IZRADA WEB APLIKACIJE ZA PRODAJU ODJEĆE

Galac, Šime

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijska tehnologija

ŠIME GALAC

Z A V R Š N I R A D

Izrada web aplikacije za prodaju odjeće

Split, rujan 2020.

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijska tehnologija

Predmet: Mrežne usluge i programiranje

Z A V R Š N I R A D

Kandidat: Šime Galac

Naslov rada: Izrada web aplikacije za prodaju odjeće

Mentor: Haidi Božiković, viši predavač

Split, rujan 2020.

Sadržaj

Sažetak

Tema ovog završnog rada je internet trgovina za prodaju odjeće. Izrada završnog rada je uključivala korisnički dio gdje se može obaviti kupnja, mogu se pregledavati narudžbe i uređivati osobni podaci. Aplikacija uključuje i administratorski dio u kojem administrator ima mogućnost uređivanja proizvoda i pregled svih narudžbi. Za izradu ovog rada je korištena biblioteka *React*, verzija 16.8, za klijentski dio aplikacije te razvojni okvir *SpringBoot*, verzija 2.2, za serverski dio. Izabrane su navedene tehnologije zbog njihovih karakteristika. *React* koristi programski jezik JavaScript koji je trenutno jedan od najpopularnijih na tržištu. Razvojni okvir *SpringBoot* programski jezik koristi Java. Pomoću njega je moguće vrlo jednostavno postaviti aplikaciju i posvetiti se razvijanju poslovne logike aplikacije.

Ključne riječi: *JavaScript*, *JSX*, komponente, *React*, *Redux*, *SpringBoot*

Summary

Development of clothing web shop

This article's main topic is web shop for clothes. Application consists of two parts. First one is a client side application where user can make a purchase, can see orders that have been made and can edit account data. There is also an administrator interface where the administrator can see all orders that have been made. Administrator can edit products, as well. This part is made using React, version 16.8. Second part is a server side application which processes all information about the application. This part was made using SpringBoot framework, version 2.2. These technologies were chosen because of their charachteristics. React uses JavaScript programming language. JavaScript is one of the most popular ones and is used in huge amount of websites. SpringBoot framework uses Java programming language, offers quick application setup and allows developers to focus on business logic.

Key words: *JavaScript*, *JSX*, komponente, *React*, *Redux*, *SpringBoot*

1. Uvod

U današnje vrijeme je najveća potreba za programerima zbog ogromnog broja tehnologija i mogućnosti koje one pružaju. To se pogotovo odnosi na programiranje na internetu gdje na dnevnoj bazi dolaze nove biblioteke koje programer može koristiti za razvijanje svog softvera. Jedna od takvih biblioteka je i *React* (*ReactJS* ili *React.js*), razvijena od *Facebooka*, pa je ona korištena za izradu svog projekta budući da je potražnja za programerima koji ju znaju koristiti velika. *React* je po definiciji biblioteka za oblikovanje korisničkog sučelja pa samo ona nije dovoljna za potpuno funkcionalnu aplikaciju. U projektu je korištena biblioteka *Redux*, pomoću koje je moguće upravljati stanjem aplikacije. Nadalje, korišten je razvojni okvir (eng. *framework*) *SpringBoot*, pomoću kojeg je moguće napraviti aplikaciju bez puno konfiguracije i koje se mogu „samo pokrenuti“. Za korištenje razvojnog okvira *SpringBoot* je potrebno poznavanje programskog jezika *Java*.

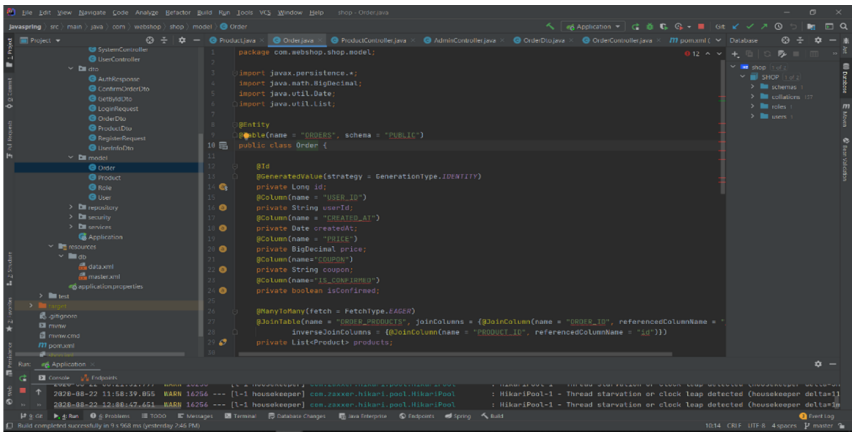
Budući da je zadatak bio napraviti internet trgovinu za prodaju odjeće, napravljene su dvije aplikacije koje su u međusobnoj interakciji. Prva aplikacija je ona izrađena u razvojnom okviru *SpringBoot* gdje su postavljene krajnje točke (eng. *endpoints*) za komunikaciju s drugom aplikacijom, onoj izrađenoj u biblioteci *ReactJS*. Nakon toga su izrađeni modeli koji će se koristiti u obje aplikacije te spajanje na bazu podataka kako bi svi podaci mogli biti spremljeni te korišteni prema potrebi korisnika. Druga aplikacija, se spaja na krajnje točke prve, dobije podatke koje zatraži, obrađuje ih te ih na kraju prikazuje korisniku. Budući da se *React* aplikacija gradi pomoću komponenti, cilj je napraviti komponente koje će se moći ponovno iskoristiti. To je samo jedna od prednosti koje pruža ova biblioteka pa je i to bio jedan od razloga zašto je izabrana. Obje aplikacije su pisane u razvojnom okruženju (eng. IDE – *Integrated Development Environment*) *IntelliJ Idea* budući da podržava pisanje gotovo svih programskih jezika, a isto tako omogućuje korištenje sustava za kontrolu verzija (eng. *version control system*) kao što je Git. U sljedećim poglavljima će biti rečeno koje su tehnologije korištene za izradu aplikacije te će biti opisan sâm način izrade aplikacije.

2. Korištene tehnologije

2.1. *IntelliJ Idea*

IntelliJ Idea je softver za pisanje kôda baziran na programskom jeziku Java. Razvijen je 2001. godine i tada je bio jedan od prvih softvera koji je omogućavao naprednu navigaciju po kôdu te refaktoriranje. Ovaj softver podržava mogućnost dodavanja brojnih dodataka kao što su dodaci za *SpringBoot*, koji je korišten u ovom završnom radu, programski jezik *Scala* te brojni drugi. Ovaj alat ima brojne prednosti koje olakšavaju rad programeru. Neki od njih su pametno nadopunjavanje kôda, uočavanje dupliciranog kôda te mogućnost umetanja kôda drugog programskog jezika u trenutno korišteni. Osim toga, ima ugrađeno otklanjanje pogrešaka (eng. *debugging*) pomoću kojega se mogu postaviti točke prekida (eng. *breakpoints*) na kojima će se aplikacija zaustaviti tijekom izvođenja što uvelike olakšava uočavanje pogrešaka. Nadalje, za sve postoje prečaci pa se tako na primjer, tijekom kreiranja modela, kada je potrebno napisati *get*() i *set*() metode, može pomoću kombinacije tipki na tipkovnici jednostavno umetnuti potrebni dio kôda [1].

Jedna od najzanimljivijih stvari je mogućnost dekompajliranja. Tijekom izrade projekta, svatko se susretne s nekakvim problem, pa je uz pomoć *IntelliJ Idea*-inog dekompajliranja moguće imati uvid u klase koje su potrebne za implementaciju nekog rješenja te tako vidjeti kako neka funkcionalnost točno radi, koji joj podaci trebaju i kako se ti podaci koriste. Kao i mnoga druga razvojna okruženja, *IntelliJ Idea* koristi više boja što uvelike olakšava čitanje kôda. Isto tako, vrlo je jednostavno pregledavati zapise u bazi podataka pomoću alatne trake ugrađene u ovo razvojno okruženje, slika 1.



Slika 1: Primjer radne okoline *IntelliJ Idea*

2.2. *SpringBoot*

Budući da je *SpringBoot* dio platforme *Spring*, za početak će biti objašnjen koncept na kojemu je *Spring* baziran.

Spring je razvojni okvir za programski jezik *Java,* otvorenog je kôda te koristi princip *Inversion of Control* gdje se mijenja paradgima u odnosu na proceduralno programiranje gdje program poziva kôd koji se treba izvršiti, a u slučaju *Inversion of Control* principa to odrađuje razvojni okvir. Ovaj princip omogućuje veću modularnost aplikacije te ima primjenu u objektno orijentiranom programiranju. Neki od modula koji su uključeni u razvojni okvir *Spring* su autentifikacija i autorizacija, pristup podacima pomoću kojeg se može raditi s relacijskim bazama podataka (u slučaju ovog rada je korišten JPA (eng. *Java Persistence API*)), MVC (*Model – View – Controller*) koji omogućuje prilagođavanje *web* aplikacija i *RESTful* (eng. *REpresentational State Transfer*) servisa. Isto tako, moguće je testirati kôd pomoću *Spring* klasi za pisanje *unit testova* [2].

SpringBoot je rješenje *Spring* tima za aplikacije koje je moguće „samo pokrenuti“. *SpringBoot* aplikacije zahtijevaju minimalnu konfiguraciju jer je već konfiguriran za korištenje sa Spring platformom i ostalim rješenjima trećih strana. Zbog navedenog, ovo

rješenje se zove konvencija prije konfiguracije (eng. *convention over configuration*). *SpringBoot* ne zahtjeva nikakvu XML(eng. *eXtensive Markup Language*) konfiguraciju, a pruža datoteku POM (eng. *Project Object Model*), koja je vidljiva u ispisu 1, pomoću koje se jednostavno mogu konfigurirati ovisnosti *Maven* (eng. *Maven dependencies*) koje će biti korištene u projektu. U *SpringBoot* je uključen i *Tomcat* – implementacija otvorenog kôda *Java servleta* – malog programa koji prima i šalje zahtjeve klijentu, najčešće preko protokola HTTP (eng. *HyperText Transfer Protocol*). *Tomcat* je razvijen i održavan odstrane *Apache Software Foundation* te omogućuje okruženje internetskog poslužitelja u kojem se *Java* kôd može pokretati [3].

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>1.4.200</version>

</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
```

Ispis 1: Izgled datoteke *pom.xml*

2.3. Baza podataka H2

Baza podataka korištena u ovom projektu je H2. To je relacijska baza podataka napisana u *Java* programskom jeziku. Može biti uključena u bilo koji *Java* projekt ili biti pokrenuta u klijentsko-poslužiteljskom načinu. H2 je baza podataka koja koristi RAM memoriju računala (eng. *in-memory database*) pa ukoliko dođe do nenadanog gašenja računala, ostaje se bez svih podataka. Međutim može se koristiti s ostalim bazama podataka koji ne čuvaju podatke u radnoj memoriji. Zbog svega toga, puno je brža jer je RAM memorija puno brža od na primjer, tvrdog diska računala [4].

2.4. *React*

Biblioteka *React JavaScript* je jedna od najviše korištenih biblioteka u današnje vrijeme za izradu korisničkih sučelja. Mnoge tvrtke koriste *React* za izradu svojih *web* stranica kao što su *Facebook, Instagram, Netflix, Reddit, Uber, WhatsApp* i brojne druge. Osim što koristi *React*, *Facebook* je razvio tu *JavaScript* biblioteku koja sadrži preko 20000 *React* komponenti na svojoj stranici. Svako *web* mjesto koje koristi *React* sadrži mnogo komponenti te se često uz *React* koriste razni drugi razvojni okviri ili *JavaScript* biblioteke kao što je *React Native* za razvoj aplikacija za mobilne platforme. *Instagram* koristi *React* zbog responzivnosti, *Reddit* zbog bržeg učitavanja linkova ili nekog drugog sadržaja i boljeg korisničkog iskustva. Svako *web* mjesto koje je koristi *React* funkcionira bolje prema mišljenjima većine korisnika [5].

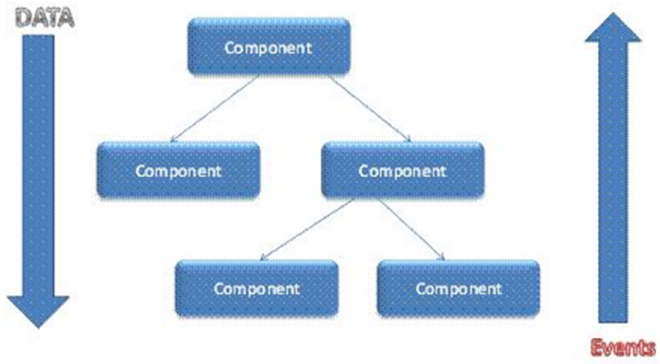
2.4.1 JSX (JavaScript XML)

Kada je riječ o *Reactu*, prva stvar koju treba spomenuti je JSX. JSX je sintaktička ekstenzija programskog jezika *JavaScript*. Razvojni tim koji stoji iza *Reacta* preporučuje korištenje JSX sintakse jer omogućava opisivanje korisničkog sučelja. Iako nije obavezno, korištenje JSX-a uvelike olakšava uklanjanje pogrešaka i upozorenja u kôdu. Nakon što se aplikacija kompajlira, JSX izrazi postaju obični pozivi JS (eng. *JavaScript*) funkcija što znači da je moguće da se JSX može koristiti unutar uvjeta i petlji. Pretvaranje JSX sintakse u JS omogućuje *Babel* [6].

2.4.2. Komponente

Komponente su najvažniji dio u *React*u. Konceptualno, komponente su kao JS funkcije. Primaju parametre, koji se nazivaju *props* (eng. *properties)* te vraćaju *React* elemente koji opisuju što će se pojaviti na ekranu. Komponente se mogu pisati kao funkcije ili kao klase po ES6 specifikaciji. U ovom radu, sve komponente su napisane kao klase. Komponente je potrebno nazivati velikim početnim slovom jer *React* komponente koje počinju malim početnim slovom tretira kao DOM elemente, a ne kao *React* elemente. Sve

React komponente se moraju ponašati kao „čiste funkcije“ što znači da ne smiju mijenjati *propse*. U *React*u je tok podataka jednosmjeran, kao što se može vidjeti na slici 2, što znači da „dijete“ komponenta ne može poslati *propse* „roditelju“ komponenti. Međutim, akcije mogu ići drugom smjeru i to pomoću JS *callback* funkcije s kojom je moguće mijenjati podatke [7].



Slika 2: Koncept *React*a

Budući da komponente trebaju biti interaktivne za korisnika, te bi se trebale mijenjati u ovisnosti o korisnikovim akcijama ili nečemu drugom, u *React*u postoji koncept stanja (eng. *state*) komponente.

Stanje komponente se inicijalizira u konstruktoru komponente. Ono služi da se u ovisnosti o korisnikovim akcijama mijenja izgled komponente, pa na primjer, ukoliko korisnik klikne na neki padajući izbornik, on će se otvoriti, a stanje će se promijeniti. Stanju komponente je moguće dodati neograničeni broj polja i tako kontrolirati više elemenata u jednoj komponenti [8].

```
class PaymentConfirmation extends Component {
    constructor(props) {
        super(props);
        this.state = {
            productDtos: this.props.items,
            price: this.props.price
        }
    }
```

```
    handleOrder = () => {
        const {order} = this.props;
        order(this.state);


    }

    render() {

        return (
            <div><Button disabled={this.props.buttonDisabled === true
?   true : false} className={"myButton"}
onClick={this.handleOrder}>Naruči</Button> </div>
        )
    }
}

export default PaymentConfirmation;
```

Ispis 2: Primjer jednostavne komponente u React-u

U ispisu 2 se može vidjeti kako je unutrašnje stanje komponente inicijalizirano u konstruktoru komponente. Stanju se pridjeljuju *props-i* iz komponente roditelja. Komponenta se samo sastoji od jednog gumba, a pritiskom na njega se izvršava narudžba. Može se vidjeti da će gumb biti neaktivan ukoliko je *prop* imena „*buttonDisable*“ postavljen na vrijednost „*true*“. Također, klikom na gumb, aktivira se događaj „*handleOrder*“ – obična funkcija u kojoj se destrukturira metoda „*order*“ iz *props-a* te se ona zatim poziva a parametar joj je unutrašnje stanje komponente.

Životnim ciklusom komponente se upravlja metodama za životni ciklus (eng. *lifecycle methods*). Pomoću njih se mogu obavljati neke akcije nakon što je komponenta postavljena u DOM ili prije nego što će komponenta biti uklonjena iz DOM-a.

2.4.3. DOM

DOM je apstrakcija HTML strukture stranice nekog *web* mjesta i to je W3C (eng. *World Wide Web Consortium*) standard. *React* uzima HTML elemente i pakira ih u jedan objekt u hijerarhijskom obliku te zadržava odnos roditelj/dijete (eng. *parent/child relationship*) kod tih HTML elemenata. Za manipuliranjem DOM-om se koristi *JavaScript*, no

treba biti oprezan jer se zbog manipuliranja smanjuje efikasnost s obzirom da je DOM bio predviđen za statične *web* stranice, a ne dinamične. Problem se javlja kod prikazivanja sadržaja (eng. *rendering*). Kada se jedna stavka ažurira, cijela lista stavki se ponovno prikazuje što nije efikasno zbog mogućeg velikog broja elemenata pa je iz tog razloga *React* počeo koristi virtualni DOM. Virtualni DOM je zapravo kopija DOM-a. *React* upravlja s dva virtualna DOM-a koja su u dva različita stanja – jedno je sljedeće stanje, a drugo je prijašnje stanje, odnosno stanje prije ažuriranja. Na taj način se pravi DOM može ažurirati u minimalnom broju koraka [9] [10].

Za razliku od DOM-a (eng. *Document Object Model*) u HTML-u (eng. *Hyper Text Markup Language*), elementi u *React*-u su obični objekti koje je lako kreirati. U pozadini, *React* DOM se brine o ažuriranju DOM-a. Svaka datoteka „*index.html"* sadrži jedan DOM element, zadano nazvan „*root*", u koji se ubacuju ostali *React* elementi.

2.5. Biblioteka *react-router*

React-router je još jedna JS biblioteka korištena u ovom projektu. Budući da je aplikacija izrađena u *Reactu* SPA (eng. *Single Page Application*), *react-router* je nužan kako bi korisniku prikazao ono što korisnik želi. *React-router* „gleda" URL (eng. *Uniform Resource Locator*) u adresnoj traci te prikazuje komponente koje su vezane za određeni URL [11].

```
<Switch>
    <Route path="/muskarci">
        <MenPage/>
    </Route>
    <Route path="/zene">
        <WomenPage/>
    </Route>
    <Route path="/djeca">
        <KidsPage/>
    </Route>
    <Route path="/contact">
        <ContactPage/>
    </Route>
    <Route path="/about">
        <AboutPage/>
    </Route>

    <Route path="/login">
```

```
        <LoginPage/>
    </Route>

    <Route path="/admin">
        <AdminDashboardPage/>
    </Route>
    <Route path="/dodaj">
        <AddProductPage></AddProductPage>
    </Route>

    <Route path="/register">
        <RegistrationPage></RegistrationPage>
    </Route>
    <Route path="/narudzbe">
        <OrdersPage></OrdersPage>
    </Route>
    <Route path="/myinfo">
        <UserInfoPage></UserInfoPage>
    </Route>
    <Route path="/cart">
        <CartPage></CartPage>
    </Route>
    <Route path="/checkout">
        <CheckoutPage></CheckoutPage>
    </Route>
    <Route path="/">
        <DashboardPage></DashboardPage>
    </Route>
    <Route path="/terms">
        <Terms></Terms>
    </Route>
</Switch>
```

Ispis 3: Izgled datoteke s rutama u aplikaciji

Na ispisu 3 se može vidjeti kako *react-router* za vrijednost u adresnoj traci „*login*“, korisniku prikazuje komponentu „*LoginPage*“. „*LoginPage*“ u ovom slučaju sadrži još jednu komponentu u sebi, a može ih sadržavati i puno više.

2.6. *Redux*

Redux je JS biblioteka otvorenog kôda, također korištena u ovom projektu, pomoću koje je moguće upravljati stanjem aplikacije. Budući da su svi podaci korišteni u *React* aplikaciji došli od *SpringBoot* aplikacije, moraju biti spremljeni na jedno mjesto odakle će im komponente moći pristupiti te ih prikazati. *Redux* omogućuje da pojedina komponenta pristupi *redux store-u* te preuzme podatke koji joj trebaju. *Redux* funkcionira na principu akcija i

reduktora (eng. *reducer*). Akcija je obični JS objekt koji objašnjava što se trenutno događa. Za svaku akciju postoji reduktor, čista funkcija – za jednake parametre vraća uvijek isti rezultat, koji prima stanje aplikacije i akciju te na temelju toga mijenja postojeće stanje i vraća novo. Na ispisu 4 se može vidjeti *redux* dio komponente. Da bi komponenta mogla pristupiti stanju aplikacije potrebno je iskoristiti metodu *connect*() pomoću koje se komponenta spaja na *store* te ona za parametre prima još dvije funkcije, a to su *mapStateToProps* i *mapDispatchToProps*. Kao što joj ime samo govori, funkcija *mapStateToProps* mapira stanje aplikacije(*redux* stanje) u *propse* komponente, a funkcija *mapDispatchToProps* mapira funkcije u *propse*. Funkcije koje su mapirane u *propse* se u komponenti mogu pozivati kao i bilo koja druga funkcija. To može biti na korisnikov pritisak tipke ili ukoliko je zadovoljen neki uvjet [12].

```
const mapStateToProps = state => {
    return {
        products: state.shop.products,

    };
};
const mapDispatchToProps = dispatch => {
    return {
        getProducts: () => dispatch(getProducts()),
        addToCart: (product) => dispatch(addToCart(product))

    }
};


export default connect(
    mapStateToProps,
    mapDispatchToProps
)(MenPage)
```

Ispis 4: Redux dio komponente

2.7. *Bootstrap*

Bootstrap je razvojni okvir, besplatan za korištenje, koji je namijenjen prvenstveno za razvoj aplikacija koji se prikazuju na mobilnim uređajima. Sadrži CSS i JS predloške za većinu HTML elemenata. *Reactstrap* je implementacija *Bootstrap*-a koja se može koristiti u

React aplikacijama i to na način da je svaka klasa u *Bootstrap*-u jedna komponenta *Reacstrap*-a. Da bi se *Reacstrap* mogao koristiti, u projekt mora biti uključen i *Bootstrap* jer, kao što je već rečeno, *React* pretvara svaku komponentu u običan HTML element te mu pridjeljuje CSS klasu, ukoliko ona postoji, da bi se on prikazao kako programer želi. Budući da je *Bootstrap* namijenjen razvijanju aplikacija za prvenstveno mobilne uređaje, koristeći njegove klase moguće je pravilno prikazati sadržaj na pametnom telefonu ili tabletu [13][14].

2.8. *Babel*

Babel je JS kompajler koji se uglavnom koristi za pretvaranje ES6 (*EcmaScript 6*) sintakse JS-a u starije verzije da bi bile kompatibilne s preglednikom koji korisnik koristi. *Babel* se jednostavno može dodati u projekt pomoću npm-a (eng. *node package manager*) [15].

2.9. *Create-react-app*

U ovome slučaju nije dodavan Babel jer je aplikacija postavljena uz pomoć *create-react-app*. *Create-react-app* je *node* paket s kojim se može vrlo jednostavno postaviti aplikacija bez ikakvih konfiguracija [16].

```
my-app/
  README.md
  node_modules/
  package.json
  public/
    index.html
    favicon.ico
  src/
    App.css
    App.js
    App.test.js
    index.css
    index.js
    logo.svg
```

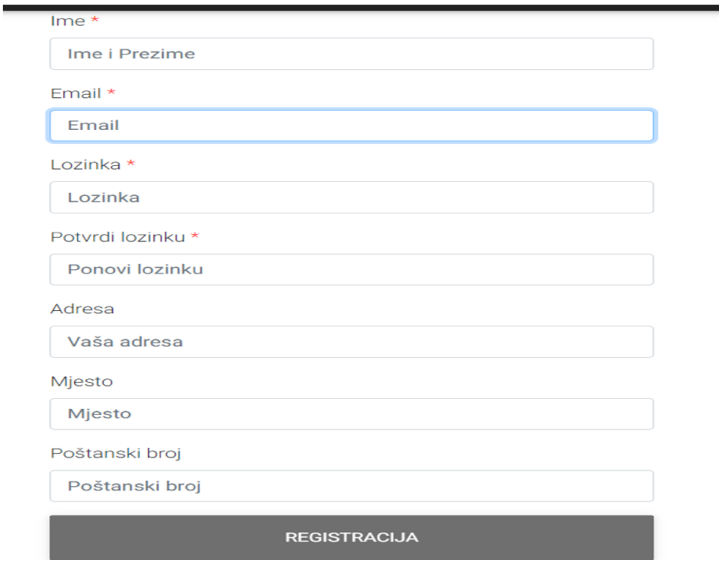
Slika 3: Struktura *React* aplikacije postavljenje pomoću *create-react-app*

3. Opis praktičnog rada

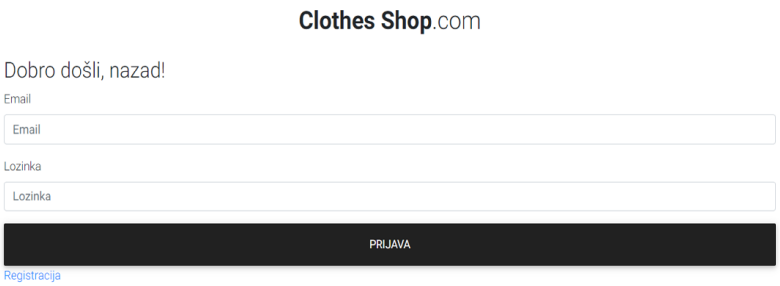
3.1. Klijentski dio

Tema ovog završnog rada je bila izrada internet trgovine odjeće u kojoj će korisnik moći pregledavati proizvode po kategorijama, dodavati ih u košaricu te na kraju obaviti kupnju. Plaćanje u ovoj internet trgovini je moguće obaviti samo pouzećem. Osim toga, korisnik može pregledati sve svoje narudžbe i ima pristup svojim podacima. Također, u aplikaciji postoji i administratorski dio gdje administrator ima sve mogućnosti kao i korisnik te još ima pristup svim narudžbama, koje može odobriti, i svim korisnicima. Administrator može i dodavati proizvode u trgovinu. Prije svega, bilo je potrebno omogućiti registraciju i prijavu korisnika u aplikaciju.

3.1.1. Registracija i prijava



Slika 4: Izgled obrasca za registraciju

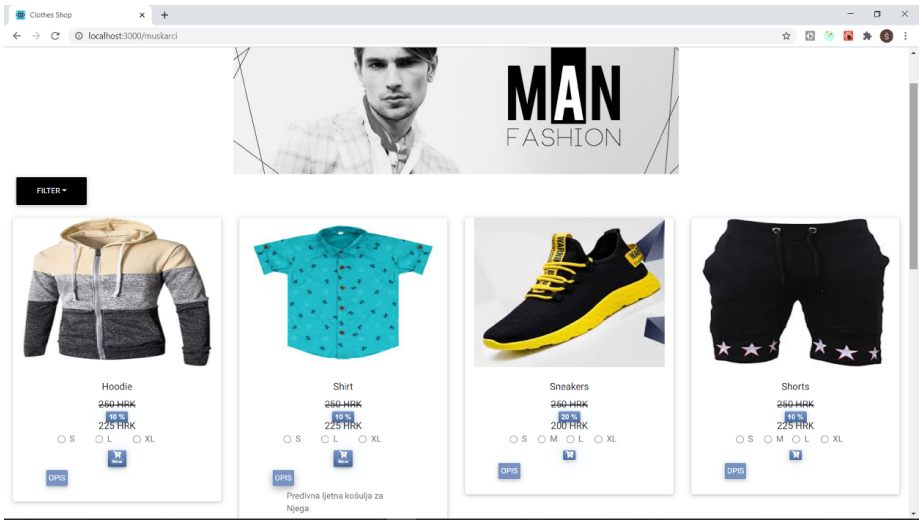


Slika 5: Izgled obrasca za prijavu u aplikaciju

Na slikama 4 i 5 se vide obrasci (eng. *forms*) za registraciju i prijavu. Nakon što se korisnik registrira, preusmjeren je na stranicu za prijavu gdje unosi adresu elektroničke pošte i lozinku. Tijekom registracije, korisnik mora unijeti polja za ime, adresu elektroničke pošte i lozinku dok ostala polja nisu obavezna te ih može kasnije mijenjati u postavkama svog profila.

3.1.2. Proizvodi

Nakon prijave, korisnik je preusmjeren na početnu stranicu gdje su prikazani novi proizvodi i oni koji su na akciji. Kao što se može vidjeti u navigaciji, kategorije između kojih korisnik može pretraživati su one za muškarce, žene i djecu. Osim tih triju kategorija, postoje i podkategorije koje korisnik može filtrirati da bi lakše pronašao proizvod koji želi.



Slika 6: Izgled stranice s muškim proizvodima

Svaka od kategorija sadrži onoliko komponenti imena „*Product*“ koliko postoji proizvoda za tu kategoriju.

```
class Product extends Component {
    constructor(props) {
        super(props);
        this.state = {
            isAdded: false,
            isOpen: false,
            size: ''
        }

    }

    onSizeSelect = (e) => {
        this.setState({size: e.target.value})

    }

    handleClick = () => {
        const {addToCart, product} = this.props;
        product.size = this.state.size;
        addToCart(product);
        this.onShowAlert();
```

```
    }
    onShowAlert = () => {
        this.setState({isAdded: true}, () => {
            window.setTimeout(() => {
                this.setState({isAdded: false})
            }, 2000)
        });
    }
    toggle = () => {
        this.setState({isOpen: !this.state.isOpen})
    }


    render() {
        const {name,
description,newProduct,price,img,category,product, sizes} =
this.props;
        const size = !!product.size ? product.size.split(' ') : [];
        const location = "/img/"


        return (

            <Card className={"container-fluid"}>
                <CardImg className={"img-fluid"} src={location + img}
                         alt="Card image cap"/>

                <CardBody
                className={"container-fluid justify-content-center"}>
              <CardTitle className={"d-flex justify-content-center"}>
                        {name}
                    </CardTitle>
                    <CardSubtitle
                        className={!!product.discount === true ?
                            "crossed d-flex justify-content-center" :
                            "none d-flex justify-content-center"}>
                        {price} HRK
                    </CardSubtitle>
                    {!!product.discount &&
                   <div className={" d-flex justify-content-center "}>
                        <Badge>{product.discount} %</Badge></div>}
                    {!!product.discount && <CardSubtitle
                <CardSubtitle
                className={"d-flex justify-content-center"}>
              {product.price - (product.price * product.discount / 100)}
                        HRK </CardSubtitle>}
                    <div>
                        <CardText className={"container-fluid"}>

                          <div className={"container-fluid d-flex"}>
```

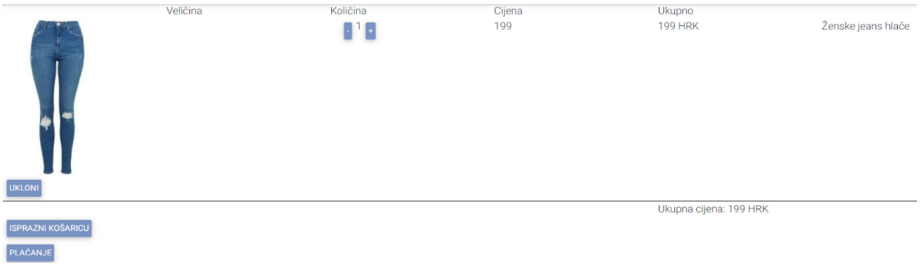
```
                    {
                  <form className={"container-fluid d-flex"}>
                                {
                   !!sizes && sizes.map((char) => !!char &&
                         <div className={"container-fluid"}>
                                      <label>
                                       <input
                                 className={"form-check-input"}
                                       type="radio"
                                       name="react-tips"
                                      value={char}
                                    onClick ={this.onSizeSelect}

                                                  />
                                                  {char}
                                              </label>
                                          </div>)}

                              </form>
                            }


                     </div>

                <div className={"d-flex justify-content-center"}>

                       <Badge>


                      <MDBIcon icon="cart-plus"
                 className={"cursor"} onClick={this.handleClick}/>
                               {
                                   newProduct &&
                       <Badge
                 className={"d-flex justify-content-center badge"}>
                                        New
                       </Badge>}
                               </Badge>
                           </div>

                           <Button
                               className={"d-flex justify-content" +
                               "-center btn-secondary2"}
                               onClick={this.toggle}>
                               Opis
                           </Button>
          <div className={"container-fluid justify-content-center"}>
                          <Collapse
                 className={"container-fluid justify-content-center"}
                                isOpen={this.state.isOpen}>
```

```
                                    {description}
                                </Collapse>
                            </div>
                        </CardText>
                    </div>


                </CardBody>
                {
                    this.state.isAdded &&
                 <Alert className={"d-flex justify-content-center"}
                           color="primary">
                        Proizvod je uspješno <br/>
                        dodan u košaricu!!
                    </Alert>
                }
            </Card>

        )
    }
}

export default Product;
```

Ispis 5: Komponenta „*Product*“

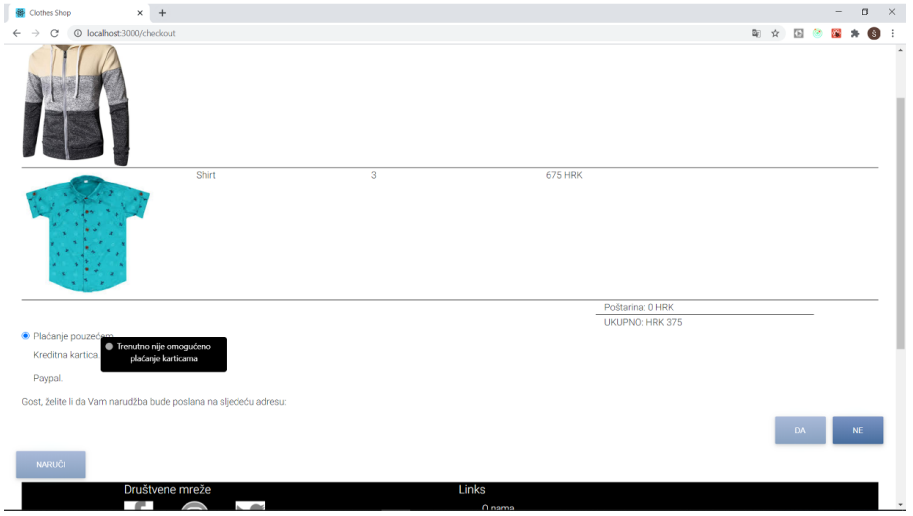
Korisnik u komponenti „*Product*“, čiji se kôd nalazi u ispisu 5, može vidjeti cijenu proizvoda, sliku te ima mogućnost dodavanja proizvoda u košaricu. Također, ima mogućnost odabira veličine odjevnog proizvoda. Isto tako, klikom na gumb imena „OPIS“, otvara se dio komponente u kojemu se vidi opis proizvoda. Nakon što odabere proizvode koje želi, korisnik može otići u košaricu koja se nalazi u zaglavlju stranice.

3.1.3. Košarica

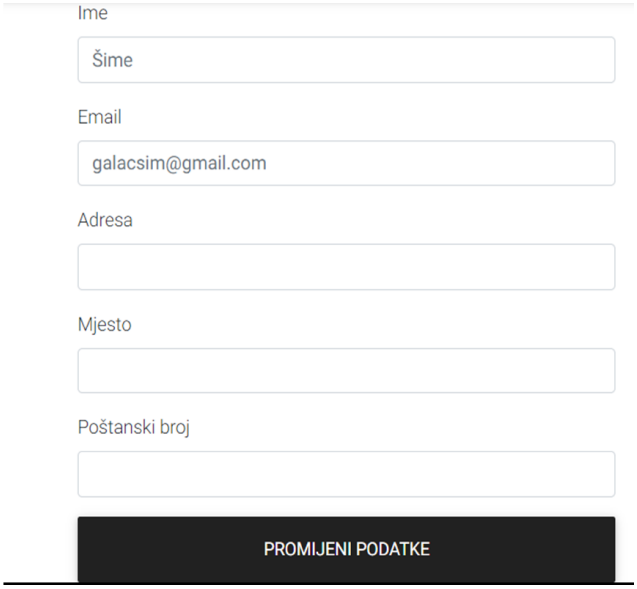


Slika 7: Izgled košarice

Kada se nalazi u komponenti „*Cart*“, korisnik može vidjeti proizvode koje je ranije dodao te ih može ukloniti iz košarice te povećati ili smanjiti količinu dodanih proizvoda, slika 7. Ukoliko je zadovoljan sa svojim izborom korisnik nastavlja na plaćanje ili može isprazniti košaricu ako nije zadovoljan s izabranim. Nakon nastavka, korisniku se prikazuje komponenta „*Checkout*“ gdje potvrđuje svoju adresu, slika 8, koju je unio tijekom registracije ili, ukoliko želi promijeniti adresu, klikom na gumb „NE“, bude preusmjeren u komponentu „*UserInfo*“, slika 9, gdje može mijenjati sve svoje podatke pa tako i adresu dostave.



Slika 8: Dovršavanje narudžbe



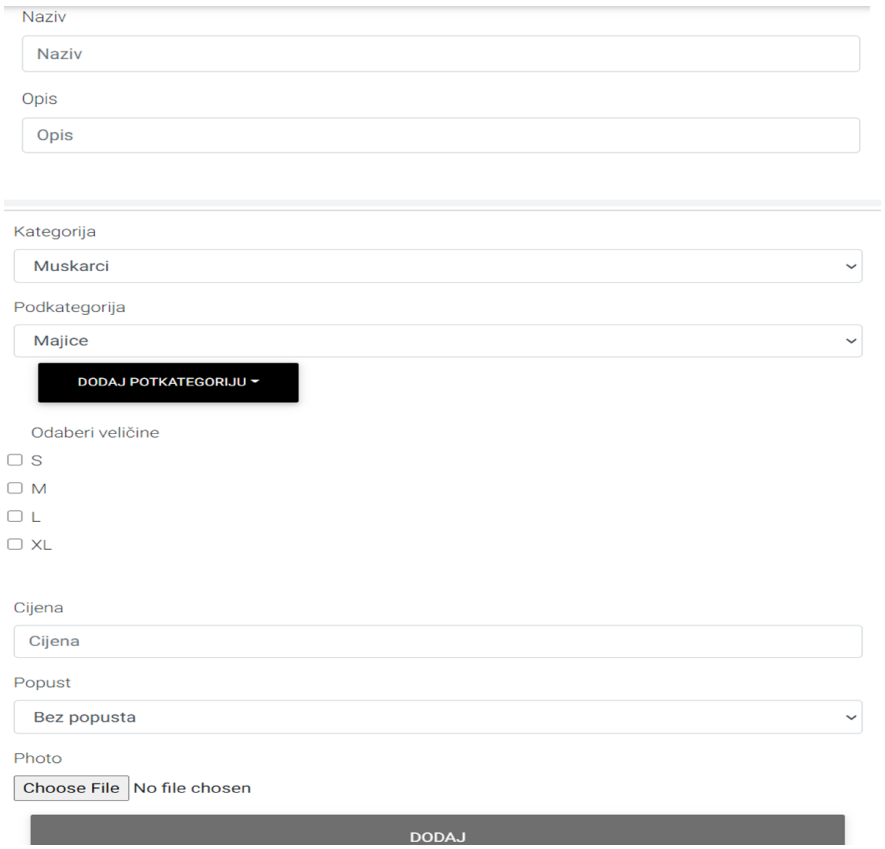
Slika 9: Izgled obrasca za mijenjanje osobnih podataka

Kao što se može vidjeti na slici 8, korisnik ima pristup košarici iako nije prijavljen u aplikaciju ali ne može obaviti narudžbu nego će biti preusmjeren na obrazac za prijavu. Nakon što ispuni sve uvjete, korisnik može naručiti željene proizvode. Prilikom narudžbe korisniku se šalje elektronička pošta u kojoj ga se obavještava da je njegova narudžba zaprimljena te da će biti obaviješten nakon što ona bude potvrđena. Kao što je vidljivo na slici 10, korisnik može u svakom trenutku vidjeti sve svoje narudžbe. Omogućeno mu je da stornira narudžbu ukoliko nije zadovoljan. Storniranje je moguće jedino ako administrator nije potvrdio narudžbu. Korisnik, također, ima mogućnost potvrditi dolazak narudžbe.

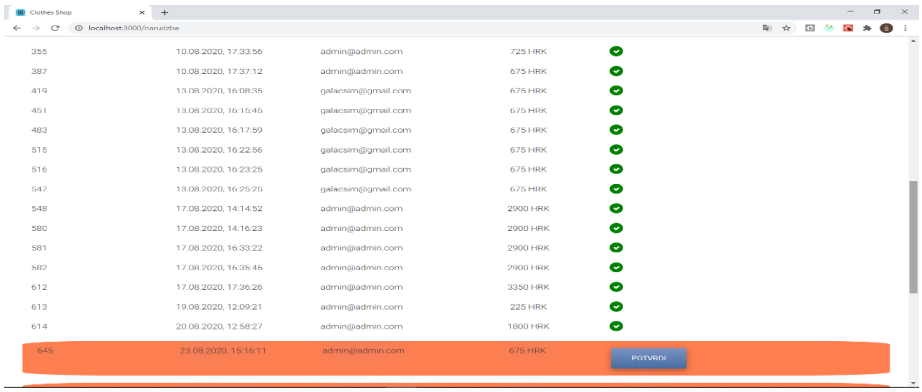
Slika 10: Pregled narudžbi korisnika

3.1.4. Administratorski dio

Administrator ima sve mogućnosti kao i običan korisnik, ali ima i mogućnost ulaska u *„AdminDashboard“*, dio aplikacije gdje može odabrati želi li dodati neki proizvodili izbrisati, , slika 10, te pregledati narudžbe, slika 11. Također ima mogućnost pregleda svih korisnika, što se vidi na slici 12, te klikom može vidjeti korisnikove narudžbe. Također ima mogućnost deaktivacije pojedinog korisnika ukoliko to želi.

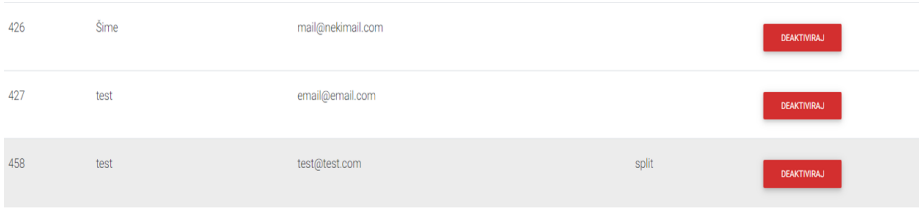


Slika 10: Obrazac za unos novog proizvoda u trgovinu



Slika 11: Pregled svih narudžbi

U obrascu za dodavanje proizvoda administrator upisuje sve podatke o proizvodu, odabire veličine proizvoda te odabire sliku za koju želi da bude prikazana za proizvod koji unosi. Administrator ima mogućnost potvrđivanja narudžbe prilikom čega korisniku dolazi elektronička pošta s detaljima narudžbe. Također, može vidjeti i sve proizvode za pojedinu narudžbu.



Slika 12: Pregled korisnika

3.2. Poslužiteljski dio

3.2.1. Korisnik

Kao što je već rečeno, aplikacija izrađena u *Reactu* je povezana sa *SpringBoot* aplikacijom. Poslužiteljski dio obrađuje sve podatke te ih predaje aplikaciji *React* preko krajnjih točaka. Da bi se korisnik mogao registrirati i prijaviti u aplikaciju, to je potrebno omogućiti na poslužiteljskoj strani aplikacije.

```
@Entity
@Table(name = "USERS", schema = "PUBLIC", uniqueConstraints =
{@UniqueConstraint(columnNames = "email")})
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Email
    @Column(nullable = false)
    private String email;

    @JsonIgnore
    private String password;

    @Column(nullable = false)
    private String name;

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(name = "USER_ROLES", joinColumns = {@JoinColumn(name =
"user_id", referencedColumnName = "id")},
            inverseJoinColumns = {@JoinColumn(name = "role_id",
referencedColumnName = "id")})
    private List<Role> roles;

    @Column
    private String adress;

    @Column
    private String city;

    @Column
    private BigInteger postalCode;
```

Ispis 6: Model korisnika

Kao što se može vidjeti na ispisu 6, model za korisnika na poslužiteljskoj strani je gotovo isti kao i obrazac za registraciju u *React* aplikaciji. Jedino polje koje ne postoji u *React* dijelu je

polje „*id*“ koje se automatski generira u bazi podataka. Prilikom registracije, korisnik je unesen u bazu podataka, a tijekom prijave u aplikaciju se provjerava postoji li uneseni korisnik u bazi te ako postoji, dodjeljuje mu se token za autentifikaciju koji se šalje aplikaciji *React* te ga ona sprema u lokalnu pohranu (eng. *local storage*) preglednika. Važno je napomenuti da se token može spremiti i u kolačić (eng. *cookie*), ali je u ovom projektu spreman u lokalnu pohranu. Prilikom svakog upita prema poslužitelju, kao što je dohvat proizvoda, obavljanje narudžbe ili promjena podataka, *React* aplikacija u zaglavlju zahtjeva šalje token za autentifikaciju te tako poslužitelj zna o kojem se korisniku radi.

3.2.2. Krajnje točke aplikacije

Krajnjim točkama poslužiteljskog dijela upravlja kontroler. U ovom projektu postoji više kontrolera pa tako ima jedan za autentifikaciju korisnika, za administratorski dio aplikacije, za obavljanje narudžbe te za dohvat proizvoda.

```
@PostMapping("/register")
public ResponseEntity<?> registerUser(@Valid @RequestBody
RegisterRequest registerRequest) {

    if
(!registerRequest.getPassword().equals(registerRequest.getRePassword(
))) {
        throw new RuntimeException("Please enter the same
password!");
    }

    Optional<User> optionalUser =
userRepository.findByEmail(registerRequest.getEmail());

    if (optionalUser.isPresent()) {
        User user = optionalUser.get();
        if (registerRequest.getAdministrator())
        {
            if (isRoleExists("ADMIN", user.getRoles()))
            {
                throw new RuntimeException("Email se koristi!!");
            }
            user.getRoles().add(new Role(1L, "ADMIN"));
        } else
            {
```

```
            if (isRoleExists("USER", user.getRoles())) {
                throw new RuntimeException("Email se koristi!!");
            }
            user.getRoles().add(new Role(2L, "USER"));
        }
        userRepository.save(user);
        return ResponseEntity.ok("Registriran korisnik!!");
    }


    User user = new User();
    user.setEmail(registerRequest.getEmail());
    user.setName(registerRequest.getName());

user.setPassword(passwordEncoder.encode(registerRequest.getPassword()
));
    user.setAdress(registerRequest.getAdress());
    user.setCity(registerRequest.getCity());
    user.setPostalCode(registerRequest.getPostalCode());

    List<Role> roles = new ArrayList<>();
    roles.add(new Role(2L, "USER"));
    if (registerRequest.getAdministrator()) {
        roles.add(new Role(1L, "ADMIN"));
    }
    user.setRoles(roles);

    userRepository.save(user);

    return ResponseEntity.ok("Registriran korisnik!!");

}
```

Ispis 7: Kontroler – registracija korisnika

Kao što se vidi na ispisu 7, krajnja točka u kontroleru je obična funkcija koja vraća entitet odgovor (eng. *Response entity*), a prima objekt „*RegisterRequest*“ koji sadržava sve podatke o registraciji. Također, može se vidjeti da se „*RegisterRequest*“ nalazi u tijelu zahtjeva. To pokazuje anotacija „*@RequestBody*“. Nadalje, u funkciji se provjerava postoji li već korisnik s istom adresom elektroničke pošte te se dodaju uloge(eng. *roles*) za korisnika. Uloge su u ovom slučaju: „*ROLE_ADMIN*“ i „*ROLE_USER*“. Ukoliko se želi registrirati korisnik, dobije samo jednu ulogu, onu za korisnika, a ukoliko to želi administrator, dobije obje. Već je spomenut token za autentifikaciju, a u njemu se prilikom prijave enkodiraju uloge koje ima te njegov identifikacijski broj koji se nalazi u bazi podataka. Pomoću toga, poslužiteljska aplikacija uvijek zna tko želi koji resurs te će mu ga dati ukoliko ima pravo na to. U ovom projektu svi

imaju pristup proizvodima iz baze podataka dok sve narudžbe može vidjeti samo administrator, a svoje narudžbe može vidjeti svaki korisnik.

3.2.3. Postavke aplikacije

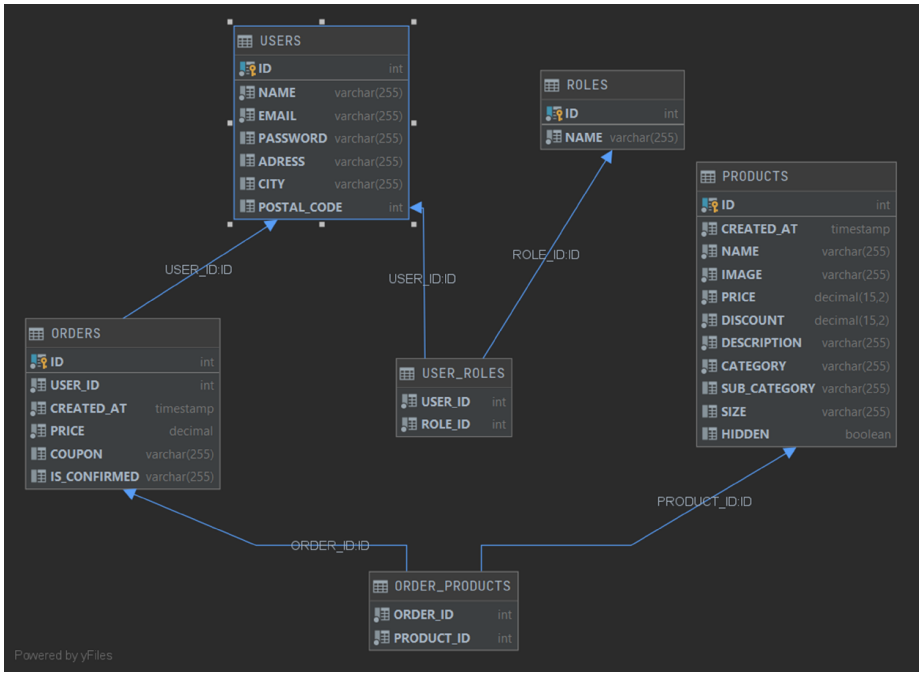
```
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.datasource.url=jdbc:h2:file:~/.db/shop;AUTO_SERVER=TRUE

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect
spring.jpa.show-sql=true
spring.h2.console.enabled=true
spring.h2.console.path=/h2console
spring.h2.console.settings.web-allow-others=true

spring.liquibase.change-log=classpath:/db/master.xml

app.auth.tokenSecret= 84037AJBHFN73485236GFE12
app.auth.tokenexpiration= 864000000

spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=galacsim
spring.mail.password=password



# Other properties
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.connectiontimeout=5000
spring.mail.properties.mail.smtp.timeout=5000
spring.mail.properties.mail.smtp.writetimeout=5000


# TLS , port 587
spring.mail.properties.mail.smtp.starttls.enable=true
```

Ispis 8: Datoteka *application.properties*

Na ispisu 8 se mogu vidjeti postavke baze podataka. Može se vidjeti korisničko ime i lozinka za bazu podataka, koji pokretački program koristi, gdje se u memoriji nalazi. Može se vidjeti gdje se čuvaju zapisi o promjeni baze podataka. Ispod postavki baze podataka se nalaze postavke za autentifikacijski token te postavke za slanje elektroničke pošte. Svaki JWT (eng. *JSON* (*JavaScript Object Notation*) *Web Token*) token, a takav je korišten u ovom projektu, treba tajnu za token kako bi ga poslužitelj mogao enkodirati te po potrebi dekodirati da bi saznao podatke koji su mu potrebni. Nakon toga slijede postavke za slanje elektroničke pošte.

Potrebno je unijeti korisničko ime te lozinku te omogućiti protokol TLS (eng. *Transport Layer Securitiy*) kako bi se elektronička pošta uspješno poslala. Da bi to bilo moguće, potrebno je i omogućiti u postavkama servera za slanje, u ovom slučaju *Google*-a.

3.2.4. Baza podataka



Slika 13: Shema baze podataka

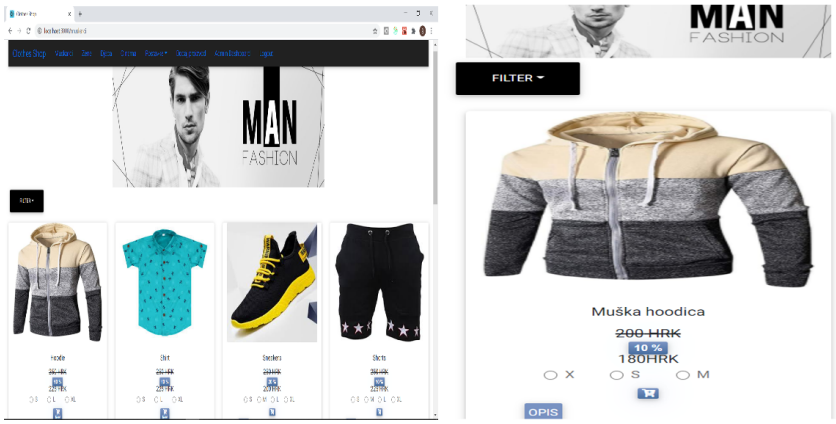
Na slici 13 se nalazi shema baze podataka koja sadržava tablice za korisnika, proizvode, narudžbe, uloge koje korisnik ima te dvije međutablice koje, kao što im imena govore, služe povezivanju dvaju entiteta. Budući da svaka narudžba može sadržavati više proizvoda, a svaki proizvod može biti dio više narudžbi, definira se veza više na više. U tom slučaju je stvorena tablica imena „*Order_Products*“ u kojoj se nalaze podaci koji pokazuju koja narudžba sadrži koji proizvod. Ista situacija je i za tablicu „*User_Roles*“ gdje su povezani korisnici s ulogama

koje mogu imati. U slučaju ove aplikacije to su dvije uloge, ali ih može biti i više pa je ponovno korišten prethodno opisan način rješavanja problema.

3.3. Izgled aplikacije

Za izgled elemenata koji se prikazuju korisniku aplikacije brine se CSS (eng. *Cascading Style Sheets*). Pomoću njega svakom se elementu mogu pridodati CSS klase koje opisuju pojedini element to jest komponentu u slučaju *Reacta*. U ovom projektu su korišteni *Reactstrap* i *Bootstrap*. Zbog razlike u veličini ekran, određeni sadržaj se može na jedan način prikazati na pametnom telefonu, a na drugi način na stolnom računalu. Tako napravljena internetska stranica je responzivna. [15] [16]

Responzivnost je u današnjem razvoju aplikacija vjerojatno najvažniji aspekt o kojem se treba brinuti. Prosječan korisnik se uopće neće zadržavati na internetskoj lokaciji na kojoj mu sadržaj nije lijepo prikazan. Iskustvo korisnika (eng. *UX – user experience*) je najvažnija stvar koju treba pružiti korisniku. Vodeći se time, i ovaj projekt je napravljen responzivno pa samim time izgleda različito na različitim uređajima. U jednom konkretnom primjeru, komponenta „*Product*“ će se na pametnom telefonu prikazivati po jedna u svakom redu, dok će se na računalu prikazivati po četiri u redu što se vidi na slici 14.



Slika 14: Usporedba izgleda proizvoda na velikom i malom zaslonu

Također, navigacija na računalu je odmah vidljiva korisniku, dok na mobitelu korisnik prvo treba stisnuti na „*Menu*“ kako bi mogao vidjeti stavke navigacije.

4. Zaključak

Nakon dovršetka aplikacije, može se zaključiti kako je *React* odlična biblioteka za stvaranje korisničkog sučelja jer daje brojne mogućnosti osobi koja je koristi. Prije samog početka, kada je trebalo odabrati temu, dvojba je bili između razvojnog okruženja *Angular* i *React* biblioteke. Prvi korak pri izradi projekta je bio naučiti koncept R*eact*-a. Nakon toga je sve bilo puno lakše. Sljedeća prepreka je bio *Redux* čiji su koncepti bili teži za shvatiti, ali bez te biblioteke nije bilo moguće izraditi aplikaciju. Ipak, kada se shvati kako upravljati stanjem cijele aplikacije i kako pojedinim komponentama predati podatke koje joj trebaju, izrada aplikacije ide mnogo jednostavnije. Tijekom cijelog vremena izrade aplikacije nametalo se pitanje koji je najbolji razvojni okvir ili biblioteka za stvaranje korisničkog sučelja, no na to pitanje nije lako odgovoriti. Najbitnije je znati ono što treba napraviti te se na temelju toga prilagoditi tehnologije izrade softvera. Za neke stvari je bolja biblioteka *React*, za neke druge stvari je bolji *Angular*, ili *Vue.js* – još jedna biblioteka za razvoj korisničkog sučelja. Za korištenje bilo koje od ove tri tehnologije potrebno je određeno predznanje vezano uz CSS, HTML i programski jezik *JavaScript*. Ujedno, ništa manje važno, odabir Reacta povrh ostalih tehnologija, je bio odličan zbog postojeće velike ponude poslova na tržištu orijentiranih na React tehnologiju.

Literatura

[1] IntelliJ Idea, „Features“, https://www.jetbrains.com/idea/features/ , JetBrains (posjećeno 24.8.2020.)

[2] Spring, „Why Spring“ https://spring.io/why-spring (posjećeno 24.8.2020.)

[3] SpringBoot, „SpringBoot“ https://spring.io/projects/spring-boot (posjećeno 24.8.2020.)

[4] H2Database, „H2 Database Engine“ https://www.h2database.com/html/main.html (posjećeno 24.8.2020)

[5] Brainhub, „ 10 Famous Apps Using ReactJS Nowadays“ https://brainhub.eu/blog/10-famous-apps-using-reactjs-nowadays/#:~:text=React%20allows%20developers%20to%20create,is%20simply%20a%20JavaScript%20runtime , Matt Warcholinski (posjećeno 24.8.2020.)

[6] React, „Why JSX“ https://reactjs.org/docs/introducing-jsx.html , Facebook (posjećeno 24.8.2020.)

[7] React, „Components and props“ https://reactjs.org/docs/components-and-props.html , Facebook (posjećeno 24.8.2020.)

[8] React, „State and lifecycle“ https://reactjs.org/docs/state-and-lifecycle.html , Facebook (posjećeno 24.8.2020)

[9] W3schools, „JavaScript HTML DOM“ https://www.w3schools.com/js/js_htmldom.asp , Refsnes Data (posjećeno 24.8.2020.)

[10] React, „Virtual DOM and Internals“ https://reactjs.org/docs/faq-internals.html#:~:text=The%20virtual%20DOM%20(VDOM)%20is,This%20process%20is%20called%20reconciliation.&text=They%20may%20also%20be%20considered,virtual%20DOM”%20implementation%20in%20React , Facebook (posjećeno 24.8.2020.)

[11] ReactRouter, „Quick Start“ https://reactrouter.com/web/guides/quick-start , ReactTraining (posjećeno 24.8.2020.)

[12] Redux, „Getting Started with Redux“ https://redux.js.org/introduction/getting-started (posjećeno 24.8.2020.)

[13]Bootstrap, „Introduction“ https://getbootstrap.com/docs/4.5/getting-started/introduction/ , Twitter(posjećeno 24.8.2020.)

[14] Reactstrap, „About the Project“ https://reactstrap.github.io/ (posjećeno 24.8.2020.)

[15] Babel, „What is Babel?“ https://babeljs.io/docs/en/index.html (posjećeno 24.8.2020.)

[16] Create-react-app, „Getting started“ https://create-react-app.dev/docs/getting-started/ (posjećeno 24.8.2020.)