Upotreba reaktivnog programiranja pri razvoju web aplikacija

Čerjan, Vlaho

SVEUČILIŠTE U DUBROVNIKU

ODJEL ZA ELEKTROTEHNIKU I RAČUNARSTVO

VLAHO ČERJAN

UPOTREBA REAKTIVNOG PROGRAMIRANJA PRI RAZVOJU WEB APLIKACIJA

DIPLOMSKI RAD

Dubrovnik, rujan, 2020

SVEUČILIŠTE U DUBROVNIKU

ODJEL ZA ELEKTROTEHNIKU I RAČUNARSTVO

UPOTREBA REAKTIVNOG PROGRAMIRANJA PRI RAZVOJU WEB APLIKACIJA

DIPLOMSKI RAD

Studij: Primijenjeno/poslovno računarstvo

Kolegij: Raspodijeljeni informacijski sustavi

Mentor: doc.dr.sc. Krunoslav Žubrinić

Komentorica: Ana Kešelj mag. ing. comp.

Student: Vlaho Čerjan JMBAG: 0023104375

Dubrovnik, rujan, 2020.

REPUBLIKA HRVATSKA

SVEUČILIŠTE U DUBROVNIKU

ODJEL ZA ELEKTROTEHNIKU I RAČUNARSTVO

STUDIJ: Primijenjeno/poslovno računarstvo

Sveučilišni diplomski studij

Ur. broj: 39/20-ER

Dubrovnik, 29. lipnja 2020.

Kolegij: Raspodijeljeni informacijski sustavi

Mentor: doc. dr. sc. Krunoslav Žubrinić

Komentor: Ana Kešelj, mag. ing. comp.

DIPLOMSKI ZADATAK

Kandidat: **VLAHO ČERJAN**

JMBAG: **0023104375**

Naslov zadatka: **UPOTREBA REAKTIVNOG PROGRAMIRANJA PRI RAZVOJU WEB APLIKACIJA**

REACTIVE PROGRAMMING IN WEB APPLICATION DEVELOPMENT

Opis zadatka: Usporediti različite programske paradigme. Opisati pojmove reaktivnog sustava i reaktivnog programiranja. Opisati povijest, specifičnosti i osnovne karakteristike reaktivnog pristupa izradi aplikacija. Usporediti monolitnu i reaktivnu arhitekturu web aplikacije. Koristeći reaktivan pristup izraditi model i prototip web aplikacije za trgovinu i servis elektroničkim proizvodima.

Zadatak je uručen kandidatu 29. lipnja 2020.
Rok za predaju diplomskog rada je do 18. rujna 2020.

Mentor:

doc. dr. sc. Krunoslav Žubrinić

SVEUČILIŠTE U DUBROVNIKU
UNIVERSITAS STUDIORUM RAGUSINA

Pročelnik Odjela:

izv. prof. dr. sc. Mario Miličević

SAŽETAK

React.js je *JavaScript* biblioteka za izgradnju korisničkih sučelja. Bazira se na komponentama koje mogu funkcionirati samostalno i kontrolirati vlastito stanje. Unutar izrađene aplikacije se koristi reaktivna izrada korisničkog sučelja tj. koristi se *React.js*. Upotrebom reaktivnog programiranja ubrzava se i olakšava proces izrade sučelja korištenjem *React* komponenti. Za izradu cijele aplikacije korišten je MERN sustav što znači da se koristila *MongoDB* baza podataka, *Express.js* okvir za *back-end* razvoj, *React.js* biblioteka za izradu korisničkog sučelja i *Node.js* kao serversko okruženje.

Ključne riječi: reaktivno programiranje, *Express.js*, MERN, *MongoDB, Node.js, React.js*

ABSTRACT

React.js is a *JavaScript* library used for building user interfaces. It's based on components that can work on their own and control their own state. Reactive building of user interfaces was used in the created application i.e., it was built using *React.js*. Using reactive programming speeds and eases the process of building interfaces through use of *React* components. The whole app was made using the MERN system. This means that *MongoDB* was used for a database, *Express.js* for a back-end system, *React.js* library for building a user interface and *Node.js* as a server environment.

Keywords: reactive programming, *Express.js*, MERN, *MongoDB, Node.js, React.js*

SADRŽAJ:

1 UVOD

Izrada aplikacije reaktivnim načinom olakšava i ubrzava razvoj korisničkim sučelja *web* aplikacija. Za razvoj na takav način razvijene su brojne biblioteke npr. *React.js, Vue.js i Angular*. Korištenjem komponenti tih aplikacija repetitivni posao programera se smanjuje čime se povećava njihova produktivnost. To je razlog što su velike internetske kompanije (primjerice *Facebook*) prihvatile takav pristup, intenzivno ga promoviraju te su ga uključile u svoje razvojne biblioteke.

U ovom radu je prikazan kompletan proces izrade reaktivne aplikacije. U razvoju je korišteno razvojno okruženje *PhpStorm*, poslužiteljsko okruženje *Node.js*, biblioteka za izradu korisničkih sučelja *React.js* te distribuirana baza podataka *MongoDB* koji zajedno utjelovljuju jedan od najmodernijih načina izrade web aplikacija.

Rad ne opisuje osnovne *web* tehnologije (HTML, CSS i JS), već se temelji na opisu naprednijih tehnika *web* programiranja (JSX - *JavaScript* XML, SASS – *Syntactically Awesome Style Sheets*), te je za razumijevanje pojmova navedenih u ovom radu potrebno imati određeno predznanje u izradi *web* stranica. Svrha ovoga rada je prikazati razliku između različitih pristupa pri razvoju *web* aplikacija, a glavni cilj izrade je opisati pristup izradi *web* aplikacije pomoću reaktivnog načina programiranja, koristeći se *React.js* bibliotekom .

Nakon uvoda, u drugom poglavlju će biti ukratko opisana povijest razvoja *web* aplikacija s naglaskom na specifičnosti *web* korisničkih sučelja te različite načine modernog razvoja korisničkih sučelja web aplikacija, uključujući reaktivni pristup. Treće poglavlje opisuje glavne dijelove *web* aplikacije. U četvrtom poglavlju detaljno je opisan razvoj korisničke aplikacije. Opisana su korištena okruženja, biblioteka i izrađena baza podataka. Unutar petoga poglavlja su opisani najvažniji dijelovi programskog koda korisničkog sučelja, dok šesto poglavlje sadrži opis izgleda najvažnijih *web* stranica aplikacije. U zaključnom poglavlju se sažima sadržaj rada i daje kratak osvrt na proces razvoja opisan u ovom radu.

2 RAZVOJ WEB APLIKACIJA

Web aplikacije se razvijaju preko 30 godina, od samog nastanka *weba* 1990-ih godina. S naglim razvojem interneta i računalne tehnologije došlo je do naglog napretka u razvoju *web* aplikacija. U ovom poglavlju će se ukratko opisati povijest i moderni načini razvoja *web* aplikacija.

2.1 Povijest razvoja web aplikacija

Razvoj *web* aplikacija se generalno pripisuje Timu Berners-Leeju, britanskom znanstveniku, koji je zajedno s Belgijancem Robertom Cailliauom, objavio prijedlog hipertekst sustava 1990. godine. Kratko iza objave, Berners-Lee je opisao i razvio glavne značajke *weba*.

Te značajke su:

- Jedinstveni lokator resursa (engl. *Uniform Resource Locator* – URL)koji služi za jedinstveno prepoznavanje resursa na *webu*.
- HTTP tj. Protokol za prijenos hiperteksta koji opisuje na koji način djeluju zahtjevi (engl. *request*) i odgovori (engl. *response*).
- Programska potpora (engl. *software*) *web* poslužitelja koji može odgovoriti na HTTP zahtjeve.
- Jezik za označavanje hiperteksta (HTML) koji se koristi za oblikovanje sadržaja u dokumenata na *webu*.
- Program (*web* preglednik) koji može uputiti HTTP zahtjeve prema URL-ovima koji mogu prikazati HTML kod koji prime [1].

Iako su osnove *weba* začete u ranim 90-tima, *web* kakav znamo danas nije postojao do izuma *Mosaica* [2], prvog popularnog grafičkog *web* preglednika, koji su 1993. godine razvili Erica Bina i Marc Andreessen. *Netscape Communications*, čiji je suosnivač bio Andreessen, je krajem 1994. godine objavio *Netscape Navigator* koji je postao dominantan *web* preglednik, sve dok *Internet Explorer* nije preuzeo tu ulogu 1995. godine.

Naglim povećanjem korištenja weba, sredinom 2000-ih, pojavio se novi pojam - *Web 2.0*. Taj pojam je bio važan i za korisnike i za programere [1].

Za korisnike je predstavljao interaktivno iskustvo u kojem korisnik može pridonijeti i konzumirati sadržaj na *webu*, stvarajući korisnički upravljano *web* iskustvo. Unutar ove kategorije spadaju neke od najpopularnijih stranica današnjice: *Facebook, YouTube* i *Wikipedija*. Ovaj način korištenja *web* aplikacija, u kojem se dopušta korisniku komentiranje sadržaja, objavljivanje sadržaja ili pravljenje profila na društvenim mrežama, revolucionarizirao je korištenje *web* aplikacija.

Za programere, *Web 2.0* je označio promjenu u načinu izrađivanja dinamičkih *web* stranica. Dio programske logike, koja je prije toga postojala samo na poslužitelju, je migrirao u okruženje *web* preglednika. To je značilo da za razvoj interaktivnih *Web* 2.0 aplikacija *web* programeri moraju savladati klijentski programski jezik poput *JavaScripta*. Takvi programski jezici se izvršavaju u okruženju *web* preglednika. Pored toga, za efikasniju komunikaciju između klijentskog i poslužiteljskog dijela aplikacije kao i rad klijentskog dijela aplikacije, programeri su morali usavršavati programerske tehnike za asinkronu komunikaciju.

Web programiranje današnjice je značajno kompliciranije nego što je bilo čak i prije 10 godina tako da je pri razvoju nastala i određena raspodjela poslova. Mnogi programeri su loši vizualni dizajneri, a mnogi dizajneri nisu dobri programeri pa je prirodno nastala raspodjela između stvaranja programske potpore i dizajniranja vizualnog korisničkog sučelja. Ovaj koncept je bitan unutar novog *Web 2.0* sustava programiranja [1].

2.2 Razvoj modernih korisničkih sučelja

Razvoj modernih korisničkih sučelja je vrlo kompliciran i opširan postupak. *Web* programeri koji razvijaju korisnička sučelja web aplikacije trebaju imati značajnu količinu predznanja prije učenja novih tehnologija, od funkcioniranja samog interneta, HTML-a, CSS-a i *JavaScripta*, do sustava za verzioniranje koda (npr. *Gita* [3]), moraju poznavati metode *web* zaštite (HTTPS [4], CORS [5]), upravitelje paketa (npr. npm [6] ili yarn [7]), razne CSS preprocesore (npr. SASS [8], *PostCSS* [9]) kao i alate za automatizaciju izgradnje programske podrške (npr. npm skripte, *webpack* [10]).

Usvajanjem svih ovih pojmova, programer odabire razvojni okvir (engl. *framework*) u kojem će raditi. Neki od danas najpopularnijih razvojnih okvira su *React.js* [11], *Angular* [12] i *Vue.js* [13]. Unutar ovih razvojnih okvira se dodatno koriste i moderni CSS (stilizirane komponente, CSS moduli, stilizirani JSX itd.), *web* komponente (HTML predlošci, prilagođeni

elementi) i CSS razvojni okviri (npr. *Bootstrap* [14], *Reactstrap* [15], *Material* UI [16], *Tailwind* CSS [17] ...).

Nakon stjecanja potrebnog predznanja, slijedi savladavanje tehnologija izrade progresivnih *web* aplikacija (PWA [18]), koja uključuje učenje različitih *web* sučelja za programiranje aplikacija (engl. *Application Program Interface* - API), alata za poboljšanje performansi PWA, prikazivanje na strani poslužitelja (*React.js* -- *Next.js* [19], *Angular – Universal* [20], *Vue.js* -- *Nuxt.js* [21]), *GraphQL* [22] koji pruža cjelovit i razumljiv opis podataka različitih *API-ja* (*Apollo* [23], *Relay Modern* [24]), generatore statičkih stranica (*Next.js* [19], *GatsbyJS* [25]), mobilne aplikacije (R*eact Native* [26]) i desktop aplikacije (*Electron* [27]) [28].

Za izradu aplikacije opisane u ovom radu morao sam savladati mnogo tehnologija spomenutih u ovom potpoglavlju. U procesu izrade korišten je *React.js* kao razvojni okvir aplikacije, *Bootstrap* kao CSS okvir, npm kao alat za izgradnju sustava i upravitelj paketa te *BitBucket* [29] sustav za verzioniranje koda. Sve osim baze podataka je izrađeno i uključeno unutar razvojnog okruženja *PhpStorm* [30] u kojem je korišteno poslužiteljsko okruženje *Node.js* [31] za rad s alatima za automatizaciju izgradnje aplikacije, upraviteljima paketa, sustavom za verzioniranje koda i izgradnju aplikacije. Baza podataka je izgrađena unutar grafičkog korisničkog sučelja *MongoDB Compass* [32].

2.3 Reaktivno programiranje

Programeri pri razvoju programa mogu koristiti različite programske paradigme (stilove ili načine programiranja). Imperativno i reaktivno programiranje su dvije programske paradigme koje mogu koristiti. Imperativno programiranje je klasičan način programiranja kod kojeg programeri u programu navode niz instrukcija koje opisuju upravljački tok odnosno označavaju na koji način se točno u programu nešto treba izvršiti. U većini klasičnih programskih jezika imperativno programiranje je standardni način razvoja programa. Primjerice kada želimo dohvatiti neki zapis iz reda, kod imperativnog pristupa izradi programa programer određuje zadanim logičkim uvjetima situaciju odnosno trenutak u kojem treba dohvatiti zapis iz reda te navodi instrukcije kojima se podatak dohvaća [33].

Reaktivno programiranje je suprotna paradigma imperativnom programiranju. Taj pristup se temelji na pojmu kontinuiranih vremenski promjenjivih vrijednosti i širenju (propagiranju)

promjena. Kod reaktivnog programiranja programer određuje što treba napraviti, a sam programski razvojni okvir je zadužen za konkretnu realizaciju. Time se olakšava deklarativni razvoj aplikacija upravljanih događajima (engl. *event-driven application*) tako da dopušta programerima da programskim instrukcijama izraze što program treba učiniti, a programskom jeziku se prepušta da automatski odredi kada i na koji način treba učiniti određenu promjenu. Uzmimo za primjer jednostavnu sumu dvije varijable *var1* i *var2*. Ako u varijablu *var3* zapišemo sumu prethodne dvije varijable, promjenom vrijednosti tih varijabli program će automatski ažurirati vrijednost *var3* varijable. Ovakav način izvršavanja operacija je jedan od glavnih pojmova reaktivnog programiranja. U ovoj paradigmi, promjene stanja se automatski i učinkovito šire preko mreže međusobno ovisnih izračuna od strane temeljnog izvršnog modela [33]. Primjerice programer će u programu definirati funkciju za dohvaćanje zapisa iz reda koja će se automatski izvršiti kada tijekom izvođenja programa nastane događaj odgovarajućeg tipa.

Postoje dvije istaknute značajke reaktivnog programiranja: ponašanje (engl. *beahviour*) i događaj (engl. *event*). U reaktivnog programiranju, ponašanja je termin korišten za opis vremenski promjenjivih vrijednosti (engl. *time-varying*). Primjer takve vrijednosti je vrijeme tako da većina reaktivnih programskih jezika pruža primitiv ponašanja (engl. *behaviour primitive*) koji predstavlja vrijeme (npr. primitiv sekunde koji predstavlja vrijednost trenutne sekunde u minuti). Druga ponašanja se mogu izraziti kao funkcija vremenskog primitiva kojeg pruža reaktivni programski jezik. Događaji se odnose na potencijalno beskonačan tok promjena vrijednosti. Za razliku od ponašanja koji se neprekidno mijenjaju tijekom vremena, događaji nastaju u diskretnim točkama u vremenu (npr. pritisak tipke na tipkovnici, promjena lokacije) [33].

2.4 Baze podataka

Baza podataka je skup međusobno povezanih podataka o nekom realnom sustavu relevantnih za neko područje primjene koji su pohranjeni u računalskoj memoriji i organizirani tako da se njihovi elementi mogu koristiti u različitim obradama uz kontroliranu zalihost ili redundanciju.

2.4.1 Relacijske baze podataka

Relacijske baze podataka su standardna tehnologija koja se dugi niz godina koristi za pohranu strukturiranih podataka u računalnim sustavima. Temelje se na pohrani podataka u tabličnom obliku gdje svakom tipu podataka odgovara jedna tablica, dok zapisima odgovaraju reci u tablici.

Stupci tablice opisuju atribute i svaka tablica bi trebala imati ključni atribut koji jednoznačno opisuje određeni redak tablice. Tablice su međusobno povezane preko ključeva. Temelje se na relacijskom modelu podataka opisanom 1970.-ih godina. Kod smještanja podataka u tablice redoslijed zapisa niti redoslijed stupaca u tablici nisu važni. Za pretraživanje i ažuriranje podataka u relacijskim bazama koristi se standardni upitni jezik SQL koji je isti ili vrlo sličan za korištenje na relacijskim bazama različitih proizvođača.

2.4.2 NoSQL baze podataka

Popularizacijom interneta pojavio je problem koji se očituje velikom količinom podataka koje treba prihvatiti i obraditi u što kraćem vremenu. Posljednjih desetak godina na tržištu su se pojavile nove vrste nerelacijskih baza podataka koje pokušavaju riješiti navedeni problem. Te baze podataka ne koriste SQL kao jezik za pretraživanje podataka pa se najčešće nazivaju NoSQL baze podataka. U tim bazama podaci se ne organiziraju prema principima relacijskog modela nego, prema znatno slobodnijem modelu podataka koji ovisi o vrsti baze podataka. Jedna od najuočljivijih karakteristika većine ovih baza je da ne zahtijevaju definiciju sheme podataka poput relacijskih baza podataka. NoSQL baze podataka su zbog toga dinamičke, u smislu da je tip i broj atributa nekog entiteta “otvoren”, to jest može ga se po potrebi mijenjati bez da to utječe na druge podatke.

Prednost takvog pristupa je u tome što je jednostavnije preslikati strukturu podataka iz baze u strukturu podataka programskog jezika čime se pojednostavljuje interakcija između aplikacije i baze podataka. Druga važna karakteristika je što su NoSQL baze podataka osmišljene i realizirane za distribuiran način upotrebe, dok su relacijske baze podataka i relacijski model izvorno osmišljeni za korištenje na jednom računalu. NoSQL sustavi su napravljeni tako da se više njih može rasporediti na veći broj računala gdje svaki može raditi sa svojom skupinom podataka čime se značajno ubrzava prihvat i obrada podataka [34].

Osnovne razlike između relacijskih i NoSQL baza podataka su sljedeće:

- NoSQL baze su većinom ne-relacijske ili distribuirane baze podataka,
- NoSQL je relativno nova dok je tehnologija relacijskih baza podataka dosta stara tehnologija,

- Relacijske baze su bazirane na tablicama u obliku redova i stupaca te se moraju pridržavati standardnih definicija sheme. NoSQL baze mogu biti bazirane na dokumentima, parovima ključ-vrijednost, grafovima ili stupcima i ne moraju se držati shema,
- Relacijske baze imaju dizajniranu predefiniranu shemu za strukturirane podatke, NoSQL baze preferiraju denormalizirane sheme,
- Relacijske baze su skupe za horizontalno skaliranje dok su NoSQL baze relativno jeftine za takav način proširivanje u usporedbi sa SQL bazama jer su prilagođene su korištenju na distribuiranim sistemima,
- Relacijske baze su vertikalno skalabilne što znači da se skaliraju povećavanjem kapaciteta fizičkih dijelova (RAM, CPU, HDD, SSD,…). NoSQL baze su horizontalno skalabilne što znači da se mogu skalirati dodavanjem više poslužitelja unutar infrastrukture,
- Za razliku od NoSQL baza podataka, relacijske baze striktno prate ACID svojstva,
- Potreba za dodavanjem novog podataka unutar relacijske baze uzrokuje promjenu strukture baze odnosno sheme tablice. Novi podaci unutar NoSQL baze se jednostavno ubacuju pošto baze nemaju strogo definiranu strukturu poput relacijskih baza.

Ako NoSQL baze podataka usporedimo s relacijskim bazama podataka, neke od prednosti NoSQL baza podataka su sljedeće:

- sposobnost rukovanja velikim količinama podataka,
- pružaju visoku razinu fleksibilnosti sa modelima podataka jer ne postoji shema baze podataka niti postoje tablice,
- jednostavna horizontalna skalabilnost. Za horizontalno skaliranje nema potrebe povećavati količinu hardvera, a sve što je potrebno je dodati još servera unutar spremišta servera (engl. *server pool*) pošto NoSQL baze nemaju shemu i prilagođene su korištenju na distribuiranim sistemima,
- detaljno modeliranje baze (ovo se odnosi na činjenicu da NoSQL baze nemaju shemu koja se treba detaljno modelirati u slučaju relacijskih baza) nije potrebno što ušteđuje vrijeme i trud.

Neki od nedostataka NoSQL baza podataka:

- prednosti NoSQL baza uzrokuju i jedan od nedostataka – smanjenje ACID (engl. *Atomicity, Consistency, Isolation, Durability*) svojstava. ACID svojstva su svojstva koja svaka akcija baze podataka mora sadržavati kako bi održala preciznost, cjelovitost i integritet podataka. NoSQL baza pruža samo eventualnu dosljednost. Distribuirani sistemi

zadržavaju kopije podataka na nekoliko uređaja kako bi pružali visoku dostupnost i skalabilnost. Kada aplikacija napravi promjenu u podacima na uređaju ta promjena se mora propagirati na ostale kopije. Pošto propagacija nije trenutačna, postoji interval u kojemu neke kopije nemaju najnovije promjene, dok druge imaju (kopije su međusobno nekonzistentne). Promjena će se eventualno primijeniti na sve kopije i od tud dolazi pojam eventualne dosljednosti,

- za razliku od SQL baza, NoSQL bazama nedostaje standardizacija što uzrokuje probleme tijekom migracije,
- međuoperativnost (mogućnost aplikacije da funkcionira sa drugačijim bazama podataka) je isto jedan od problema u slučaju NoSQL baza podataka,
- za razliku pod relacijskih baza pretraživanje podataka je složenije jer ne postoji standardni jezik za pretraživanje kakav postoji za relacijske baze podatka.

NoSQL baze podataka se trebaju koristiti:

- za rukovanje ogromnim brojem strukturiranim, polustrukturiranih i nestrukturiranih podataka,
- kada je potrebno pratiti moderne razvojne prakse i ako se treba napraviti prototip ili brza aplikacija,
- ako preferirate objektno orijentirano programiranje,
- ako relacijske baze nisu dovoljno skalabilne za prihvatljivu cijenu,
- ako imate lokalne transakcije podataka koje ne biti dugotrajne,
- ako imate podatke bez sheme i želite dodavati nova polja bez ikakvih izmjena,
- kada je prioritet lagana skalabilnost i dostupnost [46].

Prednosti specifične *MongoDB* NoSQL bazi podataka su:

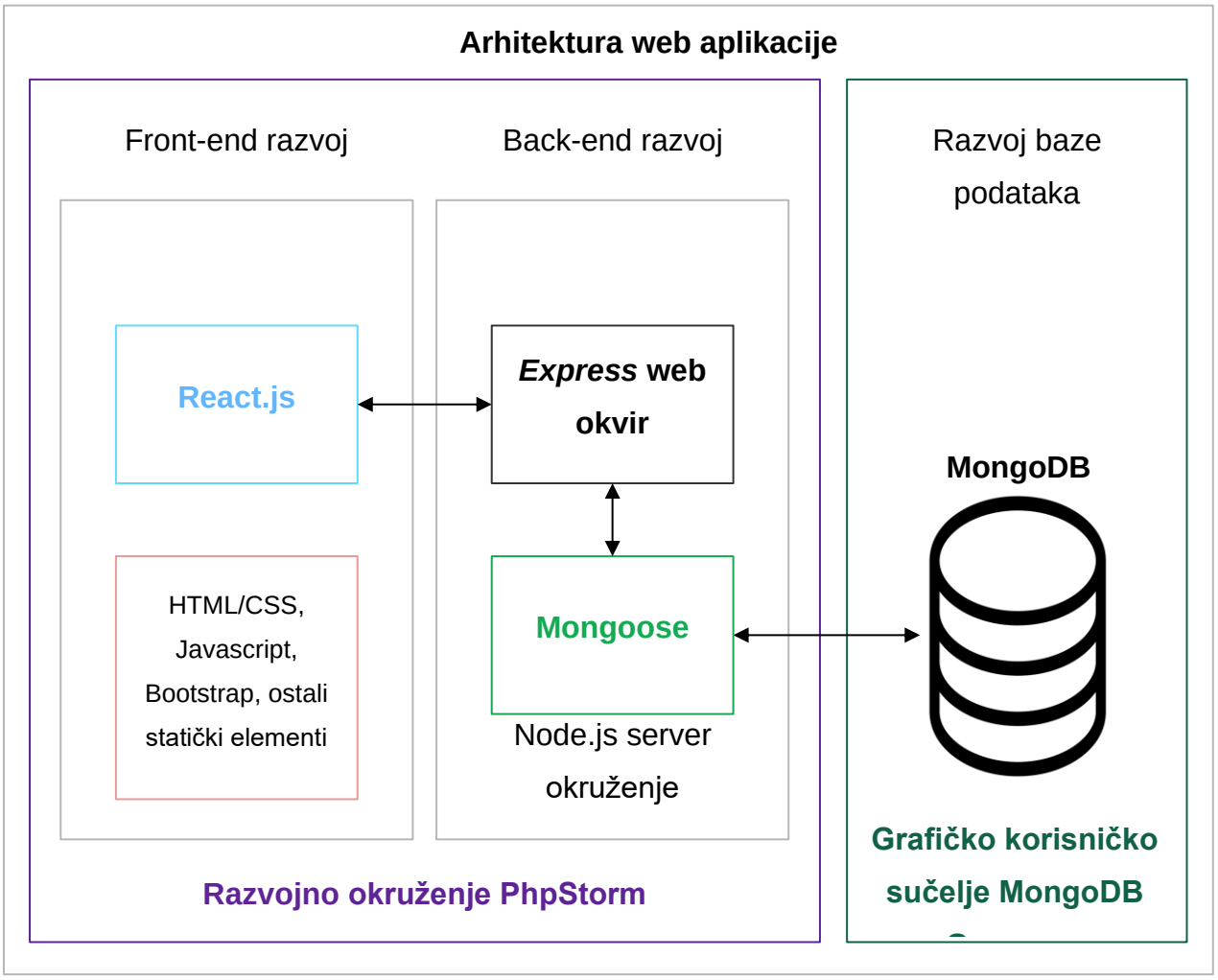
- *MongoDBov* model dokumenta dozvoljava bilo kakvu strukturu podataka i manipulaciju podacima. *Binary* JSON (BSON) format podataka dozvoljava da objekti u jednoj kolekciji imaju drugačije skupove polja (npr. jedan objekt ima polje *featured* dok drugi ne moraju imati),
- Većina baza podataka koriste velike omotače (engl. *wrappers*), poput ORM-a (engl. *Object Relational Mappers*), za pretvaranje podataka u *Object* oblik za korištenje u programima. Odluka *MongoDBa* da sprema i prikazuje podatke u dokument obliku znači da se podacima

može pristupiti iz bilo kojeg jezika u strukturi podataka koja je izvorna tom jeziku (npr. rječnici (engl. *dictionary*) u *Pythonu*, asocijativne liste u *JavaScriptu*),

- Ako je potrebno promijeniti strukturu podataka unutar *MongoDB* baze, ne treba se gasiti cijela stranica ili aplikacija za primijeniti promjene. Nema zastoja pri izmjeni sheme i unos novih podataka se može raditi bilo kada bez zastoja u operaciji.
- *MongoDB* je dizajniran da olakša pristup podacima i rijetko zahtijeva pridruživanja ili transakcije te je vrlo sposoban rukovati kompleksnim upitima. *MongoDB Query Language* (MQL) je cjelovit jezik koji omogućuje upite u dokumente i izvodi kompleksne cjevovode analitike (engl. *analytics pipelines*) sa nekoliko linija MQL-a.
- *MongoDB* je dizajniran da bude distribuirana baza podataka. Mogu se stvarati klasteri (engl. *clusters*) sa trenutačnom replikacijom i podijeliti velike ili visokopropusne kolekcije na više klastera kako bi se održale performanse i vodoravno skaliranje [36].

3 MODEL IZRAĐENE WEB APLIKACIJE

Izrađena *web* aplikacija se sastoji od 3 glavna dijela koji su prikazani na dijagramu Dijagram 1.



Dijagram 1 Arhitektura izrađene *web* aplikacije

Prvi dio uključuje razvoj *front-enda* ili korisničkog sučelja unutar kojega se koristi *React.js* biblioteka i potrebni osnovni elementi i komponente. Drugi dio se odnosi na razvoj baze podataka. U izrađenoj aplikaciji se koristila nerelacijska *MongoDB* baza podataka koja je izrađena pomoću grafičkog sučelja *MongoDB Compass*. Zadnji dio je spajanje baze podataka i grafičkog sučelja koji se naziva *back-end* razvoj ili razvoj pozadinskog sustava. Unutar ovoga dijela se koriste biblioteka za modeliranje objektnih podataka *Mongoose* i *Node.js* okvir *web* aplikacija *Express.js*.

Ovakav način izrade je poznatiji pod nazivom MERN ili *MongoDB, Express.js, React.js* i *Node.js full-stack* izrada *web* aplikacije [34]. Reaktivan pristup razvoju aplikacije se može vidjeti pri izradi sučelja pomoću *React.js* biblioteke. *React.js* koristi komponente, ponašanja (engl. *behaviours*) i događaje (engl. *events*) za izradu sučelja te su to osnovni principi na kojima se bazira reaktivno programiranje. Reaktivno programiranje uvelike ubrzava razvoj korisničkog sučelja te zajedno sa upotrebom *MongoDB* baze podataka dodatno ubrzava cijeli proces izrade *web* aplikacije. U nastavku ovog poglavlja ćemo pobliže opisati glavne elemente svih ovih dijelova.

3.1 Sučelje

Glavni dio sučelja izrađene aplikacije izrađen je korištenjem *React.js* biblioteka. *React.js* je *JavaScript* biblioteka za izgradnju korisničkog sučelja. Ostale komponente važne za spomenuti su *axios* koji se koristi za spajanje na pozadinski sustav, *redux* tj. predvidljivi spremnik stanja za *JavaScript* aplikacije, *nodemailer* koji se koristi za slanje mailova te ostali statični podatci korišteni unutar aplikacije (slike, fontovi, CSS i JS biblioteke) [34].

3.1.1 *React.js*

React.js biblioteka se koristi za izradu korisničkih sučelja. *React* je temeljen na komponentama, što znači da svaka komponenta može postojati odvojeno od drugih i upravlja vlastitim stanjem. Komponente se obično stvaraju pomoću JSX sintakse koju možemo vidjeti na slici 1 ili koristeći metodu *React.createElement* [35]. Korištenja JSX-a omogućuje izgradnju *React* aplikacije deklarativnim načinom. Deklarativan način programiranja se sastoji od podučavanja programa o tome *što* treba učiniti, umjesto da mu se kaže *kako* se to uradi. *React* komponente pružaju mogućnost izoliranja dijelova *web* aplikacije kao dijelove koda ili komponenti koje se mogu ponovno koristiti [34].

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

Slika 1 Primjer JSX sintakse

Komponente *React* biblioteke se mogu podijeliti na više vrsta: funkcijske komponente, komponente klase i izrazi. Funkcijske komponente su obične *JavaScript* funkcije koje prihvaćaju svojstva kao prvi argument i vraćaju *React* elemente (prikazano na slici 2).

```jsx
const InputText = ({ name, children }) => (
    <input
        type="text"
        name={name}
        value={children}
        readOnly
    />
)
```

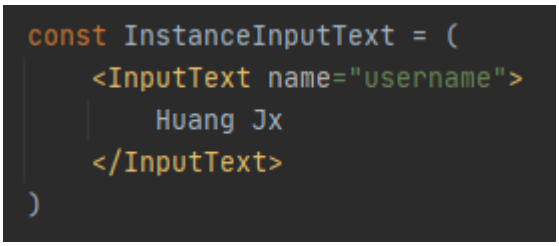
Slika 2 Primjer funkcijske JSX komponente

Komponente klase su zapravo *ECMAScript 6* - ES6 klase koje mogu definirati metode životnog ciklusa i stvoriti komponente stanja. Klase prikazuju *React* elemente pomoću *render* metode što se može vidjeti na slici 3.

```jsx
class InputText extends React.Component {
    render() {
        const { name, children } = this.props
        return (
            <input
                type="text"
                name={name}
                value={children}
                readOnly
            />
        )
    }
}
```

Slika 3 Primjer komponente JSX klase

Izrazi zadržavaju referencu na instancu *React* elementa ili komponente. Primjer izraza je prikazan na slici 4.



Slika 4 Primjer JSX izraza

Učitavanje komponenti u programersko sučelje za HTML i XML dokumente (engl. *Document Object Model* - DOM) je vrlo jednostavno. *React* pruža nekoliko metoda učitavanja komponenti korištenjem *ReactDOM* biblioteke. Korištenjem JSX-a ili *React.createElement* stvara se stablo ili prikaz DOM stabla. To se čini tako da se koristi virtualni DOM pomoću kojeg *React* transformira *React* elemente u DOM čvorove i mijenja samo čvorove koji su se promijenili.

Primjer učitavanja pomoću *ReactDOM* biblioteke se može vidjeti na slici 5 gdje se učitava <*Root/*> komponenta unutar elementa s id-em *root* [34].

Slika 5 Primjer učitavanja *React* komponenti

3.1.2 *Axios*

Axios [36] je *JavaScript* biblioteka koja se koristi za slanje HTTP zahtjeva iz poslužiteljskog okruženja *Node.js* ili *XMLHttpRequesta* iz *web* preglednika koji također podržava ES6 *Promise API* [37]. Korištenjem *axiosa* uklanjamo potrebu slanja rezultata HTTP zahtjeva u *.json()* metodu. On vraća objekt kakav treba biti. Dodatno, bilo kakva greška se uspješno hvata pomoću *.catch()* metode tako da je vrlo lagano otkloniti greške. Neke od najvažnijih značajki *axiosa* su:

- izvodi *XMLHttpRequests* iz *web* preglednika,
- obrađuje HTTP zahtjeve *iz Node.jsa*,
- podržava *Promise* API,
- presreće zahtjeve i odgovore,
- transformira podatke zahtjeva i odgovora,
- otkazuje zahtjeve,
- omogućuje automatsku transformaciju JSON podataka iuz i u *JavaScript* objekte,

- podržava klijentsku zaštitu protiv CSRF/XSRF (*Cross-Site Request Forgery*) napada.

Axios je vrlo koristan jer ubrzava kodiranje određenih HTTP zahtjeva i odgovora na zahtjeve [38]. Primjer korištenja *axiosa* je prikazan na slici 6 na kojoj je se može vidjeti povezivanje s pozadinskim sustavom i dobivanje podataka za proizvode.

```
import axios from 'axios'
const api = axios.create({
    baseURL: 'http://localhost:3000/api',
})


const getAllProducts = () => api.get( url: `/products`)

var _products = [];

getAllProducts().then(products => {
    _products = products.data.data
});


const TIMEOUT = 500

export default {
    getProducts: (cb, timeout) => setTimeout( handler: () => cb(_products),  timeout: timeout || TIMEOUT),
    buyProducts: (payload, cb, timeout) => setTimeout( handler: () => cb(),  timeout: timeout || TIMEOUT)
}
```

Slika 6 Dobivanje podataka iz *back-enda*

3.1.3 *Redux*

Redux [39] je predvidljivi spremnik stanja za *JavaScript* aplikacije. Dopušta programerima da s lakoćom kontroliraju stanje njihovih aplikacija. Pomoću *reduxa* stanje se čini nepromjenjivim. To znači da je moguće pomicanje između određenih stanja aplikacije (prošlih i idućih).

Redux se temelji na tri principa:

- Jedinstveni izvor istine (engl. *source of truth*): sva stanja aplikacije moraju biti spremljena u jednom stablu objekata unutar jednog spremišta.
- Stanje je samo za čitanje: stablo stanja se ne smije izmjenjivati. Stablo stanja se može izmijeniti samo ako se izvrši akcija za izmjenu.

- Promjene se izvršavaju pomoću čistih funkcija: ove funkcije se zovu reduktori (engl. *reducers*). To su funkcije koje primaju prošlo stanje i akciju te potom izračunavaju novo stanje. Reduktori nikad ne smiju mijenjati prošla stanja nego uvijek vraćati nova [34].

3.2 Pozadinski sustav

Pozadinski sustav ove aplikacije se sastoji od modela tablice koji koristi *mongoose* [40] biblioteku da stvori model kolekcije koja se koristi unutar aplikacije (u ovom slučaju to je *product* tablica), kontrolera koji sadrži funkcije koje se mogu vršiti nad dobivenim modelom, ruta koje koriste *Express* biblioteku da bi kreirali zasebne rute unutar našeg pozadinskog sustava i glavnog dokumenta u kojemu je omogućen poslužitelj s kojega će se dobivati podatci unutar sučelja pomoću *axios* poziva spomenutih u prethodnom poglavlju. U sljedećih nekoliko potpoglavlja bit će objašnjeni najvažniji pojmovi pozadinskog sustava: *mongoose*, *Express* i *cors*.

3.2.1 *Mongoose*

Mongoose [40] nudi jednostavno rješenje temeljeno na shemi za modeliranje podataka aplikacije. Uključuje ugrađeno tipiziranje (engl. *type casting*), provjeru valjanosti, izradu upita i još mnogo toga. Unutar izrađene aplikacije se koristi u pozadinskom dijelu sustava pri povezivanju s *MongoDB* bazom podataka što je prikazano na slici 7 i stvaranju modela kolekcije koja se koristi u aplikaciji (Slika 8) [40].

```
const mongoose = require('mongoose')

mongoose
    .connect( uri: 'mongodb://127.0.0.1:27017/electronic_store', options: { useNewUrlParser: true })
    .catch( onrejected: e => {
        console.error('Connection error', e.message)
    })

const db = mongoose.connection

module.exports = db
```

Slika 7 Ostvarivanje povezivosti s *MongoDB* bazom

```
const mongoose = require('mongoose')
require('mongoose-double')(mongoose);
const Schema = mongoose.Schema
var SchemaTypes = mongoose.Schema.Types;

const Products = new Schema(
    {
        name: { type: String, required: true },
        price: { type: SchemaTypes.Double, required: true },
        salePrice: { type: SchemaTypes.Double, required: false },
        discount: { type: Number, required: false },
        pictures: { type: [String], required: false },
        shortDetails: { type: String, required: false },
        description: { type: String, required: false },
        stock: { type: Number, required: false },
        new: { type: Boolean, required: true, default: true },
        sale: { type: Boolean, required: true, default: false },
        category: { type: [String], required: true },
        colors: { type: [String], required: true },
        tags: { type: [String], required: true },
        variants: {
            color: String,
            images: String
        },
    },
)

module.exports = mongoose.model( name: 'product', Products, collection: 'product')
```

Slika 8 Izrada modela kolekcije preuzete iz *MongoDB* baze

3.2.2 *Express*

Express.js [41] je jedan on najboljih *Node.js* okvira za izgradnju robusnih *web* aplikacija i API-ja. *Express* se koristi za stvaranje poslužitelja koji može razumjeti i prihvatiti razne HTTP zahtjeve. Svaki zahtjev očekuje odgovor koji se šalje klijentu da bi on mogao prepoznati status resursa kojeg zahtjeva. Metode zahtjeva mogu biti:

- Sigurne: metode koje rade operacije koje mogu samo čitati podatke na poslužitelju (engl. *read-only*). Jedna od tih operacija je *GET* koja ne izmjenjuje stanje poslužitelja.
- Idempotentne: ovakve operacije imaju jednak efekt na poslužitelja kada se identični zahtjevi pozivaju više puta. Na primjer, slanje *PUT* zahtjeva da se izmjeni korisnikovo ime

bi trebalo imati jednak učinak na poslužitelja kada se nekoliko identičnih zahtjeva pošalje. Metode *GET*, *PUT* i *DELETE* su idempotentne.

- Predmemorijske: Ovo su HTTP zahtjevi koji se mogu spremiti u predmemoriju (engl. *cache*). Postoje metode koje se ne mogu spremiti u predmemoriju. Odgovor je moguće spremiti u predmemoriju samo ako se kod statusa i korištena metoda mogu spremiti u predmemoriju. Na primjer, *GET* metoda i sljedeći statusni kodovi se mogu spremiti u predmemoriju: 200 (Zahtjev uspješan), 204 (Nema konteksta), 206 (Djelomični kontekst), 301 (Trajno premješteno), 404 (Nije pronađeno), 405 (Metoda nije dozvoljena), 410 (Nestalo ili sadržaj trajno otklonjen s poslužitelja) i 414 (URI predug) [34].

Na slici 9 je prikazan primjer korištenja *Expressa* unutar izrađene *web* aplikacije. Prikazani kod se koristi unutar *back-end* sustava za kreiranje poslužitelja koji *front-end* može koristiti za slanje zahtjeva za podatke iz baze podataka.

```
const express = require('express')
const bodyParser = require('body-parser')
const cors = require('cors')

const db = require('./db')
const productRouter = require('./routes/product-routes')

const app = express()
const apiPort = 3000

app.use(bodyParser.urlencoded({ extended: true }))
app.use(cors())
app.use(bodyParser.json())

db.on('error', console.error.bind(console, argArray: 'MongoDB connection error:'))

app.get('/', (req : Request<P, ResBody, ReqBody, ReqQuery> , res : Response<ResBody> ) => {
    res.send( body: 'Hello World!')
})

app.use('/api', productRouter)

app.listen(apiPort, callback: () => console.log(`Server running on port ${apiPort}`))
```

Slika 9 Kreiranje poslužitelja pomoću *Express.jsa*

3.2.3 CORS

CORS [42] je *Node.js* paket za pružanje *Connect/Express* posredničke usluge (engl. *middleware*) koja se može koristiti za omogućavanje CORS-a s raznim opcijama [43]. CORS je mehanizam koji koristi dodatna HTTP zaglavlja kako bi rekao preglednicima da *web* aplikaciji koja radi s jednim podrijetlom daju pristup odabranim resursima iz drugog podrijetla. *Web* aplikacija izvršava HTTP zahtjev s više podrijetla kada zahtijeva resurs koji ima različito podrijetlo (domenu, protokol ili port) od vlastitog [44]. Unutar izrađene aplikacije se koristi da bi se dopustio pristup pozadinskom sustavu pomoću sučelja. Primjer korištenja se može vidjeti na slici 9 unutar prethodnog poglavlja.

3.3 *MongoDB* baza podataka

MongoDB [45] je *open-source*, NoSQL, baza podataka orijentirana na dokumente. Za razliku od relacijskog sustava baze podataka, za *MongoDB* sustav ne postoji shema (struktura opisana u formalnom jeziku) baze podataka. Umjesto toga, podaci se pakiraju unutar JSON objekta, a taj objekt se sprema unutar *MongoDB* baze u obliku *JavaScript* objekta (BSON). Ova sposobnost izravnog rada s *JavaScriptom* je jedna od najkorisnijih značajki *MongoDBa* i razlog njegove popularnosti među programerima [1].

4 RAZVOJ KORISNIČKE APLIKACIJE

U ovom poglavlju opisan je postupak razvoja reaktivne aplikacije. Prvi dio izrade aplikacije je dizajniranje tj. izrada izgleda korisničkog sučelja aplikacije. Nakon dizajna slijedi izrada funkcionalnosti korisničkog sučelja te potom izrada baze podataka. Zadnja dva dijela opisuju izradu pozadinskog sustava za povezivanje s bazom podataka i funkcionalnosti spajanja sučelja s pozadinskim dijelom sustava. Razvoj reaktivne aplikacije za razliku od klasičnih ili nereaktivnih aplikacija je u tomu što se unutar reaktivnog razvoja oslanjamo na događaje i ponašanja (*behaviours and states*) umjesto reda linija u kodu. Reaktivno programiranje je programiranje pomoću asinkronih tokova podataka što znači da će se svaki događaj, ponašanje, poziv, poruke i greške prikazati pomoću toka podataka. Asinkroni tokovi su tokovi koji propuštaju vrijednosti jednu iza druge sa određenim zadržavanjem između vrijednosti.

4.1 Dijelovi i funkcionalnosti korisničke aplikacije

U praktičnom dijelu ovog rada napravljena je aplikacija za e-trgovinu računalnom opremom. Pomoću izrađene aplikacije se mogu naručivati razni dijelovi računala i oprema za računala. Unutar same aplikacije je moguće uspoređivati pojedinačne proizvode, stavljati ih u listu želja ili u košaricu te na stranici plaćanja naručiti dijelove. Kao primjer reaktivnog pristupa programiranju korištenjem *React* komponenti izrađen je svaki dio *web* aplikacije:

- zaglavlje i podnožje *web* aplikacije,
- početna stranica,
- o nama stranica,
- stranica trgovina,
- stranica pojedinog proizvoda,
- kontakt stranica,
- košarica stranica,
- stranica plaćanja,
- stranica usporedbe,
- stranica željenih proizvoda,
- stranica uspješnog plaćanja,
- stranica 404 za URL adrese koje ne postoje u *web* aplikaciji.

Kao što su svi dijelovi aplikacije izrađeni pomoću reaktivnog pristupa, tako su omogućene i sljedeće funkcionalnosti:

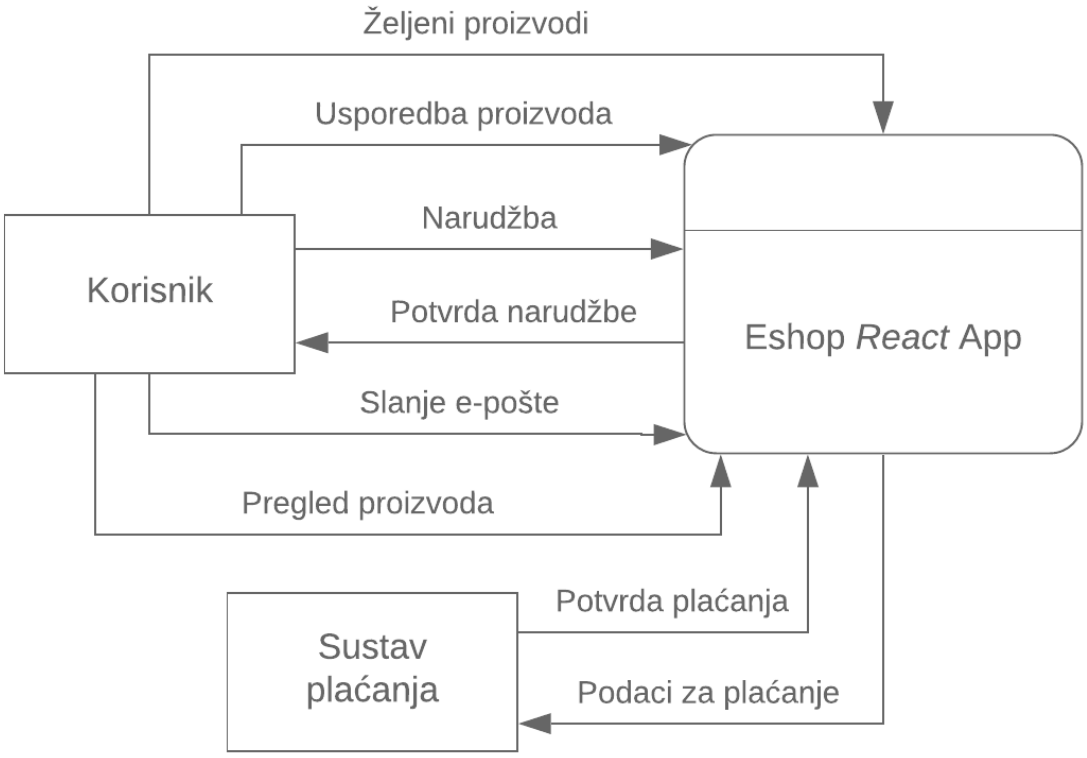
- odabir trenutne stranice,
- upravljanje jezikom stranice,
- pregled košarice u zaglavlju stranice,
- upravljanje klizačem na početnoj stranici,
- mijenjanje kategorije proizvoda na početnoj stranici (novo, izdvojeno i posebno),
- filtriranje proizvoda po marci, kategoriji, boji ili cijeni, sortiranje proizvoda po cijeni (od više ili niže), novini ili imenu (A-Z ili Z-A), mijenjanje broja kolumni i prikaza (lista ili mreža) proizvoda, stavljanje određenog proizvoda u košaricu, listu želja ili listu usporedbe na stranici trgovine i pregled novih proizvoda pomoću klizača unutar stranice trgovine,
- prikaz mape lokacije trgovine, validacija i slanje e-pošte na kontakt stranici,
- odabir količine proizvoda, stavljanje u košaricu ili listu želja i mogućnost odlaska na stranicu za plaćanje sa stranice pojedinačnog proizvoda,
- brisanje proizvoda, odlazak na stranicu trgovine ili na stranicu plaćanja na stranici košarice,
- brisanje proizvoda sa liste, stavljanje proizvoda u košaricu, odlazak na stranicu trgovine ili plaćanja sa stranice željenih proizvoda,
- prikaz liste usporedbe proizvoda, brisanje proizvoda iz liste i dodavanje proizvoda u košaricu na stranici usporedbe,
- validacija podataka i plaćanje pomoću *Stripe* ili *PayPal* sustava za online plaćanje.

Postoje također određene funkcionalnosti i stranice koje nisu realizirane kroz izradu aplikacije:

- registracija i prijava korisničkog računa,
- stranica korisničkog profila sa podacima korisnika, listom prošlih narudžbi i mogućnostima izmjene podataka,
- dodavanje na listu pretplatnika e-pošte,
- pisanje recenzija za pojedini proizvod,
- blog stranica,
- CMS sustav za izmjenu podataka spomenutih u prethodnim nerealiziranim funkcionalnostima kao i za izmjenu podataka proizvoda.

Dijagram 2 prikazuje kako sistem kao cjelina komunicira sa vanjskim entitetima. Izrađena reaktivna aplikacija komunicira sa dva vanjska entiteta, korisnikom i sustavom za plaćanje, tako da je na dijagramu prikazana veza između sustava reaktivne aplikacije i tih entiteta. Na dijagramu se vidi kako korisnik može izvršiti narudžbu proizvoda, uspoređivati proizvode,

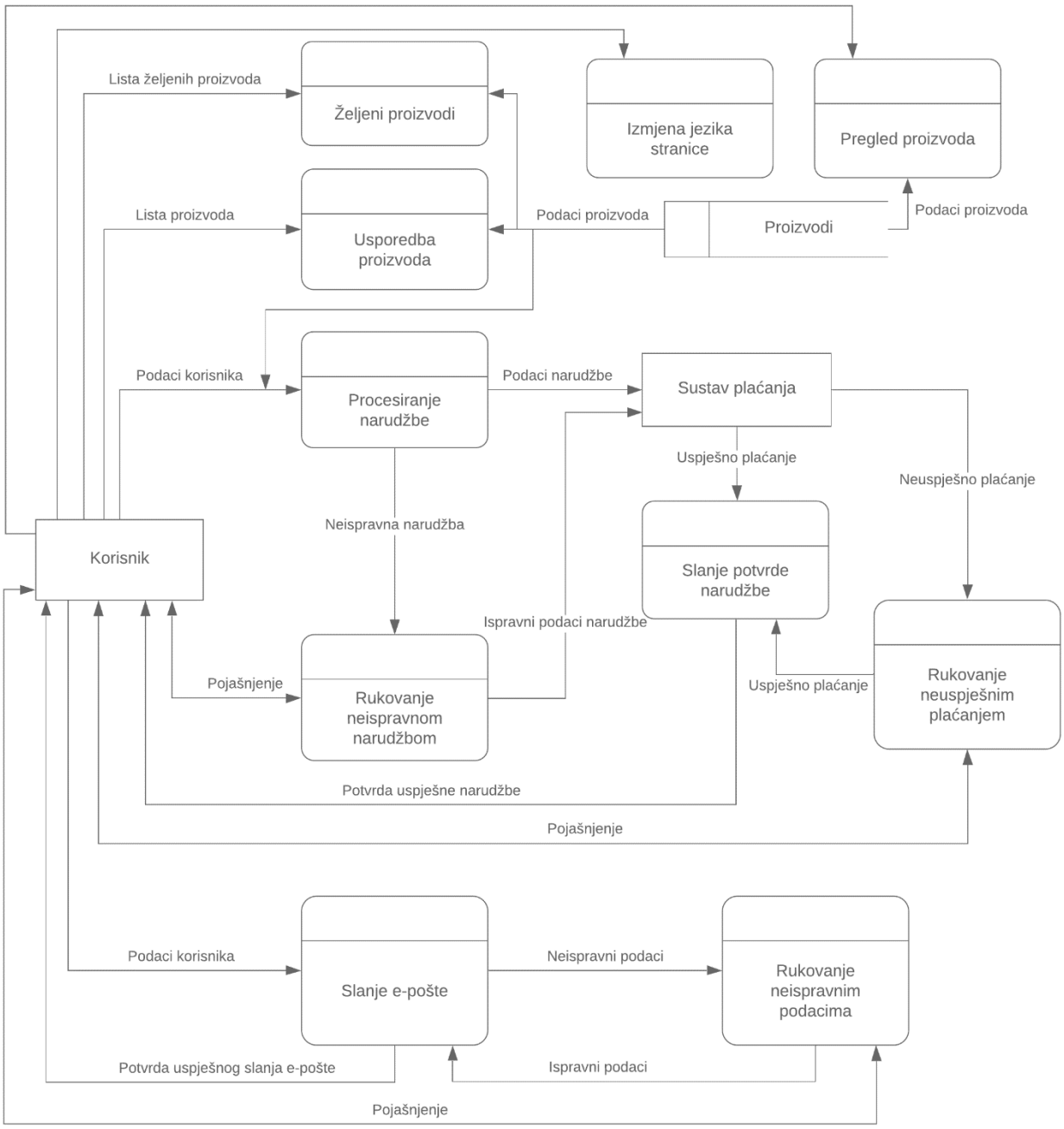
odabrati željene proizvode i slati e-poštu. Aplikacija vraća povratnu informaciju u slučaju narudžbe koja se vidi na dijagramu kao *Potvrda narudžbe*. Korisnikovi podaci za plaćanje se šalju unutar aplikacije na sustav za plaćanje, a sustav šalje aplikaciji potvrdu plaćanja nakon koje se korisniku šalje potvrda narudžbe.



Dijagram 2 Dijagram konteksta

Dijagram 3 pruža detaljniji prikaz procesa između sistema aplikacije i entiteta. Prikazuju se veze i procesi koji su bili prikazani u prošlom dijagramu između entiteta i sistema aplikacije samo pomoću proširenog prikaza. Korisnik može mijenjati listu proizvoda za usporedbu ili listu željenih proizvoda. Unutar lista se prikazuju podaci proizvoda koji se dobivaju iz baze podataka. Za izradu narudžbe su potrebni podaci korisnika kao i podaci proizvoda koji se naručuju. Korisnikovi podaci se provjeravaju te ako su ispravni odlazi se na sustav plaćanja, a ako su neispravni odlazi se na proces za rukovanje neispravnim podacima. Taj proces obavještava korisnika da su podaci neispravni te da ih je potrebno ispraviti. Ako su podaci ispravni odlazi se na sustav plaćanja gdje se u slučaju neuspješnog plaćanja ponovno obavještava korisnika za ispravak podataka, a u slučaju uspješnog plaćanja šalje se potvrda narudžbe te se obavještava korisnika o uspješnoj narudžbi. Ako korisnik želi slati e-poštu upisuje podatke te se u slučaju ispravnih podataka uspješno šalje e-pošta te se korisnika obavijesti o uspješnom slanju e-pošte, a

u slučaju neispravnih podataka obavještava se korisnika za ispravak podataka i pokušaj ponovnog slanja.



Dijagram 3 Dijagram toka podataka prve razine

4.2 Dizajn korisničkog sučelja

Dizajn sučelja treba biti što jednostavniji. U izrađenoj *web* aplikaciji korišten je postojeći *bootstrap* CSS okvir za izradu osnovnog izgleda stranice. *Bootstrap* sadrži vlastite CSS i JS dokumente pomoću kojih se može lagano mijenjati izgled i funkcionalnost elemenata pomoću imena klasa. Za dodatne izmjene izgleda su napisani posebni SASS dokumenti koji se pomoću *Node.jsa* kompiliraju u CSS dokument. SASS dokumenti su posebni preprocesirani CSS dokumenti koji mogu koristiti varijable, ugniježđena pravila, funkcije i druge dijelove koji su potpuno kompatibilni s CSS-om i omogućavaju puno brže pisanje CSS-a [8]. Također se koriste različiti fontovi, paketi ikona i slike da bi se upotpunio dizajn.

Unutar reaktivnog načina izrade stranice koriste se *React* komponente za stvaranje različitih izgleda i podstranica. Kako je prikazano na slici 10, za izradu početne stranice korištene su *header*, *footer* i *SpecialProducts* komponente koje također u sebi imaju različite komponente. Komponente *Slider* i *Helmet* su uzete iz npm paketa kojima su izgled i funkcionalnost već napravljeni. Ovaj način izrade stranica je jedan od glavnih razloga zbog kojih je *React* jedna od najkorištenijih biblioteka za izradu korisničkih sučelja. Kod je puno pregledniji i vrlo lako je izmjenjivati izgled na svim stranicama izmjenom određene komponente koja se koristi.

```
import React, { Component } from 'react';
import {Helmet} from 'react-helmet'
import '../../common/index.scss';
import Slider from 'react-slick';
// Import custom components
import SpecialProducts from "./special-products"
import {getHighestDiscount} from '../../../services'
import {connect} from "react-redux";
import withTranslate from "react-redux-multilingual/lib/withTranslate";
```

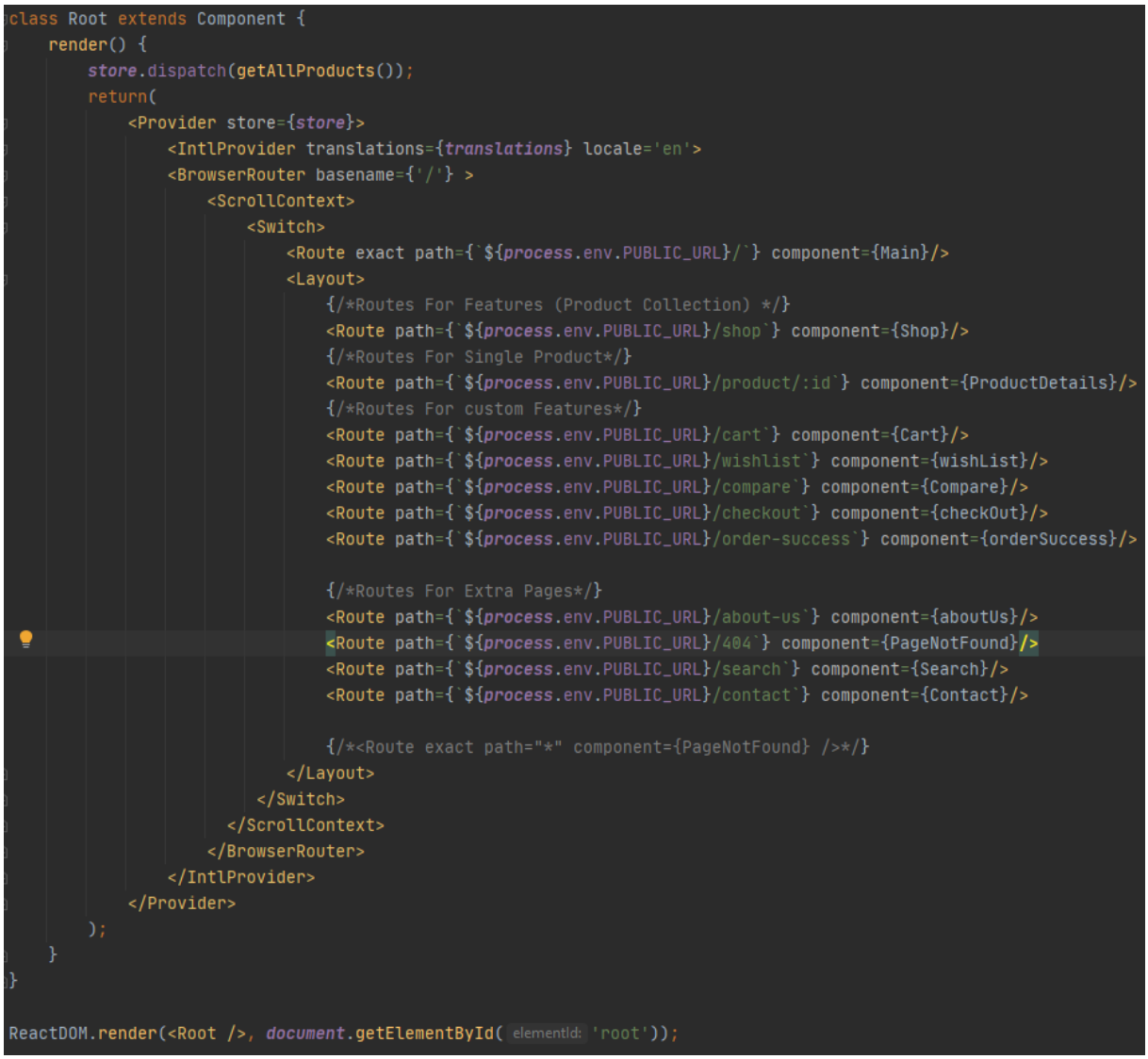
Slika 10 Prikaz komponenti početne stranice

4.3 Funkcionalnost korisničkog sučelja

Nakon što se isprogramirao dizajn korisničkog sučelja, sljedeći dio je programiranje funkcionalnosti korisničkog sučelja. Funkcionalnosti se mogu dobiti iz predodređenih paketa pomoću npm-a ili se mogu napisati sasvim nove. Na slici 10 vidljive su različite funkcionalne

komponente poput *getHighestDiscount*, *connect* i *withThranslate*. *getHighestDiscount* je ručno napisana funkcionalna komponenta dok su ostale dvije uzete iz učitanih paketa. Kao što se može vidjeti na slici paket koji se koristi je *react-redux*. Da bi koristili *redux,* u samom *rootu* aplikacije se inicijalizira *redux store* u kojega spremamo sve bitne podatke (u ovom slučaju sve proizvode). Ti podaci se nadalje mogu koristiti u bilo kojoj komponenti koristeći *redux connect* funkciju. Dodatno se koristi funkcija *withTranslate* koja omogućava korištenje višejezične podrške na *web* aplikaciji.

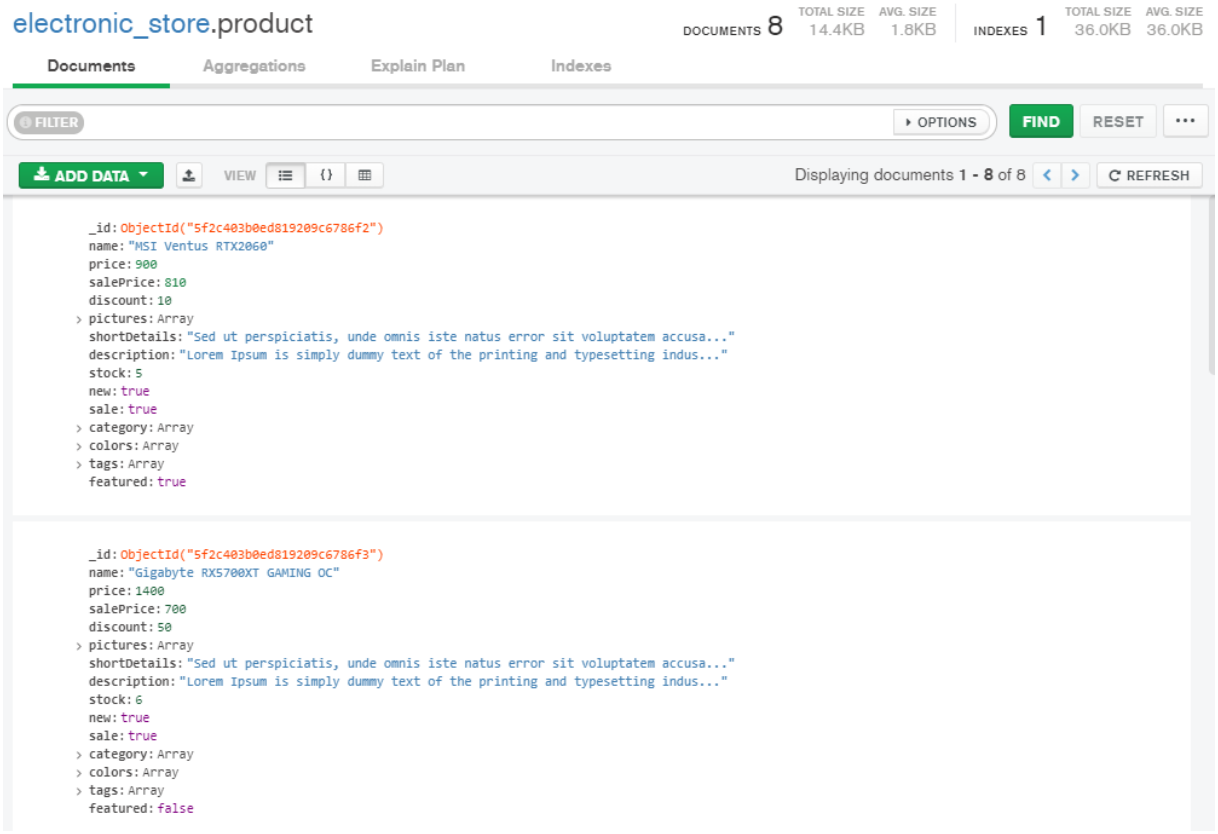
Još jedan važan paket je *react-router-dom* koji nam omogućava korištenje navigacijskih komponenti unutar aplikacije. Na slici 11 je prikazan način korištenja komponenti iz *react-router-dom* paketa za stvaranje svih ruta u aplikaciji. Ovaj paket se koristi kroz cijelu aplikaciju za kreiranje navigacijskih linkova, ne samo unutar *Root* komponente. Određene funkcije koje su specifične za određenu stranicu se mogu pisati unutar samog JSX-a, dok se često korištene pišu unutar funkcijskih komponenti za lako korištenje u aplikaciji.



Slika 11 Prikaz *'Root'* komponente aplikacije

4.4 Baza podataka

Razvoj baze podataka počinje tako da se izrade testni podaci unutar aplikacije da bi saznali koji su nam podaci potrebni za stvaranje baze podataka. Za *MongoDB* bazu podaci se upisuju unutar JSON dokumenta i učitavaju pomoću *Compass* grafičkog sučelja ili se upisuju pomoću zapovijedi u terminalu. Primjer kolekcije *MongoDB* baze je prikazan na slici 12 na kojoj su prikazani podaci proizvoda korištenih unutar aplikacije.



Slika 12 Prikaz podataka unutar *MongoDB* kolekcije

4.5 Pozadinski sustav

Unutar pozadinskog sustava se ostvaruje veza s *MongoDB* bazom. Kreira se model kolekcije koja se vuče iz baze podataka. Slika 8 iz poglavlja *Mongoose* prikazuje primjer kreiranog modela. Nakon što je kreiran model stvaraju se funkcije za obradu dobivenog modela popularnije poznate kao CRUD (engl. *Create-Read-Update-Delete*) funkcije. Za korisničko sučelje je potrebna samo *read* funkcija tj. potrebno je samo pročitati podatke iz baze i spremiti ih kao što je prikazano na slici 13. Za pravi pozadinski sustav bi se trebalo kreirati sučelje u kojem se mogu raditi sve CRUD funkcije nad proizvodima.

```
getProducts = async (req, res) => {
    await Product.find({}, (err, Products) => {
        if (err) {
            return res.status(400).json({ success: false, error: Product })
        }
        if (!Products.length) {
            return res
                .status(404)
                .json({ success: false, error: `Product not found` })
        }
        return res.status(200).json({ success: true, data: Products })
    }).catch( onrejected: err => console.log(err))
}
```

Slika 13 Funkcije čitanja podataka iz MongoDB baze

Nakon što su napravljene funkcije za čitanje potrebno je kreirati rute preko kojih će se podaci dohvatiti. Rute se stvaraju pomoću operacija *Express* paketa koristeći *Router()* objekt koji omogućava korištenje različitih HTTP zahtjev metoda (GET, PUT, POST…) koje se mogu koristiti unutar korisničkog sučelja za dohvat podataka. Primjer se može vidjeti na slici 14 na kojoj je prikazano stvaranje različitih ruta za razne operacije koje se obavljaju nad podacima.

```
const express = require('express')

const ProductCtrl = require('../controllers/product-ctrl')

const router = express.Router()

router.post( path: '/product', ProductCtrl.createProduct)
router.put( path: '/product/:id', ProductCtrl.updateProduct)
router.delete( path: '/product/:id', ProductCtrl.deleteProduct)
router.get( path: '/product/:id', ProductCtrl.getProductById)
router.get( path: '/products', ProductCtrl.getProducts)

module.exports = router
```

Slika 14 Primjer stvaranja ruta pomoću *Express* paketa

Posljednji korak je napisati skriptu za stvaranje poslužitelja preko kojega će se posluživati stvorene rute. Koriste se *Express* i *cors* paketi koji zajedno omogućuju korisničkom sučelju uspješan dohvat podataka. *Express* paket se koristi za stvaranje poslužitelja dok *cors* služi da bi se dozvolio pristup poslužitelju od vanjskih izvora, u ovom slučaju izrađene *web* aplikacije. Slika 15 sadrži prikaz skripte korištene za kreiranje poslužitelja.

```
const express = require('express')
const bodyParser = require('body-parser')
const cors = require('cors')

const db = require('./db')
const productRouter = require('./routes/product-routes')

const app = express()
const apiPort = 3000

app.use(bodyParser.urlencoded({ extended: true }))
app.use(cors())
app.use(bodyParser.json())

db.on('error', console.error.bind(console, argArray: 'MongoDB connection error:'))

app.get('/', (req : Request<P, ResBody, ReqBody, ReqQuery> , res : Response<ResBody> ) => {
    res.send( body: 'Connection successfull')
})

app.use('/api', productRouter)

app.listen(apiPort, callback: () => console.log(`Server running on port ${apiPort}`))
```

Slika 15 Back-end skripta za stvaranje servera i dohvat podataka

4.6 Funkcionalnost između sučelja i pozadinskog sustava

Za izradu funkcionalnosti između sučelja i pozadinskog sustava se koristi prethodno spomenuti *Axios*. U prethodnom poglavlju je objašnjeno kako pozadinski sustav omogućuje spajanje na poslužitelj. *Axios* koristi HTTP zahtjeve da bi pristupio poslužitelju i dohvatio ili izmijenio podatke. Na slici 16 prikazan je primjer spajanja na pozadinski poslužitelj za dohvaćanje svih proizvoda pomoću GET metode.

```
import axios from 'axios'
const api = axios.create({
    baseURL: 'http://localhost:3000/api',
})


const getAllProducts = () => api.get( url: `/products`)

var _products = [];

getAllProducts().then(products => {
    _products = products.data.data
});


const TIMEOUT = 500

export default {
    getProducts: (cb, timeout) => setTimeout( handler: () => cb(_products),  timeout: timeout || TIMEOUT),
    buyProducts: (payload, cb, timeout) => setTimeout( handler: () => cb(),  timeout: timeout || TIMEOUT)
}
```

Slika 16 *Axios* skripta za dohvat podataka

4.7 Optimizacija i ispravljanje grešaka

Zadnji dio svakoga razvoja aplikacije je pregled izgleda i funkcionalnosti te ispravljanje grešaka i optimiziranje sadržaja. Ispravljanje grešaka može se odnositi na krivi izgled dizajna ili na funkcionalnost koja ne radi ispravno ili ne obavlja zadatak koji treba ispuniti.

Optimizacija se odnosi na kompresiju podataka, posebice CSS i JS datoteka i slika. Optimiziranje slika se može uraditi preko alata na internetu ili paketa unutar *Node.jsa*, a kompresija CSS i JS datoteka se obično radi automatski nakon što se izradi aplikacija za produkciju.

Za optimiziranje slika ove aplikacije je korišten *Kraken.io* [49], online aplikacija za kompresiju slika, a za optimizaciju CSS i JS datoteka je korišten *webpack* paket koji se koristi za pakiranje statičnih modula (npr. CSS i JS datoteke). Pomoću *webpack* paketa sve neoptimizirane SASS i CSS datoteke se optimiziraju i kompresiraju zajedno u *chunk.css* datoteke. Isto kao i za SASS i CSS datoteke, JS datoteke se na isti način pakiraju u *chunk.js* datoteke.

5 PROGRAMSKI KOD REAKTIVNOG SUČELJA

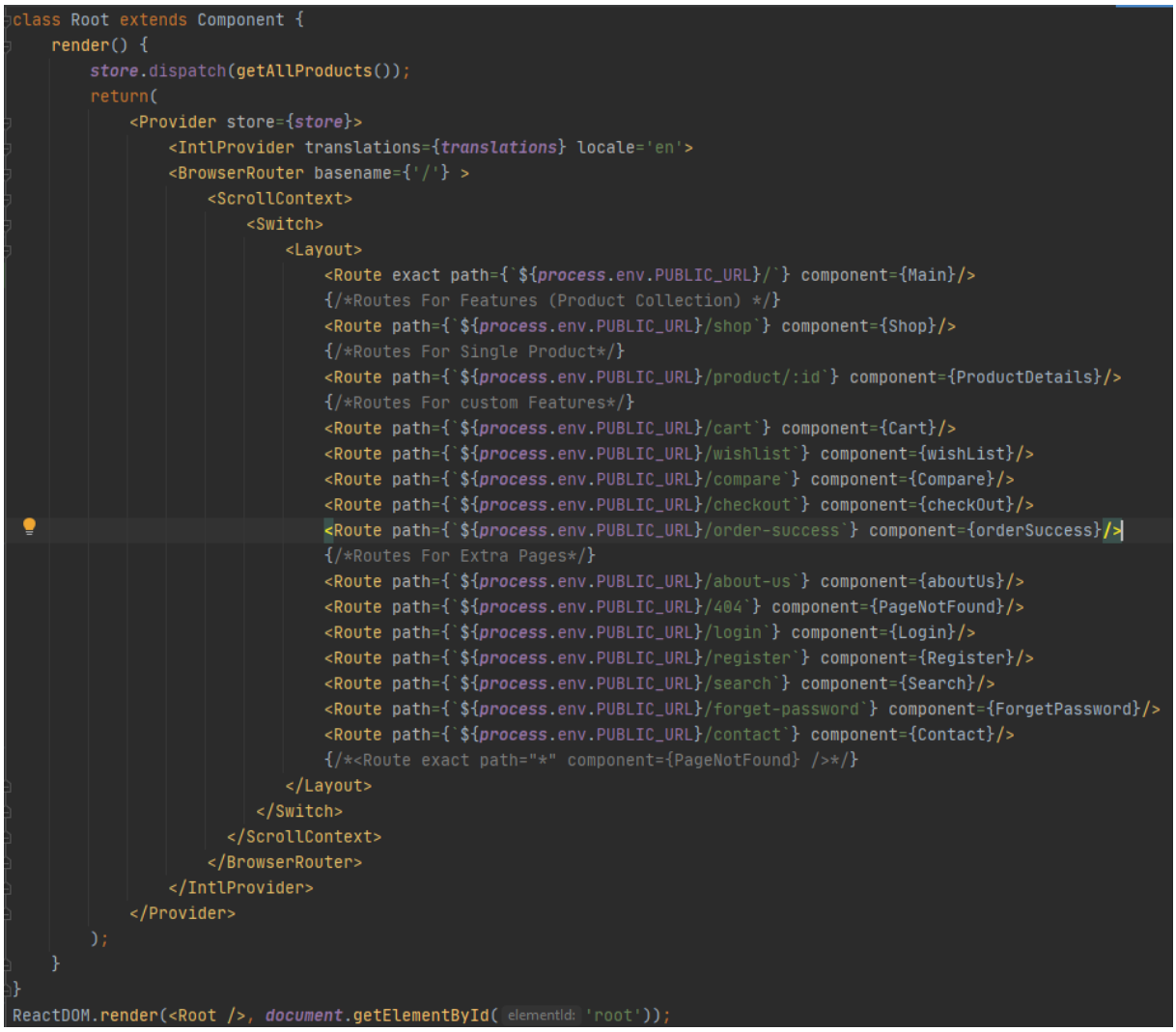
U ovom poglavlju opisat će se programski kod korisničkog sučelja. Pregledati će se sve različite komponente aplikacije te opisati način na koji su povezane unutar same aplikacije.

5.1 Glavna skripta

Glavna skripta je skripta koja učitava sav sadržaj korisničke aplikacije. Na slici 17 se vide sve učitane komponente unutar skripte. Pomoću *Provider* komponente koja se učita od *react-redux* paketa se pružaju usluge kontroliranja stanja pomoću spremnika stanja (*store*). *React-router-dom* pruža usluge usmjeravanja pomoću tri komponente: *BrowseRouter, Route* i *Switch*. *Layout* komponenta što se može vidjeti na slici 19 se koristi kao glavni dizajn stranice. Učitava *header* i *footer* komponente te se svaka komponenta *Route* prikazana na slici 18 učitava kroz ovu komponentu. Pomoću *react-redux-multilingual* paketa se pruža komponenta *IntlProvider* koji se koristi za izmjenu jezika na stranici ovisno o *locale* varijabli se iz *translate* varijable dohvaćaju prijevodi riječi.

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';
import { Provider } from 'react-redux';
import { BrowserRouter, Route, Switch } from 'react-router-dom';
import { ScrollContext } from 'react-router-scroll-4';
import { IntlReducer as Intl, IntlProvider } from 'react-redux-multilingual'
import './index.scss';

// Import custom components
import store from './store';
import translations from './constants/translations'
import { getAllProducts } from './actions'

// Layouts
import Main from './components/layouts/main/main';

//Collection Page
import Shop from "./components/collection/shop";

// Product Pages
import ProductDetails from "./components/products/product-details";

// Features
import Layout from './components/app'
import Cart from './components/cart'
import Compare from './components/compare/index'
import wishList from './components/wishlist'
import checkOut from './components/checkout'
import orderSuccess from './components/checkout/success-page'

// Extra Pages
import aboutUs from './components/pages/about-us'
import PageNotFound from './components/pages/404'
import Login from './components/pages/login'
import Register from './components/pages/register'
import Search from './components/pages/search'
import ForgetPassword from './components/pages/forget-password'
import Contact from './components/pages/contact'
```

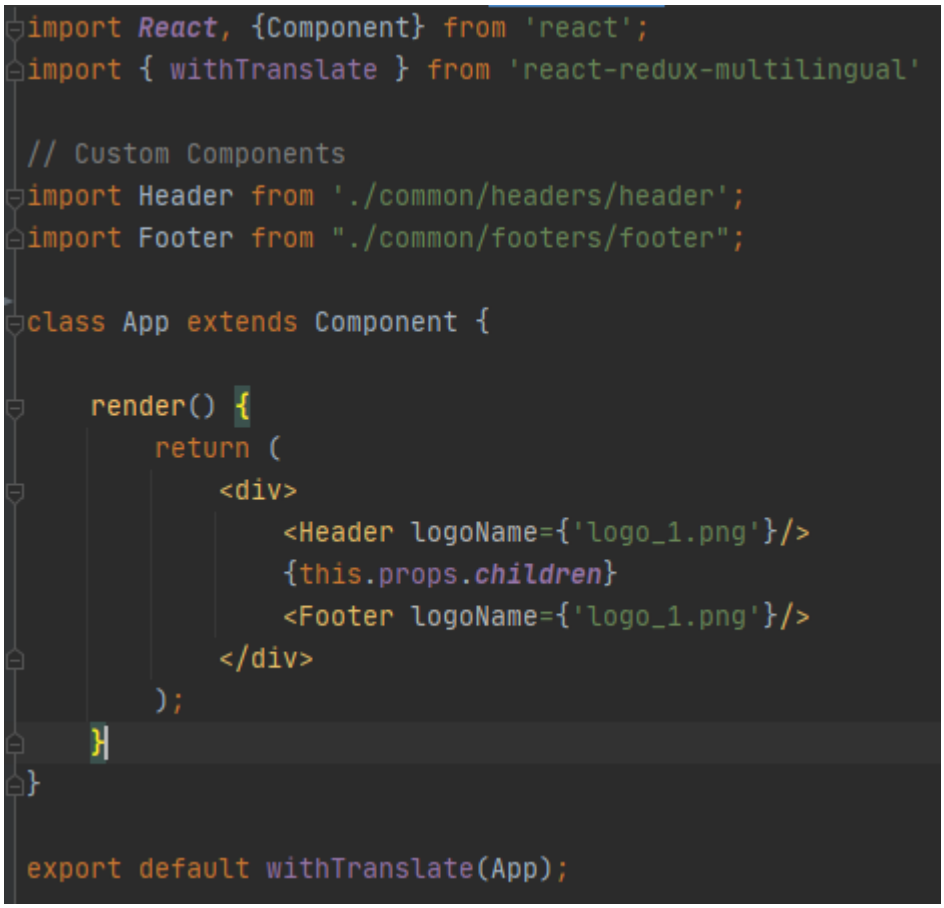
Slika 17 Komponente glavne skripte



Slika 18 Sadržaj glavne skripte

5.2 App

App komponenta (Slika 19) se koristi kao spremnik unutar kojega učitavamo sve stranice aplikacije. U sebi sadrži *header* i *footer* komponentu, a pomoću *redux* paketa se stranice spremaju unutar *props.children* varijable. Varijabla *logoName* unutar *header* i *footer* komponenti se koristi za učitavanje određene slike pomoću *LogoImage* komponente.



Slika 19 App komponenta

5.3 Zaglavlje

Unutar *header* komponente se nalaze razne funkcije i JSX sadržaj *web* stranice. Na slici 20 prva funkcija *constructor* se koristi prije učitavanja stranice za inicijalizaciju određenih stanja ili za kontrolu događaja na stranici (*event handler*). *ComponentDidMount* funkcija se poziva odmah nakon što se učita komponenta. U slučaju *header* komponente se koristi za sakriti *loader-wrapper* element i postaviti varijablu *open* na *true*. *ComponentWillMount* je funkcija koja se poziva prije nego što se učita komponenta te se u zaglavlju koristi za dodavanje slušatelja *scroll* događaja. *ComponentWillUnmount* se pozove prije nego što se komponenta izbriše. U zaglavlju se koristi za brisanje slušatelja *scroll* događaja. *HandleScroll* funkcija služi za animaciju zaglavlja kada se stranica pomiče prema dolje.

```
constructor(props) {
    super(props);

    this.state = {
        isLoading:false
    }
}
/*=====================
     Pre loader
     ==========================*/
componentDidMount() {
    setTimeout( handler: function() {
        document.querySelector( selectors: ".loader-wrapper").style = "display: none";
    }, timeout: 2000);

    this.setState( state: { open: true });
}

componentWillMount(){
    window.addEventListener( type: 'scroll', this.handleScroll);
}
componentWillUnmount() {
    window.removeEventListener( type: 'scroll', this.handleScroll);
}

handleScroll = () => {
    let number = window.pageXOffset || document.documentElement.scrollTop || document.body.scrollTop || 0;
    if (number >= 300) {
        if (window.innerWidth < 576) {
            document.getElementById( elementId: 'offset-top').style.marginTop = 0;
            document.getElementById( elementId: "sticky").classList.remove( tokens: 'fixed');
        }else
            document.getElementById( elementId: 'offset-top').style.marginTop = '186px';
            document.getElementById( elementId: "sticky").classList.add('fixed');
    } else {
        document.getElementById( elementId: 'offset-top').style.marginTop = 0;
        document.getElementById( elementId: "sticky").classList.remove( tokens: 'fixed');
    }
}
```

Slika 20 Funkcije zaglavlja

Na slici 21 su prikazane ostale funkcije zaglavlja. *ChangeLanguage* je funkcija koja služi za mijenjanje jezika stranice pomoću *react-redux-multilanguage* paketa. *OpenNav* funkcija služi za otvaranje *mySidenav* elementa. *OpenSearch* i *closeSearch* služe za otvoriti ili zatvoriti *search-overlay* element. *Load* funkcija služi za kontroliranje *Pace* komponente koja služi za automatsku kontrolu učitavanja stranice.

```
changeLanguage(lang) {
    store.dispatch(IntlActions.setLocale(lang))
}

openNav() {
    var openmyslide = document.getElementById( elementId: "mySidenav");
    if(openmyslide){
        openmyslide.classList.add('open-side')
    }
}
openSearch() {
    document.getElementById( elementId: "search-overlay").style.display = "block";
}

closeSearch() {
    document.getElementById( elementId: "search-overlay").style.display = "none";
}

load = ()=>{
    this.setState( state: {isLoading: true});
    fetch().then(()=>{
        // deal with data fetched
        this.setState( state: {isLoading: false})
    })
};
```

Slika 21 Ostale funkcije zaglavlja

Na slikama 22, 23 i 24 prikazan kod sadržaja komponente zaglavlja. Sastoji se od JSX sintakse u kojoj se nalazi HTML kod i različite komponente koje se učitavaju pomoću JSX sintakse. Unutar zaglavlja (izgled prikazuje 55) se nalazi prije spomenuta *Pace* komponenta te *TopBar, SideBar, NavBar, LogoImage* i *CartContainer* komponenta. *TopBar* sadrži elemente koji se pojavljuju u samom vrhu zaglavlja, *SideBar* i *Navbar* su komponente navigacije, *SideBar* se pojavljuje klikom na hamburger ikonicu, a *NavBar* se nalazi kao glavna navigacija u zaglavlju. *LogoImage* služi za ispisivanje logo slike na unutar zaglavlja a *CartContainer* sadrži elemente za ispisivanje sadržaja košarice unutar zaglavlja i ikonicu koja je prikazana unutar navigacije zaglavlja.

```
<header id="sticky" className="sticky">
    {this.state.isLoading ? <Pace color="#27ae60"/> : null}
    <div className="mobile-fix-option"></div>
    {/*Top Header Component*/}
    <TopBar/>

    <div className="container">
        <div className="row">
            <div className="col-sm-12">
                <div className="main-menu">
                    <div className="menu-left">
                        <div className="navbar">
                            <a href="javascript:void(0)" onClick={this.openNav}>
                                <div className="bar-style"> <i className="fa fa-bars sidebar-bar" aria-hidden="true"></i></div>
                            </a>
                            {/*SideBar Navigation Component*/}
                            <SideBar/>
                        </div>
                        <div className="brand-logo">
                            <LogoImage logo={this.props.logoName} />
                        </div>
                    </div>
                    <div className="menu-right pull-right">
                        {/*Top Navigation Bar Component*/}
                        <NavBar/>
```

Slika 22 Sadržaj *Header* komponente

```
<div>
    <div className="icon-nav">
        <ul>
            <li className="onhover-div mobile-search">
                <div><img src={`${process.env.PUBLIC_URL}/assets/images/icon/search.png`} onClick={this.openSearch}
                          className="img-fluid" alt="" />
                    <i className="fa fa-search" onClick={this.openSearch}></i></div>
            </li>
            <li className="onhover-div mobile-setting">
                <div><img src={`${process.env.PUBLIC_URL}/assets/images/icon/setting.png`} className="img-fluid" alt="" />
                    <i className="fa fa-cog"></i></div>
                <div className="show-div setting">
                    <h6>{translate('language')}</h6>
                    <ul>
                        <li><a href={null} onClick={() => this.changeLanguage( lang: 'en')}>English</a> </li>
                        <li><a href={null} onClick={() => this.changeLanguage( lang: 'hr')}>Hrvatski</a> </li>
                    </ul>
                </div>
            </li>
            {/*Header Cart Component */}
            <CartContainer/>
        </ul>
    </div>
</div>
```

Slika 23 Sadržaj *Header* komponente

```
<div id="offset-top"></div>
<div id="search-overlay" className="search-overlay">
    <div>
        <span className="closebtn" onClick={this.closeSearch} title="Close Overlay">×</span>
        <div className="overlay-content">
            <div className="container">
                <div className="row">
                    <div className="col-xl-12">
                        <form>
                            <div className="form-group">
                                <input type="text" className="form-control" id="exampleInputPassword1" placeholder="Search a Product" />
                            </div>
                            <button type="submit" className="btn btn-primary"><i className="fa fa-search"></i></button>
                        </form>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
```

Slika 24 Sadržaj *Header* komponente

5.4 Podnožje

Footer komponenta, kao i komponenta zaglavlja, sadrži *LogoImage* komponentu te još sadrži *SlideUpDown* funkcionalnu komponentu i kao svaka druga stranica *withTranslate* komponentu. *SlideUpDown* komponenta stvara elemente koji se otvaraju i zatvaraju na klik zaglavlja tih elemenata. Ta funkcionalna komponenta se učitava samo ako je stranica ispod 750 piksela širine koristeći *componentDidMount* funkciju. Sve ove komponente možemo vidjeti na slikama 25, 26 i 27 dok se izgled komponente podnožja na *web* stranici vidi na slici 56.

```
import React, {Component} from 'react';
import { Link} from 'react-router-dom';

import {SlideUpDown} from "../../../services/script"
import LogoImage from "../headers/common/logo"
import withTranslate from "react-redux-multilingual/lib/withTranslate";
```

Slika 25 *Footer* komponente

```
componentDidMount(){
    var contentwidth = window.innerWidth;
    if ((contentwidth) < 750) {
        SlideUpDown( classnames: 'footer-title');
    } else {
        var elems = document.querySelectorAll( selectors: ".footer-title");
        [].forEach.call(elems,  argArray: function(elemt) {
            let el = elemt.nextElementSibling;
            el.style = "display: block";
        });
    }
}
```

Slika 26 Funkcija *componentDidMount*

```
<div className="footer-contant">
    <div className="footer-logo">
        <LogoImage logo={this.props.logoName} />
    </div>
```

Slika 27 *LogoImage* komponenta

5.5 Početna stranica

Početna stranica sadrži određene komponente (Slika 28) koje se nalaze na svakoj pojedinoj stranici unutar *React* aplikacije. *Helmet* komponenta (Slika 29) se koristi za pisanje meta elemenata *web* stranica. *Slider* komponenta se koristi za ispisivanje klizača (engl. *slidera*) na glavnoj stranici kao što je prikazano na slici 30. Slika 31 sadrži kod koji ispisuje tri kartice vidljive na slici 59.

```
import React, { Component } from 'react';
import {Helmet} from 'react-helmet'
import '../../common/index.scss';
import Slider from 'react-slick';
// Import custom components
import SpecialProducts from "./special-products"
import {getHighestDiscount} from '../../../services'
import {connect} from "react-redux";
import withTranslate from "react-redux-multilingual/lib/withTranslate";
```

Slika 28 Učitane komponente glavne stranice

```
<Helmet>
    <title>Eshop | Electronic Store</title>
</Helmet>
```

Slika 29 *Helmet* komponenta

```
<Slider className="slide-1 home-slider">
    <div>
        <div className="home home-1">
            <div className="container">
                <div className="row">
                    <div className="col">
                        <div className="slider-contain">
                            <div>
                                <h4>{translate('save')} {discount}% {translate('on')}</h4>
                                <h1 className="text-white-50">{translate('graphic_cards')}</h1>
                                <a href={`${process.env.PUBLIC_URL}/shop`} className="btn btn-outline
                                btn-classic text-white">{translate('shop_now')}</a></div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
    <div>
        <div className="home home-2">
            <div className="container">
                <div className="row">
                    <div className="col">
                        <div className="slider-contain">
                            <div>
                                <h4 className="text-white">{translate('save_up_to')}
                                {laptopDiscount}% {translate('on')}</h4>
                                <h1>{translate('laptops')}</h1>
                                <a href={`${process.env.PUBLIC_URL}/shop`} className="btn btn-outline
                                btn-classic text-white">{translate('shop_now')}</a></div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</Slider>
```

Slika 30 *Slider* komponenta

```
<div className="col-md-4">
    <a href={`${process.env.PUBLIC_URL}/shop`}>
        <div className="collection-banner">
            <div className="img-part">
                <img src={`${process.env.PUBLIC_URL}/assets/images/electronics/gpu.jpg`}
                     className="img-fluid blur-up lazyload bg-img" alt="" />
            </div>
            <div className="contain-banner banner-3">
                <div>
                    <h4 className="text-white">{discount}% {translate('off')}</h4>
                    <h2 className="text-white">{translate('graphic_cards')}</h2>
                </div></div></div></a>
</div>
<div className="col-md-4">
    <a href={`${process.env.PUBLIC_URL}/shop`}>
        <div className="collection-banner">
            <div className="img-part">
                <img src={`${process.env.PUBLIC_URL}/assets/images/electronics/asus.jpg`}
                     className="img-fluid blur-up lazyload bg-img" alt="" />
            </div>
            <div className="contain-banner banner-3">
                <div>
                    <h4 className="text-white">{laptopDiscount}% {translate('off')}</h4>
                    <h2>{translate('laptops')}</h2>
                </div></div></div></a>
</div>
<div className="col-md-4">
    <a href={`${process.env.PUBLIC_URL}/shop`}>
        <div className="collection-banner">
            <div className="img-part">
                <img src={`${process.env.PUBLIC_URL}/assets/images/electronics/pc.jpg`}
                     className="img-fluid blur-up lazyload bg-img" alt="" />
            </div>
            <div className="contain-banner banner-3">
                <div>
                    <h4 className="text-white">{pcDiscount}% {translate('off')}</h4>
                    <h2 className="text-white">{translate('gaming_pcs')}</h2>
                </div></div></div></a>
</div>
```

Slika 31 Kartice glavne stranice

SpecialProducts komponenta sadrži kod kolekcija koje se vide na slici 58. Koristi se *Tabs, Tab, TabList i TabPanel* komponentama (Slika 32) za stvaranje zasebnih kartica koje se izmjenjuju ovisno o odabiru liste na stranici (*new, featured, special*). Također se koristi komponenta *ProductItem* koja ispisuje pojedine proizvode unutar lista.

```
<Tabs className="theme-tab">
    <TabList  className="tabs tab-title">
        <Tab>{translate('new_arrival')}</Tab>
        <Tab>{translate('featured')}</Tab>
        <Tab>{translate('special')}</Tab>
    </TabList>

    <TabPanel>
        <div className="no-slider row">
            { newProducts.map((product, index ) =>
                <ProductItem product={product} symbol={symbol}
                             onAddToCompareClicked={() => addToCompare(product)}
                             onAddToWishlistClicked={() => addToWishlist(product)}
                             onAddToCartClicked={() => addToCart(product, 1)} key={index} /> )
            }
        </div>
    </TabPanel>
    <TabPanel>
        <div className="no-slider row">
            { featuredProducts.map((product, index ) =>
                <ProductItem product={product} symbol={symbol}
                             onAddToCompareClicked={() => addToCompare(product)}
                             onAddToWishlistClicked={() => addToWishlist(product)}
                             onAddToCartClicked={() => addToCart(product, 1)} key={index} /> )
            }
        </div>
    </TabPanel>
    <TabPanel>
        <div className=" no-slider row">
            { bestSeller.map((product, index ) =>
                <ProductItem product={product} symbol={symbol}
                             onAddToCompareClicked={() => addToCompare(product)}
                             onAddToWishlistClicked={() => addToWishlist(product)}
                             onAddToCartClicked={() => addToCart(product, 1)} key={index} /> )
            }
        </div>
    </TabPanel>
</Tabs>
```

Slika 32 *SpecialProducts* sadržaj

Redux funkcija *connect* se koristi za spremanje varijabli iz *state* objekta unutar *props* objekta za korištenje unutar *web* aplikacije koristeći *mapStateToProps* funkciju (Slika 33). Razlog za korištenje ove funkcije je da bi mogli koristiti podatke spremljene unutar *redux* spremnika unutar komponente.

```
const mapStateToProps = (state) => ({
    discount: getHighestDiscount(state.data.products, category: 'graphic-card'),
    laptopDiscount: getHighestDiscount(state.data.products, category: 'laptop'),
    pcDiscount: getHighestDiscount(state.data.products, category: 'computer')
})

export default connect(mapStateToProps, null) (withTranslate(Main));
```

Slika 33 *Redux connect* funkcija

5.6 Trgovina

Stranica Trgovina (slike 60, 61 i 62) sadrži sljedeće komponente: *NewProduct* (Slika 35) ispisuje listu novih proizvoda u obliku *Slider* komponente, *Filter* (Slika 35) ispisuje filtere pomoću kojih se mogu filtrirati proizvodi, *FilterBar* (Slika 36) sadrži filtere za filtriranje izgleda liste proizvoda, *ProductListing* (Slika 36) ispisuje listu svi proizvoda ovisno o filterima a *StickyBox* (Slika 35) sadrži komponente *Filter* i *NewProduct* te se ponaša kao *sticky* element tj., prati stranicu kako se sadržaj pomiče dole. Funkcija *LayoutViewClicked* se koristi za izmjenu izgleda liste proizvoda (2., 3., 4. ili 5. kolumni), a *openFilter* se koristi za otvaranje *filter* elementa na mobilnom verziji stranice te se ove funkcije mogu vidjeti na slici 34.

```
state = {
    layoutColumns:3
}

LayoutViewClicked(colums) {
    this.setState( state: {
        layoutColumns:colums
    })
}

openFilter = () => {
    document.querySelector( selectors: ".collection-filter").style = "left: -15px";
}
```

Slika 34 Funkcije Trgovina stranice

```
<Breadcrumb title={translate('shop')}/>

<section className="section-b-space">
    <div className="collection-wrapper">
        <div className="container">
            <div className="row">
                <div className="col-sm-3 collection-filter">

                    <StickyBox offsetTop={112} offsetBottom={20}>
                        <div>
                            <Filter/>
                            <NewProduct/>
                        </div>
                    </StickyBox>
                    {/*side-bar banner end here*/}
                </div>
```

Slika 35 Sadržaj Trgovina stranice

```
<div className="product-top-filter">
    <div className="container-fluid p-0">
        <div className="row">
            <div className="col-xl-12">
                <div className="filter-main-btn">
                    <span onClick={this.openFilter}
                        className="filter-btn btn btn-theme"><i
                        className="fa fa-filter"
                        aria-hidden="true"></i> Filter</span>
                </div>
            </div>
        </div>
        <div className="row">
            <div className="col-12">
                <FilterBar onLayoutViewClicked={(colmuns) => this.LayoutViewClicked(colmuns)}/>
            </div>
        </div>
    </div>
</div>

{/*Products Listing Component*/}
<ProductListing colSize={this.state.layoutColumns}/>
```

Slika 36 Ostali sadržaj Trgovina stranice

5.7 O Nama

Stranica O Nama (slike 63 i 64) sadrži *Slider* komponentu koji ispisuje članove timova (Slika 38) te uzima *Team4* varijablu koja vraća postavke za kontrolu klizača. Također sadrži sekciju *O Nama* i *Servis* koji ispisuju određene podatke o trgovini vidljive na slikama 37 i 39.

```
<Breadcrumb title={translate('about_us')}/>
{/*about section*/}
<section className="about-page  section-b-space">
    <div className="container">
        <div className="row">
            <div className="col-lg-12">
                <div className="banner-section">
                    <img src={`${process.env.PUBLIC_URL}/assets/images/electronics/shop.jpg`} className="img-fluid" alt=""/>
                </div>
            </div>
            <div className="col-sm-12">
                <h4>Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium
                    doloremque laudantium</h4>
                <p>Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque
                    laudantium,</p>
                <p>On the other hand, we denounce with righteous indignation and dislike men who are so
                    beguiled and demoralized by the charms of pleasure of the moment, so blinded by
                    desire, that they cannot foresee the pain and trouble that are bound to ensue; and
                    equal blame belongs to those who fail in their duty through weakness of will, which
                    is the same as saying through shrinking from toil and pain. These cases are
                    perfectly simple and easy to distinguish. In a free hour, when our power of choice
                    is untrammelled and when nothing prevents our being able to do what we like best,
                    every pleasure is to be welcomed and every pain avoided. But in certain
                    circumstances and owing to the claims of duty or the obligations of business it will
                    frequently occur that pleasures have to be repudiated and annoyances accepted. The
                    wise man therefore always holds in these matters to this principle of selection: he
                    rejects pleasures to secure other greater pleasures, or else he endures pains to
                    avoid worse pains.</p>
            </div>
        </div>
    </div>
</section>
```

Slika 37 O Nama sekcija

```
{/*Team Section*/}
<section id="team" className="team section-b-space">
    <div className="container">
        <div className="row">
            <div className="col-sm-12">
                <h2>{translate('our_team')}</h2>
                <Slider {...Team4} className="team-4">
                    <div>
                        <img src={`${process.env.PUBLIC_URL}/assets/images/team/1.jpg`} className="img-fluid" alt=""/>
                        <h4>John Doe</h4>
                        <h6>CEO & Founder At Eshop</h6>
                    </div>
                    <div>
                        <img src={`${process.env.PUBLIC_URL}/assets/images/team/2.jpg`} className="img-fluid" alt=""/>
                        <h4>John Doe</h4>
                        <h6>CTO At Eshop</h6>
                    </div>
                    <div>
                        <img src={`${process.env.PUBLIC_URL}/assets/images/team/3.jpg`} className="img-fluid" alt=""/>
                        <h4>John Doe</h4>
                        <h6>COO At Eshop</h6>
                    </div>
                    <div>
                        <img src={`${process.env.PUBLIC_URL}/assets/images/team/4.jpg`} className="img-fluid" alt=""/>
                        <h4>John Doe</h4>
                        <h6>Senior Developer At Eshop</h6>
                    </div>
                    <div>
                        <img src={`${process.env.PUBLIC_URL}/assets/images/team/5.jpg`} className="img-fluid" alt=""/>
                        <h4>John Doe</h4>
                        <h6>Salesman AT Eshop</h6>
                    </div>
                </Slider>
            </div>
        </div>
    </div>
</section>
```

Slika 38 Tim sekcija

```
{/*service layout*/}
<div className="container about-cls section-b-space">
    <section className="service border-section small-section ">
        <div className="row">
            <div className="col-md-4 service-block">
                <div className="media">
                    <div dangerouslySetInnerHTML={{ __html: svgFreeShipping }} />
                    <div className="media-body">
                        <h4>free shipping</h4>
                        <p>free shipping world wide</p>
                    </div>
                </div>
            </div>
            <div className="col-md-4 service-block">
                <div className="media">
                    <div dangerouslySetInnerHTML={{ __html: svgservice }} />
                    <div className="media-body">
                        <h4>24 X 7 service</h4>
                        <p>online service for new customer</p>
                    </div>
                </div>
            </div>
            <div className="col-md-4 service-block">
                <div className="media">
                    <div dangerouslySetInnerHTML={{ __html: svgoffer }} />
                    <div className="media-body">
                        <h4>discount offer</h4>
                        <p>new online special discount offer</p>
                    </div>
                </div>
            </div>
        </div>
    </section>
</div>
{/*service layout end*/}
```

Slika 39 Servis sekcija

5.8 Kontakt

Kontakt stranica (Slika 65) sadrži mapu na kojoj se vidi lokacija trgovine koja se ispisuje pomoću *iframe* elementa (Slika 41), podatci o trgovini (Slika 41), kontakt forma (Slika 42) za slanje mailova i funkcije za kontakt formu. *HandleSubmit* funkcija šalje mail pomoću *axios* paketa,

resetForm briše sve podatke iz kontakt forme, dok ostale funkcije *Change* spremaju sadržaj kontakt forme u *state* objekt koje *axios* koristi pri slanju mailova (Slika 40).

```
handleSubmit(event) {
    event.preventDefault();
    axios({
        method: "POST",
        url:"http://localhost:3002/send",
        data:  this.state
    }).then((response : AxiosResponse<any> )=>{
        if (response.data.status === 'success'){
            alert("Message Sent.");
            this.resetForm()
        }else if(response.data.status === 'fail'){
            alert("Message failed to send.")
        }
    })
}

resetForm(){
    this.setState( state: {first_name: '', last_name: '', phone: '', email: '', message: ''})
}

onFirstNameChange(event){
    this.setState( state: {first_name: event.target.value})
}

onLastNameChange(event){
    this.setState( state: {last_name: event.target.value})
}

onPhoneChange(event){
    this.setState( state: {phone: event.target.value})
}

onEmailChange(event){
    this.setState( state: {email: event.target.value})
}

onMessageChange(event){
    this.setState( state: {message: event.target.value})
}
```

Slika 40 Funkcije Kontakt stranice

```
<div className="col-lg-7 map">
    <iframe
        src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d2934.482975560137!2d18.089455816251714!3d42.6511957916844
        allowFullScreen></iframe>
</div>
<div className="col-lg-5">
    <div className="contact-right">
        <ul>
            <li>
                <div className="contact-icon">
                    <img src={`${process.env.PUBLIC_URL}/assets/images/icon/phone.png`} alt="Generic placeholder image" />
                        <h6>{translate('contact_us')}</h6>
                </div>
                <div className="media-body">
                    <p>+385 986 - 754 - 3210</p>
                    <p>+385 986 - 754 - 3211</p>
                </div>
            </li>
            <li>
                <div className="contact-icon">
                    <i className="fa fa-map-marker" aria-hidden="true"></i>
                    <h6>{translate('address')}</h6>
                </div>
                <div className="media-body">
                    <p>Vukovarska 13</p>
                    <p>Dubrovnik 20000</p>
                </div>
            </li>
            <li>
                <div className="contact-icon">
                    <img src={`${process.env.PUBLIC_URL}/assets/images/icon/email.png`} alt="Generic placeholder image" />
                        <h6>{translate('email')}</h6>
                </div>
                <div className="media-body">
                    <p>suport@eshop.com</p>
                    <p>info@eshop.com</p>
                </div>
            </li>
            <li>
```

Slika 41 Mapa i podaci trgovine

```
<form className="theme-form" onSubmit={this.handleSubmit.bind(this)}>
    <div className="form-row">
        <div className="col-md-6">
            <label htmlFor="name">{translate('first_name')}</label>
            <input type="text" className="form-control" id="first_name"
                   placeholder={translate('first_name')} required="" value={this.state.first_name} onChange={this.onFirstNameChange.bind(this)}/>
        </div>
        <div className="col-md-6">
            <label htmlFor="email">{translate('last_name')}</label>
            <input type="text" className="form-control" id="last_name"
                   placeholder={translate('last_name')} required="" value={this.state.last_name} onChange={this.onLastNameChange.bind(this)}/>
        </div>
        <div className="col-md-6">
            <label htmlFor="review">{translate('phone_number')}</label>
            <input type="text" className="form-control" id="phone"
                   placeholder={translate('phone_number')} required="" value={this.state.phone} onChange={this.onPhoneChange.bind(this)}/>
        </div>
        <div className="col-md-6">
            <label htmlFor="email">{translate('email')}</label>
            <input type="text" className="form-control" id="email" placeholder={translate('email')}
                   required="" value={this.state.email} onChange={this.onEmailChange.bind(this)}/>
        </div>
        <div className="col-md-12">
            <label htmlFor="review">{translate('write_your_message')}</label>
            <textarea className="form-control" placeholder={translate('write_your_message')}
                      id="message" rows="6" value={this.state.message} onChange={this.onMessageChange.bind(this)}></textarea>
        </div>
        <div className="col-md-12">
            <button className="btn btn-solid" type="submit">{translate('send_your_message')}</button>
        </div>
    </div>
</form>
```

Slika 42 Kontakt forma

5.9 Košarica

Stranica Košarica (Slika 66) sadrži tablicu koja ispisuje detalje odabranih proizvoda i omogućuje izmjenu količine odabranih proizvoda te uklanjanje određenih proizvoda iz košarice. Na slici 43 je prikazan kod zaglavlja tablice i ispisivanja slike i pojedinačne cijene proizvoda. Slika 44 prikazuje opciju uklanjanja proizvoda iz košarice, opciju povećanja ili smanjenja količine proizvoda i totalne cijene proizvoda, a na slici 45 prikazan je kod za ispis ukupne cijene košarice, gumbi za kupnju ili vraćanje u trgovinu i kod za prikazivanje prazne košarice.

```
<table className="table cart-table table-responsive-xs">
    <thead>
    <tr className="table-head">
        <th scope="col">{translate('image')}</th>
        <th scope="col">{translate('product_name')}</th>
        <th scope="col">{translate('price')}</th>
        <th scope="col">{translate('quantity')}</th>
        <th scope="col">{translate('action')}</th>
        <th scope="col">{translate('total')}</th>
    </tr>
    </thead>
    {cartItems.map((item, index) => {
        return (
        <tbody key={index}>
            <tr>
                <td>
                    <Link to={`${process.env.PUBLIC_URL}/product/${item._id}`}>
                        <img src={item.variants?
                                  item.variants[0].images
                                  :item.pictures[0]} alt="" />
                    </Link>
                </td>
                <td><Link to={`${process.env.PUBLIC_URL}/product/${item._id}`}>{item.name}</Link>
                    <div className="mobile-cart-content row">
                        <div className="col-xs-3">
                            <div className="qty-box">
                                <div className="input-group">
                                    <input type="text" name="quantity"
                                           className="form-control input-number" defaultValue={item.qty} />
                                </div>
                            </div>
                        </div>
                        <div className="col-xs-3">
                            <h2 className="td-color">{symbol}{item.price-(item.price*item.discount/100)}</h2>
                        </div>
```

Slika 43 Zaglavlje tablice i dio sadržaja Košarica stranice

```
                <div className="col-xs-3">
                    <h2 className="td-color">
                        <a href="#" className="icon" onClick={() => this.props.removeFromCart(item)}>
                            <i className="icon-close"></i>
                        </a>
                    </h2>
                </div>
            </div>
        </td>
        <td><h2>{symbol}{item.price-(item.price*item.discount/100)}</h2></td>
        <td>
            <div className="qty-box">
                <div className="input-group">
                    <span className="input-group-prepend">
                        <button type="button" className="btn quantity-left-minus" onClick={() => this.props.decrementQty(item._id)}
                                data-type="minus" data-field="">
                         <i className="fa fa-angle-left"></i>
                        </button>
                    </span>
                    <input type="text" name="quantity" value={item.qty} readOnly={true} className="form-control input-number" />

                    <span className="input-group-prepend">
                    <button className="btn quantity-right-plus" onClick={() => this.props.incrementQty(item, 1)}  data-type="plus"
                            disabled={(item.qty >= item.stock)? true : false}>
                    <i className="fa fa-angle-right"></i>
                    </button>
                   </span>
                </div>
            </div>{(item.qty >= item.stock)? translate('out_of_stock') : ''}
        </td>
        <td>
            <a href="#" className="icon" onClick={() => this.props.removeFromCart(item)}>
                <i className="fa fa-times"></i>
            </a>
        </td>
        <td><h2 className="td-color">{symbol}{item.qty*(item.price-(item.price*item.discount/100))}</h2></td>
    </tr>
</tbody> )
```

Slika 44 Sadržaj Košarica stranice

```
                <table className="table cart-table table-responsive-md">
                    <tfoot>
                    <tr>
                        <td>{translate('total_price')} :</td>
                        <td><h2>{symbol} {total} </h2></td>
                    </tr>
                    </tfoot>
                </table>
            </div>
        </div>
        <div className="row cart-buttons">
            <div className="col-6">
                <Link to={`${process.env.PUBLIC_URL}/shop`} className="btn btn-solid">{translate('continue_shopping')}</Link>
            </div>
            <div className="col-6">
                <Link to={`${process.env.PUBLIC_URL}/checkout`} className="btn btn-solid">{translate('check_out')}</Link>
            </div>
        </div>
    </div>
</section>
:
<section className="cart-section section-b-space">
    <div className="container">
        <div className="row">
            <div className="col-sm-12">
                <div >
                    <div className="col-sm-12 empty-cart-cls text-center">
                        <img src={`${process.env.PUBLIC_URL}/assets/images/icon-empty-cart.png`} className="img-fluid mb-4" alt="" />
                        <h3>
                            <strong>{translate('cart_empty_text')}</strong>
                        </h3>
                    </div>
                </div>
            </div>
        </div>
    </div>
</section>
```

Slika 45 Sadržaj tablice, gumbi i ispis prazne košarice stranice Košarica

5.10 Stranica plaćanja

Na stranici Plaćanje (Slika 67) se koriste *Stripe* [50] i *Paypal* [51] paketi kao oblik online plaćanja na trgovini. Također se na stranici nalazi forma za unos podataka korisnika i detalji proizvoda za plaćanje. Na slici 46 se nalaze komponente, *constructor* funkcija u kojoj spremamo početne varijable na stranici i *setStateFromInput* funkcija koja sprema podatke iz određenih *input* elemenata kako se mijenjaju. Slika 47 prikazuje funkcije: *setStateFromCheckbox* koja sprema podatke iz *checkbox* elemenata na izmjeni elementa, *checkhandle* koja sprema vrijednost plaćanja unutar *state* objekta i *stripeClick* koji otvara prozor za plaćanje preko *Stripe* API-ja. *PayPal* funkcije se nalaze na slici 48 koje se izvršavaju ovisno o statusu naplate, a slike 49 i 50 prikazuju kod forme za unos podataka. Unutar forme se zovu funkcije za izmjenu podataka *setStateFromInput/Checkbox* koje spremaju promjenu vrijednosti unutar *state* varijabli.

```
import React, {Component} from 'react';
import {Helmet} from 'react-helmet'
import { connect } from 'react-redux'
import {Link, Redirect } from 'react-router-dom'
import PaypalExpressBtn from 'react-paypal-express-checkout';
import SimpleReactValidator from 'simple-react-validator';

import Breadcrumb from "../common/breadcrumb";
import {removeFromWishlist} from '../../actions'
import {getCartTotal} from "../../services";
import withTranslate from "react-redux-multilingual/lib/withTranslate";

class checkOut extends Component {

    constructor (props) {
        super (props)

        this.state = {
            payment:'stripe',
            first_name:'',
            last_name:'',
            phone:'',
            email:'',
            country:'',
            address:'',
            city:'',
            state:'',
            postalcode:'',
            create_account: ''
        }
        this.validator = new SimpleReactValidator();
    }

    setStateFromInput = (event) => {
        var obj = {};
        obj[event.target.name] = event.target.value;
        this.setState(obj);
```

Slika 46 Komponente i funkcije stranice Plaćanja

```
setStateFromCheckbox = (event) => {
  var obj = {};
  obj[event.target.name] = event.target.checked;
  this.setState(obj);
  if(!this.validator.fieldValid(event.target.name) && event.target.name != 'create_account')
  {this.validator.showMessages();}
}
checkhandle(value) {
    this.setState( state: {payment: value})
}
StripeClick = () => {

    if (this.validator.allValid()) {
        alert('You submitted the form and stuff!');

        var handler = (window).StripeCheckout.configure({
            key: 'pk_test_glxk17KhP7poKIawsaSgKtsL',
            locale: 'auto',
            token: (token: any) => {
                console.log(token)
                  this.props.history.push({
                      pathname: '/order-success',
                          state: { payment: token, items: this.props.cartItems, orderTotal: this.props.total, symbol: this.props.symbol }
                  })
            }
          });
          handler.open({
            name: 'Eshop',
            description: 'Online Computer Store',
            amount: this.amount * 100
          })
    } else {
      this.validator.showMessages();
      // rerender to show messages for the first time
      this.forceUpdate();
    }
}
```

Slika 47 Funkcije stranice Plaćanja

```
render (){
    const {cartItems, symbol, total, translate} = this.props;

    // Paypal Integration
    const onSuccess = (payment) => {
        console.log("The payment was succeeded!", payment);
        this.props.history.push({
            pathname: '/order-success',
                state: { payment: payment, items: cartItems, orderTotal: total, symbol: symbol }
        })

    }

    const onCancel = (data) => {
        console.log('The payment was cancelled!', data);
    }

    const onError = (err) => {
        console.log("Error!", err);
    }

    const client = {
        sandbox:    'AZ4S98zFa01vym7NVeo_qthZyOnBhtNvQDsjhaZSMH-2_Y9IAJFbSD3HPueErYqN8Sa8WYRbjP7wWtd_',
        production: 'AZ4S98zFa01vym7NVeo_qthZyOnBhtNvQDsjhaZSMH-2_Y9IAJFbSD3HPueErYqN8Sa8WYRbjP7wWtd_',
    }
```

Slika 48 Funkcije za PayPal

```
<form>
    <div className="checkout row">
        <div className="col-lg-6 col-sm-12 col-xs-12">
            <div className="checkout-title">
                <h3>{translate('billing_details')}</h3>
            </div>
            <div className="row check-out">
                <div className="form-group col-md-6 col-sm-6 col-xs-12">
                    <div className="field-label">{translate('first_name')}</div>
                    <input type="text" name="first_name" value={this.state.first_name} onChange={this.setStateFromInput} />
                    {this.validator.message( field: 'first_name', this.state.first_name,  validations: 'required|alpha')}
                </div>
                <div className="form-group col-md-6 col-sm-6 col-xs-12">
                    <div className="field-label">{translate('last_name')}</div>
                    <input type="text" name="last_name" value={this.state.last_name} onChange={this.setStateFromInput} />
                    {this.validator.message( field: 'last_name', this.state.last_name,  validations: 'required|alpha')}
                </div>
                <div className="form-group col-md-6 col-sm-6 col-xs-12">
                    <div className="field-label">{translate('phone_number')}</div>
                    <input type="text" name="phone"  value={this.state.phone} onChange={this.setStateFromInput} />
                    {this.validator.message( field: 'phone', this.state.phone,  validations: 'required|phone')}
                </div>
                <div className="form-group col-md-6 col-sm-6 col-xs-12">
                    <div className="field-label">{translate('email')} {translate('address')}</div>
                    <input type="text" name="email" value={this.state.email} onChange={this.setStateFromInput} />
                    {this.validator.message( field: 'email', this.state.email,  validations: 'required|email')}
                </div>
                <div className="form-group col-md-12 col-sm-12 col-xs-12">
                    <div className="field-label">{translate('country')}</div>
                    <select name="country" value={this.state.country} onChange={this.setStateFromInput}...>
                    {this.validator.message( field: 'country', this.state.country,  validations: 'required')}
                </div>
```

Slika 49 Forma za unos podataka

```
            <div className="form-group col-md-12 col-sm-12 col-xs-12">
                <div className="field-label">{translate('address')}</div>
                <input type="text" name="address" value={this.state.address} onChange={this.setStateFromInput} placeholder="Street address" />
                {this.validator.message( field: 'address', this.state.address,  validations: 'required|max:120')}
            </div>
            <div className="form-group col-md-12 col-sm-12 col-xs-12">
                <div className="field-label">{translate('town_city')}</div>
                <input type="text" name="city" value={this.state.city} onChange={this.setStateFromInput} />
                {this.validator.message( field: 'city', this.state.city,  validations: 'required|alpha')}
            </div>
            <div className="form-group col-md-12 col-sm-6 col-xs-12">
                <div className="field-label">{translate('state_country')}</div>
                <input type="text" name="state" value={this.state.state} onChange={this.setStateFromInput} />
                {this.validator.message( field: 'state', this.state.state,  validations: 'required|alpha')}
            </div>
            <div className="form-group col-md-12 col-sm-6 col-xs-12">
                <div className="field-label">{translate('postal_code')}</div>
                <input type="text" name="postalcode" value={this.state.postalcode} onChange={this.setStateFromInput} />
                {this.validator.message( field: 'postal code', this.state.postalcode,  validations: 'required|integer')}
            </div>
            <div className="form-group col-lg-12 col-md-12 col-sm-12 col-xs-12">
                <input type="checkbox" name="create_account" id="account-option"  checked={this.state.create_account} onChange=
                    {this.setStateFromCheckbox}/>
                  <label htmlFor="account-option">{translate('create_account')}</label>
            </div>
        </div>
    </div>
```

Slika 50 Forma za unos podataka

5.11 Stranica usporedbe

Stranica Usporedba (Slika 68) sadrži *Slider* komponentu (Slika 51) unutar koje se ispisuju proizvodi za usporedbu. Prikazuje se slika i svi podaci o proizvodu i gumbi za dodavanje u košaricu ili brisanje iz *slider* elementa.

```
<Slider {...settings} className="slide-4">
    {Items.map((item,index) =>
        <div key={index}>
            <div className="compare-part">
                <button type="button" className="close-btn" onClick={() => removeFromCompare(item)}>
                    <span aria-hidden="true">×</span>
                </button>
                <div className="img-secton">
                    <Link to={`${process.env.PUBLIC_URL}/product/${item._id}`}>
                    <img src={item.variants?
                                item.variants[0].images
                                :item.pictures[0]} className="img-fluid" alt="" />
                    <h5>{item.name}</h5></Link>
                    <h5>{symbol}{(item.price*item.discount/100)}
                        <del><span className="money">{symbol}{item.price}</span></del></h5>
                </div>
                <div className="detail-part"><div className="title-detail"><h5>discription</h5></div>
                    <div className="inner-detail"><p>{item.shortDetails}</p></div>
                </div>
                <div className="detail-part"><div className="title-detail"><h5>Brand Name</h5></div>
                    <div className="inner-detail"><p>{item.tags}</p></div>
                </div>
                <div className="detail-part"><div className="title-detail"><h5>size</h5></div>
                    <div className="inner-detail"><p>{item.size}</p></div>
                </div>
                <div className="detail-part"><div className="title-detail"><h5>color</h5></div>
                    <div className="inner-detail"><p>{item.colors}</p></div>
                </div>
                <div className="detail-part"><div className="title-detail"><h5>availability</h5></div>
                    <div className="inner-detail"><p>In stock</p></div>
                </div>
                <div className="btn-part">
                    <a href="javascript:void(0)" className="btn btn-solid" onClick={() => addToCart(item, 1)}>add to cart</a>
                </div>
            </div>
        </div>
    )}
</Slider>
```

Slika 51 *Slider* komponenta stranice Usporedba

5.12 Stranica željenih proizvoda

Stranica Željeni Proizvodi (Slika 69) se sastoji od tablice u kojoj su zapisani podaci odabranih proizvoda te ih je moguće izbrisati iz liste proizvode ili dodati u košaricu. Također postoje gumbi za odlazak na stranicu Trgovina ili na stranicu Plaćanje. Na slici 52 je prikazan kod zaglavlja tablice, ispisa slike proizvoda, cijene proizvoda i gumbova za brisanje iz liste ili dodavanje u košaricu za mobilnu verziju stranice. Slika 53 prikazuje programski kod ispisa cijene i gumba za *desktop* verziju aplikacije i dva gumba za nastavak u trgovinu ili na plaćanje.

```
<table className="table cart-table table-responsive-xs">
    <thead>
    <tr className="table-head">
        <th scope="col">image</th>
        <th scope="col">product name</th>
        <th scope="col">price</th>
        <th scope="col">availability</th>
        <th scope="col">action</th>
    </tr>
    </thead>
    {Items.map((item, index) => {
        return (
            <tbody key={index}>
            <tr>
                <td>
                    <Link to={`${process.env.PUBLIC_URL}/product/${item._id}`}>
                        <img src={item.variants?
                                    item.variants[0].images
                                    :item.pictures[0]} alt="" />
                    </Link>
                </td>
                <td><Link to={`${process.env.PUBLIC_URL}/product/${item._id}`}>{item.name}</Link>
                    <div className="mobile-cart-content row">
                        <div className="col-xs-3">
                            <p>in stock</p>
                        </div>
                        <div className="col-xs-3">
                            <h2 className="td-color">{symbol}{item.price-(item.price*item.discount/100)}
                            <del><span className="money">{symbol}{item.price}</span></del></h2>
                        </div>
                        <div className="col-xs-3">
                            <h2 className="td-color">
                                <a href="javascript:void(0)" className="icon" onClick={() => this.props.removeFromWishlist(item)}>
                                    <i className="fa fa-times"></i>
                                </a>
                                <a href="javascript:void(0)" className="cart" onClick={() => this.props.addToCartAndRemoveWishlist(item, 1)}>
                                    <i className="fa fa-shopping-cart"></i>
                                </a>
                            </h2>
```

Slika 52 Sadržaj stranice Željenih Proizvoda

```
                            <td><h2>{symbol}{item.price-(item.price*item.discount/100)}
                                <del><span className="money">{symbol}{item.price}</span></del></h2></td>
                            <td >
                                <p>in stock</p>
                            </td>
                            <td>
                                <a href="javascript:void(0)" className="icon" onClick={() => this.props.removeFromWishlist(item)}>
                                    <i className="fa fa-times"></i>
                                </a>
                                <a href="javascript:void(0)" className="cart" onClick={() => this.props.addToCartAndRemoveWishlist(item, 1)}>
                                    <i className="fa fa-shopping-cart"></i>
                                </a>
                            </td>
                        </tr>
                        </tbody> )
                })}
            </table>
        </div>
    </div>
    <div className="row wishlist-buttons">
        <div className="col-12">
            <Link to={`${process.env.PUBLIC_URL}/shop`} className="btn btn-solid">continue shopping</Link>
            <Link to={`${process.env.PUBLIC_URL}/checkout`} className="btn btn-solid">check out</Link>
        </div>
    </div>
</div>
</section>
```

Slika 53 Sadržaj stranice Željenih Proizvoda

5.13 Stranica 404

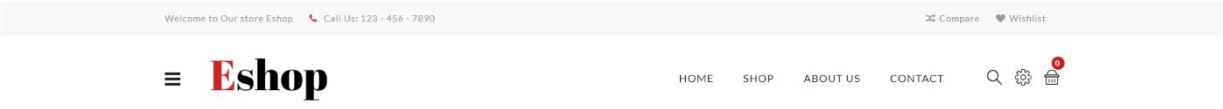
Stranica 404 se pojavi kada se pokuša pristupiti stranici koja ne postoji na *web* aplikaciji. Sadrži jednostavan kod (Slika 54) koji ispisuje grešku 404 koja govori korisniku da stranica ne postoji i opciju da se vrati na početnu stranicu.

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

import Breadcrumb from "../common/breadcrumb";

class PageNotFound extends Component {

    render (){

        return (
            <div>
                <Breadcrumb title={'404 Page'}/>
                <section className="p-0">
                    <div className="container">
                        <div className="row">
                            <div className="col-sm-12">
                                <div className="error-section">
                                    <h1>404</h1>
                                    <h2>page not found</h2>
                                    <a href="index.html" className="btn btn-solid">back to home</a>
                                </div>
                            </div>
                        </div>
                    </div>
                </section>
            </div>
        )
    }
}

export default PageNotFound
```

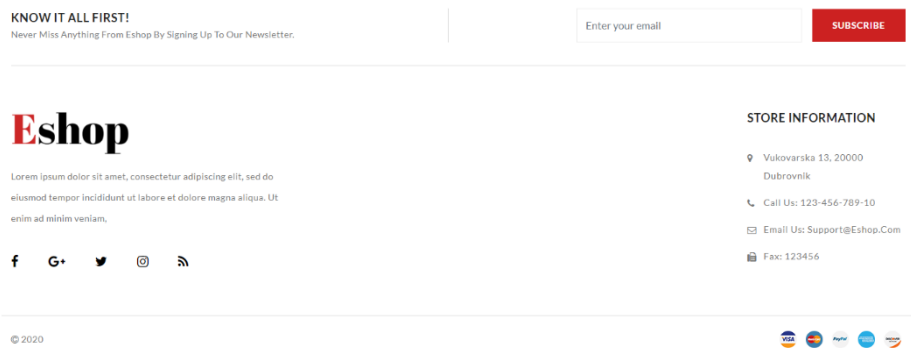
Slika 54 Sadržaj 404 stranice

6 DIJELOVI REAKTIVNOG SUČELJA

U ovom poglavlju će se proći kroz *web* aplikaciju i objasniti pojedine dijelove korisničkog sučelja. Svaka stranica ima 3 glavna dijela: zaglavlje, podnožje i sadržaj. Ulaskom na početnu stranicu se učitavaju sva tri dijela te se zaglavlje i podnožje komponente osim par dijelova ne mijenjaju, što znači da se te komponente ne učitavaju više puta. Na slikama 55 i 56 su prikazani zaglavlje i podnožje izrađene reaktivne aplikacije.



Slika 55 Zaglavlje *web* aplikacije



Slika 56 Podnožje *web* aplikacije

Glavni sadržaj stranice u sebi sadrži *Slider i SpecialProduct* komponente vidljive na slikama 57 i 58. Sadržaj također sadrži tri kartice (Slika 59) čija je namjena istaknuti posebne proizvode ili popuste na proizvode. *Slider* komponenta sadrži tri *Slider* elementa koji se izmjenjuju svakih 5 sekunda ili pritiskom na strelice za naprijed i nazad. *SpecialProduct* sadrži tri različita elementa od kojih svaki prikazuje posebnu skupinu proizvoda: novo, istaknuto i posebno.

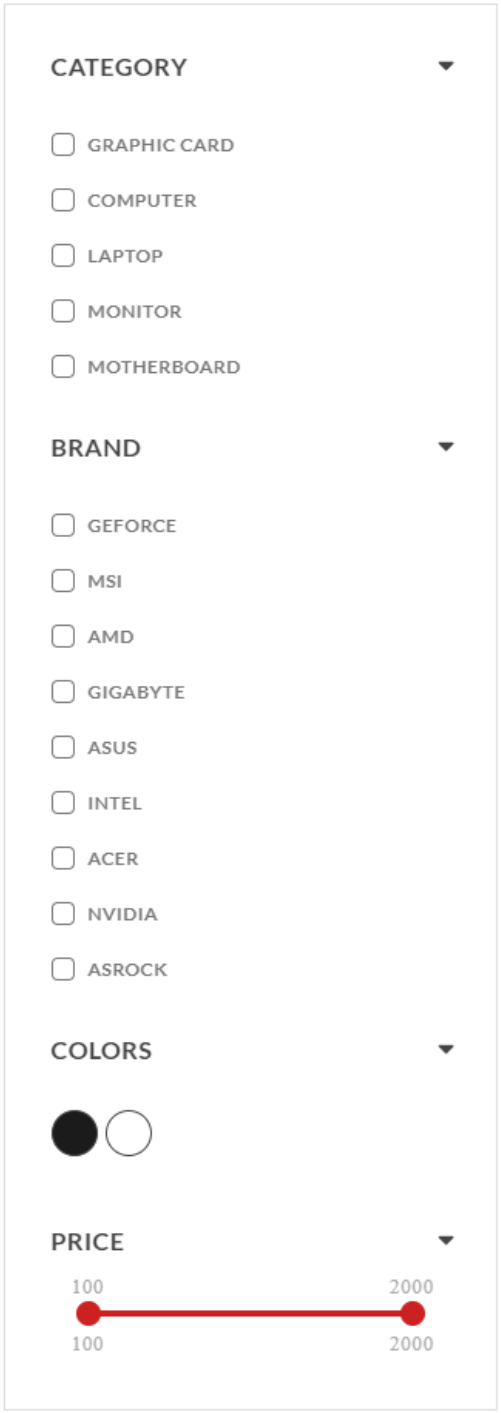
Slika 57 *Slider* komponenta

Slika 58 *SpecialProduct* komponenta

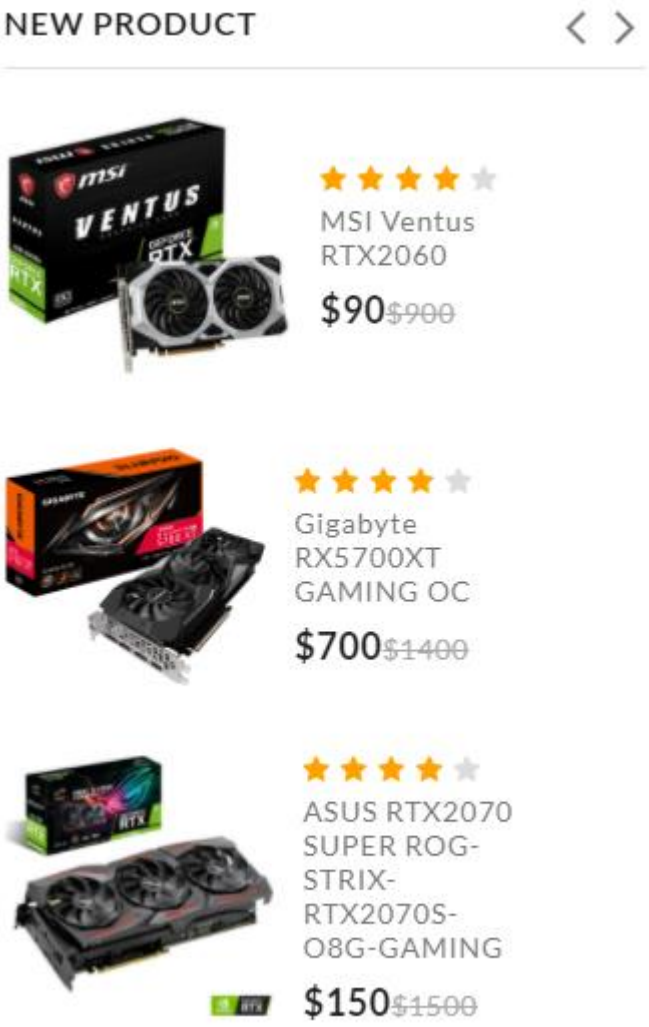
Slika 59 Kartice početne stranice

Druga stranica *web* aplikacije je stranica Trgovina koja u sebi sadrži 5 komponenti: *NewProduct, Filter, FilterBar, ProductListing, StickyBox.* Komponenta *StickyBox* služi kao kontejner za komponente *Filter* i *NewProduct*. Filter komponenta sadrži filtere pomoću kojih

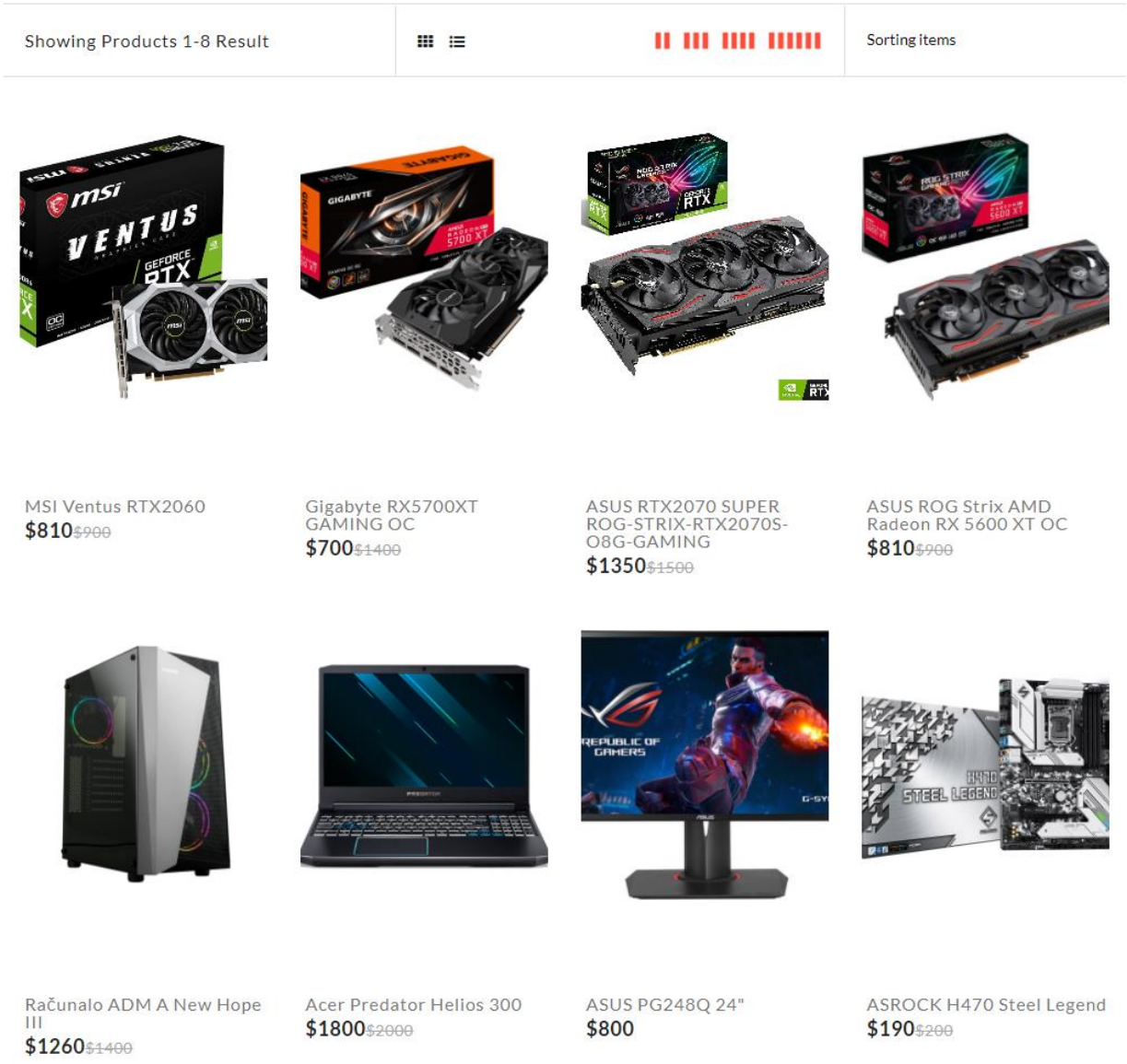
možemo filtrirati sadržaj *ProductListing* komponente. *NewProduct* komponenta sadrži listu svih novih proizvoda unutar *Slider* komponente. *FilterBar* komponenta za razliku od *Filter* komponente sadrži filtere koji mogu mijenjati izgled *ProductListing* komponente. *ProductListing* komponenta sadrži listu svih proizvoda na stranici unutar *InfiniteScroll* komponente koja služi za učitavanje komponenti kako se korisnik pomiče na dno stranice. Sve ove komponente možemo vidjeti na slikama 60, 61, i 62.



Slika 60 Filter komponenta

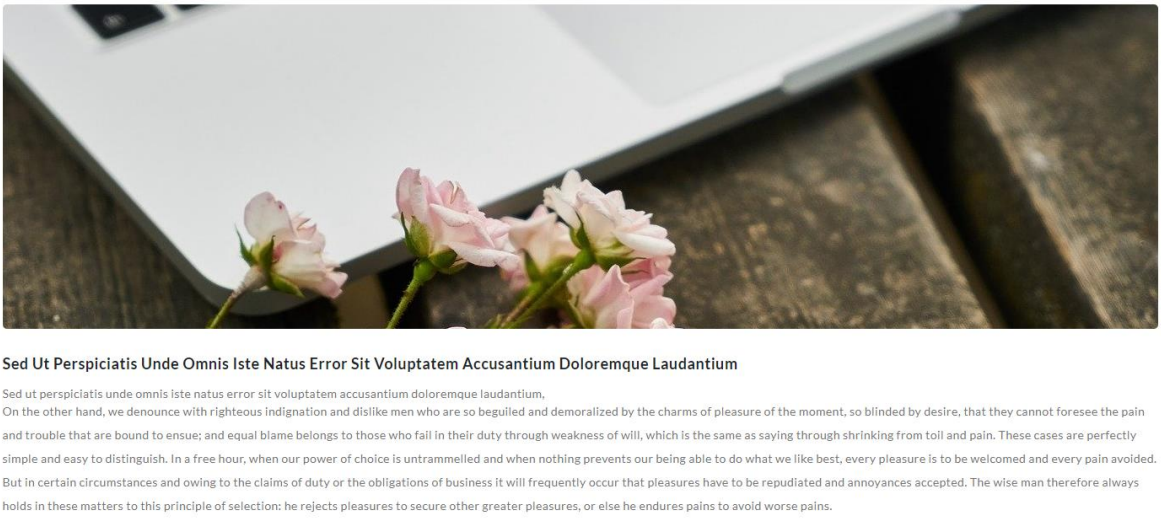


Slika 61 *NewProduct* komponenta



Slika 62 *FilterBar* i *ProductListing* komponente

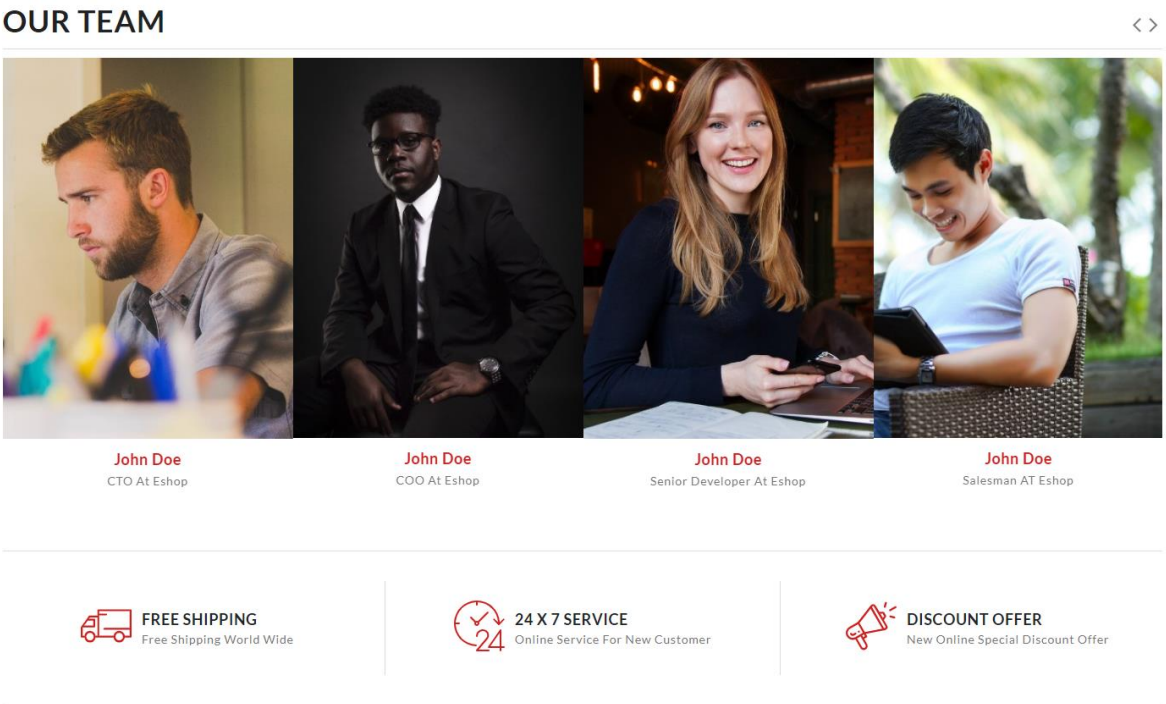
Sljedeća stranica je O Nama. Na toj stranici se nalazi *o nama* sekcija u kojoj se opisuje trgovina, *tim* sekcija koja koristi *Slider* komponentu za prikazivanje osoba unutar tima i *servis* sekcija koja opisuje pogodnosti kupovanja unutar trgovine. Sve ove sekcije i komponente su vidljive na slikama 63 i 64.



Sed Ut Perspiciatis Unde Omnis Iste Natus Error Sit Voluptatem Accusantium Doloremque Laudantium

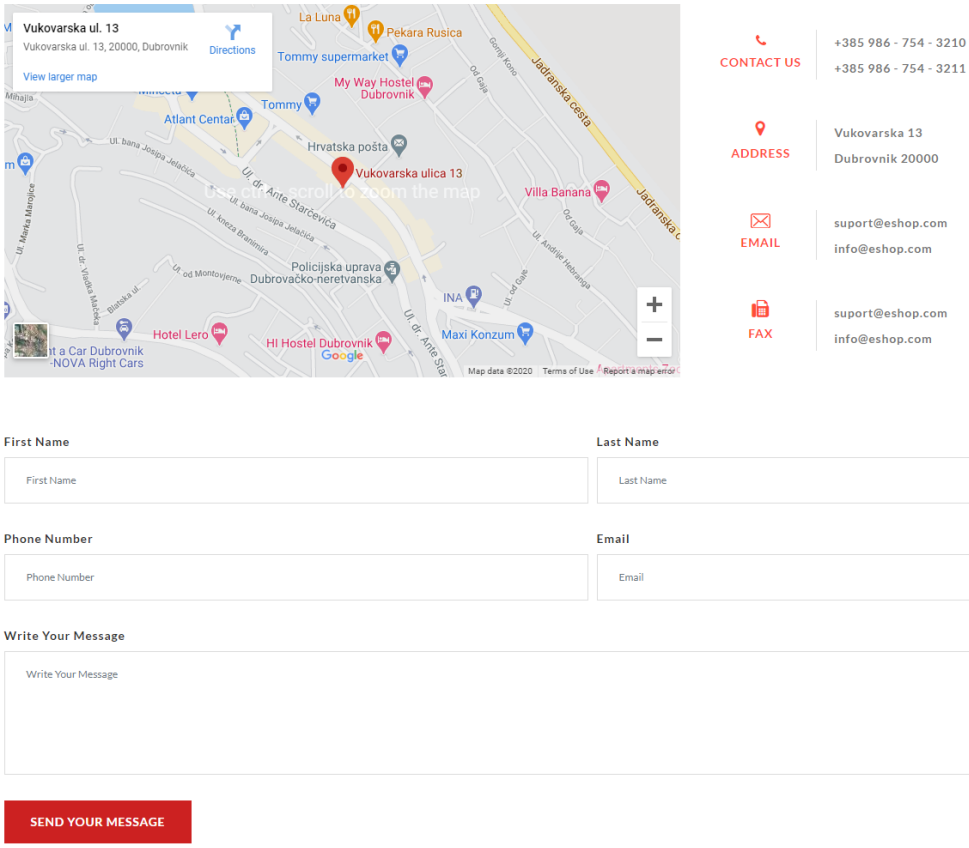
Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium,
On the other hand, we denounce with righteous indignation and dislike men who are so beguiled and demoralized by the charms of pleasure of the moment, so blinded by desire, that they cannot foresee the pain and trouble that are bound to ensue; and equal blame belongs to those who fail in their duty through weakness of will, which is the same as saying through shrinking from toil and pain. These cases are perfectly simple and easy to distinguish. In a free hour, when our power of choice is untrammelled and when nothing prevents our being able to do what we like best, every pleasure is to be welcomed and every pain avoided. But in certain circumstances and owing to the claims of duty or the obligations of business it will frequently occur that pleasures have to be repudiated and annoyances accepted. The wise man therefore always holds in these matters to this principle of selection: he rejects pleasures to secure other greater pleasures, or else he endures pains to avoid worse pains.

Slika 63 O Nama sekcija



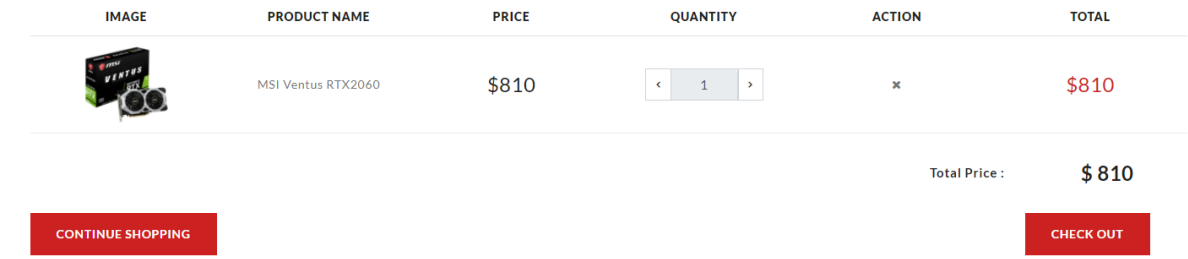
Slika 64 Tim komponenta i Servis sekcija.

Kontakt stranica sadrži mapu, listu kontakta i kontakt formu vidljive na slici 65. Kontakt stranica koristi *axios* za slanje mailova pomoću kontakt forme.

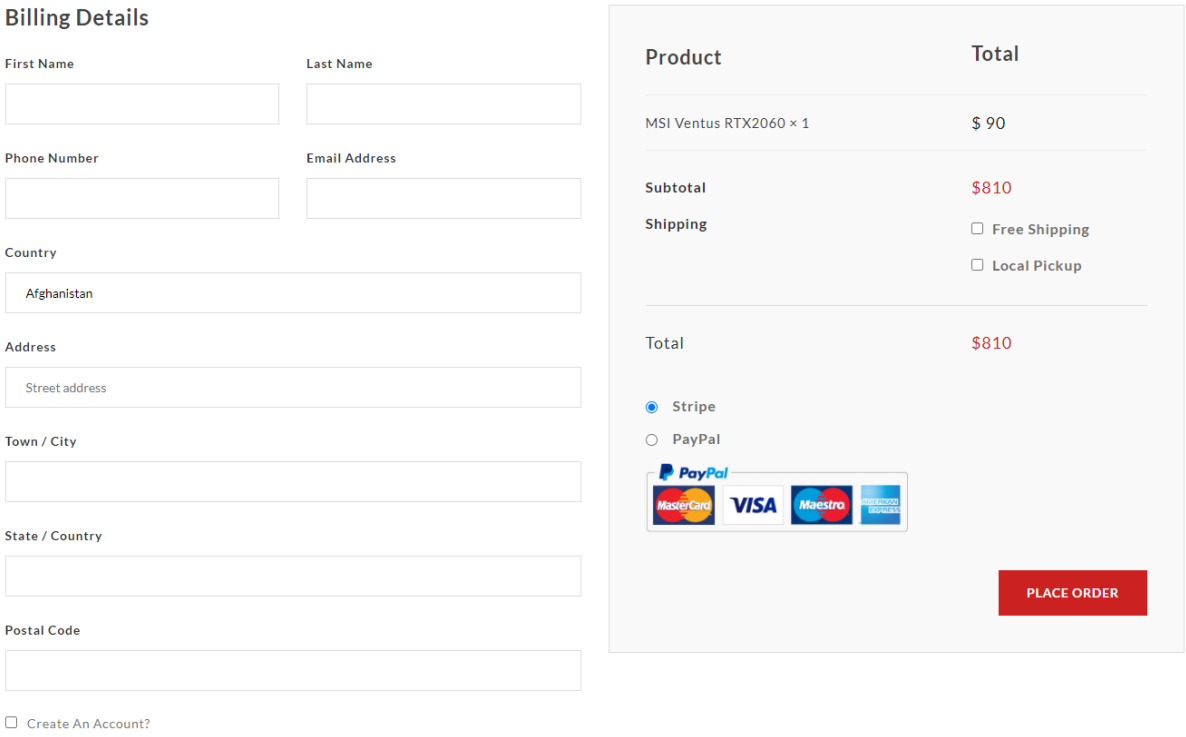


Slika 65 Sadržaj kontakt stranice

Osim spomenutih, stranice koje je vrijedno spomenuti su *Cart*, *Checkout*, *Compare i Wishlist* stranice, te *Language* komponentu. Stranica Košarica (*Cart*) (Slika 66) ispisuje sadržaj košarice korisniku. Unutar nje se ispisuje svaki izabrani proizvod, cijena pojedinačnog proizvoda, količina proizvoda u košarici, gumb za izbrisati proizvod iz košarice, ukupna cijena proizvoda i ukupna cijena košarice. Iz *Cart* stranice se može vratiti na stranicu Trgovina ili dalje na stranicu Plaćanje. *Checkout* stranica, vidljiva na slici 67, zahtjeva podatke korisnika i pruža opciju plaćanja odabranih proizvoda pomoću *PayPala* [47] ili *Stripea* [48]. Obje opcije omogućuju internetsko plaćanje.

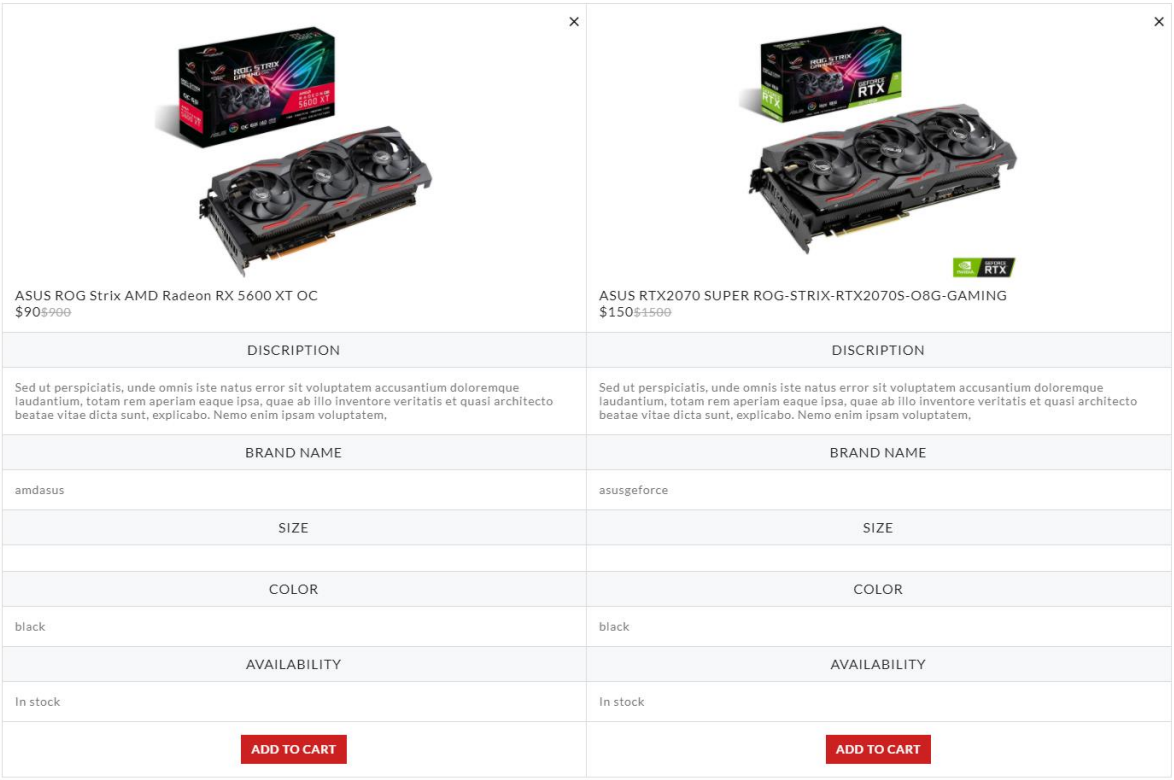


Slika 66 Sadržaj stranice Košarice

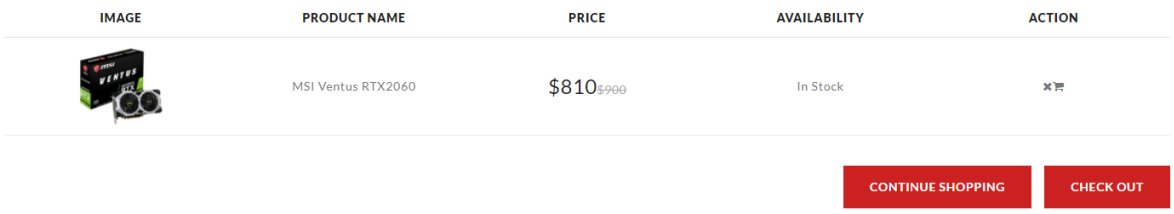


Slika 67 Sadržaj stranice Plaćanja

Stranica Usporedba omogućuje korisniku da usporedi bilo koje proizvode na stranici. Ispisuje se sadržaj svih proizvoda u obliku kartica jedna do druge pomoću kojih korisnik može usporediti sve dijelove proizvoda. Uz stranicu Usporedba postoji i stranica Željeni Proizvodi u koju korisnik može staviti sve proizvode koji mu se sviđaju. Također korisnik može izbrisati odabir proizvoda na stranici, staviti proizvod u košaricu, vratiti se u trgovinu i otići na stranicu košarice. Slike 68 i 69 prikazuju ove dvije stranice. Vrijedna komponenta za spomenuti je komponenta za mijenjanje jezika (*Language*) koja omogućuje izmjenu jezika na stranici između hrvatskog i engleskog.



Slika 68 Stranica usporedbe proizvoda

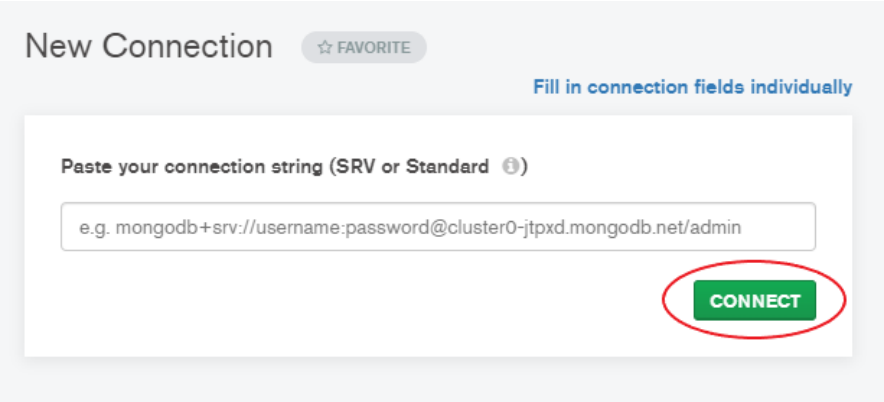


Slika 69 Stranica željenih proizvoda

7 UPUTE ZA INSTALACIJU I KORIŠTENJE APLIKACIJE

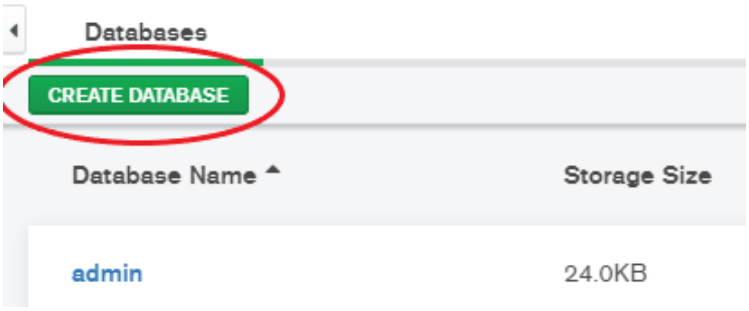
Ove upute bi trebale omogućiti jednostavnu instalaciju aplikacije korisniku na svakom modernom računalu. Preduvjet za korištenje aplikacije je instalacija nekoliko aplikacija prije nego što se može pokrenuti aplikacija: *Node.js* [31], *MongoDB Compass* [32], *Git* [3] ili nekakav drugi sustav za verzioniranje koda i editor koda ili se može koristiti terminal. Nakon što ste instalirali sve potrebne aplikacije sa sljedećeg linka skinite *products.json* dokument tako da kliknete na ime dokumenta:
https://bitbucket.org/vlahocer/diplomski/downloads/ [49].

Pokrenite *MongoDB Compass* aplikaciju. Kad se aplikacija dovrši učitavati trebali bi vidjeti opciju za novu poveznicu (*New Connection*). Nemojte ništa upisivati u prozor ispod nego kliknite na *CONNECT* (Slika 70). Na internet serveru bi se koristili terminalom umjesto aplikacijom te bi se spajali na *MongoDB* bazu tako da bi instalirali *MongoDB* ekstenziju na server te unutar terminala upisali odgovarajuće zapovijedi za upravljanje bazama podataka koji se mogu vidjeti u dokumentaciji *MongoDBa* [32].



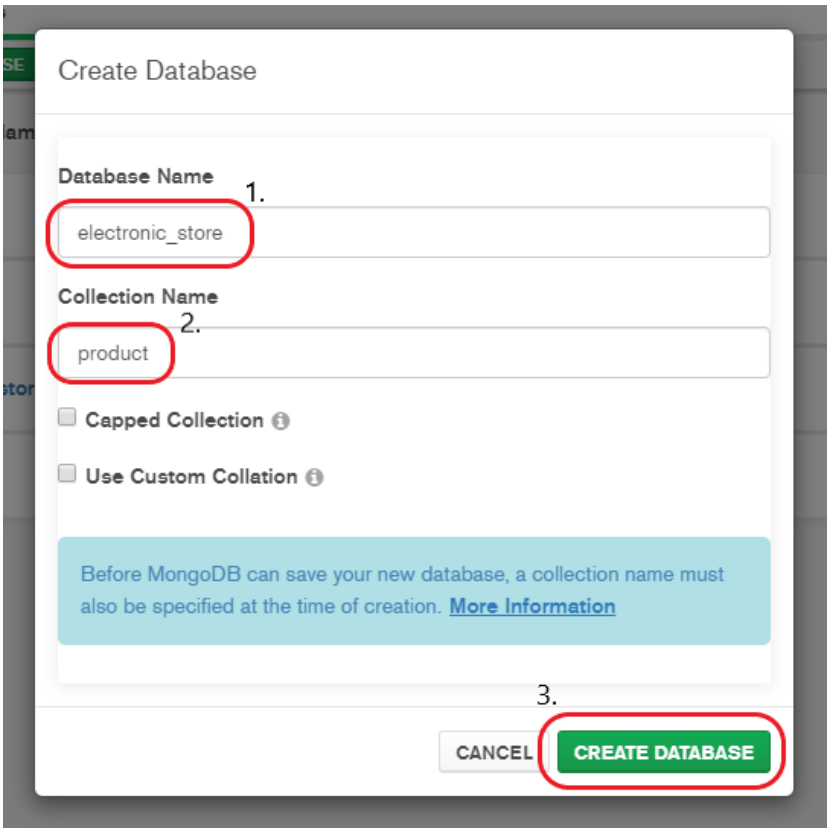
Slika 70 Spajanje na MongoDB

Nakon što se uspješno spojite na *MongoDB* poslužitelj kliknite na *CREATE DATABASE* (Slika 71).



Slika 71 Kreiranje nove baze

Otvoriti će se novi prozor. Upišite za ime baze *electronic_store* i za ime kolekcije *product* te nakon toga kliknite na *CREATE DATABASE* (Slika 72).

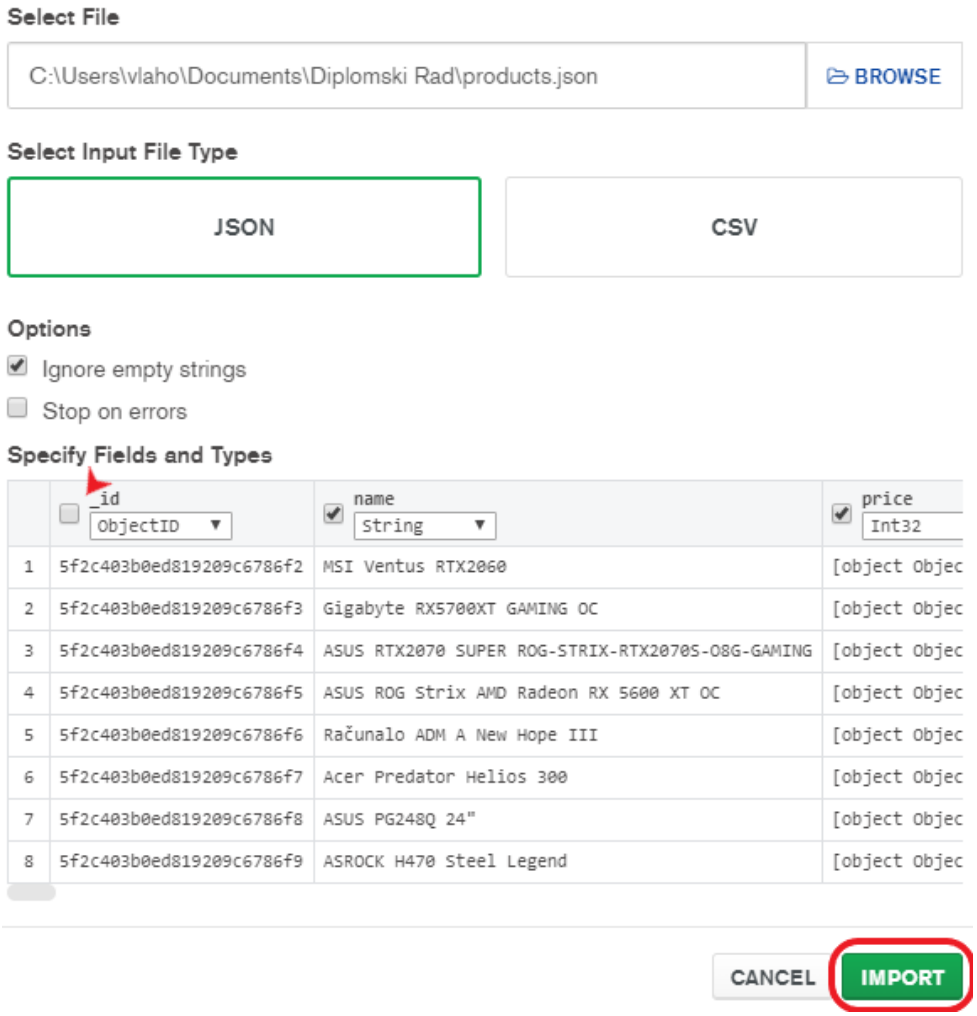


Slika 72 Upisivanje potrebnih podataka za stvaranje baze

Ako se sve uspješno izvršilo trebali bi vidjeti novu bazu u listi *electronic_store*. kliknite na nju, zatim kliknite na kolekciju *product* i trebali biste vidjeti novi prozor na kojemu piše *This collection has no data* i veliki zeleni gumb *Import Data*.

Kliknite na taj gumb, stvoriti će se novi prozor, kliknite na *BROWSE* i izaberite *product.json* dokument koji ste maloprije preuzeli. Nakon što odaberete dokument pojaviti će tablica podataka *Specify Fields and Types* u prozoru ispod izabranoga dokumenta. Kliknite na

kvačicu na *_id* kolumni tako da je poništite i nakon toga kliknite na *IMPORT* (Slika 73) i podaci će biti uneseni u bazu.



Slika 73 Unos podataka u kolekciju

Nakon što ste unijeli bazu podataka klonirajte aplikaciju s *bitbucket* *linka* [50] i pozicionirajte se unutar foldera u kojemu želite raspakirati dokumente i upišite:
git clone *https://vlahocer@bitbucket.org/vlahocer/diplomski.git*.

Sljedeći dio se tiče instalacije i podešavanja pozadinskog poslužitelja. Pričekajte da se preuzmu i raspakiraju datoteke te se potom pozicionirajte u *server* folder unutar *back-end* foldera u istom terminalu. Trebali bi se nalaziti u *.../back-end/server* poziciji unutar terminala. Upišite *npm install* u terminal te pričekajte da se dovrši instalacija paketa. Nakon dovršene instalacije u istom terminalu upišite *node index.js*. Ako je sve u redu trebali bi vidjeti sljedeću poruku unutar

terminala *Server running on port 3000*. To znači da se poslužitelj uspješno pokrenuo i da možemo dohvatiti podatke s njega.

Otvorite novi terminal i nemojte isključivati prethodni. Unutar novog terminala pozicionirajte se u *front-end* folder. Upišite *npm install* kao i prethodni put i pričekajte da se dovrši. Pozicionirajte se u *src* folder unutar *front-end* foldera. Upišite *node sendMail.js.* Trebali bi vidjeti poruku *Server is ready to take messages* ako se uspješno pokrenuo mail poslužitelj.

Sve što je preostalo je pokrenuti samu aplikaciju. Otvorite novi terminal i ostavite dva na kojima su otvoreni poslužitelji za bazu i mail uključene. Pozicionirajte se kao i za prethodni korak u *scr* folder. Upišite *npm start* i *web* aplikacija bi se trebala otvoriti u vašem pregledniku na *localhost:8000* adresi.

8 ZAKLJUČAK

Izrada *web* aplikacije reaktivnim načinom uvelike olakšava i ubrzava posao *web* programera korisničkog sučelja. *React.js* je *JavaScript* biblioteka koja se bazira na komponentama. Korištenjem *React* komponenti repetitivni posao programiranja se uklanja i sve se piše unutar JSX sintakse koja omogućava pisanje HTML-a direktno unutar *JavaScripta*. Ovakav način izrade ubrzava izradu *web* aplikacija. Virtualni DOM koji *React* koristi omogućava umjesto učitavanja cijele stranice učitavanje samo komponenti koje su se izmijenile te se na taj način ubrzava samo učitavanje stranica. Jedna od velikih mana *React.jsa* je strma krivulja učenja. Za razliku od na primjer *Vue.jsa* koji ima normalnu raspodjelu HTML-a i *JavaScripta*, unutar *Reacta* sve se programira pomoću JSX sintakse za koju treba vremena za naviknuti se i naučiti ako je netko početnik.

Unutar izrađene *web* aplikacije postoji puno izrađenih funkcionalnosti, ali je moguće dodati još određenih dijelova koji bi upotpunili cjelokupni izgled i funkcionalnost stranice. Neke od funkcionalnosti za dodavanje u budućnosti bi bile: korisnički profili (registracija i prijava), blog stranica za objavljivanje vijesti, CMS sustav za obradu podataka i stranica za podršku (*FAQ*).

Izrada reaktivnih aplikacija je u početku vrlo spora dok se ne nauči sintaksa i trikovi koji olakšavaju programiranje, ali nakon malo vremena postaje sve lakše. *React* će najvjerojatnije ostati jedna, ako ne i najviše korištena biblioteka za izradu korisničkih sučelja iako ima vrijedne suparnike poput *Vue.jsa* i *Angular.jsa*. Jedan od najvećih razloga tome je pojednostavljenje izrade *web* stranica pomoću komponenti i JSX sintakse, velika zajednica koja može pomoći svakoj osobi koja ima problema s *React* programiranjem i činjenica da ga je *Facebook*, tvrtka koja je i napravila *React*, popularizirala i koristi ga za vlastite sustave.

9 LITERATURA

[1] R. Connolly i R. Hoar, Fundamentals of Web Development, 2 ur., Pearson Education, 2018.

[2] »History Computer,« [Mrežno]. Dostupno na: https://history-computer.com/Internet/Conquering/Mosaic.html. [Pokušaj pristupa 23 9 2020].

[3] S. Chacon, »Git,« [Mrežno]. Dostupno na: https://git-scm.com/. [Pokušaj pristupa 23 9 2020].

[4] »CloudFlare,« CloudFlare, [Mrežno]. Dostupno na: https://www.cloudflare.com/learning/ssl/what-is-https/. [Pokušaj pristupa 23 9 2020].

[5] M. contributors, »MDN web docs,« Mozilla, [Mrežno]. Dostupno na: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS. [Pokušaj pristupa 23 9 2020].

[6] »npm,« npm, [Mrežno]. Dostupno na: https://www.npmjs.com/. [Pokušaj pristupa 23 9 2020].

[7] »yarn,« yarn, [Mrežno]. Dostupno na: https://yarnpkg.com/. [Pokušaj pristupa 23 9 2020].

[8] Sass, »sass-lang.com,« Sass, [Mrežno]. Dostupno na: https://sass-lang.com/documentation. [Pokušaj pristupa 12 Rujan 2020].

[9] A. Sitnik, »PostCSS,« PostCSS, [Mrežno]. Dostupno na: https://postcss.org/. [Pokušaj pristupa 23 9 2020].

[10] »Webpack,« Webpack, [Mrežno]. Dostupno na: https://webpack.js.org/. [Pokušaj pristupa 23 9 2020].

[11] Facebook, »React,« Facebook, [Mrežno]. Dostupno na: https://reactjs.org/docs/react-api.html. [Pokušaj pristupa 23 9 2020].

[12] »Angular,« Google, [Mrežno]. Dostupno na: https://angularjs.org/. [Pokušaj pristupa 23 9 2020].

[13] E. You, »Vue,« Vue, [Mrežno]. Dostupno na: https://vuejs.org/. [Pokušaj pristupa 23 9 2020].

[14] M. Otto i J. Thornton, »Bootstrap,« Bootstrap, [Mrežno]. Dostupno na: https://getbootstrap.com/. [Pokušaj pristupa 23 9 2020].

[15] »Reactstrap,« Reactstrap, [Mrežno]. Dostupno na: https://reactstrap.github.io/. [Pokušaj pristupa 23 9 2020].

[16] O. Isonen, »Material UI,« Material UI, [Mrežno]. Dostupno na: https://material-ui.com/. [Pokušaj pristupa 23 9 2020].

[17] A. Wathan i S. Schoger, »Tailwindcss,« Tailwindcss, [Mrežno]. Dostupno na: https://tailwindcss.com/. [Pokušaj pristupa 23 9 2020].

[18] S. Richard i P. LePage, »Web Dev,« Web Dev, [Mrežno]. Dostupno na: https://web.dev/progressive-web-apps/. [Pokušaj pristupa 23 9 2020].

[19] T. Neutkens, »Next.js,« Vercel, [Mrežno]. Dostupno na: https://nextjs.org/. [Pokušaj pristupa 23 9 2020].

[20] L. K. Liau, »Angular Universal,« Angular, [Mrežno]. Dostupno na: https://angular.io/guide/universal. [Pokušaj pristupa 23 9 2020].

[21] A. Chopin, S. Chopin i P. Parsa, »Nuxtjs,« Nuxt, [Mrežno]. Dostupno na: https://nuxtjs.org/. [Pokušaj pristupa 23 9 2020].

[22] L. Byron, »GraphQL,« GraphQL, [Mrežno]. Dostupno na: https://graphql.org/. [Pokušaj pristupa 23 9 2020].

[23] »Apollo Graph,« Apollo Graph, [Mrežno]. Dostupno na: https://www.apollographql.com/. [Pokušaj pristupa 23 9 2020].

[24] »Relay Modern,« Facebook, [Mrežno]. Dostupno na: https://engineering.fb.com/web/relay-modern-simpler-faster-more-extensible/. [Pokušaj pristupa 23 9 2020].

[25] »Gatsbyjs,« Gatsby, [Mrežno]. Dostupno na: https://www.gatsbyjs.com/. [Pokušaj pristupa 23 9 2020].

[26] Facebook, »React Native,« Facebook, [Mrežno]. Dostupno na: https://reactnative.dev/. [Pokušaj pristupa 23 9 2020].

[27] »Electron,« OpenJS Foundation, [Mrežno]. Dostupno na: https://www.electronjs.org/. [Pokušaj pristupa 23 9 2020].

[28] A. Kamran, »roadmap.sh,« roadmap.sh, [Mrežno]. Dostupno na: https://roadmap.sh/frontend. [Pokušaj pristupa 15 Rujan 2020].

[29] »BitBucket,« Atlassian, [Mrežno]. Dostupno na: https://bitbucket.org/product. [Pokušaj pristupa 23 9 2020].

[30] »PhpStorm,« JetBrains, [Mrežno]. Dostupno na: https://www.jetbrains.com/phpstorm/. [Pokušaj pristupa 23 9 2020].

[31] »Nodejs,« OpenJS Foundation, [Mrežno]. Dostupno na: https://nodejs.org/en/. [Pokušaj pristupa 23 9 2020].

[33] A. L. Carreton i T. V. Cutsem, »A Survey on Reactive Programming,« [Mrežno]. Dostupno na: https://www.researchgate.net/publication/233755674_A_Survey_on_Reactive_Programming. [Pokušaj pristupa 23 9 2020].

[35] »Software Testing Help,« [Mrežno]. Dostupno na: https://www.softwaretestinghelp.com/sql-vs-nosql/. [Pokušaj pristupa 23 9 2020].

[36] »Advantages of MongoDB,« MongoDB, [Mrežno]. Dostupno na: https://www.mongodb.com/advantages-of-mongodb. [Pokušaj pristupa 26 9 2020].

[37] E. W. I. Koroliova, MERN Quick Start Guide: Build web applications with MongoDB, Express.js, React, and Node, 2 ur., Packt Publishing, 2018.

[41] E. Kollegger, »What is Axios.js and why should I care?,« 14 Svibanj 2018. [Mrežno]. Dostupno na: https://medium.com/@MinimalGhost/what-is-axios-js-and-why-should-i-care-7eb72b111dc0. [Pokušaj pristupa 9 Rujan 2020].

[43] Automattic, »Mongoose ODM,« [Mrežno]. Dostupno na: https://mongoosejs.com/docs/guide.html. [Pokušaj pristupa 11 Rujan 2020].

[46] »Cross-Origin Resource Sharing,« [Mrežno]. Dostupno na: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS. [Pokušaj pristupa 11 Rujan 2020].

[47] T. Goode, »Express cors middleware,« [Mrežno]. Dostupno na: https://expressjs.com/en/resources/middleware/cors.html. [Pokušaj pristupa 11 Rujan 2020].

10 PRILOZI

10.1 Popis dijagrama

10.2 Popis slika

IZJAVA

Izjavljujem pod punom moralnom odgovornošću da sam diplomski rad izradio samostalno, isključivo znanjem stečenim na studijima Sveučilišta u Dubrovniku, služeći se navedenim izvorima i literaturom i uz stručno vodstvo mentora doc.dr.sc. Krunoslava Žubrinića i komentorice Ane Kešelj, kojima se još jednom srdačno zahvaljujem.

U Dubrovniku, 28. rujna 2020.