RESTful web aplikacija za virtualnu tržnicu

Vican Ajdinović, Diana

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Preddiplomski sveučilišni studij računarstva

Završni rad

RESTFUL WEB APLIKACIJA ZA VIRTUALNU TRŽNICU

Rijeka, rujan 2022.

Diana Vican Ajdinović

0069085340

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Preddiplomski sveučilišni studij računarstva

Završni rad

RESTFUL WEB APLIKACIJA ZA VIRTUALNU TRŽNICU

Mentor: doc. dr. sc. Marko Gulić

Rijeka, rujan 2022.

Diana Vican Ajdinović

0069085340

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 7. ožujka 2022.

Zavod: **Zavod za računarstvo**
Predmet: **Razvoj web aplikacija**
Grana: **2.09.06 programsko inženjerstvo**

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Diana Vican Ajdinović (0069085340)**
Studij: Preddiplomski sveučilišni studij računarstva

Zadatak: **RESTful web aplikacija za virtualnu tržnicu / RESTful web application for virtual marketplace**

Opis zadatka:

Razviti RESTful web aplikaciju virtualne tržnice. Aplikacija mora podržavati uslugu elektroničke trgovine uz implementaciju košarice odnosno svih njezinih funkcionalnosti, od dodavanja proizvoda do upravljanja njezinim sadržajem. Također, potrebno je implementirati registraciju novih obrta i upravljanje ponudom pojedinog obrta kao i upravljanje dobivenih narudžbi. Za razvoj poslužiteljskog dijela web aplikacije treba koristiti Laravel 8 radni okvir uz proizvoljno odabran sustav za upravljanje bazama podataka. Također, treba koristiti paket Laravel Sanctum za realizaciju autentifikacije (JWT token). Za razvoj klijentskog dijela aplikacije treba koristiti React JavaScript knjižicu za razvoj korisničkog sučelja uz učinkovito renderiranje aplikacije na uređajima s različitim veličinama zaslona.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Diana Vican Ajdinović
Zadatak uručen pristupniku: 21. ožujka 2022.

Mentor:

Doc. dr. sc. Marko Gulić

Predsjednik povjerenstva za završni ispit:

Prof. dr. sc. Kristijan Lenac

IZJAVA

Izjavljujem da sam ovaj završni rad, RESTful web aplikaciju za virtualnu tržnicu, izradila sama pomoću znanja stečenog tijekom studija i potrebne literature prema Pravilniku o završnom radu, završnom ispitu i završetku preddiplomskih sveučilišnih studija.

Diana Vican Ajdinović Potpis: _____

ZAHVALA

Zahvaljujem se svome mentoru doc. dr. sc. Marku Guliću na savjetima, razumijevanju te izdvojenom vremenu i trudu tijekom izrade ovoga rada.

Zahvaljujem se i svojoj obitelji na iskazanoj ljubavi i podršci.

SADRŽAJ

1. UVOD

Ideje i pokušaji razvoja odnosno šire primjene usluga elektroničke trgovine mogu se pratiti od sedamdesetih godina dvadesetoga stoljeća, no prekretnicom se smatra razvoj pružatelja internetskih usluga WWW-a (World Wide Web) 1999. godine za koji je zaslužan britanski profesor računarstva Timothy John Berners-Lee. Uslijedilo je osnivanje prvih velikih tvrtki koje su nudile usluge elektroničke trgovine poput Amazona, eBaya i Netscapea. Već 1998. godine utemeljen je PayPal koji omogućuje prijenos novčanih sredstava internetom. Od tada se broj korisnika elektroničke trgovine uvijek kretao uzlaznom putanjom, a informatičkom su evolucijom dodatno učvršćeni njezini temelji. Takvim se načinom poslovanja brišu geografske granice i proširuje izbor dostupnih proizvoda. Kupci mogu jednostavnije i brže uspoređivati kvalitetu i cijenu te međusobno dijeliti osobna iskustva. Prodavačima se istodobno olakšava prikupljanje velikih količina raznih podataka o trenutnim i potencijalnim kupcima što im pomaže u poboljšanju korisničkih usluga i prilagodbi ciljanim skupinama klijenata. Također, cijena oglašavanja je jeftinija nego u ostalim medijima što je čini pristupačnijom manjim tvrtkama koje se još uvijek pokušavaju prezentirati široj publici.

Manje obiteljske trgovine i obrti koji nastoje plasirati domaće proizvode na tržište često ne mogu pokrivati troškove oglašavanja, povećanja radnog osoblja te najamnine prostora na boljoj lokaciji, stoga se registracija u web aplikaciji za trgovanje nameće kao praktično rješenje, kako za trgovce, tako i za kupce kojima se naručeni proizvodi dostavljaju na kućnu adresu. Također, na taj se način proizvodi učinkovito promoviraju široj publici, čuva se tradicija te potiče domaća poljoprivreda, a manjim mjesnim trgovinama daje se prilika za razvoj.

Svrha ovog završnog rada je razviti *RESTful* web aplikaciju koja će pružati usluge elektroničke trgovine, s posebnim naglaskom na virtualnu tržnicu, uključujući implementaciju košarice s uobičajenim funkcionalnostima brisanja i dodavanja proizvoda te naručivanje. Osim toga, moći će se registrirati obrti te definirati njihova ponuda proizvoda. Također, potrebno je razviti funkcionalnosti za dodavanje novih kategorija i upravljanje zaprimljenim narudžbama.

Poslužiteljski je dio aplikacije izrađen u Laravelu 8 [1], razvojnom okviru koji se temelji na PHP-u. Zahvaljujući ugrađenim i višestruko testiranim funkcionalnostima koje Laravel posjeduje, poput

upravljanja sesijama, sustava za autentikaciju i objektnog relacijskog mapiranja, pozitivno utječe na robusnost projekta i otvara put većim nadogradnjama u budućnosti. Klijentska je strana razvijena pomoću Javascriptove knjižnice React u osamnaestoj inačici [2]. Svestranost komponenti, svojstvo deklarativnosti i koncept generiranja virtualne slike dokumenta samo su neke od karakteristika koje ga čine učinkovitim u izgradnji modernih korisničkih sučelja. Sustav autentikacije implementiran je koristeći se Sanctumom [3]. Sanctum je Laravelov paket s mnoštvom funkcija pomoću kojih je lako ugraditi autentikaciju u projekt, zauzimajući pritom relativno malo memorije. Softverski paketi XAMPP [4] i phpMyAdmin [5] korišteni su za poslužiteljske usluge i upravljanje podatcima u bazi, dok je radi oblikovanja responzivnog grafičkog sučelja upotrijebljen Bootstrap 5 [6].

2. KORIŠTENE TEHNOLOGIJE

2.1. Laravel

Začetci Laravela sežu u 2010. godinu kada je Taylor Otwell, entuzijastični programer iz Sjedinjenih Američkih Država, osmislio novo okruženje za programiranje aplikacija u PHP-u (Hypertext Preprocessor). Nezadovoljan tadašnjim karakteristikama postojećih alata poput Symfonyja i CodeIgnitera čija je namjena izrada programske podrške u PHP-u, odlučio je ispraviti nedostatke koji su mu otežavali svakodnevni posao. Iako u početku nije imao na umu stvaranje potpuno nove razvojne okoline, razmatrajući prednosti i nedostatke tadašnjih okruženja te zahtjeve s kojima se susretao na svome radnom mjestu, polako je razvijao Laravel. U lipnju sljedeće godine objavljena je njegova prva inačica, a najbitnije karakteristike obuhvaćale su ugrađenu autentikaciju, omogućen prijevod sadržaja koji se prikazuje korisniku na više jezika, razne funkcionalnosti za rad s HTML-om (HyperText Markup Language), implementaciju modela i definiranje njihovih međusobnih odnosa te upravljanje sesijama i nešto jednostavniju implementaciju ruta.

No, za Laravelovu se arhitekturu još uvijek nije moglo reći da pripada obrascu *model-view-controller* (MVC), svojstvenom mobilnim i web aplikacijama. MVC arhitektura raščlanjuje programski kod na tri komponente. U modelu se oblikuju podatci koji se čuvaju u bazi podataka te utvrđuju odnosi s drugim modelima dok je uloga jedinice *view* prikazivanje i prikupljanje tih podataka u korisničkom sučelju. Kontroleri su treća jedinica, posrednik između modela i *view* komponenti koja sadrži metode za obradu korisnikovih zahtjeva te na temelju definiranih pravila vraća određeni rezultat. Upravo su kontroleri značajna nadopuna koja je postala dio druge verzije objavljene u rujnu 2011. godine.

Osim toga, dodani su i obrasci Blade koji kombiniraju uobičajene jezike za izradu klijentske strane, HTML i JavaScript, s PHP-om kao okosnicom. Sintaksa petlji i grananja pojednostavnjena je te omogućuje lako baratanje varijablama čije se vrijednosti preuzimaju s poslužitelja. Blade skripte potpuno se prevode u PHP kod i čuvaju u privremenoj memoriji dok se god ne izmijene čime se smanjuje vrijeme učitavanja. Primjer takvog koda vidljiv je na Slici 2.1.1. koja prikazuje *foreach* petlju kroz niz objekata te dohvaćanje vrijednosti njihovih atributa poput naziva i opisa.

```
@foreach ($shops as $shop)
    <div class="col-md-4">
        <div class="card mb-4 shadow-sm">
            <img class="card-img-top"
                 src="{{ asset('trgovina.jpg) }}" alt="trgovina"
                 width="400px" height="400px">
            <div class="card-body">
                <p class="card-title">{{ $shop->name }}</p>
                <p class="card-text">{{ $shop->description }}</p>
                <a href="{{ route('shops.show-products', $shop->id) }}" class="card-link">
                    Prikaži proizvode
                </a>
                <div class="d-flex justify-content-between align-items-center">…
                </div>
            </div>
        </div>
    </div>
@endforeach
```

Slika 2.1.1. Foreach petlja kroz listu proizvoda na klijentskoj strani

Kako bi kod bio pregledniji i jasniji te kako se iste funkcionalnosti ne bi opetovano programirale i pojavljivale na više mjesta, poželjno ih je spremiti kao posebne datoteke te kasnije ugnijezditi u skripte gdje je potrebno. No, nije nužno koristiti se Blade skriptama u izradi klijentske strane. Laravel može poslužiti u programiranju samo poslužiteljskog dijela aplikacije, a za korisničko se sučelje može birati između širokog spektra raznih knjižnica.

Uslijedile su značajne nadogradnje početne verzije. Predstavljen je Artisan, komandno sučelje s puno prečaca koji znatno olakšavaju i ubrzavaju programiranje u Laravelu. Na primjer, ako je potrebno napraviti novi kontroler, dovoljno je napisati *php artisan make:controller ImeKontrolera* u terminal. Nova će datoteka već sadržavati osnovne metode za dodavanje, brisanje i pretraživanje. Sličan je postupak i prilikom stvaranja novog modela ili operacija s tablicama u bazama podataka. Popis osnovnih akcija kontrolera s ustaljenim nazivima naveden je u Tablici 2.1.1.

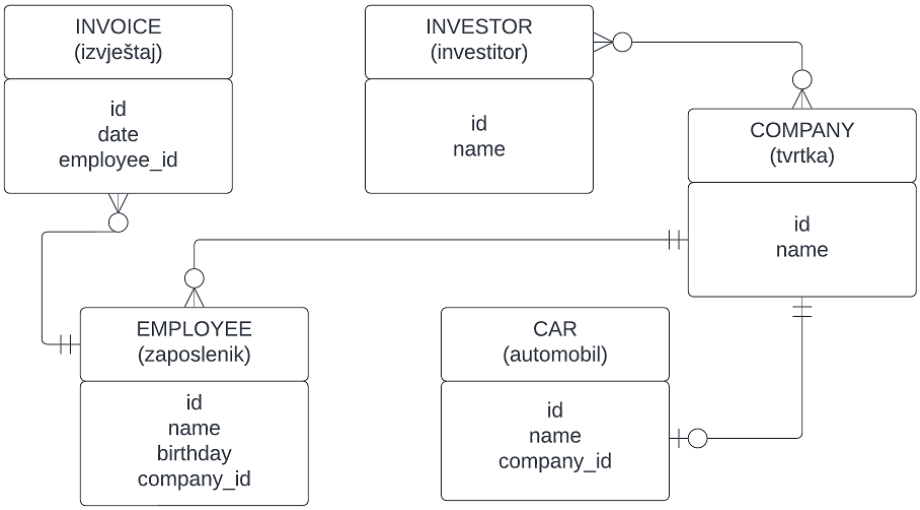
Tablica 2.1.1. Uobičajeni nazivi metoda, funkcija i ruta

Metoda	*URI*	*Funkcija*	*Naziv rute*
GET	/categories	index	categories.index
GET	/categories/create	create	categories.create
POST	/categories	store	categories.store
GET	/categories/{categoryId}	show	categories.show
GET	/categories/{categoryId}/edit	edit	categories.edit
PUT	/categories/{categoryId}	update	categories.update
DELETE	/categories/{categoryId}	destroy	categories.destroy

Na ovaj način kod će ostati jasno strukturiran i podijeljen u logičke cjeline, a pritom ga je lako prilagoditi zahtjevima klijenta. Sintaksa je instinktivna i njome su pojednostavnjene mnoge funkcije PHP-a te SQL-a, a posebno bi valjalo istaknuti Laravelov popularni Eloquent ORM (objektno relacijsko mapiranje) za brzo i učinkovito izvršavanje operacija nad podatcima u bazi podataka. Nakon odabira i konfiguracije lokalnog poslužitelja, može se krenuti u migriranje tablica te izradu modela. Modeli su klase kojima su iskazani atributi i veze tablica u bazi podataka, stoga je uobičajeno da naziv modela, koji je u jednini, odgovara nazivu tablice koji je u množini. Na primjer, modelu *Planet* odgovarala bi tablica *planets,* ali nije nužno. Modelu se može dodijeliti i neka druga tablica po želji, no tada je to potrebno ručno definirati. Kako bi se spriječilo neželjeno ubacivanje podataka, a atributima entiteta dodijelile vrijednosti koje se kose s načinom rada aplikacije, potrebno je takve atribute dodati u niz naziva *guarded*. U takvom se slučaju obično štite identifikatori i lozinka. U suprotnom, kada se želi naznačiti da su pojedina polja ispunjiva kroz HTML forme, nazivi tih stupaca dodaju se na listu sadržanu u varijabli *fillable*. Ako se želi zabraniti vraćanje pojedinih vrijednosti atributa klijentskoj strani, označi ih se kao *hidden*. Neke vrijednosti zapisane su u tablicama u obliku nule i jedinice ili običnog teksta odnosno znakovnog niza. No, ponekad ih je potrebno pretvoriti u drugi tip podatka. Tada ih se pospremi u varijabli *casts* te se im se pored naziva atributa dodijeli način na koji ih se želi iščitati, primjerice niz, logička vrijednost ili datum s točno određenim redoslijedom navođenja mjeseca, godine i dana.

U Laravelu je vrlo elegantno riješen problem povezivanja tablica i određivanja vrste veza. Njegov Eloquent zapravo je ORM alat koji je automatski uključen u svaki novi projekt. Naglasak je na objektno-orijentiranom pristupu, modelima se pristupa kao klasama, a podatci se iz baze prikazuju

kao objekti. Pritom su, osim osnovnih radnji dodavanja, brisanja i uređivanja, omogućene i puno složenije SQL operacije. Vrlo je bitno prije izrade svake aplikacije temeljito proučiti njezinu namjenu, a zatim izraditi odgovarajući dijagram. Jednostavni dijagram prikazan na Slici 2.1.2. poslužit će kao primjer povezivanja entiteta Zaposlenika (eng. *Employee*), Ulagača (eng. *Investor*), Tvrtke (eng. *Company*), Automobila (eng. *Car*) i Izvještaja (eng. *Invoice*) s atributima identifikacijskog broja (*id*), imena (eng. *name*), datuma (eng. *date*), rođendana (eng. *birthday*) i stranih ključeva, identifikatora zaposlenika (eng. *employee_id*) i tvrtke (eng. *company_id*).



Slika 2.1.2. Prikaz odnosa objekata u automobilskoj industriji

S obzirom na to da marka automobila pripada samo jednoj tvrtki, a neka tvrtka proizvodi samo jednu marku automobila, veza se između tih dvaju entiteta definira kao 1:1. Tijekom unosa vrijednosti automobila u tablicu potrebno je zapisati identifikacijski broj tvrtke kako bi se relacija mogla opisati i u Laravelovom modelu. U modelu Company dodaje se kod prikazan na Slici 2.1.3.

```php
public function car() {
    return $this->hasOne(Car::class);
}
```

Slika 2.1.3. Definiranje veze 1:1 u Company.php

Zatim se u modelu Car definira funkcija *company()* u kojoj se iskazuje pripadnost automobila točno jednoj tvrtki što prikazuje Slika 2.1.4.

```php
public function company() {
    return $this->belongsTo(Company::class);
}
```

Slika 2.1.4. Definiranje veze 1:1 u Car.php

Dakle, ključnim se riječima *hasOne* i *belongsTo* iskazuje relacija 1:1 na temelju identifikacijskog broja tvrtke u tablici s automobilima (company_id). Iznimno se, u slučajevima kada naziv stranog ključa odstupa od konvencije, kao drugi parametar nakon naziva klase šalje ime stranog ključa. Ako parametar nije definiran, pretpostavlja se da je nazvan prema imenu tablice u jednini s nastavkom *_id.*

Sljedeća vrsta veze je 1:m, odnosno, prema prije navedenom primjeru, jedna tvrtka može imati više zaposlenika, dok osoba može biti zaposlena u samo jednoj tvrtki. Za iskazivanje takve relacije koriste se ključne riječi *hasMany* i *belongsTo.* U modelu tvrtke dodaje se funkcija *employees*() prikazana na Slici 2.1.5. čime se označava da tvrtka zapošljava više osoba.

```php
public function employees() {
    return $this->hasMany(Employee::class);
}
```

Slika 2.1.5. Definiranje veze 1:m u Company.php

Potom se u modelu zaposlenika napiše kod kao što je prikazano na slici Slici 2.1.6. čime se iskazuje pripadnost radnika nekoj tvrtki na temelju njezinog identifikatora u tablici zaposlenika.

```php
public function company() {
    return $this->belongsTo(Company::class);
}
```

Slika 2.1.6. Definiranje veze 1:m u Employee.php

Posljednji tip relacije je m:n čime se može opisati odnos investitora i tvrtki. Njegova je implementacija nešto složenija te zahtijeva migriranje nove tablice u bazu. Tablica će sadržavati strane ključeve, identifikatore tvrtke te investitora, a njezin naziv prema prihvaćenoj normi kombinacija je imena tablice tvrtki i tablice zaposlenika. Lako se može dogoditi da se količina zapisa u takvoj tablici počne naglo povećavati, stoga nije na odmet razmotriti neka druga rješenja kojima bi se spriječila takva situacija. Ključna riječ kojom se definira takva vrsta veze je *belongsToMany* te je obvezno dodati i naziv nove tablice kao drugi parametar.

Nakon utvrđivanja i opisivanja gore navedenih odnosa pojednostavnjeno je rukovanje podatcima u bazi čime se manje griješi i puno brže dolazi do rješenja, pogotovo ako se barata velikim količinama zapisa. Također, kod je puno kraći, čišći i jasniji. Na primjer, želi se doznati ime marke automobila za koji je izdan neki izvještaj. Kada bi se koristilo obični SQL-om, morale bi se spojiti četiri tablice. Naravno, postoje i zamršenije situacije kada postaje nezgodno baratati stranim ključevima ili odrediti koje atribute uvrstiti u konačan rezultat, a koji će ostati neiskorišteni višak. No, u Laravelu je dovoljno napisati *$invoice->employee->company->car->name*. Ponekad je nezgodno procijeniti učinkovitost programskog koda prije nego što se aplikacija pusti u produkciju. Ako se operacije izvode nad većim brojem podataka, potrebno je pripaziti na dostupnost resursa. Eloquent ORM rabiti metodu *with()* kojom se postiže tzv. *eager loading*, odnosno *istovremeno izvršavanje SQL upita nad više povezanih entiteta* [7]. Na taj se način nastoji izbjeći *N + 1* problem, a pod tim se pojmom podrazumijeva nepotrebno ponavljanje upita za svu djecu objekte nekoga objekta roditelja. Uz to se može nadodati jedan ili više uvjeta koristeći funkciju *whereHas()* kao što je prikazano na Slici 2.1.7. gdje se traže tvrtke čiji nazivi marki automobila počinju glasovnim skupom *da* ili završavaju glasovnim skupom *er*.

```
$records = Company::with('cars')
    ->whereHas('cars', function($query) {
        return $query->where('name', 'LIKE', 'da'.'%')
            ->orWhere('name', 'LIKE', '%'.'er');
    });
```

Slika 2.1.7. Filtriranje funkcijama with(), whereHas() i orWhere()

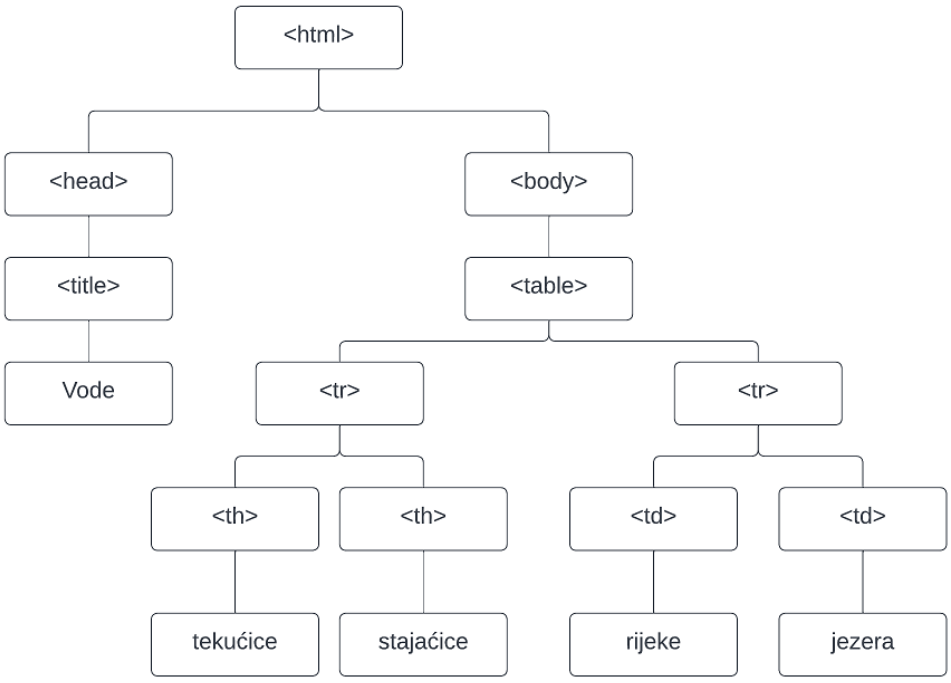
U trenutku objave ovog završnog rada Laravel je dosegao devetu inačicu, a za njegovu je nadogradnju ključna četvrta verzija iz 2013. godine u kojoj su dodane fasade, mehanizam apstrakcije i enkapsulacije koda koji pruža statičko sučelje kroz koje se pristupa mnogobrojnim metodama [8].

Po prvi je put uključena potpora za implementaciju elektroničke pošte te sijanje podataka pomoću *seeder* klasa. Sijanje baze podataka (eng. database seeding) podrazumijeva pohranu velikih količina podataka čime se nastoji simulirati rad aplikacije u produkciji. Na taj se način može točnije ocijeniti njezina učinkovitost i potrebna količina resursa. Može se deklarirati tip i oblik podataka poput adrese e-pošte, duljina imena, primijeniti *hash* funkcija prilikom dodavanja lozinke te popuniti bazu poštujući vrstu veze između dva modela. Od 2013. godine za instaliranje paketa unutar projekta i za kreiranje nove aplikacije rabi se Composer [9]. Ideja je da se knjižnice po potrebi dodaju u repozitorij projekta umjesto da se instaliraju na globalnoj razini. Nastao je po uzoru na Rubyjev bundler i Node-ov *npm*.

Iako se sustav za autentikaciju može samostalno napisati i implementirati u aplikaciju, preporučljivo je poslužiti se u tu svrhu namijenjenim paketima te tako zaobići greške i propuste koji će umanjiti kvalitetu proizvoda i ugroziti korisničke podatke. *Ako se razvija monolitična aplikacija, u izbor ulaze već ugrađeni Laravel Breeze, Laravel Jetstream ili pak Laravel Fortify čije se metode pozivaju preko Auth i Session fasada. Takav način provjere identiteta koristi se kolačićima dok se podatci pospremaju u sesiji. Autenticiranje korisnika kroz API riješava se na drukčiji način, odnosno koristeći se API tokenom* [1]. Nakon što korisnik unese potrebne podatke, zauzvrat mu se pridruži jedinstveni enkriptirani znakovni niz u bazi pomoću kojeg se zatim pristupa raznim uslugama neke web aplikacije. U tu svrhu Laravel nudi pakete Sanctum, koji je jednostavnija opcija, te Passport koji se rabi kada je nužno posegnuti za tzv. otvorenom autorizacijom (OAuth). Svrha OAuth-a je pružiti pristup raznim uslugama bez odavanja lozinke i sličnih osjetljivih informacija. U tom se slučaju provjera identiteta oslanja na tokene, a posrednik koji potražuje uslugu u korisnikovo ime često je Facebook ili Google. Za jednostranične aplikacije (eng. *single page* aplikacije, skraćeno SPA) te mobilne aplikacije, Laravel Sanctum sasvim je dovoljan, iako ne podupire otvorenu autorizaciju. Pojam SPA obuhvaća aplikacije poput Gmail-a koje, iako ažuriraju sadržaj kojeg poslužitelj vraća, ne traže učitavanje svaki put kada klijent pošalje zahtjev. Autentikacija Sanctumom uključuje tablicu u kojoj se pamte jedinstveni osobni tokeni (*personal access tokens*). Kod single page aplikacija, Sanctum najprije provjerava postoji li kolačić koji odgovara autenticiranoj sesiji te na taj način utvrđuje je li korisnik prijavljen. Ako kolačić nije prisutan, započet će s provjerom API tokena [3].

2.2. React

React je JavaScript knjižnica otvorenog koda za razvoj korisničkog sučelja objavljena u svibnju 2013. godine. Njezino autorstvo pripada tvrtki Meta, koja je tada djelovala pod imenom Facebook, i mnogim programerima diljem svijeta. U svojim je početcima služio kao knjižnica za XHP, kombinaciju programskih jezika PHP i Metinog Hacka. XHP također pripada Meti, a njegova je namjena bila olakšati programiranje klijentske strane te spriječiti XSS (eng. cross-site scripting) napade. Međutim, logika kojom je razvijan nije bila zadovoljavajuća u slučaju dinamičnih web aplikacija jer bi se stranica morala ponovo učitati kad god bi se dogodila neka promjena. Zbog toga je u React utkan koncept VDOM-a (Virtual Document Object Model). DOM (Document Object Model) predstavlja API (Application Programming Interface) te određuje logičku strukturu dokumenta. Svaka je komponenta jedan čvor, a kako izgleda tako raščlanjen prikaz na primjeru HTML dokumenta može se vidjeti na Slici 2.2.1.



Slika 2.2.1. Prikaz DOM-a HTML dokumenta

Iako grafički prikaz DOM-a vrlo često posjeduje strukturu stabla, ni u kojem slučaju ne bi ga valjalo tako definirati. Njime se nastoji prikazati logički model dokumenta, a DOM uvijek prati način

implementiranja njegovih funkcionalnosti. Virtualni DOM slika je DOM-a prije bilo kakvih izmjena u dokumentu i zapisana je u memoriji. Kada se u React dokumentu promijeni stara komponenta ili uvede nova, provjeravat će se razlika između virtualnog i trenutnog DOM-a. Tada se stranica neće morati ponovo učitati, nego će se korigirati samo izmijenjeni dio dokumenta čime se značajno poboljšava iskustvo korištenja aplikacijom. Također, može se reći da React posjeduje svojstvo deklarativnosti. Nasuprot imperativnom pristupu, u kojem je potrebno odrediti svaki korak iz seta promjena kako bi se došlo do željenog izgleda korisničkog sučelja, u Reactu se samo definira krajnji rezultat. Prije nego se krene u razvoj klijentske strane, potrebno je instalirati *npm* [10], alat za upravljanje paketima te kreiranje novih projekata.

Izgrađujući klijentsku stranu u Reactu zapravo se kreiraju komponente čiji se odnos može opisati kao odnos roditelja i djece te se manje, jednostavnije komponente gnijezde u zajedničke roditeljske komponente. Na primjer, napisane su tri komponente za navigacijsku traku (Navbar.js), izbornik koji će se prikazivati skroz lijevo na ekranu (Sidebar.js) te podnožje s podatcima o tvrtki koja predstavlja aplikaciju, kontaktima i sl. (Footer.js). Doda se nova datoteka u kojoj se pozivaju prethodno spomenute komponente te se navedu točno onim redoslijedom kojim se pojavljuju u korisničkom sučelju. Radi boljeg razumijevanja slika 2.2.2. prikazuje programski kod roditeljske *layout* komponente. Komponenta Outlet poziva se iz *react-router-dom* paketa, a njezina je uloga označiti mjesto na kojem će se smjestiti preostale komponente djeca koje su dio aplikacije i u kojima su implementirane njezine razne funkcionalnosti poput učitavanja liste podataka ili korisničkog računa.

```jsx
import React from 'react';

import { Outlet } from 'react-router-dom';
import Navbar from './Navbar';
import Sidebar from './Sidebar';
import Footer from './Footer';

import '../../assets/admin/css/styles.css';
import '../../assets/admin/js/scripts';

const AdminLayout = () => {
    return (
        <div className="sb-nav-fixed">
            <Navbar />
            <div id="layoutSidenav">
                <div id="layoutSidenav_nav">
                    <Sidebar />
                </div>
                <div id="layoutSidenav_content">
                    <main>
                        <Outlet />
                    </main>
                    <Footer />
                </div>
            </div>
        </div>
    );
}

export default AdminLayout;
```

Slika 2.2.2. Prikaz layout komponente implementirane u Reactu

Neki se dijelovi mogu preuzeti kao skice s mnogobrojnih stranica i prilagoditi željama klijenta, a ostale komponente isprogramirati potpuno samostalno. Na ovaj je način lakše izraditi bogata, responzivna i korisnicima primamljiva sučelja. Također, slične se funkcionalnosti ne moraju pisati ispočetka, već se pozivaju na mjestima gdje su potrebne. Struktura koda je tada čišća i jasnija, lakše je pronaći i ispraviti pogreške te brže implementirati nove zahtjeve.

Komponente aplikacije iz ovog završnog rada podijeljene su u dvije skupine, sučelje koje komunicira s prijavljenim korisnikom neovisno o njegovoj ulozi te administratorsko sučelje, a glavna roditeljska datoteka je App.js. U njoj su za prethodno dva spomenuta sučelja pozvana dva *layout* elementa PublicLayout.js i AdminLayout.js. Kako to izgleda napisano u Reactu, prikazuje Slika 2.2.3.

```
function App() {
  return (
    <div className="App">
      <BrowserRouter>
        <Routes>
        <Route path="/" element={<PublicLayout />} >
          <Route path='/*' element={ <PublicRoutes /> } />
          <Route
            path='/register'
            element={ localStorage.getItem('auth_token') ? <Navigate to="/home" /> : <Register /> }
          />
          <Route
            path='/login'
            element={ localStorage.getItem('auth_token') ? <Navigate to="/home" /> : <Login /> }
          />
        </Route>

        <Route path="/" element={<AdminLayout />} >
          <Route path='/admin/*' element={ <AdminRoutes /> } />
        </Route>

        </Routes>
      </BrowserRouter>
    </div>
  );
}
```

Slika 2.2.3. Glavna layout komponenta App.js

Nadalje, svakom su od *layouta* pridružene datoteke AdminRoutes.js i PublicRoutes.js s odgovarajućim rutama, odnosno poveznicama na određene komponente. Na Slici 2.2.4. može se vidjeti lista ruta namijenjenih prijavljenom korisniku. One se dohvaćaju iz datoteke PublicRoutes.js te renderiraju unutar odgovarajućeg *layouta* kako je definirano u App.js datoteci. Uloga BrowserRoutera je sinkronizacija korisničkog sučelja s URL-om, a komponenta Route povezuje URL s točnim prikazom na ekranu.

```
function PublicRoutes() {

  return (
    <Routes>
        <Route path='/home'  element={<Home />} />
        <Route path='/about-us' element={<AboutUs />} />
        <Route path='/contacts' element={<Contacts />} />
        <Route path='/categories' element={<Categories />} />
        <Route path='/shops' element={<Shops />} />
        <Route path='/categories/:categoryKeyname' element={<ProductsByCategory />} />
        <Route path='/shops/:shopKeyname' element={<ProductsByShop />} />
        <Route path='/categories/:categoryKeyname/:productKeyname' element={<Product />} />
        <Route path='/my-shopping-cart' element={<Cart />} />
        <Route path='/checkout' element={<Checkout />} />
        <Route path='/order-placed' element={<OrderPlaced />} />
    </Routes>
  );

}

export default PublicRoutes;
```

Slika 2.2.4. Rute prijavljenog korisnika

React Router DOM je *npm*-ova knjižnica za upravljanje rutama. U rujnu 2021. godine objavljena je njezina šesta inačica s mnogim izmjenama i novostima. Znatno je smanjena veličina paketa zbog čega je povećana brzina učitavanja na slabijim mrežama. Prethodni element Switch zamijenjen je novim elementom Routes koji omogućava ugnježđenja te lakše prepoznaje hijerarhiju ruta. Prije se moralo paziti na redoslijed pozivanja ruta kako bi se prikazala ispravna komponenta. Na Slici 2.2.5. definirane su dvije rute. S obzirom da su im putovi gotovo isti i da je ruta za prikaz postojeće kategorije prva navedena, može se pretpostaviti da će se prilikom kreiranja nove kategorije renderirati komponenta Category umjesto ispravne AddCategory. No, u novoj verziji React prepoznaje da je put do stranice za unos novih podataka specifičniji, stoga se će se korisnika preusmjeriti na ispravnu stranicu.

```
<Route path="categories/:categoryId" element={<Category />} />
<Route path="categories/new" element={<AddCategory />} />
```

Slika 2.2.5. Rute za pregled kategorija

To svojstvo omogućilo je relativno deklariranje ruta zbog čega je izbačen atribut *exact*. Njime se prije definirao točan put do neke stranice, no sada je moguće definirati jednu roditeljsku rutu te unutar nje renderirati rute djecu koje se definiraju u zasebnoj datoteci. Dodana je i nova kuka (eng. hook) *useNavigate()* za navigaciju koja je zamijenila do tada rabljenu *useHistory()*. Navigacija po povijesti odvija se pomoću njezine instance i cijelog broja kao parametra kojim se pomiče pokazivač po stogu. Kao parametar se može unijeti i put koji je povezan s određenom rutom. React kuke su JavaScript funkcije koje omogućuju upravljanje trenutnim stanjem i korištenje ostalim Reactovim svojstvima bez kreiranja posebnih klasa. Osim spomenute *useNavigate()* najčešće korištene kuke u ovom projektu su *useState()* i *useEffect()*. Uloga kuke *useState()* je prihvatiti početno stanje neke varijable ili objekta i vratiti trenutno stanje te funkciju kojom se stanje ažurira. Kuka *useEffect()* aktivira se prilikom ažuriranja stanja. Način na koji se ove kuke koriste prikazan je na Slici 2.2.6.

```
const [categories, setCategories] = useState([]);
const [loading, setLoading] = useState(true);

useEffect(() => {

    document.title = "Moja tržnica";

    axios.get(`/api/get-categories`).then(res => {
        if (res.status === 200) {
            setCategories(res.data.categories);
            setLoading(false);
        }
    });
}, []);
```

Slika 2.2.6. Uporaba useEffect() kuke

U ovom slučaju kukom *useState()* inicijalizira se varijabla *categories* u koju će se spremiti lista kategorija koja se vraća iz Laravelovog dijela aplikacije. Zatim se funkcijom *setCategories()* pozvanoj u kuki *useEffect()* podatci o kategorijama spremaju u varijabli *categories*. Varijabla *loading* dodana je kako se tijekom učitavanja podataka korisniku ne bi prikazivao prazan ekran, a u njoj se može upisati obavijest o učitavanju ili pak implementirati kružić za učitavanje. Na slici se također može

uočiti axios, JavaScript-ov HTTP klijent za upravljanje korisnikovim zahtjevima koji se šalju poslužiteljskoj strani te odgovorima koji dolaze klijentu.

Ponekad će prikaz pojedinih elemenata ovisiti o tome je li trenutno prijavljeni korisnik administrator ili obični korisnik ili pak o tome je li uopće prijavljen. Na primjer, nema smisla prijavljenom korisniku, za kojeg se zna da je registriran, prikazivati gumbe koji vode na stranicu za registraciju ili prijavu, već samo gumb za odjavu. Vrijedi i obrnuto. Da bi se utvrdila prijava u sustav, React nudi poseban mrežni objekt *localStorage* koji posjeduje razne metode za upravljanje mrežnom pohranom. Metodom *setItem()* definira se ključ i njemu pripadajuća vrijednost, a metodom *getItem()* može se provjeriti postoji li korisnikov token u lokalnoj pohrani te na temelju toga renderirati željene komponente. U primjeru sa Slike 2.2.7. vrijednost varijable *AuthButtons* postavit će se kao skup HTML gumbova za prijavu i registraciju ako osoba nije prijavljena. U suprotnom slučaju varijabla će sadržavati samo gumb za odjavu iz sustava.

```
var AuthButtons = '';
if (!localStorage.getItem('auth_token')) {
    AuthButtons = (
        <ul className="navbar-nav ms-auto mb-2 mb-lg-0">
            <li className="nav-item">
                <Link className="nav-link" to="/login">Login</Link>
            </li>
            <li className="nav-item">
                <Link className="nav-link" to="/register">Register</Link>
            </li>
        </ul>
    );
} else {
    AuthButtons = (
        <ul className="navbar-nav ms-auto mb-2 mb-lg-0">
            <li className="nav-item">
                <button type="button"
                    onClick={logoutSubmit}
                    className="nav-link btn btn-primary btn-sm text-white">
                        Logout
                </button>
            </li>
        </ul>
    );
}
```

Slika 2.2.7. Renderiranje varijable AuthButtons

Osim navedenih, postoje još i funkcije za uklanjanje točno određene vrijednosti na temelju njezinog ključa te potpuno čišćenje memorije.

Iako je React intuitivan te ga je lako shvatiti čak i početnicima, nedostaje mu detaljnije razrađena službena literatura. Osim toga, nema ni definiranu jedinstvenu normu što je veliki problem s obzirom na to da je otvorenog koda i na njegovom razvoju surađuje puno ljudi. Ono što se može smatrati i prednosti i manom je njegova ovisnost o mnogim vanjskim knjižnicama. Iskusnom će programeru na taj način biti ponuđene razne opcije i može birati što će uvrstiti u svoj projekt. No, nekome tko tek počinje programirati može biti teško razlučiti koje pakete odabrati, a da pritom ne dođe do njihovih međusobnih konflikata ili, još gore, konflikata između različitih inačica paketa. Bez obzira na to, na umu se mora imati da je React stvorila velika i stabilna tvrtka koja se njime koristi u svojim proizvodima. Nadogradnje su redovite i dobro promišljene i sigurne neće brzo zastarjeti ili se potpuno napustiti. 2015. godine objavljen je React Native pomoću kojeg se mogu programirati aplikacije za Android i iOS čime je smanjena potreba za učenjem objektno-orijentiranog C-a i Jave.

2.3. Bootstrap

2011. godine Marko Otto i Jacob Thornton razvili su Bootstrap. Iako je stabilna inačica objavljena u kolovozu 2021., već dugo je jedan od najpopularnijih okruženja za oblikovanje grafičkih sučelja. Njegovom uporabom izbjegavaju se mnoge nepravilnosti kada se aplikacija učitava u pretraživačima kojima i nije potpuno prilagođena, bez obzira na veličinu uređaja. Elemente koji se žele implementirati poput grafova, tablica, odjeljaka za navigaciju, administratorskih sučelja i mnogih drugih vrlo je lako prilagoditi osobnim preferencijama. Sa službene stranice može se skinuti mnoštvo besplatnih skica, pospremiti ih kao CSS ili JavaScript datoteke u direktoriju projekta i učitavati po potrebi. U izradi ovoga završnog rada korišteni su elementi verzije 5.2.

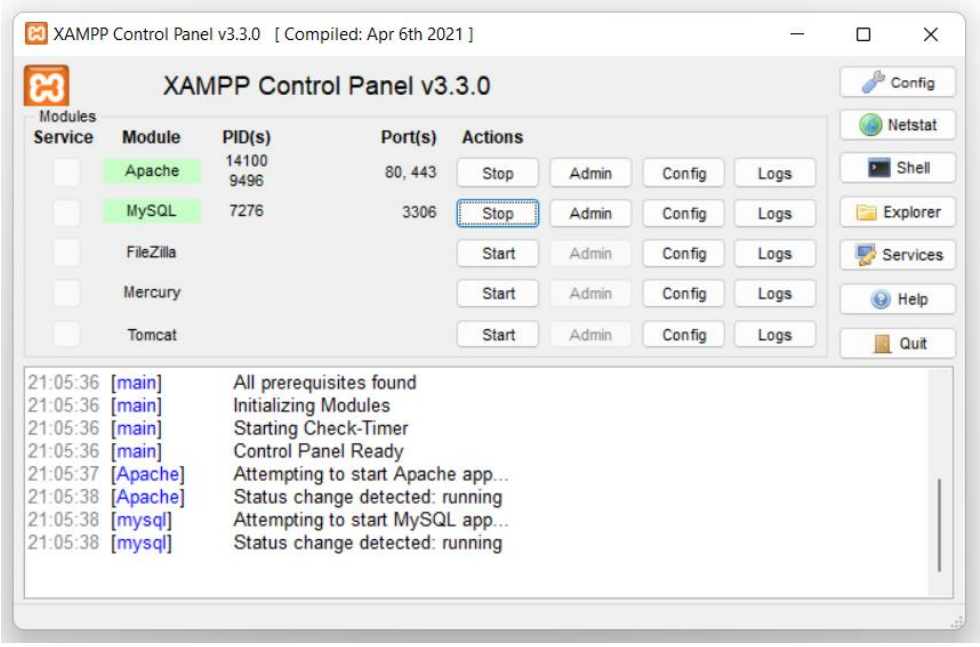
Posebnu pažnju valja posvetiti Bootstrapovom sustavu rešetki (eng. *Grid System*) na kojem su postavljeni temelji responzivnog dizajna. Ideja je da se grafičko sučelje raščlani na retke i stupce. S obzirom na specifično svojstvo redaka, negativnu marginu od -15px, retci se uvijek smještaju unutar *container* elementa koji to poništava. Time se izbjegava nekontrolirano ponašanje redaka. Containeri mogu biti prilagodljive ili točno određene širine. Zajedno s retcima u komponenti *container* može postojati bilo koji drugi element poput paragrafa ili liste. Međutim, kada se započne s dodavanjem redaka, unutar njih ne smije se nalaziti ni jedan element osim stupaca. Sadržaj koji se želi dodati u

retke, potrebno je najprije postaviti u stupce. Bootstrapovim klasama *xs, sm, md* i *lg* definira se koliko će stupaca pojedini element zauzeti ovisno o veličini ekrana. Klasa *xs* označuje mobitele, odnosno ekrane uže od 768 piksela. Veličina između 768px i 992px pripada klasi *sm* namijenjenoj tabletima, dok raspon klase *md* seže do 1200 piksela i rabi se za manja prijenosna računala. Ekranima širim od 1200px dodijeljena je klasa *lg*. Dakle, u Bootstrapu je dozvoljeno urediti veličinu pojedine komponente na način da se ona prilagođava veličini uređaja na kojem se koristi aplikacijom. Pored naziva klase dovoljno je dodati broj kojim se označava broj stupaca koji će neki element zauzeti. Bitno je pritom imati na umu da je širina ekrana podijeljena na dvanaest dijelova, stoga zbroj stupaca svih elemenata koji se nalaze jedan pored drugoga ne smije biti veći od dvanaest.

2.4. XAMPP

Tijekom izrade aplikacije kao lokalni poslužitelj upotrijebljen je XAMPP, softverski paket za pružanje mrežnih usluga. Njegov naziv zapravo je akronim glavih komponenti koje XAMPP uključuje - Apache HTTP Servera koji prihvaća nadolazeće HTTP zahtjeve, MariaDB (ili MySQL) sustava za upravljanje bazama podataka te interpretera za PHP i Pearl skripte, dok slovo X predstavlja *cross-platform* usluge. U paketu se nalazi i phpMyAdmin te FileZilla FTP, Mercury Mail i JSP Tomcat serveri.

PhpMyAdmin postoji još od 1998. godine. Aplikacija je napisana u PHP-u u svrhu izvršavanja raznih operacija u SQL-u nad tablicama i bazama te administracije podataka na mreži. Dodavanje, brisanje i uređivanje karakteristika atributa i njihovih vrijednosti izvršava se preko grafičkog sučelja, a softver se može pokretati na bilo kojem serveru. Omogućuje pričuvu sigurnosne kopije baze podataka i izvoz u formatima kao što su XML, CSV, SQL, PDF i mnogi drugi. Ako se ne koristi XAMPP, kako bi se phpMyAdmin pokrenuo, potrebna je ručna instalacija mrežnog servera (Apache, Nginx, IIS) i baze podataka (MySql, MariaDB). Prije pokretanja aplikacije na XAMPP kontrolnoj ploči potrebno je uključiti Apache i MySql kako je prikazano na Slici 2.4.1. Pored svakog modula zapisan je broj vrata (eng. port) na kojem je taj modul aktivan.



Slika 2.4.1. Komandno sučelje XAMPP-a

2.5. Postman

Prilikom razvoja web aplikacija s odvojenom klijentskom i poslužiteljskom stranom, kao što je slučaj u ovom radu kod kojeg je poslužiteljski dio izrađen u Laravelu, a klijentski dio u Reactu, poželjno je koristiti se alatima za testiranje HTTP zahtjeva poput Postmana [11]. Kroz njegovo grafičko sučelje ponuđene su uobičajene metode *GET, POST, PUT, DELETE* i *PATCH.* Nakon toga potrebno je odrediti u kojem će se formatu primiti zahtjev, npr. u JSON obliku, a mogu se unijeti parametri, odnosno vrijednosti atributa koje se šalju. Poslužiteljski dio vraća odgovore i prihvaća zahtjeve kao da komunicira sa stvarnim klijentom. Za funkcionalnosti koje traže autentikaciju, u polju za unos tokena unese se vrijednost koja je zapisana u bazi, tj. pridružena postojećem korisniku nakon čega se nastavlja uobičajena razmjena podataka. Nakon svake akcije poslužitelj vraća troznamenkasti broj kojim obavještava je li zahtjev uspješno izvršen te razlog odbijanja ako ga nije bilo moguće provesti. Ti su brojevi standardizirani i nazivaju se statusnim kodovima. Tablica 2.5.1. prikazuje njihov oblik i značenje.

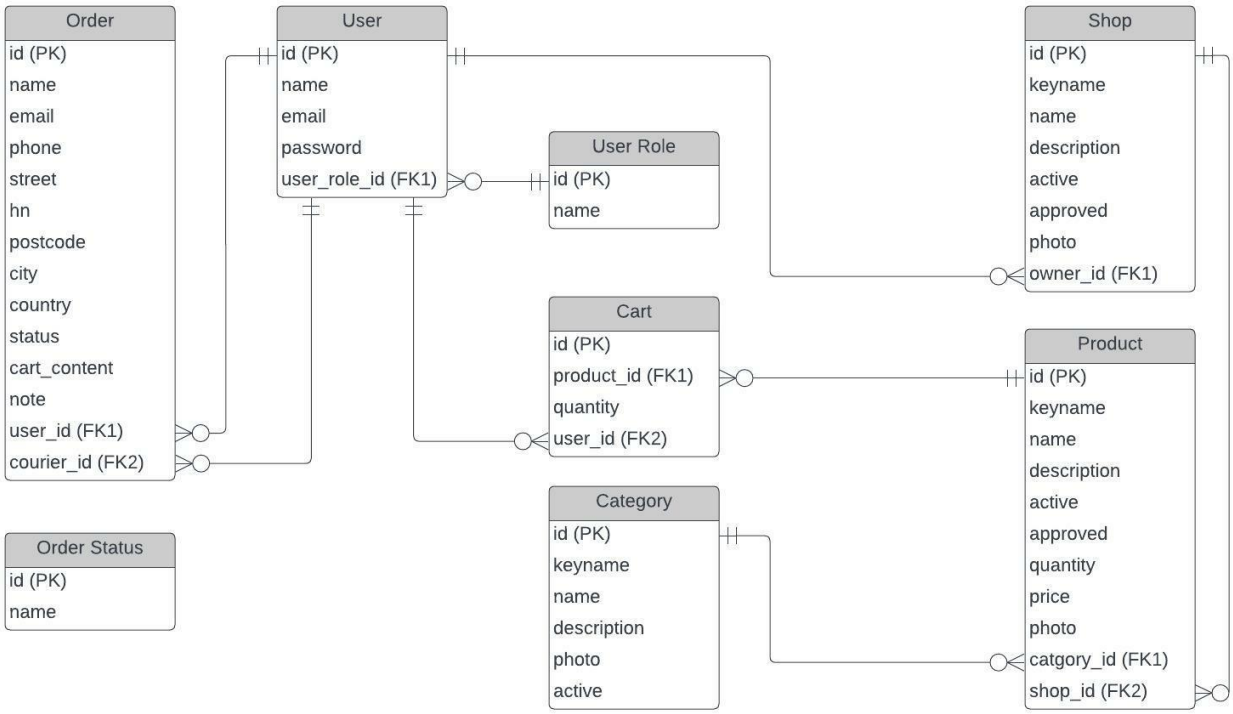
Tablica 2.5.1. HTTP zahtjevi

OBLIK	*ZNAČENJE*	*OBJAŠNJENJE*
1xx	zahtjev primljen	Zahtjev je zaprimljen te se nastavlja s obradom.
2xx	uspjeh	Zahtjev je primljen, jasno je što se traži i nastavlja se s obradom.
3xx	preusmjeravanje	Klijenta se poziva na obavljanje nove akcije.
4xx	pogreška na klijentu	Zahtjev je pogrešno oblikovan ili nije u skladu s pravilima aplikacije.
5xx	pogreška na poslužitelju	Dogodila se pogreška na poslužitelju, stoga se zahtjev ne može ostvariti.

3. MOJA TRŽNICA

3.1. Priprema prije izrade aplikacije

Aplikacija izrađena u sklopu ovog završnog rada nazvana je Moja tržnica. Prije nego se započelo s razvojem, bilo je potrebno razraditi strukturu, sve njezine funkcionalnosti i izraditi ER (*Entity Relationship*) dijagram. Kao entiteti definirani su korisnik (*user*), čiji su atributi identifikacijski broj (*id*), ime (*name*), adresa e-pošte (*email*), lozinka (*password*) te identifikator korisnikove uloge (*user_role_id*), a zatim entitet uloga korisnika (*user role*) s identifikatorom (*id*) i nazivom (*name*). Entitetu narudžbi pridruženi su atriuti identifikacijski broj (*id*), ime primatelja (*name*), adresa e-pošte i broj telefona (*phone*) primatelja, ulica (*street*) i kućni broj (*hn*) primatelja, država primatelja (*country*), status narudžbe (*status*), sadržaj (*cart_content*), bilješka (*note*) te identifikatori naručitelja i dostavljača. Atributi čija je namjena zapisati podatke o primatelju dodani su jer primatelj i naručitelj ne moraju biti ista osoba. Štoviše, primatelj ne mora biti registriran u bazi podataka ove aplikacije. Entitet status narudžbe posjeduje samo dva atributa, identifikacijski broj (*id*) i naziv (*name*). Košarica (*cart*) pamtit će podatke o proizvodu (*product_id*), količini (*quantity*) i naručitelju (*user_id*). Entitetima kategoriji (*category*), trgovini (*shop*) i proizvodu (*product*) pridruženi su atributi identifikacijski broj, generirani naziv (*keyname*), naziv (*name*), opis (*description*), aktivnost (*active*) i put do fotografije na poslužitelju (*photo*). Atributom za zabilješku odobrenja (*approved*) označit će se jesu li proizvod ili trgovina odobreni kako bi administrator mogao upravljati njihovom vidljivošću korisnicima. Svaka trgovina mora pripadati nekom korisniku koji je njezin vlasnik na temelju vrijednosti atributa vlasnika (*owner_id*). Svaki će proizvod biti svrstan u neku kategoriju, stoga mu je dodan atribut identifikator kategorije (*category_id*). Proizvodi pripadaju ponudi neke trgovine na temelju atributa identifikatora trgovine (*shop_id*). Entiteti su povezani na temelju ključeva koristeći se stranim ključevima (*foreign keys*, skraćeno FK), dok se iz baze dohvaćaju i pretražuju pomoću primarnih ključeva (*primary keys*, skraćeno PK), odnosno njihovih identifikacijskih brojeva. Kako izgleda rezultat kreiranja ER dijagrama nakon definiranja svih entiteta, atributa i prirode njihove povezanosti, prikazuje Slika 3.1.1.

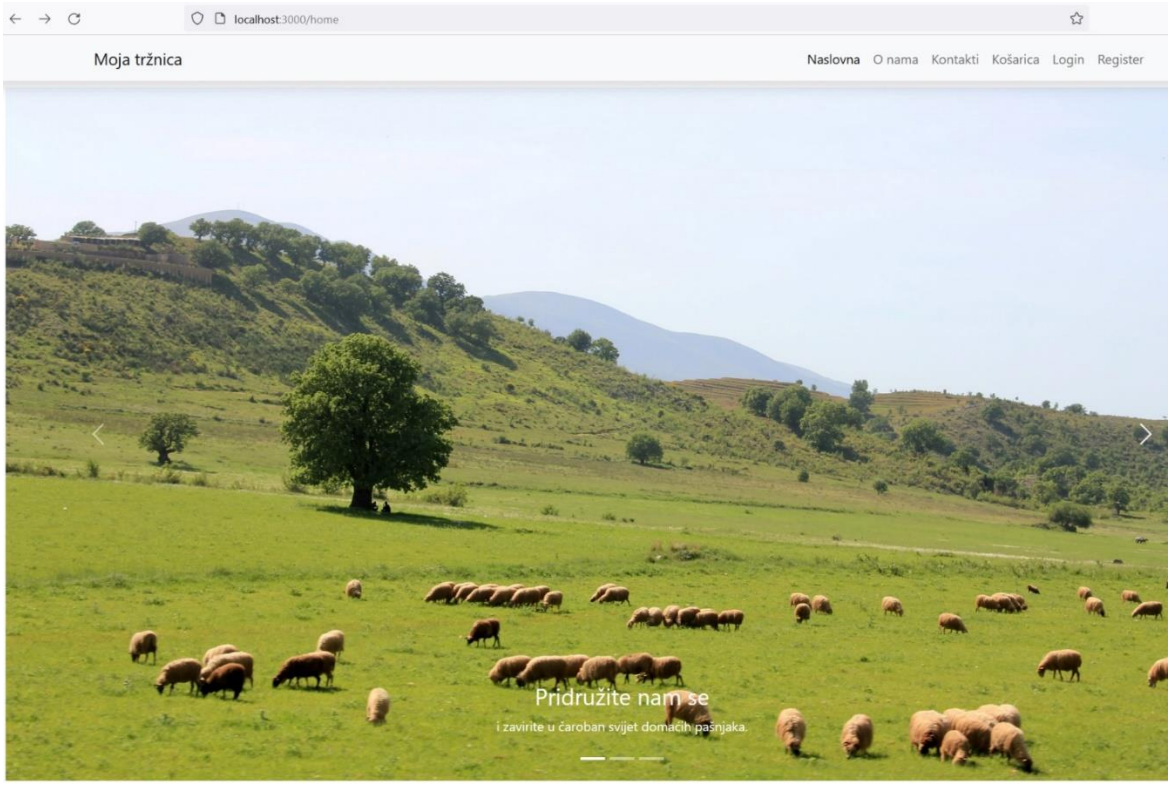


Slika 3.1.1. ER dijagram Moje tržnice

Sljedeći je korak bio napraviti dva projekta, jedan u Reactu koji će predstavljati korisnički dio aplikacije, te jedan u Laravelu koji će imati ulogu poslužitelja. Također, pomoću *npm*-a instalirani su svi dodatni paketi koji su uvelike pridonijeli razvoju ovoga projekta. XAMPP ne zahtijeva ručnu konfiguraciju, ali zato treba pripaziti da se brojevi vrata, na kojima se odvijaju procesi pridruženi modulima MySQL i Apache na upravljačkoj ploči XAMPP-a, upišu u Laravelovom projektu.

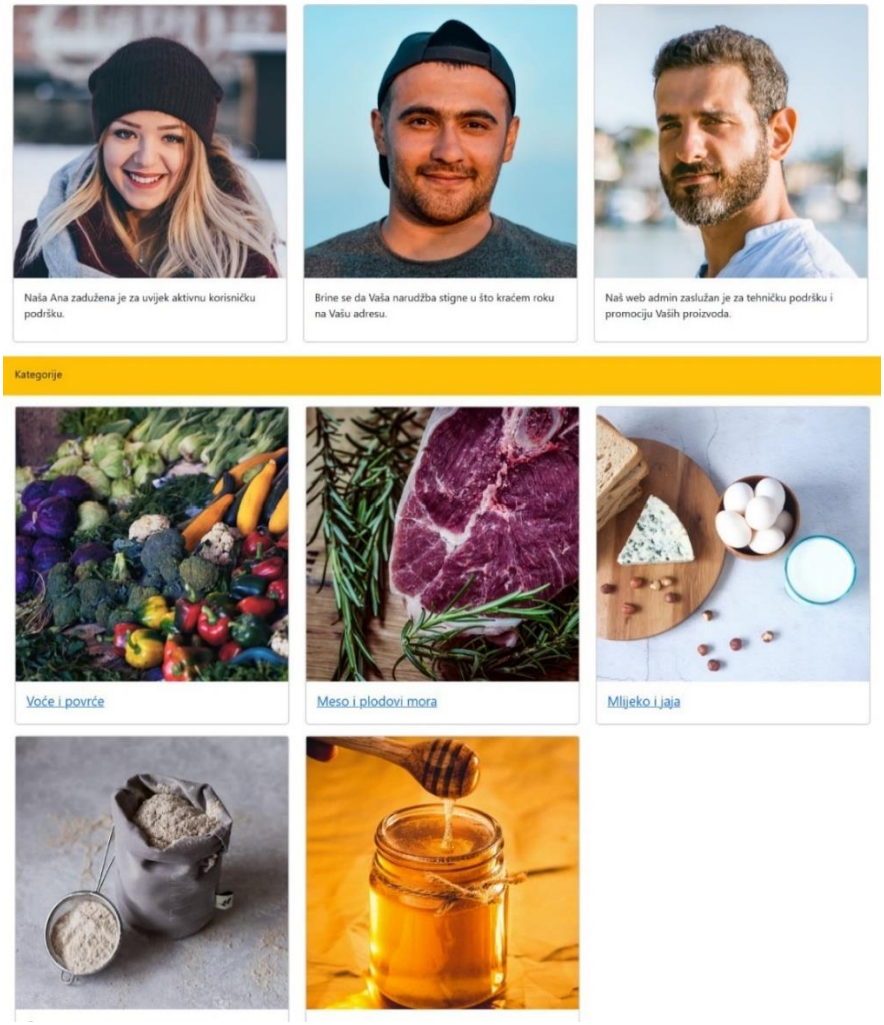
3.2. Funkcionalnosti aplikacije

Kada korisnik pristupi aplikaciji, učita se naslovna stranica prikazana na Slici 3.2.1. na kojoj se ističe prezentacija s fotografijama lokalnog krajolika.



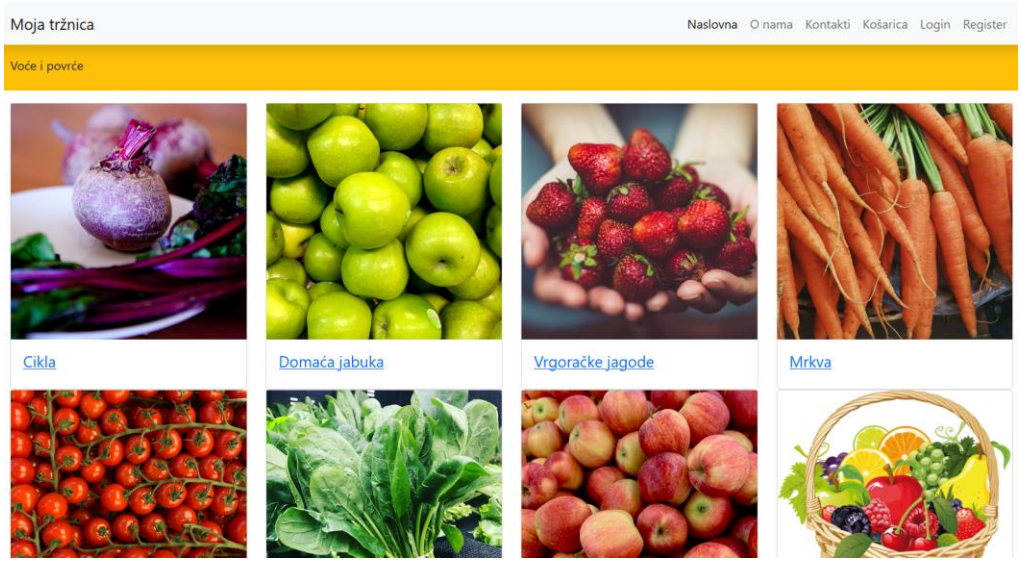
Slika 3.2.1. Naslovnica aplikacije

Na vrhu stranice nalazi se navigacijska traka sa stiliziranim imenom aplikacije i gumbima s poveznicama na uobičajene stranice s podatcima o aplikaciji i kontaktima, gumb za povratak na naslovnicu te gumbi za prijavu i registraciju. Pomičući pokazivač prema dnu stranice otkriva se odjeljak prikazan na Slici 3.2.2. u kojem su predstavljeni administratori aplikacije te kartice za svaku aktivnu kategoriju proizvoda uključujući prigodne fotografije s nazivima kategorija.



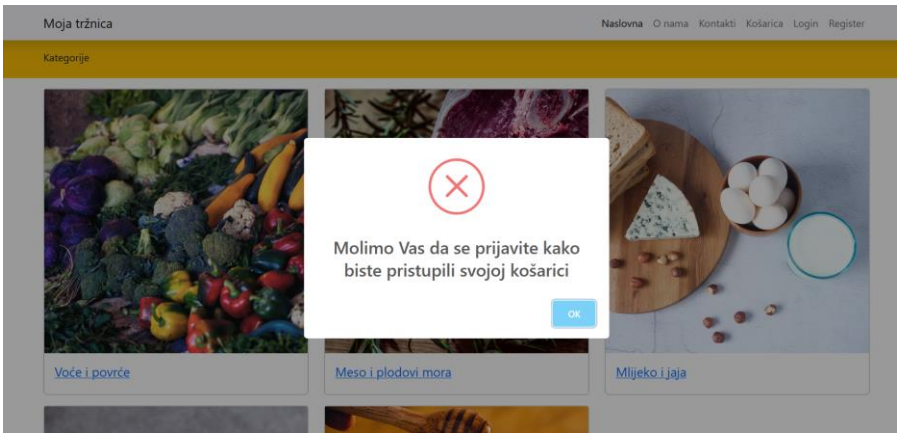
Slika 3.2.2. Prikaz kategorija

Kategorije koje je administrator aplikacije maknuo s liste aktivnih kategorija iz bilo kojeg razloga, npr. nezainteresiranost kupaca ili nepopunjenost kategorije proizvodima dulji vremenski razmak. Kartica svake kategorije poveznica je na listu proizvoda koji joj pripadaju. Na primjer, klik na kategoriju Voće i povrće prikazat će korisniku stranicu s proizvodima prikazanu na Slici 3.2.3.



Slika 3.2.3. Prikaz proizvoda

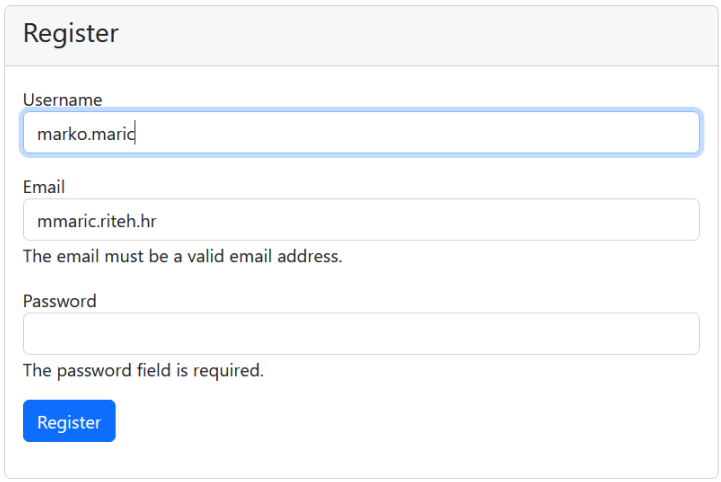
Po uzoru na listu kategorija prikazanu karticama korisnik smije pristupiti i listi svih aktivnih trgovina i obrta. Također, svaka kartica s fotografijom i nazivom obrta poveznica je na stranicu s proizvodima iz te trgovine. Klikom na gumb na navigacijskoj traci koji vodi do košarice, neprijavljenog će se korisnika upozoriti, kako se može vidjeti na Slici 3.2.4., da treba otvoriti svoj korisnički račun ili se prijaviti ispravnim podatcima ako ga je već kreirao, a zatim preusmjeriti na popis s kategorijama.



Slika 3.2.4. Obavijest korisniku koji nije prijavljen

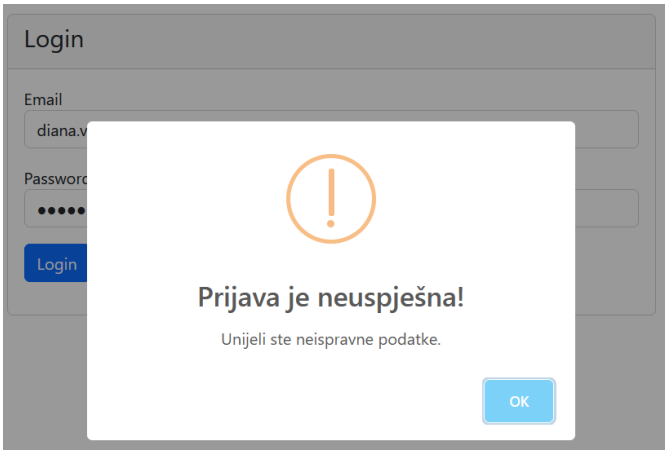
Želi li osoba napraviti novi korisnički račun, klikom na gumb Register pojavit će se upitnik u koji je potrebno unijeti osobne podatke. Nakon potvrde registracije podatci se spremaju u tablici Users na

poslužitelju. No, prije toga aplikacija vrši validaciju prema definiranim pravilima poput ispravnog formata elektroničke adrese i njezine jedinstvenosti te duljine lozinke što se može vidjeti na Slici 3.2.5. Također, zahtjev za registraciju neće biti poslan ako nisu unesena sva polja. Svakoj novoj upisanoj osobi automatski se dodjeljuje uloga običnog korisnika koja, iako nudi više funkcionalnosti nego što ih ima neregistrirani potencijalni kupac, ipak ne pokriva neke ovlasti koje su dodijeljene samo administratoru.



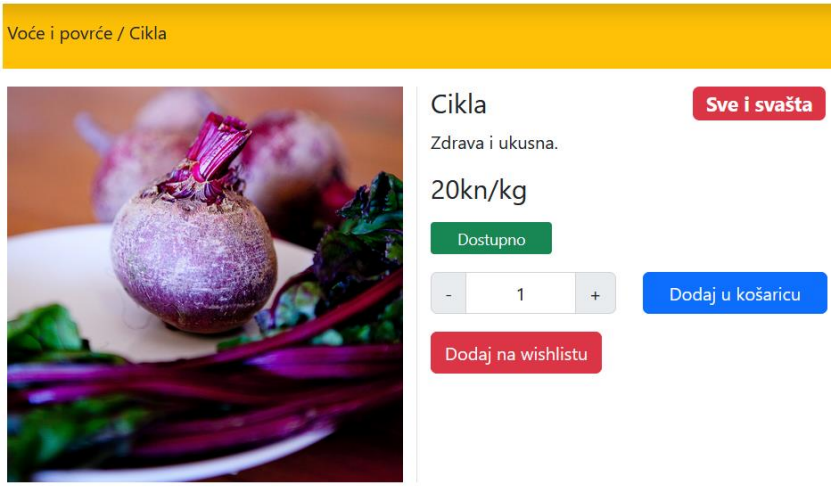
Slika 3.2.5. Upitnik za registraciju

Prijava u aplikaciju vrši se preko upitnika sličnom onome za registraciju, no traži se samo adresa e-pošte i lozinka. Uz standardnu validaciju poput formata unesenih vrijednosti ovoga se puta provjerava postoji li korisnik s unesenom adresom i lozinkom u bazi. Ako je odgovor potvrdan, preusmjerit će ga se na naslovnu stranu i moći će naručivati proizvode u aplikaciji. U suprotnom slučaju pojavit će se skočni prozor kao na Slici 3.2.6.



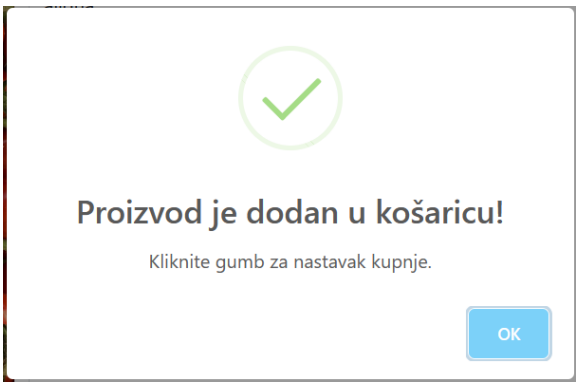
Slika 3.2.6. Obavijest o neispravnom unosu tijekom registracije

Nakon uspješne prijave korisnik može pregledavati detalje o pojedinom proizvodu poput fotografije, naziva, cijene, opisa i obrta koji ga prodaje kako se može pogledati na Slici 3.2.7.



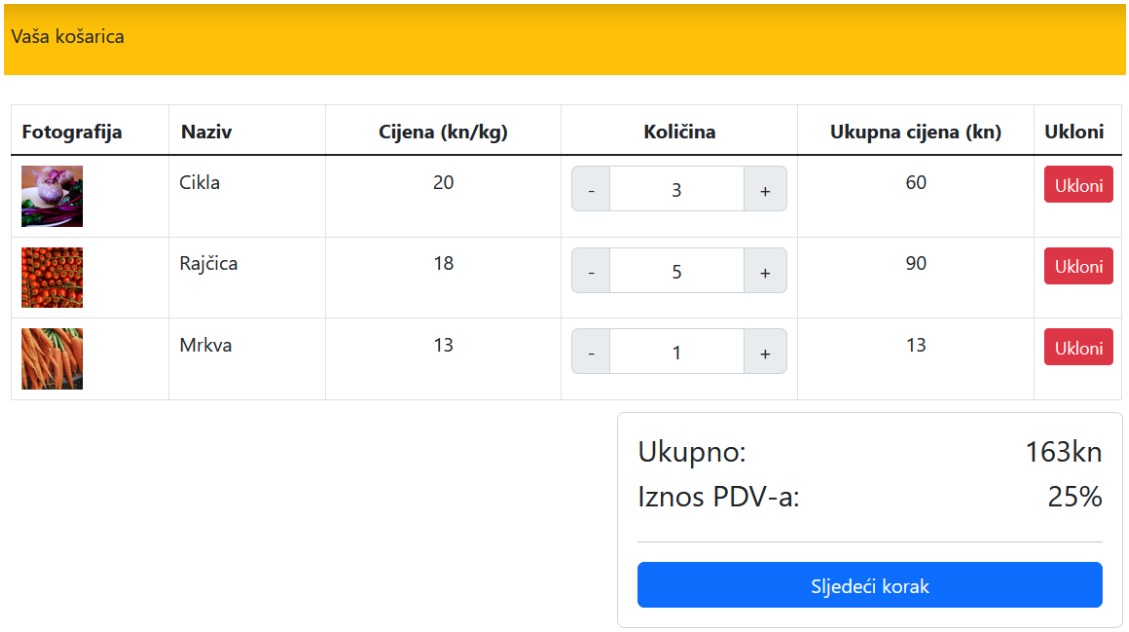
Slika 3.2.7. Prikaz podataka proizvoda

Kada proizvoda trenutno nema na stanju, ispisat će se prikladna poruka. Ako ga kupac želi dodati u košaricu, može to učiniti klikom na plavi gumb. S lijeve strane gumba može se promijeniti količina proizvoda. Na ekranu se tada ispiše poruka prikazana na Slici 3.2.8.



Slika 3.2.8. Obavijest o dodanom proizvodu

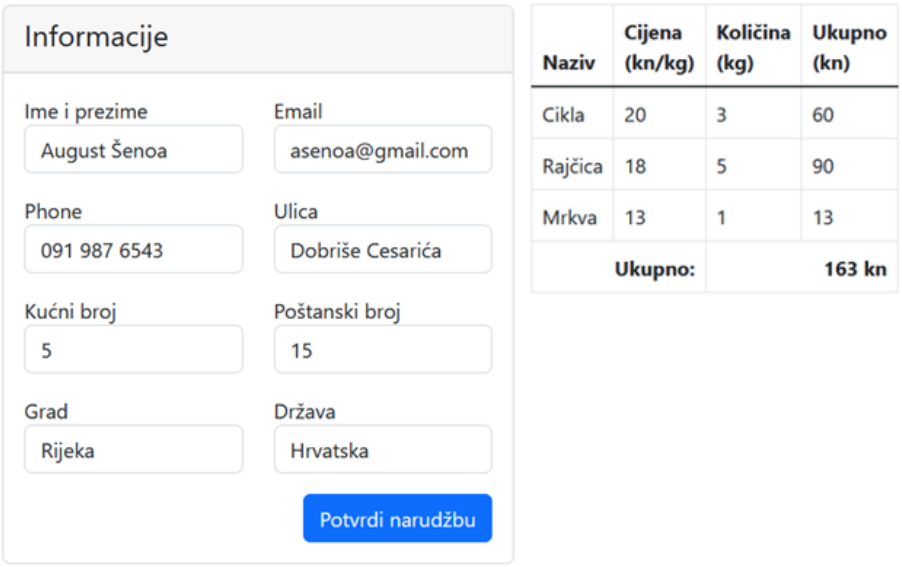
Gumb za nastavak kupnje odvest će korisnika do liste proizvoda u košarici. Izgled košarice prikazan je na Slici 3.2.9.



Slika 3.2.9. Sadržaj košarice

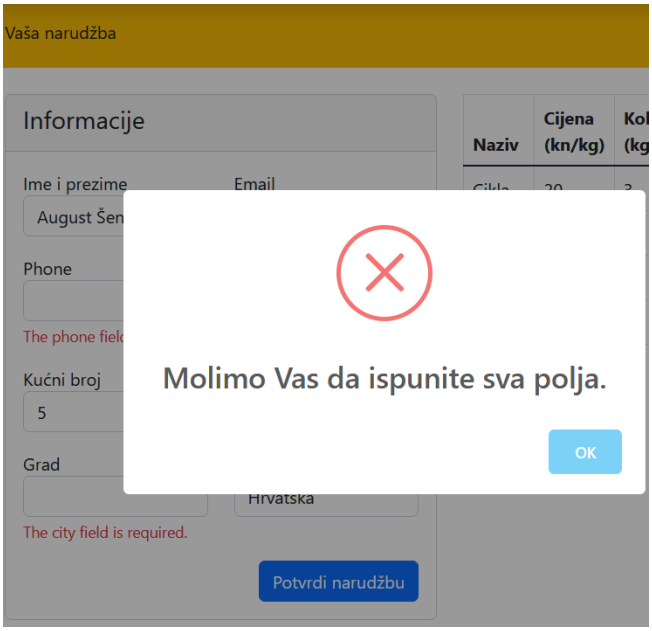
Prije nego se narudžba podnese, još uvijek se može uklanjati proizvode iz košarice ili pak mijenjati njihova količina. Sukladno tome mijenjat će se ukupna cijena. Sljedeći korak je unos podataka za dostavu kao što su ime i prezime naručitelja, broj telefona, adresa i sl. Pored upitnika za unos podataka

prikazat će se lista sa sadržajem košarice kako bi korisnik mogao još jednom provjeriti je li siguran u podnošenje narudžbe (Slika 3.2.10.).



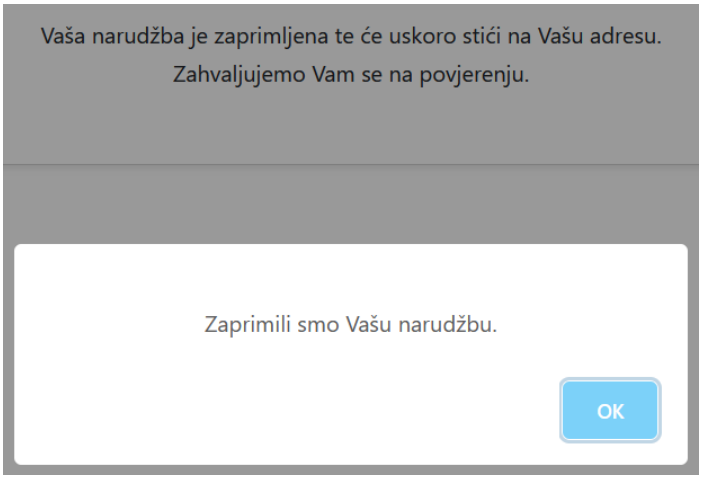
Slika 3.2.10. Upitnik za narudžbu

Ako se dogodi da nisu svi potrebni podatci uneseni, prikazat će se skočni prozor kao na Slici 3.2.11.



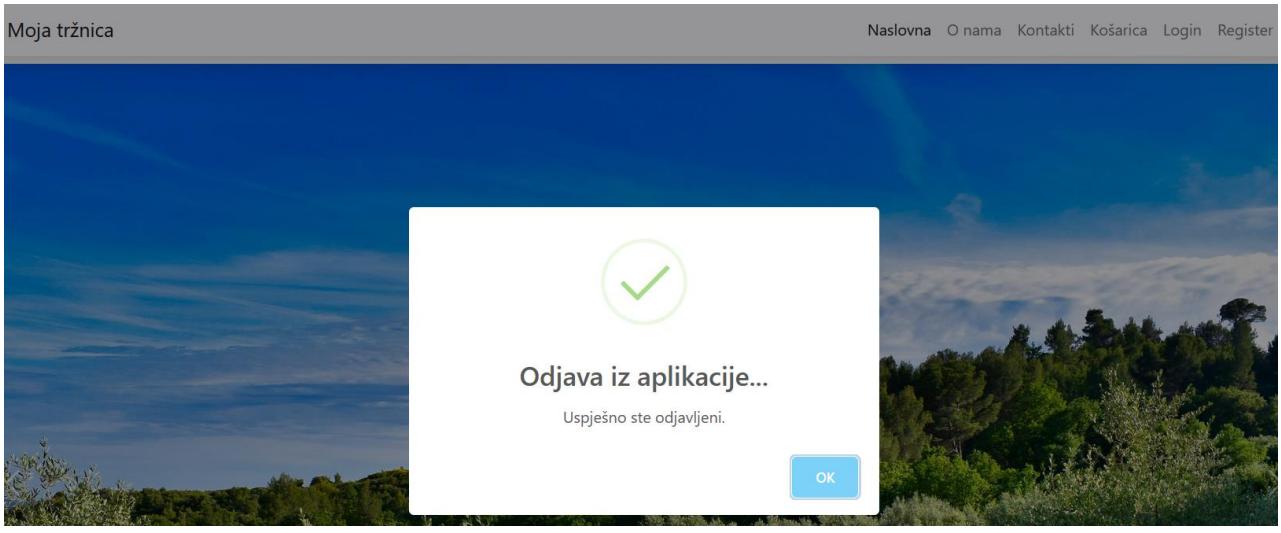
Slika 3.2.11. Upitnik za narudžbu – neispravni podatci

Sada kupac napokon može potvrditi slanje narudžbe, a nakon toga će ga se odvesti na stranicu, koja se vidi na Slici 3.2.12. s potvrdom da je narudžba zaprimljena.



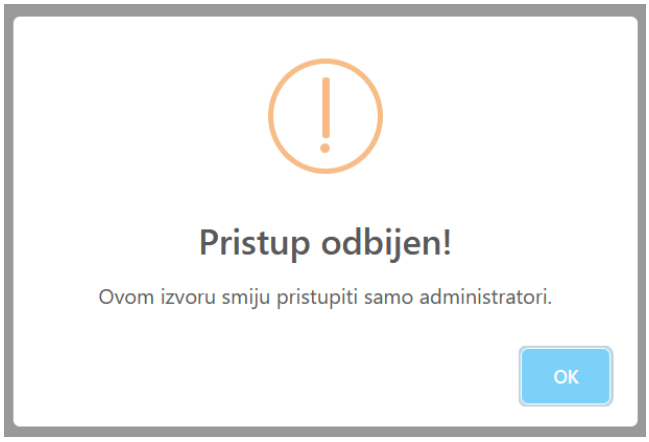
Slika 3.2.12. Potvrđena narudžba

Nakon što je korisnik pregledao i naručio željene proizvode te se želi odjaviti sa svog računa, potrebno je kliknuti na gumb *Logout* u gornjem desnom kutu. Prikazat će se prigodna poruka kako se može vidjeti na Slici 3.2.13.



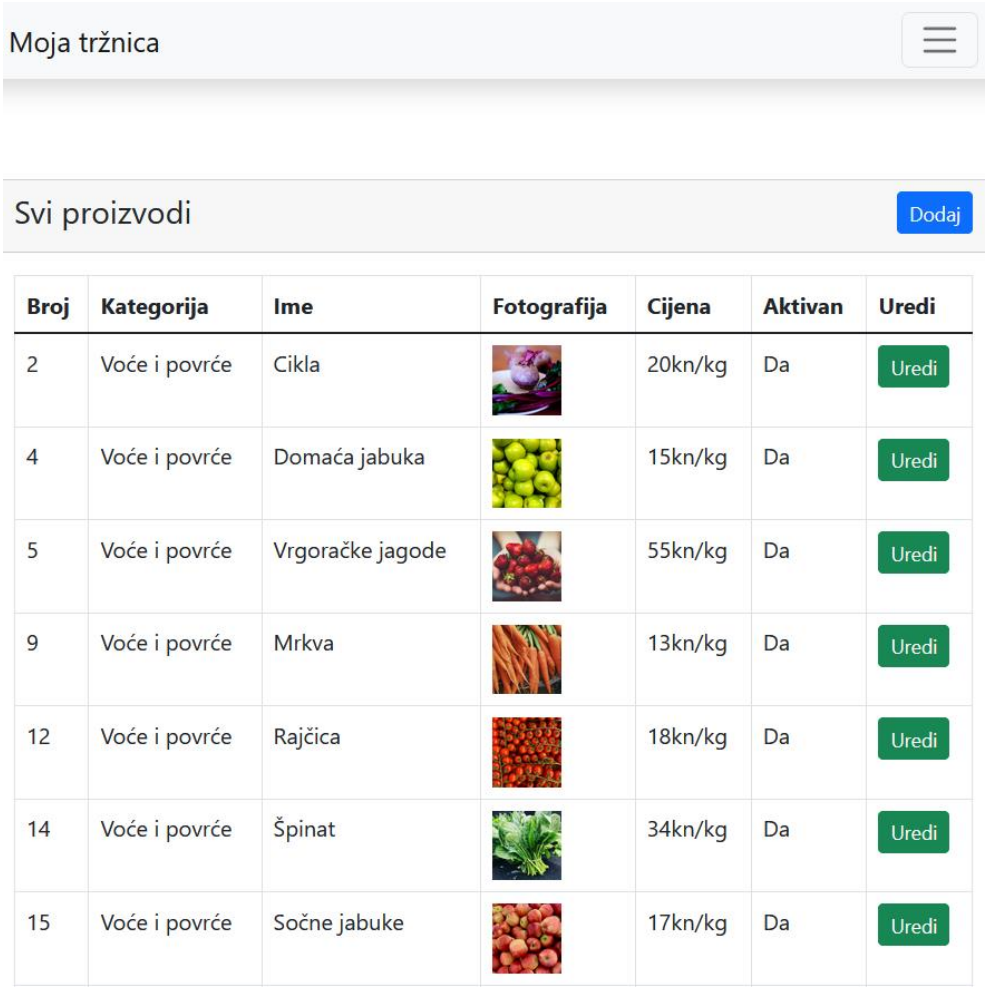
Slika 3.2.13. Odjava iz aplikacije

Može se dogoditi da korisnik koji nema administratorske ovlasti, upiše u pretraživač URL čijoj stranici pristupa samo administrator. Tada mu se na ekranu prikaže skočni prozor koji ga obavještava da nema pristup zatraženim izvorima, kako to prikazuje Slika 3.2.14.



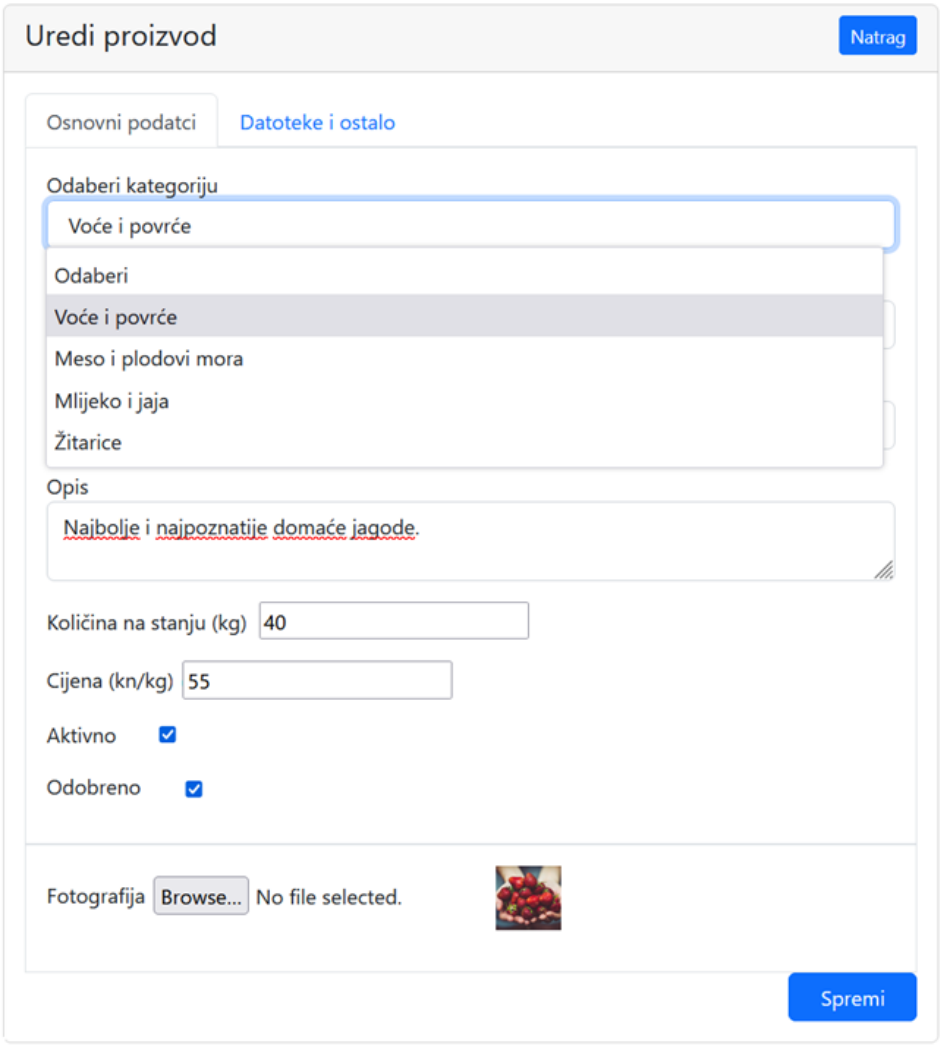
Slika 3.2.14. Poruka prilikom pristupanja nedozvoljenom resursu

Lista proizvoda koja se prikazuje administratoru drukčija je od one koja se prikazuje kupcu. Prikazuje se dodatni atribut kojim se izriče je li proizvod aktivan. Slična je lista implementirana i za popis kategorija. Na tim se listama prikazuju i oni elementi koji su u bazi označeni kao neaktivni. Svi proizvodi iz tablice, odnosno vrijednosti njihovih atributa, prikazuju se kako se može vidjeti na Slici 3.2.15.



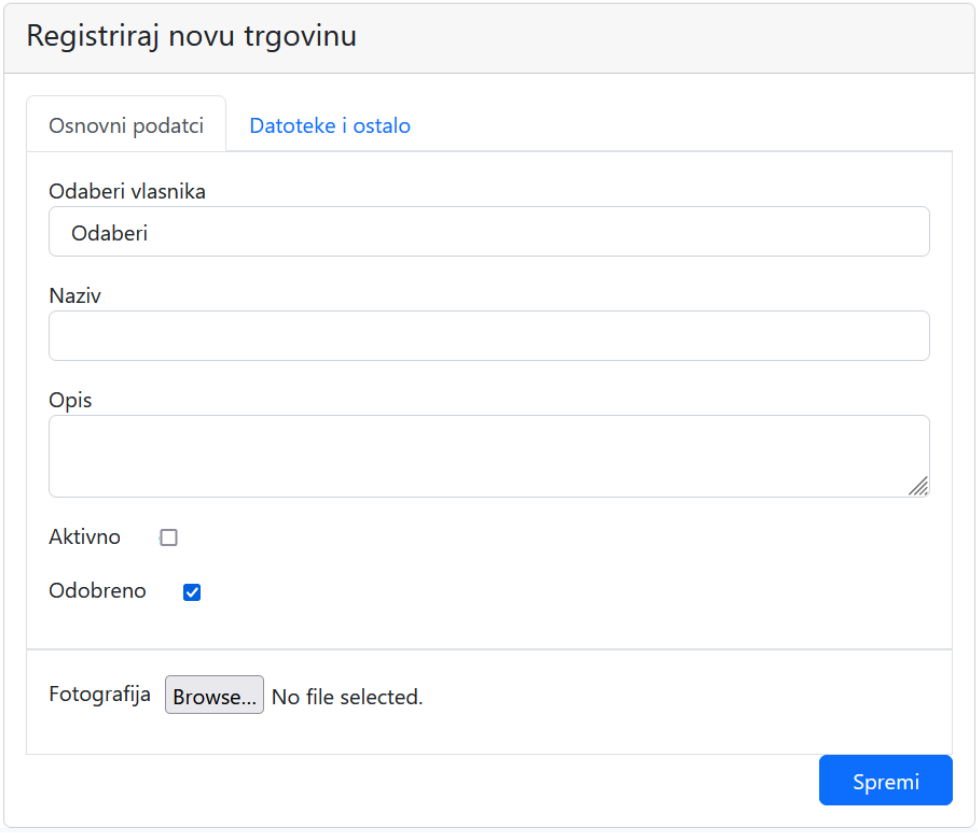
Slika 3.2.15. Lista proizvoda za administratora

Nakon što se klikne na gumb za uređivanje, prikazat će se upitnik u kojem se mogu mijenjati vrijednosti zapisane prilikom registracije. Vrijede, naravno, isti kriteriji validacije. Forma za uređivanje proizvoda može se vidjeti na slici 3.2.16.



Slika 3.2.16. Upitnik za uređivanje proizvoda

Iz padajućeg izbornika moraju se odabrati vrijednosti kategorije te obrta koji prodaje proizvod. Slični su upitnici sastavljeni za uređivanje kategorije i trgovine. Može se odabrati fotografija proizvoda te odlučiti hoće li proizvod biti aktivan i odobren, odnosno hoće li se pojavljivati na korisničkom sučelju. Kada se želi dodati (registrirati) novi objekt u sustav, mora se ispuniti forma prikazana na primjeru za dodavanje trgovine na Slici 3.2.17.



Slika 3.2.17. Upitnik za registraciju nove trgovine

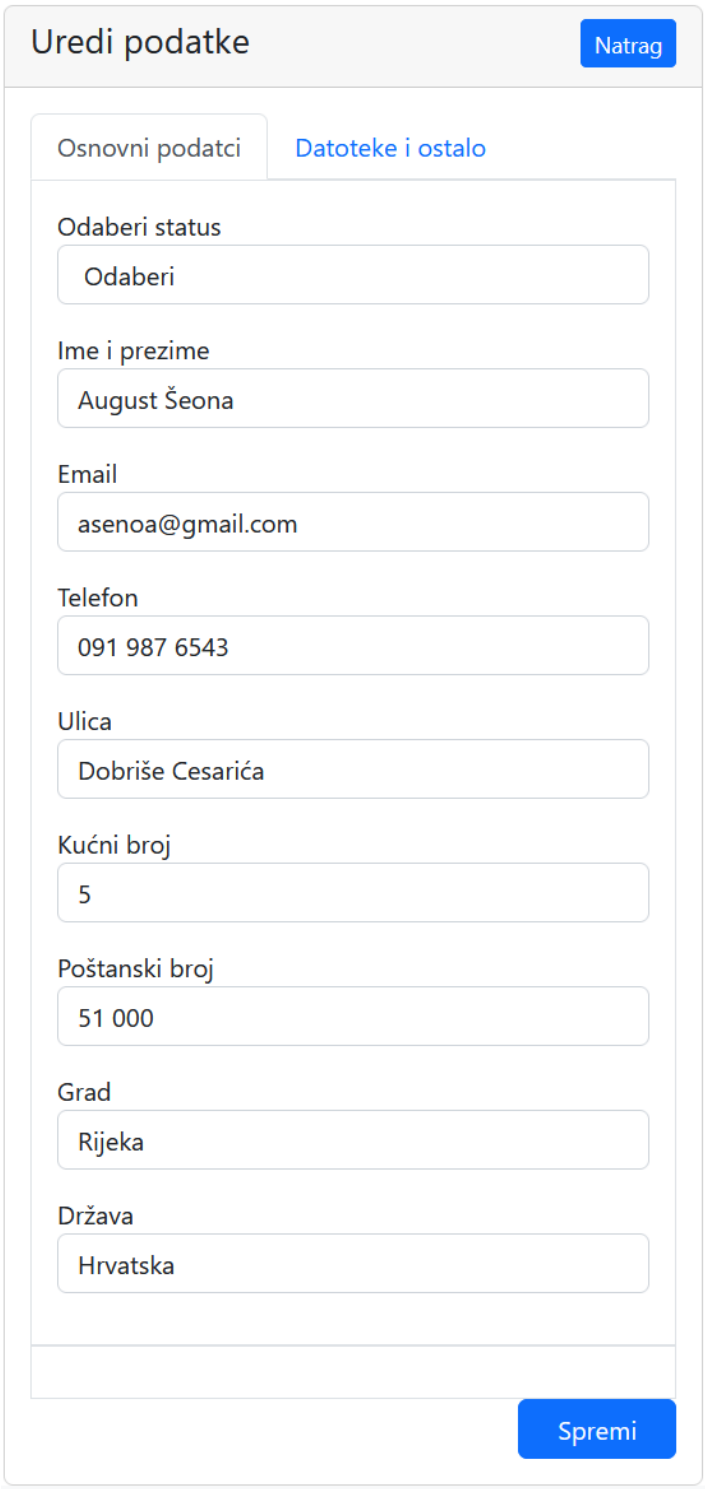
Nakon toga dodana će se trgovina prikazati na listi sa Slike 3.2.18., pod pretpostavkom da ju je administrator odobrio i aktivirao.

Slika 3.2.18. Lista obrta za administratora

Posljednja funkcionalnost za koju je administrator ovlašten je upravljanje narudžbama. On ih može vidjeti na popisu kako prikazuje Slika 3.2.19.

Slika 3.2.19. Lista narudžbi

A kada želi provjeriti detalje ili urediti podatke, može to učiniti kikom na pripadajući gumb. Tada se otvara forma sa Slike 3.2.20.



Slika 3.2.20. Detalji narudžbe za administratora

4. OPIS PROCESA NA RAZINI KODA

4.1. Registracija korisnika

Uobičajeni procesi registracije, prijave registriranog korisnika i odjave ostvareni su Laravelovim paketom Sanctumom te Reactovim mrežnim objektom za upravljanje podatcima iz sesije. U glavnoj komponenti na klijentskoj strani App.js potrebno je pripaziti na konfiguraciju CORS-a (Cross-Origin Resource Sharing). Varijabla *withCredentials* postavljena je na *true*, čime je omogućeno prosljeđivanje korisničkih podataka u zahtjevima koji se šalju između različitih domena. Axios presretači (interceptori) služe za konfiguraciju parametara zaglavlja zahtjeva, a definirano je da će se autorizacija vršiti koristeći se tokenom *Bearer*. *Bearer* token je niz znakova koji nema posebno značenje. Ne može se, kao neke druge vrste tokena, raščlaniti na dijelove koji u sebi nose enkriptirane informacije poput upotrijebljenog *hash* algoritma ili imena prijavljenog korisnika. Svi odgovori koji budu poslani iz Laravelovog dijela aplikacije na lokalnom poslužitelju, a čiji je URL *http://localhost:8000/*, prihvatit će se u JSON (JavaScript Object Notation) formatu. Navedena konfiguracija prikazana je na Slika 4.1.1.

```
axios.defaults.withCredentials = true;
axios.interceptors.request.use(function (config) {
  const token = localStorage.getItem('auth_token');
  config.headers.Authorization = token ? `Bearer ${token}` : '';
  return config;
});

axios.defaults.headers.post['Accept'] = 'application/json';
axios.defaults.headers.post['Content-Type'] = 'application/json';
axios.defaults.baseURL = "http://localhost:8000/";
```

Slika 4.1.1. Konfiguracija CORS-a

Klikom na gumb za registraciju poziva se pripadajući URL vezan s komponentom Register.js izrađenoj u Reactu s upitnikom za unos korisničkog imena, adrese elektroničke pošte i lozinke. Koristeći se mrežnim objektom utvrđuje se postoji li *Bearer* token u lokalnoj pohrani kako prikazuje Slika 4.1.2.

```
<Route
  path='/register'
  element={ localStorage.getItem('auth_token') ? <Navigate to="/home" /> : <Register /> }
/>
```

Slika 4.1.2. Provjera postoji li Bearer token u lokalnoj memoriji

Ako je klijent dobio odgovor u kojem je sadržana vrijednost tokena, znači da je korisnik prijavljen, stoga će se trenutni URL pomoću elementa *Navigate* zamijeniti URL-om naslovne strane.

Struktura programskog koda na klijentskoj strani ovoga projekta slična je za svaku stranicu. Najprije se dodaju paketi kojima se želi koristiti, a potom se kreće s pisanjem funkcije koja će na *korisničkom sučelju* vratiti vrijednosti varijabli te HTML i JavaScript elemente. Dakle, potrebno je unutar funkcije, odmah na početku, definirati varijable čijim će se vrijednostima upravljati kroz interakciju s osobom koja koristi aplikaciju kako to prikazuje Slika 4.1.3. na primjeru JavaScript komponente za registraciju. Sve su vrijednosti postavljene kao prazan znakovni niz ili prazan niz poput varijable *error_message* u kojoj će se zapisivati poruke upozorenja tijekom provjere ispravnosti formata podataka.

```
const [registerInput, setRegister] = useState({
    name: '',
    email: '',
    password: '',
    error_message: [],
});
```

Slika 4.1.3. Inicijalizacija konstante za registraciju

Zatim se napiše konstanta *handleInput* čija je svrha praćenje promjena u poljima za unos vrijednosti koristeći se pritom sintetičkim događajem (eng. *event*). Sintetički događaji omataju događaje među kojima postoje manje razlike i koji su karakteristični samo za određene vrste preglednika. Na taj se način stvara unificirano aplikacijsko programsko sučelje i osigurava besprijekoran rad aplikacije u različitim uvjetima. Primjer konstante može se vidjeti na Slici 4.1.4.

```
const handleInput = (e) => {
    e.persist();
    setRegister({...registerInput, [e.target.name]: e.target.value });
}
```

Slika 4.1.4. Konstanta handleInput()

Nakon toga potrebno je implementirati slanje podataka poslužiteljskoj strani. Prilikom definiranja konstante *registerSubmit* također je upotrijebljen sintetički događaj s obzirom da ona ima ulogu funkcije koja se poziva klikom na gumb u upitniku za unos podataka. Unutar nje prikupljaju se vrijednosti atributa i zapisuju u obliku niza. U ovom su slučaju ti atributi ime na koje se osoba želi registrirati, adresa e-pošte i lozinka. Potom se postavi CSRF (Cross-Site Request Forgery) zaštita. CSRF napadi djeluju na način da se osobu navede na ispunjavanje upitnika kojim upravlja napadač. Unesene vrijednosti pošalju se na krivo odredište čime mogu biti ugroženi mnogi osjetljivi podatci. Slijedi pozivanje rute za registraciju koja je definirana na poslužiteljskoj strani. Priroda zahtjeva je *POST*, kao i uvijek kada se u bazi podataka dodaju novi zapisi. Slika 4.1.5. prikazuje opisani proces programiranja.

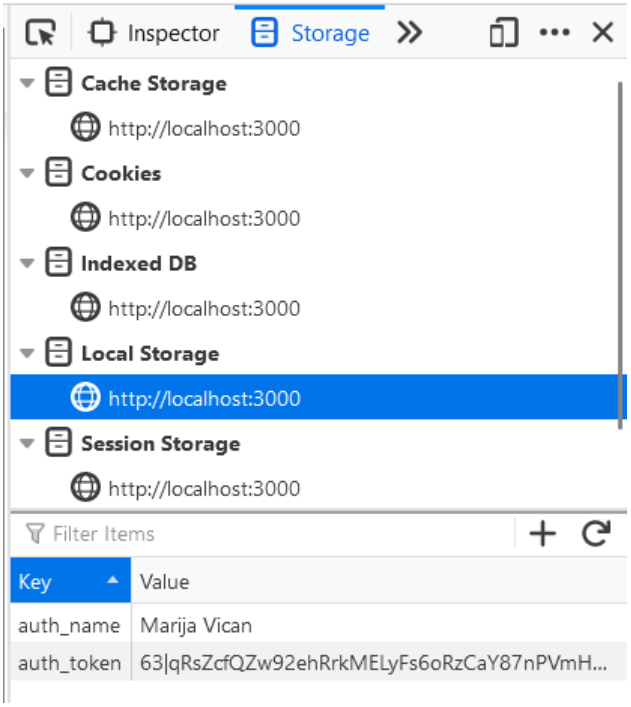
```
const registerSubmit = (e) => {
    e.preventDefault();

    const data = {
        name: registerInput.name,
        email: registerInput.email,
        password: registerInput.password,
    }

    axios.get('/sanctum/csrf-cookie').then(response => {
        axios.post(`/api/register`, data).then(res => {
            if (res.data.status === 200) {
                localStorage.setItem('auth_token', res.data.token);
                localStorage.setItem('auth_name', res.data.username);
                swal("Registrirani ste!", "", "success");
                navigate('/home');
            } else {
                setRegister({
                    ...registerInput, error_message: res.data.validation_errors
                });
            }
        });
    });
}
```

Slika 4.1.5. Slanje podataka poslužitelju

Da je registracija uspješno završena, prepoznaje se po vraćenom statusnom kodu 200. U lokalnoj se memoriji pomoću varijabli *auth_token* i *auth_name* te funkcije *setItem()* zapamte vrijednosti tokena i korisničkog imena vraćenih s poslužiteljske strane. Uvid u vrijednosti varijabli može se dobiti tako da se desnim klikom na stranici otvori izbornik, klikne na opciju *Inspect*, a zatim na odjeljak *Storage*. U njemu se pod opcijom *Local Storage* mogu očitati korisničko ime i token kako prikazuje Slika 1.4.6.



Slika 4.1.6. Token i korisničko ime u lokalnoj pohrani

Funkcija *swal()* iz paketa *Sweetalert* [12] otvara skočni prozor kako bi se korisnika obavijestilo o prijavi, a rabi se kao zamjena funkciji *window.alert()*. Kao parametre prima glavnu poruku, kratko objašnjenje napisano manjim fontom koje će se smjestiti ispod poruke te naziv prikladne ikone, ovom slučaju *success* za prikaz kvačice odobrenja. Zatim se pomoću funkcije *navigate()* korisnik preusmjeri na stranicu čiji je URL *http://localhost:8000/*.

Može se dogoditi da vrijednosti unesene u polja za registraciju ne zadovolje kriterije validacije. Tada se funkcijom *setRegister()* dohvate vraćene poruke i spreme u varijablu *error_message* kojom će se koristiti prilikom ispisa prikladnog teksta ispod svakog polja.

Preostaje još posložiti dijelove korisničkog sučelja. Komponenta Register.js u svom *return* dijelu sadrži uobičajene HTML elemente poput *div* i *span* odjeljaka, naslova poglavlja i gumba kojima stilizira upitnik za registraciju. Najčešće korištene klase su klase za oblikovanje spremnika, redova i stupaca, kartica te uređivanje gumba. Osim toga, ovdje se nalazi i forma za unos podataka koji se šalju poslužiteljskoj strani klikom na gumb za potvrdu. Prilikom kreiranja forme vrijednost parametra onSubmit govori što će se dogoditi nakon podnošenja upitnika. U ovom slučaju ta je vrijednost prije spomenuta i objašnjena konstanta *registerSubmit*. Svako polje za unos definirano *input* komponentom povezano je varijablama u *registerInputu* kroz parametar *value* čija se vrijednost mora staviti u

vitičaste zarade s obzirom na to da se koristi nazivom varijabli umjesto konkretnim vrijednostima. Odmah ispod, u *span* komponenti, pozivaju se vrijednosti poruka koje se vraćaju prilikom nezadovoljenih uvjeta validacije. Kako izgleda dovršeni upitnik, vidi se na Slici 4.1.7.

```
<form onSubmit={registerSubmit}>
    <div className='form-group mb-3'>
        <label>Username</label>
        <input type='' name='name' onChange={handleInput}
            value={ registerInput.name }
            className='form-control'>
        </input>
        <span>{ registerInput.error_message.name }</span>
    </div>
    <div className='form-group mb-3'>
        <label>Email</label>
        <input type='' name='email' onChange={handleInput}
            value={ registerInput.email }
            className='form-control'>
        </input>
        <span>{ registerInput.error_message.email }</span>
    </div>
    <div className='form-group mb-3'>
        <label>Password</label>
        <input type='' name='password' onChange={handleInput}
            value={ registerInput.password }
            className='form-control'>
        </input>
        <span>{ registerInput.error_message.password }</span>
    </div>
    <div className='form-group mb-3'>
        <button type='submit' className='btn btn-primary'>
            Register
        </button>
    </div>
</form>
```

Slika 4.1.7. Upitnik za registraciju u datoteci Register.js

Nakon što se u klijentovom zahtjevu pozove ruta definirana u datoteci api.php u Laravelovom dijelu aplikacije, poziva se funkcija za registraciju koja se nalazi u AuthControlleru. AuthController sadrži uobičajene funkcije za prijavu, odjavu i dodavanje novih korisnika. Dakle, funkcija *register()* primit će nadolazeći HTTP zahtjev s vrijednostima kojima se klijent želi registrirati. Parametar funkcije definiran je kao objekt klase *Request* koja posjeduje razne metode za lakše baratanje dobivenim

vrijednostima. Na primjer, metodom *isJson()* provjerava se šalje li klijent uistinu zahtjev formata JSON. Može se saznati i očekuje li klijent odgovor u tom obliku koristeći se metodom *wantsJson()*. Za dohvat vrijednosti, pod pretpostavkom da je varijabla *$request* instanca klase *Request*, dovoljno je napisati *$request->ime_varijable*. Ako se pak želi u jednoj varijabli spremiti cijeli sadržaj zahtjeva, dovoljno je pozvati metodu *all()* čime će se dobiti asocijativni niz podataka. Funkcijom *only()* filtriraju se podatci na način da se odbaci sve što nije napisano u njezinim parametrima. Iako se metodom *validate()* može ocijeniti ispravnost formata podataka i je li neka vrijednost nužna za unos, u ovom je projektu u tu svrhu upotrijebljen *Validator*, jedna od Laravelovih fasada. Njome se može postaviti najmanja i najveća duljina unosa, odrediti je li podatak obvezan ili zahtijevati unos isključivo u formatu adrese e-pošte. Jedan od vrlo moćnih parametara je *unique* koji se postavlja kada vrijednost mora biti jedinstvena u tablici podataka što bi se inače moralo ručno provjeriti. Dovoljno je samo navesti naziv tablice i atributa. Na Slici 4.1.8. može se vidjeti kako se Validatorom koristi prilikom validacije podataka za registraciju.

```
$validator = Validator::make($request->all(), [
    'name' => 'required',
    'email' => 'required|email|unique:users,email',
    'password' => 'required|min:8',
]);

if ($validator->fails()) {
    return response()->json([
        'validation_errors' =>$validator->messages(),
    ]);
}
```

Slika 4.1.8. Uporaba validatora prilikom registracije

Metodom *fails()*, koja vraća vrijednost *true* u slučaju nezadovoljenih kriterija, koristi se tijekom oblikovanja odgovora koji će stići klijentskoj strani. *Validator* sadrži implementirane prikladne poruke u slučaju pogreške pa ih se može upotrijebiti za ispis obavijesti u Reactovom dijelu aplikacije. Ako je sve u redu, podatci iz HTML zahtjeva zapišu se u tablicu u bazi pomoću funkcije *create()* kako prikazuje Slika 4.1.9.

```php
$user = User::create([
    'name' => $request->name,
    'email' => $request->email,
    'user_role_id' => 2,
    'password' => Hash::make($request->password),
]);
$token = $user->createToken('usertoken', [''])->plainTextToken;
```

Slika 4.1.9. Spremanje korisnika u bazi na poslužitelju

U bazi na lokalnom serveru dodan je novi korisnik kako se može vidjeti na Slici 4.1.10. Radi jednostavnosti prikazana su samo za ovaj primjer najbitnija polja. Kako se ne bi zlorabile, lozinke se zaštite koristeći se fasadom *Hash* tako da ni osoba koja ima pristup bazi neće moći iščitati njihovu vrijednost.

id 1	name	email	password	user_role_id
6	Marija Vican	marijav@gmail.com	$2y$10$iPFdxckAnpfVK4XRQjNjteXXNbxmj4018AKo32H7rl5...	2

Slika 4.1.10. Konfiguracija CORS-a

Funkcija *createToken()* registriranom će korisniku pridružiti jedinstveni token i zapisati ga u tablici *personal_access_tokens* gdje se nalaze svi tokeni trenutno prijavljenih korisnika u aplikaciji. Svakom novom korisniku nikad neće biti dodijeljena uloga administratora, stoga će se njegov token prikladno nazvati *usertoken*, a pod atributom *abilities* zapisat će se vrijednost [""], kako se može uočiti na Slici 4.1.11., što znači da vlasnik tog tokena nema administratorske ovlasti.

id	tokenable_type	tokenable_id	name	token	abilities
65	App\Models\User	6	usertoken	ef7a0259952094fd016abc3688d01f2035b932e1c3a0cb012e...	[""]

Slika 4.1.11. Token običnog korisnika bez administratorskih ovlasti

Zatim se oblikuje odgovor koji će se vratiti klijentu. On sadržava vrijednost statusa koja iznosi 200 za prihvaćen i uspješno procesuiran zahtjev, korisničko ime registrirane osobe, upravo generirani token i poruku odobrenja. Nabrojane vrijednosti pohranjene su u niz, a s obzirom da se zahtjev može poslati samo u JSON formatu, ne smije ga se zaboraviti pretvoriti u taj oblik kao na Slici 4.1.12.

```
return response()->json([
    'status' => 200,
    'username' => $user->name,
    'token' => $token,
    'message' => 'You are successfully registered.'
]);
```

Slika 4.1.12. Odgovor klijentu nakon uspješne registracije

Proces odjavljivanja iz sustava započinje kada osoba klikne na gumb za odjavu definiran u komponenti Navbar.js. Tada se izvršava kod napisan u konstanti *logoutSubmit()*. Ovdje se također koristi sintetičkim događajem te njegovom funkcijom *preventDefault()* koja sprječava učitavanje stranice svaki put kada se pošalje neki upitnik, odnosno u ovom slučaju zatraži zahtjev za odjavom. Upotrijebljena je metoda *POST* te se specificira ogovarajuća ruta koja se nalazi u datoteci api.php u Laravelovoj aplikaciji. Ako je odjava uspješno izvršena, pomoću objekta *localStorage* potrebno je ukloniti podatke iz lokalne pohrane, odnosno token i korisničko ime. Potom se funkcijom s*wal()* prikladno uredi izgled skočnog prozora koji se pojavljuje nakon odjave korisnika kako se može vidjeti na Slici 4.1.13.

```
axios.post(`/api/logout`).then(res => {
    if (res.data.status === 200) {
        localStorage.removeItem('auth_token');
        localStorage.removeItem('auth_name');
        swal("Odjava...", "Uspješno ste odjavljeni iz aplikacije.", "success");
        navigate('/home');
    }
});
```

Slika 4.1.13. Brisanje podataka iz lokalne memorije nakon odjave

Dakle, ruta *logout* definirana u Laravelu kako prikazuje Slika 4.1.14. poziva funkciju za odjavu korisnika koja je napisana u AuthControlleru.

```php
Route::middleware(['auth:sanctum'])->group(function () {
    Route::post('/logout', [AuthController::class, 'logout']);
});
```

Slika 4.1.14. Ruta za odjavu u api.php

Rabeći posrednika (eng. *middleware*) filtriraju se HTTP zahtjevi i u obzir uzimaju samo oni iz kojih se može iščitati da je korisnik autenticiran koristeći se *Sanctumom*. U Laravelu postoji funkcija *auth()* kojom se na jednostavan način mogu dohvatiti korisnikovi podatci. Ako se osoba odjavi iz sustava, mora se izbrisati i njezin token zapisan u tablici *personal_access_tokens*. To će se učiniti na način kako prikazuje Slika 4.1.15.

```php
public function logout(Request $request) {
    auth()->user()->tokens()->delete();

    return response()->json([
        'status' => 200,
        'message' => 'Uspješno ste odjavljeni.'
    ]);
}
```

Slika 4.1.15. Odgovor klijentu nakon odjave

Odgovor koji će se poslati klijentu potrebno je pretvoriti u JSON format. U njemu su sadržane vrijednost statusnog koda, koji iznosi 200 što označava uspješno izvršen zahtjev, te prikladna poruka da je korisnik odjavljen.

4.2. Registracija proizvoda

U svrhu dodavanja novog proizvoda na klijentskoj strani kreirana je datoteka AddProduct.js u kojoj je uređeno korisničko sučelje za unos vrijednosti. Na početku su definirane konstante pomoću kojih će se spremati dohvaćene kategorije i trgovine te upisane vrijednosti atributa proizvoda. Fotografija proizvoda morala se odvojiti od ostali atributa i posebno definirati zbog načina na koji se šalju datoteke. Podrazumijevat će se da novi proizvod nije odobren dok to administrator ne učini, stoga je zadana vrijednost varijable *approved* postavljena na *false*. Ostale su varijable prazni znakovni nizovi što će se poslije promijeniti kada se unesu stvarne vrijednosti u korisničkom sučelju. Upotrijebljene konstante mogu se vidjeti na Slici 4.2.1., a njihove će se vrijednosti, kao i tijekom unosa podataka za novog korisnika, postavljati metodom *setNazivKonstante()*.

```
const [selectedCategories, setSelectedCategories] = useState([]);
const [selectedShops, setSelectedShops] = useState([]);
const [productInput, setProduct] = useState({
    category_id: '',
    name: '',
    description: '',
    active: '',
    approved: false,
    quantity: '',
    price: '',

});
const [productPhoto, setProductPhoto] = useState([]);
```

Slika 4.2.1. Inicijalizacij varijabli prije unosa novog proizvoda

Sljedeći korak je napisati konstante koje će paziti na promjene koje se događaju u poljima za unos vrijednosti. Također, potrebno je posebno implementirati sličnu funkcionalnost za učitavanje fotografije kako prikazuje Slika 4.2.2. gdje se dohvaća prva vrijednost varijable *files*.

```
const handleProductPhoto = (e) => {
    e.persist();
    setProductPhoto({ photo:e.target.files[0] })
}
```

Slika 4.2.2. Učitavanje fotografije proizvoda

Unos kategorije kojoj proizvod pripada, kao i naziv trgovine je nužan. Kako bi administrator mogao odabrati željene vrijednosti u padajućem izborniku, najprije ih se mora dohvatiti iz baze podataka. Pritom će se koristiti *axiosom* i metodom *GET*. Ako je vraćeni statusni kod 200, znači da su podatci uspješno pristigli klijentskoj strani i može ih se pomoću funkcije funckije *set()* pospremiti u odgovarajuće varijable kako to prikazuje Slika 4.2.3. Ovaj je dio koda napisan unutar prethodno spomenute Reactove kuke *useEffect(),* a u njemu se definiraju rute *select-category* i *select-shop* definirane na poslužiteljskoj strani.

```
useEffect(() => {
    axios.get(`/api/select-category`).then(res => {
        if (res.data.status === 200) {
            setSelectedCategories(res.data.categories);
        }
    });
    axios.get(`/api/select-shop`).then(res => {
        if (res.data.status === 200) {
            setSelectedShops(res.data.shops);
        }
    });
}, []);
```

Slika 4.2.3. Dohvaćanje liste kategorija i trgovina s poslužitelja

Potom se u Laravelovom dijelu aplikacije metodom *GET* pozivaju odgovarajuće funkcije unutar *CategoryControllera* i *ShopControllera.* U obzir se uzimaju samo trenutno aktivne kategorije i obrti što se naznači u metodi *where()* prilikom dohvaćanja podataka iz tablice kako je prikazano na Slici 4.2.4. na primjeru selekcije kategorija.

```php
public function select() {
    $categories = Category::where('active', 1)->get();

    return response()->json([
        'categories' => $categories,
        'status' => 200
    ]);
}
```

Slika 4.2.4. Vraćanje liste kategorija i trgovina klijentu

Rezultati se vrate klijentskoj strani koji ih tada može prikazati unutar HTML-ove *select* komponente. Pritom se rabi JavaScriptova metoda *map()* kojom se iterira po nizu kategorija ili trgovina te ih se svaka od njih pridružuje HTML *option* elementu na temelju identifikacijskog broja kao ključa. Kako je ta funkcionalnost implementirana u Reactovom dijelu aplikacije, može se vidjeti na Slici 4.2.5.

```jsx
<label>Odaberi trgovinu / proizvođača</label>
<select name="category_id"
    onChange={handleInput}
    value={productInput.shop_id}
    className="form-control">
<option> Odaberi </option>
    {
        selectedShops.map((shopItem) => {
            return (
                <option value={shopItem.id} key={shopItem.id}>
                    {shopItem.name}
                </option>
            )
        })
    }
</select>
```

Slika 4.2.5. Padajući izbornik s popisom obrta

Ostali atributi unose se kroz *input* elemente čija se vrijednost parametra *value* postavlja kao neka od varijabli unutar konstante *productInput*. Iznimka je fotografija proizvoda koja se unosi na način kako je prikazano na Slici 4.2.6. Tip unosa definira se kao *file*.

```
<div className="form-group mb-3">
    <label>Fotografija</label>
    <input type="file" name="photo" onChange={handleProductPhoto}/>
</div>
```

Slika 4.2.6. HTML odjeljak za unos fotografije

Upravo radi učitavanja datoteka treba se naznačiti da je atribut *encType* u HTML-ovom upitniku vrijednosti *multipart/form-data*. Kao što i samo ime govori, podatci će se tada poslati poslužitelju u više dijelova zbog veličine fotografije. Atributi upitnika mogu se vidjeti na slici 4.2.7.

```
<div className="card-body">
    <form encType="multipart/form-data" onSubmit={addProduct} id="add_product">
        <ul className="nav nav-tabs" id="myTab" role="tablist">···
        </ul>
        <div className="tab-content" id="myTabContent">···
        </div>

        <button type="submit" className="btn btn-primary px-4 float-end">
            Spremi
        </button>
    </form>
</div>
```

Slika 4.2.7. Atributni upitnika za dodavanje proizvoda

Slanje podataka kreće klikom na gumb za potvrdu čime se poziva funkcija za dodavanje proizvoda. Kako bi se sve vrijednosti atributa proizvoda spremile u zajednički objekt, instancira se objekt klase *FormData* što se postiže metodom *append()*. Opisani primjer može se pogledati na Slici 4.2.8.

```
const data = new FormData();
data.append('category_id', productInput.category_id);
data.append('name', productInput.name);
data.append('description', productInput.description);
data.append('price', productInput.price);
data.append('quantity', productInput.quantity);
data.append('active', productInput.active);
data.append('approved', productInput.approved);
data.append('photo', productPhoto.photo);
```

Slika 4.2.8. Uporaba objekta FormData

Slijedeći je korak poslati zahtjev poslužiteljskoj strani koristeći se *axiosom,* definirati tip HTML metode kao *POST* te upisati odgovarajuću rutu. Ako je proizvod uspješno dodan, vrijednost statusnog koda bit će 200. Tada će se vrijednosti svih atributa ponovo postaviti na prazan niz, odnosno upitnik će biti spreman za unos novog zapisa u bazu. Ako je pak vraćeni kod vrijednosti 422, znači da je zahtjev primljen i ispravno oblikovan, no formati pojedinih vrijednosti atributa nisu zadovoljili kriterije validacije. Tada se ispod odgovarajućih polja u sučelju ispisuju prigodne poruke kako bi korisnik znao u čemu je pogriješio.

Funkcija za dodavanje proizvoda u *ProductControlleru* isprva provjerava jesu li unesene vrijednosti za obvezna polja. Također, vrsta datoteke smije biti samo fotografija ili slika s nastavcima *.jpeg, .png* te *.jpg*. U slučaju nepravilnosti klijentu će se poslati poruke Laravelovog *Validatora*. Ako su pravila zadovoljena, slijedi priprema podataka za unos. Iz naziva proizvoda generirat će se vrijednost atributa *keyname*. Velika slova tada će se zamijeniti malim, a razmaci znakom spojnicom. Dijakritički će se znakovi također ukloniti i zamijeniti sličnim znakovima latinske abecede kako se može vidjeti na Slici 4.2.9. Taj će atribut poslužiti za kreiranje URL-a kada se klikne na proizvod u korisničkom sučelju.

```
$data['keyname'] = strtolower(Str::ascii(str_replace(' ', '-', $data['name'])));
```

Slika 4.2.9. Generiranje vrijednosti atributa keyname

Potom je potrebno spremiti učitanu fotografiju na poslužitelj. Metode *time()* i *getClientOriginalExtension()* rabe se za kreiranje imena fotografije. No, kako se ne bi izvršavale nad objektom koji ne postoji, prvo se provjerava sadrži li zahtjev datoteku. Ako je slika učitana, spremit

će se u mapu *public/photos/products*. Kako je ova funkcionalnost implementirana, može se vidjeti na Slici 4.2.10.

```php
if ($request->hasFile('photo')) {
    $file = $request->file('photo');
    $extension = $file->getClientOriginalExtension();
    $name = time() . '.' . $extension;

    $file->move('photos/products/', $name);
    $data['photo'] = 'photos/products/' . $name;
}
```

Slika 4.2.10. Provjera sadržaja zahtjeva na poslužitelju

Proizvod se zatim zapisuje u tablici podataka na način kako je prikazano na Slici 4.2.11.

id	keyname	name	description	active	approved	quantity	price	photo	category_id	shop_id
2	cikla	Cikla	Zdrava i ukusna.	1	1	9	20	photos/products /1662592084.jpg	9	2

Slika 4.2.11. Novi proizvod u tablici

Prikaz podataka na klijentskoj strani u administratorskom sučelju ostvaren je kroz datoteku Product.js koja sadrži ista polja kao datoteka za dodavanje novog proizvoda, osim što se metodom *GET* dohvaćaju podatci s poslužitelja u JSON obliku kao na Slici 4.2.12. te ih se potom prikazuje unutar HTML-ovih elemenata.

Slika 4.2.12. Poslužitelj vraća zahtjev u JSON formatu

No, dodavanje zapisa, kao i dohvat liste kategorija te obrta tijekom unosa podataka, funkcionalnost je koja je ostvariva samo ako je trenutni korisnik administrator. U tu svrhu pripadne rute u datoteci api.php definirane su unutar posebnog posrednika (eng. *middleware*). Kao što se može uočiti na Slici 4.2.13., osim što osoba mora biti prijavljena u aplikaciju, prolazi i kroz provjeru korisničke uloge.

```
//Protected routes
Route::middleware(['auth:sanctum', 'checkForAdmin'])->group(function () {
    Route::post('add-product', [ProductController::class, 'create']);
    Route::get('select-category', [CategoryController::class, 'select']);
    Route::get('select-shop', [ShopController::class, 'select']);
```

Slika 4.2.13. Zaštićene rute

Drugi parametar naveden unutar posrednika poziva funkciju *handle()* iz *AdminMiddlewarea*. Koristeći se metodama fasade *Auth*, najprije se provjerava je li osoba prijavljena. Ako nije, vraća se statusni kod 401 i prikladna poruka. Prijavljeni korisnik može imati ulogu administratora ili običnog korisnika poput kupca i obrtnika u čijem se slučaju odbija procesuirati zaprimljeni zahtjev te vraća statusni kod 403. Metodom *tokenCan('server:admin')* prikazanoj na Slici 4.2.14. provjerava se

vrijednost atributa *abilities* u tablici tokena. Ako je njegova vrijednost ["*"], prepoznaje se da osoba ima administratorske ovlasti te smije dodavati proizvode.

```php
public function handle(Request $request, Closure $next)
{
    if (Auth::check()) {
        if (auth()->user()->tokenCan('server:admin')) {
            return $next($request);
        } else {
            return response()->json([
                'internal_message' => 'User is not admin.'
            ], 403);
        }
    } else {
        return response()->json([
         'internal_message' => 'User unauthenticated.',
         'status' => 401,
        ]);
    }
}
```

Slika 4.2.14. Provjera korisničke uloge u posredniku

Kakve će ovlasti imati korisnikov token, odlučuje se prilikom prijave u aplikaciju. Na klijentskoj se strani unose adresa e-pošte i lozinka koje se zatim pošalju Laravelovom dijelu aplikacije. U *AuthControlleru* može se pronaći funkcija koja validira unesene podatke. Potom traži unesenu adresu u bazi podataka, a ako ona ne postoji, znači da se korisnik nikad nije registrirao. Metodom *check()* implementiranoj u fasadi *Hash* provjerit će se odgovara li unesena lozinka adresi iz HTTP zahtjeva. Ako je odgovor potvrdan, na temelju identifikacijskog broja uloge korisnika, pridružit će joj se token s odgovarajućim ovlastima kako prikazuje Slika 4.2.15.

```
if ($user->user_role_id == 1) {
    $userRole = 'admin';
    $token = $user->createToken('admintoken', ['server:admin'])
        ->plainTextToken;
} else {
    $userRole = '';
    $token = $user->createToken('usertoken', [''])
        ->plainTextToken;
}
```

Slika 4.2.15. Konfiguracija CORS-a

5. ZAKLJUČAK

Cilj ovoga završnog rada bio je razviti RESTful aplikaciju koja će podržavati usluge elektroničke trgovine te osigurati CRM (*Costumer Relationship Menagement*) funkcionalnosti kojima će se pratiti rad obrta, njihova ponuda te poslovanje. Osim toga, aplikacija mora dobro izgledati na svim veličinama ekrana. Svi zadatci definirani prije izrade rada su ostvareni. U budućnosti bi se projekt mogao proširiti na mobilnu aplikaciju koju će koristiti dostavljači kako bi mogli ažurirati stanje narudžbe. Obrtnicima bi pomoglo kada bi se uredio dio korisničkog sučelja u kojem će moći pratiti rad svoje trgovine kroz godine te dobiti uvid u karakteristike svojih kupaca poput dobi i spola.

Može se zaključiti da će, ako se želi kreirati moderna, stabilna i lako nadogradiva poslužiteljska strana aplikacije, Laravel zadovoljiti sve potrebe jednog klasičnog projekta za dodavanje, brisanje, dohvaćanje i uređivanje podataka u bazi. Također, olakšano je upravljanje objektima koji su međusobno povezani složenim vezama, stoga se programer može posvetiti nekim drugim bitnijim zadatcima koji će mu oduzeti više vremena, a ključni su za kvalitetan rad aplikacije. Metode implementirane u Laravelovim fasadama olakšavaju i ubrzavaju razne operacije nad objektima. One su zapravo prečica funkcija koje bi programer morao sam napisati pazeći pritom na pogreške koje se uvijek mogu lako potkrasti i time usporiti napredak te smanjiti kvalitetu konačnog proizvoda. Ono što se posebno ističe je velika zajednica Laravelovih programera na internetu koji neprestano rade na njegovom poboljšanju i voljni su nadopunjavati literaturu kako bi pomogli neiskusnim kolegama.

S obzirom da je klijentski dio aplikacije napisan u *Reactu*, njezino korisničko sučelje je responzivno, a komponente se jednostavno i brzo slažu u skladnu cjelinu koja će sigurno privući nove korisnike. Bootstrap je dodatno pridonio poboljšanju njezinog izgleda. Intuitivna struktura *Reactovih* datoteka znatno utječe na brzinu učenja ove JavaScript knjižnice. No, u slučaju da se dogode značajnije promjene u nekim paketima, neiskusnom programeru može predstavljati problem. Službena dokumentacija još uvijek nije zadovoljavajuća, a ne olakšava ni činjenica da na njegovom razvoju radi puno programera koji nemaju zajedničku normu. No, to ne znači da je *React* zato manje vrijedan učenja. Dapače, polako istiskuje Javu iz uporabe kada se radi o programiranju aplikacija za mobilne sustave. Osim toga, u svojim aplikacijama koriste ga velike i stabilne tvrtke, stoga će još dugo biti jedan od boljih izbora kada je riječ o kreiranju korisničkog sučelja. Ovaj je završni rad iznimno koristan za svakog programera koji želi ocijeniti svoje sposobnosti, vještine i znanja te ga svakako vrijedi nadograđivati testirajući pritom novo dostupne funkcionalnosti u Reactu i Laravelu.

LITERATURA

[1] Službena dokumentacija: „Installation“, s interneta, https://laravel.com/docs/8.x , 2. rujna 2022.

[2] Službena dokumentacija: „Getting Started“, s interneta, https://reactjs.org/docs/getting-started.html , 2. rujna 2022.

[3] Službena dokumentacija: „Laravel Sanctum“, s interneta, https://laravel.com/docs/8.x/sanctum , 4. rujna 2022.

[4] Službena dokumentacija: „What is XAMPP?“, s interneta, https://www.apachefriends.org/index.html , 27. kolovoza 2022.

[5] Službena dokumentacija: „About“, s interneta, https://www.phpmyadmin.net , 1. rujna 2022.

[6] Službena dokumentacija: „Download“, s interneta, https://getbootstrap.com/docs/5.1/getting-started/download/ , 9. rujna 2022.

[7] TutorialsPoint: „Entity Framework - Eager Loading“, s interneta, https://www.tutorialspoint.com/entity_framework/entity_framework_eager_loading.htm , 28. kolovoza 2022.

[8] Decode Web: „Beginners guide to facades in laravel“, s interneta, https://www.codementor.io/@decodeweb/beginners-guide-to-facades-in-laravel-1457p3h4nd , 2. ožujka 2020.

[9] Službena dokumentacija: „Getting Started“, s interneta, https://getcomposer.org/doc/00-intro.md , 2. rujna 2022.

[10] Službena dokumentacija: „About *npm*“, s interneta, https://www.npmjs.com/ , 2. rujna 2022.

[11] Službena dokumentacija: „Download“, s interneta, https://www.postman.com/ , 5. rujna 2022.

[12] Službena dokumentacija: „Installation“, s interneta, https://sweetalert.js.org/docs/ , 9. rujna 2022.

POPIS KRATICA

1. WWW - World Wide Web
2. PHP - Hypertext Preprocessor
3. HTML - HyperText Markup Language
4. MVC - *model-view-controller*
5. ORM - objektno relacijsko mapiranje
6. OAuth – otvorena autorizacija
7. XSS - cross-site scripting
8. VDOM - Virtual Document Object Model
9. DOM - Document Object Model
10. API - Application Programming Interface
11. ER – Entity Relationship
12. CORS - Cross-Origin Resource Sharing
13. CSRF - Cross-Site Request Forgery
14. CRM - Costumer Relationship Menagement

SAŽETAK

U ovome završnom radu prezentirana je RESTful aplikacija za pružanje usluga elektroničke trgovine virtualne tržnice. Korisničko je sučelje izrađeno u Reactu, a poslužiteljska strana u Laravelu. U uvodu se navode prednosti internetske kupovine i svrha projekta. Zatim su detaljno objašnjene glavne karakteristike korištenih tehnologija te su opisane sve implementirane funkcionalnosti. Na razini koda objašnjeni su registracija korisnika i dodavanje novog proizvoda. U zaključku je opisano iskustvo programiranja u Laravelu i Reactu te su navedena moguća poboljšanja aplikacije.

Ključne riječi: Laravel, React, Sanctum, RESTful, web aplikacija, virtualna tržnica, e-trgovina

ABSTRACT

In this paper the RESTful e-commerce application for virtual marketplace is presented. The fronted has been made with React, while the backend has been made with Laravel. The introduction describes advantages of e-commerce and the purpose of this project. Then, the main characteristics of the used technologies are thoroughly explained and implemented functionalities are described. User registration and new product registration are explained on the code level. In the conclusion, programming experience in Laravel and React is discussed, as are possible improvements.

Keywords: Laravel, React, Sanctum, RESTful, web application, virtual marketplace, e-commerce