Web aplikacija za online tržnicu

Barišić, Petar

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:562831

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-15**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

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

Sveučilišni studij

Web aplikacija za online tržnicu

Diplomski rad

Petar Barišić

Osijek, 2023. godina.

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

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 18.09.2023.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Petar Barišić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1183R, 07.10.2021.
OIB studenta:	99038070496
Mentor:	doc. dr. sc. Krešimir Romić
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 1:	doc. dr. sc. Krešimir Romić
Član Povjerenstva 2:	doc. dr. sc. Hrvoje Leventić
Naslov diplomskog rada:	Web aplikacija za online tržnicu
Znanstvena grana diplomskog rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak diplomskog rada:	Potrebno je izraditi web platformu na kojoj će mali prodavači i OPG-ovi moći kreirati svoje profile i u sklopu istih objavljivati svoje proizvode za prodaju. Kupcima je potrebno omogućiti pretragu i pregled oglašenih proizvoda te narudžbu istih. U bazi podataka čuvati podatke o prodavačima, proizvodima (opis, stanje, dostupnost i sl.) i narudžbama. Implementirati prijavu u sustav s barem dvije uloge - prodavač i kupac. Za izradu web aplikacije koristiti novije razvojne okvire poput React.js (za frontend) i Spring Boot (za backend). Tema rezervirana za: Petar Barišić
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	18.09.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 11.10.2023.

Ime i prezime studenta:	Petar Barišić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1183R, 07.10.2021.
Turnitin podudaranje [%]:	3

Ovom izjavom izjavljujem da je rad pod nazivom**: Web aplikacija za online tržnicu**

izrađen pod vodstvom mentora doc. dr. sc. Krešimir Romić

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 studenta:

SADRŽAJ

1. UVOD 1

1.1. Zadatak diplomskog rada 1

2. PREGLED PODRUČJA 3

2.1. Plodovi.hr 3

2.2. ePlac 4

2.3. Online Tržnice grada Zagreba 4

2.4. Domaća web tržnica 5

2.5. Usporedba postojećih rješenja 6

3. ARHITEKTURA APLIKACIJE 7

3.1. Korištene tehnologije i alati 7

3.1.1. IntelliJ IDEA 8

3.1.2. React, TypeScript i Tailwind CSS 8

3.1.3. Java i Spring Boot 9

3.1.4. H2 i Liquibase 10

3.2. Podaci s kojima aplikacija radi 11

3.3. Dizajn korisničkog sučelja 12

4. IMPLEMENTACIJA 14

4.1. Baza podataka 14

4.2. Backend 17

4.2.1. Model 17

4.2.2. Repozitorij 21

4.2.3. Servis 22

4.2.4. Kontroler 23

4.2.5. Security 25

4.3. Frontend 26

4.3.1. Tipovi podataka i API pozivi 27

4.3.2. Komponente 28

4.3.3. Stranice 30

4.3.4. Rute 31

5. TESTIRANJE RADA APLIKACIJE 33

5.1. Backend testiranje..... 33

5.2. Frontend testiranje 34

6. ZAKLJUČAK..... 46

LITERATURA 47

SAŽETAK..... 48

ABSTRACT 49

1. UVOD

U današnjem digitalnom dobu, tehnološki napredak i sveprisutnost interneta preobrazili su način na koji ljudi reklamiraju i pregledavaju proizvode. Tradicionalni modeli trgovine postaju zastarjeli, a elektronička trgovina postaje neizostavan dio naše svakodnevnice. U skladu s tim, razvoj mrežne aplikacije za tržnicu postaje ključan kako bi se zadovoljile potrebe korisnika i stvorile nove mogućnosti za trgovce. Jedan od ključnih razloga za popularnost mrežnih tržnica je jednostavnost i praktičnost koje pružaju korisnicima u udobnosti vlastitog doma. Nažalost ovakve mrežne aplikacije su još u razvoju u Hrvatskoj i nisu toliko popularne. Stoga je jedan od glavnih ciljeva ovog rada potaknuti napredak spomenutih aplikacija i na našim regijama.

Kroz ovaj diplomski rad opisana je arhitektura aplikacije, implementacija funkcionalnosti i pružiti detaljan pregled tehnologija i alata koji su korišteni. Očekuje se da će rezultat ovog rada biti funkcionalna i skalabilna mrežna aplikacija koja će pružiti korisnicima intuitivno iskustvo pregleda i rezerviranje proizvoda putem interneta.

1.1. Zadatak diplomskog rada

Zadatak ovog diplomskog rada je razviti mrežnu aplikaciju za tržnicu koja će omogućiti korisnicima da pregledavaju i rezerviraju proizvode putem interneta. Korisnici će i dalje kupovati i prodavati proizvode uživo. Razlog tome je to što kada se rezervira i dođe uživo po proizvod, korisnik ga može pregledati, dotaknuti i time provjeriti njegovu kvalitetu prije nego što donese odluku o kupnji. Pomoću aplikacije korisnik će unaprijed znati odgovore na ključna pitanja, kao što su: što se prodaje, tko prodaje, gdje se prodaje i za koliko se prodaje. Mrežna aplikacija za tržnicu omogućit će korisnicima registraciju i prijavu na platformu, pregled i pretraživanje proizvoda i trgovaca, izvršavanje narudžbi i praćenje statusa narudžbe. Također će pružati funkcionalnosti za trgovce, kao što su upravljanje inventarom, postavljanje cijena i praćenje narudžbi. Razvoj ove aplikacije zahtijeva odgovarajuću arhitekturu i tehnologije koje će omogućiti skalabilnost, optimiziranost i jednostavnost korištenja.

Java, snažan i robustan programski jezik s bogatim ekosustavom alata, odabran je za *backend* implementaciju radi pouzdanosti i efikasnosti. Za spremanje podataka će se koristiti H2 baza podataka, koja je brza te jednostavna za korištenje i za prenošenje na produkcijsko okruženje. Uz H2, Liquibase će omogućiti upravljanje verzijama baze podataka i omogućiti lakše ažuriranje strukture podataka. Na *frontendu* će se koristiti React.js, popularan JavaScript okvir za izgradnju

korisničkog sučelja, u kombinaciji s TypeScriptom, koji pruža statičku tipizaciju i poboljšava sigurnost koda i Tailwind CSS, modernim CSS okvirom koji omogućuje brzo i jednostavno stiliziranje modernog i intuitivnog korisničkog sučelja.

2. PREGLED PODRUČJA

Online tržnice revolucionirale su način na koji ljudi kupuju i prodaju proizvode diljem svijeta. U posljednjem desetljeću, rapidan razvoj tehnologije i internetske povezanosti omogućio je stvaranje globalnih platformi koje olakšavaju online trgovinu. Namirnice su postale dostupne na dohvat prsta putem interneta. U nastavku će se opisati online tržnice koje postoje u Hrvatskoj, fokusirajući se na njihove mogućnosti i ključne aspekte koje pružaju.

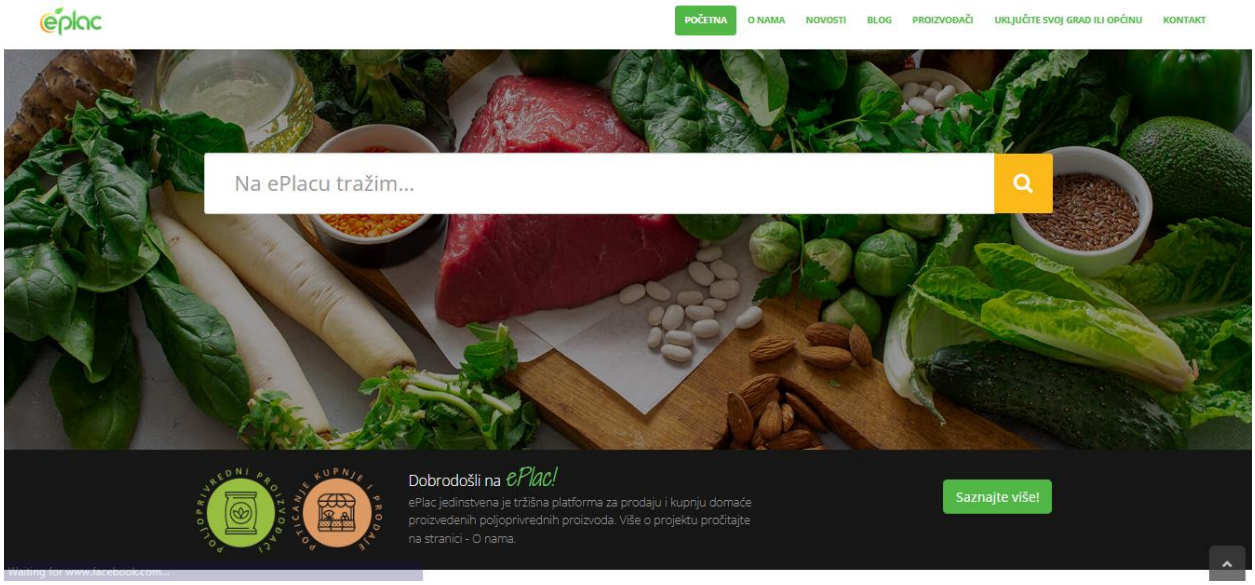
2.1. Plodovi.hr

Plodovi.hr je online tržnica između kupca i OPG-ova na području Osijeka i okolice stvorena od strane Reducos j.d.o.o (Slika 2.1.). Mrežna stranica omogućuje bogat pregled proizvoda koji nude lokalni OPG-ovci kako bi potaknuli ljude da kupuju domaće te olakšati im cijeli taj proces naručivanjem u par klikova i dostavom na kućni prag [1].

Slika 2.1. Početna stranica mrežne aplikacije *Plodovi.hr*

2.2. ePlac

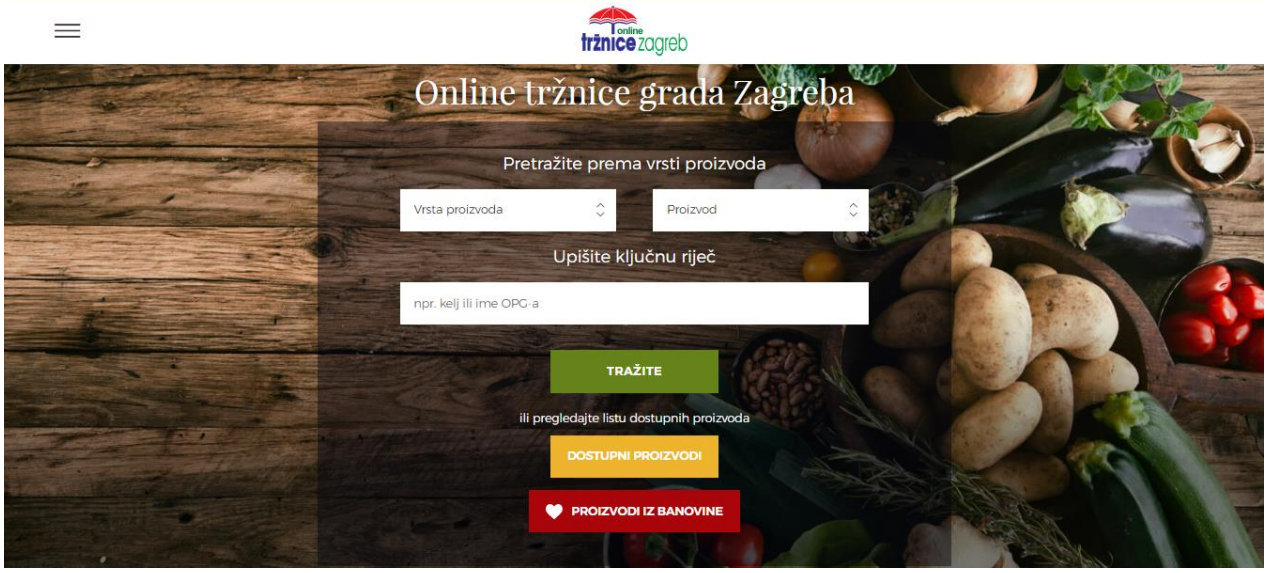
ePlac je mrežna i mobilna tržišna platforma namijenjena za prodaju i kupnju domaćih poljoprivrednih proizvoda. Platformu su 2017. godine razvili PlanITJob u suradnji sa tvrtkom Minea d.o.o. (Slika 2.2.). Potporu je pružio grad Bjelovar koji je i sufinancirao projekt. Cilj projekta je poticanje proizvodnje i prodaje lokalnih proizvođača te osiguravanje kontrolirane kupnje pravih domaćih proizvoda [2].



Slika 2.2. Početna stranica mrežne aplikacije *ePlac*

2.3. Online Tržnice grada Zagreba

Online Tržnice grada Zagreba je interaktivna mrežna aplikacija koja je nastala kao odgovor na izbijanje koronavirusa, koji je prouzročio zatvaranje tržnica u Zagrebu (Slika 2.3.). Međutim, to je istovremeno potaknulo Grad Zagreb da stvori online tržnicu na kojoj će domaći proizvođači s područja grada Zagreba moći ponuditi svoje proizvode. U suradnji s Uredom gradonačelnika, Gradskim uredom za poljoprivredu i šumarstvo te Podružnicom Tržnice Zagreb, razvijena je jedinstvena platforma koja omogućuje lokalnim proizvođačima i kupcima da se povežu kao da su na samoj tržnici [3].



Slika 2.3. Početna stranica mrežne aplikacije *Online Tržnice grada Zagreba*

2.4. Domaća web tržnica

Domaća web tržnica je mrežna aplikacija koja je nastala kao rezultat provedbe projekta pod nazivom "Organizacija sustava izravne prodaje poljoprivrednih proizvoda korištenjem internet tehnologije" (skraćenica: OSIPPPIT) (Slika 2.4.). Ovaj projekt je financiran unutar okvira Operativnog programa IPA Slovenija-Hrvatska za razdoblje od 2007. do 2013. godine. Tržnica obuhvaća promoviranje i prodaju poljoprivrednih proizvoda na području Istre [4].

Slika 2.4. Početna stranica mrežne aplikacije *Domaća web tržnica*

2.5. Usporedba postojećih rješenja

Spomenute mrežne aplikacije su bogate mogućnostima i uslugama. Svaka nudi usluge registracije i prijave te traženje i pregled proizvoda i proizvođača (Tablica 2.1.).

Tablica 2.1. Prikaz funkcionalnosti pojedinih aplikacija

Plodovi.hr	**ePlac**	**Online Tržnice grada Zagreba**	**Domaća web tržnica**
Registracija	Registracija	Registracija	Registracija
Tražilica	Tražilica	Tražilica	Tražilica
Pregled proizvoda	Pregled proizvoda	Pregled proizvoda	Pregled proizvoda
Pregled proizvođača	Pregled proizvođača	Pregled proizvođača	Pregled proizvođača
Filter proizvoda	Uključivanje svojeg grada ili općine	„Plavi ceker“	Mogućnost narudžbe
Košarica	Košarica	Mogućnost narudžbe	Interaktivna karta
Recenzije	Blog		Aktualnosti
Novosti	Novosti		Stranica na 3 jezika
Dostava			

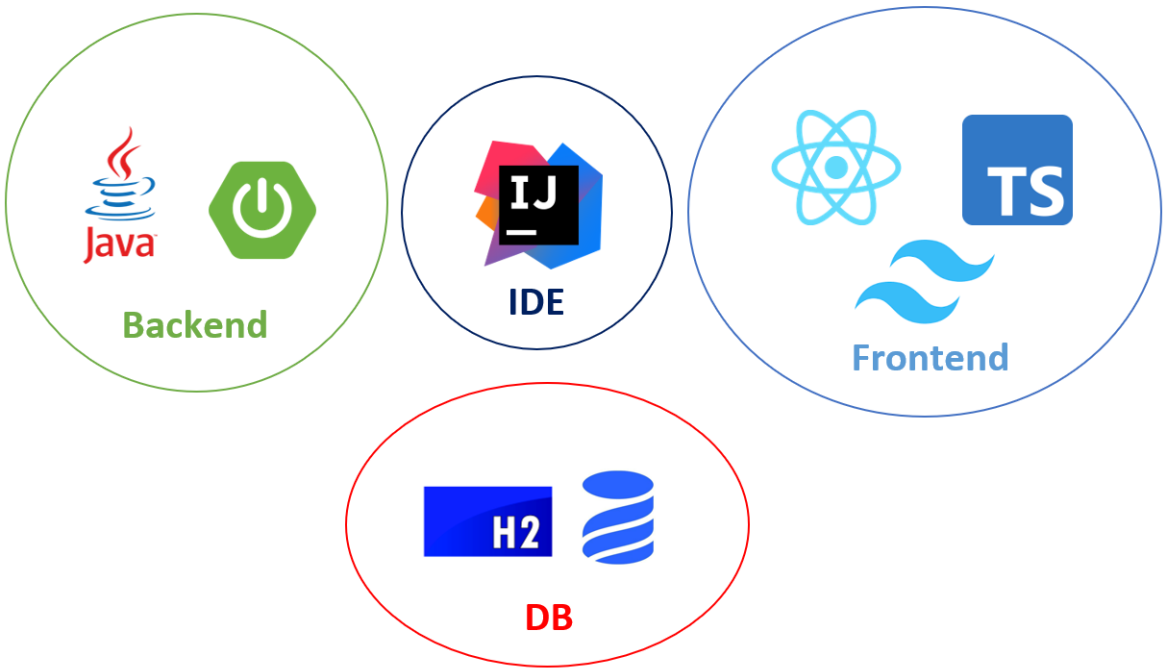
No aplikacije pružaju i ono nešto po čemu se ističu. *Plodovi.hr* nudi mogućnost recenzije, koja pomaže korisnicima pri odabiru namirnice, novosti, koji pružaju najnovije informacije o različitim sadržajima vezanim za proizvode te dostavu proizvoda u krugu od 25 km od Osijeka. Tržišna platforma *ePlac*, kao posebnu mogućnost, nudi usluge uključivanja vlastitog grada ili općine čime pruža univerzalno sučelje za cijelu Hrvatsku. Također sadrži sekciju blog, gdje korisnici objavljuju članke o receptima i jelima koja su pravili. *Online Tržnice grada Zagreba* pruža veliki spektar vrsta proizvoda koji su trenutno dostupni kao što je: voće, povrće, mliječni proizvodi, suhomesnati proizvodi i sl. Isto tako, određeni proizvodi mogu imat oznaku kvalitete „Plavi ceker“. Ona pokazuje kupcima da proizvod pripada u skupinu visokokvalitetnih prehrambenih proizvoda čija je kvaliteta više razine od zakonski propisane. Mrežna aplikacija *Domaća web tržnica* omogućuje korisnicima pristup zanimljivoj interaktivnoj karti koja prikazuje različite lokacije proizvoda na području Istre. Prikaz lokacija ovisi o proizvodima koje korisnik odabere. Pruža i uslugu kalendara događaja, gdje korisnici mogu saznati datume održavanja radionica i sajmova vezano za poljoprivredu te uslugu prijevoda stranice na tri jezika: hrvatski, engleski i slovenski.

3. ARHITEKTURA APLIKACIJE

Dobra arhitektura mrežne aplikacije predstavlja ključni temelj za njezinu funkcionalnost i uspjeh. Ona utječe na sve aspekte aplikacije, uključujući korištene tehnologije, način rukovanja podacima i dizajn korisničkog sučelja. Arhitektura treba omogućiti integraciju *frontenda* s *backendom* na način koji će omogućiti brze i nesmetane interakcije. To uključuje brzo učitavanje stranica i intuitivnu navigaciju. Skalabilnost, kako aplikacije rastu, također je ključna komponenta dobre arhitekture. Mrežna aplikacija mora biti sposobna nositi se s porastom broja korisnika i povećanjem opterećenja. Pravilna arhitektura omogućava laku skalabilnost dodavanjem novih resursa. U nastavku slijedi detaljniji opis korištenih tehnologija i alata, načina rukovanja podacima i dizajna korisničkog sučelja.

3.1. Korištene tehnologije i alati

Pri izradi mrežnih aplikacija, odabir odgovarajućih tehnologija i alata igra ključnu ulogu u postizanju uspješnog rezultata. Napredni integrirani razvojni okviri, alati za automatizaciju, upravljanje kodom i verzijama, kao i dobro dokumentirane biblioteke, mogu značajno ubrzati razvojni proces. U nastavku slijedi opis odabranih tehnologija i alata koje će se koristiti za izradu online tržnice (Slika 3.1.).



Slika 3.1. Prikaz korištenih tehnologija

3.1.1. IntelliJ IDEA

IntelliJ IDEA je integrirano razvojno okruženje (*engl. IDE*) koje se koristi za programiranje u različitim jezicima, uključujući Java, Kotlin, Groovy, Scala, JavaScript, TypeScript i mnogi drugi. Razvijen od strane kompanije JetBrains, IntelliJ IDEA je jedan od najpopularnijih i najmoćnijih IDE-ova koji se koristi u razvoju programske podrške.

Za razvoj mrežnih aplikacija, IDEA pruža napredne alate koji podržavaju HTML, CSS i JavaScript te integraciju s popularnim okvirima kao što su Angular, React i Vue.js. Također omogućava brzo i jednostavno refaktoriranje koda, što uključuje preimenovanje i izvlačenje metoda te preuređivanje koda radi poboljšanja čitljivosti i efikasnosti. Njegov moćni interaktivni *debugger* omogućava razvojnim inženjerima da detaljno prate izvršavanje svog koda, otkrivajući i ispravljajući greške i probleme u programu. IntelliJ IDEA je integriran s popularnim verzioniranim kontrolnim sustavima kao što je Git, što olakšava verzioniranje programskoga koda. Isto tako ima podršku za testiranje programskog koda, omogućavajući razvojnim inženjerima da napišu, pokreću i analiziraju rezultate testova direktno iz okoline te podržava veliki broj dodataka (engl. *plugins*) za različite programske jezike, alate i okvire [5].

3.1.2. React, TypeScript i Tailwind CSS

React je popularna JavaScript biblioteka za izgradnju korisničkih sučelja. Razvijen od strane Facebooka, osmišljen je kako bi olakšao stvaranje efikasnih, modularnih i interaktivnih mrežnih aplikacija. React se zasniva oko koncepta komponenata, dijelova koda koji predstavljaju određene dijelove korisničkog sučelja.

React koristi JSX (JavaScript XML) sintaksu koja omogućuje razvojnim inženjerima pisanje koda sličnog HTML-u, ali u JavaScriptu. Ova kombinacija omogućuje jednostavno spajanje logike i prikaza u jedinstvenom okruženju. U Reactu postoje neki ključni koncepti koje treba spomenuti. Komponente su temeljne jedinice u Reactu i predstavljaju nezavisne dijelove korisničkog sučelja. Mogu se sastojati od drugih komponenti, a unutar Reacta imaju stanje (engl. *state*) i svojstva (engl. *props*) kojima se upravlja [6]. React koristi virtualni DOM (engl. *Document Object Model*), koji je apstraktna reprezentacija stvarnog DOM-a. Kada se stanje komponente promijeni, React generira novi virtualni DOM i uspoređuje ga s prethodnim stanjem. Samo se promjene primjenjuju na stvarni DOM, što je efikasnije od ručnog ažuriranja cijelog DOM-a [7]. React također podržava i jednosmjerni protok podataka, što znači da se podaci prenose kroz hijerarhiju komponenata od roditelja prema djeci. Ovo olakšava praćenje i održavanje stanja aplikacije [8].

TypeScript, s druge strane, je programski jezik koji proširuje standardni JavaScript. Razvijen od strane Microsofta, TypeScript donosi statičko tipiziranje u JavaScript. Time omogućuje razvojnim inženjerima da deklariraju tipove varijabli, parametara funkcija i povratnih vrijednosti. Statičko tipiziranje također omogućuje ranu detekciju grešaka i pruža bolje razumijevanje koda [9].

Tailwind CSS je moderni CSS okvir koji omogućuje brzo i efikasno razvijanje dizajna suvremenih mrežnih aplikacija. Ističe se svojim pristupom "utility-first" stiliziranju, što znači da se koriste mnoge pojedinačne CSS klase kako bi se brzo primijenili stilovi nad elementima. Jedna od glavnih prednosti Tailwind CSS-a je njegova fleksibilnost. Umjesto da se oslanja na unaprijed definirane komponente, Tailwind CSS pruža široku paletu korisnih CSS klasa koje se mogu koristiti za oblikovanje elemenata prema potrebi. To omogućuje iznimnu prilagodljivost i preciznu kontrolu nad stilizacijom elemenata, bez potrebe za pisanjem prilagođenih CSS stilova. Još jedna značajka Tailwind CSS-a je njegova lakoća korištenja. Zbog obilja dostupnih klasa i intuitivnog nazivlja, razvojnim inženjerima omogućuje brzo i efikasno pisanje CSS-a. Osim toga, Tailwind CSS nudi i funkcionalnosti poput responzivnog dizajna, animacija, sjena i mnogo drugih stilskih mogućnosti koje olakšavaju izgradnju modernih i privlačnih korisničkih sučelja.

3.1.3. Java i Spring Boot

Java je popularan programski jezik koji se koristi za razvoj širokog spektra aplikacija, uključujući mrežne, mobilne, računalne i poslužiteljske aplikacije. Java je objektno orijentiran jezik, što znači da je usmjeren na organiziranje koda u objekte koji međusobno komuniciraju. Dizajniran je da bude platformski neovisan, što znači da se Java programi mogu izvršavati na različitim operativnim sustavima. Ovo se postiže upotrebom Java virtualne mašine (engl. *Java Virtaul Machine*) koja prevodi kod u izvršni kod specifičan za određenu platformu. Također, Java ima strogu sintaksu koja zahtijeva pravilno formatiranje i upotrebu elemenata poput zagrada i točke sa zarezom te koristi statičko tipiziranje, što znači da se tipovi varijabli moraju deklarirati prije korištenja. Podržava objektno orijentirano programiranje (OOP) i uključuje koncepte kao što su klase, objekti, nasljeđivanje, enkapsulacija i polimorfizam. OOP omogućava organizaciju koda na logičan način, olakšava ponovno korištenje i održavanje [11].

Spring Boot je popularan radni okvir (engl. *framework*) za razvoj Java aplikacija. Razvijen od tvrtke VMware, Spring Boot je dio šireg Spring ekosustava koji olakšava razvoj Java aplikacija. Spring Boot primjenjuje princip brzog pokretanja, koji automatski konfigurira mnoge dijelove aplikacije. To uključuje konfiguraciju baze podataka, upravljanje ovisnostima i postavljanje

mrežnog servera. Ovo smanjuje potrebu za ručnom konfiguracijom i omogućava razvojnim inženjerima da se usredotoče na razvoj poslovnih funkcionalnosti. Spring Boot dolazi s ugrađenim servlet kontejnerom, kao što su Tomcat ili Jetty, što pojednostavljuje pokretanje i testiranje mrežnih aplikacija bez potrebe za vanjskim serverom. Također ima sposobnost automatske konfiguracije temeljene na konvencijama i principu "pronalazi i konfiguriraj". Ovo olakšava razvoj aplikacija jer smanjuje potrebu za ručnom konfiguracijom. Spring Boot koristi Maven ili Gradle za upravljanje ovisnostima projekta. Time omogućava jednostavno dodavanje, uklanjanje i upravljanje bibliotekama i modulima koji su potrebni za razvoj aplikacije [12].

3.1.4. H2 i Liquibase

H2 je brza, lagana i otvorena baza podataka napisana u Javi. H2 se često koristi za razvoj i testiranje aplikacija, posebno u okruženjima koja zahtijevaju lokalnu ili privremenu bazu podataka.

H2 se može pokrenuti unutar samog aplikacijskog okruženja, bez potrebe za posebnim serverom ili zasebnom instalacijom. Ovo pojednostavljuje razvoj i testiranje aplikacija. Poznata je po svojoj visokoj performansi i malom utrošku resursa. Baza podataka se brzo pokreće i omogućuje brz pristup podacima. Također je jednostavna za upotrebu i konfiguraciju. H2 podržava različite načine pohrane podataka. To uključuje memorijski način, gdje se podaci pohranjuju samo u memoriju, lokalni način, gdje se podaci pohranjuju na disku, te klijent-poslužitelj način, gdje se podaci pohranjuju na poslužitelju. Ova fleksibilnost omogućuje prilagodbu baze podataka prema specifičnim potrebama aplikacije [13].

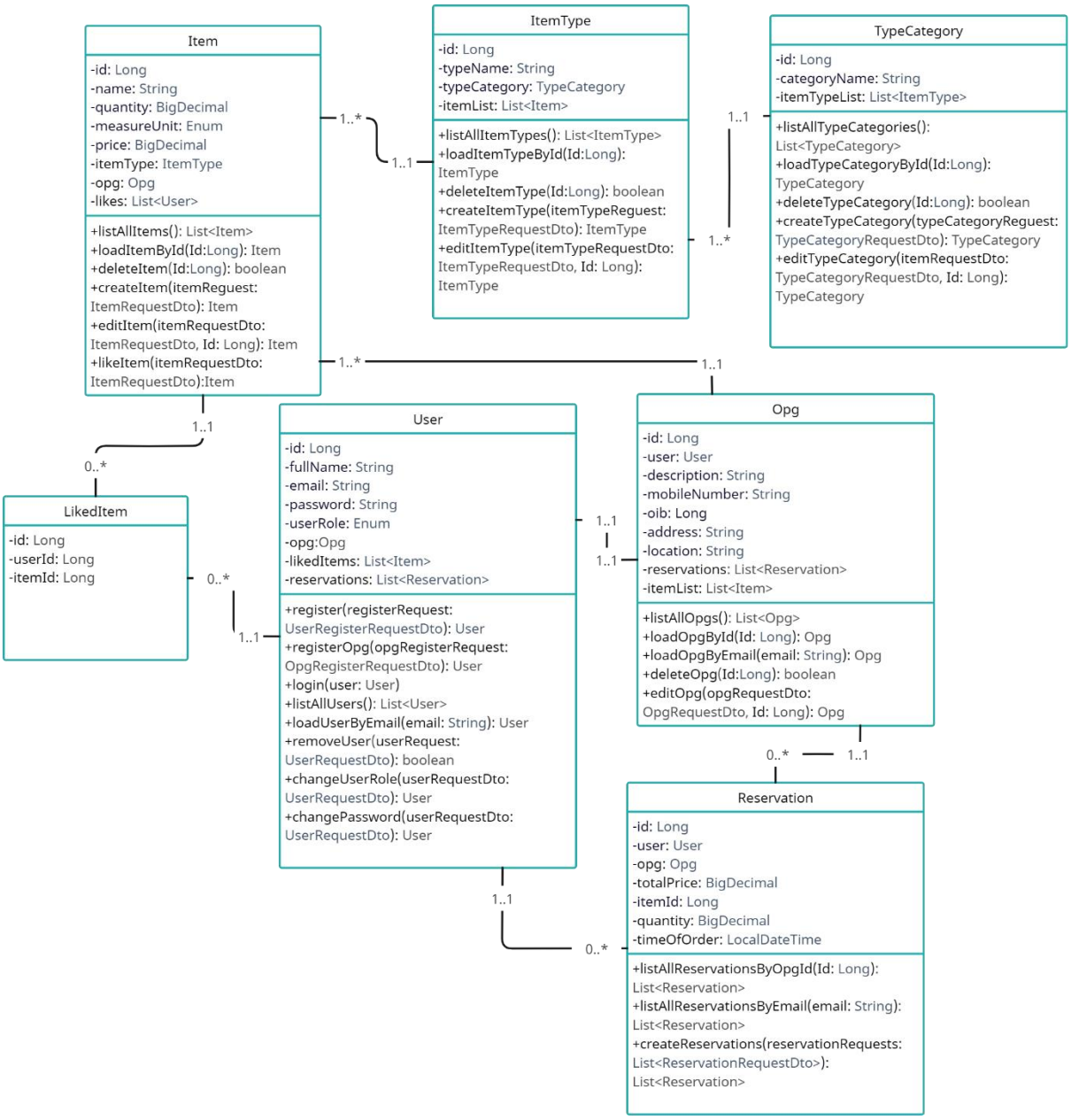
Liquibase je alat za upravljanje promjenama u strukturi baze podataka. Koristi se za verzioniranje baze podataka i automatsko izvršavanje promjena u strukturi podataka tijekom razvoja i održavanja aplikacija.

Liquibase omogućuje verzioniranje strukture baze podataka putem „changeseta". Svaka promjena u strukturi podataka, kao što su dodavanje tablica ili izmjene stupaca, definirana je kao zaseban „changeset“ s jedinstvenim identifikatorom. Ovo olakšava praćenje i upravljanje promjenama tijekom vremena. Platformski je neovisan alat koji podržava različite vrste baza podataka, uključujući MySQL, Oracle, PostgreSQL, H2 i mnoge druge. To znači da se promjene mogu konzistentno primjenjivati bez obzira na odabranu bazu podataka. Liquibase automatski izvršava promjene u strukturi baze podataka na temelju definiranih „changeseta“. Promjene se mogu primijeniti pri pokretanju aplikacije ili u određenom trenutku tijekom životnog ciklusa aplikacije, čime olakšava automatizaciju upravljanja promjenama u bazi podataka [14].

Kombinacija H2 baze podataka i Liquibase alata pruža programerima moćan alat za upravljanje strukturom podataka i promjenama u aplikacijama, olakšavajući razvoj, testiranje i održavanje baze podataka.

3.2. Podaci s kojima aplikacija radi

Pomoću klasnog dijagrama će se na jednostavni način prikazati klase koje se koriste u mrežnoj aplikaciji te njihove međusobne ovisnosti (Slika 3.2.). Dijagram je izrađen u aplikaciji Creately.

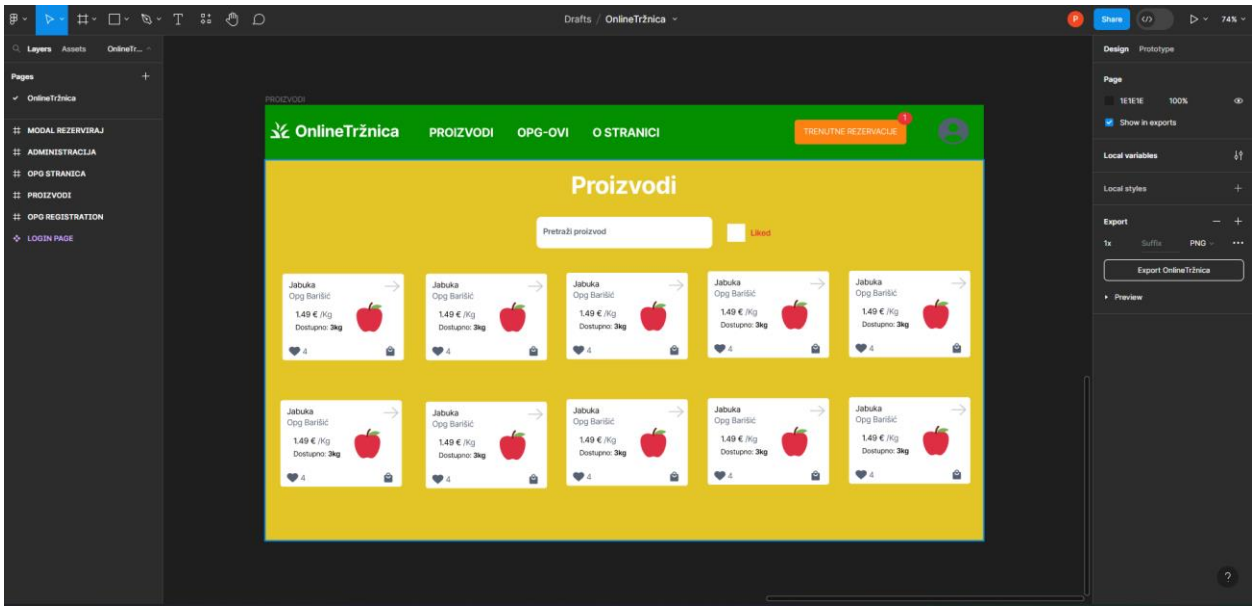


Slika 3.2. Klasni dijagram mrežne aplikacije *OnlineTržnica*

Klase *Item*, *User* i *Opg* predstavljaju glavne podatke aplikacije. *Item* predstavlja proizvod koji *Opg* pruža korisniku (*User*). Proizvod sadrži tip proizvoda (*ItemType*) koji predstavljaju objekte kao što su: jabuka, kruška, kukuruz i sl. Ti proizvodi pripadaju kategoriji (*ItemCategory*) kao što su: voće, povrće, mesni proizvodi i sl. Korisnik može imati tri uloge (*UserRole*): administrator, korisnik i prodavač. Glavne mogućnosti korisnika su odabiranje proizvoda koji im se sviđaju te rezervacija samog proizvoda. Administrator može upravljati korisnicima i prodavačima te prodavač uređuje proizvode koje pruža za rezervaciju.

3.3. Dizajn korisničkog sučelja

Za dizajn korisničkog sučelja korišten je alat Figma. Paleta boja mrežne aplikacije će najviše predstavljati zelena, žuta, narančasta i crvena. Razlog tome je njihova uska povezanost s voćem i povrćem koji čini većinu svake tržnice. Zelena simbolizira svježinu i prirodu dok žuta sunce i svijetlost. Narančasta predstavlja naranču, mandarinu ili breskvu, a crvena jagodu, jabuku ili rajčicu.



Slika 3.3. Prikaz dizajna stranice *Proizvodi* u alatu Figma

Slika 3.3. prikazuje kombinaciju ranije spomenutih boja u smislenoj cjelini. Također se vidi i prisutnost bijele boje koja se lako uočava. Ona će se koristiti najviše kao boja teksta na stranicama te kao boja pozadine kartice proizvoda koja predstavlja glavnu komponentu aplikacije (Slika 3.4.).

Slika 3.4 Prikaz dizajna kartice proizvoda

4. IMPLEMENTACIJA

Implementacija koda predstavlja ključan korak u procesu razvoja mrežne aplikacije. Nakon što su odabrani tehnologije i alati, odlučeno je s kojim podacima će se raditi i dizajnirano korisničko sučelje, sljedeće dolazi njihovo povezivanje u funkcionalan programski kod.

4.1. Baza podataka

Sve povezano s bazom podataka se nalazi u direktoriju *resources/db*. Svojstva koja će baza podataka imati su definirana u datoteci *appliacation.properties* (Slika 4.1.).

```
spring.jpa.show-sql=false
spring.jpa.hibernate.ddl-auto=none
spring.jpa.generate-ddl=false
spring.jpa.properties.hibernate.format_sql=false
spring.datasource.url=jdbc:h2:mem:onlineTrznica
spring.datasource.username=admin
spring.datasource.password=admin
spring.h2.console.enabled=true

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

server.servlet.contextPath=/api
```

Slika 4.1. Prikaz programskog koda u datoteci *application.properties*

- *spring.jpa.show-sql=false:* Ova postavka onemogućuje ispisivanje SQL upita u konzoli. Ako je postavljeno na *true*, aplikacija će ispisivati SQL upite koje izvršava prema bazi podataka u konzoli. Ovo je korisno za praćenje izvršavanja koda, ali se obično isključuje zbog sigurnosnih razloga.
- *spring.jpa.hibernate.ddl-auto=none*: Definira se način upravljanja tablicama baze podataka. Postavljanjem na *none*, Hibernate neće automatski kreirati ili mijenjati tablice u bazi podataka.
- *spring.jpa.generate-ddl=false:* Onemogućuje se Hibernateu automatski generiranje SQL skripte za stvaranje tablica na temelju entiteta aplikacije. Koristit će se prilagođene tablice pa je automatsko generiranje nepotrebno.
- *spring.jpa.properties.hibernate.format_sql=false*: Onemogućuje se formatiranje SQL upita prije nego što se ispišu u konzoli.
- *spring.datasource.url=jdbc:h2:mem:onlineTrznica:* Definira URL vezu prema bazi podataka. Koristi se H2 baza podataka koja će biti smještena u memoriji i nazvana *onlineTrznica*.

- *spring.datasource.username=admin* i *spring.datasource.password=admin*: Korisničko ime i lozinka koji će se koristiti za pristup bazi podataka.
- *spring.h2.console.enabled=true:* Omogućuje H2 konzolu, koja je korisna za pregled i upravljanje H2 baze podataka putem mrežnog sučelja.
- *spring.liquibase.change-log=classpath:db/test-changelog.xml:* Postavlja se putanja glavne *changelog* datoteke koja sadrži informacije o strukturi i inicijalnim podacima baze podataka.
- *server.servlet.contextPath=/api:* Predstavlja kontekst putanju (engl. *context path*) mrežne aplikacije. To znači da će svi API (engl. *Application Programming Interface - API*) pozivi započinjati s */api* u URL-u. Na primjer, ako je aplikacija dostupna na http://localhost:8080, API pozivi će biti dostupni na http://localhost:8080/api.

Za kreiranje i upravljanje tablica u bazi podataka je korišten Liquibase i XML (engl. *Extensible Markup Language*) koji je napisan u datoteci *changelog-verzija.xml*. Unutar <*changeSet*> elementa se upisuju promjene vezane za jedan skup podataka. Pomoću <*createTable*> elementa se kreira tablica s odabranim nazivom, a unutar elementa se dodaju novi elementi koji predstavljaju jedan red tablice, <*column*>. Red sadrži svojstva kao što su naziv i tip podatka, a može sadržavati i element <*constraints*>, koji predstavlja dodavanje ograničenja kao što su primarni ključ ili *nullable* mogućnost (Slika 4.2.).

```
8   <changeSet id="202308121424" author="pbarisic">
9       <createTable tableName="users">
10          <column name="id" type="BIGINT" autoIncrement="true">
11              <constraints primaryKey="true" unique="true"/>
12          </column>
13          <column name="full_name" type="VARCHAR(255)">
14              <constraints nullable="false"/>
15          </column>
16          <column name="password" type="VARCHAR(60)">
17              <constraints nullable="false"/>
18          </column>
19          <column name="email" type="VARCHAR(60)">
20              <constraints nullable="false"/>
21          </column>
22          <column name="role" type="enum('ADMIN','USER','SELLER')">
23              <constraints nullable="false"/>
24          </column>
25      </createTable>
26  </changeSet>
```

Slika 4.2. Kreiranje *users* tablice koristeći Liquibase i XML

Kreirana tablica je na sličan način popunjena testnim podacima u datoteci *changelog-test-verzija.xml* (Slika 4.3.).

```
<changeSet id="2023081222252" author="pbarisic">
    <insert tableName="users">
        <column name="full_name" value="Petar Barišić"/>
        <column name="email" value="pbarisic@gmail.com"/>
        <column name="password" value="$2a$10$P84plkfAJO1qKgqDaSAsS.hrZ8scFZ3hd.l2Ba6KlBQ/pLemsrR36"/>
        <column name="role" value="ADMIN"/>
    </insert>
</changeSet>
```

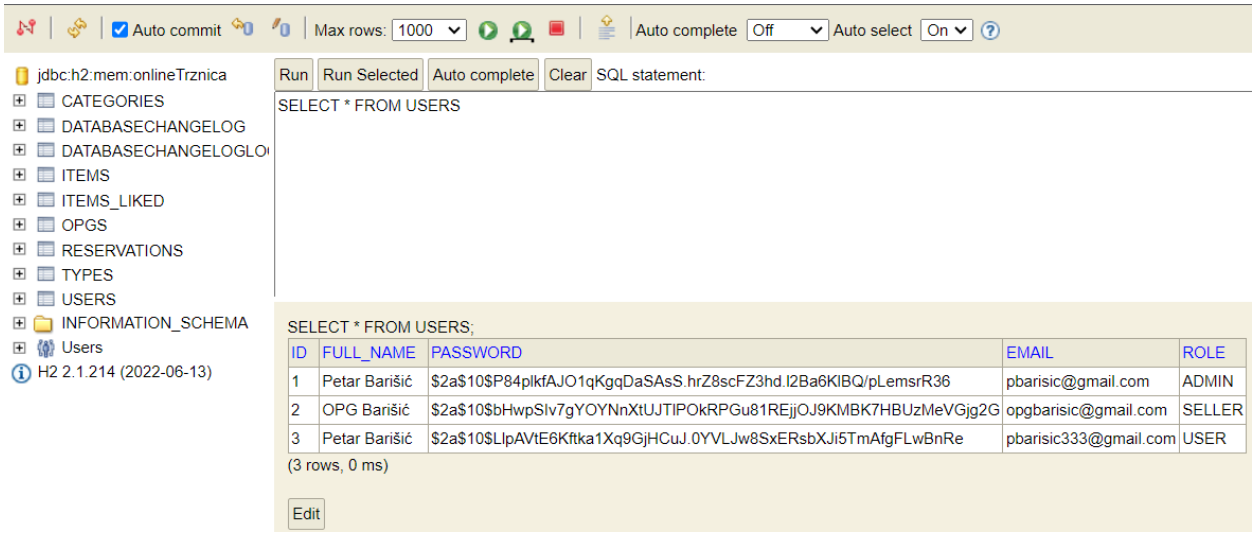
Slika 4.3. Popunjavanje *users* tablice koristeći Liquibase i XML

U datoteci *test-changelog* su dodane sve datoteke koje su korištene za upravljanje i popunjavanje tablica te je to ujedno datoteka iz koje Spring pokreće bazu. Vrlo je bitan redoslijed dodavanja datoteka, odnosno prvo se treba dodati datoteka koja kreira ili mijenja tablicu pa tek onda datoteka koja ju popunjava (Slika. 4.4.)

```
<include file="db/changelog/changelog-0.1.0.xml"/>
<include file="db/test-changelog/changelog-test-0.1.0.xml"/>
<include file="db/changelog/changelog-0.1.1.xml"/>
<include file="db/test-changelog/changelog-test-0.1.1.xml"/>
<include file="db/changelog/changelog-0.1.2.xml"/>
<include file="db/changelog/changelog-0.1.3.xml"/>
<include file="db/test-changelog/changelog-test-0.1.2.xml"/>
```

Slika 4.4. Prikaz dijela *test-changelog.xml* datoteke

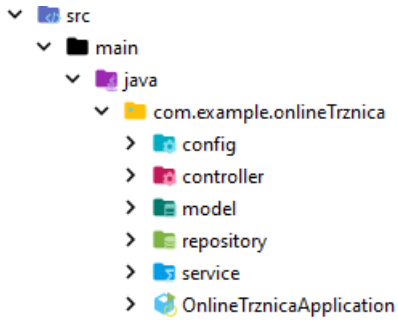
Prikaz tablica je moguće vidjeti na mrežnom sučelju koji se nalazi na poveznici *http://localhost:8080/api/h2-console* (Slika 4.5.).



Slika 4.5. Grafički prikaz tablice *users*

4.2. Backend

Uz pomoć Spring Boot anotacija i Java programskog jezika vrlo jednostavno je implementirana *backend* logika. Arhitektura se sastoji od 4 dijela: modela, repozitorija, servisa i kontrolera. Time se postigla dobra organizacija i struktura koda (Slika 4.6.).



Slika 4.6. Prikaz *backend* arhitekture

4.2.1. Model

Model predstavlja strukturu podatka ili objekta koji se koristi u mrežnoj aplikaciji. Definira kako će podaci biti organizirani i pohranjeni. Odgovoran je za definiranje svojstava entiteta, kao što su atributi i metode za pristup i manipulaciju podacima (Slika 4.7.).

```java
@Entity
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "items")
public class Item {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;

        @Column(nullable = false)
        private String name;

        @Column(nullable = false)
        private BigDecimal quantity;

        @Column(nullable = false)
        @Enumerated(EnumType.STRING)
        private MeasureUnit measureUnit;

        @Column(nullable = false)
        private BigDecimal price;

        1 usage
        @JsonIgnoreProperties("itemList")
        @ManyToOne(optional = false)
        @JoinColumn(name = "item_type_id",nullable = false)
        private ItemType itemType;

        1 usage
        @JsonIgnoreProperties("itemList")
        @ManyToOne(optional = false)
        @JoinColumn(name = "opg_id",nullable = false)
        private Opg opg;

        @ManyToMany(mappedBy = "likedItems")
        private List<User> likes;
}
```

Slika 4.7. Prikaz modela *Item*

Pomoću anotacija *@Entity* i *@Table,* Spring Boot zna da je model entitet i da pripada tablici *items.* *@NoArgsConstructor* i *@AllArgsConsctrutor* anotacije pružaju gotove konstruktore za model, a *@Getter* i *@Setter* gotove „gettere“ i „settere“. Time se skratilo vrijeme pisanja programskoga koda i skratio se broj linija, što utječe na lakšu čitljivost. Uz atribute se nalaze anotacije *@Column* čime se povezuje atribut modela i red u tablici, a *@ManyToOne* i *@ManyToMany* predstavlja odnose između tablica, odnosno modela koji su ranije definirani.

Uz model su usko vezani *RequestDto* i *ResponseDto*. DTO (engl. *Data Transfer Object)* predstavlja objekt za prijenos vrijednosti modela ovisno o tome služi li za zahtjev ili za odgovor između klijenta i poslužitelja.

RequestDto je objekt koji služi za prijenos vrijednosti koje zahtjeva poslužitelj (Slika 4.8.). U njemu se validiraju vrijednosti prije nego što se obavi poziv sa servisom. Na taj način se može spriječiti slanje nevažećih podataka, smanjujući opterećenje na poslužitelju i poboljšavajući brzinu odgovora.

```
13  @Data
14  @NoArgsConstructor
15  @AllArgsConstructor
16  @Validated
17  public class ItemRequestDto {
18          @NotBlank(message = "Obavezno polje.")
19          private BigDecimal quantity;
20
21          @NotBlank(message = "Obavezno polje.")
22          private MeasureUnit measureUnit;
23
24          @NotBlank(message = "Obavezno polje.")
25          private BigDecimal price;
26
27          @NotNull(message = "Obavezno polje.")
28          private Long itemTypeId;
29
30          @NotNull(message = "Obavezno polje.")
31          private Long opgId;
32  }
```

Slika 4.8. Prikaz *RequestDtoa* modela *item*

@Data služi za kreiranje „gettera“ i „settera“ za jednostavne objekte kao što je DTO, a *@Validated* za primjenu validacije na sve atribute klase. *@NotNull* anotacija ne dozvoljava da vrijednost bude *null*, a *@NotBlank* dodatno ne dozvoljava da vrijednost ima manje od jednog znaka.

ResponseDto je objekt koji služi za prijenos vrijednosti kao odgovor klijenta poslužitelju (Slika 4.9.). Objekt sadrži samo one vrijednosti koje su bitne za prikazati, time optimizira opterećenje mreže i pruža jasnu strukturu podataka s kojima poslužitelj zna upravljati.

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
public class ItemResponseDto {

        @Nullable
        private Long id;

        @Nullable
        private String name;

        @Nullable
        private BigDecimal quantity;

        @Nullable
        private String measureUnit;

        @Nullable
        private BigDecimal price;

        @Nullable
        private Long itemTypeId;

        @Nullable
        private List<String> likes;

        @Nullable
        private String itemType;

        @Nullable
        private String opgName;

        @Nullable
        private Long opgId;
}
```

Slika 4.9. Prikaz *ResponseDtoa* za model *item*

Za transformaciju iz *RequestDto* u model ili iz modela u *ResponseDto* je zaslužan *mapper*. *Mapper* je komponenta koja sadrži metodu *map* s kojom mapira jedan ili više DTO-a u model ili obrnuto. Po potrebi mijenja tipove podataka kako bi se vrijednosti prilagodile traženom objektu (Slika 4.10.).

```java
@RequiredArgsConstructor
@Component
public class ItemRequestDtoToItemMapperImpl implements EntityDtoMapper<ItemRequestDto, Item> {

    1 usage
    @NotNull
    private final ItemTypeService itemTypeService;

    1 usage
    @NotNull
    private final OpgService opgService;

    Petar Barisic
    @Override
    public Item map(@NotNull final ItemRequestDto requestDto) {
        final Item item = new Item();
        final ItemType itemType = itemTypeService.loadItemTypeById(requestDto.getItemTypeId());

        item.setName(itemType.getTypeName());
        item.setQuantity(requestDto.getQuantity());
        item.setMeasureUnit(requestDto.getMeasureUnit());
        item.setPrice(requestDto.getPrice());
        item.setItemType(itemType);
        item.setOpg(opgService.loadOpgById(requestDto.getOpgId()));

        return item;
    }
```

Slika 4.10. Prikaz *mappera* iz *RequestDto* u *item*

Anotacija *@Component* je obavezna kako bi Spring Boot prepoznao da je mapper dio Spring okruženja.

4.2.2. Repozitorij

Repozitorij je komponenta koja omogućuje pristup podacima iz baze podataka. Ona obavlja operacije poput spremanja, dohvaćanja, ažuriranja i brisanja podataka iz baze podataka. Odvojen od kontrolera i servisa kako bi se odvojila logika za pristup podacima od poslovne logike (Slika 4.11.).

```java
@Repository
public interface ItemRepository extends JpaRepository<Item, Long> {

    1 usage  Petar Barisic
    List<Item> findAllByOpgId(@NotNull final Long id);
}
```

Slika 4.11. Prikaz repozitorija za model *item*

@Repository govori Springu da trenutna komponenta predstavlja repozitorij. Nasljeđuje sučelje *JpaRepositoriy* koje samo stvara SQL upite te se tako može pristupiti osnovnim metodama kao što su dohvat svih elemenata, elemenata po *id-u*, brisanje elementa i sl. Ukoliko postoji specifičan zahtjev prema bazi, definira se dodatan potpis metode kao što je to metoda *findAllByOpgId()*.

4.2.3. Servis

Servis predstavlja sloj poslovne logike aplikacije. Sadrži metode i logiku koje se koriste za izvršavanje specifičnih operacija ili izračuna. Koristi pozive iz repozitorija, obavlja dodatne provjere i operacije ukoliko je potrebno te vrača vrijednost koja je potrebna kontroleru (Slika 4.12.).

```
@Service
public class ItemServiceImpl implements ItemService {

        7 usages
        private final ItemRepository itemRepository;

        Petar Barisic
        public ItemServiceImpl(@NotNull final ItemRepository itemRepository) { this.itemRepository = itemRepository; }

        1 usage  Petar Barisic
        @Override
        public List<Item> loadAllItems() {

                return itemRepository.findAll();
        }

        1 usage  Petar Barisic
        @Override
        public List<Item> loadAllItemsByOpgId(@NotNull final Long id) { return itemRepository.findAllByOpgId(id); }

        5 usages  Petar Barisic
        @Override
        public Item loadItemById(@NotNull final Long id) throws NoSuchElementException {
                return itemRepository.findById(id).orElseThrow(
                        ()->new NoSuchElementException("Proizvod ne postoji.")
                );
        }

}
```

Slika 4.12. Prikaz dijela servisa za *item* model

Kao i repozitorij, tako i servis ima svoju anotaciju *@Service*. Nasljeđuje sučelje s metodama karakteristične za svaki model te se time svaka metoda mora prepisati i prilagoditi potrebama kontrolera. Metoda *loadItemById()* prikazuje kako servis uz dohvaćanje proizvoda po *id-u* , vrši i provjeru te ukoliko dođe do greške, prikazuje definiranu poruku.

4.2.4. Kontroler

Kontroler je odgovoran za obradu HTTP (engl. *The Hypertext Transfer Protocol*) zahtjeva koji dolaze od korisnika putem mrežnog preglednika. Kontroler prima zahtjeve u oblik REST (engl. *Representational State Transfer*) poziva, obrađuje ih i utvrđuje koje akcije treba poduzeti. U REST pozive spada *get* koji služi za dohvaćanje podataka, *post* za spremanje podataka, *put* za izmjenu podataka te *delete* koji služi za brisanje podataka. Kontroler koristi servise, DTO-ove i *mappere* kako bi izvršio poslovnu logiku i dohvatio ili ažurirao podatke (Slika 4.13.).

```
21  @CrossOrigin
22  @RestController
23  @RequestMapping("/items")
24  public final class ItemController {
25
            8 usages
26          @NotNull
27          private final ItemService itemService;
28
            3 usages
29          @NotNull
30          private final UserService userService;
            6 usages
31          @NotNull
32          private final ItemToItemResponseDtoMapperImpl itemToItemResponseDtoMapper;
            3 usages
33          @NotNull
34          private final ItemRequestDtoToItemMapperImpl itemRequestDtoToItemMapper;
35
            Petar Barisic
36          @Autowired
37          public ItemController(@NotNull final ItemService itemService,
38                                @NotNull final UserService userService,
39                                @NotNull final ItemToItemResponseDtoMapperImpl itemToItemResponseDtoMapper,
40                                @NotNull final ItemRequestDtoToItemMapperImpl itemRequestDtoToItemMapper) {
41                  this.itemService = itemService;
42                  this.userService = userService;
43                  this.itemToItemResponseDtoMapper = itemToItemResponseDtoMapper;
44                  this.itemRequestDtoToItemMapper = itemRequestDtoToItemMapper;
45          }
```

Slika 4.13. Prikaz prvog dijela kontrolera *item* modela

Kontroler također ima svoju Spring Boot anotaciju *@RestController* te *@RequestMapping* anotaciju, koja predstavlja mapiranje mrežnih poziva na kontroler i putanju preko koje će se ti pozivi odvijati. Za *item* to je već predefinirani *http://localhost:8080/api* te dodatak */items* koji je dodan u *@RequestMappingu*. @CrossOrigin služi za omogućavanje razmjene podataka između dva različita servera, kao što je *localhost:8080* za *backend* i *localhost:3000* za *frontend*. *@Autowired* anotacija koja se nalazi iznad kontrolera služi za ubrizgavanje ovisnosti drugih Spring Boot komponenata, kao što su *mapperi* i servisi, u kontroler. Metoda *loadItemById()* predstavlja *get* REST poziv kojoj se uz putanju za *item* šalje i varijabla *id*. Pomoću servisa se preko

id-a pronađe *item* te ga se mapira koristeći *mapper* za *ResponseDto.* Ukoliko postoji *item* za zadani *id*, metoda vraća poslužitelju *ResponseDto* uz statusni kod 200 koji predstavlja uspješnost (Ok), no ukoliko ne postoji *item* za zadani *id* , poslužitelju vraća statusni kod 404 (Not Found). *Delete* REST poziv radi na isti princip, samo što vraća statusni kod bez DTO-a. *Post* metoda *createItem()* prima kao zahtjev JSON objekt kao *RequestDto*, mapira ga u *item* te ga pomoću servisa sprema u bazu. Za uspješnost operacije vraća statusni kod 201 (Created) (Slika 4.14.).

```
55          @GetMapping("{id}")
56          public ResponseEntity<ItemResponseDto> loadItemById(
57                  @NotNull @PathVariable final Long id
58          ) {
59                  try {
60
61                          final Item item = itemService.loadItemById(id);
62                          final ItemResponseDto responseDto = itemToItemResponseDtoMapper.map(item);
63
64                          return ResponseEntity.ok().body(responseDto);
65                  } catch (NoSuchElementException e) {
66                          return ResponseEntity.notFound().build();
67                  }
68          }
69
            Petar Barisic
70          @DeleteMapping("{id}")
71          public ResponseEntity<?> deleteItem(@NotNull @PathVariable final Long id) {
72
73                  final boolean isDeleted = itemService.deleteItem(id);
74                  if (!isDeleted) {
75                          return ResponseEntity.notFound().build();
76                  } else {
77                          return ResponseEntity.ok().build();
78                  }
79          }
80
            Petar Barisic
81          @PostMapping
82          public ResponseEntity<ItemResponseDto> createItem(
83                  @NotNull @RequestBody final ItemRequestDto itemRequestDto
84          ) {
85
86                  Item item = itemRequestDtoToItemMapper.map(itemRequestDto);
87                  item = itemService.saveItem(item);
88
89                  return ResponseEntity.status(HttpStatus.CREATED).body(itemToItemResponseDtoMapper.map(item));
90          }
```

Slika 4.14. Prikaz REST poziva u *item* kontroleru

4.2.5. Security

Spring Security je moćan okvir u Springu koji omogućuje razvojnim inženjerima da lako implementiraju sigurnost i autorizaciju u svojim mrežnim aplikacijama. Kada se radi o JWT (engl. *JSON Web Token*) autorizaciji, Spring Security pruža robustan pristup zaštiti resursa i autentikaciji korisnika koristeći JWT tokene. Prilikom uspješne autentikacije korisnika, JWT token se generira i potpisuje pomoću tajnog ključa na serveru (Slika 4.15.). Taj token se šalje klijentu, koji ga obično čuva u memoriji ili u kolačiću. Klijent šalje JWT token u svakom zahtjevu prema serveru. Spring Security zatim provjerava i dekodira token kako bi utvrdio identitet korisnika i odredio kojim resursima ima pravo pristupa.

```
32        public String generateToken(UserDetails userDetails) { return generateToken(new HashMap<>(),userDetails); }
35
                1 usage   Petar Barisic
36  @     public String generateToken(Map<String, Object> extraClaims, UserDetails userDetails) {
37            return Jwts
38                    .builder()
39                    .setClaims(extraClaims)
40                    .setSubject(userDetails.getUsername())
41                    .setIssuedAt(new Date(System.currentTimeMillis()))
42                    .setExpiration(new Date(System.currentTimeMillis() + (1000 * 60 * 60)))
43                    .signWith(getSignInKey(), SignatureAlgorithm.HS256)
44                    .compact();
45        }
```

Slika 4.15. Prikaz metode koja generira JWT token

Trajanje tokena je namješteno na 360000 milisekundi, odnosno sat vremena. Nakon sat vremena, token ističe i korisnik nema više pravo pristupu podacima sve dok se ponovno ne prijavi. Za potpisivanje tokena se koristi HS256 simetrički algoritam.

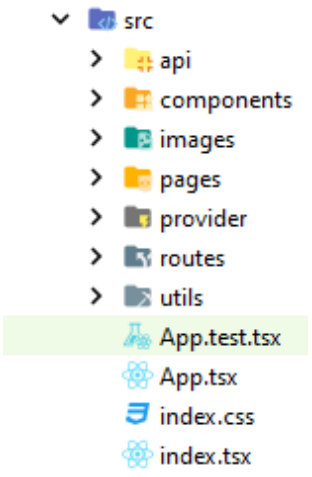
Ključna komponenta u Spring Security konfiguraciji je *SecurityConfiguration*. Ova klasa definira kako će mrežna aplikacija rukovati autentikacijom i autorizacijom. U kontekstu JWT autorizacije, *SecurityConfiguration* će sadržavati konfiguraciju za JWT filtere i pristupne točke (engl. *endpoints*) za autentikaciju. Konfiguracija će također sadržavati pravila za dozvoljeni pristup resursima na temelju JWT tokena. Na primjer, *post* REST pozivima za *items* pristupnu točku mogu pristupiti korisnici koji imaju ulogu administratora ili prodavača, dok drugi korisnici nemaju prava pristupa (Slika 4.16.).

```java
@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
public class SecurityConfiguration {

    1 usage
    private final JwtAuthenticationFilter jwtAuthFilter;
    1 usage
    private final AuthenticationProvider authenticationProvider;

    Petar Barisic
    @Bean
    @Order(1)
    public SecurityFilterChain filterChain(@NotNull final HttpSecurity http) throws Exception {
        http.csrf(AbstractHttpConfigurer::disable)
                .authorizeHttpRequests((requests) -> {

                    requests.requestMatchers(
                            AntPathRequestMatcher.antMatcher(HttpMethod.POST,  pattern: "/items")
                            )
                            .hasAnyAuthority(
                                    UserRole.ADMIN.toString(),
                                    UserRole.SELLER.toString()
                            );
```

Slika 4.16. Prikaz dijela *SecurityConfiguration* klase

4.3. Frontend

Frontend je dio mrežne aplikacije što korisnici vide i s čime vrše interakciju. Upravo React.js omogućuje izgradnju modernih i dinamičkih korisničkih sučelja koja su privlačna, brza i responzivna. To utječe na ukupno korisničko iskustvo, što je ključno za zadovoljstvo i zadržavanje korisnika. Glavni dio *Frontend* arhitekture se sastoji od tipova podataka i API poziva, komponenata, stranica i ruta (Slika 4.17.).



Slika 4.17. Arhitektura *frontenda*

4.3.1. Tipovi podataka i API pozivi

Pomoću TypeScripta su točno definirani tipovi podataka koji se očekuje od kontrolera na *backendu* i tipovi podataka koje očekuje sam kontroler (Slika 4.18.). To omogućuje jednostavnu kontrolu podacima koji se kasnije koriste kroz komponente i stranice mrežne aplikacije.

```
export type ItemResponse = {
    id: number;
    name: string;
    quantity: number;
    price: number;
    measureUnit: string;
    itemType: string;
    opgName: string;
    likes: string[];
    itemTypeId: number;
    opgId: number
};
```

Slika 4.18. Prikaz definiranog tipa podatka *ItemResponse*

ItemResponse predstavlja objekt, odnosno ranije spomenuti *ResponseDto*, koji *backend* šalje kao odgovor na zahtjev poslužitelja. Atributi unutar objekta mogu biti jednostavnog tipa kao što je *string* i *number*, ili mogu biti zahtjevnijeg tipa kao što je niz i objekt. Svojstvo *export* omogućuje pristup tipa podatka kroz cijelu mrežnu aplikaciju.

API pozivi prema *backendu* su mogući pomoću *axios* biblioteke. On omogućuje lako izvođenje HTTP zahtjeva, bez obzira na to koristi li se *get*, *post*, *put* ili *delete* poziv. Također automatski pretvara ručno definirane tipove podataka u JSON objekt kada šalje zahtjev ili obrnuto kada prima odgovor. Time je pojednostavljena i olakšana komunikacija između *frontenda* i *backenda*.

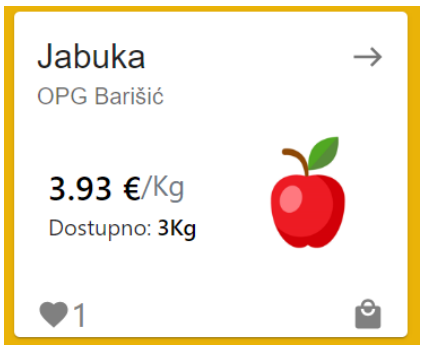
Pomoću *axios.defoults.baseURL* je definirana osnovna putanja za *axios* pozive. Konstanta *createItem()* zapravo predstavlja funkciju koja prima parametre tipa *ItemRequest* te pomoću *axiosa* šalje podatke prema osnovnoj putanji uz dodatni nastavak „/items“. Nakon toga povratnu vrijednost tipa *Promise<AxiosResponse>* sprema u konstantu. Pomoću ključne rijeci *async* funkcija postaje asikrona te se pomoću *await* omogućuje jasno upravljanje redoslijedom izvršenja operacija i čekanja na njihov završetak prije nego što se nastavi s izvođenjem ostatka programskoga koda. Povratni tip *Promise<AxiosResponse>* sadrži atribut *data*, koji predstavlja vrijednosti podataka koje šalje *backend* i *errors*, koji sadrži statusni kod pomoću kojeg se može upravljati validacijom na *frontendu*. Na kraju se sve komponente izvoze za daljnje korištenje (Slika 4.19.).

```
import axios, {AxiosResponse} from "axios";
import {createAxiosConfig} from "../utils";
import {ItemRequest, LikeRequest} from "./types";

axios.defaults.baseURL = 'http://localhost:8080/api';

const loadAllItems = async (): Promise<AxiosResponse> => {
    return axios.get( url: "/items", createAxiosConfig());
};

const createItem = async (values: ItemRequest): Promise<AxiosResponse> => {
    return axios.post( url: "/items", values, createAxiosConfig());
};

const deleteItem = async (id: number | string): Promise<AxiosResponse> => {
    return axios.delete( url: `/items/${id}`, createAxiosConfig());
};

const updateItem = async (id: number | string, value: ItemRequest): Promise<AxiosResponse> => {
    return axios.put( url: `/items/${id}`, value, createAxiosConfig());
};

const likeItem = async (values: LikeRequest): Promise<AxiosResponse> => {
    return axios.post( url: "/items/likeItem", values, createAxiosConfig());
};
export {loadAllItems, createItem, deleteItem, updateItem, likeItem}
```

Slika 4.19. Prikaz *apiCalls.ts* datoteke za *item* model

4.3.2. Komponente

Komponente su ključni koncept u React.js i predstavljaju izuzetno važnu ulogu u razvoju modernih mrežnih aplikacija. One omogućuju podjelu korisničkog sučelja na manje, samostalne dijelove. To čini kod modularnijim i olakšava razvoj, održavanje i ponovnu upotrebu. Upravo ponovna upotreba omogućuje korištenje komponente više puta u različitim dijelovima aplikacije ili čak u različitim projektima, čime se smanjuje potreba za ponovnim pisanjem istog programskog koda. Upravo takva je i komponenta *ItemCard* koja se koristi nekoliko puta kroz cijelu aplikaciju (Slika 4.20.).



Slika 4.20.Prikaz komponente *ItemCard*

Komponente su izolirane i neovisne o ostatku aplikacije, što ih čini lakšima za testiranje. Može se testirati svaka komponenta pojedinačno bez potrebe za pokretanjem cijele aplikacije. React.js omogućuje optimizaciju učitavanja stranica pomoću dinamičkog učitavanja komponenti. To znači da se komponente učitavaju samo kada su stvarno potrebne, što ubrzava inicijalno učitavanje aplikacije. Također postoji velik ekosustav dostupnih komponenti koje se mogu koristiti u vlastitoj mrežnoj aplikaciji. U *ItemCard* komponenti se koriste već gotove komponente biblioteke Material UI koje su prilagođene prema vlastitoj aplikaciji te uz pomoć TailwindCSS biblioteke kontrolirano i na jednostavan način prikazane na zaslonu. Neki od korištenih komponenata su *Card*, *CardHeader* i *CardContent* (Slika 4.21.).

```
170     return (
171         <Card
172             sx={{
173                 maxHeight: !expanded ? 230 : 380, ':hover': {
174                     boxShadow: 20,
175                 },
176                 transition: "transform 0.15s ease-in-out",
177                 "&:hover": {transform: "scale3d(1.02, 1.02, 1)"},
178             }}>
179             <CardHeader
180                 action={
181                     haveLink && <IconButton onClick={() => navigate( to: `${Paths.OpgPage}/${opgId}`)} aria-label="više">
182                         <East/>
183                     </IconButton>
184                 }
185                 title={name}
186                 subheader={opgName}
187             />
188             <div className="grid grid-cols-2 justify-center">
189                 <CardContent>
190                     <div className="mt-1 ml-2">
```

Slika 4.21. Prikaz dijela implementacije *ItemCard* komponente

Za upravljanje stanjem (engl. *state*) u funkcionalnim komponentama koristi se *useState*, koji je jedan od najvažnijih React.js-ovih *Hookova*. *Hookovi* su funkcije koje omogućuju komponentama da koriste stanje, efekte i druge funkcionalnosti koje su ranije bile dostupne samo u klasnim komponentama (Slika 4.22.). Stanje je dinamički podatak koji može varirati tijekom životnog ciklusa komponente i utječe na to kako se komponenta prikazuje i ponaša.

```
71      const cart = useContext(CartContext);
72      const {token, role} = useContext(AuthContext);
73      const [isSubmitting, setIsSubmitting] = useState<boolean>( initialState: false);
74      const [itemResponse, setItemResponse] = useState<ItemResponse>()
75      const navigate = useNavigate();
```

Slika 4.22. Prikaz React.js *Hookova* korištenih u *IteamCard* komponenti

Najvažnija komponenta u svakoj React.js mrežnoj aplikaciji je *App.tsx*. Ona čini okvir mrežne aplikacije te sadrži njenu osnovnu strukturu, uključujući navigaciju, zajedničke komponente, izgled i druge globalne aspekte. Preko nje se određuje koja će se stranica prikazivati trenutno na zaslonu korisnika (Slika 4.23.).

```
import React from 'react';
import { BrowserRouter } from "react-router-dom";
import Routes from "./routes/Routes";

function App() {
    return (
        <BrowserRouter>
            <Routes />
        </BrowserRouter>);
}

export default App;
```

Slika 4.23. Prikaz *App.tsx* komponente

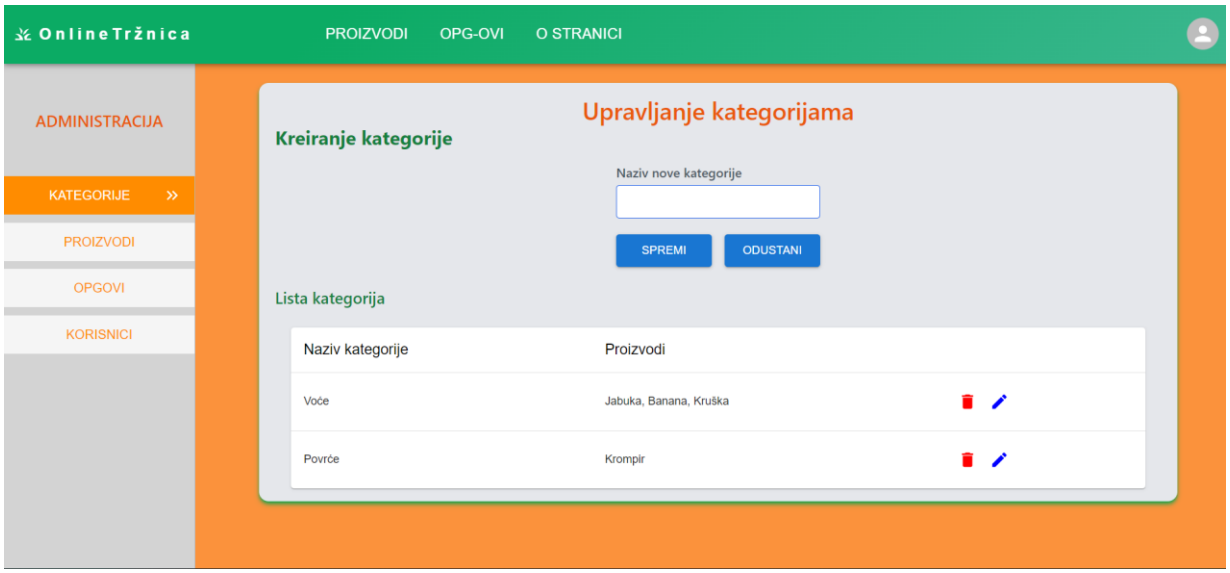
4.3.3. Stranice

Stranice (engl. *pages*) predstavljaju pojedinačne vizualne komponente ili sekcije komponenata koje se korisnicima prikazuju kao samostalne jedinice sadržaja. Svaka stranica često ima svoju svrhu i funkcionalnost te se može prikazivati kao samostalna ruta u aplikaciji. Komponente koje se nalaze unutar stranice mogu preko stranice međusobno komunicirati šaljući vrijednosti jedna drugoj (Slika 4.24.).

```
75    useEffect( effect: () => {
76        void loadCategories();
77        void loadItemTypes();
78        void loadOpgs();
79        void loadUsers();
80
81    }, deps: [loadCategories, loadItemTypes, loadOpgs, loadUsers, tabIndex]);
82    return (<>
83        <div className="bg-orange-400 min-h-screen">
84            <div className="grid">
85                <HeadBar/>
86                <SideBar setTabIndex1={setTab1} setTabIndex2={setTab2} setTabIndex3={setTab3} setTabIndex4={setTab4}
87                         tabs={tabs} title={title}/>
88            </div>
89            {tabIndex === 1 && (<EditCategoriesForm categories={categories} refreshCategories={loadCategories}/>)}
90            {tabIndex === 2 && (
91                <EditItemTypesForm itemTypes={itemTypes} categories={categories} refreshItemTypes={loadItemTypes}/>)}
92            {tabIndex === 3 && (<EditOpgs opgs={opgs} refreshOpgs={loadOpgs}/>)}
93            {tabIndex === 4 && (<EditUsers users={users} refreshUsers={loadUsers}/>)}
94
95        </div>
96        <Footer/>
97    </>);
```

Slika 4.24. Prikaz stranice *AdminPage.tsx*

Stranica *AdminPage.tsx* sadrži komponente *HeadBar.tsx*, koja predstavlja izbornik koja se nalazi na vrhu stranice, *SideBar.tsx*, izbornik sa strane te *Footer.tsx*, koji se nalazi na podnožju stranice. Ovisno o elementu koji korisnik odabere pomoću izbornika sa strane, stranica će prikazivati određenu formu za uređivanje (Slika 4.25.).



Slika 4.25. Prikaz *AdminPage.tsx* stranice

Stranica sadrži *Hook* koji se zove *useEffect*. On omogućuje da se stranica ponovno učita svaki put kada se dogodi promjena u vrijednostima koji su mu predani kroz ovisnosti (engl. *dependecies*). Stoga svaki put kada se dogodi promjena na nekoj od formi za uređivanje, komponenta će pozvati funkciju za dohvaćanje njoj potrebnih podataka te, ukoliko se ta funkcija nalazi u ovisnostima *useEffecta*, *useEffect* će ponovno osvježiti stranicu i korisniku će se prikazati najnoviji podaci.

4.3.4. Rute

Rute se koriste za upravljanje navigacijom između različitih komponenata ili stranica u mrežnoj aplikaciji. Ovaj koncept se koristi uz pomoć biblioteke za upravljanje rutama React Router. Najprije su definirane krajnje točke putanja za svaku stranicu (Slika 4.26.).

```
export enum Paths {
    Home = "/",
    Login = "/login",
    Registration = "/registration",
    RegistrationOPG = "/registration-opg",
    OpgPage = "/opg",
    AdminPage = "/administration",
    EditOpg = "/my-opg",
    UserPage = "/my-profile",
    ItemsPage = "/items",
    OpgsPage = "/opgs"
}
```

Slika 4.26. Prikaz *Paths.ts* datoteke

Zatim je kreirana konfiguracija gdje se određuje na kojoj putanji će se prikazat koja stranica te s kojom rolom joj korisnik smije pristupiti (Slika 4.27.).

```
const RouteConfiguration: RouteInfo[] = [
    { path: Paths.Registration, component: RegistrationPage, visibility: RouteVisibility.LoggedOut  },
    { path: Paths.RegistrationOPG, component: RegistrationPageOPG, visibility: RouteVisibility.LoggedOut  },
    { path: Paths.Login, component: LoginPage, visibility: RouteVisibility.LoggedOut  },
    { path: Paths.Home, component: HomePage, visibility: RouteVisibility.Everyone },
    { path: Paths.OpgPage +"/:id", component: OpgPage, visibility: RouteVisibility.Everyone },
    { path: Paths.AdminPage, component: AdminPage, visibility: RouteVisibility.Admin },
    { path: Paths.EditOpg, component: EditOpgPage, visibility: RouteVisibility.Seller },
    { path: Paths.UserPage, component: UserPage, visibility: RouteVisibility.User },
    { path: Paths.ItemsPage, component: ItemsPage, visibility: RouteVisibility.Everyone },
    { path: Paths.OpgsPage, component: OpgsPage, visibility: RouteVisibility.Everyone },
];

export default RouteConfiguration;
```

Slika 4.27. Prikaz *RouteConfig.ts* datoteke

I na kraju je dodana glavna komponenta koja upravlja u prikazu rutama u odnosu na trenutnog korisnika (Slika 4.28.).

```
const Routes = () => {
    const {token, role} = useContext(AuthContext);

    return (
        <DomRoutes>
            {RouteConfiguration.map((route : RouteInfo ) => {
                if (canShowRoute(token, role, route.visibility)) {
                    return <Route element={<route.component/>} key={route.path} path={route.path}/>;
                } else {
                    return null;
                }
            })}
        </DomRoutes>
    );
};
```

Slika 4.28. Prikaz *Routes.ts* datoteke

5. TESTIRANJE RADA APLIKACIJE

Testiranje omogućava identifikaciju i ispravak grešaka i problema u mrežnoj aplikaciji prije nego što se ona isporuči za uporabu, čime se štedi mnogo vremena i resursa. Aplikacije koje nisu testirane mogu biti nepouzdane i podložne rušenju ili neželjenom ponašanju. Pomoću testiranja se osigurava stabilnost i pouzdanost aplikacije te njeno daljnje održavanje. Testiranje je raspoređeno na *backend* i na *frontend* dijelu.

5.1. Backend testiranje

Na *backendu* je testiran servisni sloj za *Item* model koristeći funkcionalne testove. Funkcionalni testovi su vrsta testova koji se fokusiraju na to ispunjava li aplikacija funkcionalne zahtjeve i specifikacije. Ovi testovi ocjenjuju kako aplikacija reagira na različite ulazne podatke te provjeravaju ponaša li se aplikacija očekivano u skladu s funkcionalnim zahtjevima. Uz pomoć *@SpringBootTest* anotacije kreiran je testni kontekst u kojem su napisani funkcionalni testovi za sve metode iz *ItemServicea* (Slika 5.1.).

```
@SpringBootTest
public class ItemServiceTests {
        11 usages
        @Autowired
        @NotNull
        private ItemService itemService;
        1 usage
        private static final Long VALID_OPG_ID = 1L;
        1 usage
        private static final Long INVALID_OPG_ID = 0L;
        4 usages
        private static final Long VALID_ITEM_ID = 1L;
        2 usages
        private static final Long INVALID_ITEM_ID = 0L;

        @Test
        void shouldReturnItemsWhenCallingLoadAllItems(){
                final List<Item> items = itemService.loadAllItems();

                Assertions.assertFalse(items.isEmpty());
        }
        @Test
        void shouldReturnItemsWhenCallingLoadAllItemsByValidOpgId(){
                final List<Item> items = itemService.loadAllItemsByOpgId(VALID_OPG_ID);

                Assertions.assertFalse(items.isEmpty());
        }
        @Test
        void shouldNotReturnItemsWhenCallingLoadAllItemsByInvalidOpgId(){
                final List<Item> items = itemService.loadAllItemsByOpgId(INVALID_OPG_ID);

                Assertions.assertTrue(items.isEmpty());
        }
```

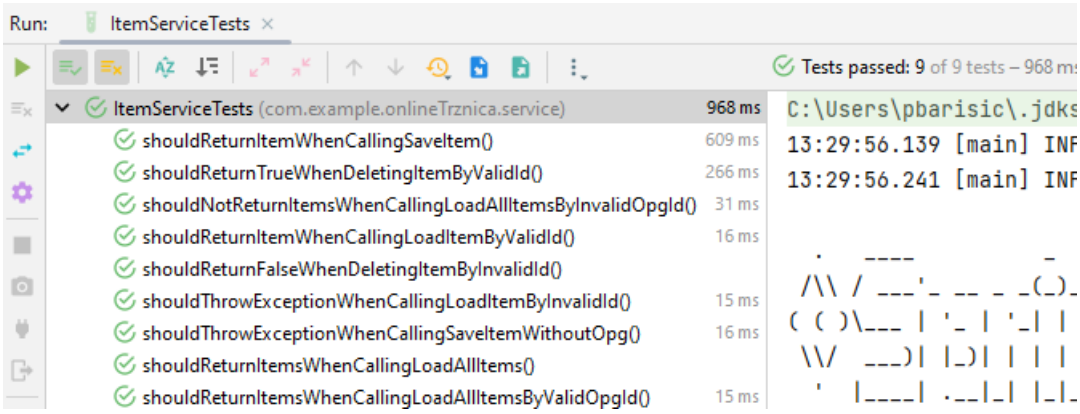
Slika 5.1. Prikaz dijela *ItemServiceTests* testne klase

Testne metode se imenuju tako da se zna što se očekuje kao povratna vrijednost testa, kako bi testovi bili pregledniji i kako bi lakše prepoznali grešku ukoliko testovi ne prođu. Metoda se označi *@Test* anotacijom te se sastoji od dijela gdje se poziva servis i od dijela gdje se provjerava odziv servisa. Najčešće se provjerava istinitost ili postojanost objekta, no može se provjeriti i bacanje određene iznimke, kao u slučaju dohvata proizvoda koji ne sadrži predanu *id* vrijednost (Slika 5.2.).

```
57
58      @Test
59      void shouldThrowExceptionWhenCallingLoadItemByInvalidId(){
60              Assertions.assertThrows(
61                      NoSuchElementException.class,
62                      () -> itemService.loadItemById(INVALID_ITEM_ID)
63              );
        }
```

Slika 5.2. Prikaz provjere bacanja specifične iznimke

Kombinacijom tipki „CTRL + SHIFT + F10“ se pokreću svi testovi koje se nalaze u trenutnom *SpringBootTest* kontekstu. Nakon što se svi testovi izvrše, prikazuje se rezultat koji pokazuje koliko je testova prošlo te koliko dugo su se izvršavali (Slika 5.3.).



Slika 5.3. Prikaz rezultata testova

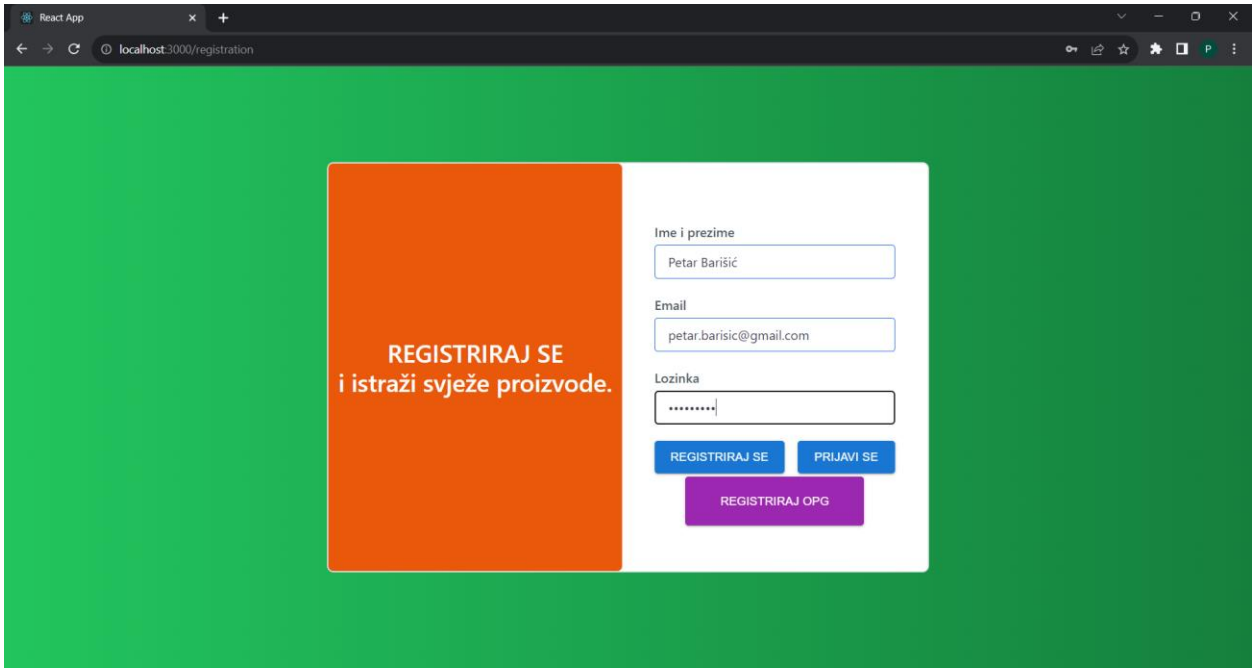
Svi testovi za servis *ItemService* su uspješno prošli te je za njihovo izvršavanje bilo potrebno 968 milisekundi.

5.2. Frontend testiranje

Frontend dio napisan u React.js-u je također moguće testirati na različite načine. Neki od načina su jedinično testiranje (engl. *Unit testing*), integracijsko testiranje (engl. *Integration testing*) i testiranje korisničkog sučelja (engl. *User interface testing*). U nastavku će biti prikazani koraci testiranja korisničkog sučelja kroz tri moguće uloge: korisnik, administrator i prodavač.

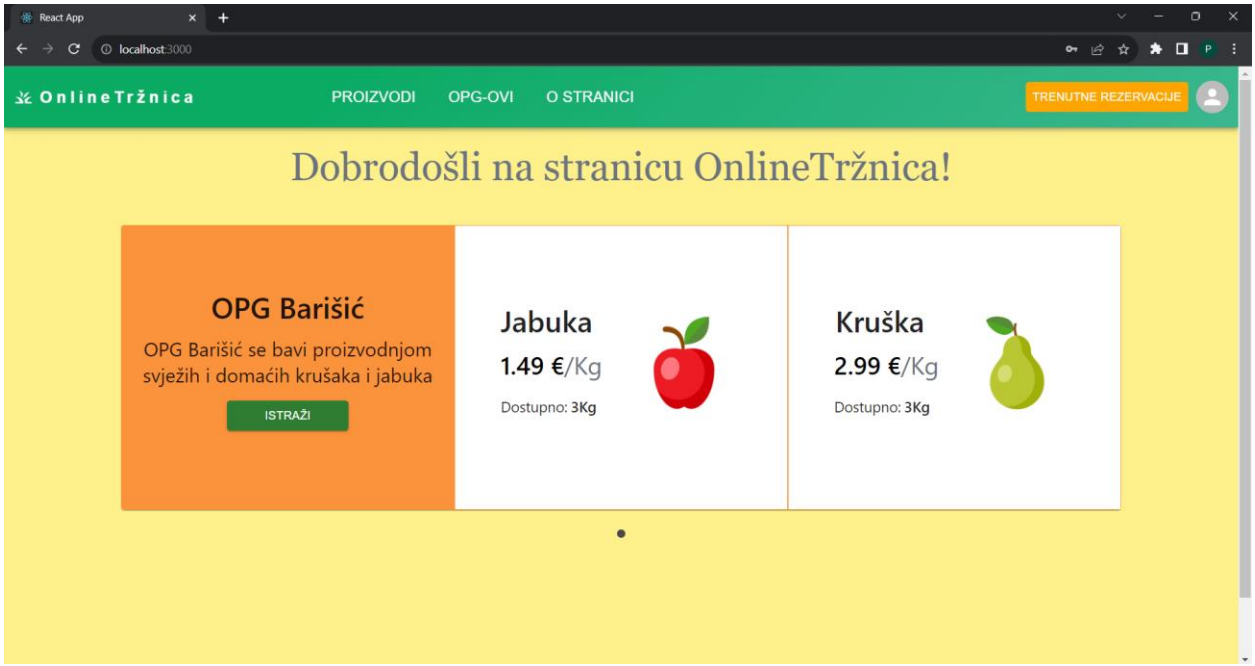
Korisnik rezervira željeni proizvod:

1. Korisnik uspješno popunjava polja za registraciju te se registrira na mrežnu stranicu *onlineTržnica* (Slika 5.4.).



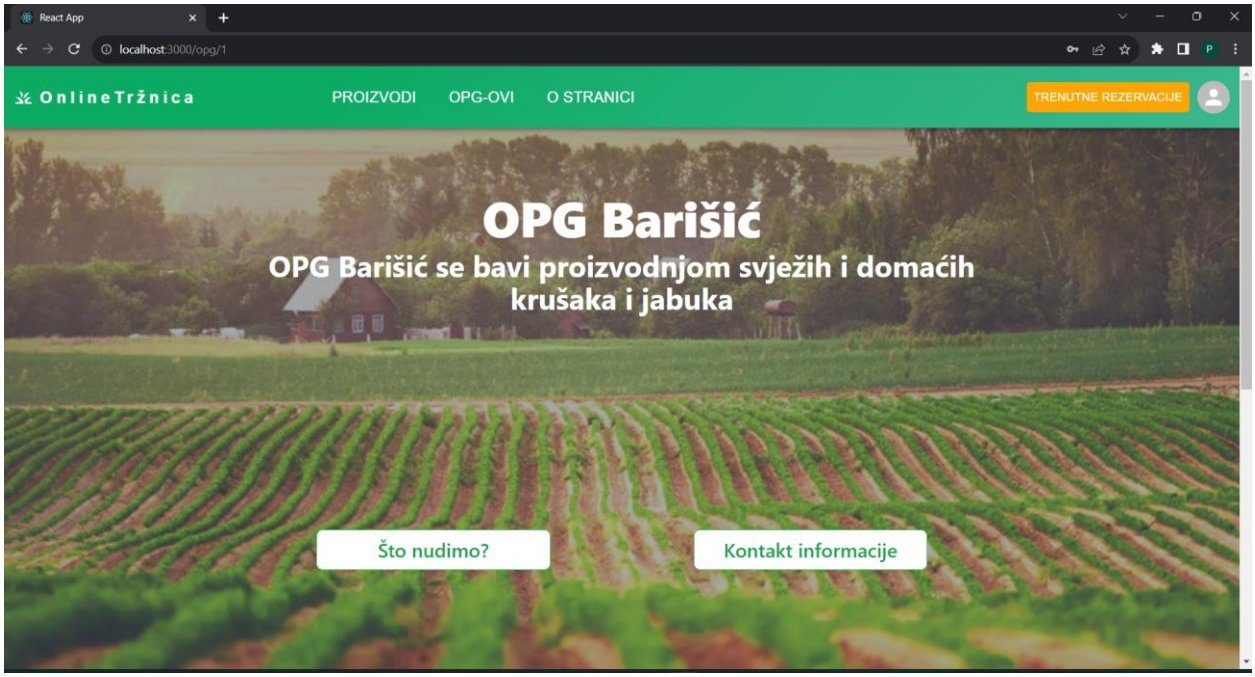
Slika 5.4. Prikaz forme za registraciju

2. Korisnik se uspješno registrirao te je proslijeđen na početnu stranicu (Slika 5.5.).



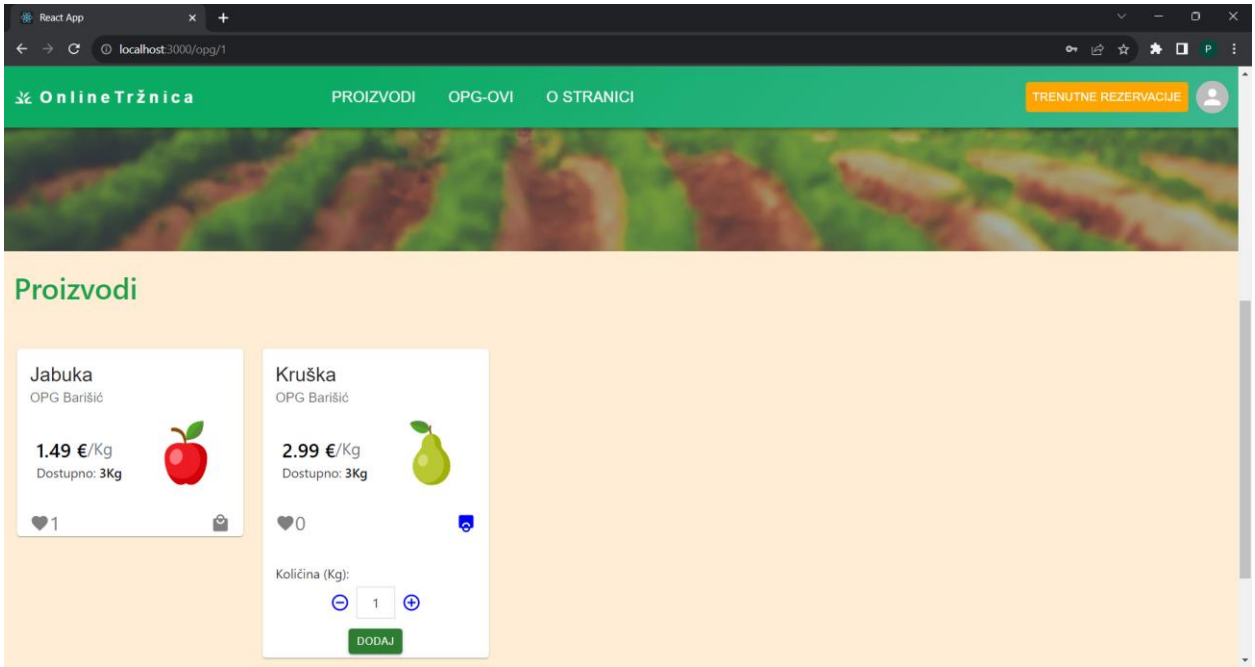
Slika 5.5. Prikaz početne stranice

3. Korisnik na „vrtuljku“ odabire željeni OPG, odabire gumb „ISTRAŽI“ te ga se prosljeđuje na stranicu željenog OPG-a (Slika 5.6.).



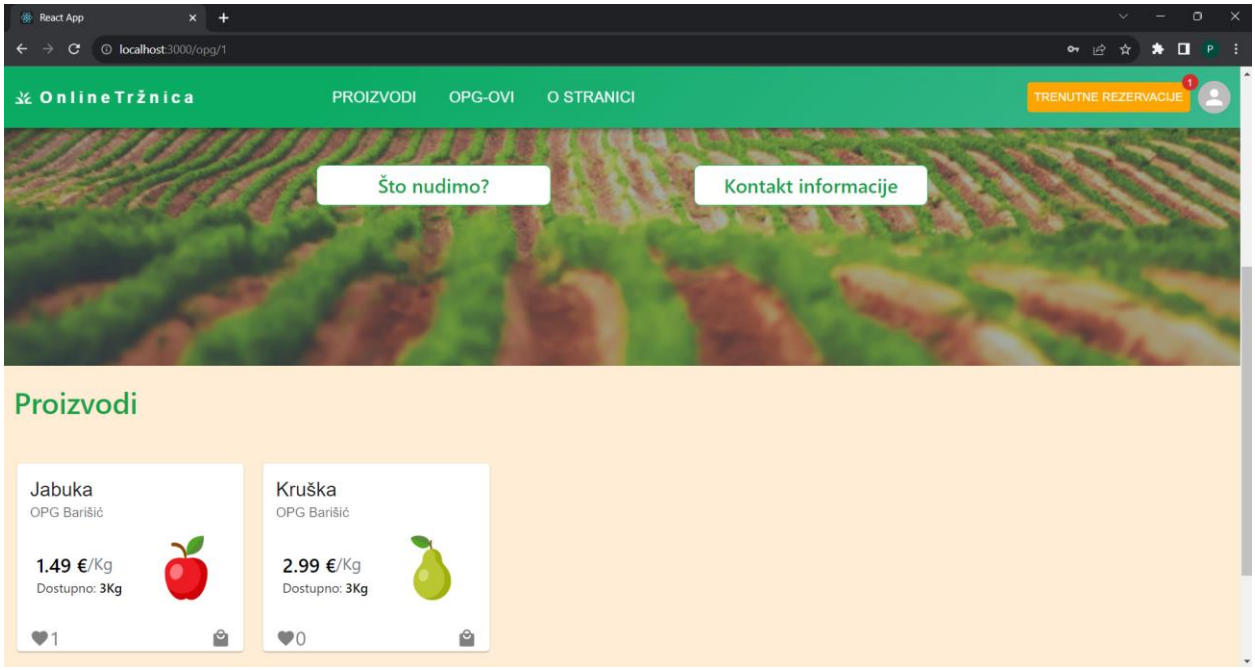
Slika 5.6. Prikaz OPG stranice

4. Korisnik odabire tipku „Što nudimo?“, stranica se spušta do sekcije „Proizvodi“ te korisnik odabire ikonicu „vrećice za kupnju“ kod željenog proizvoda za rezervaciju (Slika 5.7.).



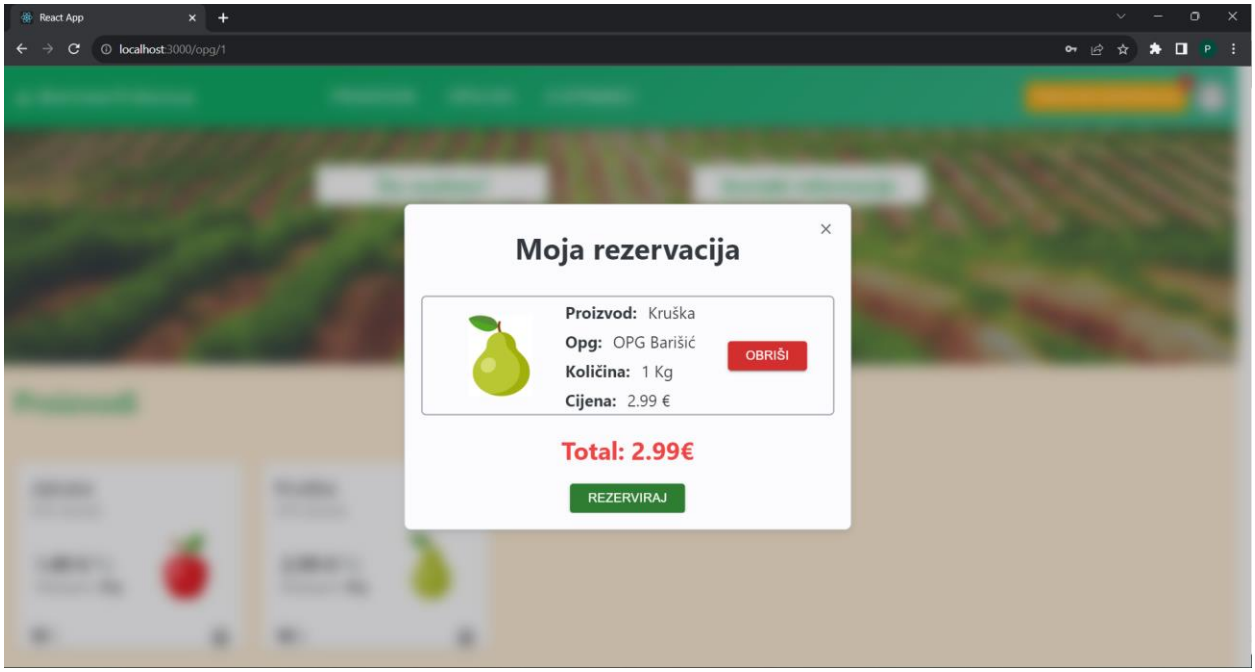
Slika 5.7. Prikaz sekcije „Proizvodi“ na OPG stranici

5. Korisnik odabire željenu količinu koristeći tipke „+“ ili „-“, odabire gumb dodaj te mu se na navigacijskoj traci, iznad tipke „trenutne rezervacije“, pojavljuje crveni kružić s brojem jedan, koji predstavlja trenutnu količinu željenih proizvoda za rezervaciju (Slika 5.8.)



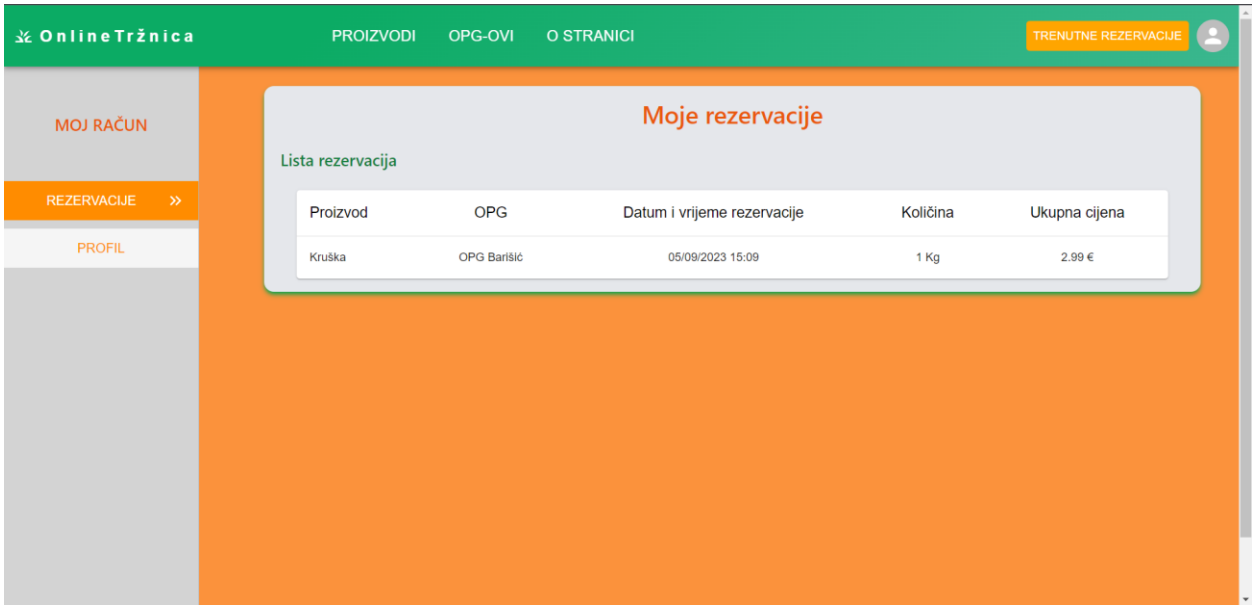
Slika 5.8. Prikaz OPG stranice i izbornika u trenutku odabira željenog proizvoda za rezervaciju

6. Korisnik odabire tipku „Trenutne rezervacije“, otvara se modal sa trenutnim željenim rezervacijama, ukupnom cijenom i tipkom „REZERVIRAJ“ (Slika 5.9.).



Slika 5.9. Prikaz modala „Moja rezervacija“

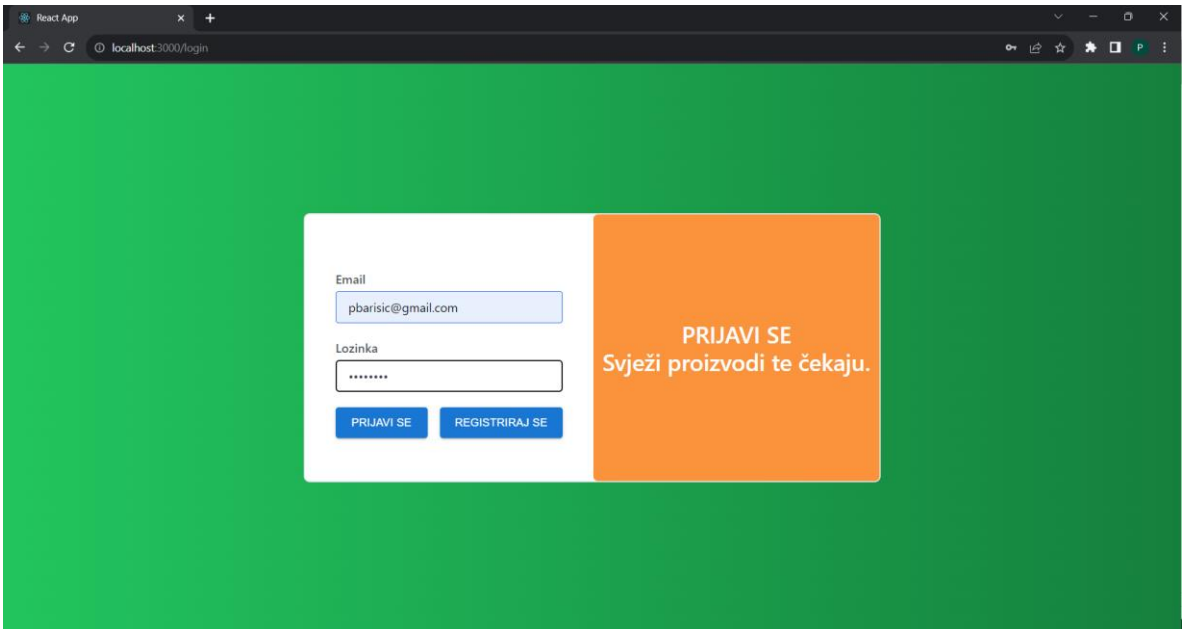
7. Korisnik odabire tipku „REZERVIRAJ“, prosljeđuje ga se na stranicu „MOJ RAČUN“ u sekciju „REZERVACIJE“, gdje se vidi rezervacija sa svim potrebnim detaljima u tablici te gumb „TRENUTNE REZERVACIJE“ više ne sadrži crveni kružić (Slika 5.10.).



Slika 5.10. Prikaz stranice „MOJ RAČUN“

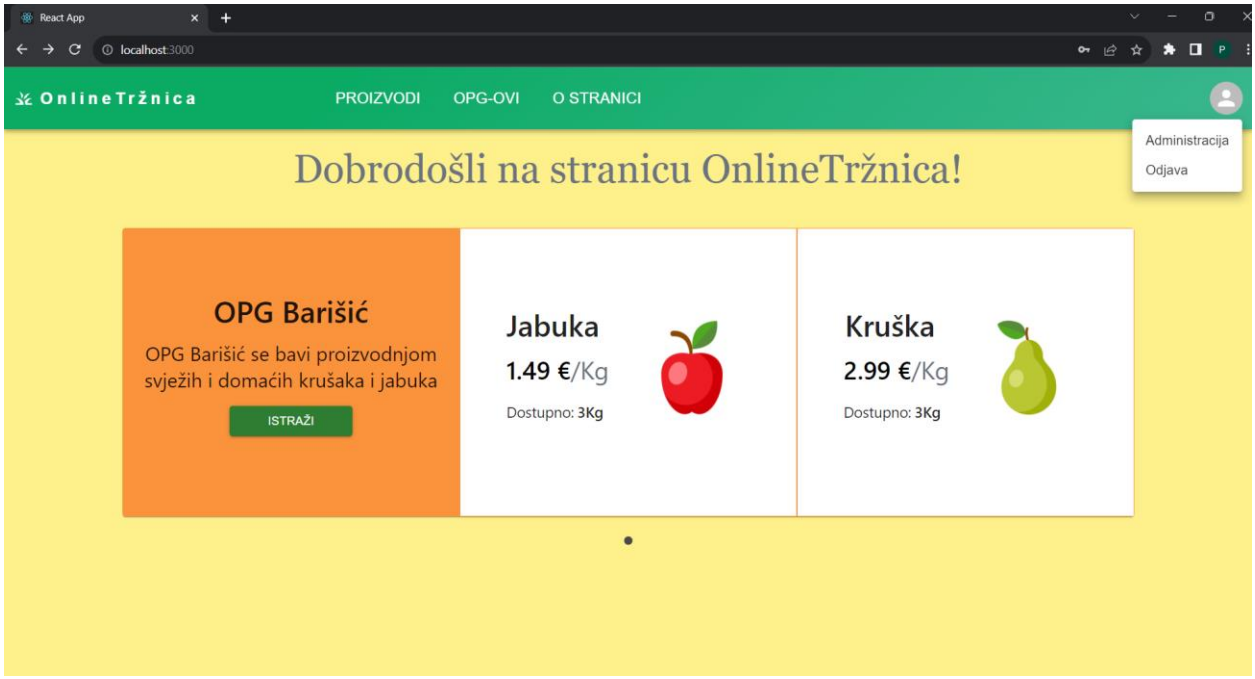
Administrator uspješno dodaje novu kategoriju i vrstu proizvoda za novu kategoriju:

1. Administrator uspješno popunjava polja za prijavu te se prijavljuje na mrežnu stranicu *OnlineTržnica* (Slika 5.11.).



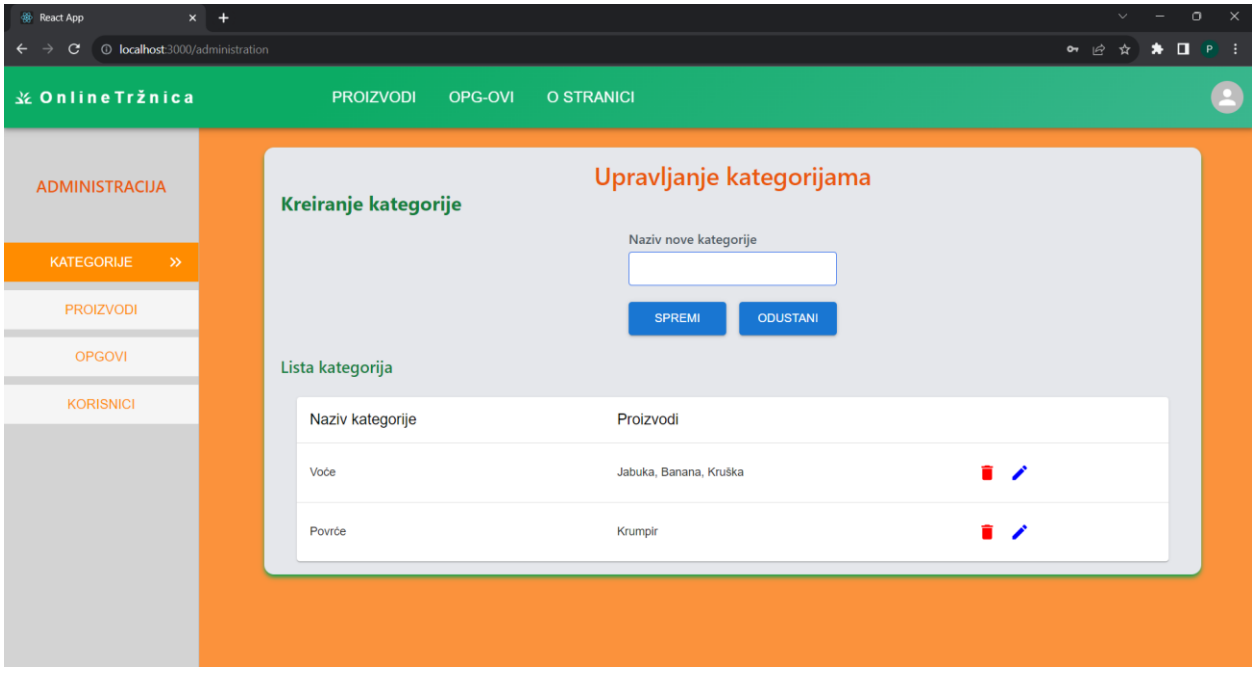
Slika 5.11. Prikaz forme za prijavu

2. Administrator se uspješno prijavio, proslijeđen je na početnu stranicu te odabire izbornik na navigacijskoj traci klikom na ikonu avatara (Slika 5.12.)



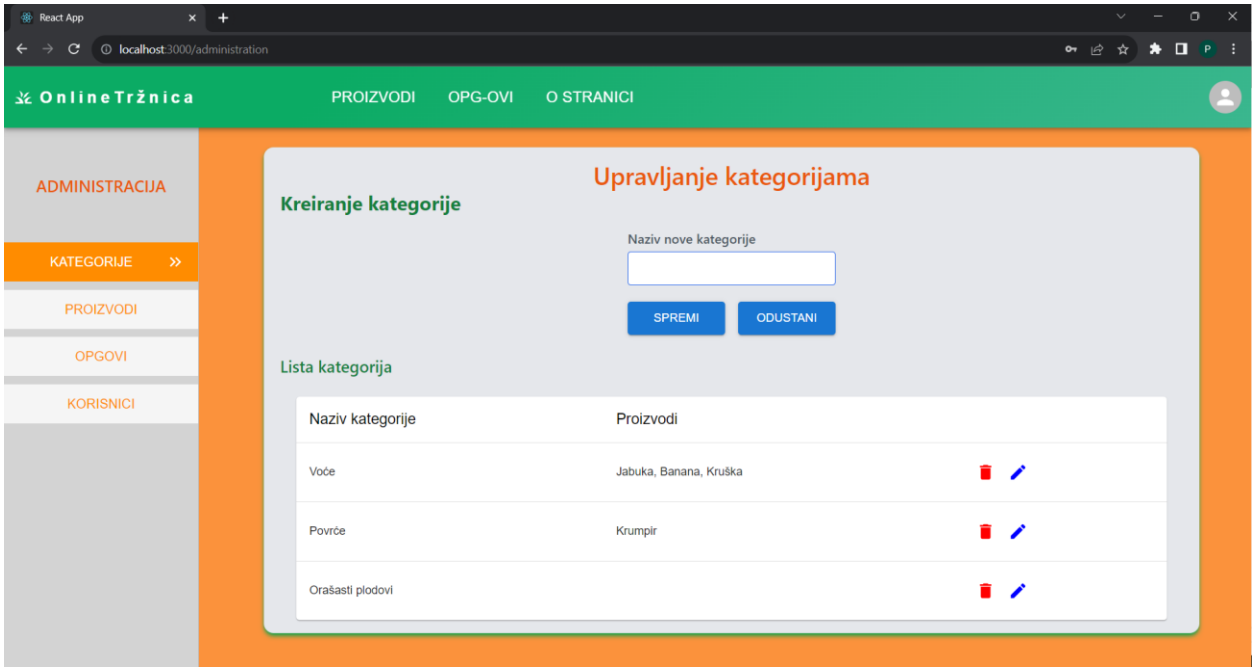
Slika 5.12. Prikaz početne stranice i izbornika

3. Administrator u izborniku odabire tipku „Administracija“ te ga se prosljeđuje na stranicu „ADMINISTRACIJA“ u sekciju „KATEGORIJE“ (Slika 5.13.)



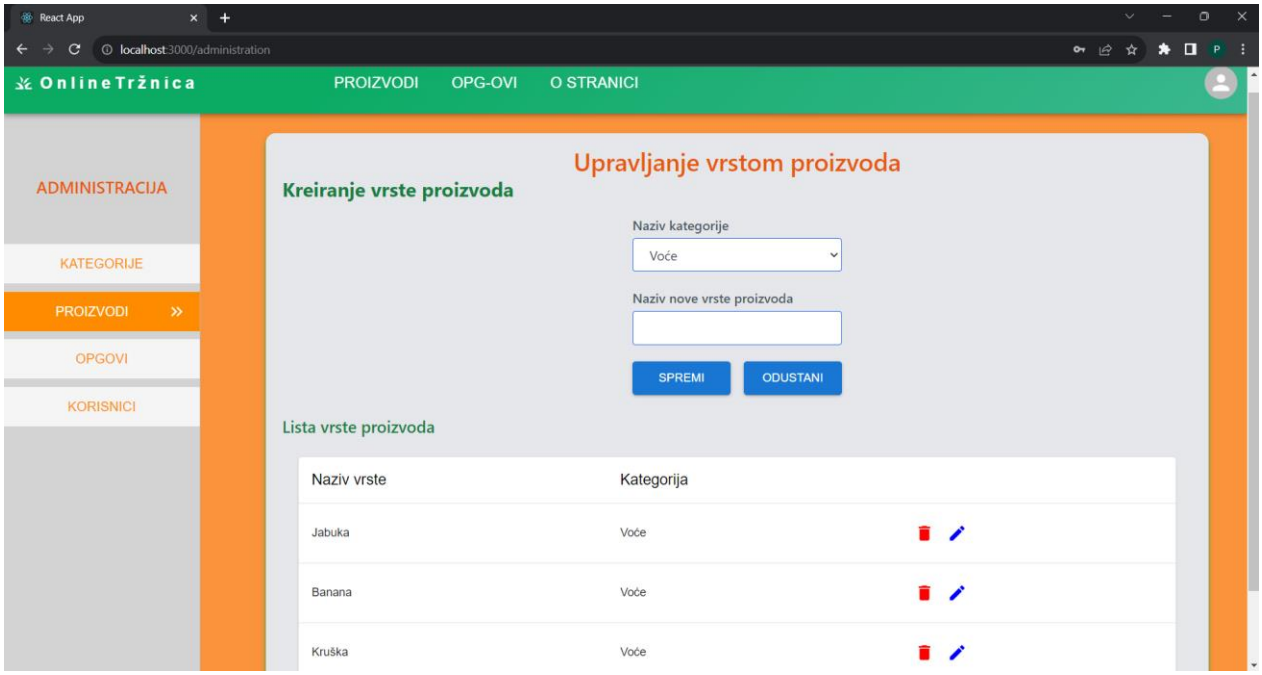
Slika 5.13. Prikaz stranice „ADMINISTRACIJA“

4. Administrator upisuje naziv nove kategorije, odabire tipku „SPREMI“ te se stranica spušta do mjesta u tablici gdje se upravo uspješno dodala nova kategorija (Slika 5.14.).



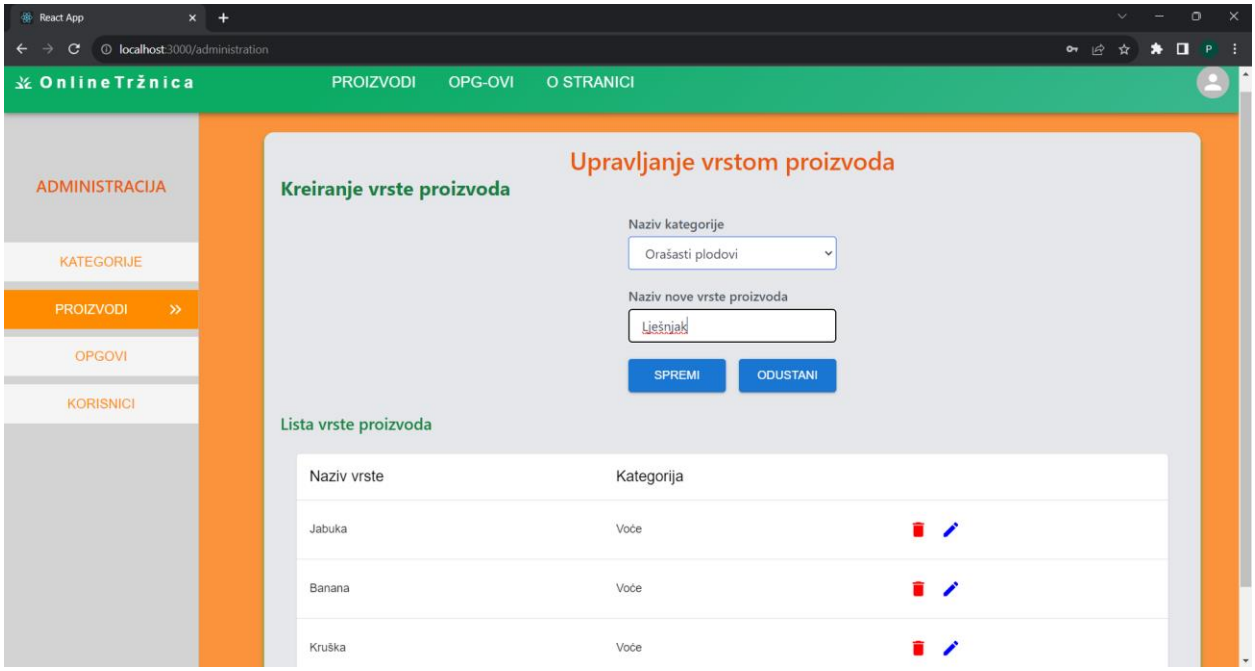
Slika 5.14. Prikaz nove kategorije u tablici

5. Administrator odabire tipku „PROIZVODI“ u izborniku koji se nalazi s lijeve strane te ga se prosljeđuje na sekciju „PROIZVODI“ (Slika 5.15.).



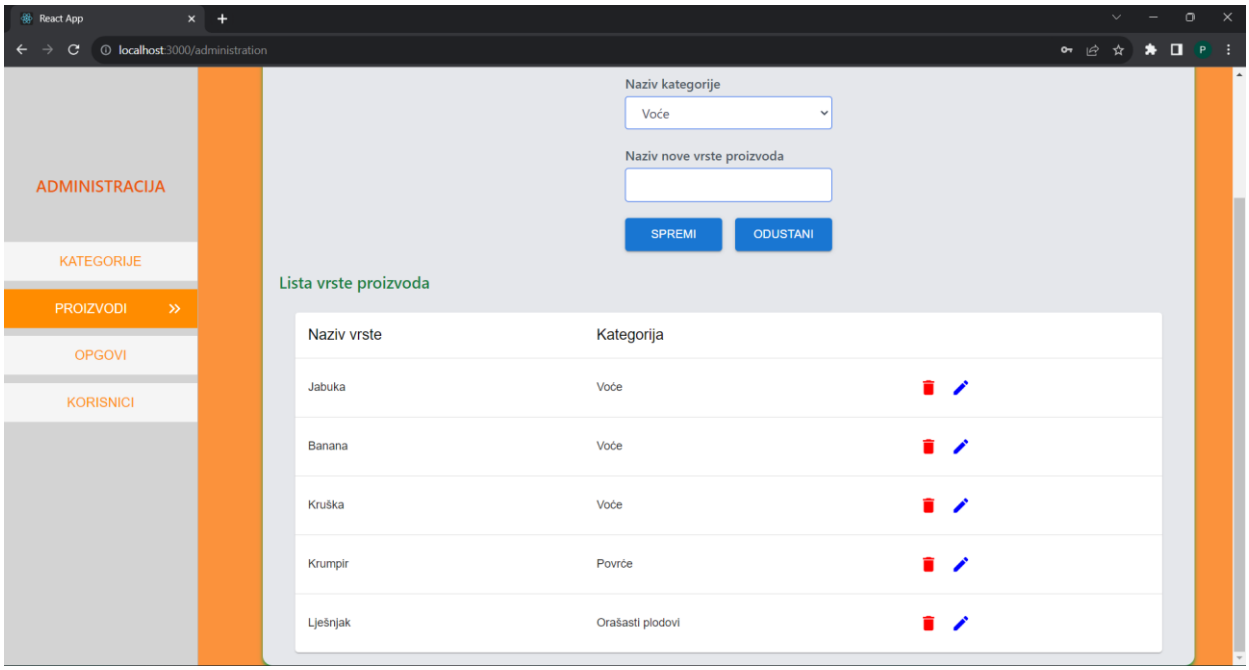
Slika 5.15. Prikaz sekcije „PROIZVODI“

6. Administrator odabire novu dodanu kategoriju u padajućem izborniku te upisuje naziv nove vrste proizvoda za novu kategoriju (Slika 5.16.).



Slika 5.16. Prikaz unosa nove vrste proizvoda

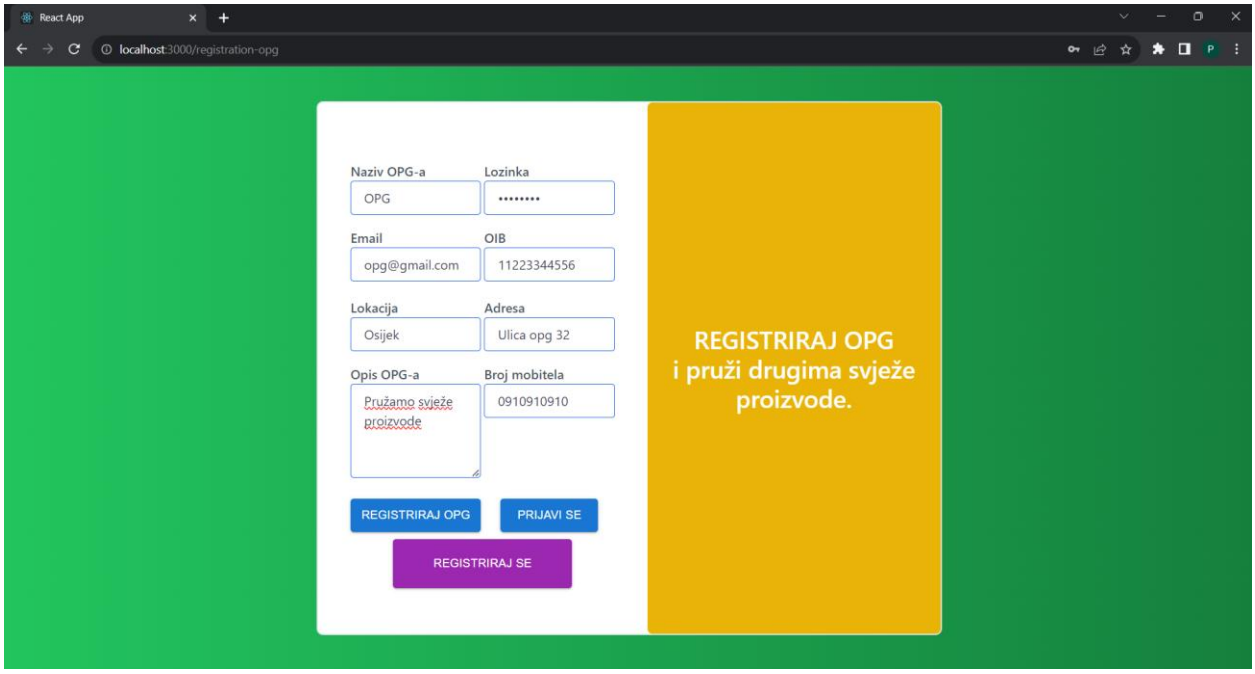
7. Administrator odabire tipku „SPREMI“, forma se vraća na početne vrijednosti te se stranica spušta do mjesta u tablici gdje se prikazuje nova vrsta proizvoda (Slika 5.17.).



Slika 5.17. Prikaz nove vrste proizvoda u tablici

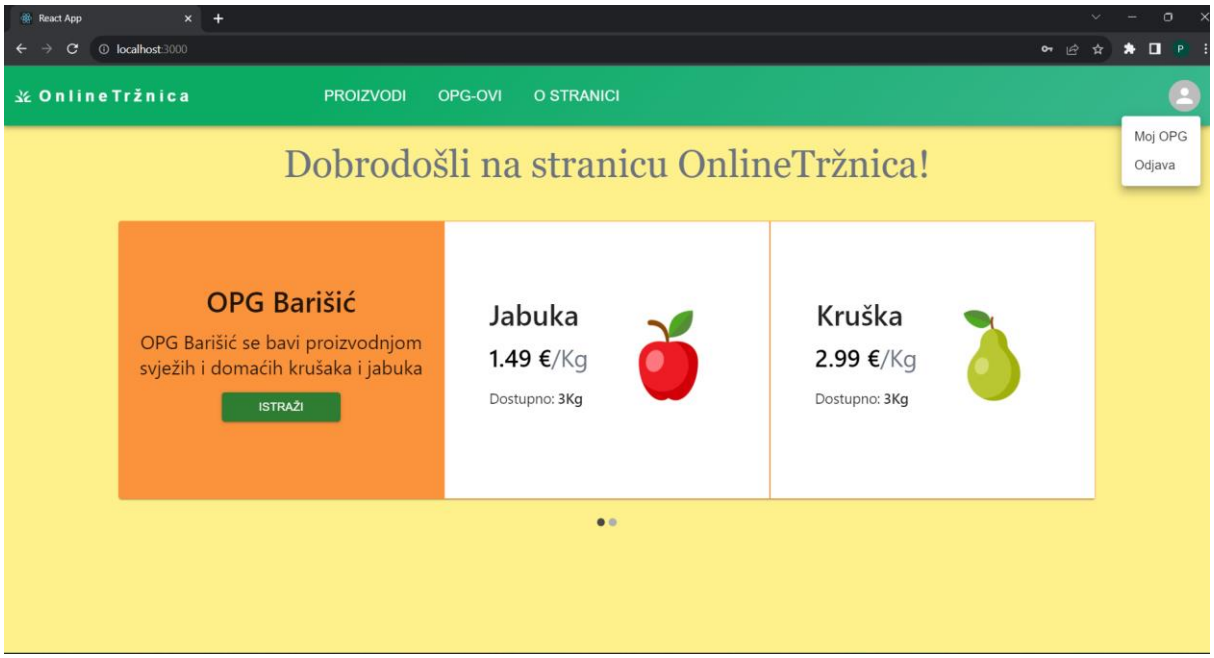
Prodavač dodaje novi proizvod u svoju ponudu:

1. Prodavač uspješno popunjava polja za registraciju OPG-a te se registrira na mrežnu stranicu *onlineTržnica* (Slika 5.18.).



Slika 5.18. Prikaz forme za registraciju OPG -a

2. Prodavač se uspješno registrirao, proslijeđen je na početnu stranicu te odabire izbornik na navigacijskoj traci klikom na ikonu avatara (Slika 5.19.).

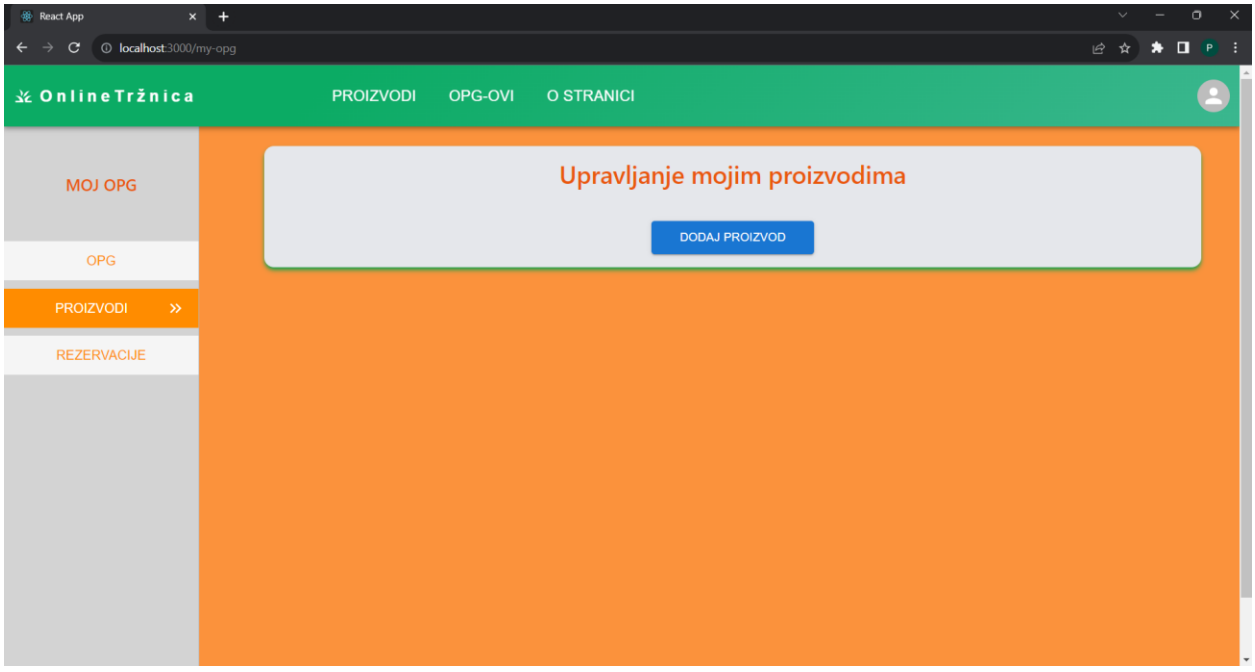


Slika 5.19. Prikaz početne stranice i izbornika za prodavača

3. Prodavač u izborniku odabire tipku „Moj OPG“ te ga se prosljeđuje na stranicu „MOJ OPG“ u sekciju „OPG“ (Slika 5.20.)

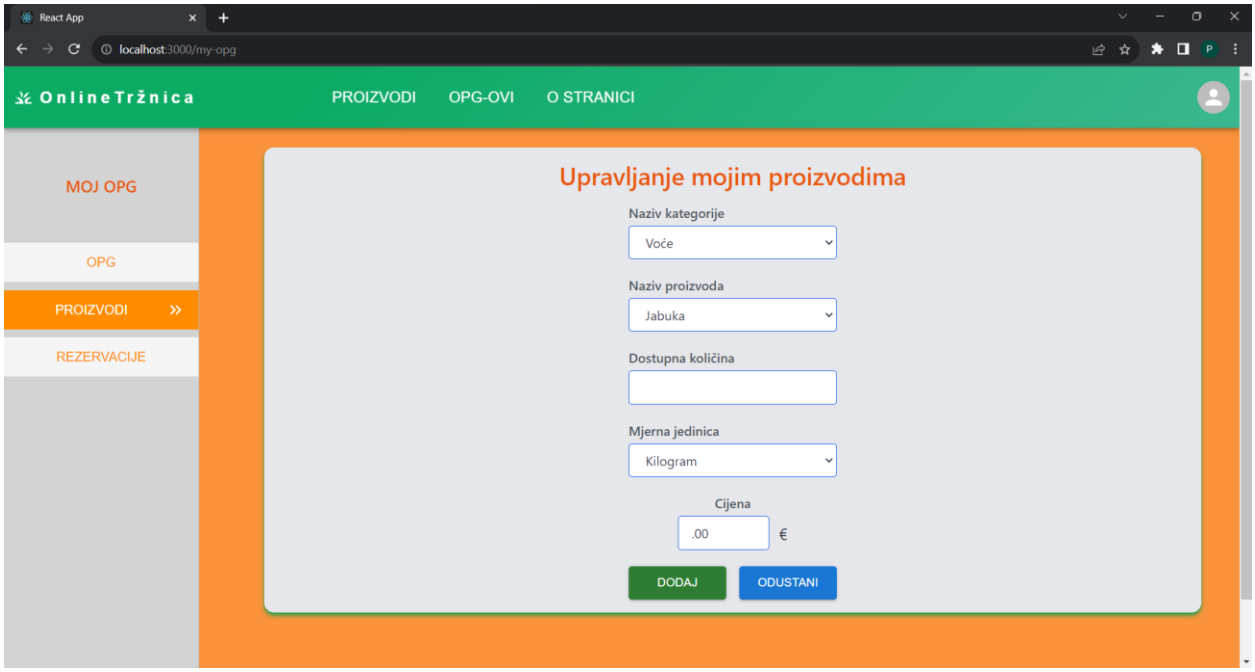
Slika 5.20. Prikaz stranice „MOJ OPG“

4. Prodavač odabire tipku „PROIZVODI“ u izborniku koji se nalazi s lijeve strane te ga se prosljeđuje na sekciju „PROIZVODI“ (Slika 5.21.).



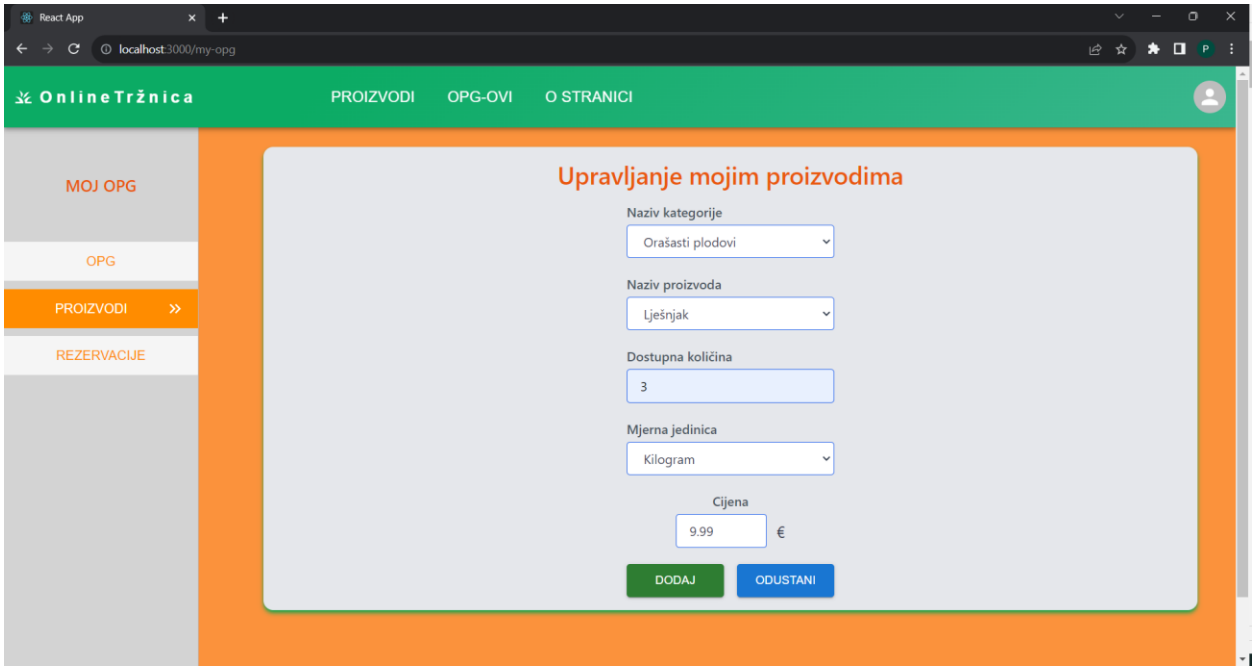
Slika 5.21. Prikaz sekcije „Proizvodi“

5. Prodavač odabire tipku „DODAJ PROIZVOD“ te ga se preusmjerava na formu za dodavanje novog proizvoda (Slika 5.22.).



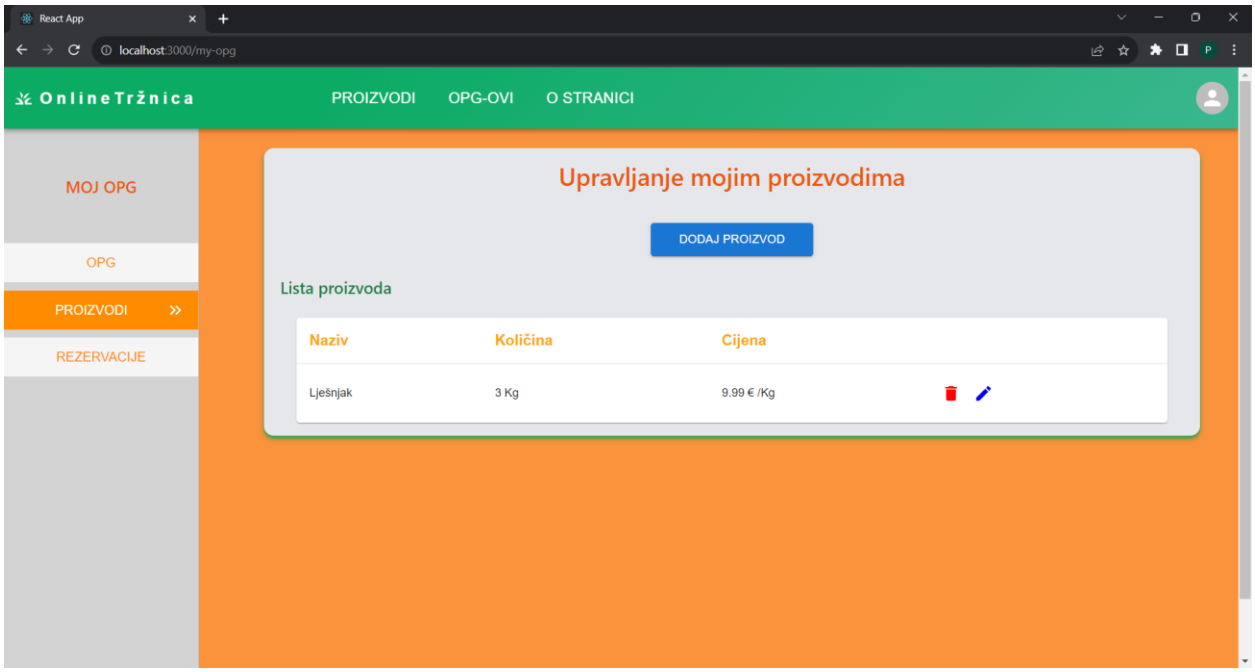
Slika 5.22. Prikaz forme za dodavanje proizvoda

6. Prodavač ispunjava formu željenim vrijednostima i odabire tipku „DODAJ“ (Slika 5.23.).



Slika 5.23. Prikaz ispunjene forme

7. Proizvođaču se prikazuje tablica s vrijednostima novog dodanog proizvoda (Slika 5.24.).



Slika 5.24. Prikaz liste proizvoda s novim dodanim proizvodom

6. ZAKLJUČAK

Mrežna aplikacija „OnlineTržnica“ nastala je kao rezultat naučenog kroz pet godina fakulteta i kroz praksu. Prikazuje komunikaciju između *frontend* dijela, napisanog u React.js i *backend* dijela, koji je napisan koristeći Java programski jezik i Spring Boot radni okvir. Aplikacija sadrži veliki broj funkcionalnosti kao što su: registracija i prijava korisnika, postojanje tri uloge (administrator, prodavač, korisnik), upravljanje korisnicima, tipovima i kategorijama proizvoda, kao i upravljanje samim proizvodima te kontrolu nad načinom kojim se pristupa podacima. No, ona ipak ne nudi mogućnost prodaje, kao postojeća rješenja, već nudi mogućnost rezervacije. Aplikacija bi se također mogla unaprijediti dodavanjem više validacija i validacijskih poruka, npr. kod registracije i promjene lozinke, popravljanjem same strukture koda raspodjelom više funkcionalnih cjelina u zasebne datoteke te dodavanjem mogućnosti učitavanje vlastite fotografije proizvoda. Zahvaljujući Spring Boot-u, implementacija programskoga koda na *backendu* je bila jednostavna i brza. Njegova kontrola nad servisima, kontrolerima i repozitorijima skratila je duljinu, ali i samo vrijeme pisanja programskoga rješenja za zadane probleme. Isto se može reći za React.js na *frontendu*. Mogućnost korištenja iste komponente više puta te korištenje drugih gotovih komponenti je također ubrzalo razvoju aplikacije. Uz pomoć TailwindCSS biblioteke, uspio se postići željeni cilj izgleda komponenata i stranice puno jednostavnije i preciznije nego korištenjem „čistog“ CSS-a. Stoga nije ni čudo što sve više firma današnjice, pogotovo u Hrvatskoj, koriste spomenute tehnologije za razvoj puno zahtjevnijih i ozbiljnijih rješenja.

LITERATURA

[1] Reducos j.d.o.o, „O nama“, dostupno na: https://plodovi.hr/o-nama/, pristupljeno: 17.6.2023.

[2] ePlac, „ePlac - O nama - Web tržnica i gradska tržnica“, dostupno na: https://eplac.eu/o-nama/, pristupljeno: 17.6.2023.

[3] Grad Zagreb, „O projektu Online Tržnice grada Zagreba“, dostupno na: https://online.trznice-zg.hr/o-projektu-online-trznice-grada-zagreba/7, pristupljeno: 17.6.2023.

[4] Institut za poljoprivredu i turizam Poreč, „O nama“, dostupno na: https://www.trznica-trg.eu/o-nama, pristupljeno: 17.6.2023.

[5] JetBrains s.r.o., „What is IntelliJ IDEA?“, dostupno na: https://www.jetbrains.com/idea/features/, pristupljeno: 20.6.2023.

[6] Meta Open Source, „Quick Start“, dostupno na: https://react.dev/learn, pristupljeno: 20.6.2023.

[7] Meta Platforms, Inc., „Virtual DOM and Internals“, dostupno na: https://legacy.reactjs.org/docs/faq-internals.html, pristupljeno: 20.6.2023.

[8] Meta Open Source, „Thinking in React“, dostupno na: https://react.dev/learn/thinking-in-react, pristupljeno: 20.6.2023.

[9] Microsoft, „The Basics“, dostupno na: https://www.typescriptlang.org/docs/handbook/2/basic-types.html, pristupljeno: 20.6.2023.

[10] Tailwind Labs Inc., „Utility-First Fundamentals“, dostupno na: https://tailwindcss.com/docs/utility-first, pristupljeno: 20.6.2023.

[11] IBM Corporation, „Advantages of Java“, dostupno na: https://www.ibm.com/docs/en/aix/7.1?topic=monitoring-advantages-java, pristupljeno: 24.6.2023.

[12] Microsoft, „What is Java Spring Boot?“, dostupno na: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-java-spring-boot/, pristupljeno: 24.6.2023.

[13] h2database, „Features“, dostupno na: https://www.h2database.com/html/features.html, pristupljeno: 24.6.2023.

[14] Liquibase Inc., „How Liquibase Works“, dostupno na: https://www.liquibase.com/how-liquibase-works, pristupljeno: 24.6.2023.

SAŽETAK

Ovaj rad prikazuje koje mogućnosti ima mrežna aplikacija „onlineTržnica“, a koje mogućnosti pružaju već postojeća rješenja. Opisane su tehnologije koje se koriste, struktura podataka s kojima aplikacija radi te su opisani ključni aspekti dizajna. Detaljno je pojašnjena implementacija na *backendu* te pripadajući funkcionalni testovi za servis. Također je pojašnjena implementacija na *frontendu* kao i pripadajući test korisničkog sučelja.

Ključne riječi: H2, Java, Liquibase, mrežna aplikacija, online tržnica, React.js, Spring Boot, TailwindCSS.

ABSTRACT

Web application for online marketplace

The thesis explains what possibilities the web application "onlineTržnica" will have, and what possibilities are provided by already existing solutions. It describes what technologies will be used, the data structure the application will work with, and the key design aspects are also described. The backend implementation and corresponding functional tests for the service are explained in detail. The frontend implementation as well as the related user interface test were also clarified.

Keywords: H2, Java, Liquibase, marketplace, React.js, Spring Boot, TailwindCSS, web appliacation.