Razvoj web aplikacije za online prodaju

Pilepić, Matija

Sveučilište u Rijeci – Odjel za informatiku

Diplomski studij informatike - modul Poslovna informatika

Matija Pilepić

Razvoj web aplikacije za online prodaju

Diplomski rad

Mentorica: doc. dr. sc. Martina Ašenbrener Katić

Rijeka, Prosinac 2021.

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 16.06.2021.

Zadatak za diplomski rad

Pristupnik: Matija Pilepić

Naziv diplomskog rada: Razvoj web aplikacije za online prodaju

Naziv diplomskog rada na eng. jeziku: Development of a web application for online sale

Sadržaj zadatka:

Zadatak diplomskog rada je objasniti postupak planiranja i stvaranja web aplikacije za online prodaju. U radu je potrebno pobliže objasniti korištene tehnologije (Okvir Django REST, CSS okvir Bootstrap, JavaScript okvir React, JavaScript biblioteka Redux i tako dalje).

Neke od glavnih funkcionalnosti koje bi web aplikacija za online prodaju trebala sadržavati su: košarica, ocjenjivanje i komentiranje proizvoda od strane korisnika, "carousel" najbolje ocjenjenih proizvoda, paginacija proizvoda, pretraživanje proizvoda, korisnički profili, admin panel, „checkout" proces (dostava, poštarina, odabir načina plaćanja) i slično.

Mentor:
Doc. dr. sc. Martina Ašenbrener Katić

Voditeljica za diplomske radove:
Izv. prof. dr. sc. Ana Meštrović

Zadatak preuzet: 17.06.2021.

(potpis pristupnika)

Sadržaj

Sažetak

Cilj ovog diplomskog rada je kreirati web aplikaciju za online prodaju. Detaljno ću objasniti postupak planiranja i stvaranja web aplikacije za online prodaju, te objašnjenja objasniti snimkama zaslona koda. Objasniti ću korištene tehnologije koje su korištene za izradu web aplikacije (Okvir Django REST, JavaScript okvir React, Redux, PostgreSQL baza podataka i tako dalje).

Neke od glavnih funkcionalnosti koje će web aplikacija sadržavati su: košarica, ocjenjivanje i komentiranje proizvoda, „carousel“ najbolje ocjenjenih proizvoda, paginaciju proizvoda, korisnički profil, admin panel, „checkout“ proces (dostava, poštarina, odabir načina plaćanja), integracija PayPal plaćanja i slično.

Ključne riječi

Aplikacija, React, JavaScript, Django, Python, Redux, PostgreSQL, AWS, REST, API

1 UVOD

U današnje vrijeme sve je moguće prodati putem interneta. Za svaki proizvod moguće je pronaći kupca. Online kupovina bilježi značajan rast u proteklih 20 godina, a nagli rast zabilježila je početkom COVID pandemije. U ovom radu ću objasniti postupak razvoja aplikacije za online prodaju. Objasnit ću sve korištene tehnologije i pokazati razlog zašto su one upotrijebljene za razvoj aplikacije.

Aplikacija je razvijena kao *'Single-page'* aplikacija, koristeći React JavaScript razvojni okvir i Python Django razvojni okvir. Za bazu podataka koristi *'Amazon Web Services RDS'* servis, a za pohranu statičnih podataka *'Amazon Web Services Simple Storage Serice'*. Neke od funkcionalnosti koje web aplikacija sadrži su:

- Košarica
- Recenzije i ocjene proizvoda
- *'Carousel'* najboljih proizvoda
- Paginaciju proizvoda
- Pretraživanje proizvoda
- Korisnički račun sa svim narudžbama
- Admin panel za upravljanje proizvodima
- Admin panel za upravljanje korisnicima
- Admin panel za upravljanje narudžbama
- *'Checkout'* proces (dostava, poštarina, odabir načina plaćanja, plaćanje)
- Plaćanje PayPalom ili kreditnom karticom.

Django je jedan od najkompletnijih razvojnih okruženja za razvoj web aplikacija. Python omogućuje brz razvoj web aplikacija, a Django se može pobrinuti o svemu, od baze podataka do krajnjeg HTML-a koji korisnik vidi. Međutim, za izradu ove aplikacije Django će služiti kao 'back-end' API koji će posluživati odgovore za '*requestove'* koje aplikacija šalje tokom korištenja. '*Front-end*' je razvijen u React JavaScript razvojnom okruženju. Kako bi izbjegli kreiranje dva servera, posebno za front-end i posebno za back-end, korišten je Django kako bi posluživao statične datoteke.

2 OPIS KORIŠTENIH TEHNOLOGIJA

2.1 *Django REST okvir*

2.1.1 Django okvir

Django je Python *'high-level'* razvojni okvir koji omogućuje brz razvoj sigurnih web stranica koje su jednostavne za održavanje. Razvijen je između 2003. i 2005. godine od strane tima zaduženog za izradu i održavanje web portala. Tim je počeo sve češće ponovno koristiti svoj kod i obrasce dizajna, što je na kraju evoluiralo u razvojni okvir, koji je objavljen kao open-source kod 2005. godine [1].

Django nam pomaže razviti aplikacije koje su [1]:

- Svestrane – Django se koristi za izradu gotovo bilo koje vrste web stranica, od CMS sustava, wikija, društvenih mreža, web portala ili web shopova. Može raditi s bilo kojim okvirom na strani klijenta i može isporučiti sadržaj u gotovo bilo kojem formatu (HTML, JSON, XML, itd.). Također se može proširiti na korištenje drugih komponenti ako je potrebno.
- Sigurne - Django pomaže programerima da izbjegnu mnoge uobičajene sigurnosne pogreške pružajući okvir koji je dizajniran da "radi prave stvari" za automatsku zaštitu web stranice. Na primjer, Django pruža siguran način upravljanja korisničkim računima i lozinkama, izbjegavajući uobičajene pogreške kao što je stavljanje informacija o sesiji u kolačiće gdje su ranjivi (umjesto kolačići sadrže ključ, a stvarni podaci pohranjeni su u bazi podataka) ili izravno pohranjivanje lozinki umjesto hash lozinke.
- Skalabilne - Django koristi arhitekturu koja se temelji na komponentama *'shared-nothing'* (svaki dio arhitekture je neovisan o drugima i stoga se može zamijeniti ili promijeniti ako je potrebno). Jasno razdvajanje između različitih dijelova znači da se može povećati radi povećanja prometa dodavanjem hardvera na bilo kojoj razini: Caching server, server baze podataka ili aplikacijski server. Neke od najprometnijih stranica uspješno su skalirali Django kako bi zadovoljili svoje zahtjeve (npr. Instagram).
- Lako održive - Django kod je napisan korištenjem načela dizajna i obrazaca koji potiču stvaranje koda za održavanje i ponovnu upotrebu. Konkretno, koristi se principom Don't Repeat Yourself (DRY) tako da nema nepotrebnog dupliciranja, smanjujući količinu koda.
- Prenosiv - Django je napisan na Pythonu, koji radi na mnogim platformama. To znači da nije vezan ni za jednu poslužiteljsku platformu i da može pokrenuti svoje aplikacije na mnogim verzijama Linuxa, Windowsa i Mac OS X.

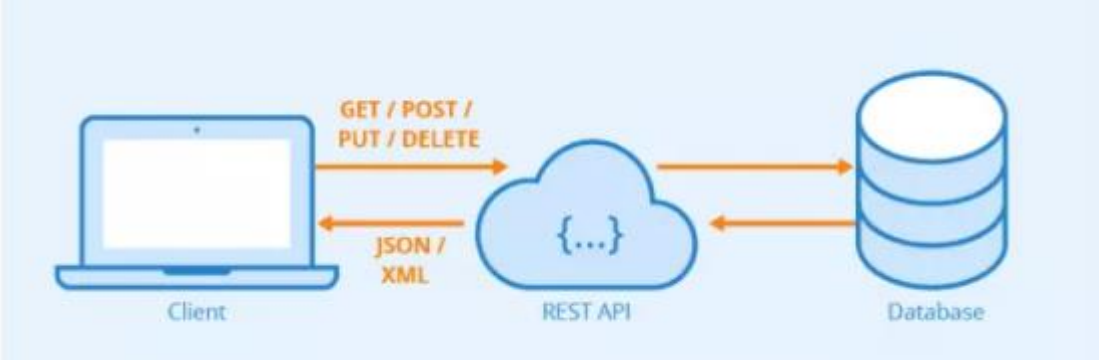
2.1.2 Django REST okvir

Django Rest Framework omogućuje stvaranje RESTful API-ja: način za prijenos informacija između sučelja i baze podataka na jednostavan način. API (Application programming interface) je skup definicija i protokola za izgradnju i integraciju aplikacijskog softvera (slika 1). Ponekad se naziva ugovorom između pružatelja informacija i korisnika informacija – utvrđivanje sadržaja koji se zahtijeva od potrošača (the call) i sadržaja koji zahtijeva proizvođač (the response) [2].

REST je skup arhitektonskih ograničenja, a ne protokol ili standard. API programeri mogu implementirati REST na razne načine.

Kada je zahtjev klijenta napravljen putem RESTful API-ja, on prenosi prikaz stanja resursa podnositelju zahtjeva ili krajnjoj točki. Ove informacije, ili prikaz, isporučuju se u jednom od nekoliko formata putem HTTP-a: JSON (Javascript Object Notation), HTML, XLT, Python, PHP ili običan tekst. JSON je općenito

najpopularniji format datoteke koji se koristi jer je, unatoč svom nazivu, neovisan o jeziku, kao i čitljiv i ljudima i strojevima [4].



Slika 1 Application programming interface

Da bi se API smatrao RESTful, mora biti u skladu s ovim kriterijima [3]:

- Arhitektura klijent-poslužitelj sastavljena od klijenata, poslužitelja i resursa, sa zahtjevima kojima se upravlja putem HTTP-a.
- Komunikacija klijent-poslužitelj bez stanja, što znači da se podaci o klijentu ne pohranjuju između zahtjeva za dobivanje i svaki je zahtjev odvojen i nepovezan.
- Jedinstveno sučelje između komponenti tako da se informacije prenose u standardnom obliku.
- *'Cacheable'* podaci koji pojednostavljuju interakcije klijent-poslužitelj.

2.2 *React Bootstrap okvir*

React je JavaScript biblioteka otvorenog koda za izgradnju korisničkog sučelja. Održava ga Meta (bivši Facebook) i zajednica pojedinačnih programera i tvrtki. React se može koristiti kao baza u razvoju '*single-page*' ili mobilnih aplikacija. Međutim, React se bavi samo upravljanjem stanjem i prikazivanjem tog stanja u DOM-u ('*Document Object Model')*, tako da kreiranje React aplikacija obično zahtijeva korištenje dodatnih biblioteka za '*routing'*, kao i određene funkcionalnosti na strani klijenta. Neke od značajki React okvira su [5]:

- Deklarativan - React se pridržava paradigme deklarativnog programiranja. Programeri dizajniraju poglede za svako stanje aplikacije, a React ažurira i generira komponente kada se podaci promijene.
- Komponente - React kod se sastoji od entiteta koji se nazivaju komponente. Komponente se mogu prikazati određenom elementu u DOM-u pomoću React DOM biblioteke.
- Virtualni DOM - React stvara predmemoriju strukture podataka u memoriji, izračunava nastale razlike, a zatim učinkovito ažurira prikazani DOM preglednika. Taj se proces naziva pomirenje (*'reconciliation')*. To omogućuje programeru da piše kod kao da se cijela stranica prikazuje pri svakoj promjeni, dok React biblioteke prikazuju samo podkomponente koje se stvarno mijenjaju. Ovo selektivno prikazivanje pruža značajno povećanje performansi.

React-Bootstrap je potpuna re-implementacija Bootstrap komponenti koristeći React. Ne ovisi ni o bootstrap.js ni o jQueryju. Metode i događaji koji koriste jQuery obavljaju se imperativno izravnim manipuliranjem DOM-om. Nasuprot tome, React koristi ažuriranja stanja za ažuriranje virtualnog DOM-a. Na taj način React-Bootstrap pruža pouzdanije rješenje ugradnjom Bootstrap funkcionalnosti u Reactov virtualni DOM. Budući da je React-Bootstrap izgrađen s React Javascriptom, stanje (*'state')* se može proslijediti unutar React-Bootstrap komponenti kao '*props*". Također olakšava upravljanje

stanjima jer se ažuriranja izrađuju pomoću Reactovog stanja umjesto izravnog manipuliranja stanjem DOM-a. To također daje veliku fleksibilnost pri stvaranju složenijih komponenti [6].

2.3 *Redux*

Redux je predvidljivi spremnik stanja za JavaScript aplikacije. Pomaže pisati aplikacije koje se ponašaju dosljedno, rade u različitim okruženjima (klijent, poslužitelj i izvorno) i koje je lako testirati. Povrh toga, pruža sjajno razvojno iskustvo, kao što je uređivanje koda uživo u kombinaciji s *'time traveling debugger-om'*. Redux omogućuje upravljanje stanjem aplikacije na jednom mjestu i da promjene u aplikaciji budu predvidljivije i sljedljive. Upravljanje stanjima je u biti način da se olakša komunikacija i dijeljenje podataka između komponenti. Stvara opipljivu strukturu podataka koja predstavlja stanje vaše aplikacije iz koje možete čitati i pisati. Na taj način možete vidjeti inače nevidljiva stanja dok radite s njima.

Iako je React izrađen na način da komponente interno upravljaju svojim stanjima bez potrebe za vanjskim alatom, to dobro funkcionira za aplikacije s nekoliko komponenata. Kako aplikacija postaje veća, upravljanje stanjima koja se dijele među komponentama postaje sve teže [7].

Način na koji Redux radi je jednostavan. Postoji središnja trgovina koja drži cijelo stanje aplikacije. Svaka komponenta može pristupiti pohranjenom stanju bez slanja '*props-a*' s jedne komponente na drugu. Postoje tri glavne značajke Reduxa [8]:

- Actions - Jednostavno rečeno, akcije su događaji. Oni su jedini način na koji možete slati podatke iz svoje aplikacije u Redux trgovinu. Podaci mogu biti iz interakcija korisnika, API poziva ili čak slanja obrazaca.
- Reducers - Reduktori su funkcije koje preuzimaju trenutno stanje aplikacije, izvode radnju i vraćaju novo stanje. Ova stanja se pohranjuju kao objekti i određuju kako se stanje aplikacije mijenja kao odgovor na radnju poslanu u pohranu.
- Store - Trgovina zadržava stanje aplikacije. Vrlo je preporučljivo imati samo jednu trgovinu u bilo kojoj Redux aplikaciji. Možete pristupiti pohranjenom stanju, ažurirati stanje i registrirati ili odjaviti slušatelje putem pomoćnih metoda.

2.4 *Single Page aplikacija*

Single Page Application (SPA) je jedna stranica (otuda i naziv) na kojoj puno informacija ostaje nepromijenjeno, a samo nekoliko dijelova treba ažurirati odjednom. Neki primjeri *Single-Page*

aplikacija su Gmail, Google Maps, AirBNB, Netflix, Pinterest, Paypal i mnogi drugi. Kada otvorimo Gmail, primijetit ćete da tijekom navigacije nema puno promjena - bočna traka i zaglavlje ostaju netaknuti dok prolazite kroz pristiglu poštu. SPA svakim klikom šalje samo ono što trebate, a preglednik prikazuje te informacije. Ovo se razlikuje od tradicionalnog učitavanja stranice gdje poslužitelj ponovno generira cijelu stranicu svakim vašim klikom i šalje je vašem pregledniku. Ova metoda smanjuje vrijeme učitavanja za korisnike i čini količinu informacija koju poslužitelj mora poslati puno manjom i puno isplativijom.

SPA je web aplikacija ili web stranica koja komunicira s korisnikom dinamičkim prepisivanjem trenutne stranice, umjesto učitavanja cijelih novih stranica s poslužitelja. Na većini web stranica ima puno sadržaja koji se ponavlja. Neki od njih ostaju isti bez obzira na to gdje korisnik ide (zaglavlja, podnožja, logotipi, navigacijska traka, itd.), neki od njih su konstantni u samo određenom odjeljku (filterske trake, banneri), a postoji mnogo ponavljajućih izgleda i predložaka (blogovi, samoposluživanje, gore spomenuto postavljanje google pošte). *Single Page* aplikacije iskorištavaju ovo ponavljanje.

Postoje mnoge prednosti SPA rješenja kao što su poboljšane performanse i konzistentnost aplikacije te smanjeno vrijeme razvoja i troškovi infrastrukture. Odvajanjem prezentacijskog sloja od sadržaja i podataka, razvojni timovi mogu raditi različitim brzinama, a da su i dalje integrirani za cjelokupno rješenje. SPA je dobar za izradu responzivnog dizajna za mobitele, stolna računala i tablete. Neke od prednosti *single page* aplikacija su [13]:

- Jednokratno učitavanje HTML, CSS, JS datoteka
- Nema dodatnih upita poslužitelju
- Brz i responzivan frontend
- Poboljšano korisničko iskustvo.

2.5 *Amazon Web Services*

AWS (Amazon Web Services) je sveobuhvatna, razvijajuća platforma za računalstvo u oblaku. AWS usluge mogu ponuditi alate kao što pohrana baze podataka i *'content delivery'* usluge. Za izradu ove web aplikacije korištene su usluge AWS S3 (AWS Simple Storage Solution) i AWS RDS (AWS Relation Database).

2.5.1 AWS Simple Storage Solution

Amazon Simple Storage Service (Amazon S3) je usluga za pohranu objekata koja nudi skalabilnost, dostupnost podataka, sigurnost i performanse. Korisnici svih veličina i industrija mogu pohraniti i zaštititi bilo koju količinu podataka za praktički bilo koji slučaj upotrebe, kao što su podatkovna jezera, aplikacije u oblaku i mobilne aplikacije. Amazon S3 pruža značajke upravljanja tako da možete optimizirati, organizirati i konfigurirati pristup svojim podacima kako biste zadovoljili svoje specifične poslovne i organizacijske zahtjeve [9].

2.5.2 Amazon Relational Database Service (RDS)

Usluga Amazon Relational Database Service (Amazon RDS) olakšava postavljanje, rad i skaliranje relacijske baze podataka u oblaku. Minimalizira potrebe održavanja baze podataka automatizacijom, kreira više instanci za veću dostupnost i lakši '*failover*'. Podržani tipovi baze podataka su PostgreSQL, MySQL, Maria DB, Oracle, SQL Server i Amazon Aurora [10].

Relacijske baze podataka podatke skladište u tablicama, '*rekordima*' i poljima. Relacije su temeljna značajka relacijskih baza. Najčešće se koriste za pohranjivanje transakcijskih i analitičkih podataka iz razloga što pružaju dobru stabilnost i pouzdanost. Ova web aplikacija koristi PostgreSQL bazu podataka.

PostgreSQL je moćan sustav objektno-relacijskih baza podataka otvorenog koda koji koristi i proširuje SQL jezik u kombinaciji s mnogim značajkama koje sigurno pohranjuju i skaliraju najkompliciranija radna opterećenja podataka. PostgreSQL dolazi s mnogim značajkama koje imaju za cilj pomoći programerima u izgradnji aplikacija, administratorima da zaštite integritet podataka i izgrade okruženja otporna na greške, te pomažu u upravljanju podacima bez obzira na to koliko je velik ili mali skup podataka [11].

2.6 *Heroku*

Heroku je *container-based cloud Platform as a Service (PaaS).* PaaS je kategorija usluga računalstva u oblaku koja korisnicima omogućuje pružanje, instaliranje, pokretanje i upravljanje modularnim paketom koji se sastoji od računalne platforme i jedne ili više aplikacija, bez složenosti izgradnje i održavanja infrastrukture koja je obično povezana s razvojem i pokretanjem aplikacije. Heroku se koristi za implementaciju, upravljanje i skaliranje modernih aplikacija. Platforma je fleksibilna i jednostavna za korištenje, te nudi jednostavan put do plasiranja aplikacija na tržište. Heroku je *'fully-managed*' usluga, što znači da se programer ne mora brinuti oko održavanja servera, hardvera ili infrastrukture već se mogu fokusirati na svoj temeljni proizvod [12].

3 Izrada aplikacije

3.1 *Preduvjeti*

Za izradu ove web aplikacije potrebno je imati barem osnovno znanje Django okvira i Python programskog jezika. Treba razumjeti i API-je, te znati kako raditi s njima. HTML i CSS su temeljni jezici za kreiranje i uređivanje, te se njihovo znanje podrazumijeva. Koristit ćemo JavaScript za rad s Reactom i API pozive, te je potrebno osnovno znanje i razumijevanje istog. Također, koristiti ćemo Bootstrap te se očekuje osnovno poznavanje njegovih koncepata.

Tehnologija koja je korištena za izradu web aplikacije je React i Django, za *state managmnet* korišten je Redux. Za postavljanje na server korišten je Heroku, za bazu podataka korišten je PostgreSQL koji ćemo postaviti na Amazon Web Services (u nastavku AWS). Za plaćanja je integriran PayPal API.

U svome radnome okruženju potrebno je instalirati Node.js i Python. Kao tekst editor korišten je Virtual Studio Code s dodatnim ekstenzijama radi lakšeg i bržeg programiranja:

- ES7 React Native snippets za brže i lakše dodavanja *snippeta* koda
- Bracket Pair Colorizer za lakše snalaženje u kodu
- Auto Rename Tag za lakše izmjene u HTML tagovima
- Prettier za popravljanje indentacije.

3.2 *Početak Front Enda*

3.2.1 Kreiranje React aplikacije

Projekt započinjemo kreiranjem front enda s Reactom, a nakon toga ćemo prijeći na back end. Počinjemo s kreiranjem React aplikacije naredbom „npx create-react-app my-app“ u command promptu. Nakon što kreiramo aplikaciju, možemo ju pokrenuti s naredbom „npm start“ (slika 2).

```
C:\Users\mpilepic\OneDrive - iOLAP, Inc\Desktop\Webshop>npx create-react-app frontend
npx: installed 67 in 13.77s

Creating a new React app in C:\Users\mpilepic\OneDrive - iOLAP, Inc\Desktop\Webshop\frontend.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...
```

Slika 2 Kreiranje React aplikacije

Kreiranu aplikaciju otvorimo u Visual Studio Code-u i spremi smo za nastavak razvoja frontenda. Prvo što ćemo napraviti je promijeniti naslov stranice i *favicon* unutar 'frontend/public' direktorija i obrisati React aplikaciju unutar *App.js* datoteke.

3.2.2 Header i Footer komponente

Krećemo s izradom komponenata za web aplikaciju, prvo ćemo napraviti direktorij 'components' unutar 'src' direktorija. Unutar 'components' direktorija, kreiramo dvije nove datoteke: 'Footer.js' i 'Header.js'. Komponente također registriramo unutar 'App.js' datoteke kako bi ih aplikacija prikazala. Na slici 3 prikazan je kod za izradu 'header' komponente.

```
22  function App() {
23    return (
24      <Router>
25        <Header />
26        
27        <Footer />
28      </Router>
29    );
30  }
```

Slika 3 Kod Headera i Footera unutar App.js

Za stil stranice koristiti ćemo React Bootstrap, kojeg moramo instalirati naredbom „npm install react-bootstrap“. Također, dodati ćemo 'bootstrap.min.css' datoteku koja sadrži temu aplikacije unutar 'src' direktorija te ju *importat* unutar 'index.js' datoteke (slika 4).

App.js | index.js

frontend > src > index.js

```
1   import React from 'react';
2   import ReactDOM from 'react-dom';
3   import { Provider } from 'react-redux'
4   import store from './store'
5   import './index.css';
6   import './bootstrap.min.css'
7   import App from './App';
8   import reportWebVitals from './reportWebVitals';
9
10  ReactDOM.render(
11    <Provider store={store}>
12      <App />
13    </Provider>,
14    document.getElementById('root')
15  );
```

Slika 4 Kod Index.js datoteke

Za kreiranje Header komponente koristiti ćemo Bootstrap komponente, što će olakšati i ubrzati razvoj aplikacije. Komponente koje su korištene za izradu Header komponente su Navbar, Nav, Container, Row, NavDropdown prikazano na slici 5.

```
<header>
    <Navbar bg="dark" variant="dark"  expand="lg" collapseOnSelect >
        <Container>
            <LinkContainer to='/'>
                <Navbar.Brand> Budi-Moderan.hr </Navbar.Brand>
            </LinkContainer>

            <Navbar.Toggle aria-controls="basic-navbar-nav" />
            <Navbar.Collapse id="basic-navbar-nav">
                <SearchBox />
                <Nav className="ml-auto">
                    <LinkContainer to='/cart'>
                        <Nav.Link ><i className="fas fa-shopping-cart"> </i> Košarica</Nav.Link>
                    </LinkContainer>
                    {userInfo ? (
                        <NavDropdown title={userInfo.name} id='username'>
                            <LinkContainer to='/profile'>
                                <NavDropdown.Item>Profil</NavDropdown.Item>
                            </LinkContainer>

                            <NavDropdown.Item onClick={logoutHandler}>Odjavi se</NavDropdown.Item>
                        </NavDropdown>
                    ) : (
                            <LinkContainer to='/login'>
                                <Nav.Link><i className="fas fa-user"></i>Prijavi se</Nav.Link>
                            </LinkContainer>
                        )}
                    {userInfo && userInfo.isAdmin && (
                        <NavDropdown title='Admin' id='adminmenue'>
                            <LinkContainer to='/admin/userlist'>
                                <NavDropdown.Item>Korisnici</NavDropdown.Item>
                            </LinkContainer>

                            <LinkContainer to='/admin/productlist'>
                                <NavDropdown.Item>Proizvodi</NavDropdown.Item>
                            </LinkContainer>

                            <LinkContainer to='/admin/orderlist'>
                                <NavDropdown.Item>Narudžbe</NavDropdown.Item>
                            </LinkContainer>

                        </NavDropdown>
                    )}
                </Nav>
            </Navbar.Collapse>
        </Container>
    </Navbar>
</header>
```

Slika 5 Kod Header komponente

Footer komponenta ostaje jednostavna, dodajemo samo dvije linije teksta (slika 6).

```
<footer>
    <Container>
        <Row>
            <Col className="text-center py-3">Copyright &copy; Matija Pilepic</Col>
        </Row>
        <Row>
            <Col className="text-center py-3">Sveučilište u Rijeci, Odjel za informatiku</Col>
        </Row>
    </Container>
</footer>
```

Slika 6 Kod Footer komponente

3.2.3 Početni zaslon

Home Screen je ekran, ne komponenta te zbog toga kreiramo novi direktorij unutar 'src' direktorija, nazovimo ga 'screens'. Napravimo datoteku 'HomeScreen.js'. HomeScreen također moramo registrirati unutar 'App.js' datoteke na isti način kao i prethodne komponente. Unutar HomeScreena prikazati ćemo *Carousel* najprodavanijih proizvoda i sve dostupne proizvode. Sve to napravljeno je API pozivima prema back-endu. Komponente koje se koriste unutar 'HomeScreen.js' datoteke još nisu kreirane, a izgled datoteke prikazan je na slici 7.

```jsx
import { useDispatch, useSelector } from 'react-redux'
import { Row, Col } from 'react-bootstrap'
import Product from '../components/Product'
import Loader from '../components/Loader'
import Message from '../components/Message'
import Paginate from '../components/Paginate'
import ProductCarousel from '../components/ProductCarousel'
import { listProducts } from '../actions/productActions'


function HomeScreen({ history }) {
    const dispatch = useDispatch()
    const productList = useSelector(state => state.productList)
    const { error, loading, products, page, pages } = productList

    let keyword = history.location.search

    useEffect(() => {
        dispatch(listProducts(keyword))

    }, [dispatch, keyword])

    return (
        <div>
            {!keyword && <ProductCarousel />}

            <h1>Najnoviji proizvodi</h1>
            {loading ? <Loader />
                : error ? <Message variant='danger'>{error}</Message>
                    :
                    <div>
                        <Row>
                            {products.map(product => (
                                <Col key={product._id} sm={12} md={6} lg={4} xl={3}>
                                    <Product product={product} />
                                </Col>
                            ))}
                        </Row>
                        <Paginate page={page} pages={pages} keyword={keyword} />
                    </div>
            }
        </div>
    )
}

export default HomeScreen
```

Slika 7 Kod HomeScreen.js datoteke

Sljedeće kreiramo 'Product.js' datoteku unutar 'Components' direktorija, te krećemo s izradom 'Product' komponente koja je korištena za prikaz proizvoda u HomeScreen zaslonu. Unutar komponente prosljeđujemo informacije koje želimo prikazati, kao što su slika, ime, ocjena i broj recenzija proizvoda (slika 8).

```jsx
import React from 'react'
import { Card } from 'react-bootstrap'
import Rating from './Rating'
import { Link } from 'react-router-dom'

function Product({ product }) {
    return (
        <Card style={{ height: 'auto' }} className="my-3 p-3 rounded">
            <Link to={`/product/${product._id}`}>
                <Card.Img  src={product.image} />
            </Link>

            <Card.Body>
                <Link to={`/product/${product._id}`}>
                    <Card.Title style={{ height: '3rem' }} as="div">
                        <strong>{product.name}</strong>
                    </Card.Title>
                </Link>

                <Card.Text as="div">
                    <div className="my-3">
                        <Rating value={product.rating} text={`${product.numReviews} recenzija`} color={'#f8e825'} />
                    </div>
                </Card.Text>


                <Card.Text as="h3">
                    {product.price} kn
                </Card.Text>
            </Card.Body>
        </Card>
    )
}

export default Product
```

Slika 8 Kod Product.js datoteke

Nastavljamo kreiranje 'Rating.js' komponente. Komponenta je korištena unutar 'Product.js' komponente, ali ima svoju zasebnu logiku te sam ju zbog toga odlučio izdvojiti. U komponentu šaljemo 3 parametra: vrijednost (ocjena), tekst i boju. Komponenta za svaki cijeli broj ocjene dodaje jednu zvjezdicu, a za 0.5 dodaje pola zvjezdice u zadanoj boji. Pored zvjezdica će ispisati tekst koji smo proslijedili (slika 9. i slika 10).

```jsx
import React from 'react'

function Rating({ value, text, color }) {
    return (
        <div className="rating">
            <span>
                <i style={{ color }} className={
                    value >= 1
                        ? 'fas fa-star'
                        : value >= 0.5
                            ? 'fas fa-star-half-alt'
                            : 'far fa-star'
                }>
                </i>
            </span>
            <span>
                <i style={{ color }} className={
                    value >= 2
                        ? 'fas fa-star'
                        : value >= 1.5
                            ? 'fas fa-star-half-alt'
                            : 'far fa-star'
                }>
                </i>
            </span>
            <span>
                <i style={{ color }} className={
                    value >= 3
                        ? 'fas fa-star'
                        : value >= 2.5
                            ? 'fas fa-star-half-alt'
                            : 'far fa-star'
                }>

                </i>
            </span>
            <span>
                <i style={{ color }} className={
                    value >= 4
                        ? 'fas fa-star'
                        : value >= 3.5
                            ? 'fas fa-star-half-alt'
                            : 'far fa-star'
                }>

                </i>
            </span>
```

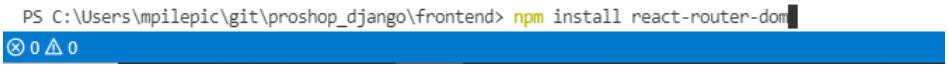
Slika 9 Kod Rating komponenta (1/2)

```jsx
            <span>
                <i style={{ color }} className={
                    value >= 5
                        ? 'fas fa-star'
                        : value >= 4.5
                            ? 'fas fa-star-half-alt'
                            : 'far fa-star'
                }>

                </i>
            </span>

            <span>{text && text}</span>
        </div>
    )
}
export default Rating
```

Slika 10 Kod Rating komponenta (2/2)

3.2.4 Implementacija React Router-a

Implementaciju React Routera započinjemo dodavanjem Routera u aplikaciju. Pozicioniramo se unutar frontend direktorija projekta te pokrećemo instalaciju (slika 11). Na slici 12 prikazano je kako ProductScreen komponenti unutar 'path' parametra prosljeđujemo parametar 'id', kako bi znao koji proizvod mora prikazati.



Slika 11 React Router DOM instalacija

Router ćemo koristiti kako bi cijelu aplikaciju '*zamotali'* unutar *'Hash Router-a'*. Unutar 'App.js' filea, prvo dodajemo potrebne importove, a zatim cijelu aplikaciju stavljamo u HashRouter u znaku (slika 12). Time omogućujemo da komponente ne navodimo direktno unutar aplikacije, već navodimo 'Route' kojem prosljeđujemo komponentu i putanju na kojoj ju treba izvršiti.

```
function App() {
  return (
    <Router>
      <Header />
      <main className="py-3">
        <Container>
          <Route path='/' component={HomeScreen} exact />
          <Route path='/login' component={LoginScreen} />
          <Route path='/register' component={RegisterScreen} />
          <Route path='/profile' component={ProfileScreen} />
          <Route path='/shipping' component={ShippingScreen} />
          <Route path='/placeorder' component={PlaceOrderScreen} />
          <Route path='/order/:id' component={OrderScreen} />
          <Route path='/payment' component={PaymentScreen} />
          <Route path='/product/:id' component={ProductScreen} />
          <Route path='/cart/:id?' component={CartScreen} />

          <Route path='/admin/userlist' component={UserListScreen} />
          <Route path='/admin/user/:id/edit' component={UserEditScreen} />

          <Route path='/admin/productlist' component={ProductListScreen} />
          <Route path='/admin/product/:id/edit' component={ProductEditScreen} />

          <Route path='/admin/orderlist' component={OrderListScreen} />
        </Container>
      </main>
      <Footer />
    </Router>
  );
}

export default App;
```

Slika 12 Kod App.js datoteke

3.2.5 Ekran proizvda

Izradu ekrana za prikaz proizvoda započinjemo kreiranjem datoteke 'ProductScreen.js' unutar 'screens' direktorija. Potrebno je *imporatiti* sve potrebne komponente i biblioteke (slika 13).

```
import React, { useState, useEffect } from 'react'
import { useDispatch, useSelector } from 'react-redux'
import { Link } from 'react-router-dom'
import { Row, Col, Image, ListGroup, Button, Card, Form } from 'react-bootstrap'
import Rating from '../components/Rating'
import Loader from '../components/Loader'
import Message from '../components/Message'
import { listProductDetails, createProductReview } from '../actions/productActions'
import { PRODUCT_CREATE_REVIEW_RESET } from '../constants/productConstants'
```

Slika 13 Kod ProductScreen.js datoteke (1/5)

Na ekranu za prikaz proizvoda potrebno je prikazati sliku proizvoda, njegov naziv, ocjenu, broj recenzija, cijenu, opis, komentare i gumb za dodavanje u košaricu. Stranica je konstruirana kao tablica, podijeljena u redove i stupce u koje su postavljene komponente, slike ili samo tekst kao što je prikazano slikama 14, 15, 16 i 17.

```
<div>
    <Link to='/' className='btn btn-light my-3'>Natrag</Link>
    {loading ?
        <Loader />
        : error
            ? <Message variant='danger'>{error}</Message>
            : (
                <div>
                    <Row>
                        <Col md={6}>
                            <Image src={product.image} alt={product.name} fluid />
                        </Col>


                        <Col md={3}>
                            <ListGroup variant="flush">
                                <ListGroup.Item>
                                    <h3>{product.name}</h3>
                                </ListGroup.Item>

                                <ListGroup.Item>
                                    <Rating value={product.rating} text={`${product.numReviews} recenzija`} color={'#f8e825'} />
                                </ListGroup.Item>

                                <ListGroup.Item>
                                    Cijena: {product.price} kn
                                </ListGroup.Item>

                                <ListGroup.Item>
                                    Opis: {product.description}
                                </ListGroup.Item>
                            </ListGroup>
                        </Col>
```

Slika 14 Kod ProductScreen.js datoteke(2/5)

```
<Col md={3}>
    <Card>
        <ListGroup variant='flush'>
            <ListGroup.Item>
                <Row>
                    <Col>Cijena:</Col>
                    <Col>
                        <strong>{product.price} kn</strong>
                    </Col>
                </Row>
            </ListGroup.Item>
            <ListGroup.Item>
                <Row>
                    <Col>Status:</Col>
                    <Col>
                        {product.countInStock > 0 ? 'Na zalihama' : 'Nema na zalihama'}
                    </Col>
                </Row>
            </ListGroup.Item>

            {product.countInStock > 0 && (
                <ListGroup.Item>
                    <Row>
                        <Col>Količina:</Col>
                        <Col xs='auto' className='my-1'>
                            <Form.Control
                                as="select"
                                value={qty}
                                onChange={(e) => setQty(e.target.value)}
                            >
                                {

                                    [...Array(product.countInStock).keys()].map((x) => (
                                        <option key={x + 1} value={x + 1}>
                                            {x + 1}
                                        </option>
                                    ))
                                }

                            </Form.Control>
                        </Col>
                    </Row>
                </ListGroup.Item>
            )}
```

Slika 15 Kod ProductScreen.js datoteke(3/5)

```
                    <ListGroup.Item>
                        <Button
                            onClick={addToCartHandler}
                            className='btn-block'
                            disabled={product.countInStock == 0}
                            type='button'>
                            Dodaj u košaricu
                        </Button>
                    </ListGroup.Item>
                </ListGroup>
            </Card>
        </Col>
    </Row>

    <Row>
        <Col md={6}>
            <h4>Recenzije</h4>
            {product.reviews.length === 0 && <Message variant='info'>Nema recenzija</Message>}

            <ListGroup variant='flush'>
                {product.reviews.map((review) => (
                    <ListGroup.Item key={review._id}>
                        <strong>{review.name}</strong>
                        <Rating value={review.rating} color='#f8e825' />
                        <p>{review.createdAt.substring(0, 10)}</p>
                        <p>{review.comment}</p>
                    </ListGroup.Item>
                ))}
```

Slika 16 Kod ProductScreen.js datoteke (4/5)

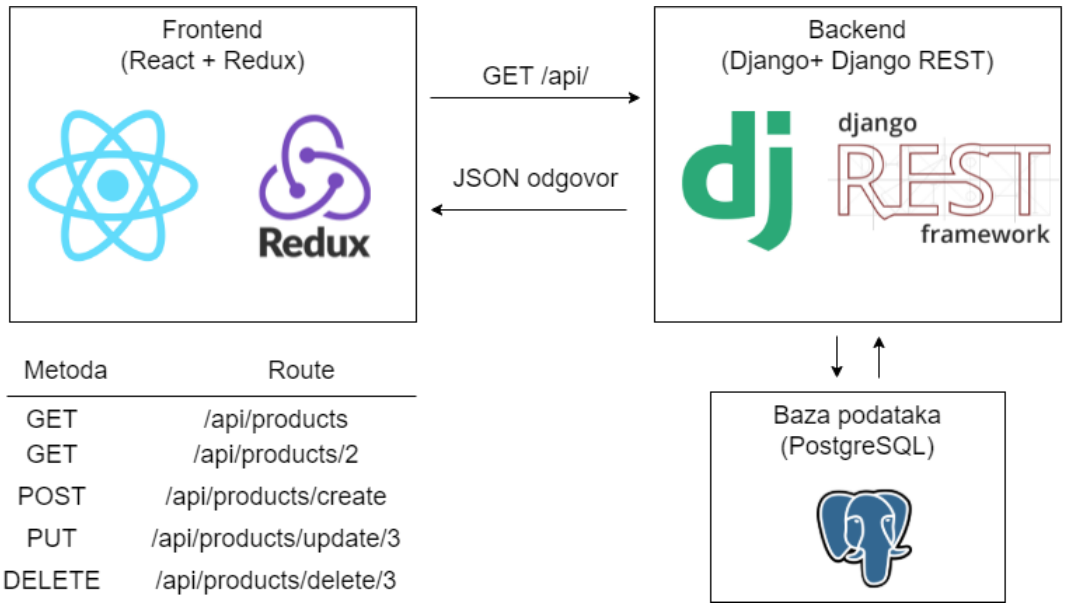
```
        <ListGroup.Item>
            <h4>Napišite recenziju</h4>

            {loadingProductReview && <Loader />}
            {successProductReview && <Message variant='success'>Recenzija poslana</Message>}
            {errorProductReview && <Message variant='danger'>{errorProductReview}</Message>}

            {userInfo ? (
                <Form onSubmit={submitHandler}>
                    <Form.Group controlId='rating'>
                        <Form.Label>Ocijena</Form.Label>
                        <Form.Control
                            as='select'
                            value={rating}
                            onChange={(e) => setRating(e.target.value)}
                        >
                            <option value=''>Izaberite...</option>
                            <option value='1'>1 - Jako loše</option>
                            <option value='2'>2 - Loše</option>
                            <option value='3'>3 - Dobro</option>
                            <option value='4'>4 - Jako dobro</option>
                            <option value='5'>5 - Odlično</option>
                        </Form.Control>
                    </Form.Group>

                    <Form.Group controlId='comment'>
                        <Form.Label>Recenzija</Form.Label>
                        <Form.Control
                            as='textarea'
                            row='5'
                            value={comment}
                            onChange={(e) => setComment(e.target.value)}
                        ></Form.Control>
                    </Form.Group>

                    <Button
                        disabled={loadingProductReview}
                        type='submit'
                        variant='primary'
                    >
                        Pošalji
                    </Button>

                </Form>
            ) : (
                <Message variant='info'>Molimo vas da se <Link to='/login'>prijavite</Link> za pisanje recenzije.</Message
                )}
        </ListGroup.Item>
    </ListGroup>
```

Slika 17 Kod ProductScreen.js datoteke (5/5)

3.3 *Kreiranje back-enda*

3.3.1 Frontend-Backend komunikacija

Na slici 18 prikazan je tok podataka kroz aplikaciju. Frontend šalje zahtjev s jednom od mogućih metoda Djangu u backendu, koji na temelju istoga generira pregled te uzima podatke iz baze podataka. Podatke zatim serializira u JSON (Java Script Object Notation) format, te ih vraća frontendu koji ih prikazuje krajnjem korisniku.



Metoda	Route
GET	/api/products
GET	/api/products/2
POST	/api/products/create
PUT	/api/products/update/3
DELETE	/api/products/delete/3

Slika 18 Frontend-backend tok podataka

Prije kreiranja backend, potrebno je kreirati virtualni environment, kako bi sve potrebne biblioteke bile instalirane unutar njega. Environment kreiramo naredbom „virtualenv env“, a aktiviramo ga naredbom „env/Scripts/activate“. Sada smo spremni za instaliranje biblioteka te ćemo prvo instalirati django s naredbom „pip install django“. Kada je Django instaliran, kreiramo backend naredbom „django-admin startproject backend“. Naredbom „python manage.py startapp base“ kreiramo aplikaciju unutar projekta. Kreiranu aplikaciju moramo registrirati unutar 'settings.py' datoteke, dodavanjem iste unutar 'INSTALLED_APPS' variable.

3.3.2 Slanje i dohvaćanje podataka

Za komunikaciju Djanga i baze podataka, unutar 'views.py' datoteke dodajemo funkcije za dohvaćanje, uređivanje ili kreiranje podataka u bazi (slika 19).

```python
@api_view(['GET'])
def getProduct(request, pk):
    product = Product.objects.get(_id=pk)
    serializer = ProductSerializer(product, many=False)
    return Response(serializer.data)


@api_view(['POST'])
@permission_classes([IsAdminUser])
def createProduct(request):
    user = request.user

    product = Product.objects.create(
        user=user,
        name='Ime Proizvoda',
        price=0,
        brand='Brand',
        countInStock=0,
        category='Kategorija',
        description=''
    )

    serializer = ProductSerializer(product, many=False)
    return Response(serializer.data)
```

Slika 19 Kod Views.py datoteke

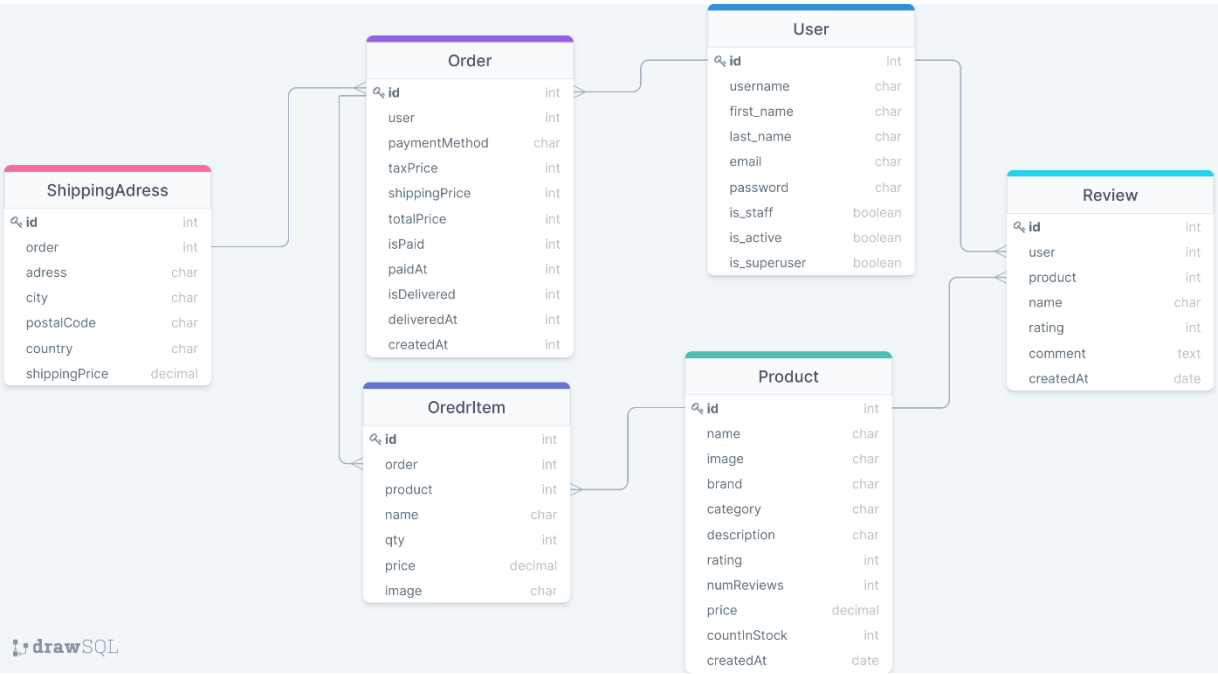
Kako bi te podatke mogao koristiti React u frontendu, oni moraju biti u JSON formatu. To ćemo postići serializacijom podataka. Kreiramo datoteku 'serializers.py' unutar 'backend' direktorija, te u njoj definiramo svoje serializere kako je prikazano na slici 20.

```python
class UserSerializer(serializers.ModelSerializer):
    name = serializers.SerializerMethodField(read_only=True)
    _id = serializers.SerializerMethodField(read_only=True)
    isAdmin = serializers.SerializerMethodField(read_only=True)

    class Meta:
        model = User
        fields = ['id', '_id', 'username', 'email', 'name', 'isAdmin']

    def get__id(self, obj):
        return obj.id

    def get_isAdmin(self, obj):
        return obj.is_staff

    def get_name(self, obj):
        name = obj.first_name
        if name == '':
            name = obj.email

        return name
```

Slika 20Kod korisničkih serializera

3.3.3 Modeliranje podataka

Nakon što je napravljen osnovni dizajn frontenda, spremni smo napraviti bazu podataka, napraviti tablice i odrediti relacije između tablica. Kao bazu podataka korištena je PostgreSQL baza. Kako bi krenuli u izradu baze podataka, prvo moramo napraviti model podataka. Na slici 21 prikazan je model podataka za izradu aplikacije.



Slika 21 Model podataka

Kada imamo model podataka, možemo raditi Django modele unutar 'models.py' datoteke. Model 'User' ne moramo kreirati jer Django pruža *Defaultni* model. Prvo moramo *imporatiti* Django modele u 'models.py' datoteku (slika 22).

```
from django.db import models
from django.contrib.auth.models import User
```

Slika 22 Kod importa Django modela

Nakon toga možemo početi graditi svoje modele. U tablici *Product* za polje *image* koristimo *ImageField* tip podatka. Kako bi njega mogli koristiti, u svoje virtualno okruženje moramo dodati proširenje *Pillow* naredbom „pip install pillow“. Na slici 23 prikazan je kod modela proizvoda.

```python
class Product(models.Model):
    user = models.ForeignKey(User, on_delete=models.SET_NULL, null=True)
    name = models.CharField(max_length=200, null=True, blank=True)
    image = models.ImageField(null=True, blank=True,
                              default='/placeholder.png')
    brand = models.CharField(max_length=200, null=True, blank=True)
    category = models.CharField(max_length=200, null=True, blank=True)
    description = models.TextField(null=True, blank=True)
    rating = models.DecimalField(
        max_digits=7, decimal_places=2, null=True, blank=True)
    numReviews = models.IntegerField(null=True, blank=True, default=0)
    price = models.DecimalField(
        max_digits=7, decimal_places=2, null=True, blank=True)
    countInStock = models.IntegerField(null=True, blank=True, default=0)
    createdAt = models.DateTimeField(auto_now_add=True)
    _id = models.AutoField(primary_key=True, editable=False)

    def __str__(self):
        return self.name


class Review(models.Model):
    product = models.ForeignKey(Product, on_delete=models.SET_NULL, null=True)
    user = models.ForeignKey(User, on_delete=models.SET_NULL, null=True)
    name = models.CharField(max_length=200, null=True, blank=True)
    rating = models.IntegerField(null=True, blank=True, default=0)
    comment = models.TextField(null=True, blank=True)
    createdAt = models.DateTimeField(auto_now_add=True)
    _id = models.AutoField(primary_key=True, editable=False)

    def __str__(self):
        return str(self.rating)


class Order(models.Model):
    user = models.ForeignKey(User, on_delete=models.SET_NULL, null=True)
    paymentMethod = models.CharField(max_length=200, null=True, blank=True)
    taxPrice = models.DecimalField(
        max_digits=7, decimal_places=2, null=True, blank=True)
```

Slika 23 Kod modela proizvoda

3.4 *Implementacija Reduxa za upravljanje stanjima*

3.4.1 Kreiranje Redux Store-a

Prije kreacije samog Redux Store-a moramo instalirati Redux unutar naše aplikacije. Redux nije specifičan za React, zato moramo instalirati i react-redux i redux-thunk koji nam omogućuje asinkrone pozive. Pozicionirani u 'frontend' direktoriju svoga projekta, izvršimo naredbu: „npm install redux react-redux redux-thunk“.

Za kreiranje 'store-a' potrebno je kreirati datoteku 'store.js' unutar 'src' direktorija. *Importamo* sva proširenja koja smo prethodno instalirali i zatim možemo kreirati 'store' (slika 24). Nakon što ga kreiramo, moramo ga dodati i u 'index.js' datoteku.

```
const initialState = {
    cart: {
        cartItems: cartItemsFromStorage,
        shippingAddress: shippingAddressFromStorage,
    },
    userLogin: { userInfo: userInfoFromStorage },
}

const middleware = [thunk]

const store = createStore(reducer, initialState,
    composeWithDevTools(applyMiddleware(...middleware)))

export default store
```

Slika 24 Kod Store.js datoteke

3.4.2 Product akcije i reduktori

Unutar 'src' datoteke kreiramo direktorije 'actions' i 'reducers'. Reduktor je funkcija koja uzima naše sadašnje stanje, i napravi s njime ono što mu kažemo da napravi. U suštini reduktori ažuriraju 'store'. Unutar direktorija 'reducers' kreiramo datoteku 'productReducers.js'. Ova datoteka sadrži više reduktora za ažuriranje više stanja unutar 'storea'. Na slici 25 prikazan je primjer jednog od reduktora.

```
export const productListReducer = (state = { products: [] }, action) => {
    switch (action.type) {
        case PRODUCT_LIST_REQUEST:
            return { loading: true, products: [] }

        case PRODUCT_LIST_SUCCESS:
            return {
                loading: false,
                products: action.payload.products,
                page: action.payload.page,
                pages: action.payload.pages
            }

        case PRODUCT_LIST_FAIL:
            return { loading: false, error: action.payload }

        default:
            return state
    }
}


export const productDetailsReducer = (state = { product: { reviews: [] } }, action) => {
    switch (action.type) {
        case PRODUCT_DETAILS_REQUEST:
            return { loading: true, ...state }

        case PRODUCT_DETAILS_SUCCESS:
            return { loading: false, product: action.payload }

        case PRODUCT_DETAILS_FAIL:
            return { loading: false, error: action.payload }
```

Slika 25 Kod reduktora proizvoda

Reduktore moramo uključiti unutar 'store.js' datoteke i definirati 'combineReducers' koja se sadrži od *dictionarya* svih reducera kao što je prikazano na slici 26.

```
const reducer = combineReducers({
    productList: productListReducer,
    productDetails: productDetailsReducer,
    productDelete: productDeleteReducer,
    productCreate: productCreateReducer,
    productUpdate: productUpdateReducer,
    productReviewCreate: productReviewCreateReducer,
    productTopRated: productTopRatedReducer,
```

Slika 26 Kod combine reducera

Definirajmo i konstante koje reduceri koriste unutar direktorija 'constants/constants.js' (slika 27).

```
export const PRODUCT_LIST_REQUEST = 'PRODUCT_LIST_REQUEST'
export const PRODUCT_LIST_SUCCESS = 'PRODUCT_LIST_SUCCESS'
export const PRODUCT_LIST_FAIL = 'PRODUCT_LIST_FAIL'
```

Slika 27 Konstante reduktora

Ponavljamo iste korake i za akcije. Kreiramo 'productActions.js' unutar 'actions' direktorija. Akcije su funkcije koje su zadužene za slanje API poziva. Nećemo kreirati nove konstante, već koristimo iste kao i za 'reducere'. Na slici 28 prikazan je kod akcije proizvoda.

```
export const listProducts = (keyword = '') => async (dispatch) => {
    try {
        dispatch({ type: PRODUCT_LIST_REQUEST })

        const { data } = await axios.get(`/api/products${keyword}`)

        dispatch({
            type: PRODUCT_LIST_SUCCESS,
            payload: data
        })

    } catch (error) {
        dispatch({
            type: PRODUCT_LIST_FAIL,
            payload: error.response && error.response.data.detail
                ? error.response.data.detail
                : error.message,
        })
    }
}
```

Slika 28 Kod akcije proizvoda

3.4.3 Korištenje Reduxa na početnom zaslonu

Kada smo kreirali akcije i reducere, moramo ih početi koristiti kako bi on mogli ažurirati store. Stanja su definirana na razini komponenti, a pomoću akcija radimo API pozive kojim ažuriramo stanja u komponentama (slika 29).

```
const dispatch = useDispatch()
const productList = useSelector(state => state.productList)
const { error, loading, products, page, pages } = productList


useEffect(() => {
    dispatch(listProducts())

}, [dispatch, keyword])
```

Slika 29 Kod poziva akcija

3.4.4 Message i Loader komponente

Za 'loader' komponentu korištena je React Bootstrap Spinner komponenta, a za 'message' komponentu React Bootstrap React Alert Message komponenta (slike 30 i 31).

```
import React from 'react'
import { Alert } from 'react-bootstrap'

function Message({ variant, children }) {
    return (
        <Alert variant={variant}>
            {children}
        </Alert>
    )
}

export default Message

```

Slika 30 Kod Message komponenta

```
import React from 'react'
import { Spinner } from 'react-bootstrap'

function Loader() {
    return (
        <Spinner
            animation='border'
            role='status'
            style={{
                height: '100px',
                width: '100px',
                margin: 'auto',
                display: 'block'
            }}
        >
            <span className='sr-only'>Učitavam...</span>
        </Spinner>
    )
}

export default Loader
```

Slika 31 Kod sLoader komponenta

Obije komponente koristimo na početnoj stranici, s podacima koje smo dobili od prethodno kreiranih akcija i reduktora (slika 32).

```
<h1>Najnoviji proizvodi</h1>
{loading ? <Loader />
    : error ? <Message variant='danger'>{error}</Message>
        :
        <div>
```

Slika 32 Kod upotrebe komponenata Message i Loader

3.4.5 *Product Details* akcije i reduktori

Kako bi na stranici 'Product Screen' definiranoj u datoteci 'productScreen.js' mogli dobiti podatke iz storea moramo kreirati akcije i reducere. Pratimo već objašnjeni postupak. Slika 33 prikazuje reduktor proizvoda, a slika 34 akcije proizvoda.

```
export const productDetailsReducer = (state = { product: { reviews: [] } }, action) => {
    switch (action.type) {
        case PRODUCT_DETAILS_REQUEST:
            return { loading: true, ...state }

        case PRODUCT_DETAILS_SUCCESS:
            return { loading: false, product: action.payload }

        case PRODUCT_DETAILS_FAIL:
            return { loading: false, error: action.payload }

        default:
            return state
    }
}
```

Slika 33 Kod Product reduktora

```
export const listProductDetails = (id) => async (dispatch) => {
    try {
        dispatch({ type: PRODUCT_DETAILS_REQUEST })

        const { data } = await axios.get(`/api/products/${id}`)

        dispatch({
            type: PRODUCT_DETAILS_SUCCESS,
            payload: data
        })

    } catch (error) {
        dispatch({
            type: PRODUCT_DETAILS_FAIL,
            payload: error.response && error.response.data.detail
                ? error.response.data.detail
                : error.message,
        })
    }
}
```

Slika 34 Kod Product akcije

3.5 *Dodaj u košaricu funkcionalnost*

3.5.1 Odabir količine i 'Dodaj u košaricu'

Prvo ćemo dodati padajući meni unutar 'ProductScreen.js' datoteke koji će nuditi broj koliko proizvoda dodati u košaricu, ovisno o broju proizvoda na zalihi (slika 35).

```
<ListGroup.Item>
    <Row>
        <Col>Status:</Col>
        <Col>
            {product.countInStock > 0 ? 'Na zalihama' : 'Nema na zalihama'}
        </Col>
    </Row>
</ListGroup.Item>

{product.countInStock > 0 && (
    <ListGroup.Item>
        <Row>
            <Col>Količina:</Col>
            <Col xs='auto' className='my-1'>
                <Form.Control
                    as="select"
                    value={qty}
                    onChange={(e) => setQty(e.target.value)}
                >
                    {

                        [...Array(product.countInStock).keys()].map((x) => (
                            <option key={x + 1} value={x + 1}>
                                {x + 1}
                            </option>
                        ))
                    }

                </Form.Control>
            </Col>
        </Row>
    </ListGroup.Item>
```

Slika 35 Kod padajučeg izbornik

Nakon toga kreiramo gumb *Dodaj u košaricu* koji će biti onemogućen ako nema proizvoda na zalihama (slika 36).

```
<ListGroup.Item>
    <Button
        onClick={addToCartHandler}
        className='btn-block'
        disabled={product.countInStock == 0}
        type='button'>
        Dodaj u košaricu
    </Button>
</ListGroup.Item>
```

Slika 36 Kod dodaj u košaricu funkcionalnosti

Kreiramo i *handler* koji određuje akciju nakon klika na dugme. Ovom akcijom biti ćemo preusmjereni u košaricu (slika 37).

```
const addToCartHandler = () => {
    history.push(`/cart/${match.params.id}?qty=${qty}`)
}
```

Slika 37 Kod dodaj u košaricu handler

3.5.2 Akcije i reduktori košarice

Da bi napravili funkcionalnost dodavanja proizvoda u košaricu, moramo dodati novi dio u Redux store, tj. akcije, reduktore i konstante košarice. Ponovno slijedimo već poznati postupak. Prvo dodajemo konstante u novu datoteku 'cartConstants.js' (slika 38).

```
export const CART_ADD_ITEM = 'CART_ADD_ITEM'
export const CART_REMOVE_ITEM = 'CART_REMOVE_ITEM'

export const CART_SAVE_SHIPPING_ADDRESS = 'CART_SAVE_SHIPPING_ADDRESS'

export const CART_SAVE_PAYMENT_METHOD = 'CART_SAVE_PAYMENT_METHOD'

export const CART_CLEAR_ITEMS = 'CART_CLEAR_ITEMS'
```

Slika 38 Konstante košarice

Nakon toga dodajemo reduktore u datoteku 'cartReducers.js' i akcije u 'cartActions.js kao što je prikazano na slikama 39 i 40.

```
export const cartReducer = (state = { cartItems: [], shippingAddress: {} }, action) => {
    switch (action.type) {
        case CART_ADD_ITEM:
            const item = action.payload
            const existItem = state.cartItems.find(x => x.product === item.product)

            if (existItem) {
                return {
                    ...state,
                    cartItems: state.cartItems.map(x =>
                        x.product === existItem.product ? item : x)
                }
            } else {
                return {
                    ...state,
                    cartItems: [...state.cartItems, item]
                }
            }

        case CART_REMOVE_ITEM:
            return {
                ...state,
                cartItems: state.cartItems.filter(x => x.product !== action.payload)
            }

        case CART_SAVE_SHIPPING_ADDRESS:
            return {
                ...state,
                shippingAddress: action.payload
            }

        case CART_SAVE_PAYMENT_METHOD:
            return {
                ...state,
                paymentMethod: action.payload
            }

        case CART_CLEAR_ITEMS:
            return {
                ...state,
                cartItems: []
            }

        default:
            return state
    }
```

Slika 39 Kod reduktora košarice

```
export const addToCart = (id, qty) => async (dispatch, getState) => {
    const { data } = await axios.get(`/api/products/${id}`)

    dispatch({
        type: CART_ADD_ITEM,
        payload: {
            product: data._id,
            name: data.name,
            image: data.image,
            price: data.price,
            countInStock: data.countInStock,
            qty
        }
    })
    localStorage.setItem('cartItems', JSON.stringify(getState().cart.cartItems))
}
```

Slika 40 Kod akcije košarice

Nakon što dohvatimo podatke u lokalnu memoriju, moramo ih poslati u store. Unutar 'Store.js' datoteke dodati ćemo te podatke unutar inicijalnog stanja (slika 41).

```
const cartItemsFromStorage = localStorage.getItem('cartItems') ?
    JSON.parse(localStorage.getItem('cartItems')) : []

const initialState = {
    cart: {
        cartItems: cartItemsFromStorage,
        shippingAddress: shippingAddressFromStorage,
    },
    userLogin: { userInfo: userInfoFromStorage },
}
```

Slika 41 Kod prosljeđivanja podataka iz lokalnog skladišta u store

3.5.3 Dodaj u košaricu funkcionalnost

Sama funkcionalnost nalazi se unutar 'CartScreen.js' datoteke. Prije svega tu datoteku treba kreirati i registrirati u 'App.js' na već objašnjeni način. Uzeti ćemo *ID* proizvoda i količinu, te upotrebom 'addtoCart' akcije dodati proizvod u košaricu. Slika 42 prikazuje kod navedene funkcionalnosti.

```
function CartScreen({ match, location, history }) {
    const productId = match.params.id
    const qty = location.search ? Number(location.search.split('=')[1]) : 1
    const dispatch = useDispatch()

    useEffect(() => {
        if (productId) {
            dispatch(addToCart(productId, qty))
        }
    }, [dispatch, productId, qty])
```

Slika 42 Kod dodaj u košaricu funkcionalnosti

3.5.4 Ekran košarice

Ekran košarice podijeljen je u dva stupca. S lijeve strane su prikazani svi proizvodi u košarici, a s desne strane je sažetak narudžbe i gumb za nastavak na plaćanje (slike 43 i 44).

```
return (
    <Row>
        <Col md={8}>
            <h1>Košarica</h1>
            {cartItems.length === 0 ? (
                <Message variant='info'>
                    Vaša košarica je prazna. <Link to='/'>Natrag</Link>
                </Message>
            ) : (
                <ListGroup variant='flush'>
                    {cartItems.map(item => (
                        <ListGroup.Item key={item.product}>
                            <Row>
                                <Col md={2}>
                                    <Image src={item.image} alt={item.name} fluid rounded />
                                </Col>
                                <Col md={3}>
                                    <Link to={`/product/${item.product}`}>{item.name}</Link>
                                </Col>

                                <Col md={2}>
                                    {item.price} kn
                                </Col>

                                <Col md={3}>
                                    <Form.Control
                                        as="select"
                                        value={item.qty}
                                        onChange={(e) => dispatch(addToCart(item.product, Number(e.target.value)))}
                                    >
                                        {

                                            [...Array(item.countInStock).keys()].map((x) => (
                                                <option key={x + 1} value={x + 1}>
                                                    {x + 1}
                                                </option>
                                            ))
                                        }

                                    </Form.Control>
                                </Col>
```

Slika 43 Kod ekrana košarice (1/2)

```
                                    </Form.Control>
                                </Col>

                                <Col md={1}>
                                    <Button
                                        type='button'
                                        variant='light'
                                        onClick={() => removeFromCartHandler(item.product)}
                                    >
                                        <i className='fas fa-trash'></i>
                                    </Button>
                                </Col>
                            </Row>
                        </ListGroup.Item>
                    ))}
                </ListGroup>
            )}
        </Col>

        <Col md={4}>
            <Card>
                <ListGroup variant='flush'>
                    <ListGroup.Item>
                        <h2>Ukupno ({cartItems.reduce((acc, item) => acc + item.qty, 0)}) stavki</h2>
                        {cartItems.reduce((acc, item) => acc + item.qty * item.price, 0).toFixed(2)} kn
                    </ListGroup.Item>
                </ListGroup>

                <ListGroup.Item>
                    <Button
                        type='button'
                        className='btn-block'
                        disabled={cartItems.length === 0}
                        onClick={checkoutHandler}
                    >
                        Nastavite na plaćanje
                    </Button>
                </ListGroup.Item>


            </Card>
        </Col>
    </Row>
```

Slika 44 Kod ekrana košarice (2/2)

3.5.5 Ukloni iz košarice funkcionalnost

Za uklanjanje predmeta iz košarice slijedimo već objašnjenu proceduru dodavanja konstanti, akcija i reduktora. Konstante su prikazane na slici 38, reduktori na slici 39, a slika 45 pokazuje akciju uklanjanja iz košarice.

```
export const removeFromCart = (id) => (dispatch, getState) => {
    dispatch({
        type: CART_REMOVE_ITEM,
        payload: id,
    })

    localStorage.setItem('cartItems', JSON.stringify(getState().cart.cartItems))
}
```

Slika 45 Kod ukloni iz košarice akcije

3.6 *Backend autentikacija korisnika*

3.6.1 JSON Web tokeni

Autentikacija je proces u kojem doznajemo tko je korisnik. Korisnik upisuje svoje korisničko ime i lozinku i šalje ih backendu. Mi tada autenticiramo korisnika i znamo tko je on. Autorizacija je ono što smo odlučili što pojedini korisnik može raditi u aplikaciji, koje stranice može posjetiti i koji rutama u API-u ima pristup. Na primjer korisnik mora imati mogućnost vidjeti proizvode i radi narudžbe, ali ne i uređivati proizvode.

JSON Web token je kriptirani zapis podataka o korisniku. Tokeni se spremaju u lokalnom skladištu te se šalju sa svakim upitom prema serveru kako bi se korisnik mogao autorizirati. Da bi smo mogli koristiti JWT (JSON Web Token) moramo ga instalirati unutar svog virtualno okruženja naredbom „pip install djangorestframework-simplejwt“ i konfigurirati settings.py datoteku da koristi JWT kao defaultno sredstvo autentikacije (slika 46).

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    )
}
```

Slika 46Kod JWT autentikacije

Također, u 'Urls.py' datoteku potrebno je uključiti tokene i postaviti 'urlpatterns' varijablu. Ponašanje JWT tokena možemo konfigurirati kroz 'Settings.py' datoteku unutar varijable 'SIMPLE_JWT' (slika 47).

```
SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(days=30),
    'REFRESH_TOKEN_LIFETIME': timedelta(days=1),
    'ROTATE_REFRESH_TOKENS': False,
    'BLACKLIST_AFTER_ROTATION': True,
    'UPDATE_LAST_LOGIN': False,

    'ALGORITHM': 'HS256',
    'VERIFYING_KEY': None,
    'AUDIENCE': None,
    'ISSUER': None,

    'AUTH_HEADER_TYPES': ('Bearer',),
    'AUTH_HEADER_NAME': 'HTTP_AUTHORIZATION',
    'USER_ID_FIELD': 'id',
    'USER_ID_CLAIM': 'user_id',

    'AUTH_TOKEN_CLASSES': ('rest_framework_simplejwt.tokens.AccessToken',),
    'TOKEN_TYPE_CLAIM': 'token_type',

    'JTI_CLAIM': 'jti',

    'SLIDING_TOKEN_REFRESH_EXP_CLAIM': 'refresh_exp',
    'SLIDING_TOKEN_LIFETIME': timedelta(minutes=5),
    'SLIDING_TOKEN_REFRESH_LIFETIME': timedelta(days=1),
}
```

Slika 47Kod JWT konfiguracije

3.6.2 Korisnički serializer

Kako bi dobili korisničke podatke u svoj API kreirati ćemo URL path za objekt korisnika, a prije nego dobijemo podatke moramo napraviti 'serializer' za njih. Unutar 'views.py' datoteke kreiramo funkciju 'getUserProfile' koja koristi serializer 'UserSerializer'. Dodavanjem dekoratora 'permission' klase zaštićujemo ovu funkciju od pristupa neautenticiranim korisnicima (slika 48).

```
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def getUserProfile(request):
    user = request.user
    serializer = UserSerializer(user, many=False)
    return Response(serializer.data)
```

Slika 48 Kod za dohvačanje podatka o korisniku

Funkciju je potrebno dodati u 'urls.py' datoteku te definirati 'UserSerializer'. Slika 49 prikazuje kod serializera podataka.

```
class UserSerializerWithToken(UserSerializer):
    token = serializers.SerializerMethodField(read_only=True)

    class Meta:
        model = User
        fields = ['id', '_id', 'username', 'email', 'name', 'isAdmin', 'token']

    def get_token(self, obj):
        token = RefreshToken.for_user(obj)
        return str(token.access_token)
```

Slika 49 Kod serializera korisnika

3.6.3 Registracija korisnika

Za registraciju korisnika ponavljamo sličan postupak. Unutar 'views.py' datoteke kreiramo funkciju 'registerUser' te ju registriramo unutar 'urls.py' datoteke (slika 50).

```
@api_view(['POST'])
def registerUser(request):
    data = request.data
    try:
        user = User.objects.create(
            first_name=data['name'],
            username=data['email'],
            email=data['email'],
            password=make_password(data['password'])
        )

        serializer = UserSerializerWithToken(user, many=False)
        return Response(serializer.data)
    except:
        message = {'detail': 'User with this email already exists'}
        return Response(message, status=status.HTTP_400_BAD_REQUEST)
```

Slika 50 Kod registracije korisnika

3.7 *Frontend autentikacija korisnika*

3.7.1 Reduktori i akcije korisnika

Kada smo napravili funkcije za login i registraciju korisnika i API pozive za detalje o korisniku, možemo krenuti raditi frontend da bi te funkcionalnosti bile i vidljive. Krećemo s izradom konstanti, reduktora i akcija kao što su prikazani na slikama 51, 52 i 53.

```
export const USER_LOGIN_REQUEST = 'USER_LOGIN_REQUEST'
export const USER_LOGIN_SUCCESS = 'USER_LOGIN_SUCCESS'
export const USER_LOGIN_FAIL = 'USER_LOGIN_FAIL'

export const USER_LOGOUT = 'USER_LOGOUT'

export const USER_REGISTER_REQUEST = 'USER_REGISTER_REQUEST'
export const USER_REGISTER_SUCCESS = 'USER_REGISTER_SUCCESS'
export const USER_REGISTER_FAIL = 'USER_REGISTER_FAIL'
```

Slika 51 Konstante korisnika

```
export const userLoginReducer = (state = {}, action) => {
    switch (action.type) {
        case USER_LOGIN_REQUEST:
            return { loading: true }

        case USER_LOGIN_SUCCESS:
            return { loading: false, userInfo: action.payload }

        case USER_LOGIN_FAIL:
            return { loading: false, error: action.payload }

        case USER_LOGOUT:
            return {}

        default:
            return state
    }
}

export const userRegisterReducer = (state = {}, action) => {
    switch (action.type) {
        case USER_REGISTER_REQUEST:
            return { loading: true }

        case USER_REGISTER_SUCCESS:
            return { loading: false, userInfo: action.payload }

        case USER_REGISTER_FAIL:
            return { loading: false, error: action.payload }

        case USER_LOGOUT:
            return {}

        default:
            return state
    }
}
```

Slika 52 Kod reduktora korisnika

```
export const login = (email, password) => async (dispatch) => {
    try {
        dispatch({
            type: USER_LOGIN_REQUEST
        })

        const config = {
            headers: {
                'Content-type': 'application/json'
            }
        }

        const { data } = await axios.post(
            '/api/users/login/',
            { 'username': email, 'password': password },
            config
        )

        dispatch({
            type: USER_LOGIN_SUCCESS,
            payload: data
        })

        localStorage.setItem('userInfo', JSON.stringify(data))

    } catch (error) {
        dispatch({
            type: USER_LOGIN_FAIL,
            payload: error.response && error.response.data.detail
                ? error.response.data.detail
                : error.message,
        })
    }
}
```

Slika 53 Kod akcije korisnika

3.7.2 Login zaslon i funkcionalnost

Kreiramo datoteku unutar 'loginScreen.js' direktorija 'screens' te unutar nje gradimo login zaslon. Uključiti ćemo neke svoje komponente i neke komponente React Bootsrapa (slika 54).

```
import React, { useState, useEffect } from 'react'
import { Link } from 'react-router-dom'
import { Form, Button, Row, Col } from 'react-bootstrap'
import { useDispatch, useSelector } from 'react-redux'
import Loader from '../components/Loader'
import Message from '../components/Message'
import FormContainer from '../components/FormContainer'
import { login } from '../actions/userActions'
```

Slika 54 Kod login zaslona (1/3)

Nakon toga radimo Login funkciju, unutar koje ćemo koristiti login akciju (slika55).

```
function LoginScreen({ location, history }) {
    const [email, setEmail] = useState('')
    const [password, setPassword] = useState('')

    const dispatch = useDispatch()

    const redirect = location.search ? location.search.split('=')[1] : '/'

    const userLogin = useSelector(state => state.userLogin)
    const { error, loading, userInfo } = userLogin

    useEffect(() => {
        if (userInfo) {
            history.push(redirect)
        }
    }, [history, userInfo, redirect])

    const submitHandler = (e) => {
        e.preventDefault()
        dispatch(login(email, password))
    }
```

Slika 55 Kod login zaslona (2/3)

Nakon toga uređujemo stranicu (slika 56).

```
return (
    <FormContainer>
        <h1>Prijavi se</h1>
        {error && <Message variant='danger'>{error}</Message>}
        {loading && <Loader />}
        <Form onSubmit={submitHandler}>

            <Form.Group controlId='email'>
                <Form.Label>Email Adresa</Form.Label>
                <Form.Control
                    type='email'
                    placeholder='Vaš Email'
                    value={email}
                    onChange={(e) => setEmail(e.target.value)}
                >
                </Form.Control>
            </Form.Group>


            <Form.Group controlId='password'>
                <Form.Label>Lozinka</Form.Label>
                <Form.Control
                    type='password'
                    placeholder='Vaša Lozinka'
                    value={password}
                    onChange={(e) => setPassword(e.target.value)}
                >
                </Form.Control>
            </Form.Group>

            <Button type='submit' variant='primary'>
                Prijava
            </Button>
        </Form>

        <Row className='py-3'>
            <Col>
                Novi korisnik? <Link
                    to={redirect ? `/register?redirect=${redirect}` : '/register'}>
                    Registrirajte se!
                    </Link>
            </Col>
        </Row>

    </FormContainer>
)
```

Slika 56 Kod login zaslona (3/3)

Kada je stranica dizajnirana, moramo ju registrirati unutar 'App.js' datoteke.

3.7.3 Ekran registracije korisnika

Već smo kreirali akcije i reduktore za registraciju korisnika, te možemo krenuti na izradu same stranice. Kreiramo datoteku 'LoginScreen.js' unutar 'screens' direktorija i ponavljamo postupak sličan onom na stranici za prijavu korisnika (slika 57, 58 i 59).

```
import React, { useState, useEffect } from 'react'
import { Link } from 'react-router-dom'
import { Form, Button, Row, Col } from 'react-bootstrap'
import { useDispatch, useSelector } from 'react-redux'
import Loader from '../components/Loader'
import Message from '../components/Message'
import FormContainer from '../components/FormContainer'
import { register } from '../actions/userActions'

function RegisterScreen({ location, history }) {

    const [name, setName] = useState('')
    const [email, setEmail] = useState('')
    const [password, setPassword] = useState('')
    const [confirmPassword, setConfirmPassword] = useState('')
    const [message, setMessage] = useState('')

    const dispatch = useDispatch()

    const redirect = location.search ? location.search.split('=')[1] : '/'

    const userRegister = useSelector(state => state.userRegister)
    const { error, loading, userInfo } = userRegister

    useEffect(() => {
        if (userInfo) {
            history.push(redirect)
        }
    }, [history, userInfo, redirect])

    const submitHandler = (e) => {
        e.preventDefault()

        if (password != confirmPassword) {
            setMessage('Passwords do not match')
        } else {
            dispatch(register(name, email, password))
        }

    }
```

Slika 57 Kod ekrana registracije korisnika (1/3)

```
return (
    <FormContainer>
        <h1>Registracija</h1>
        {message && <Message variant='danger'>{message}</Message>}
        {error && <Message variant='danger'>{error}</Message>}
        {loading && <Loader />}
        <Form onSubmit={submitHandler}>

            <Form.Group controlId='name'>
                <Form.Label>Ime</Form.Label>
                <Form.Control
                    required
                    type='name'
                    placeholder='Ime...'
                    value={name}
                    onChange={(e) => setName(e.target.value)}
                >
                </Form.Control>
            </Form.Group>

            <Form.Group controlId='email'>
                <Form.Label>Email Adresa</Form.Label>
                <Form.Control
                    required
                    type='email'
                    placeholder='Email...'
                    value={email}
                    onChange={(e) => setEmail(e.target.value)}
                >
                </Form.Control>
            </Form.Group>

            <Form.Group controlId='password'>
                <Form.Label>Lozinka</Form.Label>
                <Form.Control
                    required
                    type='password'
                    placeholder='Lozinka...'
                    value={password}
                    onChange={(e) => setPassword(e.target.value)}
                >
                </Form.Control>
            </Form.Group>
```

Slika 58 Kod ekrana registracije korisnika (2/3)

```
                <Form.Group controlId='passwordConfirm'>
                    <Form.Label>Ponovljena Lozinka</Form.Label>
                    <Form.Control
                        required
                        type='password'
                        placeholder='Ponovljena Lozinka...'
                        value={confirmPassword}
                        onChange={(e) => setConfirmPassword(e.target.value)}
                    >
                    </Form.Control>
                </Form.Group>

                <Button type='submit' variant='primary'>
                    Registracija
                </Button>

            </Form>

            <Row className='py-3'>
                <Col>
                    Imate korisnički račun? <Link
                        to={redirect ? `/login?redirect=${redirect}` : '/login'}>
                        Prijava
                        </Link>
                </Col>
            </Row>
        </FormContainer >
    )
}

export default RegisterScreen
```

Slika 59 Kod ekrana registracije korisnika (3/3)

3.7.4 Ažuriraj korisnika funkcionalnost

Kada je korisnik prijavljen u aplikaciju, mora imati mogućnost promijeniti svoje ime, prezime, email i password. Za to moramo napraviti backend podršku. U backend direktoriju projekta, unutar datoteke 'user_views.py' definiramo funkciju 'updateUserProfile'. Funkciji dodajemo 'permission_classes' dekorator koji limitira korištenje funckije na prijavljene korisnike. Slika 60 prikazuje kod za navedenu funkcionalnost.

```
@api_view(['PUT'])
@permission_classes([IsAuthenticated])
def updateUserProfile(request):
    user = request.user
    serializer = UserSerializerWithToken(user, many=False)

    data = request.data
    user.first_name = data['name']
    user.username = data['email']
    user.email = data['email']

    if data['password'] != '':
        user.password = make_password(data['password'])

    user.save()

    return Response(serializer.data)
```

Slika 60 Kod za ažuriranje korisnika

3.7.5 Stranica profila

Kada smo napravili backend komponentu za ažuriranje profila, možemo krenuti s izradom frontenda. Prije nego napravimo samu stranicu profila moramo napraviti konstante, akcije i reduktore (slika 61, 62 i 63).

```
export const userDetailsReducer = (state = { user: {} }, action) => {
    switch (action.type) {
        case USER_DETAILS_REQUEST:
            return { ...state, loading: true }

        case USER_DETAILS_SUCCESS:
            return { loading: false, user: action.payload }

        case USER_DETAILS_FAIL:
            return { loading: false, error: action.payload }

        case USER_DETAILS_RESET:
            return { user: {} }


        default:
            return state
    }
}

export const userUpdateProfileReducer = (state = {}, action) => {
    switch (action.type) {
        case USER_UPDATE_PROFILE_REQUEST:
            return { loading: true }

        case USER_UPDATE_PROFILE_SUCCESS:
            return { loading: false, success: true, userInfo: action.payload }

        case USER_UPDATE_PROFILE_FAIL:
            return { loading: false, error: action.payload }

        case USER_UPDATE_PROFILE_RESET:
            return {}

        default:
            return state
    }
}
```

Slika 61 Kod reduktora stranice profila

```
export const getUserDetails = (id) => async (dispatch, getState) => {
    try {
        dispatch({
            type: USER_DETAILS_REQUEST
        })

        const {
            userLogin: { userInfo },
        } = getState()

        const config = {
            headers: {
                'Content-type': 'application/json',
                Authorization: `Bearer ${userInfo.token}`
            }
        }

        const { data } = await axios.get(
            `/api/users/${id}/`,
            config
        )

        dispatch({
            type: USER_DETAILS_SUCCESS,
            payload: data
        })


    } catch (error) {
        dispatch({
            type: USER_DETAILS_FAIL,
            payload: error.response && error.response.data.detail
                ? error.response.data.detail
                : error.message,
        })
    }
```

Slika 62 Kod akcija stranice profila

```
export const USER_DETAILS_REQUEST = 'USER_DETAILS_REQUEST'
export const USER_DETAILS_SUCCESS = 'USER_DETAILS_SUCCESS'
export const USER_DETAILS_FAIL = 'USER_DETAILS_FAIL'
export const USER_DETAILS_RESET = 'USER_DETAILS_RESET'

export const USER_UPDATE_PROFILE_REQUEST = 'USER_UPDATE_PROFILE_REQUEST'
export const USER_UPDATE_PROFILE_SUCCESS = 'USER_UPDATE_PROFILE_SUCCESS'
export const USER_UPDATE_PROFILE_FAIL = 'USER_UPDATE_PROFILE_FAIL'
export const USER_UPDATE_PROFILE_RESET = 'USER_UPDATE_PROFILE_RESET'

export const USER_LIST_REQUEST = 'USER_LIST_REQUEST'
export const USER_LIST_SUCCESS = 'USER_LIST_SUCCESS'
export const USER_LIST_FAIL = 'USER_LIST_FAIL'
export const USER_LIST_RESET = 'USER_LIST_RESET'
```

Slika 63 Konstante stranice profila

Unutar 'screens' direktorija kreiramo 'ProfileScreen.js' datoteku unutar koje ćemo oblikovati stranicu profila. Stranica je napravljena u dva stupca. U lijevom je forma za ažuriranje profila, a na desnoj strani sve narudžbe korisnika. Kod za izradu stranice profila prikazan je na slikama 64-68.

```
import React, { useState, useEffect } from 'react'
import { Link } from 'react-router-dom'
import { Form, Button, Row, Col, Table } from 'react-bootstrap'
import { LinkContainer } from 'react-router-bootstrap'
import { useDispatch, useSelector } from 'react-redux'
import Loader from '../components/Loader'
import Message from '../components/Message'
import { getUserDetails, updateUserProfile } from '../actions/userActions'
import { USER_UPDATE_PROFILE_RESET } from '../constants/userConstants'
import { listMyOrders } from '../actions/orderActions'

function ProfileScreen({ history }) {

    const [name, setName] = useState('')
    const [email, setEmail] = useState('')
    const [password, setPassword] = useState('')
    const [confirmPassword, setConfirmPassword] = useState('')
    const [message, setMessage] = useState('')

    const dispatch = useDispatch()

    const userDetails = useSelector(state => state.userDetails)
    const { error, loading, user } = userDetails

    const userLogin = useSelector(state => state.userLogin)
    const { userInfo } = userLogin

    const userUpdateProfile = useSelector(state => state.userUpdateProfile)
    const { success } = userUpdateProfile

    const orderListMy = useSelector(state => state.orderListMy)
    const { loading: loadingOrders, error: errorOrders, orders } = orderListMy
```

Slika 64 Kod stranice profila (1/5)

```
useEffect(() => {
    if (!userInfo) {
        history.push('/login')
    } else {
        if (!user || !user.name || success || userInfo._id !== user._id) {
            dispatch({ type: USER_UPDATE_PROFILE_RESET })
            dispatch(getUserDetails('profile'))
            dispatch(listMyOrders())
        } else {
            setName(user.name)
            setEmail(user.email)
        }
    }
}, [dispatch, history, userInfo, user, success])

const submitHandler = (e) => {
    e.preventDefault()

    if (password != confirmPassword) {
        setMessage('Passwords do not match')
    } else {
        dispatch(updateUserProfile({
            'id': user._id,
            'name': name,
            'email': email,
            'password': password
        }))
        setMessage('')
```

Slika 65 Kod stranice profila (2/5)

```
<Col md={3}>
    <h2>Korisnički profil</h2>

    {message && <Message variant='danger'>{message}</Message>}
    {error && <Message variant='danger'>{error}</Message>}
    {loading && <Loader />}
    <Form onSubmit={submitHandler}>

        <Form.Group controlId='name'>
            <Form.Label>Ime</Form.Label>
            <Form.Control
                required
                type='name'
                placeholder='Ime...'
                value={name}
                onChange={(e) => setName(e.target.value)}
            >
            </Form.Control>
        </Form.Group>

        <Form.Group controlId='email'>
            <Form.Label>Email Adresa</Form.Label>
            <Form.Control
                required
                type='email'
                placeholder='Email...'
                value={email}
                onChange={(e) => setEmail(e.target.value)}
            >
            </Form.Control>
        </Form.Group>
```

Slika 66 Kod stranice profila (3/5)

```
        <Form.Group controlId='password'>
            <Form.Label>Lozinka</Form.Label>
            <Form.Control

                type='password'
                placeholder='Lozinka...'
                value={password}
                onChange={(e) => setPassword(e.target.value)}
            >
            </Form.Control>
        </Form.Group>

        <Form.Group controlId='passwordConfirm'>
            <Form.Label>Ponovljena Lozinka</Form.Label>
            <Form.Control

                type='password'
                placeholder='Ponovljena lozinka'
                value={confirmPassword}
                onChange={(e) => setConfirmPassword(e.target.value)}
            >
            </Form.Control>
        </Form.Group>

        <Button type='submit' variant='primary'>
            Ažuriraj
    </Button>

    </Form>
</Col>
```

Slika 67 Kod stranice profila (4/5)

```
<Col md={9}>
    <h2>Moje narudžbe</h2>
    {loadingOrders ? (
        <Loader />
    ) : errorOrders ? (
        <Message variant='danger'>{errorOrders}</Message>
    ) : (
                <Table striped responsive className='table-sm'>
                    <thead>
                        <tr>
                            <th>ID</th>
                            <th>Datum</th>
                            <th>Ukupno</th>
                            <th>Plaćeno</th>
                            <th>Dostavljeno</th>
                            <th></th>
                        </tr>
                    </thead>

                    <tbody>
                        {orders.map(order => (
                            <tr key={order._id}>
                                <td>{order._id}</td>
                                <td>{order.createdAt.substring(0, 10)}</td>
                                <td>${order.totalPrice}</td>
                                <td>{order.isPaid ? order.paidAt.substring(0, 10) : (
                                    <i className='fas fa-times' style={{ color: 'red' }}></i>
                                )}</td>
                                <td>{order.isDelivered ? order.deliveredAt.substring(0, 10) : (
                                    <i className='fas fa-times' style={{ color: 'red' }}></i>
                                )}</td>
                                <td>
                                    <LinkContainer to={`/order/${order._id}`}>
                                        <Button className='btn-sm'>Više</Button>
                                    </LinkContainer>
                                </td>
                            </tr>
                        ))}
                    </tbody>
                </Table>
            )}
    </Col>
</Row>
```

Slika 68 Kod stranice profila (5/5)

3.8 *Checkout funkcionalnost*

3.8.1 Podaci o dostavi

Ekran za podatke o dostavi pojavljuje se kada iz košarice korisnik krene na plaćanje. Kako bi kreirali taj ekran radimo novu datoteku 'shippingScreen.js'. Za funkcionalnost stranice moramo napraviti i

pripadajuće akcije i reduktore. Slike 69 i 70 prikazuju kod same stranice dostave, dok su na slikama 71 i 72 prikazane akcije i reduktori stranice dostave.

```
import React, { useState, useEffect } from 'react'
import { Form, Button } from 'react-bootstrap'
import { useDispatch, useSelector } from 'react-redux'
import FormContainer from '../components/FormContainer'
import CheckoutSteps from '../components/CheckoutSteps'
import { saveShippingAddress } from '../actions/cartActions'

function ShippingScreen({ history }) {

    const cart = useSelector(state => state.cart)
    const { shippingAddress } = cart

    const dispatch = useDispatch()

    const [address, setAddress] = useState(shippingAddress.address)
    const [city, setCity] = useState(shippingAddress.city)
    const [postalCode, setPostalCode] = useState(shippingAddress.postalCode)
    const [country, setCountry] = useState(shippingAddress.country)

    const submitHandler = (e) => {
        e.preventDefault()
        dispatch(saveShippingAddress({ address, city, postalCode, country }))
        history.push('/payment')
    }
```

Slika 69 Kod za stranicu dostave (1/2)

```
return (
    <FormContainer>
        <CheckoutSteps step1 step2 />
        <h1>Dostava</h1>
        <Form onSubmit={submitHandler}>

            <Form.Group controlId='address'>
                <Form.Label>Addresa</Form.Label>
                <Form.Control
                    required
                    type='text'
                    placeholder='Addresa...'
                    value={address ? address : ''}
                    onChange={(e) => setAddress(e.target.value)}
                >
                </Form.Control>
            </Form.Group>

            <Form.Group controlId='city'>
                <Form.Label>Grad</Form.Label>
                <Form.Control
                    required
                    type='text'
                    placeholder='Grad...'
                    value={city ? city : ''}
                    onChange={(e) => setCity(e.target.value)}
                >
                </Form.Control>
            </Form.Group>

            <Form.Group controlId='postalCode'>
                <Form.Label>Poštanski broj</Form.Label>
                <Form.Control
                    required
                    type='text'
                    placeholder='Poštanski broj...'
                    value={postalCode ? postalCode : ''}
                    onChange={(e) => setPostalCode(e.target.value)}
                >
                </Form.Control>
            </Form.Group>

            <Form.Group controlId='country'>
                <Form.Label>Država</Form.Label>
```

Slika 70 Kod za stranicu dostave (2/2)

```
        case CART_SAVE_SHIPPING_ADDRESS:
            return {
                ...state,
                shippingAddress: action.payload
            }
```

Slika 71 Kod reduktora adrese za dostavu

```
export const saveShippingAddress = (data) => (dispatch) => {
    dispatch({
        type: CART_SAVE_SHIPPING_ADDRESS,
        payload: data,
    })

    localStorage.setItem('shippingAddress', JSON.stringify(data))
}
```

Slika 72 Kod akcije adrese za dostavu

3.8.2 Komponenta progresa *checkout* procesa

Komponenta progresa nam govori na kojem se trenutno koraku *checkouta* nalazim i daje mogućnost vratit se natrag. Kreiramo datoteku 'CheckoutSteps.js' unutar 'components' direktorija u kojoj ćemo dizajnirati komponentu (slika 73).

```
function CheckoutSteps({ step1, step2, step3, step4 }) {

    return (
        <Nav className='justify-content-center mb-4'>
            <Nav.Item>
                {step1 ? (
                    <LinkContainer to='/login'>
                        <Nav.Link>Prijavi se</Nav.Link>
                    </LinkContainer>
                ) : (
                        <Nav.Link disabled>Prijavi se</Nav.Link>
                        )}
            </Nav.Item>

            <Nav.Item>
                {step2 ? (
                    <LinkContainer to='/shipping'>
                        <Nav.Link>Dostava</Nav.Link>
                    </LinkContainer>
                ) : (
                        <Nav.Link disabled>Dostava</Nav.Link>
                        )}
            </Nav.Item>

            <Nav.Item>
                {step3 ? (
                    <LinkContainer to='/payment'>
                        <Nav.Link>Plaćanje</Nav.Link>
                    </LinkContainer>
                ) : (
                        <Nav.Link disabled>Plaćanje</Nav.Link>
                        )}
            </Nav.Item>

            <Nav.Item>
                {step4 ? (
                    <LinkContainer to='/placeorder'>
                        <Nav.Link>Narudžba</Nav.Link>
                    </LinkContainer>
                ) : (
                        <Nav.Link disabled>Narudžba</Nav.Link>
                        )}
            </Nav.Item>
        </Nav>
    )
```

Slika 73 Kod checkout procesa

3.8.3 Odabir načina plaćanja

Na stranicu za odabir plaćanja možemo dodati izbor koji želimo. Za ovaj primjer postavljene su dvije mogućnosti: plaćanje pouzećem i PayPal. Počinjemo s kreiranjem 'paymentScreen.js' datoteke unutar 'screens' direktorija. Gradimo stranicu na isti način kao i prethodne. Slike 74 i 75 prikazuju kod stranice načina plaćanja, dok su na slikama 76, 77 i 78 prikazane konstante, akcije i reduktori.

```
import React, { useState, useEffect } from 'react'
import { Form, Button, Col } from 'react-bootstrap'
import { useDispatch, useSelector } from 'react-redux'
import FormContainer from '../components/FormContainer'
import CheckoutSteps from '../components/CheckoutSteps'
import { savePaymentMethod } from '../actions/cartActions'

function PaymentScreen({ history }) {

    const cart = useSelector(state => state.cart)
    const { shippingAddress } = cart

    const dispatch = useDispatch()

    const [paymentMethod, setPaymentMethod] = useState('')

    if (!shippingAddress.address) {
        history.push('/shipping')
    }

    const submitHandler = (e) => {
        e.preventDefault()
        dispatch(savePaymentMethod(paymentMethod))
        history.push('/placeorder')
    }
```

Slika 74 Kod za način plaćanja (1/2)

```
return (
    <FormContainer>
        <CheckoutSteps step1 step2 step3 />

        <Form onSubmit={submitHandler}>
            <Form.Group>
                <Form.Label as='legend'>Izaberite metodu plaćanja</Form.Label>
                <Col>
                    <Form.Check
                        type='radio'
                        label='PayPal ili Kreditna kartica'
                        id='paypal'
                        name='paymentMethod'
                        onChange={(e) => setPaymentMethod('PayPal')}
                    >
                    </Form.Check>,
                    <Form.Check
                        type='radio'
                        label='Plaćanje pouzećem'
                        id='pouzece'
                        name='paymentMethod'
                        onChange={(e) => setPaymentMethod('pouzece')}
                    >

                    </Form.Check>
                </Col>
            </Form.Group>

            <Button type='submit' variant='primary'>
                Dalje
            </Button>
        </Form>
    </FormContainer>
    )
}

export default PaymentScreen
```

Slika 75 Kod za način plaćanja (2/2)

Nakon toga kreiramo potrebne akcije, reduktore i konstante.

```
export const savePaymentMethod = (data) => (dispatch) => {
    dispatch({
        type: CART_SAVE_PAYMENT_METHOD,
        payload: data,
    })

    localStorage.setItem('paymentMethod', JSON.stringify(data))
}
```

Slika 76 Kod akcija načina plaćanja

```
case CART_SAVE_PAYMENT_METHOD:
    return {
        ...state,
        paymentMethod: action.payload
    }
```

Slika 77 Reduktor načina plaćanja

```
export const CART_SAVE_SHIPPING_ADDRESS = 'CART_SAVE_SHIPPING_ADDRESS'
```

Slika 78 Konstanta načina plaćanja

3.8.4 Ekran narudžbe

Na lijevoj strani ekrana narudžbe nalaze se informacije kao adresa, način plaćanja i naručene stavke. Na desnoj strani je vidljiva tablica s cijenom, dostavom (dostava se plaća za narudžbe manje od 100 kn), PDV-om i ukupnom sumom. Na dnu iste tablice je dugme 'Nastavi na plaćanje'. Stranicu gradimo prema već poznatom postupku, a počinjemo kreiranjem datoteke 'OrderScreen.js' unutar koje pišemo kod (slike 79 i 80).

```
import React, { useState, useEffect } from 'react'
import { Button, Row, Col, ListGroup, Image, Card } from 'react-bootstrap'
import { Link } from 'react-router-dom'
import { useDispatch, useSelector } from 'react-redux'
import Message from '../components/Message'
import CheckoutSteps from '../components/CheckoutSteps'
import { createOrder } from '../actions/orderActions'
import { ORDER_CREATE_RESET } from '../constants/orderConstants'

function PlaceOrderScreen({ history }) {

    const orderCreate = useSelector(state => state.orderCreate)
    const { order, error, success } = orderCreate

    const dispatch = useDispatch()

    const cart = useSelector(state => state.cart)

    cart.itemsPrice = cart.cartItems.reduce((acc, item) => acc + item.price * item.qty, 0).toFixed(2)
    cart.shippingPrice = (cart.itemsPrice > 100 ? 0 : 10).toFixed(2)
    cart.taxPrice = Number((0.25) * cart.itemsPrice).toFixed(2)

    cart.totalPrice = (Number(cart.itemsPrice) + Number(cart.shippingPrice) + Number(cart.taxPrice)).toFixed(2)


    if (!cart.paymentMethod) {
        history.push('/payment')
    }
    var payment = 'PayPal'
    if (cart.paymentMethod == 'pouzece'){
        payment ='Plačanje pouzećem'
    }

    useEffect(() => {
        if (success) {
            history.push(`/order/${order._id}`)
            dispatch({ type: ORDER_CREATE_RESET })
        }
    }, [success, history])
```

Slika 79 Kod za ekran narudžbe (1/2)

```
useEffect(() => {
    if (success) {
        history.push(`/order/${order._id}`)
        dispatch({ type: ORDER_CREATE_RESET })
    }
}, [success, history])

const placeOrder = () => {
    dispatch(createOrder({
        orderItems: cart.cartItems,
        shippingAddress: cart.shippingAddress,
        paymentMethod: cart.paymentMethod,
        itemsPrice: cart.itemsPrice,
        shippingPrice: cart.shippingPrice,
        taxPrice: cart.taxPrice,
        totalPrice: cart.totalPrice,
    }))
}

return (
    <div>
        <CheckoutSteps step1 step2 step3 step4 />
        <Row>
            <Col md={8}>
                <ListGroup variant='flush'>
                    <ListGroup.Item>
                        <h2>Podaci o dostavi</h2>

                        <p>
                            <strong>Adresa: </strong>
                            {cart.shippingAddress.address},  {cart.shippingAddress.city}
                            {' '}
                            {cart.shippingAddress.postalCode},
                            {' '}
                            {cart.shippingAddress.country}
                        </p>
                    </ListGroup.Item>

                    <ListGroup.Item>
                        <h2>Način plaćanja</h2>
                        <p>
                            <strong></strong>
                            {payment}
                        </p>
                    </ListGroup.Item>

                    <ListGroup.Item>
```

Slika 80 Kod za ekran narudžbe (2/2)

3.8.5 Kreiranje pogleda i URL ruta

Da bi dovršili *checkout* funkcionalnost moramo napraviti poglede i URL rute u backendu. Unutar 'views' direktorija u backend-u, kreiramo datoteku 'oreder_views.py'. Uz same poglede moramo kreirati i serializere podataka (slike 81 i 82).

```python
from django.shortcuts import render

from rest_framework.decorators import api_view, permission_classes
from rest_framework.permissions import IsAuthenticated, IsAdminUser
from rest_framework.response import Response

from base.models import Product, Order, OrderItem, ShippingAddress
from base.serializers import ProductSerializer, OrderSerializer

from rest_framework import status
from datetime import datetime


@api_view(['POST'])
@permission_classes([IsAuthenticated])
def addOrderItems(request):
    user = request.user
    data = request.data

    orderItems = data['orderItems']

    if orderItems and len(orderItems) == 0:
        return Response({'detail': 'Nema predmeta.'}, status=status.HTTP_400_BAD_REQUEST)
    else:



        order = Order.objects.create(
            user=user,
            paymentMethod=data['paymentMethod'],
            taxPrice=data['taxPrice'],
            shippingPrice=data['shippingPrice'],
            totalPrice=data['totalPrice']
        )



        shipping = ShippingAddress.objects.create(
            order=order,
            address=data['shippingAddress']['address'],
            city=data['shippingAddress']['city'],
            postalCode=data['shippingAddress']['postalCode'],
            country=data['shippingAddress']['country'],
        )
```

Slika 81 Kod ogleda za dodavanje narudžbe (1/2)

```python
        for i in orderItems:
            product = Product.objects.get(_id=i['product'])

            item = OrderItem.objects.create(
                product=product,
                order=order,
                name=product.name,
                qty=i['qty'],
                price=i['price'],
                image=product.image.url,
            )

            product.countInStock -= item.qty
            product.save()

        serializer = OrderSerializer(order, many=False)
        return Response(serializer.data)
```

Slika 82 Kod ogleda za dodavanje narudžbe (2/2)

3.8.6 Kreiranje narudžbe

Da bi kreirali narudžbu i spremili ju u bazu podataka, moramo napraviti reduktore i akcije narudžbe (slike 83 i 43).

```
export const orderCreateReducer = (state = {}, action) => {
    switch (action.type) {
        case ORDER_CREATE_REQUEST:
            return {
                loading: true
            }

        case ORDER_CREATE_SUCCESS:
            return {
                loading: false,
                success: true,
                order: action.payload
            }

        case ORDER_CREATE_FAIL:
            return {
                loading: false,
                error: action.payload
            }

        case ORDER_CREATE_RESET:
            return {}


        default:
            return state
    }
}
```

Slika 83 Kod reduktora narudžbe

```
export const createOrder = (order) => async (dispatch, getState) => {
    try {
        dispatch({
            type: ORDER_CREATE_REQUEST
        })

        const {
            userLogin: { userInfo },
        } = getState()

        const config = {
            headers: {
                'Content-type': 'application/json',
                Authorization: `Bearer ${userInfo.token}`
            }
        }

        const { data } = await axios.post(
            `/api/orders/add/`,
            order,
            config
        )

        dispatch({
            type: ORDER_CREATE_SUCCESS,
            payload: data
        })

        dispatch({
            type: CART_CLEAR_ITEMS,
            payload: data
        })

        localStorage.removeItem('cartItems')


    } catch (error) {
        dispatch({
            type: ORDER_CREATE_FAIL,
            payload: error.response && error.response.data.detail
                ? error.response.data.detail
                : error.message,
        })
    }
}
```

Slika 84 Kod akcije narudžbe

3.8.7 Dohvaćanje narudžbe prema ID-u

Da bi mogli dohvatiti narudžbu prema id-u narudžbe moramo napravit 'endpoint' u backendu projekta. Za to nam je potreban pogled i URL, koje ćemo kreirati po već objašnjenom postupku (slike 85 i 86).

```python
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def getOrderById(request, pk):

    user = request.user

    try:
        order = Order.objects.get(_id=pk)
        if user.is_staff or order.user == user:
            serializer = OrderSerializer(order, many=False)
            return Response(serializer.data)
        else:
            Response({'detail': 'Niste autorizirani za pregled ove narudžbe.'},
                     status=status.HTTP_400_BAD_REQUEST)
    except:
        return Response({'detail': 'Narudžba ne postoji.'}, status=status.HTTP_400_BAD_REQUEST)
```

Slika 85 Kod za dohvatiti narudžbu po ID-u

```python
from django.urls import path
from base.views import order_views as views


urlpatterns = [

    path('', views.getOrders, name='orders'),
    path('add/', views.addOrderItems, name='orders-add'),
    path('myorders/', views.getMyOrders, name='myorders'),
```

Slika 86 Kod URL-a narudžbi

3.8.8 Kreiranje ekrana narudžbe

Prije same kreacije ekrana narudžbe, potrebno je napravit akcije i reduktore kako bi dobili narudžbu putem pogleda kojeg smo već napravili (slike 87 i 88).

```
export const getOrderDetails = (id) => async (dispatch, getState) => {
    try {
        dispatch({
            type: ORDER_DETAILS_REQUEST
        })

        const {
            userLogin: { userInfo },
        } = getState()

        const config = {
            headers: {
                'Content-type': 'application/json',
                Authorization: `Bearer ${userInfo.token}`
            }
        }

        const { data } = await axios.get(
            `/api/orders/${id}/`,
            config
        )

        dispatch({
            type: ORDER_DETAILS_SUCCESS,
            payload: data
        })


    } catch (error) {
        dispatch({
            type: ORDER_DETAILS_FAIL,
            payload: error.response && error.response.data.detail
                ? error.response.data.detail
                : error.message,
        })
    }
}
```

Slika 87 Kod akcija narudžbe

```
export const orderDetailsReducer = (state = { loading: true, orderItems: [], shippingAddress: {} }, action) => {
    switch (action.type) {
        case ORDER_DETAILS_REQUEST:
            return {
                ...state,
                loading: true
            }

        case ORDER_DETAILS_SUCCESS:
            return {
                loading: false,
                order: action.payload
            }

        case ORDER_DETAILS_FAIL:
            return {
                loading: false,
                error: action.payload
            }


        default:
            return state
    }
```

Slika 88 Kod reduktora narudžbe

Nakon toga možemo kreirati 'orederScreen.js' datoteku u kojoj ćemo dizajnirati svoju stranicu narudžbe korištenjem gore definiranih akcija i reduktora kao što je prikazano na slikama 89 i 90.

```
import React, { useState, useEffect } from 'react'
import { Button, Row, Col, ListGroup, Image, Card } from 'react-bootstrap'
import { Link } from 'react-router-dom'
import { useDispatch, useSelector } from 'react-redux'
import { PayPalButton } from 'react-paypal-button-v2'
import Message from '../components/Message'
import Loader from '../components/Loader'
import { getOrderDetails, payOrder, deliverOrder } from '../actions/orderActions'
import { ORDER_PAY_RESET, ORDER_DELIVER_RESET } from '../constants/orderConstants'

function Pouzece(nacin) {
    if (nacin == 'PayPal'){return true}
  }

function OrderScreen({ match, history }) {
    const orderId = match.params.id
    const dispatch = useDispatch()


    const [sdkReady, setSdkReady] = useState(false)

    const cart = useSelector(state => state.cart)
    const { shippingAddress } = cart

    const orderDetails = useSelector(state => state.orderDetails)
    const { order, error, loading } = orderDetails

    const orderPay = useSelector(state => state.orderPay)
    const { loading: loadingPay, success: successPay } = orderPay

    const orderDeliver = useSelector(state => state.orderDeliver)
    const { loading: loadingDeliver, success: successDeliver } = orderDeliver

    const userLogin = useSelector(state => state.userLogin)
    const { userInfo } = userLogin


    if (!loading && !error) {
        order.itemsPrice = order.orderItems.reduce((acc, item) => acc + item.price * item.qty, 0).toFixed(2)
    }
```

Slika 89 Kod za ekran narudžbe (1/2)

```
useEffect(() => {

    if (!userInfo) {
        history.push('/login')
    }

    if (!order || successPay || order._id !== Number(orderId) || successDeliver) {
        dispatch({ type: ORDER_PAY_RESET })
        dispatch({ type: ORDER_DELIVER_RESET })

        dispatch(getOrderDetails(orderId))
    } else if (!order.isPaid) {
        if (!window.paypal) {
            addPayPalScript()
        } else {
            setSdkReady(true)
        }
    }
}, [dispatch, order, orderId, successPay, successDeliver])


const successPaymentHandler = (paymentResult) => {
    dispatch(payOrder(orderId, paymentResult))
}

const deliverHandler = () => {
    dispatch(deliverOrder(order))
}

return loading ? (
    <Loader />
) : error ? (
    <Message variant='danger'>{error}</Message>
) : (
            <div>
                <h1>Narudžba {order.Id}</h1>
                <Row>
                    <Col md={8}>
                        <ListGroup variant='flush'>
                            <ListGroup.Item>
                                <h2>Dostava</h2>
                                <p><strong>Ime: </strong> {order.user.name}</p>
                                <p><strong>Email: </strong><a href={`mailto:${order.user.email}`}>{order.user.email}</a></p>
                                <p>
                                    <strong>Adresa: </strong>
```

Slika 90 Kod za ekran narudžbe (2/2)

3.8.9 Označi kao plaćeno pogled

Da bi mogli narudžbu označiti kao plaćenu, moramo kreirati *endpoint* u backendu koji se sastoji od pogleda i URL-a (slike 91 i 92).

```python
@api_view(['PUT'])
@permission_classes([IsAuthenticated])
def updateOrderToPaid(request, pk):
    order = Order.objects.get(_id=pk)

    order.isPaid = True
    order.paidAt = datetime.now()
    order.save()

    return Response('Narudžba je plaćena.')
```

Slika 91 Kod za označi kao plačeno view

```python
path('<str:pk>/', views.getOrderById, name='user-order'),
path('<str:pk>/pay/', views.updateOrderToPaid, name='pay'),
```

Slika 92 Kod za označi kao plaćeno URL path

3.8.10 Akcije i reduktori za plaćanje

Kako bi implementirali plaćanje u web aplikaciju, moramo definirati konstante, akcije i reduktore plaćanja (slike 93, 94 i 95).

```javascript
export const ORDER_PAY_REQUEST = 'ORDER_PAY_REQUEST'
export const ORDER_PAY_SUCCESS = 'ORDER_PAY_SUCCESS'
export const ORDER_PAY_FAIL = 'ORDER_PAY_FAIL'
export const ORDER_PAY_RESET = 'ORDER_PAY_RESET'
```

Slika 93 Konstante plaćanja

```javascript
export const orderPayReducer = (state = {}, action) => {
    switch (action.type) {
        case ORDER_PAY_REQUEST:
            return {
                loading: true
            }

        case ORDER_PAY_SUCCESS:
            return {
                loading: false,
                success: true
            }

        case ORDER_PAY_FAIL:
            return {
                loading: false,
                error: action.payload
            }

        case ORDER_PAY_RESET:
            return {}

        default:
            return state
    }
}
```

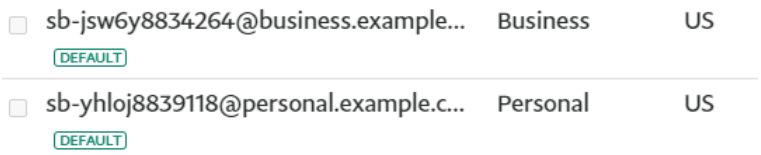
Slika 94 Kod reduktora plaćanja

```
export const payOrder = (id, paymentResult) => async (dispatch, getState) => {
    try {
        dispatch({
            type: ORDER_PAY_REQUEST
        })

        const {
            userLogin: { userInfo },
        } = getState()

        const config = {
            headers: {
                'Content-type': 'application/json',
                Authorization: `Bearer ${userInfo.token}`
            }
        }

        const { data } = await axios.put(
            `/api/orders/${id}/pay/`,
            paymentResult,
            config
        )

        dispatch({
            type: ORDER_PAY_SUCCESS,
            payload: data
        })


    } catch (error) {
        dispatch({
            type: ORDER_PAY_FAIL,
            payload: error.response && error.response.data.detail
                ? error.response.data.detail
                : error.message,
        })
    }
```

Slika 95 Kod akcija plaćanja

3.8.11 PayPal integracija

Kako bi integrirali PayPal unutar naše web aplikacije moramo se registrirati na njihovu stranicu. PayPal nudi i *developer* račun s kojim je moguće testirati integraciju samog API-a. Unutar developers računa imamo već kreirana 2 *accounta,* jedan poslovni koji ćemo integrirati unutar aplikacije i jedan privatni s kojim možemo testirati uspješnost plaćanja (slika 96).



Slika 96 PayPal računi

Unutar order screena dodajemo skripte za prikaz PayPal gumba, koje mogu biti pronađene na PayPal korisničkom računu. Dugme za PayPal smo već stavili u stranicu narudžbe, a koristili smo React Bootstrap PayPal Button (slika 97).

```
const addPayPalScript = () => {
    const script = document.createElement('script')
    script.type = 'text/javascript'
    script.src = 'https://www.paypal.com/sdk/js?client-id=AffnqKZoJo9kmPwPfF8kkqx4SwlaJ7GL3tByAb9ZIPQdchMuqbi13zEdVuFxs9_7tWVDmI
    script.async = true
    script.onload = () => {
        setSdkReady(true)
    }
    document.body.appendChild(script)
}
```

Slika 97 Kod PayPal skripte

3.8.12 Prikaz narudžbi na profilu korisnika

Kako bi prikazali narudžbe na profilu korisnika, prvo moramo napraviti pogled u backendu, a onda i dodati funkcionalnost u ekran profila u frontendu kao što je prikazano na slikama 98, 99 i 100.

```
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def getMyOrders(request):
    user = request.user
    orders = user.order_set.all()
    serializer = OrderSerializer(orders, many=True)
    return Response(serializer.data)
```

Slika 98 Kod za pregled korisničkih narudžbi

```
const orderListMy = useSelector(state => state.orderListMy)
const { loading: loadingOrders, error: errorOrders, orders } = orderListMy
```

Slika 99 Kod za dohvačanje podataka o narudžbi

```
<Col md={9}>
    <h2>Moje narudžbe</h2>
    {loadingOrders ? (
        <Loader />
    ) : errorOrders ? (
        <Message variant='danger'>{errorOrders}</Message>
    ) : (
                <Table striped responsive className='table-sm'>
                    <thead>
                        <tr>
                            <th>ID</th>
                            <th>Datum</th>
                            <th>Ukupno</th>
                            <th>Plaćeno</th>
                            <th>Dostavljeno</th>
                            <th></th>
                        </tr>
                    </thead>

                    <tbody>
                        {orders.map(order => (
                            <tr key={order._id}>
                                <td>{order._id}</td>
                                <td>{order.createdAt.substring(0, 10)}</td>
                                <td>${order.totalPrice}</td>
                                <td>{order.isPaid ? order.paidAt.substring(0, 10) : (
                                    <i className='fas fa-times' style={{ color: 'red' }}></i>
                                )}</td>
                                <td>{order.isDelivered ? order.deliveredAt.substring(0, 10) : (
                                    <i className='fas fa-times' style={{ color: 'red' }}></i>
                                )}</td>
                                <td>
                                    <LinkContainer to={`/order/${order._id}`}>
                                        <Button className='btn-sm'>Više</Button>
                                    </LinkContainer>
                                </td>
                            </tr>
                        ))}
                    </tbody>
                </Table>
            )}
</Col>
```

Slika 100 Kod za ispis narudžbi

3.9 *Admin funkcionalnost*

3.9.1 Admin ekran korisnika, akcije i reduktori

Kreirati ćemo admin ekran korisnika, na kojem će biti prikazani svi korisnici i njihovi detalji. Također moguće je brisati ili urediti korisnika. Krenimo od izrade akcije i reduktora za ekran korisnika (slike 101 i 102).

```
export const userListReducer = (state = { users: [] }, action) => {
    switch (action.type) {
        case USER_LIST_REQUEST:
            return { loading: true }

        case USER_LIST_SUCCESS:
            return { loading: false, users: action.payload }

        case USER_LIST_FAIL:
            return { loading: false, error: action.payload }

        case USER_LIST_RESET:
            return { users: [] }

        default:
            return state
    }
```

Slika 101 Kod reduktora izlistavanja korisnika

```
export const listUsers = () => async (dispatch, getState) => {
    try {
        dispatch({
            type: USER_LIST_REQUEST
        })

        const {
            userLogin: { userInfo },
        } = getState()

        const config = {
            headers: {
                'Content-type': 'application/json',
                Authorization: `Bearer ${userInfo.token}`
            }
        }

        const { data } = await axios.get(
            `/api/users/`,
            config
        )

        dispatch({
            type: USER_LIST_SUCCESS,
            payload: data
        })


    } catch (error) {
        dispatch({
            type: USER_LIST_FAIL,
            payload: error.response && error.response.data.detail
                ? error.response.data.detail
                : error.message,
        })
    }
}
```

Slika 102 Kod akcija za izlistavanje korisnika

Nakon akcija i reduktora kreiramo datoteku 'UserListScreen.js' u kojoj dizajniramo stranicu (slike 103 i 104).

```
import React, { useState, useEffect } from 'react'
import { LinkContainer } from 'react-router-bootstrap'
import { Table, Button } from 'react-bootstrap'
import { useDispatch, useSelector } from 'react-redux'
import Loader from '../components/Loader'
import Message from '../components/Message'
import { listUsers, deleteUser } from '../actions/userActions'

function UserListScreen({ history }) {

    const dispatch = useDispatch()

    const userList = useSelector(state => state.userList)
    const { loading, error, users } = userList

    const userLogin = useSelector(state => state.userLogin)
    const { userInfo } = userLogin

    const userDelete = useSelector(state => state.userDelete)
    const { success: successDelete } = userDelete


    useEffect(() => {
        if (userInfo && userInfo.isAdmin) {
            dispatch(listUsers())
        } else {
            history.push('/login')
        }

    }, [dispatch, history, successDelete, userInfo])


    const deleteHandler = (id) => {

        if (window.confirm('Are you sure you want to delete this user?')) {
            dispatch(deleteUser(id))
        }
    }
```

Slika 103 Kod admin stranica korisnika (1/2)

```
return (
    <div>
        <h1>Korisnici</h1>
        {loading
            ? (<Loader />)
            : error
                ? (<Message variant='danger'>{error}</Message>)
                : (
                    <Table striped bordered hover responsive className='table-sm'>
                        <thead>
                            <tr>
                                <th>ID</th>
                                <th>IME</th>
                                <th>EMAIL</th>
                                <th>ADMIN</th>
                                <th></th>
                            </tr>
                        </thead>

                        <tbody>
                            {users.map(user => (
                                <tr key={user._id}>
                                    <td>{user._id}</td>
                                    <td>{user.name}</td>
                                    <td>{user.email}</td>
                                    <td>{user.isAdmin ? (
                                        <i className='fas fa-check' style={{ color: 'green' }}></i>
                                    ) : (
                                        <i className='fas fa-check' style={{ color: 'red' }}></i>
                                    )}</td>
```

Slika 104 Kod admin stranica korisnika (2/2)

Kako bi dugme za brisanje korisnika bilo upotrebljivo, moramo napraviti pogled u backendu, te pripadajuće reduktore i akcije (slike 105, 106 i 107).

```
@api_view(['DELETE'])
@permission_classes([IsAdminUser])
def deleteUser(request, pk):
    userForDeletion = User.objects.get(id=pk)
    userForDeletion.delete()
    return Response('User was deleted')
```

Slika 105 Kod pogleda za brisanje korisnika

```
export const userDeleteReducer = (state = {}, action) => {
    switch (action.type) {
        case USER_DELETE_REQUEST:
            return { loading: true }

        case USER_DELETE_SUCCESS:
            return { loading: false, success: true }

        case USER_DELETE_FAIL:
            return { loading: false, error: action.payload }

        default:
            return state
    }
```

Slika 106Kod rReduktora za brisanje korisnika

```
export const deleteUser = (id) => async (dispatch, getState) => {
    try {
        dispatch({
            type: USER_DELETE_REQUEST
        })

        const {
            userLogin: { userInfo },
        } = getState()

        const config = {
            headers: {
                'Content-type': 'application/json',
                Authorization: `Bearer ${userInfo.token}`
            }
        }

        const { data } = await axios.delete(
            `/api/users/delete/${id}/`,
            config
        )

        dispatch({
            type: USER_DELETE_SUCCESS,
            payload: data
        })


    } catch (error) {
        dispatch({
            type: USER_DELETE_FAIL,
            payload: error.response && error.response.data.detail
                ? error.response.data.detail
                : error.message,
        })
    }
}
```

Slika 107 Kod akcija za brisanje korisnika

Također moramo imati mogućnost dobiti podatke o korisniku na temelju njegovog ID-a. Za to radimo pogled u backendu (slika 108).

```
@api_view(['GET'])
@permission_classes([IsAdminUser])
def getUserById(request, pk):
    user = User.objects.get(id=pk)
    serializer = UserSerializer(user, many=False)
    return Response(serializer.data)
```

Slika 108 Kod za dohvaćanje korisnika po ID-u

Napraviti ćemo i pogled za ažuriranje korisnika (slika 109).

```python
@api_view(['PUT'])
@permission_classes([IsAuthenticated])
def updateUser(request, pk):
    user = User.objects.get(id=pk)

    data = request.data

    user.first_name = data['name']
    user.username = data['email']
    user.email = data['email']
    user.is_staff = data['isAdmin']

    user.save()

    serializer = UserSerializer(user, many=False)

    return Response(serializer.data)
```

Slika 109 Kod za pogled za ažuriranje korisnika

Kada imamo sve, možemo kreirati ekran s korisničkim podacima i mogućnošću ažuriranja korisnika. Kreiramo 'UserEditScreen.js' i krećemo s oblikovanjem stranice. Kod stranice prikazan je na slikama 110 i 111.

```
import React, { useState, useEffect } from 'react'
import { Link } from 'react-router-dom'
import { Form, Button } from 'react-bootstrap'
import { useDispatch, useSelector } from 'react-redux'
import Loader from '../components/Loader'
import Message from '../components/Message'
import FormContainer from '../components/FormContainer'
import { getUserDetails, updateUser } from '../actions/userActions'
import { USER_UPDATE_RESET } from '../constants/userConstants'

function UserEditScreen({ match, history }) {

    const userId = match.params.id

    const [name, setName] = useState('')
    const [email, setEmail] = useState('')
    const [isAdmin, setIsAdmin] = useState(false)

    const dispatch = useDispatch()

    const userDetails = useSelector(state => state.userDetails)
    const { error, loading, user } = userDetails

    const userUpdate = useSelector(state => state.userUpdate)
    const { error: errorUpdate, loading: loadingUpdate, success: successUpdate } = userUpdate

    useEffect(() => {

        if (successUpdate) {
            dispatch({ type: USER_UPDATE_RESET })
            history.push('/admin/userlist')
        } else {

            if (!user.name || user._id !== Number(userId)) {
                dispatch(getUserDetails(userId))
            } else {
                setName(user.name)
                setEmail(user.email)
                setIsAdmin(user.isAdmin)
            }
        }

    }, [user, userId, successUpdate, history])

    const submitHandler = (e) => {
        e.preventDefault()
        dispatch(updateUser({ _id: user._id, name, email, isAdmin }))
    }
```

Slika 110 Kod ekrana uređivanja profila (1/2)

```
return (
    <div>
        <Link to='/admin/userlist'>
            Natrag
        </Link>

        <FormContainer>
            <h1>Uredite korisnika</h1>
            {loadingUpdate && <Loader />}
            {errorUpdate && <Message variant='danger'>{errorUpdate}</Message>}

            {loading ? <Loader /> : error ? <Message variant='danger'>{error}</Message>
                : (
                    <Form onSubmit={submitHandler}>

                        <Form.Group controlId='name'>
                            <Form.Label>Ime</Form.Label>
                            <Form.Control

                                type='name'
                                placeholder='Ime...'
                                value={name}
                                onChange={(e) => setName(e.target.value)}
                            >
                            </Form.Control>
                        </Form.Group>

                        <Form.Group controlId='email'>
                            <Form.Label>Email Adresa</Form.Label>
```

Slika 111 Kod ekrana uređivanja profila (2/2)

3.9.2 Admin ekran proizvoda, akcije i reduktori

Izrada admin prozora proizvoda slična je kao i za korisnike. Ponavljamo iste postupke, s promijenjenim pojedinostima. Započinjemo reduktorima i akcijama, a na kraju oblikujemo i sam ekran. Uz elemente u frontendu, potrebno je dodati i poglede u backendu. Nakon toga radimo ekran za kreaciju novog proizvoda te ekran za uređivanje proizvoda. Proces izrade admin ekrana prikazan je slikama 112 do 116.

```
export const deleteProduct = (id) => async (dispatch, getState) => {
    try {
        dispatch({
            type: PRODUCT_DELETE_REQUEST
        })

        const {
            userLogin: { userInfo },
        } = getState()

        const config = {
            headers: {
                'Content-type': 'application/json',
                Authorization: `Bearer ${userInfo.token}`
            }
        }

        const { data } = await axios.delete(
            `/api/products/delete/${id}/`,
            config
        )

        dispatch({
            type: PRODUCT_DELETE_SUCCESS,
        })


    } catch (error) {
        dispatch({
            type: PRODUCT_DELETE_FAIL,
            payload: error.response && error.response.data.detail
                ? error.response.data.detail
                : error.message,
        })
    }
}
```

Slika 112 Kod akcija za brisanje proizvoda

```
export const productDeleteReducer = (state = {}, action) => {
    switch (action.type) {
        case PRODUCT_DELETE_REQUEST:
            return { loading: true }

        case PRODUCT_DELETE_SUCCESS:
            return { loading: false, success: true }

        case PRODUCT_DELETE_FAIL:
            return { loading: false, error: action.payload }

        default:
            return state
    }
}
```

Slika 113 Kod reduktora za brisanje proizvoda

```python
@api_view(['DELETE'])
@permission_classes([IsAdminUser])
def deleteProduct(request, pk):
    product = Product.objects.get(_id=pk)
    product.delete()
    return Response('Proizvod obrisan.')
```

Slika 114 Kod pogled za brisanje proizvoda

```javascript
import React, { useState, useEffect } from 'react'
import { LinkContainer } from 'react-router-bootstrap'
import { Table, Button, Row, Col } from 'react-bootstrap'
import { useDispatch, useSelector } from 'react-redux'
import Loader from '../components/Loader'
import Message from '../components/Message'
import Paginate from '../components/Paginate'
import { listProducts, deleteProduct, createProduct } from '../actions/productActions'
import { PRODUCT_CREATE_RESET } from '../constants/productConstants'

function ProductListScreen({ history, match }) {

    const dispatch = useDispatch()

    const productList = useSelector(state => state.productList)
    const { loading, error, products, pages, page } = productList

    const productDelete = useSelector(state => state.productDelete)
    const { loading: loadingDelete, error: errorDelete, success: successDelete } = productDelete

    const productCreate = useSelector(state => state.productCreate)
    const { loading: loadingCreate, error: errorCreate, success: successCreate, product: createdProduct } = productCreate


    const userLogin = useSelector(state => state.userLogin)
    const { userInfo } = userLogin

    let keyword = history.location.search
    useEffect(() => {
        dispatch({ type: PRODUCT_CREATE_RESET })

        if (!userInfo.isAdmin) {
            history.push('/login')
        }

        if (successCreate) {
            history.push(`/admin/product/${createdProduct._id}/edit`)
        } else {
            dispatch(listProducts(keyword))
        }

    }, [dispatch, history, userInfo, successDelete, successCreate, createdProduct, keyword])


    const deleteHandler = (id) => {

        if (window.confirm('Jeste li sigurni da želite obirsati ovaj proizvod?')) {
            dispatch(deleteProduct(id))
        }
```

Slika 115 Kod stranica za prikaz proizvoda

```
import React, { useState, useEffect } from 'react'
import axios from 'axios'
import { Link } from 'react-router-dom'
import { Form, Button } from 'react-bootstrap'
import { useDispatch, useSelector } from 'react-redux'
import Loader from '../components/Loader'
import Message from '../components/Message'
import FormContainer from '../components/FormContainer'
import { listProductDetails, updateProduct } from '../actions/productActions'
import { PRODUCT_UPDATE_RESET } from '../constants/productConstants'


function ProductEditScreen({ match, history }) {

    const productId = match.params.id

    const [name, setName] = useState('')
    const [price, setPrice] = useState(0)
    const [image, setImage] = useState('')
    const [brand, setBrand] = useState('')
    const [category, setCategory] = useState('')
    const [countInStock, setCountInStock] = useState(0)
    const [description, setDescription] = useState('')
    const [uploading, setUploading] = useState(false)

    const dispatch = useDispatch()

    const productDetails = useSelector(state => state.productDetails)
    const { error, loading, product } = productDetails

    const productUpdate = useSelector(state => state.productUpdate)
    const { error: errorUpdate, loading: loadingUpdate, success: successUpdate } = productUpdate


    useEffect(() => {

        if (successUpdate) {
            dispatch({ type: PRODUCT_UPDATE_RESET })
            history.push('/admin/productlist')
        } else {
            if (!product.name || product._id !== Number(productId)) {
                dispatch(listProductDetails(productId))
            } else {
                setName(product.name)
                setPrice(product.price)
                setImage(product.image)
                setBrand(product.brand)
                setCategory(product.category)
```

Slika 116 Kod stranica za uređivanje proizvoda

3.10 *Ostale funkcionalnosti*

3.10.1 Recenzije proizvoda

Za izradu recenzija proizvoda, potrebno je napraviti *endpoint* u backendu (slika 117), te akcije i reduktere u frontendu (slike 118 i 119). Nakon toga recenzije proizvoda moramo uključiti u željene stranice.

```python
@api_view(['POST'])
@permission_classes([IsAuthenticated])
def createProductReview(request, pk):
    user = request.user
    product = Product.objects.get(_id=pk)
    data = request.data


    alreadyExists = product.review_set.filter(user=user).exists()
    if alreadyExists:
        content = {'detail': 'Recenzija za ovaj proizvod je već dodana!'}
        return Response(content, status=status.HTTP_400_BAD_REQUEST)


    elif data['rating'] == 0:
        content = {'detail': 'Odaberite ocjenu.'}
        return Response(content, status=status.HTTP_400_BAD_REQUEST)


    else:
        review = Review.objects.create(
            user=user,
            product=product,
            name=user.first_name,
            rating=data['rating'],
            comment=data['comment'],
        )

        reviews = product.review_set.all()
        product.numReviews = len(reviews)

        total = 0
        for i in reviews:
            total += i.rating

        product.rating = total / len(reviews)
        product.save()

        return Response('Recenzija dodana.')
```

Slika 117 Kod backend pogled za dodavanje recenzije

```
export const productReviewCreateReducer = (state = {}, action) => {
    switch (action.type) {
        case PRODUCT_CREATE_REVIEW_REQUEST:
            return { loading: true }

        case PRODUCT_CREATE_REVIEW_SUCCESS:
            return { loading: false, success: true, }

        case PRODUCT_CREATE_REVIEW_FAIL:
            return { loading: false, error: action.payload }

        case PRODUCT_CREATE_REVIEW_RESET:
            return {}

        default:
            return state
    }
```

Slika 118 Kod reduktora za dodavanje recenzije

```
export const createProductReview = (productId, review) => async (dispatch, getState) => {
    try {
        dispatch({
            type: PRODUCT_CREATE_REVIEW_REQUEST
        })

        const {
            userLogin: { userInfo },
        } = getState()

        const config = {
            headers: {
                'Content-type': 'application/json',
                Authorization: `Bearer ${userInfo.token}`
            }
        }

        const { data } = await axios.post(
            `/api/products/${productId}/reviews/`,
            review,
            config
        )
        dispatch({
            type: PRODUCT_CREATE_REVIEW_SUCCESS,
            payload: data,
        })



    } catch (error) {
        dispatch({
            type: PRODUCT_CREATE_REVIEW_FAIL,
            payload: error.response && error.response.data.detail
                ? error.response.data.detail
                : error.message,
        })
    }
```

Slika 119 Kod akcija za dodavanje recenzije

3.10.2 Pretraga proizvoda

Kako bi dodali pretraživanje proizvoda, prvo ćemo napraviti komponentu 'SearchBox.js' (slika 120). Kreiranu komponentu uključiti ćemo u zaglavlje aplikacije. Akcije i reduktori te backend *endpoint* već imaju funkcionalnost pretraživanja po ID-u.

```jsx
import React, { useState } from 'react'
import { Button, Form } from 'react-bootstrap'
import { useHistory } from 'react-router-dom'

function SearchBox() {
    const [keyword, setKeyword] = useState('')

    let history = useHistory()

    const submitHandler = (e) => {
        e.preventDefault()
        if (keyword) {
            history.push(`/?keyword=${keyword}&page=1`)
        } else {
            history.push(history.push(history.location.pathname))
        }
    }
    return (
        <Form onSubmit={submitHandler} inline>
            <Form.Control
                type='text'
                name='q'
                onChange={(e) => setKeyword(e.target.value)}
                className='mr-sm-2 ml-sm-5'
                placeholder="Pretraga..."
            ></Form.Control>

            <Button
                type='submit'
                variant='outline-success'
                className='p-2'
            >
                Traži
            </Button>
        </Form>
    )
}

export default SearchBox
```

Slika 120 Kod za komponentu pretraživanja

3.10.3 Paginacija proizvoda

Paginacija proizvoda je prikazivanje određenog broja proizvoda po stranici. Kako bi ostvarili tu funkcionalnost korištena je Django Pagination klasa (slika 121). Za prikaz komponenti korištene su React Bootstrap komponente (slika 122).

```python
try:
    products = paginator.page(page)
except PageNotAnInteger:
    products = paginator.page(1)
except EmptyPage:
    products = paginator.page(paginator.num_pages)

if page == None:
    page = 1

page = int(page)
print('Page:', page)
serializer = ProductSerializer(products, many=True)
return Response({'products': serializer.data, 'page': page, 'pages': paginator.num_pages})
```

Slika 121 Kod Django paginacije

```
        ))}
      </Row>
      <Paginate page={page} pages={pages} keyword={keyword} />
    </div>
```

Slika 122 Kod frontend paginacije

3.10.4„Carousel“ najbolje ocjenjenih proizvoda

Carousel je galerija najbolje ocjenjenih proizvoda koja se prikazuje na početnoj stranici. U carouselu se prikazuje pet najbolje ocjenjenih proizvoda bolje ocjenjenih od 4 zvjezdice. Potrebno je napraviti backend *endpoint* za dobiti najbolje proizvode (slika 123), akcije (slika 125), reduktore (slika 124) i carousel komponentu (slika 126) koju ćemo uključiti u početni ekran.

```
@api_view(['GET'])
def getTopProducts(request):
    products = Product.objects.filter(rating__gte=4).order_by('-rating')[0:5]
    serializer = ProductSerializer(products, many=True)
    return Response(serializer.data)
```

Slika 123 Kod backend endpointa za najbolje proizvode

```
export const productTopRatedReducer = (state = { products: [] }, action) => {
    switch (action.type) {
        case PRODUCT_TOP_REQUEST:
            return { loading: true, products: [] }

        case PRODUCT_TOP_SUCCESS:
            return { loading: false, products: action.payload, }

        case PRODUCT_TOP_FAIL:
            return { loading: false, error: action.payload }

        default:
            return state
    }
```

Slika 124 Kod reduktora najboljih proizvoda

```
export const listTopProducts = () => async (dispatch) => {
    try {
        dispatch({ type: PRODUCT_TOP_REQUEST })

        const { data } = await axios.get(`/api/products/top/`)

        dispatch({
            type: PRODUCT_TOP_SUCCESS,
            payload: data
        })

    } catch (error) {
        dispatch({
            type: PRODUCT_TOP_FAIL,
            payload: error.response && error.response.data.detail
                ? error.response.data.detail
                : error.message,
        })
    }
}
```

Slika 125 Kod akcije najboljih proizvoda

```jsx
import React, { useEffect } from 'react'
import { useDispatch, useSelector } from 'react-redux'
import { Link } from 'react-router-dom'
import { Carousel, Image } from 'react-bootstrap'
import Loader from './Loader'
import Message from './Message'
import { listTopProducts } from '../actions/productActions'

function ProductCarousel() {
    const dispatch = useDispatch()

    const productTopRated = useSelector(state => state.productTopRated)
    const { error, loading, products } = productTopRated

    useEffect(() => {
        dispatch(listTopProducts())
    }, [dispatch])

    return (loading ? <Loader />
        : error
            ? <Message variant='danger'>{error}</Message>
            : (
                <Carousel pause='hover' className='bg-dark'>
                    {products.map(product => (
                        <Carousel.Item key={product._id}>
                            <Link to={`/product/${product._id}`}>
                                <Image src={product.image} alt={product.name} fluid />
                                <Carousel.Caption className='carousel.caption'>
                                    <h4>{product.name} ({product.price} kn)</h4>
                                </Carousel.Caption>
                            </Link>
                        </Carousel.Item>
                    ))}
                </Carousel>
            )
    )
}

export default ProductCarousel
```

Slika 126 Kod za Carousel najboljih proizvoda

3.11 *Postavljanje baze podataka i podizanje aplikacije na Heroku server*

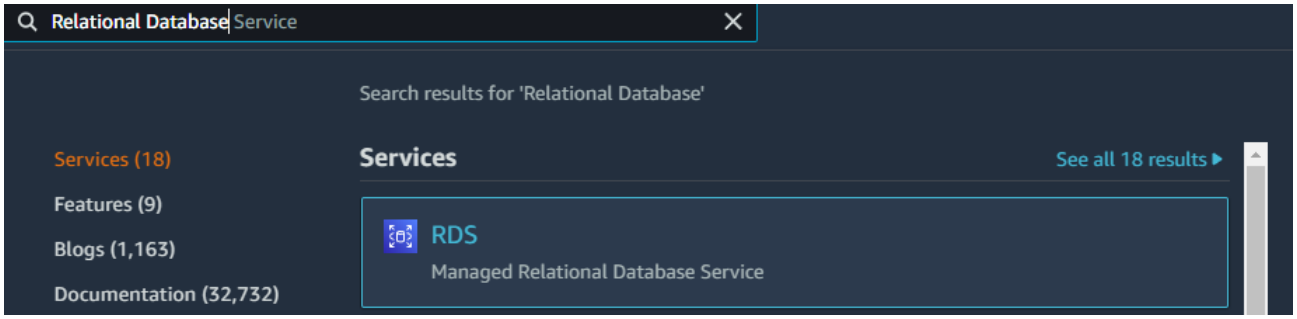
3.11.1 Spajanje React i Django projekta

Kako ne bi morali imati posebno servere za frontend i backend projekta, možemo ih spojiti i staviti na isti server. To radimo tako da frontend direktorij premjestimo u backend. Kada se nalazimo unutar tog direkotrija koristimo naredbu „npm run build“. Izvršavanjem te naredbe kreirati će se folder 'build' u koji je React pokupio i stavio sve statične datoteke iz našeg projekta.

Kada smo frontend pretvorili u statične datoteke, pozicioniramo se unutar backend direktorija i pokrenemo naredbu „python manage.py collectstatics“ kojom Django traži statične datoteke unutar projekta koje će prikazati na svome serveru.

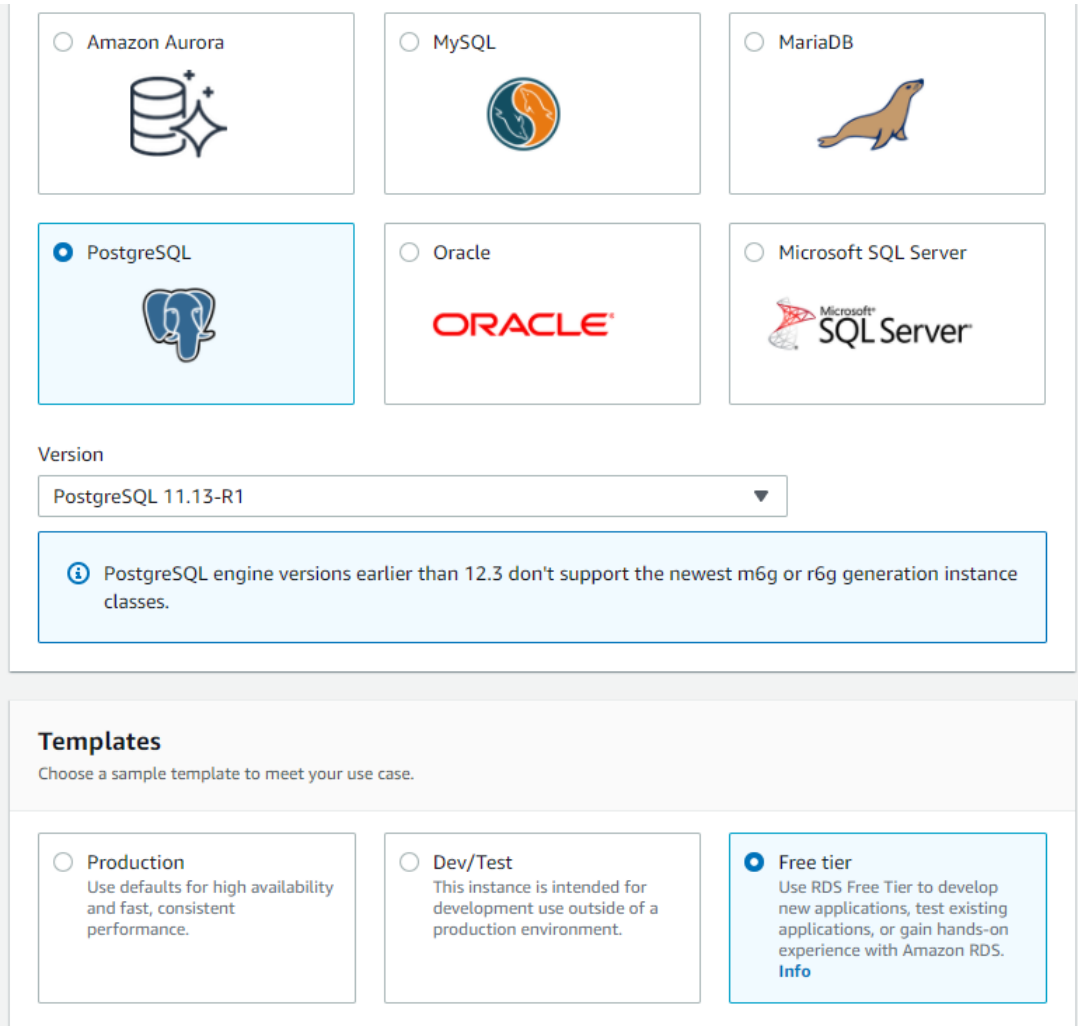
3.11.2 Postavljanje PostgreSQL baze podataka i statičnih podataka

Za bazu podataka aplikacija koristi PostgreSQL koju će nam pružiti Amazon Web Services. Prije početka treba se registrirati na AWS web stranicu. Kada se pozicioniramo u AWS konzoli, tražimo servis ' Relational Database' (slika 127).

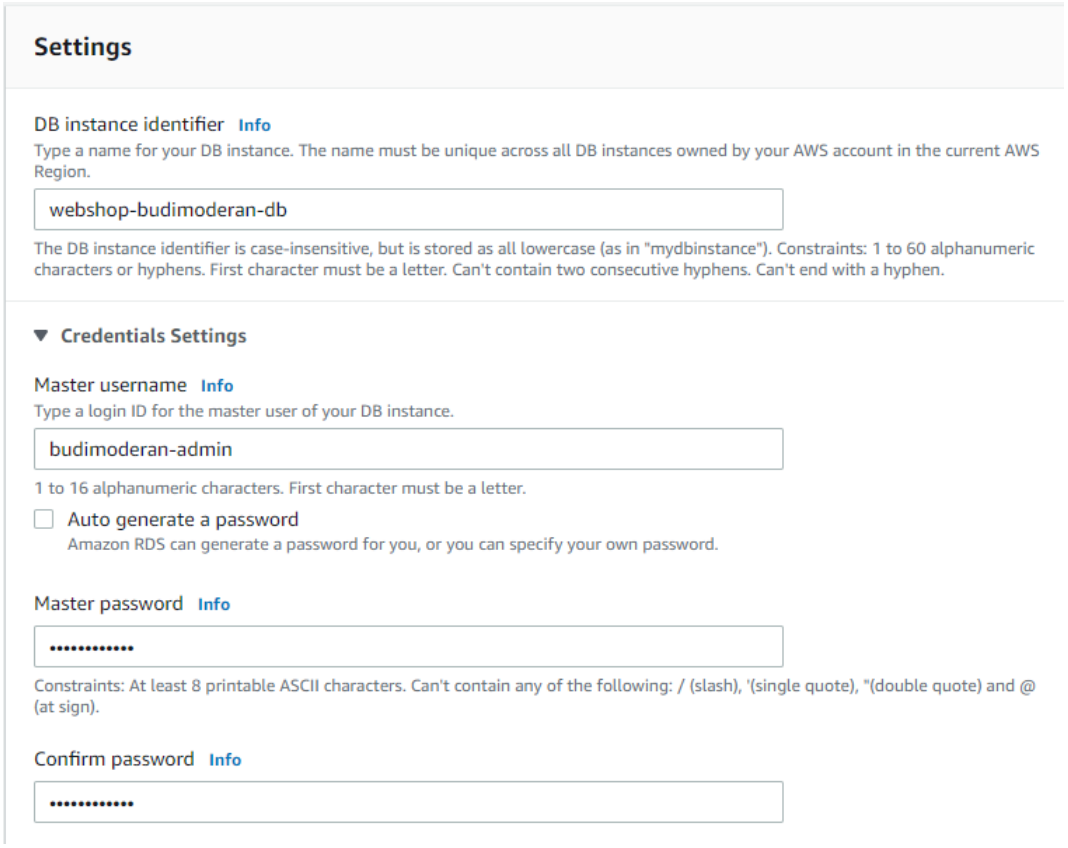


Slika 127 AWS konzola

Kreirati ćemo novu bazu tipa PostgreSQL, izabrati 'Free tier' template te postaviti *username* i *password* baze podataka kao što je prikazano na slikama 128 i 129.



Slika 128 Kreiranje baze podataka (1/2)



Slika 129 Kreiranje baze podataka (2/2)

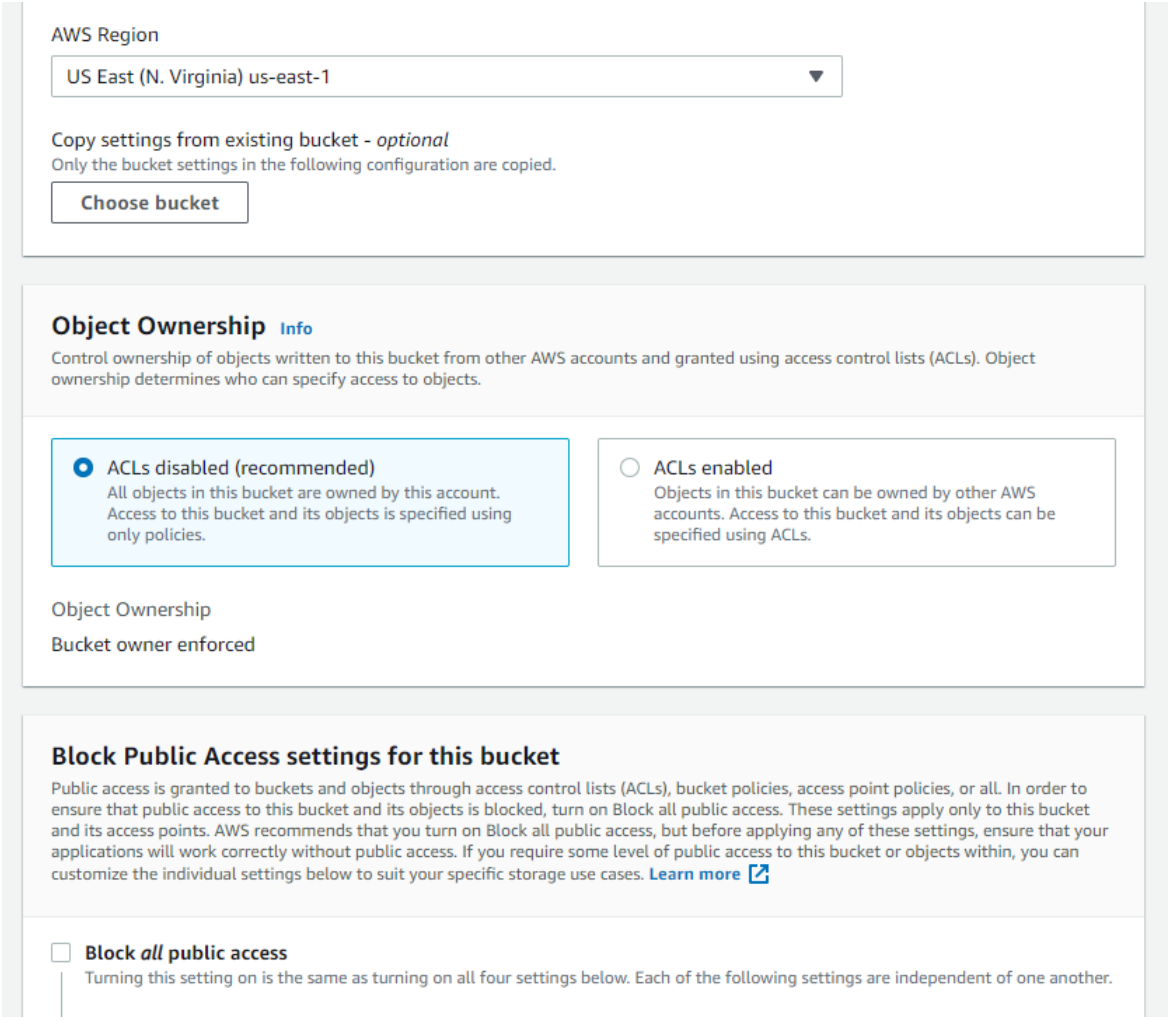
Unutar 'settings.py' datoteke moramo promijeniti konekciju na bazu podataka (slika 130).

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'mpilepicwebshop',
        'USER': 'mpilepic',
        'PASSWORD': os.environ.get('DB_PASS'),
        'HOST': os.environ.get('HOST'),
        'PORT': '5432'
    }
}
```

Slika 130 Kod za konekcija na bazu podataka

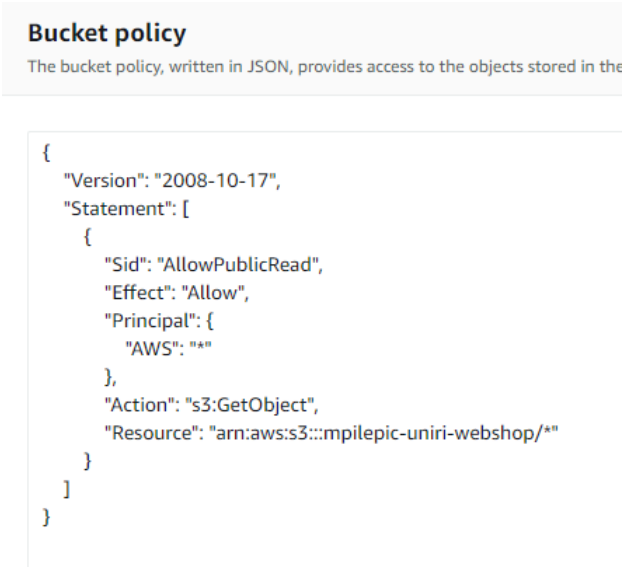
Kako bi Django znao za naše modele podataka koje smo kreirali i preslikao ih u novokreiranu bazu podataka završavamo naredbe „python manage.py makemigrations“ i „python manage.py migrate“. Kako bi kreirali admin korisnika, koristimo naredbu „python manage.py createsuperuser“. Podatke o bazi podataka, password i host, spremiti ćemo u environment variable Heroku alata iz sigurnosnih razloga.

Statične podatke, kao slike, ne možemo držati u PostgreSQL bazi, zato ćemo iskoristiti AWS S3 servis. Kako bi kreirali S3 bucket u AWS konzoli tražimo S3 te klikom na 'Create bucket' dolazimo na ekran za kreiranje bucketa. Dodijelimo mu ime i obratimo pozornost da 'Block all public access' nije označeno (slika 131).



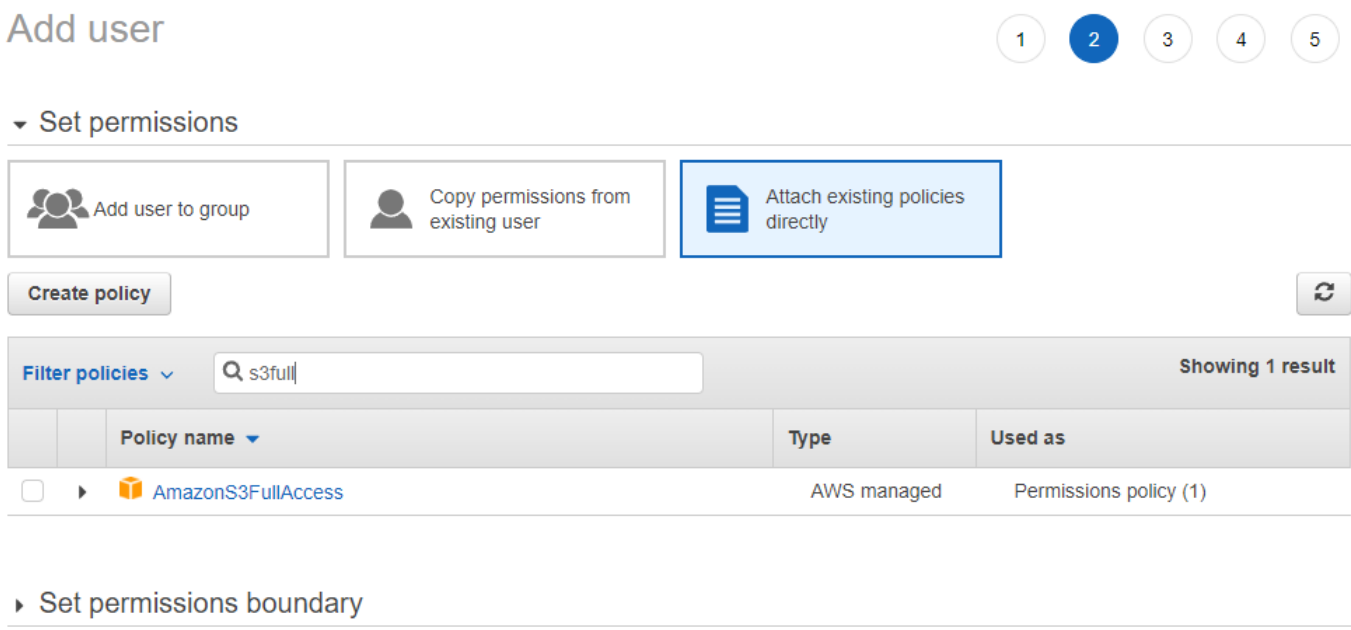
Slika 131 AWS S3

Unutar permisija *bucketa* moramo odrediti tko što može raditi s podacima. Pošto će nam on služiti želimo da svi vide naše podatke, ali ne i da ih mogu uređivati. To možemo definirati s 'bucket policy' opcijom kao na slici 132.



Slika 132 Kod S3 bucket policy-a

Unutar AWS-a moramo napraviti korisnika kojega ćemo dodijeliti našoj aplikaciji kako bi imao pristup bazi podataka. Unutar AWS konzole pronađemo IAM servis i klikom na 'Add user' u nekoliko klikova kreiramo novog korisnika. Prilikom kreacije korisnika potrebno je paziti da označimo da želimo pristupne ključeve za programski pristup, te da *user* ima pravo pristupa S3 servisu kao što je prikazano na slici 133.



Slika 133 Kreacija AWS korisnika

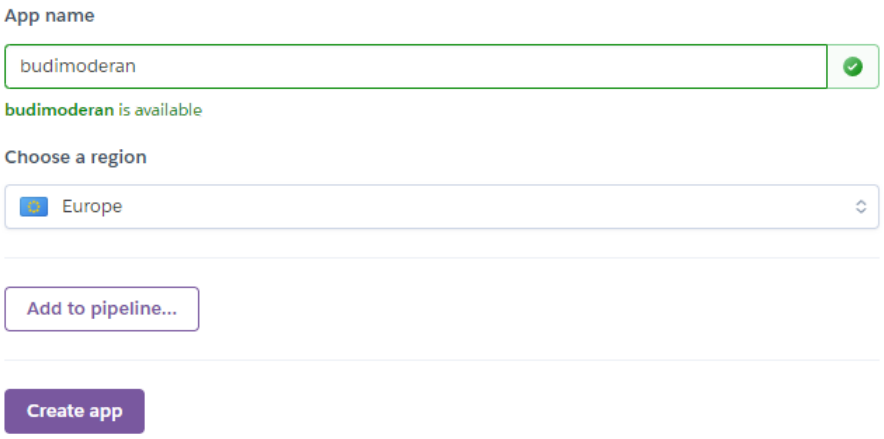
Kada smo kreirali *usera*, potrebno je pristupne podatke dati Djangu i usmjeriti ga da koristi S3 za spremanje statičnih datoteka. Slika 134 prikazuje korištenje *environment* variabli za prosljeđivanje pristupnih podataka.

```
AWS_QUERYSTRING_AUTH = False
DEFAULT_FILE_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'

AWS_ACCESS_KEY_ID = os.environ.get('AWS_ACCESS_KEY_ID')
AWS_SECRET_ACCESS_KEY = os.environ.get('AWS_SECRET_ACCESS_KEY')

AWS_STORAGE_BUCKET_NAME = 'mpilepic-uniri-webshop'
```

Slika 134 Kod za postavljanje S3 u Django okviru

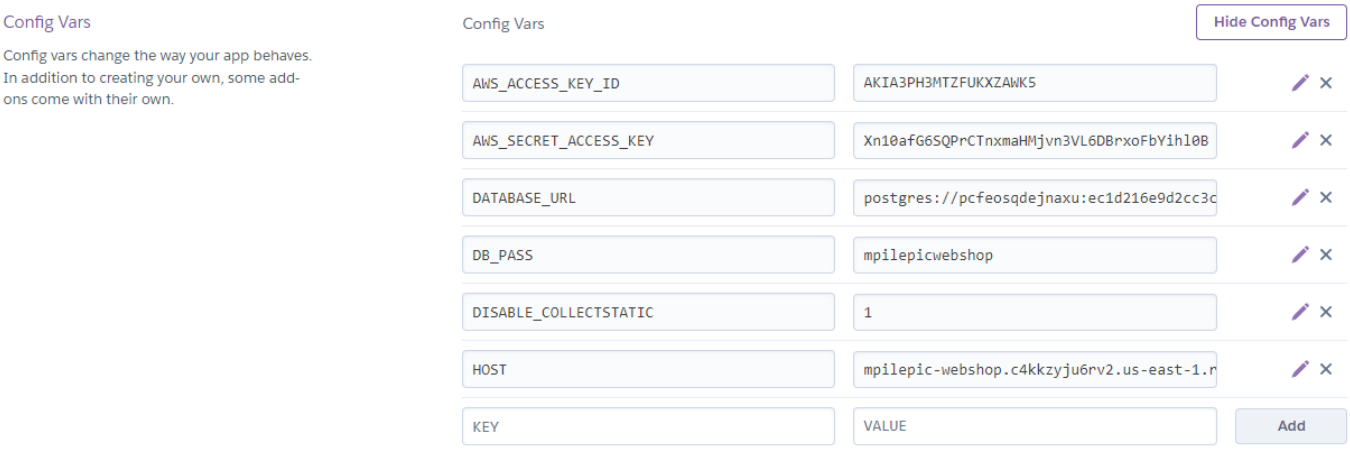
3.11.3 Postavljanje aplikacije na Heroku server

Django i Heroku nam omogućuju izrazito jednostavno postavljanje aplikacije na server. Na Heroku web stranici kreiramo novu aplikaciju (slika 135).



Slika 135 Heroku kreiranje aplikacije

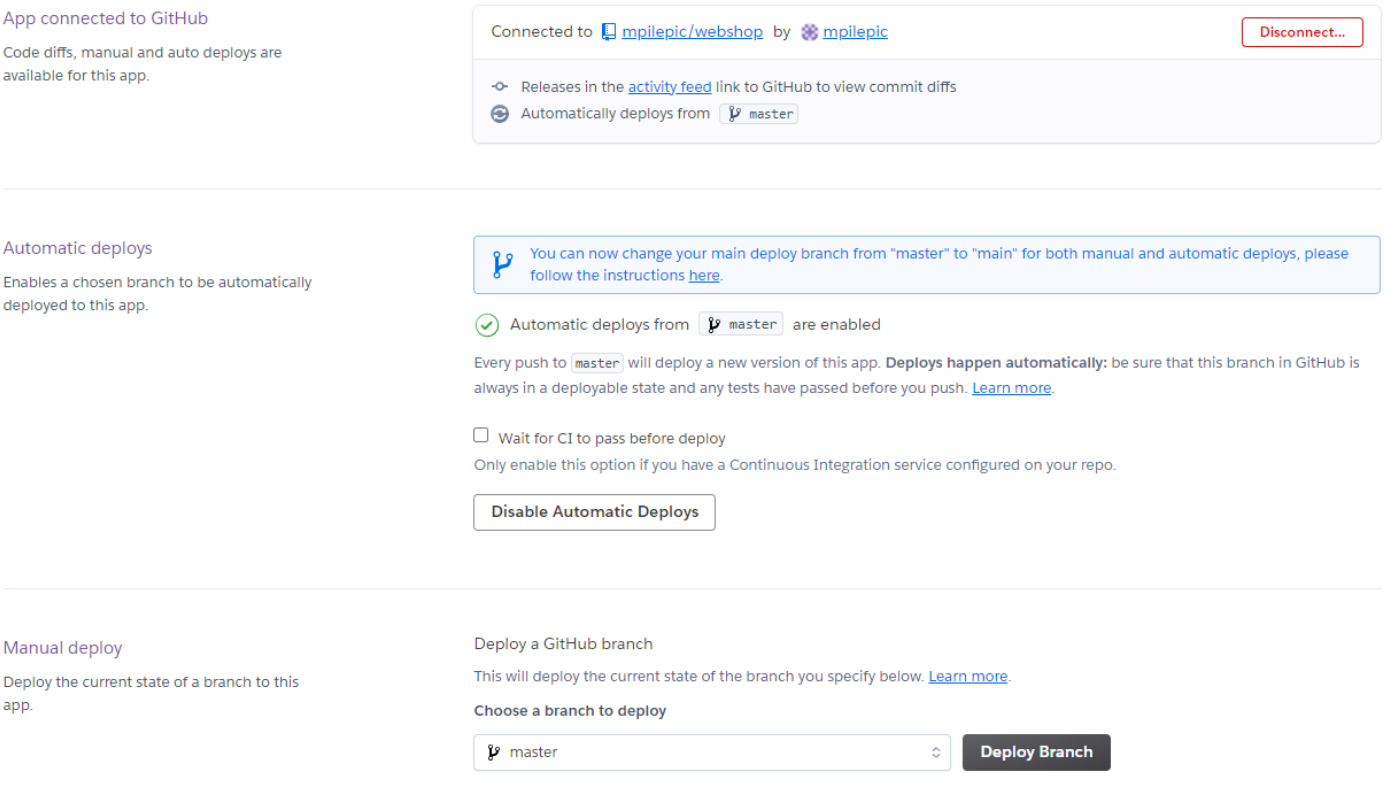
Kada smo u aplikaciji, namjestimo envirnment variable koje Django koristi kao na slici 136.



Slika 136 Heroku environment variable

Kako bi aplikaciju mogli postaviti na Heroku, moramo ju prvo staviti na GitHub i povezati GitHub račun s Heroku računom. Nakon registracije na GitHub i kreacije repozitorija, izvršavanjem naredbe 'git init' projekt postaje git projekt. 'git add . ' dodaje sve datoteke u git repozitorij, 'git commit' dodaje promjene indexu i na kraju 'git push' šalje promjene u repozitorij.

Kada smo aplikaciju povezali s GitHub računom, izaberemo koji repozitorij želimo staviti na naš server i pritisnemo dugme 'Deploy branch' (slika 137).

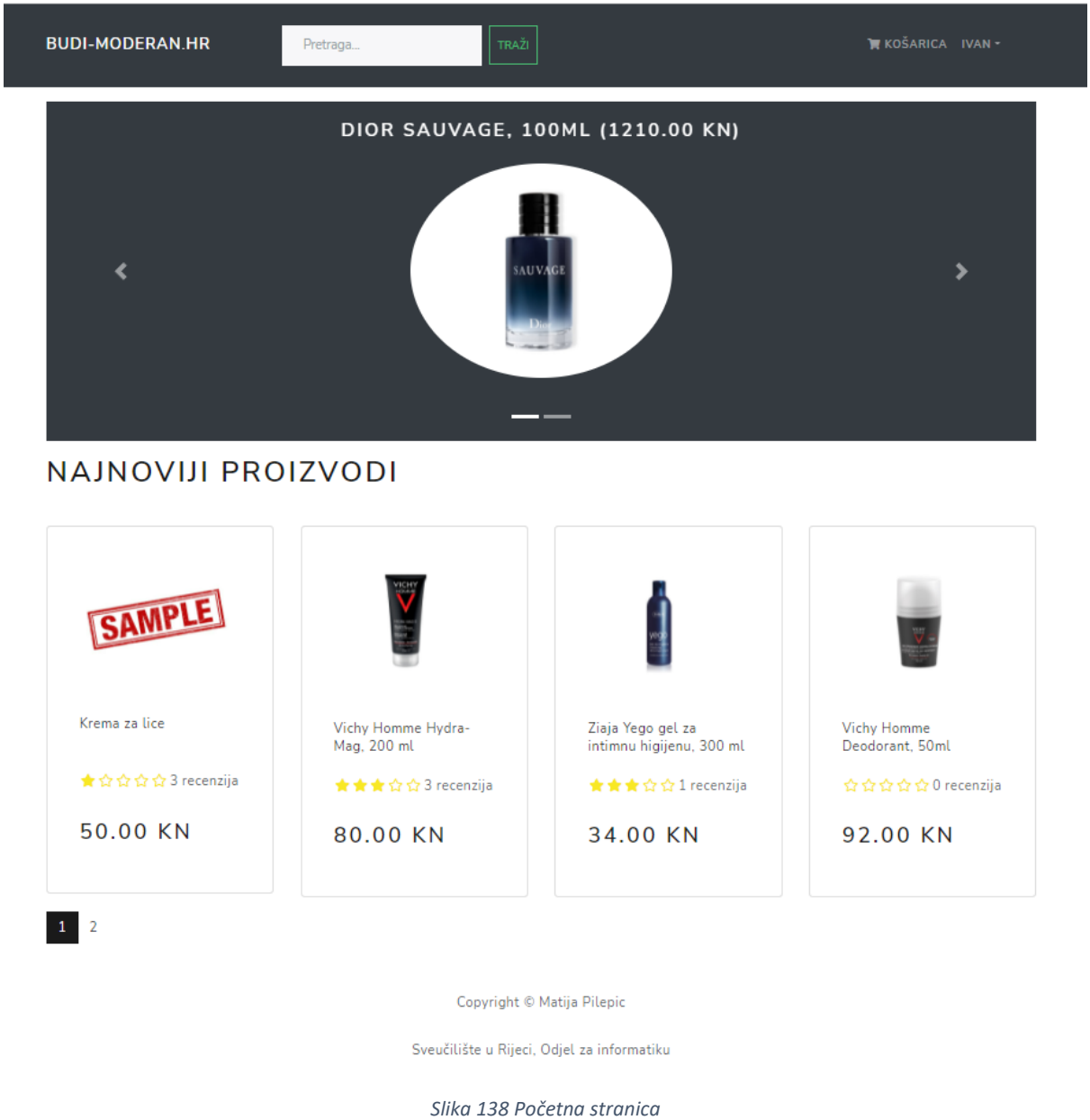


Slika 137 Heroku deploy

3.12 *Prikaz aplikacije*

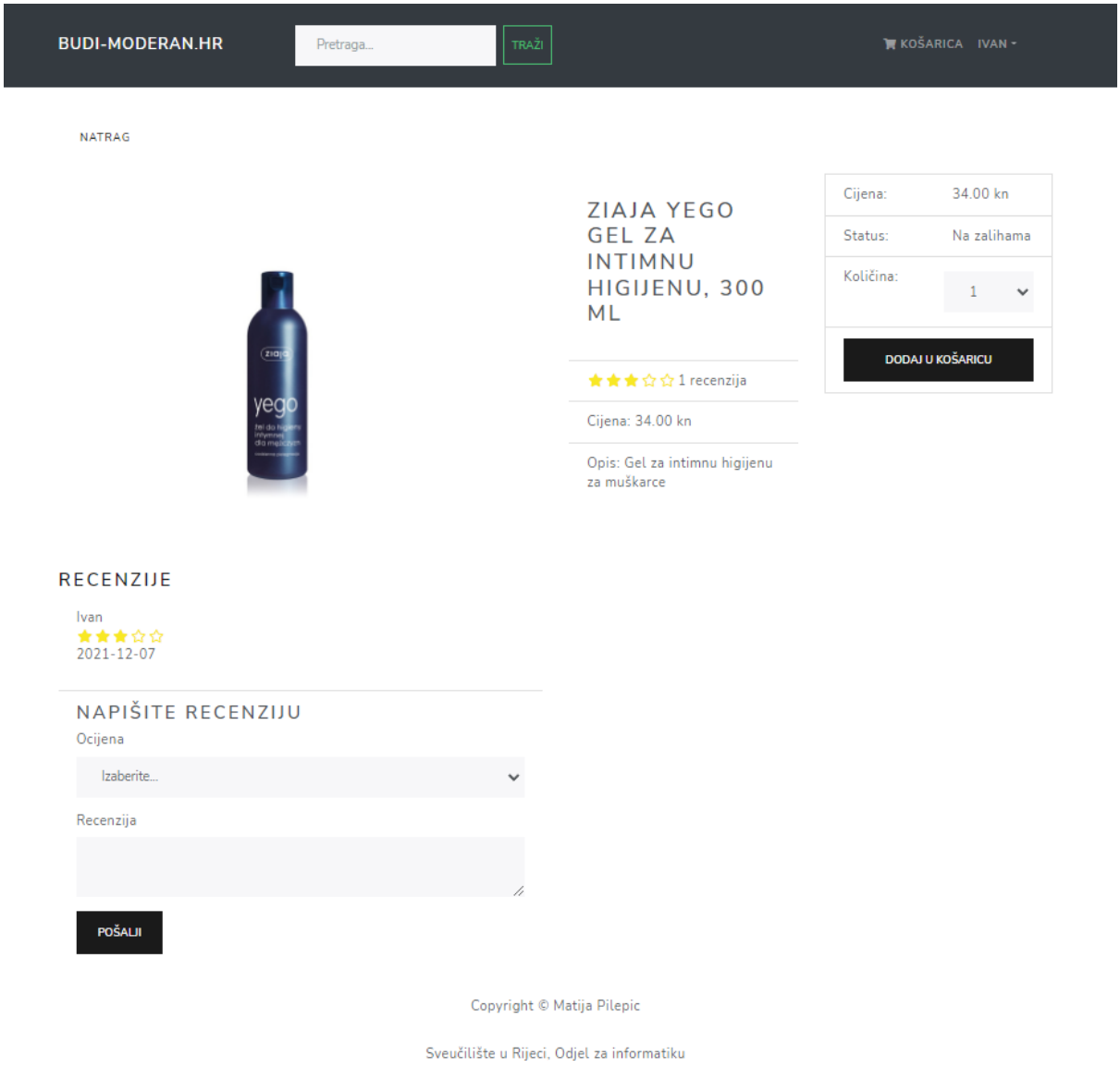
3.12.1 Prikaz iz perspektive korisnika

Kada smo u aplikaciju prijavljeni kao korisnik, na početnoj stranici vidljiva je Header komponenta koja sadrži naslov stranice, pretraživanje, koršaricu i padajući meni profila. U nastavku je vidljiv *Courosel* najbolje ocjenjenih proizvoda. Ispod *Courosela* izlistani su svi proizvodi, paginirani po 4 proizvoda po stranici (slika 138).



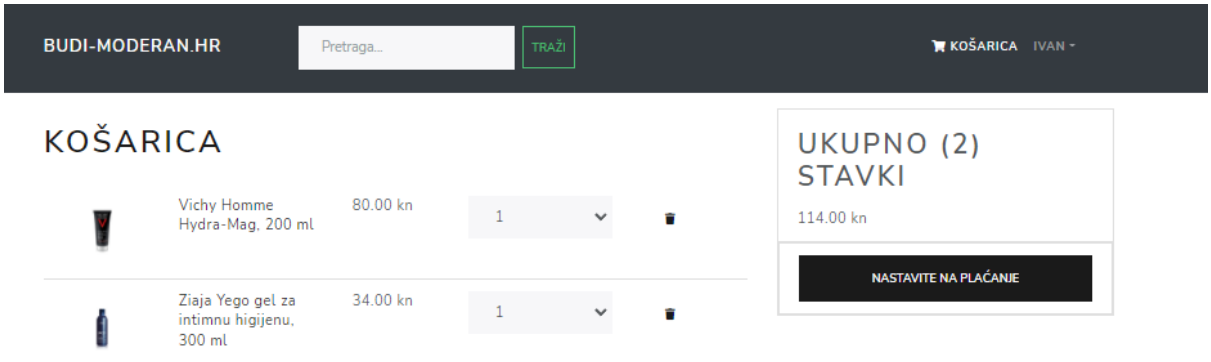
Slika 138 Početna stranica

Klik na neki od proizvoda vodi nas na detaljnu stranicu o proizvodu. S lijeve stranice vidljiva je slika proizvoda, dosada unesene recenzije i forma za unos nove recenzije. Vidljiva je ocjena, cijena, opis proizvoda i stanje zaliha. Također, tu je i dugme za dodavanje proizvoda u košaricu (slika 139).



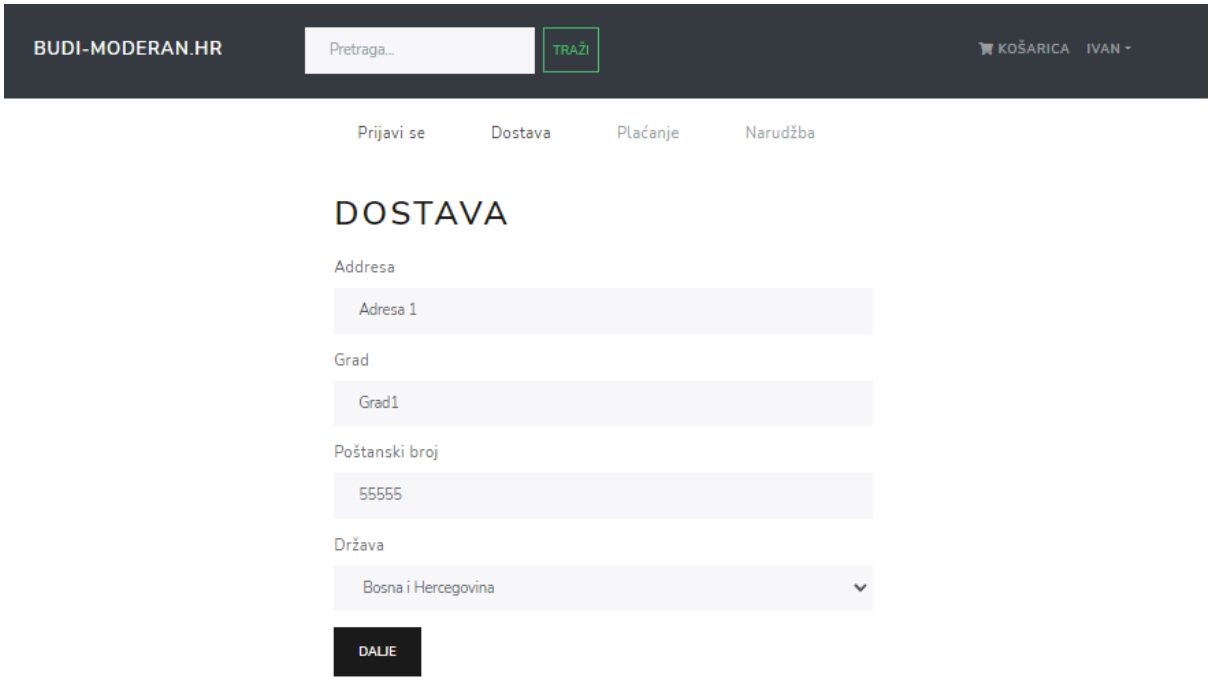
Slika 139 Stranica proizvoda

Slika 140 prikazuje košaricu u kojoj su prikazani svi dodani proizvodi i ukupna cijena.



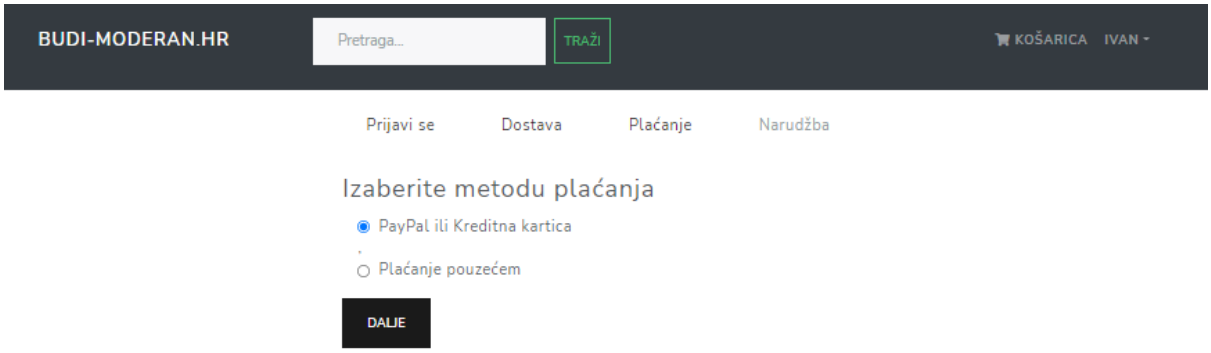
Slika 140 Košarica

Kada nastavimo na plaćanje, preusmjereni smo na ekran za upis podataka o dostavi. Ispod footer komponente, nalazi se komponenta *progresa Checkout procesa.* Odabir države je napravljen kao padajući izbornik (slika 141).



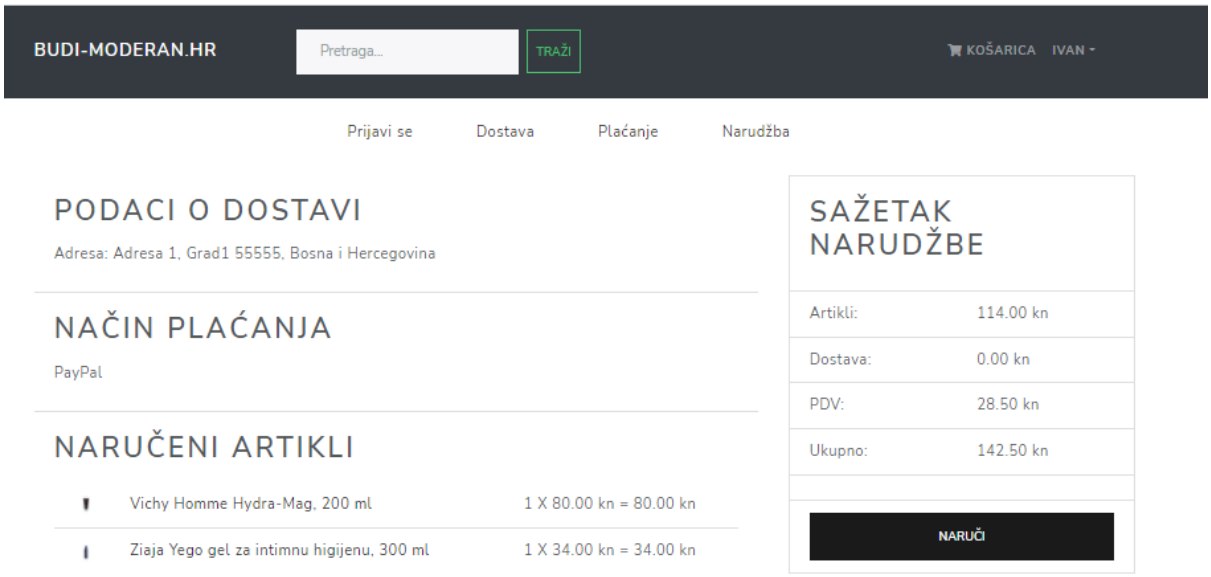
Slika 141 Podaci o dostavi

Kao metode plaćanja implementirane su PayPal i plaćanje pouzećem, a izbor se vrši pomoću 'radio buttona' (slika 142).



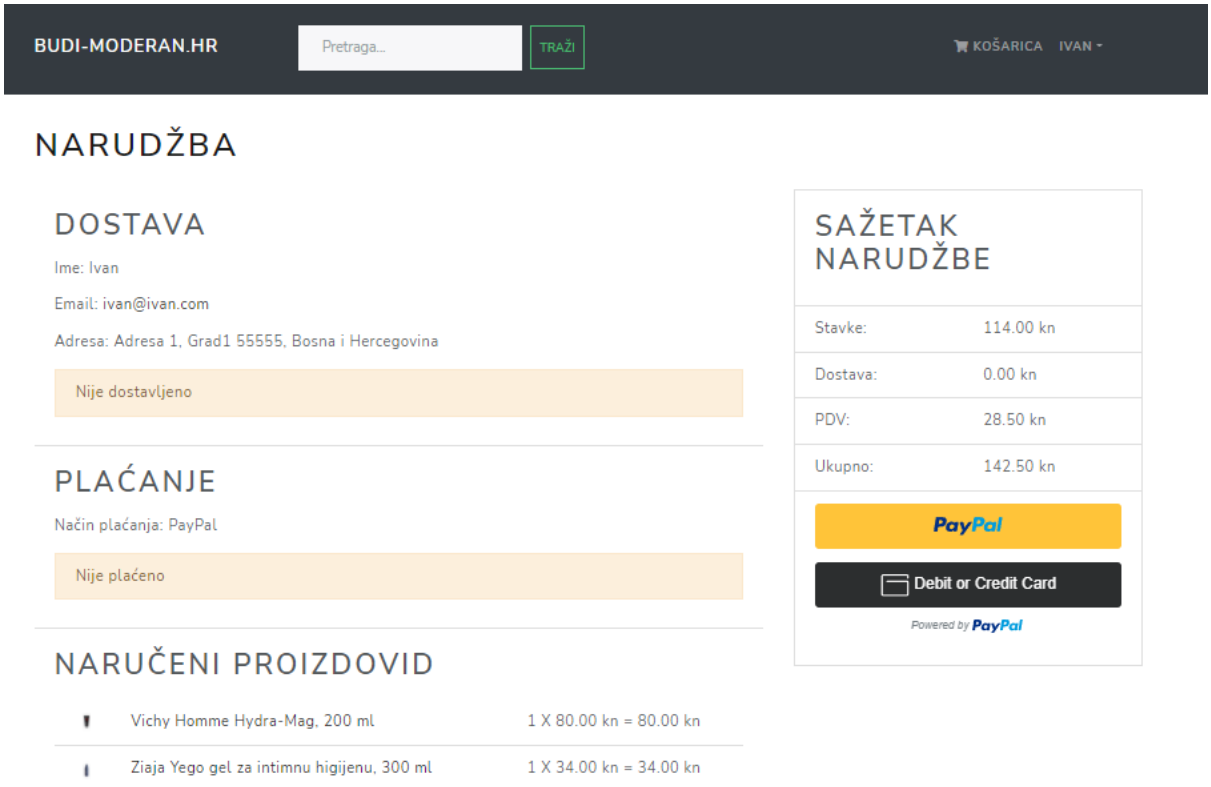
Slika 142 Odabir metode plaćanja

Kada odaberemo način plaćanja, vidimo sažetak narudžbe. Na njemu su prikazani podaci o dostavi koje smo unijeli, odabrani način plaćanja i naručeni artikli. Također vidimo ukupnu cijenu raščlanjenu na cijenu artikla, dostave, PDV i ukupno. U svakom trenutku se možemo vratiti na neki od prethodnih koraka i napraviti izmjene ako je potrebno (slika 143).



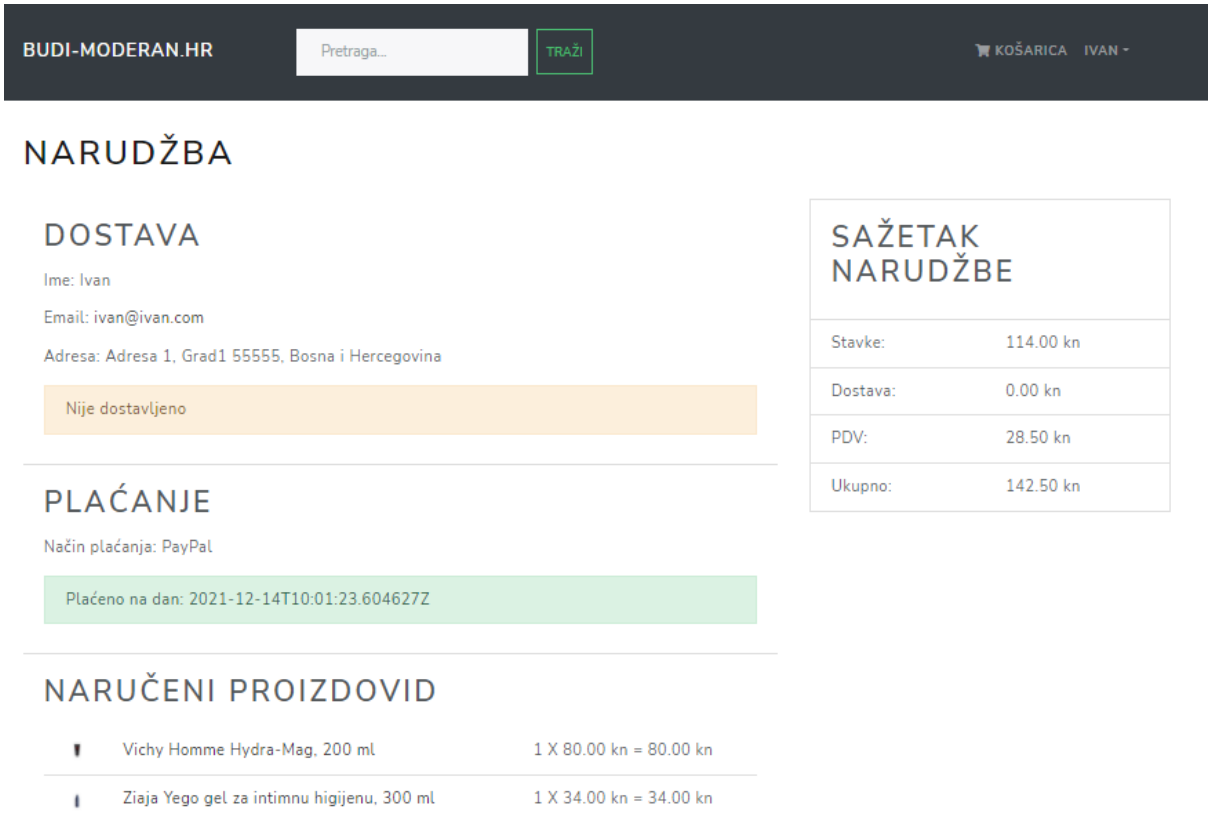
Slika 143 Sažetak narudžbe

Klikom na 'Naruči' naša narudžba je spremljena i spremna za plaćanje. Vidimo trenutni status dostave i status plaćanja te popis naručenih proizvoda (slika 144).



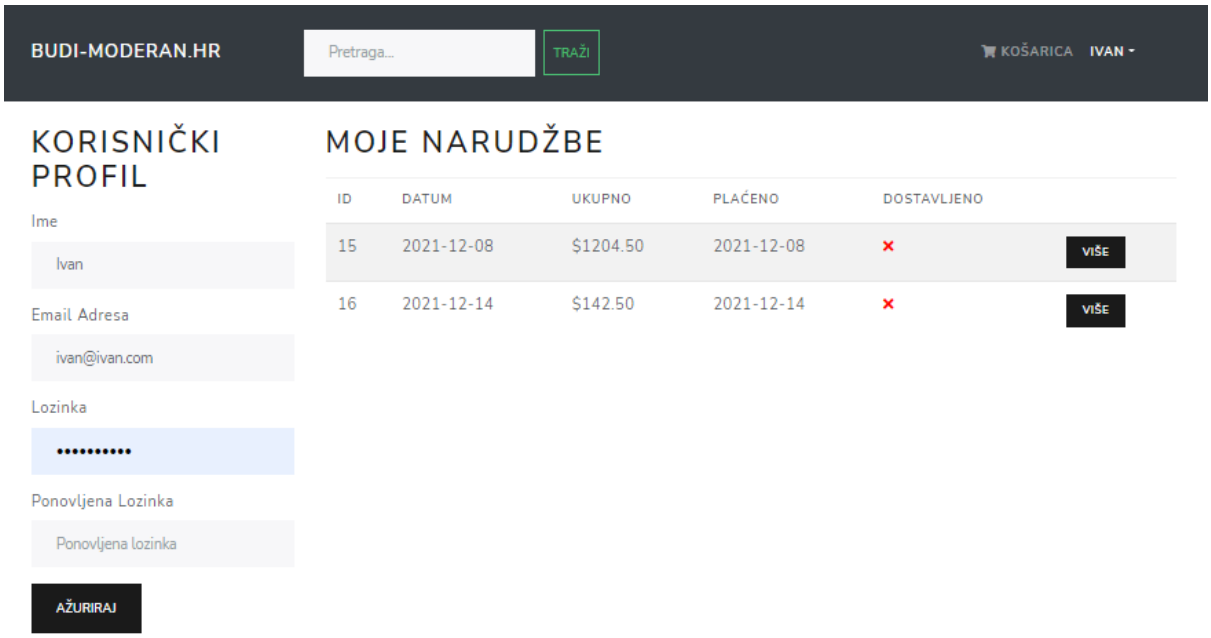
Slika 144 Narudžba

Na slici 145 prikazana je promjena na stranici narudžbe nakon što je ista plaćena. Status plaćanja je promijenjen u plaćeno, a PayPal funkcionalnost više nije vidljiva.



Slika 145 Narudžba plaćena

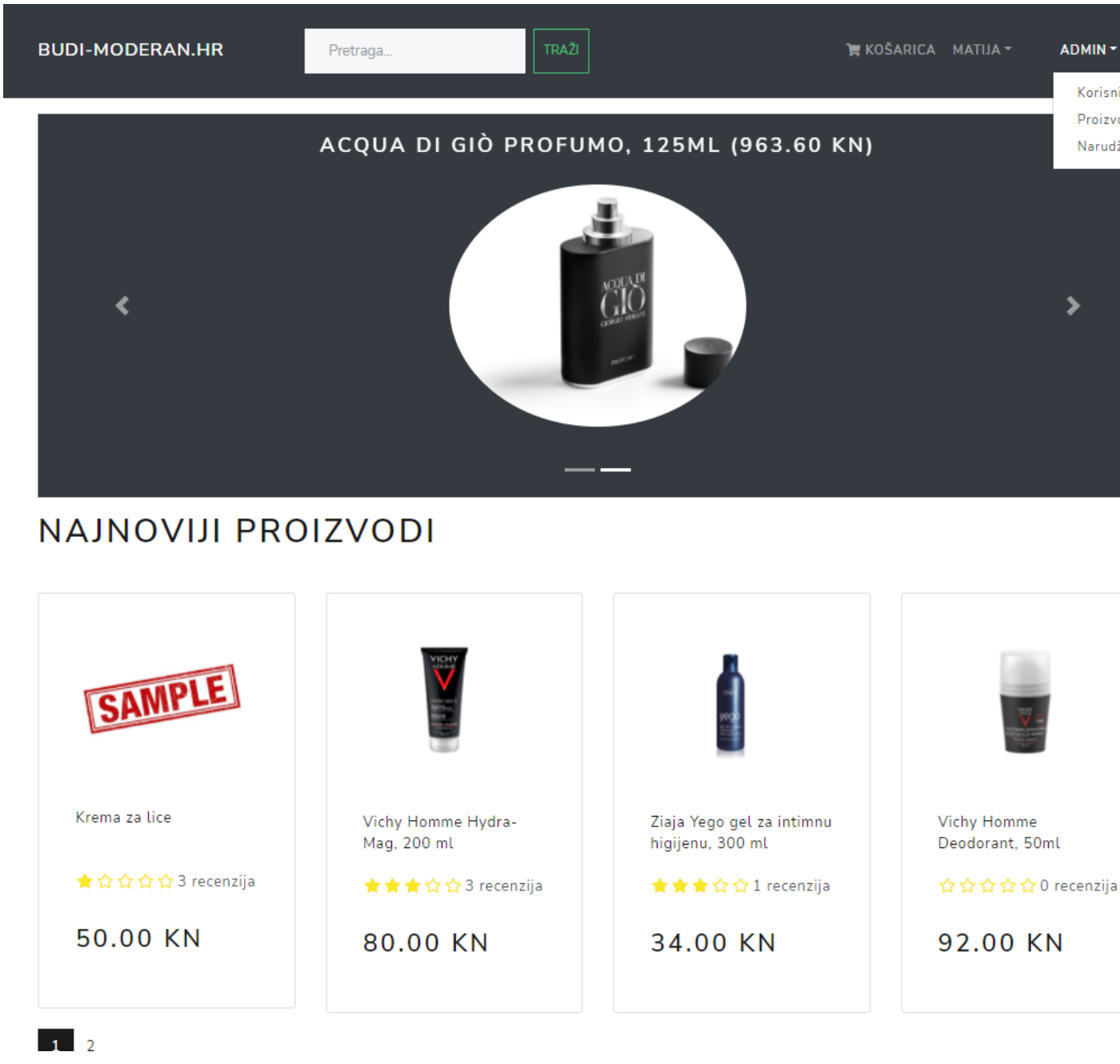
Na stranici profila, na lijevoj strani korisnik ima mogućnost urediti svoj profil. S desne strane vidljive su sve narudžbe korisnika, njihov status i dugme za pregled više pojedinosti (slika 146).



Slika 146 Stranica profila

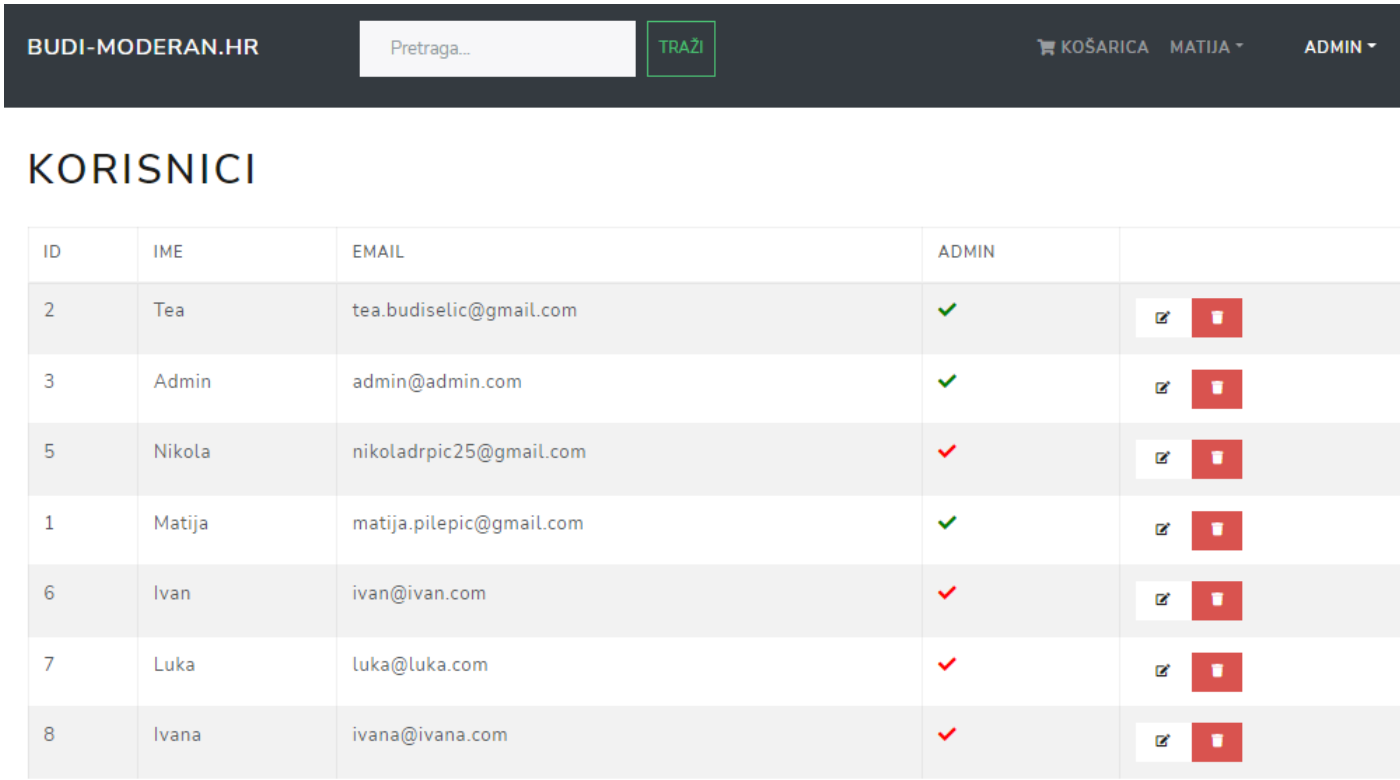
3.12.2 Prikaz iz perspektive admina

Kada se prijavimo kao admin, imamo sve funkcionalnosti korisnika i dodatne funkcionalnosti admina. Na početnoj stranici vidljiv nam je padajući izbornik 'admin' unutar kojega su stranice za pregled i uređivanje korisnika, proizvoda i narudžbi (slika 147).



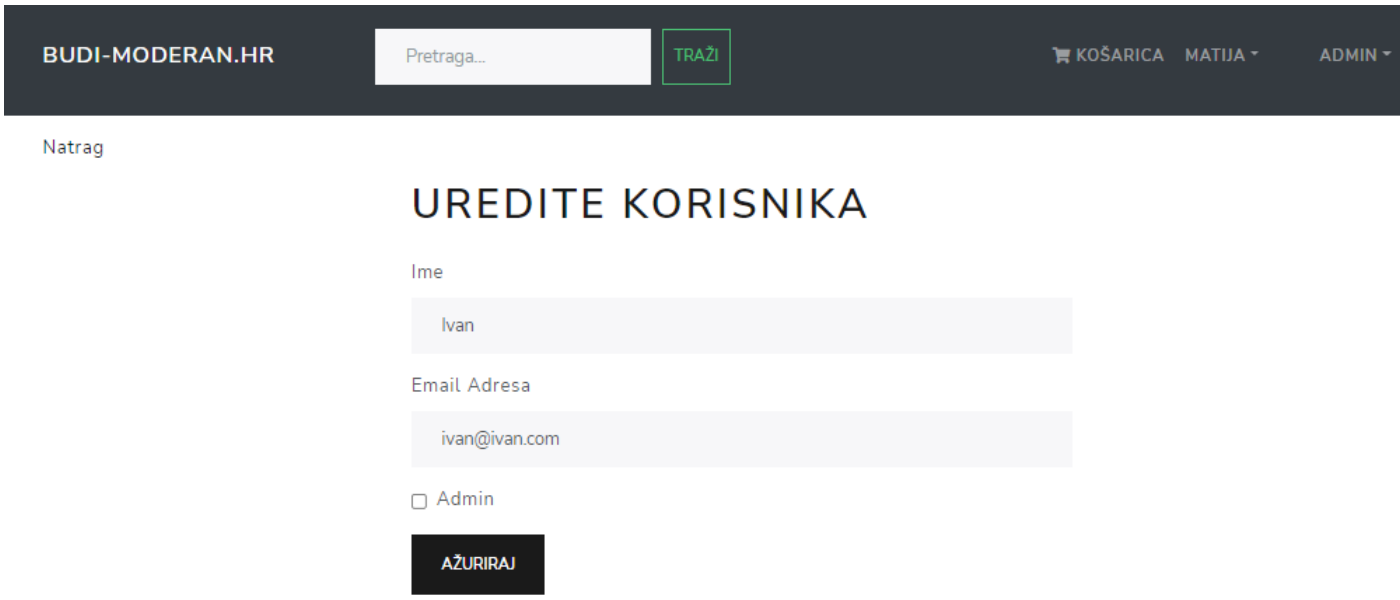
Slika 147 Admin padajući meni

Stranica za prikaz korisnika ispisuje nam sve registrirane korisnike naše aplikacije, prikazuje njihov ID, ime, email i status jesu li su admini. Također imamo mogućnost brisanja i uređivanja korisnika (slika 148).



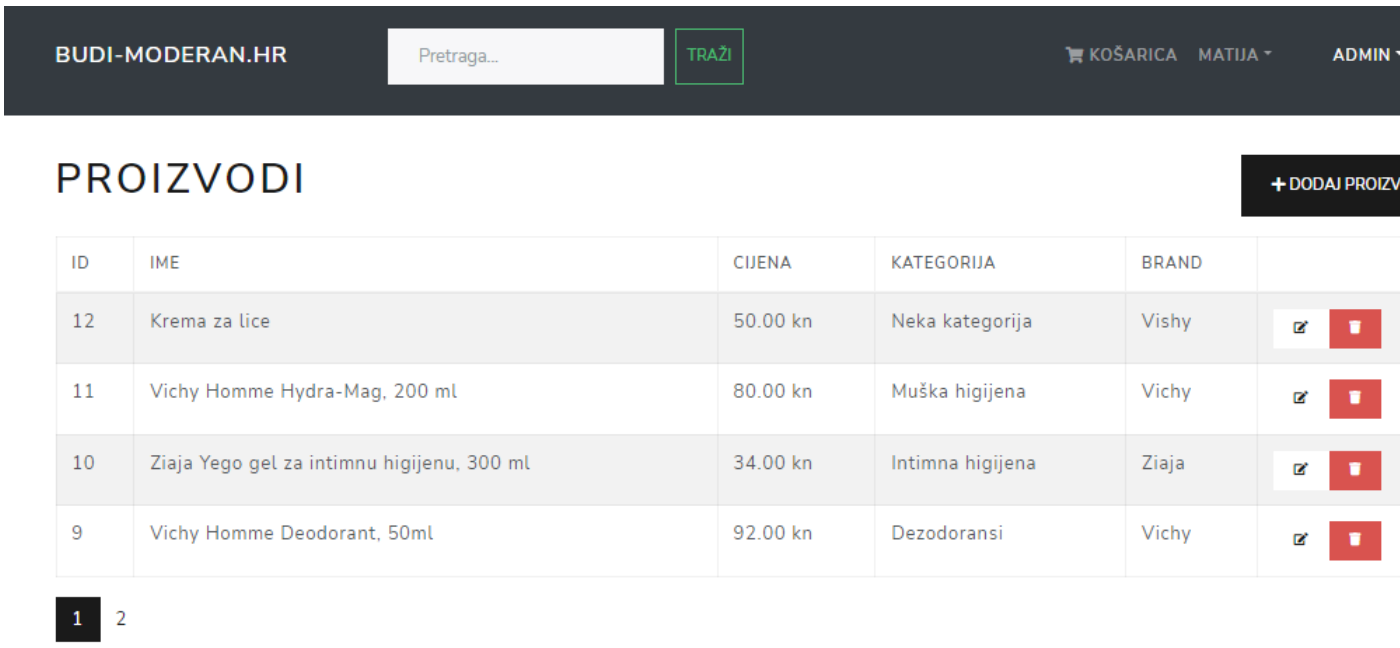
Slika 148 Admin prikaz korisnika

Slika 149 prikazuje ekran za uređivanje korisnika. Možemo mijenjati ime i email adresu korisnika ili ga označiti kao admina.



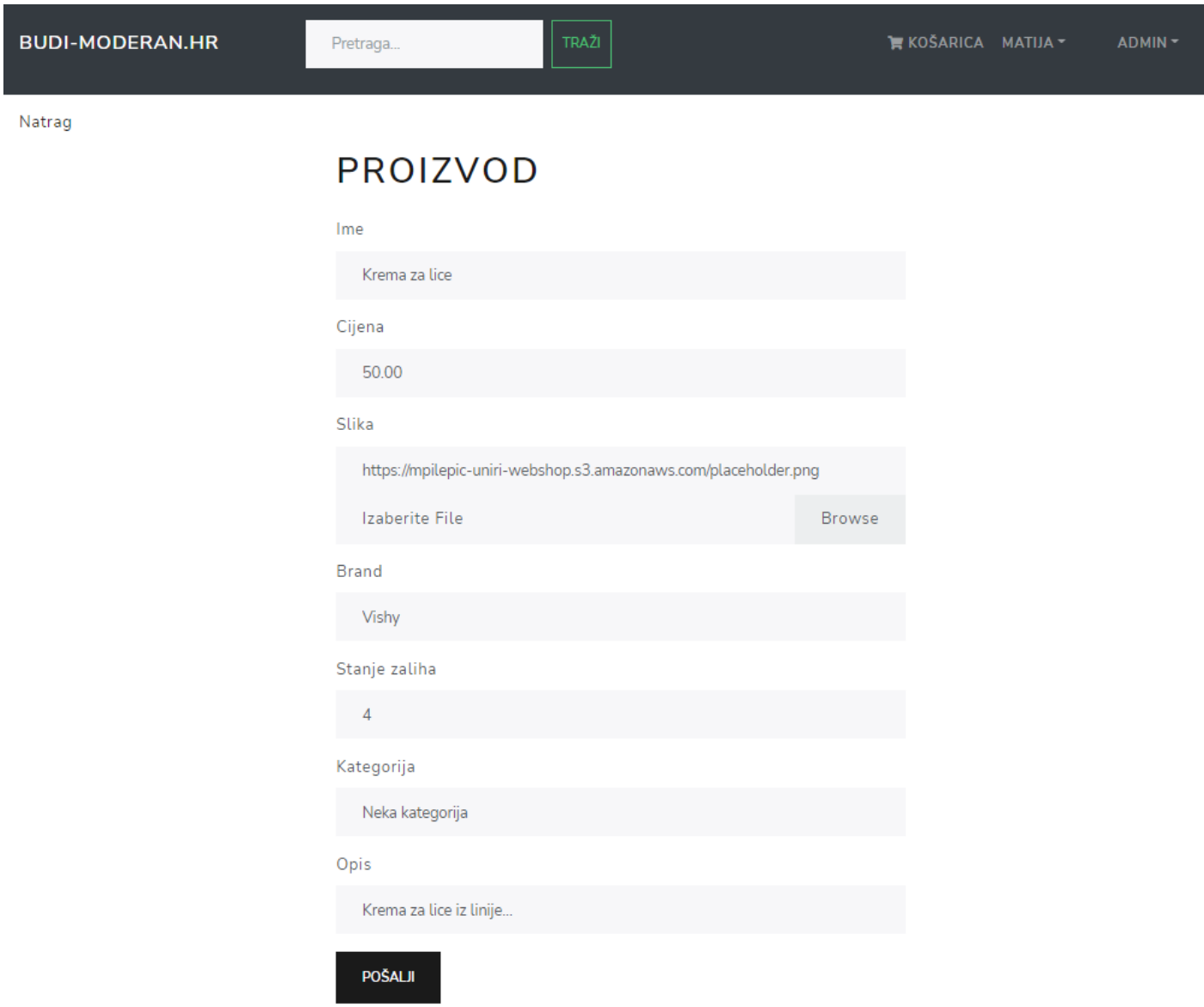
Slika 149 Uređivanje korisnika

Stranica prikaza proizvoda slična je stranici za prikaz korisnika. Svi korisnici su izlistani, mogu se uređivati, brisati ili dodavati novi (slika 150).



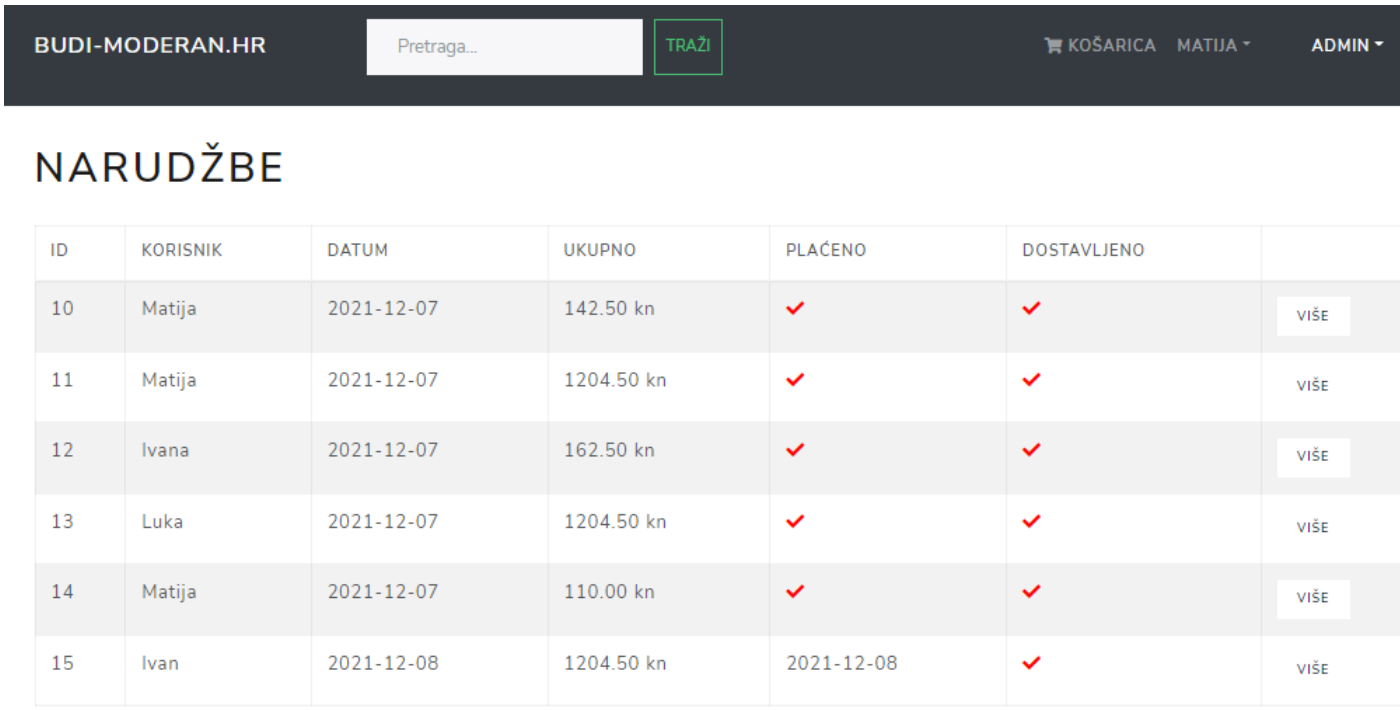
Slika 150 Admin prikaz proizvoda

Na slici 151 prikazan je ekran za uređivanje ili dodavanje proizvoda. Za obje funkcionalnosti koristi se ista forma.



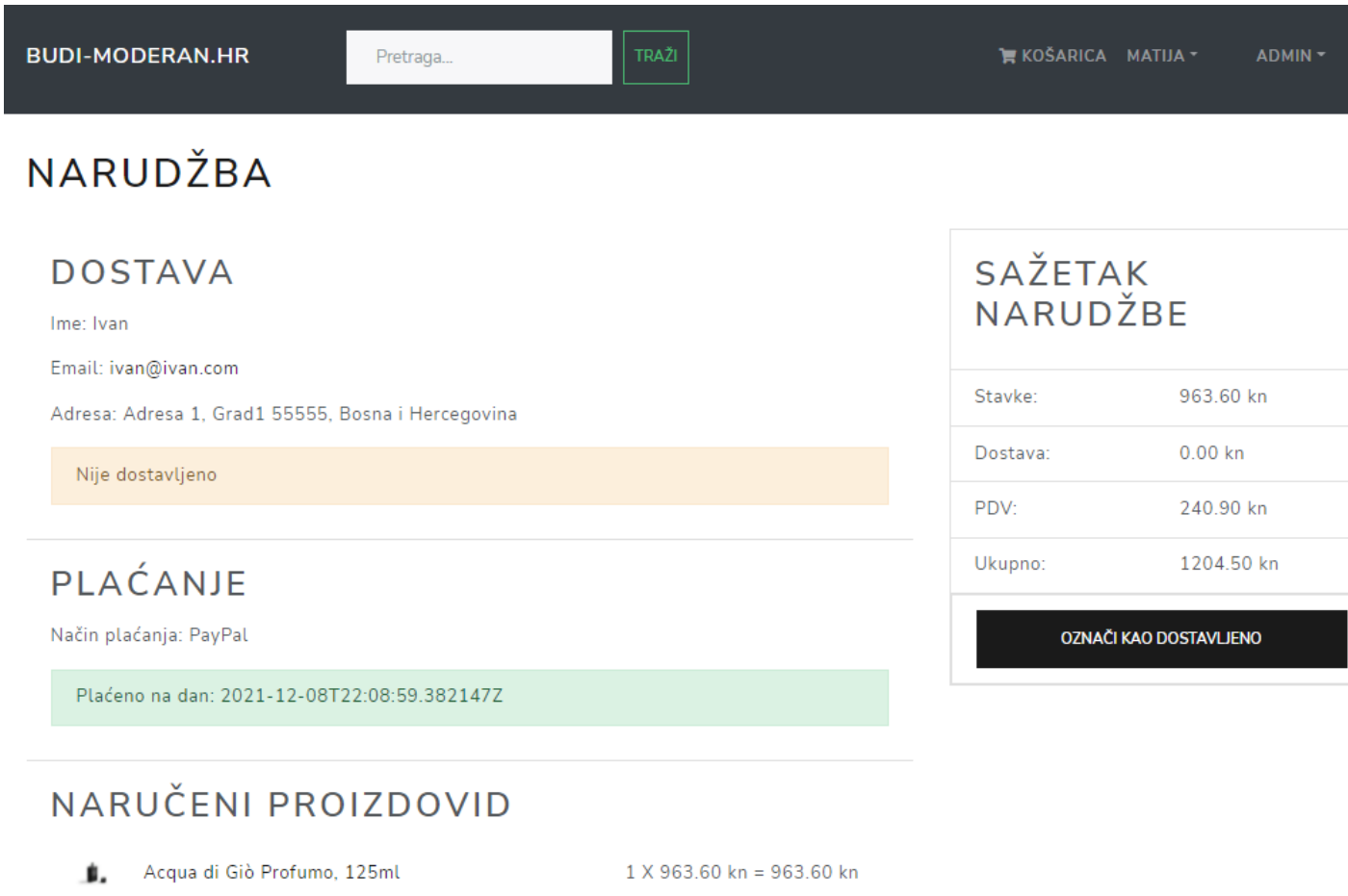
Slika 151 Uređivanje ili dodavanje proizvoda

Možemo vidjeti sve narudžbe i njihovo stanje, ali ne možemo uređivati već postojeće narudžbe (slika 152).



Slika 152 Admin popis narudžbi

Slika 153 prikazuje mogućnost označavanja plaćenih narudžbi kao dostavljenih. Ako narudžba nije plaćena, ova funkcionalnost nije omogućena.



Slika 153 Detaljan prikaz narudžbe

3.12.3 Mobilni prikaz

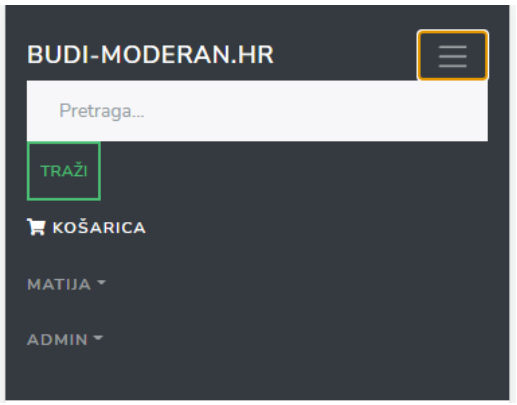
Aplikacija je u potpunosti responzivna te je funkcionalna bez obzira na kojoj veličini ekrana ju pregledavamo. Slike 154 i 155 prikazuju prikaz početne stranice i detaljan opis proizvoda otvorene na mobilnom uređaju.

Slika 154 Mobilni prikaz početne stranice

Slika 155 Mobilni prikaz proizvoda

Slika 156 prikazuje izgled košarice, a slika 157 izgled izbornika na mobilnom uređaju.

Slika 156 Mobilni prikaz stranice košarice



Slika 157 Mobilni prikaz izbornika

Zaključak

Rezultat ovog rada je funkcionalna web aplikacija za web prodaju. U aplikaciju se vrlo lako mogu postaviti bilo koji proizvodi, te se ona može prilagoditi prirodi proizvoda koji se prodaju. Izrazito je jednostavno proširiti ju s dodatnim komponentama i stranicama.

Kao backend razvojni okvir koji se brine o obradi baze podataka izabrao sam Django Python okvir. Osnovna namjena Djanga je olakšavanja izrade web aplikacija, s praktički beskonačnim mogućnostima što ga čini jednim od najpopularnijih okvira te vrste današnjice. Za kreiranje frontenda, odnosno korisničkog sučelja, koristio sam JavaScript okvir React. React je besplatna JavaScript biblioteka otvorenog koda za izgradnju korisničkih sučelja temeljenih na komponentama. Jedna je od najpopularnijih biblioteka te vrste, uz Vue.js i Angluar, današnjice. Sve navedene datoteke su temeljene na JavaScript programskom jeziku te je preduvjet za korištenje istih poznavanje jezika. Također se podrazumijeva znanje iz HTML-a i CSS-a. Za upravljanje stanjima aplikacije korišten je Redux. Koncepti Reduxa nešto su kompliciraniji, ali bez upravljanja stanjima nije moguće napraviti web aplikaciju. Aplikacija za bazu podataka koristi PostgreSQL tip baze. PostgreSQL je sustav objektno-relacijskih baza podataka otvorenog koda i kao takav smatrao sam da upotpunjuje potrebe ove aplikacije.

Tokom kreiranja, i nakon završetka, postavljalo se pitanje dali su izbrane najbolje dostupne tehnologije. Ja smatram kako na to pitanje ne postoji točan odgovor iz razloga što svaka tehnologija ima svoje prednosti i mane, tako i ove koje sam ja koristio. Bitno je izabrati tehnologiju s kojom možemo ostvariti svoj zadani cilj, što je bilo napraviti aplikaciju za web trgovinu. Prikazao sam kako se kombinacijom različitih tehnologija i programskih jezika može dobiti jedinstveno rješenje.

Literatura

[1] Django introduction, https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction, Mozilla.org (26.11.2021.)

[2] What is Django Rest Framework and why you should learn it, https://letslearnabout.net/blog/what-is-django-rest-framework-and-why-you-should-learn-it/, LetsLearnAbout.net (26.11.2021.)

[3] What is a REST API?, https://www.redhat.com/en/topics/api/what-is-a-rest-api, RedHat.com (26.11.2021.)

[4] Django Rest Framework, https://www.django-rest-framework.org/, Django-Rest-Framework.org (26.11.2021.)

[5] React (JavaScript library), https://en.wikipedia.org/wiki/React_(JavaScript_library), wikipedia.org (26.11.2021.)

[6] Why React-Bootstrap?, https://react-bootstrap.github.io/getting-started/why-react-bootstrap/, github.io (03.12.2021.)

[7] Getting Started with Redux, https://redux.js.org/introduction/getting-started, redux.js.org

[8] Why use Redux? A tutorial with examples, https://blog.logrocket.com/why-use-redux-reasons-with-clear-examples-d21bffd5835/, logrocket.com (03.12.2021.)

[9] Amazon S3, https://aws.amazon.com/s3/, amazon.com (03.12.2021.)

[10] What is AWS RDS?, https://www.w3schools.com/whatis/whatis_aws_rds.asp, w3schools.com (03.12.2021.)

[11] PostgreSQL: The World's Most Advanced Open Source Relational Database, https://www.postgresql.org/, postgresql.org (07.12.2021.)

[12] What is Heroku?, https://www.heroku.com/about, heroku.com (08.12.2021.)

[13] What Is a Single Page Application and Why Do People Like Them so Much?, https://www.bloomreach.com/en/blog/2018/07/what-is-a-single-page-application.html, Bloomreach.com (10.12.2021.)

Popis slika

Prilozi

1. Web aplikacija za internet prodaju: https://mpilepic-webshop.herokuapp.com/
2. GitHub repozitorij aplikacije: https://github.com/mpilepic/webshop