Razvoj web aplikacije za naručivanje dostave

Bazzara, Josip

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

Razvoj web aplikacije za naručivanje dostave

Rijeka, srpanj 2024.

Josip Bazzara

0069090480

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

Razvoj web aplikacije za naručivanje dostave

Mentor: Izv. prof. dr. sc. Marko Gulić

Rijeka, srpanj 2024.

Josip Bazzara

0069090480

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 21.03.2024.

Zavod: Zavod za računarstvo
Predmet: Razvoj web aplikacija

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Josip Bazzara (0069090480)**
Studij: Sveučilišni prijediplomski studij računarstva (1035)

Zadatak: **Razvoj web aplikacije za naručivanje dostave / Development of a web application for ordering delivery**

Opis zadatka:

Razviti web aplikaciju za naručivanje dostave. Aplikacija mora imati implementiranu potpunu funkcionalnosti autentifikacije uz podržavanje različitih profila korisnika kao što su glavni administrator, administrator trgovine koja pruža dostavu, prijavljeni korisnik koji naručuje dostavu i prijavljeni dostavljač koji vrši dostavu. Nadalje, aplikacija mora imati funkcionalnosti dodavanja, brisanja i ažuriranja stavki unutar trgovine koju uređuje administrator trgovine. Nadalje, aplikacija treba imati implementiran cjelokupni proces narudžbe i dostave. Plaćanje mora biti izvršeno koristeći proizvoljno odabrani online servis za plaćanje. Za razvoj poslužiteljskog dijela web aplikacije treba koristiti Spring Boot radni okvir uz proizvoljno odabran sustav za upravljanje bazama podataka. Za razvoj klijentskog dijela aplikacije treba koristiti React JavaScript knjižicu za razvoj korisničkog sučelja uz korištenje MaterialUI React biblioteku za dizajn klijentskog dijela.

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

Zadatak uručen pristupniku: 22.03.2024.

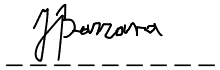
Mentor:
doc. dr. sc. Marko Gulić

Predsjednik povjerenstva za završni ispit:
prof. dr. sc. Miroslav Joler

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad

Rijeka, srpanj 2024.



Josip Bazzara

Zahvala

Zahvaljujem se mentoru doc. dr. sc. Marku Guliću na ukazanom povjerenju te pruženom vremenu za izradu ovoga rada. Također se zahvaljujem svojim prijateljima i obitelji na podršci koja mi je bila pružena tijekom studiranja.

SADRŽAJ

1 UVOD

Ovaj završni rad istražuje koncept e-trgovine kroz razvoj web aplikacije koja omogućuje korisnicima jednostavno obavljanje narudžbi, dok istovremeno pruža administratorske alate za efikasno upravljanje sustavom i poslovnim subjektima. Aplikacija zadovoljava korisničke zahtjeve mogućnosti izrade narudžbi iz više različitih poslovnih subjekata (restorana i trgovina), kao i mogućnost kartičnog plaćanja samih narudžbi. Osim toga zadovoljava potrebe administratora aplikacije time što im omogućuje kontrolu nad time koji će poslovni subjekti postojati u aplikaciji, koja će biti njihova ponuda, a također imaju uvid u sve korisnike koji imaju napravljeni račun unutar aplikacije. Naravno, uz administratora i same korisnike, u aplikaciji postoji i treća uloga koja predstavlja sve dostavljače, odnosne korisnike koji imaju uvid u kreirane narudžbe, te ih mogu preuzimati za dostavu i upravljati njima. Za sam razvoj aplikacije, na poslužiteljskoj strani korišten je Spring Boot [1] radni okvir, a na klijentskoj strani korišten je React [2]. Spring Boot [1] je radni okvir otvorenog koda za razvoj Java aplikacija. On olakšava izgradnju, konfiguraciju i upravljanje Java aplikacija putem automatskog konfiguriranja i integracije s popularnim alatima i bibliotekama. Kao što je navedeno, za razvoj sučelja odabran je React [2], također radni okvir otvorenog koda razvijen od strane Facebooka. React [2] se ističe svojom efikasnošću i jednostavnošću u izradi interaktivnih korisničkih sučelja. Za izgled stranica i komponenti korištena je programska biblioteka MaterialUI [3]. MaterialUI [3] programska biblioteka pruža širok spektar unaprijed definiranih komponenata poput tipografije, kartica, polja za unos, izbornika, dijaloških okvira itd. Ove komponente su prilagodljive i mogu se prilagoditi potrebama aplikacije kroz različite definirane opcije. Zbog svoje robusne sigurnosne funkcije i skalabilnosti te visokih performansa, PostgreSQL [8] je odabran kao relacijski sustav za upravljanje bazama podataka za ovu aplikaciju. Podržava kompleksne SQL upite, transakcije, replikaciju i razne vrste indeksa, što ga čini pogodnim za širok spektar primjena, od malih web aplikacija do velikih korporativnih sustava. Njegova fleksibilnost i podrška za standarde čine ga popularnim izborom među programerima i administratorima baza podataka širom svijeta. Osim navedenih tehnologija, važno je napomenuti da je za razvoj aplikacije korištena monolitna arhitektura koja pruža jednostavnost u razvoju i održavanju, što je vrlo pogodno za manje i/ili srednje kompleksne projekte poput ovoga. Također, komunikacija između klijenta i poslužitelja implementirana je korištenjem standarda poput HTTPS protokola, kao i implementacijom

autorizacije i autentifikacije na razini korisnika i administrativnih prava, te se na taj način osigurava zaštita osjetljivih podataka unutar sustava. U ovom radu detaljno će biti opisane korištene tehnologije, sve funkcionalnosti aplikacije kao i opis svih korisničkih sučelja. Također, u ovom radu nalazi se opis funkcionalnosti autorizacije i autentifikacije korisnika na razini programskog koda što aplikaciju čini sigurnom i pouzdanom. Uz autorizaciju i autentifikaciju, u ovom radu nalazi se i detaljan opis implementacije kartičnog plaćanja na Stripe platformi na razini programskog koda.

2 TEHNOLOGIJE

2.1 Java

Java [12] je objektno orijentirani programski jezik i računalna platforma koju je razvila tvrtka Sun Microsystems, a kasnije je preuzela Oracle Corporation. Poznata po svojoj prenosivosti, Java omogućava razvoj aplikacija koje mogu raditi na različitim uređajima i operativnim sustavima bez potrebe za ponovnim prevođenjem koda, zahvaljujući principu "Write Once, Run Anywhere" (WORA). Jezik je dizajniran s naglaskom na sigurnost, stabilnost i visoke performanse, te se koristi za razvoj raznovrsnih aplikacija, od web aplikacija i mobilnih aplikacija do *enterprise* sustava i ugrađenih sustava. *Java Virtual Machine* (JVM) omogućava izvršavanje Java programa, pružajući izolirano okruženje koje povećava sigurnost i pouzdanost aplikacija. Java također nudi bogatu standardnu biblioteku koja olakšava razvoj i smanjuje vrijeme potrebno za izradu aplikacija. Široka podrška zajednice, veliki broj dostupnih knjižnica i alata, te kontinuirana evolucija jezika čine Javu jednim od najpopularnijih i najkorisnijih programskih jezika u industriji softverskog razvoja.

2.2 Spring Boot

Spring Boot [1] je radni okvir otvorenog koda za razvoj Java aplikacija koji pruža jednostavan način za izgradnju, konfiguriranje i upravljanje Java aplikacijama. On se temelji na Spring Frameworku, ali dolazi s ugrađenim alatima i konvencijama koje olakšavaju brži razvoj aplikacija.

Što se tiče slojevite arhitekture, Spring Boot [1] podržava organizaciju aplikacija u različite slojeve kako bi se omogućila bolja modularnost, održavanje i skalabilnost. Tipična slojevita arhitektura u Spring Boot [1] aplikacijama uključuje:

1. **Sloj kontrolera (eng. Controller Layer):** Ovdje se nalaze kontroleri (eng. controllers) koji obrađuju zahtjeve klijenata. Oni komuniciraju s korisničkim sučeljem (npr. web sučelje) i proslijeđuju zahtjeve servisnom sloju.
2. **Sloj servisa (eng. Service Layer):** Servisi sadrže poslovnu logiku aplikacije. Oni obavljaju obradu podataka, provode poslovna pravila i komuniciraju s repozitorijima radi pristupa podacima. Servisi se često koriste za poslovnu logiku koja je zajednička više kontrolerima. Na slici 2.1 prikazan je primjer

jedne metode u servisnom sloju, sloju gdje se nalazi sva poslovna logika aplikacije.

```
private List<OrderResponse> mapOrderResponses(List<Order> orders) {
    List<OrderResponse> orderResponses = new ArrayList<>();
    orders.forEach(order -> {
        var orderResponse = new OrderResponse();
        orderResponse.setId(order.getId());
        orderResponse.setPhoneNumber(order.getUserPhoneNumber());
        orderResponse.setLocation(order.getUserLocation());
        orderResponse.setTotalPrice(order.getTotalPrice());
        orderResponse.setMarketParticipantId(order.getMarketParticipant().getId());
        orderResponse.setMarketParticipantName(order.getMarketParticipant().getName());
        orderResponse.setUserOwner(order.getOrderOwner().getId());
        orderResponse.setOrderStatus(order.getOrderStatus().toString());
        orderResponse.setItems(order.getItemList().stream()
                .map(itemService::mapToItemResponse).collect(Collectors.toList()));

        orderResponses.add(orderResponse);
    });

    return orderResponses;
}
```

Slika 2.1 - Primjer metode u Servisnom sloju

3. **Sloj repozitorija (eng. Repository Layer):** Ovaj sloj sadrži repozitorije koji služe za komunikaciju s baza podataka. Repozitoriji pružaju apstrakciju nad slojem za pristup podacima (eng. data access layer), omogućujući programerima jednostavan pristup podacima bez detaljnog rukovanja baza podataka.

4. **Sloj podataka (eng. Data Layer):** Ovdje se nalaze entiteti (eng. entites) koji predstavljaju strukturu podataka aplikacije. Entiteti se mapiraju na tablice u bazi podataka i koriste se za pohranu i manipulaciju podacima.

Ova slojevita arhitektura omogućuje jasnu organizaciju koda, olakšava testiranje i održavanje aplikacija te podržava princip razdvajanja odgovornosti (eng. Separation of Concerns), što rezultira boljom modularnošću i skalabilnošću aplikacija.

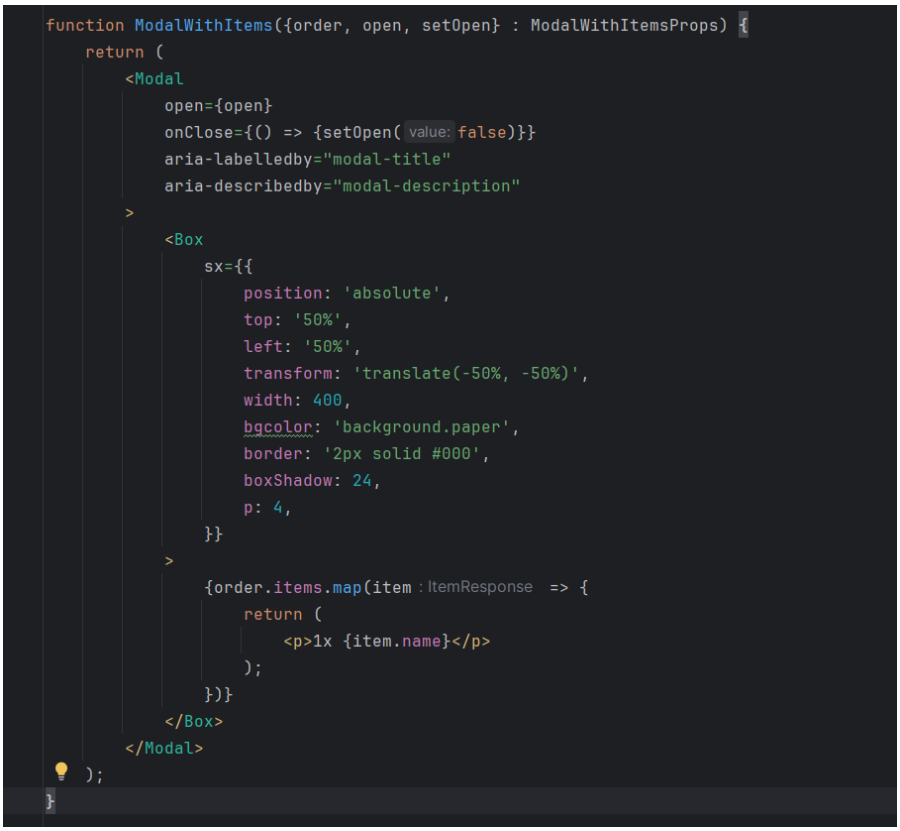
2.3 REST

REST (Representational State Transfer) [13] je arhitektonski stil za izgradnju skalabilnih i interoperabilnih web servisa. Temelji se na standardnim HTTP metodama kao što su *GET*, *POST*, *PUT* i *DELETE* za izvršavanje operacija nad resursima, koji su predstavljeni

u obliku URI-ova (Uniform Resource Identifiers). U *RESTful* arhitekturi, resursi se prenose u različitim formatima, najčešće kao *JSON* ili *XML*, čime se omogućava lakša integracija i komunikacija između klijentskih i serverskih aplikacija. REST promiče korištenje *stateless* komunikacije, što znači da svaki zahtjev od klijenta prema serveru mora sadržavati sve potrebne informacije za razumijevanje i obradu zahtjeva, bez potrebe za zadržavanjem stanja na serveru. Popularnost REST-a leži u njegovoj jednostavnosti, učinkovitosti i standardizaciji, što ga čini ključnim elementom u dizajnu API-ja i integraciji aplikacija.

2.4 React

React [2] je JavaScript radni okvir koji se koristi za izradu interaktivnih korisničkih sučelja u web aplikacijama. Razvijen od strane Facebooka, React [2] je postao jedan od najpopularnijih alata među web developerima zbog svoje efikasnosti, jednostavnosti i performansi. Ključna karakteristika Reacta [2] je koncept komponenata, modularnih i ponovno upotrebljivih jedinica sučelja koje olakšavaju organizaciju koda i održavanje aplikacija. Na slici 2.2 prikazan je primjer jedne komponente u Reactu [2] korištene u razvoju ove aplikacije.



Slika 2.2 - Primjer React komponente

Jedan od ključnih koncepta u Reactu [2] je Virtual DOM (Document Object Model), tehnika koja omogućava brzo ažuriranje prikaza sučelja. Umjesto direktnog mijenjanja stvarnog DOM-a, React [2] stvara virtualnu reprezentaciju DOM-a u memoriji, uspoređuje je s prethodnom verzijom i ažurira samo dijelove koji su se promijenili, što rezultira boljom performansom aplikacije. React [2] promovira jednosmjerno podatkovno vezanje (engl. one-way data binding), što znači da se podaci prenose kroz komponente odozdo prema gore (engl. "downward data flow"). To olakšava praćenje podataka i njihovo upravljanje u aplikaciji, čime se smanjuje mogućnost neočekivanih efekata zbog nuspojava u aplikaciji.

Jedna od prednosti Reacta [2] je *JSX* (JavaScript XML), proširenje JavaScript sintakse koje omogućuje pisanje HTML-a unutar JavaScript koda. Ovo olakšava pisanje komponenti jer omogućuje kombiniranje JavaScript logike s HTML strukturom na intuitivan način.

React [2] zajednica razvija i održava bogat ekosustav dodataka, biblioteka i alata koji olakšavaju razvoj web aplikacija. S obzirom na veliku popularnost i široku podršku, React [2] je postao standardni izbor za razvoj modernih web aplikacija. Uz to, React [2] je dobro dokumentiran, s obiljem resursa za učenje i podršku, što ga čini pristupačnim čak i za početnike u *web developmentu*.

2.5 Typescript

TypeScript [14] je programski jezik otvorenog koda koji je razvila tvrtka Microsoft, a koji nadograđuje JavaScript dodavanjem statičkog tipiziranja. Kao nadskup JavaScripta, *TypeScript* omogućava korištenje svih postojećih JavaScript biblioteka i koda, dok istovremeno donosi dodatne mogućnosti za bolju organizaciju i održavanje velikih kodnih baza. Statičko tipiziranje omogućava rano otkrivanje grešaka tijekom razvoja, poboljšavajući pouzdanost i stabilnost aplikacija. *TypeScript* također uvodi napredne značajke kao što su klase, sučelja i moduli, što olakšava pisanje objektno orijentiranog koda i promovira ponovnu upotrebu koda. Zbog kompatibilnosti s *ECMAScript* standardima, *TypeScript* se brzo prilagođava novim JavaScript specifikacijama, osiguravajući da programeri mogu koristiti najnovije značajke jezika. Alati poput Visual Studio Codea pružaju snažnu podršku za *TypeScript*, uključujući autokompletiranje,

refaktoring i navigaciju kodom. Kao rezultat, *TypeScript* je postao popularan izbor među razvojnim timovima koji žele izgraditi skalabilne, održive i robusne web aplikacije.

2.6 MaterialUI

MaterialUI [9] je popularna biblioteka komponenata za React [2], temeljena na Googleovom dizajn sistemu Material Design. Ova biblioteka pruža širok spektar predefiniranih komponenata kao što su tipografija, kartice, polja za unos, izbornici, dijaloški okviri i mnoge druge. MaterialUI [9] biblioteka objavljena je pod MIT licencom. Pošto je MaterialUI [9] biblioteka otvorenog koda po MIT licencom, MaterialUI [9] dopušta korištenje biblioteke u komercijalne i privatne svrhe bez dodatnih troškova, slobodnu modifikaciju izvornog koda kao i mogućnost daljnje distribucije izmijenjenog izvornog koda.

Jedna od ključnih karakteristika MaterialUI-a je njegova prilagodljivost i mogućnost prilagođavanja komponenata potrebama aplikacije putem različitih definiranih opcija. Također, MaterialUI nudi konzistentan i moderan izgled komponenata, što olakšava razvoj vizualno privlačnih korisničkih sučelja.

Kroz korištenje MaterialUI komponenti, programeri mogu ubrzati proces razvoja aplikacija jer imaju pristup već izgrađenim i testiranim komponentama koje su spremne za upotrebu. Osim toga, MaterialUI ima aktivnu zajednicu korisnika i razvijatelja te pruža obilje resursa za učenje i podršku, što dodatno olakšava korištenje ove biblioteke u razvoju web aplikacija.

2.7 React Auth Kit

React Auth Kit [4] je biblioteka koja olakšava implementaciju sustava za autentikaciju i autorizaciju na klijentskoj strani. Pruža gotove komponente i metode za upravljanje korisničkim sesijama, provjeru prava pristupa i zaštitu ruti. Ova biblioteka pomaže programerima da brzo i jednostavno integriraju sigurnosne funckionalnosti u svoje React [2] aplikacije, čime olakšava razvoj sigurnih web aplikacija.

Na početnom ekranu pri ulasku u aplikaciju, korisniku je prikazan ekran za ulazak u aplikaciju s mogučnošću prijave ili pak za kreiranje novog korisničkog računa.

Ukoliko korisnik odabere opciju prijave, kombinacijom unosa ispravne e-mail adrese te lozinke koje je postavio prilikom kreiranja računa, korisnik ulazi u aplikaciju te može pristupati svim ostalim ekranima.

2.8 Stripe

Stripe [5] je platforma za online plaćanja koja omogućava developerima integraciju raznih platnih usluga u njihove web aplikacije. Osnovana 2010. godine, Stripe platforma pruža API koji olakšava obradu kreditnih kartica, upravljanje pretplatama, rukovanje fakturama i sprječavanju prevarama. Njegova jednostavnost i robusna dokumentacija omogućuju brzu implementaciju, što ga čini idealnim izborom za sve aplikacije koje pružaju nekakvu vrstu naplativih usluga. Osim osnovnih platnih funkcionalnosti, Stripe podržava međunarodne transakcije, omogućavajući poslovanje na globalnoj razini. Integracija sa Stripeom omogućava web aplikacijama sigurno i efikasno upravljanje financijskim transakcijama, čime se štedi vrijeme i resursi potrebni za razvoj vlastitih rješenja.

2.9 PostgreSQL

PostgreSQL [8] je napredni objektno relacijski sustav za upravljanje bazama podataka (ORDBMS) otvorenog koda, poznat po svojoj stabilnosti, pouzdanosti i bogatstvu značajki. Razvijen je s naglaskom na sukladnost s SQL standardima, te nudi podršku za složene upite, replikaciju podataka i punu ACID sukladnost za transakcijsku sigurnost. ACID (eng. Atomicity, Consistency, Isolation, Durability) jest akronim koji se referencira na četiri svojstva koje svaka PostgreSQL transakcija mora podržavati. To su atomičnost, dosljednost, izolacija i trajnost. PostgreSQL [8] omogućava korištenje naprednih funkcionalnosti kao što su složeni tipovi podataka, JSON podrška, indeksiranje putem B-drveća i GiST indeksa, te snažni alati za kontrolu pristupa i sigurnost. Njegova arhitektura omogućava skalabilnost i učinkovito rukovanje velikim količinama podataka, čineći ga pogodnim za širok spektar aplikacija, od malih projekata do velikih sustava za analitiku i obradu podataka. PostgreSQL [8] zajednica kontinuirano doprinosi njegovom razvoju, osiguravajući redovita ažuriranja, popravke i dodatne značajke koje unapređuju performanse i korisničko iskustvo. Zbog svojih karakteristika i mogućnosti prilagodbe, PostgreSQL [8] se koristi u raznim industrijama, uključujući financije, zdravstvo, obrazovanje i tehnologiju, kao jedan od najpouzdanijih sustava za upravljanje bazama podataka na tržištu.

2.10 Docker

Docker [10] je platforma otvorenog koda koja omogućava razvoj, isporuku i pokretanje aplikacija unutar kontejnera. Kontejneri su lagani, prenosivi i konzistentni okviri koji sadrže sve potrebne komponente za pokretanje aplikacije, uključujući kod, biblioteke, okruženja za izvršavanje i sistemske alate. Ova tehnologija omogućava programerima da osiguraju da njihove aplikacije rade u bilo kojem okruženju, bilo da se radi o lokalnom računalu, podatkovnom centru ili oblaku, eliminirajući probleme povezane s razlikama u konfiguracijama sustava. Docker poboljšava agilnost i učinkovitost razvoja softvera, omogućavajući brzu i jednostavnu kreaciju, testiranje i implementaciju aplikacija. Njegova modularna priroda olakšava skaliranje aplikacija i integraciju s DevOps praksama, pružajući automatizaciju i orkestraciju kroz alate poput Docker Compose i Docker Swarm. Uz široku podršku zajednice i integraciju s drugim alatima za razvoj i upravljanje aplikacijama, Docker je postao ključan alat za moderne razvojne i operativne timove koji žele optimizirati svoje procese isporuke softvera.

2.11 Node packet manager

Node Package Manager (npm) [15] je alat za upravljanje paketima za JavaScript, koji je integralni dio Node.js okruženja. Npm omogućava programerima da lako preuzimaju, instaliraju, upravljaju i dijele pakete koda, olakšavajući razvoj aplikacija. S više od milijun paketa dostupnih u svojem registru, npm pruža širok spektar funkcionalnosti koje pokrivaju gotovo svaki aspekt razvoja softvera, od korisničkih sučelja do backend servisa. Npm također omogućava jednostavno upravljanje ovisnostima projekta, osiguravajući da svi potrebni paketi i njihove verzije budu dosljedno instalirani u razvojnom okruženju. Korištenje npm-a olakšava kolaboraciju među timovima, jer omogućava dijeljenje vlastitih paketa unutar organizacije ili s javnošću. Njegove značajke, poput semantičkog verzioniranja i skripti za automatizaciju, dodatno unapređuju produktivnost i održavanje projekata. Kao rezultat, npm je postao ključan alat za razvoj moderne JavaScript aplikacije, podržavajući brži i učinkovitiji razvojni proces.

2.12 Postman

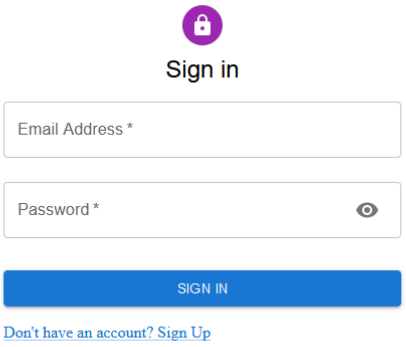
Postman [11] je alat za razvoj i testiranje API-ja koji omogućava programerima jednostavno slanje HTTP zahtjeva i pregledavanje odgovora. Pruža intuitivno korisničko sučelje za stvaranje, testiranje i dokumentiranje RESTful i SOAP API-ja, što ubrzava

proces razvoja i poboljšava suradnju unutar timova. Postman omogućava korisnicima da definiraju različite tipove zahtjeva, uključujući GET, POST, PUT i DELETE, te dodaju parametre, zaglavlja i tijela zahtjeva. Osim osnovnih funkcionalnosti, Postman podržava izradu automatiziranih testova pomoću JavaScripta, omogućavajući integraciju testiranja u CI/CD procese. Kolekcije zahtjeva, koje se mogu dijeliti s timovima, olakšavaju organizaciju i ponovnu upotrebu testnih scenarija. Također, Postman nudi napredne značajke poput praćenja API-ja, izrade detaljnih dokumentacija i simulacije API okruženja putem Postman Mock Servera. Njegova sposobnost integracije s različitim alatima i platformama čini ga nezaobilaznim alatom za moderne razvojne timove koji žele osigurati kvalitetu i pouzdanost svojih API-ja.

3 OPIS RADA APLIKACIJE ZA NARUČIVANJE DOSTAVE

3.1 Početni ekran (autentifikacija)

Za pristup aplikaciji, od korisnika se zahtjeva da ima kreiran korisnički račun te da je prijavljen u isti, odnosno da ima aktivnu sesiju s poslužiteljem. Ukoliko to nije ostvareno, odnosno ako na klijentskoj strani ne postoji poseban token izdan od strane poslužitelja, prvi ekran koji korisnik vidi jest ekran za prijavu, odnosno registraciju korisnika. Isječak ekrana sa formom za prijavu prikazan je na slici 3.1. Osim polja za upis elektroničke pošte i lozinke koji su korisniku potrebni za ulazak u aplikaciju, na formi za prijavu dostupan je gumb s tekstom *SIGN IN* (prijavi se), na čiji klik korisnik potvrđuje svoju prijavu, te ispod njega nalazi se link koji, ukoliko korisnik nema kreiran korisnički račun, odnosno prvi put koristi aplikaciju, vodi do novog ekrana koji sadrži formu za kreiranje novih korisničkih računa.



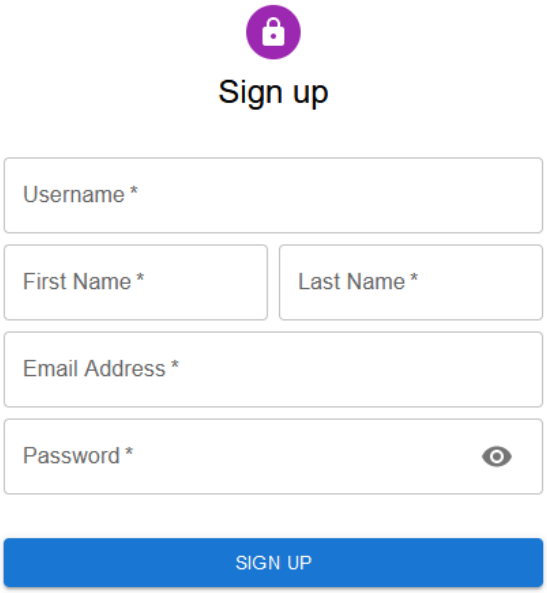
Slika 3.1 - Forma za prijavu korisnika

Na slici su prikazana dva polja u formi, prvo polje koje predstavlja elektroničku poštu korisnika koju je naveo prilikom kreiranja svog računa, te drugo polje koje predstavlja lozinku koju je također korisnik naveo prilikom kreiranja svog računa. Na slici 3.2 prikazano je polje za unos lozinke nakon što korisnik unese niz znakova gdje se vidi da su znakovi sakriveni kako bi se korisnik osigurao da treće lice neće moći vidjeti lozinku prilikom svoje prijave.

Slika 3.2 - Polje za lozinku sa skrivenim znakovima

Klikom miša na ikonicu u obliku oka koja se nalazi na samom polju za lozinku, korisniku se lozinka prikaže, što korisniku daje mogućnost garancije da je upisao točnu kombinaciju znakova koji predstavljaju lozinku prilikom prijave.

Dakle, osim same prijave, korisnik može odabrati i registraciju svog računa gdje se otvara novi ekran s formularom gdje može unesti sve potrebne informacije za kreiranje novog računa. Osim lozinke i e-mail adrese, prilikom kreiranja računa od korisnika se zahtjeva da unese svoje ime i prezime kao i *nadimak* koji će koristiti prilikom korištenja aplikacije. Na slici 3.3 može se vidjeti ekran koji služi kao formular za kreiranje korisničkog računa.

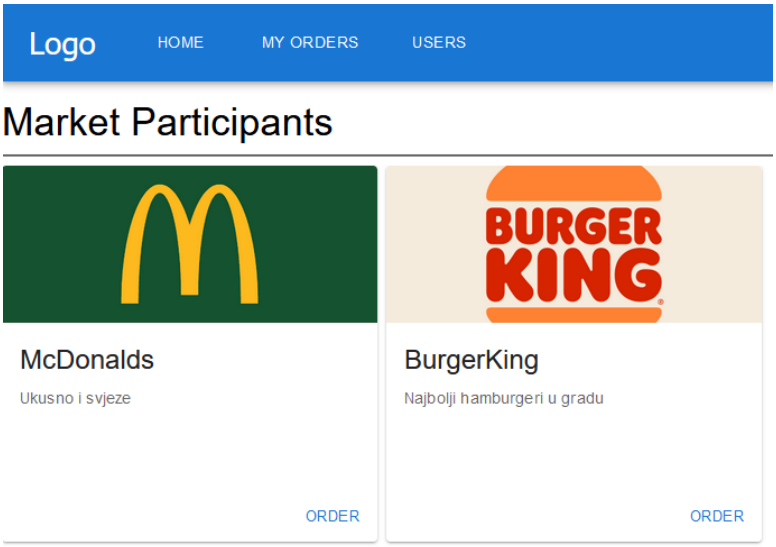


Slika 3.3 - Forma za kreiranje novog korisničkog računa

Na ekranu prikazanom na slici 3.3, također vidimo polje za lozinku s istim mehanizmom za sakrivanje/prikaz lozinke kao što je opisano na ekranu za prijavu.

3.2 Ekran s poslovnim subjektima za narudžbe

Ukoliko se korisnik aplikacije uspješno prijavio, otvara mu se novi ekran s listom svih poslovnih subjekata koji su kreirani u aplikaciji od strane administratora. Isječak s prikazanom poslovnim subjektima prikazan je na slici 3.4.



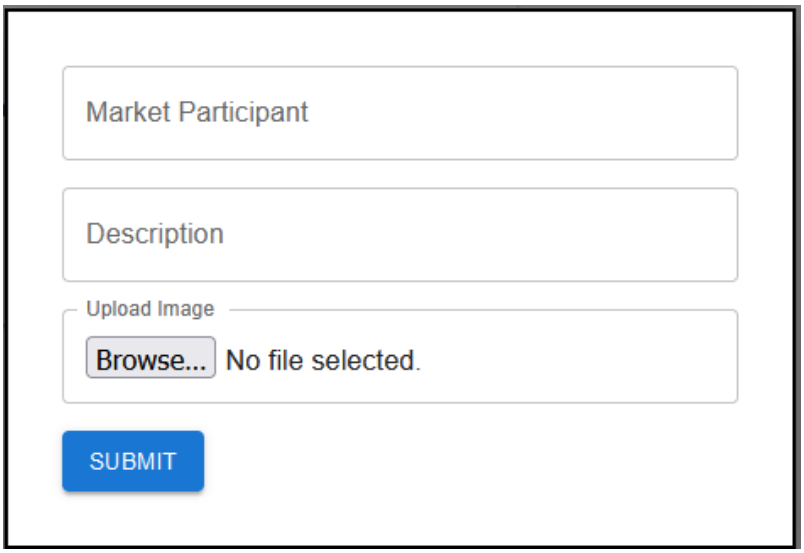
Slika 3.4 - Lista poslovnih subjekata dostupnih za kreiranje narudžbe

Osim samih poslovnih subjekata dostupnih za kreiranje narudžbe, na slici je prikazan isječak navigacijske trake koja će biti opisana u daljnjim poglavljima. Na slici se može vidjeti lista kartica koje predstavljaju određeni poslovni subjekt. Poslovni subjekt dostupan za kreiranje narudžbe od strane korisnika opisan je svojom slikom, imenom kao i kratkim opisom kojeg administrator unosi prilikom kreiranja novog poslovnog subjekta. Također na dnu kartice korisnicama je dostupan gumb s tekstom *Order* (naruči) na čiji klik korisnika aplikacija odvodi na novi ekran s detaljima poslovnog subjekta, kao i svim njegovim proizvodima dostupnim za ubacivanje u narudžbu.

Također, ukoliko prijavljeni korisnik ima dodijeljenu ulogu administratora, na samom dnu ekrana dostupan mu je gumb s tekstom *ADD MARKET PARTICIPANT* (dodaj poslovni subjekt) koji se može vidjeti na slici 3.5.

Slika 3.5 - Gumb za dodavanje novog poslovnog subjekta

Klikom na prikazan gumb, administratoru aplikacije otvara se prozor koji sadrži formular sa svim potrebnim informacijama potrebnim za kreiranje poslovnog subjekta, kao što su njegovo ime, opis te slika koja ga predstavlja na sučelju. Unosom imena poslovnog subjekta, opisa te slike koju administrator želi da bude prikazana na sučelju za određeni poslovni subjekt, te klikom na gumb *Submit* (podnesi zahtjev) na slici 3.6, u aplikaciju je dodan novi poslovni subjekt te se automatski dodaje u listu svih poslovnih subjekata na ekranu prikazanom na slici 3.4.



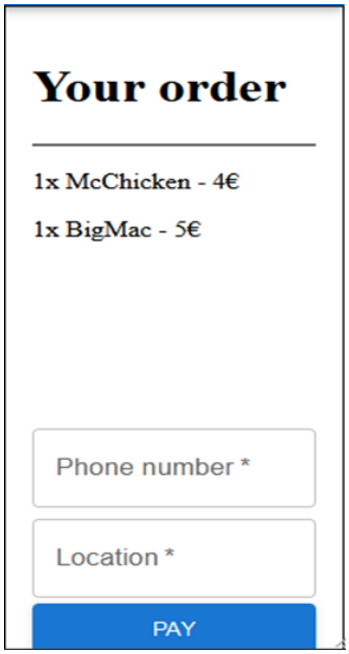
Slika 3.6 - Dodavanje poslovnih subjekata (administracija)

3.3 Detalji poslovnog subjekta

Kao što je opisano u prethodnom poglavlju, svaka kartica poslovnog subjekta, osim opisnih informacija, sadrži gumb s tekstom *Order* (naruči) na čiji klik aplikacija odvodi korisnika na novi ekran koji služi kao prikaz detalja poslovnog subjekta, odnosno prikaz svih proizvoda koje poslovni subjekt sadrži, kao i njihova cijena te opis sa slikom. Slično kao i kod ekrana s listom poslovnih subjekata, na dnu ekrana nalazi se gumb dostupan isključivo administraciji na čiji se klik otvara formular za dodavanje nove stavke koja će biti vezana za poslovni subjekt u kojem se trenutno korisnik (administrator) nalazi. Za dodavanje stavke, potrebno je unesti njeno ime, opis te sliku koju administrator želi da se prikaže na sučelju. Klikom na određenu stavku, ona se automatski dodaje u narudžbu. Na slici 3.7 prikazana je lista kartica koje predstavljaju opis proizvoda određenog poslovnog subjekta dostupnih za narudžbu.

Slika 3.7 - Detalji proizvoda određenog poslovnog subjekta

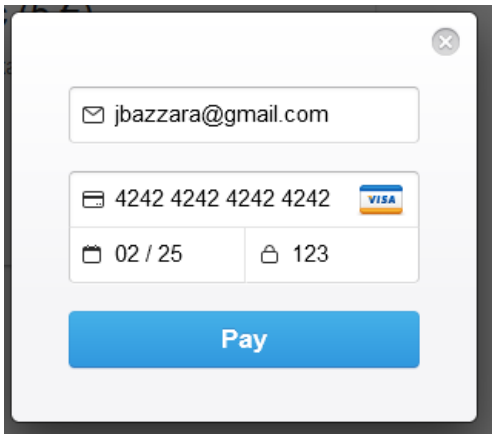
Slično kao i kod ekrana opisanog u prethodnom poglavlju s listom poslovnih subjekata, proizvodi poslovnog subjekta dostupnih za narudžbu također su opisani pripadnom slikom koju unosi administrator, naslovom uz koji je prikazana i cijena proizvoda te kratkim opisom. Na dnu same kartice nalazi se gumb s tekstom *Add* (dodaj u narudžbu), na čiji se klik proizvod koji je korisnik odabrao doda u narudžbu. Moguće je dodati isti proizvod više puta u narudžbu. Na slici 3.8 prikazana je isječak ekrana koji služi kao prikaz detalja narudžbe.



Slika 3.8 - Detalji narudžbe

Kako bi narudžba bila uspješno kreirana, osim što mora dodati proizvod u narudžbu, korisnik mora upisati broj mobitela koji služi kao kontakt, te adresu na koju želi da se narudžba dostavi. Broj mobitela i lokacija nisu vezane za samog korisnika koji vrši narudžbu zbog mogućnosti da određeni korisnik želi napraviti više narudžbi na različite adrese i s različitim kontaktom, u slučaju kada npr. korisnik nije kod kuće, ali svejedno

želi napraviti narudžbu. Nakon što korisnik odabere stavke koje želi naručiti, te unese potrebne informacije, klikom na gumb sa tekstom *Pay* (plati) sa slike 3.8, korisniku će biti prikazan novi prozor u kojem korisnik unosi detalje o svojoj kartici koja se procesira na Stripe naplatni servis. Prozorčić sa podatcima o naplati prikazan je na slici 3.9.

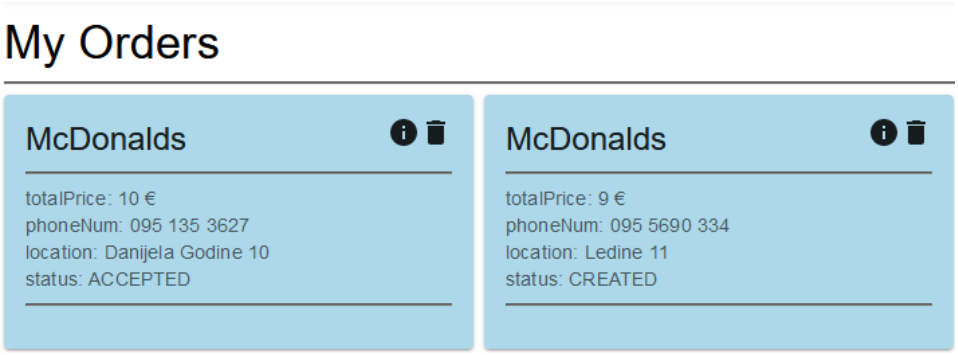


Slika 3.9 - Formular za unos podataka o naplati

Unosom elektroničke poštanske adrese koja služi kao podatak za kreiranje korisnika na Stipe servisu, čiji će detalji biti opisani u nastavku rada, kao i unosom podataka o kartici kao što su broj kartice, tajni CCV troznamenkasti broj te datum isticanja kartice, kreira se transakcija prema Stripe servisu. Podaci prikazani na slici 3.9 primjer su jednog od testnih podataka koje Stripe nudi. Iznos transakcije jednak je ukupnom zbroju cijena svih stavki dodanih u narudžbu. Ukoliko je Stripe servis potvrdio transakciju kao uspješnom, to znači da se narudžba uspješno kreirala, te slijedeći korak korisnika koji je naručio jest samo sačekat da dostavljač preuzme narudžbu te je dostavi.

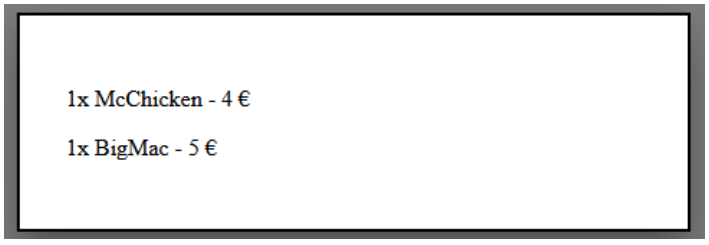
3.4 Ekran s kreiranim narudžbama

Nakon što korisnik kreira narudžbu, aplikacija korisnika odvodi na novi ekran na kojem korisnik ima uvid u sve svoje trenutno aktivne narudžbe, kao i u njihove detalje, poslovni subjekt za koji je narudžba kreirana, sve proizvode za koje je narudžba vezana, ukupnu cijenu narudžbe, broj telefona kao i adresu vezanu uz narudžbu itd. Isječak ekrana koji prikazuje trenutno aktivne narudžbe prijavljenog korisnika prikazan je na slici 3.10.



Slika 3.10 - Ekran s trenutno aktivnim narudžbama

Također uz naslov, korisniku su dostupne dvije ikonice. Klikom na prvu informativnu ikonu, korisniku se kartica proširuje te prikazuje sve stavke koje su naručene za određenu narudžbu, što je prikazano na slici 3.11.



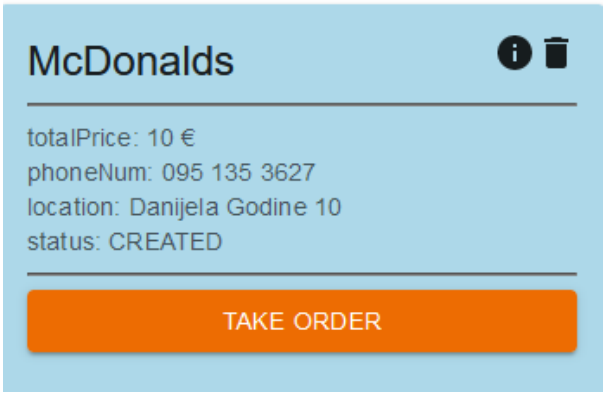
Slika 3.11 - Detalji narudžbe s pripadnim proizvodima

Klikom na drugu ikonicu, korisnik može poništiti svoju narudžbu. Također, ove ikonice dostupne su i administratoru te ukoliko smatra da je narudžba maliciozna (namjerno unesena kriva adresa ili sl.) može sam poništiti određenu narudžbu.

Nakon što se narudžba napravi, njen status je *CREATED* (narudžba kreirana), a nakon što dostavljač vidi i prihvati narudžbu, status se mijenja u *ACCEPTED* (narudžba prihvaćena). Postoji još i status *DELIVERED* (narudžba dostavljena) koji označava sve narudžbe koje su završene i uspješno dostavljene. Nakon što se narudžba završi i dostavljač ju označi

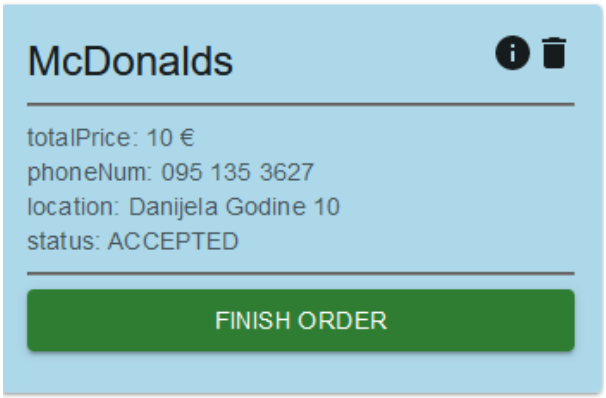
kao gotovu, korisniku više nije prikazana na sučelju jer sučelje prikazano na slici 3.10 pokazuje samo trenutno aktivne narudžbe. Narudžba je i dalje dostupna na pregled korisniku administratoru.

Na slici 3.12 prikazana je kartica koja predstavlja narudžbu u perspektivi dostavljača. Njemu nisu prikazane trenutno aktivne narudžbe trenutno prijavljenog korisnika, već trenutno aktivne narudžbe svih korisnika.



Slika 3.12 - Detalji narudžbe (dostavljač)

Prikaz narudžbe je prilično sličan kao i kod korisnika koji nema ulogu dostavljača. Dodatak je jedino gumb narančaste boje s tekstom *TAKE ORDER* (preuzmi narudžbu) na čiji klik narudžba mijenja svoj status te korisnik koji je vlasnik narudžbe na svom sučelju dobiva obavijest da je narudžba preuzeta.



Slika 3.13 - Detalji narudžbe (dostavljač)

Klikom na spomenuti narančasti gumb, narudžba mijenja svoj status, te dostavljaču je dostupan novi gumb zelene boje s tekstom *FINISH ORDER* (završi narudžbu) čiji klik označava uspješno završenu narudžbu. Dostavljač i administrator iz bilo kojeg razloga mogu otkazati narudžbu u bilo kojem trenutku, odnosno statusu, no korisnik bez uloge

dostavljača i/ili administratora može svoju narudžbu otkazati samo dok je još u statusu *CREATED*, odnosno dok ju dostavljač nije prihvatio, kako bi se spriječio potencijalno maliciozni pokušaj otkazivanja narudžbe nakon što je ona spremna i kod dostavljača.

3.5 Administracija korisnika

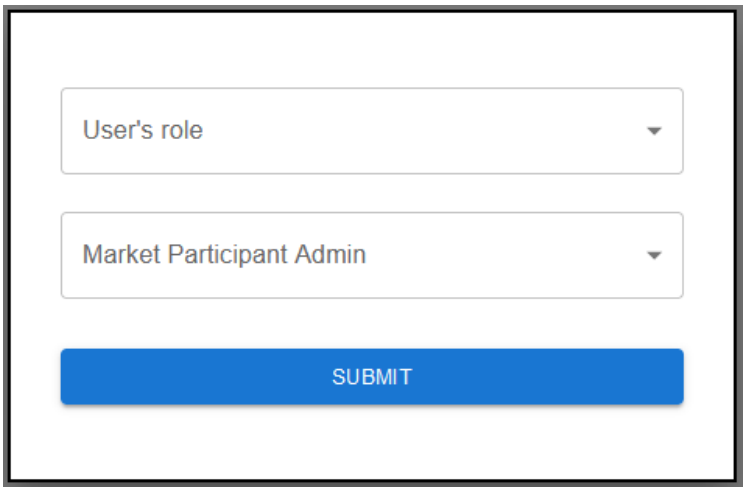
Slijedeći ekran koji će biti opisan u nastavku teksta jest ekran za prikaz svih korisnika s aktivnim računom u aplikaciji (Slika 3.14). Ekran je dostupan korisnicima s ulogom administratora u aplikaciji, te osim za prikaz svih aktivnih korisnika, tablica administratoru nudi mogućnost uređivanja korisničke uloge u aplikaciji, te mogućnost povezivanja korisnika s određenim poslovnim subjektom kako bi taj korisnik mogao administrirati taj poslovni subjekt, npr. dodavati nove stavke za narudžbu, te tako olakšati administratoru prilikom kontroliranja i vođenja sustava. Administrator uvidom u tablicu može vidjeti elektroničku poštu svih korisnika, kao i njihova imena, prezimena, nadimke te trenutno dodijeljenu ulogu.

ID	E-mail	First name	Last name	Username	Role
1	jbazzara@gmail.com	Josip	Bazzara	jbazzara@gmail.com	ADMIN
2	peroperic@gmail.com	Pero	Peric	peroperic@gmail.com	CUSTOMER
3	JankoBranko@gmail.com	Janko	Branko	JankoBranko@gmail.com	CUSTOMER

Rows per page: 10

Slika 3.14 - Tablica za administraciju korisnika

Duplim klikom na nekog od korisnika iz tablice, administratoru se otvara prozor sa formom (slika 3.15) gdje administrator može promovirati druge korisnike u dostavljače ili administratore, kao i dodijeliti korisniku neki poslovni subjekt za koji bi on bio zadužen. Ukoliko glavni administrator dodijeli poslovni subjekt nekom od korisnika, tada taj korisnik može uređivati stavke (proizvode dostupne za narudžbu kao i njihovu cijenu) isključivo za taj poslovni subjekt. Time glavni administrator može delegirati svoje zadatke na neke od drugih korisnika.



Slika 3.15 - Prozor za administraciju određenog korisnika

Na prozoru prikazanom na slici 3.15 administratoru su prikazana dva polja, prvi koji služi za dodjeljivanje nove role korisnika, te drugi za dodjeljivanje poslovnog subjekta odabranom korisniku. Klikom na gumb s tekstom *SUBMIT* (podnesi zahtjev), administrator može potvrditi svoje promjene za korisnika.

3.6 Navigacijska traka

Unutar cijele aplikacije u svakom trenutku korisniku je dostupna navigacijska traka prikazana na slici 3.16.

Slika 3.16 - Navigacijska traka

Na navigacijskoj traci korisniku su dostupne rute do svih dostupnih ekrana u aplikaciji, te ukoliko npr. prijavljeni korisnik nije administrator, neće imati pristup ekranu za administraciju korisnika, samim time taj ekran neće mu biti prikazan ni na navigacijskoj traci.

Također, s desne strane, ukoliko je korisnik uspješno prijavljen, dostupan mu je gumb s tekstom *LOGOUT* (odjavi se), na čiji klik korisnik biva odjavljen iz aplikacije. Na ekranima za autentifikaciju (prijava i registracija), taj gumb naravno nije prikazan.

4 RAZVOJ I IMPLEMENTACIJA SPECIFIČNIH APLIKACIJSKIH FUNKCIONALNOSTI

4.1 Autentifikacija i autorizacija

U nastavku teksta detaljnije će biti objašnjeno što je i kako je implementirana autorizacija i autentifikacija u ovoj aplikaciji te sve funkcionalnosti koje su dio tih procesa. Autentifikacija je proces kojim se potvrđuje identitet korisnika, u ovom slučaju putem lozinke, i odnosi se na određivanje nivoa pristupa korisnika određenim resursima. Dakle autentifikacija odgovara na pitanje tko je korisnik, a autorizacija odgovara na pitanje što korisnik smije raditi i čemu smije pristupati. Oba procesa su ključna za sigurnost informacijskih sustava i osiguravaju da samo ovlašteni korisnici imaju pristup osjetljivim podacima.

Prvi korak u implementiranju autorizacije i autentifikacije jest definiranje modela korisnika, odnosno što sve opisuje i čini distinkciju među korisnicima. Na slici 4.1 prikazan je kod koji opisuje klasu korisnika na poslužiteljskoj strani. Na slici se može vidjeti da je svaki korisnik opisan unikatnim identifikacijskim ključem, svojim imenom i prezimenom, elektroničkom poštom, lozinkom, rolom itd. Za implementaciju Spring Boot [1] sigurnosnog paketa važno je da klasa koja opisuje korisnika implementira *UserDetails* sučelje iz *org.spingframework.security.core* paketa.

```
public class User implements UserDetails {

    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "user_id_gen")
    @SequenceGenerator(name = "user_id_gen", sequenceName = "user_id_seq", allocationSize = 1)
    private Long id;

    @Column(name = "username")
    private String username;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    @Column(name = "email")
    private String email;

    private String password;

    @Enumerated(EnumType.STRING)
    private UserRole role;

    @OneToMany(mappedBy = "orderOwner")
    private List<Order> orderList;
```

Slika 4.1 - Prikaz modela korisnika

Daljnji korak u procesu autorizacije korisnika bio bi kreiranje tokena prilikom prijave korisnika. U ovoj aplikaciji koristi se *JWT* (JSON Web Token) vrsta tokena. JWT je kompletan te siguran način predstavljanja zahtjeva između dvije strane te je jedan od najčešćih načina implementacije autentifikacije i autorizacije u raznim aplikacijama. Sastoji se od zaglavlja, tijela te potpisa, te su svi dijelovi u samom tokenu odvojeni točkama. Zaglavlje sadrži informacije o tipu tokena te algoritmu enkripcije koji su potrebni poslužitelju. Tijelo sadrži tvrdnje (*claims*), koje su zapravo kriptirane informacije o korisniku. Nadalje, potpis kao dio tokena koristi se za verifikaciju autentičnosti tokena i osigurava da podaci nisu izmijenjeni nakon što je token izdan. Na slici 4.2 može se vidjeti isječak koda koji služi za generiranje JWT tokena.

```
public String generateToken(Map<String, Object> claims, UserDetails userDetails) {
    return Jwts.builder().setClaims(claims).setSubject(userDetails.getUsername())
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() + 3600000 * 24))
            .signWith(getSignInKey(), SignatureAlgorithm.HS256)
            .compact();
}
```

Slika 4.2 - Algoritam za generiranje JWT tokena

Na slici se može vidjeti, da konkretno za ovu aplikaciju, token u svoje tijelo sprema informacije o korisnikovom *nadimku* kojeg je postavio prilikom kreiranja računa. Također se može vidjeti da token traje točno jedan dan te da se za potpis tokena koristi *HS256* algoritam. Kako bi korisnik uopće dobio svoj token, potrebno je naravno unijeti točnu elektroničku poštu i pripadajuću lozinku definiranu prilikom kreiranja računa. Za provjeru ispravnosti unesenih podataka zaslužna je klasa *AuthenticationManager* iz *org.springframework.security.authentication* paketa. Na slici 4.3 prikazana je metoda u kojoj se može vidjeti korištenje klase iz spomenutog paketa.

```
public AuthResponse login(UserLoginRequest request) {
    authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(request.getEmail(), request.getPassword())
    );

    var user = userRepository.findByEmail(request.getEmail()).orElseThrow(() -> new UsernameNotFoundException("User not found"));

    var jwt = jwtService.generateToken(user);

    return AuthResponse.builder().jwt(jwt).build();
}
```

Slika 4.3 - Metoda za prijavu korisnika

Može se vidjeti da metoda prikazana na slici prima *UserLoginRequest* kao parametar koji u sebi sadrži elektroničku poštu te lozinku koju je korisnik unio prilikom pokušaja prijave te onda te informacije šalje u *authenticate*() metodu iz već spomenute *AuthenticationManager* klase. Ukoliko se lozinka podudara sa korisnikom čija je elektronička pošta dobivena iz zahtjeva, generira se JWT token te prijava prolazi uspješno. Ukoliko pak korisnik unese krivu lozinku ili elektroničku poštu koja se ne podudara niti s jednim korisnikom u bazi podataka aplikacije, korisnik će na pokušaj prijave od poslužitelja dobiti HTTP 403 status. HTTP 403 status jest HTTP status koji označava da su resursi s poslužitelja kojima korisnik pokušava pristupiti zaštićena, odnosno korisnik nema dozvolu pristupiti im.

Pravilo koje je implementirano u aplikaciji jest da svaki korisnik prilikom pokušaja pristupa bilo kojeg resursa na poslužitelju mora priložiti jedinstveni, već spomenuti JWT token koji mu je izdan prilikom prijave. To je implementirano uz pomoć metode prikazane na slici 4.4.

```
final String authHeader = request.getHeader( s: "Authorization");
final String jwtToken;
final String userEmail;

if(authHeader == null || !authHeader.startsWith("Bearer ")) {
    filterChain.doFilter(request, response);
    return;
}

jwtToken = authHeader.substring( beginIndex: 7);
userEmail = jwtService.extractUsername(jwtToken);

if (userEmail != null && SecurityContextHolder.getContext().getAuthentication() == null) {
    UserDetails userDetails = this.userDetailsService.loadUserByUsername(userEmail);

    if(jwtService.isTokenValid(jwtToken, userDetails)) {
        UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken
                (userDetails, credentials: null, userDetails.getAuthorities());
        authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
        SecurityContextHolder.getContext().setAuthentication(authToken);
    }
}
filterChain.doFilter(request, response);
```

Slika 4.4 -Metoda za provjeru JWT tokena

Na slici 4.4 može se vidjeti da klasa iz aplikacije *JwtAuthenticaionFilter* nasljeđuje *OncePerRequestFilter* klasu iz *org.springframework.web.filter* paketa. Klasa *OncePerRequestFilter* sadrži metodu *doFilterInternal*() u kojoj se može definirati logika koja će se izvršiti prilikom svakog zahtjeva koji dođe na poslužitelj. Na slici 4.5 može se vidjeti da metoda kao parametre, između ostalih, prima tijelo zahtjeva koji je došao na poslužitelj.

```
@Override
protected void doFilterInternal(@NonNull HttpServletRequest request, @NonNull HttpServletResponse response, @NonNull FilterChain filterChain)
        throws ServletException, IOException {
```

Slika 4.5 - Zaglavlje metode doFilterInternal()

Konkretno, u *doFilterInternal*() metodi jedino je zanimljivo što sadrži *Authorization* dio zahtjeva na poslužitelju. Ukoliko on ne postoji ili je krivog oblika, korisniku će biti vraćen već spomenuti 403 HTTP status te korisnikov zahtjev prema poslužitelju biti će odbijen. Nadalje, ukoliko *Authorization* dio zahtjeva postoji, ispravnog je oblika te sadrži ispravan JWT token, metoda na slici iz tokena izvlači podatke o korisniku, provjerava trajanje tokena i njegovu aktivnost. Ukoliko je sve prošlo uspješno, u *SecurityContextHolder* zapisuju se podaci o zahtjevu i prijavljenom korisniku. Nadalje, pošto je, kao što je već spomenuto, ovo samo filter koji se okida kao mehanizam provjere prije nego se sam zahtjev korisnika krene izvršavati, nakon što se filter odradi i sve prođe uspješno, zahtjev se dalje delegira na tražene resurse. Spomenuti *SecurityContextHolder* jest zapravo klasa iz *org.springframework.security.core.context* paketa te služi kao mjesto gdje se spremaju informacije o prijavljenom korisniku, te njegovoj roli i ostalim sigurnosnim atributima. Uz pomoć njega, u aplikaciji u bilo kojem trenutku može se pristupiti o podacima prijavljenog korisnika.

Za samu autorizaciju važno je još napomenuti da su lozinke kreiranih korisnika spremljene u bazi podataka aplikacije. Kako bi lozinke bile zaštićene i sigurne, one se prije spremanja kodiraju uz pomoć *BCrypt hash* algoritma. Implementacija *BCrypt* algoritma nalazi se u *org.springframework.security.crypto.password* paketu, a funkcionira na način da za svaku lozinku kreira jedinstveni niz nasumičnih bitova koji se dodaje lozinki prije nego što se izvrši heširanje. On osigurava da čak ukoliko dva korisnika imaju istu lozinku, njihovi heševi će biti različiti te će lozinke spremljene u bazi biti različite. Također, *BCrypt* algoritam uključuje više rundi procesiranja kako bi se proces dodatno usporio te se time povećava otpornost na različite vrste napada.

Na slici 4.6 prikazana je metoda koja je zadužena za autentifikaciju, odnosno provjeru kojim resursima može pristupati koji korisnik.

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
            .cors().and().csrf().disable()
            .authorizeHttpRequests()
            .requestMatchers("/auth/**").permitAll()
            .anyRequest().authenticated()
            .and()
            .sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and()
            .authenticationProvider(authenticationProvider)
            .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);

    return http.build();
}
```

Slika 4.6 - Metoda za autentifikaciju

Svaki resurs u aplikaciji definiran je svojom rutom. Npr. ukoliko korisnik želi pristupiti svojim trenutno aktivnim narudžbama, ruta za dohvat tih podataka biti će */orders/getCurrent*. U *securityFilterChain*() metodi definirano je kojim rutama smiju pristupati neprijavljeni korisnici te koje rute su strogo zaštićene. Zaštićenim rutama može se pristupati samo ukoliko zahtjev sadrži već opisani JWT token koji sadrži informacije o korisniku koji je zahtjev podnio. Na slici se može vidjeti da resursima opisanima sa */auth* rutom može pristupati bilo tko.

Na klijentskoj strani aplikacije korištena je programska biblioteka za React [2] aplikacije zvana react-auth-kit [4]. Na slici 4.7 prikazano je kako se react-auth-kit [4] može integrirati s React [2] aplikacijom, te što je sve potrebno za to.

Slika 4.7 - Korištenje react-auth-kit biblioteke

Na slici 4.7 vidi se da je definirana varijabla imena *store* koja je potrebna kako bi se definirali parametri ključni za implementaciju autorizacije na klijentskoj strani aplikacije. U njoj jest definirano gdje će se spremati JWT token dobiven sa poslužitelja, pod kojim ključem će se spremiti na definiranu lokaciju kako bi se poslije JWT token, ukoliko je potrebno, po istom ključu mogao i dohvatiti. Također, definirano je želi li se osigurati lokacija pod kojom je JWT token spremljen, no to znači da se JWT tokenu ne može pristupiti unutar koda aplikacije. Tako se može vidjeti konkretno da je na slici 4.6 odabrano da se JWT token dobiven s poslužitelja sprema u web kolačić pod ključem *_auth*, te zbog potrebe pristupa tokena u kodu, web kolačić nije dodatno osiguran. Web kolačić je mala tekstualna datoteka koja se pohranjuje na računalu ili općenito uređaju korisnika kada posjete web stranicu. Web kolačići omogućuju web aplikacijama da prepoznaju korisnike i njihove preferencije pri budućim posjetima. Također, na slici 4.7 prikazano je da se opisana varijabla imena *store* proslijeđuje u komponentu zvanu *AuthProvider*. *AuthProvider* komponenta je komponenta iz *react-auth-kit* [4] programske biblioteke, te se u njoj nalaze funkcije i algoritmi za spremanje JWT tokena, provjere je li korisnik koji pokušava pristupiti web aplikaciji autoriziran, funkcije za prijavu korisnika, odjavu itd. *AuthProvider* komponenta također je zadužena za zaštitu sučelja i ekrana za koje je potrebno biti prijavljen kako bi im se pristupilo. Ukoliko u web kolačiću ne postoji spremljeni JWT token za određenog korisnika, te taj korisnik pokušava pristupiti npr. ekranu s prikazom proizvoda određenog poslovnog subjekta, tada će korisnik automatski biti premješten na ekran za prijavu korisnika. Tim mehanizmom postoji zaštita i na

klijentskoj strani od neautoriziranog pristupa provjerljivim podacima i sučeljima. Kako bi *AuthProvider* komponenta služila svojoj svrsi, ona mora biti roditelj svim ostalim komponentama u aplikaciji, odnosno svi ekrani moraju imati *AuthProvider* kao krovnu komponentu. Koncept roditeljskih i dječjih komponenti u React [2] aplikacijama odnosni se na strukturu i odnos između različitih komponenti, a koristi se kako bi se postigao modularan i organiziran način razvoja korisničkog sučelja. Također, važno je napomenuti da dječje komponente imaju pristup podacima koji su definirani u roditeljskoj komponenti te na taj način svi ekrani u aplikaciji mogu pristupati metodama i varijablama definiranim u *AuthProvider* komponenti. Na slici 4.8 može se vidjeti primjer korištenja funkcije za prijavu korisnika unutar autorizacijskog ekrana definiranu u *AuthProvider* komponenti.

```
function handleSubmit(data: LoginRequest) {
    PostLogin(data).then(response : JwtResponse  => {
        signIn( signInConfig: {
            auth: {
                token: response.jwt,
                type: "Bearer"
            },
            userState: {
                email: data.email
            }
        });
        navigate( to: "/home");
    })
}
```

Slika 4.8 - Funkcija za prijavu korisnika u React aplikaciji

PostLogin() funkcija, čiji se kod može vidjeti na slici 4.9, služi tome da podatke (elektronička pošta i lozinka korisnika) potrebne za autorizaciju proslijedi poslužitelju.

```
export async function PostLogin(request: LoginRequest) : Promise<JwtResponse> {
    const response = await fetch( input: 'api/auth/login', init: {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(request),
    });

    return response.json();
}
```

Slika 4.9 - PostLogin() funkcija

Na slici se može vidjeti da metoda *PostLogin*() zahtjev tipa *LoginRequest* koji sadrži podatke o korisnikovoj elektroničkoj pošti i lozinki koje je unio prilikom pokušaja prijave šalje na poslužitelj, konkretno na rutu */api/auth/login*. Također, može se vidjeti da *PostLogin*() funkcija prikazana na slici 4.9 kao odgovor od poslužitelja očekuje već spomenuti JWT token koji je garancija da je korisnik unio ispravne podatke za prijavu te da je uspješno autoriziran za ulazak u aplikaciju.

Nakon što poslužitelj odgovori s JWT tokenom, tada se token prosljeđuje u *signIn*() funkciju definiranu u već spomenutom *AuthProvideru*, čija se implementacija nalazi u react-auth-kit [4] programskoj biblioteci. Zadatak *signIn*() funkcije iz *react-auth-kit* [4] programske biblioteke jest da pohrani JWT token dobiven iz spomenute PostLogin() funkcije sa slike 4.9 na odgovarajuću definiranu lokaciju. Pogledom na sliku 4.7 može se podsjetiti kako su definirane varijable koje opisuju lokaciju i način spremanja JWT tokena. Te varijable spremaju se uz pomoć AuthProvider komponente iz react-auth-kit [4] biblioteke u kontekst aplikacije prilikom svakog pokretanja aplikacije.

Također, kako bi svaki zahtjev korisnika prema poslužitelju imao odgovarajući JWT token u svome zaglavlju, za to se brine funkcija *useAuthHeader*() iz react-auth-kit [4] biblioteke. Uporaba *useAuthHeader*() funkcije vidi se na slici 4.10.

```
const authHeader = useAuthHeader();

useEffect( effect: () => {
    PostGetAllUsers(authHeader).then(response : UserResponse[]  => {
        setUsers(response)
    });
}, deps: [authHeader]);
```

Slika 4.10 - Uporaba authHeadera

Konkretno na slici 4.10 može se vidjeti da je definirana varijabla imena *authHeader* te se dohvaća s spomenutom *useAuthHeader*() funkcijom. Nadalje, u primjeru sa slike prikazano jest kako je varijabla *authHeader* proslijeđena u *PostGetAllUsers*() funkciju. *PostGetAllUsers*() funkcija jest funkcija zadužena za dohvat svih korisnika aplikacije sa poslužitelja. Kod *PostGetAllUsers*() funkcije prikazan je na slici 4.11.

```
const response = await fetch( input: '/api/users/getAll', init: {
    method: "POST",
    headers: new Headers( init: {
        "Authorization": `${authHeader}`,
        "Content-Type": "application/json"
    })
});
```

Slika 4.11 - PostGetAllUsers() funkcija

Važno jest da spomenuta *authHeader* varijabla u zahtjevu bude spremljena pod ključem *Authorization* kao što je prikazano na slici 4.11. To je važno zbog toga što poslužitelj očekuje JWT token na toj lokaciji u zaglavlju zahtjeva. Ukoliko JWT token iz *authHeader* varijable postoji, ali je spremljen pod drugačijim ključem, poslužitelj će odbiti zahtjev te korisnik neće dobiti zatražene resurse, u ovom slučaju listu svih korisnika aplikacije za administratora. Opisanim slijedom akcija na klijentu i poslužitelju postiže se autorizacija i autentifikacija korisnika u ovoj web aplikaciji.

4.2 Stripe

Nadalje, funkcionalnost koja će u nastavku teksta biti detaljnije objašnjena jest Stripe [5], to jest funkcionalnost kartičnog plaćanja u ovoj web aplikaciji. Za integraciju s Stripe [5] platformom i implementaciju mogućnosti kartičnog plaćanja unutar web aplikacije potrebno je imati Stripe [5] korisnički račun. Prilikom kreiranja računa potrebno je navesti potrebne informacije o sebi i svojoj tvrtki, kao i povezati bankovni račun za isplate. U verziji za testiranje koja je korištena u ovoj web aplikaciji nije potrebno navesti informacije o svojoj tvrtki te bankovnom računu. Nadalje, nakon kreiranja računa korisnik dobiva API (*application programming interface*) ključeve, i to privatni i javni API ključ, koji su potrebni za komunikaciju između Stripe [5] platforme i web aplikacije korisnika. API, odnosno aplikacijsko programsko sučelje jest skup određenih pravila i specifikacija koje programeri slijede tako da se mogu služiti uslugama ili resursima operacijskog sustava ili nekog drugog složenog programa, strukture podataka ili protokola. Pošto je ova aplikacija izrađena u *Spring Boot* [1] razvojnom okruženju, za pomoć pri komunikaciji između Stripe [5] platforme i poslužitelja aplikacije korištena je *com.stripe.java* programska biblioteka [6] koja sadrži metode i algoritme koji nam pomažu u komunikaciji sa *Stripe* [5] platformom, za kreiranje transakcija, dohvat transakcija, dohvat i kreiranje korisnika na Stripe [5] platformi i sve ostale mogućnosti koje Stripe [5] nudi. Na slici 4.12 može se vidjeti način kako se privatni API ključ dodijeljen od Stripe [5] platforme rabi i integrira sa već spomenutom Stripe-java [6] knjižnicom.

```
@Service
public class PaymentService {

    @PostConstruct
    private void postConstruct() {
        Stripe.apiKey = "sk_test_51PFFBD03WFT1Ir1I8bm2OvwaLBTXDK88TRSPAzAb7RQzdsdJtCMm0ISSl0mjDPMtrLknJKUrSeZFOzh6CtBTodOB00xsftZImH";
    }
```

Slika 4.12 - Uporaba Stripe privatnog ključa

Dakle, pošto je metoda u kojoj se privatni ključ nalazi anotirana sa *@PostConstruct* anotacijom, to znači da će se metoda okinuti odmah nakon što se *PaymentService* klasa instancira u aplikaciji, te će Stripe klasa iz Stripe-java [6] knjižnice pročitati ponuđeni ključ te će ga spremiti u svoj kontekst kako bi ga pri svakom pokušaju komunikacije sa Stripe platformom mogao iskoristiti. Na slici 4.13 prikazana je metoda zaslužna za kreiranje transakcije prilikom kreiranja narudžbe.

```java
public Charge chargeNewCard(String token, Long amount) throws Exception {
    Map<String, Object> chargeParams = new HashMap<>();
    chargeParams.put("amount", amount);
    chargeParams.put("currency", "EUR");
    chargeParams.put("source", token);

    return Charge.create(chargeParams);
}
```

Slika 4.13 - Metoda za kreiranje Stripe transakcije

Stripe [5] platforma u komunikaciji s poslužiteljem web aplikacije u zahtjevu očekuje parametre definirane u službenoj Stripe [5] dokumentaciji. Konkretno, za kreiranje transakcije potrebno je priložiti iznos za koji se transakcija kreira (u ovom slučaju ukupan iznos cijele narudžbe), valuta u kojoj je iznos prikazan, te poseban jednokratan token koji traje za vrijeme jedne sesije između aplikacije i Stripe [5] platforme. Kako dobiti jednokratni token biti će objašnjeno u nastavku rada, gdje će biti opisana komunikacija između Stripe platforme i klijentske strane ove web aplikacije. Ukoliko se transakcija uspješno obavila, poslužitelj će klijentu vratiti informaciju da se narudžba uspješno kreirala zajedno sa svim informacijama o kreiranoj transakciji.

Na klijentskoj strani ove aplikacije, kao pomoć pri implementaciji kartičnog plaćanja te komunikacije sa Stripe platformom, korištena je *react-stripe-checkout* [7] programska biblioteka. *React-stripe-checkout* [7] programska biblioteka sadrži Stripe komponentu koja se sastoji od ugrađenih funkcija potrebnih za komunikaciju sa Stripe platformom te formularom za unos podataka potrebnih za kreiranje transakcije kao što su elektronička pošta korisnika, iznos transakcije, broj kartice s kojom se transakcija želi kreirati te ostali potrebni podaci o kartici kao što su datum isticanja te tajni troznamenkasti CCV broj. Na slici 4.14 prikazano je kako je Stripe komponenta iz *React-stripe-checkout* [7] programske biblioteke iskorištena.

```jsx
return(
    <Stripe
        token={handlePayment}
        stripeKey="pk_test_51PFFBD03WFT1Ir1IHmnMkQrizxqQMd1x6fINlFuoF8AjPTRGhuWKzFuMFgKtzdoHdIpETOafERBUTYPXLlweaodS00Mf5bPwgw"
    >
        <Button type="submit" fullWidth variant="contained" color="primary">PAY</Button>
    </Stripe>
);
```

Slika 4.14 - Stripe React komponenta

Stripe [5] komponenti potrebno je priložiti javni API ključ sa Stripe [5] platforme, kao i implementaciju funkcije za kreiranje transakcije koja će pozvati već opisan *chargeNewCard*() servis na poslužitelju sa slike 4.13. Na slici 4.15 prikazane su funkcije implementirane na klijentskoj strani aplikacije koje služe za komunikaciju s poslužiteljem, te izračun ukupnog iznos narudžbe kako bi mogao kreirati transakciju s ispravnim iznosom.

```
async function handlePayment(token: Token) {
    await fetch( input: `/api/payment/charge`, init: {
        method: "POST",
        headers: new Headers( init: {
            "Authorization": `${authHeader}`,
            "Content-Type": "application/json",
        }),
        body: JSON.stringify({token: token.id, amount: calculateAmount()} as PaymentRequest)
    }).then((response : Response ) => {
        if (response.ok) {
            handleSubmit()
        }
    })
}

function handleSubmit() {
    PostCreateOrder({itemList: items, mpId: params.id} as OrderRequest, authHeader).then(() => {
        navigate( to: "/order/my")
    })
}

function calculateAmount() {
    let totalPrice = 0
    items.map(item : ItemResponse  => totalPrice += item.price)
    return totalPrice * 100
}
```

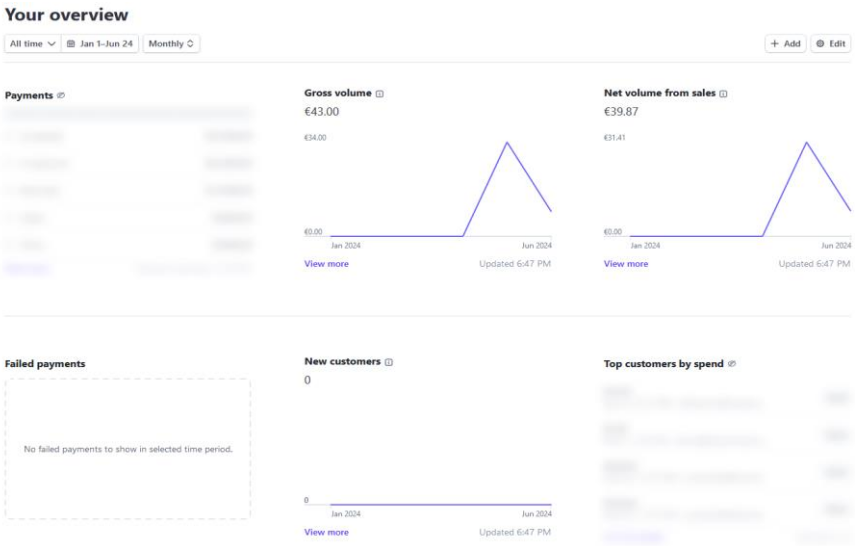
Slika 4.15 - React funkcije za kreiranje transakcije

Također, u funkciji *handleSubmit*() prikazano je korištenje *PostCreateOrder*() funkcije što je funkcija čiji je cilj poslužitelju signalizirati da je narudžba spremna za kreiranje te mu prosljeđuje listu proizvoda odabranih od strane korisnika koji ulaze u narudžbu kao i identifikacijski kod poslovnog subjekta za koji se narudžba kreira. Funkcija *PostCreateOrder*() korisnikov zahtjev prosljeđuje na poslužitelj na rutu */api/orders*, a na slici 4.16 gdje je prikaza kod funkcije *PostCreateOrder*() također se vidi da funkcija koristi već spomenutu *authHeader* varijablu iz *react-auth-kit* programerske biblioteke kako bi poslužitelj znao da zahtjev dolazi od strane autoriziranog korisnika.

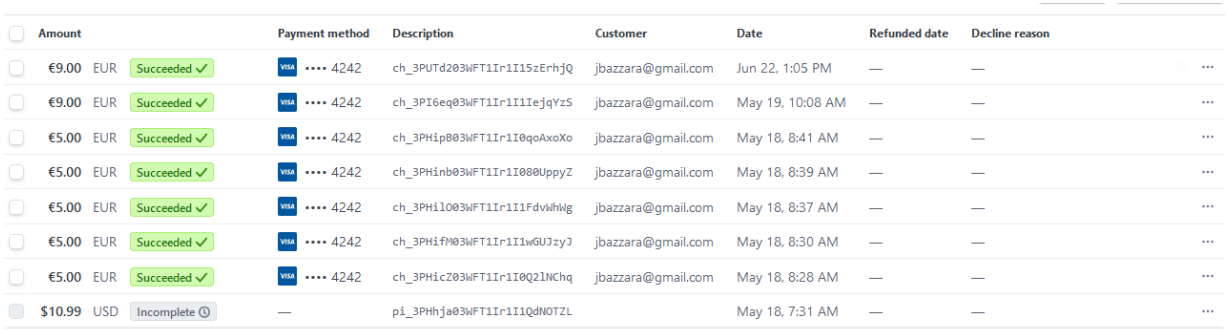
Slika 4.16 - Funkcija PostCreateOrder()

Prvi korak u procesu kartičnog plaćanja u aplikaciji jest uspostava komunikacije klijenta i Stripe [5] platforme. Implementacija za uspostavu komunikacije klijenta i Stripe platforme nalazi se u već spomenutoj *Stripe React* komponenti prikazanoj na slici 4.14. U tom koraku, klijentska strana aplikacije na Stripe [5] platformu šalje podatke unesene od strane korisnika, a to su broj kartice, vrijeme isticanja te tajni troznamenkasti CCV broj. Uz korisnikove kartične podatke, Stripeu [5] se prosljeđuje javni *API* ključ koji služi za autorizaciju korisnika na Stripe [5] platformi. Javni *API* ključ proslijeđen je *Stripe React* komponenti što je prikazano na slici 4.14. Nakon toga, na Stripe [5] platformi provjerava se ispravnost tih podataka. Ukoliko je Stripe [5] platforma zaključila da je korisnik unio ispravne podatke o svojoj kartici, tada će klijentskoj strani aplikacije odgovoriti s već spomenutim jednokratnim tokenom potrebnim za komunikaciju između poslužitelja i Stripe platforme. Na slici 4.15 može se vidjeti da u *handlePayment*() funkciji, nakon što Stripe platforma odgovori s jednokratnim tokenom, taj se token prosljeđuje na poslužitelja na rutu */payment/charge*, konkretno na *chargeNewCard*() metodu zaslužnu za finalno kreiranje transakcije. Na slici 4.15 može se vidjeti da poslužitelj u zahtjevu očekuje objekt tipa *PaymentRequest* koji uz opisani token sadrži i ukupni iznos narudžbe. Funkcija *calculateAmount*() zaslužna je za izračun ukupnog iznosa narudžbe, a funkcionira tako što prolazi kroz petlju svih proizvoda u narudžbi te ukupnoj cijeni pridodaje cijenu proizvoda u trenutnoj iteraciji petlje. Nakon što se prođu svi proizvodi u narudžbi, ukupna cijena množi se sa 100, zbog toga što Stripe platforma za kreiranje transakcije očekuje cijenu u centima, a unutar aplikacije cijena proizvoda u bazi podataka iskazana je u eurima. Opisanim slijedom akcija između klijenta, poslužitelja i Stripe platforme omogućene su kartične transakcije unutar aplikacije. Na slikama 4.17 te 4.18 prikazani su isječci

upravljačke ploče gdje vlasnik Stripe korisničkog računa ima uvid u sve kreiranje transakcije, postotak uspješnosti obavljenih transakcija, ukupan prihod svih transakcija, koliko različitih korisnika je pokušalo obaviti transakciju i još mnogo toga.



Slika 4.17 - Statistika na Stripe platformi



Amount			Payment method	Description	Customer	Date	Refunded date	Decline reason
€9.00	EUR	Succeeded ✓	•••• 4242	ch_3PUTd203WFT1Ir1I15zErhjQ	jbazzara@gmail.com	Jun 22, 1:05 PM	—	—
€9.00	EUR	Succeeded ✓	•••• 4242	ch_3PI6eq03WFT1Ir1I1IejqYzS	jbazzara@gmail.com	May 19, 10:08 AM	—	—
€5.00	EUR	Succeeded ✓	•••• 4242	ch_3PHipB03WFT1Ir1I0qoAxoXo	jbazzara@gmail.com	May 18, 8:41 AM	—	—
€5.00	EUR	Succeeded ✓	•••• 4242	ch_3PHinb03WFT1Ir1I080UppyZ	jbazzara@gmail.com	May 18, 8:39 AM	—	—
€5.00	EUR	Succeeded ✓	•••• 4242	ch_3PHilO03WFT1Ir1I1FdvWhWg	jbazzara@gmail.com	May 18, 8:37 AM	—	—
€5.00	EUR	Succeeded ✓	•••• 4242	ch_3PHifM03WFT1Ir1I1wGUJzyJ	jbazzara@gmail.com	May 18, 8:30 AM	—	—
€5.00	EUR	Succeeded ✓	•••• 4242	ch_3PHicZ03WFT1Ir1I0Q2lNChq	jbazzara@gmail.com	May 18, 8:28 AM	—	—
$10.99	USD	Incomplete	—	pi_3PHhja03WFT1Ir1I1QdNOTZL		May 18, 7:31 AM	—	—

Slika 4.18 - Detalji transakcija na Stripe platformi

5 ZAKLJUČAK

U ovom radu napravljena je aplikacija s korisničkim sučeljem koje omogućuje jednostavno obavljanje narudžbi iz različitih poslovnih subjekata s mogućnošću kartičnog plaćanja, uz administratorske alate za učinkovito upravljanje sustavom.

Tehnologije koje su korištene, počevši od Spring Boot [1] razvojnog okruženja, Reacta [2] kao programske biblioteke za razvoj klijentskog dijela aplikacija, PostgreSQL [x] kao sustava za upravljanje bazom podataka itd. omogućile su brzu, jednostavnu i pouzdanu implementaciju svih potrebnih funkcionalnosti. Uz poznavanje načina funkcioniranja weba i razvoja web aplikacija, uz navedene tehnologije otvara se mnoštvo mogućnosti koje je moguće ostvariti sa alatima opisanima u ovome radu.

U tijeku procesa razvoja, naišlo se na mnogo izazova koji su naposlijetku uspješno savladani. Implementacija slojevite arhitekture omogućila je jasnu organiziranost koda te olakšala testiranje i održavanje aplikacije.

U pogledu budućnosti projekta, planirano je nastaviti s njegovim unaprjeđenjem i nadogradnjom. To uključuje dodavanje novih funkcionalnosti poput poboljšanja korisničkog iskustva kroz implementaciju naprednijih algoritama pretrage i filtriranja, proširenje administracijskih mogućnosti kao što su dodavanje grafova za prikaz statistike, poboljšanja na sigurnosnim funkcionalnostima poput mogućnosti oporavka lozinke ili dvostruke razine autentifikacije kako bi podaci i računi u aplikaciji bili što sigurniji.

Uz razna navedena planirana poboljšanja, web aplikacija za naručivanje dostave s trenutnom verzijom spremna je za uspješno korištenje te omogućuje sve najvažnije zamišljene funkcionalnosti, a to su mogućnost korisnika za kreiranje narudžbe uz mogućnost kartičnog plaćanja, mogućnost dostavljača da upravlja kreiranim narudžbama te mogućnost administratora da upravlja cijelim sustavom.

LITERATURA

[1] Spring boot službena dokumentacija, https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/, 25. veljače 2024.

[2] React službena dokumentacija, https://react.dev/, 23. veljače 2024.

[3] MaterialUI službena dokumentacija, https://mui.com/, 23. veljače 2024.

[4] ReactAuthKit službena dokumentacija, https://authkit.arkadip.dev/, 23. veljače 2024.

[5] Stripe službena dokumentacija, https://docs.stripe.com/api, 19. svibnja 2024.

[6] Stripe-java programska biblioteka, javni GitHub repozitorij, https://github.com/stripe/stripe-java, 15. lipnja 2024.

[7] React-stripe-checkout programska biblioteka, javni GitHub repozitorij, https://github.com/azmenak/react-stripe-checkout, 15. lipnja 2024.

[8] PostgreSQL službena dokumentacija, https://www.postgresql.org/docs/current/index.html, 15. lipnja 2024.

[9] MaterialUI, službena dokumentacija, https://mui.com/, 15.lipnja 2024.

[10] Docker alat, službena dokumentacija, https://docs.docker.com/, 15. lipnja 2024.

[11] Postman alat, službena dokumentacija, https://www.postman.com/, 15. lipnja 2024.

[12] Oracle Java, službena dokumentacija, https://docs.oracle.com/en/java/, 25. veljače 2024.

[13] IBM: „What is a REST API?“, članak s interneta, https://www.ibm.com/topics/rest-apis, 15. lipnja 2024.

[14] Typescript, službena dokumentacija, https://www.typescriptlang.org/docs/, 15. lipnja 2024.

[15] NPM packet manager, službena dokumentacija, https://docs.npmjs.com/, 15. lipnja 2024.

POPIS SLIKA

SAŽETAK

U ovom radu predstavljena je web aplikacija za naručivanje dostava. Aplikacija svojim funkcionalnostima ostvaruje potrebe korisnika za mogućnost dostave putem aplikacije uz podršku kartičnog plaćanja, kao i potrebe administratora za upravljanje sustavom za naručivanje dostava te potrebe dostavljača da upravlja narudžbama. Aplikacija je izgrađena prateći REST standarde. Tehnologija korištena za izgradnju poslužiteljske strane aplikacije jest Spring Boot, dok je za klijentsku stranu korištena React programska biblioteka. Od ostalih tehnologija, korišten je PostgreSQL kao sustav za upravljanje relacijskom bazom podataka, MaterialUI kao biblioteka za izgradnju sučelja te Stripe kao platforma za podršku kartičnog plaćanja.

Ključne riječi: Web, aplikacija, narudžbe, Spring Boot, React, REST

ABSTRACT

This paper presents a web application for ordering deliveries. With its functionalities, the application fulfills the needs of the user for the possibility of delivery via the application with the support of card payment, as well as the needs of the administrator to manage the delivery ordering system and the needs of the delivery person to manage orders. The application was built following REST standards. The technology for building the server side of the application is Spring Boot, while the React program library was used for the client side. Among other technologies, PostgreSQL was used as a relational database management system, MaterialUI as a library for building interfaces, and Stripe as a platform for card payment support.

Keywords: web, application, deliveries, Spring Boot, React, REST