UNIVERZITET SINGIDUNUM

TEHNIČKI FAKULTET

WEB APLIKACIJA ONLINE TRGOVINE BACKEND

- diplomski rad -

Mentor: **Kandidat:**

prof. dr Milan Paroški **Predrag Radak**

Novi Sad, 2023.

UNIVERZITET SINGIDUNUM

TEHNIČKI FAKULTET

21000 NOVI SAD, Bulevar Mihajla Pupina 4a

KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj:	
Tip dokumentacije:	Monografska publikacija
Tip zapisa:	Tekstualni štampani dokument
Vrsta rada:	Diplomski rad
Autor:	Predrag Radak
Mentor:	Dr Milan Paroški
Naslov rada:	Web aplikacija online trgovine - Backend
Jezik publikacije:	Srpski (latinica)
Jezik izvoda:	Srpski (latinica)
Država publikovanja:	Srbija
Godina:	2023
Mesto i adresa:	Univerzitet Singidunum Danijelova 32, Beograd
Fizički opis rada: (poglavlja/strana/ citata/tabela/slika/grafika/priloga)	4/54/0/4/34/0/0
Naučna oblast:	Primenjene računarske nauke i informatika
Naučna disciplina:	Softverske i informacione tehnologije
Predmetna odrednica/Ključne reči:	Web aplikacija, online trgovina.
UDK	
Čuva se:	Biblioteka, Beograd
Važna napomena:	
Izvod:	
Datum određivanja teme:	

Datum odbrane:			
Članovi komisije:	Predsednik:	Dr Đorđe Obradović	
	Član, mentor:	Dr Milan Paroški	

UNIVERZITET SINGIDUNUM 21000 NOVI SAD, Bulevar Mihajla Pupina 4a	Broj:
ZADATAK ZA IZRADU RADA	Datum:

NA OSNOVU PODNETE PRIJAVE, PRILOŽENE DOKUMENTACIJE I ODREDBI STATUTA FAKULTETA

IZDAJE SE ZADATAK ZA ZAVRŠNI RAD, SA SLEDEĆIM ELEMENTIMA:

- problem – tema rada;
- način rešavanja problema i način praktične provere rezultata rada, ako je takva provera neophodna;
- literatura

NASLOV ZAVRŠENOG RADA:

Web aplikacija online trgovine - Backend

TEKST ZADATKA:

Web aplikacija onlajn trgovine pića treba da pruži korisnicima sledeće funkcionalnosti i opcije:
Pregledajte i pretražujte ponudu:Korisnici bi trebalo da budu u mogućnosti da pretražuju različita pića u ponudi. Aplikacija treba da omogući korisnicima da lako traže proizvode prema kategorijama, brendovima ili drugim relevantnim parametrima.
Detalji o proizvodu:Aplikacija treba da pruži detaljne informacije o svakom proizvodu, kao što su opis, tip i ostali atributi koji dodatno opisuju proizvod.
Korpa i narudžbina:Korisnici bi trebalo da imaju mogućnost da dodaju željene proizvode u korpu, pregledaju naručene proizvode, promene količinu ili uklone proizvode iz korpe i jednostavno poruče.
Personalizacija i preporuke:Aplikacija može da koristi algoritme personalizacije da korisnicima pruži preporuke proizvoda na osnovu njihovih preferencija, prethodnih kupovina ili drugih relevantnih podataka.
Korisnička podrška: Aplikacija treba da pruži korisničku podršku, bilo putem ćaskanja uživo, telefona ili e-pošte, tako da korisnici mogu da dobiju odgovore na svoja pitanja ili rešenja za moguće probleme.
Administrativni panel:Za upravljanje sadržajem aplikacije, administrativni panel treba da omogući dodavanje, uređivanje i brisanje proizvoda, upravljanje porudžbinama, praćenje zaliha i druge administrativne funkcije.
Ove funkcionalnosti su ključne za web aplikaciju onlajn pića, a ostale specifičnosti se mogu prilagoditi potrebama i zahtevima aplikacije.

Rukovodilac studijskog programa:	Mentor rada:
	Dr. Milan Paroški

SADRŽAJ

Predgovor

Kao mladi ljudi, iz domaćinskih kuća, odrasli smo uz domaće proizvode. Navikli smo na svežu hranu a tako isto i na cedjene sokove koje nam je baka pravila. Vremenom kako smo odrasli okusili smo i tradicinoalna alkoholna pića koja nam dolaze baš iz našeg domaćinstva. Samim tim smatramo da na ovaj način možemo prezentovati naš stručan rad tokom studija i ljubav prema tradiciji.

U današnjem svetu gde se sve više ljudi oslanja na onlajn platforme za obavljanje svojih svakodnevnih aktivnosti, onlajn kupovina hrane i pića postaje sveprisutna. Upravo iz tog razloga, odlučili smo da razvijemo online prodavnicu posvećenu prodaji pića(Domaće rakije).

Cilj ove web aplikacije je da korisnicima pruži zgodan način da pretražuju, biraju i naručuju pića. Takođe želim da osiguram da korisnici imaju pristup detaljnim informacijama o proizvodu, kao i mogućnost da ocenjuju i ostavljaju recenzije kako bi stvorili zajednicu korisnika od poverenja.

Ova web aplikacija ne samo da će korisnicima pružiti pogodnost u procesu naručivanja pića, već će podržati i industriju hrane i pića na mreži koja se stalno razvija.

Uvod

U savremenom društvu, web online shopovi su postali neizostavan deo našeg svakodnevnog života, omogućavajući nam jednostavno i udobno kupovanje proizvoda putem interneta. Sa sve većim brojem ljudi koji prelaze s tradicionalnog načina kupovine u fizičkim trgovinama na online platforme, online shopovi pružaju brojne prednosti. Oni nam omogućavaju pregled širokog raspona proizvoda, poređenje cena, detaljne informacije o proizvodima i sigurnu online transakciju. Internet je potpuno promenio način poslovanja, rušeći geografske granice, jezičke barijere i valutna ograničenja. Omogućena je jednostavna i brza komunikacija, gotovo trenutno prenošenje velikih količina podataka na velike udaljenosti, jednostavno objavljivanje i ažuriranje multimedijalnih dokumenata i njihova kontinuirana globalna dostupnost, digitalna isporuka dobara i usluga, direktno plaćanje putem Interneta, stvaranje virtuelnih organizacija…

Sve to predstavlja elemente novog oblika poslovanja, tzv. ***elektronskog poslovanja*** *(electronic business).* Elektronsko poslovanje je opšti koncept koji obuhvata sve oblike poslovnih transakcija ili razmene informacija koje se izvode korišćenjem informacione i komunikacione tehnologije i to: *između preduzeća, između preduzeća i njihovih kupaca, ili između preduzeća i javne administracije.* Sa aspekta komunikacija: elektronsko poslovanje je elektronska isporuka informacija, proizvoda i usluga i elektronsko plaćanje korišćenjem računarskih i drugih komunikacijskih mreža. Sa poslovnog aspekta: to je primena tehnologije u svrhu automatizacije poslovnih transakcija i poslovanja. Sa stanovišta usluga: to je alat koji omogućava smanjenje troškova poslovanja uz istovremeno povećanje kvaliteta i brzine pružanja usluga.

Elektronsko poslovanje uključuje i ***elektronsku trgovinu*** *(electronic commerce)* koja opisuje proces kupovine, prodaje, transfera ili razmene proizvoda, usluga ili informacija putem kompjuterskih mreža, uključujući i Internet. Elektronska trgovina je skup neopipljivih veza koje održavaju ekonomski agenti. Ova definicija podrazumeva bilo koju transakciju koja se odvija preko Interneta, telefona, bankarske mreže, itd. kao i bilo koji drugi metod plaćanja nezavisno od toga da li se koristi stvarni ili elektronski novac. E-trgovina se može posmatrati sa šireg i užeg aspekta, pa tako šira definicija obuhvata razmenu poslovnih informacija, održavanje poslovnih odnosa i vođenje poslovnih transakcija sredstvima telekomunikacionih mreža. A uža definicija obuhvata kupovinu i prodaju robe, usluga i informacija putem mreže. To znači da je e-trgovina pojavljujući koncept koji opisuje procese kupovine i prodaje, odnosno razmenu proizvoda, usluga i informacija putem kompjuterskih mreža uključujući i Internet. Elektronsku trgovinu možemo definisati i iz perspektive: komunikacija – kao isporuka robe, servisa, informacija ili isplata preko računarske mreže; trgovine – omogućavanje kupovine i prodaje robe, servisa, informacija preko Interneta. Pod pojmom elektronske trgovine (*e-commerce*), sastavnim delom elektronskog poslovanja (*e-business*), obično se podrazumevaju operacije kupovine i prodaje proizvoda i usluga, koje se obavljaju preko Interneta. Ali *koncept elektronske trgovine je mnogo širi i ne ograničava se samo na Internet*. Kupovina putem Interneta je samo deo elektronske trgovine, što znači da je Internet trgovina uži pojam od elektronske trgovine.

Kupovina preko Interneta može biti najbrži, a često i najjeftiniji način kupovine nekog proizvoda. Trgovina preko Interneta se smatra najprofitabilnijim oblikom trgovine zbog jednostavnosti i niskih troškova. Kupovina je moguća u bilo koje doba dana ili noći, 365 dana u godini, nema ograničenja samo na kupce iz komšiluka, ne plaća se zakup poslovnog

prostora... Bilo gde da se nalazi, prodavac postavljajući svoj proizvod na Internet omogućuje ostalim korisnicima Interneta da vide njegov proizvod i ako su hiljadama kilometara daleko. Uspostavljanje klasičnog maloprodajnog lanca zahteva velika ulaganja u infrastrukturu, prostor, zaposlene i prateću opremu. Organizovanje Internet maloprodaje dosta je jeftinije, s obzirom da sve fizičke trgovine zamenjuje jedan Internet sajt. Kada je u pitanju "*online*" kupovina u Srbiji, korisnici još uvek nemaju dovoljno poverenja kako bi koristili svoje platne kartice na Internetu. Primena elektronske kupovine i prodaje, kasni u Srbiji iz dva razloga: *prvi je finansijske prirode*
– standard većine stanovništva u Srbiji ne pokriva čak ni osnovne životne potrebe tako da je višak novca, koji bi ostao na računu i bio iskorišćen za kupovinu putem Interneta, puka teorija. *Drugi razlog je pasivnost domaće privrede prema Internetu* – u većini slučajeva, Internet se koristi za poslovnu komunikaciju unutar samih preduzeća. Pojavom virtuelnih prodavnica na Internetu, stvoreno je novo "*online*" tržište koje ne poznaje granice.

Elektronsko poslovanje i elektronska trgovina je isprva prihvaćeno sa velikom dozom skepticizma, prvenstveno zbog prelaska na novu tehnologiju. Ali problem koji elektronsko poslovanje i elektronsku trgovinu prati od nastanka do danas je problem ***sigurnosti.*** Ovaj problem potiče od samih karakteristika računarskih mreža na koje se oslanja suština elektronskog poslovanja i elektronske trgovine. I danas se radi na novim rešenjima koje će obezbediti maksimalnu sigurnost. Sigurnost je proces održavanja prihvatljivog nivoa rizika. Onaj koji napada sistem bira sredstvo, napada u određeno vreme i na određenom mestu, dok onaj koji brani, koristi sva raspoloživa sredstva u svako vreme i na svakom mestu. Zbog ovoga pojam bezbednosti je apstraktan i ne postoji apsolutna sigurnost. Ali to nije razlog da elektronsko poslovanje i elektronska trgovina stagniraju. Današnji sigurnosni mehanizmi štite sve učesnike u transakciji čak i kada se krađa dogodi.

Ključne rieči: web online shop, e-trgovina, razvoj aplikacija, korisničko sučelje, performanse, sigurnost.

1. Teorijsko metodološke osnove

1.1. Predmet rada

Predmet aplikacije, koja se zasniva na web onlajn prodavnici, je razvoj i implementacija sofisticirane platforme koja korisnicima omogućava pregled, izbor i kupovinu proizvoda putem interneta. Ova aplikacija ima za cilj da pruži intuitivno korisničko iskustvo i olakša proces kupovine na mreži.

Glavne karakteristike i funkcionalnosti ove aplikacije uključuju:

- Korisnički interfejs: Aplikacija će imati jednostavan korisnički interfejs koji će korisnicima omogućiti lak pristup različitim kategorijama proizvoda, pretražuju i filtriraju proizvode i upravljaju korisničkim nalozima.

- Katalog proizvoda: Aplikacija će sadržati detaljan katalog proizvoda koji će korisnicima omogućiti da vide sve dostupne proizvode. Proizvodi će biti organizovani po kategorijama, a korisnici će moći da filtriraju i sortiraju proizvode prema različitim kriterijumima kao što su cena, ocene ili popularnost.

- Porudžbina, Korpa i plaćanje: Korisnici će moći da poruče i dodaju proizvode u korpu, pregledaju sadržaj korpe i izvrše bezbedno plaćanje putem različitih načina plaćanja kao što su kreditne kartice, PayPal ili onlajn bankarstvo.

- Korisnički nalog: Svaki korisnik će imati svoj korisnički nalog sa ličnim podacima, istorijom porudžbina i opcijama za upravljanje podešavanjima naloga. Ovo će korisnicima omogućiti brz pristup njihovim prethodnim porudžbinama i pružiti personalizovano iskustvo.

- Recenzije i ocene: Aplikacija će omogućiti korisnicima da ostave recenzije i ocene proizvoda kako bi pomogli drugim korisnicima da donesu informisanu odluku o kupovini.

- Bezbednost i zaštita podataka: Aplikacija će primeniti mere bezbednosti kako bi zaštitila korisničke podatke i obezbedila sigurnost transakcija. Ovo će uključivati upotrebu bezbednih protokola za prenos podataka i šifrovanje osetljivih informacija.

- Praćenje porudžbina: Korisnici će moći da prate status svojih porudžbina, uključujući praćenje isporuke i primanje obaveštenja o promenama statusa porudžbine.

Svrha ove aplikacije je da korisnicima pruži intuitivan i praktičan način naručivanja.

1.2. Naučni ciljevi rada

Naučni ciljevi aplikacije zasnovane na web prodavnici mogu biti pogodni za analiziranje i rešavanje problema u narednim slučajevima upotrebe, pod tim smatramo sledeće:

- Istraživanje ponašanja korisnika: Cilj je analizirati kako korisnici komuniciraju sa aplikacijom, njihove preferencije i obrasce ponašanja prilikom kupovine proizvoda na mreži. Ovo istraživanje može da pruži uvid u načine na koje se korisnici kreću u onlajn prodavnici i da pruži smernice za poboljšanje korisničkog iskustva.

- Procena učinka aplikacije: Cilj je da se proceni brzina učitavanja stranice, odziv korisničkog interfejsa i ukupni učinak aplikacije. Ova evaluacija omogućava identifikaciju mogućih problema i optimizaciju aplikacije za poboljšanje korisničkog iskustva.

- Bezbednost i zaštita podataka: Cilj je istraživanje i sprovođenje bezbednosnih mera kako bi se obezbedila bezbednost korisničkih podataka, zaštitila aplikacija od napada i sprečila zloupotreba informacija. Ova istraživanja mogu pružiti smernice za razvoj bezbednih aplikacija onlajn prodavnica zasnovanih na vebu.

- Optimizacija korisničkog interfejsa: Cilj je istražiti najbolje prakse u dizajnu korisničkog interfejsa za web prodavnicu na mreži kako bi se poboljšalo korisničko iskustvo i olakšala navigacija i interakcija sa aplikacijom. Ove studije mogu pružiti smernice za kreiranje intuitivnog i privlačnog korisničkog interfejsa.

- Poboljšanje personalizacije i preporuka: Cilj je istražiti mogućnosti personalizacije i preporuke proizvoda na osnovu preferencija korisnika, istorije kupovine i drugih relevantnih faktora. Ovo istraživanje može doprineti razvoju naprednih algoritama koji poboljšavaju relevantnost preporuka i prilagođavaju se potrebama korisnika.

- Procena učinka aplikacije: Cilj je da se sprovede evaluacija aplikacije na osnovu povratnih informacija korisnika, mera zadovoljstva korisnika i drugih pokazatelja kako bi se procenio učinak aplikacije i identifikovale mogućnosti za dalje poboljšanje.

Kroz postizanje ovih naučnih ciljeva, aplikacija zasnovana na web onlajn prodavnici može dati važan doprinos istraživanju korisničkog iskustva, sigurnosti i optimizaciji web onlajn prodavnice. Njena upotreba može omogućiti bolje razumevanje potreba i ponašanja korisnika, poboljšati performanse aplikacije, obezbediti sigurnost i zaštitu podataka i optimizovati korisnički interfejs i personalizaciju. Pored toga, procena performansi aplikacije pružiće uvid u njenu delotvornost i omogućiti dalje unapređenje kako bi se zadovoljile potrebe korisnika i ostvarilo uspešno iskustvo kupovine na mreži.

2. Proces razvoja softvera sa aspekta modelovanja i implementacije

Za rešenje ovog domena problema sagradio sam monolitnu arhitekturu aplikacije. Springboot framework je korišćen za implementaciju serverskog dela.

2.1. Funkcionalni zahtevi

Date su tabele funkcionalnih zahteva serverskog dela po sledećim modelima:

➢ Grubi model sistema

ID	Naziv	Opis
0001	Registrovanje Na Sistem	Sistem treba omogućiti registrovanje korisnika.
0002	Logovanje Na Sistem	Sistem treba omogućiti logovanje korisnika sa svojim parametrima.
0003	Organizacija Prava Pristupa	Sistem treba uskladiti dodelu prava pristupa u zavisnosti od kredencijala nakon logovanja.
0004	Pregled Entiteta	Sistem treba omogućiti pregled entiteta.
0005	Uređivanje Entiteta	Sistem treba omogućiti uređivanje entiteta.
0005.1	Brisanje Entiteta	Sistem treba omogućiti brisanje entiteta.
0005.2	Izmena Entiteta	Sistem treba omogućiti izmenu entiteta.
0005.3	Kreiranje Entiteta	Sistem treba omogućiti kreiranje entiteta.
0006	Pretraga Entiteta	Sistem treba omogućiti pretragu entiteta.
0007	FiltriranjeEntiteta	Sistem treba omogućiti filtriranje cene entiteta.
0008	Kreiranje PDF Izveštaja	Sistem treba omoguciti kreiranje izveštaja trenutnog stanja entiteta u vidu tabele.
0009	Zapisivanje Log Podataka	Sistem treba omogućiti ispis log podataka i korisnika koji je to radio nakon poziva entiteta i taj podatak smestiti u Dokument orijentisanu bazu.

Tabela 1. Funkcionalni zahtevi serverskog dela – Grubi model sistema

➢ Administrator(Prodavac) – Model zahteva

ID	Naziv	Opis
R001	Login	Omogućiti registrovanje i logovanje Administratora sa njegovim parametrima.
R002	Vršenje administracije Proizvoda	Omogućiti da admin vrši administraciju sifarnika proizvoda
R002.1	Dodavanje proizvoda	Omogućiti dodavanje novih proizvoda
R002.2	Brisanje proizvoda	Omogućiti brisanje proizvoda.
R002.3	Izmena proizvoda	Omogućiti izmenu proizvoda
R003	Vršenje administracije registrovanih korisnika	Omogućiti da admin vrši administraciju registrovanih korisnika
R003.1	Pregled korisnika	Omogućiti da admin pregleda sve korisnike
R003.2	Brisanje korisnika	Omogućiti da admin vrsi brisanje korisnika
R004	Vrsenje administracije Porudžbine	Omogućiti da admin vrsi administraciju porudžbine
R004.1	Pregled Porudžbine	Omogućiti pregled svih porudžbine kao evidenciju.
R004.2	Brisanje Porudžbine	Omogućiti brisanje porudžbine.
R005	Vršenje administracije placanja	Omogućiti da admin vrši administraciju organizacije placanja
R005.1	Plaćanje karticom	Omogućiti da admin vrši administraciju organizacije placanja karticom.
R005.2	Plaćanje gotovinom	Omogućiti da admin vrši administraciju organizacije placanja gotovinom nakon isporuke.

Tabela 2. Funkcionalni zahtevi serverskog dela – Administrator(Prodavac)

➢ Klijent(Kupac) – Model zahteva

ID	Naziv	Opis
R001	Login	Omogućiti registrovanje i logovanje Korisnicima sa njihovim parametrima
R002	Pregled proizvoda	Omogućiti korisnicima da Pregledaju sve dostupne proizvode
R003	Pretraga proizvoda	Omogućiti korisnicima da pretrazuju proizvode
R004	Filtriranje proizvoda	Omogućiti korisnicima da filtriraju proizvod po dozvoljenim parametrima.
R004.1	Filtriranje po ceni	Omogućiti korisnicima da filtriraju proizvod po ceni u rasponu od do.
R004.2	Filtriranje po nazivu	Omogućiti korisnicima da filtriraju proizvod po nazivu.
R004.3	Filtriranje po tipu	Omogućiti korisnicima da filtriraju proizvod po tipu sorte.
R004.4	Filtriranje po godini	Omogućiti korisnicima da filtriraju proizvod po godini proizvodnje.
R005	Porucivanje proizvoda	Omogućiti korisnicima da porucuju proizvode.
R006	Placanje porudzbine	Omoguciti korisnicima da placaju porucen proizvod
R007	Uklanjanje proizvoda iz korpe	Omogućiti korisnicima da uklanjaju proizvod iz korpe.
R008	Dodeljivanje recenzije za proizvod	Omogućiti korisnicima da dodeljuju ocene i recenzije za proizvode.
R009	Pregled porudžbine	Omogućiti korisnicima da pregledaju porudžbinu.
R010	Otkazivanje porudžbine	Omogućiti korisnicima da otkažu svoju porudžbinu.

Tabela 3. Funkcionalni zahtevi serverskog dela – Klijent(Kupac)

2.2. Nefunkcionalni zahtevi

Tabela nefunkcionalnih zahteva koju sistem mora podržati:

ID	Naziv	Opis
R007	Performanse	Predstavljaju zahteve u vezi sa izvođenjem određenih operacija (brzina odgovora sistema na radnje korisnika). Specificirati: broj transakcija (konkretnih) koje se mogu izvršiti u sekundi, vreme odgovora, preciznost rezultata proračuna. Zatim se može odrediti količina podataka sa kojom sistem mora da se bavi: zahtevi za memorijom, maksimalan broj konkurentskih korisnika, maksimalan broj zapisa u tabeli.
R008	Bezbednost	Specifikacija uključuje zahteve koji se odnose na gubitak i štetu (LJUDIMA, IMOVINI ili ŽIVOTNOJ SREDINI) koji mogu nastati kao rezultat korišćenja sistema. Definisati mere predostrožnosti i radnje koje treba preduzeti, kao i potencijalno opasne radnje koje treba sprečiti. Identifikujte sertifikate, pravila, propise koji se odnose na bezbednost koje sistem mora da poštuje.
R009	Sigurnost	Podacima u sistemu aplikacije neće moći da pristupaju svi korisnici. Samo vlasnik, odnosno Admin.
R010	Pouzdanost	Do iznenadnog prestanka rada sistema može doći jedino kao posledica grešaka u radu operativnog sistema korisnika ili problema vezanih za funkcionisanje servera.

Tabela 4. Nefunkcionalni zahtevi serverskog dela

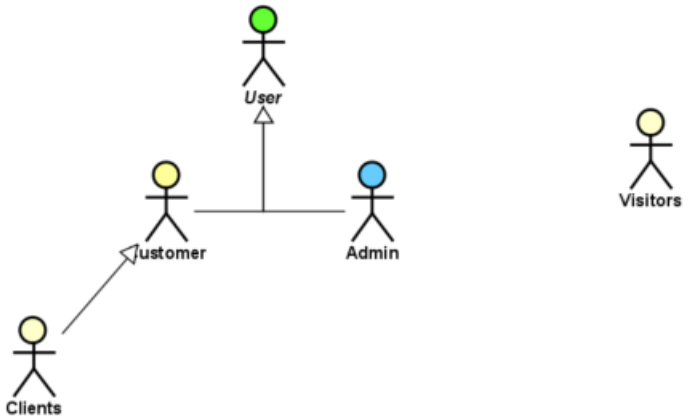
2.3. Model Interakcije

Model interakcije obezbeđuje osnovnu strukturu ili nacrt za to kako se proizvod ili sistem ponašaju na osnovu poznatog ponašanja korisnika.

2.3.1. Dijagrami slučajeva korišćenja

Ovo poglavlje ima za cilj da opiše slučajeve korišćenja sistema uz priložene dijagrame. Biće definisani po sledećim fazama upotrebe:

- **Hijerarhija aktera**



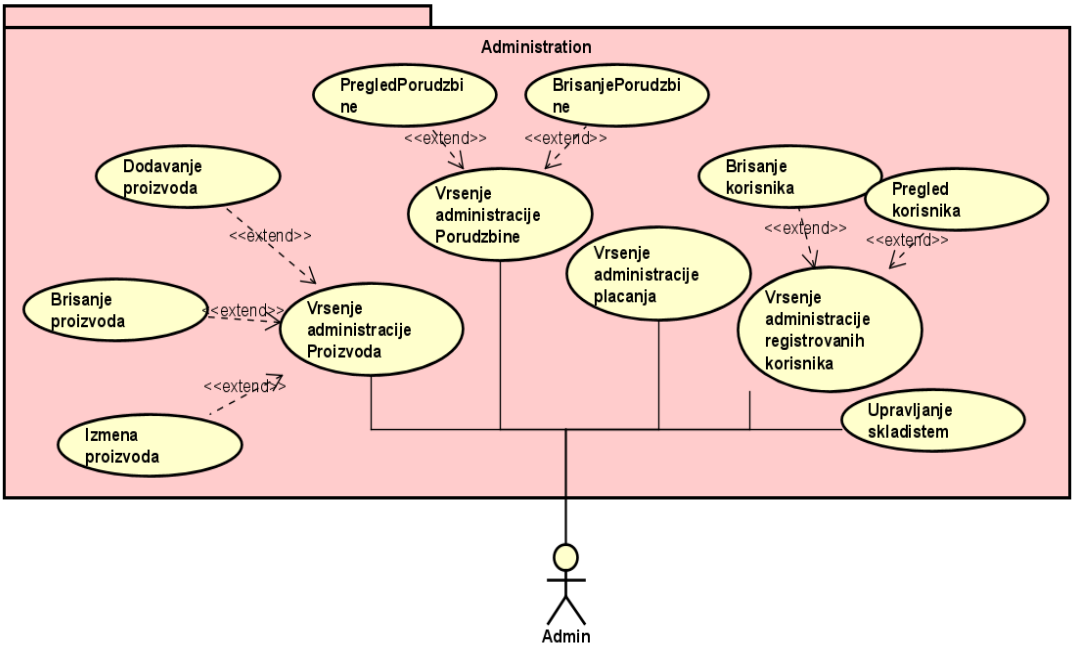
Slika 1. UseCase diagram Hijerarhija aktera

Sa slike 1. može se zaključiti:

Na dijagramu je predstavljena hijerarhija korisnika koja pristupa aplikaciji. Dakle, imamo korisnika koji je zapravo apstraktnog tipa što znači da njegovu ulogu nasleđuju Customer (Kupac koji podrazumeva da su to novi klijenti kao potencijalni kupci porizvoda) i Admin(Administrator/Prodavac). Naravno, oba korisnika imaju svoja prava pristupa određenim funkcionalnostima aplikacije o čemu ćemo malo detaljnije u nastavku obrade teme.
Takođe, imamo i posetioce koji imaju pristup početnoj strani aplikacije.

➢ Prava administratora

-Korisnik: **Admin**



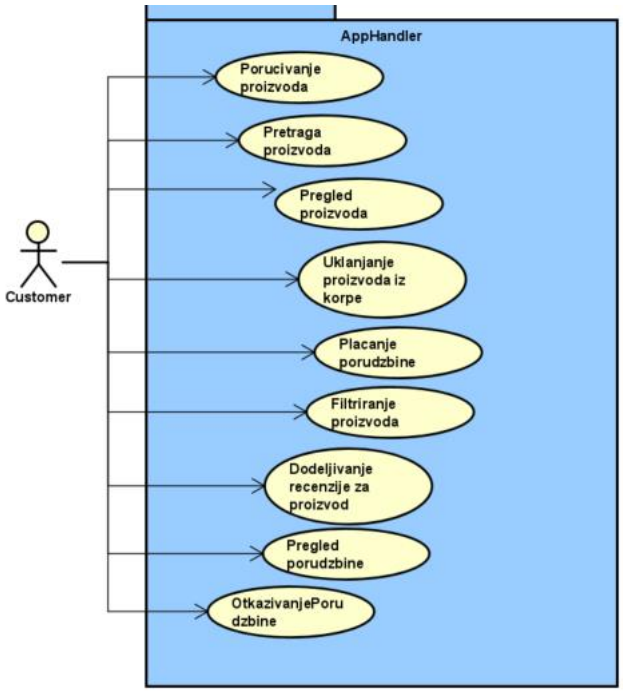
Slika 2. UseCase diagram AdminRules

Sa slike 2. može se zaključiti:

- **Preduslov**: Korisnik mora biti ulogovan na svoj nalog sa permisijom: ROLE_ADMIN.
- **Postupak**: Administrator ima sva prava na sistemu koja su u dodiru sa uređivanjem poslovnog sistema, kao i upravljanju svim proizvodima. Naravno ima i ulogu u magacinskom poslovanju, odnosno, upravljanu skladištem.

➢ Prava klijenta(Kupaca)

-Korisnik: **Customer**



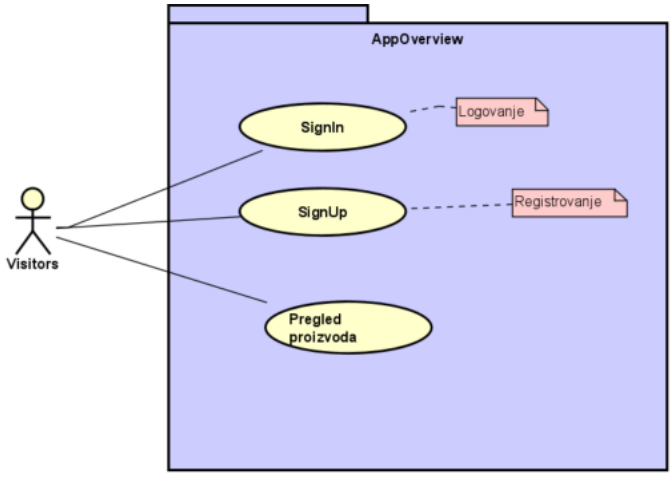
Slika 3. UseCase diagram CustomerRules

Sa slike 3. može se zaključiti:

- **Preduslov**: Korisnik mora biti ulogovan na svoj nalog sa permisijom: ROLE_CUSTOMER.
- **Postupak**: Kupac ima sva prava na sistemu koja su u skladu sa njegovima potrebama kako bi izvršio sledeće korake:
  - Mogućnost da pregleda dostupne proizvode,
  - Mogućnost da pretraži i filtrira proizvod,
  - Mogućnost da poruči proizvod i stavi u svoju korpu,
  - Mogućnost da plati karticom ili gotovinom,
  - Mogućnost da oceni sistem nakon uspešnog završetka rada,
  - Mogućnost da pregleda svoju porudžbinu,
  - Mogućnost da otkaže svoju porudžbinu.

➢ Neregistrovani korisnici(Posetioci)

-Korisnik: **Visitors**



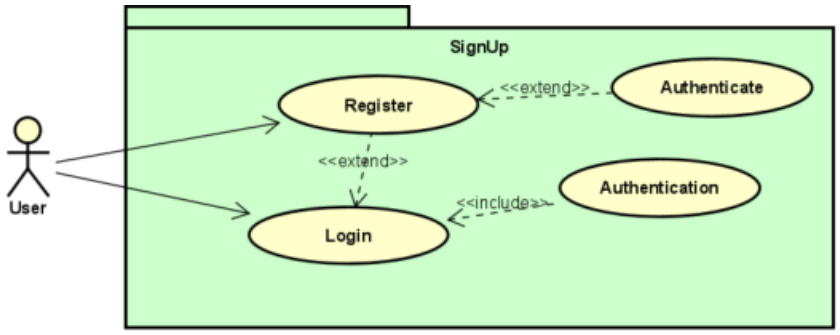
Slika 4. UseCase diagram Visitors

Sa slike 4. može se zaključiti:

- **Preduslov**: Posetilac ne mora biti ulogovan.
- **Postupak**: Posetilac može pristupiti pregledu proizvoda ili početnoj stranici aplikacije. Naravno, nudi mu se mogućnost registrovanja kako bi mogao da izvrši neku radnju u aplikaciji.

➢ Prijava na sistem

-Korisnik: **User**



Slika 5. UseCase diagram SignUp

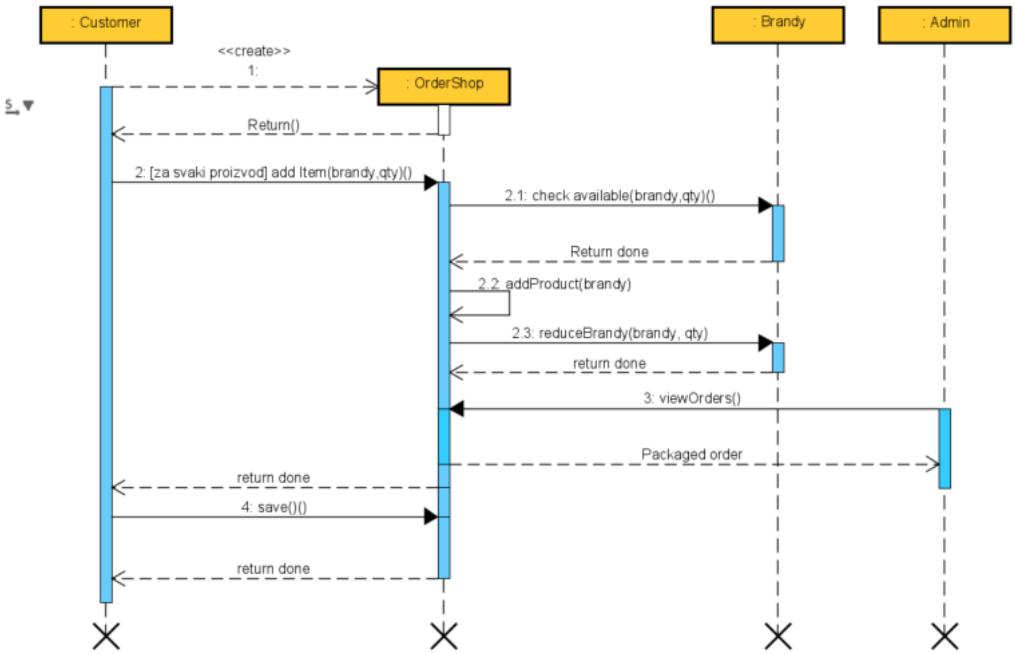
Sa slike 5. može se zaključiti:

- **Preduslov**: Korisnik pristupa sistemu.
- **Postpuak**: Korisnik može pristupiti sledećim slučajevima upotrebe:
  - Registrovanje - Prilikom registrovanja proširuje se slučaj uvođenjem autentikacije koja utvrđuje identitet korisnika.
  - Logovanje – Prilikom logovanja, naravno postoji proširiva mogućnost i registrovanja ukoliko korisnik nije registrovan. Ako je registrovan i izvršava logovanje automatski se uključuje i slučaj autentikacije koji proverava identitet korisnika.

2.3.2. Dijagram sekvenci

U ovom delu prikazaću Vam ponašanje koje obuhvata skup poruka koje se razmenjuju između skupa objekata u nekom kontekstu sa nekom namenom. Poruka predstavlja komunikaciju između objekata koja prenosi informaciju.

➢ Interakcija aplikacije



Slika 6. Sequence Diagram – ApplicationSystem

Na slici 6. možemo zaključiti:

Primer dijagrama sekvence prikazuje tri objekta koji učestvuju: „Customer“, „OrderShop“ i „Brandy“. Čak i bez formalnog poznavanja notacije, verovatno možete dobiti prilično dobru predstavu o tome šta se dešava.

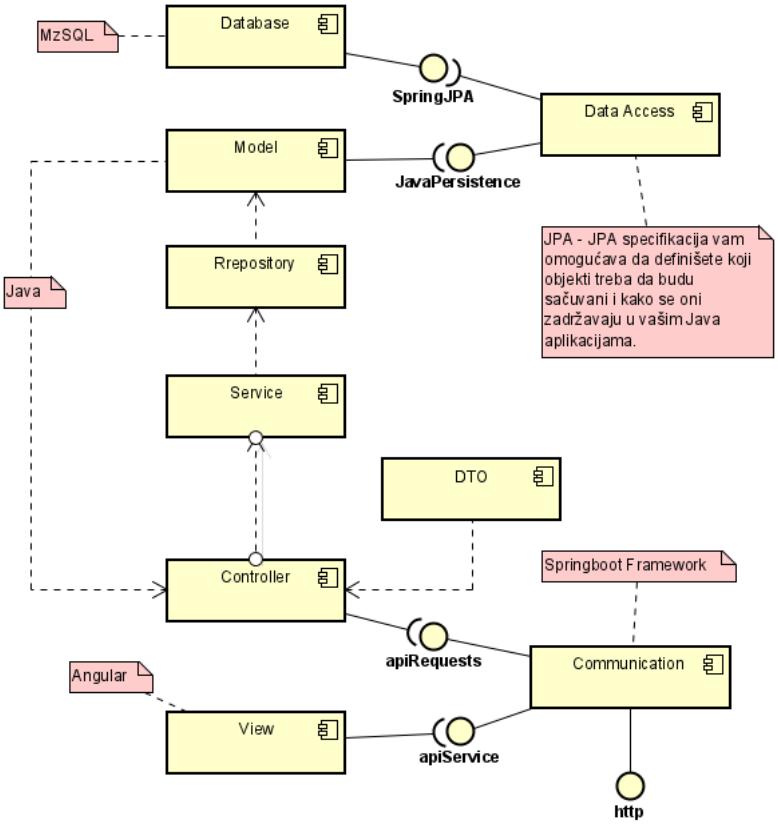
- Korak 1 : Kupac kreira porudžbinu.
- Korak 2: Kupac dodaje stavke(proizvode) u porudžbinu.
- Korak 2.1: Svaka stavka se proverava da li je dostupna.
- Korak 2.2: Ako je proizvod dostupan, dodaje se narudžbini.
- Korak 2.3: Redukovanje stanja u magacinu.
- Korak 3: Uvid u porduzbine koju je kupac izvršio.
- Korak nakon dodavanja: Povratak
- Korak 4: Sačuvajte i uništite red

2.4. Model statičke strukture

U nastavku biće prikazani dijagram komponenti koji predstavljaju svaku tehnologiju koja se koristi za razvoj sistema, takođe tu su i dijagrami klasa koji prikazuju skup klasa, interfejsa, saradnji i drugih stvari strukture, povezanih relacijama. Odlučio sam da ovaj segment podelim na dve celine kako bih detaljnije prikazao obe strane statičke strukture aplikacije.

2.4.1. Dijagram komponenti – Komunikaciona arhitektura

U prilogu data je slika koja predstavlja komunikaciju između komponenti kao arhitekture cele aplikacije.



Slika 7. Component diagram - CommunicationSystem

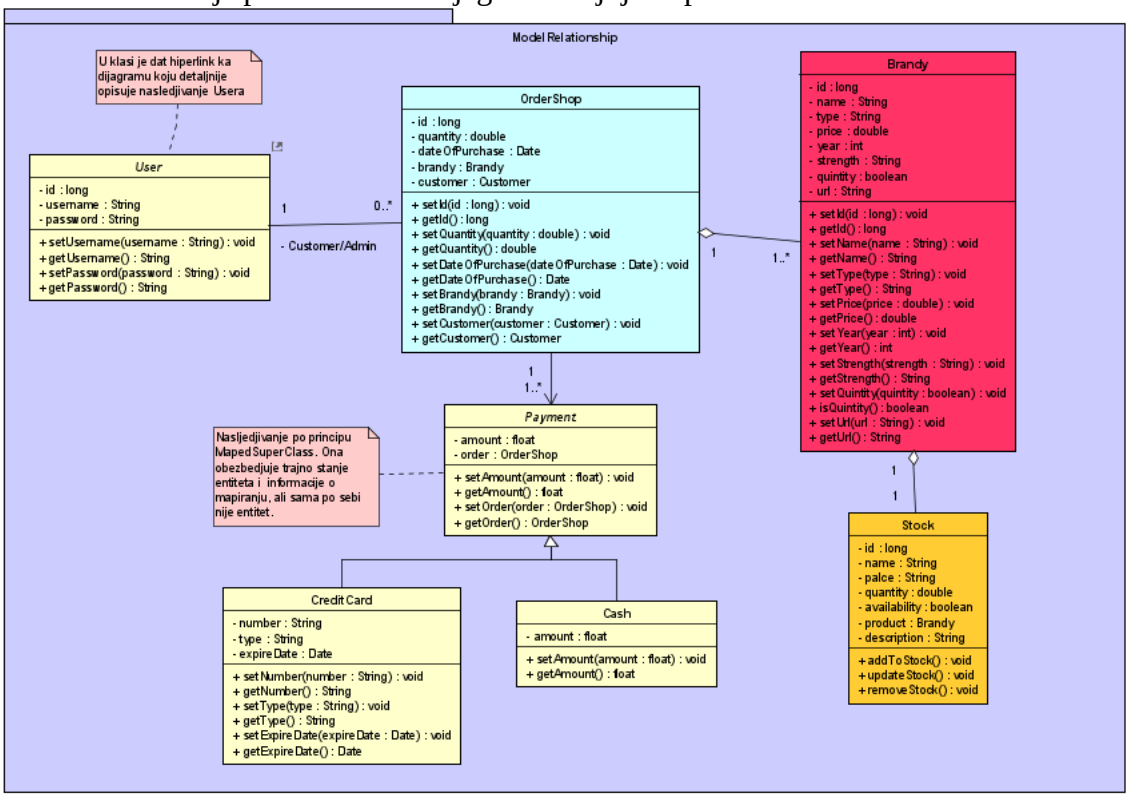
Komponente web aplikacije (pogledajte sliku 7) komuniciraju kroz Springboot framevork. Dakle, Java tehnologija podržava sve modele i kontrolere. Zahtevi za performanse kontrolora prenose se preko web servisa (RESTful) povezanih sa entitetima putem pristupa podacima. Baza podataka je konstruisana pomoću MySQL-a, kome se pristupa preko Java Persistence API-ja (JPA). Taj mehanizam vam omogućava da definišete koji objekti treba da budu sačuvani i kako se oni zadržavaju u vašim Java aplikacijama.

2.4.2. Klasni dijagram - Arhitektura aplikacije

U nastavku biće prikazani klasni modeli koji detaljno predstavljaju arhitekturu sistema i relacije između objekata kojim se bavimo.

➢ Model aplikacije

Na slici 8. dat je prikaz klasnih dijagrama koji je implementiran na backend-u:



Slika 8. Class diagram - ModelApp

Sa slike 8. može se zaključiti:

Klase:

- ***User*** je akter koji zapravo opisuje korisnika koji pristupa sistemu. Od važnijih atributa imamo *username* i *password* koji predstavljaju unosne parametre za logovanje na sistem. Od operacija imam generisane getere i setere za sve prethodno navedene atribute koji služe za dobavljanje vrednosti tamo gde je to potrebno. Kao što možemo videti ovo je apstraktna klasa gde je implementiran veoma zanimljiv način nasleđivanja i iz tog razloga želeo sam tu temu da odvojim u zasebnu stavku rada(detaljnije na slici 11.).

- **Brandy** je entitet koji suštinski predstavlja naš proizovd. U njoj se nalaze privatni ključ i atributi koji detaljnije opisuju objekat. Od važnijih osobina u njoj se nalazi mapiranje klase Brandy u kolekciju objekta OrderShop koja će zapravo biti roditeljska klasa upravo zbog tog mehanizma mapiranja i hibernate-a koji definiše tu promenljivu. U ovom primeru želeo bih da Vam predstavim kako to direktno izgleda u kodu:

```
@OneToMany(mappedBy = "brandy")
private Set<OrderShop> orders = new HashSet<OrderShop>();
```

Naravno kao i svuda od operacija imam generisane getere i setere za sve prethodno navedene atribute koji služe za dobavljanje vrednosti tamo gde je to potrebno.

- **Stock** predstavlja entitet skladišta proizvoda u aplikaciji. U njoj se nalaze privatni ključ i atributi koji detaljnije opisuju objekat. Od važnijih osobina u njoj se nalaze količina, dostupnost i lista proizvoda(Brandy). Ova klasa je inicijalno predstavljena kako bi prikazao i sistem skladištenja. Dakle, moguće je dodavanje zaliha kao i ažuriranje nakon svake porudžbine.

- **OrderShop** predstavlja entitet porudžbine u aplikaciji. Ova klasa ima svoj privatni ključ naravno koji se generise pomoću antoacije *@GeneratedValue(strategy = GenerationType.IDENTITY)*. Svakako tu su i ostali atributi kao što vidite na slici 8. koji upotpunjuju objekat kako bih imao sve potrebne informacije o porudžbini. Od važnijih segmenata u klasi nalaze se promenljive brandy i customer. U jednostavnom prevodu to znači da su oni pripadnici entiteta OrderShop. Taj mehanizam je implementiran relacionim mapiranjem kao i u prethodnoj klasi Brandy ali sa drugčijom perspektivom. Dakle, korišćena je relacija @ManyToOne i to izgleda u demonstraciji ovako:

```
@ManyToOne(optional = false)
private Customer customer;
@ManyToOne(optional = false)
private Brandy brandy;
```

  Odnos „ManyToOne“ je tip odnosa između dve klase entiteta u kontekstu ORM (Object-Relational Mapping) sistema kao što je Spring Boot. Ova veza znači da se jedan objekat jedne klase može povezati sa više objekata druge klase, dok svaki objekat druge klase može biti povezan samo sa jednim objektom prve klase.
  U ovom primeru, klasa „OrderShop“ ima odnos „ManyToOne“ sa klasom „Customer“ i klasom „Brandy“. To znači da se jedna porudžbina (objekat „OrderShop“) može povezati sa jednim korisnikom (objekat „Customer“) i jednom vrstom pića (objekat „Brandy“), dok se jedan korisnik ili jedna vrsta pića može povezati sa više porudžbina.

    - Relacije klasnog dijagrama:
      - Asocijacija:
        - Neodređena asocijacija koja predstavlja povezanost sa objektom *User* .
        - Usmerena asocijacija koja predstavlja povezanost sa objektom *Payment* .
      - Agregacija:
        - Relacija ka objektu „Brandy“ ukazuje na to da jedan objekat entiteta „OrderShop“ sadrži referencu na objekat entiteta „Brandy“. To znači da objekat „OrderShop” koristi objekat „Brandy” kao njegov deo, ali objekat „Brandy” može postojati i sam, nezavisno od objekta „OrderShop”.

- ***Payment*** je objekat koji predstavlja sistem plaćanja. Kao što možemo i primetiti ona je roditeljska klasa koja ima svoje izvedene klase(naslednice). To su „CrediCard“ i „Cash“. One naravno preuzimaju sve osobine roditelja.
  Kako bih se upoznao sa što više primenjenih veština i tehnologija prilikom izrade diplomskog rada odlučio sam da u ovom segmentu plaćanja primenim drugačiji način nasleđivanja kako ne bi bilo identično gradivo kao i kod slučaja modela Usera.
  Naime, odlučio sam da implemenitram strategiju pod nazivom *MappedSuperclass*.

Taj princip u JPA (Java Persistence API) omogućava kreiranje zajedničke superklase koja sadrži zajedničke atribute i metode za druge klase entiteta. Ta superklasa je označena napomenom „@MappedSuperclass“. Primer definisane klase:

```
@MappedSuperclass
public class Payment {}
```

U ovom primeru, klasa „*Payment*“ je definisana kao „MappedSuperclass“. To znači da ona neće biti direktno mapirana u bazu podataka kao nezavisna tabela, ali će njene atribute i metode naslediti klase entiteta koje je koriste(CreditCard i Cash).
Kada klasa entiteta nasledi klasu "MappedSuperclass", ona dobija sve atribute i metode definisane u superklasi. Klasa entiteta može dopuniti ili zameniti ove atribute i metode, ali i dalje nasleđuje osnovne karakteristike iz klase "MappedSuperclass".
Ova tehnika nasleđivanja se često koristi kada postoji skup zajedničkih atributa ili metoda koji bi trebalo da budu dostupni za više klasa entiteta. Na primer, u slučaju klase „*Payment*“, ona može da sadrži zajedničke atribute kao što su broj transakcije, iznos plaćanja, datum i slično, koje će naslediti sve klase entiteta koje predstavljaju različite vrste plaćanja.
Važno je napomenuti da klase entiteta koje nasleđuju klasu „MappedSuperclass“ moraju biti mapirane u bazu podataka korišćenjem drugih napomena, kao što je „@Entiti“, kako bi se osiguralo da su podaci pravilno uskladišteni i da im se pristupa.

- **CreditCard** je naslednik klase „*Payment*” koja je označena napomenom „@Entity” i dodatno je definisana napomenom „@Table” koja navodi naziv tabele u bazi („credit_card”). Primer implementacije:

```
@Entity
@Table(name = "credit_card")
public class CreditCard extends Payment{}
```

Nasleđivanje omogućava klasi „CreditCard“ da nasledi sve atribute i metode iz nadređene klase „Plaćanje“. To znači da će klasa „CreditCard“ naslediti uobičajene atribute kao što je iznos plaćanja i referenca za porudžbinu iz klase „Payment“.
Klasa "CreditCard" se koristi za predstavljanje informacija o kreditnoj kartici koja se koristi za plaćanje. Ova klasa može da sadrži dodatne specifične atribute vezane za kreditnu karticu, kao što su broj kartice, tip, datum isteka.
Da bi se informacije o kreditnoj kartici sačuvale i pristupile u bazi podataka, potrebno je dodatno mapiranje klase entiteta „CreditCard“ korišćenjem odgovarajućih JPA napomena kao što su „@Column“, „@Id“ i druge, u zavisnosti od zahteva aplikacije i baze podataka.
Na slici 9. dat je prikaz kako izgleda rezultat tabele u MySQL bazi nakon generisanja:

Naziv tabele: credit_card

id	amount	expire_date	number	type
1	3000	2023-09-09	1234-333	Visa

Slika 9. MySQL Workbench – CreditCardTable

  - Relacije klasnog dijagrama:
    - Generalizacija:
      - Relacija objekata koja preuzima (nasleđuje) obeležja i operacije klase „*Payment*“.

- **Cash** je naslednik klase „*Payment*“ koja je označena anotacijom „@Entiti“ i dodatno je definisana napomenom „@Table“ koja navodi naziv tabele u bazi podataka („cash“). Primer implementacije:

```
@Entity
@Table(name = "cash")
public class Cash extends Payment{}
```

Nasleđivanje omogućava klasi „Cash“ da nasledi sve atribute i metode iz nadređene klase „Payment“. To znači da će klasa „Cash“ naslediti uobičajene atribute kao što je iznos plaćanja i referenca za porudžbinu iz klase „*Payment*“.
Klasa "Cash" se koristi za predstavljanje informacija o gotovinskom plaćanju. Pošto gotovinsko plaćanje ne uključuje dodatne specifične informacije kao što su broj kartice ili ime vlasnika, klasa „Cash“ je relativno jednostavna.
Da bi se podaci o gotovinskom plaćanju pohranili i pristupili u bazi podataka, potrebno je dodatno mapiranje klase entiteta "Cash" koristeći odgovarajuće JPA napomene kao što su "@Column", "@Id" i druge, u zavisnosti od aplikacije i baze podataka.
Na slici 10. dat je prikaz kako izgleda rezultat tabele u MySQL bazi nakon generisanja:

Naziv tabele: cash

id	amount	cash_tendered
1	3000	3000

Slika 10. MySQL Workbench – CashTable

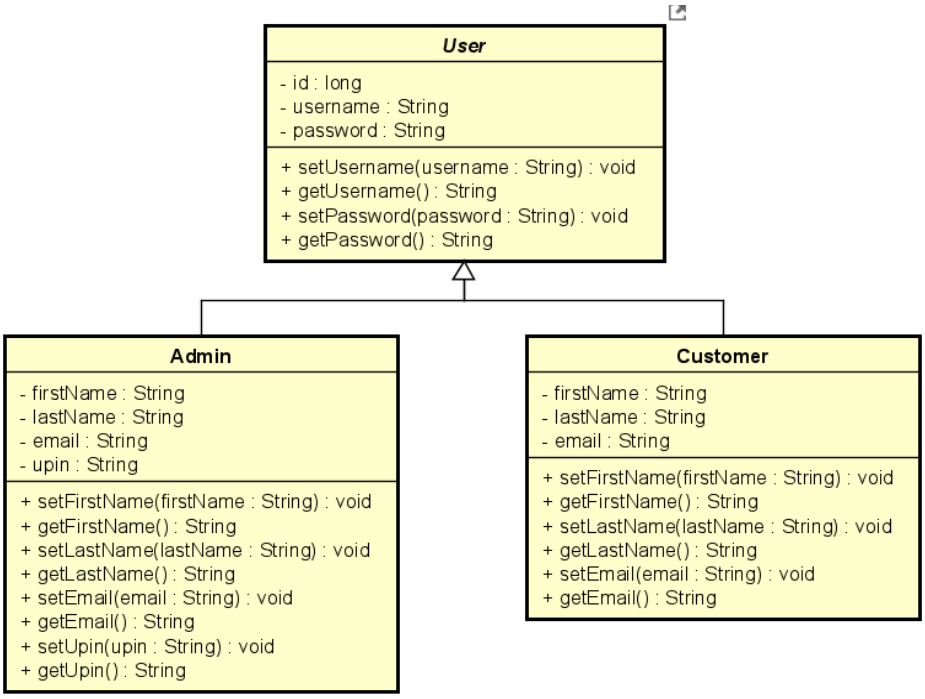
  - Relacije klasnog dijagrama:
    - Generalizacija:
      - Relacija objekata koja preuzima (nasleđuje) obeležja i operacije klase „*Payment*“.

2.4.3. Klasni dijagram - Autentikacioni sistem

U nastavku biće prikazani klasni modeli koji detaljno predstavljaju autentitkacioni sistem korisnika i relacije između objekata kojim se bavimo.

➢ Model korisnika

Na slici 11. dat je prikaz klasnog dijagrama(Detaljniji prikaz User-a sa slike 8.) koji je implementiran na backend-u:



Slika 11. Class diagram - User

Sa slike 11. može se zaključiti:

U Spring Boot-u, nasleđivanje se može primeniti korišćenjem principa jedne tabele kada se radi sa relacionom bazom podataka. Ovaj princip vam omogućava da modelujete hijerarhiju klasa gde više podklasa deli zajedničku tabelu baze podataka.

U mom scenariju imam osnovnu klasu koja se zove **„*User*“** i dve podklase koje se zovu **„Customer“** i **„Admin“**. Da biste ovo implementirali koristeći princip nasleđivanja 'SINGLE_TABLE', možete kreirati jednu tabelu u bazi podataka koja predstavlja klasu User i koja uključuje kolone za zajedničke atribute koje dele sve podklase. Dodatne kolone specifične za svaku podklasu takođe mogu biti uključene.U nastavku želim da Vam demonstriram kako to izgleda u implementaciji:

User:

```
@Entity(name = "User")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public class User {}
```

Admin:

```
@Entity
public class Admin extends User {}
```

Customer:

```
@Entity
public class Customer extends User {}
```

U gornjem primeru, napomena **@Inheritance** sa strategijom = **InheritanceType.*SINGLE_TABLE*** ukazuje da je strategija nasleđivanja zasnovana na principu jedne tabele. U toj tabeli postoji i nešto što nazivamo diskriminatorom koji predstavlja zapravo kolonu koja je ispunjena vrednostima podklase, da li je tu upisan Admin ili Customer. To je suština i tako razlikujemo podklase. Naravno postoji i mehanizam anotacije **@DiscriminatorColumn** koja se koristi za specifikaciju naziva kolone koja će se koristiti, a **@DiscriminatorValue** napomena se koristi da obezbedi jedinstvenu vrednost za svaku podklasu u koloni diskriminatora. Ukoliko ne koristimo pomenute anotacije, podrazumevan naziv u tabeli za diskriminator je **'dtype'**.

Sa ovim podešavanjem, kada sačuvate instance klase Customer ili Admin, one će biti uskladištene u istoj tabeli korisnika sa vrednošću diskriminatora koja ukazuje na njihov specifični tip. Kada preuzima podatke iz tabele, Spring Boot će automatski kreirati instance odgovarajuće podklase na osnovu vrednosti diskriminatora.

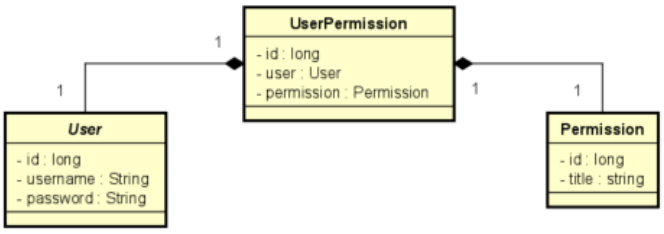
Na slici 12. dat je prikaz kako izgleda rezultat tabele u MySQL bazi nakon generisanja:

dtype	id	password	username	email	first_name	last_name	upin
Admin	4	{bcrypt}$2a$10$yTw0Ix81GwlMprPbYBZ3DulqT...	predrag10	predrag@gmail.com	Predrag	Radak	2102939123
Customer	6	{bcrypt}$2a$10$C8Y38HKXgcS//zJ7JSdCj.C9O...	pera	perar@gmail.com	Petar	Peric	NULL

Slika 12. MySQL Workbench - UserTable

➢ Model Prava pristupa

Na slici dat je prikaz klasnog dijagrama koji je implementiran na backend-u sa ciljem da podeli prava pristupa aplikaciji:



Slika 13. Class diagram – UserPermission

Na slici 13. možemo zaključiti:

Da bi se dodelila prava pristupa za ove tri klase, koristi se veza između klase User i klase Permission preko klase UserPermission, koja predstavlja asocijativnu klasu. Imajući to na umu, evo kako se dodeljivanje prava pristupa može uraditi:

Klasa User(Registrovani korisnik):

- Klasa User predstavlja korisnika aplikacije.
- Svaki objekat klase User ima jedinstveno korisničko ime i lozinku.
- Korisnik takođe sadrži kolekciju objekata klase UserPermission, koji predstavljaju dozvole dodeljene korisniku.
- Koristeći ovu kolekciju, možete pristupiti svim dozvolama dodeljenim određenom korisniku.

Klasa Permission(Uloga):

- Klasa Permission predstavlja dozvolu ili pravo pristupa u sistemu.
- Svaki objekat klase Permission ima naslov koji opisuje dozvolu.
- Svaka dozvola takođe sadrži kolekciju objekata klase UserPermission, koji predstavljaju korisnike kojima je ta dozvola dodeljena.

Klasa UserPermission:

- Klasa UserPermission predstavlja vezu između klase User i klase Permission.
- Svaki objekat klase UserPermission ima referencu na objekat klase User i objekat klase Permission preko @ManyToOne relacije.
- Na osnovu ovih referenci, možete pristupiti korisniku i permisiji.

Na slici je prikazan rezultat tabele UserPermission u MySQL bazi podataka:

id	permission_id	user_id
1	1	4
3	2	6
5	2	8

Slika 14. MySQL Workbench – UserPermission

Zaključak: Važno je napomenuti da specifična implementacija prava pristupa u vašoj aplikaciji zavisi od poslovnih pravila i logike koju želite da primenite.Konkretno za ovaj slučaj upotrebio sam dve uloge: ROLE_ADMIN i ROLE_USER koje se nalaze u skladištu tabele Permission. Kao što je i već rečeno klasa UserPermission je zapravo veza koja spaja te uloge i User-a.

2.5. Obezbeđenje sistema(Spring Security with JWT)

Spring Security je moćan framework koji pruža podršku za autentifikaciju i autorizaciju za Java aplikacije. Pomaže u zaštiti aplikacije tako što kontroliše pristup resursima i štiti od uobičajenih bezbednosnih pretnji.

JWT (JSON Web Token) je popularan mehanizam autentifikacije zasnovan na tokenima koji se često koristi sa Spring Security-om. Omogućava klijentu da se autentifikuje i dobije token, koji se zatim šalje sa svakim narednim zahtevom za pristup zaštićenim resursima.

Da bismo razumeli integraciju Spring Security-a sa JWT-om, razmotrimo jednostavan primer slučaja koji je pogodan kao podloga za implementaciju u aplikaciji:

Registracija korisnika: Kada se korisnik registruje u aplikaciji, njegovi akreditivi (npr. korisničko ime i lozinka) se čuvaju bezbedno, obično u bazi podataka.

Autentifikacija korisnika: Kada korisnik pokuša da se prijavi, on daje svoje akreditive. Spring Security mehanizam za autentifikaciju proverava ove akreditive u odnosu na sačuvane podatke.

Generisanje tokena: Kada se korisnik uspešno autentifikuje, generiše se JWT token. Ovaj token uključuje informacije o korisniku, kao što su korisničko ime ili ID korisnika, zajedno sa vremenom isteka i digitalnim potpisom.

Izdavanje tokena: generisani JWT token se vraća klijentu kao deo autentifikacionog odgovora, obično kao zaglavlje odgovora ili u telu odgovora. Klijent bezbedno čuva ovaj token za sledeće zahteve.

Autorizacija zasnovana na tokenu: U narednim zahtevima, klijent uključuje JVT token u zaglavlja zahteva, obično u zaglavlju „Ovlašćenja“ sa prefiksom „Nosilac“. Server proverava integritet i autentičnost tokena proverom digitalnog potpisa i osigurava da nije istekao.

Kako bih implementirao Spring Security sa JWT-om, izvršio sam sledeće korake:

-Konfiguracija Spring Security: Podesio sam potrebne konfiguracije u svom konfiguracionom fajlu i klasi Spring Security da bih omogućio autentifikaciju zasnovanu na JWT-u. Ovo uključuje navođenje krajnje tačke autentifikacije, omogućavanje autentifikacije zasnovane na tokenima i definisanje bezbednosnih pravila.Ovako izgleda konfiguracioni fajl:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.0</version>
```

-Generisanje tokena: Implementirao sam komponentu koja generiše JWT tokene nakon uspešne autentifikacije korisnika. Ova komponenta treba da sadrži neophodne informacije o korisniku, vreme isteka i digitalni potpis.
Ovako izgleda opis klase:

```
@Component
public class TokenUtils {
    @Value("${token.secret}")
    private String secret;

    @Value("${token.expiration}")
    private Long expiration;
//U nastavku naravno nalze se metode koje definišu potrebne
informacije
}
```

-Validacija tokena: Konfigurisao sam filter ili presretač koji presreće dolazne zahteve i potvrđuje JWT token uključen u zaglavlja zahteva. Proverava integritet, autentičnost i rok trajanja tokena. Ako je token ispravan, dozvoljava da se zahtev nastavi, u suprotnom, zabranite pristup. Ovako izgleda klasa:

```
public class AuthenticationTokenFilter extends
UsernamePasswordAuthenticationFilter {}
```

-Bezbedni resursi: Definišem pravila pristupa resursima aplikacije na osnovu korisničkih uloga ili dozvola. Odlučio sam se da taj deo logike primenim u servisu. Klasa će naslediti interfejs *UserDetailsService* koji dolazi iz paketa springframework.
Ovako izgleda klasa:

```
@Service
public class UserDetailsServiceImpl implements
UserDetailsService {}
```

Poruka: Integracijom Spring Security-a sa JWT-om, moguće je kreirati robustan i siguran mehanizam autentifikacije za svoju aplikaciju. JWT-ovi obezbeđuju autentifikaciju bez stanja, omogućavajući vam da skalirate svoju aplikaciju i lako autentifikujete zahteve različitih klijenata bez potrebe za upravljanjem sesijom.

2.6. Aspekt - orijentisano programiranje(AOP)

Pisanje log poruka u Spring Boot aplikaciji kroz Aspect Oriented Programming (AOP) omogućava centralizovano upravljanje evidentiranjem određenih operacija ili metoda u aplikaciji, bez potrebe za ručno dodavanje logovanja svakoj od ovih metoda.

Prođimo kroz osnovne korake pisanja log poruka preko AOP-a u ovoj aplikaciji:

- Dodavanje zavisnosti: U konfiguracionom fajlu projekta uključujem neophodne zavisnosti za AOP i Spring Boot. Za AOP, biće potrebna zavisnost „spring-boot-starter-aop“. Primer iz konfiguracionog fajla aplikacije:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
</dependency>
```

- AOP konfiguracija: Konfigurišem aplikaciju da prepozna aspekt i primeni ga na izabrane tačke preseka. Ovo se može postići korišćenjem anotacije kao što je „@Aspect“ na aspektu i korišćenjem konfiguracionih napomena kao što je „@EnableAspectJAutoProxy“ na klasi konfiguracije.

- Definisanje aspekta: Kreiram aspekt koji sadrži logiku za pisanje log poruka. Aspekt je poseban Java objekat koji definiše ponašanje koje treba primeniti na određene metode ili operacije u aplikaciji. Primer klase:

```
@Aspect
@Component
public class Logger {}
```

- Definisanje tačaka: Unutar svake klase aspekta definišem tačke koje specificiraju tačke spajanja ili metode na koje treba primeniti aspekt. Izraz pointcut definiše kriterijume podudaranja za tačke spajanja.

- Primena akcija: Unutar klase aspekta primenjujem metode koje sadrže logiku koja treba da se izvrši pre, posle ili oko navedenih tačaka spajanja. Mogu se koristitei napomene kao što su „@Before“, „@After“ ili „@Around“ kako bih označio metode akcija. Primer metode u klasu Aspekta:

```
@Before("@annotation(Logged)")
public void logStartedExecution(JoinPoint jp){
    System.out.println("Pre izvršavanja metode. [LOGGED]. ");
    System.out.println(jp.getSignature());
    //Ispis argumenata u konzoli sa vrednostima
    for(Object o : jp.getArgs()){
        System.out.println(o);
    }}
```

Zaključak: Kada se aplikacija pokrene, aspekt će automatski primeniti snimanje log poruka na izabrane tačke preseka, prema definisanim akcijama. Na ovaj način možemo da nadgledamo izvršavanje određenih metoda ili evidentiranje događaja u aplikaciji.
Korišćenjem AOP-a u Spring Boot aplikaciji, možemo da odvojimo sveobuhvatne probleme, kao što su evidentiranje, bezbednost ili praćenje performansi, zavisi od poslovne logike, što rezultira čistijim kodom koji se može održavati.

2.7. Baza podataka

Server komunicira sa MySQL i Mongo(NoSQL) bazom podataka. Relaciona baza podataka sadrži objekte koji se automatski kreiraju prilikom pokretanja aplikacije uz pomoć Spring Data JPA mehanizma.

- Predstaviću Vam nekoliko koraka uključenih u komunikaciju Spring Boot aplikacije sa ***MySQL***:

-Konfiguracija zavisnosti: Uključujem neophodne zavisnosti u konfiguraciju izgradnje Spring Boot projekta (npr. korišćenjem Maven-a ili Gradle-a). Dodajem zavisnosti kao što su „spring-boot-starter-data-jpa“ i „miskl-connector-java“ da bih omogućio Spring Data JPA i MySQL drajver baze podataka. Ovako to izgleda u konfiguracionom fajlu:

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

-Konfiguracija baze podataka: U konfiguracionoj datoteci aplikacije (application.properties) navodim neophodne detalje o vezi sa bazom podataka, uključujući URL baze podataka, korisničko ime i lozinku. Ova svojstva će koristiti Spring Boot za uspostavljanje veze sa MySQL bazom podataka. Pogledajmo kako to izgleda:

```
                #MySQL database
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5
InnoDBDialect
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost:3306/brandy_project
spring.datasource.username=root
spring.datasource.password=root
```

-Mapiranje entiteta: Definišem entitete ili modele podataka koji predstavljaju tabele baze podataka u aplikaciji. Označavam ove entitete JPA napomenama kao što su „@Entiti“, „@Table“ i „@Column“ kako bih ih mapirali u odgovarajuće tabele i kolone baze podataka. Primer kako izgleda taj entitet u klasi:

```
@Entity
public class Brandy {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String name;
...}
```

-Interfejs skladišta: Kreiram interfejs skladišta koji proširuje Spring Data JPA spremište (npr. „CrudRepositori“ , „JpaRepositori“ ili „PagingAndSortingRepository“). Ovaj interfejs će obezbediti neophodne metode za obavljanje operacija baze podataka kao što su ispitivanje, čuvanje, ažuriranje i brisanje entiteta. Primer nekog interfejsa:

```
@Repository
public interface BrandyRepository extends
PagingAndSortingRepository<Brandy, Long> {
    List<Brandy> findByPriceBetween(double min, double max);

    @Query("SELECT a FROM Brandy a WHERE a.price > :min AND a.price <
:max")
    List<Brandy> findByPrice(double min, double max);

    @Query(value = "SELECT brandy.* FROM brandy WHERE brandy.name =
:name", nativeQuery = true)
    Optional<Brandy> findBrandyName(@Param("name") String name);
}
```

Zaključak: Sa ovim koracima, aplikacija će moći da komunicira sa MySQL bazom podataka. Možemo da izvodimo operacije baze podataka koristeći date metode skladišta, a Spring Boot će upravljati osnovnom vezom baze podataka, transakcijama i mapiranjem između Java objekata i tabela baze podataka.

- Predstaviću Vam nekoliko nivoa koraka uključenih u komunikaciju Spring Boot aplikacije sa ***MongoDB*** - isključivo za potrebe AOP(Aspect-Oriented Programming):

-Konfiguracija zavisnosti: Uključujem neophodne zavisnosti u konfiguraciju izgradnje Spring Boot projekta (npr. korišćenjem Maven-a ili Gradle-a). Dodajem zavisnosti kao što su „spring-boot-starter-data-mongodb“ da bih omogućio integraciju Spring Data MongoDB. Ovako to izgleda u konfiguracionom fajlu:

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
```

-Konfiguracija baze podataka: U konfiguracionoj datoteci aplikacije (application.properties) navodim neophodne detalje o MongoDB konekciji, uključujući host baze podataka, port i akreditive za autentifikaciju (ako je primenljivo).U mom slučaju konfiguracija je prilagođena Docker kontejneru koji zahteva uri specifikaciju(*mongodb://host:password@localhost:port*) kao potpunu akreditaciju za konekciju Pogledajmo kako to izgleda:

```
                #For Docker container - MongoDB
spring.data.mongodb.uri=mongodb://root:root@localhost:27017
spring.data.mongodb.database=brandy_springboot
```

-Mapiranje entiteta: Definišem entitete ili modele podataka koji predstavljaju MongoDB kolekciju aplikaciji. Označavam ove entitete odgovarajućim napomenama kao što su „@Document“, i „@Id“ zatim ih mapiram u odgovarajuće MongoDB kolekcije i polja. Primer kako izgleda taj entitet u klasi:

```
@Document(collection = "orderLogs")
public class OrderLog {}
```

-Interfejs skladišta: Kreiram interfejs skladišta koji proširuje Spring Data spremište (npr. „MongoRepositori“). Ovaj interfejs će obezbediti neophodne metode za obavljanje operacija baze podataka kao što su ispitivanje, čuvanje, ažuriranje i brisanje entiteta. Primer nekog interfejsa:

```
@Repository
public interface OrderLogRepository extends
MongoRepository<OrderLog, String> {
}
```

Zaključak: Prateći ove korake, aplikacija će moći da komunicira sa MongoDB bazom podataka. Možemo iskoristiti karakteristike MongoDB-a, kao što su fleksibilni dizajn šeme, visoka skalabilnost i bogate mogućnosti upita, da efikasno skladištimo i preuyimamo podatke za našu aplikaciju.

3. Korišćene tehnologije

U nastavku je dat kratak opis korišćenih tehnologija tokom rada na implementaciji sistema rešenja.

3.1. IntelliJ IDEA

IntelliJ IDEA je besplatno integrisano razvojno okruženje (IDE) za Java programiranje koje je odlično za programere koji žele da maksimiziraju svoju produktivnost. Dostupan je pod licencom Apache 2.0, koja omogućava korisnicima da razvijaju besplatne i nekomercijalne proizvode, dodatke i IDE. Program je razvila kompanija JetBrains, kompanija za razvoj softvera.

Ovaj IDE ima verziju otvorenog koda pod nazivom IntelliJ IDEA Community Edition i plaćenu verziju pod nazivom IntelliJ IDEA Ultimate. Dve verzije imaju jasne razlike, ali platforma Communiti Edition je prilično slična Ultimate. Osim toga, ima sve osnovne karakteristike koje bi svakom programeru bile potrebne.

IntelliJ IDEA ima mnogo funkcija, ali njegova najpopularnija ključna karakteristika je sposobnost pametnog dovršavanja koda. To u osnovi znači da program može da predloži klase, metode, polja i ključne reči koje se očekuju u trenutnom kodu koji pokušavate da kreirate. Pored toga, IDE takođe predviđa šta vam treba i omogućava vam da automatizujete ponavljajuće razvojne zadatke. Može vam pomoći da brzo popunite polje, pristupite prozoru sa alatkama, podesite prekidače, lako pretražite listu brojnih elemenata i još mnogo toga.

Funkcija dovršavanja koda ima dve vrste lista predloga: osnovno dovršavanje i dovršavanje pametnog podudaranja tipa. Osnovni završetak koda se pojavljuje kada pritisnete Ctrl+Space, a zatim predlaže listu mogućih imena kada se primeni na deo polja, deklaraciju promenljive ili parametar. Pritiskom na Ctrl+Space dvaput će se prikazati nedostupni članovi, klase i statička polja i metode. U međuvremenu, pritiskom na komandu tri puta proširiće se lista predloga kompletiranja osnovnog koda na sve klase u projektu.

Ukratko, IntelliJ IDEA je veoma moćan IDE koji opslužuje mnogo programera i programskih jezika. Njegova platforma i funkcije su dizajnirane da vam pomognu da kodirate svoje programe brže i efikasnije. Iako se može zamrznuti ako imate malo RAM-a, to je manji problem u poređenju sa njegovim mogućnostima. Pametno dovršavanje koda definitivno prija gomili, jer vam ne daje samo nasumične predloge.

Izvor [1]

3.2. Git – alat za kontrolu verzija

Git je DevOps alat korišćen za upravljanje kodom. Besplatan i distribuiran sistem za kontrolu verzija otvorenog koda, dizajniran da podrži rad na malim i velikim projektima sa velikom brzinom i efikasnošću. Jednostavan je za učenje i nadmašuje druge SCM (Software Configuration Management) alate, zbog odlika poput lokalnog grananja i više radnih tokova.

Git Vas podstiče da imate više lokalnih grana, koje mogu biti kompletno međusobno nezavisne, budući da je njihovo kreiranje, spajanje i brisanje jednostavno i brzo. Skoro sve operacije se lokalno izvršavaju što je jedna od odlika koja se tiče brzine. Brzina i performansa su primarni dizajnerski ciljevi od samog početka razvoja Git-a. Budući da je Git distribuiran SCM sistem, pruža Vam mogućnost kloniranja repozitorijuma. Dakle svaki korisnik ima potpun backup glavnog server-a.

Izor[2][3]

3.3. Docker – kontejner za pokretanje aplikacije

Docker je otvorena platforma za razvoj, isporuku i pokretanje aplikacija. Docker vam omogućava da odvojite svoje aplikacije od svoje infrastrukture tako da možete brzo da isporučite softver. Pomoću Docker-a možete upravljati infrastrukturom na isti način na koji upravljate svojim aplikacijama. Koristeći prednosti Docker-ovih metodologija za brzo isporuku, testiranje i implementaciju koda, možete značajno smanjiti kašnjenje između pisanja koda i njegovog pokretanja u proizvodnji.

Docker pojednostavljuje životni ciklus razvoja omogućavajući programerima da rade u standardizovanim okruženjima koristeći lokalne kontejnere koji pružaju vaše aplikacije i usluge. Kontejneri su odlični za kontinuiranu integraciju i kontinuiranu isporuku (CI/CD) radne tokove.

Docker koristi arhitekturu klijent-server. Docker klijent razgovara sa Docker demonom, koji obavlja teške poslove izgradnje, pokretanja i distribucije vaših Docker kontejnera. Docker klijent i demon mogu da rade na istom sistemu, ili možete povezati Docker klijent sa udaljenim Docker demonom. Docker klijent i demon komuniciraju koristeći REST API, preko UNIKS soketa ili mrežnog interfejsa. Još jedan Docker klijent je Docker Compose, koji vam omogućava da radite sa aplikacijama koje se sastoje od skupa kontejnera.

Kada koristite Docker, vi kreirate i koristite slike, kontejnere, mreže, volumene, dodatke i druge objekte. Ovaj odeljak je kratak pregled nekih od tih objekata.

Slika je šablon samo za čitanje sa uputstvima za kreiranje Docker kontejnera. Često se slika zasniva na drugoj slici, sa nekim dodatnim prilagođavanjem.

Kontejner je instanca slike koja se može pokrenuti. Možete da kreirate, pokrenete, zaustavite, premestite ili izbrišete kontejner koristeći Docker API ili CLI. Možete da povežete kontejner sa jednom ili više mreža, priložite mu skladište ili čak kreirate novu sliku na osnovu njegovog trenutnog stanja.

Izvor[4]

3.4. Korišćene tehnologije na serverskom delu

3.4.1.Java Spring Boot

Među svim ostalim okvirima Jave, Spring je najpopularniji. Spring se sastoji od mnogih pod-okvira, uključujući Spring Boot. Spring Boot čini gornji sloj Spring okvira i sadrži sve karakteristike koje Spring obećava programeru.

Spring Boot je lakši za korišćenje od Spring i štedi programeru neko vreme ako se iskoristi odgovarajuća konfiguracija za određenu aplikaciju. Bonus Spring Boot okvira je to što je mnogo korisniji u razvoju REST API-ja.

Koliko god da je Spring Framevork sposoban i sveobuhvatan, i dalje je potrebno značajno vreme i znanje za konfigurisanje, podešavanje i primenu Spring aplikacija. Spring Boot ublažava ovaj napor sa tri važne mogućnosti.

1. Autokonfiguracija - znači da se aplikacije inicijalizuju sa unapred podešenim zavisnostima koje ne morate ručno da konfigurišete. Pošto Java Spring Boot dolazi sa ugrađenim mogućnostima autokonfiguracije, on automatski konfiguriše i osnovni Spring Framevork i pakete nezavisnih proizvođača na osnovu vaših podešavanja.
2. Spring Boot ima snažan pristup dodavanju i konfigurisanju početnih zavisnosti, na osnovu potreba vašeg projekta.
3. Spring Boot pomaže programerima da kreiraju aplikacije koje se samo pokreću. Konkretno, omogućava vam da kreirate samostalne aplikacije koje se pokreću same, bez oslanjanja na spoljni web server, ugrađivanjem web servera kao što je Tomcat ili Netti u vašu aplikaciju tokom procesa inicijalizacije.

Izvor[5][6]

3.4.2. MySQL

MySQL je sistem za upravljanje relacionim bazama podataka razvijen od strane Oracle-a baziran na strukturiranom upitnom jeziku (SQL).

Svojim dokazanim performansama, pouzdanošću i lakoćom korišćenja, MySQL je vodeći izbor baze podataka za veb bazirane aplikacije, korišćen od strane kompanija kao što su Facebook, Twitter, YouTube i druge.

Izvor[7][8]

3.4.3. MongoDB

MongoDB je program za upravljanje NoSQL bazom podataka otvorenog koda. NoSQL (ne samo SQL) se koristi kao alternativa tradicionalnim relacionim bazama podataka. NoSQL baze podataka su veoma korisne za rad sa velikim skupovima distribuiranih podataka. MongoDB je alatka koja može da upravlja informacijama orijentisanim na dokumente, čuva ili preuzima informacije.

MongoDB se koristi za skladištenje podataka velikog obima, pomažući organizacijama da skladište velike količine podataka dok i dalje rade brzo. Organizacije takođe koriste MongoDB za svoje 'ad-hoc' upite, indeksiranje, balansiranje opterećenja, agregaciju, izvršavanje JavaScript-a na strani servera i druge funkcije.

Izvor[9]

3.5. Korišćene tehnologije za testiranje na klijentskom delu

3.5.1. Insomnia(Testiranje API-ja)

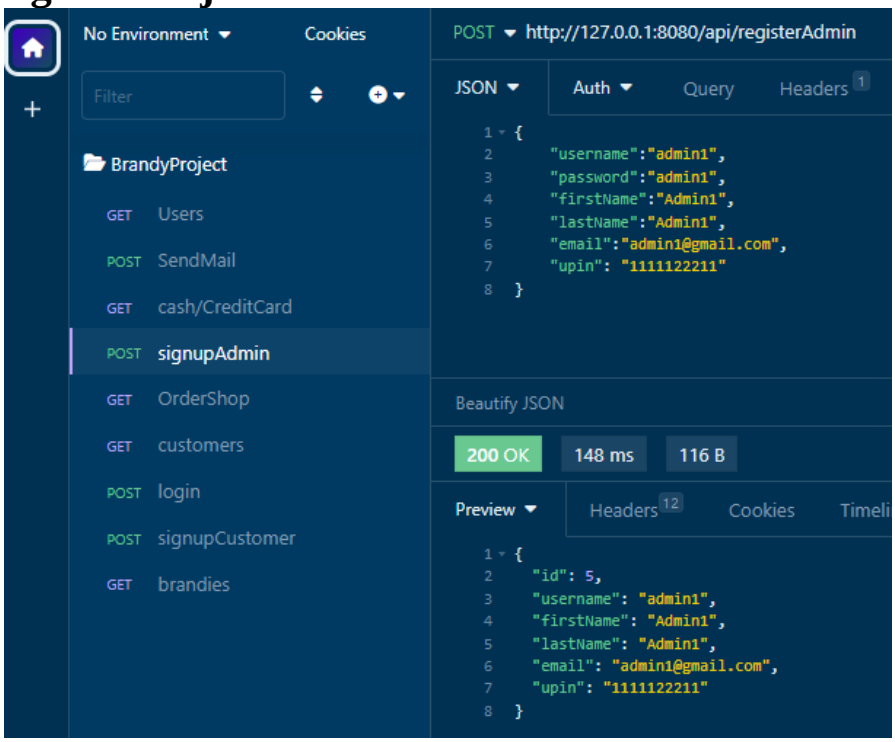
Insomnia REST Client je moćan REST API klijent koji se koristi za elegantno organizovanje, skladištenje i izvršavanje RESTful API zahteva. Insomnia je jedan od brzih REST klijenata dostupnih za Windows, Mac i Linux. Insomnia REST klijent je besplatan višeplatformski okvir za radnu površinu za testiranje RESTful aplikacija. Uključuje sofisticirane karakteristike kao što su kreiranje koda, bezbednosni pomoćnici, varijable okruženja i korisnički interfejs prilagođen korisniku. Možete da iskoristite prednosti Insomnia da testirate GraphQL API-je i RESTful API-je zasnovane na HTTP-u. To je odlična alternativa za Postman za slanje GraphQL i REST zahteva sa podrškom za promenljive okruženja, upravljanje kolačićima, autentifikaciju i generisanje koda.

Izvor[10]

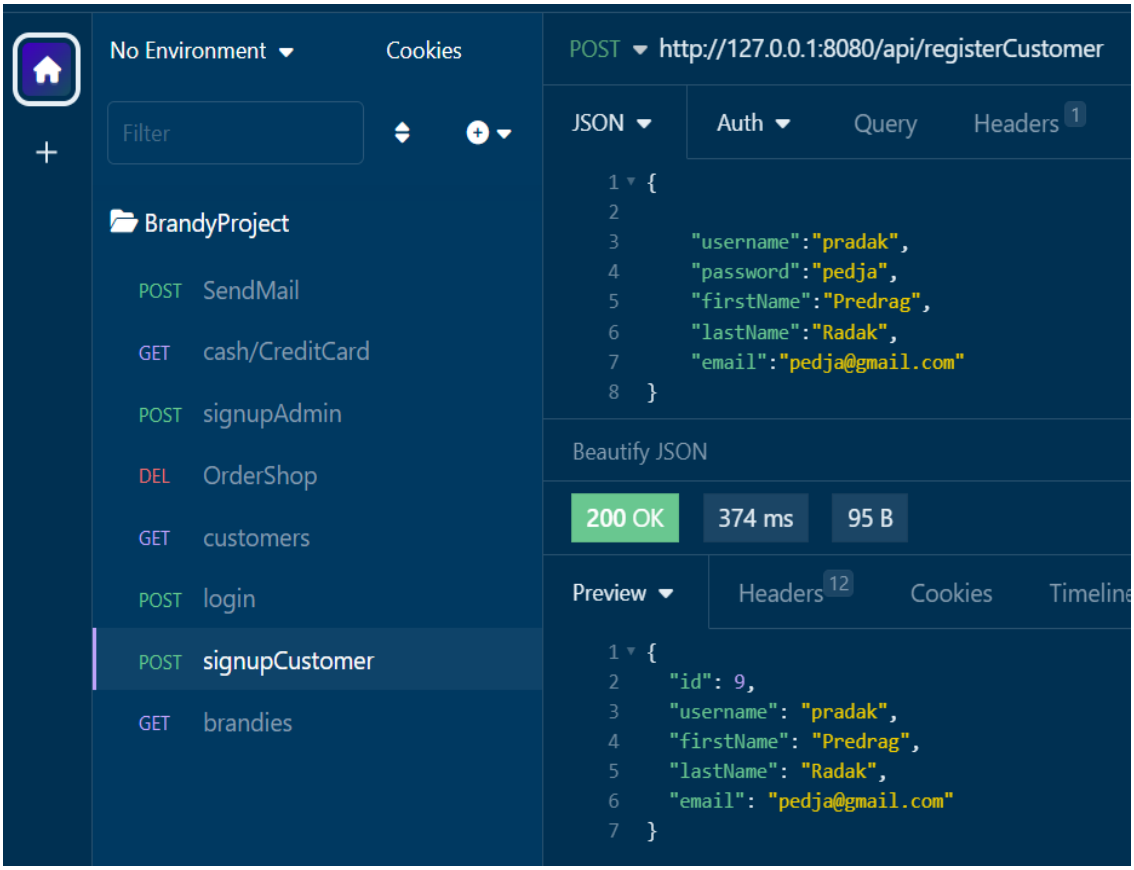
4. Primeri testiranja i rezultati aplikacije

4.1. Prijavljivanje na sistem

➢ Registrovanje korisnika

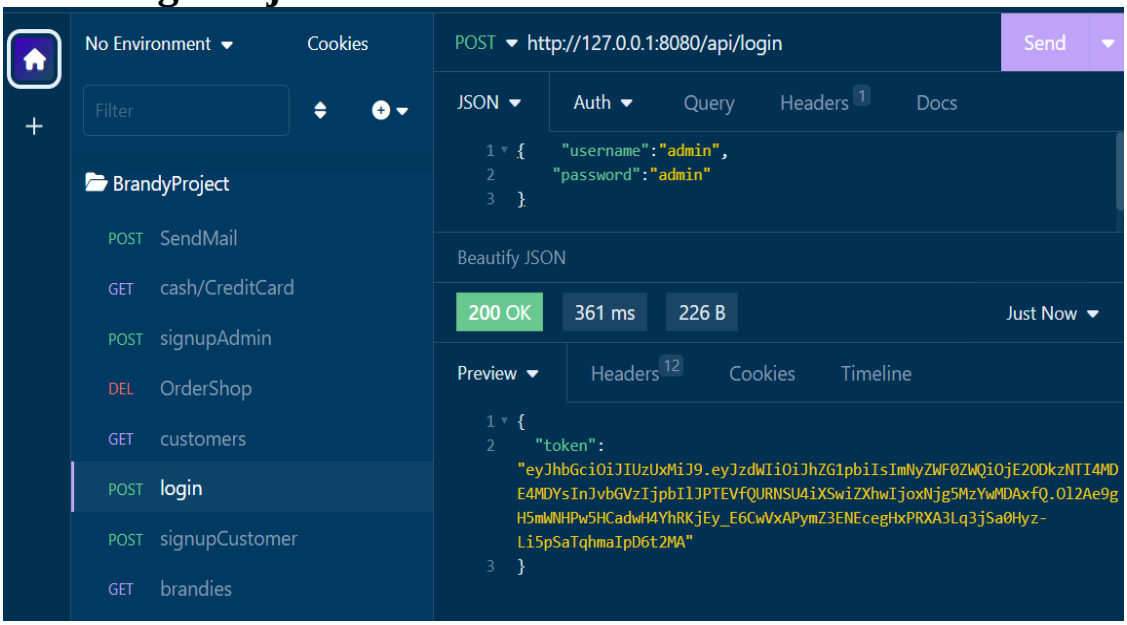


Slika 15. Registrovanje administratora - predstavlja endpoint koji se koristi za registraciju novog **Administratora** u sistem. Podaci se šalju HTTP POST metodom. Ovi podaci se koriste za kreiranje novog administratorskog naloga u sistemu. Nakon slanja zahteva, backend aplikacija treba da obradi podatke i kreira novog administratora sa navedenim parametrima. Naravno, primenjena je mera bezbednosti poput validacije podataka i enkripcije lozinke.



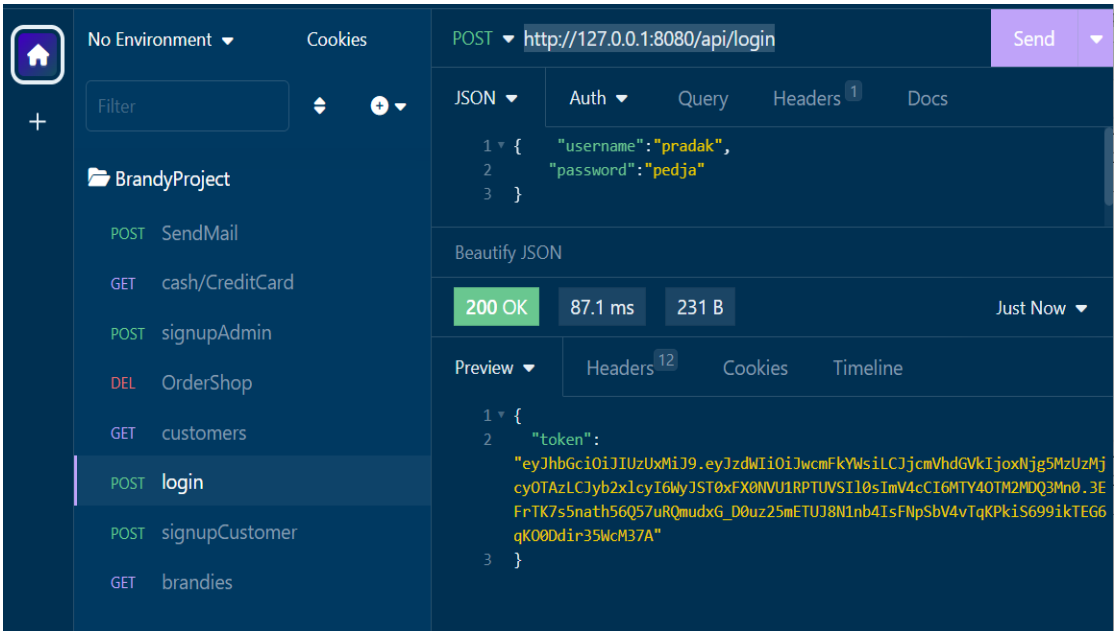
Slika 16. Registrovanje kupca - predstavlja endpoint koji se koristi za registraciju novog **Customera**(Kupca) u sistem. Podaci se šalju HTTP POST metodom. Ovi podaci se koriste za kreiranje novog naloga kupca u sistemu. Nakon slanja zahteva, backend aplikacija treba da obradi podatke i kreira novog kupca sa navedenim parametrima. Naravno, primenjena je mera bezbednosti poput validacije podataka i enkripcije lozinke.

➢ Logovanje korisnika



Slika 17. Logovanje administratora - predstavlja endpoint koji se koristi za autentifikaciju **Administratora** i generisanje tokena za pristup sistemu. Ovi podaci se koriste za proveru identiteta admina. Nakon slanja zahteva, backend aplikacija treba da proveri podatke i ako su ispravni, generiše token koji će se koristiti za dalji pristup resursima u sistemu.

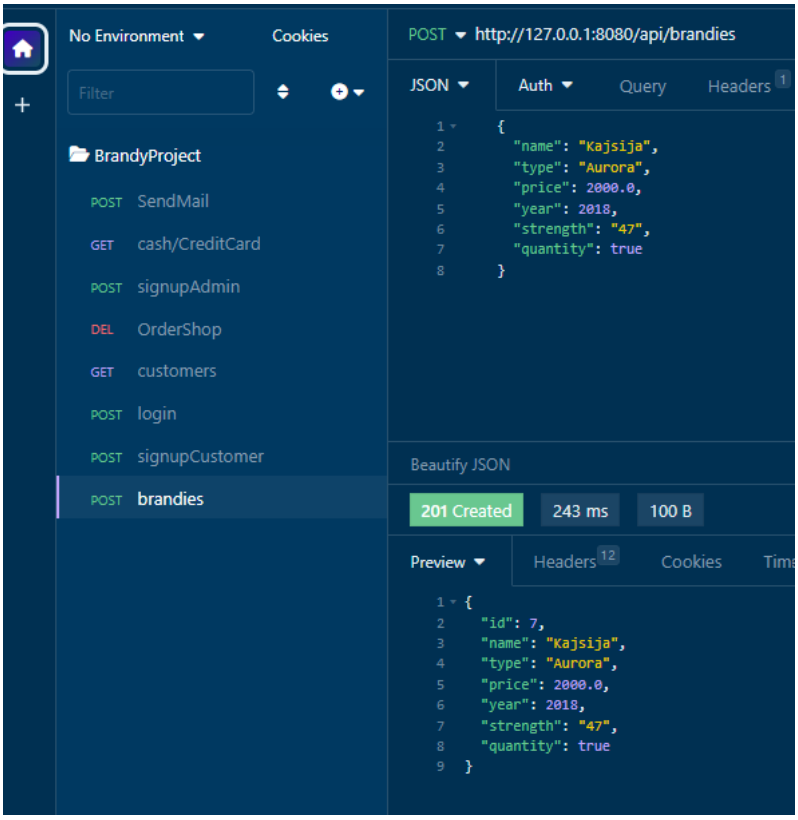
Odgovor na zahtev treba da sadrži generisani token koji će admin koristiti za autorizaciju prilikom pristupa drugim delovima aplikacije. Token se obično vraća kao deo HTTP odgovora.



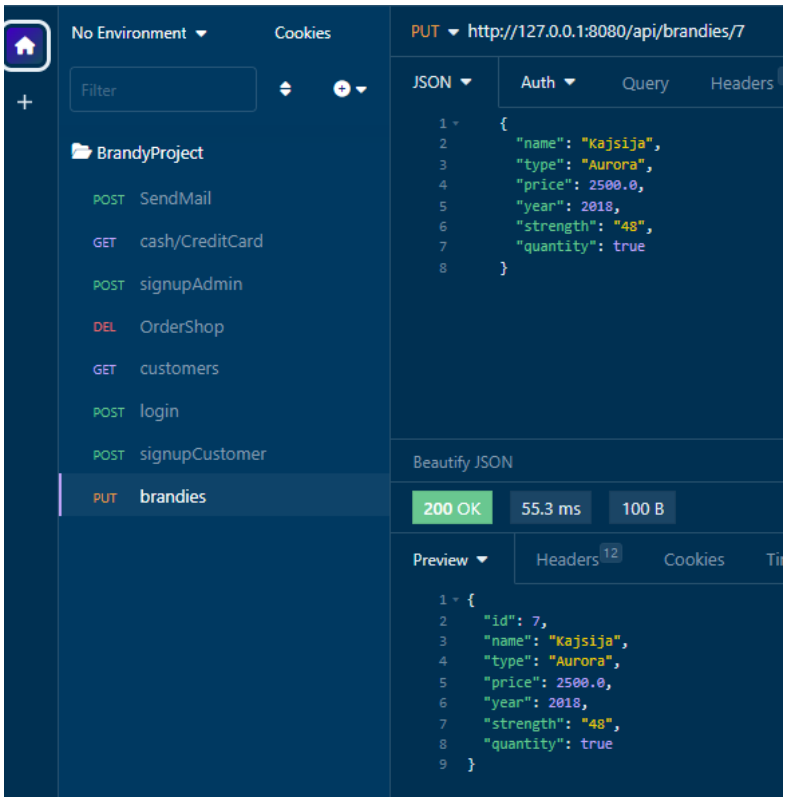
Slika 18. Logovanje kupca - predstavlja endpoint koji se koristi za autentifikaciju **Customera** i generisanje tokena za pristup sistemu. Ovi podaci se koriste za proveru identiteta kupca. Nakon slanja zahteva, backend aplikacija treba da proveri podatke i ako su ispravni, generiše token koji će se koristiti za dalji pristup resursima u sistemu.

Odgovor na zahtev treba da sadrži generisani token koji će kupac koristiti za autorizaciju prilikom pristupa drugim delovima aplikacije. Token se obično vraća kao deo HTTP odgovora.

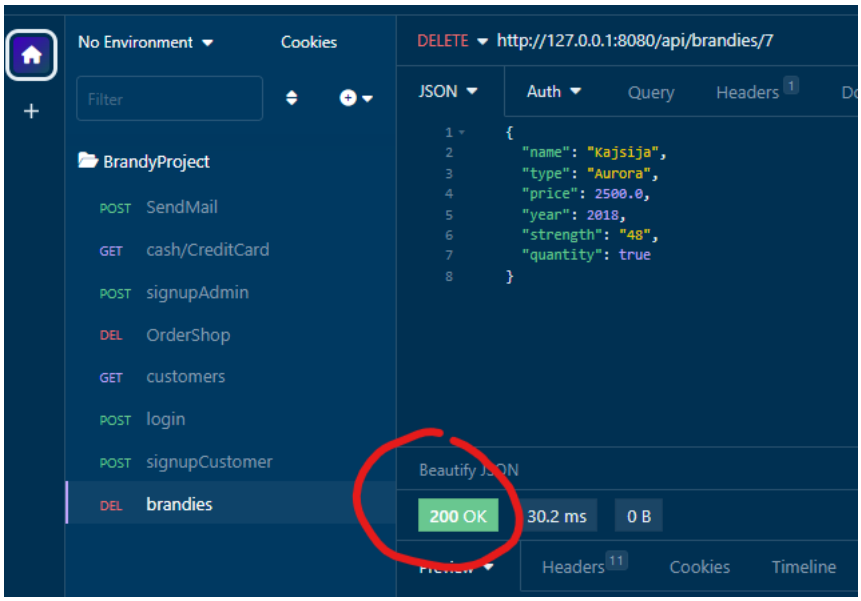
4.2. Pristup administratora



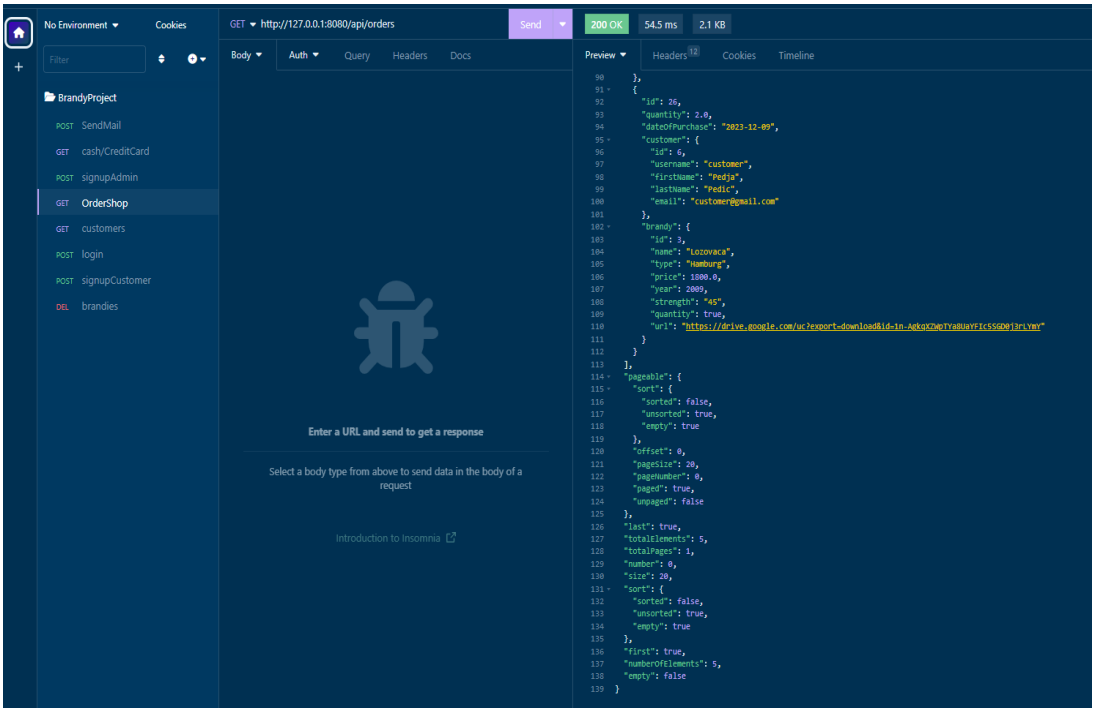
Slika 19. Dodavanje novog proizvoda (POST method).



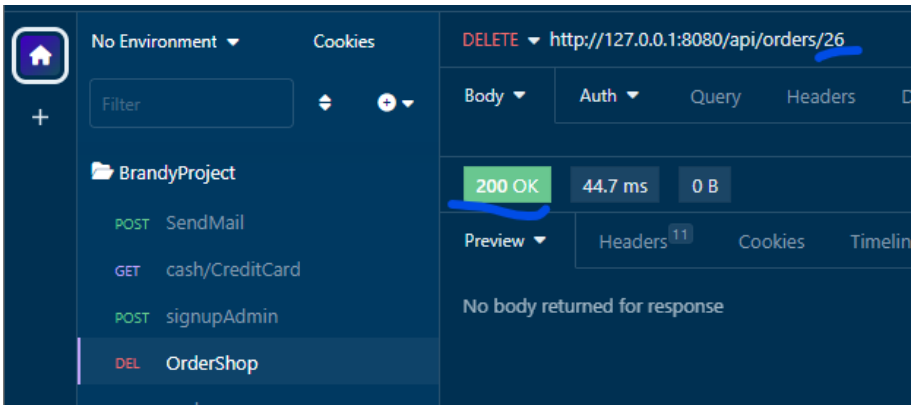
Slika 20. Izmena postojećeg proizvoda (PUT method).



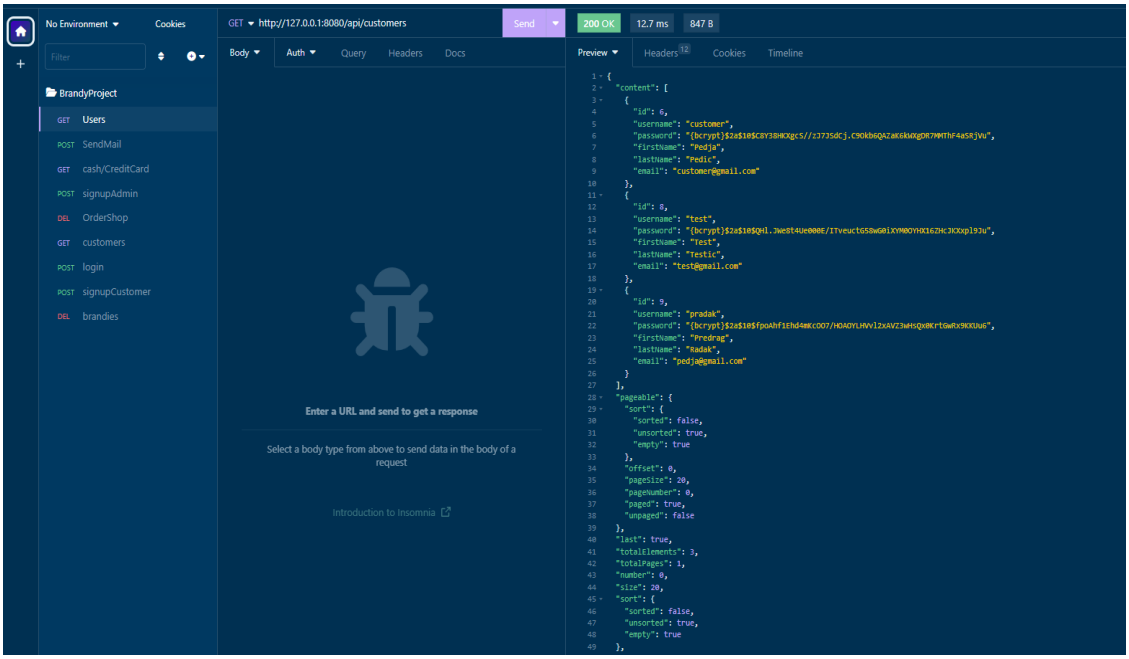
Slika 21. Brisanje proizvoda (DELETE method, po id-ju).



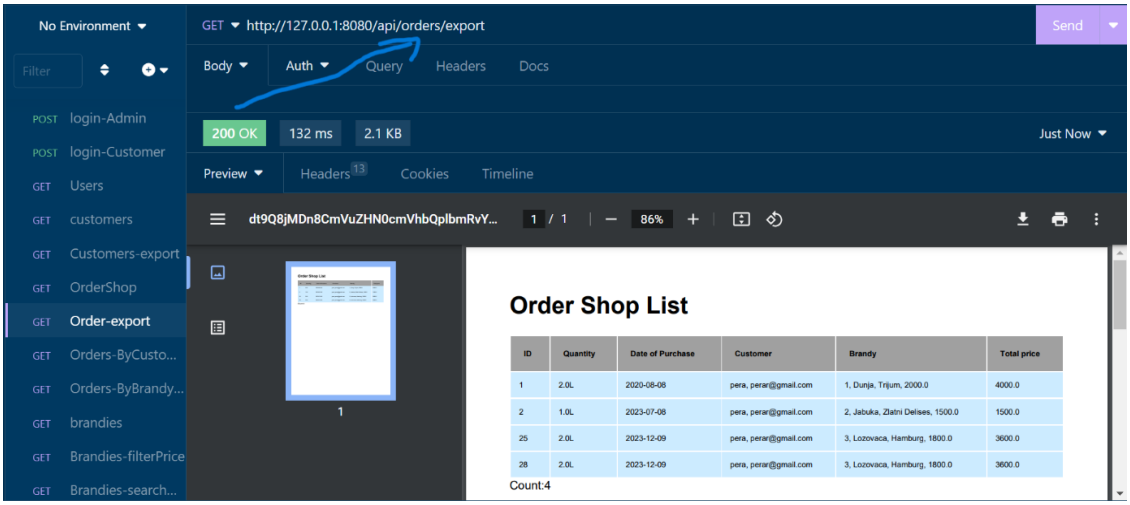
Slika 22. Pregled svih porudžbina (GET method).



Slika 23. Brisanje porudžbine (DELETE method, po id-ju).



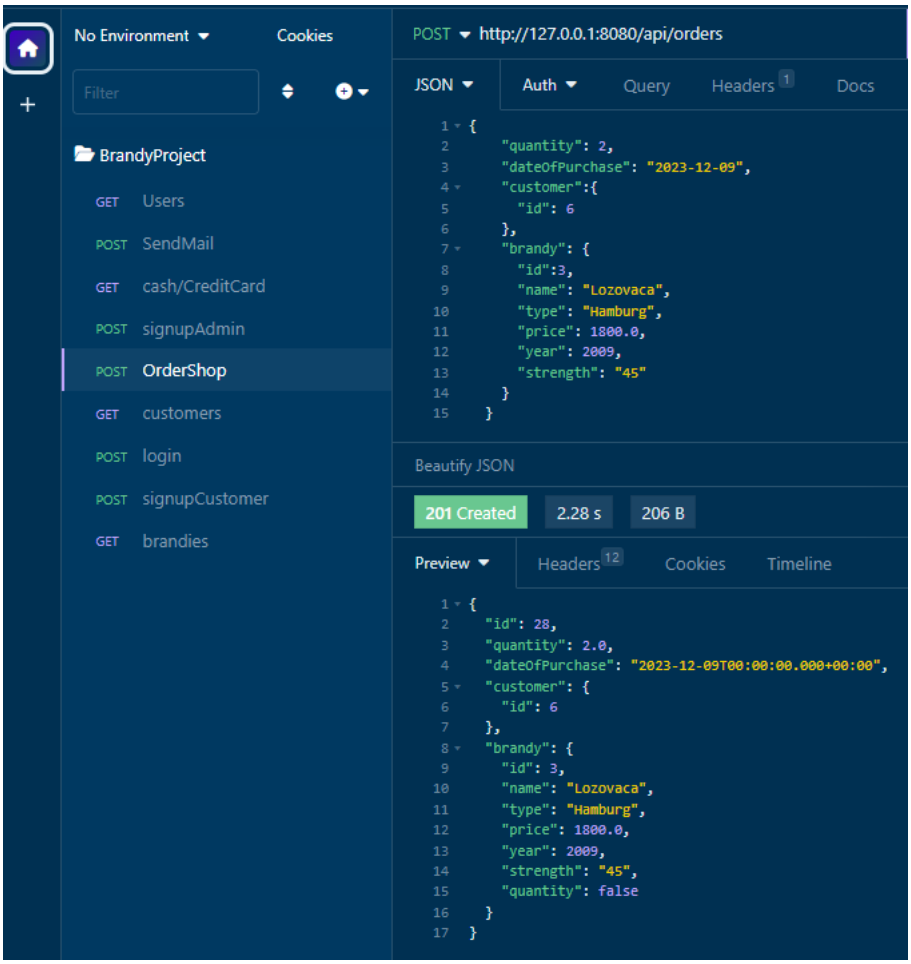
Slika 24. Pregled svih Kupaca.



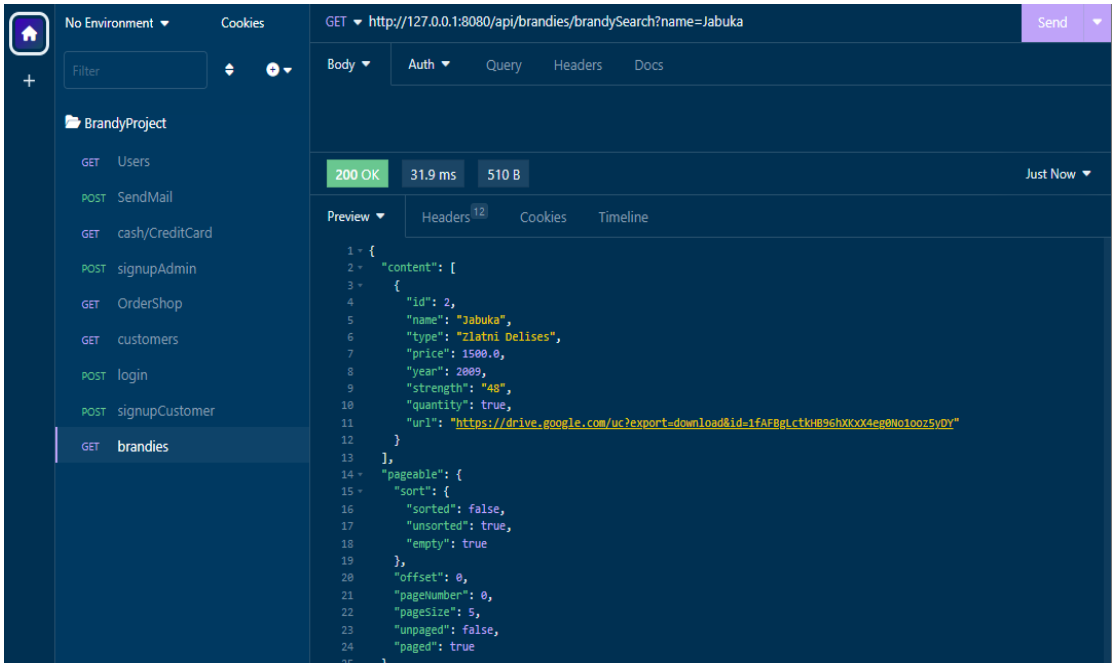
Slika 25. Eksportovanje izveštaja aktivne porudžbine.

Rezime: Prava administratora u aplikaciji su posebne privilegije i ovlašćenja koja su dodeljena korisnicima sa administratorskim ulogama kako bi im omogućili pristup i izvršavanje određenih funkcionalnosti koje su rezervisane samo za administratore. Uloga za administratora je označena je „ROLE_ADMIN“.

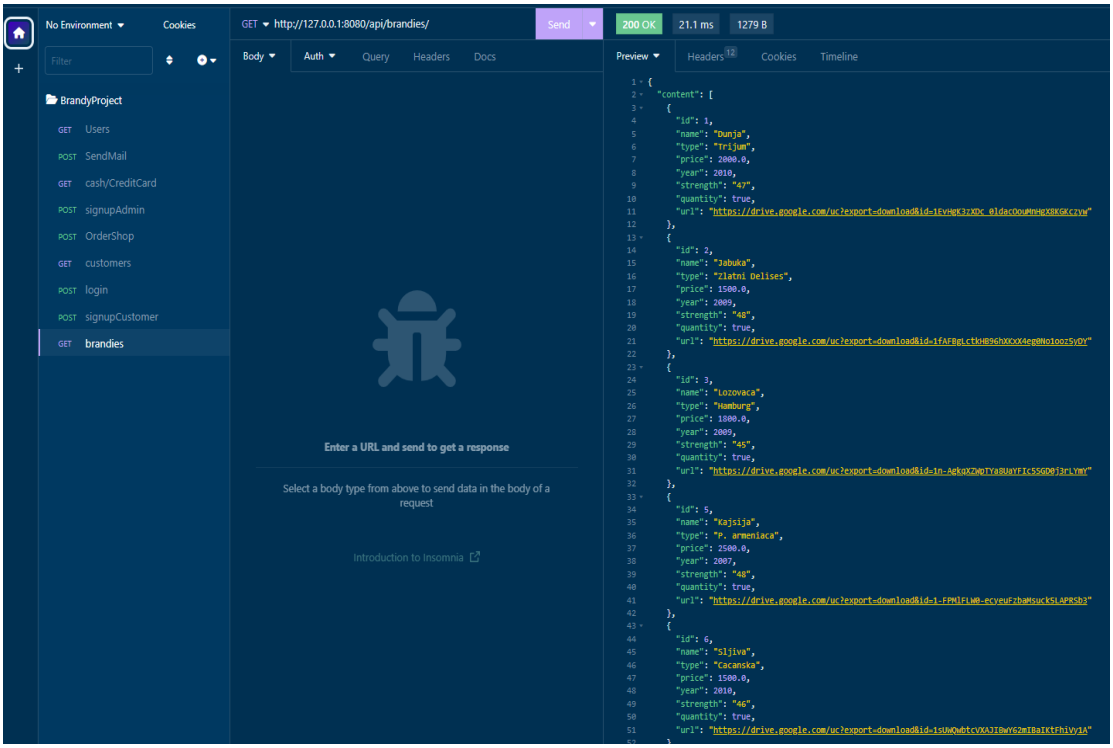
4.3. Pristup kupaca



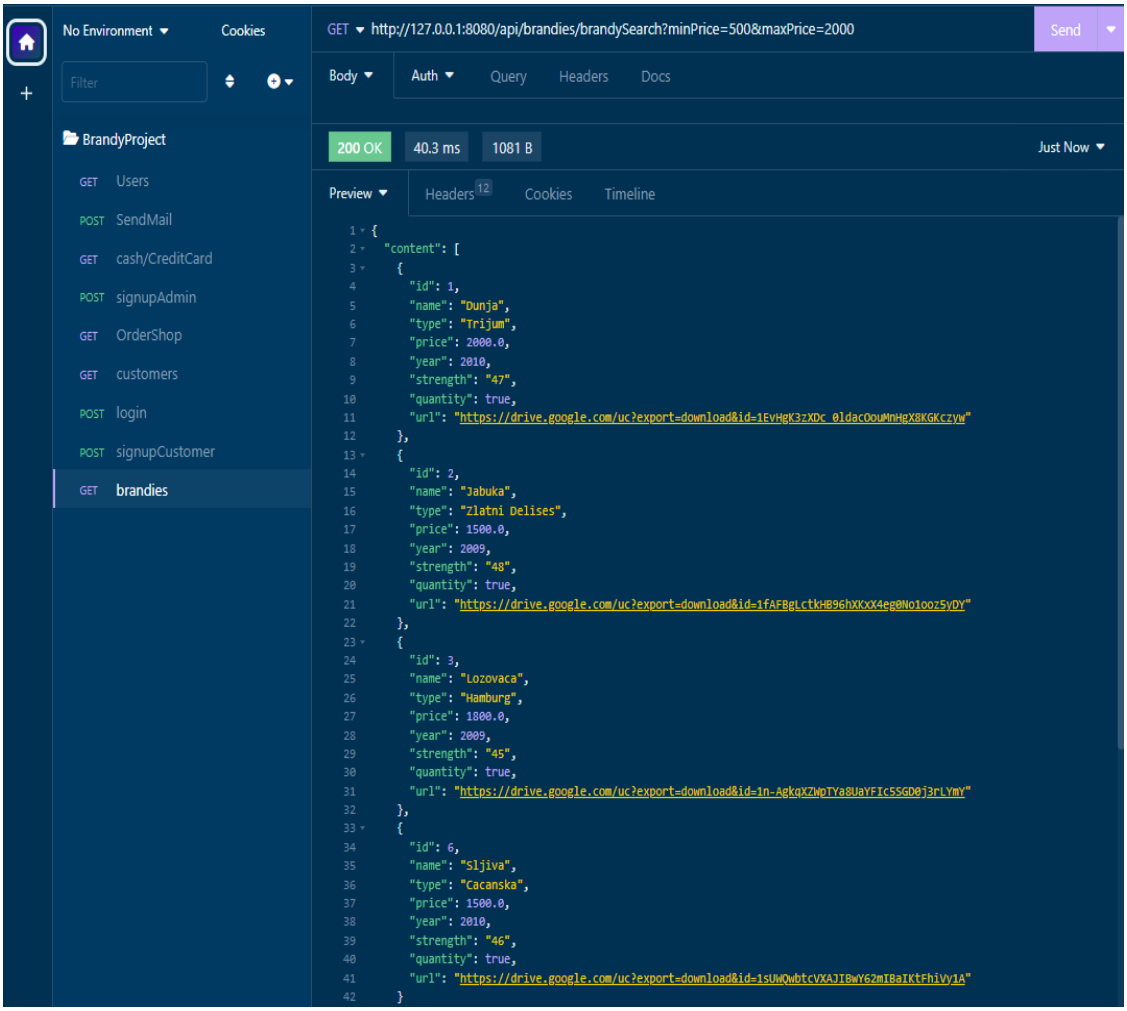
Slika 26. Poručivanje proizvoda.



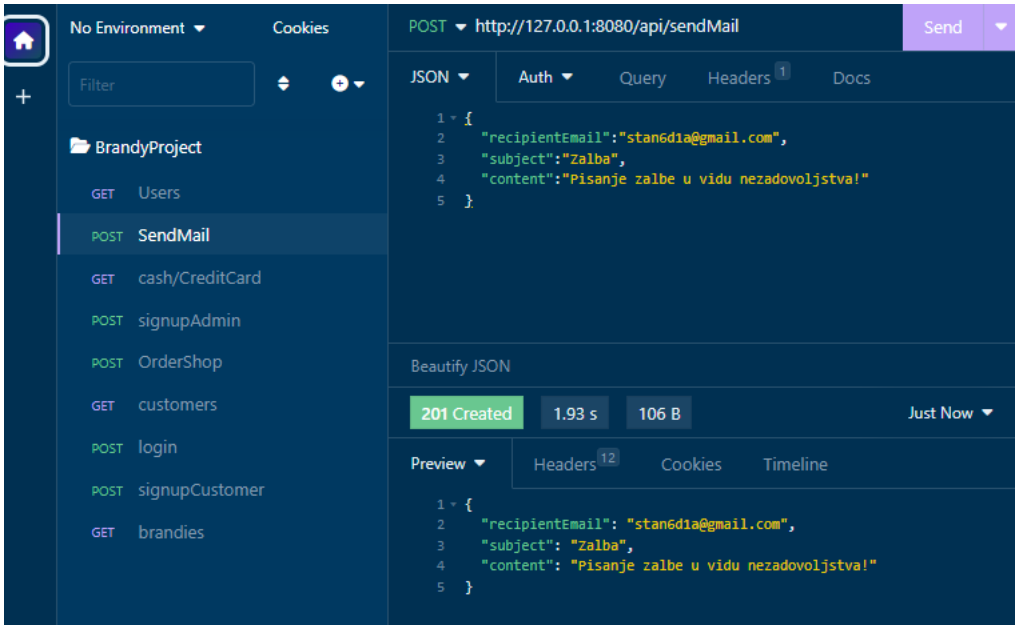
Slika 27. Pretraga proizvoda po nazivu.



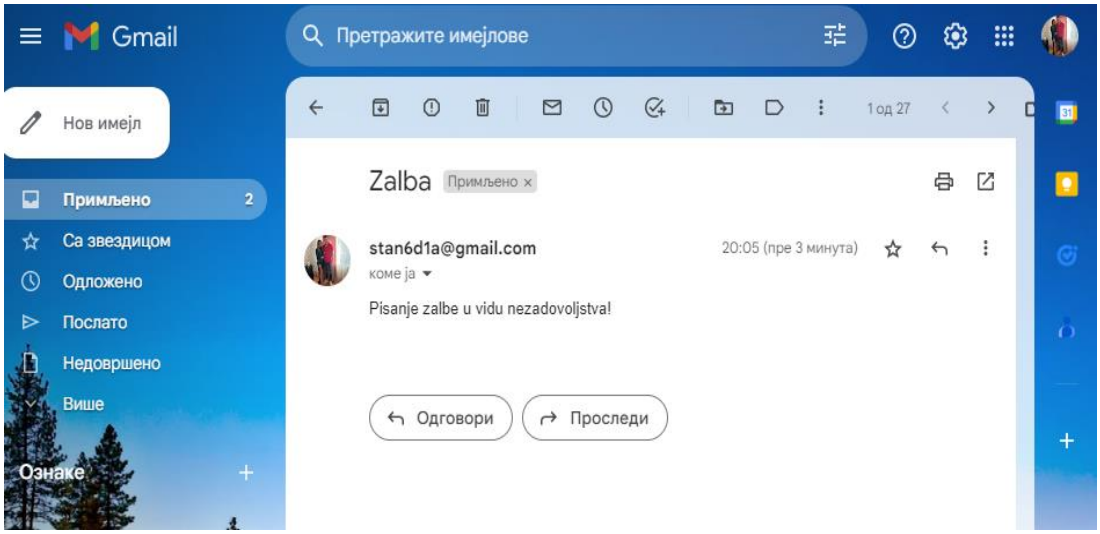
Slika 28. Pregled svih dostupnih proizvoda.



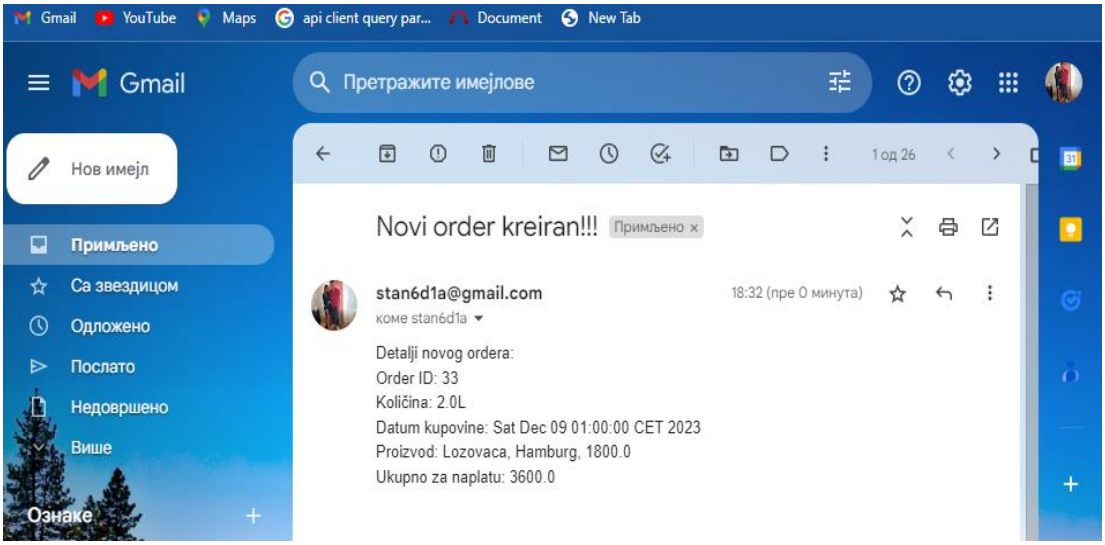
Slika 29. Filtriranje po ceni proizvoda.



Slika 30. Slanje Email pošte kao pisane recenzije za proizvod.



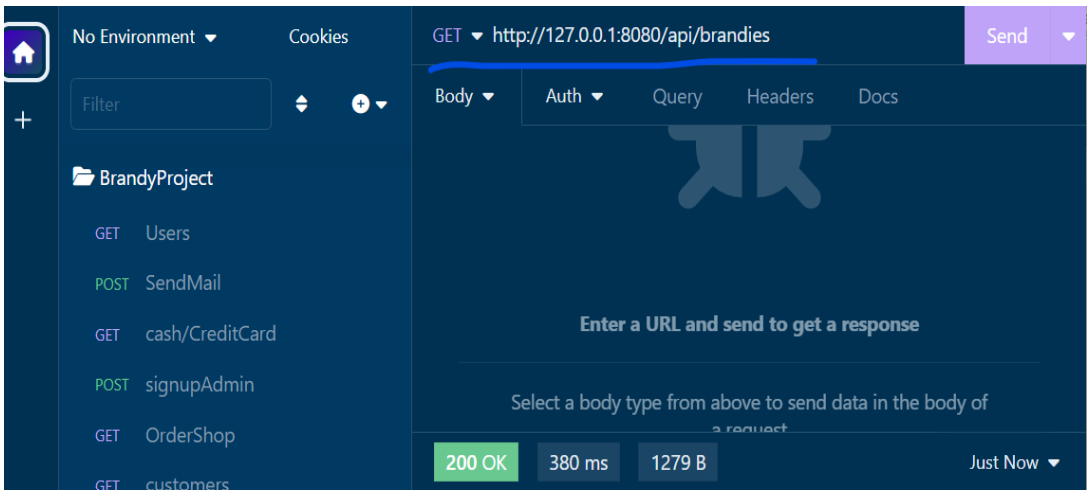
Slika 31. Rezultat prijema Email pošte.



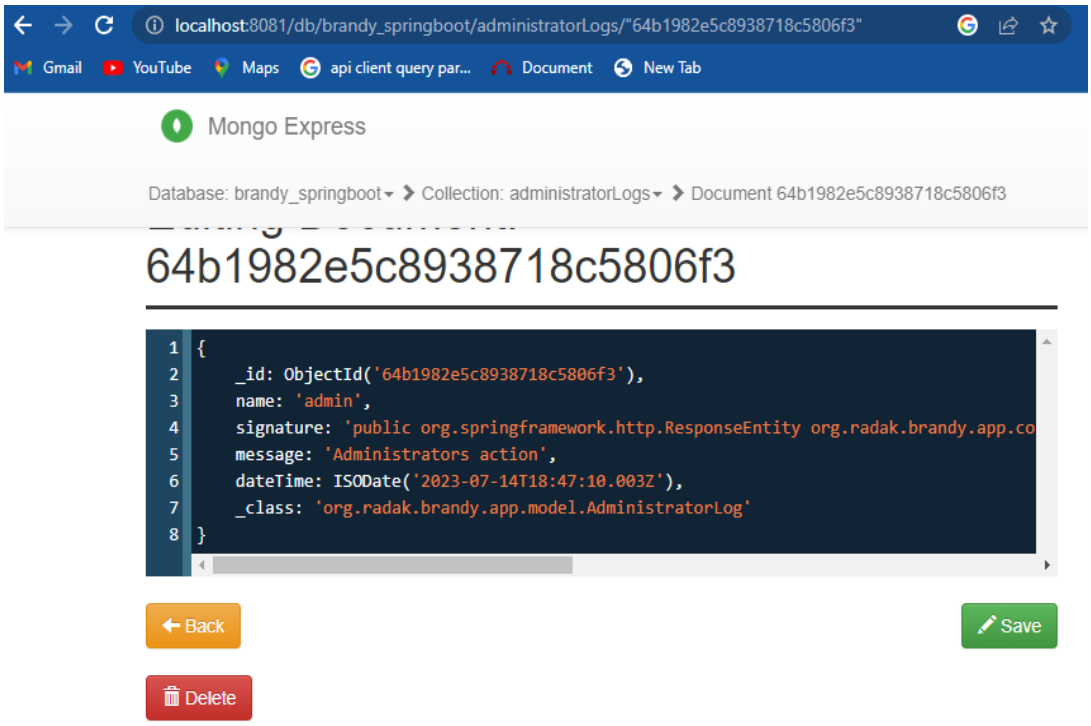
Slika 32. Rezultat prijema Email pošte za novu kreiranu porudžbinu.

Rezime: Prava kupaca u aplikaciji su posebne privilegije i ovlašćenja koja su dodeljena korisnicima sa ulogama kupaca kako bih im omogućio pristup i izvršavanje određenih funkcionalnosti koje su rezervisane samo za administratore. Uloga za kupca je označena je „ROLE_CUSTOMER“.

4.4. Skladištenje i zapis log poruka (AOP)



Slika 33. AOP poziv - Nakon dobavljanja svih proizvoda trigeruje se kreirani aspekt i akcija koja se upisuje u Mongo bazu kao log poruke.



Slika 34. Rezultat log poruka u MongoDB.

Zaključak: Ovim pristupom, log poruke će biti zapisane u MongoDB bazu podataka, što omogućava kasnije pretraživanje, analizu ili praćenje u toku izvršavanja aplikacije.

5. Zaključak

Ukupno gledano, ova aplikacija pruža praktičan način kupovine pića korisnicima, istovremeno smanjujući negativan uticaj na okolinu kroz smanjenje fizičke mobilnosti, podršku lokalnim proizvođačima poput moje porodice i digitalizaciju poslovanja. Moram da priznam, nije bilo jednostavno pokriti celu logiku sistema. Nakon silnih dilema odlučio sam se za pravi potez u pravom trenutku. Primenio sam svo iskustvo svoje porodice u ovaj rad na koji su oni veoma ponosni. Naime, u celokupnom procesu razvoja želeo sam da sistem bude jednostavan za interakciju kako bi se kupci što lakše snašli. Takav sistem na backend-u je tražio puno izazova i upoznavanja sa novim strategijama i veštinama. Dosta sam provodio vremena tokom razvoja prijavljivanja na sistem i strategije nasleđivanja. Isključivo iz razloga bezbednosti korisnika i zaštite podataka, smatram da je to na prvom mestu. Uz par dana istraživanja i testiranja sve je bilo u redu i vrlo sam zadovoljan kako to sada izgleda. Ne bih da izostavim i naravno deo koji se odnosi na CRUD operacije, tu nije bilo prevelikih poteškoća zahvaljujući kursu koji sam slušao tokom studija na predmetu Internet softverske arhitekture. Ono što jeste zahtevalo novo upoznavanje bilo je kreiranje izveštaja u vidu PDF formata i slanje maila kao princip recenzije za korisnike. Na tom slučaju proveo sam nedelju dana gde sam prikupljao informacije i primere sa raznih internet resursa i dokumentacija. Ponosno mogu reći da sam to uspešno savladao i da sam vrlo srećan jer sam se sam izborio sa tim problemom.

Šta sam uspeo da postignem od onoga što mi je bila želja? Uspeo sam da napravim web aplikaciju koju svi naši korisnici mogu da koriste.Da li sam pokrio sve slučaje korišćenja u aplikaciji? Nadam se da su sve funkcionalnosti dovoljne za jednostavno poručivanje proizvoda.Koliko je zaista dobro, to ću prepustiti Vama da ocenite.

Šta bi se moglo popraviti ili doraditi? Svakako smatram da je sistem plaćanja veoma veliki izazov za mene u ovom trenutku i da se na tome može poraditi u bliskoj budućnosti. Implementacija i model je tu ali funkcionalnosti , „knjiženje“, dnevnik blagajne, itd. je zapravo problem. To je jedan zaseban deo na kome se naravno može dodatno poraditi kako bi ova aplikacija otišla u produkciju. Takođe moram spomenuti i magacinsko poslovnaje koje jednim delom i implementirano. Moguće je dodavati zalihe u skladište. Ono što je prirotitet jeste uraditi ažuriranje stanja magacina nakon svake porudžbine.

Kada se sve sabere, moram reći da sam zadovoljan urađenim. Jeste, uvek može bolje i mora ali smatram da sam pokrio sva gradiva koja sam savladao tokom studija, što i jeste cilj ovog rada. Dobio sam ono što sam želeo, serverski deo aplikacije koji je funkcionalan po svim zahtevima koji su navedeni. Uspešno je svaki API zahtev testiran, podaci se zapisuju u bazu i iz tog razloga mogu biti samo srećan.

6. Biografija

Predrag Radak rođen je 12.01.2000. godine, u Vršcu. Završio je osnovnu skolu "Đura Jakšić" 2013. godine u Pavlišu. Iste godine upisuje srednju školu ŠC"Nikola Tesla", smer finansijski administrator, koji završava 2017. godine u Vršcu. 2019. godine upisuje Tehnički fakultet na univerzitetu Singidunum u Novom Sadu. Već u ranom uzrastu počinje da se bavi sportom i sa 5 godina započinje svoju karijeru u fudbalskom klubu "OFK Vršac", da bi kasnije postao poluprofesionalni fudbaler i reprezentativac u futsalu. U slobodno vreme voli da putuje, trenira i uživa u svom domaćinstvu.

Literatura

[1] „IntelliJ IDEA,“ [Online]. Available: https://intellij-idea.en.softonic.com/

[2]"About Git," [Online]. Available: https://git-scm.com/about

[3]"Simplilearn - What is git," [Online]. Available: https://www.simplilearn.com/tutorials/git-tutorial/what-is-git?source=sl_frs_nav_playlist_video_clicked

[4] "Docker Overview," [Online]. Available: https://docs.docker.com/get-started/overview/

[5] "What is the Java Spring Boot," [Online]. Available: https://www.educative.io/answers/what-is-the-java-spring-boot-framework-used-for

[6] "What Spring Boot adds to Spring Framework," [Online]. Available: https://www.ibm.com/topics/java-spring-boot

[7] "What is MySQL," [Online]. Available: https://www.talend.com/resources/what-is-mysql/

[8] "Talend - MySQL," [Online]. Available: https://www.talend.com/resources/what-is-mysql/

[9] "What is MongoDB," [Online]. Available: https://www.techtarget.com/searchdatamanagement/definition/MongoDB

[10] "Introduction to Insomnia REST Client," [Online]. Available: https://hevodata.com/learn/insomnia-rest-client/