Internetska trgovina informatičkom opremom

Banković, Nikola

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1500

INTERNETSKA TRGOVINA INFORMATIČKOM OPREMOM

Nikola Banković

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1500

INTERNETSKA TRGOVINA INFORMATIČKOM OPREMOM

Nikola Banković

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

ZAVRŠNI ZADATAK br. 1500

Pristupnik:	**Nikola Banković (0035214385)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	prof. dr. sc. Vedran Mornar

Zadatak: **Internetska trgovina informatičkom opremom**

Opis zadatka:

Izraditi internetsku trgovinu koja će omogućiti korisniku kupnju nove ili rabljene informatičke opreme, poput gotovih računala, komponenti za izgradnju vlastitog računala, perifernih uređaja te raznih softverskih licenci. Aplikacija treba imati mogućnost sortiranja i filtriranja po kategorijama, cijeni te ostalim parametrima za koje to bude moguće. U aplikaciju će biti ugrađene i posebne administrativne funkcionalnosti poput dodavanja novih kategorija, artikala i ostalih potrebnih atributa. Trgovina treba imati košaricu za kupnju i razrađen postupak naplate. Postojat će tri vrste korisnika, svaka sa svojom razinom autoriteta: korisnik koji kupuje artikle u trgovini, dobavljač koji prodaje svoje artikle te administrator koji će imati omogućene sve funkcionalnosti u aplikaciji uključujući i administraciju drugih korisnika. Posebnu pozornost obratiti na jednostavnost i ergonomiju korisničkog sučelja. Za razvoj aplikacije koristit će se Java, Spring Boot, Lombok, Hibernate, Maven, Angular te PostgreSQL.

Rok za predaju rada: 14. lipnja 2024.

ZAHVALA

Zahvaljujem se mentoru prof. dr. sc. Vedranu Mornaru na svim smjerincama, savjetima i pruženoj pomoći tijekom izrade završnog rada te se zahvaljujem na mentorstvu na kolegiju Projekt R.

Zahvaljujem se svojim roditeljima koji su me podržavali i pružali mi pomoć kada god je to bilo potrebno. Želim se zahvaliti svojem bratu koji mi je bio podrška i prijatelj tijekom cijelog studija te ostatku obitelji koji su mi bili motivacija i poticaj da uspijem.

Zahvaljujem se i svojim prijateljima koji su mi pomagali i podržavali me u studiju i izvan njega, a posebno se želim zahvaliti prijateljima Leoni i Nikoli na svim savjetima i velikoj podršci tijekom izrade završnog rada.

Sadržaj

Uvod

Razvoj modernih web aplikacija postao je složen i zahtjevan proces koji zahtijeva korištenje naprednih tehnologija i alata. U svijetu internetske trgovine, gdje korisnici očekuju brzu, sigurnu i responzivnu uslugu, izbor pravih tehnologija za razvoj aplikacije može biti ključan za uspjeh poslovanja. Ovaj rad se fokusira na izradu internetske trgovine specijalizirane za prodaju informatičke opreme, koristeći moderne tehnologije i radne okvire.

Java, kao jedan od najpopularnijih programskih jezika, pruža fleksibilne alate za razvoj aplikacija. Uz korištenje radnih okvira kao što su Spring Framework i Angular, moguće je izgraditi kompleksne i skalabilne sustave. Spring Framework, sa svojim komponentama poput Spring Boota, Spring Securityja i Spring Data JPA-a, omogućava brzi razvoj, sigurnost i efikasno upravljanje podacima. Angular, radni okvir razvijen od strane Googlea, omogućava izradu dinamičkih i responzivnih korisničkih sučelja koristeći programski jezik TypeScript.

U ovom radu također će se istražiti PostgreSQL, jedan od sustava za upravljanje bazama podataka, te razvojna okruženja IntelliJ IDEA i WebStorm, koja olakšavaju proces razvoja aplikacija. Kroz detaljnu analizu i praktične primjere, rad će prikazati cjelokupni proces razvoja web aplikacije, od inicijalizacije projekta do završne implementacije.

Cilj ovog rada je pružiti sveobuhvatan pregled tehnologija koje omogućuju izradu moderne, efikasne i sigurne internetske trgovine informatičkom opremom. Kroz praktične primjere, steći će se uvid u prakse i metodologije za razvoj web aplikacija, te razumjeti važnost korištenja ispravnim alata i tehnologija u procesu razvoja.

1. Tehnologije u izradi aplikacije

U ovom poglavlju istražit će se ključne tehnologije koje omogućavaju razvoj modernih softverskih aplikacija, fokusirajući se na razvoj web aplikacije u programskom jeziku Java, korištenje radnih okvira u sklopu programskog jezika Java i radnog okvira Angular te izrade baze podataka.

Java, kao jedan od najpopularnijih programskih jezika, nudi alate poput Mavena koji omogućava lakše upravljanje projektima i Lomboka koji se koristi kako bi se smanjila količina standardnih i često ponavljajućih dijelova kôda. Spring Framework je radni okvir za programski jezik Java koji u sebi sadrži brojne komponente i druge radne okvire poput Spring Boota, Spring Securityja i Spring JPA-a te tako omogućava brzi razvoj, efikasno upravljanje bazama podataka i nudi vrlo dobro rješenje za spremanje i upravljanje podacima. Angular je radni okvir kojeg je razvio Google kako bi omogućio razvoj dinamičkih i responzivnih web aplikacija uz korištenje programskog jezika TypeScript te HTML (*engl.* HyperText Markup Language) i CSS (*engl.* Cascading Style Sheets) kôdova.

Kako bi se olakšao svakodnevni rad programera koriste se razvojna okruženja koja pružaju širok spektar funkcionalnosti za razvoj i upravljanje projektima. Primjer takvog razvojnog okuženja su IntelliJ IDEA i WebStorm. IntelliJ IDEA razvojno okruženje namijenjeno je za pisanje programa u programskom jeziku Java, a razvojno okruženje WebStorm namijenjeno je za pisanje i uređivanje kôda web stranica. Takva kombinacija tehnologija pruža vrlo dobru i stabilnu osnovu za izgradnju modularnih, sigurnih i efikasnih softverskih rješenja.

1.1. Java

Java je objektno orijentirani programski jezik koji se koristi za razvoj web i mobilnih aplikacija, razvoj desktop aplikacija te igara. Razvila ga je tvrtka Sun Microsystems, a kasnije je razvoj programskog jezika preuzela tvrtka Oracle Corporation. Ranih 90-ih James Gosling je sa svojim timom započeo razvoj sa željom da se stvori jezik za digitalne uređaje poput televizora, digitalnih prijemnika i slično. Java je prošla kroz nekoliko verzija od svog prvog izdanja 1995. godine donoseći značajna poboljšanja i nove značajke sa svakom novom verzijom [1].

Prva verzija pod nazivom Java 1.0 omogućila je osnovne funkcionalnosti i princip "napiši jednom, pokreni svuda" zbog svoje mogućnosti da se izvršava na bilo kojem operacijskom sustavu koji podržava JVM (*engl*. Java Virtual Machine). JVM služi kao izvršni motor za pokretanje aplikacija pisanih u Javi i zbog korištenja Java *bytecode*-a, koji nastaje prevođenjem originalnog Java kôda, JVM ga može interpretirati na bilo kojoj platformi na kojoj je instaliran [2]. Uslijedile su verzije Java 2 (J2SE 1.2) i Java 5 (J2SE 5.0) koje su uvele značajke poput Swing biblioteke za izradu grafičkih korisničkih sučelja i generika koji omogućavaju pisanje univerzalnog kôda i bolju kontrolu tipova. U Javi 8, izdanoj 2014. godine, uvedeni su lambda izrazi i *Streams API* (*engl.* Application Programming Interface), omogućavajući funkcionalno programiranje. Najnovije verzije, kao što su Java 11 i Java 17, donose dugoročnu podršku i dodatna poboljšanja u performansama, sigurnosti i sintaksi jezika, čime se omogućava razvoj suvremenih i učinkovitih aplikacija [1]. Trenutno najnovija verzija jezika Java koja ima dugoročnu podršku je Java 21 (Java SE 21), a najnovija verzija bez plana za dugoročnu podršku je Java 22 (Java SE 22) koja je izašla u ožujku 2024. godine.

Prednosti programskog jezika Java u odnosu na ostale programske jezike su prenosivost koju omogućava JVM, sigurnost zbog rada unutar virtualnog stroja što omogućuje izolaciju pri izvršavanju kôda i nepostojanje eksplicitnih pokazivača čime se smanjuje rizik od neovlaštenog pristupa memoriji. Dodatne prednosti jezika Java su jednostavnost učenja zbog toga što je sintaksa temeljena na sintaksi jezika C++, robusnost zbog automatskog upravljanja memorijom i mehanizama za rukovanje iznimkama te visokih performansi u odnosu na ostale tradicionalno interpretirane jezike zbog toga što je *bytecode* jako sličan izvornom kôdu [3].

1.1.1. Maven

Maven (*jid*. „Sakupljač znanja“) započeo je kao pokušaj pojednostavljivanja procesa izgradnje u Jakarta Turbine projektu. Cilj je bio stvoriti standardizirani način izgradnje projekata, jasnu definiciju njihovih opsega, jednostavan način za objavljivanje informacija o projektu te način za dijeljenje JAR (*engl.* Java ARchive) datoteka između nekoliko projekata. Kao rezultat tog pokušaja stvoren je alat koji se može koristiti za izgradnju i upravljanje bilo kojim projektom temeljenom na programskom jeziku Java [4].

Maven se koristi radi izbjegavanja većinskog dijela same konfiguracije projekta pružajući razne projektne predloške koji su nazvani arhetipovi (*engl.* Archetypes). Osim predložaka ključna značajka Mavena je upravljanje ovisnostima (*engl.* Dependency management). Te ovisnosti predstavljaju vanjske knjižnice koje se koriste u projektu Maven uključujući automatsko ažuriranje, preuzimanje te provjeru kompatibilnosti ovisnosti što značajno smanjuje količinu rada potrebnu za postavljanje i održavanje projekta.

Ovisnosti se preuzimaju iz repozitorija ovisnosti, a ovisnosti dodataka se preuzimaju iz repozitorija dodataka što rezultira manjim brojem sukoba kada dodaci počnu preuzimati dodatne ovisnosti koje su im potrebne. Potrebne ovisnosti projekata moguće je učitati iz lokalnog datotečnog sustava ili javnih repozitorija ovisnosti poput Maven Centrala što omogućuje lak pristup velikom broju alata i knjižnica koje su potrebne za razvoj projekta. Postavljanje Maven projekta vrši se putem POM-a (*engl.* Project Object Model) kojeg predstavlja XML (*engl.* Extensible Markup Language) datoteka naziva pom.xml.

POM upravlja ovisnostima projekta, opisuje ga te postavlja dodatke potrebne za izradu softvera [5]. Primjer izgleda pom.xml datoteke dan je u nastavku pod nazivom: Kôd 1.1.

```
<project xmlns="..."
      <modelVersion>4.0.0</modelVersion>
      <parent>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-
parent</artifactId>
            <version>3.2.5</version>
            <relativePath/> <!-- lookup parent from
repository -->
      </parent>
      <groupId>hr.fer.zavrsni1500</groupId>
      <artifactId>it-shop</artifactId>
      <version>0.0.1-SNAPSHOT</version>
      <name>IT Shop</name>
      <description>IT Shop</description>
      <properties>
            <java.version>21</java.version>
            <mapstruct.version>1.5.5.Final</mapstruct.version
      >
      </properties>
      <dependencies>
            <dependency>
                  <groupId>org.springframework.boot</groupId>
                  <artifactId>spring-boot-starter-data-
jpa</artifactId>
            </dependency>
            <dependency>
                  <groupId>org.springframework.boot</groupId>
                  <artifactId>spring-boot-starter-
security</artifactId>
            </dependency>
```

Kôd 1.1 Primjer pom.xml datoteke

Svaka aplikacija koja se gradi koristeći Maven slijedi određeni životni ciklus. Moguće je izvršiti nekoliko ciljeva životnog ciklusa, uključujući kompilaciju kôd-a projekta, stvaranje paketa te instalaciju arhivske datoteke u lokalni Maven repozitorij. Neki od važnijih životnih ciljeva su: *validate*, *compile*, *test*, *package*, *integration-test*, *verify*, *install*, *deploy*.

U životnom ciklusu *validate* provjerava se ispravnost projekta, u ciklusu *compile*, kompilira se izvorni kôd. Životni ciklusi *test* i *integration-test* pokreću jedinične i integracijske testove, a *package* pakira kompilirani kôd u arhivsku datoteku. *Verify* provjerava je li paket važeći dok *install* instalira paket u lokalni Maven repozitorij, a *deploy* taj paket šalje na udaljeni server ili repozitorij [5].

1.1.2. Lombok

Lombok je alat koji se koristi u programskom jeziku Java kako bi pojednostavio razvojni proces na način da eliminira potrebu za ručnim pisanjem repetitivnih dijelova kôda. U java projekt moguće ga je implementirati kao ovisnost kroz Maven, a svojom implementacijom omogućava programerima korištenje brojnih anotacija za generiranje standardnih metoda poput *gettera* i *settera* koji se koriste za dohvaćanje ili postavljanje vrijednosti varijable unutar java objekta korištenjem `@Getter` i `@Setter` anotacija [6].

Omogućava zaobilaženje pisanja konstruktora koristeći se anotacijama `@RequiredArgsConstructor` koji generira konstruktor samo s potrebnim argumentima, `@NoArgsConstructor` omogućava generiranje konstruktora bez ulaznih argumenata te `@AllArgsConstructor` generira konstruktor koji u sebi sadrži sve varijable tog objekta kao argumente. Neke od bitnih anotacija koje Lombok omogućava su `@ToString`, `@EqualsAndHashCode` i `@Data`. `@ToString` anotacija generira toString metodu koja se koristi za ispis objekta u obliku stringa. Anotacija `@EqualsAndHashCode` generira implementacije metode *equals* koja služi za usporedbu i metode *hashCode* koja vraća brojčanu vrijednost generiranu *hashing* algoritmom. Anotacija `@Data` kombinira funkcionalnosti anotacija `@Getter`, `@Setter`, `@ToString` te `@EqualsAndHashCode` [7].

1.1.3. Liquibase

Liquibase je alat za upravljanje bazom podataka koji omogućava bržu i sigurniju provjeru i obavljanje promjena na shemama baze podataka od razvoja do puštanja aplikacije u produkciju. Promjene se mogu napisati jednom i primjenjivati na različitim platformama baza podataka, a moguće ih je definirati u različitim formatima poput XML-a, JSON-a (engl. JavaScript Object Notation) i slično.

Izvršavanjem promjena, Liquibase u bazi podataka stvara dvije nove tablice: „DATABASECHANGELOG“ u kojoj su pohranjeni podaci o promjenama i „DATABASECHANGELOGLOCK“ koja onemogućava da više instanci Liquibase-a istovremeno ažurira bazu podataka. Liquibase koristi ranije navedene formate i slične datoteke za navođenje promjena u bazi podataka slijednim redoslijedom. Promjene su u obliku *changeset*-ova koji u sebi sadrže tipove promjena koje se primjenjuju nad bazom podataka poput dodavanja stupaca, promjene ključeva i slično. [8]

1.2. Spring Framework

Spring Framework je radni okvir za programski jezik Java kojeg je 2003. godine razvio Rod Johnson [9]. Spring kao radni okvir omogućava lakši razvoj aplikacija temeljenih na programskom jeziku Java.

Ključan element Springa je infrastrukturna podrška na razini aplikacije koja se fokusira na tok poslovnih aplikacija što omogućuje razvojnim timovima da usmjere svoju pažnju na poslovnu logiku aplikacije umjesto na nepotrebne veze sa specifičnim implementacijskim okruženjima [10]. Spring se može smatrati okvirom koji podržava druge okvire, poput Spring Security, Spring JPA, i Spring Boot. Ovi okviri omogućuju jednostavniju integraciju sigurnosti, pristupa podacima i brzu izgradnju aplikacija.

Spring omogućava razvoj aplikacija visokih performansi korištenjem POJO-a [11]. POJO (*engl*. Plain old Java object) je definicija klase koja nije vezana za niti jedan Java okvir, pa ga zbog toga može koristiti bilo koji Java program. Glavna prednost POJO-a je ponovna iskoristivost i jednostavnost jer ne postoje posebne konvencije za imenovanje svojstava i metoda, niti druga posebna ograničenja [12].

U nastavku se nalazi primjer objekta `Person` pod nazivom Kôd 1.2 koji prikazuje osnovnu implementaciju POJO klase s privatnim varijablama, konstruktorom s parametrima, te *getter* metodama za pristup punom imenu i dobi osobe.

```
public class Person {
    private String firstName;
    private String lastName;
    private int age;

    public Person(String firstName, String lastName, int age)
{
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
    }

    public String getFullName() {
        return firstName + " " + lastName;
    }

    public int getAge() {
        return age;
    }
}
```

Kôd 1.2 Primjer POJO klase

Osim POJO-a spring koristi tehnike poput AOP-a (*engl.* aspect-oriented programming) i posebnog oblika IoC-a (*engl.* Inversion of Control), također poznato kao ubrizgavanje ovisnosti (*engl.* Dependency Injection). IoC ili inverzija kontrole je načelo dizajna u razvoju softvera koja prepušta kontrolu izvršavanja radnom okviru. Koristeći IoC Spring olakšava izradu modularnih programa, testiranje i zamjenu između različitih implementacija na način da preuzme kontrolu nad tijekom programa i pozivanjem prilagođenog kôda. U objektno orijentiranom programiranju, ubrizgavanje ovisnosti je tehnika u kojoj jedan objekt pruža ovisnosti drugom objektu [11]. Ovaj pristup pomaže u odvajanju stvaranja objekata od njihove upotrebe, što poboljšava modularnost i testabilnost kôda. Glavna ideja je da se objekti ne bi trebali sami stvarati, već bi sve potrebne ovisnosti trebale biti ubrizgane u njih. To poboljšava modularnost, olakšava testiranje i promiče labavo spajanje između komponenti [13].

JavaBean je posebna vrsta Java klasa koje se koriste u Spring Frameworku i poštuju određene konvencije. Prema konvencijama mora sadržavati konstruktor bez argumenata, mora podržavati sučelje koje omogućava serijalizaciju tog objekta (*engl.* Serializable interface) i mora imati *Gettere* i *Settere* koji služe za dohvaćanje i postavljanje vrijednosti vlastitih svojstava [14].

Primjer JavaBean klase se nalazi u nastavku pod nazivom Kôd 1.3 i prikazuje klasu koja ispunjava sve uvjete navedene konvencijom. Sadrži privatnu varijablu `value`, konstruktor bez ulaznih argumenata, *getter* i *setter* metode za dohvaćanje i postavljanje imena te implementaciju sučelja `Serializable`.

```
public class ExampleLongValueBean implements Serializable {
    private Long value;

    public ExampleBean() {
    }

    public Long getValue() {
        return this.value;
    }

    public void setValue(Long value) {
        this.value = value;
    }
}
```

Kôd 1.3 Primjer JavaBean klase

Spring Framework se također uvelike oslanja na korištenje XML konfiguracijske datoteke kako bi definirao kontekst aplikacije i povezivanje između njezinih komponenata. Datoteka beans.xml definira prethodno navedene Beanove i njihove ovisnosti, a datoteka applicationContext.xml definira cjelokupni kontekst aplikacije [15]. Primjer XML konfiguracijske datoteke koja u beans.xml definira Bean i naziv njegove varijable iz ranije navedenog primjera Kôd 1.3 prikazan je Kôd 1.4:

```
<bean id="exampleLongValueBean"
class="hr.fer.zavrsnirad.ExampleLongValueBean">
      <property name="value" value=1L/>
</bean>.
```

Kôd 1.4 Primjer Bean-a i varijable

1.2.1. Spring Boot

Spring Boot je radni okvir u sklopu programskog jezika Java i jedan je od brojnih okvira unutar Spring Frameworka koji služi za jednostavniju izradu i pokretanje Java aplikacija. Razvio ga je Pivotal tim u svrhu ubrzanja procesa postavljanja, konfiguracije i pokretanja jednostavnih i web aplikacija.

Glavni cilj Spring Boota je skratiti vrijeme razvoja i pisanja jediničnih testova u kojima se testiraju pojedinačne komponente te integracijskih testova u kojima se kombiniraju pojedinačne komponente i testiraju kao cjelina [16]. Jedna od značajki Spring Boota je mogućnost izbjegavanja pisanja teških XML konfiguracijskih datoteka, koje su prisutne u Spring Frameworku, te njihova automatska konfiguracija. Potrebno je samo pravilno koristiti konfiguraciju za određenu funkcionalnost.

Na primjer, za korištenje Hibernatea, dovoljno je dodati anotaciju `@Entity`, te ako se ne želi koristiti automatski naziv tablice istovjetan nazivu klase ili stupca istovjetnog nazivu varijable mogu se dodati anotacije `@Table` i/ili `@Column`.

Stvaranje REST API-ja je vrlo jednostavno i može se ostvariti uporabom anotacija `@RestController` i `@RequestMapping(„/nazivOdredišta“)` na razini klase upravljača u kombinaciji s anotacijama poput `@GetMapping`, `@PostMapping`, `@PutMapping`, `@PatchMapping`, `@DeleteMapping` kao što je prikazano kôdom: Kôd 1.5 [17].

```
@RestController
@RequiredArgsConstructor
@RequestMapping("/api/auth")
public class AuthController {

    private final AuthService authService;

    @GetMapping("/current")
    public UserDto getCurrentUser() {
        return authService.getCurrentUser();
    }

    @PostMapping("/login")
    public LoginResponse login(
      @RequestBody final LoginDto user) {
```

```
        return authService.login(user);
    }

    @PostMapping("/register")
    public void register(
      @RequestBody final RegisterDto user) {
        authService.register(user);
    }
}
```

Kôd 1.5 Primjer korištenja mappinga u upravljaču

Posebna verzija anotacije koja se koristi u upravljaču, anotacija `@RestController`, u sebi uključuje anotacije `@Controller` i `@ResponseBody` i na taj način pojednostavljuje njihovu implementaciju.

Primjer jednostavnijeg načina pisanja metoda upravljača korištenjem `@RestController` anotacije dan je u nastavku pod nazivom Kôd 1.6.

```
@GetMapping("/current")
public UserDto getCurrentUser() {
    return authService.getCurrentUser();
}
```

Kôd 1.6 Primjer korištenja anotacije @RestController

U usporedbi s korištenjem `@Controller` prikazanih isječkom Kôd 1.7.

```
@GetMapping("/current")
Public @ResponseBody UserDto getCurrentUser() {
    return authService.getCurrentUser();
}
```

Kôd 1.7 Primjer korištenja anotacije @Controller

Pri izradi upravljača koristi se i anotacija `@RequestMapping` za mapiranje zahtjeva metode. `@RequestMapping` sadrži razne atribute za podudaranje po URL-u (*engl.* Uniform Resource Locator) te vrsti HTTP (*engl.* Hypertext Transfer Protocol) zahtjeva, njegovim parametrima, zaglavljima i tipovima medija, a može se koristiti na razini upravljača ili njegove metode.

Uz `@RequestMapping` postoje i skraćene specifičnije varijante anotacija za HTTP zahtjeve:

- `@GetMapping`
- `@PostMapping`
- `@PutMapping`
- `@PatchMapping`
- `@DeleteMapping`

Za rukovanje HTTP zahtjevom *GET*, može se koristiti anotacija `@RequestMapping` u sljedećem obliku `@RequestMapping(method=HttpMethod.GET)` ili se može koristiti anotacija `@GetMapping` kako bi se izbjeglo pisanje argumenta unutar `@RequestMapping` anotacije. Ostale skraćene anotacije se koriste na istovjetan način.

Koriste se i anotacije poput `@RequestBody`, koja označava tijelo HTTP zahtjeva s ulaznim podacima u obliku JSON-a ili XML-a te se ti ulazni podaci pretvaraju u objektni oblik putem metoda za rukovanje zahtjevima. Uz `@RequestBody` postoje anotacije `@PathVariable` i `@RequestParam` koje označavaju varijablu/e koje se pojavljuju u zahtjevu. Pri korištenju `@PathVariable` redoslijed varijabli je bitan, a za `@RequestParam` nije, zbog toga što se varijable dohvaćaju preko naziva umjesto preko redoslijeda, kao što je to u slučaju korištenja anotacije `@PathVariable` [18].

1.2.2. Spring Data JPA

Spring Data JPA dio je šire obitelji Spring Data komponenti koja pojednostavljuje implementaciju repozitorija čija je osnova JPA. JPA (*engl.* Java Persistence API), je Java specifikacija koja je prvobitno objavljena 2006. godine i omogućava određene funkcionalnosti i standarde ORM (*engl.* Object-Relational Mapping) alatima.

Neke od funkcionalnosti koje JPA omogućava su ispitivanje, kontrola i pohrana podataka između relacijskih baza podataka i Java objekata. Kao specifikacija, JPA ne nudi nikakve funkcije već zahtjeva implementaciju te zbog toga ORM alati poput ranije spomenutog Hibernatea implementiraju JPA specifikacije, koje se smatraju vezom između objektno orijentiranog modela i relacijskog sustava baza podataka.

Neke od ključnih značajki JPA specifikacije su:

- JPA nije implementacija već samo specifikacija
- Pruža skup pravila i smjernica za postavljanje sučelja za implementaciju ORM-a
- Zahtijeva nekoliko klasa i sučelja
- Podržava jednostavno, integrirano i čisto objektno-relacijsko mapiranje
- Podržava polimorfizam i nasljeđivanje
- Omogućuje dinamičke i imenovane upite

Za implementaciju JPA potrebno je koristiti neki radni okvir kao što je Hibernate. Hibernate je radni okvir i ORM alat za programski jezik Java koji pojednostavljuje razvoj Java aplikacija koje za rad koriste interakciju s bazom podataka. Korištenje Hibernatea omogućuje pohranu Java objekata u relacijsku bazu podataka [19].

Za omogućavanje JPA-a u projektu potrebno ga je dodati kao ovisnost kroz Maven. Kôd 1.8 prikazuje primjer iz pom.xml datoteke.

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

Kôd 1.8 Primjer ovisnosti korištene za dodavanje JPA-a u pom.xml kroz Maven

Korištenjem Spring Boota, Hibernate se postavlja kao zadani pružatelj JPA usluga stoga nije nužno definirati `entityManagerFactory` bean, osim u slučaju kada ga je potrebno dodatno prilagođavati. Osim toga, Spring Boot može automatski konfigurirati `DataSource` bean ovisno o bazi podataka koja se koristi.

U slučaju *in-memory* baze podataka, Spring Boot će automatski konfigurirati `DataSource` ako je odgovarajuća ovisnost prisutna na *classpath*-u, odnosno mjestu gdje Java runtime i Java compiler traže klase i resurse. Ako se želi koristiti JPA s MySQL bazom podataka potrebno je dodati `mysql-connector-java` ovisnost u pom.xml [20].

1.2.3. Spring Security

Spring Security je radni okvir i komponenta Springa koja pruža autentifikaciju, autorizaciju i zaštitu od uobičajenih napada u Java aplikacijama. Glavne značajke Spring Securityja su:

- Sveobuhvatna i nadogradiva podrška za autorizaciju i autentifikaciju
- Zaštita od CSRF (*engl.* Cross Site Request Forgery), *clickjacking*, *session fixation* napada, itd.
- Opcionalna integracija sa Spring Web MVC-jem
- Integracija Servlet API-a [21].

Korištenjem Spring Securityja aplikacija se štiti od raznih vrsta napada poput CSRF-a, napada *clickjacking*-om i *session fixation* napada. U CSRF napadu, napadač pokušava prevariti korisnika da nehotice izvrši neželjenu radnju na web stranici na kojoj je prethodno bio autentificiran. *Clickjacking* napadom, napadač zavarava korisnika da klikne na nešto drugačije od onog što misli da klikće, a tijekom *session fixation* napada, napadač preuzima kontrolu nad korisničkom sjednicom prije nego što je korisnik autentificiran [22].

Pri izradi aplikacije koriste se Spring Security anotacije poput `@EnableWebSecurity`, koja se koristi na razini konfiguracijske klase da bi se omogućila značajka web sigurnosti i označila konfiguracijska klasa koja definira sigurnosne postavke [23]. Također, koriste se anotacije poput `@EnableMethodSecurity` i `@PreAuthorize`. Anotacija `@EnableMethodSecurity` se koristi kako bi se omogućila sigurnosna zaštita na razini metoda aplikacije, a anotacija `@PreAuthorize` služi za provjeru pristupa prije izvršavanja metode te se može koristiti na razini klase za koju želimo postaviti provjeru pristupa ili same metode unutar te klase. Ako se anotacija `@PreAuthorize` koristi na razini klase i na razini metode unutar te klase, anotacija na razini metode će nadjačati anotaciju na razini klase [24].

1.3. PostgreSQL

PostgreSQL ili često zvan samo Postgres je objektno-relacijski sustav za upravljanje bazama podataka koji koristi i proširuje jezik SQL (*engl.* Structured Query Language) na način da ga kombinira sa značajkama koje omogućavaju sigurno pohranjivanje podataka sa zahtjevnim radnim opterećenjima.

Razvijen je na Berkeleyju 80-ih godina kao dio POSTGRES projekta te se i dalje aktivno razvija na osnovnoj platformi. Postgres se koristi zahvaljujući svojoj arhitekturi, pouzdanosti, integritetu podataka, proširivosti koju dopušta te mogućnosti da radi na svim značajnim operacijskim sustavima [25]. Korištenjem upita pisanih u jeziku SQL moguće je dohvaćati podatke iz tablica u bazi, kreirati nove tablice, dodavati podatke, ažurirati i brisati podatke, spajati podatke iz više tablica povezujući ih na temelju povezanih stupaca i slično.

Neke od bitnih naredbi u SQL upitima su `SELECT` za dohvaćanje stupaca tablice preko njihovog naziva ili korištenje * kako bi se dohvatili svi podaci unutar izabrane tablice, naredba `FROM` koja definira tablicu iz koje se podaci dohvaćaju, a uz dodatak naredbe `WHERE` dohvatit će se samo podaci koji ispunjavaju navedeni uvjet. Postoje i naredbe poput `DELETE` i `UPDATE` koje brišu ili ažuriraju podatak u tablici te naredba `INSERT INTO` koja dodaje nove retke u tablicu. SQL upit se piše kao kombinacija ranije navedenih naredbi kako bi se čim bolje specificiralo što se točno u upitu želi obaviti i nad kojom tablicom ili njezinim stupcima i redcima [26]. Primjer SQL upita dan je kôdom u nastavku pod nazivom: Kôd 1.9

```
SELECT *
FROM product
WHERE price > 199
```

Kôd 1.9 Primjer SQL upita

SQL upit prikazan kôdom Kôd 1.9 ostvaruje dohvaćanje svih stupaca naredbom `SELECT *` iz tablice product što se postiže korištenjem naredbe `FROM product` te na ekranu ispisuje stupce ispunjene podacima za proizvode čiji je iznos cijene veći od 199, a to se postiže korištenjem naredbe `WHERE price > 199`.

1.4. Angular

TypeScript radni okvir Angular koristi se za razvoj web aplikacija koje predstavljaju i manipuliraju podacima na efikasan i dinamičan način. Aktivnim ažuriranjem radnog okvira Angular, razvoj aplikacija postaje sve efikasnije, a efikasnost se posebno povećala uvođenjem podijele u različite module. Uvođenjem Angular CLI-a (*engl.* Command Line Interface) omogućeno je stvaranje aplikacije i pretvaranje složenih struktura u modularni oblik zbog mogućnosti korištenja naredbenog retka za instalaciju potrebnih paketa. Angular CLI rješava i probleme poput inicijalizacije novih projekata, generiranja komponenti, servisa i sl., uvođenjem automatizacije tih procesa.

Neke od ključnih značajki Angulara su:

- Korištenje komponenti i direktiva
- Podržan je na svim popularnim mobilnim preglednicima
- Za povezivanje podataka između prikaza i modela koriste se svojstva unutar „()“ i „[]“
- Pruža podršku za TypeScript i JavaScript

Zbog fleksibilnosti i ranije navedenih značajki, Angular je pogodan radni okvir za izradu jednostranih aplikacija, gdje se stvara jedna web stranica pa se sadržaj dinamički ažurira kako korisnik njome upravlja.

Angular se koristi i u razvoju aplikacija u stvarnom vremenu poput raznih aplikacija za razmjenu poruka, aplikacija za vremensku prognozu i sličnih u kojima je ključno trenutno ažuriranje [27]. Koristeći Angular CLI moguće je preko komandne linije pomoću naredbe `ng generate` ili skraćeno `ng g` automatizirati izradu potrebnih datoteka za Angular komponentu, servis i slično. Komponente se stvaraju u komandnoj liniji korištenjem naredbe `ng generate component naziv_komponente` ili u skraćenoj verziji `ng g c naziv_komponente`. Poželjno je nalaziti se u željenom direktoriju u kojem želimo stvoriti komponentu kako bi se izbjeglo nepotrebno premještanje datoteka. Pri stvaranju komponente pomoću naredbe, Angular stvara poddirektorij naziva istovjetnog nazivu komponente te se u njemu nalaze TypeScript, HTML i CSS datoteke potrebne za tu komponentu. Stvaranje servisa, modula i ostalih dijelova aplikacije odvija se na isti način koristeći naredbu `ng generate`, ali uz drugačiji nastavak [28].

1.5. Razvojna okruženja IntelliJ IDEA i WebStorm

Integrirano razvojno okruženje ili IDE (*engl.* Integrated Development Environment) je aplikacija čija je namjena pružanje pomoći prilikom pisanja kôda neke aplikacije. Cilj IDE-a je povećati produktivnost na način da kombinira funkcionalnosti poput uređivanja kôda, izgradnje, testiranja te na kraju pakiranja u jednostavnu aplikaciju. Iako se za pisanje kôda može koristiti bilo koji uređivač teksta, većina IDE-a uključuje dodatne funkcionalnosti koje uređivači teksta ne pružaju poput centralnog sučelja za zajedničke alate u razvoju softvera te tako čine proces razvoja bržim i jednostavnijim.

Automatizacija uređivanja kôda moguća je zbog toga što radna okruženja u sebi imaju napisana pravila kako bi kôd određenog programskog jezika trebao biti strukturiran. Pružaju i mogućnost isticanja sintakse koristeći se raznim oblikovanjima fonta za isticanje ključnih riječi i sl. Mnoga razvojna okruženja dodatno pružaju mogućnost inteligentnog dovršavanja kôda što uvelike skraćuje vrijeme potrebno za pisanje. Podrška za restrukturiranje, lokalna automatizacija izgradnje, mogućnost kompilacije kôda, testiranja, otklanjanja poteškoća i slično neke su od ostalih značajku razvojnih okruženja [29].

IntelliJ IDEA je integrirano razvojno okruženje za programske jezike Java i Kotlin. Razvila ga je tvrtka JetBrains kako bi programerima maksimalno povećala produktivnost. Razvojno sučelje IntelliJ pruža automatizaciju rutinskih i ponavljajućih zadataka, inteligentno dovršavanje kôda, statičku analizu i podršku za restrukturiranje. Iako je IntelliJ namijenjen programskim jezicima Java i Kotlin, može se koristiti i s drugim jezicima koji se mogu kompilirati u JVM *bytecode* poput jezika Scala i Groovy [30].

Samo razvojno sučelje je vrlo prilagodljivo i korisnici mogu mijenjati način isticanja ključnih riječi ili cijelu temu uređivača te ga mogu u potpunosti prilagoditi svojim potrebama u vizualnom izgledu i funkcionalnostima koje im pruža. Osim vizualne i funkcionalne prilagodljivosti IntelliJ podržava i razne dodatke koji mu omogućavaju dodavanje novih funkcionalnosti za ostale programske jezike.

U razvojno okruženje integrirana je kontrola verzija koja se može koristiti s Git-om, GitHub-om ili GitLab-om kroz integrirano grafičko sučelje [31]. WebStorm je razvojno okruženje koje značajkama i izgledom jako nalikuje IntelliJ-u iz razloga što je to, također, proizvod kojeg je razvila tvrtka JetBrains. WebStorm i IntelliJ dijele značajke poput ranije navedenog automatskog dovršavanja kôda, olakšanog restrukturiranja, statičke analize, korištenja

kontrolom verzija kroz sučelje i slično, što ga čini brzim i efikasnim radnim okruženjem za razvoj web aplikacija [32].

Bitna razlika između IntelliJ-a i WebStorm-a nalazi se u njihovoj namjeni. Glavna namjena razvojnog okruženja WebStorm, je olakšati pisanje programskog kôda i povećati produktivnost pri razvoju web aplikacija ili njihovih dijelova koji se pišu u programskim jezicima JavaScript, TypeScript, HTML, CSS i ostalim povezanim tehnologijama [33].

2. Proces izrade web aplikacije

Proces izrade web aplikacije započinje njezinim planiranjem nakon kojeg slijedi izrada njezinih ključnih dijelova, a to su: baza podataka, pozadinski dio aplikacije i korisničko sučelje. Za izradu i razvoj aplikacije korištene su ranije navedene tehnologije, Postgres baza podataka, radni okvir Spring za izradu pozadinskog dijela aplikacije poznatijeg kao *backend*, a za izradu korisničkog sučelja ili takozvanog *frontenda* korišten je radni okvir Angular.

Inicijalizacija Spring projekta na *backendu* može se započeti korištenjem alatom Spring Initializr koji omogućuje stvaranje praznog projekta uz odabir trenutno potrebnih ovisnosti kroz grafičko sučelje tog alata. Nove potrebne ovisnosti moguće je dodati u pom.xml datoteku u slučaju potrebe u kasnijem razvoju projekta. Potrebno je prvo inicijalizirati *backend* kako bi se Spring Boot projekt mogao koristiti pri stvaranju modela baze podataka.

2.1. Planiranje i izrada modela baze podataka

Izrada baze podataka započinje izradom modela prema potrebama i zahtjevima korisnika definiranih u fazi planiranja aplikacije. Nakon definiranja zahtjeva, prelazi se na konceptualni dizajn pri čemu se koristi entitetsko-relacijski (ER) model za vizualizaciju podataka i njihovih odnosa. Svaka tablica ima jedan atribut koji služi kao primarni ključ.

Primarni ključ je atribut, koji je jedinstven za svaki podatak upisan u neku tablicu te se u pravilu dodjeljuje automatski. Kako bi se tablice mogle stvarati u bazi podataka moguće je koristiti SQL upite, ali ih je također moguće stvoriti na ranije spomenut način tako da se baza podataka poveže sa Spring Boot aplikacijom koristeći se application.yml datotekom u koju se upisuju podaci potrebni za povezivanje.

U application.yml datoteku potrebno je upisati URL baze podataka, korisničko ime te lozinku za pristup bazi kao što prikazuje isječak kôda, Kôd 2.1.

```
spring:
  datasource:
    url: jdbc:postgresql://localhost:5432/itshop-db
    username: itshop
    password: S3cret
  jpa:
    hibernate:
      ddl-auto: update
  servlet:
    multipart:
      max-file-size: 10MB
      max-request-size: 10MB
```

Kôd 2.1 Primjer application.yml datoteke

Nakon uspješnog povezivanja s bazom podataka, započinje se sa stvaranjem klasa objekata čije će varijable postati istoimeni atributi u tablici entiteta. Klase objekta koje će se pretvarati u tablice entiteta u bazi podataka, potrebno je označiti anotacijom `@Entity` i anotacijom `@Table` kojom će se odrediti naziv tablice. Potrebno je jednu od varijabli objekta označiti anotacijom `@Id` kako bi se ta varijabla mogla koristiti kao primarni ključ, nakon čega je potrebno tu istu varijablu označiti anotacijom `@GeneratedValue` u kojoj se odabire strategija `GenerationType.IDENTITY`.

Kôd 2.2 prikazuje klasu `User` koja je označena anotacijama `@Entity` i `@Table` te njezine varijable `private Long id` koja je označena anotacijama `@Id` i `@GeneratedValue`.

```
@Entity
@Getter
@Setter
@Table(name = "`user`")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String firstName;
    private String lastName;
    private String username;
    private String email;
    private String password;
    private String phoneNumber;
    private String address;

    @Enumerated(EnumType.STRING)
    private Role role;

}
```

Kôd 2.2 Primjer entiteta

Osim klasa objekata koji predstavljaju entitete, u paketu model, nalaze se i enumeracijske klase. Enumeracijske klase u sebi sadrže konstante koje se koriste za određivanje mogućih vrijednosti neke varijable u objektu te dodatne metode poput *gettera* ako je to potrebno.

Slika 2.1 Primjer paketa model prikazuje paket model u kojem se nalaze sve klase koje su označene anotacijom `@Entity` jer predstavljaju jedan od entiteta u bazi podataka.

Slika 2.1 Primjer paketa model

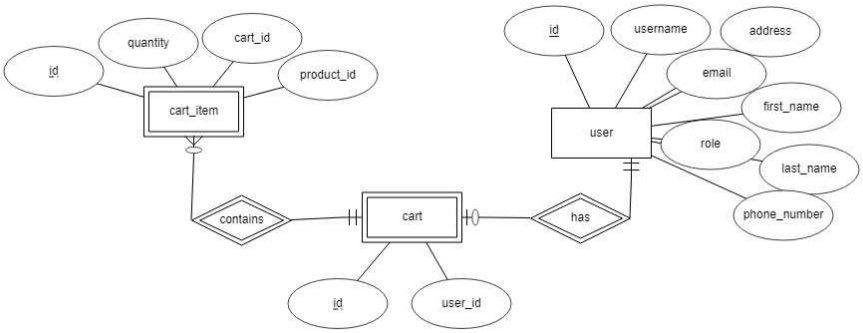
Osim ranije navedenih anotacija, u klasama objekata koji čine entitete još se koriste anotacije: `@OneToOne`, `@OneToMany`, `@ManyToOne` koje određuju vrstu veze između entiteta i anotacija `@JoinColumn` koja određuje naziv atributa koji predstavlja strani ključ.

Kôd 2.3 prikazuje primjer korištenja ranije navedenih anotacija u klasi koja predstavlja entitet.

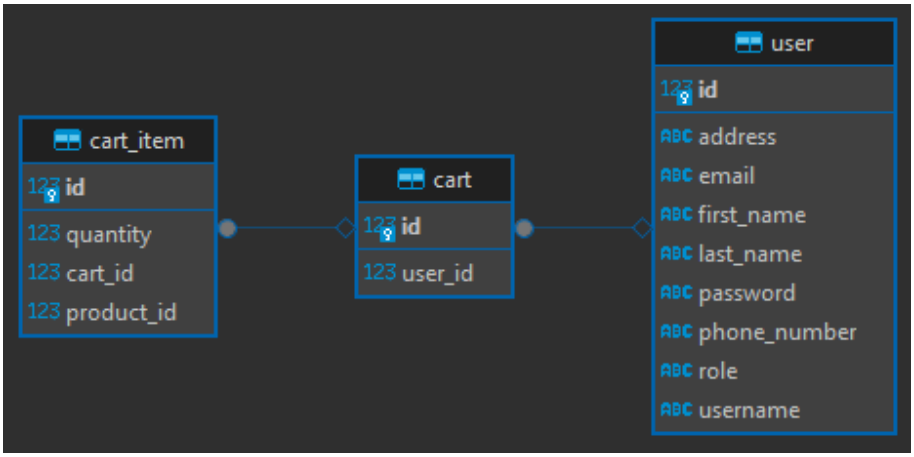
```
@Entity
@Getter
@Setter
@Table(name = "`cart`")
public class Cart {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @OneToOne
    @JoinColumn(name = "user_id")
    private User user;

    @OneToMany(mappedBy = "cart", cascade = CascadeType.ALL)
    private List<CartItem> cartItemList = new ArrayList<>();

}
```

Kôd 2.3 Primjer ostalih anotacija entiteta

Nakon stvaranja svih objekata koji će se koristiti kao entiteti, pokretanjem aplikacije u bazi se stvaraju tablice. Slika 2.2 prikazuje primjer ER modela, a Slika 2.3 prikazuje primjer relacijske sheme.



Slika 2.2 Primjer ER modela baze podataka



Slika 2.3 Primjer relacijske sheme

2.2. Proces izrade pozadinskog dijela aplikacije (backenda)

Proces izrade pozadinskog dijela aplikacije započinje nakon inicijalizacije projekta i izrade modela baze podataka. Prvi korak u izradi *backenda* je postavljanje sigurnosne konfiguracije. Za postavljanje sigurnosne konfiguracije koristi se klasa `WebSecurityConfiguration` u kojoj se koriste anotacije iz Spring Security radnog okvira. Kôd 2.4 prikazuje primjer korištene `WebSecurityConfiguration` klase i potrebne Spring Security anotacije.

```
@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
@EnableMethodSecurity
public class WebSecurityConfiguration {

      ...
}
```

Kôd 2.4 Primjer WebSecurityConfiguration klase

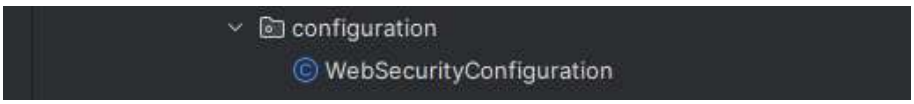
Klasa se označava anotacijom `@Configuration` kako bi Spring znao da predstavlja konfiguracijsku klasu i dodaju se još anotacije `@EnableWebSecurity` i `@EnableMethodSecurity` kako bi se omogućilo korištenje web sigurnosti i sigurnosti na razini metoda. Zatim se u klasu `WebSecurityConfiguration` počinju pisati metode koje se označavaju anotacijom `@Bean` kako bi se stvorili JavaBeanovi iz vrijednosti koje te metode vraćaju.

Kôd 2.5 prikazuje primjer jedne metode koja se nalazi u klasi `WebSecurityConfiguration` i označena je anotacijom `@Bean`.

```
@Bean
public AuthenticationProvider authenticationProvider() {
      final DaoAuthenticationProvider authenticationProvider
= new DaoAuthenticationProvider();
      authenticationProvider.setUserDetailsService(userDetail
      sService);

      authenticationProvider.setPasswordEncoder(passwordEncod
      er());
      return authenticationProvider;
}
```

Kôd 2.5 Primjer metode iz klase WebSecurityConfiguration

Metodu `authenticationProvider()` nužno je označiti anotacijom `@Bean` jer se rade izmjene u odnosu na uobičajeni kôd metode. Osim te metode, u klasu `WebSecurityConfiguration` dodaju se i ostale potrebne metode za autorizaciju i autentifikaciju. Sve klase koje služe za konfiguraciju projekta i označene su anotacijom `@Configuration`, nalaze se u zasebnom paketu pod nazivom configuration što prikazuje Slika 2.4.



Slika 2.4 Primjer paketa configuration

Nakon postavljanja konfiguracijskih klasa prelazi se na stvaranje sučelja repozitorija koje služi za komunikaciju *backenda* s bazom podataka. Za komunikaciju se koristi Hibernate koji je implementacija JPA-a i omogućuje da se rad s podacima u bazi podataka obavlja kroz klase objekata umjesto direktnog pisanja SQL upita.

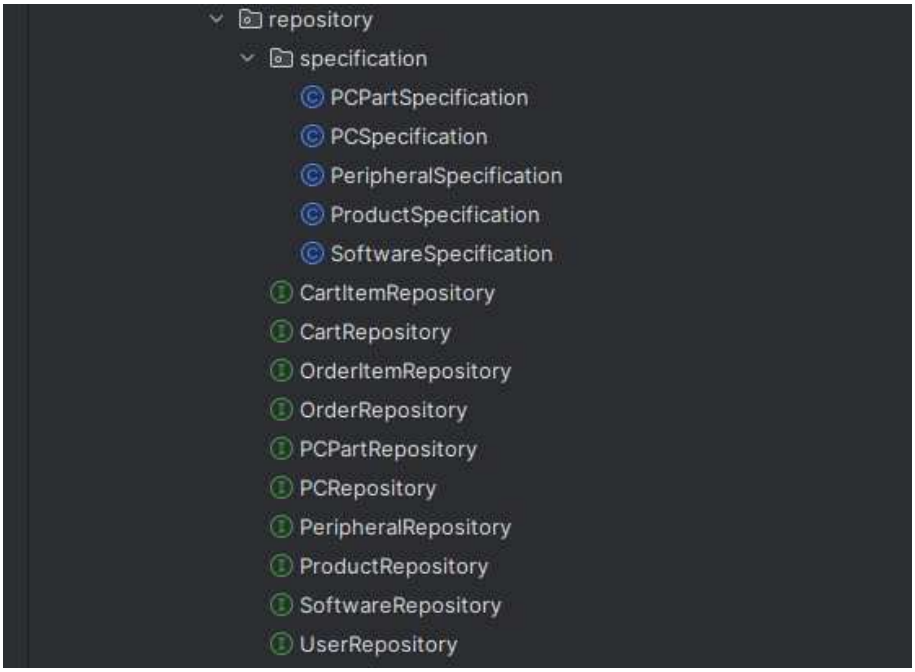
Svako sučelje koje predstavlja jedan od repozitorija označava se anotacijom `@Repository` te nasljeđuje generičko sučelje `JpaRepository` koje pruža osnovne CRUD (engl. *Create, Read, Update, Delete*) operacije i nekoliko dodatnih metoda za rad s entitetima u bazi podataka.

Kôd 2.6 prikazuje primjer sučelja `UserRepository` koje nasljeđuje generičko sučelje `JpaRepository` u kojem se kao generički tipovi redom postavljaju entitet `User` i tip varijable koja predstavlja primarni ključ u bazi podataka.

```
@Repository
public interface UserRepository extends JpaRepository<User,
Long> {

    User findByUsername(String username);
    Boolean existsByUsername(String username);
    Boolean existsByEmail(String email);
}
```

Kôd 2.6 Primjer sučelja UserRepository

U sučelju `UserRepository` nalaze se metode za rad s bazom podataka koje nije potrebno dodatno implementirati zbog nasljeđivanja sučelja `JpaRepository`. Sučelja repozitorija nalaze se u paketu repository kao što to prikazuje Slika 2.5.



Slika 2.5 Primjer paketa repository

Osim sučelja repozitorija, u paketu se nalazi paket *specification* koji sadrži klase koje su potrebne da bi se omogućila filtracija entiteta po određenim vrijednostima njihovih atributa.

Nakon stvaranja potrebnih sučelja repozitorija, može se započeti sa stvaranjem DTO-a (*engl.* Data Transfer Object) koji služe za prijenos podataka između različitih slojeva aplikacije i

nalaze se u paketu pod nazivom dto. Kôd 2.7 prikazuje primjer *recorda* `UserDto` koji u sebi sadrži potrebne varijable koje je potrebno poslati *frontendu*.

```
public record UserDto(
        String firstName,
        String lastName,
        String username,
        String email,
        String phoneNumber,
        String address,
        Role role
) {
}
```

Kôd 2.7 Primjer recorda UserDto

Pretvorba DTO-a u objekte entiteta vrši se pomoću `Mapper` sučelja specifičnih za taj dto ili objekt, a korištenjem ovisnosti `MapStruct` može se postići automatsko generiranje kôda tih mappera uz označavanje sučelja anotacijom `@Mapper`. U tim sučeljima je dovoljno postaviti objekt kao tip podatka koji metoda vraća i kao ulazni argument postaviti objekt koji prima, kako bi se izvršila pretvorba ulaznog objekta u izlazni. Kôd 2.8 prikazuje primjer `mapper` sučelja koje pretvara entitet `Cart` u DTO `CartDto` i obratno.

```
@Mapper(componentModel = "spring", uses =
{CartItemMapper.class, UserMapper.class})
public interface CartMapper {
    Cart cartDtoToCart(CartDto cartDto);
    CartDto cartToCartDto(Cart cart);
}
```

Kôd 2.8 Primjer sučelja CartMapper

Ako objekt u sebi sadrži drugi objekt, može se koristiti svojstvo `uses`. Korištenjem tog svojstva, navode se nazivi drugih postojećih *mappera* za objekt koji je sadržan u objektu nad kojim se vrši pretvorba. To se radi kako bi *mapper* koristio neki od već postojećih *mappera* umjesto da u sebi stvara dodatne metode za pretvorbu tog sadržanog objekta.

Po završetku implementacije repozitorija, objekata za prijenos i *mappera*, započinje se sa stvaranjem servisa i pisanjem implementacija njihovih metoda. Servisi se nalaze u paketu *service*, a njihova implementacija u paketu pod nazivom impl unutar paketa *service*.

U implementaciji servisa nalaze se kôdovi svih metoda koje se nalaze u njegovom sučelju. Implementaciju klase servisa označava se anotacijom `@Service` te joj se dodaje anotacija `@RequiredArgsConstructor`. Klasa implementacije servisa koristi se *beanovima* repozitorija i *mappera* za ostvarivanje potrebnih funkcionalnosti poput dohvaćanja entiteta iz baze podataka i slično.

Kôd 2.9 prikazuje primjer implementacija servisa `ProductServiceImpl` koja u sebi sadrži metode za dohvaćanje proizvoda iz baze podataka, brisanje određenog proizvoda te povećavanja broja posjeta određenog proizvoda kroz korisničko sučelje.

```
@Service
@RequiredArgsConstructor
public class ProductServiceImpl implements ProductService {

    private final ProductRepository productRepository;
    private final ProductMapper productMapper;

    public ProductDto getProductById(final Long id) {
        final Product product =
productRepository.findById(id)
                .orElseThrow(() -> new
EntityNotFoundException(String.format("Product with ID(%d)
not found!", id)));

        return productMapper.productToProductDto(product);
    }

    public List<ProductDto> getAllProducts(final
ProductFilter filter) {
        final ProductSpecification<Product> specification =
new ProductSpecification<>(filter);
        final List<Product> products =
productRepository.findAll(specification);
        return productMapper.productsToProductDtos(products);
    }

    public void deleteProduct(final Long id) {
        final Product product =
productRepository.findById(id)
```

```
                .orElseThrow(() -> new
EntityNotFoundException(String.format("Product with ID(%d)
not found!", id)));
        productRepository.delete(product);
    }

    public void incrementTimesVisitedForProduct(final Long
id) {
        productRepository.incrementTimesVisited(id);
    }
}
```

Kôd 2.9 Primjer implementacije servisa za proizvode

Posljednji dio razvoja *backenda* predstavlja stvaranje upravljača. Upravljači se označavaju ranije spomenutom anotacijom `@RestController` te u sebi sadrže metode od kojih svaka predstavlja jedan API. Kôd 2.10 prikazuje primjer `AccountControllera` koji u sebi sadrži krajnje točke koje se koriste za dohvaćanje korisnika (*engl*. user), izradu korisnika, ažuriranje njegovih podataka, promjenu lozinke i brisanje korisnika.

```
@RestController
@RequestMapping("/api/account")
@RequiredArgsConstructor
public class AccountController {

    private final UserService userService;

    @GetMapping("/all")
    @PreAuthorize("hasRole('ROLE_ADMIN')")
    public List<UserDto> getUsers() {
        return userService.getAllUsers();
    }
    @GetMapping("/{userId}")
    @PreAuthorize("hasRole('ROLE_ADMIN') ||
hasRole('ROLE_USER')")
    public UserDto getUserById(@PathVariable final Long
userId) { return userService.getUserById(userId); }

    @PostMapping
    @PreAuthorize("hasRole('ROLE_ADMIN')")
```

```
    public void createUser(@RequestBody final RegisterDto
registerDto) {
        userService.createUser(registerDto);
    }

    @PutMapping("/{userId}")
    @PreAuthorize("hasRole('ROLE_ADMIN')")
    public void updateUser(@PathVariable final Long userId,
@RequestBody final UpdateUserDto userUpdateDto) {
        userService.updateUser(userId, userUpdateDto);
    }

    @PatchMapping("/{userId}")
    @PreAuthorize("hasRole('ROLE_ADMIN') ||
@currentUserService.isCurrentUserMakingRequest(#userId)")
    public void changeUserPassword(@PathVariable final Long
userId,
                                   @RequestBody final
PasswordChangeDto changePasswordDto) throws
            PasswordComplexityException,
            SamePasswordException,
            WrongPasswordException {
        userService.changeUserPassword(userId,
changePasswordDto);
    }

    @DeleteMapping("/{userId}")
    @PreAuthorize("hasRole('ROLE_ADMIN')")
    public void deleteUser(@PathVariable final Long userId) {
        userService.deleteUser(userId);
    }
}
```

Kôd 2.10 Primjer AccountController upravljača

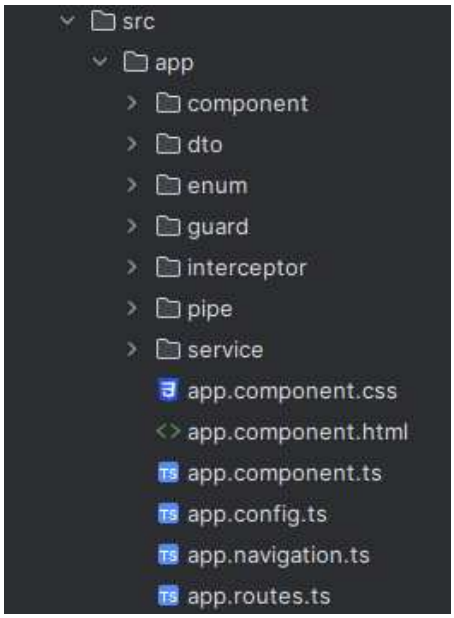
U upravljaču, koristi se anotacija `@RequestMapping` za postavljanje zadanog dijela URL adrese za sve metode, dok se na razini svake metode koristi anotacija u ovisnosti o željenom tipu zahtjeva.

Ako se anotacija tipa zahtjeva, na razini metode, koristi bez zagrada dovoljno je poslati samo zahtjev tog tipa na URL adresu zadanu anotacijom `@RequestMapping`. U suprotnom se na

URL adresu zadanu u anotaciji `@RequestMapping` nadovezuje dodatak adresi zadan u anotaciji tipa zahtjeva. Osim *mapping* anotacija, koristi se i anotacija `@PreAuthorize` kako bi neke krajnje točke bile dostupne samo korisnicima s određenom razinom prava.

2.3. Proces izrade korisničkog sučelja (frontenda)

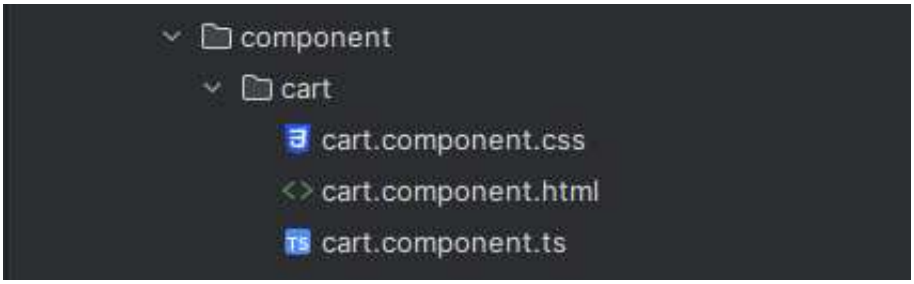
Proces izrade korisničkog sučelja započinje inicijalizacijom Angular projekta. Inicijalizacija projekta izvršava se korištenjem komandnom linijom pomoći Angular CLI alata. U direktoriju u kojem se želi inicijalizirati projekt potrebno je otvoriti komandnu liniju te upisati naredbu `ng new naziv_projekta`, u ovom slučaju `ng new zavrsni-rad-it-shop-fe`. Nakon inicijalizacije projekta, unutar direktorija *app*, koji se nalazi u novonastalom direktoriju *src*, može se započeti sa stvaranjem novih direktorija u koje će se smještati komponente, servisi i ostali potrebni dijelovi. Slika 2.6 prikazuje jednu od mogućih struktura projekta.



Slika 2.6 Primjer strukture projekta

Pozicioniranjem u direktorij component može se započeti sa stvaranjem komponente. Komponenta se stvara tako da se nakon pozicioniranja u željeni direktorij u komandnoj liniji koristi naredba `ng generate component naziv_komponente`.

Za stvaranje komponente `cart` koristi se naredba `ng generate component cart` koja stvara novi direktorij naziva *cart* u direktoriju component te u njega dodaje TypeScript, HTML i CSS datoteke naziva cart.component. Slika 2.7 prikazuje strukturu direktorija *component* i novonastale komponente `cart`.



Slika 2.7 Primjer strukture direktorija component

Nakon stvaranja nove komponente, započinje se s pisanjem TypeScript kôda koji određuje kojim funkcionalnostima raspolaže ta komponenta. Kôd 2.11 prikazuje izgled TypeScript dijela komponente `LoginComponent`, koja sadrži metodu `onSubmit()` i koristi se servisom `AuthService`.

```
@Component({
  selector: 'app-login',
  standalone: true,
  imports: [
    MatFormField, MatButton, MatInput, MatFormFieldModule,
    ReactiveFormsModule
  ],
  templateUrl: './login.component.html',
  styleUrl: './login.component.css'
})
export class LoginComponent {
  private readonly authService = inject(AuthService);
  protected loginForm = new FormGroup({
    'username': new FormControl(null, Validators.required),
    'password': new FormControl(null, Validators.required)
  });
  onSubmit() {
    this.authService.login(this.loginForm.value);
  }
}
```

Kôd 2.11 Primjer TypeScript kôda komponente LoginComponent

Funkcionalnost metode `onSubmit()` koristi se u HTML dijelu komponente, a Kôd 2.12 prikazuje isječak kôda u kojem se koristi.

```
<form (ngSubmit)="onSubmit()" [formGroup]="loginForm">
  <h1>Login</h1>
      ...
</form>
```

Kôd 2.12 Primjer isječka HTML kôda komponente LoginComponent

Kao što je ranije navedeno, komponenta `LoginComponent` koristi `AuthService`. Stvaranje servisa postiže se tako da se u *app* direktoriju napravi novi direktorij naziva *service*. Nakon pozicioniranja u direktorij *service*, u komandnoj liniji se koristi naredba `ng generate service auth` čime se stvara nova TypeScript datoteka auth.service.

Servis `AuthService` u sebi sadrži metode koje omogućavaju funkcionalnosti poput registracije, prijave, odjave i slično uz komunikaciju s *backend* dijelom aplikacije. Kôd 2.13 prikazuje primjer `AuthService` servisa i njegovih metoda.

```
export class AuthService {
  private readonly http = inject(HttpClient);
  private readonly navigation = inject(AppNavigation);
  private readonly snackBar = inject(MatSnackBar);
  protected url = environment.apiUrl + '/auth';
  protected user: User | undefined = undefined;

  register(data: any) {
    return this.http.post<User>(this.url + '/register',
data).subscribe(() => {
      this.snackBar.open("Successfully Registered!", "✔", {
duration: 3000, verticalPosition: 'top' });
      this.navigation.navigateToLogin();
    }); }
  login(data: any) {
    return this.http.post(this.url + '/login',
data).subscribe((response: any) => {
      localStorage.setItem('access_token', response.token);
      this.user = response.user;
      this.navigation.navigateToHome();
    }); }
```

```
  logout() {
    localStorage.removeItem('access_token');
    this.user = undefined;
    this.navigation.navigateToLogin();
  }
  getUserByToken() {
    return this.http.get<User>(this.url +
'/current').pipe(map((user: User) => {
      this.user = user;
      return user;
    }));
  }
  isLoggedIn() {
    if (this.user !== undefined) {
      return true;
    }
    if (this.tokenExists()) {
      return this.getUserByToken().subscribe();
    } else {
      return false;
    }
  }
  isAdmin() {
    if (this.user !== undefined) {
      return this.user.role === Role.ROLE_ADMIN;
    }
    return false;
  }
  tokenExists() {
    const token = localStorage.getItem('access_token');
    return !!token;
  }
  getCurrentUser() {
    return this.user;
  }
}
```

Kôd 2.13 Primjer AuthService klase

Nakon izrade komponenti i servisa, prelazi se na postavljanje putanja u TypeScript datoteci app.routes. Kôd 2.14 prikazuje primjer postavljanja ruta u datoteci app.routes.

```
export const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent },
  { path: 'login', component: LoginComponent, canActivate:
[notLoggedInGuard] },
  { path: 'register', component: RegisterComponent,
canActivate: [notLoggedInGuard] },
  { path: 'create-product', component: ProductFormComponent,
canActivate: [adminGuard] },
  { path: 'edit-product/:id', component:
ProductFormComponent, canActivate: [adminGuard]  },
  { path: 'product/:id', component: ProductComponent },
  { path: 'product' , component: ProductListComponent},
  { path: 'cart', component: CartComponent, canActivate:
[loggedInGuard] },
  { path: 'order', component: OrderHistoryComponent,
canActivate: [loggedInGuard] },
  { path: 'order/:id', component: OrderDetailsComponent,
canActivate: [loggedInGuard] },
  { path: '**', redirectTo: '/home' }
];
```

Kôd 2.14 Primjer sadržaja datoteke app.routes

Putanje zapisane u datoteci app.routes koriste se za navigaciju između komponenti aplikacije. Osim komponenti, servisa i putanja, aplikacija može sadržavati *pipe* ili više njih, a koriste se za oblikovanje podataka kao što su datumi, brojevi, tekst i slično tako da budu čitljiviji i lakši za održavanje.

Isječak kôda pod nazivom: Kôd 2.15 prikazuje *pipe* koji se koristi za prikaz skraćenog dijela teksta opisa proizvoda u HTML datoteci.

```
<div class="product-info-container">
      <div class="product-info">
            ...
             <p>{{ product.description | truncate:200 }}</p>
            ...
      </div>
      ...
</div>
```

Kôd 2.15 Primjer korištenja pipe alata

Kôd koji definira funkcionalnost koju obavlja *pipe*, zapisan je u datoteci truncate.pipe.ts koja se nalazi u direktoriju *pipe*. Kôd 2.16 prikazuje primjer definiranja funkcionalnosti koju obavlja `truncate pipe`.

```
@Pipe({
  name: 'truncate',
  standalone: true
})
export class TruncatePipe implements PipeTransform {

  transform(value: string, limit: number = 100, trail: string
= '...'): string {
    return value.length > limit ? value.substring(0, limit) +
trail : value;
  }

}
```

Kôd 2.16 Primjer kôda datoteke truncate.pipe.ts

Po završetku stvaranja svih ranije navedenih dijelova *frontend* projekta, može se početi vizualno uređivati korisničko sučelje koristeći CSS datoteke u komponentama tako da se u svakoj komponenti napiše CSS kôd koji će toj komponenti izmijeniti izgled u korisničkom sučelju.

3. Funkcionalnosti web aplikacije

Ključni aspekti aplikacije uključuju funkcionalne zahtjeve, arhitekturu sustava, dizajn baze podataka i vizualni prikaz korisničkog sučelja. Funkcionalni zahtjevi predstavljaju sve specifične funkcionalnosti koje web aplikacija mora ispuniti kako bi zadovoljila potrebe korisnika. Njima se opisuje na koji način sustav radi, fokusirajući se na konkretne zadatke, ponašanja i usluge koje aplikacija treba pružiti.

Arhitektura sustava predstavlja organizaciju i strukturu sustava, definirajući njegove ključne komponente te njihov međusobni odnos. Osim toga pruža osnovni okvir na kojem se temelji proces razvoja softvera, osiguravajući da sve komponente rade zajedno kako bi se postigli funkcionalni i nefunkcionalni zahtjevi sustava.

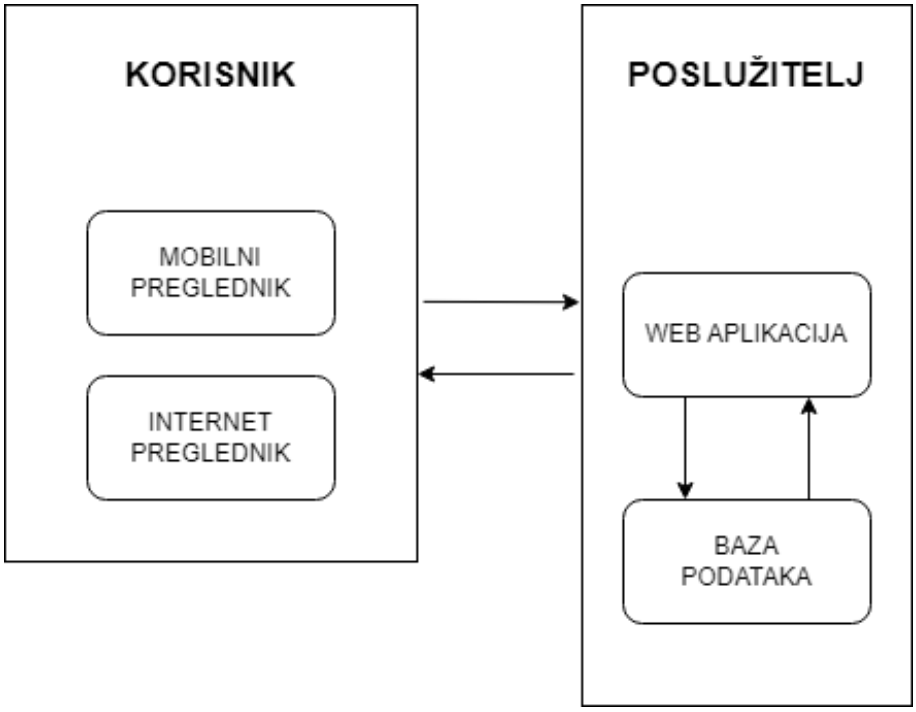
U ovom poglavlju prikazan je dizajn baze podataka korištene u aplikaciji i slike ekrana korisničkog sučelja. Cilj ovog poglavlja je pružiti sveobuhvatan pregled tehničkih elemenata koji podržavaju rad aplikacije i osiguravaju njezinu funkcionalnost.

3.1. Funkcionalni zahtjevi i arhitektura aplikacije

Funkcionalni zahtjevi obuhvaćaju izradu web aplikacije koja će služiti kao internetska trgovina za novu i rabljenu informatičku opremu. Aplikacija korisnicima mora omogućiti pregledavanje i pretraživanje proizvoda. Korisnicima mora biti omogućeno sortiranje proizvoda po nazivu i cijeni, filtriranje po kategorijama, vrsti, rasponu cijene i ostalim parametrima koji su specifični za odabranu kategoriju proizvoda. Svim korisnicima će biti dostupne ranije navedene funkcionalnosti i mogućnosti prijave ili registracije korisničkog računa, a za pristup ostalim funkcionalnostima, korisnici će morati biti prijavljeni u sustav. Prijavljeni korisnici imat će jednu od tri razine prava, koja će se određivati prema vrsti korisničkog računa: korisnik, dobavljač ili administrator sustava. Ključne funkcionalnosti internetske trgovine uključuju dodavanje proizvoda u integriranu košaricu za kupnju, proces naplate te mogućnost pregleda svih obavljenih kupnji. Korisnicima s ulogom dobavljača moraju biti dostupne sve do sad navedene funkcionalnosti uz mogućnost dodavanja novih proizvoda. Administratorima sustava moraju biti omogućene sve funkcionalnosti i dodavanje novih vrsta proizvoda za određenu kategoriju, brisanje proizvoda, administracija postojećih korisnika te stvaranje novih korisničkih računa za sve vrste korisnika.

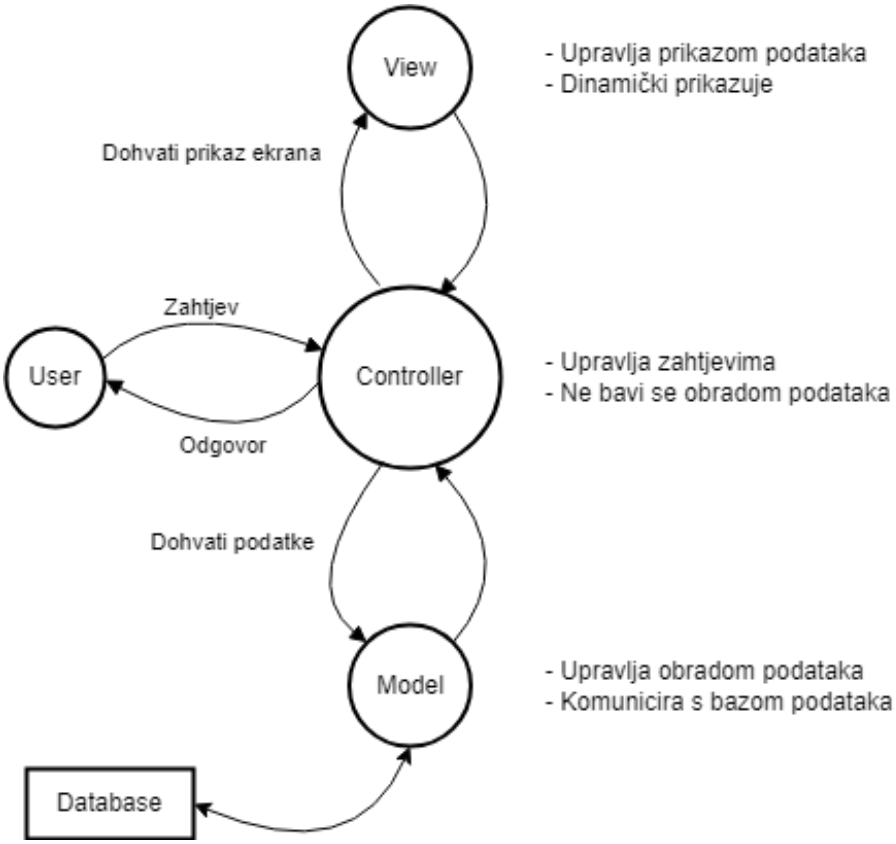
Arhitektura web aplikacije sastoji se od dva glavna dijela: klijent i poslužitelj. Klijent obavlja interakciju s korisnikom aplikacije, prikazuje podatke i šalje zahtjeve na poslužitelj. Poslužitelj obavlja obradu zahtjeva od strane klijenta, pristupa bazi podataka te šalje odgovore klijentu. Takva arhitektura omogućava efikasno raspoređivanje zadataka, centralizirano upravljanje podacima te rast i upravljanje povećanom potražnjom. Klijenti mogu biti različiti uređaji poput osobnih računala i mobitela, a funkciju poslužitelja obavlja jedno ili više računala na kojima se nalaze *frontend* i *backend* dijelove od kojih je sačinjena web aplikacija te baza podataka.

Slika 3.1 prikazuje primjer arhitekture web aplikacije gdje korisnik ima ulogu klijenta koristeći se mobilnim preglednikom ili internet preglednikom na računalu, a poslužitelj sadrži web aplikaciju koja mu šalje tražene podatke i komunicira s bazom podataka.



Slika 3.1 Prikaz arhitekture aplikacije

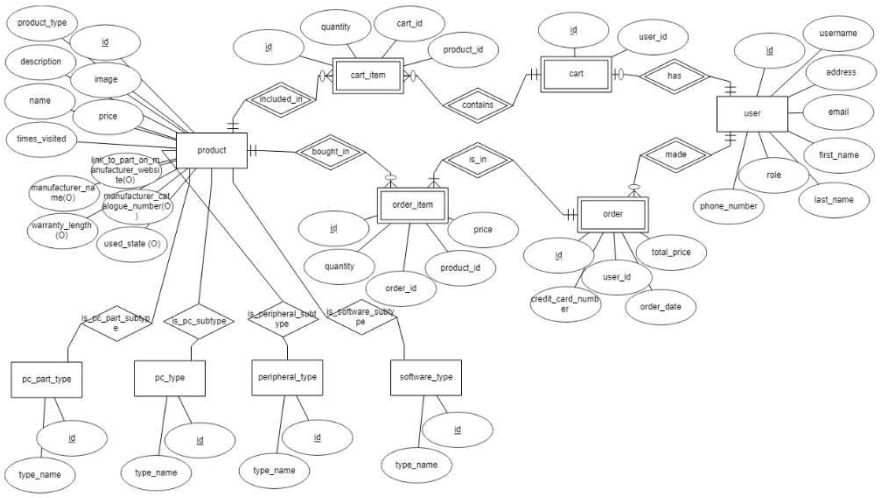
U arhitekturi poslužitelja koristi se MVC (*engl*. Model-View-Controller) obrazac u svrhu organizacije kôda. *Model* upravlja podatkovnom logikom i komunicira s bazom podataka. *View* komponenta stvara korisničko sučelje koje popunjava podacima koje dobiva iz *Modela*, a *Controller* obrađuje korisničke zahtjeve i upravlja protokom podataka između *Model* i *View* komponenti. Primjer MVC-a prikazan je Slika 3.2.



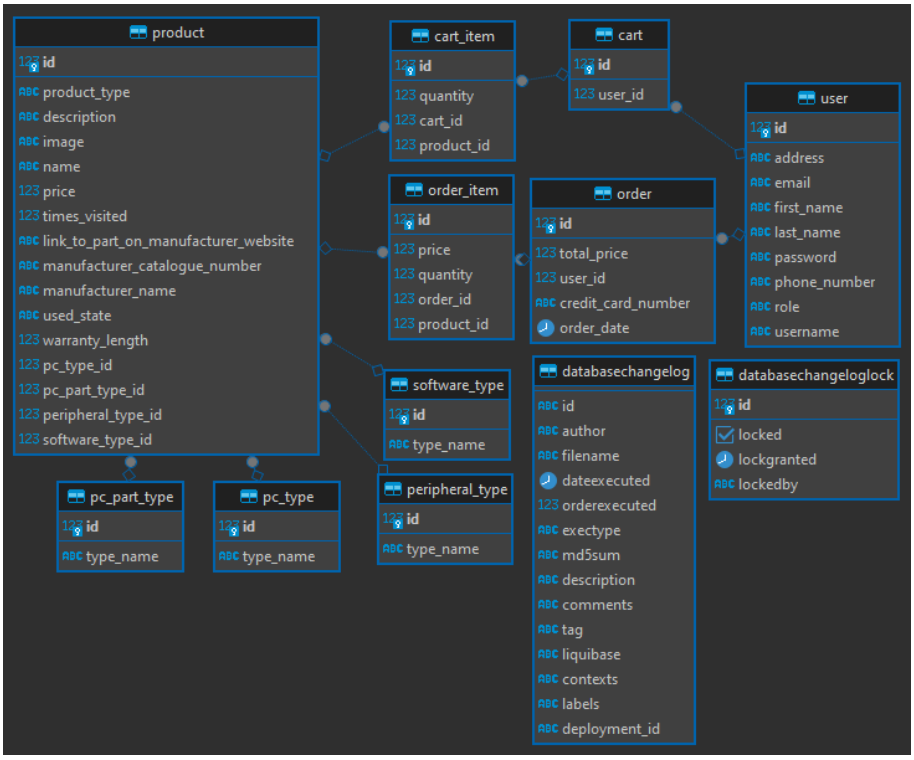
Slika 3.2 Prikaz MVC arhitekture

Arhitektura koja koristi MVC omogućava jasnu podjelu odgovornosti, olakšava održavanje i nadogradnju sustava te ponovnu upotrebu kôda.

Baza podataka korištena u aplikaciji prikazana je Slika 3.3 koja sadrži ER dijagram, a izgled tablica u bazi podataka prikazan je relacijskom shemom koju prikazuje Slika 3.4.



Slika 3.3 Prikaz ER dijagrama baze podataka



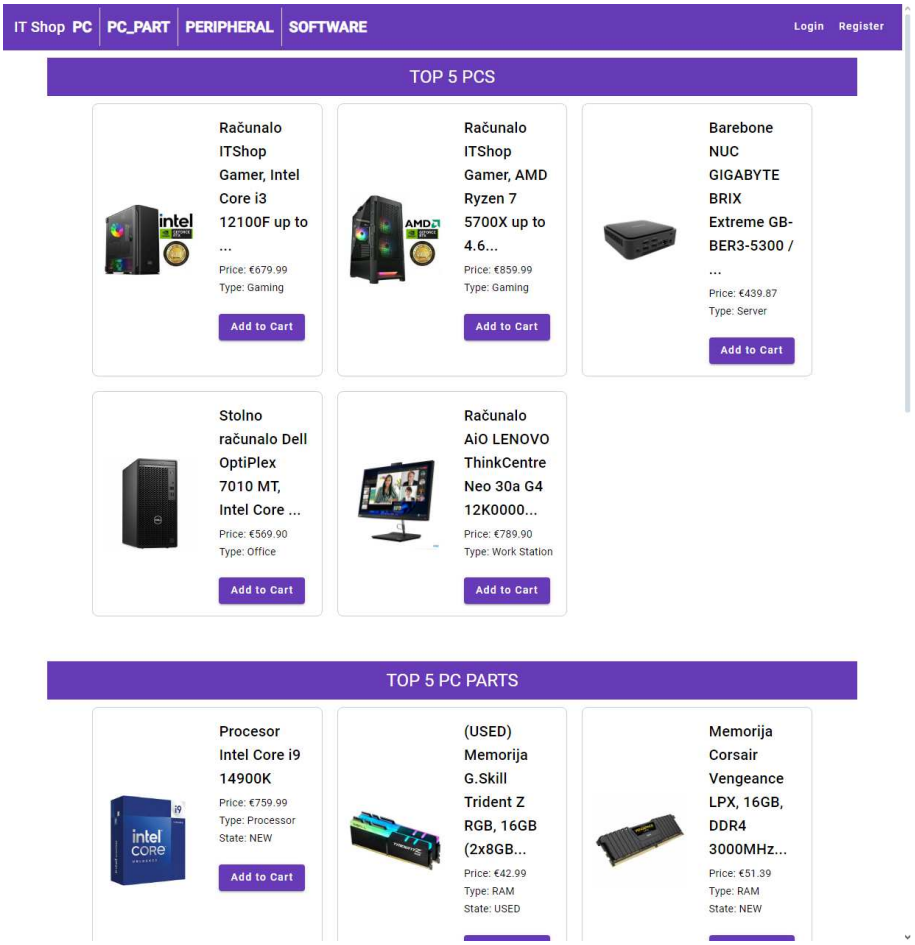
Slika 3.4 Prikaz relacijske sheme baze podataka

Tablice „databasechangelog“ i „databasechangeloglock“ se ne nalaze u ER dijagramu zato što se automatski stvaraju korištenjem liquibase-a. Te tablice u sebi ne sadrže podatke koje će aplikacija koristiti već samo podatke o promjenama koje se vrše nad bazom podataka.

Baza podataka prikazana Slika 3.4 sadrži središnju tablicu „*user*“ u kojoj se nalaze podaci o registriranim korisnicima poput njihovog imena, adrese, vrste korisničkog računa i slično. Osim tablice „*user*“, u bazi podataka nalazi se i središnja tablica „*product*“ koja sadrži informacije o proizvodima poput naziva proizvoda, kategorije, cijene, vrste, slike, itd. Tablica „*cart*” u sebi sadrži podatak o pripadnosti korisniku (*user_id*), a tablica „*cart_item*“ u sebi sadrži podatke o identifikatoru košarice (*cart_id*) kojoj pripada, identifikator proizvoda (*product_id*) i količinu tog proizvoda (*quantity*). Podaci o narudžbama spremaju se u tablicu „*order*“ koja sadrži podatak o ukupnoj cijeni narudžbe (*total_price)*, korisniku kojem pripada (*user_id*) te broju kreditne kartice i datumu narudžbe. U tablici „*order_item*“ nalaze se svi podaci sadržani u tablici „cart_item“ uz dodatak o podatku cijene proizvoda u trenutku narudžbe. Tako se omogućava točan povrat novca u slučaju povrata proizvoda ako se cijena proizvoda u međuvremenu promijenila. U bazi podataka nalaze se još četiri tablice koje u sebi sadrže nazive potkategorija za svaku od mogućih kategorija proizvoda.

3.2. Opis ekrana internetske trgovine

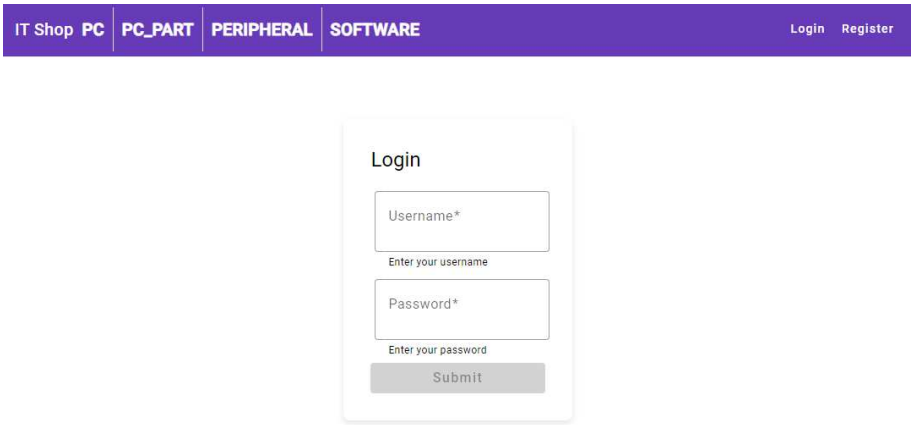
Otvaranjem aplikacije u internet pregledniku, korisniku se prikazuje početna stranica na kojoj se prikazuju najposjećeniji proizvodi iz svake kategorije što prikazuje Slika 3.5.



Slika 3.5 Prikaz početne stranice i navigacijske trake

Korisnik može pogledati detalje određenog proizvoda klikom na bilo koji dio kartice prikaza pojedinog proizvoda. Postoji mogućnost odabira između prikaza proizvoda određene kategorije ili potkategorije kroz navigacijsku traku koja se nalazi na vrhu zaslona.

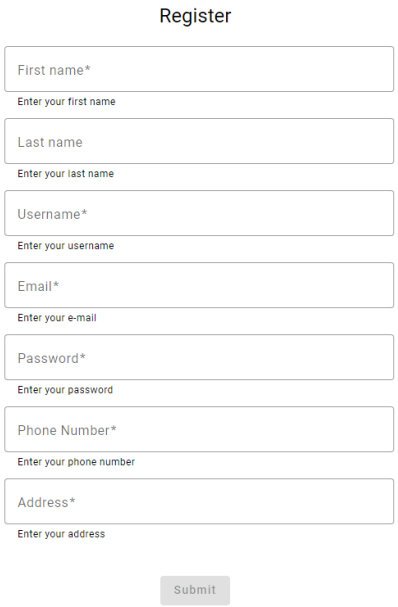
Klikom na jedan od gumba koji se nalaze na desnoj strani navigacijske trake moguće je prijaviti se u aplikaciju ili stvoriti novi korisnički račun.Klikom na gumb za dodavanje u košaricu, korisnika se preusmjerava na stranicu za prijavu što prikazuje Slika 3.6.



Slika 3.6 Prikaz stranice za prijavu

Ako korisnik ne posjeduje korisnički račun, potrebno ga je napraviti prije prijave klikom na gumb „*Register*“ koji otvara stranicu s obrascem za registraciju korisnika u koji se upisuju potrebni podaci.

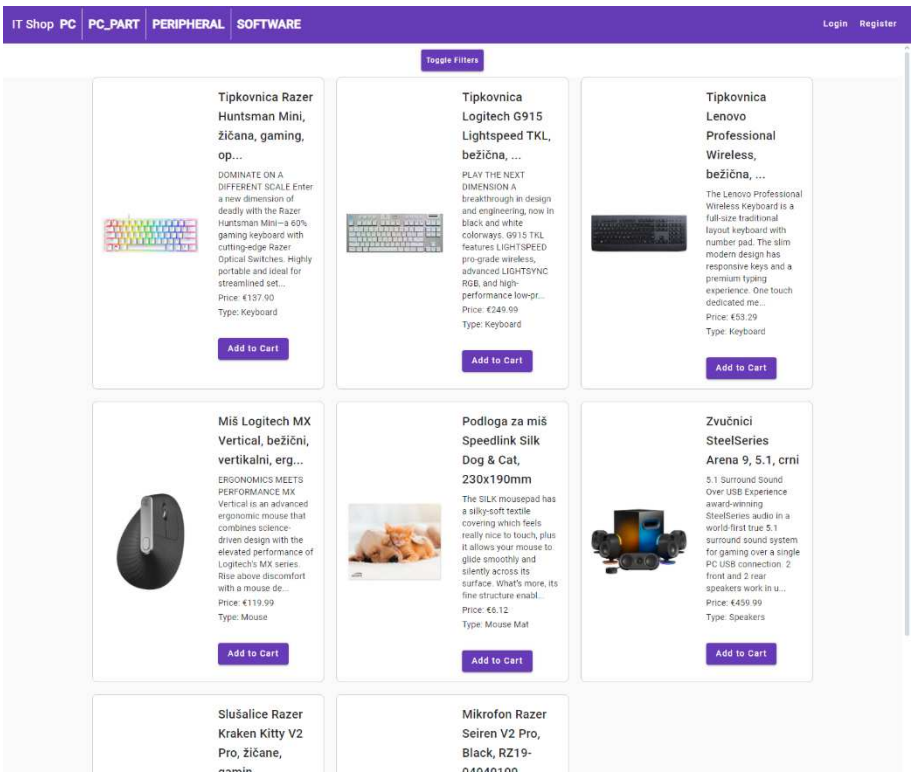
Slika 3.7 prikazuje stranicu s obrascem za registraciju korisničkog računa.



Slika 3.7 Prikaz stranice za registraciju

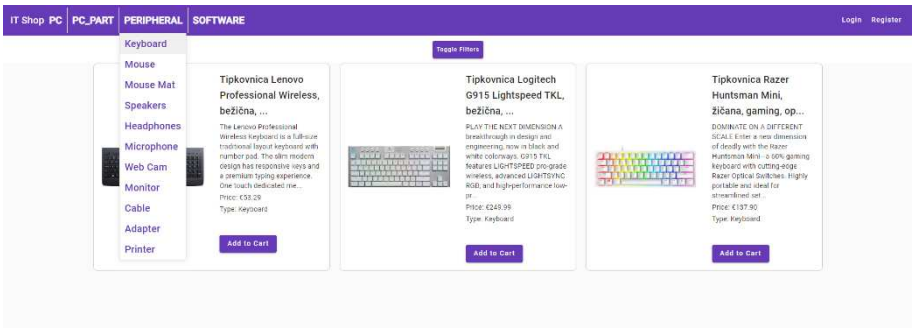
Na lijevoj strani navigacijske trake korisnik može odabrati jednu od četiri ponuđene kategorije, jednu od potkategorija iz padajućeg izbornika koji se prikazuje prelaskom

pokazivača preko naziva kategorije ili se vratiti na početnu stranicu klikom na „*IT Shop*“. Slika 3.8 prikazuje stranicu koja se otvara kada korisnik izabere kategoriju „*PERIPHERAL*“.



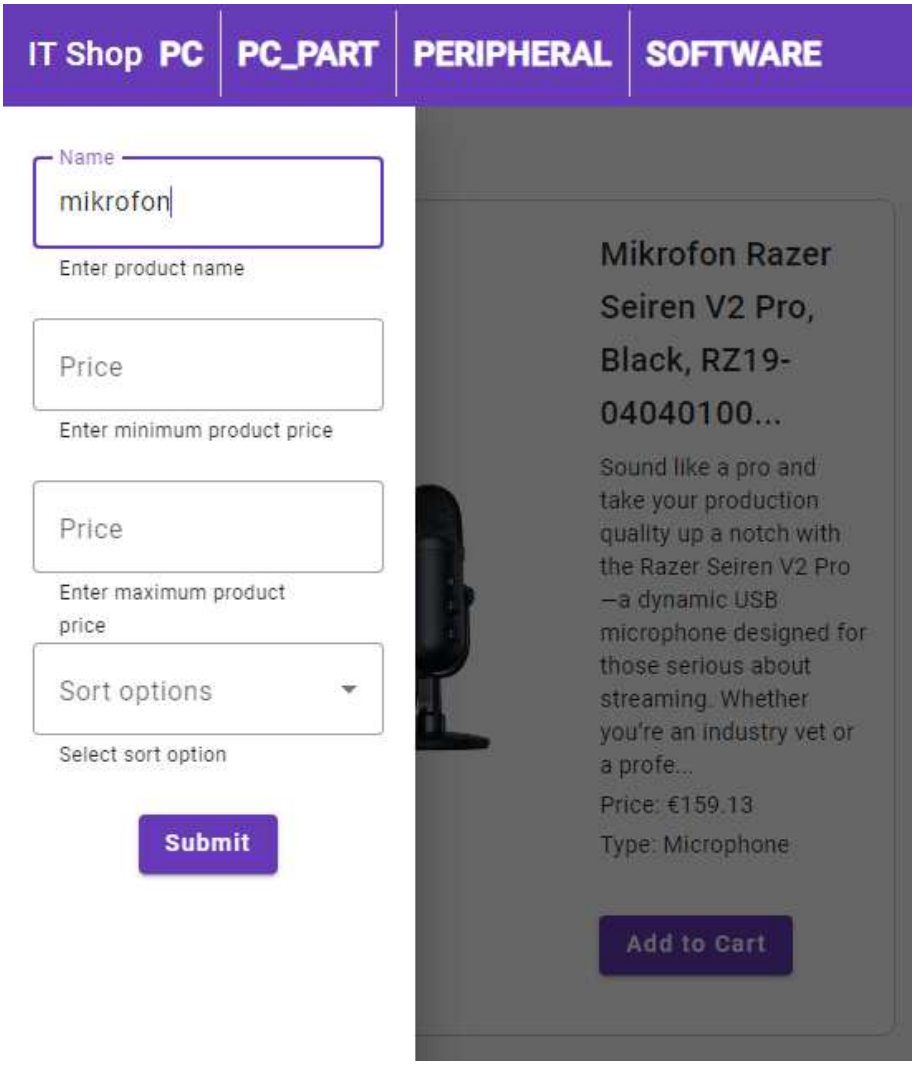
Slika 3.8 Prikaz kategorije periferije (*engl.* Peripheral)

Prelaskom pokazivača preko kategorije periferije i odabirom potkategorije tipkovnice (*engl.* Keyboard) iz padajućeg izbornika, korisniku se otvara stranica u čijem sadržaju su prikazani samo proizvodi iz odabrane potkategorije što je prikazano Slika 3.9

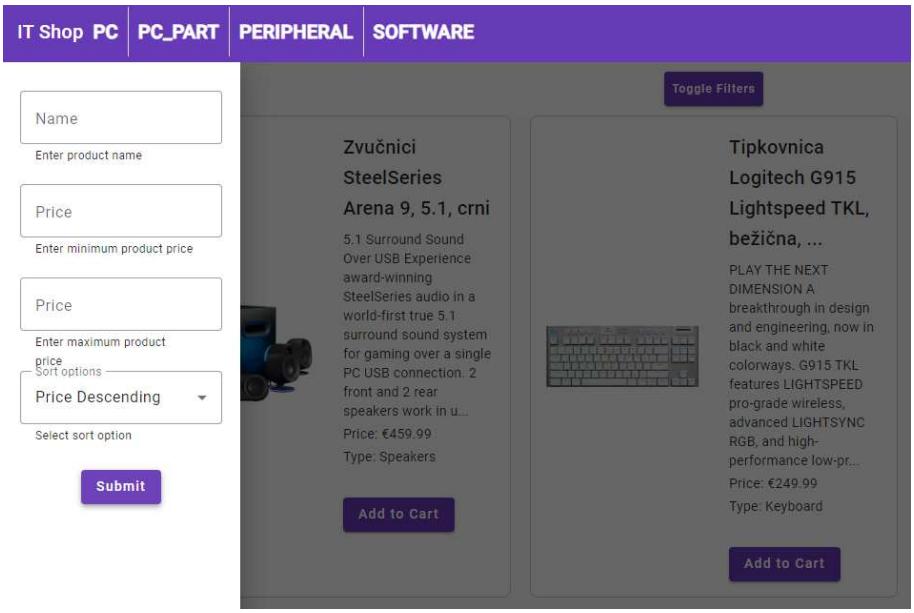


Slika 3.9 Prikaz stranice s tipkovnicama

Korisniku je omogućeno dodatno filtriranje i sortiranje proizvoda po različitim parametirima kroz obrazac koji se prikazuje s lijeve strane zaslona nakon klika mišem na gumb „*Toggle filters*“. Slika 3.10 i Slika 3.11 prikazuju obrazac u koji se mogu upisati parametri prema kojima se proizvodi filtriraju i sortiraju.



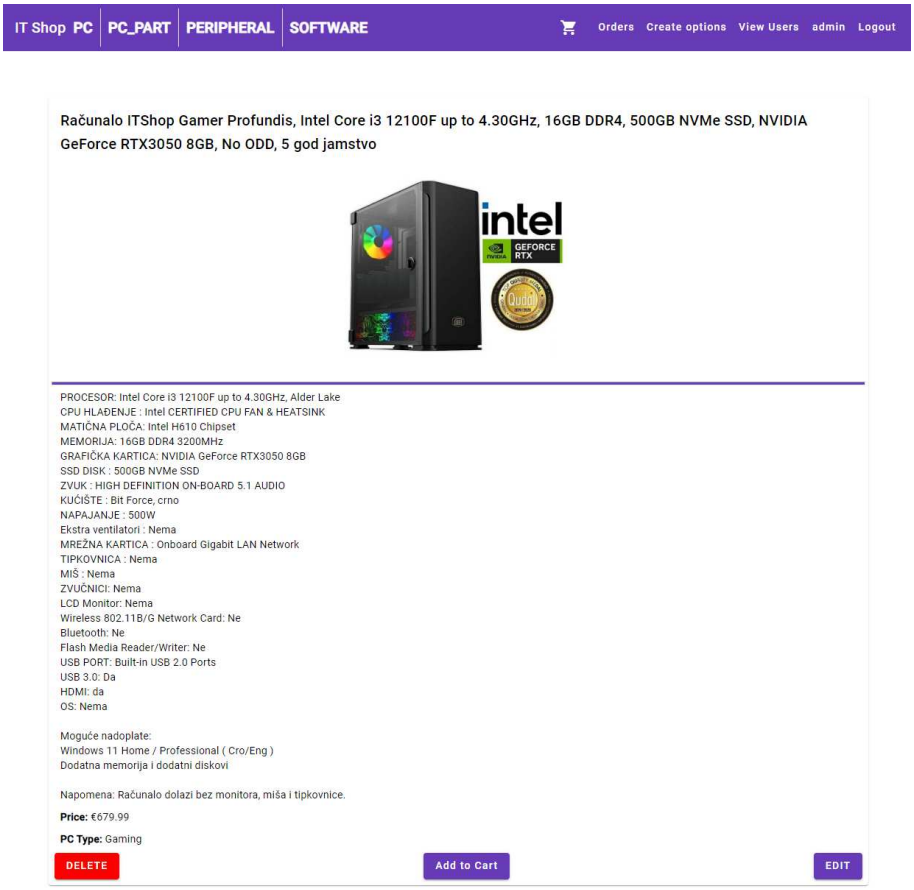
Slika 3.10 Prikaz filtiranja proizvoda po nazivu



Slika 3.11 Prikaz sortiranja proizvoda padajuće po cijeni

Klikom na karticu proizvoda, otvara se prikaz stranice proizvoda s dodatnim detaljima i gumbom koji omogućava dodavanje proizvoda u košaricu ako je korisnik prijavljen, u suprotnom korisnika se preusmjerava na stranicu za prijavu.

Uz gumb za dodavanje u košaricu, prikazuju se i dodatni gumbi za uređivanje kartice i brisanje proizvoda ako prijavljeni korisnik ima ulogu administratora stranice što prikazuje Slika 3.12. Kartica proizvoda sastoji se od naziva, slike, detaljnog opisa proizvoda te njegove cijene i vrste.

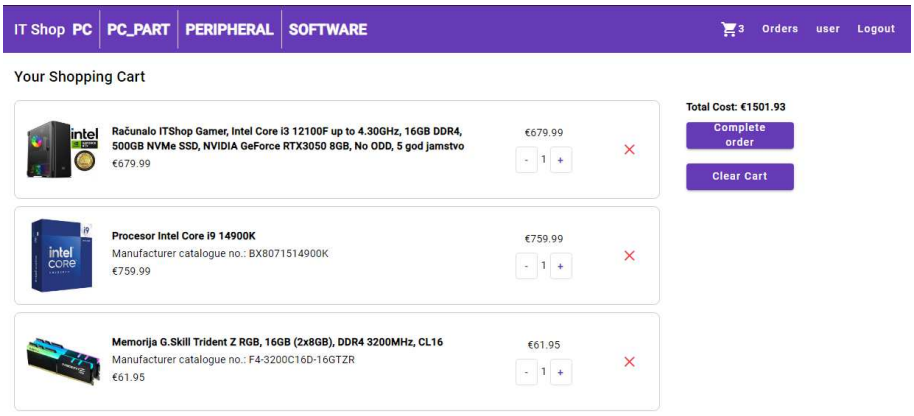


Slika 3.12 Prikaz detalja o proizvodu

Otvaranjem detalja o proizvodu, povećava se broj posjeta tom proizvodu te se taj statistički podatak koristi za prikaz pet najposjećenijih proizvoda. Ako je korisnik prijavljen u sustav, klikom na gumb „*Add to cart*“ proizvod se dodaje u košaricu. Korisnik mora biti prijavljen zbog toga što se trenutno stanje košarice čuva u bazi podataka kako bi korisnik mogao nastaviti kupnju na bilo kojem drugom računalu na kojem je prijavljen.

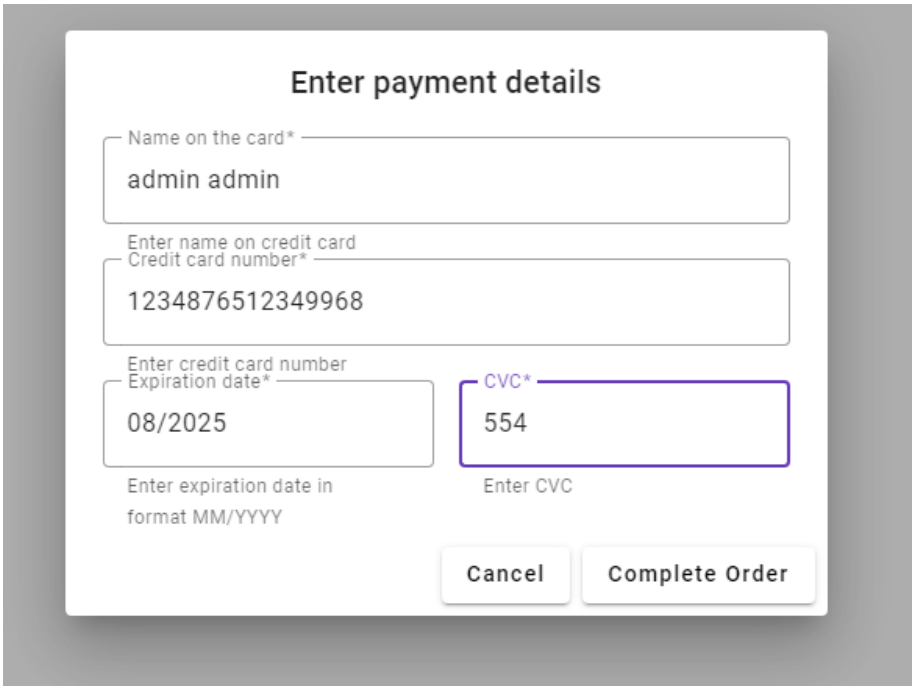
Za dodavanje veće količine proizvoda korisnik može ponovno stisnuti gumb „*Add to cart*“ ili dodati još primjeraka kada otvori prikaz košarice. Prikaz košarice otvara se klikom na oznaku košarice u navigacijskoj traci. Dodavanjem proizvoda u košaricu povećava se broj uz oznaku košarice koji označava broj različitih proizvoda koji se nalaze u njoj. Slika 3.13 prikazuje praznu košaricu, a Slika 3.14 prikazuje košaricu koja sadrži proizvod te gumbe za povećavanje ili smanjivanje količine, za brisanje proizvoda iz košarice te gumbe koji omogućavaju pražnjenje košarice ili proces naplate.

Slika 3.13 Prikaz prazne košarice



Slika 3.14 Prikaz košarice koja sadrži proizvod

Klikom na gumb „*Complete order*“ korisniku se prikazuje prozor za upisivanje podataka o kreditnoj kartici. Slika 3.15 prikazuje prozor s upisanim podacima za naplatu koji se satoje od imena i prezimena vlasnika, broja i sigurnosnog koda kartice te datuma do kojeg je kartica važeća.

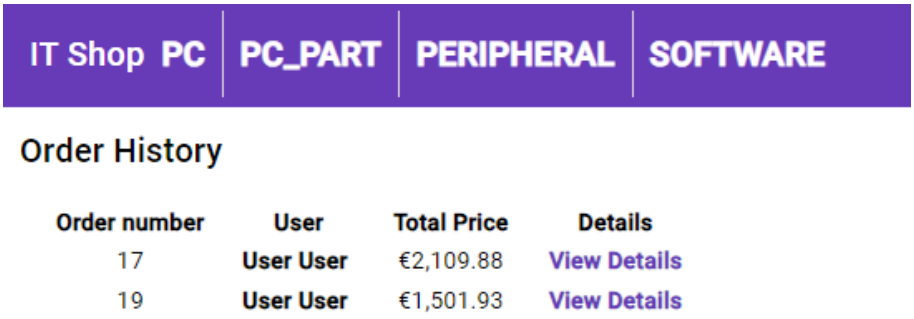


Slika 3.15 Prikaz prozora s podacima za naplatu

Nakon popunjavanja podataka, ponovnim klikom na gumb „*Complete order*“, korisnika se preusmjerava na stranicu s detaljima narudžbe. Slika 3.16 prikazuje stranicu *Order details* na kojoj se nalaze svi detalji narudžbe nakon obavljenog plaćanja.

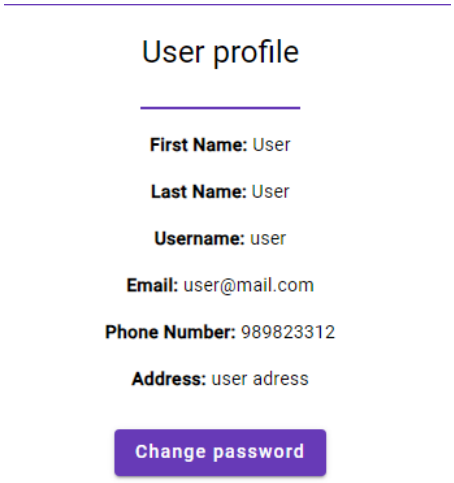
Slika 3.16 Prikaz detalja narudžbe (*engl.* Order details)

Do detalja o narudžbi, može se doći i klikom na gumb „*Orders*“ na lijevoj strani navigacijske trake, te se tamo može izabrati narudžba za koju korisnik želi pogledati detalje klikom na gumb „*View detalis*“, kao što prikazuje Slika 3.17.



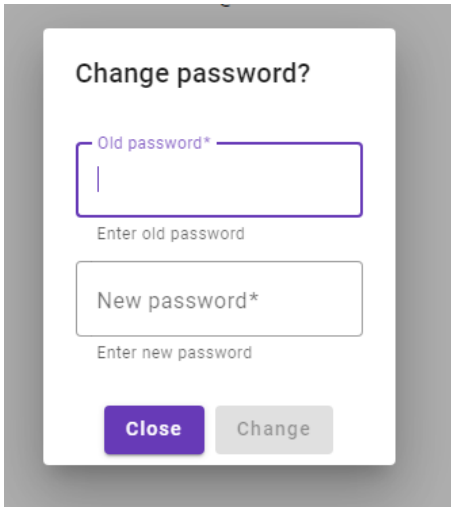
Slika 3.17 Prikaz povijesti narudžbi (*engl.* Order history)

Osim ranije navedenih funkcionalnosti, korisnik može pregledati podatke vlastitog korisničkog računa te promijeniti lozinku. Stranica korisničkog računa otvara se klikom na korisničko ime na desnoj strani navigacijske trake. Slika 3.18 prikazan je izgled stranice korisničkog računa s podacima korisnika i gumbom „*Change password*“ koji služi za promjenu lozinke korisničkog računa.



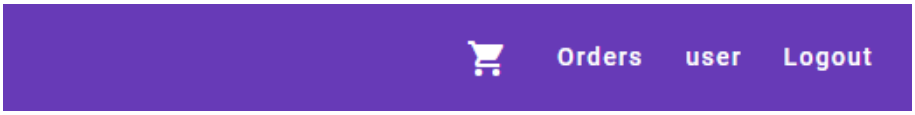
Slika 3.18 Prikaz stranice korisničkog računa

U trenutnoj implementaciji stranice korisničkog računa, korisnik može mijenjati samo lozinku. U slučaju da korisnik želi promijeniti ostale podatke potrebno je kontaktirati administratore. Klikom na gumb „*Change password*“ otvara se prozor s obrascem za promjenu lozinke korisničkog računa prikazan Slika 3.199.

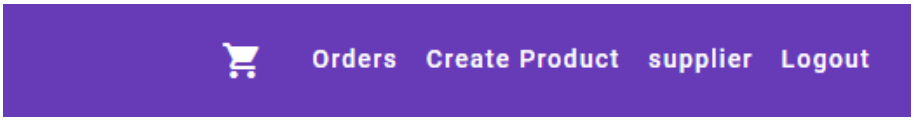


Slika 3.19 Prikaz obrasca za promijenu lozinke

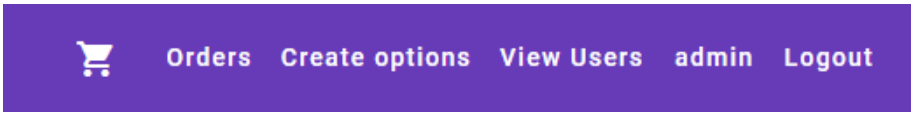
Sve ranije navedene funkcionalnosti dostupne su svim vrstama prijavljenih korisnika, a korisnicima s posebnim ovlastima dostupne su neke od dodatnih funkcionalnosti koje se nalaze na desnoj strani navigacijske trake. Slika 3.20 prikazuje desnu stranu navigacijske trake običnog korisnika, Slika 3.21 prikazuje desnu stranu navigacijske trake korisnika koji ima ovlasti dobavljača, a Slika 3.22 prikazuje izgled desne strane navigacijske trake korisnika s administratorskim ovlastima.



Slika 3.20 Prikaz navigacijske trake običnog korisnika



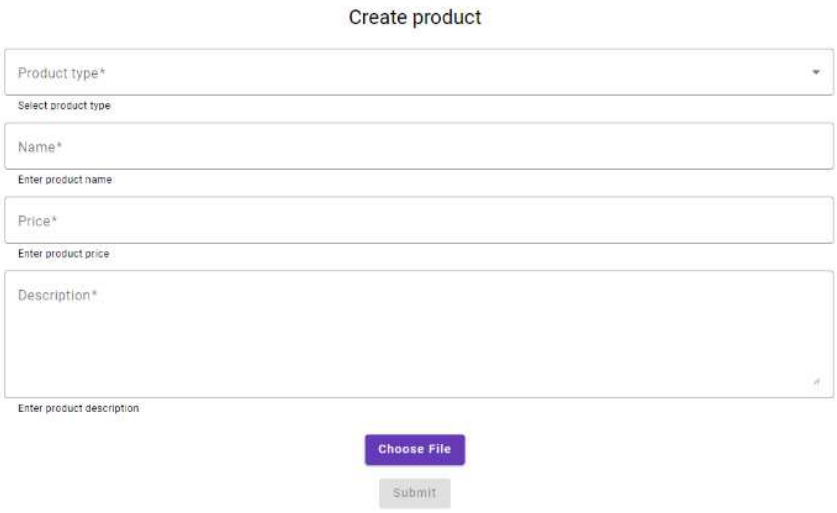
Slika 3.21 Prikaz navigacijske trake dobavljača



Slika 3.22 Prikaz navigacijske trake administratora

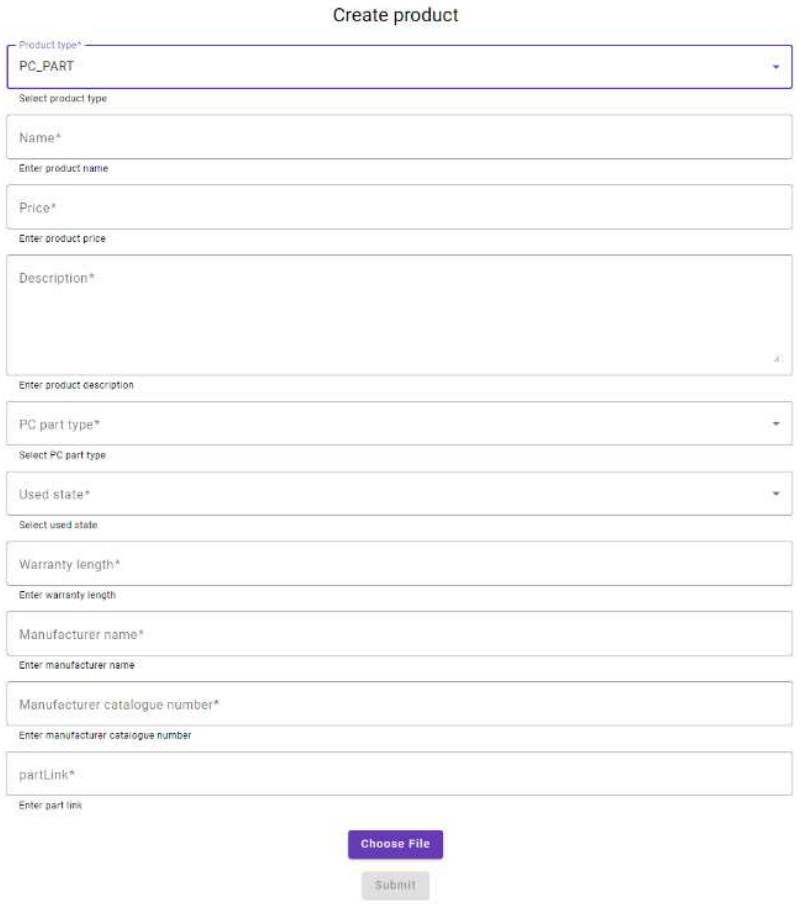
Korisniku s ulogom dobavljača, dostupna je funkcionalnost stvaranja novog proizvoda. Klikom na gumb „*Create product*“ otvara se nova stranica koja sadrži obrazac kroz koji se mogu dodati novi proizvodi u internetsku trgovinu.

Obrazac za stvaranje novog proizvoda prikazan je Slika 3.23, a dodatna polja za popunjavanje informacija o proizvodu prikazuju se nakon što korisnik izabere kategoriju proizvoda koji želi stvoriti.



Slika 3.23 Prikaz stranice za stvaranje novog prizvoda

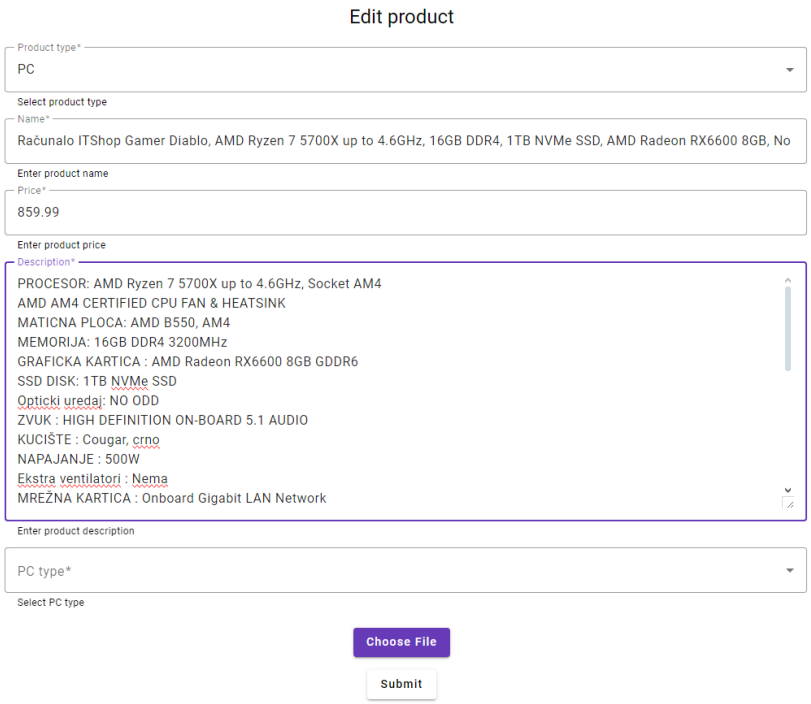
Primjer izgleda obrasca u kojem je korinik odabrao kategoriju proizvoda koji želi stovriti prikazuje Slika 3.24.



Slika 3.24 Prikaz obrasca za stvanje proizvoda s izabranom kategorijom proizvoda

Korisniku je omogućeno upisivanje podataka koji su specifični za odabranu kategoriju proizvoda te dodavanje slike klikom na gumb „*Choose file*“. Nakon uspješnog stvaranja proizvoda korisnika se preusmjerava na stranicu s prikazom detalja o proizvodu.

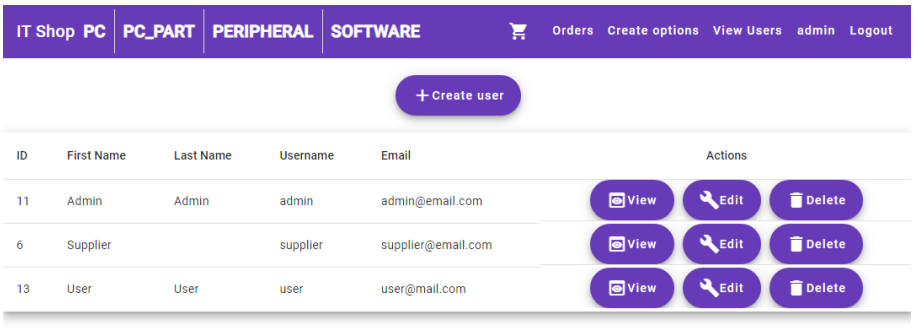
Funkcionalnost uređivanja podataka o proizvodu dostupna je samo administratorima sustava klikom na gumb „*EDIT*“ koji se nalazi na stranici s detaljima o proizvodu. Klikom na gumb za uređivanje, otvara se identičan obrazac kao kod stvaranja proizvoda, ali s već popunjenim podacima što prikazuje Slika 3.25. U slučaju da korisnik ne izabere novu sliku proizvoda ili potkategoriju ti podaci ostat će nepromijenjeni.



Slika 3.25 Prikaz obrasca za uređivanje detalja o proizvodu

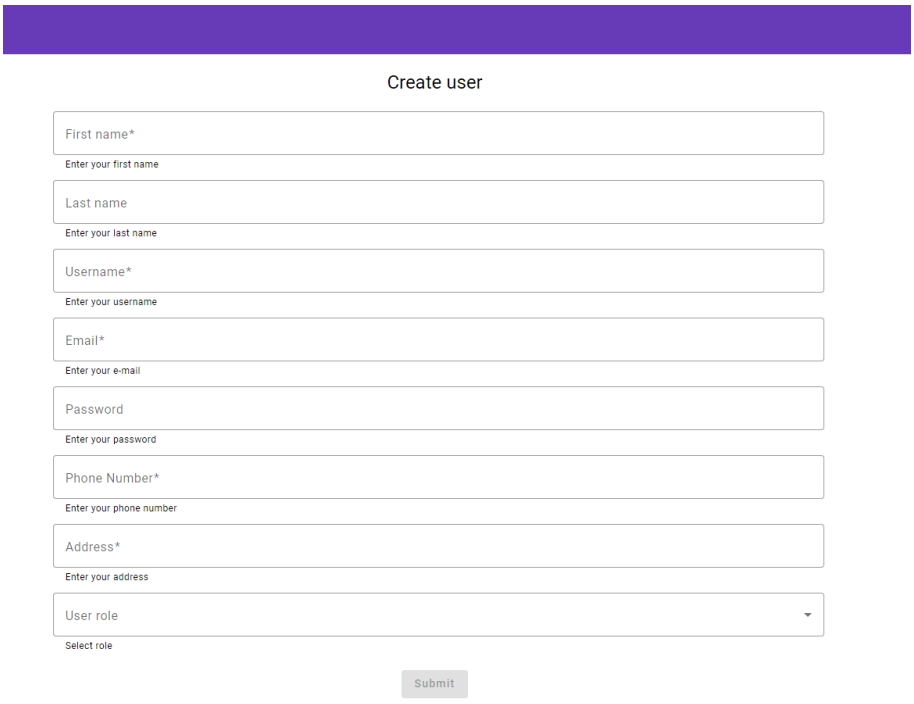
Administratorima sustva omogućeno je brisanje proizvoda klikom na gumb „*DELETE*“ koji se nalazi na stranici s detaljima o proizvodu.

Dodatna mogućnost, dostupna samo administratorima, je pregled svih korisnika klikom na gumb „*View users*“ na desnoj strani navigacijske trake. Klikom na taj gumb otvara se nova stranica s popisom svih korisničkih računa i dodatnim gumbima koji služe za pregled i uređivanje njihovih podataka, brisanje korisničkih računa te stvaranje novih korisnika prikazana Slika 3.26.



Slika 3.26 Prikaz stranice s popisom korisničih računa

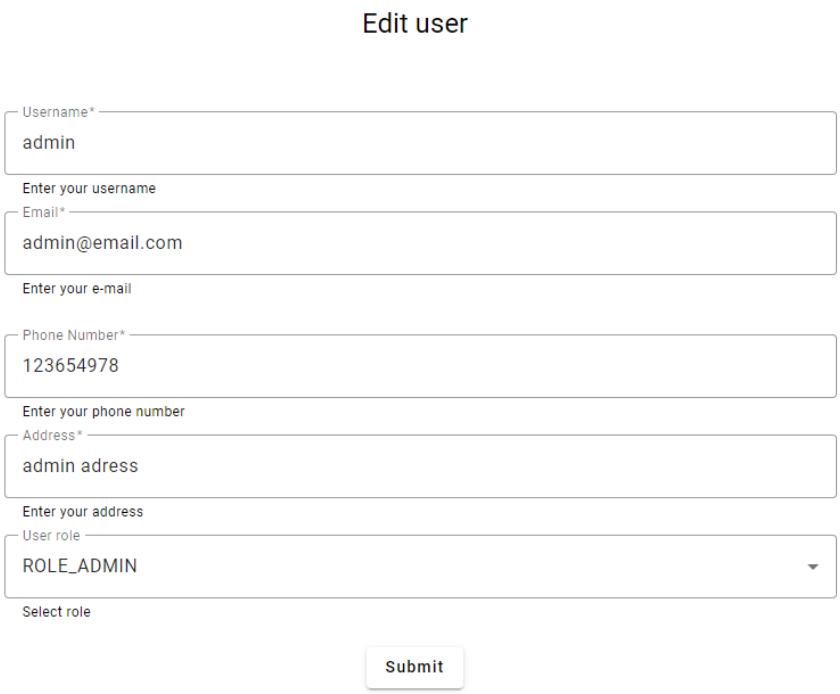
Nove korisničke račune moguće je stvoriti klikom na gumb „*Create user*“ nakon čega se otvara obrazac za stvaranje korisničkog računa kojeg prikazuje Slika 3.27.



Slika 3.27 Prikaz obrasca za stvaranje korisničkog računa

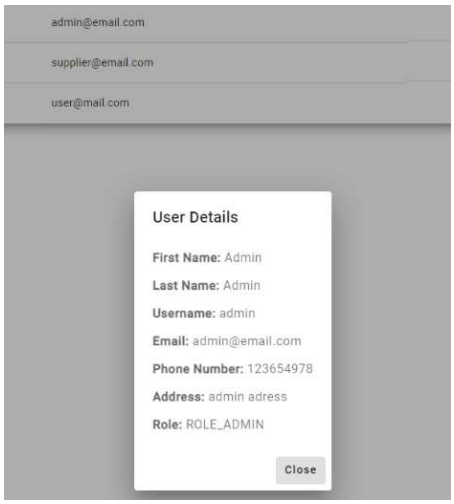
Stvaranjem korisničkog računa putem obrasca za registraciju, korisničke ovlasti se automatski postavljaju na razinu običnog korisnika.

Putem obrasca za stvaranje novog korisnika na već postojećem administratorskom računu moguće je stvoriti nove korisničke račune s ovlastima običnog korisnika, dobavljača ili administratora. Klikom na gumb „*Edit*“ otvara se obrazac s već popunjenim podacima za odabrani korisnički račun prikazan Slika 3.28.



Slika 3.28 Prikaz obrasca za uređivanje podataka korisničkog računa

Administratorima je omogućeno i pregledavanje korisničkih podataka klikom na gumb „*View*“ nakon čega se otvara prozor s podacima korisničkog računa kao što prikazuje Slika 3.29.



Slika 3.29 Prikaz prozora s podacima korisničkog računa

Zaključak

Razvoj modernih web aplikacija zahtijeva temeljito planiranje, poznavanje naprednih tehnologija, korištenje primjerenih alata i radnih okvira. Kroz ovaj rad, istražene su ključne tehnologije kao što su Java, Spring Framework, Angular, PostgreSQL te razvojna okruženja IntelliJ IDEA i WebStorm. Kombinacija ovih tehnologija pruža osnovu za izgradnju sigurnih i efikasnih aplikacija.

Java i njezini alati, poput Mavena i Lomboka, omogućavaju jednostavnije upravljanje projektima i smanjenje ponavljajućeg kôda. Spring Framework, sa svojim komponentama kao što su Spring Boot, Spring Security i Spring Data JPA, omogućava brzu izradu aplikacija visokih performansi i sigurnosti. Angular svojom podrškom za TypeScript, omogućava stvaranje dinamičkih korisničkih sučelja koja poboljšavaju korisničko iskustvo.

Kroz praktične primjere, prikazan je cjelokupan proces razvoja web aplikacije, uključujući inicijalizaciju projekta, izradu modela baze podataka, implementaciju *backend* i *frontend* dijela aplikacije te postavljanje sigurnosnih konfiguracija. Korištenjem PostgreSQL-a za upravljanje bazama podataka, osigurava se pouzdanost i integritet podataka.

Ovaj rad naglašava važnost korištenja modernih tehnologija i radnih okvira u razvoju web aplikacija, ističući njihove prednosti i mogućnosti. Zaključno, kombinacija ovih tehnologija omogućava izradu kvalitetnih i suvremenih softverskih rješenja koja zadovoljavaju potrebe korisnika i standarde industrije.

Literatura

[1] *The Complete History of Java Programming Language*, GeeksforGeeks, (2024, siječanj). Poveznica: https://www.geeksforgeeks.org/the-complete-history-of-java-programming-language/; pristupljeno: 3. lipnja 2024.

[2] *How JVM Works – JVM Architecture?*, GeeksforGeeks, (2023, rujan). Poveznica: https://www.geeksforgeeks.org/jvm-works-jvm-architecture/?ref=lbp, pristupljeno: 3. lipnja 2024.

[3] *Features of Java*, Javatpoint. Poveznica: https://www.javatpoint.com/features-of-java, pristupljeno: 3. lipnja 2024.

[4] *What is Maven?*, Apache Maven Project. Poveznica: https://maven.apache.org/what-is-maven.html, pristupljeno: 3. lipnja 2024.

[5] *Apache Maven Tutorial*, Baeldung, (2024, siječanj). Poveznica : https://www.baeldung.com/maven, pristupljeno: 4. lipnja 2024.

[6] *Introduction to Project Lombok*, Baeldung, (2024, ožujak). Poveznica: https://www.baeldung.com/intro-to-project-lombok, pristupljeno: 7. lipnja 2024.

[7] *Package lombok*, Project Lombok. Poveznica: https://projectlombok.org/api/lombok/package-summary, pristupljeno: 7. lipnja 2024.

[8] *Introduction to Liquibase,* Liquibase, Poveznica: https://docs.liquibase.com/concepts/introduction-to-liquibase.html, pristupljeno: 11. srpnja 2024..

[9] *Spring Tutorial*, Javatpoint. Poveznica: https://www.javatpoint.com/spring-tutorial, pristupljeno: 7. lipnja 2024.

[10] *Spring Framework 6.1.8*, spring by VMware Tanzu. Poveznica: https://spring.io/projects/spring-framework#overview, pristupljeno: 7. lipnja 2024.

[11] R. Awati, *Spring Framework (Spring)*, TechTarget, (2024, ožujak). Poveznica: https://www.techtarget.com/searchapparchitecture/definition/Spring-Framework, pristupljeno: 7.lipnja 2024.

[12] S. Swiniarski, *POJO*, codecademy, (2022, kolovoz). Poveznica: https://www.codecademy.com/resources/docs/java/pojo, pristupljeno: 7.lipnja 2024.

[13] *Dependency Injection(DI) Design Pattern*, GeeksforGeeks, (2024, siječanj). Poveznica: https://www.geeksforgeeks.org/dependency-injectiondi-design-pattern/, pristupljeno: 7. lipnja 2024.

[14] *JavaBean*, Javatpoint. Poveznica: https://www.javatpoint.com/java-bean, pristupljeno: 7. lipnja 2024.

[15] *XML – Based Injection in Spring*, GeeksforGeeks, (2024, veljača). Poveznica: https://www.geeksforgeeks.org/xml-based-injection-in-spring/, pristupljeno: 8. lipnja 2024.

[16] *Spring Boot Tutorial*, GeeksforGeeks, (2023, kolovoz). Poveznica: https://www.geeksforgeeks.org/spring-boot/, pristupljeno: 8. lipnja 2024.

[17] *Introduction to Spring Boot*, GeeksforGeeks, (2024, svibanj). Poveznica: https://www.geeksforgeeks.org/introduction-to-spring-boot/, pristupljeno: 9. lipnja 2024.

[18] devs5003, *Spring Boot MVC REST Annotations With Examples*, JavaTechOnline, (2023, srpanj). Poveznica: https://javatechonline.com/spring-boot-mvc-rest-annotations-with-examples/, pristupljeno: 9. lipnja 2024.

[19] *Java – JPA vs Hibernate*, GeeksforGeeks, (2022, rujan). Poveznica: https://www.geeksforgeeks.org/java-jpa-vs-hibernate/, pristupljeno: 9. lipnja 2024.

[20] E. Paraschiv, *A Guide to JPA with Spring*, Baeldung, (2023, listopad). Poveznica: https://www.baeldung.com/the-persistence-layer-with-spring-and-jpa, pristupljeno: 10. lipnja 2024.

[21] *Spring Security 6.3.0*, spring by VMware Tanzu. Poveznica: https://spring.io/projects/spring-security, pristupljeno: 10. lipnja 2024.

[22] MDN, *Types of attacks.* Poveznica: https://developer.mozilla.org/en-US/docs/Web/Security/Types_of_attacks, pristupljeno: 11. lipnja 2024.

[23] *Securing a Web Application*, spring by VMware Tanzu. Poveznica: https://spring.io/guides/gs/securing-web, Pristupljeno: 11. lipnja 2024.

[24] *Introduction to Spring Method Security*, Baeldung, (2024, siječanj). Poveznica: https://www.baeldung.com/spring-security-method-security, Pristupljeno: 11. lipnja 2024.

[25] *About*, PostgreSQL. Poveznica: https://www.postgresql.org/about/, Pristupljeno: 11. lipnja 2024.

[26] M. Drake, *An Introduction to Queries in PostgreSQL*, DigitalOcean, (2018, listopad). Poveznica: https://www.digitalocean.com/community/tutorials/introduction-to-queries-postgresql#creating-a-sample-database, pristupljeno: 11. lipnja 2024.

[27] *Angular Tutorial*, GeeksforGeeks, (2024, ožujak). Poveznica: https://www.geeksforgeeks.org/angular-tutorial/, pristupljeno: 11. lipnja 2024.

[28] *Build an App with Angular and Angular CLI*, GeeksforGeeks, (2024, ožujak). Poveznica: https://www.geeksforgeeks.org/build-an-app-with-angular-and-angular-cli/, pristupljeno: 11. lipnja 024.

[29] *What is an IDE (Integrated Development Environment)?*, AWS. Poveznica: https://aws.amazon.com/what-is/ide/, pristupljeno: 11. lipnja 2024.

[30] *IntelliJ IDEA overview*, JetBrains, (2024, svibanj). Poveznica: https://www.jetbrains.com/help/idea/discover-intellij-idea.html, pristupljeno: 11. lipnja 2024.

[31] *Features overview (IntelliJ IDEA)*, JetBrains. Poveznica: https://www.jetbrains.com/idea/features/, pristupljeno: 11. lipnja 2024.

[32] *JetBrains*, WebStorm Features, JetBrains. Poveznica: https://www.jetbrains.com/webstorm/features/, pristupljeno: 11. lipnja 2024.

[33] *Meet WebStorm,* JetBrains, (2024, svibanj). Poveznica: https://www.jetbrains.com/help/webstorm/meet-webstorm.html, pristupljeno: 11. lipnja 2024.

Sažetak

Internetska trgovina informatičkom opremom

Cilj ovog rada je pružiti detaljan pregled tehnologija koje omogućuju izradu moderne, učinkovite i sigurne internetske trgovine informatičkom opremom. U radu se istražuju ključne tehnologije za razvoj modernih web aplikacija, s naglaskom na programski jezik Java, Spring Framework, Angular, PostgreSQL te razvojna okruženja IntelliJ IDEA i WebStorm. Ističe se važnost korištenja alata poput Mavena i Lomboka, kao i komponenti Spring Frameworka za izradu kvalitetnih i efikasnih softverskih rješenja. Kroz praktične primjere prikazan je proces razvoja aplikacije internetske trgovine specijalizirane za prodaju informatičke opreme, od planiranja do implementacije.

Ključne riječi: razvoj web aplikacije, Java, Java Spring Framework, Angular, PostgreSQL, Maven, Lombok

Abstract

Online Store for Computer Equipment

This paper explores key technologies for developing modern web applications, focusing on Java, Spring Framework, Angular, PostgreSQL, and development environments IntelliJ IDEA and WebStorm. Tools such as Maven and Lombok, as well as components of the Spring Framework, are described in detail. Through practical examples, the development process of an application is presented, from planning to implementation. The paper highlights the importance of using these technologies to create high-quality and efficient software solutions.

Key words: web application development, Java, Java Spring Framework, Angular, PostgreSQL, Maven, Lombok