paket kao zaseban skup ulaznih točaka. Nije obavezan za korištenje, no može eliminirati potrebu samostalnog pisanja logike za dohvaćanje podataka.

Pomoću Redux-a se odlučuje kako se želi upravljati svime, poput postavljanja skladišta, što sadrži stanje i kako izgraditi reduktore. To je u nekim slučajevima dobro jer daje fleksibilnost, ali ta fleksibilnost nije uvijek potrebna. Ponekad se želi započeti rad na najjednostavniji mogući način, s dobrim zadanim ponašanjem. Cilj Redux Toolkit-a je pomoći u pojednostavljenju uobičajenih slučajeva korištenja Redux-a. Nije namijenjen da bude potpuno rješenje za sve što se može raditi s Redux-om, ali bi trebao pojednostaviti kod koji je povezan s njim [9]. Uključuje različite API-je (engl. *Application Programming Interface*), kao što su:

- configureStore()
- createReducer()
- createAction()

- createSlice()
- createAsyncThunk
- createEntityAdapter
- createSelector

5.4.1 Slice

Kako bi kreirali odsječak (engl. *slice*) potrebno je koristiti `createSlice` funkciju. Riječ je o funkciji višeg reda koja kao parametre prima početno stanje, objekt sa reduktorskim funkcijama i naziv odsječka. U Redux Toolkit-u, `createSlice` funckija pomaže stvoriti isječak Redux skladišta. Funkcija je jako korisna jer i automatski generira kreatore radnji i tipove akcija koji odgovaraju reduktorima i stanju. To se postiže jer interno koristi `createAction` i `createReducer`. Zbog toga nije potrebno raditi *if* provjere kako bi se saznalo o kojoj je akciji riječ. To se obavi automatski i smanjuje potrebu za pisanjem tog koda. Također pruža jednaku funkcionalnost kao i Redux samo što se ista funkcionalnost postigne uz manje koda koji je jednostavniji i čitljiviji.

```
import { createSlice } from '@reduxjs/toolkit';

export const genreOrCategory = createSlice({
  name: 'genreOrCategory',
  initialState: {
    genreIdOrCategoryName: '',
    page: 1,
    searchQuery: '',
  },
  reducers: {
    selectGenreOrCategory: (state, action) => {
      state.genreIdOrCategoryName = action.payload;
      state.searchQuery = '';
    },
    searchMovie: (state, action) => {
      state.searchQuery = action.payload;
    },
  },
});
```

Slika 6 Kreiranje Redux Toolkit slice-a

Još jedna stvar koju Redux Toolkit olakšava je mijenjanje stanja u reduktorima. Ovdje je dozvoljeno manipulirati trenutnim stanjem, što je prije bilo zabranjeno. Odnosno, i dalje nije dozvoljeno manipuliranje stanjem, ali dobra stvar kod `createSlice` funkcije je što nemože doći do toga jer Redux Toolkit interno koristi drugi paket. Riječ je o paketu *imgur* koji će detektirati takav kod i automatski će klonirati postojeće stanje, stvoriti novi objekt stanja, sačuvati sve unutar objekta što ne mijenjamo i *override*-at sve što želimo promijeniti [9].

5.4.2 Skladište

Za razliku od klasičnog Reduxa gdje se za stvaranje skladišta koristi metoda `createStore`, kod Redux Toolkit-a se za navedeno koristi `configureStore` metoda. To je metoda koja stvara skladište i čini spajanje više reduktora u jedan jednostavnije. Metoda prima konfiguracijski objekt unutar kojeg se postavi *reducer property* i riječ je o očekivanom *property*-u za `configureStore` metodu.

```
import { configureStore } from '@reduxjs/toolkit';

import { tmdbApi } from '../services/TMDB';
import genreOrCategoryReducer from '../features/currentGenreOrCategory';
import userReducer from '../features/auth';

export default configureStore({
  reducer: {
    [tmdbApi.reducerPath]: tmdbApi.reducer,
    currentGenreOrCategory: genreOrCategoryReducer,
    user: userReducer,
  },
});
```

Slika 7 Kreiranje Redux Toolkit skladišta

5.4.3 Akcije

Redux Toolkit čini otpremanje (engl. *dispatch*) akcija vrlo jednostavnim. Automatski stvori jedinstvene akcijske identifikatore za različite reduktore.

```
genreOrCategory.actions.searchMovie()
```

Kod 20 Pristup akciji u Redux Toolkit-u

Kod iznad pokazuje na koji način se može pristupiti određenim akcijama. `Actions` je objekt koji sadrži različite ključeve, gdje nazivi ključeva odgovaraju nazivima metoda koji se nalaze u `createSlice` funkciji, odnosno nazivima koji se nalaze unutar reduktora. Možemo pristupiti tim ključevima koje sadrži akcijski objekt, no s tim se ne pristupa metodama koje se

nalaze u reduktoru, nego metodama koje Redux Toolkit automatski stvori. Te metode će, kada se pozovu, stvoriti akcijski objekt za nas i zato se te metode zovu kreatori akcije. Svaki od tih akcijskih objekata će već sadržavati *type property* sa jedinstvenim identifikatorom po akciji. Stoga se mi, kao developeri, ne trebamo brinuti o stvaranju akcijskih objekata i smišljanju jedinstvenih identifikatora za svaku akciju.

```
import React, { useState } from 'react';
import { TextField, InputAdornment } from '@mui/material';
import { Search as SearchIcon } from '@mui/icons-material';
import { useDispatch } from 'react-redux';
import { useLocation } from 'react-router-dom';

import { searchMovie } from '../../features/currentGenreOrCategory';
import useStyles from './styles';

const Search = () => {
  const [query, setQuery] = useState('');
  const classes = useStyles();
  const dispatch = useDispatch();
  const location = useLocation();

  const handleKeyPress = (event) => {
    if (event.key === 'Enter') {
      dispatch(searchMovie(query));
    }
  };
```

Slika 8 *Dispatch* Redux Toolkit akcije unutar komponente

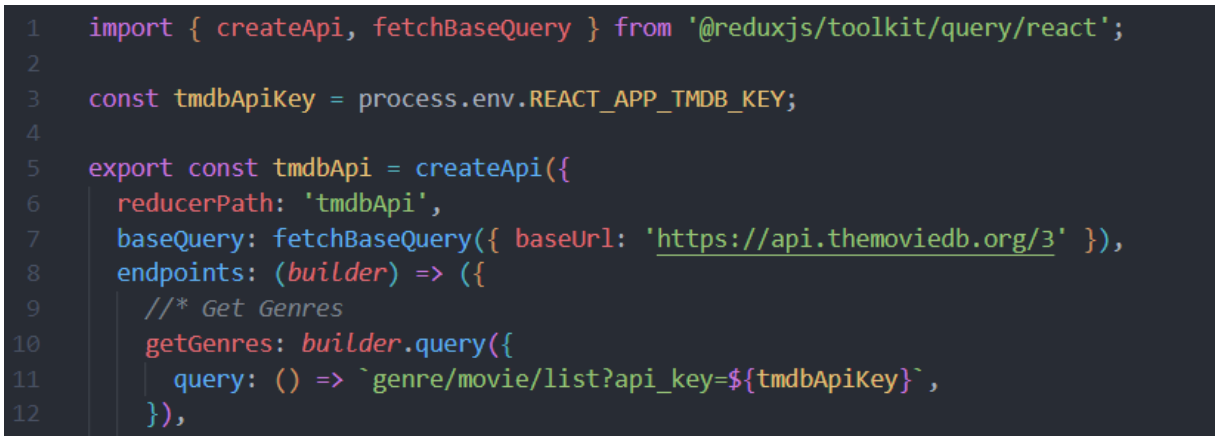
5.4.4 RTK Query

Redux Toolkit Query je moćan alat za dohvaćanje podataka i predmemoriranje. Osmišljen je da pojednostavi uobičajene slučajeve učitavanja podataka u web aplikaciji, eliminirajući potrebu da sami ručno pišete logiku za dohvaćanje podataka i predmemoriranja. Riječ je o izbornom dodatku koji je uključen u Redux Toolkit paket, a njegova funkcionalnost je izgrađena na vrhu ostalih API-ja u Redux Toolkit-u. Web aplikacije obično trebaju dohvatiti podatke s poslužitelja kako bi ih prikazale. Također obično trebaju izvršiti ažuriranja tih podataka, poslati ta ažuriranja poslužitelju i održavati predmemorirane podatke na klijentu sinkroniziranim s podacima na poslužitelju. RTK Query crpi inspiraciju iz drugih alata koji su bili rješenja za dohvaćanje podataka, kao što su Apollo Client, React Query, Urql i SWR, ali dodaje jedinstveni pristup svom API dizajnu:

- Logika dohvaćanja podataka i predmemoriranja izgrađena je na temelju `createSlice` i `createAsyncThunk` API-ja
- Budući da Redux Toolkit ne ovisi o korisničkom sučelju, funkcionalnost RTK upita može se koristiti s bilo kojim slojem korisničkog sučelja
- API krajnje točke definirane su unaprijed, uključujući kako generirati parametre upita iz argumenata i transformirati odgovore za predmemoriju
- RTK Query također može generirati React *hooks* koje enkapsuliraju cijeli proces dohvaćanja podataka, pružaju `data` i `isLoading` polja komponentama i upravljaju vijekom trajanja predmemoriranih podataka dok se komponente montiraju i demontiraju
- RTK Query je u potpunosti napisan u TypeScript-u i osmišljen je da pruži izvrsno iskustvo korištenja TypeScript-a

Za kreiranje API odsječka koristi se `createApi` metoda koja je srž funkcionalnosti RTK Query-a. Omogućuje definiranje skupa krajnjih točaka koje opisuju kako dohvatiti podatke iz pozadinskih API-ja, uključujući konfiguraciju kako dohvatiti i transformirati te podatke. Generira strukturu API odsječka koja sadrži Redux logiku (i po izboru React *hooks*) koja enkapsulira proces dohvaćanja podataka i predmemoriranja. Obično bi trebali imati samo jedan API odsječak po osnovnom URL-u s kojim vaša aplikacija treba komunicirati. Na primjer, ako web lokacija dohvaća podatke iz */api/posts* i */api/users*, imali bi jedan API odsječak s */api/* osnovnim URL-om i zasebne definicije krajnjih točaka za posts i users. To omogućuje učinkovito iskorištavanje prednosti automatiziranog ponovnog dohvaćanja. Osnovni parametri koje createApi prihvaća su:

- reducerPath
- baseQuery
- endpoints



```
import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react';

const tmdbApiKey = process.env.REACT_APP_TMDB_KEY;

export const tmdbApi = createApi({
  reducerPath: 'tmdbApi',
  baseQuery: fetchBaseQuery({ baseUrl: 'https://api.themoviedb.org/3' }),
  endpoints: (builder) => ({
    //* Get Genres
    getGenres: builder.query({
      query: () => `genre/movie/list?api_key=${tmdbApiKey}`,
    }),
```

Slika 9 Kreiranje Redux Toolkit Api-ja

`ReducerPath` je jedinstveni ključ na koji na koji će vaša usluga biti podignuta u skladištu. Ako se `createApi` pozove više puta unutar aplikacije, potrebno je za svaki od njih navesti jedinstvenu vrijednost. Zadana vrijednost je 'api'.

`BaseQuery` je osnovni upit koji koristi svaka krajnja točka (engl. *endpoint*). RTK Query izvozi pomoćni program pod nazivom `fetchBaseQuery` unutar kojeg je moguće postaviti osnovni URL koji će koristiti svaka krajnja točka.

Krajnje točke su samo skup operacija koje želimo da se izvedu na serveru. Definiraju se kao objekt i postoje dvije osnovne vrste krajnjih točaka, a to su *query* i *mutation*. Query može biti funkcija koja vraća ili string ili objekt koji se prosljeđuje baseQuery.

API odsječak sadrži automatski generirani reduktor koji je potrebno dodati u Redux skladište. Na kraju je moguće uvesti automatski generirane *hooks* iz API odsječka u komponente gdje je potrebna i pozvati *hook*-s sa svim potrebnim parametrima. RTK Query će automatski dohvatiti podatke pri postavljanju, ponovno dohvatiti kada se parametri promijene, pružiti { `data`, `isFetching`} vrijednosti u rezultatu i ponovno prikazati komponentu ovisno o tome kako se te vrijednosti mijenjaju [10].

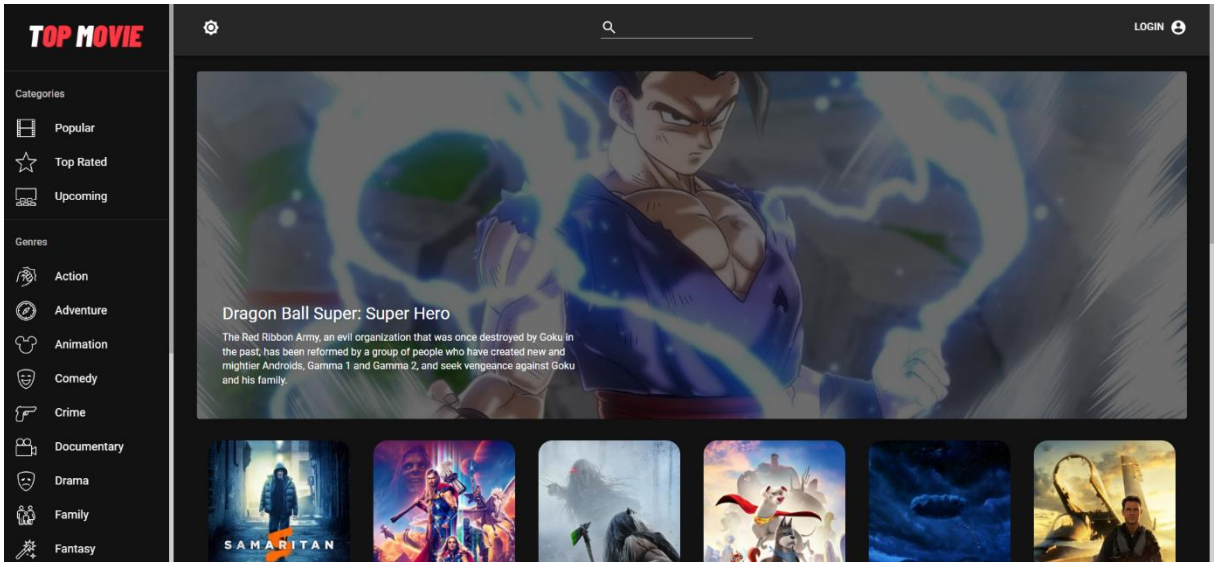
```
import React, { useState } from 'react';
import { Box, CircularProgress, useMediaQuery, Typography } from '@mui/material';
import { useSelector } from 'react-redux';

import { useGetMoviesQuery } from '../../services/TMDB';
import { FeaturedMovie, MovieList, Pagination } from '..';

const Movies = () => {
  const [page, setPage] = useState(1);
  const { genreIdOrCategoryName, searchQuery } = useSelector((state) => state.currentGenreOrCategory);
  const { data, error, isFetching } = useGetMoviesQuery({ genreIdOrCategoryName, page, searchQuery });
  const lg = useMediaQuery((theme) => theme.breakpoints.only('lg'));
```

Slika 10 Dohvaćanje podataka sa kreiranog Api-ja

6 Oblikovanje i izrada aplikacije

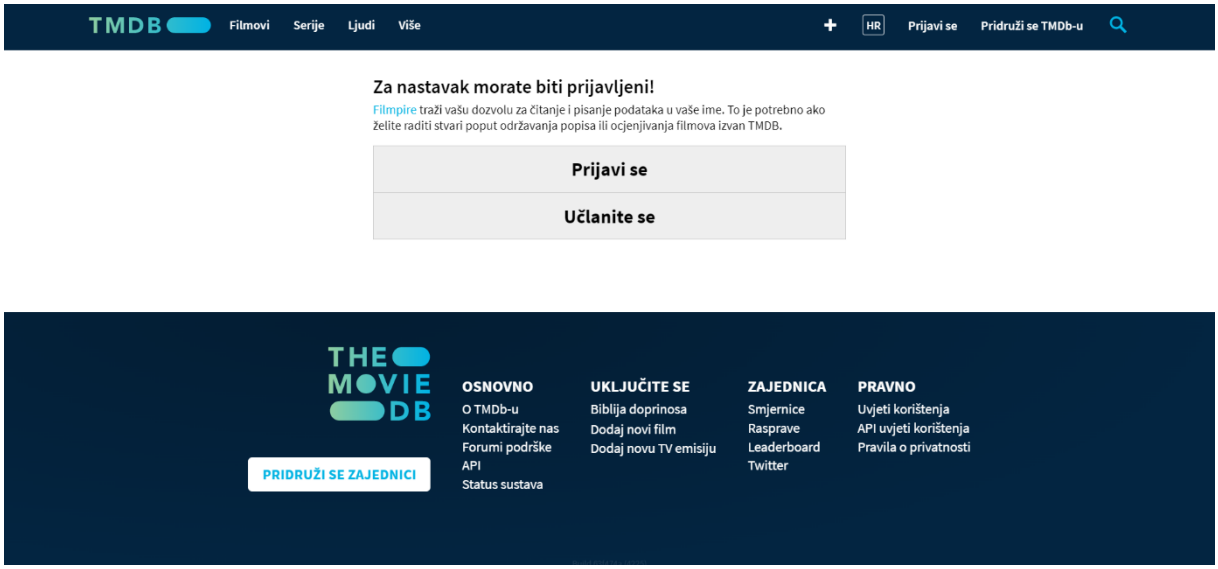
Projekt koji sam izradio kao podlogu i podršku ovom radu je web aplikacija kreirana u React programskom okviru. Riječ je o aplikaciji koja omogućava korisniku pretraživanje i pregled filmova. Filmove je moguće pretraživati po kategorijama, žanrovima i nazivu. Aplikacija koristi i Redux Toolkit kako bi prosljeđivanje podataka različitim komponentama bilo što jednostavnije. Za izradu aplikacije koristio sam i The Movie Database (skraćeno TMDB) API. To je API s kojeg dohvaćam sve filmove i serije, omogućava pretraživanje po kategorijama i žanru, pregled detalja o filmu i glumcima te još mnogo toga. Još jedna funkcionalnost je prijava u aplikaciju. Nakon što se korisnik prijavi dostupne su mu akcije kao dodavanje sadržaja u favorite ili listu gledanja. Cijeli proces prijave i registracije ide preko TMDB API.

Prije rada na projektu sam krenuo sa razvojem vještina u React-u s kojim sam već imao iskustva. Nešto novo što sam trebao savladati je Redux Toolkit. Nakon istraživanja sam shvatio da je dosta jednostavniji od „klasičnog“ Redux-a, te da pruža bolju funkcionalnost. Za implementaciju tog dijela sam se poslužio različitim videima i dokumentacijom koja je detaljno opisana na internetu. Ulaskom u aplikaciju otvara se početna stranica. Sastoji se od više elemenata. Možemo vidjeti opcije filtriranja po različitim kategorijama i logo, koji vodi na početnu stranicu. Zatim tu je dio sa filmskim sadržajem, opcija pretraživanja po nazivu, tamni način rada i gumb koji vodi na stranicu za stvaranje korisničkog profila. Filmski sadržaj se sastoji od jednog istaknutog dijela gdje je riječ o filmskom sadržaju koji je prvi na popisu API-a s kojeg dohvaćamo podatke. Ostatak sadržaja se nastavlja ispod istaknutog dijela.



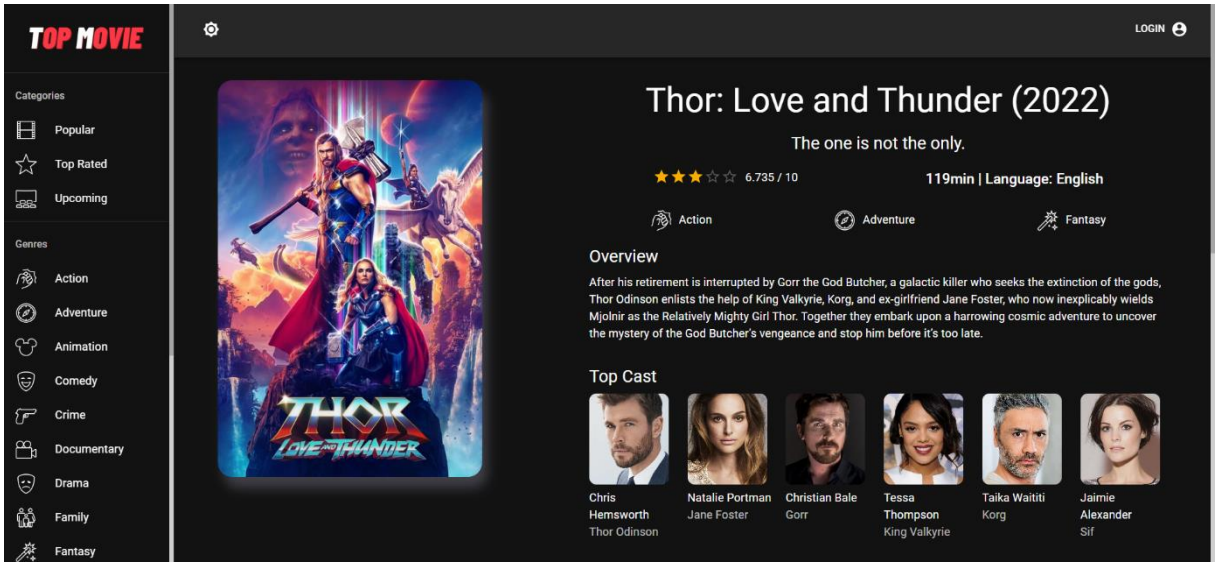
Slika 11 Početna stranica aplikacije

Klik na Log In gumb nas vodi na TMDB stranicu gdje je potrebno kreirati novi račun ili se prijaviti u račun koji već postoji. Nakon toga će korisnik moći dodavati filmove u favorite i listu gledanja.



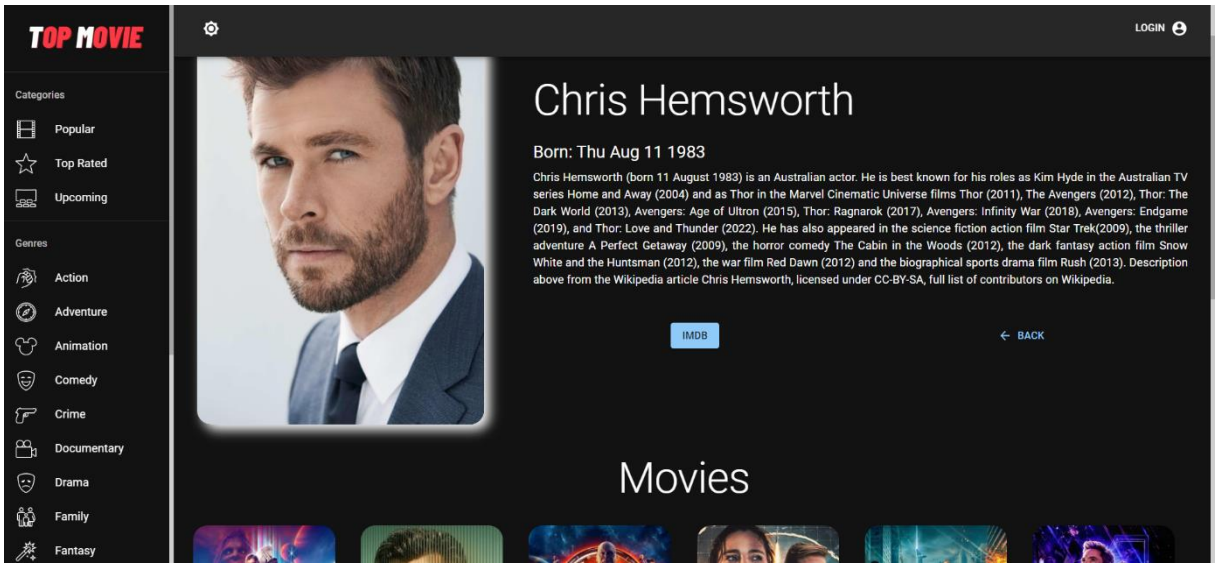
Slika 12 Kreiranje računa ili prijava u već postojeći

Aplikacija omogućava i pregled detalja o svakom filmu. Kikom na određeni film otvara se stranica koja prikazuje različite informacije kao što su naziv, ocjena, jezik i trajanje, kratak pregled, glumačka postava, linkovi na web stranicu, imdb stranicu i poveznicu na trailer. Na kraju se nalazi dio sa preporučenim sadržajem ako je korisnik zainteresiran za neki specifični film.



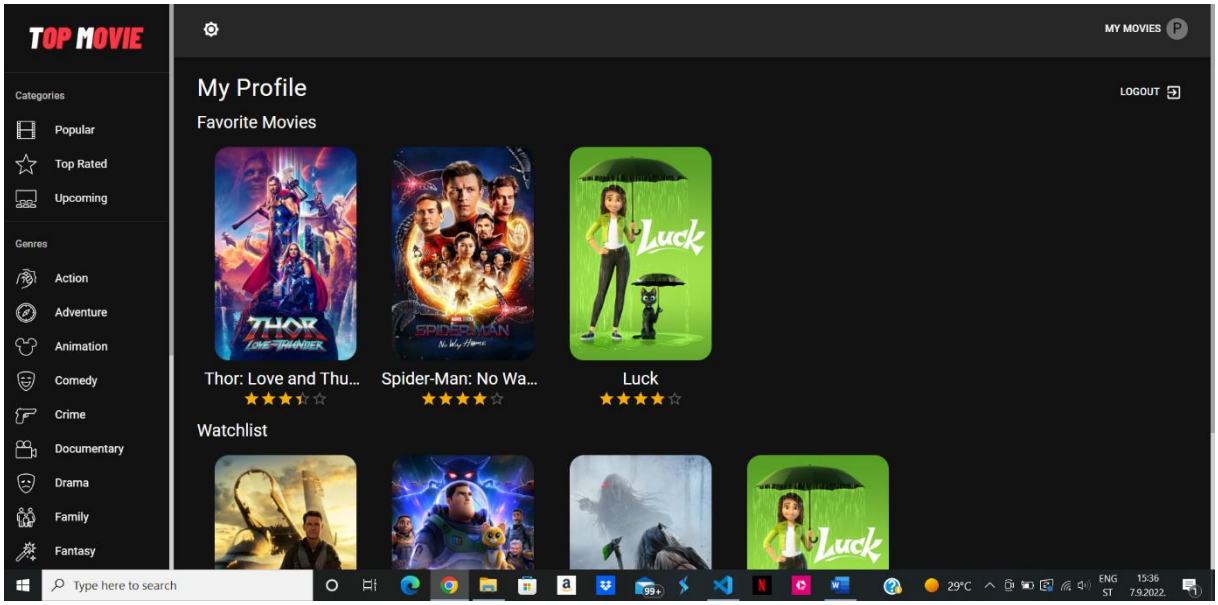
Slika 13 Detalji o odabranom filmu

Klikom na određenog glumca idemo na sljedeću stranicu koja prikazuje detalje o tom glumcu. Detalji se odnose na ime i prezime, datum rođenja, kratak opis karijere te link imdb stranicu tog glumca. Također TMDB Api omogućava i dohvaćanje svih filmova u kojem je određeni glumac sudjelovao, stoga postoji i taj dio.



Slika 14 Detalji o odabranom glumcu

I na kraju, nakon prijave u aplikaciju, korisnik ima mogućnost pregleda svih filmova koje je dodao u favorite ili listu gledanja. To je vrlo korisno za ovu aplikaciju jer joj je glavna namjena pretraživanje filmova i spremanje istih u jednu od ove dvije kategorije kako bi korisnik kasnije mogao vidjeti što je točno odlučio gledati i koji su mu filmovi omiljeni.



Slika 15 Pregled filmova koji su dodani u favorite i listu gledanja

7 Zaključak

U ovome radu je opisan jedan od najpopularnijih JavaScript okvira za razvoj web aplikacija, React. Istaknuti su različiti elementi i prednosti, a jedna od najvećih je brzo i jednostavno kreiranje aplikacija. Opisano je kreiranje komponenata koje se sastoje od kombinacije HTML-a i JavaScript-a, odnosno od JSX-a koji govori što komponenta treba prikazati na zaslonu. Također, opisane su različite *hooks* koje se koriste i način prosljeđivanja podataka komponentama. Podaci se mogu prosljeđivati putem props-a ili koristeći Redux biblioteku. Redux se koristi kako bi se izbjeglo prosljeđivanje podataka svakoj komponenti zasebno koristeći svojstva. Sastoji se od centralnog skladišta podataka koje je dostupno svim komponentama. Kada je nekoj komponenti potreban određen podatak, samo treba pristupiti skladištu i odabrati dio koji joj treba. To čini protok podataka u React mnogo jednostavnijim i dinamičnijim. Osim „klasičnog“ Reduxa, u radu je opisan i Redux Toolkit. Riječ je o skupu alata koji omogućavaju da se pojednostavni Redux razvoj. Uključuje pomoćne programe za stvaranje i upravljanje Redux skladištem, kao i pisanje Redux akcija i reduktora.

Na kraju ovog rada prikazan je React u praksi, odnosno kreirana je web aplikacija koja služi za pregled filmskog sadržaja. Aplikacija ima različite mogućnosti, kao filtriranje po različitim kategorijama i žanrovima, prikaz detaljnih informacija o filmu i glumcima, registraciju i dodavanje sadržaja u favorite ili listu gledanja, te pretraživanje sadržaja po nazivu. Aplikacija je primjer kako zapravo iskoristiti sav teorijski sadržaj pri stvaranju stvarnog projekta i koji je najbolji način primjene svih navedenih koncepata.

Virtualni svijet je bitan element današnjeg društva i možemo reći da ljudi na neki način ovise o internetu jer većina zadataka se danas može obaviti uz par klikova miša. Sa sve bržim razvojem virtualnog svijeta javlja se potreba za što boljim i naprednijim web aplikacijama. U prošlosti su sve web stranice bile statičke, gdje je korisnik mogao samo dobiti određene informacije, bez ikakve interakcije. Današnje web aplikacije se temelje na dinamičnosti i interakciji sa korisnikom. Također, brzina je bitan element jer korisnici više ne toleriraju sporo učitavanje ili spore reakcije aplikacije nakon određene akcije. Tu nam može pomoći React.

Zaključno, React je vrlo moderan i često korišten programski okvir koji omogućuje kreiranje kompleksnih, ali i brzih web aplikacija i kojeg koriste neke od najvećih svjetskih kompanija za razvoj svojih usluga.

Literatura

[1] »Web application (Web app),« [Mrežno]. Available: https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app.

[2] »Complete Path to JavaScript Mastery,« [Mrežno]. Available: https://www.jsmastery.pro/complete-path-to-javascript-mastery.

[3] S. Khan, »What is Material UI in React?,« [Mrežno]. Available: https://www.educative.io/answers/what-is-material-ui-in-react.

[4] C. Deshpande, »The Best Guide to Know What is React,« 18 July 2022. [Mrežno]. Available: https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs.

[5] T. Adhikary, »React Hooks Fundamentals for Beginners,« 15 March 2022. [Mrežno]. Available: https://www.freecodecamp.org/news/react-hooks-fundamentals/.

[6] G. Singhal, »Pros and Cons of Clinet-side Routing with React,« 4 April 2020. [Mrežno]. Available: https://www.pluralsight.com/guides/pros-and-cons-of-client-side-routing-with-react.

[7] »Redux Essentials, Part 1: Redux Overview and Concepts,« [Mrežno]. Available: https://redux.js.org/tutorials/essentials/part-1-overview-concepts.

[8] R. Wieruch, Taming the State in React, 2017.

[9] »Getting Started with Redux Toolkit,« [Mrežno]. Available: https://redux-toolkit.js.org/introduction/getting-started.

[10] »RTK Query Overview,« [Mrežno]. Available: https://redux-toolkit.js.org/rtk-query/overview.

[11] R. Wieruch, The Road to React, 2021.

[12] A. Sirotka, »What is React?,« 18 February 2022. [Mrežno]. Available: https://flatlogic.com/blog/what-is-react/.

[13] Z. Sajjad, »Smarter Redux with Redux Toolkit,« 8 April 2022. [Mrežno]. Available: https://blog.logrocket.com/smarter-redux-redux-toolkit/.