Metode za upravljanje stanjem React komponenti

Kurevija, Ivan

Undergraduate thesis / Završni rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Split, University of Split, Faculty of science / Sveučilište u Splitu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:166:089160

Rights / Prava: Attribution-NonCommercial-NoDerivatives 4.0 International/Imenovanje-Nekomercijalno-Bez prerada 4.0 međunarodna

Download date / Datum preuzimanja: **2024-07-27**

Repository / Repozitorij:

Repository of Faculty of Science

zir.nsk.hr

UNIVERSITY OF SPLIT

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U SPLITU

PRIRODOSLOVNO MATEMATIČKI FAKULTET

ZAVRŠNI RAD

METODE ZA UPRAVLJANJE STANJEM REACT KOMPONENTI

Ivan Kurevija

Split, rujan 2021.

SVEUČILIŠTE U SPLITU

PRIRODOSLOVNO MATEMATIČKI FAKULTET

ZAVRŠNI RAD

METODE ZA UPRAVLJANJE STANJEM REACT KOMPONENTI

Student:

Ivan Kurevija

Mentor:

prof. dr. sc. Divna Krpan

Split, rujan 2021.

Temeljna dokumentacijska kartica

Završni rad

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

METODE ZA UPRAVLJANJE STANJEM REACT KOMPONENTI

Ivan Kurevija

Sažetak: React je jedna od najpopularnijih biblioteka u svijetu programiranja internetskih aplikacija. Jedan od razloga za tu popularnost je konstantno unaprjeđivanje i dodavanje novih funkcionalnosti. Jedna od takvih se krije pod imenom Hook. Hook-ovi su metode koje zamjenjuju klase, olakšavaju promjenu i upravljanje stanjem, te doprinose responzivnosti aplikacije. Najbolji su saveznik za uštedu vremena i koda svakom web programeru. U ovom radu, Hook-ovi su fokus, uz analizu i ostalih tehnologija korištenih u mom projektu.

Ključne riječi: react, hook, upravljanje stanjem, web aplikacija, internet trgovina, mern

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

Rad sadrži: 34 stranice, 24 grafička prikaza i 10 literaturnih navoda.

Izvornik je na hrvatskom jeziku.

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

Neposredni voditelj: Dr.sc. Divna Krpan, viši predavač Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

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

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

Divna Krpan, viši predavač Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad prihvaćen: **rujan 2021.**

Basic documentation card

Bachelor Thesis

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

METHODS FOR MAINTAINING STATE OF REACT COMPONENTS

Ivan Kurevija

Summary: React is one of the most popular libraries in the world of web application programming. One of the reasons for this amount of popularity is the constant improvement and addition of new functionalities. One of these is hidden under the name Hook. Hooks are methods that replace classes, make it easier to change and manage the state, and contribute to application responsiveness. They are the best ally for saving time and code for any web developer. In this thesis, Hooks are the focus, and I analyzed other technologies needed to create a web application, which were also use in this project.

Key words: react, hook, state management, web application, internet store, mern

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

Thesis consists of: 34 pages, 24 pictures and 10 references.

Original language: Croatian

Supervisor: **Goran Zaharija, Ph.D.** Assistant Professor of Faculty of Science, University of Split

Co-Supervisor: **Divna Krpan**, **Ph.D.** Senior Lecturer of Faculty of Science, University of Split

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

Thesis accepted: **September 2021.**

Zahvala:

Zahvaljujem se svojim roditeljima i bratu koji su mi pružali bezuvjetnu podršku tijekom trajanja mog preddiplomskog studija.

Zahvalu upućujem i svojoj voditeljici dr. sc. Divni Krpan na odličnom mentorstvu, potpori i svim savjetima koji su mi pomogli u izradi ovog rada.

Također želim zahvaliti i prijateljima, kolegama koji su učinili moje studiranje posebnim razdobljem u životu i što su mi uvijek bili podrška.

Sadržaj

1 UVOD 2

2 MERN Stack tehnologije 3

2.1 React 3

2.2 MongoDb 4

2.3 Node Js 6

2.3.1 NPM 7

2.4 Express 10

2.4.1 Rutiranje 10

2.4.2 Posrednici 12

2.4.3 Zahtjev/odgovor i veza sa posrednicima 13

3 React Hooks 15

3.1 useState Hook 16

3.2 useEffect Hook 18

3.3 useContext Hook 20

3.4 Izrada vlastitih hook-ova 21

4 Oblikovanje i izrada aplikacije 23

4.1 Aplikacija PMF Web Shop 24

5 ZAKLJUČAK 28

6 Literatura 29

7 Popis slika 30

1 UVOD

Internet je u našem dobu postao jedna od osnovnih ljudskih potreba. Omogućava nam povezanost i komunikaciju s cijelim svijetom, bez obzira gdje se nalazimo. Uz neograničene mogućnosti, javlja se i potreba za konstantnim unaprjeđenjem brzine i usluge internetskih stranica i platformi. Pandemija coronavirusa je također stvorila neke nove navike, pogotovo vezano za internet kupovinu te rad i školovanje od kuće. Tako je proširen raspon korisnika ovakvih internetskih usluga i sigurno neće opadati.

U izradi i programiranju rješenja za navedene zahtjeve, najčešće se bira programski jezik Javascript. Iskombiniran sa HTML[1]-om i CSS[2]-om omogućava programiranje kreativnih rješenja. Pošto živimo u brzom i dinamičnom svijetu u kojemu svakodnevno imamo sve veće brzine internetskih usluga, očekujemo da tu brzinu prate i internetske stranice i aplikacije. Oku ugodne i fluidne aplikacije nerijetko puno bolje prolaze na tržištu. React biblioteka odlično odgovara na ove zahtjeve. Ova Javascript biblioteka otvorenog koda služi za izradu interaktivnih korisničkih sučelja koji znatno ubrzavaju aplikaciju i doprinose fluidnosti, pa i općem vizualnom dojmu.

Ovaj završni rad sastoji se od teoretskog i praktičnog rada. Praktični rad se odnosi na web aplikaciju internetske trgovine. Za izradu su korištene slijedeće tehnologije: React, Mongo, Express, React, Node. Aplikacija je programirana u Javascriptu, a za dizajn su korišteni: CSS, HTML, Bootstrap i React Bootstrap. U ovom teoretskom radu opisat će se korištene tehnologije, njihove prednosti, mane i primjene, kao i izrađena aplikacija. Analizirat će se glavni aspekti aplikacije uz priložene dijelove programskog koda.

Uz sve navedeno, fokus rada se stavlja na React hooks[3]. Ova važna značajka iz React biblioteke će mi biti okosnica rada i na primjerima ću opisati važnost i velike prednosti korištenja hook-ova u programskom kodu.

[1] HTML – Prezentacijski jezik za izradu web stranica (HyperText Markup Language)
[2] CSS – Stilski jezik (Cascading Style Sheets)
[3] Hook- metoda za upravljanje stanjem React komponenti

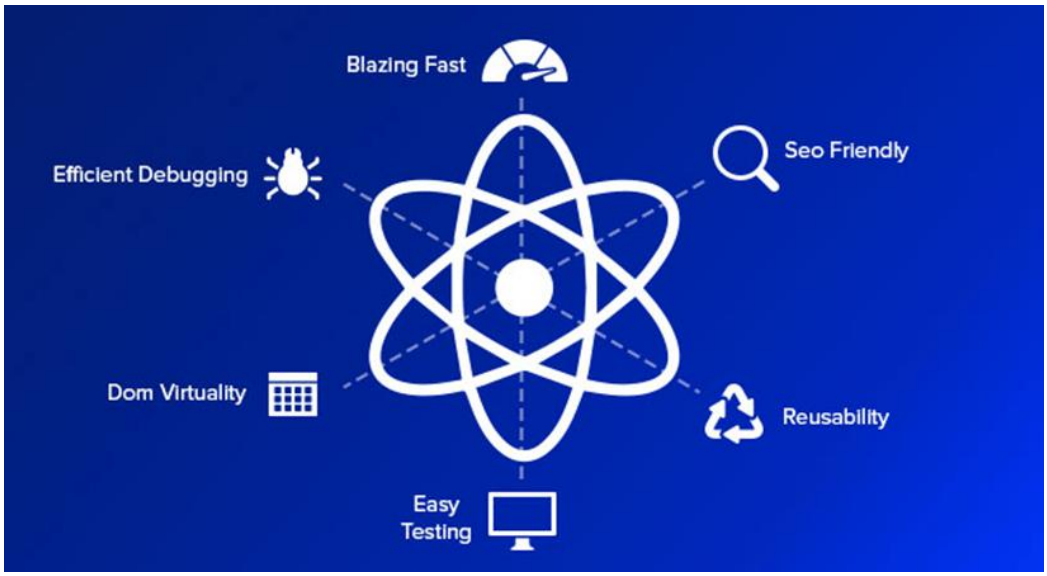
2 MERN Stack tehnologije

MERN stack je akronim za skup tehnologija koje su korištene za izradu aplikacije. To su redom: Mongo, Express, React i Node. Opis i razrada ovih tehnologija će poslužiti kao dobar uvid u funkcioniranje modernih internetskih aplikacija.

2.1 React

React je besplatna Javascript biblioteka otvorenog koda. Koristi se za izradu korisničkih sučelja. Razvio ga je Jordan Walke, jedan od programera u Facebook-u [1]. Nastao 2011. i od tada mu strelovito raste popularnost, te je postao jedna od najkorištenijih biblioteka Javascripta. React ima ulogu u prijenosu podataka modelu objekta dokumenta(poznatiji kao DOM[4]), te zbog toga aplikacije programirane uz pomoć React-a često zahtijevaju preuzimanje dodatnih biblioteka za rutiranje i održavanje stanja. Zbog toga se javlja potreba za korištenjem Redux-a i React Router-a. Redux se koristi za rutiranje, dok React Router, kako mu i ime kaže, služi za rutiranje.

Najbolja odlika React-a je ta da programeri mogu mijenjati podatke, a da se u tom slučaju ne mora osvježavati stranica. Ovo čini aplikaciju bržom i responzivnijom, te je jedan od vodećih razloga za popularnost ove biblioteke.



Slika 1 Glavne karakteristike React-a[5]

[4] DOM – Document Object Model
[5] Izvor: https://teknotrait.com/wp-content/uploads/2016/07/All-about-react-JS.png

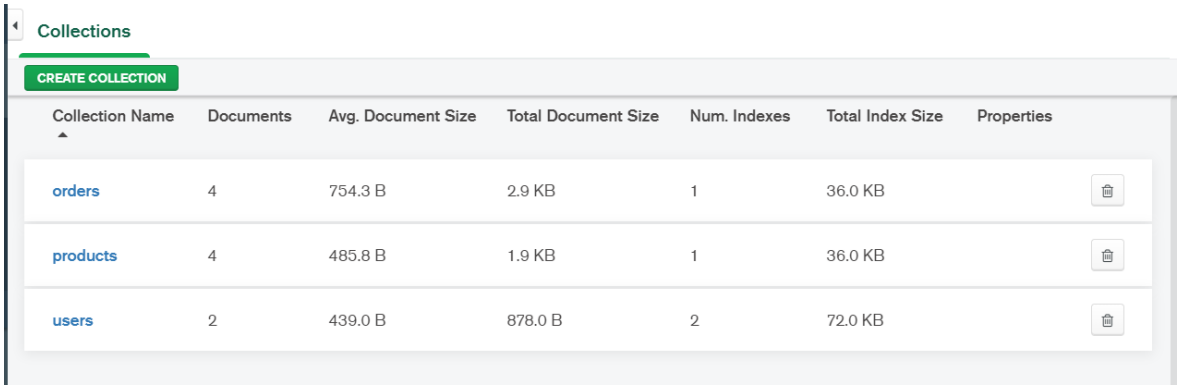
Najvažnijim karakteristikama se smatraju: JSX, Virtual DOM i React native.

JSX je ekstenzija koja omogućava podešavanje modela objekta dokumenta u HTML-u. Pojednostavljuju ažuriranje istog i time puno utječu na brzinu i efikasnost aplikacije. Kod složenijih aplikacija Virtual DOM predstavlja kopiju originalnog DOM-a koji prati koji su dijelovi promijenjeni prije samog ažuriranja. React Native je dobro poznat programerima mobilnih aplikacija. Sadrži iste principe kao i React, što pruža velik broj funkcija za korisnička sučelja.

Ogromna prednost React-a je puno funkcionalnosti, lako učenje uz potrebno prethodno iskustvo u Javascriptu, HTML-u i CSS-u. Dao je novu dimenziju internet aplikacijama i stranicama, te je preporuka svakom programeru koji želi početi sa razvojem internet aplikacija da se usmjeri na učenje ovih tehnologija.

2.2 MongoDb

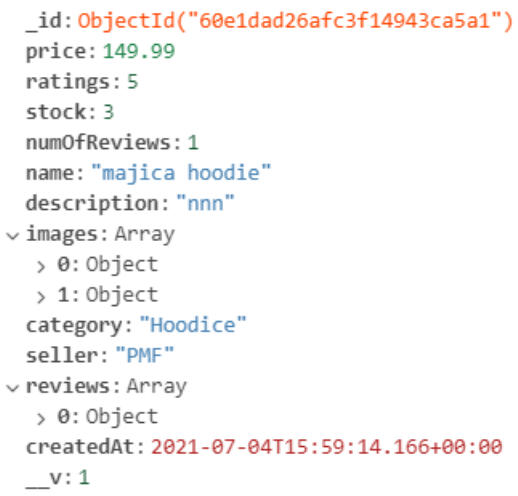
Mongo Db spada pod nerelacijski tip baza podataka. Nerelacijske baze podataka nemaju upite i ne spremaju podatke u tablicu. Podaci se kod nerelacijskih baza spremaju na različite načine. Kod Mongo baza podataka, spremljeni su u obliku JSON[6] objekata. Svaka baza se sastoji od kolekcija, koje u sebi sadrže dokumente.



Slika 2 MongoDb Kolekcije

Pošto su podaci zapisani u obliku JSON objekata, mogu sadržavati razne vrste podataka i lako se distribuiraju u različite sustave. JSON je jako univerzalan i brz, te pridodaje popularnosti korištenja Mongo baza među programerima.

[6] JSON (JavaScript Object Notation) je tekstualni format dizajniran za čitljiv i razumljiv prijenos strukture podataka ljudima i strojevima.



Slika 3 Primjer proizvoda spremljenog u Mongo bazu podataka u obliku JSON objekta

Mongo se također može pohvaliti i svojom prirodom bez sheme. Kod relacijskih baza potrebno je na vrijeme definirati strukturu svih podataka koji se pohranjuju. Podaci kod Mongo baze ne slijede unaprijed kreiranu shemu, osim ako je programer ne definira na aplikacijskoj razini. Ovo pogoduje programerima koji često rade izmjene podataka.

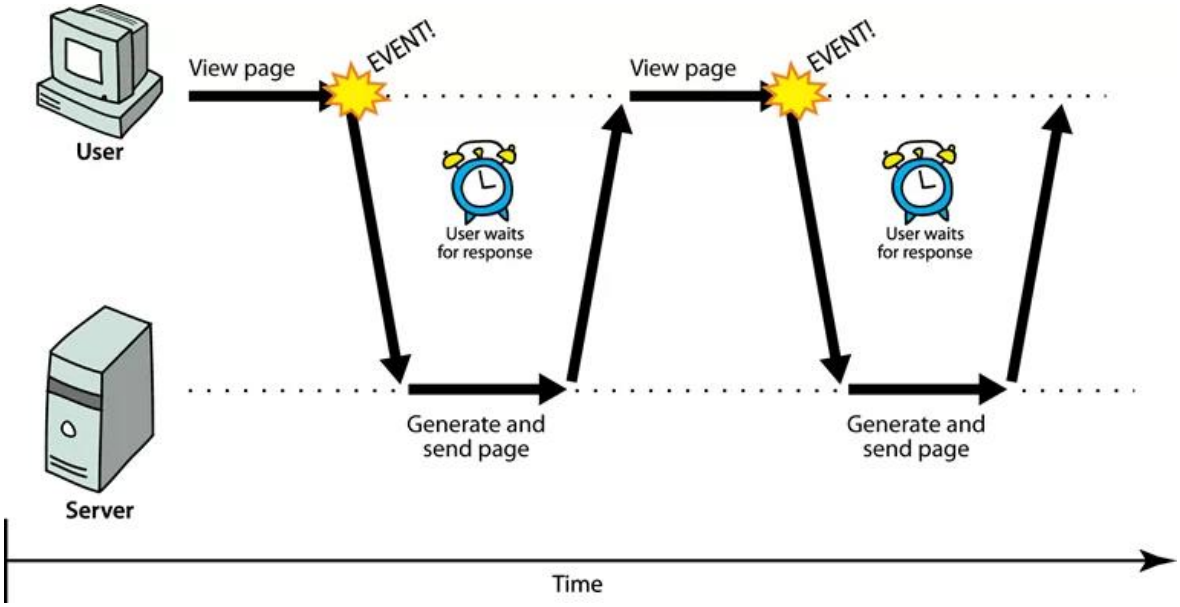
MongoDb sam vodi brigu oko uravnoteženja podataka i učitavanju preko klastera. Također vodi raspodjelu dokumenata te čitanje i pisanje usmjeri na ispravan poslužitelj i tako olakšava rad i omogućava brže ostvarivanje upita.

Uz čitanje, pisanje, ažuriranje i brisanje Mongo podržava i dodatne funkcije: indeksiranje, agregaciju, datotečni sustav i posebne kolekcije. Sve navedene funkcije ne utječu na brzinu rada. Mongo radnu memoriju koristi koliko može tako da nikad neće preopteretiti sustav.

Jedna od loših strana MongoDb-a je ta što ne podržava transakcije. Transakcija predstavlja logičku, atomsku jedinicu rada koja sadrži jedan ili više SQL izraza. Sve manje aplikacija zahtijeva transakcije, ali još uvijek postoje one kojima su potrebne transakcije za ažuriranje više dokumenata/zbirki. Ako je to nužna funkcija za vaš tim, MongoDb se ne bi trebao koristiti. Postoji mogućnost oštećenja podataka [2]. Udruživanje dokumenata je također jedna od mana Mongo baza, no programeri rade na tom problemu. Naime izvlačenje podataka iz više kolekcija zahtjeva brojne upite, a to dovodi do neurednog koda. Treba paziti sa indeksiranjem, jer sa loše izvedenim indeksiranjem, baza će davati spore i trome odgovore.

2.3 Node Js

Node se definira kao serverska platforma pogonjena V8 Javascript Engine-om. U programiranju se često javlja problem dugog čekanja odgovora na neki zahtjev, na primjer čekanje na dohvat podataka iz baze.



Slika 4 Proces učitavanja web stranice nakon nekog događaja[7]

Popularno rješenje za ovakav problem je raspodjela izvršavanja na *niti* (eng. threads[8]). Konkretno kod Javascript-a se uvijek izvršava samo jedna nit. Kod izvršavanja sporijih operacija, kao što je čitanje iz baze, program neće čekati, već će ići na izvršavanje slijedeće linije koda. Kada se I/0[9] operacija vrati, pokreće se callback, te se na taj način procesuira rezultat. Node Js nam nudi jednostavan, asinkroni, događajima pogonjen model programiranja za izradu modernih i brzih web rješenja.

NodeJs odvodi javascript izvan preglednika i izvršava ga. Omogućuje nam izvršavanje JS koda izvan preglednika. Također uklanja ograničenja koja dolaze s izvršavanjem JS -a u pregledniku. Omogućuje nam pristup datotečnom sustavu. Može nam pomoći u izgradnji jakih, sigurnih i skalabilnih web aplikacija.

[7] Izvor: https://www.popwebdesign.net/popart_blog/wp-content/uploads/2015/06/synchronous.png
[8] Thread - hrv. dretva ili nit, služi za podjelu programa na više dijelova radi jednostavnijeg i bržeg izvršavanja
[9] I/O – ulaz/izlaz

Danas je NodeJs najpopularnija tehnologija koja se koristi za izradu pozadine web aplikacije. Također se može koristiti za izradu RESTful API -ja.

Node Js je najbolje koristiti u aplikacijama koje imaju značajniju količinu podataka i kojima su podaci okosnica. Primjer takvih aplikacija su video streaming servisi, aplikacije za komunikaciju, internetske trgovine itd.

Za razvoj aplikacija koje intenzivno koriste procesorsku snagu, Node nije dobar izbor. Za takve aplikacije najbolje je koristiti drugu tehnologiju kao što su Django, Flask, Ruby on Rails itd. Čvor nije dobar za aplikacije koje zahtijevaju intenzivniju obradu na poslužiteljskoj strani. Primjeri aplikacija s intenzivnim korištenjem procesora su aplikacije za manipulaciju slikama, aplikacije za pretvorbu videa, aplikacije za kompresiju videa.

2.3.1 NPM

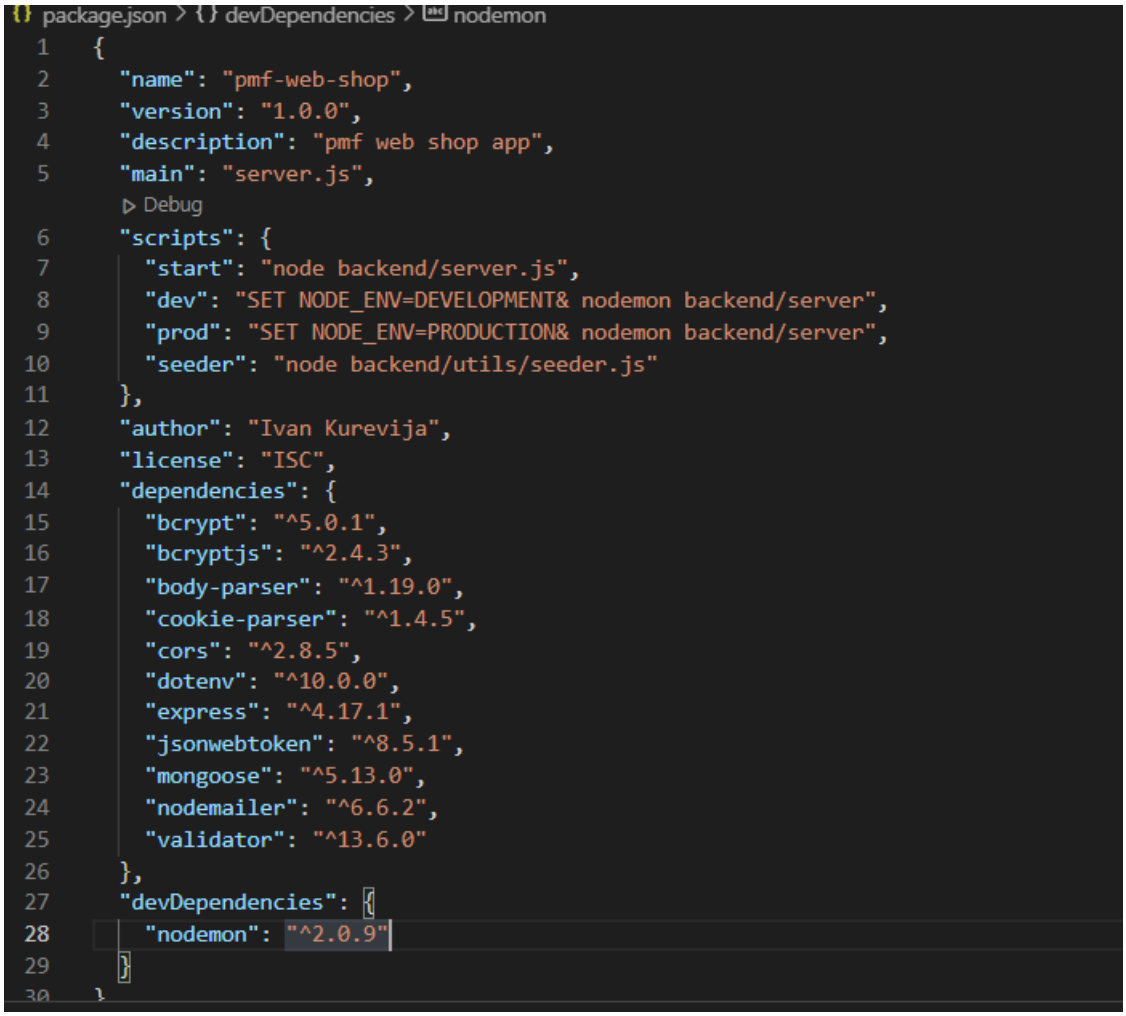
Kratica npm označava "upravitelj paketa čvorova". Kada se radi na nekom JavaScript projektu, može se koristiti npm za instaliranje paketa kodova drugih ljudi u svoj projekt.

Npm je alat koji se instalira direktno na računalo, te je svaki paket također pohranjen lokalno, na što treba paziti prilikom premještanja datoteka. Za instaliranje, deinstaliranje ili ažuriranje paketa koristit se npm iz naredbenog retka. Pod "paketom" se smatra bilo koji dio koda koji je netko odlučio objaviti na npm.

Neki primjeri npm paketa:

- Angular
- React
- jQuery
- Socket.io
- Express

Za upotrebu npm paketa u projektu, on treba sadržavati datoteku pod nazivom package.json. Ova datoteka čuva popis svih paketa koje koristite i koju ste verziju svakog od njih odabrali.



Slika 5 Popis NPM paketa iz mog projekta

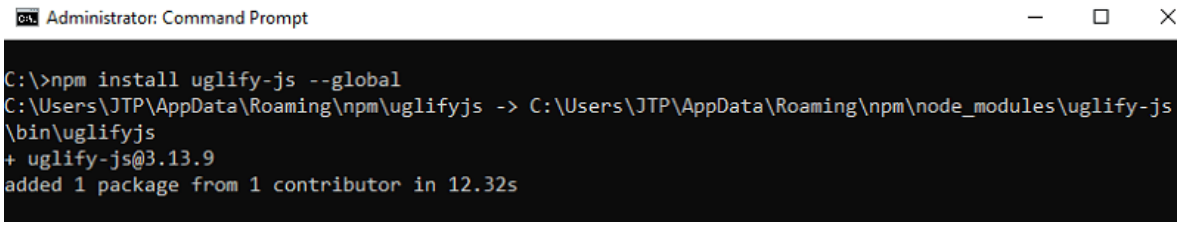
Na slici 5 vidimo kako izgleda package.json datoteka, ona sadrži nazive i broj verzije određenog paketa. U slučaju da projekt ne sadrži package.json datoteku ona se može kreirati tako što se u terminal upiše naredba npm init.

Svaki put kad se otvori projekt na novom računalu, bit će potrebno ponovo instalirati sve pakete. To je posebno istaknuto kad programeri koriste različite operacijske sustave (npr. Mac i Windows računala). Razlog je što svi paketi nisu međusobno kompatibilni te je potrebno preuzeti verziju koja odgovara operacijskom sustavu koji se koristi.

Kad se prvi put otvori projekt na drugom računalu, potrebno je ponovo instalirati pakete upisivanjem naredbe u terminalu: *npm install*

Nakon install naredbe u mapi projekta se stvara mapa pod nazivom node_modules. Ta mapa sadrži kodove svih paketa, te nerijetko zna biti masivna.

Novi npm paket se dodaje upisivanjem i pokretanjem slijedeće naredbe u terminalu: npm install ime_paketa –save(Slika 6).



Slika 6 Primjer instaliranja paketa[10]

Neke od prednosti korištenja npm paketa:

- Automatski generira datoteku package-lock.json. Korisno je posvetiti se sustavu kontrole verzija. Na taj način drugi programeri mogu lako instalirati ovisnosti na svoje lokalne strojeve
- S lakoćom se upravlja lokalnim ili globalnim ovisnostima
- Npm je dobro opremljen za rukovanje s više verzija ovisnosti
- Ima službeni registar koji ima više paketa nego pypi, rubygems ili packagist

Najbitnije npm naredbe:

- npm i: instalacija lokalnog paketa
- npm i -g: instalacija globalnog paketa
- npm un: brisanje lokalnog paketa
- npm up: ažuriranje paketa
- npm t: testiranje
- npm ls: ispis instaliranih paketa

Npm je odlična funkcionalnost Node Js-a koja pojednostavljuje izradu zahtjevnih i velikih projekata, te su ovakvi sustavi s paketima značajno unaprijedili programiranje, jer su omogućili kopiranje ponavljajućih komada kodova. Time su omogućili programerima da se posvete kompliciranijim konceptima i algoritmima, pa imamo sve naprednije i brže aplikacije.

[10] Izvor: https://static.javatpoint.com/blog/images/what-is-npm25.png

2.4 Express

Kada govorimo o Express-u, govorimo o Javascript okviru. Besplatan je i otvorenog koda. Express nam pomaže u rutiranju te povezivanju posrednika[11] sa http zahtjevom. Upotrebljava se i za kreiranje JSON API-ja[12]. Pruža nam mnoštvo alata kao što su: handleri, postavke i uslužne programe.

Serveri uobičajeno izvršavaju zadaće poput: slušanja zahtjeva koji dolazi na server, prima http zahtjeve na krajnjoj točki i strukturira informacije poslane sa zahtjevom, odgovara zahtjevu HTTP-a[13] i pokreće kod, te generalno odgovara na zahtjev. Sve ove zadaće uz alate koje nam Express nudi, postignu se u samo nekoliko linija koda.

Tri su koncepta s kojima radi Express a to su:

1. Rutiranje
2. Posrednici
3. Zahtjev/odgovor

2.4.1 Rutiranje

Cilj rutiranja je opisati koji dio koda bi se trebao pokrenuti kao odgovor na zahtjev, kojeg je zaprimio server. Opisuje se na način koji je zasnovan na kombinaciji URL[14] uzorka i HTTP metode povezane sa zahtjevom[3].

Primjer kako radi rutiranje: ako imamo GET zahtjev za /Početna, vratit će nam početnu stranicu. Neki od ostalih zahtjeva koje server primi: PUT, DELETE, POST, itd.

[11] Posrednici su još poznatiji kao middlewares
[12] API – aplikacijsko programsko sučelje
[13] HTTP - Request/response protokol
[14] URL – usklađeni lokator sadržaja

```
19
20
21
22  router.route('/products').get(getProducts);
23  router.route('/product/:id').get(getSingleProduct);
24
25
26
27  router.route('/admin/product/new').post(isAuthenticatedUser,authorizeRoles('admin'),newProduct);
28
29  router.route('/admin/product/:id').put(isAuthenticatedUser,authorizeRoles('admin'),updateProduct);
30
31  router.route('/admin/product/:id').delete(isAuthenticatedUser,authorizeRoles('admin'),deleteProduct);
32
33
34  router.route('/review').put(isAuthenticatedUser, createProductReview);
35  router.route('/reviews').get(isAuthenticatedUser, getProductReviews);
36  router.route('/reviews').delete(isAuthenticatedUser, deleteReview);
37
38
39
40  module.exports=router;
41
42
43
```

Slika 7 Rutiranje

Na slici 7 možemo vidjeti kako se stvaraju rute. Programski kod prvo uvozi Express aplikacijski objekt, pa ga koristi da dođe do objekta rutera (Slika 8). Onda koristeći GET metodu pridružuje objektima navedene rute. Nakon svega navedenog modul izvozi router objekt (Slika 7).

```
backend > routes > JS product.js > ...
1   const express=require('express')
2   const router=express.Router();
3
```

Slika 8 Uvoz objekata

Da bi koristili modul rutera moramo prvo postaviti zahtjev za router modulom pozivanjem require(). Kada pozovemo metodu use() ruter se dodaje u putanju posrednika i tako stvara URL putanju stranice. (Slika 9)

```
backend > middlewares > JS auth.js > isAuthenticatedUser
1   const user = require("../models/user");
```

Slika 9 Zahtjev za rutom

Rutiranje ima funkcije koje dohvaćaju zahtjev i daju odgovor na njega šaljući mu podatke. Primjer takve funkcije možemo vidjeti na slici 10.

```
// dohvati sve podatke   =>   /api/v1/products?keyword=hoodica
exports.getProducts = catchAsyncErrors(async (req, res, next) =>{


    const resPerPage = 4;
    const productsCount = await Product.countDocuments();

    const APIFeaturess = new APIFeatures(Product.find(),req.query)
                            .search()
                            .filter()
                            .pagination(resPerPage)
    const products=await APIFeaturess.query;

    res.status(200).json({
        success: true,
        productsCount,
        resPerPage,
        products
    })
})
```

Slika 10 Funkcija rutiranja

2.4.2 Posrednici

Posrednik (eng. middleware) je kod koji se izvršava u vremenu zahtjev/odgovor ciklusa. Koristi se za oblikovanje ponašanja mrežnog servera i za dodavanje novih funkcionalnosti. Ove funkcije imaju pristup objektima zahtjeva i odgovora. Izvršavaju se jedna nakon druge, ovisno o rasporedu istih. Dodavanje funkcionalnosti se obavlja slanjem istih u req i res objekte.

Dobar primjer Express posrednika je cookie-parser. On analizira zaglavlje kolačića dolaznog zahtjeva i popunjava vrijednosti kolačića na objektu req pod req.cookies kao objekt. To znači da će svi drugi ruteri ili posrednici koji se pokreću nakon raščlanjivanja kolačića imati pristup kolačićima poslanim sa zahtjevom.

Express je sam po sebi kolekcija posrednika, uz mogućnosti rutiranja. Posrednici su ključni za dodavanje funkcionalnosti Express serveru, fleksibilni su i mogu se izrađivati vlastiti (Slika 11).

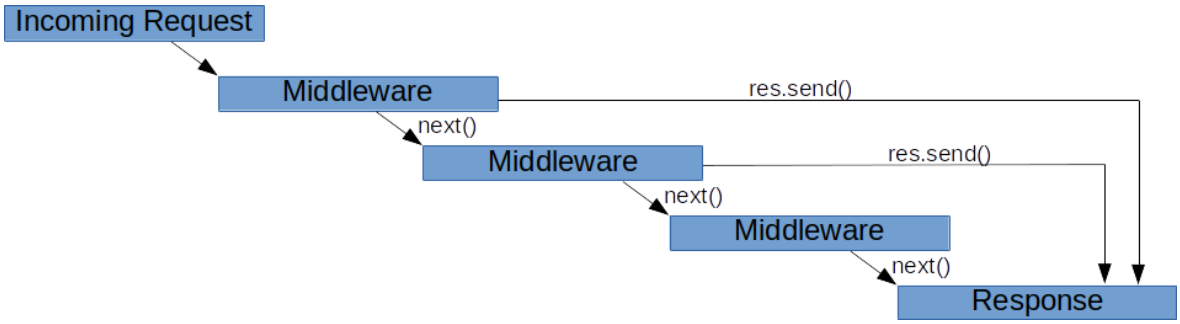
Slika 11 Primjer vlastitog posrednika

2.4.3 Zahtjev/odgovor i veza sa posrednicima

U funkciji možemo primjetiti sadržane req i res parametre. Oni su zapravo zahtjev kojeg je primio server, te odgovor koji se eventualno šalje natrag. Jedna HTTP transakcija se može opisati pomoću ciklusa zahtjeva i odgovora. Express nam pomaže da dovršimo ove transakcije korisno povećavajući ugrađene objekte zahtjeva i odgovora koje node-ov jezgroviti HTTP modul pruža kada vaš poslužitelj primi zahtjev. [4]

Spomenuti se zahtjev/odgovor ciklus odvija kroz slijedeće korake:

- Kada netko pristupi serveru, express aplikacija primi zahtjev
- Stvara se zahtjev/odgovor objekt
- Podaci se iskorištavaju za stvaranje i slanje smislenog odgovora
- Za procesuiranje tih podataka, koristimo posrednike, koji mogu manipulirati zahtjev/odgovor objekt



Slika 12 Prolazak zahtjeva kroz svaki posrednik dok se ne dobije odgovor[15]

Nakon stvaranja objekta zahtjeva i odgovora, oni prolaze kroz svaki posrednik gdje se obrađuju ili se izvršava neki drugi kod. Na kraju svakog posrednika poziva se next() funkcija. Njoj imamo pristup u svakom posredniku, baš kao što ima i objekt zahtjeva i odgovora. Kada se pozove next() funkcija izvršava se slijedeći posrednik po redu s točno određenim zahtjev i odgovor objektima. Na ovaj način oba objekta, korak po korak, prolaze kroz svaki posrednik pojedinačno. Cijeli ovaj proces možemo vidjeti, slikovito prikazan, na slici 12. Ovo možemo zamisliti kao neku vrstu cjevovoda, kroz koji naši podaci prolaze i prenose se od zahtjeva do konačnog odgovora.

Najčešće je posljednji posrednik rukovatelj rutera, gdje više ne pozivamo funkciju next() nego konačni objekt šaljemo klijentu kao odgovor.

Ako posrednik ne završi ciklus zahtjeva i odgovora, morao bi pozvati next() funkciju koja će kontrolu proslijediti slijedećem posredniku i tako dok ne dobijemo odgovor. Ako ne pozove funkciju next, tu nastaje problem da zahtjev na neki način ostaje visjeti u sredini. Ovako se završava cijeli ciklus zahtjeva i odgovora.

[15] Izvor: https://miro.medium.com/max/953/1*NdJKXX4mSin7D3QENE0RJw.png

3 React Hooks

React je biblioteka koja se konstantno širi i dobiva nove značajke i komponente. U svom v16.7.0 ažuriranju predstavio nam je novitet pod nazivom *hooks* koje predstavljaju metode za upravljanje stanjem React komponenti.

Hook-ovi su značajka koja nam omogućuje korištenje stanja ali i ostalih značajki Reacta bez potrebe pisanja klase. Možemo reći su funkcije koje "zakače" React stanje i značajke životnog ciklusa iz funkcijskih komponenti. Jako je važno napomenuti da ne rade unutar klasa.[5]

Ako želite dodati neko stanje u funkcijsku komponentu za to bi trebali koristiti klase. No React se ne bi složio jer upravo to se može postići hook-ovima unutar postojeće funkcijske komponente.

Hook-ovi su karakteristični i po tome jer imaju dva bitna pravila:

1. Hook-ovi se pozivaju samo na najvišoj razini. Nemojte ih pozivati unutar petlji, ugniježđenih funkcija ili uvjeta. Ovo pravilo osigurava da su hook-ovi pozvani u istom redoslijedu svaki put kad se komponenta iznova rendera.[5]
2. Hook-ove pozivajte samo iz komponenata funkcije React. Ne pokušavajte ih pozivati iz uobičajenih JavaScript funkcija. Umjesto toga možete pozivati hook-ove iz React funkcijskih komponenti. Mogu se pozivati i iz ručno izrađenih hook-ova.[5]

Hook-ove je također jako bitno započeti sa use. Ukoliko se to ne napravi React će upozoriti na tu grešku.

Jedna od glavnih snaga React-a je način na koji sinkronizira stanje aplikacije sa korisničkim sučeljem. Kako se stanje mijenja, zbog interakcije korisnika ili ažuriranja podataka iz sustava ili mreže, React inteligentno i učinkovito utvrđuje koje promjene treba uputiti na DOM-u u pregledniku ili na korisničkom sučelju općenito u drugim sredinama.[6]

Ako hook-ove uspoređujemo sa klasama, te njihove funkcijske komponente, doći ćemo do zaključka kako oni omogućavaju čišće i preglednije kodiranje, te kod koji je lak za testiranje, održavanje pa tako i ponovno korištenje i „recikliranje“. Funkcijske komponente s hook-ovima više ne trebaju sve metode životnog ciklusa jer se učinci mogu inkapsulirati u hook-

ove. Kod je bio mnogo bolje organiziran s ta dva različita efekta po strani razdvojena, te njihov kod konsolidiran na jednom mjestu za svaki efekt.[6]

Dva najpoznatija i najčešće korištena hook-a su useState i useEffect. O njima ću posebno govoriti, no postavlja se pitanje, kada se koriste više puta u istom kodu, kako kod zna koje stanje pripada kojem hook-u? Hook-ovi uvijek gledaju prema svom redoslijedu unutar komponente kako bi utvrdile s kakvim stanjem odgovaraju. Ako bi se ovaj redoslijed mijenjao pri svakom iscrtavanju, React ne bi znao što vratiti na svaki poziv, što bi dovelo do neželjenih nuspojava i grešaka! Ako se javi potreba za uvjetno izvršavanje, jednostavno se stavi uvjet unutar određenog hook-a. Ako programer slučajno zaboravi na ključna pravila, React je stvorio poseban dodatak ESlint koji će primijeniti ova dva pravila.

Spomenuo sam useEffect i useState hook-ove, posebno ću ih obraditi, a uz njih je važno predstaviti i useContext hook te na kraju i kreiranje vlastitih hook-ova.

3.1 useState Hook

Stanje je mjesto odakle podaci dolaze. Uvijek bismo trebali nastojati učiniti naše stanje što jednostavnijim. Uz React 16.8 omogućeni su hook-ovi koji nam omogućuju da se nosimo sa stanjem s manje kodiranja.

Slika 13 Korištenje useState hook-a za pretragu po ključnoj riječi

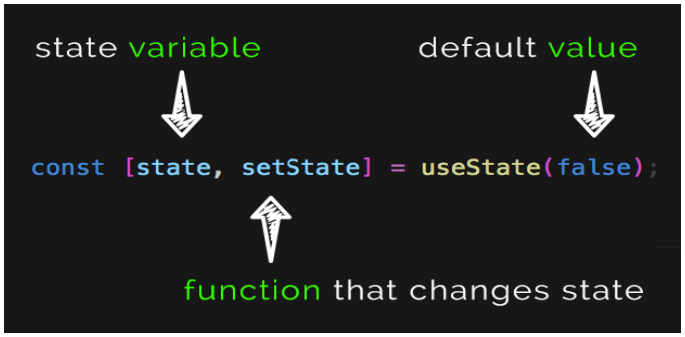
Kako vidimo na primjeru na slici 13, prije svega se uvozi useState hook iz React-a. Naravno bez toga ju ne bi mogli koristiti, a tako je i sa svim ostalima. Nakon uvoza prva stavka bit će naziv naše varijable stanja, druga stavka bit će naziv funkcije koju koristimo za ažuriranje naše varijable stanja. Možemo ih nazvati kako god želimo. Našem stanju početnu vrijednost dajemo proslijedivši ga useState-u.

Iz slike 13 možemo vidjeti da useState mijenja stanje ključne riječi u prazan string. Nakon toga searchHandler prima ono što je upisano u tražilicu te prosljeđuje rezultat na novi URL. Cijela funkcija za tražilicu vraća traku za pretragu iz koje se šalje vrijednost onog što je upisano da bi hook baratao time i mijenjao stanje.

Općenito govoreći jedini argument za useState hook je početno stanje. Za razliku od klasa, stanje ne mora biti objekt. Možemo zadržati broj ili niz ako je to sve što nam treba. Vraća par vrijednosti: trenutno stanje i funkciju koja ga ažurira. Zbog toga pišemo const [keyword, setKeyword] = useState () kao što vidimo na slici 14. Ovo je slično this.state.keyword i this.setKeyword u klasi, osim što ih dobivate u paru. Deklariranje varijabli stanja kao par [nešto, setNešto] također je zgodno jer nam omogućuje davanje različitih naziva različitim

varijablama stanja ako želimo koristiti više od jednog. Ne moraju se koristiti mnoge varijable stanja. Varijable stanja mogu se dobro nositi sa objektima i nizovima, pa se još uvijek mogu grupirati povezani podaci. Međutim, za razliku od this.setState u klasi, ažuriranje varijable stanja uvijek je zamjenjuje umjesto da je spaja.

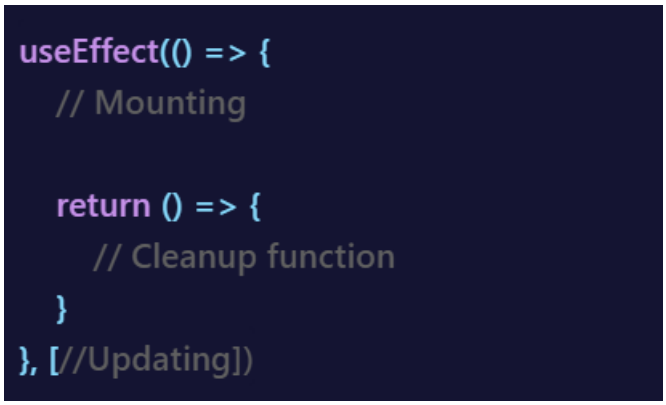
Kada deklariramo varijablu stanja pomoću useState funkcije, ona vraća par, niz s dvije stavke. Prva stavka je trenutna vrijednost, a druga je funkcija koja nam omogućuje ažuriranje. Korištenje [0] i [1] za pristup njima pomalo je zbunjujuće jer imaju određeno značenje. Zato umjesto toga koristimo destrukturiranje niza.[7]



Slika 14 Deklaracija i značenje varijabli prilikom korištenja useState funkcije

3.2 useEffect Hook

Korištenje useEffect hook-a podrazumijeva razumijevanje životnog ciklusa komponente. Ovaj životni ciklus sastoji se od tri glavna dijela: montaže, ažuriranja i demontaže. Životni ciklus komponente funkcionira na slijedeći način: kada korisnik dođe na stranicu, komponenta se montira, a zatim se ažurira stanje, komponenta se također ažurira i, konačno, komponenta se demontira kada korisnik napusti stranicu.[8]



Slika 15 Anatomija useEffect hook-a

Na slici 15 imamo uvid kako se piše useEffect hook odnosno samu anatomiju istog. Životni ciklus je podijeljen u tri cjeline unutar hook-a. Prva je montiranje, a ono označava trenutak kada korisnik uđe na stranicu i tu se komponenta rendera po prvi put. Return funkcija služi za „čišćenje“. Kada korisnik napusti stranicu komponenta će se demontirati. Posljednja cjelina je red. Sadržaj reda su sva stanja koja će se za vrijeme trajanja životnog ciklusa ažurirati. Ako komponentu ne želimo ažurirati, red se može odbaciti.

Ovaj se hook koristi kada se žele pokrenuti neke funkcije za vrijeme životnog ciklusa komponente. Dobar primjer korištenja je ako želimo da se ažurira korisničko sučelje prilikom promjene stanja. Također valja napomenuti da se stanje može definirati tijekom prvog učitavanja (componentDidMount), pa i očistiti stanje dok se komponenta demontira (componentWillUnmount).

I ovaj hook se mora prvo uvesti iz React-a da bi se koristio. Iza toga stvorimo neku komponentu na kojoj ćemo raditi.

```
40      useEffect(() => {
41          if (error) {
42              return alert.error(error)
43          }
44          console.log(category)
45          dispatch(getProductsStorePage(keyword, currentPage, price, category));
46
47
48
49      }, [dispatch, alert, error, keyword, currentPage, price, category])
50
51      function setCurrentPageNo(pageNumber) {
52          setCurrentPage(pageNumber)
53      }
54      return (
55          <Fragment>
56              <div class="heading-background">
57                  <h2 id="products-heading"> Proizvodi </h2>
58              </div>
59              <section id="products" className="container mt-5">
60                  <div className="row">
61                      {/* filter po cijeni i kategorijama (prikazuje se samo kada se traži određeni proizvod) */}
62
63                          <Fragment>
```

Slika 16 Primjena useEffect hook-a

Na slici 16 prikazana je primjena ovog hook-a. Prvo provjeravamo postoji li neka greška, te ukoliko ona postoji šaljemo upozorenje na istu. Pomoću dispatch funkcije šaljemo akciju, u ovom slučaju getProductsStorePage. Ovo je proces montiranja na primjeru koda iz mog projekta. Na kraju ide ažuriranje, a sa slike također možemo vidjeti koje se sve komponente ažuriraju, izlistane u uglatim zagradama.

Funkcija će u ovom konkretnom slučaju vraćati filtriranje po cijeni, filtriranje po kategoriji i raspodjelu po stranicama(paginaciju).

3.3 useContext Hook

Ovo je jedan od hook-ova koje nisam koristio u svom projektu, ali je član „trojstva“ najvažnijih pa je potrebno napisati par rečenica i o njemu.

Da bi koristili useContext hook moramo prvo znati koja je to uloga konteksta u React-u. React Context API uveden je kako bi se prevladao problem prenošenja rekvizita niz stablo komponenti. Stanje aplikacije može se globalno pohraniti i podijeliti na više komponenti. Sada s dodavanjem hook-ova u React arhitekturu, dobivamo novi hook koji se zove useContext.[9]

Prije smo mogli koristiti kontekst API samo unutar komponenti klase. No s ovim hook-om možemo s lakoćom koristiti kontekst unutar funkcionalnih komponenti. Također je mnogo lakše čitati i pisati.

```
1  import React from 'react';
2
3  const ThemeContext = React.createContext("gray");
4
5
6  export default function App() {
7    return (
8      <ThemeContext.Provider value="gray">
9        <ChildComponent />
10     </ThemeContext.Provider>
11   );
12 }
13
14 function ChildComponent() {
15   const theme = React.useContext(ThemeContext);
16   return (
17     <button style={{ background: theme }}>
18       I am styled by theme context!
19     </button>
20   );
21 }
```

Slika 17 Primjena useContext hook-a[16]

Prvo se kao i uvijek uveze hook iz React-a. Primjer sa slike 17 će nam poslužiti za uvid u mogućnosti useContext hook-a. Siva boja je postavljena kao zadana. Zatim se koristi Context.Provider na komponenti najviše razine kako bismo ponudili temu svim dolje podređenim komponentama. ChildComponent sada može pristupiti ili konzumirati kontekst

[16] Izvor: https://www.educative.io/edpresso/how-to-use-the-usecontext-hook-in-react

teme pomoću useContext hook-a. Na kraju kad god se promijeni vrijednost ThemeContext, promjena će se odraziti na potrošačke komponente djeteta.

3.4 Izrada vlastitih hook-ova

Vlastiti hook-ovi su ustvari normalne Javascript funkcije koje mogu koristiti drugi hook-ovi unutar njega i sadrže zajedničku logiku stanja koja se može ponovno koristiti unutar više komponenti. Ove funkcije imaju prefiks use. One štede programeru ne samo vrijeme, nego i broj linija koda, koji je s njima znatno manji.

```
const useCounter = () => {
  const [value, setValue] = useState(0)

  const increase = () => {
    setValue(value + 1)
  }

  const decrease = () => {
    setValue(value - 1)
  }

  const zero = () => {
    setValue(0)
  }

  return {
    value,
    increase,
    decrease,
    zero
  }
}
```

Slika 18 Primjer vlastitog hook-a[17]

[17] Izvor: https://fullstackopen.com/en/part7/custom_hooks

Izrađeni hook sa slike 18 koristi interni useState hook za stvaranje vlastitog stanja. Hook vraća objekt čija svojstva uključuju vrijednost brojača kao i funkcije za upravljanje vrijednošću.

Možemo prepoznati da se u ovom primjeru radi o brojaču koji mijenja vrijednosti ovisno o zahtjevu. Po zahtjevima imamo pribrajanje, oduzimanje i postavljanje na nulu.

Vlastiti hook-ovi nude fleksibilnost dijeljenja logike koja prije nije bila moguća u React komponentama. Mogu se izraditi vlastiti hook-ovi koji pokrivaju velik broj slučajeva upotrebe, poput rukovanja obrascima, animacije, mjerača vremena i vjerojatno još mnogo toga.[10]

4 Oblikovanje i izrada aplikacije

Projekt koji sam izradio kao podlogu i podršku ovom radu, temelji se na MERN stack tehnologijama. MERN Stack je grupa tehnologija za izradu internetskih aplikacija, a obuhvaća: MongoDb, Express, React i Node. Koristio sam svaku od njih da bi izradio internetsku trgovinu. Poznat nam je koncept internetskih trgovina, te je projekt zamišljen da bude jednostavna, brza internet trgovina s proizvoda koji nose logo našeg fakulteta.

Rad na projektu sam započeo sa istraživanjem tehnologija iz MERN stack-a. Od svih najviše sam se posvetio Reactu o kojem sam znao jako malo, a s Javascriptom sam imao malo iskustva.

Nakon istraživanja i učenja nekih novih stvari, počeo sam rad na backend dijelu aplikacije. Za korištenje tehnologija u backendu i za samu izradu, poslužio sam se raznim tutorijalima i internetskim člancima.

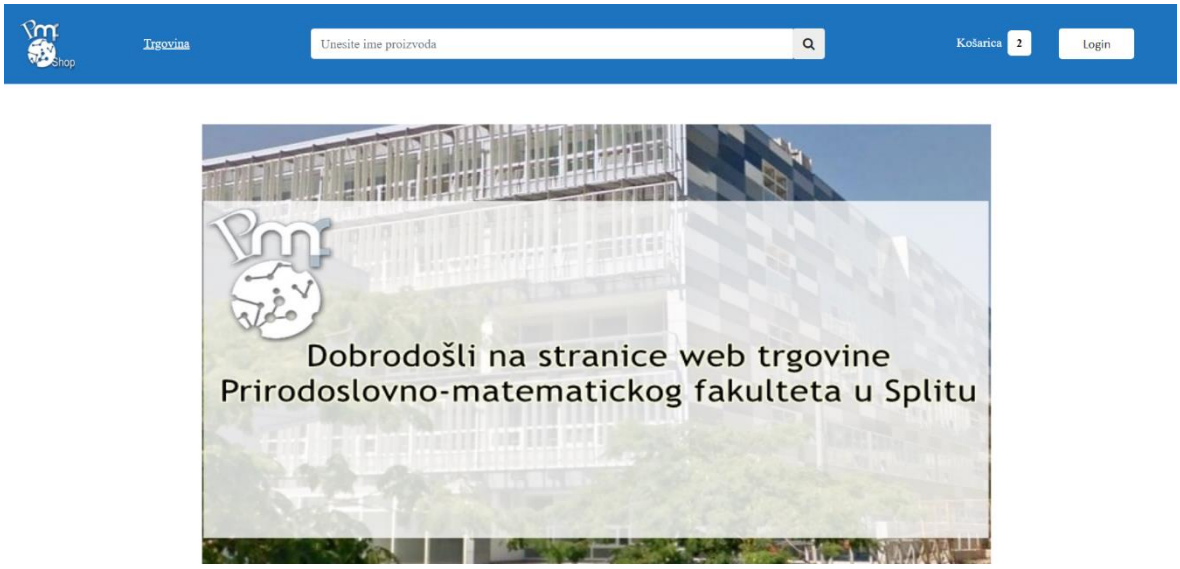
```
12  //CONTROLLER METODE
13
14  // Registriraj korisnika => /api/v1/register
15  exports.registerUser = catchAsyncErrors( async(req, res, next) => {
16
17      const result = await cloudinary.v2.uploader.upload(req.body.avatar, {
18          folder: 'avatars',
19          width: 150,
20          crop: "scale"
21      })
22
23      const {name, email, password} = req.body;
24
25      const user = await User.create({
26          name,
27          email,
28          password,
29          avatar:{
30              public_id: result.public_id,
31              url: result.secure_url
32          }
33      })
34
35      sendToken(user, 200, res)
36  })
37
```

Slika 19 Backend – primjer controllera

Obzirom da su rute i posrednici već opisani ranije, iz backenda je važno izdvojiti još jednu bitnu funkciju, a to je controller čiji primjer možemo vidjeti na slici 19. Ovaj controller služi za registraciju korisnika. Prvo zahtjeva s klijentske strane podatke koji su potrebni za

stvaranje novog korisnika, u ovom slučaju to su: ime, email i lozinka. Kada dobije tražene podatke pomoću create() funkcije stvara se novi korisnik s pripadajućim podacima i sprema se u bazu podataka. Na kraju se šalje token, koji služi da bi korisnik nakon registracije ostao prijavljen sve dok se sam ne odjavi ili dok ne istekne potrebno vrijeme dok ga sustav sam ne odjavi.

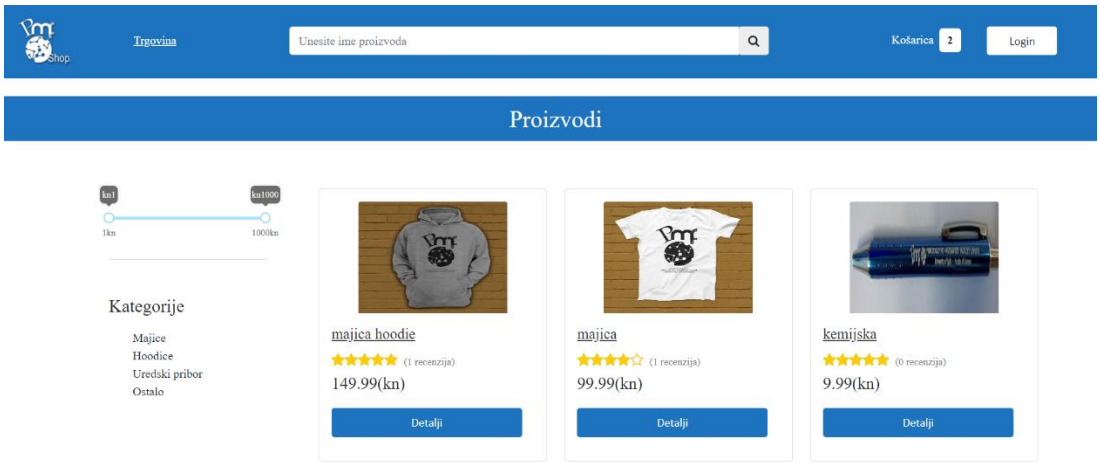
4.1 Aplikacija PMF Web Shop



Slika 20 Početna stranica aplikacije

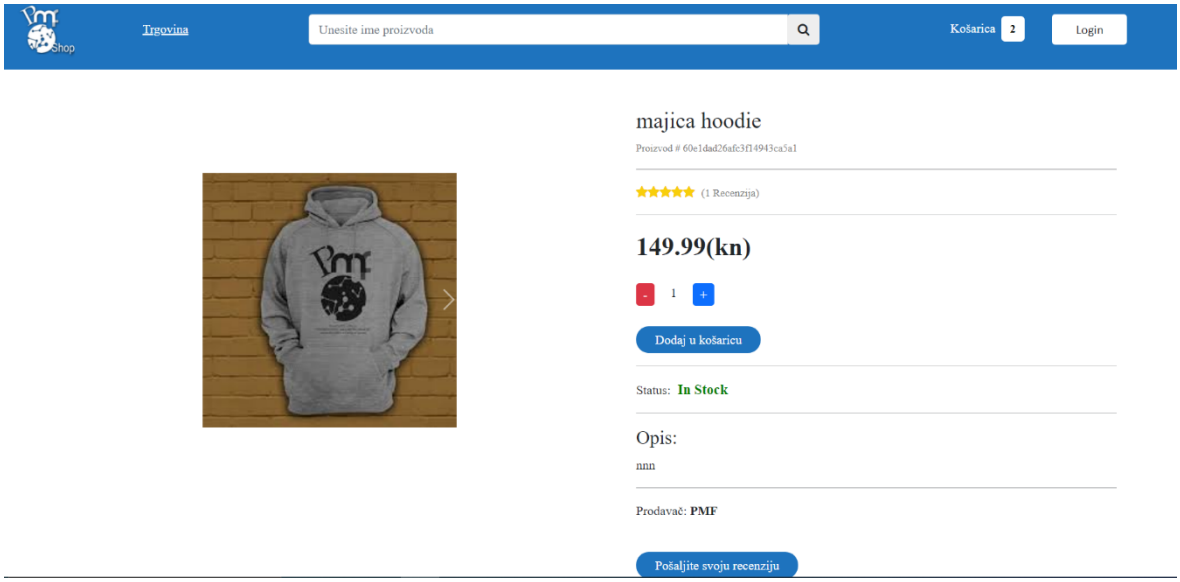
Kada se otvori aplikacija, korisnika dočeka početna stranica (Slika 20). Na njoj možemo vidjeti logo, koji vodi na početnu stranicu, stranicu za trgovinu, tražilicu i gumb za prijavu.

Klikom na vezu „Trgovina“ vidjet ćemo slijedeće (Slika 21):



Slika 21 Trgovina

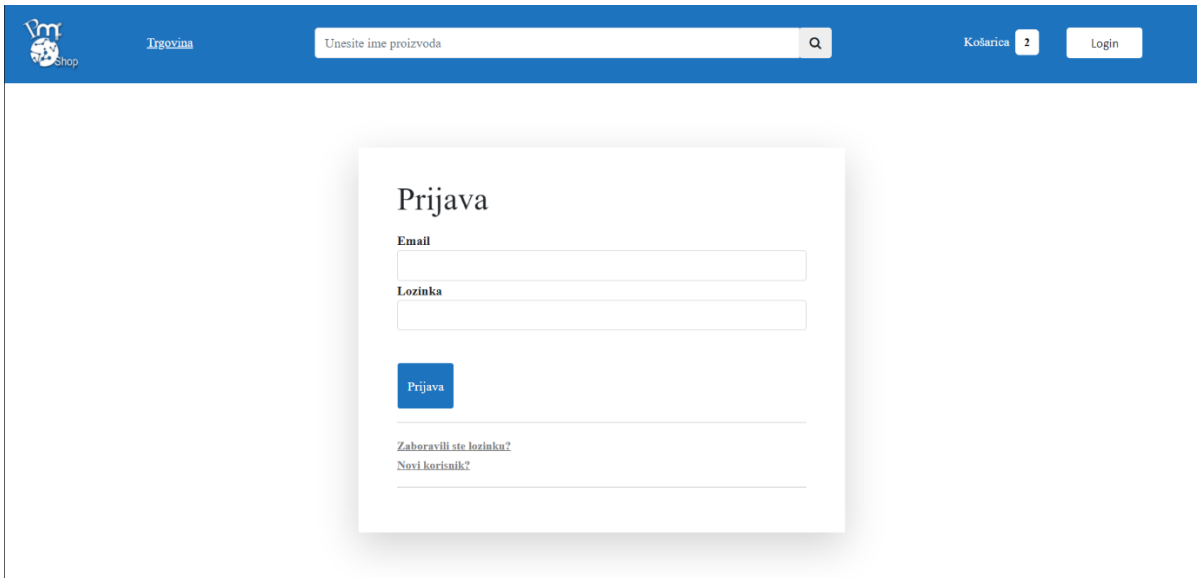
Proizvodi se iz baze prikazuju na ovoj stranici i raspoređuju prema kategorijama. Dakle filtrirati ih možemo po cijeni i kategorijama. Detalje i recenzije o svakom od prozivoda možemo vidjeti klikom na gumb „Detalji“ ili na samu sliku proizvoda.(Slika 22)



Slika 22 Detalji o proizvodu

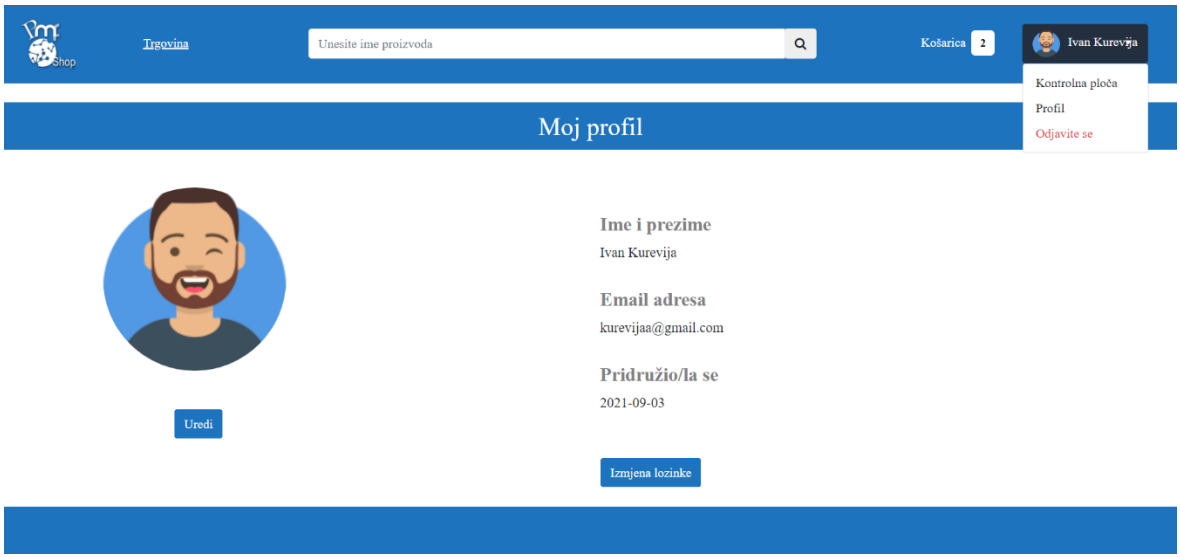
Kada otvorimo detalje o proizvodu (Slika 22), možemo vidjeti prije svega ime, sliku i jedinstveni id proizvoda. Između ostalog tu je gumb za naručivanje, te je moguće napisati i objaviti vlastitu recenziju za svaki proizvod.

Aplikacija sadrži mogućnost registracije i prijave korisnika. Ovo omogućuje korisniku praćenje i pregled svojih narudžbi. Postoje dvije uloge: administrator i korisnik. Administrator ima mogućnost manipulacije proizvodima, narudžbama i opće upravljanje stranicom. Ako osoba ima administratorske ovlasti, sve navedeno može izvršavati klikom na kontrolnu ploču. Obični korisnik može naručiti, pratiti narudžbe i recenzirati. Osim toga omogućeno je uređivanje profila i promjena podataka, te slike profila. Slika profila se automatski sprema na Cloudinary server i prikazuje na korisničkom profilu.



Slika 23 Stranica za prijavu korisnika

Na slici 23 vidimo stranicu za prijavu korisnika. Kao i na svakoj web aplikaciji, korisnik upisuje svoje podatke i prijavljuje se na stranicu. Na stranici za prijavu korisnik ima mogućnost zatražiti token za promjenu lozinke. Ukoliko je zaboravio lozinku, na mail mu dolazi link(token) uz pomoć kojeg može promijeniti lozinku. Stranica za prijavu sadrži i link za registraciju novog korisnika, ako već nema svoj račun.



Slika 24 Korisnički profil

Uz sve navedeno aplikacija omogućava naručivanje proizvoda, dodavanjem u košaricu. Narudžbom proizvoda na mail dolazi narudžbenica sa svim podacima. Za sad još uvijek nema mogućnost online plaćanja, ali moguća je nadogradnja u budućnosti uz pomoć Stripe tehnologije. Stripe je zapravo tvrtka koja nudi software za obradu plaćanja, te API za aplikacije internetskih trgovina. Omogućavaju implementaciju kartičnog sustava plaćanja u programski kod, te imaju visoke sigurnosne standarde.

5 ZAKLJUČAK

Često možemo čuti da se u zadnja dva desetljeća cijeli svijet „preselio“ na internet. Ponekad stvarno djeluje tako, jer puno svakodnevnih obaveza možemo obavljati online. Komunikacija sa ostatkom svijeta nikad nije bila bolja, u usporedbi sa prošlim vremenima, čak i nevjerojatna. Starije generacije nisu mogle ni zamisliti da će besplatno moći satima razgovarati s osobom na drugom kontinentu, kupovati odjeću i hranu na internetu, te imati pristup svekolikom znanju i količini informacija. Kada pogledamo brzinu razvoja virtualnog svijeta, možemo zaključiti da je sve veća potreba za boljim, bržim i naprednijim internetskim aplikacijama. Više nisu dovoljne samo „obične“ stranice sa osnovnim informacijama i telefonskim brojem, već aplikacije koje nude usluge i proizvode „sada i odmah“. Osim toga ni jedan korisnik više neće trpjeti predugo učitavanje te sporu reakciju stranice. Tu nastupa React sa popratnim tehnologijama omogućava kreaciju brzih i oku ugodnih, te prije svega kompleksnih web aplikacija. Ja sam to pokazao na primjeru web trgovine, gdje korisnici mogu naručiti proizvod brzo i bezbolno.

Budućnost svijeta je u simbiozi između online i stvarnog života, a programeri su glavni alat za postizanje te simbioze, pretvaranjem „dosadnih“ dnevnih obaveza u par klikova mišem. Na taj način se svijetu može vratiti toliko bitno slobodno vrijeme. Tehnologije poput opisanih olakšavaju programeru kreaciju takvog svijeta i korisniku daju brzinu i efikasnost u obavljanju kupovine, plaćanja računa, rješavanju birokracije i tako dalje.

6 Literatura

[1] Što je React https://www.popwebdesign.net/sta-je-react.html

[2] Prednosti i mane MongoDb https://www.virtual-dba.com/blog/pros-and-cons-of-mongodb/

[3] Routing https://heynode.com/tutorial/what-express-nodejs-framework/

[4] Request/Response https://heynode.com/tutorial/what-express-nodejs-framework/

[5] React hooks https://www.javatpoint.com/react-hooks

[6] John R. Larsen (2020). React Hooks in Action With Suspense and Concurrent Mode Version 3, 18, 20-21

[7] Using the State Hook https://reactjs.org/docs/hooks-state.html

[8] useEffect Hook https://designcode.io/react-hooks-handbook-useeffect-hook

[9] Learn the useContext Hook https://programmingwithmosh.com/javascript/learn-the-usecontext-hook-in-react/

[10] Custom Hooks https://reactjs.org/docs/hooks-custom.html

7 Popis slika

Slika 1 Glavne karakteristike React-a 3
Slika 2 MongoDb Kolekcije 4
Slika 3 Primjer proizvoda spremljenog u Mongo bazu podataka u obliku JSON objekta 5
Slika 4 Proces učitavanja web stranice nakon nekog događaja 6
Slika 5 Popis NPM paketa iz mog projekta 8
Slika 6 Primjer instaliranja paketa 9
Slika 7 Rutiranje 11
Slika 8 Uvoz objekata 11
Slika 9 Zahtjev za rutom 11
Slika 10 Funkcija rutiranja 12
Slika 11 Primjer vlastitog posrednika 13
Slika 12 Prolazak zahtjeva kroz svaki posrednik dok se ne dobije odgovor 14
Slika 13 Korištenje useState hook-a za pretragu po ključnoj riječi 17
Slika 14 Deklaracija i značenje varijabli prilikom korištenja useState funkcije 18
Slika 15 Anatomija useEffect hook-a 18
Slika 16 Primjena useEffect hook-a 19
Slika 17 Primjena useContext hook-a 20
Slika 18 Primjer vlastitog hook-a 21
Slika 19 Backend – primjer controllera 23
Slika 20 Početna stranica aplikacije 24
Slika 21 Trgovina 24
Slika 22 Detalji o proizvodu 25
Slika 23 Stranica za prijavu korisnika 26
Slika 24 Korisnički profil 26