Web aplikacija za prodaju odjeće

Mikulić, David

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Stručni prijediplomski studij Računarstvo

WEB APLIKACIJA ZA PRODAJU ODJEĆE

Završni rad

David Mikulić

Osijek, 2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1S: Obrazac za ocjenu završnog rada na stručnom prijediplomskom studiju

Ocjena završnog rada na stručnom prijediplomskom studiju

Ime i prezime pristupnika:	David Mikulić
Studij, smjer:	Stručni prijediplomski studij Računarstvo
Mat. br. pristupnika, god.	AR4867, 27.07.2021.
JMBAG:	0246094261
Mentor:	doc. dr. sc. Tomislav Galba
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 1:	doc. dr. sc. Tomislav Galba
Član Povjerenstva 2:	izv. prof. dr. sc. Alfonzo Baumgartner
Naslov završnog rada:	Web aplikacija za prodaju odjeće
Znanstvena grana završnog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rada:	Potrebno je napraviti web aplikaciju koja će omogućiti korisnicima odabir, odnosno, kupovinu različitih odjevnih predmeta. Web aplikacija treba imati više korisničkih uloga (minimalno dvije) sa pripadajućim funkcionalnostima. Također, potrebno je koristiti neke od dostupnih relacijskih baza podataka. Napomena: tema rezervirana za David Mikulić

Datum ocjene pismenog dijela završnog rada od strane mentora:	18.09.2024.
Ocjena pismenog dijela završnog rada od strane mentora:	Izvrstan (5)
Datum obrane završnog rada:	27.09.2024.
Ocjena usmenog dijela završnog rada (obrane):	Izvrstan (5)
Ukupna ocjena završnog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije završnog rada čime je pristupnik završio stručni prijediplomski studij:	30.09.2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O IZVORNOSTI RADA

Osijek, 30.09.2024.

Ime i prezime Pristupnika:	David Mikulić
Studij:	Stručni prijediplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	AR4867, 27.07.2021.
Turnitin podudaranje [%]:	7

Ovom izjavom izjavljujem da je rad pod nazivom**: Web aplikacija za prodaju odjeće**

izrađen pod vodstvom mentora doc. dr. sc. Tomislav Galba

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

Sadržaj

1. Uvod

Ovaj završni rad bavi se izradom web aplikacije za prodaju odjeće, s ciljem unapređenja korisničkog iskustva s ugodnim korisničkim sučeljem i olakšavanja kupovine proizvoda. Implementacija aplikacije omogućuje jednostavno i brzo pregledavanje ponude odjeće po kategorijama, te olakšava proces naručivanja i plaćanja.

Aplikacija je osmišljena kao web shop brenda namijenjenog za glazbeni kolektiv, omogućujući korisnicima jednostavan pristup odjevnim predmetima te brzu i sigurnu kupovinu. Kroz niz poglavlja detaljno je objašnjen proces izrade navedene web aplikacije. Prvo poglavlje, „Korištene tehnologije“, opisuje tehnologije korištene pri izradi web aplikacije, uključujući razloge zašto su odabrane te njihovu sintaksu i strukturu.

U poglavlju „Slična rješenja“ su predstavljene već postojeće web aplikacije koje nude slične mogućnosti, te koje su poslužile kao inspiracija za izgled i funkcionalnost aplikacije ovog završnog rada. Poglavlje „Opis web aplikacije“ opisuje strukturu korisničkog sučelja, sve uloge koje aplikacija ima te dijelove korisničkog sučelja koji su specifični za pojedinu ulogu.

Poglavlje „Programsko rješenje“ sa svojim potpoglavljima „Backend“ i „Frontend“ opisuje arhitekturu servera, rukovanje bazom podataka, implementaciju interaktivnih elemenata objašnjavajući kako se postiže responzivnost i interaktivnost u korisničkom sučelju.

Ovaj sveobuhvatni pregled omogućuje čitateljima bolje razumijevanje izrade web trgovine te pruža uvid u korištenje modernih web tehnologija za izradu aplikacija namijenjenih online prodaji odjeće.

1.1. Zadatak završnog rada

Potrebno je napraviti web aplikaciju koja će omogućiti korisnicima odabir, odnosno, kupovinu različitih odjevnih predmeta. Web aplikacija treba imati više korisničkih uloga (minimalno dvije) sa pripadajućim funkcionalnostima. Također, potrebno je koristiti neke od dostupnih relacijskih baza podataka.

2. Slična rješenja

S obzirom da je brend za koji se izrađuje web aplikacija za prodaju odjeće namijenjen za glazbeni kolektiv, web aplikacija je inspirirana trgovinama drugih glazbenih izvođača.

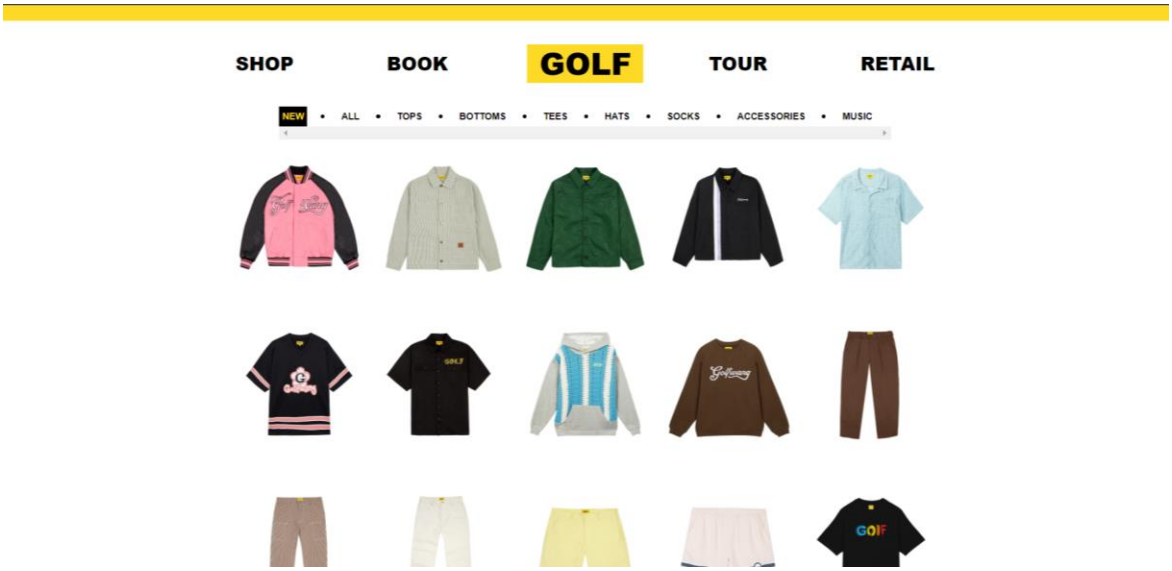
2.1. Travis Scott Official Store

Travis Scott Official Shop je web aplikacija, odnosno internetska trgovina poznatog hip hop izvođača Travisa Scotta koja služi kao izvrstan primjer za izradu web trgovine s prodajom odjeće i drugih promotivnih artikala (engl. *merch*) kao što su CD-ovi. Na slici 2.1 prikazan je minimalistički, ali intuitivan dizajn stranice s prikazom svih proizvoda na početnoj stranici. [1]

Slika 2.1 Početna stranica web trgovine Travisa Scotta

2.2. GolfWang

GolfWang je web aplikacija za prodaju odjeće izvođača Tyler-a, The Creator-a koja za razliku od Travis Scottove sadrži različite kategorije i ne prikazuje sve proizvode na početnom ekranu što je prikazano na slici 2.2. Ostatak web trgovine (osim korisničkog sučelja) prilično je sličan prvom primjeru. [2]



Slika 2.2 Početna stranica na web trgovini GolfWang

3. Korištene tehnologije

U ovom poglavlju ukratko su objašnjene korištene tehnologije za izradu klijentskog i poslužiteljskog dijela ove web trgovine.

3.1. Vite

Vite je *build tool* korišten za izgradnju i postavljanje *frontend* dijela ovog projekta koji pruža brže razvojno iskustvo za izgradnju web aplikacija. Sadrži razvojni poslužitelj koji omogućava brzo osvježavanje putem nativnih ES modula, te naredbe za izgradnju koje optimiziraju i minificiraju *kod* kada je spreman za produkciju. [3]

3.2. Vue.js

Vue.js je progresivni JavaScript *framework* za stvaranje modernih i dinamičkih korisničkih sučelja. Glavna značajka Vue.js-a je njegova pristupačnost i lakoća učenja, što ga u današnje vrijeme čini popularnim izborom za razvoj web aplikacija. Vue.js je temeljen na komponentama, što znači da aplikacija može biti podijeljena na manje, ponovo upotrebljive dijelove. [4]

Arhitektura Vue.js-a sastoji se od nekoliko osnovnih koncepata, a glavni koncept su komponente. Komponente definiraju prikaze (engl. *view*), skupove elemenata na ekranu kojima se uz pomoć Vue.js-a upravlja u skladu s logikom aplikacije. Komponenta se sastoji od tri dijela:

- HTML (engl. *HyperText Markup Language*) predložak (engl. *template*) – sadržaj koji se prikazuje korisniku unutar te komponente definira se unutar <*template*> oznake
- JavaScript ili TypeScript klasa (za ovaj projekt korišten je JavaScript) – ponašanje i logika komponente upravljaju se unutar <*script*> oznake
- CSS (engl. *Cascading Style Sheets*) datoteka stilova – klase i stilovi za izgled prikaza definiraju se unutar <*style*> oznake

Vue.js komponente predstavljaju zasebnu jedinicu funkcionalnosti unutar aplikacije. Sadrže dodatne informacije vezane uz *kod* koje upućuju Vue.js-u kako koristiti i prikazati komponentu. Primjer strukture i ponovnog korištenja Vue.js komponente prikazan je u *kodu* na slici 3.1:

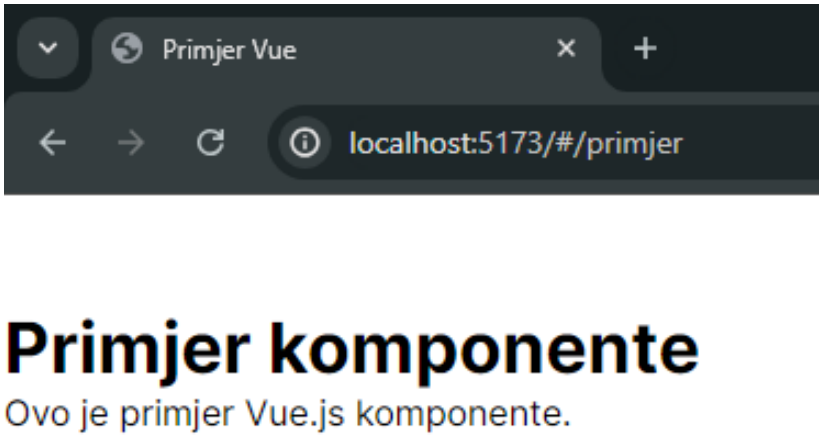
```vue
<template>
    <div class="container">
        <h1>{{ title }}</h1>
        <p>{{ message }}</p>
    </div>

</template>

<script setup>
const props = defineProps(["title","message"]);
</script>
```

Slika 3.1 Komponenta *Primjer.vue*

```vue
<template>
    <Primjer
    title="Primjer komponente"
    message="Ovo je primjer Vue.js komponente."
    >
    </Primjer>
</template>

<script setup>
import Primjer from './Primjer.vue';
</script>
```

Slika 3.2 Prikaz *PrimjerView.vue*

Ponovnim iskorištavanjem komponente *Primjer.vue* može se dinamički mijenjati određeni sadržaj kao što je u ovom slučaju *title* i *message*, što je prikazano na slici 3.2. Rezultat ovog *koda* u web pregledniku prikazan je na slici 3.3:



Slika 3.3 Primjer Vue.js prikaza i komponente u web pregledniku

3.3. SCSS

CSS je jezik za opisivanje stila HTML dokumenta. Koristi se za definiranje izgleda i formatiranja web stranica, uključujući boje, raspored, fontove i sl. [5] U ovom projektu korišten je SCSS (engl. *Sassy Cascading Style Sheets*), koji je proširenje CSS-a. SCSS nudi dodatne funkcionalnosti koje olakšavaju pisanje i održavanje stilova. SCSS je dio Sass (engl. *Syntactically Awesome Stylesheets*) predprocesora kojim je omogućeno korištenje naprednih funkcionalnosti koje nisu dostupne u čistom CSS-u. [6]

Neke od naprednih funkcionalnosti SCSS-a:

- Varijable – definiranje varijabli koje se koriste za pohranu vrijednosti kao što su boje, fontovi i veličine
- Ugnježđivanje – ugnježđivanje selektora unutar drugih selektora, što pomaže u organiziranju i čitanju *koda*
- *Mixins* – grupiranje CSS deklaracija koje se mogu ponovno koristiti

Na slici 3.4 prikazane su sve tri navedene funkcionalnosti, ponovno iskoristiv stil za gumb *buttonStyle* u kojem se također koriste varijable i ugnježđivanje:

Slika 3.4 Primjer korištenja varijabli, ugnježđivanja i *mixina* u SCSS-u

3.4. Node.js

Node.js je platforma izgrađena na Chromeovom V8 JavaScript *engineu* koja omogućava izvršavanje JavaScript *koda* izvan web preglednika. Node.js je dizajniran za izgradnju skalabilnih mrežnih aplikacija i posebno je prikladan za razvoj poslužiteljskih aplikacija. [7] Neki od ključnih koncepata Node.js-a su: asinkrono programiranje, modularnost, korištenje Express razvojnog okvira te korištenje ruta i upravljača (engl. *controller*) gdje se rute koriste za definiranje krajnjih točaka (engl. *endpoints*) aplikacije a *controlleri* sadrže logiku koja se izvršava prilikom poziva krajnjih točaka. [8]

Primjer postavljenih ruta za upravljač za autentifikaciju korisnika prikazan je na slici 3.6

Slika 3.5 Primjer postavljanja poslužitelja koristeći Node.js

```
const router = express.Router();

router.post('/register', createUser);
router.post('/login', loginUser);
router.get('/user/:email', getUserByEmail);

export default router;
```

Slika 3.6 Primjer postavljenih ruta za autentifikaciju korisnika

3.5. MySQL

MySQL je relacijski sustav za upravljanje bazama podataka (RDBMS – engl. *Relational Database Management System*) koji koristi SQL (engl. *Structured Query Language*) za manipuliranje podacima u bazi. U ovom projektu MySQL se koristi za pohranu i upravljanje podacima vezanim uz korisnike, proizvode, narudžbe i ostale bitne entitete aplikacije za prodaju odjeće. [9] Za komunikaciju s MySQL bazom podataka koristi se Knex.js: SQL *query builder* za Node.js. Pisanje SQL upita u Javascriptu olakšavaju se pomoću Knex.js-a. [10]

Primjer spajanja na MySQL bazu podataka sa Knex.js-om prikazan je na slici 3.7.

```
import knex from 'knex';
import dotenv from 'dotenv';

dotenv.config({ path: './.env' });

const db = knex({
  client: 'mysql2',
  connection: {
    host: process.env.DATABASE_HOST,
    user: process.env.DATABASE_USER,
    password: process.env.DATABASE_PASSWORD,
    database: process.env.DATABASE,
  },
});

db.raw('SELECT 1')
  .then(() => {
    console.log('MySQL connected');
  })
  .catch((error) => {
    console.error('Error connecting to MySQL:', error);
  });

export default db;
```

Slika 3.7 Primjer konfiguracije Knex.js i povezivanja s MySQL bazom poadataka

4. Opis web aplikacije

Ova web aplikacija namijenjena je kupovini raznih promotivnih artikala, poput majica, hlača i sličnih proizvoda.

4.1. Korisničko sučelje

U ovom poglavlju prikazan je izgled korisničkog sučelja ove web trgovine.

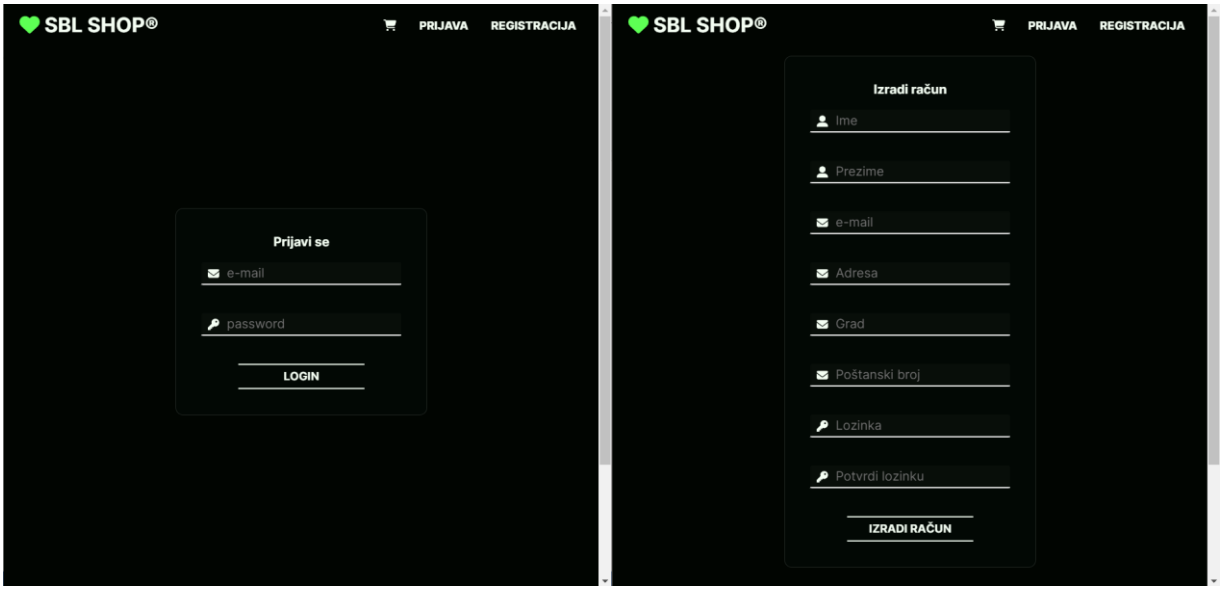
4.1.1. Početna stranica

Na slici 4.1 prikazana je početna stranica, na kojoj se pri vrhu stranice nalazi zaglavlje. U gornjem lijevom kutu prikazan je logo web trgovine a u desnom gumbi za košaricu, korisnički profil te gumb za odjavu koji se pojavljuje ako je korisnik prijavljen. Ako postoji barem jedan proizvod u košarici, pojavljuje se broj uz ikonicu košarice koji prikazuje koliko proizvoda ima u košarici. Ispod zaglavlja nalazi se izbornik s kategorijama odjeće koji zelenom bojom prikazuje koja je trenutna odabrana kategorija. Ispod izbornika prikazani su svi proizvodi iz te kategorije u obliku kartica s prikazom njihove glavne slike, imenom i cijenom. Na dnu stranice je prikazano podnožje, a u desnom kutu prikazan je link na nasumičnu pjesmu kolektiva što je osposobljeno koristeći Spotify API (engl. *Application Programming Interface*). Zaglavlje i podnožje koriste se na svakoj stranici web aplikacije kao ponovno iskoristive komponente.

Slika 4.1 Prikaz početne stranice web aplikacije

4.1.2. Prikaz stranica za prijavu i registraciju

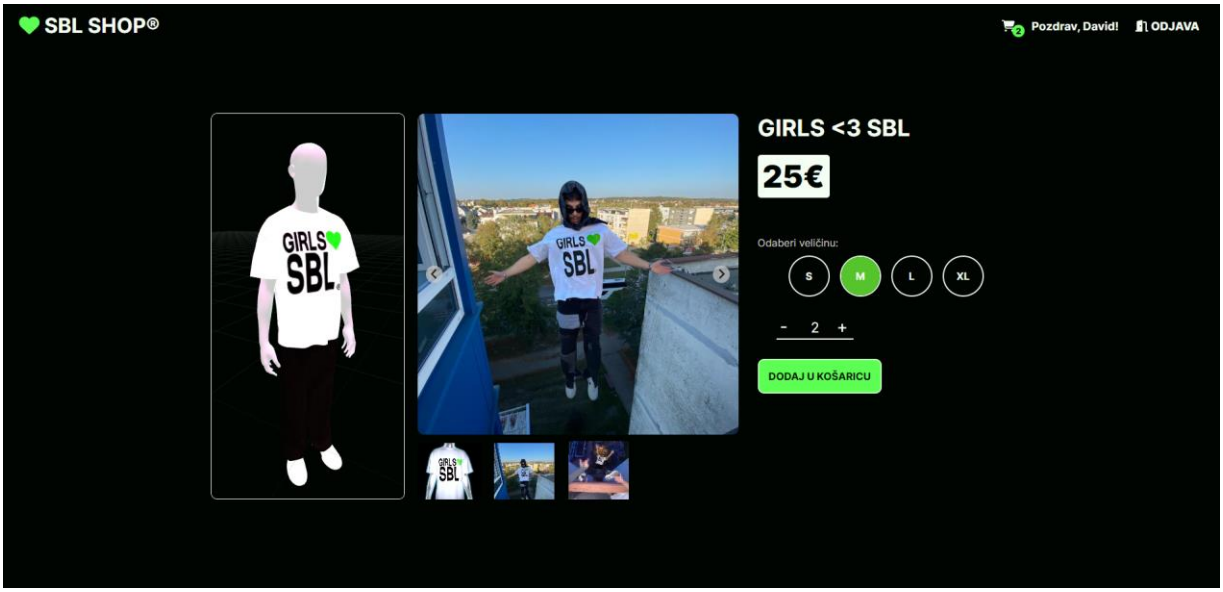
Na slici 4.2 prikazan je obrazac za prijavu u kojem se unose email adresa i lozinka s kojima je kreiran korisnički račun. Na stranici za registraciju korisnik mora unijeti i podatke za dostavu kako bi kasnije prilikom narudžbe korisničko iskustvo bilo bolje te kako bi se proces pojednostavio.



Slika 4.2 Prikaz stranica za prijavu (lijevo) i registraciju (desno)

4.1.3. Prikaz stranice pojedinog proizvoda

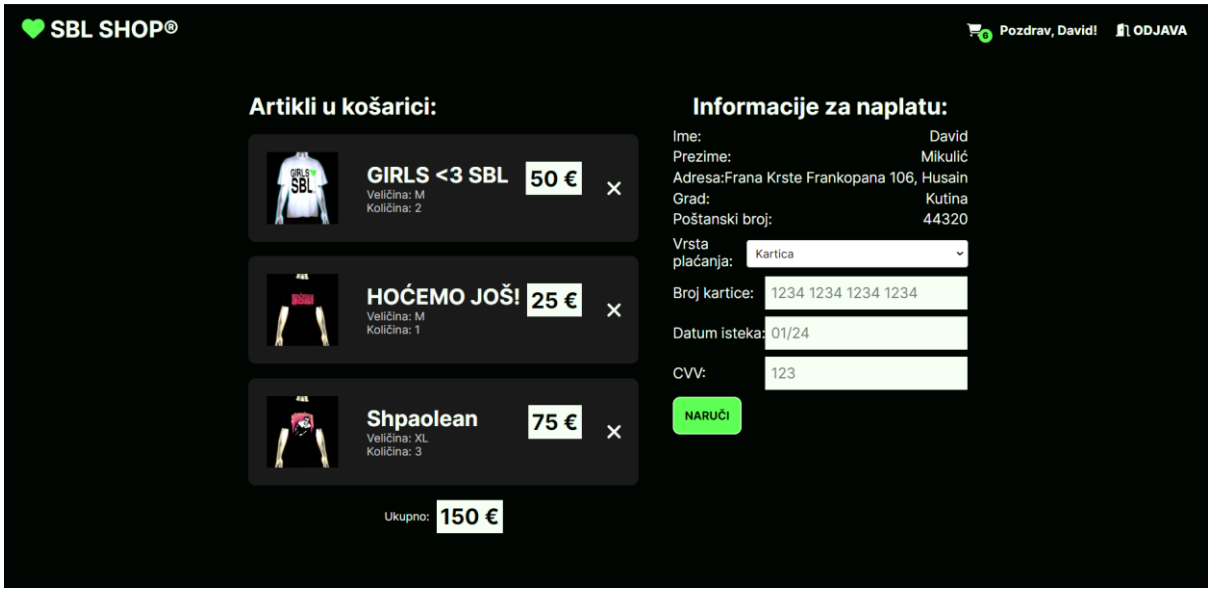
Na slici 4.3 prikazana je stranica pojedinog proizvoda. S desne strane ove stranice, prikazan je interaktivni 3D model pomoću Three.js biblioteke koji korisnik može zumirati i rotirati pomoću miša kako želi. Model je dodan u prikaz kako bi proizvod bio prikazan na zanimljiviji način te kako bi se poboljšalo korisničko iskustvo. Model animiran beskonačnom animacijom laganog rotiranja udesno oko svoje y-osi. Uz 3D model nalazi se galerija slika proizvoda. Korisnik može stiskati na male ikonice slika koje se prikazuju na mjestu velike slike. Također je moguće koristiti i gumbe za listanje slika. Na desnoj strani prikazan je naslov proizvoda, cijena te opcija za odabir veličine. Omogućeno je i biranje veličine proizvoda koji je moguće dodati u košaricu klikom na gumb „DODAJ U KOŠARICU“.



Slika 4.3 Prikaz stranice pojedinog proizvoda

4.1.4. Prikaz stranice naplate

Na stranici naplate koristi se komponenta košarice koja se također koristi i na stranici košarice. Na lijevoj strani slike 4.4 prikazani su svi proizvodi s njihovom cijenom, imenom, odabranim veličinama i količinama te ukupna vrijednost košarice. S desne strane, ako je korisnik prijavljen, prikazani su svi podaci uneseni tijekom registracije. Ako korisnik nije prijavljen, svi potrebni podaci za dostavu moraju biti uneseni. Postoji opcija plaćanja karticom i opcija plaćanja pouzećem.

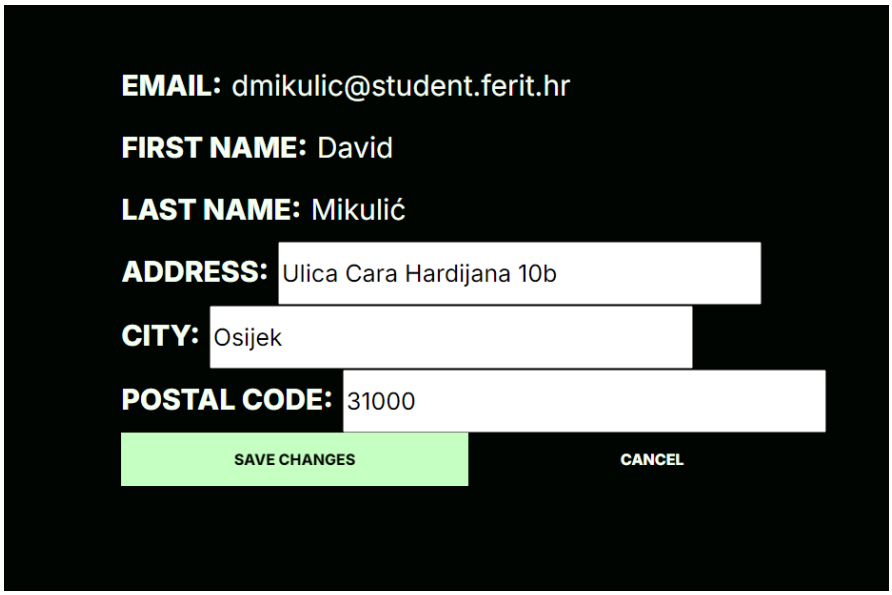


Slika 4.4 Prikaz stranice naplate

4.2. Uloge korisnika

4.2.1. Korisnik – registrirani korisnik

Registriranim korisnicima omogućeno je pregledavanje svih proizvoda, kao i pristup posebnoj stranici prikazanoj na slici 4.5, na kojoj se mogu mijenjati njihovi podaci za dostavu.



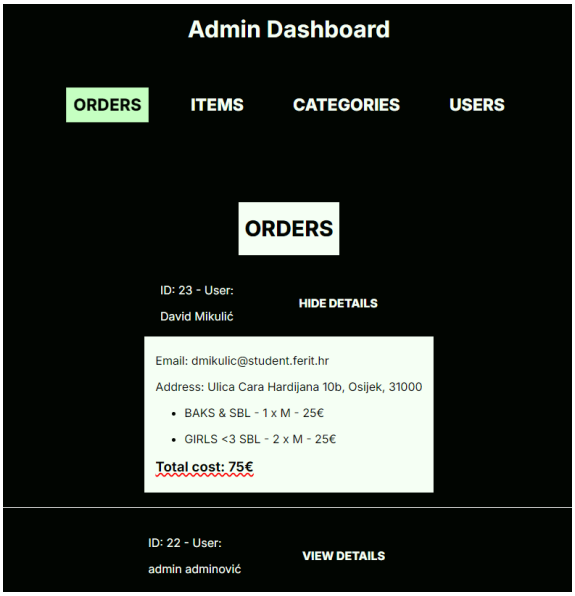
Slika 4.5 Prikaz stranice gdje korisnici mogu uređivati svoje podatke

4.2.2. Korisnik – gost

Korisnicima bez registriranog profila omogućeno je naručivanje proizvoda kao i registriranim korisnicima, no prilikom svake narudžbe podaci za dostavu moraju biti uneseni.

4.2.3. Administrator

Administratori imaju pristup posebnom prikazu s nadzornom pločom koji je prikazan na slici 4.6. Na nadzornoj ploči omogućeno je pregledavanje i kreiranje novih korisnika, kategorija i proizvoda, kao i pregled svih narudžbi.



Slika 4.6 Prikaz nadzorne ploče s ekranom za narudžbe

5. Programsko rješenje

5.1. Baza podataka

U ovom poglavlju objašnjena je struktura korištene baze podataka.

Za ovu web aplikaciju korištena je MySQL relacijska baza podataka. Baza podataka sastoji se od tablica: *users*, *categories*, *items*, *images*, *models*, *orders*, *carts* i *cart_items*.

5.1.1. Tablica *users*

U tablici s korisnicima na slici 5.1 svaki redak predstavlja jednog korisnika. Korisniku je dodjeljen ID (identifikacijski broj) te se od ostalih podataka sprema email adresa, ime, prezime, kriptirana lozinka, uloga i adresa korisnika.

ID	email	firstname	lastname	password	role	address	city	postal_code
21	admin@admin.hr	admin	admin	$2b$10$nN21wltPwu75IFZLAUnqreGNd5ZXaoK7dq1gT6tx2mB...	admin	Frankopanska 106, Husain	Kutina	44320
33	mavvid44@gmail.com	David	Mikulić	$2b$10$KwGrPD2pkNhfgCaT9Wn0hO4aRpPDTpHB0DLSn7pCd4n...	0	Frana Krste Frankopana 106, Husain	Kutina	44320

Slika 5.1 Prikaz tablice *users*

5.1.2. Tablica *categories*

U tablici kategorija na slici 5.2 spremljeni su identifikacijski brojevi svake kategorije te ime kategorije.

ID	category
1	majice
2	hlače
0	ostalo

Slika 5.2 Prikaz tablice *categories*

5.1.3. Tablica *items*

U tablici za proizvode na slici 5.3 spremljen je identifikacijski broj svakog proizvoda, njegova cijena te strani ključ na kategoriju kojoj taj proizvod pripada.

ID	name	price	category_id
1	GIRLS <3 SBL	25	1
2	HOĆEMO JOŠ!	25	1
3	Baraka 2022	25	1
5	BAKS & SBL	25	1
6	Shpaolean	25	1
7	Pattern Hlače	40	2

Slika 5.3 Prikaz tablice *items*

5.1.4. Tablica *images*

U tablici na slici 5.4 sa slikama spremljen je identifikacijski broj svake slike, putanja slike te strani ključ na identifikacijski broj proizvoda za koji je slika.

ID	image	item_id
1	assets/images/girlslovesbl.jpg	1
2	assets/images/girlslovesbl1.jpeg	1
3	assets/images/girlslovesbl2.jpeg	1
4	assets/images/hocemojos.jpg	2
5	assets/images/hocemojos1.jpg	2
6	assets/images/hocemojos3.jpg	2
7	assets/images/hocemojos2.jpg	2
8	assets/images/baraka22.jpg	3
9	assets/images/kset23.jpg	5
10	assets/images/shpaolean.jpg	6
11	assets/images/hlace.jpg	7

Slika 5.4 Prikaz tablice *images*

5.1.5. Tablica *models*

U tablici s 3D modelima na slici 5.5 spremljene su putanje do datoteke modela (kao kod slika) i strani ključ na identifikacijski broj proizvoda koji predstavljaju.

ID	model	item_id
1	assets/models/girlslovesbl/scene.gltf	1
2	assets/models/hocemojos/scene.gltf	2
3	assets/models/baraka22/scene.gltf	3
5	assets/models/kset23/scene.gltf	5
6	assets/models/shpaolean/scene.gltf	6
7	assets/models/hlace/scene.gltf	7

Slika 5.5 Prikaz tablice *models*

5.1.6. Tablice *carts, cart_items* i *orders*

Na slici 5.6 prikazane su tri tablice koje je potrebno objasniti zajedno jer su međusobno povezane. Prva tablica *carts,* sadrži identifikacijski broj košarice, strani ključ koji se odnosi na identifikacijski broj korisnika kojem košarica pripada, stupac *is_paid* koji bilježi je li košarica plaćena te ukupnu cijenu košarice. U tablici *cart_items* spremljen je svaki pojedini proizvod unutar neke košarice sa stranim ključem na identifikacijski broj košarice u kojoj se nalazi, stranim ključem na proizvod te veličina i količina tog proizvoda u košarici. Tablica *orders* predstavlja tablicu narudžbi u kojoj se novi red stvara prilikom izvršetka narudžbe, odnosno u trenutku kada vrijednost *is_paid* neke košarice postane 1.

tablica 'carts'

ID	user_ID	is_paid	cost
87	33	1	100

tablica 'cart_items'

ID	cart_id	item_id	size	quantity
237	87	5	L	3
238	87	1	M	1

tablica 'orders'

id	user_id	cart_id
16	33	87

Slika 5.6 Prikaz tablica *carts, cart_items* i *orders*

5.1.7. Tablice *guests*, *guest_cart_items*, *guest_carts* i *guest_orders*

Na slici 5.7 prikazane su četiri tablice potrebne za omogućavanje korištenja web-aplikacije korisnicima bez registriranja. U trenutku dodavanja bilo kojeg proizvoda u košaricu, kreira se novi red u tablicama *guests*, *guest_carts* i *guest_cart_items*, u tablicu *guests* zapisuje se nasumični identifikacijski broj ID, kako bi se znalo kojem gostu pripada koja košarica. Prilikom završavanja narudžbe, kreira se red u tablici *guest_orders* sa podacima koje je gost unio te identifikacijskim brojevima gosta i košarice gosta. U tablici *guest_carts* stupac *is_paid* je postavljen na vrijednost 1.

tablica 'guest_orders'

id	firstname	lastname	email	address	city	postal_code	cart_id	guest_id
1	Gost	Gostović	gost@guest.hr	Kneza Trpimira 2B	Osijek	31000	4	40

tablica 'guest_cart_items'

id	guest_cart_id	item_id	size	quantity
2	2	2	M	1
3	3	3	M	1
4	4	3	M	1
5	4	2	M	1
6	4	3	M	1
7	5	3	M	1

tablica 'guest_carts'

id	guest_id	is_paid	cost
2	0	0	25
3	52	0	25
4	40	1	75

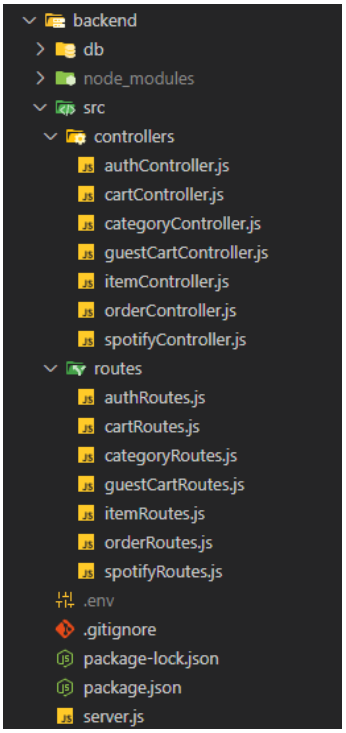
tablica 'guests'

id
82
4
0
52
40

Slika 5.7 Prikaz tablica *guests*, *guest_carts*, *guest_cart_items* i *guest_orders*

5.2. Backend

U ovom poglavlju obrađuje se način na koji je postavljen poslužiteljski dio aplikacije koristeći Node.js. Također je potrebno kreirati i datoteke i mape na pregledan način za lakše refaktoriranje i preglednost *koda*:

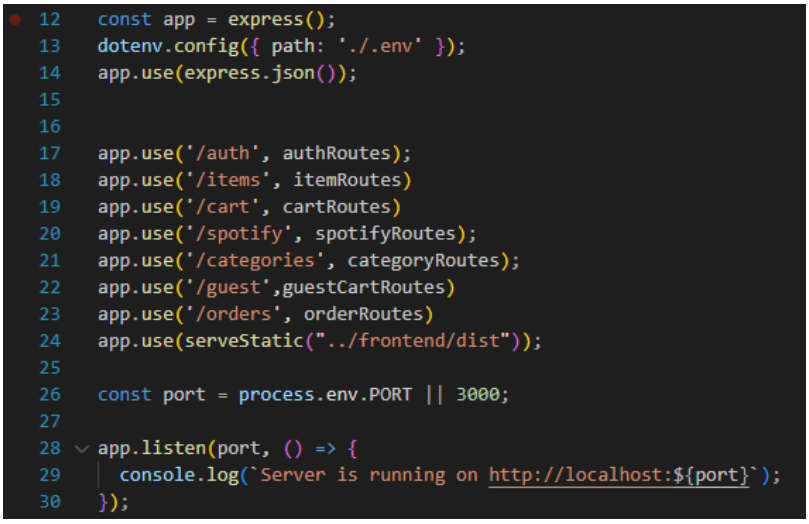


Slika 5.8 Struktura datoteka i mapa za backend dio aplikacije

Na slici 5.8 prikazano je kako je *kod* razdijeljen; na dio za povezivanje s bazom podataka (mapa db), upravljače i rute. Sve datoteke su objašnjene u daljnjim poglavljima.

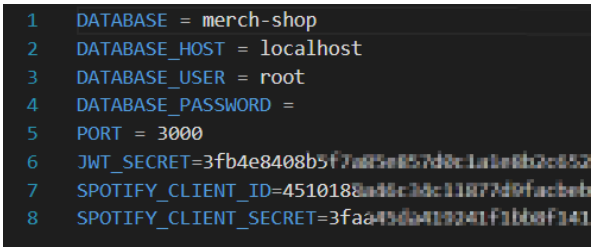
5.2.1. Postavljanje poslužitelja i povezivanje s bazom podataka

Kod na slici 5.9 se izvršava prilikom pokretanja aplikacije – datoteka *server.js*. Počinje kreiranjem instance Express aplikacije pomoću *express()* funkcije. Zatim se konfigurira učitavanje varijabli okruženja (engl. *environment variables*) iz *.env* datoteke korištenjem *dotenv.config({ path: './.env' })*. Datoteka *.env* sadrži informacije o bazi podataka te slične povjerljive informacije kao što su API ključevi ili nasumični *hash* za *hashiranje* korisničkih podataka kao što su tokeni. Na slici 5.10 prikazana je *.env* datoteka.



Slika 5.9 *Kod* za pokretanje poslužitelja – datoteka *server.js*

Nadalje, aplikacija se konfigurira tako da koristi JSON *parser* za obradu dolaznih zahtjeva koristeći *express.json()*. Ovo omogućava Express aplikaciji da automatski parsira JSON podatke iz dolaznih zahtjeva, što olakšava daljnji rad s JSON formatom u aplikaciji. Nakon toga se registriraju rute za autentifikaciju, artikle, košaricu, Spotify API i kategorije, koje su implementirane u odvojenim datotekama *authRoutes, itemRoutes, cartRoutes, spotifyRoutes* i *categoryRoutes*. Aplikacija se konfigurira da poslužuje statičke *frontend* datoteke aplikacije iz direktorija *../frontend/dist* koristeći *serveStatic()*, u toj mapi se nalaze minificirane i optimizirane datoteke iz svih Vue.js komponenata. Aplikacija se pokreće na određenom portu koji se čita iz varijable PORT u *.env* datoteci, u slučaju da datoteka *.env* ne postoji ili varijabla PORT ne postoji automatski se postavlja vrijednost za port na 3000. Nakon pokretanja, u konzolu se ispisuje na kojoj adresi je server pokrenut.



Slika 5.10 Datoteka *.env* – varijable okruženja

5.2.2. Korisnici i autentifikacija

Za rad s korisnicima koristi se upravljač *authController.js* koji sadrži sve funkcije za upravljanje korisnicima kao što su kreiranje novog računa, autentifikacija i dohvaćanje pojedinog korisnika preko email adrese.

Funkcija *createUser* sa slike 5.11 služi za registraciju novih korisnika. Uzima parametre iz zahtjeva (engl. *request body*) i prvo provjerava postoji li već korisnik s istom email adresom. Ako ne postoji, *hash* lozinke koju je korisnik unio generira se pomoću bcrypt biblioteke, kako bi se umjesto prave lozinke u bazu spremila kriptirana verzija te lozinke. Zatim se svi podaci upisuju u tablicu *users*, i korisnički račun je uspješno stvoren za novog korisnika.

```
export const createUser = async (req, res) => {
  try {
    const { firstname, lastname, email, password, address, city, postal_code } = req.body;

    const existingUser = await knex('users').where({ email }).first();
    if (existingUser) {
      return res.status(400).json({ message: 'Email is already registered' });
    }

    const hashedPassword = await bcrypt.hash(password, 10);

    await knex('users').insert({
      firstname,
      lastname,
      email,
      password: hashedPassword,
      role: 0,
      address,
      postal_code,
      city,
    });

    res.status(201).json({ message: 'User created successfully' });
  } catch (error) {
    console.error('Error registering user:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
};
```

Slika 5.11 Funkcija *createUser* u *authController.js*

Funkcija *loginUser* sa slike 5.12 služi za autentifikaciju korisnika prilikom prijave. Parametri iz zahtjeva koriste se za traženje korisnika prema unesenu email adresu. Ako korisnik nije pronađen, javlja se greška, pri čemu se, radi sigurnosti, ne otkriva točna priroda greške. Ako korisnik s navedenom email adresom postoji, provodi se provjera lozinke; ako je lozinka netočna, vraća se ista poruka kao i kada je email netočan. Nakon toga se generira JSON WebToken pomoću biblioteke jsonwebtoken, koji se koristi zbog svoje sposobnosti da omogući sigurnu i efikasnu autentifikaciju korisnika bez potrebe za čuvanjem sesijskih podataka na poslužitelju. [11] U ovom

slučaju, korisnički ID uključuje se kao dio *payloada* tokena, koji ima ograničeno trajanje valjanosti od jednog sata, čime se dodatno povećava sigurnost.

```
34  export const loginUser = async (req, res) => {
35    try {
36      const { email, password } = req.body;
37      const user = await knex('users').where({ email }).first();
38
39      if (!user) {
40        return res.status(401).json({ message: 'Invalid email or password' });
41      }
42
43      const passwordMatch = await bcrypt.compare(password, user.password);
44      if (!passwordMatch) {
45        return res.status(401).json({ message: 'Invalid email or password' });
46      }
47
48      const token = jwt.sign({ userId: user.id }, process.env.JWT_SECRET, { expiresIn: '1h' });
49
50      res.status(200).json({ token });
51    } catch (error) {
52      console.error('Error logging in:', error);
53      res.status(500).json({ message: 'Internal server error' });
54    }
55  };
```

Slika 5.12 Funkcija *loginUser* u *authController.js*

Funkcija *getUserByEmail* sa slike 5.13 koristi se za dohvaćanje korisničkih podataka na temelju email adrese. Email adresa iz parametara zahtjeva koristi se za pronalaženje korisnika u tablici *users* pomoću biblioteke Knex. Ako korisnik s danom email adresom ne postoji, vraća se poruka o pogrešci s statusom 404. Ako korisnik postoji, korisnički podaci vraćaju se kao odgovor s statusom 200. U slučaju greške tijekom procesa dohvaćanja podataka, vraća se poruka o internoj pogrešci servera s statusom 500.

Ova funkcija koristi se u *frontendu* prilikom prijave korisnika kako bi se osigurale dodatne informacije o korisniku nakon uspješne autentifikacije. Nakon uspješne prijave korisnika, potreban je pristup detaljnim korisničkim podacima za postavljanje korisničkog okruženja i prikaz specifičnih informacija korisniku u *frontend* dijelu aplikacije. Dohvaćanjem korisničkih podataka putem emaila, dobivene su potrebne informacije koristeći JWT token za autorizaciju zahtjeva. Ovaj dodatni korak omogućuje lakše održavanje *koda* obzirom da u ovom slučaju svaka funkcija radi samo jednu stvar.

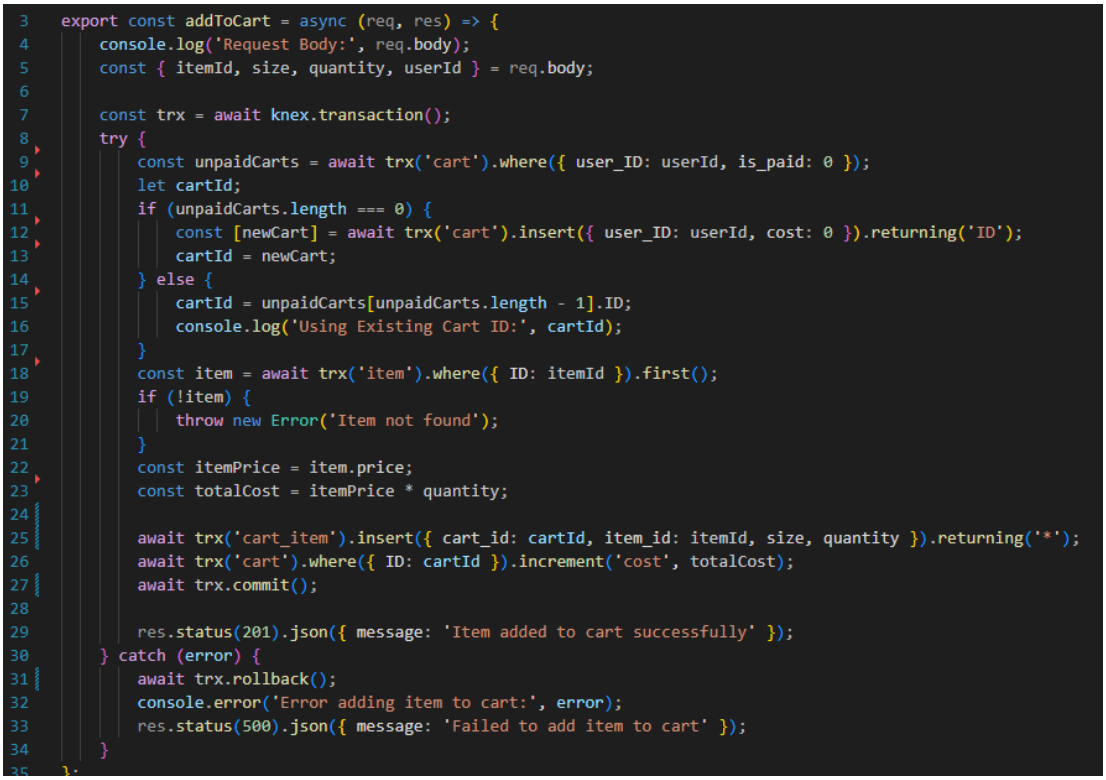
```
export const getUserByEmail = async (req, res) => {
  const { email } = req.params;

  try {
    const user = await knex('users').where({ email }).first();
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }
    console.log(user);
    res.status(200).json(user);
  } catch (error) {
    console.error('Error getting user by email:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
};
```

Slika 5.13 Funkcija *getUserByEmail* u *authController.js*

5.2.3. Košarica

Za rad s košaricom i proizvodima u košarici koristi se upravljač *cartController.js*, koji sadrži funkcije za dodavanje i brisanje proizvoda iz košarice, dohvaćanje trenutne aktivne košarice korisnika te funkcije za ažuriranje statusa košarice.

Funkcija *addToCart* sa slike 5.14 služi za dodavanje proizvoda u košaricu. Podaci iz tijela zahtjeva, koji uključuju ID artikla, veličinu, količinu i ID prijavljenog korisnika, koriste se u funkciji. Transakcija iz Knex biblioteke koristi se kako bi se osigurala nedjeljivo izvršavanje svih koraka procesa. Prvo se provjerava postoje li neplaćene košarice za korisnika; ako ne postoje, stvara se nova košarica. Ako postoje, koristi se najnovija neplaćena košarica. Zatim se dohvaća cijena artikla, računa ukupni trošak i artikl se dodaje u tablicu *cart_items*. Na kraju se ažurira ukupni trošak košarice i transakcija se potvrđuje. U slučaju greške, transakcija se poništava kako bi se osigurala konzistentnost podataka.



```
export const addToCart = async (req, res) => {
    console.log('Request Body:', req.body);
    const { itemId, size, quantity, userId } = req.body;

    const trx = await knex.transaction();
    try {
        const unpaidCarts = await trx('cart').where({ user_ID: userId, is_paid: 0 });
        let cartId;
        if (unpaidCarts.length === 0) {
            const [newCart] = await trx('cart').insert({ user_ID: userId, cost: 0 }).returning('ID');
            cartId = newCart;
        } else {
            cartId = unpaidCarts[unpaidCarts.length - 1].ID;
            console.log('Using Existing Cart ID:', cartId);
        }
        const item = await trx('item').where({ ID: itemId }).first();
        if (!item) {
            throw new Error('Item not found');
        }
        const itemPrice = item.price;
        const totalCost = itemPrice * quantity;

        await trx('cart_item').insert({ cart_id: cartId, item_id: itemId, size, quantity }).returning('*');
        await trx('cart').where({ ID: cartId }).increment('cost', totalCost);
        await trx.commit();

        res.status(201).json({ message: 'Item added to cart successfully' });
    } catch (error) {
        await trx.rollback();
        console.error('Error adding item to cart:', error);
        res.status(500).json({ message: 'Failed to add item to cart' });
    }
};
```

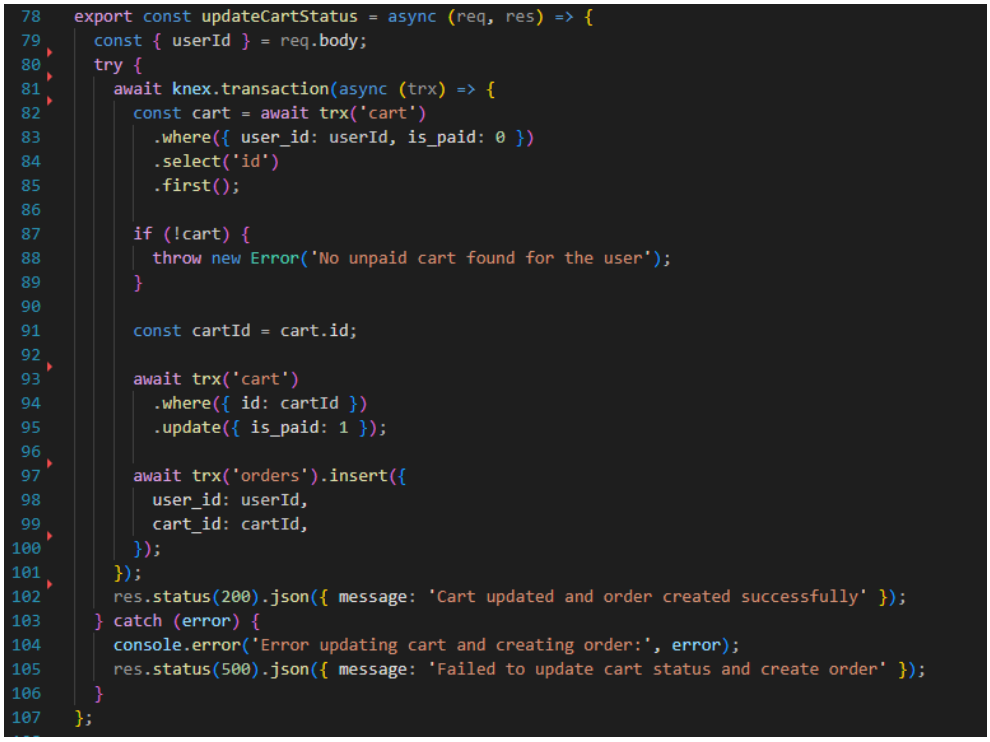
Slika 5.14 Funkcija *addToCart* u *cartController.js*

Funkcija *getActiveCart* sa slike 5.15 koristi se za dohvaćanje aktivne košarice korisnika, tj. košarice koja još nije plaćena. *UserId* se dohvaća iz URL parametara zahtjeva, a pomoću Knex upita spajaju se tablice *cart, cart_item, item* i *images* kako bi se prikupili svi potrebni podaci o košarici i stavkama unutar nje. Dohvaćeni podaci uključuju identifikator košarice, status plaćenosti, ukupni trošak, detalje stavki (naziv, cijena, opis) te grupirane slike. Ako ne postoji aktivna košarica za korisnika, vraća se HTTP status 404 s porukom "Nema aktivne košarice". Ako je aktivna košarica pronađena, grupirane slike razdvajaju se u nizove kako bi sve slike bile pravilno prikazane u *frontend* dijelu aplikacije. Konačno, obrađeni podaci vraćaju se kao JSON odgovor s HTTP statusom 200. U slučaju greške tijekom izvršavanja upita, ispisuje se poruka o pogrešci u konzolu i vraća se JSON odgovor s porukom "Neuspješno dohvaćanje aktivne košarice" te HTTP statusom 500, čime se osigurava pravilno rukovanje pogreškama i obavještavanje korisnika o problemima.

```
export const getActiveCart = async (req, res) => {
  const { userId } = req.params;
  try {
    const cart = await knex('cart')
      .leftJoin('cart_item', 'cart.ID', 'cart_item.cart_id')
      .leftJoin('item', 'cart_item.item_id', 'item.ID')
      .leftJoin('images', 'item.ID', 'images.item_id')
      .select(
        'cart.ID as cartId',
        'cart.is_paid',
        'cart.cost',
        'cart_item.*',
        'item.name',
        'item.price',
        'item.description',
        knex.raw('GROUP_CONCAT(images.image) as images')
      )
      .where({ 'cart.user_ID': userId, 'cart.is_paid': 0 })
      .groupBy('cart.ID', 'cart_item.ID', 'item.ID');

    if (cart.length === 0) {
      return res.status(404).json({ message: 'No active cart found' });
    }

    const cartWithImages = cart.map(cartItem => ({
      ...cartItem,
      images: cartItem.images ? cartItem.images.split(',') : []
    }));

    res.status(200).json(cartWithImages);
  } catch (error) {
    console.error('Error fetching active cart:', error);
    res.status(500).json({ message: 'Failed to fetch active cart' });
  }
};
```

Slika 5.15 Funkcija *getActiveCart* u *cartController.js*

Funkcija *updateCartStatus* sa slike 5.16 koristi se za završetak narudžbe, pri čemu se status košarice mijenja iz *is_paid = 0* u *is_paid = 1* i stvara novi redak u tablici *orders*. *UserId* se preuzima iz tijela zahtjeva i koristi se transakcija kako bi se osigurala konzistentnost podataka tijekom operacije. Prvo se provjerava postoji li neplaćena košarica za korisnika; ako ne postoji, javlja se greška. Ako se pronađe neplaćena košarica, njezin status se ažurira na „plaćena“ (*is_paid*: 1), a nova narudžba unosi se u tablicu *orders*, s ID-jem korisnika i ID-jem košarice. Nakon završetka transakcije, vraća se odgovor s porukom da je košarica ažurirana i narudžba kreirana, s HTTP statusom 200. U slučaju greške tijekom izvršavanja, ispisuje se poruka o grešci u konzolu i vraća se odgovor s porukom da je ažuriranje statusa košarice neuspješno te da narudžba u tablici *orders* nije kreirana, s HTTP statusom 500.



```
export const updateCartStatus = async (req, res) => {
  const { userId } = req.body;
  try {
    await knex.transaction(async (trx) => {
      const cart = await trx('cart')
        .where({ user_id: userId, is_paid: 0 })
        .select('id')
        .first();

      if (!cart) {
        throw new Error('No unpaid cart found for the user');
      }

      const cartId = cart.id;

      await trx('cart')
        .where({ id: cartId })
        .update({ is_paid: 1 });

      await trx('orders').insert({
        user_id: userId,
        cart_id: cartId,
      });
    });
    res.status(200).json({ message: 'Cart updated and order created successfully' });
  } catch (error) {
    console.error('Error updating cart and creating order:', error);
    res.status(500).json({ message: 'Failed to update cart status and create order' });
  }
};
```

Slika 5.16 Funkcija *updateCartStatus* u *cartController.js*

Funkcija *removeItemFromCart* sa slike 5.17 koristi se za uklanjanje artikala iz košarice korisnika. Parametar *cartItemId* iz URL-a koristi se za pronalazak odgovarajućeg zapisa u tablici *cart_item*. Prvo se provjerava postoji li stavka u košarici s danim ID-om; ako stavka ne postoji, vraća se poruka da proizvod u košarici nije pronađen s HTTP statusom 404. Nakon pronalaska stavke, njezina cijena dohvaća se iz tablice *item*, zatim se pronalazi trenutna cijena košarice i ažurira oduzimanjem cijene uklonjene stavke. Stavka se briše iz tablice *cart_item*. Ako nakon brisanja u košarici ne preostane nijedna stavka, košarica se također briše, a vraća se poruka da je zadnji proizvod izbrisan iz košarice te je cijela košarica izbrisana (engl. *„Last item removed, cart deleted successfully“).* U suprotnom, cijena košarice se ažurira i vraća se poruka da je proizvod uspješno izbrisan iz košarice, zajedno s novom cijenom košarice. U slučaju pogreške tijekom izvršavanja, ispisuje se poruka o pogrešci u konzolu i vraća se odgovor s porukom da je brisanje iz košarice neuspješno s HTTP statusom 500.

```
export const removeItemFromCart = async (req, res) => {
  const { cartItemId } = req.params;
  try {
    const cartItem = await knex('cart_item').where('ID', cartItemId).first();
    if (!cartItem) {
      return res.status(404).json({ message: 'Cart item not found' });
    }
    const { cart_id: cartId, item_id: itemId } = cartItem;

    const item = await knex('item').where('ID', itemId).first();
    if (!item) {
      return res.status(404).json({ message: 'Item not found' });
    }
    const { price } = item;

    const cart = await knex('cart').where('ID', cartId).first();
    if (!cart) {
      return res.status(404).json({ message: 'Cart not found' });
    }

    let { cost } = cart;
    cost -= price;

    await knex('cart_item').where('ID', cartItemId).del();
    const remainingItems = await knex('cart_item').where('cart_id', cartId).count('* as count').first();

    if (remainingItems.count === 0) {
      await knex('cart').where('ID', cartId).del();
      res.status(200).json({ message: 'Last item removed, cart deleted successfully' });
    } else {
      await knex('cart').where('ID', cartId).update({ cost });
      res.status(200).json({ message: 'Item removed from cart successfully', cost });
    }
  } catch (error) {
    console.error('Error removing item from cart:', error);
    res.status(500).json({ message: 'Failed to remove item from cart' });
  }
};
```

Slika 5.17 Funkcija *removeItemFromCart* u *cartController.js*

5.2.4. Košarica za neregistrirane korisnike

Upravljač *guestCartController.js* sadrži funkcije za upravljanje košaricama neregistriranih korisnika: *addToGuestCart* koristi se za dodavanje u košaricu gosta, *getGuestId* služi za dobivanje identifikacijskog broja gosta, *getGuestActiveCart* koristi se za dobivanje aktivne košarice gosta, *updateGuestCartStatus* ažurira status košarice gosta, *removeItemFromGuestCart* koristi se za brisanje proizvoda iz košarice gosta, dok *getGuestCartId* služi za dobivanje identifikacijskog broja košarice gosta. Te sve funkcije postavljene su za krajnje točke (engl. *endpoints*) u *guestCartRoutes.js* datoteci prikazanoj na slici 5.18.

```
router.post('/add-to-guest-cart', addToGuestCart);
router.get('/create', getGuestId);
router.get('/active-cart/:guestId', getGuestActiveCart);
router.post('/update-guest-cart-status', updateGuestCartStatus);
router.delete('/remove-item-from-guest-cart/:cartItemId', removeItemFromGuestCart);
router.get('/guest-cart-id/:guestId', getGuestCartId);
```

Slika 5.18 *guestCartRoutes.js* datoteka

5.2.5. Kategorije

Upravljač *categoryController.js* sadrži funkciju *getAllCategories*. Funkcija *getAllCategories* prikazana na slici 5.19 koristi se za dohvaćanje svih kategorija iz tablice *categories* u bazi podataka. Kada se funkcija pozove, pokušava se dohvatiti svi zapisi iz tablice *categories*, birajući samo stupac *category*. Ako je dohvaćanje uspješno, dohvaćene kategorije ispisuju se u konzolu i vraćaju se kao JSON odgovor s HTTP statusom 200.

```
import knex from '../../db/knex.js';

export const getAllCategories = async (req, res) => {
    console.log('getAllCategories controller function called');
    try {
        const categories = await knex('categories').select('category');
        res.status(200).json(categories);
    } catch (error) {
        console.error('Error fetching categories:', error);
        res.status(500).json({ message: 'Internal server error' });
    }
};
```

Slika 5.19 Funkcija *getAllCategories* u *categoryController.js*

5.2.6. Proizvodi

Upravljač *itemController.js* sadrži funkcije za dohvaćanje svih proizvoda, dohvaćanje jednog proizvoda prema njegovom ID-u, te funkcije za kreiranje, brisanje i ažuriranje proizvoda, koje su dostupne administratorima web aplikacije.

Funkcija *getItems* prikazana na slici 5.20 koristi se za dohvaćanje proizvoda iz baze podataka, s mogućnošću filtriranja prema kategoriji. Knex.js se koristi za kreiranje SQL upita koji pridružuje tablicu *images* s tablicom *items*, grupirajući sve slike povezane s određenom stavkom. Ako je kategorija specificirana u upitnim parametrima URL-a, upit se dodatno filtrira prema ID-u kategorije. Nakon izvršavanja upita, rezultati se obrađuju tako da se sve slike za svaki proizvod razdvajaju u niz. Obrađeni podaci zatim se vraćaju kao JSON odgovor s HTTP statusom 200, omogućujući *frontend* dijelu aplikacije pristup potrebnim podacima za prikaz proizvoda i njihovih slika. U slučaju pogreške tijekom izvršavanja upita, ispisuje se poruka o pogrešci u konzolu i vraća se JSON odgovor s porukom „Interna pogreška poslužitelja“ (engl. „*Internal server error*“) te HTTP statusom 500.

```
export const getItems = async (req, res) => {
  const { category } = req.query;
  try {
    let query = knex('item')
      .leftJoin('images', 'item.id', 'images.item_id')
      .select('item.*', knex.raw('GROUP_CONCAT(images.image) as images'))
      .groupBy('item.id');

    if (category) {
      query = query.where('item.category_id',
        knex('categories')
        .select('id')
        .where('category', category));
    }
    const items = await query;

    const itemsWithImages = items.map(item => ({
      ...item,
      images: item.images ? item.images.split(',') : []
    }));

    res.status(200).json(itemsWithImages);
  } catch (error) {
    console.error('Error fetching items:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
};
```

Slika 5.20 Funkcija *getItems* u *itemController.js*

Funkcija *getItemById* prikazana na slici 5.21 koristi se za dohvaćanje pojedinog proizvoda iz tablice *items* prema zadanom ID-u. Osnovni podaci proizvoda dohvaćaju se iz tablice *items* koristeći Knex, a zatim se provjerava postoji li proizvod s navedenim ID-em. Ako proizvod ne postoji, vraća se HTTP status 404 i poruka „Proizvod nije pronađen“ (engl. „*Item not found*“). Ako je proizvod pronađen, dodatno se dohvaćaju sve pridružene slike iz tablice *images* i 3D modeli iz tablice *models*, te se dodaju u objekt *item*. Rezultat se zatim vraća kao JSON odgovor s HTTP statusom 200, omogućujući *frontend* dijelu aplikacije pristup svim relevantnim podacima o proizvodu, uključujući slike i modele. U slučaju pogreške tijekom izvršavanja upita, ispisuje se poruka o pogrešci u konzolu i vraća se JSON odgovor s porukom „Interna pogreška poslužitelja“ te HTTP statusom 500, čime se osigurava pravilno rukovanje pogreškama i obavještavanje korisnika o problemima.

```
export const getItemById = async (req, res) => {
  const { id } = req.params;
  try {
    const item = await knex('item')
      .where({ id })
      .first();

    if (!item) {
      return res.status(404).json({ message: 'Item not found' });
    }

    const images = await knex('images')
      .where('item_id', id)
      .select('image');

    const models = await knex('models')
      .where('item_id', id)
      .select('model');

    item.images = images.map(img => img.image);
    item.models = models.map(mod => mod.model);

    res.status(200).json(item);
  } catch (error) {
    console.error('Error fetching item:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
};
```

Slika 5.21 Funkcija *getItemById* u *itemController.js*

5.2.7. Spotify API

U upravljaču *spotifyController.js* postoji jedna krajnja točka ali su potrebne dvije funkcije za dohvaćanje nasumične pjesme iz nekog popisa za reprodukciju. Za Node.js postoji biblioteka za korištenje Spotifyevog API-ja: spotify-web-api-node.

Kod prikazan na slici 5.22 prikazuje stvaranje instance klase SpotifyWebApi sa tajnim ključevima. Kako bi Spotify API bio osposobljen potrebno je izraditi račun i aplikaciju na web stranici „Spotify for developers“. [12]

```
const spotifyApi = new SpotifyWebApi({
  clientId: process.env.SPOTIFY_CLIENT_ID,
  clientSecret: process.env.SPOTIFY_CLIENT_SECRET,
});
```

Slika 5.22 Stvaranje instance klase *SpotifyWebApi*

Kod prikazan na slici 5.23 koristi Spotify API za dohvaćanje nasumičnih pjesama iz određene Spotify playliste. Funkcija *getAccessToken* koristi se za dobivanje pristupnog tokena pomoću

Spotify API-jevog „Client Credentials“ toka. Token je zatim postavljen na *spotifyApi* instancu kako bi se omogućili daljnji autorizirani API pozivi. Funkcija *getRandomSongsFromPlaylist* dohvaća ID popisa za reprodukciju iz URL parametara zahtjeva poslanog s frontend dijela aplikacije, poziva *getAccessToken* za osvježavanje tokena, a zatim koristi *spotifyApi.getPlaylistTracks* za dobivanje svih pjesama iz navedenog popisa za reprodukciju. Nakon što su pjesme dohvaćene, pet pjesama se nasumično odabire i šalje kao JSON odgovor klijentu. U slučaju bilo kakve greške, greška se zapisuje u konzolu i HTTP status 500 šalje se kao odgovor.

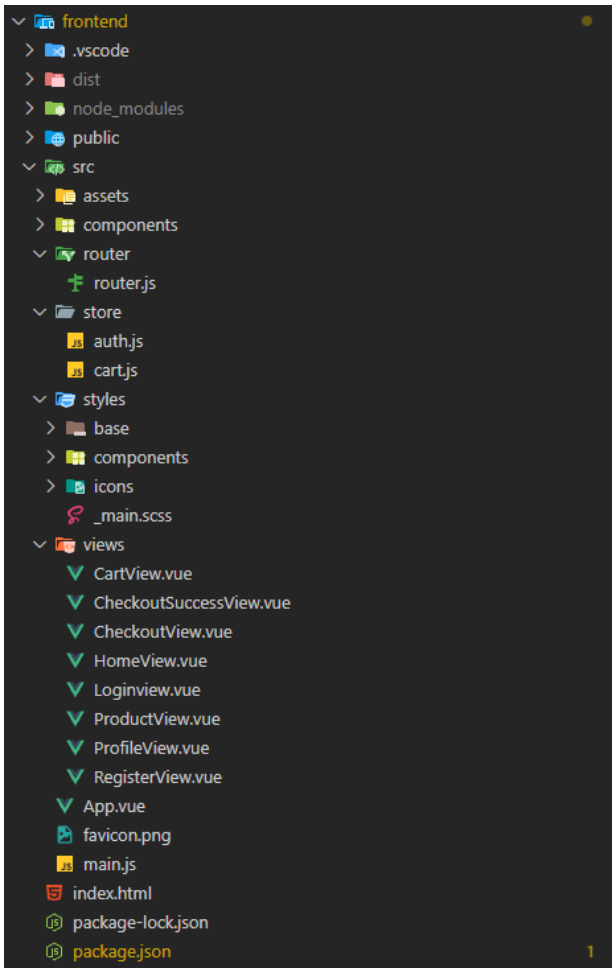
```
13  async function getAccessToken() {
14    try {
15      const data = await spotifyApi.clientCredentialsGrant();
16      spotifyApi.setAccessToken(data.body['access_token']);
17    } catch (error) {
18      console.error('Something went wrong when retrieving an access token', err);
19    }
20  }
21
22  export async function getRandomSongsFromPlaylist(req, res) {
23    const playlistId = req.params.playlistId;
24    try {
25      await getAccessToken();
26
27      const data = await spotifyApi.getPlaylistTracks(playlistId);
28      const tracks = data.body.items;
29
30      if (tracks.length === 0) {
31        return res.status(404).json({ message: 'No tracks found in the playlist' });
32      }
33
34      const randomIndex = Math.floor(Math.random() * tracks.length);
35      const randomTrack = tracks[randomIndex].track;
36
37      res.json(randomTrack);
38    } catch (error) {
39      console.error('Error retrieving playlist tracks', error);
40      res.status(500).send('Internal Server Error');
41    }
42  }
```

Slika 5.23 Funkcija *getAccessToken* i *getRandomSongsFromPlaylist* u *spotifyController.js*

5.3. Frontend

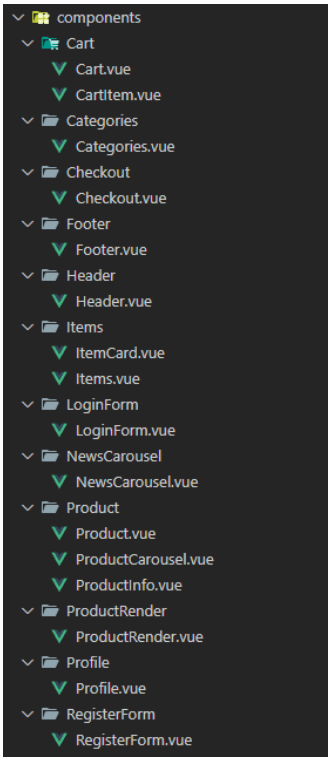
U ovom poglavlju obrađuje se način na koji je riješen *frontend* dio ove web aplikacije koristeći Vue.js.

Na slici 5.24 prikazan je način na koji su posložene datoteke u *frontend* mapi radi bolje preglednosti i lakšeg refaktoriranja *koda*.



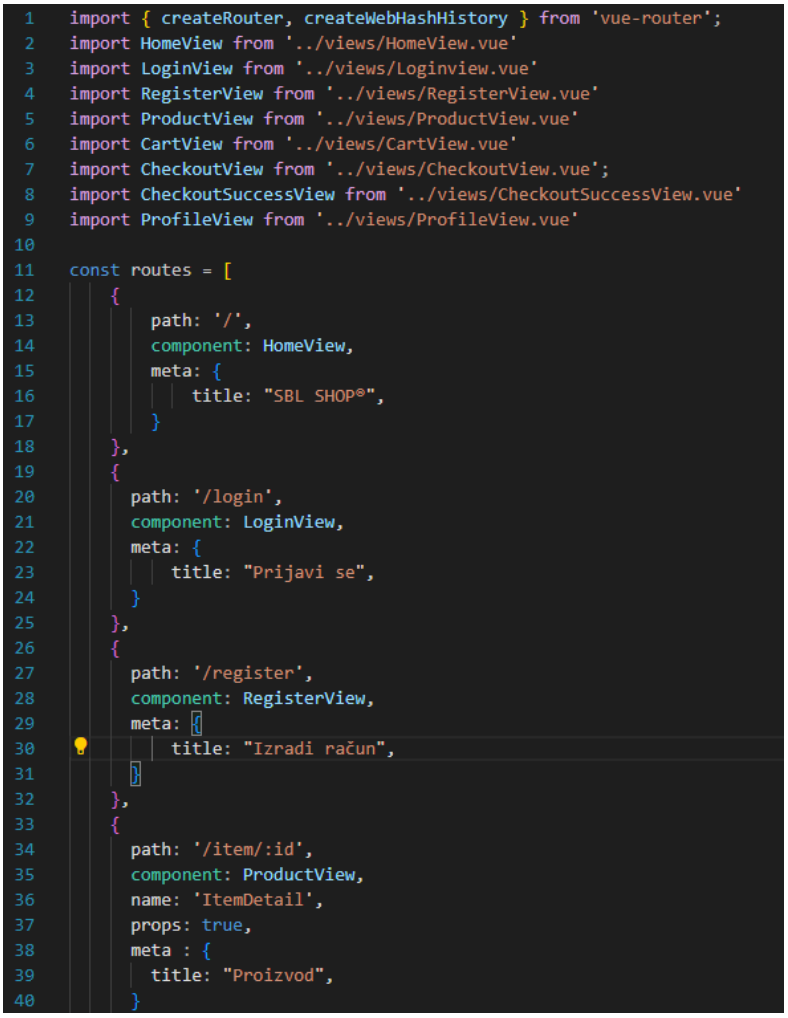
Slika 5.24 Struktura frontend mape

Na slici 5.25 prikazane su mape s Vue komponentama koje se koriste unutar svih prikaza.



Slika 5.25 Mape s Vue komponentama

Na slici 5.26 prikazan je dio datoteke router.js u kojoj su definirane *frontend* rute od kojih se svaka koristi za prikaz jednog *viewa*. Određuje se putanja i naslov za svaki *view* unutar svih ruta.



```
import { createRouter, createWebHashHistory } from 'vue-router';
import HomeView from '../views/HomeView.vue'
import LoginView from '../views/Loginview.vue'
import RegisterView from '../views/RegisterView.vue'
import ProductView from '../views/ProductView.vue'
import CartView from '../views/CartView.vue'
import CheckoutView from '../views/CheckoutView.vue';
import CheckoutSuccessView from '../views/CheckoutSuccessView.vue'
import ProfileView from '../views/ProfileView.vue'

const routes = [
    {
        path: '/',
        component: HomeView,
        meta: {
            title: "SBL SHOP®",
        }
    },
    {
      path: '/login',
      component: LoginView,
      meta: {
          title: "Prijavi se",
      }
    },
    {
      path: '/register',
      component: RegisterView,
      meta: {
          title: "Izradi račun",
      }
    },
    {
      path: '/item/:id',
      component: ProductView,
      name: 'ItemDetail',
      props: true,
      meta : {
        title: "Proizvod",
      }
```

Slika 5.26 Datoteka *router.js*

5.3.1. Početni prikaz

U ovom poglavlju detaljno je obrađen pisani dio koda za *frontend* dio aplikacije.

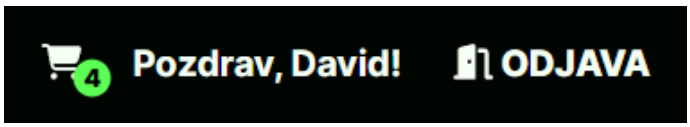
Korisničko sučelje početnog prikaza prikazano je na slici 4.1 .

Na slici 5.27 prikazana je datoteka *HomeView.vue* koja prikazuje strukturu komponenata za početnu stranicu.

```vue
<script setup>
    import Header from '../components/Header/Header.vue';
    import Items from '../components/Items/Items.vue';
    import Footer from '../components/Footer/Footer.vue';
</script>

<template>
    <Header></Header>
    <Items></Items>
    <Footer></Footer>
</template>
```

Slika 5.27 Datoteka *HomeView.vue*

Na slikama 5.29, 5.30 i 5.31 prikazan je *kod* korišten za osposobljavanje elemenata korisničkog sučelja na slici 5.28.



Slika 5.28 Brojač proizvoda u košarici, prikazivanje imena korisnika i gumb za odjavu u *headeru*

```
import { ref } from "vue";
import axios from "axios";

const cartItemCount = ref(0);
const sum = ref(0);

const updateCart = (count, total) => {
    cartItemCount.value = count;
    sum.value = total;
};

const fetchCartItemCount = async (userId) => {
    try {
        const response = await axios.get(`/cart/active-cart/${userId}`);
        const itemCount = response.data.length;
        let totalQuantity = 0;
        for (let i = 0; i < response.data.length; i++) {
            totalQuantity += response.data[i].quantity;
        }
        updateCart(itemCount, totalQuantity);
    } catch (error) {
        if (error.response && error.response.status === 404) {
            updateCart(0, 0);
        } else {
            console.error("Failed to fetch cart item count:", error);
        }
    }
};

export { cartItemCount, sum, updateCart, fetchCartItemCount };
```

Slika 5.29 Skripta cart.js

```
import { reactive, toRefs } from 'vue';
import axios from 'axios';

const state = reactive({
  user: null,
});

const storedUser = JSON.parse(localStorage.getItem('user'));
if (storedUser) {
  state.user = storedUser;
}

export const setUser = (user) => {
  console.log('Setting user:', user);
  state.user = user;
  localStorage.setItem('user', JSON.stringify(user));
};

export const useUser = () => {
  return toRefs(state);
};

export const clearUser = () => {
  state.user = null;
  localStorage.removeItem('user');
};
```

Slika 5.30 Dio skripte auth.js korišten u headeru

```vue
<script setup>
import { ref, computed, onMounted, watch } from "vue";
import { useUser, clearUser } from "../../store/auth.js";
import Heart from "../../assets/icons/Heart.vue";
import CartIcon from "../../assets/icons/CartIcon.vue";
import LogOutIcon from "../../assets/icons/LogOutIcon.vue";
import { useRouter } from "vue-router";
import { cartItemCount, sum, fetchCartItemCount } from "../../store/cart.js";

const router = useRouter();

const { user } = useUser();

const fetchCartItemCountForUser = async () => {
    if (user.value) {
        await fetchCartItemCount(user.value.ID);
    }
};

onMounted(() => {
    fetchCartItemCountForUser();
});

watch(user, () => {
    fetchCartItemCountForUser();
});

const logout = () => {
    clearUser();
    fetchCartItemCountForUser();
    router.push("/");
};

const hasItemsInCart = computed(() => cartItemCount.value > 0);
</script>
```

Slika 5.31 <script setup> dio *Header.vue* komponente

Na slici 5.28 prikazan je desni dio *headera* u kojoj postoji gumb za košaricu koji također prikazuje trenutni broj proizvoda u košarici. Ovo je osposobljeno pomoću zasebne skripte *cart.js* sa slike 5.29. Datoteka cart.js sadrži logiku za upravljanje brojem artikala u košarici i ukupnom količinom artikala. Korištenjem Vueove *ref* funkcije, definira dva reaktivna stanja: *cartItemCount* i *sum*. Ovi reaktivni podaci omogućuju automatsko ažuriranje korisničkog sučelja kada se promijeni broj artikala ili ukupna količina u košarici. Na slici 5.31 prikazana je i funkcija *onMounted* koja se poziva prilikom inicijalizacije komponente *Header.vue* i ona u ovom slučaju poziva funkciju *fetchCartItemCountForUser* koja ako je korisnik prijavljen poziva funkciju *fetchCartItemCount* s parametrom identifikacijskog broja tog korisnika. Nakon košarice u korisničkom sučelju postoji pozdrav korisniku koji prikazuje njegovo ime, ovo je također osposobljeno pomoću zasebne skripte, *auth.js* sa slike 5.30 na kojoj je prikazan samo prvi dio skripte koji se koristi unutar *headera*. Na početku se stvara reaktivni objekt *state* koji sadrži podatke korisnika nakon prijave. Korisnik se nakon prijave sprema u *localStorage* što je detaljnije objašnjeno u sljedećem poglavlju. Skripta *auth.js* sadrži i funkcije *useUser* i *clearUser*, čije je korištenje prikazano na slici 5.31. Funkcija *useUser* se koristi za dohvaćanje podataka o korisniku, dok se funkcija *clearUser* koristi prilikom odjave.

```vue
<template>
    <div>
        <Categories @categorySelected="fetchItems" />
        <div v-if="items.length" class="items-container">
            <ItemCard v-for="item in items" :item="item" :key="item.id" />

        </div>
        <div v-else class="items-container no-items">
            <h1>Trenutno nema proizvoda u ovoj kategoriji.</h1>
        </div>
    </div>
</template>

<script setup>
import { ref } from "vue";
import Categories from "../Categories/Categories.vue";
import ItemCard from "./ItemCard.vue";
import axios from "axios";

const items = ref([]);

const fetchItems = async (category) => {
    try {
        const response = await axios.get(`http://localhost:3000/items`, {
            params: { category }
        });

        items.value = response.data;
    } catch (error) {
        console.error("Error fetching items:", error);
    }
};
</script>
```

Slika 5.32 *Items.vue* komponenta

Na slici 5.32 prikazana je komponenta *Items* koja se koristi za prikazivanje proizvoda unutar odabrane kategorije. Ova komponenta koristi *Categories* komponentu za prikaz svih kategorija iz baze i omogućuje odabir kategorija. Također se koristi *ItemCard* komponenta za prikaz pojedinačnih proizvoda, pri čemu se pomoću Vueove v-for petlje prikazuju svi proizvodi u kategoriji. Glavni dio predloška sadrži uvjetni prikaz koji provjerava broj proizvoda unutar kategorije; ako postoji barem jedan proizvod, prikazuje se unutar *items-container* div elementa, gdje se svaki proizvod prikazuje pomoću *ItemCard* komponente. Ako nema proizvoda, prikazuje se poruka „Trenutno nema proizvoda u ovoj kategoriji“.

Unutar <*script setup*> dijela komponente definira se reaktivna varijabla *items* pomoću Vue-ove *ref* funkcije koja sadrži popis proizvoda. Funkcija *fetchItems* koristi se za dohvaćanje proizvoda iz *backend* API-ja kada je odabrana neka kategorija. Ova funkcija šalje *GET* zahtjev na *http://localhost:3000/items* s parametrima kategorije i ažurira *items* s dobivenim podacima. Ako dođe do iznimke tijekom dohvaćanja podataka, greška se ispisuje u konzolu. Ova komponenta omogućava dinamičko prikazivanje proizvoda ovisno o odabranoj kategoriji, čime se korisničko iskustvo čini brzim i intuitivnim.

5.3.2. Prikaz za prijavu korisnika

Korisničko sučelje prikaza za prijavu korisnika prikazano je na slici 4.2.

```vue
import { loginUser } from "../../store/auth.js";
const email = ref("");
const password = ref("");
const router = useRouter();
const errorMessage = ref("");

const login = async () => {
  try {
    errorMessage.value = "";
    await loginUser(email.value, password.value, router);
  } catch (error) {
    console.error("Login failed:", error.message);
    errorMessage.value = "Invalid email or password";
  }
};
```

Slika 5.33 <script setup> dio *LoginForm.vue* komponente koju koristi *LoginView* prikaz

```vue
export const loginUser = async (email, password, router) => {
  try {
    const response = await axios.post("http://localhost:3000/auth/login", {
      email,
      password,
    });

    if (response.status !== 200) {
      throw new Error("Login failed: Invalid response");
    }

    const token = response.data.token;
    localStorage.setItem("token", token);
    const userInfoResponse = await axios.get(
      `http://localhost:3000/auth/user/${email}`,
      {
        headers: {
          Authorization: `Bearer ${token}`,
        },
      }
    );
    if (userInfoResponse.status !== 200) {
      throw new Error("Login failed: Invalid user info response");
    }

    const userInfo = userInfoResponse.data;
    setUser(userInfo);
    router.push("/");

  } catch (error) {
    console.error("Login failed:", error.message);
    throw error;
  }
};
```

Slika 5.34 *loginUser* funkcija iz skripte *auth.js*

Prilikom prijave, unose se email adresa i lozinka, nakon čega se poziva funkcija *login* prikazana na slici 5.33 koja zatim aktivira funkciju *loginUser* iz *auth.js* skripte prikazane na slici 5.34. Funkcija *loginUser* se koristi za autentifikaciju korisnika putem API poziva na */auth/login* krajnju točku. Prvo se šalje *POST* zahtjev na *backend server* s email adresom i lozinkom korisnika, a ako odgovor nije 200, vraća se greška.

Ako nema greške, dobiveni JWT token sprema se u *localStorage* i koristi za autorizaciju *GET* zahtjeva na krajnju točku */auth/user/${email}* kako bi se dohvatili podaci o korisniku. Ako je i ovaj zahtjev uspješan, podaci o korisniku se spremaju lokalno pomoću *setUser* funkcije i korisnik se preusmjerava na početnu stranicu aplikacije pomoću Vue *Routera* (*router.push("/")*).

5.3.3. Prikaz za registraciju korisnika

Korisničko sučelje prikaza za registraciju korisnika prikazano je na slici 4.2.

Na slici 5.35 prikazan je <*script setup*> dio *RegisterForm.vue* komponente. U ovoj komponenti definiraju se reaktivne reference za sve korisničke unose, kao i *errorMessage* za prikaz grešaka. Funkcija *register* prvo resetira *errorMessage*, a zatim provjerava podudaranje lozinki. Ako se lozinke ne podudaraju, prikazuje se odgovarajuća poruka na ekranu. Ako se lozinke podudaraju, šalje se *POST* zahtjev na */auth/register* krajnju točku sa svim podacima koje je korisnik unio. Ako registracija bude uspješna, automatski se pokušava prijaviti korisnika pozivajući funkciju *loginUser* i pomoću Vue *Routera* preusmjerava korisnika na početnu stranicu.

```
const firstname = ref("");
const lastname = ref("");
const email = ref("");
const password = ref("");
const confirmPassword = ref("");
const address = ref("");
const city = ref("");
const postal_code = ref("");
const errorMessage = ref("");
const router = useRouter();

const register = async () => {
    errorMessage.value = "";  // Reset the error message
    if (password.value !== confirmPassword.value) {
        errorMessage.value = "Lozinke se ne podudaraju";
        return;
    }

    try {
        await axios.post("/auth/register", {
            firstname: firstname.value,
            lastname: lastname.value,
            email: email.value,
            password: password.value,
            address: address.value,
            city: city.value,
            postal_code: postal_code.value,
        });
        try {
            await loginUser(email.value, password.value, router);
        } catch (error) {
            errorMessage.value = "Prijava nije uspjela";
            console.error("Login failed:", error.message);
        }

    } catch (error) {
        errorMessage.value = "Registracija nije uspjela";
        console.error("Registration failed:", error);
    }
};
</script>
```

Slika 5.35 <*script setup*> dio komponente *RegisterForm.vue* koju koristi *RegisterView* prikaz

5.3.4. Prikaz pojedinog proizvoda

Korisničko sučelje prikaza pojedinog proizvoda prikazano je na slici 4.3.

Na slici 5.36 prikazan je <*template*> dio *Product.vue* komponente. U ovoj komponenti koriste se tri dodatne komponente, komponenta *ProductRender.vue* koristi se za prikaz 3D modela vezanog za proizvod, komponenta *ProductCarousel.vue* koristi se za prikaz slika u obliku galerije slika, a *ProductInfo.vue* komponenta služi za prikaz informacija o proizvodu i omogućivanje odabira veličine i količine proizvoda prije dodavanja u košaricu.

```vue
<template>
  <div class="single-product-container" v-if="item">
    <article class="product-view-container">
      <section class="product-visuals-container">
        <ProductRender :modelPath="modelPath"></ProductRender>
        <ProductCarousel :images="images"></ProductCarousel>
      </section>
      <ProductInfo :item="item"></ProductInfo>
    </article>
  </div>
  <div v-else>
    <p>Loading...</p>
  </div>
</template>
```

Slika 5.36 <*template*> dio *Product.vue* komponente koja se koristi u *ProductView* prikazu

Na slici 5.37 prikazan je <*script setup*> dio *Product.vue* komponente koji šalje zahtjev na krajnju točku za dohvaćanje pojedinačnog proizvoda pomoću njegovog identifikacijskog broja. Nakon što se proizvod dohvati, vrijednosti objekata *item*, *images* i *modelPath* postavljaju se na vrijednosti dobivene iz baze podataka.

```vue
<script setup>
import { ref, onMounted } from 'vue';
import { useRoute } from 'vue-router';
import ProductCarousel from './ProductCarousel.vue';
import ProductRender from '../ProductRender/ProductRender.vue';
import ProductInfo from './ProductInfo.vue';
import axios from 'axios';

const route = useRoute();
const item = ref(null);
const images = ref([]);
const modelPath = ref('');

const fetchItem = async () => {
  try {
    const response = await axios.get(`http://localhost:3000/items/${route.params.id}`);
    item.value = response.data;
    images.value = response.data.images;
    modelPath.value = response.data.models[0];

    document.title = response.data.name;
  } catch (error) {
    console.error('Error fetching item:', error);
  }
};

onMounted(fetchItem);
</script>
```

Slika 5.37 <*script setup*> dio *Product.vue* komponente koja se koristi u *ProductView* prikazu

5.3.5. Prikaz košarice

Korisničko sučelje prikaza košarice prikazano je lijevoj strani slike 4.4.

Prikaz košarice *CartView* koristi *Cart.vue* komponentu.

Na slici 5.38 prikazana je funkcija *fetchActiveCart* koja koristi funkciju *useUser* iz skripte *auth.js* za slanje *GET* zahtjeva krajnjoj točki koja dohvaća aktivnu košaricu korisnika. Funkcija zatim provjerava postoji li aktivna košarica s barem jednim proizvodom; ako košarica ne sadrži proizvode, ispisuje se poruka da je košarica prazna.

```
const fetchActiveCart = async () => {
  try {
    const response = await axios.get(`/cart/active-cart/${user.value.ID}`);
    if (response.data.length === 0) {
      cartItems.value = [];
      totalCost.value = 0;
    } else {
      cartItems.value = response.data;
      totalCost.value = cartItems.value[0].cost;
    }

    itemsInCartAmount.value = cartItems.value.length;

    console.log("Fetched cart items:", cartItems.value);
  } catch (error) {
    console.error("Error fetching active cart:", error);
    if (error.response && error.response.status === 404) {
      cartItems.value = [];
      totalCost.value = 0;
    } else {
      console.error("Failed to fetch active cart:", error);
    }
  }
};
```

Slika 5.38 Funkcija *fetchActiveCart* u *Cart.vue* komponenti

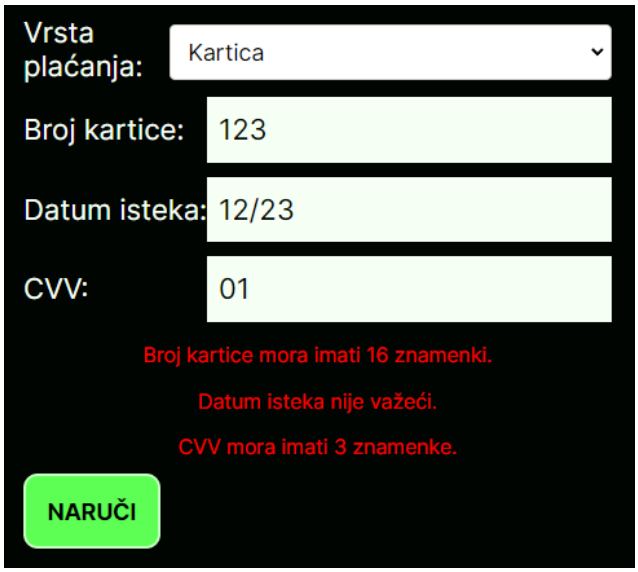
Na slici 5.39 prikazana je funkcija *removeItemFromCart* koja se koristi za uklanjanje stavke iz korisničke košarice. Ovom funkcijom prvo se šalje *DELETE* zahtjev na */cart/remove-item/{cartItemId}*, gdje *{cartItemId}* predstavlja ID stavke koja se uklanja. Ako je zahtjev uspješno proveden, poziva se *fetchActiveCart* funkcija kako bi se dohvatile ažurirane stavke u košarici te se koristi funkcija *fetchCartItemCount* za ažuriranje broja stavki u košarici. Nakon toga, emitira se događaj *removeItem* kako bi roditeljska komponenta mogla reagirati na promjenu.

```
const removeItemFromCart = async (cartItemId) => {
  try {
    await axios.delete(`/cart/remove-item/${cartItemId}`);
    await fetchActiveCart();
    await fetchCartItemCount(user.value.ID);
    emit('removeItem');
  } catch (error) {
    console.error("Failed to remove item from cart:", error);
  }
};
```

Slika 5.39 Funkcija *removeItemFromCart* u *Cart.vue* komponenti

5.3.6. Prikaz za naplatu

Korisničko sučelje prikaza za naplatu prikazano je desnoj strani slike 4.4.



Slika 5.40 Prikaz mogućih grešaka prilikom plaćanja karticom u korisničkom sučelju

Na slici 5.40 prikazane su poruke greške koje se ispisuju na ekran ako korisnik unese nevažeće podatke prilikom plaćanja karticom.

Prilikom pritiska tipke "Naruči" šalje se *POST* zahtjev na *https://localhost:3000/cart/update-status*, gdje se završava narudžba. Kao što je ranije spomenuto, ovim postupkom stvara se novi redak u tablici *orders* i status košarice se postavlja na *is_paid = 1*.

5.3.7. Prikazi za neregistrirane korisnike

Prikaz za korisnike bez registriranog profila za sve funkcionalnosti izrađen je na način da se unutar prikaza proizvoda, košarice i naplate provjerava status prijave korisnika pomoću funkcije *useUser* iz *auth.js* datoteke. Ako korisnik nije prijavljen, gumbi i mogućnosti koji se prikazuju izgledaju slično kao za prijavljene korisnike, ali su prilagođeni za korisnike bez registriranog profila. Na slici 5.41 prikazano je kako se provjerava je li korisnik prijavljen te što se prikazuje za prijavljene i neprijavljene korisnike.

```
<button
    v-if="userValue === null"
    type="submit"
    class="submit-button"
    @click="addToGuestCart"
>
    Dodaj u košaricu kao gost
</button>
<button
    v-else
    type="submit"
    class="submit-button"
    @click="addToCart"
>
    Dodaj u košaricu
</button>
```

Slika 5.41 Provjera je li korisnik prijavljen za dodavanje u košaricu

U svim ostalim komponentama za dodavanje u košaricu, prikaz košarice i narudžbi, provjera statusa prijave korisnika vrši se na isti način.

6. Zaključak

Glavni cilj ovog završnog rada bio je omogućiti ispis svih proizvoda te jednostavno dodavanje proizvoda u košaricu sa izvršetkom narudžbe uz to da cijela stranica lijepo izgleda te da je dizajn responzivan. Svi postavljeni ciljevi uspješno su realizirani. Tijekom izrade aplikacije, najveći izazovi bili su rad s košaricama i integracija različitih metoda plaćanja. Iako su ova područja već poznata, njihova primjena u kontekstu ove aplikacije predstavljala je određene poteškoće, koje su na kraju uspješno savladane. Osim toga, dodavanje i prikaz slika proizvoda zahtijevali su pažljivo planiranje i optimizaciju za različite uređaje. Uz korištenje odgovarajućih tehnologija i alata, postignuto je intuitivno i interaktivno korisničko iskustvo. Ova aplikacija može se primijeniti u bilo kojem web trgovinskom okruženju, s mogućnošću prilagodbe za različite tipove proizvoda.

Literatura

[1] »Travis Scott Official Shop,« [Mrežno]. Available: https://shop.travisscott.com/. [Pokušaj pristupa 26 June 2024].

[2] »Golf Wang,« [Mrežno]. Available: https://golfwang.com/. [Pokušaj pristupa 26 June 2024].

[3] »Vite documentation,« [Mrežno]. Available: https://vitejs.dev/guide/. [Pokušaj pristupa 26 June 2024].

[4] »Vue.js documentation,« [Mrežno]. Available: https://vuejs.org/guide/introduction.html. [Pokušaj pristupa 26 June 2024].

[5] »CSS documentation,« [Mrežno]. Available: https://developer.mozilla.org/en-US/docs/Web/CSS. [Pokušaj pristupa 26 June 2024].

[6] »SCSS documentation,« [Mrežno]. Available: https://sass-lang.com/documentation/. [Pokušaj pristupa 26 June 2024].

[7] »Introduction to Node.js,« [Mrežno]. Available: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs. [Pokušaj pristupa 26 June 2024].

[8] »Express routes and controllers,« [Mrežno]. Available: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes. [Pokušaj pristupa 26 June 2024].

[9] »MySQL About,« [Mrežno]. Available: https://www.oracle.com/mysql/what-is-mysql/. [Pokušaj pristupa 26 June 2024].

[10] »Knex.js documentation,« [Mrežno]. Available: https://github.com/knex/knex. [Pokušaj pristupa 26 June 2024].

[11] »JWT About,« [Mrežno]. Available: https://jwt.io/introduction. [Pokušaj pristupa 26 June 2024].

[12] »Spotify for developers Web API documentation,« [Mrežno]. Available: https://developer.spotify.com/documentation/web-api/tutorials/getting-started. [Pokušaj pristupa 27 June 2024].

Sažetak

Ovaj rad prikazuje razvoj web aplikacije za prodaju odjeće korištenjem tehnologija Node.js za *backend*, Vue.js za *frontend*, SCSS za stiliziranje, te MySQL za bazu podataka. Analizirana su dva slična rješenja, „Travis Scott Official Store“ i „GolfWang“, kako bi se identificirale najbolje prakse i pristupi ovom zadatku. Detaljno su opisane komponente korisničkog sučelja, uključujući početnu stranicu, stranice za prijavu i registraciju, prikaz proizvoda i stranicu za naplatu. Opisane su različite uloge korisnika - registrirani korisnici, gosti i administratori. Programsko rješenje obuhvaća strukturu baze podataka, *backend* implementaciju s autentifikacijom korisnika i upravljanjem košaricom te *frontend* komponente koje omogućuju interaktivno korisničko iskustvo.

Ključne riječi: Node.js, odjeća, prodaja odjeće, Vue.js

Abstract

This paper goes over the development of a web application for selling merchendise clothes using Node.js for the backend, Vue.js for the frontend, SCSS for styling and MySQL for the database. In the paper, two similar solutions were analyzed: „Travis Scott Official Store“ and „GolfWang“, to identify the best practices and approaches for the task. The components of the user interface are described in detail, including the home page, login and registration pages, product display, and checkout page. The different user roles - registered users, guests, and administrators - are also described. The software solution encompasses the database structure, backend implementation with user authentication and cart management, as well as frontend components that enable an interactive user experience.

Keywords: clothes, Node.js, selling clothes, Vue.js

Životopis

David Mikulić rođen je 28. siječnja 2002. godine u Zagrebu. Svoje školovanje započinje u OŠ Zvonimira Franka u Kutini te zatim upisuje srednju Tehničku Školu u Kutini, smjer tehničar za računalstvo. Nakon položene državne mature upisuje stručni prijediplomski studij računarstva na FERIT-u u Osijeku.

_____

Potpis autora

Prilozi

GitHub repozitorij programskog *koda* aplikacije: https://github.com/davidmikulic44/merch-shop-fullstack