Web trgovina za prodaju nogometne opreme

Aleksić, Ivan

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

Stručni studij

WEB TRGOVINA ZA PRODAJU NOGOMETNE OPREME

Završni rad

Ivan Aleksić

Osijek, 2023.

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

Obrazac Z1S: Obrazac za imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Osijek, 19.09.2023.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Ime i prezime Pristupnika:	Ivan Aleksić
Studij, smjer:	Stručni prijediplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	AR 4710, 19.07.2019.
OIB Pristupnika:	22611209696
Mentor:	Marina Peko, dipl. ing.
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Robert Šojo, mag. ing. comp.
Član Povjerenstva 1:	Marina Peko, dipl. ing.
Član Povjerenstva 2:	mr. sc. Željko Štanfel
Naslov završnog rada:	Web trgovina za prodaju nogometne opreme
Znanstvena grana završnog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rada	Web trgovina za prodaju nogometne opreme. Detalji teme ce biti naknadno definirani sa studentom. Tema rezervirana za: Ivan Aleksić
Prijedlog ocjene pismenog dijela ispita (završnog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	19.09.2023.

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

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 02.10.2023.

Ime i prezime studenta:	Ivan Aleksić
Studij:	Stručni prijediplomski studij Računarstvo
Mat. br. studenta, godina upisa:	AR 4710, 19.07.2019.
Turnitin podudaranje [%]:	4

Ovom izjavom izjavljujem da je rad pod nazivom**: Web trgovina za prodaju nogometne opreme**

izrađen pod vodstvom mentora Marina Peko, dipl. ing.

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Tema ovog završnog rada je realizirati web aplikaciju, odnosno web trgovinu za prodaju nogometne opreme. S obzirom da me zanimaju programski jezici i okruženja iz područja web programiranja i s kojima se namjeravam baviti u budućnosti, odabrao sam ovaj zadatak.

Općenito, bavljenje sportom vrlo je važno za čovjekovo zdravlje i prema istraživanjima mnogobrojnih stručnjaka pozitivno utječe na postizanje discipline kod pojedinca, ali i na njegovu bolju fizičku spremu. Zbog nedostatka bavljenja aktivnošću sporta možemo uzrokovati negativne promjene i bolesti svojem ljudskom organizmu, kao što su u velikoj mjeri pretilost, dijabetes ili visoki krvni tlak te kolesterol. Budući da se tema ovog završnog rada bazira na sportu nogometu i kako je kroz povijest utemeljeno da je to „najvažnija sporedna stvar na svijetu“, za istu nam je potrebna i adekvatna sportska (nogometna) oprema. Značajan izbor sportske opreme omogućuje i lakše izvođenje određene sportske aktivnosti, ali isto tako i korištenjem ispravne sportske opreme za određeni sport smanjuju se i određene vrste ozljeda. Web trgovine, kao takve, pružaju lakši i pregledniji izbor potrebne sportske opreme, osim samog odlaska i izravne posjete trgovinama uživo.

Ovim završnim radom utemeljena je web trgovina za prodaju nogometne opreme. Web trgovina implementirana je tako da omogućuje vrlo jednostavan pronalazak potrebne nogometne opreme od kopački, tenisica, dresova, lopti ili ostale opreme potrebne za bavljenje ovom vrstom sportske aktivnosti. Korisnicima je omogućeno dodavanje ili brisanje proizvoda iz formirane košarice te nakon toga i narudžba tih krajnje odabranih proizvoda.

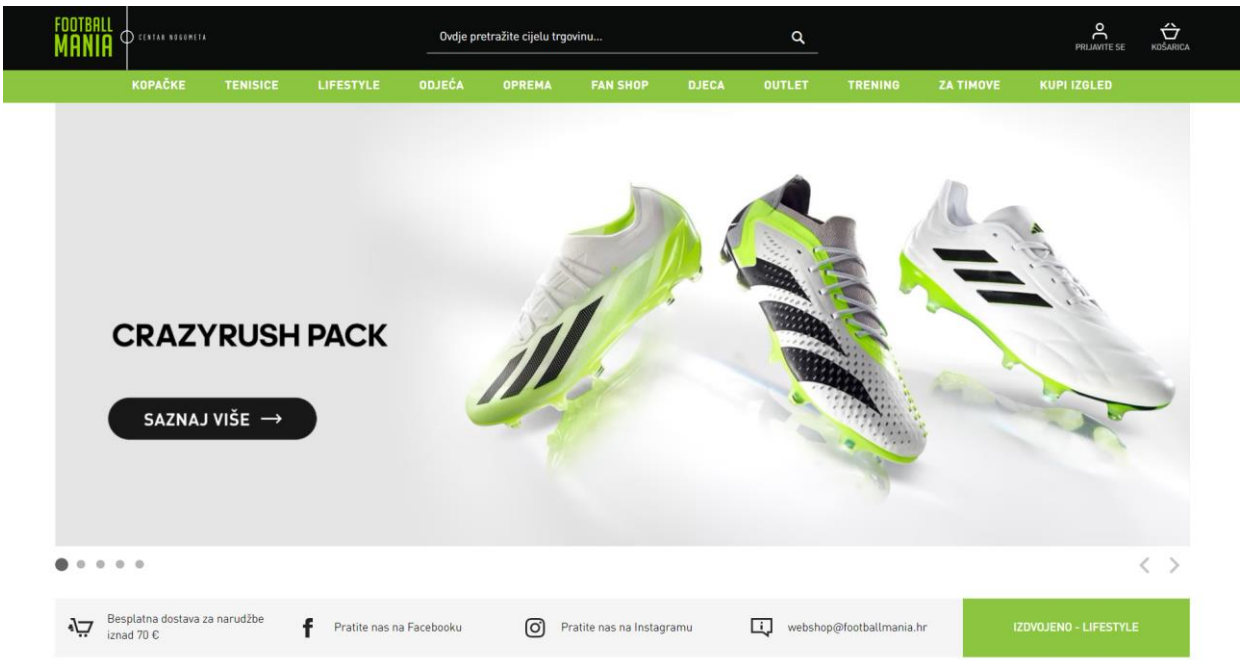
Izrada web trgovine za prodaju nogometne opreme opisana je kroz nekoliko poglavlja. Drugo poglavlje prikazuje neka od već dostupnih programskih rješenja koja su temeljna ideja i asocijacija za implementaciju ove web trgovine. Trećim poglavljem opisane su primijenjene tehnologije i programski jezici za razvoj programske podrške, odnosno tehnologije i alati koji su korišteni za *backend* i *frontend* dio aplikacije. Četvrto poglavlje obuhvaća temu implementacije programskog rješenja što opisuje svaki pojedini dio realizacije ovog završnog rada, počevši od implementacije baze podataka pa sve do opisa realizacije svakog pojedinog dijela aplikacije, uz priložene slike i odgovarajući interpretirani izvorni kod.

1.1. Zadatak završnog rada

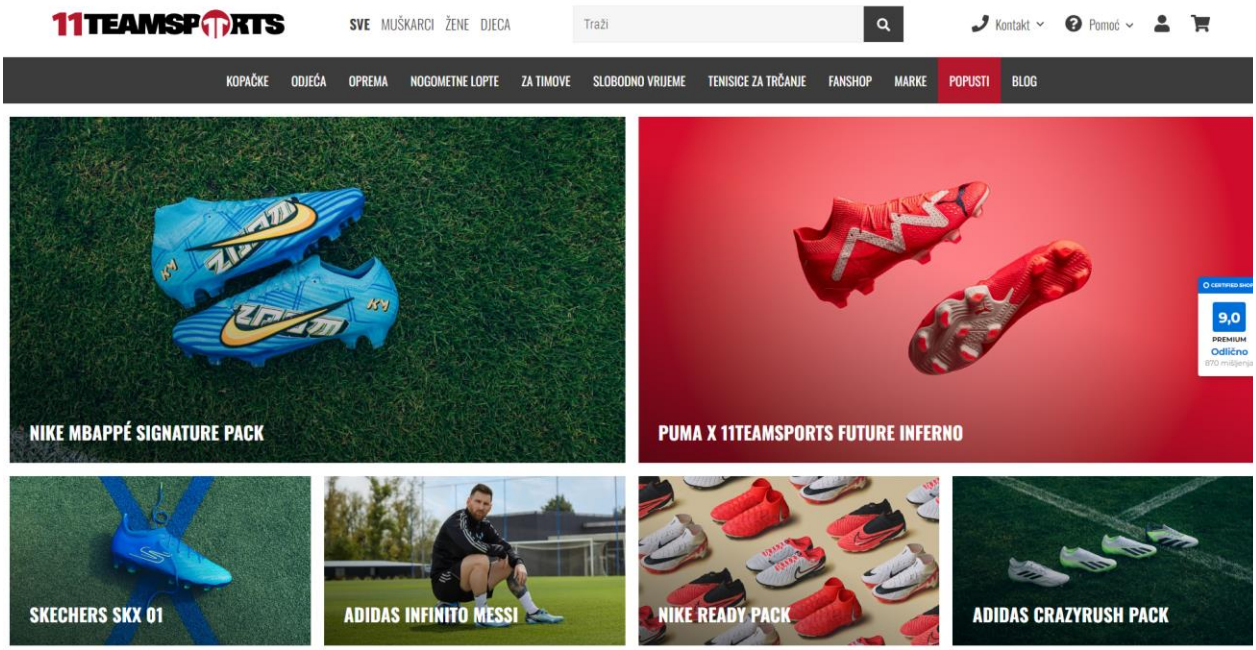
Zadatak završnog rada je izraditi web aplikaciju koja će služiti kao web trgovina za prodaju nogometne opreme, pomoću Spring Boot i Angular razvojnih programskih okvira (engl. *framework*) te ostalih pripadajućih web tehnologija i programskih jezika za razvoj ovakvog tipa web aplikacije. Potrebno je omogućiti upravljanje web trgovinom od strane registriranih i anonimnih (neregistriranih) korisnika u smislu jednostavnog odabira vrste proizvoda, pretraživanja, pregleda detalja proizvoda te dodavanja proizvoda u košaricu. Nakon dodavanja proizvoda u košaricu, korisniku je potrebno omogućiti narudžbu i plaćanje odabranih proizvoda.

2. DOSTUPNA PROGRAMSKA RJEŠENJA

Budući da je ideja o web trgovinama već davno zaživjela i internet bankarstvo, uz kartično plaćanje postala svakodnevica, sve je veća potreba za razvojem programskih rješenja koja će nuditi brzu i laku kupovinu potrebnih stvari i na taj način olakšati svakodnevno i masovno posjećivanje trgovačkih centara, te gubljenje vremena i strpljenja. Mnogobrojne su web aplikacije koje nude različite usluge prodaje, radilo se o sportu, modi, zabavi ili drugoj vrsti interesa, ipak najveća pažnja ostaje na prodaji odjeće i obuće i stoga je teško napraviti unikatno programsko rješenje koje već ne postoji. S ciljem da se napravi novo i originalno programsko rješenje ovaj završni ipak je morao imati nekakvu bazu informacija što bi trebala sadržavati jedna web trgovina za prodaju nogometne opreme. Web stranice čije programsko rješenje već postoji i na kojem je bazirana ideja za ovaj završni rad su web trgovina Football Mania (slika 2.1.) i 11teamsports.hr (slika 2.2.) [1, 2]. Obje aplikacije posjeduju forme za prijavu i registraciju korisnika, pretragu proizvoda unošenjem ključne riječi te odabir proizvoda prema različitim vrstama kategorija, što predstavlja ključni dio svake slične web aplikacije za prodaju ne samo nogometne opreme, već i drugih vrsta artikala složenih prema principu kupnje i prodaje stvari.



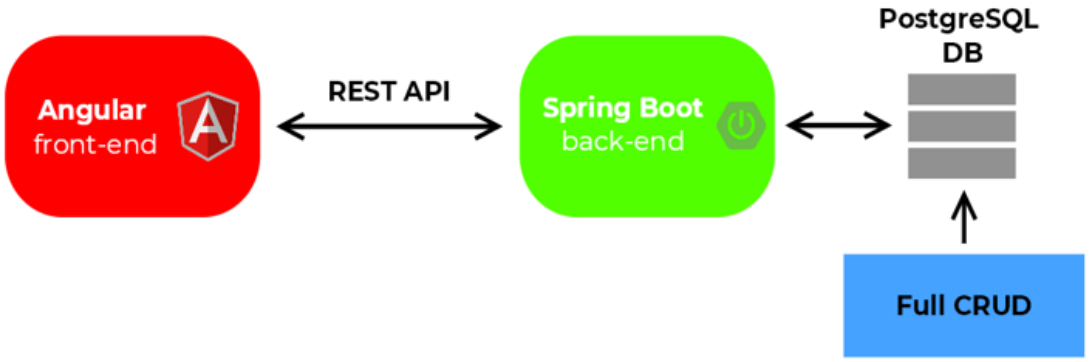
Sl. 2.1. Football Mania web shop – dostupno programsko rješenje.



Sl. 2.2. 11teamsports.hr web shop – dostupno programsko rješenje.

3. PRIMIJENJENE TEHNOLOGIJE I PROGRAMSKI JEZICI

Trećim poglavljem obuhvaća se tema tehnologija i programskih jezika koji su korišteni za izradu web trgovine za prodaju nogometne opreme, implementirane za interakciju s korisnicima pomoću četiri osnovne funkcionalnosti za rad s podacima u bazi podataka (engl. *Full CRUD*). Četiri osnovne funkcionalnosti opisanog *CRUD* sustava su operacija kreiranja novih podataka i zapisa (engl. ***C**reate*), operacija čitanja, pregledavanja ili dohvaćanja podataka iz baze (engl. ***R**ead*), operacija ažuriranja postojećih podataka (engl. ***U**pdate*) i operacija brisanja postojećih podataka i zapisa (engl. ***D**elete*). Opis tehnologija i programskih jezika važno je početi od razvojnih programskih okvira, a to su Spring Boot i Angular. Za razvoj aplikacije pomoću Spring Boot razvojnog okvira potrebno je poznavati programski jezik Java, ali isto tako i rad u IntelliJ IDEA integriranom razvojnom okruženju koje je odabrano kao optimalan IDE (engl. ***I**ntegrated **D**evelopment **E**nviroment*) za razvoj *backend* dijela aplikacije te rad s REST (engl. *Representational State Transfer*) sučeljem. REST arhitektura određuje način komunikacije između korisničke strane, odnosno klijenta i serverske strane aplikacije. Glavna značajka REST sučelja je omogućiti razmjenu podataka putem HTTP (engl. *Hypertext Transfer Protocol*) protokola. Kod Angular razvojnog okvira važno je poznavati osnovne jezike za razvoj web aplikacija, a za izradu ovog završnog rada korišteni su HTML, CSS, Bootstrap te TypeScript baziran na JavaScript programskom jeziku. Razvojno okruženje koje je korišteno za pisanje koda je Visual Studio Code. Za potrebe rada s bazom podataka korišten je PostgreSQL sustav, pokrenut pomoću pgAdmin sučelja. Slikom 3.1. prikazana je opisana shema funkcionalnosti.



Sl. 3.1. Shema funkcionalnosti aplikacije.

3.1. Programski jezik Java

Java programski jezik je objektno orijentirani programski jezik, kreiran 1995. godine unutar tvrtke Sun Microsystems, pod vodstvom Jamesa Goslinga i Patricka Naughtona. Sada je pod vlasništvom tvrtke Oracle i više od tri milijarde uređaja je pokrenuto pomoću programskog jezika Java.

Java je vrlo prihvatljiv programski jezik i koristi se za izradu mobilnih aplikacija, računalnih aplikacija, računalnih igara te komunikaciju s bazom podataka i izgradnju web aplikaciju zbog čega će i najviše služiti tijekom izrade ovog završnog rada. Zbog svoje jednostavnosti smatra se jednim od najpopularnijih programskih jezika na svijetu, prihvatljiv je na različitim vrstama platformi (Windows, Mac, Linux i ostali), besplatan je, siguran i brz te najvažnije od svega je da daje vrlo jasnu strukturu programima i omogućuje da se napisani kod ponovno upotrijebi i na taj način smanji troškove prilikom razvoja.

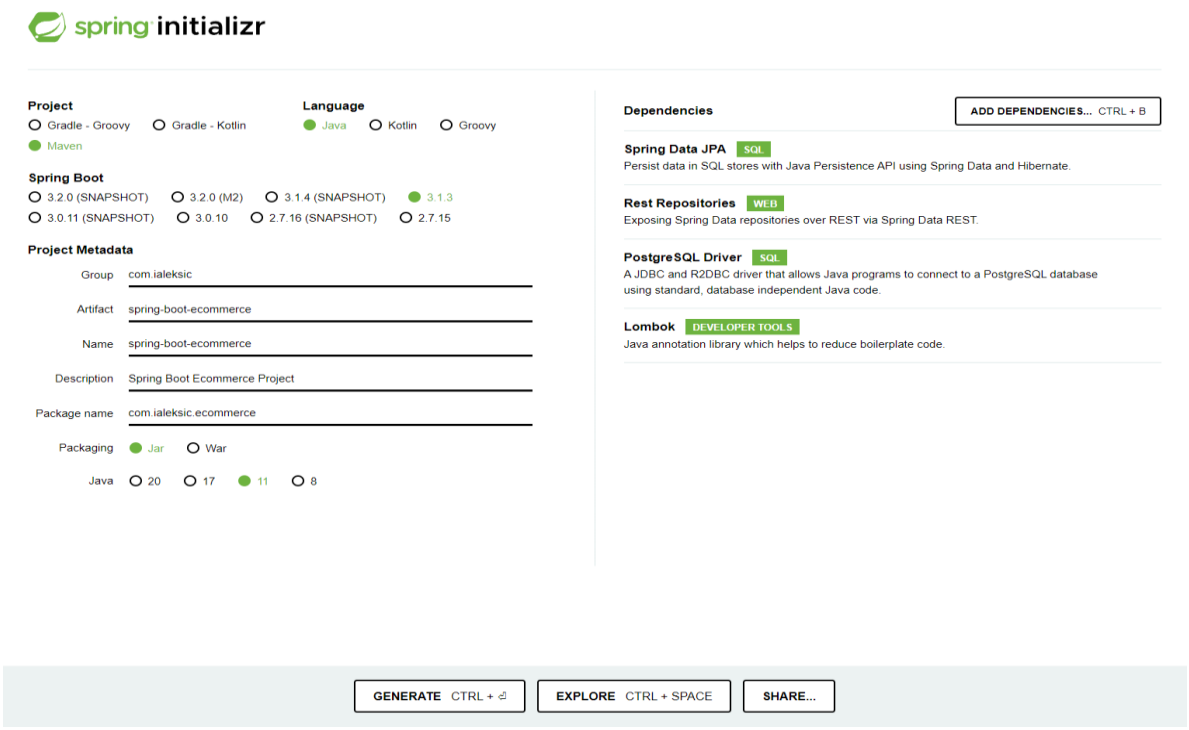
Baziran je na radu s klasama i dizajniran tako da ima što manje ovisnosti o implementaciji. Instanca jedne klase naziva se objekt, dok su osnovni članovi klase polja i metode. Polja označavaju varijable podataka te predstavljaju stanje objekta i klase, a metode su biblioteke tvrdnji, odluka ili izjava koje manipuliraju ponašanjem polja. Navedenim izjavama određuje se ponašanje klase, odnosno dodjeljuje se vrijednost poljima i ostalim varijablama, prati se i nadzire slijed izvođenja i drugo. Primjer koda u programskog jeziku Java prikazan je slikom 3.2. [3].

```
public class Main {
  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }
}
```

Sl. 3.2. Programski jezik Java – primjer koda.

3.2. Programski okvir Spring Boot

Spring Boot razvojni programski okvir predstavlja razvojno okruženje otvorenog koda koje je kreirano 2013. godine. Rod Johnson osmislio je rad razvojnog okvira kao poboljšanje na do tad već korišteni Java Enterprise Edition (JEE) standard za izradu većih i zahtjevnij ih web aplikacija. Spring Boot programski okvir naklonjen je dvjema karakteristikama, a to su inverzija kontrole te injektiranje objekata (engl. *dependency injection*). Inverzija kontrole govori kako će sam razvojni okvir kontrolirati izvršavanje programskog koda, dok injektiranje objekata ili svima poznatiji *dependency injection* predstavlja dio Spring razvojnoj okvira o kojem ovisi rad koda. Sam rad na Springovom razvojnom programskom okviru preuzela je tvrtka Pivotal Software te na taj način pridonijela poboljšanju razvojnog okruženja na način da isporučuje module, odnosno nezavisne projekte zasnovane na Springu (Spring Initializr). Može se zaključiti kako je Spring Boot osmišljen s ciljem da se uz što manje komplikacija i podešavanja, projekt inicijalizira i pripremi za daljnji razvoj. Glavne prednosti i karakteristike Spring Boot programskog okvira su takozvane *Out of the box functionalities* tj. već unaprijed „pripremljene“ funkcionalnosti i nema generiranja klasa i koda, nego se koriste već unaprijed definirane biblioteke te sigurnost. Spring Boot razvojni programski okvir osigurava cjeloviti i detaljan model programiranja i konfiguracije za moderne poslovne aplikacije temeljene na Java programskom jeziku (na bilo kojoj vrsti programske platforme za implementaciju) [4].



Sl. 3.3. Inicijalizacija projekta pomoću online servisa Spring Inizializr.

Inicijalizacija Spring Boot projekta obavlja se pomoću online servisa Spring Initializr. Prema slici 3.3. prikazuju se temeljne postavke projekta pomoću navedenog servisa i na kraju generiranje projekta. Na desnoj strani ekrana nalazi se odabir ovisnosti (engl. *dependencies*) koje će definirati namjenu te sve mogućnosti projekta na kojem radimo, dok se s lijeve strane nalaze općenite postavke za implementaciju projekta na kojem radimo. Na početku biramo hoće li naš projekt biti zasnovan na Mavenu ili Gradleu koji predstavljaju razvojne alate za izgradnju, razvijanje i implementaciju projekta te upravljanje projektima općenito. Nakon toga biramo programski jezik na kojem će biti zasnovan naš projekt. Zatim slijedi odabir verzije Spring Boota te primarni podaci koje je potrebno zadati. Primarni podaci koje moramo zadati su nazivi grupa kojima će pripadati programski sadržaj (engl. *group*), naziv artefakta koji stvaraju strukturu paketa projekta kojeg želimo inicijalizirati (engl. *artefact*), ime projekta (engl. *name*), kratki opis projekta koji nije primaran da se ispuni (engl. *description*), ime paketa (engl. *package name*), paketiranje (engl. *packaging*) u Jar ili War način arhiviranja našeg projekta i sam odabir verzije programskog jezika kojeg smo prethodno postavili na računalu. Postavke od najvažnijeg značaja odabrane su pod stavkom ovisnosti (engl. *dependency*), a to su naprimjer ovisnosti za realizaciju projektnog zadatka, ovisnosti za realizaciju grafičkog sučelja, ovisnosti za bazu podataka, ovisnosti za smanjivanje ponavljajućeg koda i druge ovisnosti [5].

Za potrebe izrade ovog projekta odabrane su sljedeće ovisnosti (Slika 2.2.):

- Spring Data JPA – ovisnost koja daje mogućnost pretvaranja programskog koda Java u upite u SQL-u
- Rest repositories – ovisnost koja omogućuje prikaz Spring Dana repozitorija putem Rest arhitekture
- PostgreSQL Driver – ovisnost koja omogućuje komunikaciju između Spring Boota i PostgreSQL baze podataka
- Lombok – ovisnost koja pridonosi da kod bude pregledniji te uklanja tzv. *boilerplate* kod

Prikaz svih ovisnosti (engl. *dependencies*) moguće je vidjeti ili mijenjati unutar pom.xml dokumenta (engl. *Project Object Model*) koji sadrži sve informacije o projektu koje smo definirali pomoću Spring Initializr *online* servisa te isto tako sve potrebne konfiguracijske detalje koje Maven sadrži za pokretanje projekta.

3.3. Programski okvir Angular

Angular predstavlja razvojni programski okvir i platformu za stvaranje suvremenih, učinkovitih i inovativnih *single-page* aplikacija. Angular je razvojna programska platforma stvorena pomoću TypeScripta. Jedna od značajki koja opisuje Angular je da je to okvir baziran na komponentama za stvaranje skalabilnih aplikacija za web. Također, Angular uključuje zbirku biblioteka koje obuhvaćaju širok raspon elemenata kao što su kontroliranje obrascima, komunikacija klijenta i poslužitelja i ostale. Angular sadrži paket alata za razvoj, implementaciju, testiranje i ažurnost kod pisanja koda [6].

Arhitektura Angular razvojnog programskog okvira je takva da se sastoji od sljedećih ključnih pojmova:

- Komponenta (engl. *component*) – predstavlja glavni dio u implementaciji Angular aplikacije. Sastoji se od HTML predloška za prikaz korisničkog sučelja stranice, TypeScript klase koja definira ponašanje i logiku aplikacije i CSS selektora koji definira kako se određena komponenta koristi u predlošku [7].

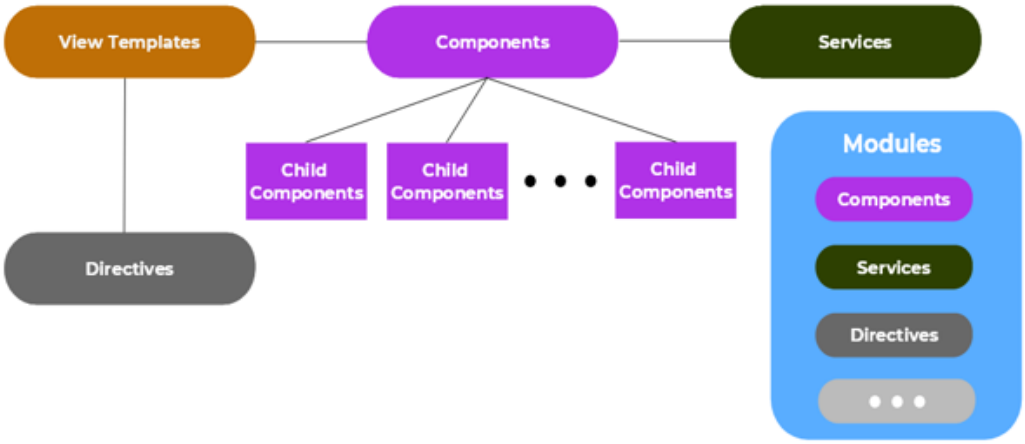
  Generiranje Angular komponente u naredbenom retku (engl. *terminal*) prikazano je slikom 3.4.

```
ng generate component <component-name>
```

Sl. 3.4. Naredba za generiranje Angular komponente.

- Prikaz korisničkog predloška (engl. *view template*) – označava nacrt za prikaz korisničkog sučelja za komponentu, napisan je kao statički HTML s dinamičkim elementima [8].
- Direktive (engl. *directives*) – klase koje dodaju određena pravila te dodatno ponašanje na elemente u Angular aplikaciji i koriste se za različite petlje i uvjete [9].
- Usluga/servis (engl. *service*) – pomoćna klasa s jasno definiranom svrhom za pružanje funkcionalnosti za određeni element, dohvaćanje podataka s poslužitelja i provjeru ispravnosti validacije [10].
- Modul (engl. *module*) – predstavlja biblioteku svih povezanih komponenti, direktiva, usluga i ostalih funkcionalnosti Angular aplikacije [11].

Slikom 3.5. prikazana je opisana shema arhitekture Angular programskog okvira.



Sl. 3.5. Arhitektura Angular programskog okvira.

Kreiranje Angular projekta izvodi se pomoću alata naredbenog retka za generiranje projekta. Unutar Angular CLI-a (engl. *Angular Command Line Interface*) generiraju se početne datoteke koje će pomoći u daljnjoj izgradnji, odnosno pokretanju projekta ili instalaciji Angular klijenta.

Slika 3.6. prikazuje naredbu za instalaciju Angular CLI-a.

```
npm install -g @angular/cli
```

Sl. 3.6. Naredba za generiranje Angular CLI-a.

Kreiranje, odnosno generiranje novog Angular projekta izvršava se na sljedeći način (Slika 3.7.):

```
ng new <ime-projekta>
```

Sl. 3.7. Naredba za generiranje Angular projekta.

Pokretanje projekta/aplikacije vrši se iz direktorija u kojem se nalazi aplikacija, na sljedeći način (Slika 3.8.):

```
ng serve
```

Sl. 3.8. Naredba za pokretanje Angular projekta.

Aplikacija se prema unaprijed zadanim postavkama otvara na port-u 4200 – http://localhost:4200.

Slika 3.9. prikazuje naredbu za jednostavnu promjenu porta na kojem se izvodi aplikacija.

```
ng serve --port <novi-broj-porta>
```

Sl. 3.9. Naredba za promjenu port-a na kojem se izvodi aplikacija.

Projektne datoteke (engl. *project files*) koje nastaju generiranjem novog Angular projekta su:

- angular.json – datoteka za konfiguraciju postavki projekta
- node_modules – lokalni repozitorij svih node modula
- package.json – meta podaci projekta, predstavljaju listu node ovisnosti
- src – glavni direktorij projekta
- app – sve komponente projekta (korisnički predlošci, klase, …)
- assets – datoteka u koju su pohranjene slike, videozapisi itd.
- enviroments – različita projektna okruženja (dev, test, prod, …)
- index.html – glavna stranica za pokretanje projkta
- polyfills.ts – podrška za različite verzije web preglednika
- test.ts – unit testovi
- tsconfig.json – TypeScript konfiguracija kompajlera

3.4. Programski jezik TypeScript

TypeScript je programski jezik razvijen od strane Microsoft kompanije 2012. godine. Besplatan je i otvorenog koda (engl. *open-source*) programski jezik, što znači da predstavlja softver otvorenog koda ili softver čiji je izvorni kod dostupan svim licenciranim korisnicima s pravima mijenjanja, prepravljanja ili unapređivanja sadržaja tog koda. Budući da web pretraživači ne razumiju izvorni TypeScript jezik, kompajliranje koda se izvodi na način da se TypeScript kod pretvara u JavaScript kod. Takav postupak se naziva *Transiling* (*Transform* + *Compile*).

Osnovni tipovi podataka TypeScript programskog jezika su:

- boolean – tip podatka koji predstavlja istinitu ili lažnu vrijednost (*true* ili *false*)
- number – tip podatka koji označava brojeve, cijele i decimalne vrijednosti (*integer* i *floating*)
- string – tip podatka koji predstavlja tekstualne podatke koji mogu biti implementirani u obliku s jednostrukim ili dvostrukim navodnicima
- any – predstavlja tip varijable koji podržava bilo koji tip podatka.

Ključna riječ kod definiranja TypeScript varijabli je *let*, koja zamjenjuje JavaScript ključnu riječ *var*. Tipove varijabli se određuje na način da se nakon imena varijable dodaje dvotočka i zatim nakon nje tip i inicijalna vrijednost. Na taj način smo osigurali da varijabla ne može biti bilo

kojeg tipa kao u JavaScript programskom jeziku. TypeScript funkcijama, također je moguće naznačiti koji će tip podatka vratiti. Ispis TypeScript koda vrši se ključnom riječi *console.log* [12].

Slika 3.10. prikazuje primjer TypeScript koda:

```
let message: string = 'Hello, World!';
console.log(message);
```

Sl. 3.10. Programski jezik TypeScript – primjer koda.

3.5. Prezentacijski opisni jezik HTML

Prema [13], HTML (engl. *HyperText Markup Language*) je prezentacijski i opisni jezik za rad i izradu web stranica, odnosno strukturu sadržaja na stranici. HTML se sastoji od niza elemenata koji se koriste prilaganje, odnosno omotavanje različitih dijelova sadržaja kako bi na određeni način djelovali i izgledali. HTML obrađuje informacije na način da ih organizira i prikaže u obliku teksta koji stvara hiperveze. Hiperveze daju mogućnost prelaska iz jedne na drugu web stranicu, odnosno prelazak iz jednog HTML dokumenta na drugi. Organiziranje te obrada i prikaz informacija i podataka pomoću HTML-a izvodi se oznakama i atributima (engl. *tags and attributes*). Oznake služe za određivanje svrhe svakog pojedinog podatka i što on predstavlja, lako ih je prepoznati unutar šiljatih zagrada gdje su naznačene. Atributi opisuju određene značajke HTML elemenata i uvijek se nalaze u početnoj zagradi za pojedini element, sastoje se od naziva svojstva koje želimo definirati za pojedinu oznaku i vrijednosti koju će poprimiti to definirano svojstvo.

Jedan HTML element sastoji se od oznaka za početak i kraj elementa te sadržaja koje taj element nosi. Oznaka za početak elementa se sastoji od naziva elementa, koji se nalazi unutar otvorene i zatvorene uglate zagrade, što navodi gdje element počinje, odnosno počinje djelovati unutar HTML dokumenta. Oznaku za kraj elementa može se opisati isto kao i oznaku za početak elementa, jedina razlika je u tome što uključuje kosu crtu ispred naziva elementa. U HTML dokumentu navodi gdje je određenom elementu kraj, tj. gdje završava. Uključivanjem atributa unutar oznake za početak elementa daju se dodatne informacije o elementu za koje ne želimo da se pojavljuju u stvarnom sadržaju, odnosno da se pozivamo na njega. Slika 3.11. prikazuje HTML element.

```
<p> Hello, World! </p>
```

Sl. 3.11. HTML element.

Prema slici 3.12. prikazan je primjer HTML dokumenta koji se sastoji od elemenata:

- **<!DOCTYPE html>** - obavezna oznaka unutar HTML dokumenta koja djeluje kao skup pravila koje HTML stranica slijedi, odnosno služi kako bi osigurali da se dokument ispravno ponaša.
- **<html></html>** - element koji obavija cijeli sadržaj na stranici i poznat je kao korijenski element. Također, uključuje i atribut *lang* čime se postavlja primarni jezik dokumenta.
- **<head></head>** - element koji djeluje kao spremnik za sve što se može uključiti u HTML stranicu, a nije sadržaj koji je vidljiv posjetiteljima stranice. To su elementi poput ključnih riječi, opisa stranice ili stiliziranje sadržaja i drugo.
- **<meta charset="utf-8">** - element koji postavlja skup znakova koje HTML dokument treba koristiti prema UTF-8 standardu koji uključuje većinu znakova iz velike većine pisanih jezika.
- **<meta name="viewport" content="width=device-width">** - element koji osigurava prikaz stranice prema širini okvira za prikaz
- **<title></title>** - element koji postavlja naslov stranice, a to je naslov koji se pojavljuje na kartici preglednika u kojem je učitana stranica.
- **<body></body>** - element koji označava sadržaj koji će biti prikazan korisnicima kada posjete web stranicu, bilo da se radi o tekstu, slikama, videozapisima, igrama, zvučnim zapisima, itd.

```
<!DOCTYPE html>
<html lang="en-US">
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width" />
<title>Page Title</title>
</head>
<body>
<p> Hello, World! </p>
</body>
</html>
```

Sl. 3.12. HTML dokument.

3.6. Stilski jezik CSS

Cascading Style Sheets ili skraćeno CSS je stilski jezik koji je označava kod koji stilizira web sadržaj. Koristi se za selektivno stiliziranje HTML elemenata. Najvažnija značajka CSS stilskog jezika je u tome da štedi vrijeme, u odnosu kad bi sav sadržaj uređivali uporabom HTML atributa i oznaka, budući da CSS ima mogućnost upravljanja izgledom na više dokumenata istovremeno. Primjenom CSS-a postiže se raspored elemenata na web stranici (određivanje margina, način prikaza i sl.), izgled teksta (font, veličina slova, boja, razmaci, podvlake i sl.), postavljanje animacija te ostalih zanimljivih i oku privlačnih sadržaja [14].

3.7. Bootstrap programski okvir

Bootstrap je besplatni programski okvir koji služi za razvoj *frontend* dijela web aplikacije. Bootstrap programski okvir temelji se na HTML, CSS i JavaScript programskim jezicima. Temeljna svrha ovog programskog okvira je u tome da je utemeljen kako bi se koristio za razvoj responzivnih sadržaja na web stranici s ciljem boljeg prikaza stranice na mobilnim uređajima. Responzivni dizajn omogućuje web aplikacijama da se automatski prilagode vrsti uređaja kojeg korisnik koristi prilikom posjete web stranice i omogući mu prikaz u skladu s tim uređajem. Također, Bootstrap je temeljen na takozvanom grid sustavu koji se sastoji od redaka i stupaca što olakšava upravljanje sadržajem i elementima na web stranici [15].

3.8. IntelliJ IDEA razvojno okruženje

IntelliJ IDEA je integrirano razvojno okruženje namijenjeno za razvoj aplikacija baziranih u većoj mjeri prema Java programskog jeziku te pruža svu potrebnu programsku i korisničku podršku za rad s web aplikacijama i Spring Boot programskim okvirom, ali i ostalim srodnim razvojnim programskim okvirima. Osim za rad s Javom, ovaj inteligentni sustav pruža rad i s drugim JVM programskim jezicima kao što su Kotlin, Scale ili Groovy. IntelliJ IDEA razvojno okruženje uvelike pomaže pri razvoju web aplikacija budući da posjeduje dinamične integrirane alate te pruža podršku JavaScript i bliskim tehnologijama. IntelliJ IDEA vrlo je prihvatljivo razvojno okruženje, budući da se sastoji od jako dobrih produktivnih svojstava pri dovršavanju koda. Takozvani algoritam predviđanja na vrlo precizan način pretpostavlja što programer želi napisati unutar linije koda i automatski dovršiti umjesto njega. Za lakšu organizaciju prilikom rada na projektima, IntelliJ IDEA daje mnoge alate, kao što su prevoditelji, Docker podrška, FTP ili preglednik bajt koda i ostali [16].

3.9. Visual Studio Code razvojno okruženje

Prema [17], Visual Studio Code (VS Code) predstavlja razvojno okruženje za oblikovanje otvorenog izvornog koda, primarna svrha mu je ispravljanje i popravak grešaka prilikom kodiranja u najčešćem slučaju web aplikacija. VS Code razvijen je od strane Microsoft kompanije, a podržan je na skoro svim najpoznatijim operativnim sustavima, poput macOS-a, Linuxa i Windowsa. Prednost VS Code-a je u tome što ugradnjom više FTP ekstenzija, korisnici mogu sinkronizirati kod između poslužitelja i uređivača, bez dodatne potrebe za novim softverom. Ključne značajke VS Code-a su tzv. Kontrola izvora (engl. *Source Control*) koja predstavlja posebnu karticu na traci izbornika te korisnicima omogućuje pristup postavkama verzija kontrole, pristup i podrška velikom broju programskih jezika kao što su Python, C++, JavaScript, TypeScript, Node.js itd., IntelliSense alat za dovršavanje koda (engl. *IntelliSense code completion*) koji razvojnom programeru omogućuju pomoć pri pisanju sintakse, parametara, funkcija ili varijabli. VS Code vrlo je prihvatljivo razvojno okruženje budući da pruža brzo kretanje između više alata kako bi se napravile potrebne izmjene u kodu. Također, veliko olakšanje su i prečaci na tastaturi za uobičajene kombinacije tipki i operacije koje se često ponavljaju. Osim već naznačenog rada VS Code-a s web stranicama, VS Code je mnogo zastupljen i za rad na *desktop* aplikacijama. Vrlo je popularan među *frontend* developerima, zbog čega je izabran za rad na završnom radu i radu s Angular razvojnim programskim okvirom.

3.10. PostgreSQL baza podataka

PostgreSQL je objektno-relacijski sustav za baze podataka otvorenog koda. Nastao je u okvirima projekta POSTGRES u Berkeleyu na Kalifornijskom sveučilištu. PostgreSQL predstavlja proširenje na standardni SQL jezik u smislu mnogobrojnih kombinacija sa specifikacijama koje pohranjuju i najzahtjevnija opterećivanja podacima. PostgreSQL sustav stoji iza softvera koji pružaju efikasna i napredna rješenja, zbog čega zaslužuje mjesto među najboljim sustavima za baze podataka. Jamči pouzdanost, dobro složenu arhitekturu i cjelovitost te točnost napisanih podataka. Cilj PostgreSQL sustava je da razvojnim programerima pruži pomoć prilikom izgradnje aplikacija, zaštitu podataka administratorima sustava i upravljanje različitim veličinama podataka. Velika prednost PostgreSQL sustava je što je besplatan i otvorenog koda, ali isto tako i proširiv. Proširiv je u smislu da se mogu definirati osobni tipovi podataka, prilagoditi funkcije ili napisati kod drugih programskih jezika bez ponovnog definiranja baze podataka [18].

4. IMPLEMENTACIJA PROGRAMSKOG RJEŠENJA

Web trgovina za prodaju nogometne opreme predstavlja aplikaciju koja omogućuje naručivanje nogometne opreme. Web aplikacija omogućuje prijavu korisnika, kojom postaje član (engl. *Member*) i na taj način ostvaruje neke od pogodnosti, odnosno popuste na određene proizvode na mjesečnoj bazi, koje nudi Web trgovina. Također, prijavljeni korisnik ima uvid u sve svoje provedene narudžbe. Proizvode koje nudi Web trgovina moguće je naručiti kao anonimni korisnik, odnosno ne-ulogirani korisnik, ali i kao prijavljeni korisnik.

Prikaz početne stranice web aplikacije (prema slici 4.1.) sadrži navigacijsku traku u kojoj se nalazi logo web trgovine (klikom gumba našeg pokazivača na logo vraćamo se svaki put na početnu stranicu), područje za pretraživanje proizvoda prema njihovom definiranom nazivu spremljenog u bazu podataka, gumb za prijavu korisnika te alat za prikaz trenutnog stanja količine dodanih proizvoda i ukupna cijena svih dodanih proizvoda u košaricu.

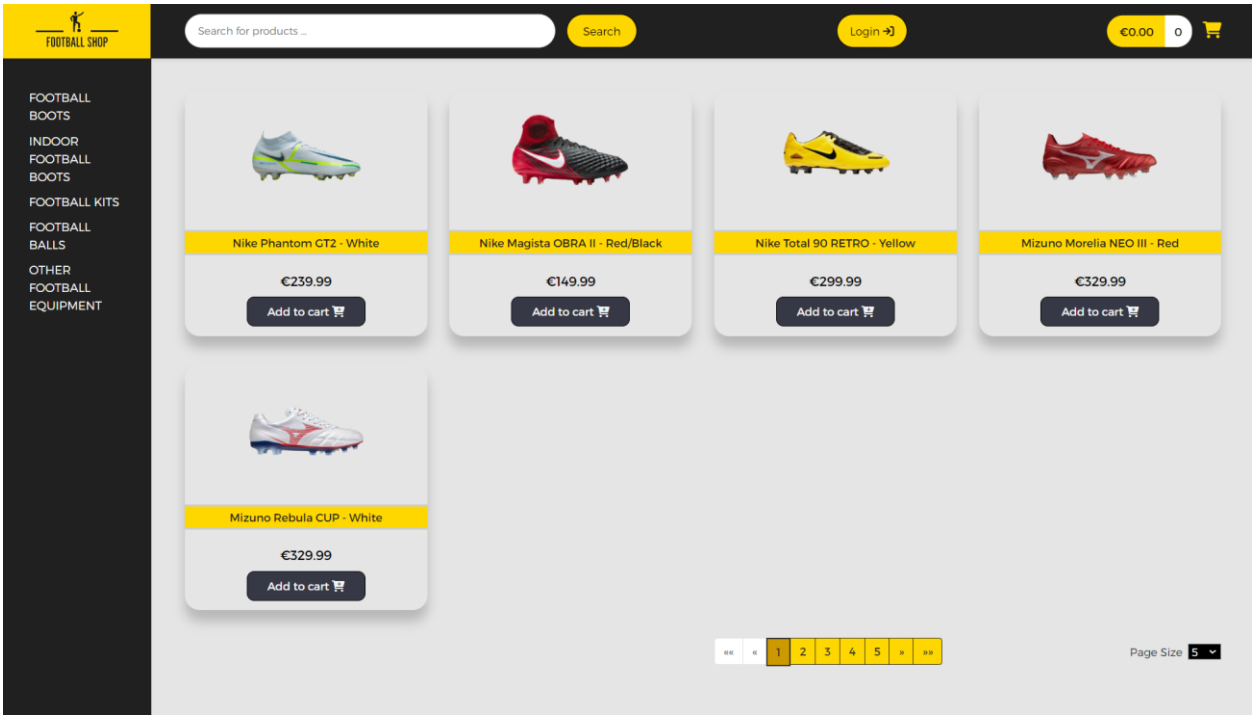
Klikom gumba našeg pokazivača na ikonu košarice, otvara se novo sučelje u vidu tablice koja nudi prikaz svih trenutno dodatnih proizvoda unutar web košarice s općim informacijama o proizvodu (naziv, cijena i slika), mogućnost povećavanja, odnosno smanjivanja količine za pojedini proizvod te brisanje samog proizvoda iz košarice. Na dnu tablice nalaze se informacije o ukupnoj količini dodanih proizvoda unutar košarice, ukupnoj cijeni i informacija o besplatnoj dostavi. Ispod navedenih stavki o važnim informacijama, nalazi se gumb koji vodi na sučelje za popunjavanje forme o korisničkim podacima, poput osnovnih podataka o kupcu, adrese za naplatu i dostavu naručenih proizvoda. Nakon čega je opet ispisan ukupni pregled svih dodatnih proizvoda te ukupna cijena, poslije čega je moguće odabirom gumba *Purchase* obaviti narudžbu, nakon kojeg dobijemo ispis poruke kako je kupnja uspješno obavljena.

Za prijavljenog korisnika unutar navigacijske trake pojavljuje se simbolična poruka kojom se daje do znanja da je korisnik uspješno prijavljen unutar web aplikacije. Također, prijavom korisnika, pojavljuju se i gumbi *Member* te *Orders* kojima prijavljeni korisnik može pristupiti popustima na određenim proizvodima, odnosno pristupiti svim narudžbama koje je do sada izvršio.

Na lijevom dijelu prikaza početne stranice nalazi se izbornik sa svim kategorijama koje nudi web trgovina, a predstavlja glavnu navigaciju do traženja željenog proizvoda. U ovisnosti o odabiru kategorije otvaraju se nove stranice koje nude popis svih proizvoda iz baze podataka za svaku specifičnu kategoriju.

Glavni dio početne stranice je sučelje koje nudi popis svih proizvoda za određenu kategoriju, gdje se nalaze slike proizvoda, naziv i cijena te gumb za dodavanje proizvoda u košaricu. Klikom gumba na sliku ili naziv pojedinog proizvoda, otvara se nova stranica koja sadrži povećanu i jasniju sliku odabranog proizvoda, gumb za dodavanje u košaricu, opis proizvoda te gumb za vraćanje na stranicu s popisom svih proizvoda iz birane kategorije.

Na dnu stranice nalazi se paginacija, odnosno dio koji obilježava na kojoj se stranici trenutno nalazimo te alat koji omogućuje odabir količine proizvoda koji će se ispisati odjednom na stranici za određenu kategoriju proizvoda.



Sl. 4.1. Prikaz izgleda početne stranice projekta.

4.1. Implementacija baze podataka

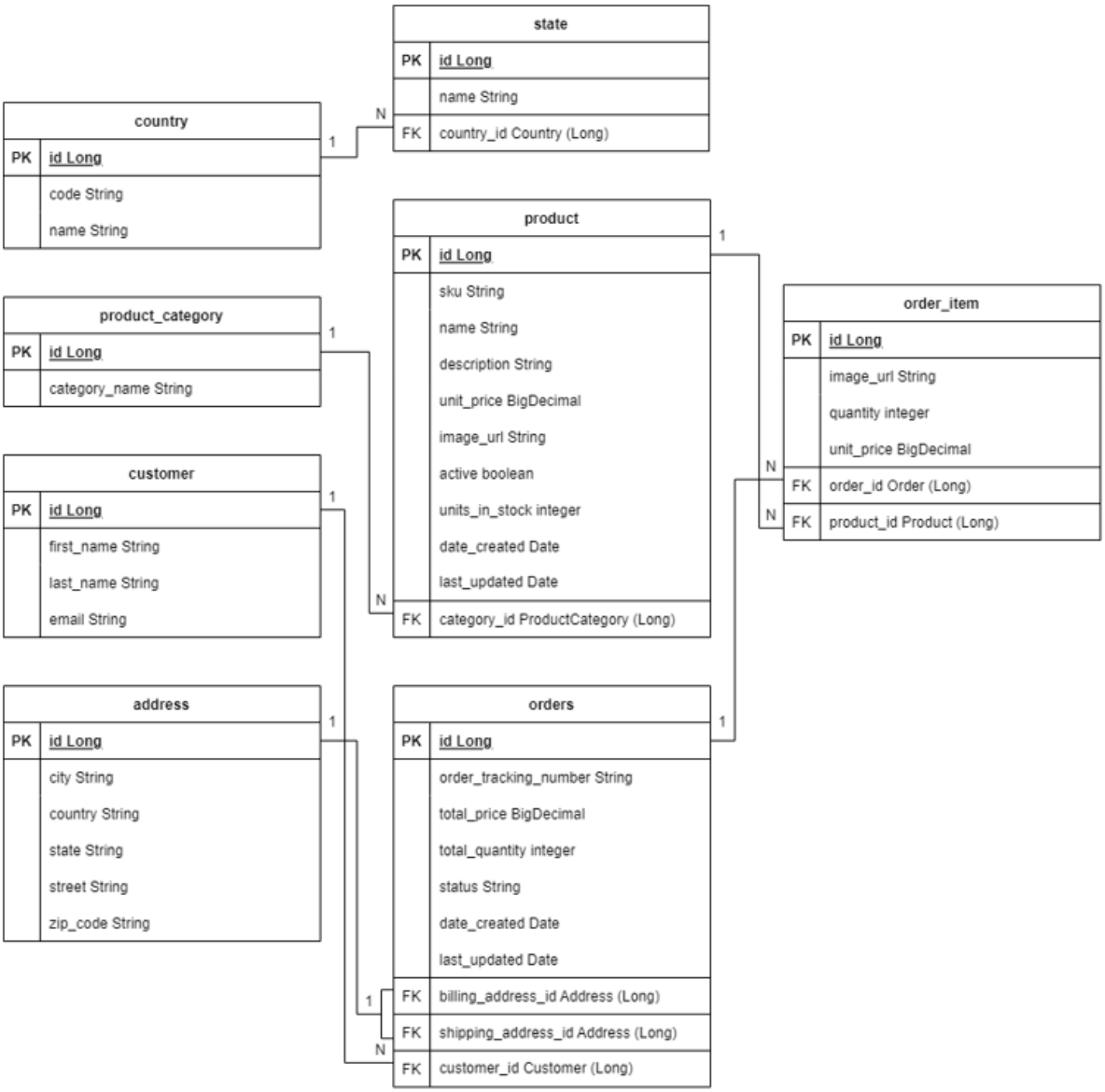
Baza podataka za potrebe izrade web trgovine za prodaju nogometne opreme je PostgreSQL, implementirana je unutar Spring Boot razvojnog okvira i upravljana pomoću sučelja pgAdmin 4. Rad baze podataka temeljen je na nekoliko operacija koje omogućuju kreiranje, pretragu, dodavanje, čitanje ili brisanje podataka, te ostale operacije. Baza podataka za ovaj završni rad nosi naziv *full-stack-ecommerce*. Unutar navedene baze podataka nalaze se strukture entiteta (tablice) pod nazivima *address*, *country*, *customer*, *order_item*, *orders*, *product*, *product_category* i *state*.

Implementacija, odnosno podešavanje konfiguracije baze podataka vrši se najprije instalacijom PostgreSQL Driver ovisnosti, prilikom inicijalizacije projekta unutar Spring Initializr alata. Nakon kreiranja projekta potrebno je implementirati bazu podataka unutar konfiguracijske datoteke – slika 4.2. Implementiranjem baze podataka ostvarili su se potrebni uvjeti za komunikaciju i razmjenu informacija između baze podataka i aplikacije. Također, kreiranjem baze podataka omogućeno je i kreiranje entiteta (tablica) pomoću Spring Boot programskog okvira.

```
### DB connection ###
spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.url=jdbc:postgresql://127.0.0.1:5432/full-stack-ecommerce
spring.datasource.username=ecommerceapp
spring.datasource.password=ecommerceapp
spring.jpa.database=POSTGRESQL
spring.jpa.database-platform = org.hibernate.dialect.PostgreSQL92Dialect
spring.jpa.open-in-view=true
spring.jpa.show_sql=false
spring.jpa.generate-ddl=false
spring.jpa.hibernate.use-new-id-generator-mappings=true
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
```

Sl. 4.2. Konfiguracija PostgreSQL baze podataka.

Prema Spring Bootu, entitet obično predstavlja tablicu u bazi podataka, ukoliko klasu označimo pomoću anotacije *@Entity* Spring Boot će automatizmom stvoriti pripadajuću tablicu za tu klasu u bazi podataka koja ukazuje na strukturu entiteta. Entiteti predstavljaju klase koje služe za definiranje strukture podataka u aplikaciji koji će biti pohranjeni, odnosno mapirani u tablice baze podataka. Budući da Spring Boot koristi JPA (engl. *Java Persistance API*) za rad s entitetima, potrebno je prilikom inicijalizacije projekta uključiti ovisnost Spring Data JPA koja omogućuje pretvorbu programskog koda Java na upite u SQL-u.



Sl. 4.3. E-R dijagram baze podataka.

Prema slici 4.3. prikazan je E-R dijagram (engl. *Entity-Relationship diagram*), odnosno model tablica podataka u bazi podataka kojim su vizualizirani entiteti i njihovi međusobni odnosni unutar baze podataka s pripadajućim atributima koji označavaju karakteristike i svojstva pojedinog entiteta.

Struktura entiteta *address* omogućuje spremanje svih podataka koji će biti upisani prilikom popunjavanja formi za adresu dostave, odnosno adresu za naplatu, što unutar baze podataka stvara tablicu koja se sastoji od stupaca:

- *id* – identifikacijska oznaka, predstavlja primarni ključ (engl. *primary key*) tablice
- *city* – označava grad kojeg je korisnik upisao prilikom popunjavanja forme
- *country* – odnosi se na državu koju je korisnik odabrao tijekom popunjavanja forme

- *state* – odnosi se na županije, države, pokrajine unutar država koje je korisnik odabrao tijekom popunjavanja forme
- *street* – predstavlja ulicu s kućnim brojem koju unosi korisnik
- *zip_code* – označava poštanski broj za određeni grad kojeg korisnik unosi

Tablica *address* povezana je s tablicom *orders* vezom 1:1 (engl. *one-to-one relationship*), budući da se u tablici *orders* nalaze stupci *shipping_address_id* i *billing_address_id* koji su strani ključevi (engl. *foreign key*) te tablice. Veza 1:1 je iz razloga što će za svaku pojedinu narudžbu od strane kupca postojati samo jedna adresa dostave, odnosno adresa naplate, ali s posebnom identifikacijskom oznakom za svaku (korisniku je omogućeno da adrese dostave i naplate budu na istu adresu).

```java
@Entity
@Table(name="address")
@Getter
@Setter
public class Address {

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

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

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

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

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

    @Column(name="zip_code")
    private String zipCode;

    @OneToOne
    @PrimaryKeyJoinColumn
    private Order order;
}
```

Sl. 4.4. Primjer implementacije entiteta.

Slikom 4.4. prikazan je primjer implementacije jednog entiteta, u ovom slučaju to je entitet *address* unutar Spring Boot programskog okvira, što je posljedica stvaranja tablice *address* unutar baze podataka. Prema danom Java kodu vidljivo je kako u implementaciji postoje različite anotacije koje označavaju pripadajuće karakteristike, veze i ponašanje te klase, odnosno entiteta i tablice. Anotacije definirane ovim primjerom bit će korištene i kod implementacije ostalih entiteta, a to su:

- *@Entity* – anotacija koja klasu *Address* označava kao JPA entitet i kao takvu će ju mapirati na tablicu u bazi podataka.
- *@Table(name="address")* – anotacija koja se koristi za navedeni naziv tablice u bazi podataka koja će se koristiti za pohranu podataka entiteta *Address*. U ovom slučaju, zadani naziv tablice je *address*.
- *@Getter* i *@Setter* – anotacije koje označavaju generiranje *get* i *set* metoda za sva polja u klasi. Ove anotacije pripadaju Lombok ovisnosti koja ih automatski generira, što znači da ih ne moramo ručno pisati
- *@Id* – anotacija koja označava polje *id* kao primarni ključ entiteta
- *@GeneratedValue(strategy = GenerationType.IDENTITY)* – anotacija koja postavlja strategiju generiranja vrijednosti za primarni ključ. U ovom slučaju vrijednost primarnog ključa će se generirati prema strategiji koju podržava baza podataka.
- *@Column(name="...")* – anotacija koja označava pojedinačni atribut entiteta i njegov odgovarajući stupac u bazi podataka.
- *@OneToOne* i *@PrimaryKeyJoinColumn* – anotacije koje označavaju kako postoji relacija „jedan naprema jedan" između entiteta *address* i entiteta *order*, s *@PrimaryKeyJoinColumn* anotacijom naznačeno je da će se primarni ključ entiteta *address* koristiti kao i primarni ključ entiteta *order* što stvara jednostruku vezu između adrese i narudžbe.

Nadalje, struktura E-R dijagrama opisuje kako je tablica *country* povezana s tablicom *state* vezom 1:N (engl. *one-to-many relationship*), budući da se u tablici *state* nalazi stupac *country_id* koji je označen kao strani ključ (engl. *foreign key*). Veza 1:N je iz razloga što svaka pojedina država ima neki veći određeni broj županija, država, odnosno pokrajina koje će korisnik odabrati prilikom popunjavanja forme za narudžbu proizvoda. Za potrebe baze podataka unutar ovog web shop-a ukupno je određeno sedam država prema kojima je moguća transakcija proizvoda, a to su Brazil (engl. *Brazil*), Kanada (engl. *Canada*), Njemačka (engl. *Germany*), Indija (engl. *India*),

Turska (engl. *Turkey*), Sjedinjene Američke Države (engl. *United* States) i Hrvatska (engl. *Croatia*).

Tablica *country* daje popis svih dostupnih zemalja za dostavu koje su dodijeljene korisniku na odabir prilikom popunjava forme za narudžbu. Tablica *country* sastoji se od stupaca:

- *id* – identifikacijska oznaka, predstavlja primarni ključ tablice
- *code* – kratka oznaka koja označava domenu države
- *name* – predstavlja ime države

Tablica *customer* omogućuje spremanje svih podataka koji će biti upisani prilikom popunjavanja forme za osnovne podatke o korisniku (ime, prezime, e-mail) kod naručivanja proizvoda. Tablicom *customer* moguće je dobiti uvid u sve korisnike koji su naručivali proizvode unutar web trgovine.

Tablica *customer* sastoji se od stupaca:

- *id* – identifikacijska oznaka za svakog korisnika koji je naručio proizvod u web trgovini, predstavlja primarni ključ tablice
- *first_name* – označava ime korisnika koje je upisano prilikom popunjavanja forme
- *last_name* – označava prezime korisnika koje je upisano prilikom popunjavanja forme
- *email* – označava e-mail korisnika koji je upisan prilikom popunjavanja forme (prijavljenim korisnicima automatski je generiran e-mail koji su upisali prilikom prijave u sustav web trgovine)

Tablica *customer* povezana je s tablicom *orders* vezom 1:N (engl. *one-to-many relationship*), jer se u tablici *orders* nalazi stupac *customer_id* koji je određen kao strani ključ. Vezom 1:N određeno je da svaki korisnik ima svoju jedinstvenu identifikacijsku oznaku prema kojoj će biti identificiran u bazi podataka kao naručitelj jedne ili više narudžbi.

Tablica *order_item* sadrži podatke o proizvodu koji je dodan u košaricu, a kasnije i potencijalno naručen od strane korisnika. Odnosno, tablica se sastoji od slike proizvoda, sadrži i stupce za ukupnu količinu pojedinog proizvoda u košarici te cijenu u odnosu na ukupnu količinu. Također, ova tablica sadrži i podatke o id-u narudžbe i id-u proizvoda dodanog u košaricu. Tablica *order_item* sastoji se od stupaca:

- *id* - identifikacijska oznaka, predstavlja primarni ključ tablice
- *image_url* - putanja (*url*) do slike proizvoda

- *quantity* - ukupna količina određenog proizvoda unutar košarice
- *unit_price* - jedinstvena cijena za određenu količinu pojedinog proizvoda
- *order_id* - predstavlja strani ključ (engl. *foreign key*) tablice koji označava identifikacijsku oznaku narudžbe u kojoj je dodan proizvod
- *product_id* - predstavlja strani ključ tablice koji označava identifikacijsku oznaku proizvoda koji naručujemo

Tablica *order_item* povezana je s tablicom *orders* vezom N:1 (engl. *many-to-one relationship*), budući da se u tablici *order_item* nalazi stupac *order_id* (strani ključ). Veza N:1 je iz razloga što može postojati jedan ili više proizvoda u pojedinoj narudžbi. Tablica *order_item* ima vezu N:1 i s tablicom *product*, jer se u njoj nalazi stupac *product_id*. Ova veza je iz razloga što je moguće odabrati jedan proizvod više puta unutar košarice te nakon toga ga i naručiti.

Tablica *orders* označava sve narudžbe od strane korisnika web shop-a, uz prikaz stupaca pojedinih detalja svake narudžbe. Tablica se sastoji od stupaca:

- *id* - identifikacijska oznaka, predstavlja primarni ključ tablice
- *order_tracking_number* - jedinstveni broj dodijeljen korisniku za praćenje stanja narudžbe
- *total_price* - ukupna cijena narudžbe
- *total_quantity* - ukupna količina proizvoda unutar košarice
- *status* - status narududžbe
- *date_created* - datum kada je narudžba izvršena
- *last_updated* - datum kada je narudžba zadnji put izmijenjena
- *billing_address_id* - predstavlja strani ključ tablice koji označava identifikacijsku oznaku adrese za naplatu narudžbe (u nekim slučajevima je ista kao i adresa dostave)
- *shipping_address_id* - predstavlja strani ključ tablice koji predstavlja identifikacijsku oznaku adrese za dostavu narudžbe
- *customer_id* - predstavlja strani ključ tablice koji predstavlja identifikacijsku oznaku korisnika koji je izvršio narudžbu

Tablica *orders* povezana je s tablicom *address* vezom 1:1, budući da se u tablici *orders* nalaze stupci *billing_address_id* i *shipping_address_id*. Veza 1:1 je iz razloga što kupac može ostaviti jednu adresu narudžbe koja sadrži podatke o jednoj adresi za naplatu narudžbe, odnosno adresi za dostavu narudžbe. Također, tablica *orders* povezana je i s tablicom *customer* vezom N:1, jer se u tablici *orders* nalazi stupac *customer_id*. Veza N:1 je zbog toga što jedna ili više narudžbi može biti izvršeno od strane jednog korisnika web shopa.

Tablica *product* označava sve potrebne podatke o svakom pojedinom proizvodu koje nudi web shop. Tablica je sastavljena od stupaca:

- *id* - identifikacijska oznaka, predstavlja primarni ključ tablice
- *sku* - šifra pojedinog proizvoda ili artikla (engl. *stock keeping unit*)
- *name* - naziv proizvoda
- *description* - opis proizvoda za što je namijenjen
- *unit_price* - jedinična cijena proizvoda
- *image_url* - putanja (*url*) do slike proizvoda
- *active* - dostupnost proizvoda u web shop-u
- *units_in_stock* - količina pojedinog proizvoda u skladištu
- *date_created* - datum dodavanja proizvoda
- *last_updated* - zadnje ažuriranje proizvoda
- *category_id* - predstavlja strani ključ tablice koji predstavlja identifikacijsku oznaku kategorije kojoj pripada pojedini proizvod

Tablica *product* povezana je s tablicom *product_category* vezom N:1, budući da se u tablici *product* nalazi strani ključ *category_id*. Veza N:1 je iz razloga što može postojati više proizvoda vezanih uz jednu određenu kategoriju.

Unutar baze podataka ovog web shop-a ukupno se nalazi 25 proizvoda po svakoj kategoriji proizvoda (5 kategorija).

Tablica *product_category* označava podatke o imenu i identifikacijskoj oznaci kategorije kojoj određeni proizvod pripada. Tablica *product_category* sastavljena je od stupaca:

- *id* - identifikacijska oznaka, predstavlja primarni ključ tablice
- *category_name* - naziv kategorije proizvoda

Tablica *product_category* povezana je s tablicom *product* vezom 1:N budući da se u tablici *product* nalazi strani ključ *category_id*. Veza 1:N je iz razloga što po jednoj kategoriji proizvoda može biti spremljeno više proizvoda.

Unutar baze podataka za ovaj web shop ukupno je pet kategorija. Nazivi kategorija proizvoda su *Football boots*, *Indoor football boots*, *Football kits*, *Football balls* i *Other football equipment*.

Tablica *state* sadrži podatke o pokrajinama ili županijama unutar država prema kojima je moguće vršiti narudžbu proizvoda. Tablica se sastoji od stupaca:

- *id* - identifikacijska oznaka, predstavlja primarni ključ tablice
- *name* - naziv pokrajine ili županije određene države
- *country_id* - predstavlja strani ključ tablice koji predstavlja identifikacijsku oznaku države kojoj pripada određena pokrajina ili županija

Tablica *state* povezana je s tablicom *country* vezom N:1 budući da se u tablici *state* nalazi strani ključ *country_id*. Veza N:1 je iz razloga što može postojati više pokrajina ili županija unutar jedne države.

4.2. Realizacija i korištenje aplikacije na korisničkoj strani

U ovom potpoglavlju predstavljena je realizacija aplikacije s korisničke strane te način rada aplikacije u komunikaciji s korisnikom. Kao što je već prikazano u prethodnim poglavljima, *front-end* dio aplikacije izvršen je pomoću programskog okvira Angular, što je ovim poglavljem detaljno objašnjeno, od stvaranja funkcija, petlji i metoda do rada pojedinih gumba (engl. *button*) i što se događa u aplikaciji u interakciji s korisnikom. Dijelovi aplikacije vidljivi korisniku i opisani ovim poglavljem su prikaz popisa proizvoda, pretraživanje proizvoda prema kategoriji i nazivu (ključnoj riječi), prikaz detalja o proizvodu, prikaz stanja u košarici te narudžba proizvoda iz košarice.

4.2.1. Implementacija popisa proizvoda

Stvaranje prikaza popisa proizvoda vidljivog na početnoj stranici projekta web trgovine započinje kreiranjem nove Angular komponente koristeći Angular CLI. Generiranje komponente i spremanje u poddirektorij izvršava se na način prikazan slikom 4.5.

```
ng generate component components/product-list
```

Sl. 4.5. Naredba za generiranje Angular komponente product-list.

Sljedeći korak u implementaciji popisa proizvoda je razvijanje klase proizvoda. Klasu je također potrebno generirati, što radimo na sličan način kao i generiranje komponente, odnosno generiramo klasu *Product* i spremamo ju u poddirektorij *common* čime nam Angular generira TypeScript klasu. Prikaz stvaranja klase proizvoda izvršava se na način prikazan slikom ispod.

```
ng generate class common/product
```

Sl. 4.6. Naredba za generiranje Angular klase Product.

Razvijanje klase *Product* koja će sadržavati i spremati podatke izvršava se na način da je potrebno dodati svojstva koja će proizvod sadržavati što se izvršava korištenjem svojstva parametara koja su na isti način predočena REST API-em. Slika 4.7. prikazuje klasu *Product* sa svim potrebnim parametrima u komunikaciji s bazom podataka.

```
export class Product {
    id!: string;
    sku!: string;
    name!: string;
    description!: string;
    unitPrice!: number;
    imageUrl!: string;
    active!: boolean;
    unitsInStock!: number;
    dateCreated!: Date;
    lastUpdated!: Date;
}
```

Sl. 4.7. Implementacija klase Product.

Servis ili usluga (engl. *service*) potreban je kod implementacije pozivanja REST API arhitekture kao posrednika komunikacije između servera i klijenta. Servis predstavlja pomoćnu klasu koja pruža željenu funkcionalnost aplikacije, pokretana na klijentskoj strani aplikacije. REST klijent osiguran od strane Angular programskog okvira naziva se HTTP client koji je dio HttpClientModule ovisnosti. Glavni dijelovi servisa označavaju kako on mora imati mogućnost implementiranja u druge klase ili komponente pa ga je potrebno naznačiti anotacijom @Injectable. Također, u servisu je određen i primarni URL koji će korisnika preusmjeriti na stranicu proizvoda.

Prikaz podataka o svakom pojedinom proizvodu vrši se na način da se iterira listom proizvoda u nizu i nakon toga se generira HTML element (u ovom slučaju paragraf) za svaki element u nizu, prikazano kodom na slici 4.8.

```
<p *ngFor="let tempProduct of products">
    {{ tempProduct.name }}: {{ tempProduct.unitPrice | currency:'EUR' }}
</p>
```

Sl. 4.8. Implementacija product-list.component.html stranice.

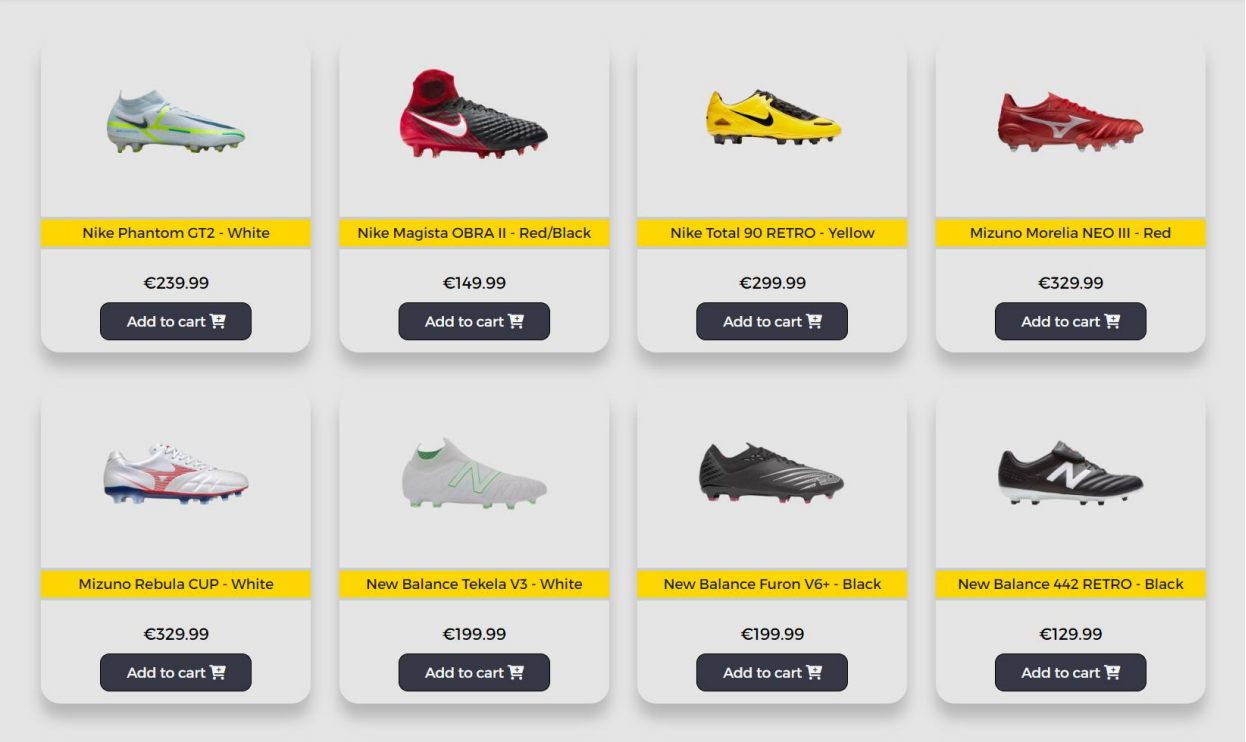
Detaljan prikaz elemenata tablično ili u tzv. obliku rešetke (engl. *grid*) izvršava se na sličan način, gdje su implementirani usmjerivači na prikaz slike, naziva i cijene proizvoda, kao i gumb za dodavanje proizvoda u košaricu. Implementacija je prikazana slikom 4.9. postavljenjem svih elemenata unutar jednog odjeljka koji predstavlja blok elementa s pripadajućim hipervezama, gumbima i elementima za postavljanje slike.

```
<div class="row">

    <!-- loop over the collection of products -->
    <div *ngFor="let tempProduct of products" class="col-md-3">

        <div class="product-box">

            <!-- link on the product image-->
            <a routerLink="/products/{{ tempProduct.id }}">
                <img src="{{ tempProduct.imageUrl }}" class="img-responsive">
            </a>


            <!-- link on the product name-->
            <a routerLink="/products/{{ tempProduct.id }}">
            <h1>{{ tempProduct.name }}</h1>
            </a>

            <div class="price">{{ tempProduct.unitPrice | currency:'EUR' }}</div>

            <button (click)="addToCart(tempProduct)" class="btn-add-to-cart-list mt-2">
                Add to cart
            <i class="fa fa-cart-plus" aria-hidden="true"></i></button>
        </div>
    </div>

    <div *ngIf="products?.length == 0" class="alert alert-war col-md-12" role="alert">
        No products found.
    </div>

</div>
```

Sl. 4.9. Implementacija product-list-grid.component.html stranice.

Kasnije, sva HTML struktura za komponentu prikaza liste proizvoda spaja se u HTML dokument *app.component.html* što je zapravo prikaz početne stranice, gdje je vidljiva i lista svih proizvoda koji se nalaze spremljeni u bazi podataka. Selektor kojim se identificira komponenta liste proizvoda u HTML kodu i gdje će biti producirana u korisničkom sučelju je *<app-product-list>* kasnije zamijenjen *<router-outlet>* selektorom kako bi se svaka promjena na stranici dinamički učitavala na temelju trenutne rute.

Iteracija listom proizvoda koji se nalaze na početnoj stranici aplikacije je vrlo jednostavna za korisnika aplikacije. Svaki proizvod se nalazi u svojoj zasebnoj kartici prema *grid* načinu prikaza, gdje se nalazi slika proizvoda, naziv proizvoda, cijena proizvoda i gumb za dodavanje proizvoda u košaricu. Klikom gumba našeg pokazivača na sliku ili naziv proizvoda, stranica nas preusmjerava na prikaz detalja o proizvodu, što je tema potpoglavlja 4.2.4. Slikom 4.10. prikazan je izgled liste proizvoda na početnoj stranici web trgovine.



Sl. 4.10. Prikaz izgleda liste proizvoda na početnoj stranici aplikacije.

4.2.2. Implementacija pretraživanja proizvoda prema kategoriji

Prvi korak kod implementacije pretraživanja proizvoda prema kategoriji je podešavanje aplikacije na *back-end* dijelu, odnosno „detektiranje“ ID-a pojedinog entiteta kako bi se dobila pojedina kategorija proizvoda iz liste te kasnije dobio prikaz svih proizvoda iz odabrane kategorije. Unutar Spring Boot klase *MyDataRestConfig* stvorena je metoda *exposeIds* gdje se dohvaća lista svih klasa entiteta iz upravitelja entiteta (engl. *Entity Manager*). *EntityManager* je sučelje Spring Boot-a koje omogućuje interakciju s JPA (engl. *Java Persistance Api*) zavisnosti podataka, uključujući pritom upravljanje entitetima, operacijama i upitima prema bazi podataka. Nadalje, u metodi *exposeIds* kreirano je polje tipova entiteta. Nakon toga, iteracijom *for* petlje dobivaju se tipovi entiteta, i na kraju se „izlažu“ ID-ovi entiteta za niz tipova entiteta/domena. Slikom 4.11. prikazana je implementacija metode *exposeIds* unutar klase *MyDataRestConfig*.

```
private void exposeIds(RepositoryRestConfiguration config) {

    // list of all entity classes from the entity manager
    Set<EntityType<?>> entities = entityManager.getMetamodel().getEntities();

    // array of the entity types
    List<Class> entityClasses = new ArrayList<>();

    // entity types for the entities
    for (EntityType tempEntityType : entities) {
        entityClasses.add(tempEntityType.getJavaType());
    }

    // expose entity ids for the array of entity/domain types
    Class[] domainTypes = entityClasses.toArray(new Class[0]);
    config.exposeIdsFor(domainTypes);
}
```

Sl. 4.11. Implementacija exposeIds metode.

Sljedeći korak u implementaciji je stvaranje Angular klase za kategoriju proizvoda – *ProductCategory*. Slikom 4.12. prikazana je navedena klasa s pripadajućim parametrima – id kategorije i naziv kategorije proizvoda.

```
export class ProductCategory {
    id!: number;
    categoryName!: string;
}
```

Sl. 4.12. Implementacija klase ProductCategory.

Nakon kreiranja klase, slijedi kreiranje komponente koja će predstavljati kategorije proizvoda u web trgovini. Komponenta je stvorena pod nazivom *product-category-menu.* Komponentu je potrebno implementirati na način da čita kategorije iz usluge servisa. Implementacija je prikazana slikom 4.13. gdje je anotacijom *@Component* naznačeno kako se radi o komponenti s pripadajućim selektorom za kasnije korištenje u implementaciji pripadajućeg HTML-a za kategoriju proizvoda, dok se *templateUrl i styleUrls* odnose na HTML predložak koji će biti stvoren i pripadajuće CSS stilove koji će se koristiti. Daljnjom analizom koda, vidljivo je kako je na početku deklarirano polje *productCategories* koje će sadržavati objekte tipa *ProductCategory* i kasnije će se inicijalizirati u kodu. Nadalje se definira konstruktor za komponentu koja prima *productService* objekt tipa *ProductService* putem definiranja ovisnosti. Metodom *ngOnInit* definirano je kako će se pozivati metoda *listProductCategories* kada se inicijalizira komponenta. Metoda *listProductCategories* poziva metodu za dohvaćanje kategorija

proizvoda – *getProductCategories()*, odnosno metodu servisa *productService*. Kada su podaci dostupni, tada se poziva funkcija unutar *subscribe* bloka koja postavlja *productCategories* na željene dobivene podatke i ispisuje ih konzolom. *Subscribe* metoda omogućuje preuzimanje podataka iz servisa.

```
import { Component, OnInit } from '@angular/core';
import { ProductCategory } from 'src/app/common/product-category';
import { ProductService } from 'src/app/services/product.service';

@Component({
  selector: 'app-product-category-menu',
  templateUrl: './product-category-menu.component.html',
  styleUrls: ['./product-category-menu.component.css']
})
export class ProductCategoryMenuComponent implements OnInit {

  productCategories!: ProductCategory[];

  constructor(private productService: ProductService) { }

  ngOnInit(): void {
    this.listProductCategories();
  }

  listProductCategories() {

    this.productService.getProductCategories().subscribe(
      data => {
        console.log('Product Categories=' + JSON.stringify(data));
        this.productCategories = data;
      }
    );
  }
```

Sl. 4.13. Implementacija funkcionalnosti klase za komponentu ProductCategoryMenuComponent.

Prilikom implementacije pretraživanja proizvoda prema kategoriji potrebno je i ažurirati klasu *ProductService* u smislu postavljanja novog URL-a za dohvaćanje podataka o kategoriji proizvoda iz baze podataka. Također, potrebno je implementirati metodu *getProductCategories()* koja je odgovorna za dohvaćanje kategorija proizvoda s poslužitelja koristeći HTTP GET zahtjev i vraćanje tih kategorija kao *Observable* koji kao dio Angular biblioteke služi u ovom slučaju za emitiranje niza objekata tipa *ProductCategory*. Prema slici 4.14. prikazana je metoda *getProductCategories()*. Definiranje sučelje (engl. *interface*) također je nužno implementirati, sučelje *GetResponseProductCategory* opisuje očekivani format odgovora koji će se dobiti prilikom dohvaćanja podataka o kategoriji proizvoda s poslužitelja.

```
  getProductCategories(): Observable<ProductCategory[]> {

    return this.httpClient.get<GetResponseProductCategory>(this.categoryUrl).pipe(
      map(response => response._embedded.productCategory)
    );
  }
}
```

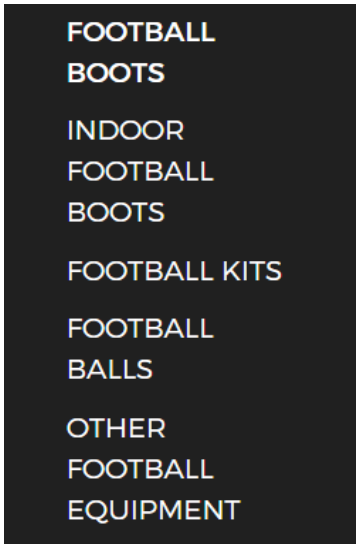
Sl. 4.14. Implementacija metode getProductCategories u ProductService.

Stvaranje HTML dokumenta za listu proizvoda prikazan je slikom 4.15. gdje je izvedena petlja koja prolazi kroz kategorije proizvoda te stvara dinamičke linkove, linka se ID kategorije za kasniju iteraciju korisnika stranicom i ostvaruje se prikaz kategorije proizvoda prema imenu.

```
<div class="menu-sidebar-content js-scrollbar1">
    <nav class="navbar-sidebar">
        <ul class="list-unstyled navbar-list">

            <li *ngFor="let tempProductCategory of productCategories">
                <a routerLink="/category/{{ tempProductCategory.id }}" routerLinkActive="active-link">
                    {{ tempProductCategory.categoryName }}
                </a>
            </li>


        </ul>
    </nav>
</div>
```

Sl. 4.15. Implementacija HTML dokumenta product-category-menu.component.html.

Prikaz liste proizvoda web trgovine lako je uočljiv za korisnika web trgovine, budući da se nalazi ispod loga aplikacije i lijevo od prikaza liste proizvoda. Klikom gumba našeg pokazivača na jednu od opcija kategorije dobije se prikaz svih proizvoda koje ta određena kategorija nudi. Prikaz popisa proizvoda detaljno je opisan u prethodnom potpoglavlju. Slika 4.16. odnosi se na vizualni prikaz liste s kategorijom proizvoda implementiranog na web trgovini.



Sl. 4.16. Prikaz izgleda liste kategorija proizvoda.

4.2.3. Implementacija pretraživanja proizvoda prema ključnoj riječi

Stvaranje pretraživača proizvoda prema ključnoj riječi, odnosno prema nazivu koji proizvod nosi unutar baze podataka započinje definiranjem metode *findByNameContaining*. Prema Spring Boot-u i njegovog ugrađenom JPA-u (engl. *Java Persistance API*) ova metoda će pratiti određenu deklaraciju koja omogućuje automatsko generiranje SQL upita. U ovom slučaju metoda *findByNameContaining* traži proizvode, odnosno entitete koji su tipa *Product* čije ime sadrži određeni niz znakova koji je tražen. Kao rezultat, očekuje se vraćanje stranice proizvoda (engl. *Page*). Nužno je da se metodi proslijedi parametar zahtjeva *Page* putem HTTP zahtjeva. Implementacija metode unutar sučelja *ProductRepository* prikazana je slikom 4.17.

```
Page<Product> findByNameContaining(@RequestParam("name") String name, Pageable pageable);
```

Sl. 4.17. Implementacija metode findByNameContaining.

Kao i u prethodnim potpoglavljima, dalje slijedi generiranje komponente – komponenta *Search*. Prije definiranja metoda koje opisuju ovu komponentu, potrebno je implementirati novu Angular rutu (engl. *route*) u *app.module.ts* za upravljanje navigacijom aplikacije. Prema slici 4.18. definirana je ruta s imenom *'search/:keyword'* gdje *search* predstavlja temeljni *path* koji će se koristiti u URL-u, dok *keyword* označava rutu parametra koja očekuje da se iza *search/* nalazi nekakav dinamički parametar, u ovom slučaju to je upravo *keyword* koji predstavlja naziv komponente proizvoda. Dio rute *component: ProductListComponent* definira komponentu koja će se prikazivati kada se korisnika usmjerava na ovu rutu. U ovom slučaju Angular će prikazati komponentu *ProductListComponent*, kada korisnik posjeti URL koji odgovara implementiranoj ruti *'search/:keyword'*.

```
{path: 'search/:keyword', component: ProductListComponent}
```

Sl. 4.18. Implementacija rute za pretraživanje proizvoda prema ključnoj riječi.

Ažuriranje komponente *SearchComponent* za slanje podataka na rutu pretraživanja započinje implementiranjem *Event Binding* elemenata unutar *input* parametra u HTML dokumentu *search.component.html*. Korisnik unutar polja za pretraživanje unosi tekst, odnosno ključnu riječ koja predstavlja naziv proizvoda. Pristup elementu pretraživanja naznačen je s *#myInput* gdje će unos (engl. *input*) bit tekst. Za pretraživanje proizvoda prema ključnoj riječi potrebno je implementirati metodu *doSearch* koju je moguće potvrditi pritiskom tipke *enter* na tastaturi ili klikom gumba našeg pokazivača na gumb *Search*. Implementacija *search.component.html* dokumenta prikazana je slikom 4.19.

```
<div class="form-header">

    <input #myInput type="text"
           placeholder="Search for products ..."
           class="au-input au-input-xl"
           (keyup.enter)="doSearch(myInput.value)"/>

    <button (click)="doSearch(myInput.value)" class="au-btn-submit">
        Search
    </button>

</div>
```

Sl. 4.19. Implementacija HTML dokumenta search.component.html.

Slikom 4.20. prikazana je već spomenuta *doSearch* funkcija. Ova funkcija prima jedan parametar *value* koji je tipa *string*, što znači da očekuje string kao ulazni parametar kada se poziva. Metodom je definirano kako će se u konzolu ispisivati poruka koja sadrži vrijednost parametra *value*. Kako bi se promijenila ruta koristi se *router* objekt koji poziva metodu *navigateByUrl* za navigaciju na određenu rutu koja koristi već spomenuti dinamički generirani URL koji uključuje *value* kao dio rute.

```
doSearch(value: string) {
  console.log(`value=${value}`);
  this.router.navigateByUrl(`/search/${value}`);
}
```

Sl. 4.20. Implementacija doSearch funkcije.

Prilikom implementacije pretraživanja proizvoda prema ključnoj riječi potrebno je i ažurirati klasu *ProductService* postavljanjem URL-a za dohvaćanje proizvoda prema ključnoj riječi iz baze podataka. Također, potrebno je implementirati *getProducts()* metodu, koja je odgovorna za dohvaćanje proizvoda prema ključnoj riječi s poslužitelja koristeći HTTP GET zahtjev i vraćanje tih proizvoda kao *Observable* koji služi u ovom slučaju za emitiranje niza objekata tipa *Product*. Dok definiranje sučelja (engl. *interface*) *GetResponseProducts* opisuje očekivani format odgovora koji će se dobiti prilikom dohvaćanja podataka prema ključnoj riječi s poslužitelja.

Interakcija korisnika u smislu pretraživanja proizvoda prema ključnoj riječi je vrlo praktično. Budući da se područje za pretraživanje proizvoda s pripadajućim gumbom *Search* nalazi u zaglavlju stranice, lako je vidljivo korisniku. Unese li korisnik ispravan naziv proizvoda kojeg želi pretraživati, tada će mu se na zaslon ispisati svi proizvodi koji sadrže ključnu riječ koju se korisnik unio. Unese li korisnik naziv proizvoda koji ne postoji u web trgovini, tada će mu se na zaslonu ispisati poruka kako nije pronađen nijedan proizvod - *No products found*. Potvrđivanje

unesene riječi korisnik može potvrditi pritiskom tipke *enter* ili klikom pokazivača na gumb *Search*. Slikom 4.21. prikazano je polje za pretraživanje proizvoda s pripadajućim gumbom za potvrdu.

Sl. 4.21. Prikaz izgleda polja za pretraživanje i gumba Search.

4.2.4. Implementacija prikaza detalja proizvoda

Za implementaciju prikaza detalja proizvoda jedan od uvjeta je kreirati novu komponentu, kao za dosad opisane komponente. Naziv nove generirane komponente je *ProductDetails*. Nakon generiranja komponente potrebno je dodati novu Angular rutu. Prema slici 4.22. novodefinirana ruta je *'products/:id'* gdje *product* predstavlja temeljni *path* koji će se koristiti u URL-u, dok *id* označava rutu parametra koja očekuje da se iza *producst/* nalazi nekakav dinamički parametar koji će predstavljati identifikator proizvoda (ID). Kada korisnik posjeti URL definiran na ruti *'products/id'* prikazat će se komponenta *ProductDetailsComponent*. Za dohvaćanje proizvoda iz usluge *ProductService* potrebno je optimizirati komponentu *ProductDetailsComponent*. Slika 4.22. upravo prikazuje optimizaciju komponente stvaranjem funkcije *handleProductDetails()* gdje je na početku funkcije deklarirana konstanta *idNumber* u koju se sprema dohvaćena vrijednost parametra *id* iz trenutne rute. Također, potrebno je definirati konstantu *theProductId* koja čuva vrijednost konvertiranog ID-a u cijeli broj pomoću funkcije *parseInt*. Zadnji korak u implementaciji ove metode je pozivanje usluge za dohvaćanje podataka o proizvodu. Putem *productService* servisa (veza s *back-end* dijelom), poziva se funkcija *getProduct* s *theProductId* argumentom. Funkcija *getProduct* je asinkrona i vraća Observable tip Angular datoteke. Metodom *subscribe* prati se asinkrono učitavanje podataka i kada se podaci preuzmu, funkcija unutar bloka *subscribe* će se izvršiti tako što će se postaviti podaci proizvoda (engl. *dana*) u lokalnu varijablu *product*.

```
handleProductDetails() {

  const idNumber: any = this.route.snapshot.paramMap.get('id');

  const theProductId = parseInt(idNumber);

  this.productService.getProduct(theProductId).subscribe(
    data => {
      this.product = data;
    }
  )
}
```

Sl. 4.22. Implementacija funkcije handleProductDetails().

Kako bi se dohvatili podaci o proizvodu na temelju njegovog ID-a, potrebno je ažurirati klasu servisa proizvoda – *ProductService*. Stvara se metoda koja predstavlja logiku za pristup i dohvaćanje podataka o proizvodu iz baze podataka, odnosno iz *back-end* dijela aplikacije, na temelju proslijeđenog ID-a proizvoda te koristi Angular *HttpClient* instancu za izvođenje HTTP GET zahtjeva za dohvaćanje proizvoda. Povratna vrijednost je *Observable<Product>* koji emitira podatke o proizvodu. Slika 4.23. prikazuje implementaciju metode *getProduct*.

```
getProduct(theProductId: number): Observable<Product> {

  const productUrl = `${this.baseUrl}/${theProductId}`;

  return this.httpClient.get<Product>(productUrl);
}
```

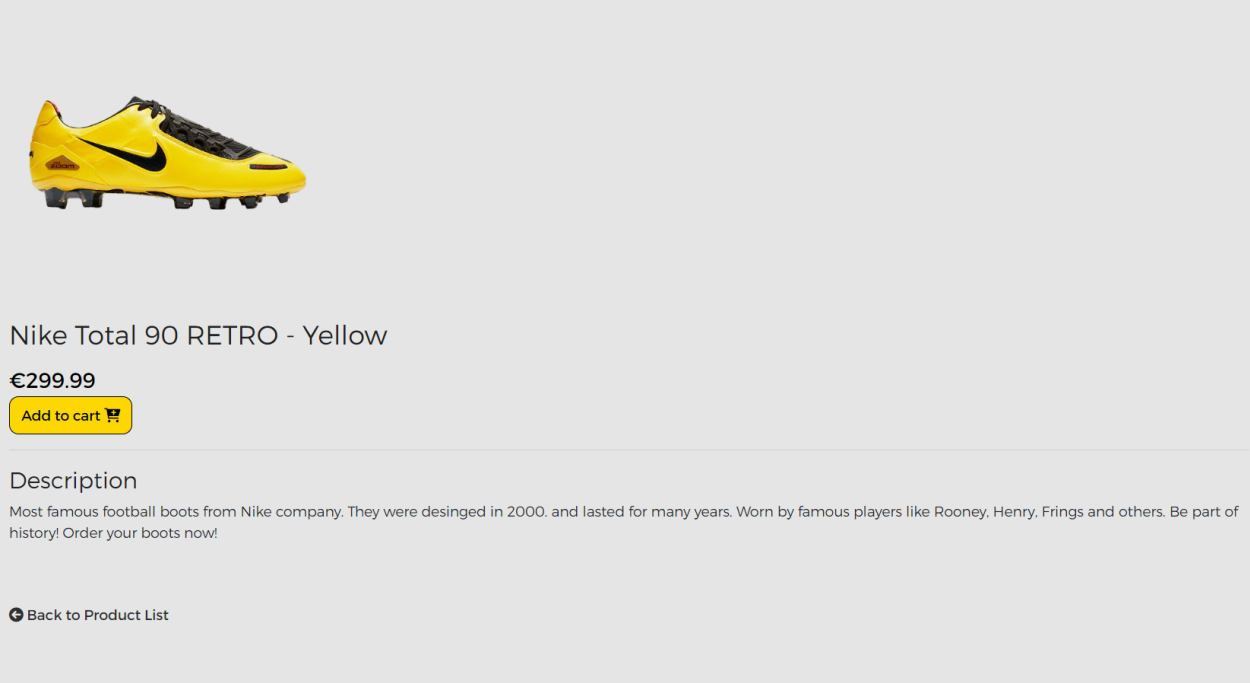
Sl. 4.23. Implementacija metode getProduct.

Prema slici 4.24. stvoren je HTML dokument koji prikazuje stranicu s detaljima proizvoda pojedinog proizvoda. Stranica se sastoji od slike proizvoda preuzeta pomoću poziva *product.imageUrl*, naziva proizvoda dohvaćenog pomoću *product.name*, cijene proizvoda prikazane u eurima, gumba za dodavanje proizvoda u košaricu, opisa proizvoda preuzetog iz baze podataka pozivom *product.description* i gumba *Back to Product List* koji će korisnika vratiti na listu proizvoda koristeći *routerLink* poziv.

```
<div class="detail-section">
    <div class="container-fluid">

        <img src="{{ product.imageUrl }}" class="detail-img">

        <h3>{{ product.name }}</h3>
        <div class="price" style="font-size: 22.5px">{{ product.unitPrice | currency:'EUR' }}</div>

        <button (click)="addToCart()" class="btn-add-to-cart">Add to cart  <i class="fa fa-cart-plus" aria-hidden="true"></i></button>

        <hr>
        <h4>Description</h4>
        <p>{{ product.description }}</p>

        <a routerLink="/products" class="mt-5" style="font-weight: bold; color: #303030;">
            <i class="fa fa-arrow-circle-left" aria-hidden="true"></i>  Back to Product List</a>

    </div>
</div>
```

Sl. 4.24. Implementacija HTML dokumenta product-details.component.html.

Interakcija s korisnikom izvedena je način da ukoliko korisnik na početnoj stranici klikne gumbom pozivača na sliku ili naziv proizvoda tada će ga se preusmjeriti na stranicu detalja proizvoda, gdje se nalaze slika proizvoda u većoj rezoluciji, naziv proizvoda, cijena u eurima, gumb za dodavanje u košaricu, opis proizvoda i gumb koji korisnika preusmjerava nazad na prikaz liste svih proizvoda koje web trgovina posjeduje. Prikaz stranice s detaljima pojedinog proizvoda sa svim opisanim detaljima predstavljen je slikom 4.25.



Sl. 4.25. Prikaz izgleda stranice s detaljima proizvoda.

4.2.5. Implementacija stanja proizvoda u košarica

Stanje košarice za kupovinu implementirano je na desnoj strani zaglavlja stranice (slika 4.26). Korisnik mijenja stanje košarice u ovisnosti dodavanja proizvoda u nju. Prikaz ukupne cijene i količine proizvoda će rasti ukoliko korisnik odabere jedan ili nekoliko proizvoda za kupovinu. Proizvode je moguće dodati u košaricu implementiranim gumbima s nazivom *Add to cart*. Gumbi se nalaze na početnoj stranici kod prikaza proizvoda te na prikazu detalja o proizvodu. Stanje košarice moguće je vidjeti ukoliko korisnik odabere svojim pokazivačem ikonu košarice u zaglavlju ili ikonu prikaza cijene i količine proizvoda u košarici. Prikaz stanja proizvoda u košarici opisan je u daljnjem tekstu ovog potpoglavlja.

Sl. 4.26. Prikaz izgleda stanja košarice.

Implementacija stanja proizvoda u košarici započinje stvaranjem nove komponente pod nazivom *cart-status*. Nakon toga slijedi stvaranje HTML dokumenta za tu komponentu prikazanog slikom 4.27. koji sadrži poveznicu koja će korisnika preusmjeriti na putanju */cart-details*, implementaciju prikaza ukupne cijene i količine proizvoda u košarici te ikonu košarice implementiranu pomoću „Font Awesome“ biblioteke za stvaranje ikona.

```
<div class="cart-area d-n">
    <a routerLink="/cart-details">
        <div class="total" style="font-weight: bold">{{ totalPrice | currency: 'EUR' }}
            <span>{{ totalQuantity }}</span>
        </div>
        <i class="fa fa-shopping-cart" aria-hidden="true"></i>
    </a>
</div>
```

Sl. 4.27. Implementacija HTML dokumenta cart-status.component.html.

Između ostalog, potrebno je ažurirati klasu *ProductListComponent* stvaranjem metode za dodavanje proizvoda u košaricu. Implementacija metode prikazana je slikom 4.28., a metodom je određeno dodavanje proizvoda u košaricu na način da metoda prima proizvod kao parametar nakon čega ispisuje informacije o proizvodu u konzolu, zatim se stvara instanca klase *CartItem* na temelju proizvoda i nakon toga se dodaje u košaricu uslugom *cartService*.

```
addToCart(theProduct: Product) {
  console.log(`Adding to cart: ${theProduct.name}, ${theProduct.unitPrice}`);

  const theCartItem = new CartItem(theProduct);

  this.cartService.addToCart(theCartItem);
}
```

Sl. 4.28. Implementacija metode addToCart.

Implementacija već navedene klase *CartItem* započinje razradom modela te klase koja sadrži važna polja proizvoda za korištenje u košarici, a također sadrži i polje koje označava ukupnu količinu tog proizvoda u košarici. Konstruktor postavlja vrijednosti klase *CartItem* na vrijednosti iz proizvoda koji je proslijeđen kao argument. Početna količina proizvoda (engl. *quantity*) postavljena je na jedan. Prema slici 4.29. implementiran je model klase *CartItem*.

```
export class CartItem {
    id: string;
    name: string;
    imageUrl: string;
    unitPrice: number;

    quantity: number;

    constructor(product: Product) {
        this.id = product.id;
        this.name = product.name;
        this.imageUrl = product.imageUrl;
        this.unitPrice = product.unitPrice;

        this.quantity = 1;
    }
}
```

Sl. 4.29. Implementacija klase CartItem.

Nakon implementacije klase potrebno je implementirati *CartService* uslugu, odnosno razviti metode za dodavanje proizvoda u košaricu – *addToCart* i metodu za izračunavanje ukupnog iznosa cijene u košarici – *computeCartTotals*. Metoda *addToCart* prikazana je slikom 4.30. gdje su na početku definirane varijable *alreadyExistsInCart* i *existingCartItem* kojima će se izvršavati provjera postojanosti artikla u košarici. Iteracijom kroz listu artikala u košarici radimo provjeru postojanosti i na taj način dokazujemo postoji li već u košarici dodan artikl od strane korisnika. Ukoliko je zaključeno da postoji već jedan ili više artikala u košarici tada je potrebno povećati količinu košarice za jedan više artikl od trenutno postojećih, ukoliko ne postoji nijedan artikl u košarici tada se stanje košarice povećava na jedan artikl. Na kraju se poziva metoda koja računa ukupnu količinu proizvoda u košarici i ukupnu cijenu svih proizvoda u košarici - *computeCartTotals*.

```
addToCart(theCartItem: CartItem){
  // check if already have the item in our cart
  let alreadyExistsInCart: boolean = false;
  let existingCartItem: CartItem = undefined;

  if (this.cartItems.length > 0) {

    // find the item in the cart based on item id
    for (let tempCartItem of this.cartItems) {
      if (tempCartItem.id === theCartItem.id) {
        existingCartItem = tempCartItem;
        break;
      }
    }

    //check if we found it
    alreadyExistsInCart = (existingCartItem != undefined);
  }

  if (alreadyExistsInCart) {
    // increment the quantity
    existingCartItem.quantity++;
  }
  else{
    // add the item to the array
    this.cartItems.push(theCartItem);
  }

  // compute cart total price and total quantity
  this.computeCartTotals();
}
```

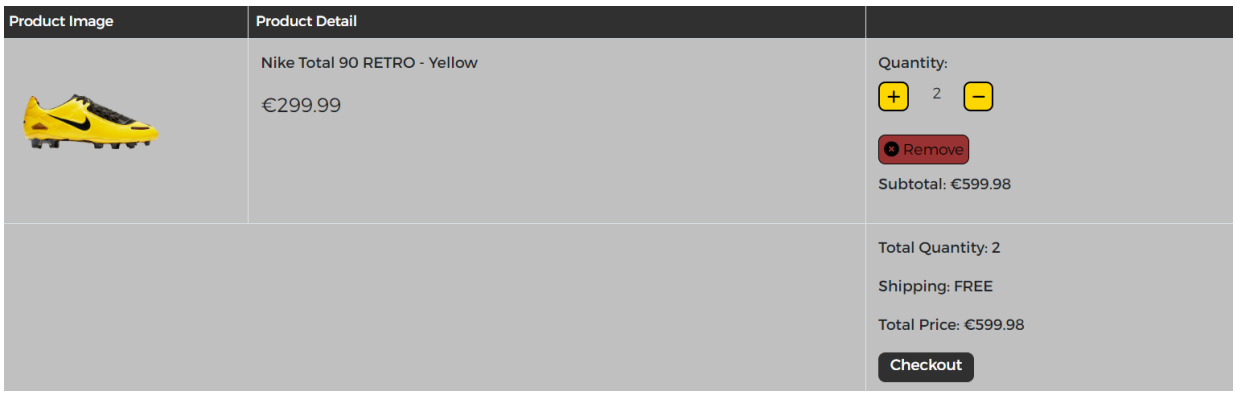
Sl. 4.30. Implementacija metode addToCart u CartService-u.

Stanje ukupne količine proizvoda i ukupne cijene implementirano je metodom *updateCartStatus* unutar klase *CartStatusComponent*. Metoda služi za ažuriranje stanja košarice, odnosno prikazuje ukupnu količinu i cijenu proizvoda u košarici. „Pretplata“ komponenti izvršava se preko *Observable* Angular pomoćne datoteke za ukupnu cijenu, odnosno količinu proizvoda koju pruža usluga *cartService*. Kada se promijene ukupna cijena ili količina artikla u košarici, emitiraju se novi podaci preko *observable-a*. Nakon što se to dogodi, tada se ažuriraju statusi komponente *totalPrice* i *totalQuantity*. Implementacija metode *updateCartStatus* prikazana je slikom 4.31.

```
updateCartStatus() {

  this.cartService.totalPrice.subscribe(
    data => this.totalPrice = data
  );

  this.cartService.totalQuantity.subscribe(
    data => this.totalQuantity = data
  );
}
```

Sl. 4.31. Implementacija metode updateCartStatus u klasi CartStatusComponent.

Stanje košarice, odnosno popis svih stavki moguće je vidjeti kada korisnik svojim pokazivačem bira ikonu košarice ili ikonu prikaza cijene i količine proizvoda u košarici u zaglavlju stranice. Popis stavki u košarici korisniku je prikazan tablicom (slika 4.32.) u kojoj se nalaze stupac s prikazom slike, stupac nazivom i cijenom proizvoda, ukupna količina proizvoda s mogućnosti povećavanja ili smanjivanja, gumb za brisanje proizvoda iz košarice, ukupna suma jednog proizvoda s obzirom na njegovu količinu te ukupan zbroj količine i cijene proizvoda. Ispod informacija o ukupnoj količini i cijeni proizvoda u košarici, nalazi se gumb *Checkout* koji preusmjerava korisnika na stranicu za unos informacija o narudžbi, detaljniji opis gumba *Checkout* i stranice za narudžbu proizvoda je u sljedećem potpoglavlju.



Sl. 4.32. Prikaz izgleda stranice s tablicom popisa stavki u košarici.

Implementacija ovog dijela aplikacije započinje stvaranje Angular komponente s nazivom *cart-details*. Nakon toga potrebno je konfigurirati novu rutu koja će korisnika preusmjeriti na stazu (engl. *path*) *cart-details*, kada treba prikazati komponentu *CartDetailsComponent*. Konfiguracija rute unutar *app.module.ts* dokumenta za modifikaciju prikazana je slikom 4.33.

```
{path: 'cart-details', component: CartDetailsComponent}
```

Sl. 4.33. Implementacija rute za cart-details komponentu.

Implementacija metode *listCartDetails()* nužna je za dohvat stavki koje su korisniku vidljive u ispisu liste proizvoda u košarici. Metoda *listCartDetails()* prikazana je slikom 4.34. i služi za pripremu podataka o košarici za prikazivanje na korisničkom sučelju, na način da dohvaća podatke o svim stavkama košarice i ostvaruje pretplatu na promjene ukupne cijene i količine proizvoda, dok na kraju poziva funkciju za izračun ukupnih vrijednosti košarice.

```
listCartDetails() {

  this.cartItems = this.cartService.cartItems;

  this.cartService.totalPrice.subscribe(
    data => this.totalPrice = data
  );

  this.cartService.totalQuantity.subscribe(
    data => this.totalQuantity = data
  );

  this.cartService.computeCartTotals();
}

incrementQuantity(theCartItem: CartItem) {
  this.cartService.addToCart(theCartItem);
}
```

Sl. 4.34. Implementacija metode listCartDetails.

4.2.6. Implementacija obrasca za narudžbu proizvoda

Implementacija obrasca za narudžbu proizvoda sastoji se od nekoliko dijelova gdje je potrebno ostaviti podatke o naručitelju, odnosno kupcu u vidu unosa podataka o imenu, prezimenu i e-mail adresi. Nadalje, potrebno je ostaviti podatke o adresama za dostavu i naplatu naručenih proizvoda i forma za popunjavanje podataka o kreditnoj kartici putem koje naručujemo proizvode.

Potrebno je generirati novu Angular komponentu s nazivom *checkout*. Kao i u prethodnim potpoglavljima potrebno je dodati i novu rutu koja omogućuje navigaciju na stranicu *checkout*. Kada se korisnik usmjeri na *'/checkout'*, Angular će prikazati *CheckoutComponent*. Definiranje nove rute prikazano je slikom 4.35.

```
{path: 'checkout', component: CheckoutComponent}
```

Sl. 4.35. Implementacija rute za navigaciju korisnika na formu za narudžbu proizvoda.

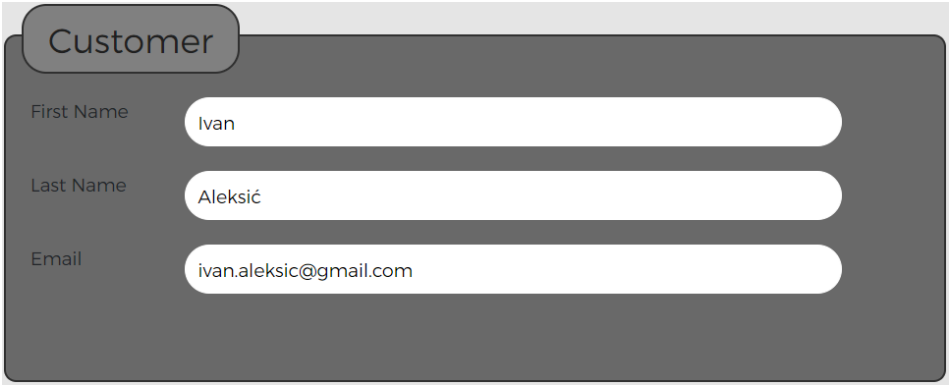
Gumb *checkout* koji preusmjerava korisnika na stranicu s obrascem za narudžbu proizvoda implementiran je na stranici s prikazom tablice proizvoda koje korisnik ima u planu naručiti.

Nadalje slijedi implementacija reaktivnih oblika obrazaca (engl. *reactive forms*) koje koriste programski API (*back-end*) za izradu formi/obrazaca. Budući da su laki za implementaciju i testiranje, u ovom završnom radu predstavljaju skalabilno rješenje za velike, složene oblike obrazaca. Programsku podršku za reaktivne oblike formi potrebno je uključiti unutar *imports* dijela u *app.module.ts*. Potrebno je definirati obrazac unutar *CheckoutComponent* klase. Slikom 4.36. prikazana je implementacija obrasca gdje je potrebno deklarirati grupu obrazaca, ugraditi alat za izradu obrazaca, stvoriti (za početak) formu za unos imena, prezime i e-maila s pripadajućim validatorima za minimalan unos riječi kod pisanja imena ili prezimena te validator za pravilan unos e-maila.

```
this.checkoutFormGroup = this.formBuilder.group({
  customer: this.formBuilder.group({
    firstName: new FormControl('',
                        [Validators.required,
                         Validators.minLength(2),
                         WebShopValidators.notOnlyWhitespace]),

    lastName: new FormControl('',
                       [Validators.required,
                        Validators.minLength(2),
                        WebShopValidators.notOnlyWhitespace]),

    email: new FormControl(theEmail,
                          [Validators.required, Validators.pattern('^[a-z0-9._%+-]+@[a-z0-9.-]+\\.[a-z]{2,4}$')])
  }),
```

Sl. 4.36. Implementacija obrasca za narudžbu proizvoda.

Prema slici 4.37. prikazan je jedan dio implementiranog obrasca za naručivanje proizvoda, odnosno dio koji korisniku daje mogućnost unos imena, prezimena i e-maila naručitelja proizvoda.



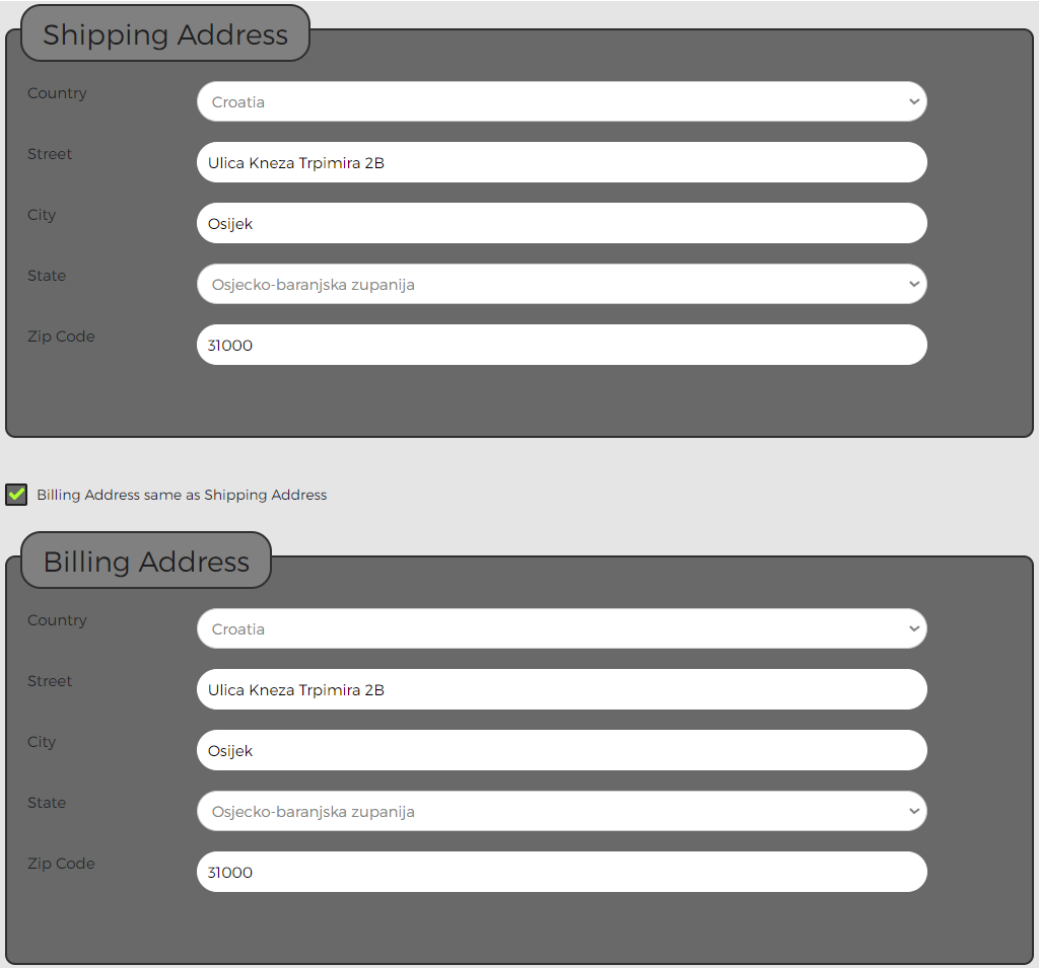
Sl. 4.37. Prikaz dijela obrasca za unos podataka o naručitelju.

Kreiranje dijela obrasca za unos podataka o adresi narudžbe započinje kreiranje tablica u bazi podataka s pripadajućim entitetima *Country* i *State*, što je već objašnjeno u poglavlju 4.1. Dalje slijedi implementacija Angular klasa za *Country* i *State* entitete s pripadajućim parametrima *id, code* i *name*. Metode za dohvaćanje *Country* i *State* entiteta implementirani su u *web-shop-form* servisu čija je implementacija objašnjena u daljnjem tekstu. Metodom *getCountries()* dohvaća se popis država. Metoda vraća *Observable* koji emitira polje objekata tipa *Country*. Za dohvaćanje podataka o izvršava se HTTP GET zahtjev prema *this.countriesUrl* URL-u te se očekuje kako se vratiti podaci u odgovoru koji je iščitan kao *GetResponseCountries*. Metoda *getStates* prima parametar *theCountryCode* koji predstavlja kod zemlje za koju želimo dohvatiti županije ili pokrajine. Povratnu vrijednost također predstavlja *Observable*, samo što umjesto emitiranja polja objekta tipa *Country*, emitira polje objekta tipa *State*. Zatim se formira URL za pretragu županija ili pokrajina i poslije toga se poziva HTTP GET zahtjev za dohvaćanje podataka. Slika 4.38. prikazuje implementaciju metoda *getCountries i getStates*.

```
getCountries(): Observable<Country[]> {

  return this.httpClient.get<GetResponseCountries>(this.countriesUrl).pipe(
    map(response => response._embedded.countries)
  );
}

getStates (theCountryCode: string): Observable<State[]> {

  // search url
  const searhStatesUrl = `${this.statesUrl}/search/findByCountryCode?code=${theCountryCode}`;

  return this.httpClient.get<GetResponseStates>(searhStatesUrl).pipe(
    map(response => response._embedded.states)
  );
}
```

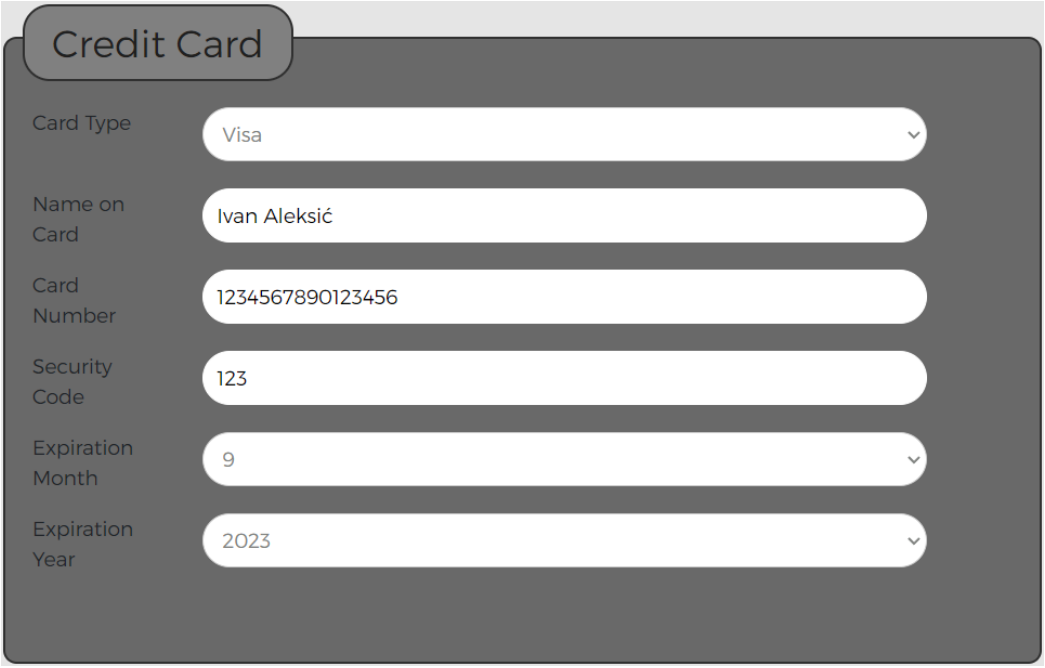
Sl. 4.38. Implementacija getCountries i getStates metoda.

Prema slici 4.39. prikazan je dio obrasca implementiran tim dokumentom, a služi za unos podataka o adresama narudžbe i naplate. Korisniku naručitelju omogućen je jednostavan unos podataka o imenu države, županije ili pokrajine, ulice, grada i poštanskog broja za narudžbu proizvoda.



Sl. 4.39. Prikaz dijela obrasca za unos podataka o adresi narudžbe i adresi naplate.

Za već istaknutu generiranu komponentu *web-shop-form* potrebno je implementirati i dio obrasca za unos podataka o kreditnoj kartici. Unutar ovog dijela obrasca nalaze se polja za odabir Visa ili Mastercard kartice, polje za unos imena, broja ili sigurnosnog broja kartice te polja za odabir mjeseca i godine isteka kartice. Prikaz dijela obrasca za unos podataka o kreditnoj kartici s koje će se preuzeti podaci za plaćanje narudžbe istaknut je slikom 4.40.



Sl. 4.40. Prikaz dijela obrasca za unos podataka o kreditnoj kartici.

5. ZAKLJUČAK

Ovim završnim realizirano je rješenje web aplikacije koja predstavlja web trgovinu za prodaju nogometne opreme. Realizacija aplikacije krenula je analiziranjem mnogobrojnih dostupnih rješenja te samim time i nastanak ideje implementacije web shopa. Implementacija je započela izradom dizajna sa svim osmišljenim dijelovima koje će web trgovina sadržavati. Okvirnim opisom ideje nastala je i shema izvedbe projekta za ovaj završni rad. Točnije, određeni su alati kojim će biti definirane funkcionalnosti i svi zahtjevi koje traži izrada ovakve vrste web aplikacije. Prema tome, za potrebe baze podataka i cijeli *backend* dio aplikacije, korišteni su sustavi PostgreSQL s programskim okvirom Spring Boot, dok je za *front-end* dio aplikacije korišten programski okvir Angular. Također, pri izradi aplikacije korišteni su programski jezici Java i TypeScript s pripadajućim programskim razvojnim okruženjima – IntelliJ IDEA i VS Code. Kroz poglavlja su objašnjena stvaranja serverske i korisničke strane. Na serverskoj strani ostvareno je stvaranje entiteta koji su vjerni prikaz tablica u bazi podataka, repozitorija za komunikaciju s bazom podataka te usluga i kontrolera za pohranu, raspored i upravljanje podacima s kojima će korisnik imati interakciju. Vizualnim dijelom aplikacije, koji ima izravnu komunikaciju i interakciju s korisnicima ostvarena je konstrukcija, izgled te ponašanje, odnosno interaktivnost podataka i informacija na web aplikaciji. Kao rezultat toga, omogućena je prijava korisnika u sustav, pretraga proizvoda prema nazivu i kategoriji, dodavanje proizvoda u košaricu te pregled iste i narudžba proizvoda iz košarice.

Povezivanjem nekoliko tehnologija u jednu cjelinu, ovim završnim radom ustanovljeno je kako je stvorena funkcionalna web aplikacija. Ova web aplikacija otvorena je i za mogućnosti implementacije novih stvari kako bi se unaprijedila interakcija s korisnicima, poput dodavanja recenzija i ocjenjivanja proizvoda od strane korisnika, pružanje preporuka i promocija proizvoda korisnicima na temelju njihovih kupovina ili pretrage proizvoda te dodavanje slika s detaljnijim prikazom proizvoda.

LITERATURA

[1] „Football Mania - Prva nogometna trgovina u Hrvatskoj“ [online]. Dostupno na: https://footballmania.hr/. [Pristupljeno: 18.9.2023.].

[2] R., Bílek, „11teamsports.hr“ [online]. Dostupno na: https://11teamsports.hr/. [Pristupljeno: 18.9.2023.].

[3] K., Arnold, J., Gosling, D., Holmes, „THE Java™ Programming Language, Fourth Edition“.

[4] „Spring Boot“ [online]. Dostupno na: https://spring.io/projects/spring-boot. [Pristupljeno: 13.9.2023.].

[5] „Spring Initializr“ [online]. Dostupno na: https://start.spring.io. [Pristupljeno: 13.9.2023.].

[6] „Angular - What is Angular?“ [online]. Dostupno na: https://angular.io/guide/what-is-angular. [Pristupljeno: 13.9.2023.].

[7] „Angular - Angular components overview“ [online]. Dostupno na: https://angular.io/guide/component-overview. [Pristupljeno: 13.9.2023.].

[8] „Angular - Understanding templates“ [online]. Dostupno na: https://angular.io/guide/template-overview. [Pristupljeno: 13.9.2023.].

[9] „Angular - Built-in directives“ [online]. Dostupno na: https://angular.io/guide/built-in-directives. [Pristupljeno: 13.9.2023.].

[10] „Angular - Introduction to services and dependency injection“ [online]. Dostupno na: https://angular.io/guide/architecture-services. [Pristupljeno: 13.9.2023.].

[11] „Angular - Introduction to modules“ [online]. Dostupno na: https://angular.io/guide/architecture-modules. [Pristupljeno: 13.9.2023.].

[12] „The starting point for learning TypeScript“ [online]. Dostupno na: https://www.typescriptlang.org/docs/. [Pristupljeno: 13.9.2023.].

[13] „HTML basics - Learn web development | MDN“ [online], 29-kol-2023. Dostupno na: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics. [Pristupljeno: 13.9.2023.].

[14] „CSS Introduction“ [online]. Dostupno na: https://www.w3schools.com/css/css_intro.asp. [Pristupljeno: 13.9.2023.].

[15] „Bootstrap 5 Get Started“ [online]. Dostupno na: https://www.w3schools.com/bootstrap5/bootstrap_get_started.php. [Pristupljeno: 13.9.2023.].

[16] „Features - IntelliJ IDEA“ [online]. Dostupno na: https://www.jetbrains.com/idea/features/. [Pristupljeno: 13.9.2023.].

[17] „Documentation for Visual Studio Code“ [online]. Dostupno na: https://code.visualstudio.com/docs. [Pristupljeno: 13.9.2023.].

[18] „PostgreSQL: About“ [online]. Dostupno na: https://www.postgresql.org/about/. [Pristupljeno: 13.9.2023.].

SAŽETAK

Naslov: Web trgovina za prodaju nogometne opreme

Web trgovina za prodaju nogometne opreme omogućuje korisniku narudžbu proizvoda iz domene sporta nogometa, pružajući pritom jednostavan i pregledan izbor proizvoda. Praktičnim dijelom rada stvoren je funkcionalan sustav web trgovine sa serverskom i korisničkom stranom. Sustav je realiziran primjenom Spring Boot programskog okvira u programskog razvojnom okruženju IntelliJ IDEA na serverskoj strani, odnosno Angular programskog okvira na korisničkoj strani. Za upravljanje i spremanje podataka korištena je PostgreSQL baza podataka. Priloženim teorijskim dijelom rada opisane su primijenjene programske tehnologije i njihova uloga u realizaciji aplikacije te implementacija programskog rješenja. Spajanjem više softverskih tehnologija stvorena je platforma web trgovine s mogućnostima nadogradnje s novim značajkama.

Ključne riječi: Angular, narudžba proizvoda, nogometna oprema, Spring Boot, web trgovina

ABSTRACT

Title: Web store for the sale of football equipment

The web store for the sale of football equipment enables the user to order products from the domain of the sport of football, while providing a simple and clear product selection. A functional web store system with a server-side and user-side was created as a practical part of work. The system was implemented using the Spring Boot framework in the IntelliJ IDEA development enviroment on the server-side and the Angular framework on the user-side. PostgreSQL database was used for data management and storage. The theoretical part of the work describes the applied software technologies and their role in the realization of application's and the implementation of the software solution. By combining multiple software technologies, a web store platform was created, with the possibility of upgrading with new features.

Keywords: Angular, football equipment, product order, Spring Boot, web store

ŽIVOTOPIS

Ivan Aleksić rođen je 30. kolovoza 1999. godine u Novoj Gradiški, Hrvatska. Pohađao je osnovnu školu Okučani u Okučanima. Nakon završene osnovne škole 2014. godine, upisuje Elektrotehničku i Ekonomsku školu, Nova Gradiška, smjer tehničar za računalstvo, koju završava 2018. godine. Obrazovanje nastavlja na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija, preddiplomski stručni studij računarstvo. Tijekom studija pohađao je stručnu praksu u tvrtki Inovativni trendovi. Od 2022. radi kao programer u tvrtki Neos d.o.o.