



УНИВЕРЗИТЕТ У НОВОМ САДУ  
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА  
У НОВОМ САДУ




Луна Живковић

# **ВЕБ АПЛИКАЦИЈА ЗА ПРОДАЈУ РЕКЛАМНОГ МАТЕРИЈАЛА**

ДИПЛОМСКИ РАД  
- Основне академске студије -

Нови Сад, 2021

	УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Датум:
	<b>ЗАДАТАК ЗА ИЗРАДУ ДИПЛОМСКОГ (BACHELOR) РАДА</b>	Лист/Листов

(Податке уноси предметни наставник - ментор)

Врста студија:	<input type="checkbox"/> Основне академске студије <input checked="" type="checkbox"/> Основне струковне студије
Студијски програм:	<b>Рачунарство и аутоматика</b>
Руководилац студијског програма:	<b>Проф. др Милан Видаковић</b>

Студент:	<b>Луна Живковић</b>	Број индекса:	<b>РА47/2017</b>
Област:	<b>Веб програмирање</b>		
Ментор:	<b>Проф. др Милан Видаковић</b>		
НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ДИПЛОМСКИ (Bachelor) РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА: <ul style="list-style-type: none"> <li>- проблем – тема рада;</li> <li>- начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;</li> <li>- литература</li> </ul>			

### НАСЛОВ ДИПЛОМСКОГ (BACHELOR) РАДА:

**Веб апликација за продају рекламног материјала**

### ТЕКСТ ЗАДАТКА:

Задатак рада представља развој веб апликације за продају рекламног материјала. Серверски део апликације ће бити реализован у програмском језику Јава, уз помоћ Spring окружења, док ће клијентски део веб апликације бити развијан у React.js окружењу, а клијентски део мобилне апликације у React Native окружењу. Апликација ће служити за објављивање, преглед и поручивање рекламног материјала.

Руководилац студијског програма:	Ментор рада:

Примерак за: ☐ Студента; ☐ Ментора



# SADRŽAJ

---

1.	Uvod.....	2
2.	Opis korišćenih tehnologija .....	4
2.1	Java programski jezik.....	4
2.2	Spring okruženje .....	7
2.2.1	Spring MVC.....	8
2.2.2	Spring Data JPA.....	10
2.2.3	Spring Security.....	10
2.3	PostgreSQL .....	11
2.4	React radni okvir .....	11
2.4.1	Komponenta.....	12
2.4.2	React Bootstrap.....	13
2.4	React Native radni okvir .....	14
3.	Specifikacija aplikacije .....	15
3.1	Dijagram slučajeva korišćenja .....	18
3.2	Dijagram klasa.....	21
3.3	Dijagram sekvence .....	24
4.	Opis implementacije .....	26
4.1	Serverski deo .....	26
4.1.1	Model .....	27
4.1.2	Repository .....	29
4.1.3	Servis.....	30
4.1.4	DTO .....	33
4.1.5	Controller .....	34
4.2	Klijentski deo .....	37

5. Zaključak.....	55
6. KLJUČNA DOKUMENTACIJSKA INFORMACIJA .....	59
7. KEY WORDS DOCUMENTATION.....	61

# 1. Uvod

Problem na kom se zasniva aplikacija jeste sve veća potreba da se proizvodi prodaju preko interneta, što povećava broj kupaca, ali traži i veći broj opcija koje bi svim kupcima bile jasne i dostupne. Obavljanje kupovine ne predstavlja i kraj “problema”, jer aplikacija mora da omogući prodavcu da ima uvid u to šta je prodato, koliko robe je ostalo u zalihama, kao i da obavesti kurira da postoji paket za isporuku koji treba da pokupi.

Rešenje koje će biti dalje analizirano predstavlja veb aplikaciju čija je namena pristup svim artiklima firme, pregled karakteristika pojedinačnih, poručivanje istih, kao i dodavanje novih proizvoda. Takođe, važan deo aplikacije čini korisnikova mogućnost skeniranja QR koda paketa sa porudžbinom, kao i mogućnost kontaktiranja kompanije preko aplikacije.

Sama aplikacija razvijena je i kao veb i kao mobilna aplikacija, gde korisnik mobilne aplikacije, pored svih mogućnosti koje nudi veb aplikacija, ima dodatnu mogućnost skeniranja QR koda.



## 2. Opis korišćenih tehnologija

Serverski deo aplikacije implementiran je u programskom jeziku Java [1], uz pomoć Spring okruženja [2] i njegovog razvojnog okvira Spring Boot [3]. Za potrebe dodatne sigurnosti aplikacije i implementacije autentifikacije i autorizacije upotrebljen je Spring Security radni okvir.

Za usluge skladištenja podataka korišćen je PostgreSQL [4], objektno-relacioni sistem za upravljanje bazama podataka.

Za klijentski deo veb aplikacije korišćena je React [5] biblioteka za JavaScript [7] jezik, dok je za mobilnu aplikaciju korišćen React Native[6]. Za dodatno vođenje računa o klijentskom delu veb aplikacije, korišćen je Bootstrap [8] radni okvir, kao i neki od brojnih dodataka biblioteci koji, u zavisnosti od svoje namene, pomažu u raznim delovima razvoja aplikacije.

### 2.1 Java programski jezik

Java predstavlja objektno-orijentisani programski jezik koji je dizajniran tako da bude pogodan za razvoj programa na svakom tipu računara. Razvila ga je kompanija “Sun Microsystems” početkom devedestih godina XX veka. Njena prednost je to što je ona prost jezik u poređenju sa drugim programskim jezicima i zbog toga se relativno lako uči. Ta jednostavnost je neophodna da bi se podržala nezavisnost Java aplikacija od tipa platforme. Sintaksa ovog jezika zasniva se na programskim jezicima C i C++, uz dodatno pojednostavljenje.

Pomoću Jave je moguće razvijati jednostavne, teksturalno-zasnovane programe koji se nazivaju konzolne aplikacije, kao i aplikacije sa grafičkim korisničkim interfejsom (engl. Graphical User Interface – GUI). Takođe, moguće je praviti i aplikacije koje se nazivaju apleti (engl. Applets). To su male GUI aplikacije koje mogu da se izvršavaju unutar veb stranice.



Prilikom stvaranja programskog jezika Java, oslanjalo se na pet osnovnih ciljeva:

- *Jednostavnost i objektna orijentisanost*

Java mora biti jednostavan, objektno orijentisani jezik programski jezik. Zahvaljujući jednostavnosti, u njoj se može lako programirati, bez potrebe za prethodnim uhođavanjem jer je bazirana na postojećem načinu razmišljanja. Objektno orijentisano projektovanje predstavlja tehniku programiranja fokusiranu na objekte (podatke) i na interfejsa ka tim objektima.

- *Sigurnost*

Programski jezik mora biti siguran, s obzirom da je namenjen korišćenju u mrežnim/distribuiranim okruženjima. Uloženo je mnogo vremena u implementaciju bezbednosti, time omogućavajući izgradnju sistema koji su zaštićeni od zlonamernih napada i modifikacije.

- *Prenosivost*

Java programski jezik mora biti prenosiv i nezavisan od platforme. Format datoteke stvorene Java kompajlerom je neutralan i nezavisan od operativnog sistema na kom će biti pokrenut.

- *Performantnost*

Mora se izvršavati sa visokim performansama.

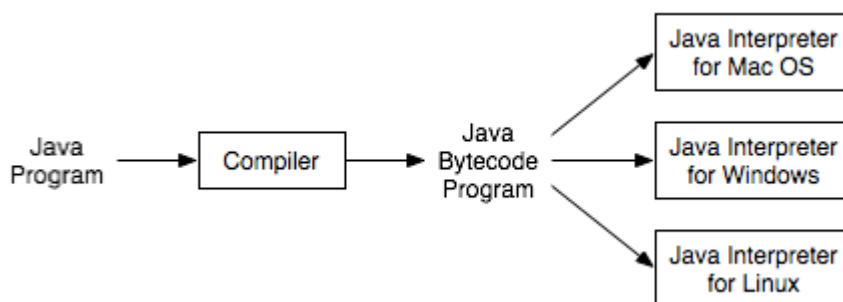
- *Dinamičnost i interpretiranost*

Zahvaljujući dinamičnosti, Java se prilagođava okruženju koje se stalno unapređuje. Java bajtkod se može izvršavati na svakom kompjuteru za koji postoji Java interpreter.

- *Neutralnost*

Java platforma, tj. Java okruženje služi za izvršavanje programa i dizajnirana je tako da što manje zavisi od specifičnih karakteristika konkretnog računarskog sistema.

Korišćenjem JVM (*Java Virtual Machine*) moguće je izvršavati Java kod, gde se izvorni kod, nakon kompajliranja, prevodi u bajtkod, koji je isti za sve arhitekture računara i predstavlja instrukcije za JVM. U zavisnosti od JVM, za jedan izvorni kod mogu se dobiti različiti izvršni kodovi, za različite platforme. Na slici 2.1.1 se može videti opisani ciklus.



Slika 2.1.1 Tok Java koda

Java je strogo tipiziran jezik, jer zahteva da se za svaki podatak u svakom trenutku zna kom tipu pripada. U Javi postoje dva tipa podatka: *prosti tipovi podataka* (int, char, byte, long, float, boolean) i *složeni tipovi podataka* (objekti i nizovi). Svaki od prostih tipova podataka ima tačno definisanu veličinu, bez obzira na platformu na kojoj se izvršava Java kod.

U Javi je moguće imati više objekata iste vrste koji imaju neke zajedničke osobine. Koristeći tu činjenicu mogu se kreirati šabloni za ovakve objekte. Ovi šabloni zovu se *klase*. Dakle, klasa predstavlja kolekciju podataka i metoda (procedura i funkcija) koje manipulišu sa tim podacima.

Klasa sadrži jedan ili više konstruktora koji se razlikuju po broju argumenata. Oni nose informacije o tome koji se podaci iz spoljnog sveta moraju obezbediti prilikom inicijalizacije objekta i na koji način se ti podaci koriste. Konstruktori nose isto ime kao i klasa.

JAVA manipuliše objektima preko reference. Kada se objekat prosleđuje nekoj metodi samo se prenosi referenca na objekat.

Većina promenljivih pripada određenom objektu. To znači da svaki objekat koji se zasniva na šablonu klase može da ima različitu vrednost za istu promenljivu.

## 2.2 Spring okruženje

Razvojno okruženje Spring je Java platforma koja pruža veliki broj opcija i predstavlja podršku razvoju složenih Java aplikacija. Spring upravlja infrastrukturom, tako da se programer može usredsrediti na razvoj aplikacije.

Karakteristike Spring okruženja:

- Organizovan je kroz module
- Koristi poznate koncepte
- Testiranje aplikacija pisanih u Springu je jednostavno jer je celo okruženje potrebno za testiranje integrisano u radni okvir
- Inversion of Control (IoC) kontejner je jednostavniji u odnosu na npr. EJB Java kontejner
- Spring pruža mogućnost korišćenja dobro razvijenog transakcionog menadžmenta
- Upotreba dodatnih biblioteka koje pomažu pri izgradnji bilo kakve aplikacije uz dodatnu fleksibilnost i jednostavnost.

*Dependency Injection (DI)* mehanizam [9] i *Inversion of Control (IoC)* [10], odnosno kontrola toka programa, predstavljaju ključni deo koji omogućava funkcionisanje Java aplikacije pisane korišćenjem tehnologije Spring razvojnog okruženja.

*Dependency Injection* se odnosi na prosleđivanje, ‘ubrizgavanje’, objekata drugim objektima. Može da se implementira kroz konstruktor, prosleđivanjem zavisne komponente kao parametar, zatim kroz set metodu, gde kontejner poziva set metodu nakon konstruisana objekta. Najčešće se koristi kroz atribut klase uz pomoć anotacije, najčešće anotacijom `@Autowired`

*Inversion of Control (IoC)* je tehnika kojom se povezivanje objekata vrši u toku izvršavanja (runtime), a ta veza nije poznata u toku kompajliranja (compile time). Da bi kontejner mogao da izvrši povezivanje, objekti moraju imati odgovarajuće uputstvo. Jedna forma IoC je dependency injection (DI).

Spring kontejner predstavlja srž Spring radnog okvira. On koristi dependency injection (DI) da upravlja komponentama koje čine aplikaciju i način na koji funkcioniše je kreiranje objekata, njihovo uvezivanje, konfigurisanje i upravljanje životnim ciklusom od kreiranja do uništenja.

### 2.2.1 Spring MVC

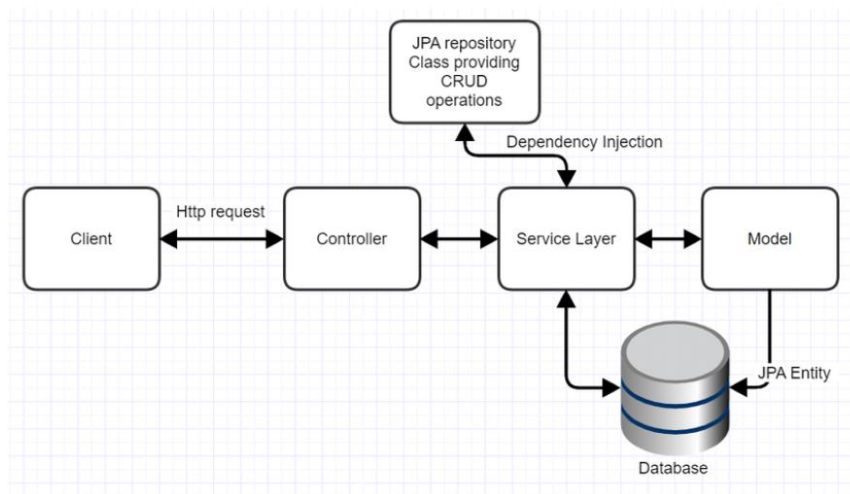
Osnovni princip dizajna u Spring MVC i Spring-u uopšte je “Otvoren za proširivanje, zatvoren za modifikovanje”, što bi značilo da aplikacije koje su rađene u Spring razvojnom okruženju omogućuju proširivanje raznim dodacima, ali ne dozvoljavaju izmene na samoj aplikaciji. (prema Deinum, M. i saradnici - Deinum, M., Serneels, K., Yates, C., Ladd, S., Vanfleteren, C. and Vervaet, E. (2012). *Pro Spring MVC*. [S.l.]: Apress.)

Spring Web MVC [11] je radni okvir koji podržava izgradnju veb aplikacije sa *Model-View-Controller* arhitekturom. Većina savremenih aplikacija ne razvija se po MVC šablonu, već se deli na serverski i klijentski deo. Međutim, ukoliko klijentsku aplikaciju realizujemo kao *view* deo koji je odgovoran za ono šta klijent vidi, dobre strane MVC-a biće očuvane.

REST (*Representational State Transfer*) [12] je stil softverske arhitekture namenjen distribuiranim hipermedijalnim sistemima kao što je WWW (World Wide Web). Najčešće se zasniva na HTTP [13] (Hypertext Transfer Protocol) protokolu za komunikaciju.

Za upravljanje resursima koriste se uglavnom predefinisane CRUD (Create, Read, Update, Delete) operacije, koje HTTP protocol podržava svojim POST, GET, PUT, DELETE metodama.

Na slici 2.1.1.1 je prikazana uobičajena arhitektura Spring aplikacije.



**Slika 2.1.1.1 Arhitektura Spring aplikacije**

Prva dužnost jeste primanje HTTP zahteva poslatog sa klijentske strane.

*Controller* predstavlja klasu koja obrađuje zahteve upućene njemu i poziva odgovarajuće servisne metode. *Controller* je Java klasa označena sa *@Controller* ili *@RestController* anotacijama. Metode unutar ovih klasa su označene sa *@RequestMapping*. Uz pomoć anotacija određuje se način pristupanja svakom zahtevu, a zahtev može sadržati parametre i oni mogu biti prosleđeni kontroleru.

Uobičajeni pristup ovom šablonu podrazumeva postojanje servisnih komponenti. Kada kontroler prosledi podatke servisu i kada se izvrše željene akcije, kontroler prima odgovor koji servis vraća. Odgovor može biti vraćen kao objekat ili kao bilo koji drugi tip informacije.

Model predstavlja opis objekata pomoću varijabli i tipova podataka. Sadrži konstruktor za inicijalizaciju objekta i getere i setere, odnosno metode za prikaz i umetanje vrednosti varijabli. Podaci koji su prikazani u modelu su skladišteni u bazi podataka.

*Repository* predstavlja interfejs implementiran pomoću *JPA* (*Java Persistence API*) [13] o kom će biti reč u sledećem poglavlju.

### 2.2.2 Spring Data JPA

Spring Data JPA (*Java Persistence API*) [14] je deo Spring Data okruženja, koji olakšava implementaciju i upravljanje repozitorijumima. JPA predstavlja specifikaciju skladištenja, pristupa i upravljanja podacima u relacionoj bazi podataka, zasnovanih na Java objektima. Iako je prvobitno bio namenjen isključivo za relacione baze, danas postoje JPA implementacije koje su proširene tako da ih je moguće koristiti i sa NoSQL bazama.

Najpopularnija implementacija JPA je Hibernate koja predstavlja ORM (Object-relational mapping) biblioteku za Java jezik. Postoje brojne anotacije koje pomažu pri, recimo, kreiranju tabela, kao što su `@Entity` pomoću koje označavamo koje Java klase predstavljaju entitete, `@Table` pomoću koje definišemo naziv tabele, `@Column` pomoću koje eksplicitno navodimo naziv atributa na koji će se mapirati polje u klasi, `@Id` pomoću kojeg definišemo id entiteta i mnogi drugi. Takođe, entiteti gotovo uvek sadrže reference ka nekim drugim entitetima, a veze između njih lako se mogu definisati pomoću `@OneToOne`, `@OneToMany`, `@ManyToOne` ili `@ManyToMany` anotacija.

### 2.2.3 Spring Security

Spring Security [15] je framework koji se koristi za autentifikaciju i kontrolu pristupa Java aplikacija. Osnovne karakteristike predstavljaju izuzetna podrška za autorizaciju i autentifikaciju i zaštita od napada kao što su CSRF (Cross Site Request Forgery) i XSS (Cross Site Scription).

Autentifikacija predstavlja proces u kojem se utvrđuje identitet korisnika. Dozvola i zabrana pristupa određenim resursima, na osnovu njegovih permisija predstavlja autorizaciju koja je ponekad i suvišna, ukoliko ne postoje različite permisije.

Autentikacija koja se najčešće upotrebljava i koja je implementirana u ovoj aplikaciji bazirana je na tokenima, u ovom slučaju JWT (JSON Web Token) koji se sastoje iz tri dela, a moguće je ubaciti i dodatne parametre. Korisnik nakon prijavljivanja na sistem, u slučaju uspešne prijave, dobija od sistema jedinstveni token, koji šalje u zaglavlju HTTP zahteva kao potvrdu identiteta.

Kako bismo implementirali Spring Security, potrebno je dodati anotaciju `@EnableSpringSecurity` u Java klasi koja će naslediti ugrađenu klasu `@WebSecurityConfigurerAdapter`.

## 2.3 PostgreSQL

*PostgreSQL* je jedan od najčešće korišćenih objektno-relacionih sistema za upravljanje bazama podataka. Zasniva se na SQL standardu i omogućava mnoge savremene karakteristike kao što su kompleksni upiti, trigeri, pogledi, upotreba stranih ključeva i drugi. Ovu bazu možemo prilagođavati svojim potrebama, jer je PostgreSQL baza otvorenog koda. Prednost joj je što je dostupna na većini najčešće korišćenih platformi i programskih jezika, a što se sigurnosti tiče - predstavlja svetski priznatu bazu podataka kao najsigurnija i najskalabilnija.

## 2.4 React radni okvir

Osnivač ReactJS-a je Jordan Walke, programski inženjer Facebook-a. Prva stabilna verzija je objavljena 2013. godine, a danas je prema statističkim podacima Stack Overflow-a ReactJS omiljeni alat za razvoj klijentske strane veb aplikacija.

React je JavaScript biblioteka otvorenog koda za razvoj klijentskih aplikacija. Iako je reč o biblioteci, zahvaljujući velikoj popularnosti, razvijeno je mnoštvo paketa koji proširuju biblioteku kako bi se omogućile funkcije koje nedostaju. Kompatibilna je sa nizom alata zahvaljujući fleksibilnosti biblioteke, čijom se

integracijom postiže potpuna funkcionalnost radnog okvira. Zato u praksi većina smatra ReactJS radnim okvirom.

React odlikuje mnoštvo pozitivnih osobina među kojima su najvažnije mogućnost razvijanja aplikacije u JavaScript jeziku, mogućnost razgradnje kompleksnog UI-a na jednostavnije komponente i skladištenje svih podataka na jednoj lokaciji (*state*).

Još jedna specifična odlika predstavlja upotreba virtual DOM-a što čini aplikaciju znatno bržom. Zahvaljujući čuvanju u keš memoriji i računajući razlike, DOM se ažurira samo ukoliko je došlo do izmena.

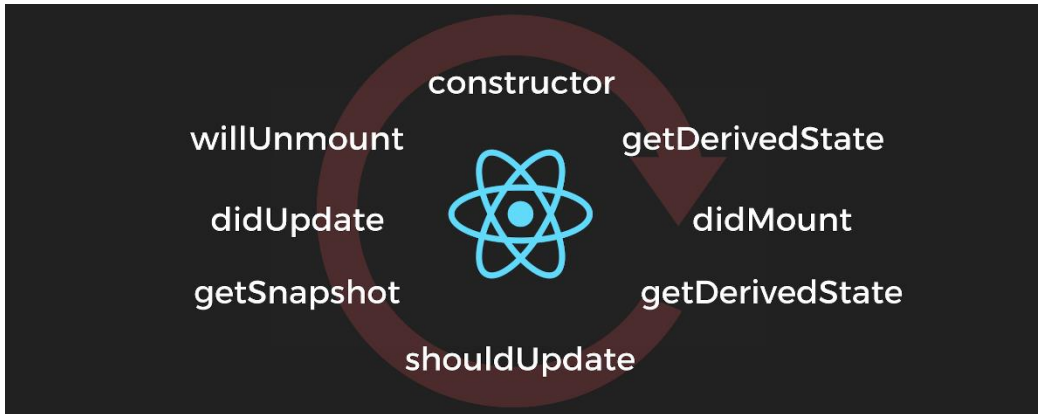
### 2.4.1 Komponenta

Komponente su ono šta omogućava da naš kod bude ponovo iskoristiv i šta omogućava podelu kompleksnog UI-a na jednostavnije celine. Identične su JavaScript funkcijama i prihvataju inpute koji se nazivaju *props*, a vraćaju React elemente koji opisuju šta treba biti renderovano korisniku.

Postoje dva načina deklarisanja komponenti ReactJS-a. Jednostavnije komponente se mogu definisati kao JavaScript funkcije koje vraćaju kod pisan JSX sintaksom, što postaje vrlo nepregledno ukoliko komponenta ima mnogo funkcionalnosti. Tada je praksa da se koristi ES6 sintaksa definišući komponentu ključnom rečju *class*. Nužno je da klasa koji predstavlja komponentu nasledi *Component* koji se importuje iz biblioteke *react*. Naziv komponente mora započinjati velikim slovom, jer u suprotnom ReactJS smatra da se radi o standardnom HTML elementu.

Komponente se sastoje iz elementa za prikaz (*HTML template*) i aplikativne logike koja ih kontroliše. Svaka komponenta predstavlja klasu sa stanjem promenljivih sa kojim manipuliše i prikazuje njegove podatke. U *state* delu definiše se stanje promenljivih kojem se može pristupiti ili zadavanjem komande servisu preko HTTP zahteva ili prilikom nekog dela životnog ciklusa komponente koji je prikazan na slici 2.4.1.1. Omogućeno je i prosleđivanje promenljivih iz jedne u drugu komponentu preko konstruktora, gde se one označavaju kao *props*.





**Slika 2.4.1.1 Životni ciklus React komponente**

Najčešće metode koje se izvršavaju u zavisnosti od dela životnog ciklusa i pomoću kojih je moguće izvršiti deo koda u određenom momentu su:

- `componentDidMount()`,
- `componentDidUpdate()` ,
- `componentWillUnmount()`,
- `componentGetSnapshot()`,
- `componentGetDerivedState()` i
- `componentShouldUpdate()`.

Na osnovu životnog ciklusa komponente lako se može zaključiti kada će koja metoda biti obavljena.

## **2.4.2 React Bootstrap**

Bootstrap je open-source JavaScript radni okvir, odnosno kombinacija HTML-a, CSS-a i JavaScript-a, razvijen sa ciljem da omogući i olakša razvoj web formi (interface-a, tj. *layout*-a) kao i razvoj naprednih veb komponenti. Predstavlja kolekciju razvijenih CSS i JavaScript alata i biblioteka.

Osim što omogućava i olakšava integrisanje raznih vrsta komponenti (formi, buttons-a, akcija sa tekstom, i drugim), odlično “sarađuje” sa JavaScript-om i sa jQuery bibliotekama.

Najveća prednost Bootstrap-a je ta što u sebi ima ugrađen set alata i biblioteka za kreiranje fleksibilnih i responsive veb formi sa svim pripadajućim elementima. Takođe, olakšava i ubrzava development, omogućava jednostavniji razvoj veb formi i interfejsa namenjenih za prikazivanje sadržaja na mobilnim telefonima i mobilnim uređajima, identično se prikazuje na svim modernim *web browser*-ima, relativno je jednostavan za upotrebu i *open source* je.

```
import { Form, Button, FormGroup, FormControl, ControlLabel, Col, Modal } from "react-bootstrap";
```

**Listing 2.4.2.1** Ubacivanje Bootstrap komponente

## 2.4 React Native radni okvir

*React Native* generisan je kao besplatni izvorni JavaScript okvir, koji ima kao cilj stvaranje mobilnih aplikacija za iOS i Android. Govoreći o svom poreklu, generisan je poput ReactJS-a od strane Facebook-a i zasnovan na React JavaScript biblioteci za razvoj interfejsa.

Neke od važnijih karakteristike React Native-a su:

- Mogućnost pokretanja aplikacije i na iOS i na Android uređajima što skraćuje vreme i resurse potrebne za prevođenje,
- Kreiran je na osnovu ReactJS-a koji je trenutno najrasprostranjeniji jezik za razvoj klijentske strane aplikacije,

- Složene animacije više ne predstavljaju problem, već su pojednostavljene uvođenjem biblioteke *Animated*. Sada je moguće kreirati vizuelno privlačne i fluidne animacije,
- Programeri mogu sa lakoćom preneti svoj rad sa veb aplikacije na mobilnu i obrnuto,
- Ukoliko je potrebno izmeniti sitne greške, ne mora se čekati da one budu odobrene od strane Google i/ili Apple store-a. React Native omogućava korišćenje usluga kao što su CodePush i ažuriranje aplikacije u bilo koje vreme gde se na taj način ubrzava ceo proces i
- Svaki put kada se desi izmena koda, izgled se automatski ažurira i na taj način developeri mogu mnogo brže i lakše da prate validnost promena.

### 3. Specifikacija aplikacije

Zadatak se odnosi na izradu veb i mobilne aplikacije za kompaniju koja se bavi prodajom reklamnog materijala (majica, dukseva i kačketa). Aplikacija omogućava izvršavanje osnovnih funkcionalnosti nad artiklima, kao što su prikaz, kreiranje i brisanje. Pored toga, program nudi registrovanje korisnika, prijavu, kao i izmene podataka o korisnicima i kompaniji. Omogućeno je i kreiranje i skeniranje QR koda pomoću kojeg se potvrđuje isporuka artikala.

Pri otvaranju početne stranice veb aplikacije, korisnik se može ulogovati, registrovati ili pregledati sve artikle koje kompanija nudi, bez mogućnosti poručivanja istih. Mobilna aplikacija, sa druge strane, odmah otvara login formu i nudi mogućnost registracije. Aplikacija ne nudi ni jednu drugu mogućnost.

Uloge koje korisnik može imati i na osnovu kojih mu se prikazuju određeni podaci i pripisuju odgovornosti su:

- Neregistrovani korisnik
- Registrovani korisnik
- Admin
- Kurirska služba

#### *NEREGISTROVANI KORISNIK:*

Neregistrovanom korisniku pružaju se opcije pregleda svih artikala u sistemu, registracija ili slanje poruke kompaniji.

#### *REGISTROVANI KORISNIK:*

Nakon uspešne prijave korisnika na sistem, na veb aplikaciji, prikazuje se početna stranica i glavni meni, dok se korisniku mobilne aplikacije nudi izbor majica, dukseva ili kapa. Sve funkcionalnosti osim skeniranja QR koda, koja je moguća samo na mobilnoj aplikaciji, korisnik je u mogućnosti da obavi i sa veb i sa mobilne aplikacije. Klikom na željeni artikal, korisnik vidi informacije o istom, kao i opciju poručivanja. Osim toga, prijavljen korisnik može otići na veb stranicu koja prikazuje već poručene artikle ili prikazati njegov profil. Ukoliko je registrovani korisnik ulogovan na mobilnoj aplikaciji, nudi se dodatna funkcija – skeniranje QR koda, kojom se potvrđuje dostava artikala od strane kurira.

Registrovani korisnik takođe može videti polje sa kontakt podacima firme, gde se dodatno vidi i polje za slanje poruke Apeironu. Popunjavanjem osnovnih ličnih podataka, pisanjem poruke i pritiskom na dugme “Send”, Apeiron će na svoj mail dobiti poruku koja je upravo poslata.

Prilikom dolaska na svoj profil, korisnik ima mogućnost ažuriranja svojih podataka, kao i pregled svih artikala iz svoje kolekcije i svih zahteva koje je kreirao u prošlosti, uz prikazan status tog zahteva.

### *ADMIN:*

Meni admina sastoji se iz nekoliko dodatnih funkcionalnosti. On može da dodaje, briše ili menja boje artikala koje su na raspolaganju u kompaniji, dodaje i briše kurire sa kojima sarađuje, dodaje i briše artikle koji su dostupni i koje registrovani korisnici mogu da poručuju. Prilikom dodavanja kurira, unose se polja kao što su e-mail, naziv kompanije, broj telefona, adresa i šifra.

Ukoliko admin želi da doda novi artikal, potrebno je dodati sliku(e), selektovati vrstu (majica, duks, kapa), boju i veličinu. Pored toga unosi se količina koja se odnosi na selektovanu boju, vrstu i veličinu, kao i naziv tog artikla. Takođe, admin može da pregleda sve porudžbine, da ih odbaci ili označi kao poslate. Ukoliko odabere da su poslate, potrebno je odabrati i kurirsku službu preko koje će paket biti poslat, a koju je prethodno registrovao u sistem. Nakon odabira, stvara se i QR kod koji će moći da pregledaju admin i kurirska služba koju je prethodno selektovao.

Admin može pristupiti stranici sa kontakt podacima firme i, ukoliko želi, izmeniti ih.

Sva brisanja u sistemu može obaviti jedino Admin, osim otkazivanja porudžbina. On na svom profilu može pogledati i sve porudžbine iz sistema raspoređene na osnovu njihovih stanja.

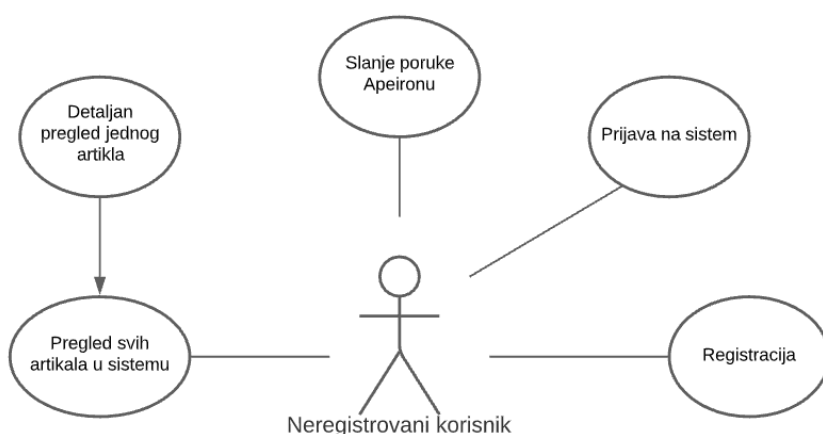
### *KURIRSKA SLUŽBA:*

Kurir, logovanjem na svoj profil može takođe da kontaktira Apeiron, pregleda dostupne artikle i pregleda QR kodove koje registrovani korisnik treba da očita sa svog mobilnog telefona i na taj način potvrdi isporuku.

U narednim poglavljima biće prikazani dijagrami slučajeva korišćenja, dijagram klasa i dijagram sekvence.

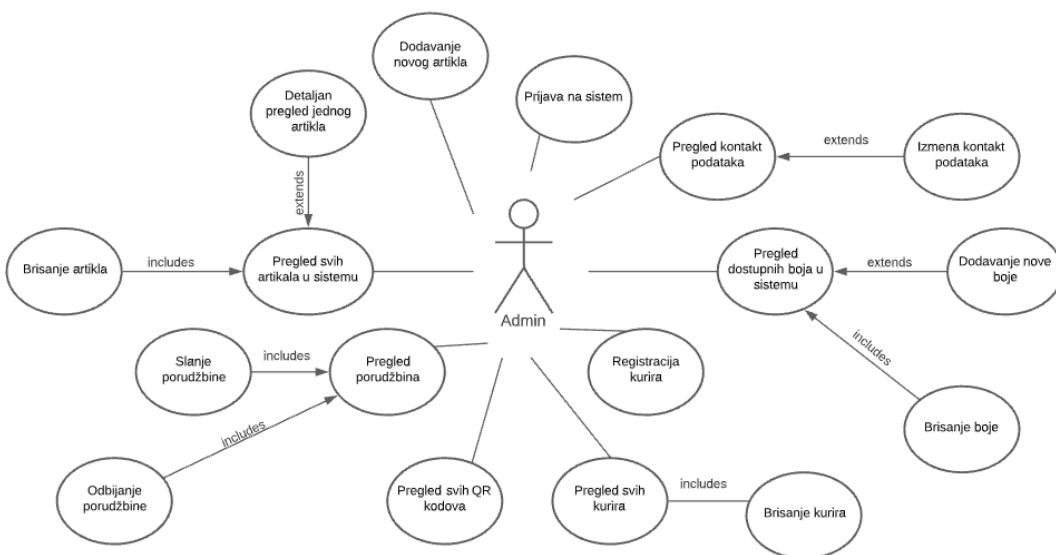
### 3.1 Dijagram slučajeja korišćenja

Kao što je već navedeno, aplikacija podržava četiri tipa korisnika. Dijagram slučajeja korišćenja za prvog, neregistrovanog korisnika je prikazan na slici 3.1.1. On ima mogućnost da se prijavi na sistem ili registruje. Takođe, neregistrovani korisnik može pregledati sve artikle iz sistema, pristupiti informacijama o jednom, određenom ili poslati poruku Apeironu.



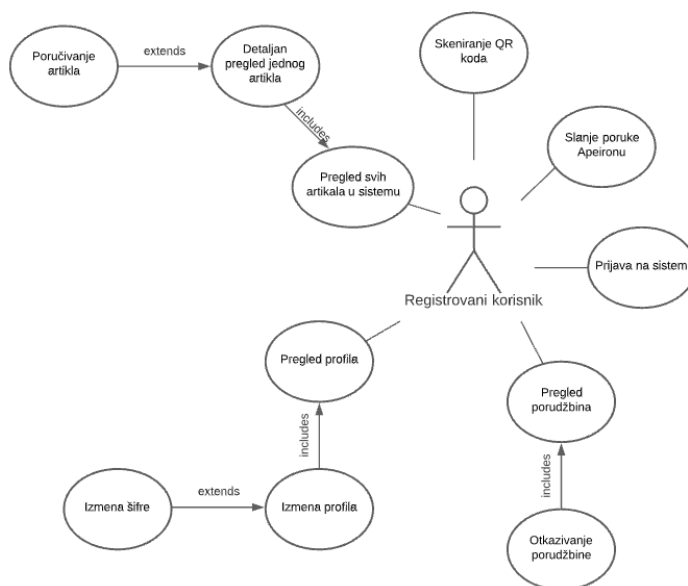
Slika 3.1.1 Dijagram slučajeja korišćenja za neregistrovanog korisnika

Na slici 3.1.2 nalazi se dijagram slučajeja korišćenja za registrovanog korisnika sa ulogom Admin. On može otići na stranicu sa svim artiklima i pregledati ili obrisati svaki od njih. Porudžbine na čekanju može da odbije ili označi kao poslate. Takođe, admin može registrovati kurira ili obrisati ga sa stranice gde su pobrojani svi. Admin ima mogućnost dodavanja ili brisanja boja artikala koje se nalaze u sistemu.



**Slika 3.1.2** Dijagram slučajeva korišćenja za korisnika sa ulogom Admin

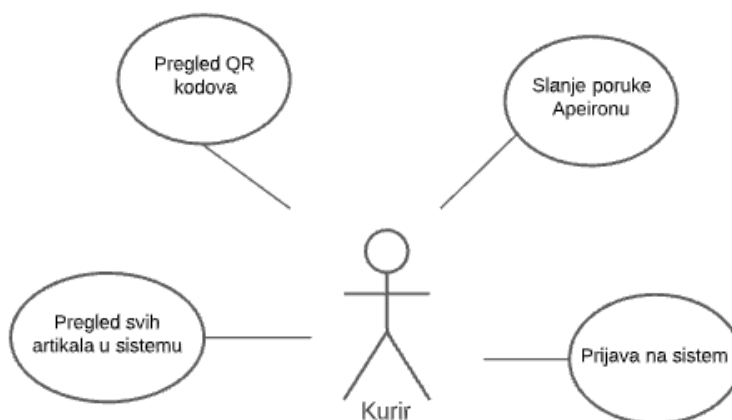
Dijagram slučajeva korišćenja za korisnika sa ulogom Registrovani korisnik se može videti na slici 3.1.3. Registrovani korisnik može dodati porudžbinu, a može je i obrisati. Na stranici svog profila ima mogućnost ažuriranja korisničkih podataka i pregleda svih artikala iz sistema. Pored toga, ima i mogućnost da kontaktira Apeiron slanjem poruke.



**Slika 3.1.3 Dijagram slučajeva korišćenja za korisnika sa ulogom User**

Na slici 3.1.4 prikazan je dijagram slučajeva korišćenja za kurira, gde je navedeno da on, osim prijave na sistem i pregleda svih artikala, može slati poruke kompaniji ili pristupiti QR kodovima koji se odnose na porudžbine za koje je on zadužen.





Slika 3.1.4 Dijagram slučajeva korišćenja za korisnika sa ulogom Kurir

## 3.2 Dijagram klasa

Model podataka posmatrane aplikacije se sastoji od četrnaest klasa.

PersonUser predstavlja interfejs koji sadrži osnovna polja, potrebna svim korisnicima sistema, kao što su *email*, *password* (šifra) ili *authority* (lista uloga). Uloge (authority) su predstavljene samo poljem *name* (naziv).

Admin, Courier i RegisteredUser implementiraju klasu PersonUser i sadrže polje *address* (adresa). Ove klase se sastoje od osnovnih podataka o korisnicima sistema kao što su *email* (e-mail adresa), *password* (lozinka) i *phoneNumber* (broj telefona). Pored osnovnih podataka, sve klase sadrže referencu ka adresi, dok registrovani korisnik i kurir sadrže i referencu ka porudžbinama (*orders*).

Order klasu opisuju polja *dueDate* (datum do kada porudžbina treba biti dostavljena), *dateOfReservation* (dan rezervacije), *status* (status porudžbine

(CREATED, DENIED, SENT, DELIVERED)) i nekoliko referenci kao što su *registeredUser* (registrovani korisnik koji je poručio artikal), *items* (poručeni artikli) i *courier* (kurir koji je zadužen za dostavu).

Artikli koji se nalaze u porudžbini predstavljaju posebnu klasu *ItemsInOrder* i sadrže polja koja ukazuju na koji se artikal porudžbina odnosi (*item*), boja poručenog artikla (*color*), veličina (*size*), kolicina (*quantity*) i datum do kada je potrebno dostavi taj određeni artikal (*date*).

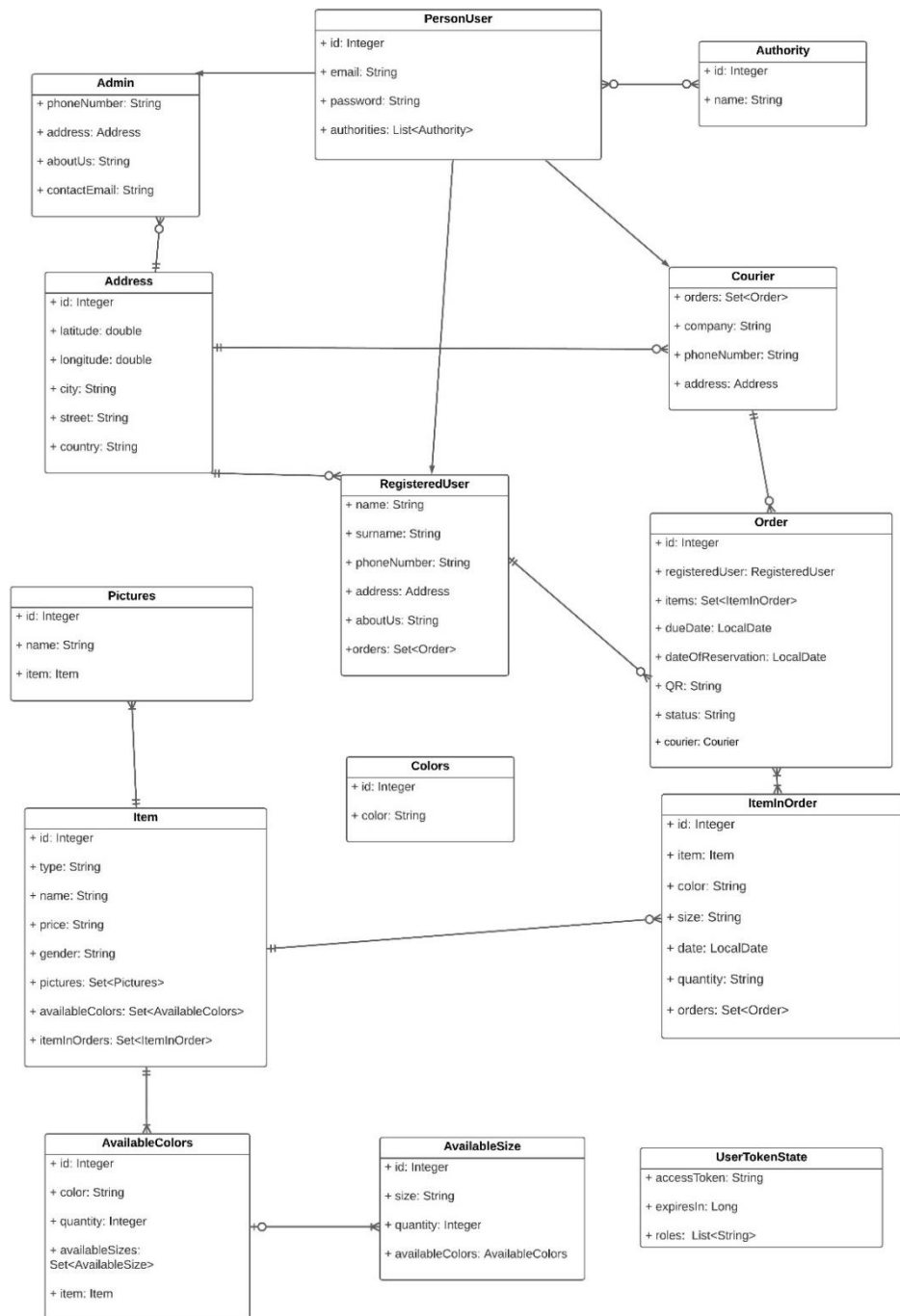
Klasa *Item* čvrsto je povezana sa tri druge klase – *Pictures*, *AvailableColors* i *AvailableSize*. Klasa *Pictures* čuva podatke o slikama koje su vezane za artikle, dok klasa *AvailableColors* sadrži podatke o tome koje boje nekog artikla su dostupne i koja je količina istih. Polje *availableSize* (dostupne veličine) predstavlja referencu na istoimenu klasu gde se nalaze podaci o dostupnim veličinama određene boje artikla.

*Colors* klasa u sebi sadrži polje *color* (boja) i koristi se pri dodavanju novih boja u sistem.

Formiranje JWT tokena obavlja se pomoću klase *UserTokenState*.

Svaka klasa pored navedenih polja sadrži i *id*, odnosno polje koje je jedinstveno identifikuje.

Dijagram klasa prikazan je na slici 3.2.1.

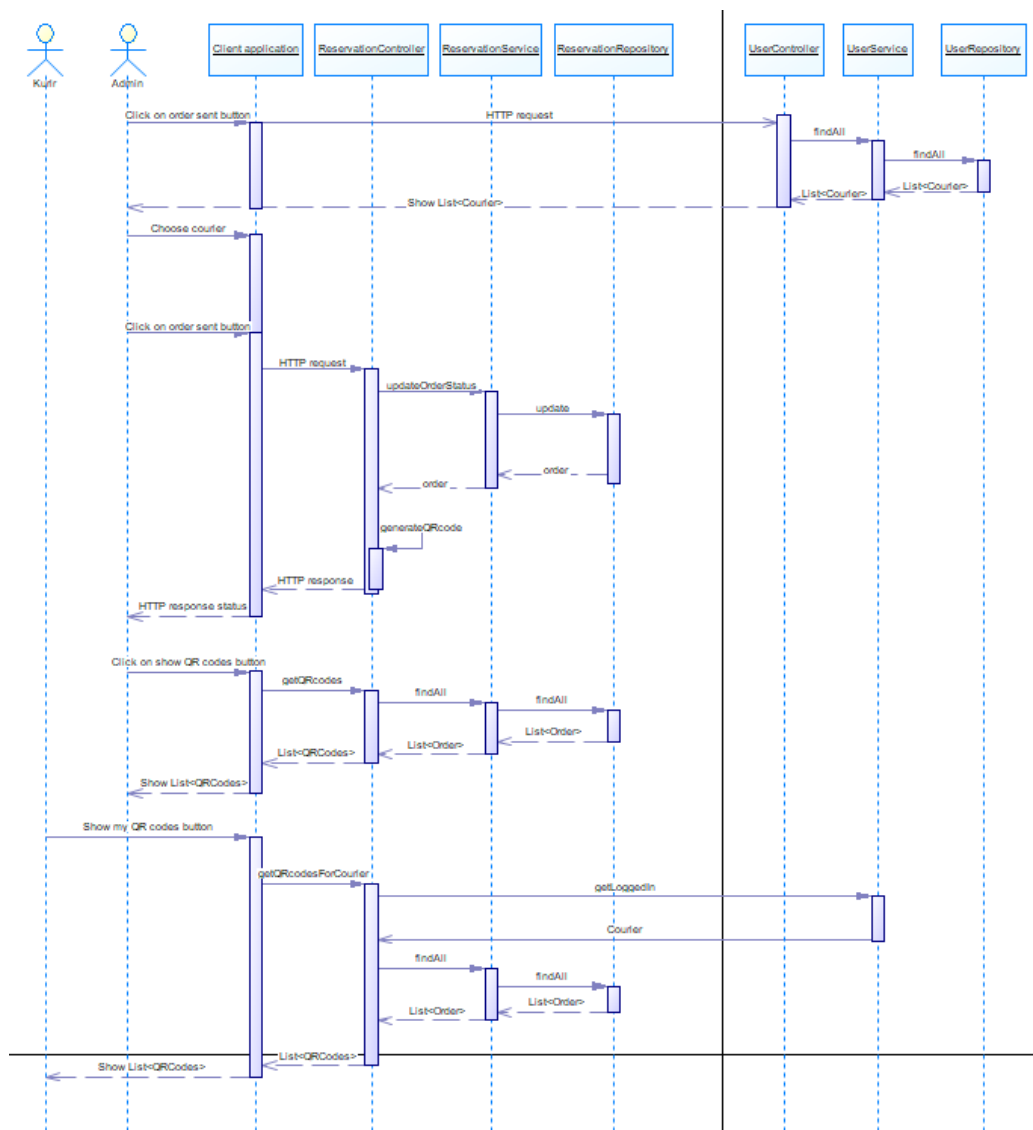


Slika 3.2.1 Dijagram klasa

### 3.3 Dijagram sekvence

Dijagram sekvence opisuje redosled razmenjivanja poruka između objekata kojim se realizuje odgovarajuća operacija u sistemu.

Kreiranje QR koda počinje na stranici sa svim porudžbinama kojoj može jedino Admin da pristupi. Samim tim, kreiranje QR koda može samo Admin da obavi. Pretpostavićemo da se na stranici sa svim porudžbinama već nalaze neke koje obavezno moraju imati status CREATED. Ukoliko je porudžbina označena sa DECLINED, DELIVERED ili SENT, dugme *Order sent* neće biti ni prikazano. Kod porudžbina koje su označene sa CREATED, klikom na dugme *Order sent* šalje se zahtev za dobavljanje svih kurira od kojih Admin mora da odabere onog preko kog će slati paket. Kada se dobavi lista kurira, Admin odabere željenog i klikom na dugme *Send* šalje HTTP zahtev na server i prolazi kroz *Order Controller*, *Service* i *Repository* kako bi promenio status sa CREATED u SENT. Nakon toga, iz kontrolera se poziva i metoda *createQRcode* koja u fajl sistemu čuva sliku QR koda sa svim potrebnim informacijama o poslatoj porudžbini. Kako bi Admin pregledao kreirani QR kod, mora otići na stranicu sa svim QR kodovima. U kontroleru se dobavljaju sve porudžbine sa podacima koji QR kod odgovara kojoj porudžbini i zatim se iz fajl sistema dobavljaju slike QR kodova i šalju u klijentski deo aplikacije. Isti postupak se dešava i ako kurir želi da pregleda svoje QR kodove, gde se proverava i koji tačno kurir je trenutno registrovan, kako bi se dobavili odgovarajući QR kodovi. Dijagram sekvence za ovaj proces prikazan je na slici 3.3.1.



Slika 3.3.1 Dijagram sekvence za kreiranje i prikaz QR koda

## 4.Opis implementacije

Aplikacija se sastoji iz dve celine, serverske i klijentske. Serverski deo aplikacije implementiran je u programskom jeziku Java, korišćenjem Spring okruženja i njegovih biblioteka. Za klijentski deo veb aplikacije korišćena je React biblioteka, dok je za klijentski deo mobilne aplikacije korišćena React Native biblioteka.

Serverski deo aplikacije zasnovan je na Spring MVC (*model, view, controller*) principu, gde je view sloj predstavljen kao klijentska aplikacija.

### 4.1 Serverski deo

Serverski deo aplikacije je podeljen u više slojeva, po ugledu na Spring MVC i oni predstavljaju:

- *Model* – klase koje opisuju modele podataka,
- *Repository* – objekti koji sadrže metode za pristup i komunikaciju sa bazom podataka,
- *Service* – servisni sloj, biznis logika aplikacije,
- *DTO (Data Transfer Object)* – objekti koji se prenose preko mreže i
- *Controller* – sloj za komunikaciju sa klijentom.

*Model* se sastoji iz četrnaest klasa. Interfejs *PersonUser* implementiraju klase *Admin*, *Courier* i *RegisteredUser* koje opisuju korisnike sistema. Svaki od ovih korisnika ima adresu koja je opisana klasom *Address*. *Item* klasa odnosi se na artikle kompanije i povezana je sa klasama *Pictures*, *AvailableColors*, *AvailableSize*, kao i *ItemInOrder* koja se odnosi na artikle u okviru porudžbine. Porudžbine su opisane klasom *Order*.

*Repository* sloj se sastoji iz repozitorijuma zaduženih za komunikaciju sa bazom podataka i obezbeđivanje podataka koji su predstavljeni modelima.

*Servisi* koriste repozitorijume radi komunikacije sa bazom i, zahvaljujući tome, omogućavaju izvršenje svih funkcionalnosti koje aplikacija nudi. Glavni servisi posmatrane aplikacije su *AdminService*, *AvailableColorsService*, *AvailableSizeService*, *ItemInOrderService*, *ColorsService*, *ItemService*, *ReservationService*, *PicturesService* i *RegisteredUserService*. Pored njih, postoje i dodatni servisi čija uloga je pomoćna i oni vrše posao obezbeđivanja dodatnih funkcionalnosti.

*DTO* sloj čini nekoliko klasa koje služe za prenos podataka. Klijent nije u dodiru sa podacima direktno iz baze, već se podaci šalju kao DTO objekti. U zavisnosti od potreba zahteva i odgovora, nastaju određene DTO klase sa odgovarajućim poljima.

Sloj *kontrolera* čine *AdminController*, *UserController*, *ItemsController*, *ReservationController* i *AuthenticationController* čija je uloga autentifikacija korisnika i obezbeđivanje registracije i prijave na sistem. Uloga kontrolera je svedena samo na komunikaciju sa klijentom i prosleđivanje parametara odgovarajućim servisima, gde se odvija cela biznis logika aplikacije.

#### 4.1.1 Model

Model čini četrnaest klasa, kao što je navedeno u prethodnom tekstu. Dve klase modela prikazane su na listinzima 4.1.1.1 i 4.1.1.2. Zbog preglednosti na slikama neće biti prikazane *property* metode kao ni *import* sekcija.

```
@Entity
@DiscriminatorValue("RegisteredUser")
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
public class RegisteredUser extends PersonUser{
```

```

@Column(name = "firstname", nullable = true)
private String name;

@Column(name = "surname", nullable = true)
private String surname;

@Column(name = "phoneNumber", nullable = true)
private String phoneNumber;

@ManyToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
private Address address;

@Column(name = "aboutUs", nullable = true)
private String aboutUs;

@JsonManagedReference
@OneToMany(mappedBy = "registeredUser", fetch = FetchType.LAZY, cascade = Cascade
adeType.ALL)
private Set<Order> orders = new HashSet<Order>();

public RegisteredUser(String name, String surname, String phoneNumber, Address
address) {
    this.name = name;
    this.surname = surname;
    this.phoneNumber = phoneNumber;
    this.address = address;
}

public RegisteredUser() {
}

```

#### Listing 4.1.1.1 Model RegisteredUser klase

```

@Entity
@Table(name="order_table")
public class Order {

    @Id
    @GeneratedValue
    @Column(name="id", unique=true, nullable=false)
    private Integer id;

    @JsonBackReference
    @ManyToOne(cascade = CascadeType.MERGE, fetch = FetchType.LAZY)

```



```

    @JoinColumn(name = "registeredUser_id", referencedColumnName = "id", nullable
= true,        unique = false)
    private RegisteredUser registeredUser;

    @ManyToMany
    @JsonIgnore
    @JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class, prop
erty = "id")
    @JoinTable(name = "items_in_order", joinColumns = @JoinColumn(name = "item_id"
, referencedColumnName = "id"),
        inverseJoinColumns = @JoinColumn(name = "order_id", referencedColumnName
me = "id"))
    private Set<ItemInOrder> items = new HashSet<ItemInOrder>();

    @Column(name = "dueDate", nullable = true)
    private LocalDate dueDate;

    @Column(name = "dateOfReservation", nullable = true)
    private LocalDate dateOfReservation;

    @Column(name = "itemId", nullable = true)
    private Integer itemId;

    @Column(name = "qrCode", nullable = true)
    private String qrCode;

    @Column(name = "status", nullable = true)
    private String status;

    @JsonBackReference
    @ManyToOne(cascade = CascadeType.MERGE, fetch = FetchType.LAZY)
    @JoinColumn(name = "courier_id", referencedColumnName = "id", nullable = true,
unique = false)
    private Courier courier;

    public Order() {
    }

```

**Listing 4.1.1.2 Model Order klase**

## 4.1.2 Repository

Sloj repozitorijuma čini dvanaest interfejsa koji obezbeđuju potrebnu komunikaciju sa bazom. Za konkretnu implementaciju repozitorijuma korišćen

je *JpaRepository*. Na listinzima 4.1.2.1 i 4.1.2.2 mogu se videti implementacije dva repozitorijuma.

```
public interface ItemRepository extends JpaRepository<Item, Integer> {  
    Item findByName(String name);  
}
```

**Listing 4.1.2.1 ItemRepository**

```
public interface RegisteredUserRepository extends JpaRepository<RegisteredUser, Integer> {  
    RegisteredUser findByEmail( String email );  
}
```

**Listing 4.1.2.2 RegisteredUserRepository**

### 4.1.3 Servis

Servisi predstavljaju sloj interfejsa i klasa koje implementiraju interfejse. Ovaj sloj zadužen je za čitavu poslovnu logiku aplikacije i predstavlja vezu između kontrolera i repozitorijuma. Takođe, u servisima se vrši konverzija objekata modela u DTO objekte, pogodne za rad sa klijentom, a moguća je konverzija i u suprotnom pravcu. Na listinzima 4.1.3.1 i 4.1.3.2 se vidi jedan par interfejsa i implementacije interfejsa. Kao što se može primetiti, interfejs sadrži samo deklarisanе metode, dok je implementacija interfejsa zadužena za celu logiku.

```
public interface IRegisteredUserService {  
    RegisteredUser findById(Integer id);  
    List<RegisteredUser> findAll ();  
    RegisteredUser save(PersonUserDTO userRequest);  
    RegisteredUser update(PersonUserDTO personUserDTO);  
    RegisteredUser findByEmail(String email);  
}
```

**Listing 4.1.3.1 IRegisteredUserService interfejs**

```

@Service
public class RegisteredUserService implements IRegisteredUserService {

    @Autowired
    private RegisteredUserRepository registeredUserRepository;

    @Autowired
    private CourirRepository courirRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Autowired
    private AuthenticationManager authenticationManager;

    @Override
    public RegisteredUser findById(Integer id) {
        return registeredUserRepository.findById(id).get();
    }

    @Override
    public List<RegisteredUser> findAll() throws AccessDeniedException {
        List<RegisteredUser> result = registeredUserRepository.findAll();
        return result;
    }

    @Override
    public RegisteredUser save(PersonUserDTO userRequest) {
        return null;
    }

    @Override
    public RegisteredUser update(PersonUserDTO personUserDTO) {
        RegisteredUser user = getLoggedUser();
        user.setName(personUserDTO.getFirstname());
        user.setSurname(personUserDTO.getSurname());
        user.setPhoneNumber(personUserDTO.getPhonenumber());
        user.setAboutUs(personUserDTO.getAboutUs());
        com.apeironapp.apeironapp.Model.Address address = new Address();
        address.setCity(personUserDTO.getAddress().getCity());
        address.setCountry(personUserDTO.getAddress().getCountry());
        address.setLatitude(personUserDTO.getAddress().getLatitude());
        address.setLongitude(personUserDTO.getAddress().getLongitude());
        address.setStreet(personUserDTO.getAddress().getStreet());
        user.setAddress(address);
        return registeredUserRepository.save(user);
    }
}

```

```

@Override
public RegisteredUser findByEmail(String email) throws UsernameNotFoundException {
    RegisteredUser u = registeredUserRepository.findByEmail(email);
    return u;
}

public PersonUserDTO getUserDTO() {
    RegisteredUser user = getLoggedInUser();

    PersonUserDTO userDTO = new PersonUserDTO();
    userDTO.setEmail(user.getEmail());
    userDTO.setFirstname(user.getName());
    userDTO.setSurname(user.getSurname());
    userDTO.setId(user.getId());
    userDTO.setPhonenumber(user.getPhoneNumber());
    AddressDTO addressDTO = new AddressDTO();
    addressDTO.setCity(user.getAddress().getCity());
    addressDTO.setCountry(user.getAddress().getCountry());
    addressDTO.setLatitude(user.getAddress().getLatitude());
    addressDTO.setLongitude(user.getAddress().getLongitude());
    addressDTO.setStreet(user.getAddress().getStreet());
    userDTO.setAddress(addressDTO);
    return userDTO;
}

public RegisteredUser changePassword(String oldPassword, String newPassword) {
    RegisteredUser user = getLoggedInUser();
    authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(user.getEmail(), oldPassword));

    if(newPassword.isEmpty())
        throw new IllegalArgumentException("Invalid new password");
    user.setPassword(passwordEncoder.encode(newPassword));
    return registeredUserRepository.save(user);
}

public RegisteredUser getLoggedInUser() {
    Authentication currentUser = SecurityContextHolder.getContext().getAuthentication();
    String email = currentUser.getName();
    return registeredUserRepository.findByEmail(email);
}

```

```

public Courier getLoggedInUser () {

    Authentication currentUser = SecurityContextHolder.getContext().getAuthent
ication();
    String email = currentUser.getName();
    return courirRepository.findByEmail(email);

}
}

```

**Listing 4.1.3.2 RegisteredUserService implementacija**

#### 4.1.4 DTO

DTO (Data Transfer Object) sloj čine objekti za prenos podataka preko mreže. Objekti u ovom sloju sadrže samo ona polja klasa koja će biti prikazana korisniku na klijentskoj strani aplikacije. Glavna prednost DTO objekata je smanjenje podataka koje je potrebno poslati preko mreže i mogućnost objedinjavanje podataka za koje bi potencijalno bilo potrebno uputiti više zahteva. Listing 4.1.4.1 prikazuje jednu DTO klasu koja služi za prenos podataka koji su vezani za artikle u okviru jedne porudžbine.

```

public class ItemInOrderDTO {

    private Integer itemId;

    private String color;

    private String size;

    private LocalDate date;

    private String quantity;

    public ItemInOrderDTO() {

    }
}

```

**Listing 4.1.4.1 ItemInOrderDTO**

### 4.1.5 Controller

Sloj kontrolera predstavlja posrednika između klijentskog dela aplikacije i servisnog sloja aplikacije. Kontrolera prihvata zahteva klijenata, prosleđivanje zahteva servisnom sloju za obradu i vraćanje odgovora klijentu. Komunikacija između kontrolera i servisnog dela aplikacije se isključivo obavlja pomoću DTO objekata. Sloj kontrolera posmatrane aplikacije čine AdminController, UserController, ItemsController, ReservationController i AuthenticationController. Na listingu 4.1.5.1 može se videti primer implementacije kontrolera. Prikazan je UserController sa nekoliko svojih metoda.

```
@RestController
@RequestMapping(value = "api/users")
@CrossOrigin(origins = "*", allowedHeaders = "*")
public class UserController {

    @Autowired
    private Environment environment;

    @Autowired
    JavaMailSenderImpl mailSender;

    @Autowired
    private UserServiceImpl userService;

    @Autowired
    private RegisteredUserService registeredUserService;

    @GetMapping("/couriers")
    @PreAuthorize("hasRole('ROLE_ADMIN')")
    ResponseEntity<List<Courier>> getAllCouriers()
    {
        List<Courier> couriers = userService.findAllDelivery();
        return couriers == null ?
            new ResponseEntity<>(HttpStatus.NOT_FOUND) :
            ResponseEntity.ok(couriers);
    }

    @GetMapping("/delete/{id}")
    @PreAuthorize("hasRole('ROLE_ADMIN')")
```

```

public ResponseEntity<String> delete(@PathVariable Integer id)
{
    userService.delete(id);
    return new ResponseEntity<>(HttpStatus.NOT_FOUND);
}

@PutMapping("/update")
@PreAuthorize("hasRole('ROLE_USER')")
public ResponseEntity<?> updateUser(@RequestBody PersonUserDTO userDTO) {
    try {
        registeredUserService.update(userDTO);
        return new ResponseEntity<>(HttpStatus.NO_CONTENT);
    } catch (IllegalArgumentException e) {
        return new ResponseEntity<>(e.getMessage(), HttpStatus.BAD_REQUEST);
    } catch (Exception e) {
        e.printStackTrace();
        return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
    }
}

@PostMapping("/changePassword")
@PreAuthorize("hasRole('ROLE_USER')")
public ResponseEntity<?> changePasswordUser(@RequestBody PasswordChanger passwordChanger) {
    try {
        registeredUserService.changePassword(passwordChanger.getOldPassword(), passwordChanger.getNewPassword());
        return new ResponseEntity<>(HttpStatus.NO_CONTENT);
    } catch (IllegalArgumentException e) {
        return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }
    catch (Exception e) {
        return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
}

```

#### Listing 4.1.5.1 UserController

Uz postojanje klasičnih metoda, implementirane su i metode za slanje e-mail poruka i kreiranje QR kodova. Na listingu 4.1.5.2 prikazana je metoda koja, zahvaljujući podešavanjima u *application.properties* datoteci, pruža mogućnost slanja mailova.

```

@PostMapping("/sendMessage")
public ResponseEntity<?> sendMessage(@RequestBody MessageDTO messageDTO) {

    SimpleMailMessage mail = new SimpleMailMessage();
    mail.setTo(environment.getProperty("spring.mail.username"));
}

```

```

        mail.setSubject("Contact message!");
        mail.setFrom(messageDTO.getEmail());
        mail.setText("You have a message from : "
            + messageDTO.getFirstName() + " " + messageDTO.getSurname() + " : " + messageDTO.getMessage() + "." +
            "\n Phone number is " + messageDTO.getPhoneNumber()+".");

        mailSender.send(mail);

        return new ResponseEntity<>(HttpStatus.NO_CONTENT);

```

#### Listing 4.1.5.2 Slanje maila

Još jedna važna funkcionalnost predstavlja formiranje QR koda kao slike. Pomoću BarcodeFormat-era i MatrixToImageWriter-a QR kod se formira i kasnije prikazuje na klijentskom delu aplikacije. Očitavanjem istog, dobija se obaveštenje da je pošiljka uspešno isporučena i status pošiljke se menja sa SENT u DELIVERED.

Metode *generateQRCode()* i *confirmOrder()* koje su zadužene za prethodno opisane funkcionalnosti, prikazane su u listingu 4.1.5.3.

```

@GetMapping("/generateQR/{id}/{courier}")
@CrossOrigin
public void generateQRcode(@PathVariable Integer id,@PathVariable Integer courier)
    throws WriterException, IOException
{

    Order order = reservationService.findById(id);
    Courier courier1 = userService.findByIdCourier(courier);

    String path = "src/main/resources/images/" + id + ".png";
    order.setQr(path);
    order.setStatus("SENT");
    order.setCourier(courier1);

    reservationService.update(order);

    String charset = "UTF-8";
    Map<EncodeHintType, ErrorCorrectionLevel> hashMap = new HashMap<EncodeHintType
, ErrorCorrectionLevel>();
    hashMap.put(EncodeHintType.ERROR_CORRECTION, ErrorCorrectionLevel.L);

```



```

        String message = "You have successfully received your order number "+order.getItemId();

        BitMatrix matrix = new MultiFormatWriter().encode(message, BarcodeFormat.QR_CODE, 200, 200);
        MatrixToImageWriter.writeToFile(matrix, path.substring(path.lastIndexOf('.') + 1), new File(path));
    }

@GetMapping("/confirmOrder/{data}")
public ResponseEntity<String> confirmOrder(@PathVariable String data) {

    String[] split = data.split(" ");
    String orderNumber = split[7];

    Order order = reservationService.findById(Integer.parseInt(orderNumber));
    order.setStatus("DELIVERED");
    reservationService.update(order);

    return new ResponseEntity<>("Order is successfully delivered.",HttpStatus.OK);
}

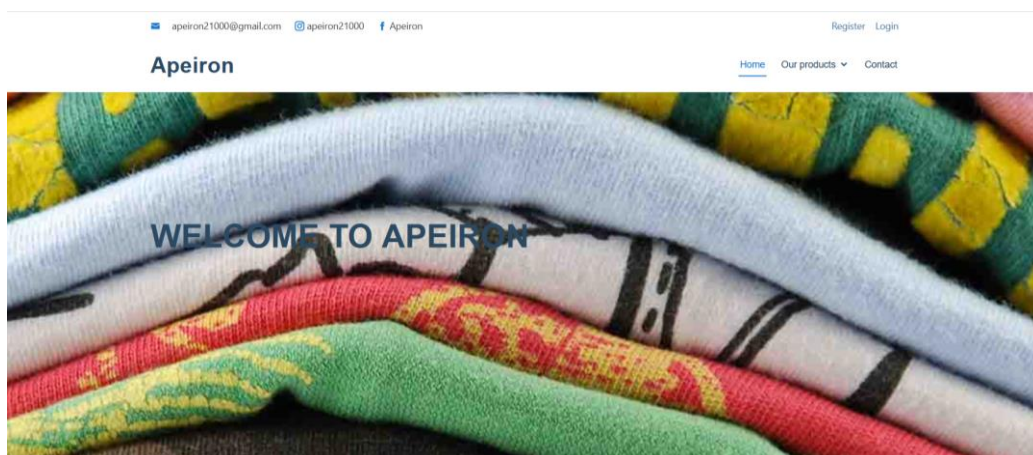
```

**Listing 4.1.5.3 generateQRCode() i confitmOrder() metode**

## 4.2 Klijentski deo

Klijentski deo veb aplikacije je rađen uz pomoć React biblioteke JavaScript jezika, dok je klijentski deo mobilne aplikacije rađen uz pomoć React Native biblioteke. Ovaj deo aplikacije predstavlja *view* komponentu arhitekture i omogućava nam prikaz podataka iz sistema. Veb aplikacija je *Single Page Application*, što znači da jedna stranica služi za prikaz različitih komponenti koje se dinamički smenjuju promenom putanja i na taj način pružaju korisniku sve potrebne informacije. Za uniformniji dizajn veb aplikacije i lakšu implementaciju dizajna korišćen je React Bootstrap radni okvir.

Prilikom otvaranja veb sajta, korisniku je prikazana početna stranica i meni iz kog može da bira pregled svih proizvoda, registraciju, prijavu ili slanje poruke kompaniji (slika 4.2.1).



**Slika 4.2.1 HomePage**

Nakon klika na dugme za registraciju ili prijavu na sistem, otvara se jedna od stranica, gde korisnik može uneti svoje podatke i poslati zahtev za registraciju ili prijavu. Na Slikama 4.2.2 i 4.2.3 su prikazani prozori sa formama za unos podataka.

The image displays two side-by-side forms on the Apeiron website. The left form is titled 'REGISTRATION' and contains the following input fields: 'Email address', 'Name', 'Surname', 'Address', 'Phone number', 'Password', and 'Repeat password'. A blue 'Sign Up' button is located at the bottom of this form. The right form is titled 'LOGIN' and contains two input fields: 'Email address' and 'Password'. A blue 'Login' button is positioned below the 'Password' field.

### Slike 4.2.2 i 4.2.3 Forme za prijavu i registraciju

Na Listingu 4.2.4 nalazi se implementacija LoginPage-a. LoginPage u sebi sadrži *handleLogin()* metodu koja šalje HTTP zahtev klikom na Login dugme i prosleđuje na taj način serveru podatke o korisniku. U zavisnosti od odgovora servera, korisnik će ili biti prijavljen na sistem i preusmeren na početnu stranicu, ili će mu izaći obaveštenje o grešci prilikom prijavljivanja. Ukoliko je korisnik uspešno prijavljen, u *localStorage*-u se čuva njegov JWT (*Json Web Token*) koji se šalje uz svaki sledeći izvršen zahtev serveru radi autentifikacije korisnika.

```
class LoginPage extends Component {
  state = {
    errorHandler: "",
    errorMessage: "",
    hiddenErrorAlert: true,
    email: "",
    password: "",
    redirect: false,
    emailError: "none",
    passwordError: "none",
  };

  handleEmailChange = (event) => {
    this.setState({ email: event.target.value });
  };

  handlePasswordChange = (event) => {
    this.setState({ password: event.target.value });
  };

  handleLogin = () => {
    this.setState({ hiddenErrorAlert: true, emailError: "none", passwordError: "none" });

    if (this.validateForm()) {
      let loginDTO = { email: this.state.email, password: this.state.password };
      Axios.post(BASE_URL + "/api/auth/login", loginDTO, { validateStatus: () => true })
        .then((res) => {
          if (res.status === 500) {
            this.setState({ errorHandler: "Bad credentials!", errorMessage: "Wrong username or password.", hiddenErrorAlert: false });
          } else {
            localStorage.setItem("keyToken", res.data.accessToken);
          }
        });
    }
  };
}
```

```

        localStorage.setItem("keyRole", JSON.stringify(res.data.role
s));
        localStorage.setItem("expireTime", new Date(new Date().getTi
me() + res.data.expiresIn).getTime());
        this.setState({ redirect: true });
    }
    })
    .catch((err) => {
        this.setState({ errorHandler: "Error!", errorMessage: "Something
went wrong, please try again.", hiddenErrorAlert: false });
    });
}
};

validateForm = () => {
    if (this.state.email === "") {
        this.setState({ emailError: "inline" });
        return false;
    } else if (this.state.password === "") {
        this.setState({ passwordError: "inline" });
        return false;
    }

    return true;
};

handleCloseAlert = () => {
    this.setState({ hiddenErrorAlert: true });
};

render() {
    if (this.state.redirect) return <Redirect push to="/" />;
    return (
        <React.Fragment>
            <TopBar />
            <Header />

            <div className="container" style={{ marginTop: "10%" }}>
                <HeadingAlert
                    hidden={this.state.hiddenErrorAlert}
                    header={this.state.errorHeader}
                    message={this.state.errorMessage}
                    handleCloseAlert={this.handleCloseAlert}
                />
                <h5 className=" text-center mb-0 text-
uppercase" style={{ marginTop: "2rem" }}>
                    Login
                </h5>

                <div className="row section-design">
                    <div className="col-lg-8 mx-auto">

```

```

        <br />
        <form id="contactForm" name="sendMessage" novalidate="no
validate">
            <div className="control-group">
                <div className="form-group controls mb-0 pb-
2" style={{ color: "#6c757d", opacity: 1 }}>
                    <input
                        placeholder="Email address"
                        className="form-control"
                        id="name"
                        type="text"
                        onChange={this.handleEmailChange}
                        value={this.state.email}
                    />
                </div>
                <div className="text-
danger" style={{ display: this.state.emailError }}>
                    Email must be entered.
                </div>
            </div>

            <div className="control-group">
                <div className="form-group controls mb-0 pb-
2" style={{ color: "#6c757d", opacity: 1 }}>
                    <input
                        placeholder="Password"
                        className="form-control"
                        id="password"
                        type="password"
                        onChange={this.handlePasswordChange}
                        value={this.state.password}
                    />
                </div>
                <div className="text-
danger" style={{ display: this.state.passwordError }}>
                    Password must be entered.
                </div>
            </div>

            <div className="form-group">
                <Button
                    style={{ background: "#1977cc", marginTop: "
15px", marginLeft: "40%", width: "20%" }}
                    onClick={this.handleLogin}
                    className="btn btn-primary btn-xl"
                    id="sendMessageButton"
                    type="button"
                >
                    Login
                </Button>
            </div>

```

```

        </form>
      </div>
    </div>
  </div>
</React.Fragment>
  );
}
}



export default LoginPage;

```

#### Listing 4.2.4 Login komponenta

Prijavljen korisnik klikom na željenu vrstu artikala iz Header-a može biti preusmeren na stranicu sa svim muškim ili ženskim majicama, duksevima ili kapama kao što je prikazano na slici 4.2.5.

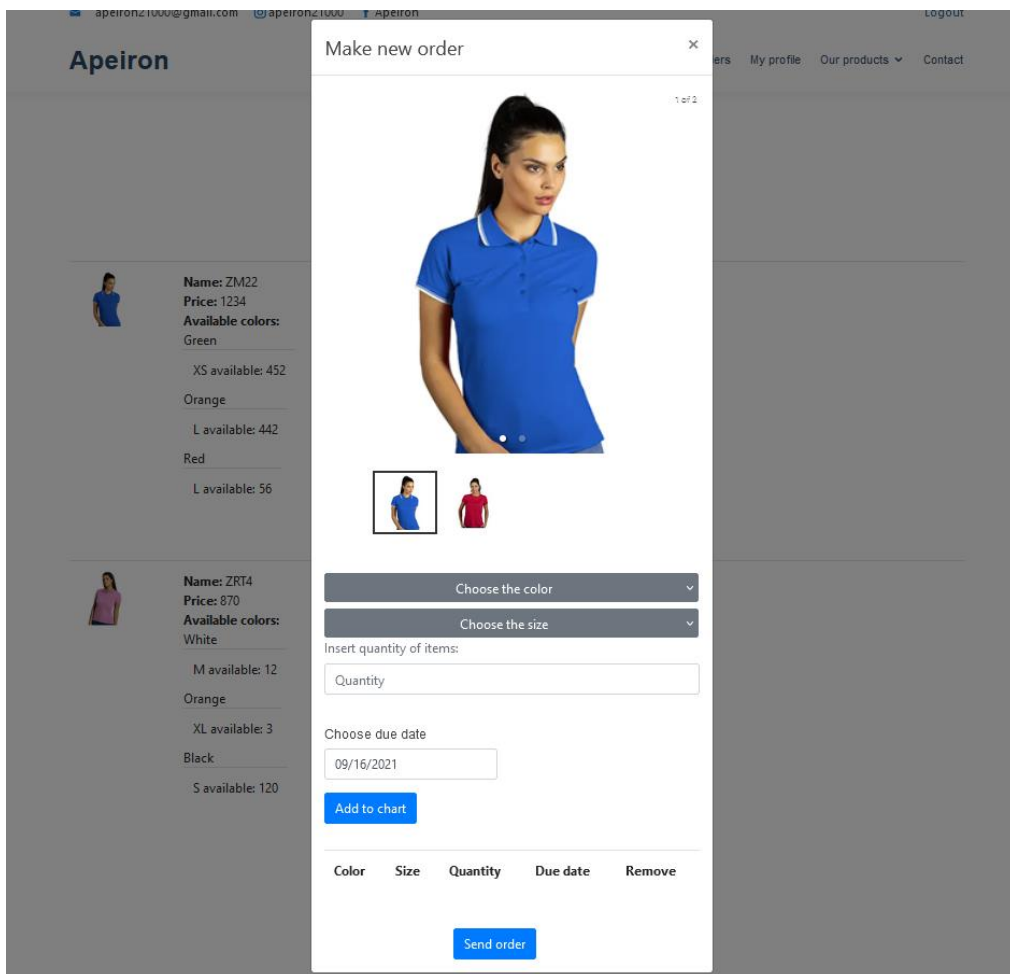
## WOMEN T-SHIRTS

	<b>Name:</b> ZM22 <b>Price:</b> 1234 <b>Available colors:</b> <div>Green</div> <div>XS available: 452</div> <div>Orange</div> <div>L available: 442</div> <div>Red</div> <div>L available: 56</div>
<div>Order</div>	
	<b>Name:</b> ZRT4 <b>Price:</b> 870 <b>Available colors:</b> <div>Orange</div> <div>XL available: 3</div> <div>White</div> <div>M available: 12</div> <div>Black</div> <div>S available: 120</div>
<div>Order</div>	

### Slika 4.2.5 Prikaz svih ženskih majica

Svaki registrovani korisnik zatim može poručiti željeni artikal klikom na dugme *Order*. Tom akcijom otvoriće se modalni dijalog u kom je potrebno ispuniti zahtevana polja kako bi porudžbina bila uspešno poslata. Ukoliko se ne popune neka od polja, korisnik će dobiti obaveštenje da je njih obavezno popuniti.

Od klijenta se traži da odabere boju, veličinu i količinu odabranog artikla, kao i datum do kada želi da roba stigne. Klikom na sliku robe omogućeno je listanje slika, kako bi kupac video sve ponuđene boje željenog artikla (slika 4.2.6)



**Slika 4.2.6** Modalni dijalog za poručivanje artikala

Osim mogućnosti pregleda i izmene svog profila, svaki registrovani korisnik i kurir mogu da šalju poruke Apeironu, koje kompaniji stižu na email adresu. Korisnik tu stranicu vidi na način prikazan na slici 4.2.7, dok je admin vidi na nešto drugačiji način, prikazan na slici 4.2.8. Klikom na dugme Edit info, polja koja su do tad bila uneditable, postaju editable i podatke je moguće menjati (slika 4.2.9).



CONTACT INFORMATION

admin@example.com

Serbia, Autonomna Pokrajina Vojvodina, Južnobački upravni okrug, grad Novi Sad, grad Novi Sad, Ulica Pavla Simića

065897999

This is a company that sells advertising material.

CONTACT US

First name

Last name

Email

Phone number

Message

Send

**Slika 4.2.7** Contact stranica za registrovanog korisnika

CONTACT INFORMATION

admin@example.com

Serbia, Autonomna Pokrajina Vojvodina, Južnobački upravni okrug, grad Novi Sad, grad Novi Sad, Ulica Pavla Simića

065897999

This is a company that sells advertising material.

Edit Info

**Slika 4.2.8** Contact stranica za admina

CONTACT INFORMATION

admin@example.com

Serbia, Autonomna Pokrajina Vojvodina, Južnobački upravni okrug, grad Novi Sad, grad Novi Sad, Ulica Pavla Simića

065897999

This is a company that sells advertising material.

Change information

**Slika 4.2.9** Izmena kontakt informacija za admina

Na listingu 4.2.10 nalazi se prikaz metode *handleChangeInfo()*. Uz pomoć *axios* dodatka, koji služi za razmenu HTTP zahteva klijenta sa serverom, na server se šalju podaci koji su u ovom primeru izdvojeni u *userDTO* varijabli. Polja su definisana tako da ispunjavaju standarde koje zahteva DTO objekat na serveru za primanje podataka. Ukoliko je zahtev prošao uspešno, u odgovoru se dobijaju novi podaci koji se zatim čuvaju u *state* delu komponente. Na taj način korisniku je omogućen automatski prikaz najnovijih podataka.

Takođe, interesantna je upotreba *Ymaps*-a pomoću koje je korisnicima omogućeno da pretražuju adresu koju unose prilikom registracije ili izmene profila.

```
handleChangeInfo = () => {  
  
  let street;  
  let city;  
  let country;  
  let latitude;  
  let longitude;  
  let found = true;  
  
  this.ymaps  
    .geocode(this.addressInput.current.value, {  
      results: 1,  
    })  
    .then(function (res) {  
      if (typeof res.geoObjects.get(0) === "undefined") found = false;  
      else {
```

```

        var firstGeoObject = res.geoObjects.get(0),
        coords = firstGeoObject.geometry.getCoordinates();
        latitude = coords[0];
        longitude = coords[1];
        country = firstGeoObject.getCountry();
        street = firstGeoObject.getThoroughfare();
        city = firstGeoObject.getLocalities().join(", ");
    }
})
.then((res) => {
    let userDTO = {
        address: { street, country, city, latitude, longitude },
        phoneNumber: this.state.phoneNumber,
        contactEmail: this.state.email,
        aboutUs: this.state.aboutUs,
    };

    if (this.validateForm(userDTO)) {
        if (found === false) {
            this.setState({ addressNotFoundError: "initial" });
        } else {
            Axios.put(BASE_URL + "/api/admin/update", userDTO, {
                validateStatus: () => true,
                headers: { Authorization: getAuthHeader() },
            })
                .then((res) => {
                    if (res.status === 400) {
                        this.setState({ hiddenFailAlert: false, failHeader:
"Bad request", failMessage: "Invalid argument." });
                    } else if (res.status === 500) {
                        this.setState({ hiddenFailAlert: false, failHeader:
"Internal server error", failMessage: "Server error." });
                    } else if (res.status === 204) {
                        this.setState({
                            hiddenSuccessAlert: false,
                            successHeader: "Success",
                            successMessage: "You successfully updated your i
nformation.",
                            hiddenEditInfo: true,
                        });
                    }
                })
                .catch((err) => {
                    console.log(err);
                });
        }
    }
});
};

```

#### Listing 4.2.10 Metoda za slanje izmenjenih podataka

U nastavku će funkcionalnosti biti objašnjene u React Native jeziku. Admin odabirom stavke Add Item iz menija, prelazi na stranicu koja je prikazana na slikama 4.2.11 i 4.2.12. Stranica je napravljena da bude scrollable (počinje tagom <ScrollView>). Prilikom dodavanja, admin može odabrati sliku iz galerije telefona, čija je implementacija prikazana na listingu 4.2.13.

The image contains two side-by-side screenshots of a mobile application interface for adding a new item. Both screenshots show a form with a back arrow and a menu icon at the top. The left screenshot (Figure 4.2.11) shows the initial form with a 'CHOOSE IMAGE' button, a 'Choose an item type' dropdown, and input fields for 'Name', 'Price', 'Color', 'Size', and 'Quantity'. The right screenshot (Figure 4.2.12) shows the form after selection, with radio buttons for 'Male' and 'Female', a 'Price' field, and a table with columns 'Color', 'Size', 'Quantity', and 'Remove'. A red 'ADD ITEM' button is at the bottom.

Slike 4.2.11 i 4.2.12 Dodavanje novog artikla

```
imageGalleryLaunch = () => {  
  let options = {  
    storageOptions: {  
      skipBackup: true,  
      path: 'images',  
    },  
  };  
  ImagePicker.launchImageLibrary({}, options);  
}
```

```

    },
  };

  launchImageLibrary(options, (res) => {

    if (res.didCancel) {
      console.log('User cancelled image picker');
    } else if (res.error) {
      console.log('ImagePicker Error: ', res.error);
    } else if (res.customButton) {
      console.log('User tapped custom button: ', res.customButton);
      alert(res.customButton);
    } else {
      const source = { uri: res.uri };
      this.setState({
        filePath: res.assets[0].uri,
        fileData: res,
        fileUri: res.uri
      });
    }
  });
}

{this.state.filePath !== "" &&
  <Image
    source={{ uri: this.state.filePath }}
    style={{ width: 200, height: 200 }}
  />

  <TouchableHighlight
    style={{
      height: 40,
      width: 300,
      borderRadius: 80,
      marginLeft: 50,
      marginRight: 50,
      marginTop: 20
    }}>
    <Button
      onPress={this.imageGalleryLaunch}
      type="button"
      title=" Choose image"
    >
  </Button>
</TouchableHighlight>
}

```

**Listing 4.2.13 Dodavanje slike**

Drop down meni i Radio Buttons polja se uz pomoć React Native biblioteke implementiraju na nešto drugačiji način u odnosu na način u React.js, što je prikazano na listingu 4.2.14.

```
<View>
  <Picker
    class="btn btn-secondary dropdown-toggle"
    aria-haspopup="true"
    aria-expanded="false"
    onChange={(e) => this.handleTypeChange(e)}
    selectedValue={this.state.selectedType}
  >
    <Picker.Item style={{ marginLeft: "12rem" }} label="Choose an item type" value="Choose an item type">
    </Picker.Item>
    {this.state.showedTypes.map((chain) => (
      <Picker.Item key={chain} label={chain} value={chain}></Picker.Item>
    ))} </Picker>
</View>

<View style={styles.text}>
  <RadioButton.Group
    onChange={value => this.setState({ gender: value })}
    value={this.state.gender}
  >
    <View>
      <RadioButton value='Male' label='Male' onPress={(e) => this.handleGenderChange('Male')} /><Text>Male</Text>
    </View>
    <View>
      <RadioButton value='Female' label='Female' onPress={(e) => this.handleGenderChange('Female')} /><Text>Female</Text>
    </View>
  </RadioButton.Group>
</View>
```

**Listing 4.2.13 DropDown meni i Radio Buttons**

Još jedna implementacija koja se razlikuje od one u React-u je formiranje tabele koje su u ovom projektu formirane importovanjem *react-native-paper* biblioteke. Prikaz koda tabele demonstriran je u listingu 4.2.14.

```

<DataTable>
  <DataTable.Header>
    <DataTable.Title>Color</DataTable.Title>
    <DataTable.Title >Size</DataTable.Title>
    <DataTable.Title >Quantity</DataTable.Title>
    <DataTable.Title >Remove</DataTable.Title>
  </DataTable.Header>

  {this.state.orders.map((c) => (
    <DataTable.Row>
      <DataTable.Cell>{c.color}</DataTable.Cell>
      <DataTable.Cell >{c.size}</DataTable.Cell>
      <DataTable.Cell >{c.quantity}</DataTable.Cell>
      <DataTable.Cell ><Button className="mt-
3" onPress={ (e) => this.handleRemoveChange(e, c)} title="Remove">

        </Button></DataTable.Cell>
      </DataTable.Row>
    )
  )}
</DataTable>

```

**Listing 4.2.14 Kreiranje tabele**

Kao i veb, mobilna aplikacija koristi velik broj modalnih dialoga gde podatke iz jedne komponente prenosimo u drugu pomoću *props*-a. Način dobavljanja podataka kroz *props* prikazano je u listingu 4.2.15.

```

render() {
  const {isVisible, handleModalClose } = this.props;

  return (

    isVisible &&

    <Modal isVisible={true} transparent={false}>
  </Modal>

  );
}

```

**Listing 4.2.15 Modalni dijalog i props**

Pretraga lokacije koja se unosi prilikom registracije ili izmene profila korisnika i kurirskih službi u mobilnoj aplikaciji implementirana je pomoću Google mapa, a pretraga sa predlogom lokacija ostvarena je importovanjem GooglePlacesAutocomplete biblioteke (listing 4.2.16)

```
<GooglePlacesAutocomplete

    fetchDetails={true}
    placeholder="Search the address"
    onPress={({data, details = null}) => {
this.setState({ street: details.formatted_address,
    city: details.getPlace,
    country: details.getCountry,
    latitute : details.geometry.location.lat,
    longitude: details.geometry.location.lng})
    }}
    query={{
      key: 'AIzaSyAsno5WRMrrU_XX-ur8CBRtndECG-0kAfs',
      language: 'en',
    }}
    styles={{
      textInputContainer: {
        backgroundColor: 'grey',
      },
      textInput: {
        height: 38,
        color: '#5d5d5d',
        fontSize: 16,
      }, predefinedPlacesDescription: {
        color: '#1faadb',
      },
    }}

/>
```

**Listing 4.2.16 GooglePlacesAutocomplete**

Vrlo važna komponenta projekta predstavlja skeniranje QR koda koje je omogućeno jedino na mobilnoj aplikaciji. Kod koji ovo omogućava predstavljen je na listingu 4.2.17.

```
export default function App() {
  const [hasPermission, setHasPermission] = useState(null);
  const [scanned, setScanned] = useState(false);
```



```

useEffect(() => {
  (async () => {
    const { status } = await BarCodeScanner.requestPermissionsAsync();
    setHasPermission(status === 'granted');
  })();
}, []);

const handleBarCodeScanned = ({ type, data }) => {
  setScanned(true);
  alert(`${data}`);
  Axios.get(BASE_URL + "/api/reservations/confirmOrder/" + data)
    .then((res) => {
      console.log(res.data);
    })
    .catch((err) => {
      console.log(err);
    });
};

if (hasPermission === null) {
  return <Text>Requesting for camera permission</Text>;
}
if (hasPermission === false) {
  return <Text>No access to camera</Text>;
}

return (
  <View style={styles.container}>
    <BarCodeScanner
      onBarCodeScanned={scanned ? undefined : handleBarCodeScanned}
      style={StyleSheet.absoluteFillObject}
    />
    {scanned && <Button title={'Tap to Scan Again'} onPress={() => setScanned(false)} />}
  </View>
);
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    flexDirection: 'column',
    justifyContent: 'center',
  },
});

```

**Listing 4.2.17** Skeniranje QR koda



## 5. Zaključak

Projekat posmatran u radu predstavlja aplikaciju za omogućavanje kupovine preko interneta, evidenciju te kupovine i slanje poruke kuririma da preuzmu paket, kao i potvrda porudžbine putem skeniranja QR koda. Osim osnovnih funkcionalnosti aplikacije, koje podržavaju prijavu i registraciju korisnika na sistem, ažuriranje, pregled i brisanje artikala, bitan deo čine e-mail poruke koje stižu adminu.

Postoji potencijal za dalji razvoj i implementaciju novih funkcionalnosti koje bi se mogle pokazati kao korisne i potrebne. Za sada, ona pomaže korisnicima da pronađu željeni artikal, da ga jednostavno i bezbedno poruče, da se automatski kontaktira kurirska služba i da prodavac ima potpuni uvid u stanje svih porudžbina. Dalji razvoj aplikacije mogao bi se unaprediti u smislu automatske ponude sličnih i kompatibilnih artikala sa onim koji je već kupljen, komunikacije sa drugim korisnicima uz razmenu iskustava vezanim za korišćenje robe, praćenje stanja pošiljke - gde se nalazi, kada je poslata, obračuna popusta za kupovinu preko određenog iznosa...

Aplikacija je rađena u savremenim tehnologijama koje se sve više proširuju i napreduju i time pružaju mogućnost za napredak samih aplikacija koje su u njima implementirane.

- [1] *Java*  
<https://docs.oracle.com/en/java/>
- [2] *Spring*  
<https://spring.io/>
- [3] *Spring Boot*  
<https://spring.io/projects/spring-boot>
- [4] *PostgreSQL*  
<https://en.wikipedia.org/wiki/PostgreSQL>
- [5] *React*  
<https://reactjs.org/>  
[https://en.wikipedia.org/wiki/React\\_\(web\\_framework\)/](https://en.wikipedia.org/wiki/React_(web_framework)/)
- [6] *React Native*  
<https://reactnative.dev/>
- [7] *JavaScript*  
[https://developer.mozilla.org/en-US/docs/Learn/Getting\\_started\\_with\\_the\\_web/JavaScript\\_basics](https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics)
- [8] *Bootstrap*  
<https://react-bootstrap.github.io/>
- [9] *Dependency Injection (DI)*  
[https://en.wikipedia.org/wiki/Dependency\\_injection](https://en.wikipedia.org/wiki/Dependency_injection)
- [10] *Inversion of Control (IoC)*  
<https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring>
- [11] *Spring Web MVC*  
<https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html>
- [12] *REST*  
<https://restfulapi.net/>  
[https://en.wikipedia.org/wiki/Representational\\_state\\_transfer](https://en.wikipedia.org/wiki/Representational_state_transfer)

[13] *HTTP*

[https://en.wikipedia.org/wiki/Hypertext\\_Transfer\\_Protocol](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol)

[14] *JPA (Java Persistence API)*

<https://spring.io/projects/spring-data-jpa>

[15] *Spring Security*

<https://www.baeldung.com/security-spring>



## 6. KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Редни број, <b>РБР:</b>	
Идентификациони број, <b>ИБР:</b>	
Тип документације, <b>ТД:</b>	монографска публикација
Тип записа, <b>ТЗ:</b>	текстуални штампани
Врста рада, <b>ВР:</b>	дипломски-бечелор рад
Аутор, <b>АУ:</b>	Луна Живковић
Ментор, <b>МН:</b>	ПрофДр Милан Видаковић
Наслов рада, <b>НР:</b>	Веб апликација за продају рекламног материјала
Језик публикације, <b>ЈП:</b>	српски
Језик извода, <b>ЈИ:</b>	српски / енглески
Земља публиковања, <b>ЗП:</b>	Србија
Уже географско подручје, <b>УГП:</b>	Војводина
Година, <b>ГО:</b>	2021
Издавач, <b>ИЗ:</b>	ауторски репринт
Место и адреса, <b>МА:</b>	Нови Сад, Факултет техничких наука,
Физички опис рада, <b>ФО:</b> (поглавља/страна/цитата/табела/слика/графика/прилога)	5 / 56 / 0 / 0 / 16 / 0 / 0
Научна област, <b>НО:</b>	Информатика
Научна дисциплина, <b>НД:</b>	Рачунарске науке
Предметна одредница/Кључне речи, <b>ПО:</b>	Java, ВЕБ, REST, React, React Native
<b>УДК</b>	

Чува се, <b>ЧУ:</b>		Библиотека Факултета техничких наука, Трг Доситеја Обрадовића 6, Нови Сад	
Важна напомена, <b>ВН:</b>			
Извод, <b>ИЗ:</b>		Задатак рада представља развој веб апликације за продају рекламног материјала. Серверски део апликације ће бити реализован у програмском језику Јава, уз помоћ Spring окружења, док ће клијентски део веб апликације бити развијан у React.js окружењу, а клијентски део мобилне апликације у React Native окружењу. Апликација ће служити за објављивање, преглед и поручивање рекламног метеријала.	
Датум прихватања теме, <b>ДП:</b>			
Датум одбране, <b>ДО:</b>			
Чланови комисије, <b>КО:</b>	Председник:		
	Члан:		Potpis mentora
	Члан, ментор:	др Милан Видаковић, ред. проф., ФТН Нови Сад	



## 7. KEY WORDS DOCUMENTATION

Accession number, <b>ANO</b> :	
Identification number, <b>INO</b> :	
Document type, <b>DT</b> :	monographic publication
Type of record, <b>TR</b> :	textual material
Contents code, <b>CC</b> :	BSc thesis
Author, <b>AU</b> :	Luna Živković
Mentor, <b>MN</b> :	Prof Dr Milan Vidaković
Title, <b>TI</b> :	WEB APPLICATION FOR SELLING ADVERTISING MATERIAL
Language of text, <b>LT</b> :	Serbian
Language of abstract, <b>LA</b> :	Serbian / English
Country of publication, <b>CP</b> :	Serbia
Locality of publication, <b>LP</b> :	Vojvodina
Publication year, <b>PY</b> :	2021
Publisher, <b>PB</b> :	author's reprint
Publication place, <b>PP</b> :	Novi Sad, Faculty of Technical Sciences, Trg Dositeja Obradovića 6
Physical description, <b>PD</b> : (chapters/pages/ref./tables/pictures/graphs/appendixes)	5 / 56 / 0 / 0 / 16 / 0 / 0
Scientific field, <b>SF</b> :	Electrical Engineering
Scientific discipline, <b>SD</b> :	Computer Science
Subject/Key words, <b>S/KW</b> :	Java, BEB, REST, React, React Native
<b>UC</b>	
Holding data, <b>HD</b> :	Library of the Faculty of Technical Sciences,
Note, <b>N</b> :	

Abstract, <b>AB</b> :		The thesis deals with the implementation of a web application for selling advertising material. The back-end part is implemented using Java programming language and Spring framework, while the front-end for web application is implemented using React.js framework, and mobile application using React Native. The application will provide overview, publishing and accessing to meaningful information about advertising material.	
Accepted by the Scientific Board on, <b>ASB</b> :			
Defended on, <b>DE</b> :			
Defended Board, <b>DB</b> :	President:		
	Member:		Menthor's sign
	Member, Mentor:	Milan Vidaković, PhD, full prof., FTN Novi Sad	

# Biografija

---

Luna Živković, rođena 14.02.1999. u Novom Sadu. Završila osnovnu školu “Đorđe Natošević” i gimnaziju “Jovan Jovanović Zmaj” u Novom Sadu.

Kontakt e-mail adresa:

[lunazvkovic@gmail.com](mailto:lunazvkovic@gmail.com)

