IZRADA WEB TRGOVINE KORIŠTENJEM ANGULAR I JAVA SPRING BOOT TEHNOLOGIJE

Jurlina, Marin

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijska tehnologija

MARIN JURLINA

Z A V R Š N I R A D

IZRADA WEB TRGOVINE KORIŠTENJEM ANGULAR I JAVA SPRING BOOT TEHNOLOGIJE

Split, lipanj 2023.

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijska tehnologija

Predmet: Sigurnost računala i podataka

Z A V R Š N I R A D

Kandidat: Marin Jurlina

Naslov rada: Izrada web trgovine korištenjem Angular i Java Spring Boot tehnologije

Mentor: Lada Sartori, viši predavač

Split, lipanj 2023.

SADRŽAJ

Sažetak

Razvoj interneta popratio je i razvoj internetskih trgovina koje su poslužile kao alternativa fizičkom odlasku u trgovinu te omogućile jednostavno, brzo i učinkovito odabiranje i kupovanje željenih proizvoda. Paralelno time, u sve digitalnijem društvu potrebna je mogućnost lake nabave raznih elektroničkih komponenti, od sagrađenih računala i laptopa, uređaja osjetljivih na dodir te procesora, grafičkih i mrežnih kartica do prodaje softvera tipa licence za Windowse. Web aplikacija Tyfer upravo se specijalizirala za prodaju takve opreme. Korisnik se može registrirati, ukoliko nije registriran te prijaviti ukoliko jest. Prijavom korisnik dobiva mogućnost pregleda prethodnih narudžbi. Naposlijetku, korisnik može iz udobnosti svoga doma naručiti bilo koju komponentu koja mu je potrebna i doći će mu u razumnom roku (kroz 3-5 radna dana) neovisno u kojem dijelu Hrvatske se nalazi.

Poslužiteljski dio aplikacije je izrađen u Java Spring Bootu, radnom okviru (engl. *framework*) koji je među najkorištenijima pri izradu web aplikacija zbog lake uporabe, jednostavnijeg pisanja i čitanja kôda i smanjenju *boilerplate* kôda. Klijentski dio izrađen u Angularu. Korisničko sučelje je izrađeno pomoću HTML (*HyperText Markup Language*), CSS (*Cascading Style Sheets*) i Bootstrap biblioteke (engl. *library*) koja omogućuje jednostavno implementiranje raznih dizajnerskih komponenti.

Ključne riječi: Angular, Java Spring Boot, web aplikacija

Summary

Creating a web shop using Angular and Java Spring Boot technology

Development of the internet was accompanied by the development of online stores that served as an alternative to physically going to the store. They made choosing and shopping simpler, faster, and more efficient. At the same time, an increasingly digital society requires an easier obtainment of various electronic components from pre-built computers and laptops, touch-sensitive devices, processors, graphic and network cards, to the sale of various software such as Windows license keys. The Tyfer web application specialized in selling such equipment. Users can order any component they need, view order history, register if they're not already registered and login if they are registered. All from the comfort of their home.

Server side of the application is made using Java Spring Boot, one of the most useful frameworks in building web applications due to its' ease of use, simpler and more readable code, and reduction of boilerplate code. The client side of the application is made using Angular. User interface is made using HTML (HyperText Markup Language), CSS (Cascading Style Sheets) and Boostrap library, which enables easier implementation of diverse design components.

Keywords: Angular, Java Spring Boot, web application

1. Uvod

Sve veći broj kupaca odlučuje se na kupnju preko interneta, dok se istovremeno sve veći broj prodavača odlučuje otvoriti internetsku trgovinu čineći fizičku trgovinu suvislom. Zbog toga je internet trgovina postala jednom od najvažnijih i najkorištenijih načina opskrbe raznim potrepštinama. Prednosti otvorenja internet trgovine su uočljive bez profesionalne analize troškova: omogućuje poslovanje bez poslovnog prostora, nisu potrebna velika početna ulaganja, moguće je proširiti ponudu uz mali rizik odbijanja, manja konkurencija među trgovinama istog ili sličnog sadržaja te samom prisutnošću na internetu omogućuje se otkrivanje sadržaja velikom broju potencijalnih kupaca.

Cilj ovoga rada je izrada web aplikacije s tematikom internet trgovine specijalizirane za prodaju informatičke opreme, koja će korisniku ove aplikacije omogućiti plasiranje svojih proizvoda na internetsko tržište. S druge strane, kupcu omogućuje odabir i kupovinu željenih proizvoda te jednostavno kartično plaćanje. Aplikacija se sastoji od administrativnog i korisničkog dijela. U administrativnom dijelu, administrator je u mogućnosti dodavati i uređivati proizvode, pomagati u slučaju nejasnoća korisnika te pregledavati korisničke račune i njima upravljati. Korisnik može pretraživati proizvode ili ih odabrati putem izbornika kategorija. Opcionalno se može prijaviti kako bi aplikacija zapamtila njegove prethodne kupovine, iako prijava i/ili registracija nisu obavezne za dovršetak kupovine.

Završni rad se sastoji od 5 cjelina, od kojih se prva odnosi na uvodni dio rada u kojem se opisuje motivacija izrade aplikacije. Druga cjelina se odnosi na detaljnije opisivanje korištenih tehnologija. Treća cjelina rada je detaljni opis praktične izrade aplikacije. Četvrta cjelina se odnosi na zaključivanje rada, dok se u petoj cjelini popisuju sve korištene literature pri izradi ovoga rada.

2. O tehnologijama

U ovoj cjelini opisane su tehnologije korištene pri izradi ovoga rada. Rad se sastoji od poslužiteljskog dijela za koji je korištena Java Spring Boot [1] i Firebase te korisničkog dijela za koji je korišten Angular ([2], [3]), HTML ([4]), CSS ([5]), Bootstrap ([6], [7]). Baze podataka su spremane u MySQL ([8], [9]) i Firestore.

2.1. Java Spring Boot

Java Spring Boot (u daljnjem tekstu *Spring Boot)* je popularni radni okvir otvorenog kôda kojeg održava kompanija Pivotal. Spring Boot pruža Java programerima platformu koja im pomaže s automatskom konfiguracijom produkcijskog stanja Spring aplikacije. Cilj Spring Boota je olakšati izradu samostalnih (*standalone*) i proizvodno spremnih (*production-ready*) aplikacija.

Glavne značajke Spring Boota uključuju:

1) Automatska konfiguracija: Spring Boot automatski konfigurira razne aspekte aplikacije tako što konfigurira bazu podataka, sigurnost, web sloj i druge ovisno o potrebama aplikacije. Ova značajka uvelike ubrzava razvoj aplikacije i smanjuje ručno namještanje.
2) Samostalni izvršni JAR (engl. *executable JAR*): Spring Boot omogućuje zapakiranje Java aplikacije kao jednu samostalnu izvršnu JAR datoteku, što olakšava distribuciju i pokretanje aplikacije.
3) Upravljanje ovisnostima (engl. *dependency*): Spring Boot koristi Maven ili Gradle alate za upravljanje ovisnostima. To olakšava dodavanje i upravljanje bibliotekama i drugim vanjskim modulima potrebnim za razvoj aplikacije.
4) Ugrađeni poslužitelj (engl. *embedded server*): Spring Boot dolazi s ugrađenim poslužiteljem što omogućuje brzo pokretanje aplikacije na lokalnom poslužitelju. Najčešće su to integrirani Tomcat ili Jetty poslužitelji.

2.2. Angular

Angular je popularni radni okvir otvorenog kôda za razvoj web aplikacija. Razvio ga je Google i koristi se za izgradnju modernih jednostraničnih aplikacija (*Single Page Applications – SPA*) i dinamičnih korisničkih sučelja. Angular je pisan u TypeScriptu, „proširenom“ jeziku JavaScripta te se on koristi kao glavni razvojni jezik. TypeScript donosi statičku tipizaciju i dodatne značajke koje olakšavaju razvoj i održavanje aplikacija.

Glavne značajke i neki koncepti Angulara su:

1) Komponente (eng. *components*): Angular koristi koncept komponenti za izgradnju korisničkog sučelja. Komponente su modularne jedinice koje kombiniraju HTML, CSS i TypeScript datoteke kako bi definirale korisničke elemente, funkcionalnosti i ponašanje.
2) Direktive: Angular ima razne ugrađene direktive koje omogućuju manipulaciju DOM-om, dinamičko prikazivanje ili skrivanje elemenata, iteraciju kroz podatke. Također moguće je definirati vlastite direktive prema potrebama projekta.
3) Upravljanje ovisnostima i njihovo ubrizgavanje: Angular ima mehanizam ubrizgavanja ovisnosti (engl. *dependency injection*). Taj mehanizam olakšava upravljanje ovisnostima između komponenata i servisa. To pruža bolju modularnost, testiranje i ponovnu iskoristivost kôda
4) Navigacija (engl. *routing*): Angular omogućuje navigaciju između komponenata unutar jednostranične aplikacije. To olakšava upravljanje stanjem aplikacije i omogućuje korisnicima glatko prelaženje između različitih dijelova aplikacije.
5) Servisi: Angular podržava koncept servisa za izvršenje poslovne logike, dohvaćanje podataka s poslužitelja ili obavljanje drugih operacija koje nisu specifične za korisničko sučelje. Servisi se, kao i ovisnosti mogu ubrizgati u komponente i koristiti za dijeljenje podataka i funkcionalnosti.

2.3. MySQL

MySQL je jedan od najpopularnijih sustava za upravljanje bazama podataka (DBMS). Glavni autor DBMS-a je Ulf Michael Widenius, koji je ujedno i jedan od osnivača MySQL AB. MySQL je relacijski sustav za upravljanje bazama podataka koji se temelji na strukturi podataka tablica, a podaci se organiziraju u redove i stupce. Podržava standardni SQL (*Structured Query Language*) jezik za upravljanje bazama podataka. MySQL se može koristiti i kao poslužiteljski sustav za različite vrste aplikacija, uključujući web aplikacije, poslovne aplikacije i druge sustave koji zahtijevaju pohranu i upravljanje podacima.

Glavne značajke MySQL-a uključuju:

1) Tablice: Podaci u MySQL-u su organizirani u tablice. Svaka tablica ima definirane stupce s odgovarajućim tipovima podataka. Tablice se koriste za pohranu i organizaciju podataka.
2) SQL: MySQL koristi SQL jezik za upravljanje bazama podataka. SQL se koristi za stvaranje, mijenjanje i upravljanje podacima u tablicama. Njegovim korištenjem moguće je kreirati upite na bazu (engl. *Query*), umetanje, dohvaćanje, ažuriranje i brisanje podataka.
3) Ovisnosti: MySQL podržava definiranje ovisnosti među tablicama, što omogućuje povezivanje podataka te kreiranje kompleksnih operacija kao što su JOIN-i za spajanje podataka iz više tablica.
4) Transakcije: Transakcijske operacije su operacije koje osiguravaju dosljednost i integritet podataka. Ujedno i omogućuju grupiranje više operacija u jednu logičku cjelinu koja se izvršava kao jedna jedinica.
5) Sigurnost: MySQL pruža mehanizme za sigurnost podataka. Omogućuje upravljanje korisnicima (engl. *user management*), njihovim pravima pristupa i upravljanja te šifriranjem podataka kako bi se osigurala njihova povjerljivost i integritet.

2.4. HTML, CSS, Bootstrap

HTML (*HyperText Markup Language*) je jezik za označavanje koji se koristi za strukturiranje sadržaja na web stranicama. To je osnovni jezik weba i koristi se za definiranje elemenata. HTML koristi oznake (engl. *tag*) za označavanje elemenata što omogućuje Internet pregledniku da interpretira i prikaže stranicu prema tim oznakama.

Neke od oznaka su: `<title>`, `<p>`, `<h1-5>`, `<a>`, `<hr>`, `<br>` i drugi.

CSS (*Cascading Style Sheets*) je stilski jezik koji se gotovo uvijek koristi zajedno s HTML-om. CSS se koristi za detaljno uređivanje izgleda i ponašanja stranice. Uglavnom se koristi za primjenu različitih stilova, kao što su boje, fontovi, razmaci, granice, slike i animacije. Dobrim manipuliranjem CSS-a, postiže se profesionalni izgled stranice, koja zatim može lako privući kupce i zadržati ih na samoj stranici. Za čitljivost kôda preporuča se odvojiti CSS u zasebnu datoteku.

Bootstrap je jedan od najpopularnijih CSS radnih okvira. On pruža set predložaka, stilova i komponenata za brz i jednostavan razvoj responzivnih web stranica. Bootstrap sadrži već gotove CSS stilove i JavaScript komponente koji se mogu iskoristiti za izgradnju modernih i mobilno prilagodljivih korisničkih sučelja.

Ukratko, HTML se koristi za strukturiranje sadržaja web stranica, CSS se koristi za njihovo stiliziranje i dizajniranje, dok je Boostrap CSS okvir koji olakšava izgradnju modernih, mobilno prilagodljivih i atraktivnih web sučelja.

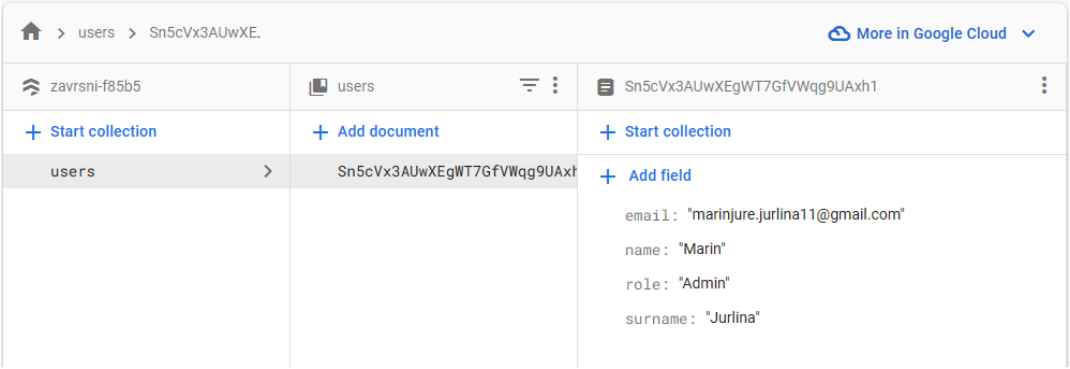
2.5. Firebase & Firestore

Firebase je platforma za razvoj aplikacija u oblaku koju je razvio Google. Pruža različite usluge i alate koji olakšavaju razvoj, testiranje, upravljanje i posluživanje mobilnih i web aplikacija.

Firestore je jedna od usluga koju pruža Firebase platforma. Firestore je fleksibilna, skalabilna i dokumentno-orijentirana baza podataka u oblaku koja se koristi za pohranu podataka aplikacije. Podržava razne platforme, uključujući web, mobilne uređaje i servere, što ga čini pogodnim za razvoj višeplatformskih aplikacija.

Neki ključni koncepti i značajke Firestorea i Firebasea uključuju:

1) Baze podataka i dokumenti: Firestore organizira podatke u kolekcijama koje sadrže dokumente. Dokumenti su strukturirani u JSON formatu i mogu sadržavati različite vrste podataka kao što su tekst, brojevi, nizovi i objekti. Dokumenti se identificiraju jedinstvenim putem (putanjom) u bazi podataka prikazanom na slici 1.



Slika 1: Prikaz Kolekcija -> Dokument

2) Sinkronizacija u stvarnom vremenu: Firestore podržava sinkronizaciju podataka u stvarnom vremenu između klijentske i poslužiteljske strane. To znači da se promjene podataka od strane jednog korisnika automatski reflektiraju u promjene na drugim korisničkim uređajima bez potrebe za osvježavanjem ili ručnom sinkronizacijom.
3) Autentikacija: Firebase omogućuje administrativnu podršku kao što je registracija i prijavljivanje. Omogućuje registraciju i prijavljivanje putem maila i lozinke, kao i putom posrednika kao što su Google, Twitter i GitHub. Ujedno omogućuje i kreiranje dvostrukog faktora autentikacije kao što je slanje potvrdnog SMS-a na mobitel ili slanje mail poruke za aktivaciju računa.

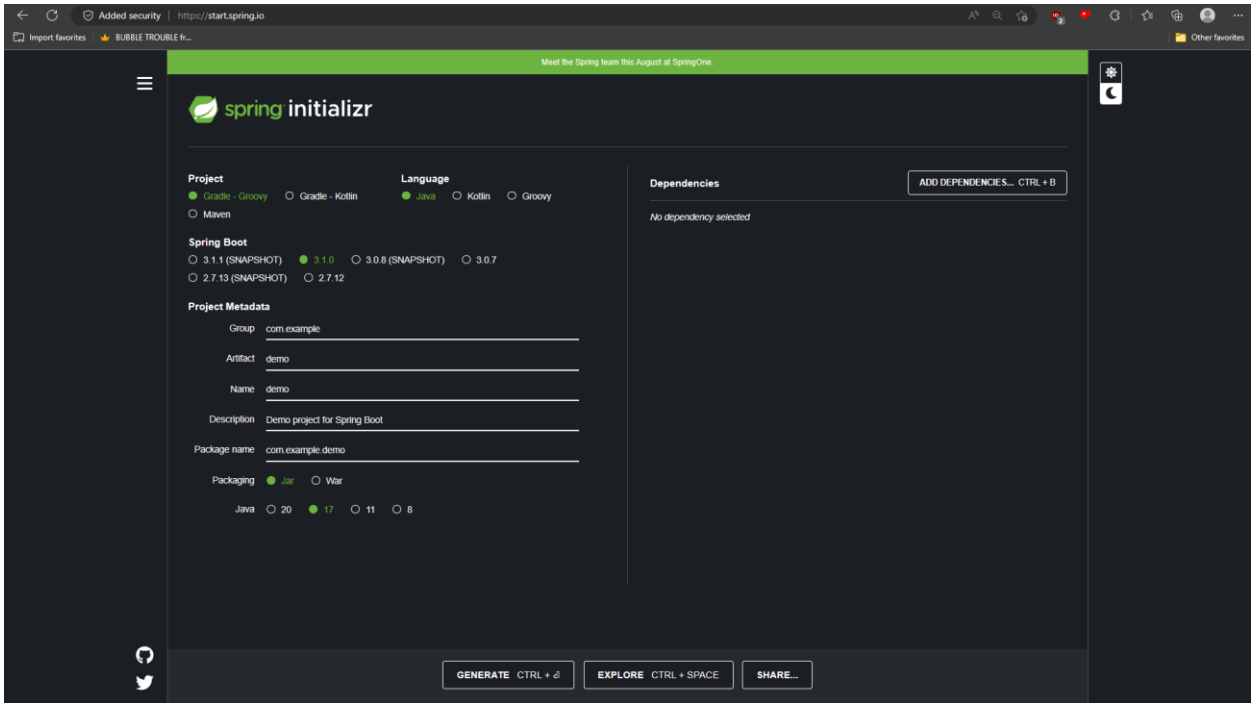
4) Pregled posjećenosti stranice: Firebase omogućuje pregled posjete stranici. Za to nije potrebno instalirati ili nadograditi stranicu, jer sve potrebno dolazi s Firebaseom. Moguće je pregledati dnevnu aktivnost korisnika kao i mjesečnu. Ujedno je moguće i pregledati koliko je poslano, a koliko ovjereno SMS poruka.

3. Praktična izvedba

U ovoj cjelini opisuje se izvedba završnog rada od postavljanja projekta korištenjem online alat za kreiranje gotovog predloška poslužiteljske strane, generiranja inicijalnih podataka u bazi podataka, dohvaćanja i radu s podacima, do prikazivanju istih korisniku.

3.1. Postavljanje projekta

U ovoj pod cjelini, opisuje se postavljanje projekta. Postavljanje projekta vrši se putem online alata na stranici: https://start.spring.io prikazan na slici 2.



Slika 2: Početna stranica start.spring.io *izvor: start.spring.io*

Na toj stranici postoje razni odabiri koji se tiču odabira vrste projekta [„Project“], programskog jezika [„Language“], verzija Spring Boota [„Spring Boot“], dodatni podaci projekta [„Project Metadata“] te odabir ovisnosti [„Dependencies“]. U dodatnim podacima projekta, može se odabrati grupa, artefakt, ime, opis te se pod „*Package name“* prikaže kako bi izgledala struktura projekta ukoliko bi se generirao. Za potrebe završnog rada, projekt će biti nazvan: „com.marin.zavrsniecommerce“. Takvim nazivom dobiva se na unikatnosti projekta te se čak i postiže razumljivost postavki projekta. Pod *„Packaging“* se odabere Jar, a pod Java verzija se odabere najkasnija verzija (*verzije Jave prilikom kreiranja projekta i pisanja dokumentacije se mogu razlikovati*).

Nakon što se dodaju podaci o projektu, vrijeme je za dodavanje ovisnosti. Ovisnosti se dodavaju klikom na *„ADD DEPENDENCIES“* te se iz odabira odaberu ovisnosti koje bi najviše koristile za ovaj projekt. Za potrebe projekta koriste se: Lombok, Spring Web, Rest Repositories, MySQL Driver, JDBC API, Spring Data JPA i Spring Data JDBC. Neke od ovisnosti mogu biti višak te se, ukoliko se ne koriste, mogu izbaciti iz projekta brišući ih iz datoteke `pom.xml`. Svaka od navedenih ovisnosti biti će ukratko opisana.

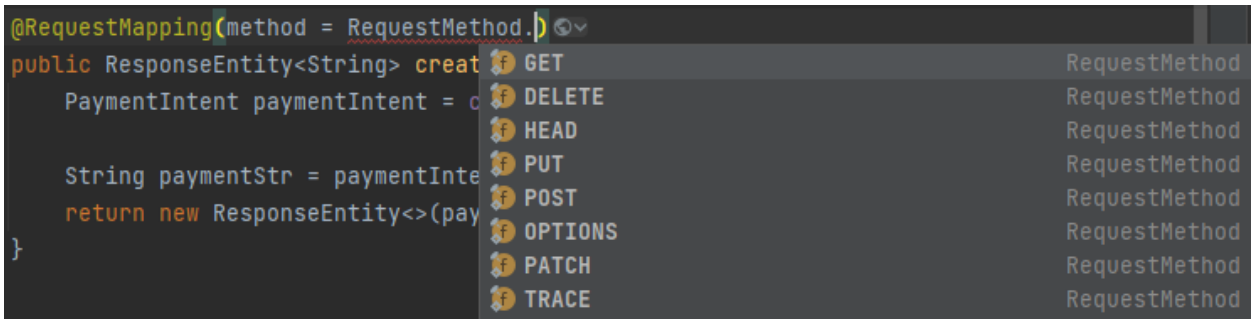
LOMBOK:

Lombok je Spring Boot ovisnost, koja pomaže pri smanjenju *boilerplate* kôda. *Boilerplate* kôd je kôd koji je višak. To su najčešće *getteri* i *setteri*, konstruktori bez argumenata, konstruktori s argumentima. Lombok ima još puno raznih „zamjenica“ za ostatke kôda. Oni će biti prikazani u detaljnijoj analizi kôda.

SPRING WEB:

Spring Web je Spring Boot ovisnost, koja automatski posloži Spring Boot aplikaciju na način da uključi izgradnju RESTful web aplikacije koristeći Spring MVC. Ujedno ima i ugrađeni Tomcat server koji se diže istovremeno s aplikacijom. Spring MVC je abrevijacija za Spring *Model-View-Controller*, gdje su „*Model*“ podaci koji opisuju entitet (najčešće se nalazi u bazi podataka), „*View*“ implementira poslovnu logiku, dok se u „*Controller*“ kreiraju metode

koje se zovu preko URL-ova. Postoji više anotacija koje se mogu koristiti pri pozivanju metoda preko URL-ova, a to su: `@GetMapping`, `@PostMapping`, `@PutMapping`, `@DeleteMapping` i drugi… Moguće je zamijeniti svaku od navedenih anotacija jednom zajedničkom, a to je: `@RequestMapping(method = RequestMethod.[„unesite vrijednost poziva"])`, gdje je vrijednost poziva *GET, POST, PUT* i drugi, već prethodno navedena. Vrijednosti poziva su prikazane na slici 3.



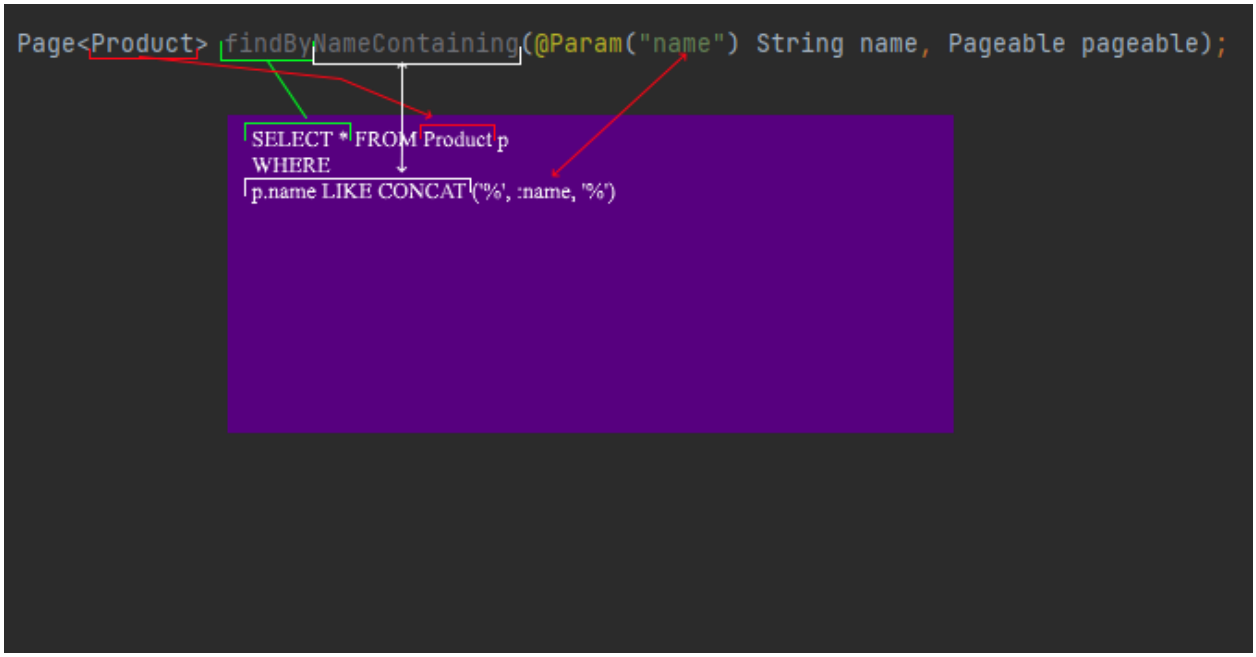
Slika 3: Izgled `@RequestMapping(method = RequestMethod.[])` anotacije

REST REPOSITORIES:

Rest Repositories je Spring Boot ovisnost, koja dodaje mogućnost repozitorija. Repozitorij, u ovom slučaju, služi kao veza između aplikacije i baze podataka. Za povezivanje zaslužna je `@Repository` anotacija. Ona je specijalizacija `@Component` anotacije, jer isto omogućuje odvijanje CRUD operacija.

`@Repository` anotacija se implementira pomoću sučelja (engl. *interface*), tako da se implementira `JpaRepository<T,ID>` gdje je *T* ime klase na koju se referencira, a *ID* tip ID varijable te klase. Najčešće to je tip *Long*.

`@Repository` anotacija ujedno i omogućuje pisanje skripti preko konvencije imenovanja. Naime, ukoliko je potrebno nešto sortirati, iščitati, prebrojati, potrebno je upisati odgovarajući ključ, što će Spring Boot repozitorij prepoznati i ovisno koja se *follow-up* metoda definira:



Slika 4: Primjer naziva metode u jednom od repozitorija

Na slici 4, definirana je metoda `findByNameContaining(@Param(„name“) String name, Pageable pageable);`. Ona je zaslužna za kreiranje SQL *Querya* napisanog na ljubičastoj ploči. `findBy` je ekvivalentno `SELECT *`, *tip* `Product Productu p`, Parametar `name` korespondira `@Param(„name“)` dok je `NameContaining` ekvivalent `p.name LIKE CONCAT()`. Korištenjem ovakvih metoda, postiže se jednostavnije manipuliranje podacima u bazi, tj. olakšava se njihovo dohvaćanje, sortiranje, čitanje i još druge operacije.

MySQL Driver, JDBC API, Spring Data JPA i Spring Data JDBC

MySQL Driver je softver komponenta koja omogućuje aplikaciji povezivanje i interakciju s MySQL bazom podataka. Ona nudi osnovne funkcionalnosti za utvrđivanje veze, slanja upita i dohvaćanje podataka iz baze podataka.

JDBC API (*Java Database Connectivity*) je standardni Java API koji nudi set sučelja i klasa za povezivanje na bazu. Omogućuju Java aplikacijama pristup raznim sustavima baza koristeći uniformni set poziva metoda.

Spring Data JPA je dio većeg, Spring Data *radnog okvira*, koji nudi visoki nivo apstrakcije i pojednostavljenog programskog modela za pristup podacima u Java aplikaciji. JPA (*Java Persistence API*) je standardna specifikacija za ORM (*Object Relational Mapping*) u Javi. Nadograđuje se na JPA i nadodaje set jednostavno uporabljivih apstrakcija, anotacija i alata.

Spring Data JDBC je još jedan modul unutar Spring Data radnog okvira koji nudi alternativni pristup pristupu podacima umjesto JPA.

3.2. Generiranje inicijalnih podataka za bazu podataka

Prilikom kreiranja novog projekta, baza podataka je prazna. Stoga, potrebno je kreirati neke inicijalne podatke, koji se kasnije trebaju prikazati korisniku na korisničkom sučelju. Za potrebe završnog rada, korišten je MySQL Workbench jer on omogućuje grafičko sučelje kao i pisanje i pokretanje SQL skripti. Čitavo generiranje inicijalnih podataka izvršavano je putem SQL skripti zbog seljenja baze s računala na laptop, stoga je jednostavnije pokrenuti skriptu nego ručno namještati podatke unutar baze.

Skripte se sastoje od više dijela. Stvaranja baze podataka, stvaranja tablice i definiranje tipova podataka unutar iste te na kraju i samo popunjavanje tablice podacima. Podaci su uzeti sa stranice: https://www.instar-informatika.hr. Baza podataka ima šest (6) kategorija od kojih svaka ima po četiri (4) proizvoda, čineći bazu podataka velikom dvadeset četiri (24) proizvoda. Kategorije su: „*LAPTOPS*“, „*DESKTOPS*“, „*MOBILE PHONES*“, „*TABLETS*“, „*SMART WATCHES*“ i „*COMPONENTS*“ prikazane na ispisu 1.

```
-- ----------------------------------------------------
-- Categories
-- ----------------------------------------------------

INSERT INTO product_category(category_name) VALUES ('LAPTOPS');
INSERT INTO product_category(category_name) VALUES ('DESKTOPS');
INSERT INTO product_category(category_name) VALUES ('MOBILE PHONES');
INSERT INTO product_category(category_name) VALUES ('TABLETS');
INSERT INTO product_category(category_name) VALUES ('SMART WATCHES');
INSERT INTO product_category(category_name) VALUES ('COMPONENTS');
```

Ispis 1: Prikaz skripte za generiranje kategorija u bazi

Navođenje svih proizvoda bi zauzelo previše teksta, stoga isti neće biti nabrojani u ovom radu, iako će neki biti prikazani na ispisima 2 i 3.

```
-- -----------------------------------------------------
-- LAPTOPS
-- -----------------------------------------------------
INSERT INTO product (sku, name, description, image_url, active, units_in_stock,
unit_price, category_id, date_created)
VALUES ('LAPTOP-WORK-0000', 'Notebook Acer TravelMate Spin B3, NX.VN2EX.005, 11.6" FHD IPS, Intel Celeron N4120 up to 2.6GHz, 4GB DDR4, 64GB SSD, Intel UHD Graphics 600, Win 10', 'Work Laptop 0',
'assets/images/products/acer-prijenosno-racunalo-tmb311rn-31-c0u1-nxvn2ex005-69131-0852726_1.png'
,0,100,262.20,1, NOW());

INSERT INTO product (sku, name, description, image_url, active, units_in_stock,
unit_price, category_id, date_created)
VALUES ('LAPTOP-WORK-0001', 'Notebook Acer Extensa 15, NX.EG9EX.01W, 15.6" FHD IPS, AMD Ryzen 3 3250U up to 3.5GHz, 8GB DDR4, 256GB SSD, AMD Radeon Graphics, no OS', 'Work Laptop 1',
'assets/images/products/notebook-acer-aspire-3-nxeg9ex01w-156-fhd-ips-amd-ryzen-3-32-76018-0001277336_1.png'
,1,100,434.81,1, NOW());

INSERT INTO product (sku, name, description, image_url, active, units_in_stock,
unit_price, category_id, date_created)
VALUES ('LAPTOP-WORK-0002', 'Notebook Dell Vostro 3525, 15.6" FHD IPS, AMD Ryzen 5 5625U up to 4.3GHz, 8GB DDR4, 512GB NVMe SSD, AMD Radeon Graphics, Win 11 Pro', 'Work Laptop 2',
'assets/images/products/notebook-dell-vostro-3525-n1005vnb3525em-0001272364_1.png'
,1,100,935.48,1, NOW());

INSERT INTO product (sku, name, description, image_url, active, units_in_stock,
unit_price, category_id, date_created)
VALUES ('LAPTOP-WORK-0003', 'Notebook Asus ExpertBook B1, B1500CEAE-BQ3072, 15.6" FHD IPS, Intel Core i5 1135G7 up to 4.2GHz, 8GB DDR4, 256GB NVMe SSD, Intel Iris Xe Graphics, no OS', 'Work Laptop 3',
'assets/images/products/notebook-asus-expertbook-b1-b1500ceae-bq-asu-b1500-bq027r_1.png'
,1,100,29.99,1, NOW());
```

Ispis 2: Prikaz skripte za stvaranje podataka za kategoriju „*LAPTOPS*“

Uz kategorije i proizvode, u bazu podataka dodane su i sve države svijeta, uključujući i njihove regije te su i kreirane tablice vezane uz naručivanje proizvoda. Te tablice su: *adresa*, *kupac*, *narudžba* te sam proizvod koji se referencira na proizvod naveden gore u ispisu 2 i njemu slične (*ostatak proizvoda*).

```
--
-- Table structure for table `orders`
--
CREATE TABLE `orders` (
  `id` bigint NOT NULL AUTO_INCREMENT,
  `order_tracking_number` varchar(255) DEFAULT NULL,
  `total_price` decimal(19,2) DEFAULT NULL,
  `total_quantity` int DEFAULT NULL,
  `billing_address_id` bigint DEFAULT NULL,
  `customer_id` bigint DEFAULT NULL,
  `shipping_address_id` bigint DEFAULT NULL,
  `status` varchar(128) DEFAULT NULL,
  `date_created` datetime(6) DEFAULT NULL,
  `last_updated` datetime(6) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `UK_billing_address_id` (`billing_address_id`),
  UNIQUE KEY `UK_shipping_address_id` (`shipping_address_id`),
  KEY `K_customer_id` (`customer_id`),
  CONSTRAINT `FK_customer_id` FOREIGN KEY (`customer_id`) REFERENCES
`customer` (`id`),
  CONSTRAINT `FK_billing_address_id` FOREIGN KEY (`billing_address_id`)
REFERENCES `address` (`id`),
  CONSTRAINT `FK_shipping_address_id` FOREIGN KEY (`shipping_address_id`)
REFERENCES `address` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
```

Ispis 3: Prikaz skripte za stvaranje tablice „*orders*“

3.3. Poslužiteljski dio

U ovoj cjelini bit će opisana izrada poslužiteljskog djela aplikacije, što uključuje entitete, kontrolere, DAO-ve (*Data Access Object)*, DTO-ve (*Data Transfer Object)* te *config* datoteku zaslužnu za otkrivanje identifikacijskih brojeva tamo gdje je to potrebno. Detaljno će biti analizirano par važnijih implementacija kao što su registracija, prijava te dodavanje i uređivanje proizvoda.

3.3.1. Dohvaćanje podataka

Dohvaćanje podataka u poslužiteljskom dijelu rađeno je pomoću Java Spring Boota. Spring Boot omogućuje jednostavno dohvaćanje podataka preko *repositorya* koristeći se

jednom od prethodno definiranih metoda. Naročito je potrebno paziti na imenovanje metoda, jer su osjetljive na velika i mala slova. Konvencija nalaže da se metode, koje se pozivaju u Spring Boot *repositoryu*, nazivaju *Camel-Case*, to jest, da se prvo slovo metode piše malim slovom, a zatim svaka druga riječ spoji s prethodnom i kapitalizira. To bi rezultiralo ovakvim nazivom metode: `findByCategoryID()`. IDE će sam prepoznati što se želi upisati pa će izborom u padajućem izborniku ponuditi metodu koju se želi pozvati, prikazanog na slici 5:

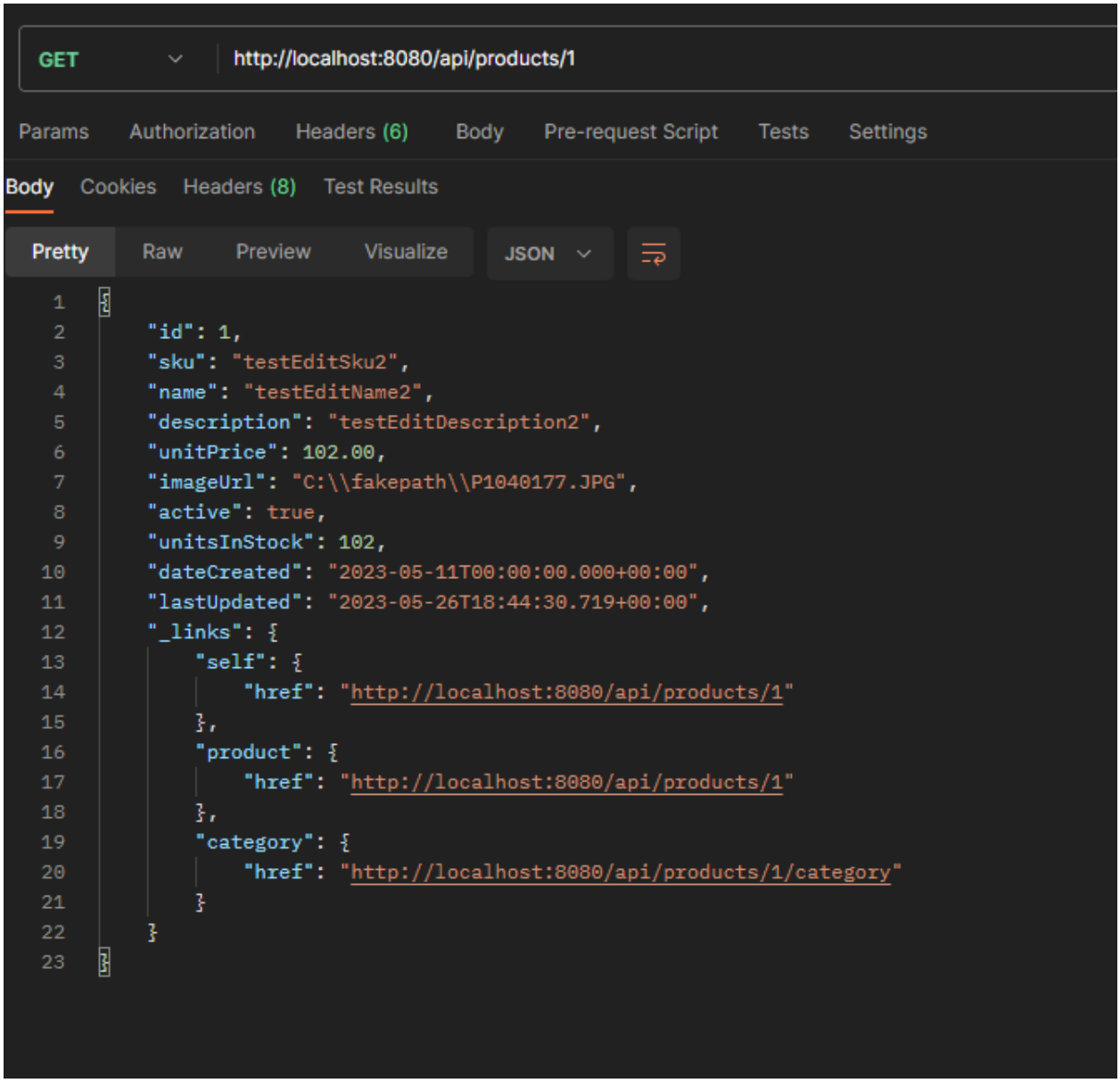
Slika 5: Prikaz automatske preporuke IDE-a za metodu find…

Nakon što se odredi koja će se metoda koristiti, potrebno je imenovati upit tako, da Spring Boot može prepoznati o čemu se radi, kako da to izvrši te po čemu da sortira. Sve se to odvija bez potrebe korisnikovog znanja (slika 4). Kad se definira metoda u *repositoryu*, nije potrebna anotacija `@Query()`, jer se upit izvršava prema konvenciji imenovanja metode.

Nakon kreiranja metode unutar *repositorya*, u većini slučajeva je potrebno definirati metodu unutar kontrolera i anotirati ju s `@GetMapping` ili `@RequestMapping(method = RequestMethod.GET)` kako bi prilikom pozivanja URL-a kontroler mogao pozvati metodu unutar *repositorya*. U ovom slučaju, to nije potrebno jer Spring Boot automatski omogućuje dohvaćanje entiteta bez definiranja popratnih metoda koje bi trebale omogućiti njihovo dohvaćanje. Ukoliko se želi dohvatiti neki podatak, koji nije definiran pod entitetima ili ako se taj podatak nalazi izvan baze podataka, potrebno je definirati i implementirati metodu unutar repositorya, kontrolera i ukoliko je potrebno, servisa. (*kontroleri* bi trebali biti napisani samo jednom linijom, a to je pozivanje servisa i metode unutar njih)

Kako je u ovom poglavlju rečeno kako se ne treba definirati metoda unutar kontrolera, potrebno je poslati *GET* zahtjev u aplikaciju Postman, koja će primiti neki argument te ga proslijediti prema bazi, koja će izvršiti zadani upit te poslati tražene podatke, koje će program poslati putem *ResponseBodya*. Postman je aplikacija koja može slati razne zahtjeve (*GET, POST, PUT, DELETE…*) te prikazati odgovor nastao izvršavanjem upita i povratnom vrijednosti metode.

U aplikaciji Postman za dohvaćanje podataka odabere se *GET* zahtjev iz padajućeg izbornika u adresnoj „tražilici“. Nakon što se odabere ispravni zahtjev, u „tražilicu“ se upise URL resursa. Na primjer, ukoliko je potrebno dohvatiti podatak o određenom proizvodu ID-a broj 1, u URL se upiše: http://localhost:8080/api/products/1 gdje je: http protokol za prijenos hiperteksta, localhost naziv računala koji se odnosi na trenutni uređaj ili lokalni mrežni priključak, 8080 je broj priključne točke, `/api/product/1` je putanja ili krajnja točka (engl. *endpoint*) URL-a. To predstavlja određeni resurs, u ovom slučaju dohvaćanje proizvoda s ID-em 1. „`/api`“ je definiran u `application.properties` kao ekstenzija URL-a zbog lakšeg raspoznavanja od ostalih putanja. U tijelu odgovora dobije se JSON koji prikazuje sve podatke dohvaćenog objekta. U ovom slučaju to je id, SKU (*stock keeping unit*), ime, opis, jedinična cijena i druge popratne vrijednosti. Uz njih, Postman pošalje i dodatne opcije po kojima se može pretraživati kao što su: pregled kategorije proizvoda i pregled svih proizvoda prikazanog na slici 6.



Slika 6: Prikaz odgovora prilikom upućivanja *GET* zahtjeva

3.3.2. Upisivanje podataka

U ovom poglavlju bit će opisano upisivanje (*dodavanje*) podataka u bazu preko *POST* zahtjeva. *POST* zahtjev se vrši isto preko Postmana, kao i *GET* zahtjev, uz malu promjenu. Naime, uz URL koji se upisuje unutar adresne „tražilice“, potrebno je definirati *RequestBody* koji će sadržavati sve potrebne podatke o entitetu kojeg se dodaje. Uz podatke unutar Postmana, potrebno je definirati i anotaciju `@RequestBody` unutar metode kontrolera. Prilikom spremanja proizvoda u bazu preko kontrolera, potrebno je pozvati metodu `save()` u

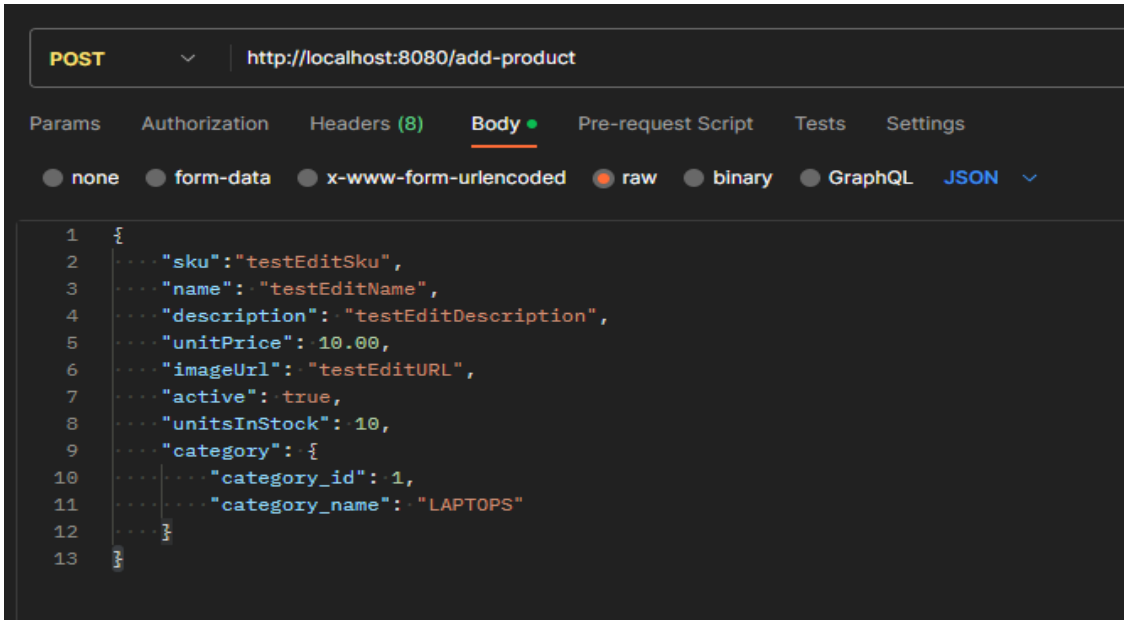
repozitoriju zaslužnom za taj entitet (*u ovom slučaju Product*). Izgled metode je prikazan na ispisu 4.

```
@PostMapping("/add-product")
    public Product createNewProduct(@RequestBody Product p){
        return productService.createNewProduct(p);
    }
```

Ispis 4: Prikaz metode *POST* zahtjeva

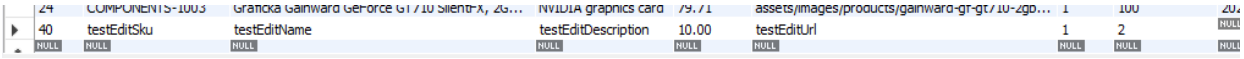
Suprotno *GET* zahtjevu, *POST* zahtjevu nije potrebna zasebna metoda unutar repositorya, jer se *POST* zahtjev izvršava putem *RequestBody*a.

Kako bi se upisao entitet u bazu podataka, unutar Postmana, u „*Body*“ potrebno je, unutar vitičastih zagrada (JSON format) definirati sve atribute tog entiteta osim ID-a jer se on autoinkrementira. Klikom na „*SEND*“, „*Body*“ se šalje na putanju „*add-product*“, gdje se aktivira metoda `createNewProduct()` koja prima „`RequestBody`“ (ekvivalent „*Body*“ unutar Postmana), koji zatim unutar metode u `productServiceu` postavi zadane vrijednosti od „*RequestBody*“ u odgovarajuće vrijednosti entiteta *Product*. Izgled tijela zahtjeva prikazan je na slici 7.



Slika 7: Kartica s pregledom tijela zahtjeva unutar Postmana

Nakon uspješnog dodavanja proizvoda, moguće ga je pregledati u bazi te kao što je moguće ustvrditi, proizvod je dodan u bazu što je prikazano na slici 8:



Slika 8: Prikaz novog proizvoda u bazi

Potvrđivanjem proizvoda dodanog u bazu, završeno je dodavanje entiteta u bazu podataka.

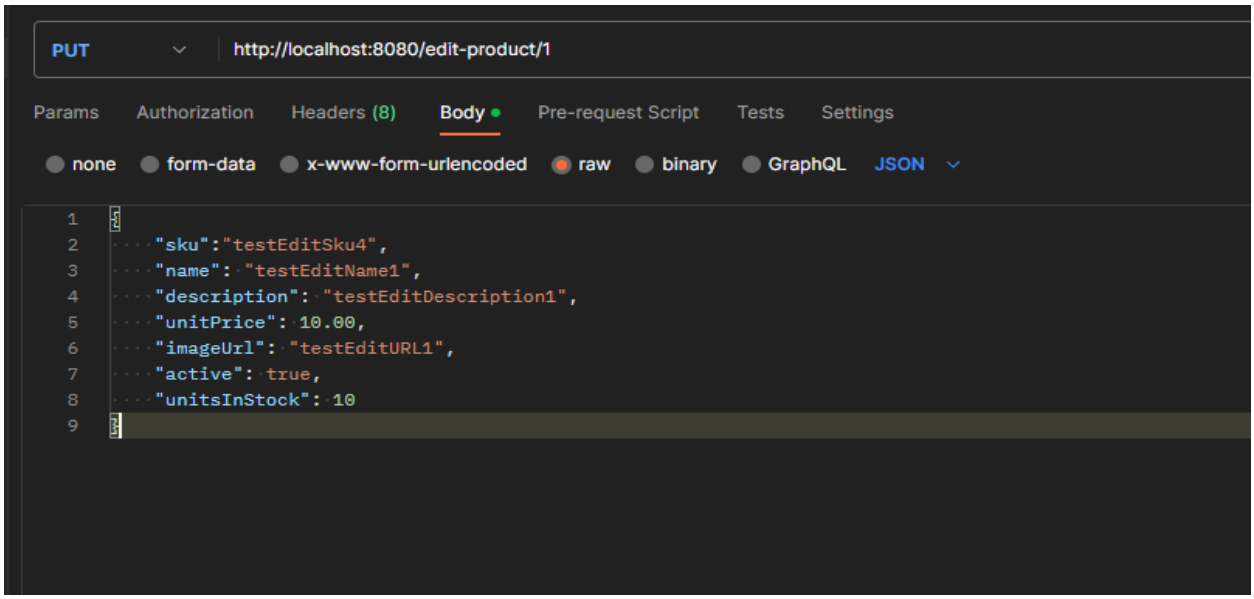
3.3.3. Promjena podataka

U ovom poglavlju bit će opisano kako se određenom proizvodu, odabranog preko identifikacijskog broja (ID), mijenjaju svi ili samo pojedini atributi. Ova radnja se odvija isto preko Postmana jer taj program nudi i *PUT* zahtjev te ujedno kao i *POST* zahtjev, nije potrebno kreirati posebne metode, to jest, nije ih potrebno definirati jer se *POST* zahtjev izvršava preko „`RequestBody`“.

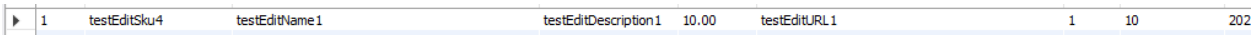
Kreiranjem nove metode, `editProduct`, unutar Java Spring Boota koja se ujedno označava oznakom: `@PutMapping(„/edit-product/{id})`. Isto kao i u *POST* zahtjevu, ovdje se koristi „`/edit-product`“ kako bi se razaznala *PUT* metoda od *POST* ili *GET* metode. (iako se samo može promijeniti vrsta zahtjeva, a putanja se ostavi ista, ali u ovom projektu nije se koristilo mnoštvo različitih metoda pa su se mogle dodavati nove putanje). Zatim, nakon otkrivanja nove krajnje točke, potrebno je definirati koji se točno proizvod želi izmijeniti. Za identifikaciju proizvoda pri pozivanju putanje zaslužan je „*{id}*“. „*{id}*“ je pozicijska varijabla unutar pozivanja putanja. To označava neko mjesto gdje se očekuje nekakva varijabla. Konačan poziv putanje bi izgledao kao prikazan na slici 9:

Slika 9: Izgled putanje za poziv *PUT* zahtjeva

Odabirom ispravne putanje i popunjavanjem tijela zahtjeva, mijenja se proizvod pri identifikacijskom broju 1. Takav upit, a zatim i prikaz u bazi podataka bi izgledao prikazan na slici 10, dok je izmijenjen proizvod prikazan na slici 11:



Slika 10: Prikaz kartice s tijelom zahtjeva za promjenu proizvoda



Slika 11: Prikaz izmijenjenog proizvoda u bazi podataka

3.4. Korisnički dio

Korisnički dio aplikacije izrađen je u Angularu, JavaScript radnom okviru pogodnom za izradu dinamičkih jednostraničnih web aplikacija te će u ovom dijelu biti opisana izrada korisničkog sučelja, dohvaćanje, ispisivanje te izmjena podataka, registracija i prijava korisnika te izgled mogućnosti prijavljenog korisnika pod ulogom administratora i ulogom člana. Ujedno će biti opisana i demonstrirana mogućnost detaljnijeg pregleda proizvoda, dodavanje proizvoda u košaricu, pregled košarice te na kraju plaćanje i potvrda o plaćanju.

3.4.1. Dohvaćanje i ispisivanje podataka

Bilo koja operacija u Angularu, bilo to dohvaćanje, ažuriranje ili brisanje, zahtijeva da se prvo generira pripadajuća komponenta koja će sadržavati sve potrebne metode, operacije i varijable koje će biti korištene za izradu neke određene funkcionalnosti. U ovom slučaju, dohvaćanje podataka bit će demonstrirano na prikazu svih proizvoda filtriranih po kategorijama. Kako postoji šest (6) kategorija, a svaka kategorija ima po četiri (4) proizvoda, sveukupna količina proizvoda iznosi dvadeset četiri (24).

Kako je prije navedeno, prva operacija koja se mora izvršiti pri početku dohvaćanja podataka jest generiranje potrebnih komponenti. To se može uraditi upisivanjem naredbe `ng generate component [putanja]/[ime-komponente]` ili skraćeno `ng g c [putanja]/[ime-komponente]`. Putanja i ime komponente su obavezni dijelovi naredbe `ng generate` neovisno o čemu se generira. Putanja i ime komponente moraju biti definirane neovisno generira li se komponenta, klasa, servis ili neki drugi tip.

Nakon uspješno generirane komponente, potrebno je upisati putanju pomoću koje će se pozvati prethodno generirana komponenta. To se radi u datoteci `app.module.ts`. To je datoteka koja sadrži podatke o putanja na stranici, deklaracije, uvoze i pružatelje servisa. Putanje čine većinu informacija veznih uz samu aplikaciju. Njih je potrebno definirati od najgeneralnije do najspecifičnije putanje. To znači da će putanja, koja se ujedno naziva i *wildcard*, biti ona koja se podudara s bilo kojom putanjom. To su najčešće putanje koje nemaju smisla (krivo upisane, nepoznate ili nepostojeće putanje). Ukoliko se u aplikaciju upiše jedna takva putanja, tada bi, najpoželjnije bilo, da aplikacija prebaci na sljedeću ili najpoznatiju putanju. U ovom slučaju to bi bilo na prikaz svih proizvoda, filtriranih po kategorijama. Nakon te *wildcard* putanje, idu one putanje koje su specifičnije od ostalih. To su uglavnom one putanje koje u sebi sadržavaju neke parametre, kao što je najčešće to identifikacijski broj (*ID*), dok nakon njih idu sve ostale. Iako je moguće razbacati putanje po nizu putanja (`routes: Routes = []`), to nije dobra praksa zato što bi se, u slučaju neispravnog rada aplikacije, teško došlo do saznanja gdje se nalazi pogreška i popravljanje greške bi bilo znatno otežano.

Definiranje putanje, unutar niza putanja izgleda:

a) Ključnom riječi `path` definira se putanja `kao string`, unutar jednostrukih navodnika
b) Definira se komponenta koju podudara odabrana putanja
c) Definira se `canActivate` metoda koja onemogućuje neovlašten pristup putanji, a posljedično i komponenti (opcionalno)
d) Ukoliko upisana putanja ne odgovara niti jednoj od navedenih ruta u datoteci `app.module.ts`, potrebno je definirati prethodno navedenu *wildcard* putanju. Ona se označuje dvama zvjezdicama *(**)* te se definira mjesto na koje će prebaciti korisnika u slučaju upisa nepoznate putanja.

Na sljedećim ispisima, bit će prikazan izgled niza putanja, kao i izgled deklaracija i uvoza komponenti i modula.

```
const routes: Routes = [
  { path: 'card-help', component: CardInfoFormHelpComponent},
  { path: 'form-help', component: CheckOutFormHelpComponent},
  { path: 'order-history', component: OrderHistoryComponent},
  { path: 'verify-email', component: VerifyEmailComponent},
  { path: 'reset-password', component: ForgotPasswordComponent},
  { path: 'login', component: LoginComponent},
  { path: 'register', component: RegisterComponent},
  { path: 'checkout', component: CheckoutComponent},
  { path: 'add-product', component: AddProductComponent, canActivate:
[RoleGuard]},
  { path: 'edit-product/:id', component: EditProductComponent, canActivate:
[RoleGuard]},
  { path: 'about-us', component: AboutUsComponent},
  { path: 'contact-us', component: ContactUsComponent},
  { path: 'help', component: HelpComponent},
  { path: 'cart-details', component: CartDetailsComponent},
  { path: 'products/:id', component: ProductDetailsComponent },
  { path: 'search/:keyword', component: ProductListComponent },
  { path: 'category/:id', component: ProductListComponent },
  { path: 'category', component: ProductListComponent },
  { path: 'products', component: ProductListComponent },
  { path: '', redirectTo: '/products', pathMatch: 'full' },
  { path: '**', redirectTo: '/products', pathMatch: 'full' },
];
```

Ispis 5: Prikaz putanja unutar *app.module.ts* datoteke

Na ispisu 5 može se uočiti kako se definiraju putanje za pristup raznim komponentama. Ujedno se mogu vidjeti i putanje kojima običan korisnik, onaj koji nema administratorska prava (ulogu) ne može pristupiti. To su putanje `add-product` i `edit-product/:id`. Na kraju niza, mogu se uočiti dvije putanje koje imaju drugačiji izgled od ostatka putanja. To su posljednje dvije putanje. Predzadnja putanja označuje prazan *string*, to jest, prazan ostatak putanje nakon protokola, naziva računala i priključne točke. Na primjer, ova putanja bi se pozvala nakon što bi se pozvala sljedeća putanja: `http://localhost:4200/`. Ova putanja nema dodatnih argumenata, stoga aplikacija ne zna što se točno od nje traži pa vrati preusmjeravanje na listu svih proizvoda dodajući na putanju „`/products`“. Ta putanja će pozvati `ProductListComponent` komponentu te će korisniku prikazati sve proizvode filtrirane po kategorijama.

```
@NgModule({
  declarations: [
    AppComponent,
    ProductListComponent,
    ProductCategoryMenuComponent,
    SearchComponent,
    ProductDetailsComponent,
    AddProductComponent,
    CartStatusComponent,
    CartDetailsComponent,
    AboutUsComponent,
    ContactUsComponent,
    HelpComponent,
    CheckoutComponent,
    LoginComponent,
    RegisterComponent,
    ForgotPasswordComponent,
    VerifyEmailComponent,
    UpdateProductComponent,
    EditProductComponent,
    OrderHistoryComponent,
    CheckOutFormHelpComponent,
    CardInfoFormHelpComponent,
  ]
```

Ispis 6: Prikaz deklaracija svih korištenih komponenti

Na ispisu 6 prikazane su sve deklaracija svih korištenih komponenti. Niz `declarations` se koristi za deklariranje svih komponenti, direktiva i *„prosljeđivanica*“ (*engl. pipe)* koji su dio NgModula. Ovaj niz omogućuje korištenje ugrađenih direktiva kao što su `ngFor` ili `ngIf` unutar tog modula. Isto tako, ukoliko se žele koristiti „*prosljeđivanice*“, mora se definirati komponenta u kojoj se ista želi koristiti, kao što su prosljeđivanica za valutu, vrijeme, veliko slovo i još mnoge druge.

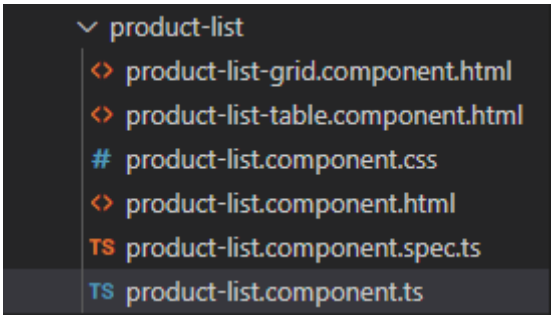
```
imports: [
    BrowserModule,
    HttpClientModule,
    RouterModule.forRoot(routes),
    NgbModule,
    AngularFireDatabaseModule,
    AngularFirestoreModule,
    FormsModule,
    AngularFireModule.initializeApp(environment.firebaseConfig),
    provideFirebaseApp(() => initializeApp(environment.firebaseConfig)),
    provideAuth(() => getAuth()),
    provideFirestore(() => getFirestore()),
    provideStorage(() => getStorage()),
    ReactiveFormsModule,
    AuthModule.forRoot({
      domain: 'dev-gumhee1zxgvdei6o.us.auth0.com',
      clientId: 'Uaxgrtoga5oWWJYeZ3xRjDNisMMhNDL9',
      authorizationParams: {
        redirect_uri: window.location.origin
      }
    }),
  ]
```

Ispis 7: Prikaz uvoza u datoteci *app.module.ts*

Na ispisu 7, prikazan je izgled niza imena `imports`. To je niz u kojemu se definiraju svi važni moduli koji se koriste za poboljšanje funkcionalnosti aplikacije. To su `BrowserModule`, `FormsModule` ili `HttpClientModule`. Navedeni moduli omogućuju funkcionalnosti kao što su upravljanje formama, HTTP komunikacija i funkcionalnosti vezane uz Internet preglednik. Funkcionalnosti vezane uz Internet preglednik su one funkcionalnosti koje omogućuju upravljanje DOM-om (*Document Object Model)*, prikazivanju komponenti i upravljanje životnim vijekom aplikacije.

Nakon uređene datoteke `app.module.ts`, to jest, dodavanja potrebnih putanja, modula i komponenti, potrebno je kreirati komponentu za određenu putanju. To se izvodi u datoteci koja je bila generirana prilikom izvođenja `ng generate` naredbe. To će kreirati komponentu `ProductListComponent`.

Komponenta `ProductListComponent`, prikazana na slici 12, je skupina datoteka koju čine datoteke ekstenzija: `html`, `css`, `ts`, `spec.ts`. Datoteka ekstenzije `.ts` nije relevantna za aplikaciju jer ona samo sadrži meta podatke o komponenti. Datoteke ekstenzija `.html`, `.css` *i* `.ts` su datoteke u kojima se piše kôd. U datoteci ekstenzije `.html` se piše kôd pomoću kojega se prikazuju dohvaćeni podaci. U toj datoteci se piše običan HTML kôd uz poneki CSS kôd, ali preporuka je da se CSS kôd odijeli od HTML-a radi lakšeg čitanja koda. Uz to, nije potrebno definirati lokaciju CSS datoteke u HTML datoteci jer Angular automatski omogućuje komunikaciju datoteka unutar iste komponente prikazanoj na slici 12.



Slika 12: Izgled `product-list` komponente

Ulaskom u datoteku `product-list.component.ts`, može se uočiti kôd zaslužan za dohvaćanje podataka, njihovo manipuliranje te prikazivanje podataka korisniku na korisničko sučelje. Najprije se definiraju varijable koje će se koristiti prilikom pozivanja putanje i metode zaslužne za dohvaćanje proizvoda. To je varijabla `products: Product[] = [];`. Koja sadržava niz imena products tipa `Product[]` koji je instanciran na prazan niz. On će u sebi sadržavati sve dohvaćene proizvode. Zatim se u konstruktoru definiraju potrebni servisi, u ovom slučaju `productService: ProductService`. Ovaj servis sadrži sve potrebne metode koje se pozivaju unutar `.ts` datoteke. Zatim se unutar `ngOnInit()` metode definira `this.route.paramMap` metoda definirana na ispisu 8:

```
this.route.paramMap.subscribe(() => {
      this.listProducts();
    });
```

Ispis 8: Prikaz `this.route.paramMap` poziva

Metoda `ngOnInit` je jedna od metoda životnog vijeka aplikacija. To je metoda koja se pokreće prilikom pokretanja aplikacija te osigurava da se svaka metoda, varijabla ili bilo kakvo ponašanje aplikacija pokrene i završni dok se ne završi učitavanje same aplikacije. U ovom slučaju, potrebno je osigurati da se učitaju svi proizvodi i prikažu na korisničko sučelje prije nego li se sama aplikacija otvori. To omogućuje korisniku trenutan prikaz proizvoda, bez osjećaja da je aplikaciji potrebno dulje učitavanje sadržaja da se prikaže korisniku. Unutar `ngOnInit` metode, poziva se `this.route.paramMap.subscribe(() => {this.listProducts();});` metoda prikazana na ispisu 8. Ova metoda se može raščlaniti na više dijelova:

1) `this.route.paramMap`: `'this.route'` predstavlja objekt `ActivatedRoute` koji omogućuje pristup informacijama trenutne putanje, dok je `'paramMap'` svojstvo `'this.route'` koje šalje novu vrijednost svakog puta kada se parametar putanje promijeni.
2) `subscribe(() => {this.listProducts();})`: `'subscribe();'` predstavlja metodu koja se zove na `'paramMap'` povratnu vrijednost koja je tipa *Observable*. Kao argument prima *call-back* metodu koja se izvršava svakog puta kada se parametri putanje promijene.
3) `this.listProducts();`: to je metoda koja se poziva unutar iste komponente i provjerava u kakvom je statusu pretraživanje proizvoda. Postoje dva statusa pretraživanja, tražilica ili listanje početne stranice.

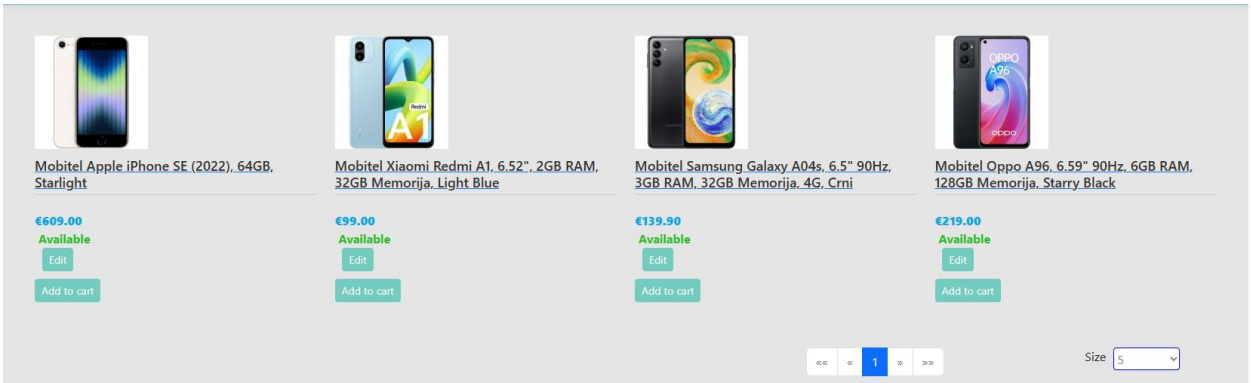
Metoda `listProducts()` (ispis 9) poziva metodu koja provjerava postoji li u trenutnoj verziji ove putanje (engl. *snapshot*) neka ključna riječ (*engl. keyword*) koja vrijednost sprema u varijablu `this.searchMode` definiranoj iznad metode. Zatim ulazi u `if-else` blok koji provjerava vrijednost `this.searchMode` varijable. Varijabla je tipa *boolean*, to jest, provjerava `True-False`. Ukoliko je vrijednost `True`, prikazuju se proizvodi po ključnoj riječi tražilice, a ukoliko je

vrijednost `False`, prikazuju se proizvodi kao lista bez tražilice. Rezultati pretrage bez i s tražilicom su prikazani na slikama 13 i 14.

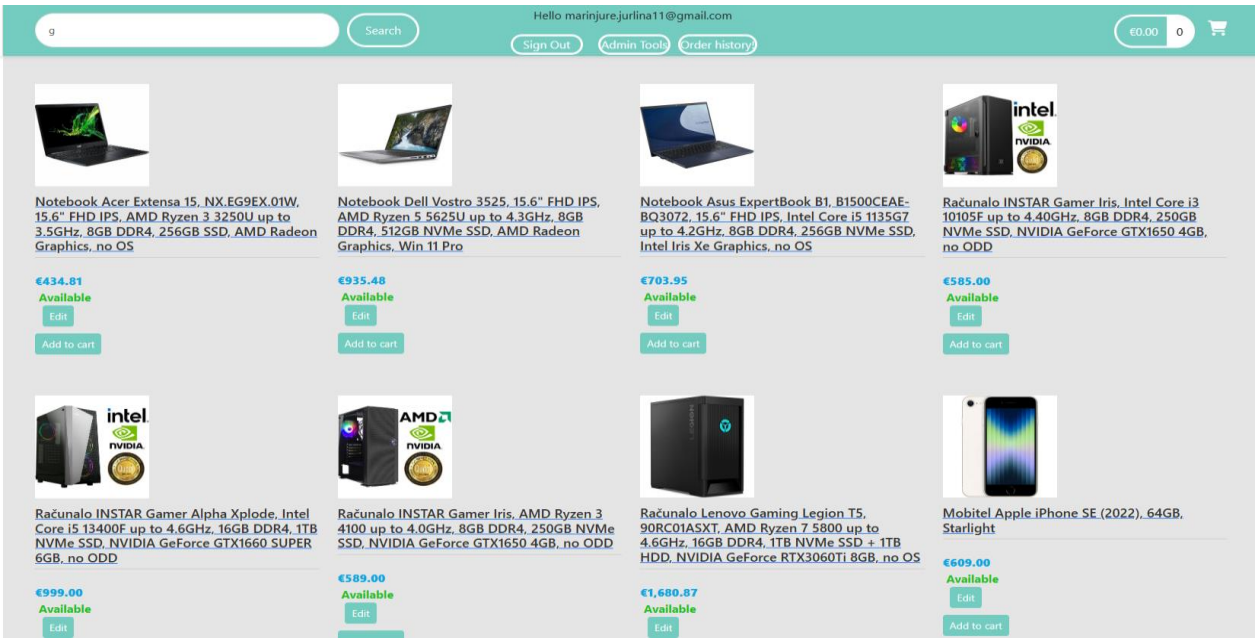
```
listProducts() {
    this.searchMode = this.route.snapshot.paramMap.has('keyword');

    if (this.searchMode) {
      this.handleSearchProducts();
    } else {
      this.handleListProducts();
    }
  }
```

Ispis 9: Prikaz `listProducts()` metode



Slika 13: Prikaz kartice sa listom proizvoda (mobiteli)



Slika 14: Prikaz kartice sa listom proizvoda s ključnom riječju „g“

3.4.2. Registracija korisnika

U ovom poglavlju bit će opisana registracija korisnika. Registracija korisnika obavlja se putem Firebase autentikacije. Firebase autentikacija omogućuje registraciju korisnika putem obrasca, a korisnik se identificira mailom i lozinkom.

Prilikom izvođenja operacija autentikacije, registracije ili prijave, potrebno je uvesti ispravne module u datoteci `app.module.ts`, tako da je unutar niza `imports`, potrebno definirati `AngularFireModule` te koristeći `initializeApp` inicijalizirati postavke firebase projekta. Postavke firebase projekta nalaze se u environment datoteci. U environment datoteci, potrebno je ubaciti svojstvo `firebaseConfig` koji u sebi sadrži *api* ključ, domenu, identifikacijski kôd projekta te još par ključeva i lokacija na kojima se spremaju podaci o registriranim korisnicima. Ti podaci, prikazani na ispisu 10, nisu vidljivi vanjskim operaterima, to jest, vidljivi su samo administratoru firebase sučelja.

```
export const environment = {
  production: false,
  firebaseConfig : {
    apiKey: "AIzaSyB9o--5mOgcLBvVwTCzY1_f8NqZAN0qM1A",
    authDomain: "zavrsni-f85b5.firebaseapp.com",
    projectId: "zavrsni-f85b5",
    storageBucket: "zavrsni-f85b5.appspot.com",
    messagingSenderId: "901445042281",
    appId: "1:901445042281:web:9c6952d4ea3eb0234d22f8"
  },
  stripePublishableKey:
"pk_test_51IKE8xG1Sr7bz9t6Bhg0rcrGhQUbgA8MLb05c4wrvDWZezw2LrEXwawDDFdMr6AS7
ruNnIhblTCw1Mv8Vku3zdS600QTtnX7C4"
};
```

Ispis 10: Prikaz *environments.ts* datoteke

Nakon integracije potrebnih svojstava i atributa *firebase* autentikacijskog sustava, potrebno je definirati metode, forme i sučelje pomoću kojega će se korisnici moći registrirati. Registracija, a na kraju i prijava, omogućuju korisniku praćenje svoje prijašnje narudžbe. Registracija se obavlja u svojoj zasebnoj komponenti te unutar autentikacijskog servera koji je definiran naredbom `ng g s services/auth`. Unutar registracijske komponente nalaze se varijable zaslužne za upisivanje korisnika u autentikacijsku bazu, nalazi se metoda `register()`

koja provjerava podudaraju li se lozinke te naposlijetku zove metodu `register()` koja se nalazi unutar autentikacijskog servisa. Metoda `register()` unutar autentikacijskog servisa kao parametre prima: mail, lozinku, ime i prezime zbog bilježenja istih unutar *firestore database*, to jest, baze podataka usluge *firestorea.* Komponenta, metoda za registriranje i forma prikazani su na ispisima 11, 12 i 13.

```
export class RegisterComponent implements OnInit {
  name: string = '';
  surname: string = '';
  email: string = '';
  password: string = '';
  repeatPassword: string = '';
  role: string = '';

  constructor(private auth: AuthService) { }

  ngOnInit(): void {
  }

  register(){
    if (this.password != this.repeatPassword){
      alert("Please, repeat your password correctly!");
      return;
    }

    this.auth.register(this.email, this.password, this.name, this.surname);

    this.email = '';
    this.password = '';
    this.repeatPassword = '';
  }

}
```

Ispis 11: Prikaz metode `register()` unutar `register.component.ts`

```
register(email: string, password: string, name: string, surname: string) {
    this.fireAuth.createUserWithEmailAndPassword(email,  password).then(res
=> {
      if (res && res.user){
        this.fireServices.collection('users').doc(res.user.uid).set({
          name: name,
          surname: surname,
          email: email,
          role: 'Member'
        });
      }
      alert("Registration successful!");
      this.sendEmailForVerification();
      alert("Verification email has been sent to your email!");
      this.router.navigate(['login']);
    }, err => {
      alert(err);
      this.router.navigate(['register']);
    })
  }
```

Ispis 12: Prikaz `register()` metode unutar `auth.service.ts`

```
<div class="mb-3">
                <label for="Name" class="form-label">Name</label>
                <input      type="text"      class="form-control"      aria-
describedby="Name" name="Name"
                [(ngModel)]="name">
            </div>
            <div class="mb-3">
                <label for="Surname" class="form-label">Surname</label>
                <input      type="text"      class="form-control"      aria-
describedby="Surname" name="Surname"
                [(ngModel)]="surname">
            </div>
            <div class="mb-3">
                <label     for="EmailRegister"     class="form-label">Email
address</label>
                <input      type="email"      class="form-control"      aria-
describedby="emailRegister" name="EmailRegister"
                [(ngModel)]="email">
            </div>
```

Ispis 13: Prikaz registracijske forme unutar `register.component.html`

Nakon što se korisnik registrirao, račun mu nije aktiviran, stoga mora aktivirati račun putem poveznice koja mu dođe na email adresu. Ukoliko korisnik pokuša pristupiti bilo kojoj stranici, a da nije potvrdio svoju mail adresu, aplikacija ga preusmjeri natrag na putanju da potvrdi svoj mail. Tek nakon što korisnik potvrdi svoj mail, ima pristup čitavoj stranici osim onih elemenata koji su zaštićeni samo za autorizirane korisnike. Metoda za potvrdu mail adrese je prikazana na ispisu 14.

```
sendEmailForVerification() {
    this.fireAuth.authState.subscribe(user => {
      if (user) {
        user.sendEmailVerification().then((res: any) => {
          this.router.navigate(['verify-email']);
        }, (err: any) => {
          alert("Something went wrong! Please check if you've entered correct
email!");
        })
      }
    });
  }
```

Ispis 14: Prikaz metode za potvrdu email adrese

3.4.3. Prijava korisnika

Prijava korisnika obavlja se jednakim koracima kao i registracija, uz izmjenu da se u formi ne traže nikakvi podaci osim maila i lozinke, osim u slučaju da se korisnik odluči prijaviti putem Googleovog prijavnog servisa.

Proces prijave započinje prikazivanjem forme neprijavljenom korisniku. Od njega se očekuje da u formu upiše svoj mail i lozinku ili ako želi, može se prijaviti putem Googleovog prijavnog servisa. Ukoliko je korisnik zaboravio svoju lozinku, može je promijeniti uz uvjet da mail adresa koju upiše u polje za mail, postoji. Ukoliko mail postoji, na mail, koji je korisnik upisao u polje, šalje se potvrdni mail za promjenu lozinke. Nakon uspješne promjene lozinke, korisnik se mora pokušati prijaviti s novo odabranom lozinkom. Nakon što je korisnik kliknuo „ENTER“, tada se pozove `login()` metoda unutar `login.component.ts`, koja provjeri jesu li mail i lozinka uneseni. Ako mail i lozinka nisu uneseni, u prozorčiću se, na vrhu monitora,

ispiše poruka koja moli korisnika da unese mail i/ili lozinku. Ukoliko je sve u redu, metoda zove drugu metodu `login()` koja se nalazi unutar autentikacijskog servisa te kao parametar prima mail i lozinku. Ta metoda zatim zove `signInWithEmailAndPassword()` metodu koja unutar sebe prima mail i lozinku. Zatim se, ukoliko je prijava uspješna, u lokalno spremište sačuva token s vrijednošću `true`. Taj token je zaslužan za pamćenje trenutno prijavljenog korisnika. Nakon toga, provjerava se je li mail korisnika potvrđen te ukoliko je, na vrhu monitora se ispisuje poruka da je korisnik uspješno prijavljen te ga se preusmjeri na stranicu s proizvodima., a ukoliko nije, na mail mu se šalje mail za potvrdu svoga maila i preusmjeri na stranicu koja mu kaže da potvrdi svoju email adresu. U slučaju nekakve pogreške, stranica ispiše poruku o grešci te korisnika preusmjeri na stranicu za prijavu. Forma, komponenta i metoda za prijavu su prikazani na ispisima 15, 16 i 17.

```
<div class="mb-3">
            <label      for="inputEmailLogIn"      class="form-label">Email
address</label>
            <input       type="email"       class="form-control"       aria-
describedby="emailLogIn" name="inputEmailLogIn"
              [(ngModel)]="email">
          </div>
          <div class="mb-3">
            <label for="passwordLogIn" class="form-label">Password</label>
            <input      type="password"      class="form-control"      aria-
describedby="passwordLogIn" name="passwordLogIn"
              [(ngModel)]="password">
          </div>
          <button        type="submit"        class="btn        btn-primary"
(click)="login()">Log in</button>
```

Ispis 15: Prikaz forme za prijavu

```
export class LoginComponent implements OnInit {

  email: string = '';
  password: string = '';

  constructor(private auth: AuthService) { }

  ngOnInit(): void {
  }

  login(){
    if (this.email == ''){
      alert("Please enter an email");
      return;
    }

    if (this.password == ''){
      alert("Please enter a password");
      return;
    }

    this.auth.login(this.email, this.password);

    this.email = '';
    this.password = '';
  }

  signInWithGoogle() {
    this.auth.loginGoogle();
  }

  signInWithTwitter() {
    this.auth.loginTwitter();
  }

  signInWithGithub() {
    this.auth.loginGithub();
  }

}
```

Ispis 16: Prikaz `login()` metode unutar `login.component.ts`

```
login(email: string, password: string) {
    this.fireAuth.signInWithEmailAndPassword(email, password).then(res => {
      localStorage.setItem('token', 'true');

      if (!res.user?.emailVerified){
        this.router.navigate(['verify-email']);
        this.sendEmailForVerification();
      }else{
        this.router.navigate(['products']);
        alert("You have been successfuly logged in!");
      }

    }, err => {
      alert(err);
      this.router.navigate(['login']);
    })
  }
```

Ispis 17: Prikaz `login` metode unutar `auth.service.ts`

Ukoliko se korisnik ne želi prijaviti putem maila i lozinke, može se prijaviti koristeći Googleov autentikacijski sustav. Taj sustav je implementiran putem Googleovog autentikacijskog poslužitelja. On se šalje kao argument metode `signInWithPopup()` koja se nalazi unutar `fireAuth` autentikacijske usluge. Nakon što je korisnik uspješno prijavljen, aplikacija ga preusmjeri na stranicu s proizvodima te se u lokalnom spremištu pohrani token, s vrijednošću ID-a korisnika. Ukoliko dođe do nekakve pogreške, korisniku se na vrhu monitora ispiše poruka o grešci. Forma i metoda za prijavu putem Googlea su prikazani na ispisima 18 i 19.

```
<div class="row mt-2">
            <a (click)="signInWithGoogle()" class="links-UM">
              <div class="card">
                <div class="card-body">
                  <img  src="../../../assets/images/products/icons8-google-
48.png" alt="sign-in-with-google">Sign in
                  using Google
                </div>
              </div>
            </a>
          </div>
```

Ispis 18: Prikaz kartice za prijavu putem Googlea

```
loginGoogle() {
    return  this.fireAuth.signInWithPopup(new  GoogleAuthProvider).then(res
=> {
      this.router.navigate(['products']);
      localStorage.setItem('token', JSON.stringify(res.user?.uid));
    }, err => {
      alert(err);
    });
  }
```

Ispis 19: Prikaz implementacije prijave putem Googlea

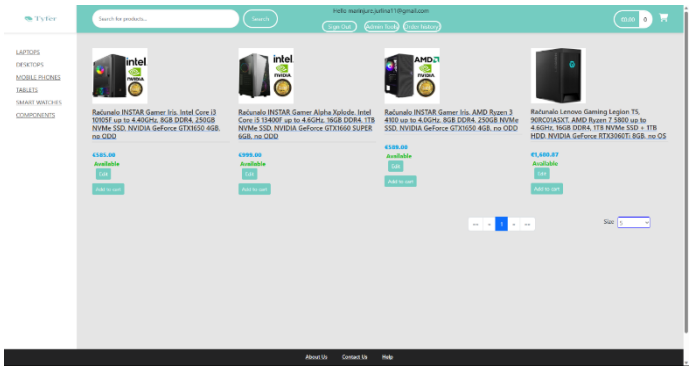
3.5. Cjelokupna izvedba

U ovoj cjelini, bit će prikazana cjelokupna izvedba aplikacije. Ova cjelina sadržavat će pregled proizvoda, pregled detalja jednog od proizvoda, pokušaj pretraživanja i paginaciju, pregled povijesti naručivanja te sam postupak naručivanja i bit će opisan postupak plaćanja.

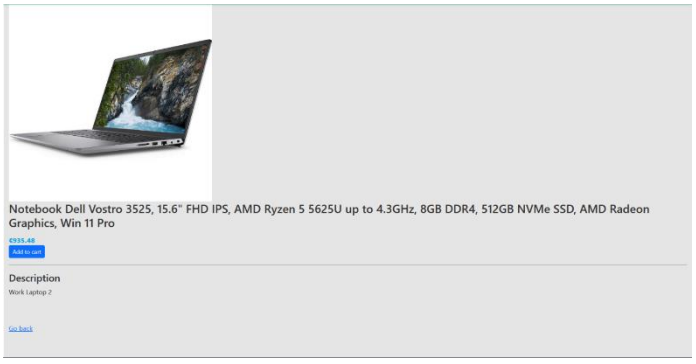
Kada korisnik prvi puta dođe na stranicu, prikažu mu se svi proizvodi. Proizvodi se nalaze u najvećem okviru koji zauzima najveći postotak prostora na monitoru. S lijeve strane glavnog izbornika, nalazi se filtracija po kategorijama te kada korisnik stisne na jednu od kategorija, proizvodi se promijene u one kojima kategorija odgovara odabranoj. Na dnu stranice nalazi se *footer* koji sadržava *About Us, Contact Us* i *Help* stranice. Stranica *About Us* predstavlja stranicu koja se fokusira na predstavljanje aplikacije. U ovom slučaju, onda služi kao upozorenje korisnicima, da proizvodi, imena, cijene te moguće usluge nisu stvarne te da ova aplikacija služi samo za izradu završnog rada. *Contact Us* i *Help* stranica služe kako bi korisnici mogli kontaktirati administratora preko njegove mail adrese. Na vrhu stranice nalazi se tražilica te ukoliko korisnik nije prijavljen i gumbovi za registraciju i prijavu. Ukoliko korisnik je prijavljen, umjesto gumbova za prijavu i registraciju, nalaze se gumbovi za odjavu i pregled povijesti naručivanja. S desne strane nalazi se područje za računanje cijena proizvoda te ukupnu količinu proizvoda, koju kad se stisne izađu detalji o narudžbi. Ti detalji uključuju sliku proizvoda, njegovi detalji, količina te sveukupna cijena tog jednog proizvoda. Korisnik može dodavati, brisati ili ukloniti sve proizvode iz svoje košarice klikom na jednog od tri gumba. Ispod liste proizvoda, nalazi se sveukupna količina svih proizvoda, cijena dostave te

ukupna cijena s 25% poreza. Ispod toga nalazi se gumb *Checkout*. Klikom na njega ispiše se forma koja očekuje od korisnika njegovo ime prezime i mail, adresu za pošiljku i naplatu te formu za upis broja kartice, trajanja kartice i sigurnosnog broja. Implementacija za kartično plaćanje je izvršena putem *Stripeovog* API-a. Nakon što je korisnik uspješno naručio i platio odabrane proizvode, u obavijesti na vrhu stranice bit će mu prikazan broj pošiljke te će isti biti dostavljen u povijesti narudžbi. Nakon toga, korisnika se preusmjerava na početnu stranicu, brišu podaci o njemu te mu se iz košarice miču svi prethodno odabrani proizvodi.

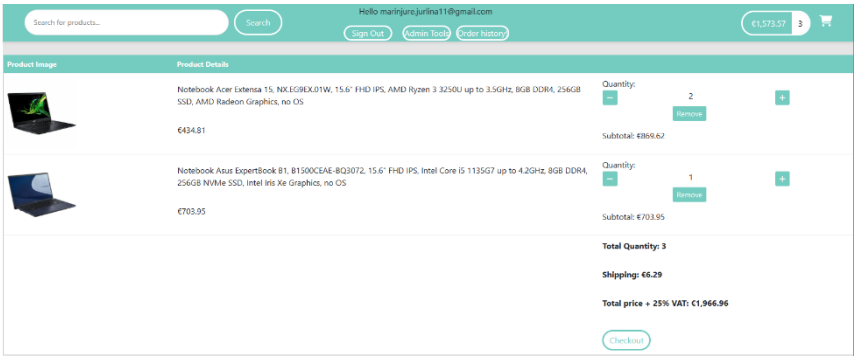
Ako je korisnik administrator, u području za pretraživanje i ispred svakog proizvoda, bit će mu prikazani gumbovi za dodavanje novih proizvoda i promjenu već postojećih. Proizvodi se ne mogu brisati, jer se isti deaktiviraju. To znači da kad su proizvodi deaktivirani, nije ih moguće kupiti, ali je pregledati. Cjelokupni izgled aplikacije je prikazan na slikama 15, 16, 17, 18, 19 i 20.



Slika 15: Prikaz početne stranice sa svim proizvodima i kategorijama

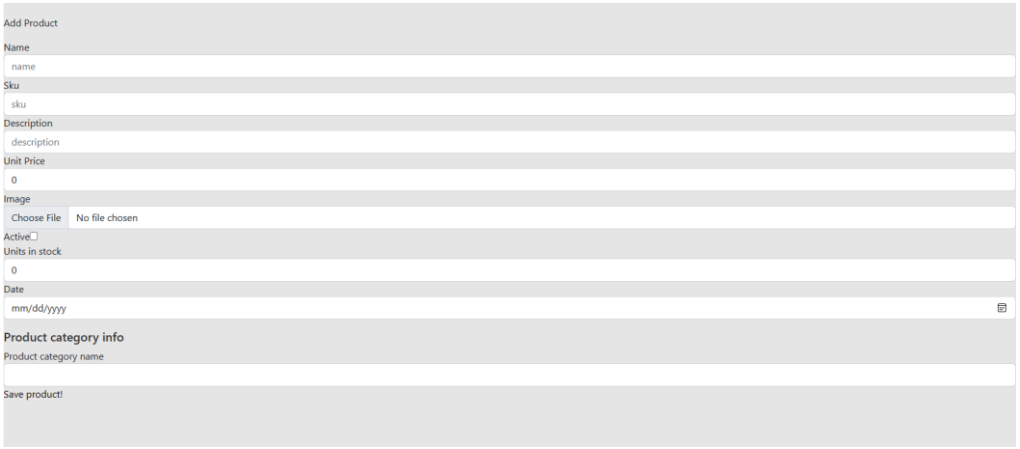


Slika 16: Kartica s detaljnim pregledom proizvoda

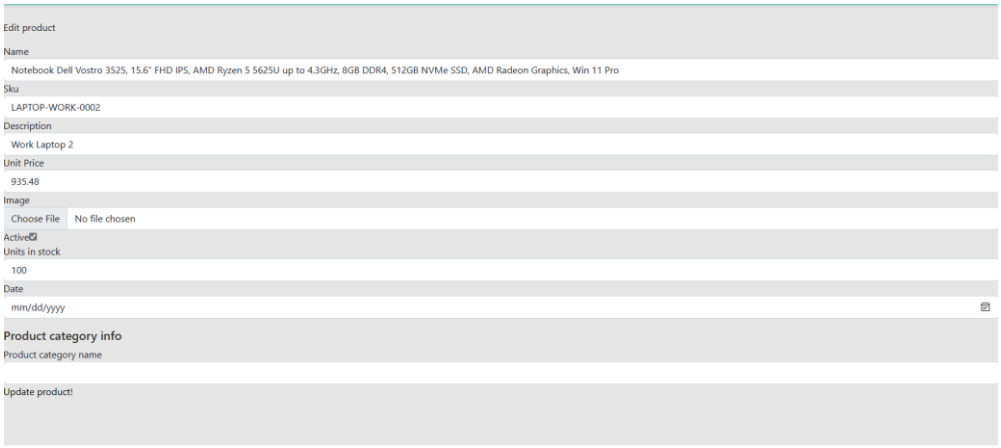


Slika 17: Prikaz košarice

Slika 18: Stranica za povijest narudžbi



Slika 19: Stranica za dodavanje novog proizvoda



Slika 20: Stranica za uređivanje jednog od proizvoda

4. Zaključak

Tema završnog rada je bila izrada web aplikacije specijalizirane za prodaju informatičke opreme. Tehnologije koje su se koristile za izradu navedenog rada su: Angular, Java Spring Boot, HTML, CSS, Bootstrap, MySQL i Firebase & Firestore Database. Razlog korištenja odabranih tehnologija je upoznatost za njima, njihova jednostavnost korištenja, čitljivo pisanje kôda, interoperabilnost tehnologija, kao i jednostavno namještanje potrebnih svojstava. Angular, glavna karika ove aplikacije, je odličan programski jezik za stvaranje jednostraničnih, responzivnih aplikacija jer sa svojom jednostavnošću, efikasnošću i jako velikom bazom korisnika, omogućuje programeru brzo i efektivno pisanje kôda. Java Spring Boot, još jedna od glavnih karika ove aplikacije, je ništa manje atraktivan programski jezik za programiranje poslužiteljskog dijela aplikacije. Omogućuje komunikaciju između korisničkog djela i baze podataka. Java Spring Boot isto tako omogućuje brzo pisanje kôda stoga ne čudi činjenica da je i dalje jedan od najpopularnijih programskih jezika za programiranje na internetu.

Aplikacija je jednostavna za korištenje, nije potrebna duga prilagodba da bi se počelo s efektivnim korištenjem aplikacije. Za obavljanje kupovine, prijava nije potrebna, ali nudi mogućnost pamćenja prijašnjih narudžbi. Korisnik može iz udobnosti svoga doma naručiti željeni proizvod, pregledavati proizvode koji ga zanimaju te ukoliko ima kakvih primjedbi, može kontaktirati korisničku službu na mail administratora. Kartično plaćanje je intuitivno, brzo i efikasno, stoga korisnik ne treba dugo čekati prilikom obrade njegovih osobnih podataka. Podaci se čuvaju u Firestore Database, radi identifikacije korisnika i omogućavanju administrativnih poslova.

Iako je aplikacija razvijana dugo vremena, još uvijek postoji mogućnost napretka. Jedna osoba teško može razviti sustav koji bi zadovoljavao sve moguće uvjete da bi korištenje aplikacije prošlo bez problema. Neke od nadogradnji uključuju: uređivanje korisničkog računa, prijavljivanje proizvoda kao defektnog, ocjenjivanje proizvoda, gledanje recenzija i još mnogo drugih. Ovaj rad je započet kao proces učenja odabranih tehnologija za stjecanje vještina koje su tražene na tržištu rada.

LITERATURA

[1]. Stackify, „What Is Spring Boot“, (16.09.2019). [Na internetu]. Dostupno: https://stackify.com/what-is-spring-boot/ [pristupano 03.06.2023]

[2]. simplilearn, „What is Angular?: Architecture, Features, and Advantages“, (24.02.2023). [Na internetu]. Dostuno: https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angular [pristupano 03.06.2023]

[3]. MDN, „Getting started with Angular“, (09.05.2023). [Na internetu]. Dostupno: https://developer.mozilla.org/enUS/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Angular_getting_started [pristupano 03.06.2023]

[4]. W3schools, „HTML Introduction“, (bez datuma). [Na internetu]. Dostpuno: https://www.w3schools.com/html/html_intro.asp [pristupano 03.06.2023]

[5]. W3schools, „HTML Styles – CSS“, (bez datuma). [Na internetu]. Dostpuno: HTML Styles CSS (w3schools.com) [pristupano 03.06.2023]

[6]. Codecademy, „A Beginner's Guide to Bootstrap“, (16.09.2021). [Na internetu]. Dostupno: What Is Bootstrap: A Beginner's Guide to Bootstrap (codecademy.com) [pristupano 03.06.2023]

[7]. W3schools, „What is Bootstrap?“, (bez datuma). [Na internetu]. Dostupno: What is Bootstrap (w3schools.com) [pristupano 03.06.2023]

[8]. DigitalOcean, „What is MySQL?“, (14.12.2020). [Na internetu]. Dostupno: https://www.digitalocean.com/community/tutorials/what-is-mysql [pristupano 03.06.2023]

[9]. builtin, „What is MySQL?“, (04.01.2023). [Na internetu]. Dostupno: https://builtin.com/data-science/mysql [pristupano 03.06.2023]