



UNIVERZITET CRNE GORE  
ELEKTROTEHNIČKI FAKULTET



Sario Al Mustafa

**Razvoj web platforme i mobilne aplikacije,  
sa prijedlogom algoritma za odabir optimalne  
saobraćajne rute**

MAGISTARSKI RAD

Podgorica, 2022. godine



UNIVERZITET CRNE GORE  
ELEKTROTEHNIČKI FAKULTET



Sario Al Mustafa

**Razvoj web platforme i mobilne aplikacije,  
sa prijedlogom algoritma za odabir optimalne  
saobraćajne rute**

MAGISTARSKI RAD

Podgorica, 2022. godine

## **PODACI I INFORMACIJE O MAGISTRANDU**

Ime i prezime: Sario Al Mustafa

Datum i mjesto rođenja: 02.04.1990. godine, Podgorica, Crna Gora

Naziv završenog osnovnog i specijalističkog studijskog programa i godina završetka: Studijski program Primijenjeno računarstvo, Elektrotehnički fakultet, Univerzitet Crne Gore, 240 ECTS kredita, 2016. godine.

## **INFORMACIJE O MAGISTARSKOM RADU**

Naziv postdiplomskih studija: Postdiplomske magistarske studije primijenjenog računarstva

Naslov rada: Razvoj web platforme i mobilne aplikacije, sa prijedlogom algoritma za odabir optimalne saobraćajne rute

Fakultet na kojem je rad odbranjen: Elektrotehnički fakultet

## **UDK, OCJENA I ODBRANA MAGISTARSKOG RADA**

Datum prijave magistarskog rada: 13.03.2019. godine

Datum sjednice Vijeća na kojoj je prihvaćena tema: 12.09.2019. godine

Komisija za ocjenu teme i podobnosti magistranda: Prof. dr Milutin Radonjić  
Prof. dr Nikola Žarić  
Prof. dr Vesna Popović-Bugarin

Mentor: Prof. dr Nikola Žarić

Komisija za ocjenu rada: Prof. dr Milutin Radonjić  
Prof. dr Nikola Žarić  
Prof. dr Vesna Popović-Bugarin

Komisija za odbranu rada: Prof. dr Milutin Radonjić  
Prof. dr Nikola Žarić  
Prof. dr Vesna Popović-Bugarin

Datum odbrane: 22.12.2022. godine

Ime i prezime autora: Sario Al Mustafa, Spec.App

## ETIČKA IZJAVA

U skladu sa članom 22 Zakona o akademskom integritetu i članom 24 Pravila studiranja na postdiplomskim studijama, pod krivičnom i materijalnom odgovornošću, izjavljujem da je magistarski rad pod naslovom

**"Razvoj web platforme i mobilne aplikacije, sa prijedlogom algoritma za odabir optimalne saobraćajne rute"**

moje originalno djelo.

Podnosilac izjave,  
  
Sario Al Mustafa, Spec.App

U Podgorici, dana 08.11.2022. godine

## **PREDGOVOR**

Svome mentoru, prof. dr Nikoli Žariću, zahvaljujem se na povjerenju koje mi je ukazao i prihvatio mentorstvo, kao i na stručnim sugestijama i savjetima, uloženom trudu, izdvojenom vremenu i posvećenosti tokom cjelokupne izrade ovoga magistarskog rada.

Posebnu zahvalnost dugujem svojim roditeljima, majci Verici, ocu Mohamadu i sestri Sari. Oni su bili i ostali moja bezrezervna podrška i sigurna luka svih ovih godina.

Sario Al Mustafa, Spec.App

# IZVOD RADA

Predmet istraživanja je razvoj veb platforme i mobilne aplikacije koje će koristiti odgovarajuće alate za: prikupljanje i dijeljenje korisničkih informacija o saobraćaju, odabir optimalne saobraćajne rute i/ili parking mjesta, upotrebu glasovnih komandi za interakciju sa korisnicima, kao i sistem nagrađivanja. Poseban osvrt u istraživanju biće na analizi algoritama za pronalaženje optimalne saobraćajne rute, kao i predlaganju algoritma za selekciju iste na osnovu informacija prikupljenih od strane korisnika, kako bi se smanjilo vrijeme vožnje, a samim tim i zagađivanje vazduha.

U predmetnom istraživanju koristit će se sledeće naučne metodologije: analiza podataka, kontrola log fajlova, simulacija pronalaženja optimalne saobraćajne ruta između jedne ili više lokacija, parking zone i sistema nagrađivanja, kao i analiza rezultata postignutih od strane anonimne grupe koja je koristila predloženo rješenje.

Korisnici, posebno turisti, lakše će se kretati u saobraćaju primjenom predloženog rješenja za optimalno pronalaženje saobraćajne rute. Očekivani benefiti za korisnika su pronalaženje optimalne saobraćajne rute sa najkraćim vremenskim intervalom i/ili najmanjim rastojanjem između dvije ili više lokacija, a samim time i smanjenje goriva. Takođe, krajnji rezultat je smanjenje emisije štetnih čestica/gasova koji nastaju pri korištenju motornih vozila.

Kroz korišćenje veb platforme i Android mobilne aplikacije, ovo istraživanje nastoji da pruži inovativno rješenje koje će korisnicima omogućiti da ostvare efikasan saobraćaj. Izraz „efikasnost saobraćaja“ se odnosi na količinu vremena potrebnog korisniku ovog rješenja da do svog odredišta stigne u što kraćem vremenskom intervalu, koristeći manje goriva i kao rezultat toga emitujući manje štetnih gasova i čestica. Ovaj pristup ima dobar uticaj i na životnu sredinu smanjenjem zagađenja vazduha.

**Ključne riječi:** algoritam, optimalna saobraćajna ruta, parking, glasovne komande, sistem nagrađivanja.

# ABSTRACT

The subject of this research is the development of a web platform and mobile application that will use appropriate tools for: collecting and sharing user traffic information, selecting the optimal traffic route and/or parking space, using voice commands to interact with users, and a reward system. Special attention in the research will be on the analysis of algorithms for finding the optimal traffic route, as well as proposing an algorithm for its selection based on information collected by users, in order to reduce driving time and thus air pollution.

The following scientific methodologies will be used in the subject research: data analysis, control of log files, simulation of finding the optimal traffic route between one or more locations, parking zones and reward systems, as well as analysis of the results achieved by an anonymous group that used the proposed solution.

Users, especially tourists, will find it easier to navigate in traffic by applying the proposed solution for optimal traffic route finding. The expected benefits for the user are finding the optimal traffic route with the shortest time interval and/or the shortest distance between two or more locations, and thus fuel reduction. Also, the end result is a reduction in the emission of harmful particles/gases that occur when using motor vehicles.

Through the use of a web platform and an Android mobile application, this research seeks to provide an innovative solution that will enable users to achieve traffic efficiency. The term "traffic efficiency" refers to the amount of time it takes the user of this solution to reach his destination in the shortest possible time, using less fuel and as a result emitting less harmful gases and particles. This approach also has a good impact on the environment by reducing air pollution.

**Keywords:** algorithm, optimal traffic route, parking, voice commands, reward system.

# Sadržaj

<b>1. UVOD .....</b>	<b>1</b>
<b>1.1. Motivacija .....</b>	<b>2</b>
<b>1.2. Predmet istraživanja.....</b>	<b>6</b>
<b>1.3. Metodologija istraživanja.....</b>	<b>8</b>
<b>1.4. Očekivani rezultati i doprinosi .....</b>	<b>9</b>
<b>2. RAZVOJNA OKRUŽENJA.....</b>	<b>10</b>
<b>3. RAZVOJ VEB PLATFORME.....</b>	<b>13</b>
<b>3.1. Struktura baze podataka.....</b>	<b>19</b>
<b>3.2. Serverski procesi .....</b>	<b>22</b>
3.2.1. Sistem identifikacije korisnika .....	23
3.2.2. Sistem nagrađivanja .....	26
3.2.3. Sistem informacija.....	30
<b>4. RAZVOJ MOBILNE APLIKACIJE.....</b>	<b>35</b>
<b>4.1. Profil korisnika.....</b>	<b>40</b>
<b>4.2. Glavna mapa.....</b>	<b>49</b>
<b>4.3. Planiranje saobraćajnih ruta .....</b>	<b>66</b>
<b>4.4. Parking zone .....</b>	<b>70</b>
<b>4.5. Glasovne komande .....</b>	<b>75</b>
<b>5. REZULTATI I DISKUSIJE.....</b>	<b>81</b>
<b>6. ZAKLJUČAK .....</b>	<b>84</b>
<b>7. LITERATURA.....</b>	<b>86</b>



# 1. Uvod

Tehnologija je suma znanja, tehnika, vještina, iskustava i metoda pomoću kojih ljudi mijenjaju, transformišu i koriste okruženje u cilju stvaranja alata, mašina, proizvoda i usluga koje zadovoljavaju naše potrebe i želje. Mi primjenjujemo tehnologije za ostvarivanje raznih zadataka u svom svakodnevnom životu, npr. koristimo tehnologije na poslu, za komunikaciju, učenje, transport, proizvodnju, osiguranje podataka, biznis i još mnogo toga. Tehnologiju možemo opisati kao proizvode i procese koji se koriste za pojednostavljivanje našeg svakodnevnog života. Ako se dobro primijeni ona može biti korisna za ljude i okolinu, ali isto tako može imati suprotan efekat ako se loše primijeni. Nema sumnje da je tehnološki napredak promijenio naš način razmišljanja, bivanja i življenja kao i okolno okruženje. Iskopali smo velike površine zemlje za traženje i vađenje metala i minerala koji nam omogućavaju proizvodnju mašina i alata, raskrčili smo velike površine šuma da bismo ih preradili u daske i papire, koristimo proizvode sagorijevanja koji u atmosferu emituju štetne materije itd. Naša svakodnevna aktivnost stvara mnogo otpada i samim time uzrokuje brojne probleme, među njima i zagađenje životne sredine.

Poznato je da su prevozna sredstva kao što su motorcikli, automobili, kombiji, autobusi, kamioni i ostala vozila koja koriste fosilna goriva jedan od glavnih zagađivača vazduha, što negativno utiče na životnu sredinu i zdravlje ljudi. Savremeni gradovi pored brojnih problema imaju problem gužvi u saobraćaju, što rezultira dužim vremenom korišćenja vozila koja proizvode štetne posljedice na životnu sredinu, odnosno povećavaju zagađivanje vazduha, pa je neophodno pronalaženje načina za postizanje efikasnijeg saobraćaja.

Ovo istraživanje ima za cilj da objedini višenamjenske platforme i korišćenjem naprednih tehnologija pruži inovativno rješenje koje će pored mogućnosti za postizanje efikasnijeg saobraćaja imati za posljedicu pozitivan uticaj na životnu sredinu, odnosno na smanjenje zagađivanja vazduha.

## 1.1. Motivacija

Životna sredina je sve što čini naše okruženje (vazduh, zemlja, voda itd.). Ona je kombinacija svega živoga i neživoga, odnosno biotičke i abiotske komponente, kao i njihovih međusobnih odnosa.

Zagađenje, ili kako se često naziva zagađivanje životne sredine (ekološko zagađivanje), je svaka promjena u prirodi životne sredine usljed uvođenja štetnih materija koje izazivaju negativne efekte. Postoji više različitih vrsta zagađenja, ali glavne vrste koje se klasifikuju po okruženju su: zagađenje vazduha, zemljišta i vode. Zagađivanje svih vrsta može imati negativne efekte na životnu sredinu i ekosistem, kao i na zdravlje i dobrobit ljudi. Zagađenje može da se odnosi na:

- štetne gasove, tečnost ili druge štetne materije koje se emituju u životnoj sredini,
- toksični gasovi i materijali koji čine vazduh i zemljište nečistim,
- ispuštanje hemikalija koje degradiraju ili mijenjaju biološko svojstvo voda, kao što su rijeke, jezera, podzemne vode itd.,
- zagađivači, kontaminanti ili opasne materije koje životnu sredinu čine nebezbjednom ili neprikladnom,
- nepodnošljivi i neželjeni nivo energije kao što su zračenja, buka itd.,
- rezultirajući efekat aktivnosti koje narušavaju biološki ekosistem ili predstavljaju opasnost za održivost životne sredine.

Zagađenje vazduha je miješanje prirodnih i vještačkih materija (čestice/gasovi) pri čemu mogu dostići štetne koncentracije koje imaju negativno dejstvo na životnu sredinu, kao i zdravlje ljudi. Razlikujemo dvije vrste zagađenja vazduha: spoljašnje i unutrašnje zagađenje (engl. outdoor and indoor pollution).

Spoljašnje zagađenje (Slika 1.1.) vazduha uključuje izloženost koja se odvija na otvorenom, kao što su:

- štetne čestice koje nastaju sagorijevanjem fosilnih goriva (tj. uglja i nafte koji se koriste u proizvodnji, transportu itd.),
- štetni gasovi (azotni oksidi, sumpor dioksid, ugljen monoksidske pare itd.),

- ozon u prizemnim slojevima atmosfere (reaktivni oblik kiseonika i primarna komponenta gradskog smoga),
- duvanski dim.



Slika 1.1. Spoljašnja zagađivanja

Unutrašnje zagađivanje (Slika 1.2.) vazduha, odnosno zagađivanje u zatvorenom prostoru uključuje izloženost štetnim česticama kao što su:

- proizvodi i hemikalije za domaćinstva,
- gasovi (radoni, ugljen monoksid itd.),
- građevinski materijali (olovo, metanal itd.),
- buđ,
- duvanski dim.



Slika 1.2. Unutrašnja zagađivanja vazduha

Takođe, postoji mogućnost da spoljašnja zagađenja prodru u zatvorene prostorije kroz otvorene prozore, vrata, ventilaciju itd. i na taj način da postanu unutrašnje zagađenje.

Problem zagađenja vazduha je globalni problem, i kao takav on je prisutan i u našem okruženju.

Kombinovani efekti spoljašnjeg i unutrašnjeg zagađivanja povezani su sa 7 (sedam) miliona prijevremenih smrti godišnje širom svijeta na osnovu podataka svjetske zdravstvene organizacije (engl. „World Health Organization“)<sup>1</sup>. Ovaj podatak ukazuje na potrebu pronalaženja rješenja koja bi umanjila emitovanje štetnih čestica/gasova u vazduhu.

Pronalaženje optimalne saobraćajne rute sa najkraćim vremenskim intervalom i/ili najkraćim rastojanjem između više lokacija je čest problem ljudima koji nijesu upoznati sa lokacijom. Kao rezultat pronalaženje optimalne saobraćajne rute može biti izazov, posebno za strance (npr. turiste) i to predstavlja prvi problem. Drugi problem je mogućnost da je saobraćajna ruta kojom vozač motornog vozila planira da prođe blokirana i kao rezultat dolazi do neprijatne situacije. Vozači koji se zadese u ovakvoj ili sličnoj situaciji nemaju drugog izbora osim da čekaju ili da se usmjere drugom saobraćajnom rutom što rezultira dodatnom potrošnjom goriva. Ova dva problema mogu biti finansijski zahtjevna, dugotrajna i frustrirajuća, a utiču i na povećanje zagađenosti vazduha.

<sup>1</sup> [https://www.who.int/health-topics/air-pollution#tab=tab\\_2](https://www.who.int/health-topics/air-pollution#tab=tab_2)

Informacione i komunikacione tehnologije (engl. Information and Communication Technologies – ICT) mogu doprinijeti rješavanju navedenih problema. ICT je široki pojam i njegov koncept se svakodnevno razvija. On obuhvata sve proizvode koji skladište, preuzimaju, manipulišu, prenose ili primaju podatke elektronskim putem u digitalnoj formi, odnosno putem računara, pametnih telefona i televizora, elektronskom poštom i slično. ICT sadrži šest (6) komponenti i to (Slika 1.3.):

- podaci (engl. data) – izvorni podaci (engl. raw data),
- hardver (engl. hardware) – fizičke mašine (računari, pametni telefoni, serveri itd.),
- softver (engl. software) – računarski program koji je razvijen za definisane operacije,
- informacije (engl. information) – obrađeni podatak ili spuk podataka koji se dalje koristi/e,
- procedure (engl. procedures) – niz definisanih aktivnosti, odnosno procesa koji se obavljaju u određenom redosljedu kako bi sistem funkcionisao uredno,
- ljudi (engl. people) – podaci se unose od strane ljudi, preko određenih uređaja, kao na primjer tastatura ili prikupljanjem podataka o kretanju ili drugim aktivnostima.



Slika 1.3. Komponente informacionih i komunikacionih tehnologija

## 1.2. Predmet istraživanja

Predmet proučavanja ovog rada je razvoj veb platforme i mobilne aplikacije koje će koristiti odgovarajuće alate za: prikupljanje i dijeljenje korisničkih informacija o saobraćaju, odabir optimalne saobraćajne rute i/ili parking mjesta, upotrebu glasovnih komandi za interakciju sa korisnicima, kao i sistem nagrađivanja. Poseban osvrt u istraživanju dat je na analizi algoritama za pronalaženje optimalne saobraćajne rute, kao i predlaganju algoritma za selekciju iste na osnovu informacija prikupljenih od strane korisnika, kako bi se smanjilo vrijeme vožnje, a samim tim i zagađivanje vazduha.

Obuhvaćene tehnologije i aplikacije zajedno sa informacijama, procesima i korisnicima veb platforme i mobilne aplikacije su predstavljene kroz razvojni proces veb platforme i Android mobilne aplikacije, kao i analizom postignutih rezultata. Takođe je predstavljena analiza okruženja za razvoj veb platforme i mobilnih aplikacija, njihove prednosti i mane, kao i obrazloženja odabranih razvojnih okruženja.

Primjena predloženog rješenja koje je predstavljeno kao veb platforma i Android mobilna aplikacija može pružiti korisnicima za pronalaženje optimalne saobraćajne rute između dvije ili više lokacija. Kao rezultat korisnicima ovog rješenja se pruža mogućnost da manje koriste motorna vozila u saobraćaju što rezultira manjim emitovanjem štetnih čestica/gasova. Takođe, prikupljanjem podataka i njihovom obradom doći će se do praktičnih rezultata koje donosioci odluka mogu da implementiraju u neka dalja rješenja za efikasniji saobraćaj.

Dosadašnja istraživanja koja se bave sličnim temama [1 – 7] ukazuju na potrebu korisnika za pronalaženje optimalne saobraćajne rute u cilju smanjenja potrebnog vremena kao i troškova vožnje, a za posljedicu imaju smanjenje zagađenosti vazduha. Takođe, pojedina istraživanja ukazuju na urgentnost u oblasti zaštite životne sredine, kao što je smanjenje zagađivanja vazduha [11].

U istraživanju [1] predstavljen je scenario gdje se osoba našla u nezgodnoj situaciji, odnosno nekoj vrsti nesreće i pomoću slanja GPS koordinata i informacija o nesreći sistem će predložiti odgovarajuće akcije koje treba preuzeti, kao i najkraću putanju do najbliže bolnice. U radu [2] je predstavljen sistem koji pruža mapiranje gustoće saobraćaja u realnom vremenu

koristeći ugrađeni modul u vozilu i Android aplikaciju koja prikazuje sva vozila između dvije lokacije u bojama za vozila u pokretu, kao i stacionarna.

U istraživanju [3] je predstavljen modul koji se može implementirati u svrhu predstavljanja dodatnih informacija koje se mogu naći na saobraćajnoj ruti do željene destinacije. U istraživanju [4] je predstavljena mobilna aplikacija koja pruža korisnicima informacije u realnom vremenu posebno turistima koji nijesu upoznati sa javnim prevozom. Takođe se posvetila pažnja na istraživanjima sistema za pronalaženje parking mjesta ili kako ga nazivaju „Pametno Parkiranje“ [7, 8]. Korišćenjem bežičnih senzora koji se postavljaju na definisane lokacije detektuju se vozila i podaci se šalju serveru koji ih obrađuje. U istraživanja [7, 8] korišteni su posebni algoritmi za pronalaženje parking mjesta, kao i pronalaženje optimalne saobraćajne rute do parking mjesta, odnosno do parking zona.

Istraživanja koja se bave glasovnim komandama imaju za cilj da olakšaju korišćenje mobilnih uređaja. Tako se u istraživanju [9] omogućava korisnicima mobilnih uređaja da glasovnim komandama obavljaju uobičajene zadatke kao što su postavljanje alarma, upućivanje poziva, pa čak i pokretanje aplikacija. Tako se mogu kreirati posebni algoritmi za postizanje kompleksnih zadataka. Istraživanja u oblasti sistema nagrađivanja imaju za cilj da putem virtuelnih nagrada podstaknu i motivišu korisnike da nastave sa korišćenjem datog proizvoda bilo to veb platforme, mobilne aplikacije ili video igrice. U istraživanju [10] se ispituje da li korisnici preferiraju da biraju vid nagrade ili jednostavno da prime nagradu putem slučajnog dodjeljivanja. U svakom slučaju rezultati ukazuju na bolje performanse korisnika kada ih očekuje neki vid nagrade na kraju zadatka.

U istraživanju [11] je predstavljeno rješenje koje koristi posebne senzore koji vrše monitoring kvaliteta vazduha u gradu i javnih prevoza kao što su autobusi u cilju prikupljanja podataka o stanju zagađenosti vazduha. Koriste se dvije vrste senzora, stacionarni i pokretni. Stacionarni su postavljeni na različitim lokacijama u gradu, a pokretni u vozilima javnog prevoza. Ovaj vid mreže senzora efikasnije prikuplja podatke na osnovu kojih se vrše analize.

Nedostatak informacija o stanju u saobraćaju veoma često dovodi do stvaranja saobraćajnih gužvi što rezultira dužom upotrebom saobraćajnih vozila i negativno utiče na zagađenje vazduha, a samim tim i zdravlje ljudi. Upotrebom algoritma koji će biti predložen za odabir optimalne saobraćajne rute, a sve na bazi informacija prikupljenih putem aplikacije i

relevantnih informacija koje korisnici putem iste učine dostupnim treba da doprinese rješavanju ovog problema. Naime, korisnici veb platforme i aplikacije će imati mogućnost da preko odgovarajućih alata odaberu optimalnu ili alternativne saobraćajne rute. Kao rezultat korisnici brže dolaze do željene destinacije što smanjuje troškove vožnje i zagađivanje vazduha tako što korisnici manje koriste saobraćajna vozila.

### **1.3. Metodologija istraživanja**

Naučne metode koje će biti primijenjene u ovom istraživanju su: metoda analize prikupljenih podataka, kontrolisanja log fajlova, simulacije pronalaženja optimalnih saobraćajnih ruta, parking zona i sistema nagrađivanja, kao i metoda korišćenja rješenja od strane anonimne test grupe.

Metodom simulacije pronalaženja optimalnih saobraćajnih ruta i parking zona biće pokazano kako će dijeljenje i prikupljanje korisničkih informacija moći da doprinese selekciji optimalne saobraćajne rute, kao i informacija o stanju parking zona. Simulacijom sistema nagrađivanja biće pokazano kako se korisnici mogu podstaknuti i motivisati za dalje dobrovoljno dijeljenje informacija o saobraćaju.

Metodom korišćenja ovog rješenja od strane anonimne testne grupe će se ustanoviti koliko je veb platforma i aplikacija primjenljiva, kao i prikupljanje informacija. Informacije će se koristiti za svrhu analiza i procjenu smjera razvoja platforme i aplikacije.

Metodama prikupljanja i analize podataka treba da se dobiju informacije o korišćenju platforme i povratne informacije od korisnika, Prikupljanje podataka će se vršiti tako što korisnici dobrovoljno postavljaju informacije o stanju u saobraćaju. Ove metode imaju za cilj statističke analize, kao i davanje smjernica za unapređenje platforme i aplikacije.

Posljednja metoda je metoda kontrolisanja log fajlova koji se nalaze na serveru. Log fajlovi služe za prikupljanje informacija o svim eventualnim problemima koji mogu da nastanu u toku rada. Informacije iz log fajlova se zatim koriste kao smjernice za pronalaženje rješenja eventualnih problema.



## 1.4. Očekivani rezultati i doprinosi

Očekivani rezultat ovog istraživanja je kreiranje web platforme i mobilne aplikacije koje za cilj imaju da se postigne efikasniji saobraćaj, što podrazumijeva smanjenje potrebnog vremene vožnje, a kao rezultat pruža mogućnost da se saobraćajna vozila manje koriste i samim time smanji zagađivanje vazduha.

Kako bi se podstakla primjena nova veb platforma i mobilna aplikacija će biti besplatne i jednostavne za korišćenje. Korisnici aplikacije će imati mogućnost da dijele informacije o saobraćaju tako što postavljaju definisane ikonice koje predstavljaju određeno značenje na mapi, a sve putem glasovnih komandi kako ne bi koristili mobilni uređaj tokom vožnje.

Za određivanje optimalne saobraćajne rute potrebno je da korisnik unese željenu destinaciju ili više lokacija kako bi se putem platforme pronašla optimalna saobraćajna ruta ili optimalni plan vožnje uzimajući u obzir sve informacije koje korisnici dijele.

Doprinosi ovog istraživanja su rješenja u vidu platformi za pronalaženje optimalne saobraćajne rute i/ili parking mjesta što može podstaknuti korisnike da manje koriste saobraćajna vozila i rezultira u smanjenju zagađenosti vazduha i podizanju svijesti ekološkog i zdravog stila života, posebno među mladima uzimajući u obzir popularnost korišćenja mobilnih uređaja i aplikacija.

Kao doprinos treba naglasiti i definisanje algoritma za selekciju optimalne rute na bazi informacija koje korisnici dijele u realnom vremenu.

## 2. Razvojna okruženja

Razvojna okruženja koja se koriste u realizaciji veb platforme i Android mobilne aplikacije su odabrana u cilju postizanja brzog i efikasnog koda, kao i stabilnog protoka podataka između korisnika i servera. Veb platforma i Android mobilna aplikacija realizovane su korišćenjem različitih razvojnih okruženja, dok koriste istu bazu podataka. Za razvoj baze podataka imamo dvije opcije SQL (engl. Structured Query Language) [12, 13] ili NoSQL (engl. non Structured Query Language).

SQL je jezik za pisanje upita koji se koriste u komunikaciji sa bazom podataka (npr. za odabir, dodavanje, brisanje i ažuriranje podataka). Komunikacija se odvija prema relacionoj bazi podataka koju sačinjava jedna ili više tabela. Tabele su skup podataka koji su definisani na osnovu šeme. Šema je strogo definisani skup polja koji predstavljaju odgovarajuće podatke za razne potrebe. Svaki novi zapis u tabeli je skup vrijednosti koji ispunjavaju definisanu šemu, odnosno polja. Ova polja moraju biti popunjena podacima pa makar da su prazna ili *NULL* (specijalna naznaka da je vrijednost u polju prazna). Odnosi između tabela, odnosno njihove relacije možemo podijeliti u tri vrste, i to:

- jedan-na-jedan (engl. one-to-one) – ova relacija se primjenjuje kada samo jedna stavka, odnosno jedno polje u tabeli A se može odnositi na samo jednu stavku, odnosno jedno polje u tabeli B,
- jedan-na-više (engl. one-to-many) – ova relacija se primjenjuje kada se jedna stavka, odnosno jedno polje u tabeli A može odnositi na više stavki, odnosno više polja u tabeli B,
- više-na-više (many-to-many) – ova relacija se primjenjuje kada se više stavki, odnosno više polja u tabeli A može koristiti na više stavki, odnosno više polja u tabeli B.

NoSQL baza podataka ne koristi tabele i relacije, takođe ne zavisi od definisane šeme. Ova baza podataka koristi kolekcije (engl. Collections) i u njima su smješteni dokumenti (engl. Documents) koji sadrže podatke. Ovi dokumenti ne moraju da koriste istu šemu kao što je slučaj kod SQL baze podataka. Dakle, NoSQL ne koristi relacije, već se svi podaci smještaju na jednom mjestu, kao na primjer u jednoj kolekciji.

Ovaj pristup je u jednoj mjeri fleksibilan jer možemo imati više različitih dokumenata sa različitim podacima u jednoj kolekciji, dok sa druge strane ne zavisi od definisanog formata, odnosno šeme što može da predstavlja problem kada se filtriraju podaci. Takođe, ne koristi relacije što rezultira brzim i efikasnim upitima prema bazi podataka, ali nedostatak u tome je što se neki podaci mogu ponavljati više puta u dokumentima što bi predstavljalo problem u slučaju da se isti podaci ažuriraju i time bi morali na svakom mjestu gdje se ponavljaju da se ažuriraju.

U ovom radu smo odabrali SQL pristup jer koristi strogo definisanu šemu što predstavlja stabilnost i preciznost pri unosu i filtriranju podataka. Takođe, koristi relacije što omogućava lakše ažuriranje podataka, kao i tabele sa definisanim poljima što nam pruža jasnu sliku za šta se koristi tabela.

Za razvoj veb platforme smo odabrali nativno kodiranje (engl. Hand Coding) u odnosu na korišćenje veb okvira (engl. Framework). Nativno kodiranje uključuje korišćenje više tehnologija za razne potrebe, u našem slučaju odabrali smo sljedeće tehnologije:

- HTML – je jezik za označavanje hiper teksta (engl. Hyper Text Markup Language), on se koristi za dizajniranje strukture veb stranica [14],
- CSS – (engl. Cascading Style Sheets) je jezik koji se koristi za definisanje stila veb stranice [14],
- JavaScript – je objektno skriptni jezik koji ima za cilj da poboljša dinamiku veb stranice kao i korisničko iskustvo (engl. User Experience) [15],
- PHP – je skriptni jezik koji se izvodi na strani servera i služi za izradu dinamičnih veb stranica, obradu podataka u bazi podataka itd. [16].

Ovim pristupom postizemo brzo i efikasno korišćenje veb platforme, kao i komunikaciju sa bazom podataka. Takođe, nadogradnja veb platforme sa novim funkcionalnostima je jednostavna i fleksibilna u ovom slučaju.

Veb okvire možemo posmatrati kao kolekciju već spremnih funkcionalnosti koje se najviše koriste u razvoju veb aplikacija. Ovo je prednost ako planiramo da razvijamo veb aplikaciju koja bi koristila već postojeće funkcionalnosti u veb okviru i time bi smanjili uloženo vrijeme za razvoj iste. U slučaju da nemamo funkcionalnosti koje želimo morali bi da razvijamo nove funkcionalnosti u okruženju veb okvira koji koristi svoju definisanu strukturu. Razvijanje

novih funkcionalnosti u veb okvirima može da predstavlja problem, jer je struktura strogo definisana i kao rezultat neke nove funkcionalnosti koje bi se jednostavno razvile u ručnom kodiranju mogle bi se iskomplikovati u veb okviru.

Razvojno okruženje za Android mobilne aplikacije je „Android Studio“, zvanično preporučeno razvojno okruženje za razvoj Android aplikacija. Android Studio koristi dva programska jezika i to Javu (engl. Java) [17, 18] ili Kotlin (engl. Kotlin). Ova dva programska jezika su takozvani maternji jezici (engl. Native Language) za razvoj Android mobilnih aplikacija. Ovo je velika prednost u odnosu na sve ostale opcije za razvoj Android mobilnih aplikacija, jer obezbjeđuje najbrže i najefikasnije korišćenje Android aplikacija. Iz tog razloga je odabrano ovo razvojno okruženje.

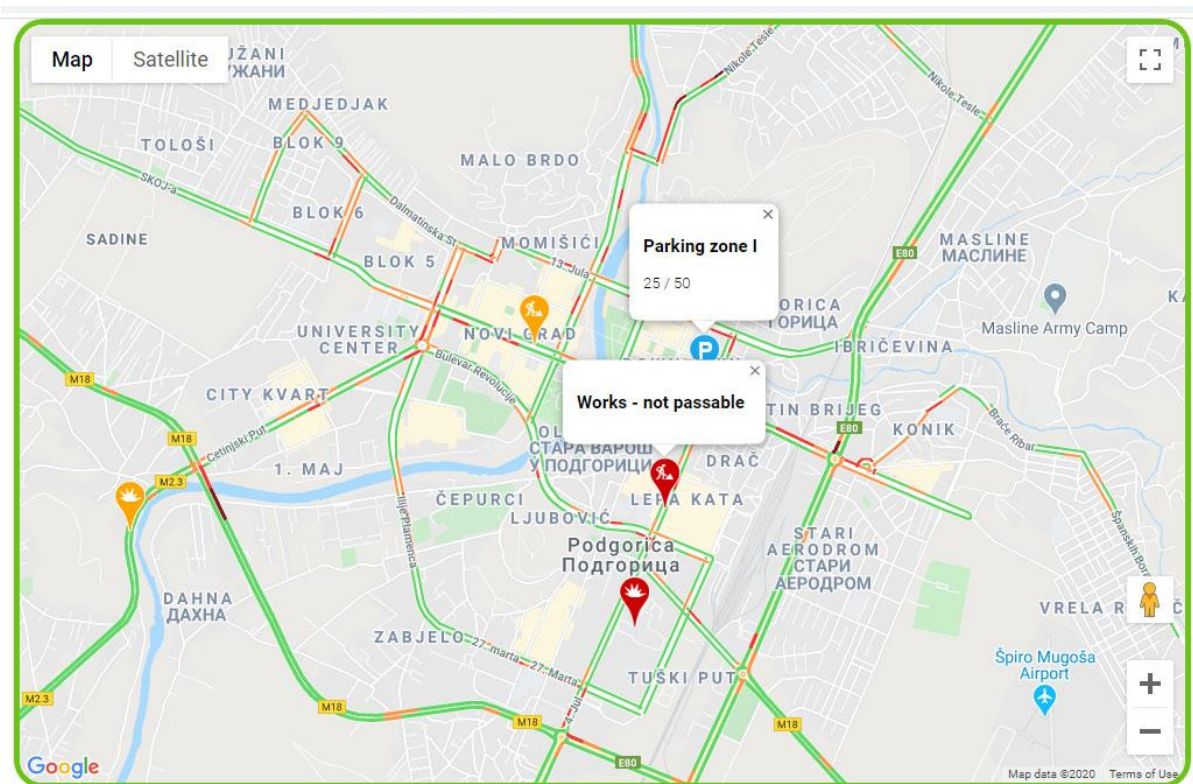
Drugi pristup su hibridne mobilne aplikacije (engl. Hybrid Mobile Applications). Hibridne mobilne aplikacije se razvijaju u posebnim razvojnim okruženjima koja obezbjeđuju razvoj mobilnih aplikacija za sve platforme, odnosno za Android i iOS. Prednost u ovome je što se samo jednom kodira mobilna aplikacija, a možda samo neke funkcionalnosti dva ili više puta zavisno od operativnih sistema mobilnih uređaja, u svakom slučaju mnogo manje se kodira u odnosu na korišćenje nativnih jezika kao što su Java ili Kotlin koji se isključivo koriste za razvoj Android mobilnih aplikacija, dok se za iOS koriste Objective-C i Swift. Nedostatak hibridnih mobilnih aplikacija je u tome što zavise od posebnih dodataka (engl. Plugins) koji komuniciraju sa ugrađenim funkcionalnostima uređaja što rezultira sporijom komunikacijom. Takođe, neki dodaci mogu biti zastarjeli ili nepouzdana što dodatno predstavlja problem pri razvoju mobilnih aplikacija. U nekim slučajevima je moguće i da ne postoje gotova rješenja koja omogućavaju pristup određenom dijelu funkcionalnosti mobilnog uređaja što dovodi do toga da se mora kreirati taj dodatak kao novo rješenje.

Pregled svih navedenih razvojnih okruženja je rezultiralo da se u ovom radu, odnosno u razvoju veb platforme i Android mobilne aplikacije sa predlogom algoritma za odabir optimalne saobraćajne rute isključivo odaberu razvojna okruženja.

### 3. Razvoj veb platforme

Razvoj veb platforme ima za cilj da pruži korisnicima brz i efikasan pristup informacijama o trenutnom stanju u saobraćaju u realnom vremenu, odnosno da predstavi gustinu saobraćaja koja je prikazana linijama u boji (Slika 3.1.). Takođe, platforma omogućava prikaz lokacije saobraćajnih nezgoda, radova na putu i policijskih patrola koje su predstavljene ikonicama ili markerima, odnosno grafički prikazi nečega, osobe ili stvari što pomaže korisnicima u navigaciji i/ili razumijevanju. Linije u boji koje predstavljaju gustinu su podijeljene u tri kategorije, tj. u tri boje, i to:








- zelena linija – gustina saobraćaja je na niskom nivou,
- narandžasta linija - gustina saobraćaja je na srednjem nivou,
- crvena linija - gustina saobraćaja je na visokom nivou.



Slika 3.1. Mapa Podgorice sa ikonicama i gustinom saobraćaja

Ikonice su predstavljene u dvije kategorije sa po tri podkategorije. Kategorije su podijeljene u dvije boje i to crvena i narandžasta. Sve podkategorije koje spadaju u kategoriju crvene boje ukazuju da ruta na kojoj se nalaze nije prohodna, dok sve podkategorije koje spadaju u kategoriju narandžaste boje su prohodne, ali sa određenim poteškoćama. Podkategorije su podijeljene u tri ikonice koje predstavljaju nezgodu, radove i policijske patrole. Takođe, ikonica parking je odvojena od gore navedene strukture kategorija. Parking ikonica je predstavljena plavom bojom i ona služi da predstavi korisnicima lokacije parking zona, kao i trenutno stanje parking mjesta, odnosno koliko ima slobodnih i koji je maksimalan broj parking mjesta. Broj ikonica se može proširiti po potrebi, tokom budućeg razvoja platforme. U tabeli (Tabela 3.1.) su prikazane sve ikonice koje se koriste na veb platformi i Android mobilnoj aplikaciji, kao i slika koja predstavlja gore navedene informacije (Slika 3.1.).

Tabela 3.1. Sve ikonice koje se koriste za prikaz informacija

	Crvena ikonica nezgode, ruta nije prohodna		Narandžasta ikonica nezgode, ruta je prohodna
	Crvena ikonica radova, ruta nije prohodna		Narandžasta ikonica radova, ruta je prohodna
	Crvena ikonica policije, ruta nije prohodna		Narandžasta ikonica policije, ruta je prohodna
		Plava ikonica za parking zone	

Ovaj dizajn (Slika 3.1.) je osmišljen tako da bude korisnicima jednostavan za razumijevanje, te su stoga informacije predstavljene u bojama i ikonicama na mapi, a postoje i opisi navedenih grafičkih oznaka. Kako bi postigli gore navedeni dizajn prvo je potrebno preuzeti sve neophodne informacije iz baze podataka, a to su podaci o postavljenim markerima koje nazivamo „bad\_markers“ i parking markerima koje nazivamo „parking\_markers“. Ovo radimo sa definisanim PHP-MySQL upitima koji su prikazani na sljedećoj slici (Slika 3.2.):

```
$resultBadMarkers = $db ->
query("SELECT * FROM bad_markers WHERE active = 1");
```

```

$resultBadMarkerInfoWindows = $db ->
query("SELECT * FROM bad_markers WHERE active = 1");
$resultParkingMarkers = $db -> query("SELECT * FROM parking_markers");
$resultParkingMarkerInfoWindows = $db ->
query("SELECT * FROM parking_markers");

```

Slika 3.2. PHP-MySQL upit koji je predviđen za preuzimanje podataka (podaci o postavljenim markerima i parking markerima) za veb platformu

Nakon preuzimanja podataka prelazimo na funkciju koja je u JavaScript-u i ima za cilj da preko navedenih upita (Slika 3.2.) sortira podatke iz baze podataka. To radimo tako što kombinujemo kodove JavaScript i PHP koje možemo podijeliti u tri sekcije.

Prva sekcija je za provjeru upita i sortiranje podataka postavljenih markera i parking markera. Sa upitima koji se nazivaju „*\$resultBadMarkers*” i „*\$resultBadMarkerInfoWindows*“ provjeravamo da li sadrži podatke, nakon čega ih preuzimamo iz tabele „bad\_markers“. Podatke koje preuzimamo su geografska širina i dužina, kao i kategorija markera. Potom ove podatke sortiramo u JSON formatu kako bi se mogli predstaviti na mapi veb platforme. Prvi upit koristimo za sortiranje podataka koje na kraju koristimo za postavljanje markera na mapi. Drugi upit koristimo za sortiranje podataka koji prolaze niz kategorijskih provjera, kako bi podatke pravilno sortirali i koje na kraju koristimo upit za postavljanje dodatnih informacija (u ovom slučaju naziv markera – „*\$title*“) o odgovarajućem markeru u slučaju da korisnici kliknu na jedan od markera. Ovaj proces je predstavljen na sljedećoj slici (Slika 3.3.):

```

var bad_markers = [
  <?php if($resultBadMarkers->num_rows > 0){
    while($row = $resultBadMarkers->fetch_assoc()){
      echo '['.$row['id'].', '.$row['latitude'].', '.$row['longitude'].
', '.$row['color'].', '.$row['type'].'],';
    }
  }
  ?>
];

var infoWindowContent_badMarkers = [
  <?php
  if($resultBadMarkerInfoWindows->num_rows > 0){

```

```

while($row = $resultBadMarkerInfoWindows->fetch_assoc()){
    if ($row['color'] == 1) {
        if ($row['type'] == 1) {
            $title = "Accident - not passable";
        } else if ($row['type'] == 2) {
            $title = "Works - not passable";
        } else if ($row['type'] == 3) {
            $title = "Police - not passable";
        }
    } else if ($row['color'] == 2) {
        if ($row['type'] == 1) {
            $title = "Accident - passable";
        } else if ($row['type'] == 2) {
            $title = "Works - passable";
        } else if ($row['type'] == 3) {
            $title = "Police - passable";
        }
    }
}

?>
['<div class="info_content">' +
 '<h3><?php echo $title; ?></h3></div>'],
<?php
}
}
?>
];

```

Slika 3.3. JavaScript kod za sortiranje podataka postavljenih markera za veb platformu

Nakon sortiranja podataka o postavljenim markerima potrebno je iste postaviti na mapi veb platforme. Ovaj proces postizemo tako što definišemo promjenljivu za prikaz dodatnih informacija, odnosno naziv markera (ili sadržaj). Zatim prolazimo kroz „for“ petlju koja sadrži niz kategorijskih provjera kako bi markerima pravilno dodijelili ikonice. Takođe, dodajemo sadržaj markera, tj. naziv kako bi se ta informacija prikazala kada korisnik klikne na jedan od markera. Na kraju kreirani marker dodajemo na mapi veb platforme i tako prolazimo kroz ovu „for“ petlju sve dok se svi preuzeti markeri ne postave na mapi (Slika 3.4.).

```

var infoWindow_badMarkers = new google.maps.InfoWindow(), bad_marker, i;
for( i = 0; i < bad_markers.length; i++ ) {

```



```

if (bad_markers[i][3] == "1") {
    if (bad_markers[i][4] == "1") {
        var icon = 'img/marker_red_accident.PNG';
    } else if (bad_markers[i][4] == "2") {
        var icon = 'img/marker_red_works.PNG';
    } else if (bad_markers[i][4] == "3") {
        var icon = 'img/marker_red_police.PNG';
    }
} else if (bad_markers[i][3] == "2") {
    if (bad_markers[i][4] == "1") {
        var icon = 'img/marker_orange_accident.PNG';
    } else if (bad_markers[i][4] == "2") {
        var icon = 'img/marker_orange_works.PNG';
    } else if (bad_markers[i][4] == "3") {
        var icon = 'img/marker_orange_police.PNG';
    }
}
var position = new google.maps.LatLng(bad_markers[i][1],
bad_markers[i][2]);
bounds.extend(position);
bad_marker = new google.maps.Marker({
    position: position,
    map: map,
    title: bad_markers[i][0],
    icon: icon
});

google.maps.event.addListener(bad_marker, 'click',
(function(bad_marker, i) {
    return function() {
        infoWindow_badMarkers.setContent(infoWindowContent_badMarkers
[i][0]);
        infoWindow_badMarkers.open(map, bad_marker);
    }
})(bad_marker, i));
map.fitBounds(bounds);

```

Slika 3.4. JavaScript kod za postavljanje podataka postavljenih markera za veb platformu

Na sličan način provjeravamo i postavljamo parking markere preko upita “\$resultParkingMarkers“ i “\$resultParkingMarkerInfoWindows„ (Slika 3.2.) koji preuzimaju podatke iz tabele „parking\_markers“. Razlika u ovom slučaju je što parking markeri pripadaju

jedinstvenoj kategoriji koja je predviđena samo za ovaj tip informacija i kao rezultat nema kategorijskih provjera. Takođe, ima svoju definisanu ikonicu, kao i dodatne informacije u slučaju kada korisnik klikne na jedan od parking markera. Dodatne informacije su broj slobodnih i maksimalan broj parking mjesta, kao i naziv parking lokacije (markera).

Druga sekcija sadrži parametre za podešavanje uveličanja/umanjena prikaza (engl. zoom level) i dodavanje informacija o gustini saobraćaja. Informacije o gustini saobraćaja se preuzimaju od strane Google Maps API-a koji predstavlja ove informacije u bojama. Kod ove sekcije je predstavljen na sljedećoj slici (Slika 3.5.):

```
var boundsListener = google.maps.event.addListener((map), 'bounds_changed', function(event) {
    this.setZoom(14);
    google.maps.event.removeListener(boundsListener);
});

var trafficLayer = new google.maps.TrafficLayer();
trafficLayer.setMap(map);
```

Slika 3.5. JavaScript kod za podešavanje zum nivoa i dodavanje informacija o gustini saobraćaja

Treća i posljednja sekcija sadrži kod kojim se pokreće funkcija pod nazivom „*initMap*“ koja obuhvata kod prve i druge sekcije (Slika 3.6.).

```
function initMap() {

    // Kod prve i druge sekcije
}
google.maps.event.addDomListener(window, 'load', initMap);
```

Slika 3.6. JavaScript kod kojim se pokreće funkcija „*initMap*“

### 3.1. Struktura baze podataka

Baza podataka za veb platformu i Android mobilnu aplikaciju je rađena u MySQL-u i implementirana na serveru kako bi bila dostupna svim korisnicima ove veb platforme i aplikacije koji su povezani na Internet. Baza se sastoji od četiri tabele, i to:

- tabela za korisnike – „users“,
- tabela za markere koji su postavljeni od strane korisnika Android mobilne aplikacije – „bad\_markers“,
- tabela za parking markere – „parking\_markers“,
- tabela za sistem nagrađivanja – „rewards“.

Dizajn gore navedenih tabela, odnosno baze podataka, se može vidjeti na sljedećoj slici (Slika 3.7.):

users	bad_markers	parking_markers	rewards
id : int(11)	id : int(11)	id : int(11)	id : int(11)
username : varchar(100)	user_id : int(11)	title : varchar(255)	user_id : int(11)
email : varchar(150)	color : int(11)	latitude : double	user_bad_marker_id : int(11)
password : varchar(255)	type : int(11)	longitude : double	bad_marker_id : int(11)
points : int(11)	latitude : double	radius : double	answer : int(11)
profile_image : int(11)	longitude : double	max_places : int(11)	
date_created : datetime	date : date	free_places : int(11)	
verified : int(1)	time : time		
	active : int(11)		

Slika 3.7. Dizajn baze podataka

Sve tabele sadrže svoju identifikacionu kolonu koju nazivamo „id“ i one su primarni ključevi. Primarni ključ (koji se takođe naziva jedinstveni ključ) je polje koje je dodijeljeno tabeli kao jedinstveni identifikator. Tip podataka ovog polja je cijeli broj (engl. Integer, INT). Vrijednost ovog polja je jedinstvena za svaki zapis i ne može da se ponovi, odnosno dva ili više zapisa ne mogu imati istu vrijednost.

Tipovi polja koja se koriste u ovoj bazi podataka su:

- INT – cijeli broj,
- VARCHAR – skup karaktera,
- DATETIME – datum i vrijeme,

- DOUBLE – decimalni brojevi,
- DATE – datum,
- TIME – vrijeme.

Tabela korisnika pod nazivom „users“ sadrži informacije o korisniku koje se koriste za podešavanje naloga u Android mobilnoj aplikaciji. Nalog je ličnog karaktera i samo ga korisnik može pregledati, dok ostali korisnici nemaju pristup tome. Na ovaj način korisnici koji postavljaju informacije o saobraćaju, odnosno postavljaju markere su anonimni i nijedan drugi korisnik ne može imati pristup informaciji ko ih je postavio. Ova tabela sadrži sljedeće informacije:

- username – korisnički nalog,
- email – elektronska pošta (engl. email) korisnika i koristi se za aktivaciju korisničkog naloga i komunikaciju sa korisnikom,
- password – šifra korisnika koja se koristi pri prijavljivanju na Android mobilnoj aplikaciji,
- points – broj poena koje korisnik prikuplja tokom korišćenja Android mobilne aplikacije,
- profile\_image – broj koji se koristi za predstavljanje slike korisničkog profila koji se bira u Android mobilnoj aplikaciji,
- date\_created – datum kada je korisnički nalog kreiran,
- verified – potvrđuje da li je korisnički nalog verifikovan.

Tabela sa postavljenim markerima pod nazivom „bad\_markers“ sadrži informacije koje su dobijene putem dobrovoljnog postavljanja markera od strane korisnika Android mobilne aplikacije. Ove informacije se koriste za prikazivanje markera na mapi veb platforme i Android mobilne aplikacije, a one su:

- user\_id – korisnički jedinstveni identifikator koji se preuzima iz tabele korisnika, odnosno tabele „users“,
- color – kategorija markera koja se dijeli na crvenu i narandžastu,
- type – podkategorija koja može biti nezgoda, radovi ili policijske patrole,
- latitude – geografska širina koja se koristi u lociranju markera na mapi,
- longitude – geografska dužina koja se koristi u lociranju markera na mapi,

- date – datum postavljanja markera
- time – vrijeme postavljanja markera
- active – označava da li je korisnički nalog aktiviran.

Tabela sa parking markerima pod nazivom „parking\_markers“ sadrži informacije o parking zonama. Ove informacije se koriste za prikazivanje postavljenih markera na mapi veb platforme i Android mobilne aplikacije, a to su:

- title – naziv parking zone,
- latitude – geografska širina koja se koristi u lociranju parking markera na mapi,
- longitude – geografska dužina koja se koristi u lociranju parking markera na mapi,
- radius – radius kruga koji se koristi u Android mobilnoj aplikaciji za vizuelni prikaz parking zone na mapi,
- max\_place – maksimalan broj parking mjesta u parking zoni,
- free\_places – broj slobodnih parking mjesta.

Tabela za sistem nagrađivanja pod nazivom „rewards“ sadrži informacije korisnika koji su potvrdili da li je postavljeni marker tačan ili netačan. Na ovaj način jedan korisnik ne može dva (ili više) puta da potvrdi tačnost jednog markera. Ova tabela sadrži sljedeće informacije:

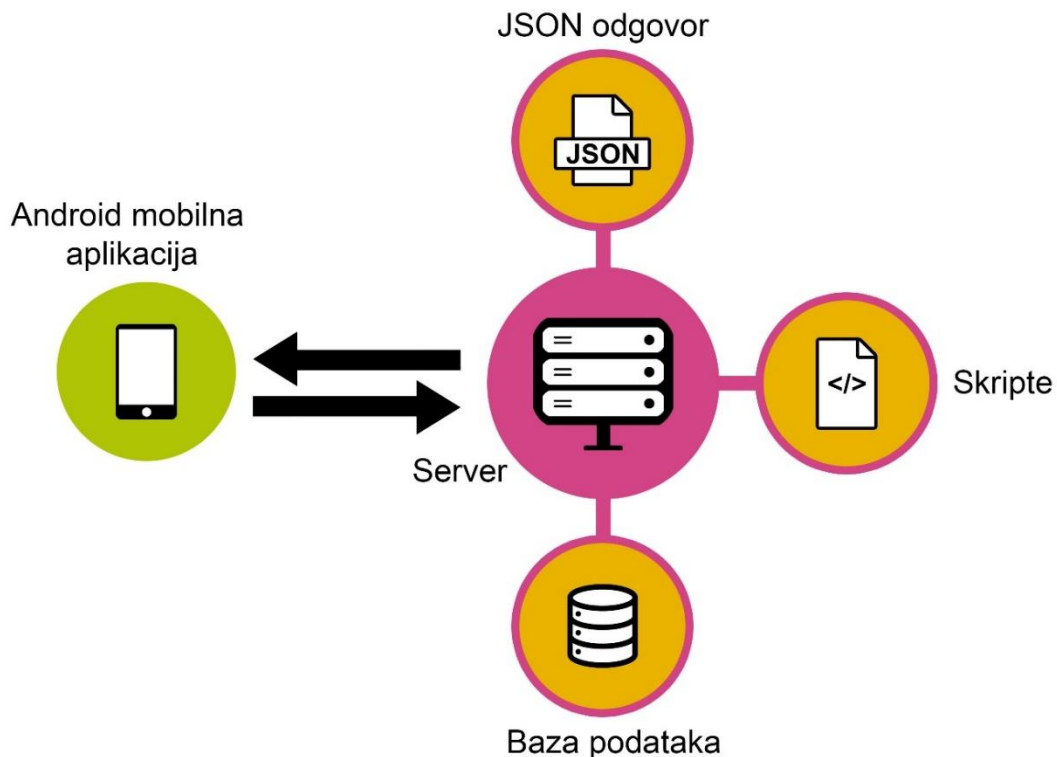
- user\_id – jedinstveni identifikator korisničkog naloga koji potvrđuje tačnost markera i preuzima se iz tabele korisnika, odnosno tabele „users“,
- user\_bad\_marker\_id – korisnički jedinstveni identifikator korisničkog naloga koji je postavio marker i on se preuzima iz tabele „bad\_markers“,
- bad\_marker\_id – jedinstveni identifikator postavljenog markera i on se preuzima iz tabele „bad\_markers“,
- answer – odgovor korisnika koji je potvrdio tačnost postavljenog markera.

Sve tabele u ovoj bazi podataka se koriste za veb platformu i Android mobilnu aplikaciju, kao i za pozadinske procese koji se koriste u izvršavanju aplikacije.

## 3.2. Serverski procesi

Serverski procesi obrađuju zahtjeve klijenata tako što izvršavaju zadate radnje. Ovi procesi preuzimaju zahtjeve od strane korisnika, obrađuju jednostavne ili složene zadatke, izvršavaju pretraživanje i ažuriranje baze podataka, upravljaju integritetom podataka i šalju odgovore na zahtjeve klijenata.

Dizajn serverskih procesa za Android mobilnu aplikaciju je osmišljen tako da korisnici aplikacije mogu po potrebi da šalju zahtjeve serveru, dok pojedini pozadinski procesi aplikacije šalju zahtjeve serveru po definisanim vremenskim intervalima. Ti zahtjevi sadrže informacije koje se koriste u skriptama za identifikaciju korisnika i obrađivanju podataka. Svi zahtjevi imaju definisanu destinaciju, odnosno svoju skriptu preko koje se i ista pokreće. Kada skripta ispuni svoj set zadatah radnji, šalje odgovor korisnicima u JSON formatu kako bi Android mobilna aplikacija mogla da preuzme odgovor. Ovaj proces je ilustrovan na sljedećoj slici (Slika 3.8.):



Slika 3.8. Dizajn komunikacije između Android mobilne aplikacije i servera

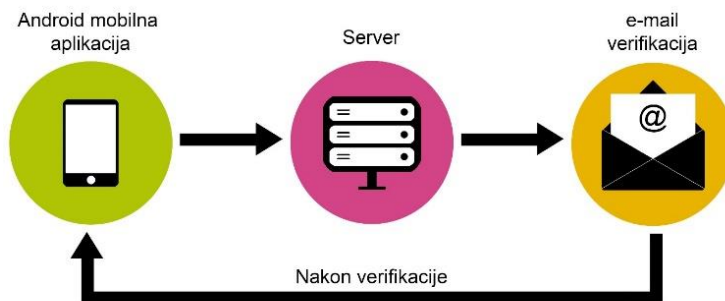
Procesi koji se nalaze na serveru su u većini predviđeni za brzu i efikasnu komunikaciju između Android mobilne aplikacije i servera. Takođe, na veb serveru se nalaze procesi koji se pokreću u određenim vremenskim intervalima. Jedan od takvih procesa je proces za deaktivaciju markera, odnosno da poslije određenog definisanog vremenskog intervala deaktivira marker. Ovim pristupom obezbjeđujemo tačnost informacija tako što skripte neće preuzimati i slati zastarjele informacije.

Kako je razvoj veb platforme i Android mobilne aplikacije uvezan, odnosno posjeduju blizak međusobni odnos tako što brojni procesi (skripte) preuzimaju dolazne podatke sa aplikacije nakon čega se obrađuju na serveru, dok veb platforma, kao što smo na početku naveli, pruža korisnicima brz i efikasan pristup informacijama o trenutnom stanju u saobraćaju.

Za uspješan razvoj ovih procesa i same komunikacije potrebno je definisati sistem identifikacije korisnika, sistem nagrađivanja i sistem prikazivanja i ažuriranja informacija.

### 3.2.1. Sistem identifikacije korisnika

Sistem identifikacije korisnika je osmišljen tako da se korisnici registruju putem Android mobilne aplikacije. Nakon toga informacije o novom korisniku pristižu na server i one se obrade, odnosno unesu u bazu podataka, ali korisnici još uvijek nijesu verifikovani. Odmah nakon unosa informacija u bazu podataka šalje se elektronska pošta sa linkom novokreiranog naloga kako bi taj korisnik klikom na link verifikovao isti. Nakon toga kreirani korisnik može da koristi sve funkcije Android mobilne aplikacije. Dizajn ovog sistema je prikazan na sljedećoj slici (Slika 3.9.):



Slika 3.9. Dizajn sistema za identifikaciju korisnika

Skripta koja procesira dolazne podatke iz Android mobilne aplikacije za registraciju novog korisnika se naziva „register.php“. Ova skripta, kao što joj samo ime govori, služi za registraciju novog korisnika. Informacije koje pristižu od strane Android aplikacije su korisnički nalog (username), elektronska pošta (email), šifra (password), dok se podaci kao što su datum kreiranja novog korisnika (date\_created) i verifikacija (verified) generišu u samoj skripti. Nakon prikupljanja podataka slijedi hešovanje (engl. hashing) šifre tako što se šifra novog korisnika hešuje pomoću PHP funkcije *password\_hash()*. Ova funkcija kreira novu hešovanu šifru koja koristi snažan matematički algoritam koji mapira podatke proizvoljne veličine u mali niz fiksne veličine i ova funkcija je isključivo u jednom smjeru, odnosno praktično je nemoguće vratiti u prvobitno stanje (Slika 3.10.). Takođe, PHP funkcija *password\_hash()* koristi **bcrypt** algoritam za hešovanje koji sadrži *so* (engl. salt) za zaštitu od *duginih tabela* (engl. rainbow tables). *So* je nasumični podatak koji se koristi kao dodatni unos u jednosmjernu funkciju za hešovanje, dok *dugine* tabele su tabele koje sadrže unaprijed definisane vrijednosti namjenjene za razbijanje hešovane šifre (engl. cracking password).

Primjer hešovane šifre pomoću **bcrypt** algoritama „*sifra12345*“ je „*\$2a\$12\$48UxhTNvAiKoPV16VzhNXek4UUfDjzAvW4FKw8iD.6nBFTWMMFfk2*“, gdje je:

- *\$2a\$* – identifikator algoritma za hešovanje,
- *12* – faktor troška (engl. cost factor) koji služi da kontroliše vrijeme koje je potrebno da se izračuna jedan heš,
- *48UxhTNvAiKoPV16VzhNXe* – kodirani nasumični podatak, odnosno *so*,
- *xk4UUfDjzAvW4FKw8iD.6nBFTWMMFfk2* – kodirana šifra, odnosno heš šifre.

Ovim procesom obezbjeđujemo autentičnost i integritet unešenih podataka, u ovom slučaju korisničke šifre kako bi se izbjeglo skladištenje iste kao čisti tekst u bazu podataka. Takođe se koristi za provjeru fajlova, dokumenata i drugih vrsta podataka.

```
$password = password_hash($password, PASSWORD_DEFAULT);
```

Slika 3.10. Funkcija koja hešuje šifru korisnika



Nakon hešovanja šifre novog korisnika potrebno je unijeti sve korisničke podatke u bazu podataka, odnosno u tabelu korisnika pod nazivom „users“ (Slika 3.11.).

```
$sql = "INSERT INTO users (username, email, password, date_created,
verified) VALUES (?, ?, ?, ?, ?)";
$stmt = $conn->prepare($sql);
$stmt ->
bind_param("ssssi", $username, $email, $password, $date_created, $verified);

$results = array();

if ($stmt->execute()) {
    $results["answer"] = "1";
    $results["message"] = "Registration completed, verify your account
with the e-mail we sent you.";
    echo json_encode($results);
} else {
    $results["answer"] = "0";
    $results["message"] = "Registration not completed, try again later.";
    echo json_encode($results);
}

$stmt->close();
$conn->close();
```

Slika 3.11. Kod za unos novo kreiranog korisnika u bazu podataka

Za unos podataka koristimo pristup „pripremljenih izjava“ (engl. prepared statements). Ovaj pristup se koristi za izvršavanje istih (ili sličnih) SQL upita više puta sa visokom efikasnošću. One funkcionišu tako što se prvo pripremi SQL upit i pošalje u bazu podataka pri čemu vrijednosti ostaju neodređene, odnosno postavljamo znak upitnik „?“ za svaku potrebnu vrijednost (Slika 3.11.). Zatim baza podataka analizira, kompajlira i vrši optimizaciju upita na osnovu zadatog SQL šablona i čuva rezultat bez izvršavanja. Kasnije kod veže vrijednosti za zadate parametre, u ovom slučaju podatke novog korisnika. Potom se izvršava pripremljena izjava i to na način da korisnik kasnije može dobiti povratnu informaciju, odnosno ako je uspješno izvršen ovaj korak onda korisnik dobija informaciju da se uspješno registrovao, u suprotnom dobija informaciju da se nije uspješno registrovao. Odgovor se šalje u JSON formatu

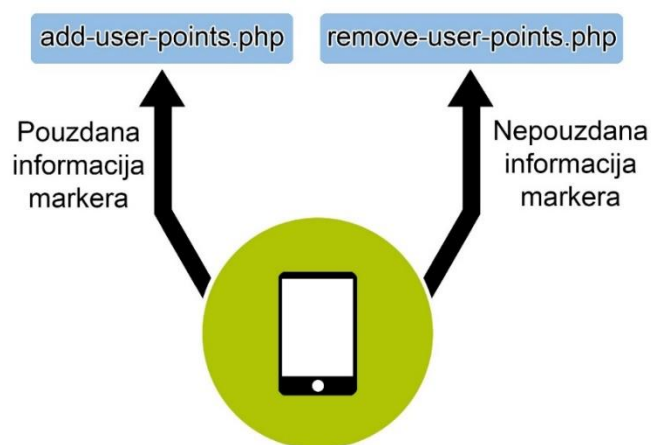
kako bi Android mobilna aplikacija mogla da preuzme i pročita odgovor. Na samom kraju zatvaramo konekciju sa bazom podataka i pripremljenu izjavu. Ovaj pristup je takođe, pored svoje efikasnosti, i bezbjedan, tako što štiti bazu podataka od SQL napada (engl. „SQL injection“).

Sljedeći korak za identifikaciju korisnika je slanje verifikacione elektronske pošte (email). Novi korisnici koji su unijeli svoju elektronsku poštu će dobiti generisani link preko kojeg će verifikovati svoj nalog klikom na njega. Link se generiše od dobijenih informacija pri registraciji korisnika, kao i datuma i vremena kreiranja korisničkog naloga.

Na ovaj način svi novi korisnici Android mobilne aplikacije moraju verifikovati svoj nalog kako bi mogli koristiti sve funkcije aplikacije. Takođe, elektronska pošta je jedinstvenog karaktera i ne može se ponoviti dva (ili više) puta u bazi podataka.

### 3.2.2. Sistem nagrađivanja

Sistem nagrađivanja je dizajniran tako da korisnici Android mobilne aplikacije potvrđuju tačnost markera na mapi i te informacije se šalju ka serveru kako bi se obradile. Skripte koje obrađuju dobijene informacije su skripta za pouzdanu informaciju koju smo nazvali „add-user-points.php“ i skripta za nepouzdanu informaciju koju smo nazvali „remove-user-points.php“. Ove skripte se pozivaju po potrebi od strane Android mobilne aplikacije, odnosno kada neki korisnik aplikacije potvrdi tačnost markera (Slika 3.12.).



Slika 3.12. Dizajn sistema za nagrađivanje korisnika

Ove skripte prvo preuzimaju dobijene podatke i skladište ih u definisane promjenljive. Podatke koje skripta preuzima su korisnički jedinstveni identifikator iz tabele korisnika pod nazivom „users“ i jedinstveni identifikator korisnika koji je postavio marker iz tabele markera. Pored navedenih podataka definišemo i još dvije promjenljive za poene, i to promjenljivu za tačnost markera koja se dodjeljuje korisniku koji je postavio marker i promjenljivu za potvrdu tačnosti koja se dodjeljuje korisniku koji je potvrdio. Ovim pristupom se nagrađuje korisnik koji je postavio marker i korisnik koji je potvrdio tačnost markera.

Sljedeći korak je da se korisnik koji je postavio marker odabere iz baze podataka, odnosno njegov ukupan broj poena iz tabele korisnika. To radimo tako što šaljemo SQL upit u kojem koristimo jednu od dobijenih informacija iz Android mobilne aplikacije, tj. jedinstveni identifikator korisnika koji je postavio marker. Za odabir ukupnog broja poena korisnika, takođe koristimo pristup „pripremljenih izjava“, ali se razlikuje u tome što ne unosimo podatke već preuzimamo podatke iz baze podataka. Pošto je princip isti kao za unos podataka ovdje imamo još jedan korak kojim se vežu rezultati izvršene pripremljene izjave. U ovom slučaju preuzimamo samo jedan podatak i to ukupan broj poena korisnika koji je postavio marker (Slika 3.13.).

```
$sql="SELECT points FROM users WHERE id=?";
if ($stmt = $conn->prepare($sql)) {
    $stmt->bind_param("i", $marker_user_id);
    $stmt->execute();
    $stmt->bind_result($marker_user_current_points);
    if ($stmt->fetch()) {
        $index["points"] = $marker_user_current_points;
    }
}
```

Slika 3.13. Kod za odabir informacije o poenima korisnika koji je postavio marker

Nakon ovog koraka skripte počinju da se razlikuju. Skripta koja vodi računa o pouzdanosti informacija o markerima pod nazivom „add-user-points.php“ ažurira poene korisnika koji je potvrdio tačnost informacije i poene korisnika koji je postavio marker tako što im uvećava poene za definisanu vrijednost, odnosno nagrađuje i jednog i drugog korisnika. Za

ažuriranje podataka, takođe koristimo pristup „pripremljenih izjava“. U ovom slučaju imamo dva šablona, po jedan za svakog korisnika. Prvo izvršavamo ažuriranje poena korisnika koji je potvrdio tačnost informacije postavljenog markera i ako se uspješno ažuriraju poeni tog korisnika, onda se ažuriraju poeni korisnika koji je postavio marker, nakon čega se šalje odgovor korisniku koji je potvrdio tačnost markera (Slika 3.14.). Na ovaj način oba korisnika dobijaju pozitivne poene, odnosno poeni se uvećavaju za već definisani broj.

```
$sql = "UPDATE users SET points=? WHERE id=?";
if ($stmt = $conn->prepare($sql)) {
    $user_points_total = $user_current_points + $user_add_points;
    $stmt->bind_param("ii", $user_points_total, $user_id);
    if($stmt->execute()) {

        $sql_marker_user = "UPDATE users SET points=? WHERE id=?";
        $stmt_marker_user = $conn->prepare($sql_marker_user);
        $marker_user_points_total = $marker_user_current_points + $marker_user_add_points;
        $stmt_marker_user->bind_param("ii", $marker_user_points_total, $marker_user_id);
        $stmt_marker_user->execute();
        $stmt_marker_user->close();

        $results["answer"] = "1";
        $results["message"] = "You and the marker provider received points.";
        $results["user_total_points"] = $user_points_total;
        echo json_encode($results);
    }
}
```

Slika 3.14. Kod za dodavanje poena korisnicima (add-user-points.php)

U slučaju da je postavljena informacija o markeru nepouzdana, korisnik koji je naišao na marker upotrebom Android mobilne aplikacije može da opovrgne tačnost ove informacije, odnosno da pošalje informaciju o nepouzdanom markeru. Ovu informaciju obrađuje skripta za nepouzdanu markere. Na ovaj način se nagrađuje korisnik koji potvrđuje tačnost markera, dok korisnik koji je postavio nepouzdanu marker dobija negativne poene, tj. oduzimaju mu se poeni (Slika 3.15.). Takođe se koristi pristup „pripremljenih izjava“. Prvi dio je identičan skripti za

pouzdanu informaciju, gdje izvršavamo ažuriranje poena korisnika koji je potvrdio tačnost informacije postavljenog markera i pri uspješnom ažuriranju prelazimo na korisnika koji je postavio nepouzdan marker. U ovom slučaju korisniku nepouzdanog markera se oduzimaju već definisani poeni pri čemu vodimo računa da ne dobije negativnu zbirnu vrijednost poena. Ako je ukupan broj poena negativan poslije ažuriranja, korisniku se dodjeljuje vrijednost od nula poena, nakon čega se šalje odgovor korisniku Android mobilne aplikacije. Na ovaj način korisniku koji je podijelio nepouzdanu informaciju o stanju u saobraćaju se oduzimaju poeni, dok se korisniku koji je potvrdio tačnost informacije uvećavaju poeni.

```
$sql = "UPDATE users SET points=? WHERE id=?";
if ($stmt = $conn->prepare($sql)) {
    $user_points_total = $user_current_points + $user_add_points;
    $stmt->bind_param("ii", $user_points_total, $user_id);
    if($stmt->execute()) {

        $sql_marker_user = "UPDATE users SET points=? WHERE id=?";
        $stmt_marker_user = $conn->prepare($sql_marker_user);
        $marker_user_points_total = $marker_user_current_points -
$marker_user_remove_points;

        if ($marker_user_points_total <= 0) {
            $marker_user_points_total = 0;
        }

        $stmt_marker_user-
>bind_param("ii", $marker_user_points_total, $marker_user_id);
        $stmt_marker_user->execute();
        $stmt_marker_user->close();

        $results["answer"] = "1";
        $results["message"] = "You received points while marker provider poin
ts have been dropped.";
        $results["user_total_points"] = $user_points_total;
        echo json_encode($results);
    }
}
```

Slika 3.15. Kod za dodavanje i oduzimanje poena korisnika (remove-user-points.php)

Ovaj pristup nagrađivanja garantuje da korisnici prikupljaju poene pri čemu im se mogu omogućiti dodatne funkcije u Android mobilnoj aplikaciji ili neki drugi vid nagrađivanja. Dok korisnicima koji postavljaju nepouzdana informacije se oduzimaju poeni i samim tim im se mogu uskratiti funkcije u aplikaciji ili možda neki drugi vid sankcionisanja kako bi spriječili bilo kakvu zloupotrebu aplikacije.

### 3.2.3. Sistem informacija

Sistem informacija ima za cilj da pruži korisnicima Android mobilne aplikacije uvid u trenutno stanje u saobraćaju. Jedan dio informacija se prikuplja od strane korisnika aplikacije koji dobrovoljno postavljaju iste, dok je drugi dio informacija predefinisani. Informacije koje se prikupljaju su o stanju u saobraćaju, a predefinisane informacije su o parking zonama. Sve informacije se prikazuju korisnicima aplikacije preko PHP skripti koje se pokreću u toku korišćenja aplikacije.

Skripte koje se koriste služe za prikaz informacija o parking zonama, prikaz informacija o stanju u saobraćaju i prikupljanje informacija o stanju u saobraćaju. Ove informacije se prikazuju pomoću definisanih grafičkih prikaza, odnosno markera.

Skripta za prikaz parking zona kojoj smo dodijelili naziv „show-parking-markers.php“ je osmišljena tako da iz baze podataka preuzima predefinisane informacije o svim parking zonama i šalje ih kao odgovor Android mobilnoj aplikaciji u JSON formatu. Za realizaciju ove skripte koristimo „pripremljene izjave“. Prvo je potrebno preuzeti potrebne podatke iz baze podataka i pripremiti izjavu. Zatim provjeravamo da li je uspješno izvršena pripremljena izjava, pri čemu se vrijednosti vežu za zadate parametre. Nakon toga se vrši obrada preuzetih podataka i njihovo raspoređivanje u predefinisani niz koji kasnije šaljemo kao odgovor u JSON formatu. Takođe, na samom kraju zatvaramo konekciju sa bazom podataka i pripremljenu izjavu. Ova skripta je predstavljena na sljedećoj slici (Slika 3.16.):

```
$results = array();
$results["show-parking-markers"] = array();
$sql = "SELECT id, title, latitude, longitude, radius, max_places, free_places FROM parking_markers";
```

```

    if ($stmt = $conn->prepare($sql)) {
        if ($stmt->execute()) {
            $stmt ->
bind_result($mId, $mTitle, $mLatitude, $mLongitude, $mRadius, $mMaxPlaces, $mFree
places);
            while ($stmt->fetch()) {
                $index["id"] = $mId;
                $index["title"] = $mTitle;
                $index["latitude"] = $mLatitude;
                $index["longitude"] = $mLongitude;
                $index["radius"] = $mRadius;
                $index["max-places"] = $mMaxPlaces;
                $index["free-places"] = $mFreeplaces;
                array_push($results["show-parking-markers"], $index);
            }
            $results["answer"] = "1";
            $results["message"] = "";
            echo json_encode($results);
        } else {
            $results["answer"] = "0";
            $results["message"] = "test";
            echo json_encode($results);
        }
    }
    $stmt->close();
    $conn->close();

```

Slika 3.16. Kod skripte za prikaz parking zona (show-parking-markers.php)

Skripta za prikaz markera koji predstavljaju stanje u saobraćaju pod nazivom „show-bad-markers.php“ je osmišljena na sličan način kao prethodna. Razlikuje se u tome što preuzima dodatnu informaciju od strane Android mobilne aplikacije. Informacija koju preuzima je status markera, u ovom slučaju je aktivnost tj. da li je marker aktivan ili nije. Kod ove skripte je prikazan na sljedećoj slici (Slika 3.17.):

```

$active = $_POST["active"];
$results = array();
$results["show-bad-markers"] = array();
$sql = "SELECT id, user_id, color, type, latitude, longitude FROM bad_markers
WHERE active = ?";

```

```

    if ($stmt = $conn->prepare($sql)) {
        $stmt->bind_param("i", $active);
        if ($stmt->execute()) {
            $stmt ->
bind_result($mId, $mUser_id, $mColor, $mType, $mLatitude, $mLongitude);
            while ($stmt->fetch()) {
                $index["id"] = $mId;
                $index["user_id"] = $mUser_id;
                $index["color"] = $mColor;
                $index["type"] = $mType;
                $index["latitude"] = $mLatitude;
                $index["longitude"] = $mLongitude;
                array_push($results["show-bad-markers"], $index);
            }
            $results["answer"] = "1";
            $results["message"] = "";
            echo json_encode($results);
        } else {
            $results["answer"] = "0";
            $results["message"] = "test";
            echo json_encode($results);
        }
    }
}
$stmt->close();
$conn->close();

```

Slika 3.17. Kod skripte za prikaz markera parking zona (show-bad-markers.php)

Skripta za prikupljanje informacija o stanju u saobraćaju od strane korisnika Android mobilne aplikacije je realizovana tako da preuzima informacije koje pristižu od aplikacije, obrađuje iste i unosi ih u bazu podataka. Takođe, pored informacija koje se preuzimaju, skripta sadrži i predefinisane promjenljive za datum i vrijeme koje se definišu na osnovu vremenske zone. Informacije koje se preuzimaju su:

- user\_id – korisnički jedinstveni identifikator,
- color – kategorija markera,
- type – podkategorija markera,
- latitude – geografska širina,
- longitude – geografska dužina.



Nakon preuzimanja informacija potrebno je provjeriti koja je kategorija, a to radimo tako što prolazimo kroz niz kategorijskih provjera, pri čemu dodjeljujemo vrijednosti tačne kategorije. Zatim pripremamo SQL upit koristeći „pripremljene izjave“ i iste vezujemo sa zadatim parametrima. Potom se izvršava pripremljeni upit i po završetku šalje odgovor korisniku aplikacije u JSON formatu. Takođe, zatvaramo konekciju sa bazom podataka i pripremljenom izjavom. Ova skripta je prikazana na sljedećoj slici (Slika 3.18.):

```
date_default_timezone_set("Europe/Podgorica");
$user_id = $_POST["user_id"];
$color = $_POST["color"];
$type = $_POST["type"];
$latitude = $_POST["latitude"];
$longitude = $_POST["longitude"];
$date = date("Y-m-d");
$time = date("H:i:s");
$active = 1;
if ($color == "red") {
    $color = 1;
}
if ($color == "orange") {
    $color = 2;
}
if ($type == "accident") {
    $type = 1;
}
if ($type == "works") {
    $type = 2;
}
if ($type == "police") {
    $type = 3;
}
$sql = "INSERT INTO bad_markers (user_id, color, type, latitude, longitude, date, time) VALUES (?, ?, ?, ?, ?, ?, ?)";
$stmt = $conn->prepare($sql);
$stmt->bind_param("iiissss", $user_id, $color, $type, $latitude, $longitude, $date, $time);
$results = array();
if ($stmt->execute()) {
    $results["answer"] = "1";
    $results["message"] = "Marker is added.";
    echo json_encode($results);
}
```

```
} else {  
    $results["answer"] = "0";  
    $results["message"] = "Marker is not added, try again.";  
    echo json_encode($results);  
}  
$stmt->close();  
$conn->close();
```

Slika 3.18. Kod skripte za unos podataka/markera (add-bad-marker.php)

Ovim putem sistem informacija će pružiti korisnicima veb platforme sve dostupne informacije koje se nalaze na serveru. Informacije se prikazuju na mapi (Slika 3.1.) u vidu markera koji su definisani specifičnim bojama i simbolima (Tabela 3.1.). Klikom na navedene markere korisnici dobijaju više informacija, kao što je precizan problem u saobraćaju ili broj slobodnih parking mjesta u parking zoni. Na ovaj način korisnici veb platforme mogu doći do informacija u saobraćaju na jednostavan način.

## 4. Razvoj mobilne aplikacije

Cilj razvoja Android mobilne aplikacije je da korisnicima brzo i efikasno pruži potrebne informacije o trenutnom stanju u saobraćaju. Takođe, aplikacija treba da pruži brojne funkcije kojima se omogućava korisnicima pojedinačni doprinos u dijeljenju korisnih informacija o stanju u saobraćaju. Ovaj vid prikupljanja informacija od korisnika koji dobrovoljno postavljaju iste se zove „izvor publike“ (engl. crowdsourcing). Izvor publike je praksa pridobijanja informacija ili doprinosa nekom projektu ili zadatku uključivanjem usluga velikog broja ljudi, bilo plaćenih ili neplaćenih, obično putem interneta.

Mobilna aplikacija komunicira sa serverom gdje se nalazi baza podataka. Takođe, na serveru su procedure, koje smo opisali u prethodnom poglavlju koje su neophodne za rad aplikacije.

Struktura Android mobilne aplikacije je dizajnirana tako da korisnicima bude jednostavna za korišćenje i da korisničko iskustvo (engl. user experience) bude zadovoljavajuće. Kroz aplikaciju korisnici šalju zahtjeve serveru gdje odgovarajuće skripte obavljaju pozadinske procese (engl. backend process) i odgovore šalju u JSON formatu. Zatim aplikacija preuzima odgovore i obrađuje ih u istoj kako bi korisnicima pružila potrebne informacije. Informacije su prezentovane u vizuelnom i tekstualnom obliku. Pojedini vizuelni prikazi su identični kao u slučaju veb rješenja, odnosno grafički prikaz ikonica je isti (Tabela 3.1.).

Razvoj ove Android mobilne aplikacije uključuje MVVM arhitekturu (engl. Model View View-Model architecture). Ovim putem obezbjeđujemo korisniku podatke uživo, odnosno kada korisnik ažurira pojedine podatke svi elementi koji predstavljaju isti se ažuriraju bez ponovnog pokretanja klase tj. aktivnosti.

Nakon registracije i/ili prijavljivanja korisnika na mobilnoj aplikaciji slijedi jedan od fragmenata u strukturi fragmenata. Ova struktura fragmenata je podijeljena na:

- profil korisnika,
- glavna mapa,
- planiranje saobraćajnih ruta,
- parking zone,

- glasovne komande.

Ovi fragmenti, kao što smo i prethodno naveli, su povezani sa pojedinim sistemima na serveru, kao i posebnim pozadinskim procesima koji obavljaju definisani set instrukcija koji su vezani za fragmente kojima je to potrebno.

Fragmenti su kontrolisani pomoću glavne aktivnosti (MainActivity.class). Takođe, kroz glavnu aktivnost se vrši i navigacija između pomenutih fragmenta. Kako bi postigli kontrolu navigacije potrebno je definisati element u XML fajlu za glavnu aktivnost koja će služiti za postavljanje/mijenjanje željenog fragmenta, a taj element je `<FrameLayout>` (Slika 4.1.).

```
<FrameLayout
    android:id="@+id/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

Slika 4.1. Dio programskog koda XML fajla glavne aktivnosti za postavljanje/mijenjanje fragmenata

Nakon definisanja XML elementa potrebno je u klasi glavne aktivnosti definisati prikaz „glavnog“ fragmenta. Glavni fragment je programski odabran da bude prvi prikaz, odnosno prvi fragment koji korisnik vidi i ne može da se promijeni (Slika 4.2.).

```
if (savedInstanceState == null) {
    getSupportFragmentManager().beginTransaction().replace(R.id.fragment_container, new MapFragment()).commit();
    navigationView.setCheckedItem(R.id.nav_map);
}
```

Slika 4.2. Dio programskog koda klase glavne aktivnosti za postavljanje glavnog fragmenata

Navigaciju između fragmenata obezbjeđujemo tako što prvo definišemo sve potrebne XML elemente. U XML fajlu glavne aktivnosti definišemo pomoću trake sa alatima koja sadrži dugme za prikaz navigacionog interfejsa i naziv mobilne aplikacije (Slika 4.3.).

```

<androidx.appcompat.widget.Toolbar
    android:id="@+id/main_toolbar"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="@color/_6CC417"
    android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
    app:popupTheme="@style/Theme.AppCompat.Light" />

```

Slika 4.3. Dio programskog koda XML fajla glavne aktivnosti za definisanje trake sa alatima

Potom definišemo meni koji sadrži nazive svih fragmenata i dodatne nazive koje koristimo za druge svrhe. Ovaj meni je zaseban XML fajl koji se kreira i unutar njega definišemo potrebne stavke u okviru kojih su pomenuti nazivi (Slika 4.4.).

```

<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <group android:checkableBehavior="single">

        <item
            android:id="@+id/nav_profile"
            android:title="@string/my_profile" />

        <item
            android:id="@+id/nav_map"
            android:title="@string/main_map" />

        <item
            android:id="@+id/nav_route_planning"
            android:title="@string/route_planning" />

        <item
            android:id="@+id/nav_parking_zones"
            android:title="@string/parking_zones" />

        <item
            android:id="@+id/nav_voice_commands"
            android:title="@string/voice_commands" />

        <item
            android:id="@+id/nav_logout"
            android:title="@string/logout" />

    </group>

</menu>

```

Slika 4.4. Programski kod XML fajla za meni

Zatim u XML fajlu glavne aktivnosti definišemo elemenat koji je zadužen za prikaz navigacionog interfejsa i to `<androidx.drawerlayout.widget.DrawerLayout>` (Slika 4.5.).

```
<androidx.drawerlayout.widget.DrawerLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/main_drawer_layout"
    tools:openDrawer="start">
    ...
</androidx.drawerlayout.widget.DrawerLayout>
```

Slika 4.5. Dio programskog koda XML fajla glavne aktivnosti za prikaz navigacionog interfejsa

Ovaj elemenat sadrži dva elementa i to element za fragmente koji smo prethodno opisali (Slika 4.1.) i element za navigacioni interfejs (Slika 4.6.).

```
<com.google.android.material.navigation.NavigationView
    android:id="@+id/nav_view"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:layout_gravity="start"
    app:headerLayout="@layout/navigation_header"
    android:fitsSystemWindows="true"
    app:menu="@menu/navigation_menu"/>
```

Slika 4.6. Dio programskog koda XML fajla glavne aktivnosti za navigacioni interfejs

Element za navigacioni interfejs je sastavljen iz dva segmenta, odnosno dva posebna XML fajla. Jedan segment je meni koji smo prethodno obrazložili, a drugi segment je navigaciono zaglavlje koje sadrži elemente za prikaz informacija o korisničkom profilu. Informacije koje se prikazuju su: korisničko ime (engl. username), elektronska pošta (engl. email), poeni (engl. points) i profilna slika (Slika 4.7.). Interakcija sa ovim informacijama je detaljno obrazložena u poglavlju „Profil korisnika“.

```
?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
```

```

android:background="@color/_ffffff"
android:padding="15dp"
android:theme="@style/ThemeOverlay.AppCompat.Dark"
android:layout_height="wrap_content"
android:orientation="horizontal">

<ImageView
    android:id="@+id/nav_profile_image"
    android:layout_width="105dp"
    android:layout_height="110dp"
    android:layout_gravity="center"
    android:src="@drawable/logo"/>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:orientation="vertical">

    <TextView
        android:id="@+id/nav_username"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textColor="@color/_333333"
        android:gravity="center"
        android:text="@string/username"/>

    <TextView
        android:id="@+id/nav_email"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textColor="@color/_333333"
        android:layout_marginTop="5dp"
        android:gravity="center"
        android:text="@string/email"/>

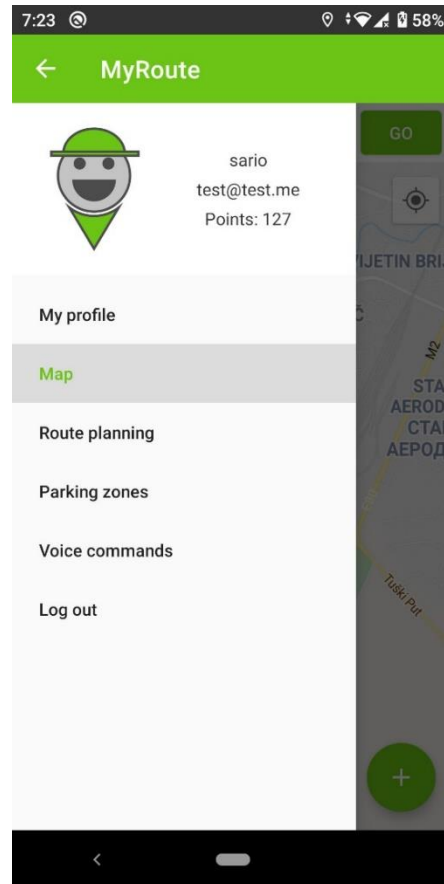
    <TextView
        android:id="@+id/nav_points"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textColor="@color/_333333"
        android:layout_marginTop="5dp"
        android:gravity="center"
        android:text="@string/points"/>

</LinearLayout>
</LinearLayout>

```

Slika 4.7. Programski kod XML fajla za navigaciono zaglavlje

Interfejs navigacionog menija u aplikaciji je prikazan na sljedećoj slici (Slika 4.8.):



Slika 4.8. Interfejs navigacionog menija

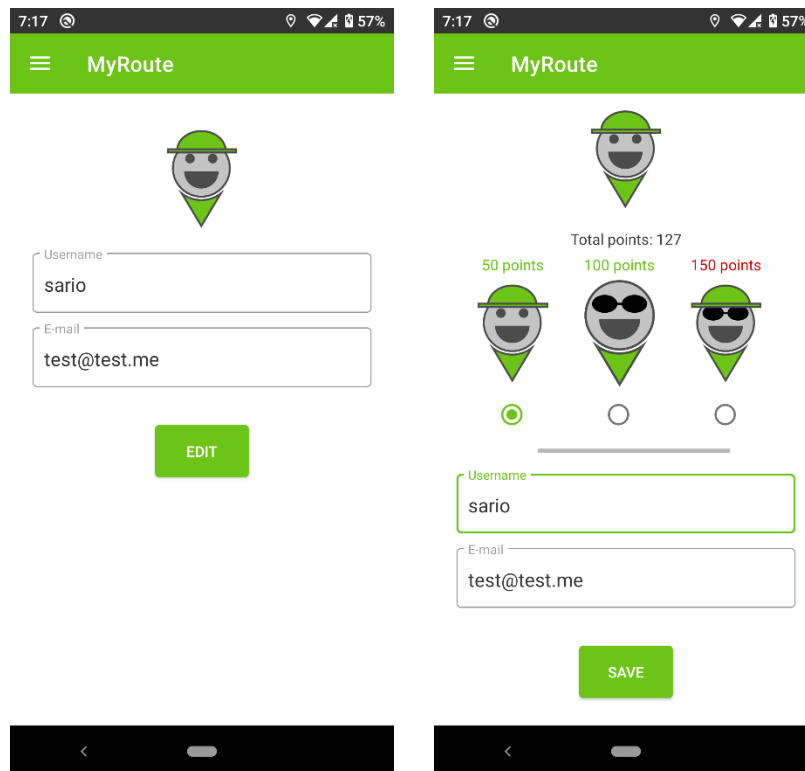
Navigacioni meni sadrži elemente kao što su profil korisnika, glavnu mapu, planiranje saobraćajnih ruta, parking zone, glasovne komande i odjavljivanje sa aplikacije. Svi navedeni elementi sadrže specifične funkcionalnosti koje pružaju korisnicima sve potrebne informacije za korišćenje aplikacije.

## 4.1. Profil korisnika

Profilni fragment ima za cilj da pruži korisnicima Android mobilne aplikacije prostor za individualnost, odnosno mogućnost da mijenjaju set informacija u skladu sa njihovim potrebama. Nakon što korisnik izvrši registraciju i/ili prijavu na mobilnu aplikaciju dobija svoj profil sa korisničkim imenom, elektronskom poštom i lozinkom. Ove informacije korisnik definiše pri



registraciji i može ih promijeniti bilo kada korišćenjem profilnog fragmenta. Takođe, dobija profilnu sliku koja je predefinisana za svakog novog korisnika aplikacije. Pored standardne profilne slike tu su i nekoliko dodatnih koje korisnik ne može da odabere sve dok ne skupi određeni broj poena. Ovo je jedan vid nagrađivanja korisnika koji je povezan sa sistemom nagrađivanja. Interfejs ovog fragmenta je na sljedećoj slici (Slika 4.9.) i to sa lijeve strane kada su podaci korisnika u trenutnom stanju, a sa desne strane kada korisnik odabere opciju za ažuriranje podataka:



Slika 4.9. Interfejs profilnog fragmenta, sa lijeve strane u trenutnom stanju, sa desne strane u stanju ažuriranja

Kako bi postigli funkcionalnosti za gore navedeni interfejs (Slika 4.9.) potrebno je prvo definisati sve potrebne elemente u XML fajlu za profilni fragment. Ovaj XML fajl fragmenta ima dva stanja, prvo u trenutnom stanju i drugo u stanju ažuriranja. Trenutno stanje sadrži četiri elementa. Jedan elemenat za prikaz profilne slike, dva tekstualna elementa za prikaz korisničkog imena i elektronske pošte, i jedno dugme koje služi za prikaz dodatnih polja u drugom stanju (Slika 4.10.).

```

<ImageView
    android:id="@+id/fragment_profile_profile_image"
    android:layout_width="100dp"
    android:layout_height="100dp"
    android:src="@drawable/logo"
    android:layout_gravity="center"
    android:layout_marginTop="10dp"
    android:layout_marginBottom="15dp" />

...

<com.google.android.material.textfield.TextInputLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
    android:layout_marginBottom="8dp">
    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/fragment_profile_username"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/username" />
</com.google.android.material.textfield.TextInputLayout>

<com.google.android.material.textfield.TextInputLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
    android:layout_marginBottom="8dp">
    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/fragment_profile_email"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/email" />
</com.google.android.material.textfield.TextInputLayout>

...

<com.google.android.material.button.MaterialButton
    android:id="@+id/fragment_profile_edit"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/edit"
    style="@style/Widget.AppCompat.Button.Colored"
    android:layout_marginTop="20dp"
    android:padding="15dp"
    android:layout_gravity="center"
    android:visibility="visible">
</com.google.android.material.button.MaterialButton>

```

Slika 4.10. Dio XML fajla profilnog fragmenta, prvo stanje

Stanje ažuriranja sadrži elemente koji su skriveni sve dok korisnik aplikacije ne klikne na dugme za ažuriranje, odnosno na isto dugme iz prvog stanja. Potom, korisniku se prikažu mogućnosti za promjenu trenutne elektronske pošte, kao i odabir nove profilne slike. Korisnik bira novu profilnu sliku tako što selektuje željenu sliku iz niza ponuđenih. Odabir je moguć samo u slučaju kada korisnik ima dovoljan broj poena u suprotnom nije moguć. Iznad svake profilne slike je prikaz potrebnog broja poena i on je zelene boje, ako korisnik ima ukupan broj poena isti ili veći od definisanog, u suprotnom je crvene boje (Slika 4.9.). Interfejs za prikaz profilnih slika realizovan je preko elemenata za linearni prikaz `<LinearLayout>` i listanje horizontalnog prikaza `<HorizontalScrollView>`. Ova dva elementa kombinujemo tako što prvom definišemo listanje horizontalnog prikaza koji sadrži dva linearna prikaza. Prvi linearni prikaz je definisan sa vertikalnom orijentacijom dok je drugi prikaz, koji je unutar prvog, definisan horizontalnom orijentacijom. Drugi linearni prikaz sadrži dva linearna prikaza koji su definisani horizontalnom orijentacijom. Prvi prikaz je predviđen da sadrži dva elementa koji predstavljaju prikaz broja potrebnih poena i profilnu sliku. Ova dva elementa su smještena u jedan linearni prikaz koji je definisan vertikalnom orijentacijom i on se ponavlja onoliko puta koliko ima profilnih slika.

Drugi prikaz sadrži jedan element za grupisanje radio dugmadi `<RadioGroup>` koji sadrži elemente radio dugmadi `<RadioButton>`. Radio dugmad služe za selekciju jedne opcije od niza ponuđenih i ovaj element koristimo da omogućimo korisnicima mobilne aplikacije da odaberu jednu profilnu sliku. Programski kod drugog stanja, odnosno stanje ažuriranja je prikazan na sljedećoj slici (Slika 4.11.).

```
<HorizontalScrollView
    android:id="@+id/fragment_profile_horizontalscrollview"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:visibility="visible"
    android:paddingTop="5dp"
    android:paddingStart="15dp"
    android:paddingEnd="15dp"
    android:paddingBottom="15dp">

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical">
```

```

<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_marginBottom="10dp">

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        <TextView
            android:id="@+id/fragment_profile_txt_profile_0"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Default"
            android:layout_gravity="center"
            android:textColor="@color/_333333"
            android:paddingBottom="5dp"/>
        <ImageView
            android:id="@+id/fragment_profile_profile_0"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:src="@drawable/logo"
            android:layout_gravity="center"/>
    </LinearLayout>

    ...

</LinearLayout>

<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <RadioGroup
        android:id="@+id/fragment_profile_rg"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:layout_gravity="center_horizontal"
        android:gravity="center_horizontal"
        android:layout_marginBottom="@dimen/global_padding_20">

        <RadioButton
            android:id="@+id/fragment_profile_rd_profile_0"
            android:layout_width="34dp"
            android:layout_height="wrap_content"
            android:layout_marginStart="33dp"
            android:layout_marginEnd="33dp"/>

        ...

    </RadioGroup>
</LinearLayout>

```

```
</LinearLayout>
</HorizontalScrollView>
```

Slika 4.11. Dio XML fajla profilnog fragmenta, drugo stanje

Nakon definisanja svih potrebnih elemenata u XML fajlu profilnog fragmenta potrebno je definisati potrebne funkcije u klasi profilnog fragmenta. Prvi korak prema realizaciji funkcije je definisanje XML elemenata u klasi kako bi mogli da ih koristimo, kao i potrebne promjenljive. Potom kreiramo funkciju koja je u sklopu fragmenta i povezana je sa MVVM arhitekturom, odnosno klasom koja sadrži definisanu strukturu za praćenje podataka uživo – *SharedViewModel.class*. U ovom fragmentu pratimo podatke kao što su korisnički jedinstveni identifikator (id), korisnički nalog, elektronska pošta (email), ukupan broj poena i profilna slika. Na ovaj način obezbjeđujemo ažuriranje pomenutih podataka bez ponovnog otvaranja klase. Programski kod ove funkcije je predstavljen na sljedećoj slici (Slika 4.12.):

```
@Override
public void onActivityCreated(@Nullable Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);
    viewModel = ViewModelProviders.of(getActivity()).get(SharedViewModel.class);

    viewModel.getId().observe(getViewLifecycleOwner(), new Observer<String>() {
        @Override
        public void onChanged(String s) {
            id = s;
        }
    });
    viewModel.getUsername().observe(getViewLifecycleOwner(), new Observer<String>() {
        @Override
        public void onChanged(String s) {
            fragment_profile_username.setText(s);
        }
    });
    viewModel.getEmail().observe(getViewLifecycleOwner(), new Observer<String>() {
        @Override
        public void onChanged(String s) {
            fragment_profile_email.setText(s);
        }
    });
    viewModel.getPoints().observe(getViewLifecycleOwner(), new Observer<String>() {
        @Override
        public void onChanged(String s) {
            points = s;
        }
    });
}
```

```

});
viewModel.getProfileImage().observe(getViewLifecycleOwner(), new
Observer<String>() {
    @Override
    public void onChanged(String s) {
        profile_image = s;
        if (Integer.valueOf(s) == 0) {
            im_profile_image.setImageResource(R.drawable.logo);
        } else if (Integer.valueOf(s) == 1) {
            im_profile_image.setImageResource(R.drawable.logo_1);
        } else if (Integer.valueOf(s) == 2) {
            im_profile_image.setImageResource(R.drawable.logo_2);
        } else if (Integer.valueOf(s) == 3) {
            im_profile_image.setImageResource(R.drawable.logo_3);
        } else if (Integer.valueOf(s) == 4) {
            im_profile_image.setImageResource(R.drawable.logo_4);
        }
    }
});
}
}
}

```

Slika 4.12. Programski kod funkcije profilnog fragmenta za praćenje promjene podataka

Sljedeći korak je kreiranje dvije funkcije za kontrolisanje mogućnosti prikaza i ažuriranja XML elemenata. Jedna funkcija je da omogući ažuriranje tekstualnih polja pomoću komande `.setEnabled(true)` i prikaz skrivenih elemenata pomoću komande `.setVisibility(View.VISIBLE)` kako bi korisnik aplikacije mogao da ažurira željena polja. Druga funkcija je da onemogući ažuriranje tekstualnih polja i sakrije elemente pomoću komandi `.setEnabled(false)` i `.setVisibility(View.GONE)` kako korisnik aplikacije ne bi slučajno promijenio podatke. Ove funkcije se realizuju kada korisnik klikne na dugme za ažuriranje gdje se pokreće prva funkcija, a pri čuvanju ažuriranih podataka pokreće se druga funkcija. Navedene funkcije na ovaj način obezbjeđuju pravilno skladištenje ažuriranih podataka u bazi podataka.

Sljedeći korak je da kreiramo funkciju kojom se korisniku omogućava selekcija profilne slike. Selekcija slike se vrši tako što korisnik klikne na jedno od ponuđenih radio dugmadi koje se nalaze ispod željene profilne slike. Nakon odabira i eventualno tekstualnog ažuriranja korisnik treba da klikne na dugme za ažuriranje podataka. U ovom trenutku se registruje odabrana slika u definisanoj promjenljivoj preko MVVM definisane strukture. Programski kod za postizanje odabira profilne slike koristi niz provjera gdje provjerava koja je slika odabrana iz niza ponuđenih radio dugmadi (Slika 4.13.).

```

int selectedProfileImage = radioButton_profileImage.getCheckedRadioButtonId();
radioButton_profileImage = getView().findViewById(selectedProfileImage);
if (radioButton_profileImage == radioButton_profileImage_0) {
    profile_image = "0";
    viewModel.setProfileImage(profile_image);
} else if (radioButton_profileImage == radioButton_profileImage_1) {
    profile_image = "1";
    viewModel.setProfileImage(profile_image);
} else if (radioButton_profileImage == radioButton_profileImage_2) {
    profile_image = "2";
    viewModel.setProfileImage(profile_image);
} else if (radioButton_profileImage == radioButton_profileImage_3) {
    profile_image = "3";
    viewModel.setProfileImage(profile_image);
} else if (radioButton_profileImage == radioButton_profileImage_4) {
    profile_image = "4";
    viewModel.setProfileImage(profile_image);
}

```

Slika 4.13. Programski kod funkcije profilnog fragmenta za odabir profilne slike korisnika

Nakon kreiranja funkcija za odabir profilne slike i omogućavanje ažuriranja tekstualnih polja potrebno je definisati funkciju za ažuriranje istih u bazi podataka. Ova funkcija pored preuzimanja odabrane slike, takođe preuzima podatke iz tekstualnih polja i priprema ih za slanje prema serveru. Slanje podataka se obavlja preko HTTP zahtjeva koji definišemo unutar funkcije i on sadrži metodu slanja podataka i destinaciju (Slika 4.14.). Metoda kojom šaljemo podatke je *POST* metoda i ona obezbjeđuje bezbjedan prenos podataka. Destinacija je adresa servera i destinacionog fajla, odnosno skripte koja obrađuje podatke.

```

StringRequest stringRequest = new StringRequest(Request.Method.POST,
URL_SAVE_EDITED_DATA,
    new Response.Listener<String>() {
        ...
    },
    new Response.ErrorListener() {
        ...
    })
{
    ...
};

RequestQueue requestQueue = Volley.newRequestQueue(getActivity());

```

```
requestQueue.add(stringRequest);  
}
```

Slika 4.14. Dio programskog koda funkcije za slanje ažuriranih podataka

Nakon što skripta obradi podatke ona vraća odgovor koji može biti pozitivan ili negativan. Aplikacija preuzima odgovor i obavještava korisnika o ishodu. U slučaju uspješne obrade, odnosno uspješnog ažuriranja korisničkih podataka aplikacija postavlja novu sesiju (engl. session) koja mijenja prethodnu. Prethodna sesija se kreira pri prijavi korisnika i koristi postojeće podatke. Sesija nam omogućava da korisnik ostane prijavljen u aplikaciji, tako da ne mora da se prijavljuje svaki put kada pokrene aplikaciju (Slika 4.15.).

```
jsonObject = new JSONObject(response);  
String answer = jsonObject.getString("answer");  
String message = jsonObject.getString("message");  
if (answer.equals("1")) {  
    Toast.makeText(getActivity(), message, Toast.LENGTH_SHORT).show();  
  
    viewModel.setUsername(new_username);  
    viewModel.setEmail(new_email);  
    viewModel.setPoints(points);  
    viewModel.setProfileImage(profile_image);  
  
    sessionManager = new SessionManager(getActivity());  
    sessionManager.createSession(id, new_username, new_email, points, profile_image);  
} else {  
    Toast.makeText(getActivity(), message, Toast.LENGTH_LONG).show();  
}
```

Slika 4.15. Dio programskog koda funkcije za slanje ažuriranih podataka – preuzimanje odgovora

U slučaju da dođe do greške pri slanju podataka destinacionoj skripti korisnik dobija obavještenje da je došlo do greške i da pokuša kasnije, tako što se koristi komanda *Toast.makeText()*.

Podatke koje šaljemo skripti su definisani sa posebnim ključevima (engl. key) koji služe za prepoznavanje i smještanje istih u odgovarajuće promjenljive koje su definisane u skripti i



koristimo ih za dalju obradu. Na sljedećoj slici je programski kod ovog dijela funkcije (Slika 4.16.).

```
@Override
protected Map<String, String> getParams() throws AuthFailureError {
    Map<String, String> params = new HashMap<>();
    params.put("id", id);
    params.put("new_username", new_username);
    params.put("new_email", new_email);
    params.put("points", points);
    params.put("new_profile_image", profile_image);

    return params;
}
```

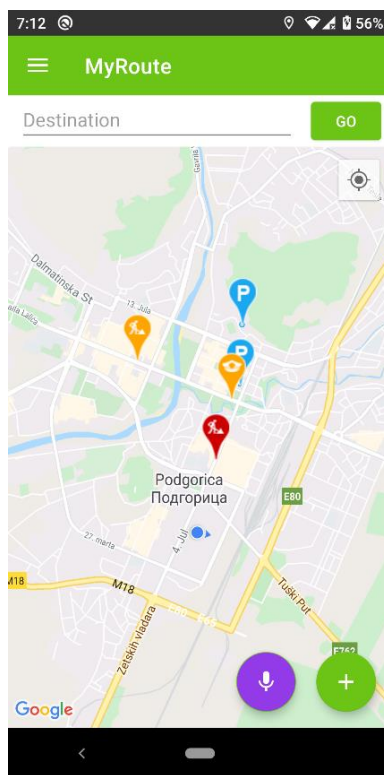
Slika 4.16. Dio programskog koda funkcije za slanje ažuriranih podataka – podaci koji se šalju

Na ovaj način profilni fragment pruža korisniku mobilne aplikacije jednostavan interfejs za pregled i bezbjedno ažuriranje podataka u bilo koje doba.

## 4.2. Glavna mapa

Fragment za prikaz glavne mape je prvi fragment koji se prikazuje korisniku nakon registracije/prijave na Android mobilnoj aplikaciji. Ovaj fragment je osmišljen sa ciljem da pruži korisniku glavne funkcionalnosti aplikacije na jednom mjestu. Korisnici aplikacije na ovom fragmentu mogu da vide sve prijavljene markere koji predstavljaju nezgode, radove i prisustvo policijskih patrola u saobraćaju, kao i sve parking zone sa relevantnim informacijama. Prikaz ovih informacija je predstavljen na mapi. Korisnici takođe mogu da sami prijave jedan od pomenutih markera putem plutajućeg menija (engl. floating menu) ili preko glasovnih komandi. Pored mogućnosti prijavljivanja i prikaza navedenih markera, korisnici mogu da potraže optimalnu saobraćajnu rutu do željene destinacije. Ovaj fragment je povezan sa ostalim fragmentima, a sve u cilju korišćenja jedne lokacije, odnosno jednog fragmenta za prikaz i unos informacija, prikaz rezultata i korišćenje glasovnih komandi za unos informacija. Dizajn ovog interfejsa je osmišljen tako da korisnicima aplikacije bude jednostavno prepoznavanje elemenata

koji su predstavljeni ikonicama i kratkim opisima. Prikaz ovog interfejsa dat je na sljedećoj slici (Slika 4.17.):



Slika 4.17. Interfejs fragmenta glavne mape

Za realizaciju prethodno navedenog interfejsa potrebno je prvo definisati sve elemente koje ćemo koristiti u XML fajlu. Interfejs ovog fragmenta možemo da podijelimo u tri sekcije. Prvu sekciju koristimo za pronalaženje optimalne saobraćajne rute do željene destinacije. U ovoj sekciji su dva elementa smještena u jedan linearni prikaz koji je definisan sa horizontalnom orijentacijom. Prvi element je za unos teksta dok je drugi dugme za pokretanje funkcije za pronalaženje saobraćajne rute. Programski kod prve sekcije je predstavljen na sljedećoj slici (Slika 4.18.).

```
<LinearLayout
    android:id="@+id/fragment_map_destination_go"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:weightSum="4"
    android:paddingStart="@dimen/global_padding_10"
```

```

android:paddingEnd="@dimen/global_padding_10"
android:visibility="visible">

<EditText
    android:id="@+id/fragment_map_destination"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_weight="3"
    android:layout_marginEnd="15dp"
    android:hint="@string/destination"/>

<com.google.android.material.button.MaterialButton
    android:id="@+id/fragment_map_go"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="@string/go"
    android:layout_gravity="center"
    style="@style/Widget.AppCompat.Button.Colored"
    android:visibility="visible">
</com.google.android.material.button.MaterialButton>
</LinearLayout>

```

Slika 4.18. Dio programskog koda XML fajla za fragment glavne mape – prva sekcija

Druga sekcija sadrži mapu koju koristimo za lociranje korisnika, prikaz informacija i rezultata. Mapa je implementirana pomoću *Maps SDK for Android API*-a koji preuzima informacije od strane *Google Maps API*-a. Ovaj dio programskog koda je prikazan na sljedećoj slici (Slika 4.19.).

```

<fragment
    android:id="@+id/fragment_map_mymap"
    class="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>

```

Slika 4.19. Dio programskog koda XML fajla za fragment glavne mape – druga sekcija

Treća sekcija se koristi za prikaz plutajućeg menija. Ova sekcija sadrži dva plutajuća dugmeta. Prvo dugme je za unos markera putem glasovnih komandi, dok je drugo dugme za otvaranje plutajućeg menija preko kojeg korisnici mogu da ručno unose markere. Ova dva

dugmeta su povezana sa fragmentima koje ćemo kasnije opisati. Kako bi postigli plutajući meni potrebno je pripremiti grafički prikaz ikonica (Tabela 3.1.) i definisati XML strukturu. Struktura sadrži dva linearna prikaza sa definisanom vertikalnom orijentacijom koji su smješteni u jedan linearni prikaz sa definisanom horizontalnom orijentacijom. Ova dva linearna prikaza sadrže dugmad koja se koriste za unos markera koji predstavljaju nezgode, radove i prisustvo policijskih patrola u saobraćaju. Ovu XML strukturu možemo da vidimo Slici 4.20.

```
<LinearLayout
    android:id="@+id/fragment_map_ll_fab_menu"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_alignParentBottom="true"
    android:layout_alignParentEnd="true"
    android:gravity="center_horizontal"
    android:layout_marginEnd="15dp"
    android:layout_marginBottom="90dp"
    android:visibility="visible">
    <LinearLayout
        android:id="@+id/fragment_map_ll_markers_red"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:gravity="center_horizontal">

        <com.google.android.material.floatingactionbutton.FloatingActionButton
            android:id="@+id/fragment_map_fab_accident_red"
            android:backgroundTint="@color/_cc0000"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/ic_white_accident"
            app:fabSize="mini"
            android:layout_margin="10dp"/>
        <com.google.android.material.floatingactionbutton.FloatingActionButton
            android:id="@+id/fragment_map_fab_works_red"
            android:backgroundTint="@color/_cc0000"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/ic_white_works"
            app:fabSize="mini"
            android:layout_margin="10dp"/>
        <com.google.android.material.floatingactionbutton.FloatingActionButton
            android:id="@+id/fragment_map_fab_police_red"
            android:backgroundTint="@color/_cc0000"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/ic_white_police"
            app:fabSize="mini"
            android:layout_margin="10dp"/>
    </LinearLayout>
</LinearLayout>
```

```

</LinearLayout>

<LinearLayout
    android:id="@+id/fragment_map_ll_markers_orange"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:gravity="center_horizontal">

    <com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/fragment_map_fab_accident_orange"
        android:backgroundTint="@color/_ffa500"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_white_accident"
        app:fabSize="mini"
        android:layout_margin="10dp"/>

    <com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/fragment_map_fab_works_orange"
        android:backgroundTint="@color/_ffa500"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_white_works"
        app:fabSize="mini"
        android:layout_margin="10dp"/>

    <com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/fragment_map_fab_police_orange"
        android:backgroundTint="@color/_ffa500"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_white_police"
        app:fabSize="mini"
        android:layout_margin="10dp"/>

</LinearLayout>

</LinearLayout>

```

Slika 4.20. Dio programskog koda XML fajla za fragment glavne mape – treća sekcija

Poslije definisanja strukture i svih neophodnih elemenata u XML fajlu fragmenta za glavnu mapu potrebno je definisati sve neophodne funkcije u klasi istog fragmenta. Prvi korak u postizanju funkcionalnosti gore navedenog interfejsa je definisati sve XML elemente u klasi kako bi ih koristili, kao i potrebne promjenljive. Potom definišemo mapu tako što implementiramo *OnMapReadyCallback* na klasu fragmenta i time dobijamo metodu *onMapReady*. Ova metoda služi za podešavanje mape i pokretanje pojedinih funkcija. Parametre koje podešavamo su ugrađeni elementi koji se pojavljuju na mapi. Ugrađeni elementi su

predstavljani kao dugmad na mapi i omogućili smo prikaz jednog takvog elementa za lociranje korisničkog uređaja dok smo prikaz ostalih onemogućili (Slika 4.21.).

```
mMap.setMyLocationEnabled(true);  
mMap.getUiSettings().setZoomControlsEnabled(false);  
mMap.getUiSettings().setMapToolbarEnabled(false);  
mMap.getUiSettings().setMyLocationButtonEnabled(true);
```

Slika 4.21. Dio programskog koda metode *onMapReady* za podešavanje ugrađenih elemenata za mapu

Funkcije koje pokrećemo preko ove metode su lociranje korisničkog uređaja i prikaz svih aktivnih markera.

Lociranje uređaja se vrši pomoću klase *LocationManager* koja pruža pristup uslugama istog. Ove usluge omogućavaju aplikacijama da dobijaju periodično ažurirane informacije o geografskom položaju uređaja.

Prikaz svih markera postizemo preko dvije slične funkcije. Jedna i druga koriste HTTP zahtjeve koje definišemo unutar pomenutih funkcija. Ove funkcije sadrže metodu slanja zahtjeva koja je *POST* i svoje destinacije, odnosno adrese skripti na destinacionom serveru. Jedna funkcija predstavlja lokacije aktivnih markera na mapi za nezgode, radove i prisustvo policijskih patrola, dok druga predstavlja parking zone sa maksimalnim i slobodnim brojem parking mjesta. Nakon što skripte uspješno obrade zahtjeve ovih funkcija one šalju odgovore u JSON formatu. U suprotnom, ako slanje nije uspješno obavještava se korisnik da je došlo do greške pri slanju podataka. Ove funkcije preuzimaju JSON odgovor za dalju obradu. Prva funkcija preuzima sljedeće informacije:

- *id* – jedinstveni identifikator markera,
- *user\_id* – jedinstveni identifikator korisnika koji je postavio marker,
- *color* – kategorija markera koja se dijeli na crvenu i narandžastu,
- *type* – podkategorija markera koji može biti nezgoda, radovi ili prisustvo policijskih patrola,
- *latitude* – geografska širina koja se koristi u lociranju markera na mapi,
- *longitude* – geografska dužina koja se koristi u lociranju markera na mapi.

Preuzete informacije se smiještaju u posebni niz za markere koji koristimo u nizu kategorijskih provjera. Na ovaj način markerima dodjeljujemo odgovarajuće definisane ikonice prije prikazivanja istih na mapi. Potom dodjeljujemo podatke prozoru za informacije (engl. Info Window) koji će se prikazati kada korisnik aplikacije klikne na jedan od markera. Programski kod ove funkcije je predstavljen na sljedećoj slici (Slika 4.22.).

```
JSONObject jsonObject = null;
try {
    jsonObject = new JSONObject(response);
    String answer = jsonObject.getString("answer");
    String message = jsonObject.getString("message");
    JSONArray jsonArray = jsonObject.getJSONArray("show-bad-markers");
    if (answer.equals("1")) {
        for (int i = 0; i < jsonArray.length(); i++) {
            JSONObject object = jsonArray.getJSONObject(i);
            String json_id = object.getString("id").trim();
            String json_user_id = object.getString("user_id").trim();
            String json_color = object.getString("color").trim();
            String json_type = object.getString("type").trim();
            String json_latitude = object.getString("latitude").trim();
            String json_longitude = object.getString("longitude").trim();

            LatLng json_latlng = new LatLng(Double.valueOf(json_latitude),
            Double.valueOf(json_longitude));
            MarkerOptions bad_markers = new MarkerOptions();

            if (json_color.equals("1")) {
                bMarkers.add(json_latitude + ", " + json_longitude);
                if (json_type.equals("1")) {
                    mMap.addMarker(bad_markers
                    .icon(BitmapDescriptorFactory.fromResource(R.drawable.marker_red_accident))
                    .title("Accident - not passable")
                    .position(json_latlng));
                } else if (json_type.equals("2")) {
                    mMap.addMarker(bad_markers
                    .icon(BitmapDescriptorFactory.fromResource(R.drawable.marker_red_works))
                    .title("Works - not passable")
                    .position(json_latlng));
                } else if (json_type.equals("3")) {
                    mMap.addMarker(bad_markers
                    .icon(BitmapDescriptorFactory.fromResource(R.drawable.marker_red_police))
                    .title("Police - not passable")
                    .position(json_latlng));
                }
            } else if (json_color.equals("2")) {
                if (json_type.equals("1")) {
                    mMap.addMarker(bad_markers
```

```

.icon(BitmapDescriptorFactory.fromResource(R.drawable.marker_orange_accident))
    .title("Accident - passable")
    .position(json_latlng));
} else if (json_type.equals("2")) {
    mMap.addMarker(bad_markers

.icon(BitmapDescriptorFactory.fromResource(R.drawable.marker_orange_works))
    .title("Works - passable")
    .position(json_latlng));
} else if (json_type.equals("3")) {
    mMap.addMarker(bad_markers

.icon(BitmapDescriptorFactory.fromResource(R.drawable.marker_orange_police))
    .title("Police - passable")
    .position(json_latlng));
}
}
InfoWindowAdapter adapter = new InfoWindowAdapter(getActivity());
mMap.setInfoWindowAdapter(adapter);
}
} else if (answer.equals("0")) {
    Toast.makeText(getActivity(), message, Toast.LENGTH_SHORT).show();
}
} catch (JSONException e) {
    e.printStackTrace();
    Toast.makeText(getActivity(), "Error: " + e.toString(),
Toast.LENGTH_SHORT).show();
}
}

```

Slika 4.22. Dio programskog koda funkcije za prikaz markera na mapi

Druga funkcija, slično kao prethodna, preuzima tipove podataka kao što su *id*, *latitude* i *longitude*, ali za parking markere. Takođe preuzima podatke koji su relevantni za parking mjesta, kao što su:

- title – naziv parking zone,
- radius – radius kruga koji se prikazuje na mapi kao vizuelni prikaz veličine,
- max-places – maksimalan broj parking mjesta,
- free-places – broj slobodih parking mjesta.

Ove podatke smještamo u posebni niz za parking markere koji koristimo za prikaz istih na mapi. Zatim dodjeljujemo ikonicu predviđenu za parking zone, kao i boju kruga koji predstavlja veličinu iste. Nakon čega dodjeljujemo podatke prozoru za informacije koji se



prikazuje kada korisnik aplikacije klikne na jedan od parking markera. Programski kod druge funkcije je prikazan na slici 4.23.

```
JSONObject jsonObject = null;
try {
    jsonObject = new JSONObject(response);
    String answer = jsonObject.getString("answer");
    String message = jsonObject.getString("message");
    JSONArray jsonArray = jsonObject.getJSONArray("show-parking-markers");
    if (answer.equals("1")) {

        for (int i = 0; i < jsonArray.length(); i++) {
            JSONObject object = jsonArray.getJSONObject(i);
            String json_id = object.getString("id").trim();
            String json_title = object.getString("title").trim();
            String json_latitude = object.getString("latitude").trim();
            String json_longitude = object.getString("longitude").trim();
            String json_radius = object.getString("radius").trim();
            String json_max_places = object.getString("max-places").trim();
            String json_free_places = object.getString("free-places").trim();

            LatLng json_latlng = new LatLng(Double.valueOf(json_latitude),
            Double.valueOf(json_longitude));
            String json_places = json_free_places + " / " + json_max_places ;

            MarkerOptions parking_markers = new MarkerOptions();

            Marker parking_marker = mMap.addMarker(parking_markers
            .icon(BitmapDescriptorFactory.fromResource(R.drawable.marker_blue_parking))
            .title(json_title)
            .snippet(json_places)
            .position(json_latlng));
            mMap.addCircle(new CircleOptions()
            .center(json_latlng)
            .radius(Double.parseDouble(json_radius))
            .strokeWidth(3f)
            .strokeColor(Color.rgb(27, 168, 241))
            .fillColor(Color.argb(70,27, 168, 241)));

            pMarkers.add(parking_marker.getId());

            InfoWindowAdapter adapter = new InfoWindowAdapter(getActivity());
            mMap.setInfoWindowAdapter(adapter);
        }

    } else if (answer.equals("0")) {
        Toast.makeText(getActivity(), message, Toast.LENGTH_SHORT).show();
    }
} catch (JSONException e) {
    e.printStackTrace();
    Toast.makeText(getActivity(), "Error: " + e.toString(),
```

```
Toast.LENGTH_SHORT).show();  
}
```

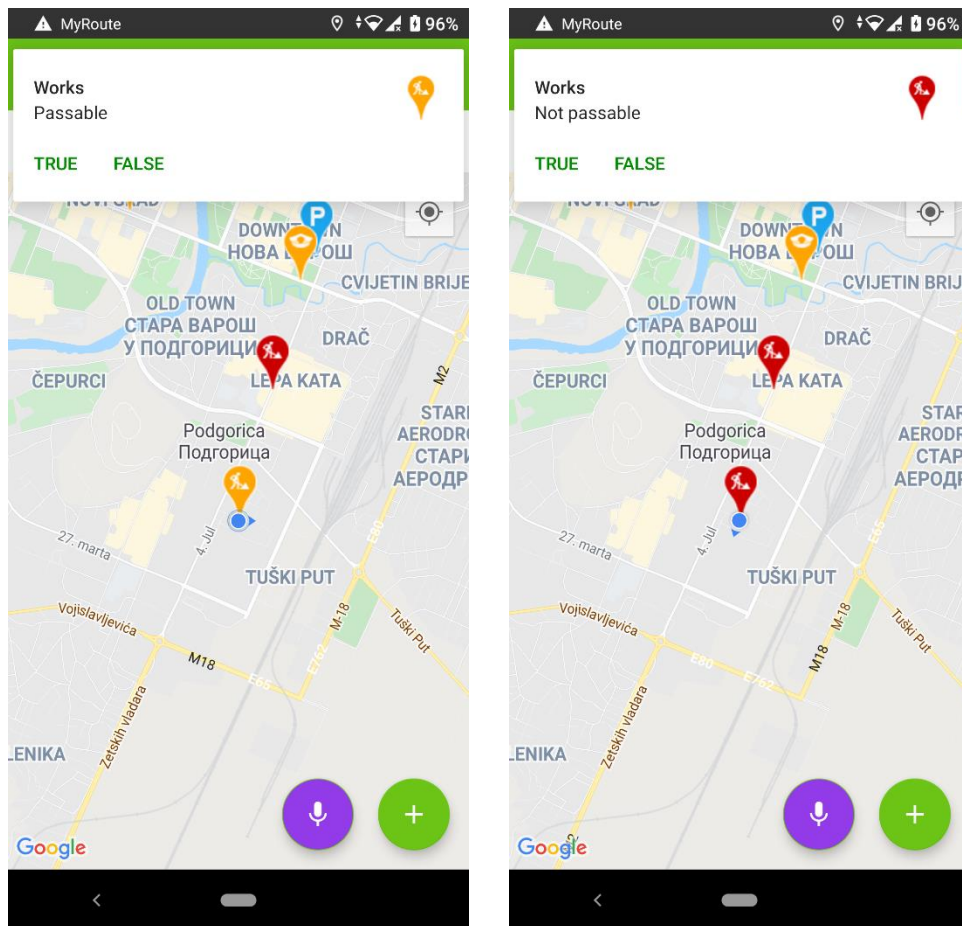
Slika 4.23. Dio programskog koda funkcije za prikaz parking markera na mapi

Pored funkcija za prikaz markera i parking markera korisnici aplikacije mogu i da postavljaju markere. Ovi markeri kao što smo i ranije naveli predstavljaju nezgode, radove i prisustvo policijskih patrola. Funkcija za postavljanje markera šalje HTTP zahtjev putem metode POST serveru na kojem se nalazi destinaciona skripta. Ova skripta obrađuje poslate podatke nakon čega ih unosi u bazu podataka i šalje odgovor aplikaciji u JSON formatu. Podatke koje korisnik aplikacije šalje su sopstveni jedinstveni identifikator (*user\_id*), boja markera (*color*), tip markera (*type*), geografska širina (*latitude*) i dužina (*longitude*). Odgovor od strane skripte može biti pozitivan ili negativan, odnosno da je uspješno unešen marker ili da nije. Aplikacija preuzima ovaj odgovor i prikazuje ga korisniku pomoću komande *Toast.makeText()* (Slika 4.24.). U slučaju da dođe do greške pri slanju ili prijemu podataka korisniku se prikazuje poruka da je došlo do problema i da pokuša kasnije. Na ovaj način omogućavamo korisniku unos podataka preko plutajućeg menija gdje se unos obavlja jednostavnim klikom na ikonicu koja predstavlja određeni problem. Ovo je jedan način unosa podataka, a drugi je preko glasovnih komandi koji ćemo opisati kasnije.

```
try {  
    JSONObject jsonObject = new JSONObject(response);  
    String answer = jsonObject.getString("answer");  
    String message = jsonObject.getString("message");  
    if (answer.equals("1")) {  
        Toast.makeText(getActivity(), message, Toast.LENGTH_SHORT).show();  
    } else {  
        Toast.makeText(getActivity(), message, Toast.LENGTH_LONG).show();  
    }  
} catch (JSONException e) {  
    e.printStackTrace();  
    Toast.makeText(getActivity(), "Error: " + e.toString(),  
    Toast.LENGTH_SHORT).show();  
}
```

Slika 4.24. Dio programskog koda funkcije za postavljanje novih markera

Nakon kreiranja funkcija koje služe za prikaz i unos markera potrebno je obezbijediti neki vid obavještenja, odnosno notifikacija (engl. notifications) u slučajevima kada se korisnik nađe u blizini markera. Na ovaj način korisnik aplikacije dobija obavještenje o markeru koji je u neposrednoj blizini i time se pruža korisniku mogućnost da se adekvatno pripremi za predstojeću prepreku na putu koja može biti nezgoda, radovi ili prisustvo policijskih patrola u saobraćaju. Primjeri ovih obavještenja su prikazani na sljedećoj slici (Slika 4.25).



Slika 4.25. Prikaz obavještenja

Kako bi postigli ovaj vid obavještenja potrebno je prvo kreirati funkciju koja će periodično da šalje koordinate mobilnog uređaja serveru. Na serveru se nalazi skripta koja obrađuje navedene podatke i vraća odgovor u JSON formatu. Aplikacija prima odgovor i provjerava da li je pozitivan ili negativan. U slučaju da je odgovor negativan aplikacija obavještava korisnika da je došlo do greške u suprotnom prolazi niz kategorijskih provjera. Ove

kategorijske provjere (Slika 4.26.) će obezbijediti odgovarajuće informacije koje se šalju drugoj funkciji za prikaz obavještenja, odnosno notifikaciju.

```
if (json_color.equals("1")) {
    bMarkers.add(json_latitude + ", " + json_longitude);
    if (json_type.equals("1")) {
        sendNotification("Accident","Not passable", "1-1", user_id);
    } else if (json_type.equals("2")) {
        sendNotification("Works","Not passable", "1-2", user_id);
    } else if (json_type.equals("3")) {
        sendNotification("Police","Not passable", "1-3", user_id);
    }
} else if (json_color.equals("2")) {
    if (json_type.equals("1")) {
        sendNotification("Accident","Passable", "2-1", user_id);
    } else if (json_type.equals("2")) {
        sendNotification("Works","Passable", "2-2", user_id);
    } else if (json_type.equals("3")) {
        sendNotification("Police","Passable", "2-3", user_id);
    }
}
```

Slika 4.26. Dio programskog koda funkcije za provjeru potrebe obavještanja

Druga funkcija preuzima podatke kao što su naslov (*title*), poruku (*message*), broj ikonice (*icon*) i jedinstveni identifikator korisnika (*id*). Naslov, poruka i jedinstveni identifikator korisnika su podaci koji su sami po sebi razumljivi, dok broj ikonice predstavlja jedinstveni broj u ovoj funkciji za kategorijsku provjeru. Ova provjera dodjeljuje odgovarajuću ikonicu koja se koristi kao vizuelni prikaz u notifikaciji. Programski kod za ovu kategorijsku provjeru je predstavljan na sljedećoj slici (Slika 4.27.).

```
if (nIcon.equals("1-1")) {
    imgIcon =
    BitmapFactory.decodeResource(getResources(),R.drawable.marker_red_accident);
} else if (nIcon.equals("1-2")) {
    imgIcon =
    BitmapFactory.decodeResource(getResources(),R.drawable.marker_red_works);
} else if (nIcon.equals("1-3")) {
    imgIcon =
    BitmapFactory.decodeResource(getResources(),R.drawable.marker_red_police);
} else if (nIcon.equals("2-1")) {
    imgIcon =
    BitmapFactory.decodeResource(getResources(),R.drawable.marker_orange_accident);
} else if (nIcon.equals("2-2")) {
```

```

    imgIcon =
    BitmapFactory.decodeResource(getResources(), R.drawable.marker_orange_works);
} else if (nIcon.equals("2-3")) {
    imgIcon =
    BitmapFactory.decodeResource(getResources(), R.drawable.marker_orange_police);
}

```

Slika 4.27. Dio programskog koda funkcije za slanje obavještenja – kategorijske provjere za dodjeljivanje ikonice

Nakon dodjeljivanja odgovarajuće ikonice potrebno je kreirati notifikaciju sa svim trenutno raspoloživim podacima. Za kreiranje ove notifikacije koristimo komandu *NotificationCompat.Builder()*, a potom je i šaljemo (Slika 4.28.).

```

Notification notification = new NotificationCompat.Builder(getActivity(),
NOTIFICATION)
    .setSmallIcon(R.drawable.ic_warning)
    .setContentTitle(nTitle)
    .setContentText(nMessage)
    .setLargeIcon(imgIcon)
    .setPriority(NotificationCompat.PRIORITY_HIGH)
    .setCategory(NotificationCompat.CATEGORY_MESSAGE)
    .setColor(Color.GREEN)
    .addAction(R.drawable.ic_mic_on, "True", actionIntentTrue)
    .addAction(R.drawable.ic_mic_off, "False", actionIntentFalse)
    .setAutoCancel(true)
    .setOnlyAlertOnce(true)
    .build();
notificationManager.notify(1, notification);

```

Slika 4.28. Dio programskog koda funkcije za slanje obavještenja – kreiranje i slanje notifikacije

Notifikacija sadrži dvije akcije, odnosno dva dugmeta. Dugmad se koriste za potvrdu tačnosti informacije, odnosno korisnik mobilne aplikacije potvrđuje da li je informacija o markeru u saobraćaju pouzdana ili nije. Ovaj vid notifikacija pruža korisnicima aplikacije mogućnost interakcije sa aktivnim podacima i dobrovoljni doprinos tačnosti informacija.

Notifikacije su povezane sa sistemom nagrađivanja tako što korisnici potvrđuju tačnost postavljenih aktivnih markera u saobraćaju i na osnovu toga dobijaju određeni broj poena. Korisnik aplikacije bira jednu od dvije opcije (Slika 4.25.) tako što klikne na jednu od ponuđenih

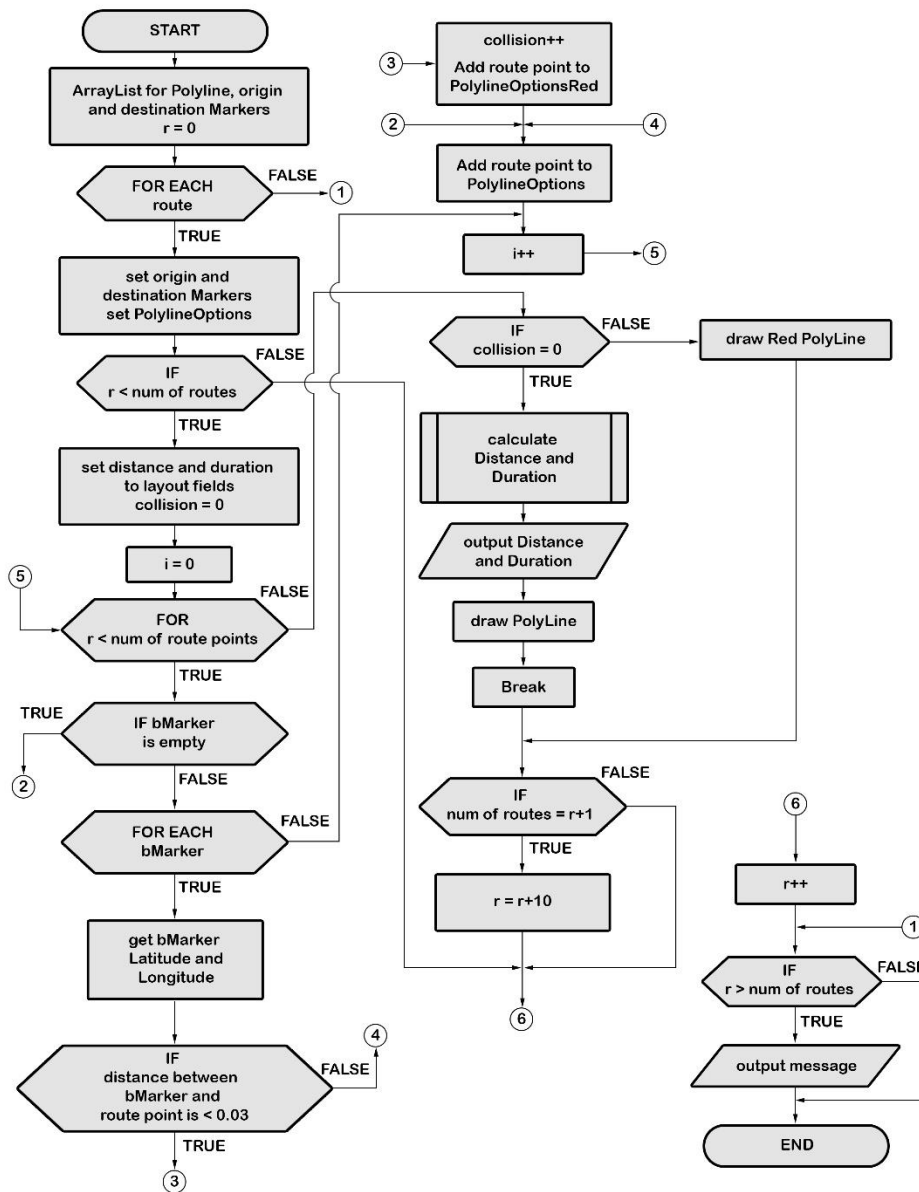
opcija pri čemu se odabrani odgovor šalje serveru gdje jedna od definisanih skripti preuzima i obrađuje podatke. Programski kod za odabir jedne od opcija je na sljedećoj slici (Slika 4.29.).

```
if (nIntent.equals("1")) {  
    notificationManager.cancel(1);  
    addPoints(nMarkerUserId);  
} else {  
    notificationManager.cancel(1);  
    removePoints(nMarkerUserId);  
}
```

Slika 4.29. Dio programskog koda funkcije za slanje odabrane opcije za ažuriranje poena korisnika

Dvije funkcije za ažuriranje poena korisnika su *addPoints()* i *removePoints()*. Ove dvije funkcije imaju za cilj da pošalju jedinstveni identifikator korisnika koji potvrđuje tačnost aktivnog markera i jedinstveni identifikator korisnika koji je postavio marker. Ovi podaci se obrađuju na serveru od strane odgovarajućih skripti nakon čega se šalje odgovor aplikaciji u JSON formatu. Odgovor sadrži podatke kao što su poruka i ukupan broj poena korisnika. Poruka prikazuje uspješnost izvršene procedure dok se ukupan broj preuzima i ažurira se interfejs koji prikazuje isti.

Ovaj fragment, glavna mapa, sadrži klase koje se koriste za pronalaženje optimalne saobraćajne rute između dvije ili više lokacija pri čemu se koristi algoritam za izbjegavanje aktivnih markera koji onemogućavaju prolaznost u saobraćaju. Klase koje pronalaze optimalnu saobraćajnu rutu su povezane sa Google Maps Directions API-om, odnosno šalju zahtjev sa potrebnim podacima kako bi isti vratio odgovarajući odgovor. Podaci koji se šalju su dvije lokacije i to trenutna lokacija mobilnog uređaja i željena destinacija. Potom Google Maps Directions API šalje odgovor u JSON formatu. Ovaj odgovor sadrži optimalnu rutu između dvije lokacije kao i maksimum dvije alternativne rute ako postoje, kao i distancu i potrebno vrijeme između dvije lokacije. Algoritam preuzima ovaj odgovor i obrađuje podatke kako bi prikazao optimalnu ili alternativnu saobraćajnu rutu. Algoritam je predviđen da dobijene podatke i podatke o postavljenim aktivnim markerima obradi i pruži korisniku vizuelni prikaz na mapi u vidu iscrtane linije između dvije ili više lokacija. Algoritam za pronalaženje optimalne saobraćajne rute je prikazan na sljedećoj slici (Slika 4.30.).



Slika 4.30. Algoritam za pronalaženje optimalne saobraćajne rute

Algoritam preuzima podatke o optimalnoj saobraćajnoj ruti, odnosno njene tačke koje sačinjavaju istu. Potom kalkuliše rastojanje između preuzete tačke sa podacima aktivnih markera koji su postavljani od strane korisnika Android mobilne aplikacije. Kalkulacija udaljenosti ove dvije tačke, odnosno geografske koordinate se vrši pomoću Haversinove formule (engl. Haversine formula):

$$d = 2r \sin^{-1} \left( \sqrt{\sin^2 \left( \frac{\varphi_2 - \varphi_1}{2} \right) + \cos(\varphi_1) \cos(\varphi_2) \sin^2 \left( \frac{\omega_2 - \omega_1}{2} \right)} \right)$$

Formula 4.1. Haversinova formula

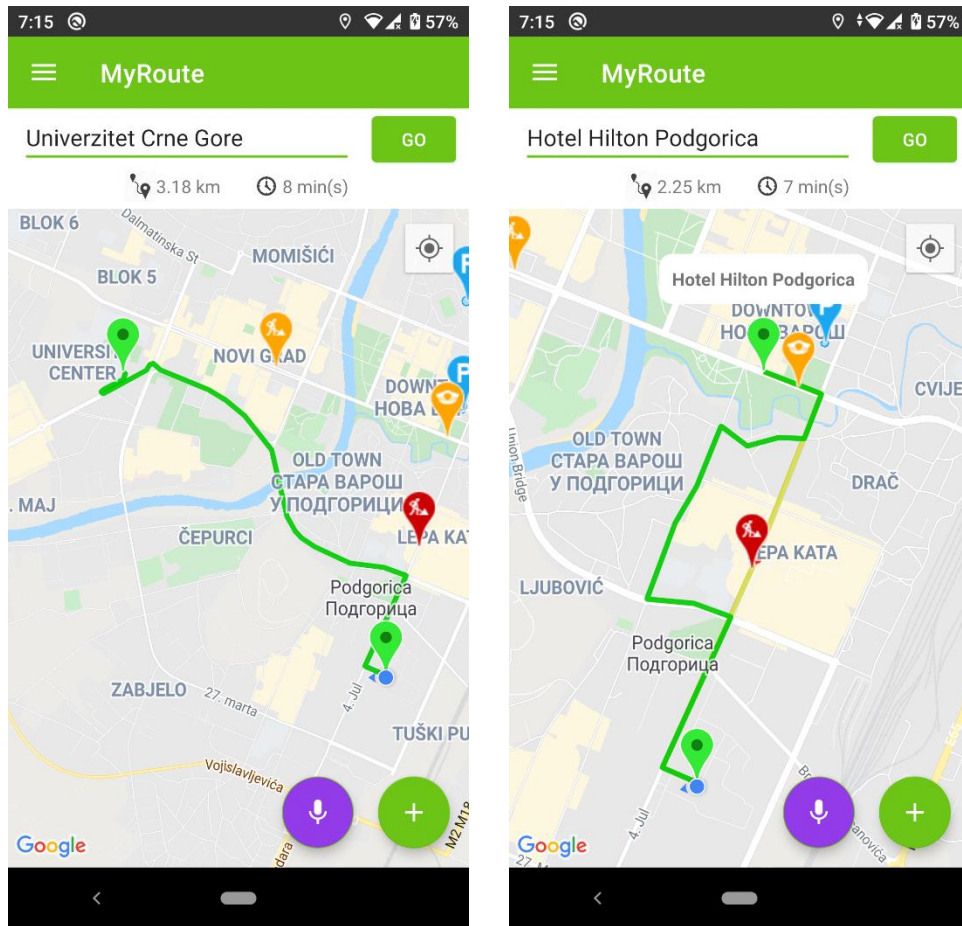
gdje su:

- $d$  – udaljenost između dvije lokacije, odnosno koordinate na sferi,
- $r$  – radijus Zemlje,
- $\varphi_1$  i  $\omega_1$  – geografska širina i dužina aktivnog markera,
- $\varphi_2$  i  $\omega_2$  – geografska širina i dužina preuzete tačke.

Ako je udaljenost između dvije koordinate dovoljna onda algoritam preuzima koordinatu i označava je zelenom bojom, odnosno da je prohodna saobraćajna ruta i nastavlja kalkulaciju sa sljedećim koordinatama aktivnog markera. U suprotnom označava koordinatu sa crvenom bojom, odnosno da nije prohodna saobraćajna ruta i prelazi na sljedeću koordinatu aktivnog markera. U procesu označavanja crvenom bojom algoritam povećava predefinisane promjenljivu koja predstavlja broj prepreka u optimalnoj saobraćajnoj ruti. Ovaj proces se ponavlja sve dok ne prođe aktivne markere nakon čega prelazi na sljedeću tačku, odnosno njenu koordinatu i ponavlja proces kalkulacije rastojanja sa aktivnim markerima.

Nakon ovih kalkulacija algoritam provjerava da li postoje prepreke u optimalnoj saobraćajnoj ruti. U slučaju da nema prepreka algoritam preuzima i kalkuliše podatke kao što su ukupna distanca i potrebno vrijeme između dvije lokacije. Ovi podaci se kalkulišu u predefinisane funkcije koje vraćaju odgovore u čitljivom formatu kao što su kilometri (km) i sati/minuti (h/min). Zatim sve ove rezultate predstavlja korisniku mobilne aplikacije u tekstualnom formatu kao što su ukupno potrebno vrijeme i distanca između lokacija, kao i zelenu liniju na mapi koja predstavlja optimalnu saobraćajnu rutu (Slika 4.31.).





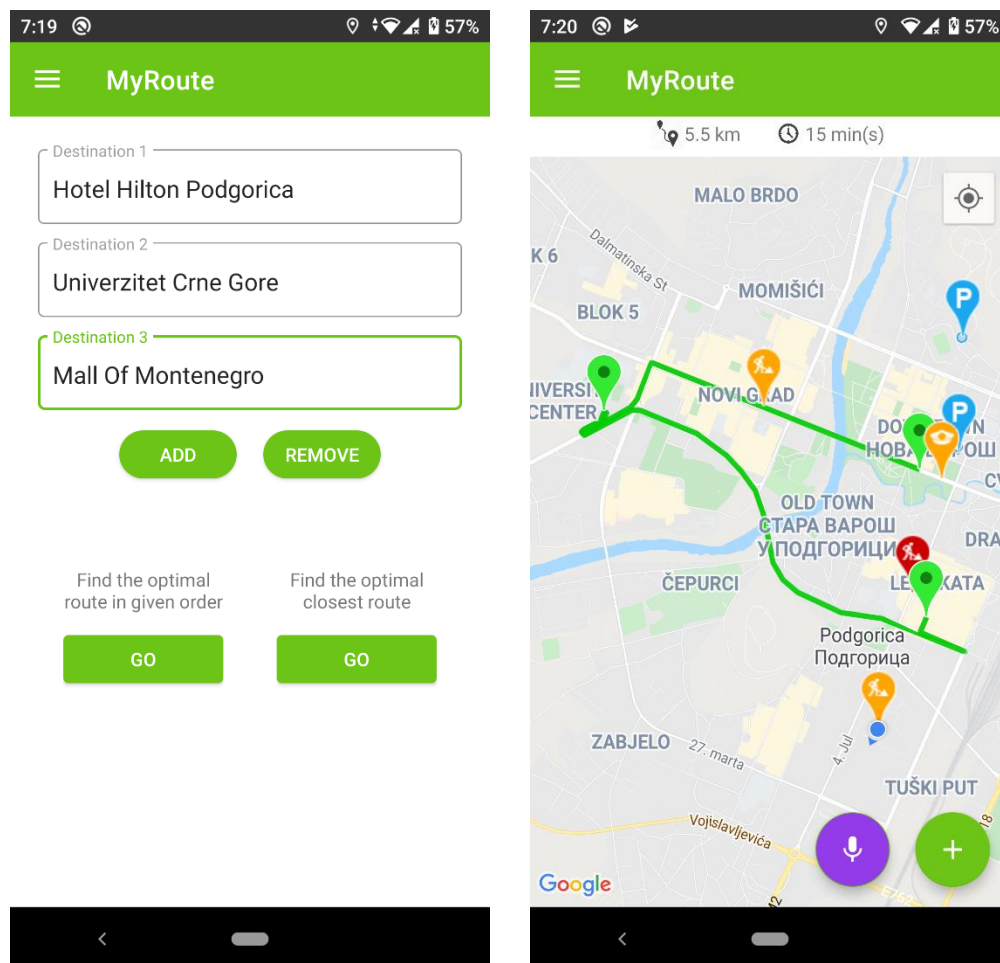
Slika 4.31. Prikaz optimalne (lijeva slika) i alternativne (desna slika) saobraćajne rute

U slučaju da ima prepreka na optimalnoj saobraćajnoj ruti, algoritam će predložiti alternativnu rutu koju će iscrtati zelenom bojom. Algoritam će za optimalnu saobraćajnu rutu koja je imala prepreku promijeniti boju u žutu kako bi korisniku vizuelno prikazao da je prvobitna ruta blokirana. Takođe, algoritam će dio blokirane rute u blizini aktivnog markera prikazati crvenom bojom kao indikator razloga nemogućće prohodnosti u saobraćaju (Slika 4.31.).

U slučaju da su sve saobraćajne rute blokirane algoritam će obavijestiti korisnika da su sve moguće rute trenutno blokirane i da pokuša kasnije ili da odabere neku drugu destinaciju koja je u blizini željene, odnosno prvobitno odabrane destinacije.

### 4.3. Planiranje saobraćajnih ruta

Fragment za planiranje saobraćajnih ruta je osmišljen tako da pruži korisnicima Android mobilne aplikacije opciju za planiranje ruta između više destinacija. Korisnici aplikacije mogu da unesu do 5 (pet) željenih destinacija nakon čega biraju jednu od dvije opcije za pronalaženje optimalne saobraćajne rute. Prva opcija je za pronalaženje optimalne saobraćajne rute u datom rasporedu, odnosno u rasporedu koji je korisnik zadao. Druga opcija je da aplikacija pronade optimalnu saobraćajnu rutu između zadatih destinacija. Na ovaj način korisnici mogu unaprijed da planiraju rute između željenih destinacija. Unos destinacija i rezultat ovog fragmenta je prikazan na sljedećoj slici (Slika 4.32.).



Slika 4.32. Prikaz za unos više destinacija (lijeva slika) i rezultat pronađene optimalne saobraćajne rute između više destinacija (desna slika)

Za postizanje ovog mehanizma u XML fajlu ovog fragmenta smo omogućili korisniku 5 (pet) polja za unos destinacija i to tako da se prvo prikazuju dva polja i po želji da korisnik dodaje ostala ili da ih uklanja (Slika 4.33.). Ovaj pristup pruža korisniku vizuelnu sliku koliko će destinacija da sadži planirana saobraćajna ruta.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:padding="@dimen/global_padding_20">

    <com.google.android.material.textfield.TextInputLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
        android:visibility="visible"
        android:layout_marginBottom="8dp">
        <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/fragment_route_destination_1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="@string/destination_1"/>
        </com.google.android.material.textfield.TextInputLayout>

    <com.google.android.material.textfield.TextInputLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
        android:visibility="visible"
        android:layout_marginBottom="8dp">
        <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/fragment_route_destination_2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="@string/destination_2"/>
        </com.google.android.material.textfield.TextInputLayout>

    ...
</LinearLayout>
```

Slika 4.33. Dio programskog koda XML fajla za planiranje saobraćajnih ruta

U klasi fragmenta za planiranje saobraćajnih ruta prvo definišemo sve elemente iz XML fajla ovog fragmenta kako bi ih koristili, kao i potrebne promjenljive. Potom definišemo funkcije za dodavanje, odnosno prikazivanje polja za unos više destinacija, kao i funkciju za uklanjanje,

odnosno skrivanje polja za unos više destinacija. Ove dvije funkcije jednostavnom manipulacijom vidljivosti prikazuju, odnosno skrivaju dodatna polja za unos destinacija (Slika 4.34.).

```
add_destination.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        if (counter == 2) {
            til_destination_3.setVisibility(View.VISIBLE);
            counter = 3;
        } else if (counter == 3) {
            til_destination_4.setVisibility(View.VISIBLE);
            counter = 4;
        } else if (counter == 4) {
            til_destination_5.setVisibility(View.VISIBLE);
            counter = 5;
            add_destination.setVisibility(View.GONE);
        }
        remove_destination.setVisibility(View.VISIBLE);
    }
});

remove_destination.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        if (counter == 5) {
            til_destination_5.setVisibility(View.GONE);
            destination_5.getText().clear();
            counter = 4;
        } else if (counter == 4) {
            til_destination_4.setVisibility(View.GONE);
            destination_4.getText().clear();
            counter = 3;
        } else if (counter == 3) {
            til_destination_3.setVisibility(View.GONE);
            destination_3.getText().clear();
            counter = 2;
            remove_destination.setVisibility(View.GONE);
        }
        add_destination.setVisibility(View.VISIBLE);
    }
});
```

Slika 4.34. Dio programskog koda klase za planiranje saobraćajnih ruta – funkcije za vidljivost dodatnih polja za unos destinacija

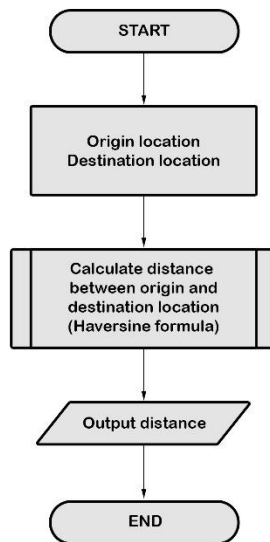
Sljedeći korak je da kreiramo funkciju koja će preuzeti unešene destinacije od strane korisnika. Ova funkcija takođe ažurira promjenljivu u MVVM arhitekturi, odnosno klasi koja

sadrži definisanu strukturu za praćenje podataka uživo – *SharedViewModel.class*. Ovaj pristup je predviđen za kontrolu funkcija za pronalaženje optimalne saobraćajne rute. Programski kod ove funkcije je prikazan na sljedećoj slici (Slika 4.35.).

```
public void getDestinations() {
    viewModel.setCounterControl("true");
    if (!destination_1.getText().toString().isEmpty()) {
        txt_destination_1 = destination_1.getText().toString();
        viewModel.setDestination1(txt_destination_1);
    }
    if (!destination_2.getText().toString().isEmpty()) {
        txt_destination_2 = destination_2.getText().toString();
        viewModel.setDestination2(txt_destination_2);
    }
    if (!destination_3.getText().toString().isEmpty()) {
        txt_destination_3 = destination_3.getText().toString();
        viewModel.setDestination3(txt_destination_3);
    }
    if (!destination_4.getText().toString().isEmpty()) {
        txt_destination_4 = destination_4.getText().toString();
        viewModel.setDestination4(txt_destination_4);
    }
    if (!destination_5.getText().toString().isEmpty()) {
        txt_destination_5 = destination_5.getText().toString();
        viewModel.setDestination5(txt_destination_5);
    }
}
```

Slika 4.35. Programski kod klase za preuzimanje unešenih destinacija od strane korisnika

Ova promjenljiva se koristi u klasi fragmenta glavne mape za provjeru da li će se koristiti planiranje saobraćajnih ruta. U slučaju da se koristi onda se algoritam poziva onoliko puta koliko je potrebno za pronalaženje optimalne saobraćajne rute između zadatih destinacija i kao rezultat dobijamo jednu zelenu liniju, odnosno saobraćajnu rutu na mapi (Slika 4.32.). Svaki poziv algoritma prolazi kroz već obrazloženi proces u fragmentu glavne mape. U slučaju da je korisnik odabrao pretragu optimalne saobraćajne rute između više lokacija, odnosno ne u zatom redosljedu, koristi se Haversinova formula (Formula 4.1.) i algoritam za sortiranje (Slika 4.36).

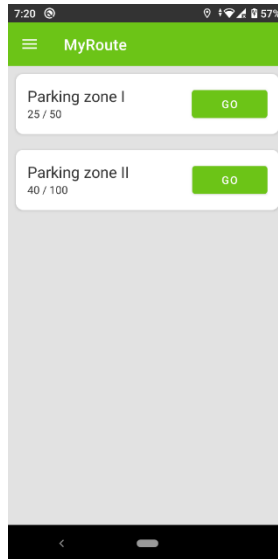


Slika 4.36. Algoritam za sortiranje

Algoritam za sortiranje funkcioniše tako što za polaznu tačku uzima lokaciju mobilnog uređaja i pomoću Haversinove formule upoređuje sa ostalim zadatim lokacijama i kao rezultat dobija najbližu zadatu lokaciju od lokacije mobilnog uređaja. Zatim se za polaznu tačku uzima rezultat prethodnog poređenja i upoređuje se sa preostalim zadatim lokacijama. Navedeni proces se ponavlja dok se ne iscrpe sve zadate lokacije i kao rezultat dobijamo sortirani niz lokacija, odnosno destinacija. Ovaj niz destinacija se zatim koristi u fragmentu glavne mape gdje se algoritam za pronalaženja optimalne saobraćajne rute poziva onoliko puta koliko je potrebno kroz već obrazloženi proces.

## 4.4. Parking zone

Fragment za parking zone ima za cilj da pruži korisnicima Android mobilne aplikacije listu postojećih parking zona sa relevantnim podacima. Podaci koji se prikazuju u listi su ime parking zone, ukupan i slobodan broj parking mjesta. Takođe, pored svake parking zone se nalazi dugme koje služi za pronalaženje optimalne saobraćajne rute. Korisnici mobilne aplikacije u ovom fragmentu imaju jednostavan interfejs preko kojeg mogu da pregledaju sve parking zone i pronađu slobodno mjesto za parking, kao i da pronađu optimalnu saobraćajnu rutu do odabrane parking zone. Interfejs liste sa parking zonama je prikazan na sljedećoj slici (Slika 4.37.).



Slika 4.37. Interfejs liste sa parking zonama

Kako bi postigli ovaj interfejs potrebno je definisati dva XML fajla. Jedan fajl služi za prikazivanje jednog zapisa, u ovom slučaju jedne parking zone. Ovaj zapis se koristi kao standardni šablon za prikazivanje svih ostalih parking zona (Slika 4.38.).

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:padding="@dimen/global_padding_15"
    android:weightSum="2">

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:layout_weight="1.8">

        <TextView
            android:id="@+id/parking_zone_item_title"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Parking zone name"
            android:textColor="@color/_333333"
            android:textSize="20sp"/>

        <TextView
            android:id="@+id/parking_zone_item_places"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Parking zone places"
            android:textColor="@color/_333333"/>
    </LinearLayout>
    <TextView
        android:id="@+id/go_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="go"
        android:textColor="@color/_333333"
        android:textSize="20sp"
        android:background-color="@color/_333333"
        android:color="white"
        android:padding="10px"
        android:margin-left="10px"/>
</LinearLayout>
```

```

</LinearLayout>

<com.google.android.material.button.MaterialButton
    android:id="@+id/parking_zone_item_go"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="0.2"
    android:text="@string/go"
    android:clickable="false"/>

</LinearLayout>

```

Slika 4.38. Programski kod XML fajla za prikaz jednog zapisa, odnosno šablon

Drugi XML fajl služi za prikazivanje svih parking zona, odnosno svih zapisa iz baze podataka koristeći navedeni šablon (Slika 4.39.). Ovaj pristup koristi *RecyclerView* dodatak (engl. widget). Ovaj dodatak je naprednija i fleksibilnija verzija *ListView* dodatka i koriste se za izlistavanje zapisa iz baze podataka u vertikalnoj listi.

```

<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/fragment_parking_zones_recyclerview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/_e2e2e2"/>

```

Slika 4.39. Programski kod XML fajla za izlistavanje zapisa

Izlistavanje parking zona se postiže pomoću funkcije koja šalje HTTP zahtjev putem metode POST serveru na kojem se nalazi destinaciona skripta. Ova skripta prikuplja sve potrebne podatke iz baze podataka i šalje odgovor mobilnoj aplikaciji u JSON formatu. Podaci koji se nalaze u odgovoru su jedinstveni identifikator parking zone (*id*), naziv (*title*), geografska širina i dužina (*latitude*, *longitude*), radius (*radius*), maksimalni broj mjesta (*max-places*) i slobodan broj mjesta (*free-places*). Ovi podaci se zatim koriste u klasi za kreiranje zapisa za parking zone nakon čega se smješta u predefinisanu listu za istu (Slika 4.40.).

```

JSONObject jsonObject = null;
try {
    jsonObject = new JSONObject(response);
    String answer = jsonObject.getString("answer");
}

```



```

String message = jsonObject.getString("message");
JSONArray jsonArray = jsonObject.getJSONArray("show-parking-markers");
if (answer.equals("1")) {

    for (int i = 0; i < jsonArray.length(); i++) {
        JSONObject object = jsonArray.getJSONObject(i);
        String json_id = object.getString("id").trim();
        String json_title = object.getString("title").trim();
        String json_latitude = object.getString("latitude").trim();
        String json_longitude = object.getString("longitude").trim();
        String json_radius = object.getString("radius").trim();
        String json_max_places = object.getString("max-places").trim();
        String json_free_places = object.getString("free-places").trim();

        String json_latlng = json_latitude + "," + json_longitude;

        mParkingZoneList.add(new ParkingZoneItem(json_title, json_max_places,
json_free_places, json_latlng));
    }

} else if (answer.equals("0")) {
    Toast.makeText(getActivity(), message, Toast.LENGTH_SHORT).show();
}

mParkingZoneAdapter = new ParkingZoneAdapter(getActivity(), mParkingZoneList);
mRecyclerView.setAdapter(mParkingZoneAdapter);
mParkingZoneAdapter.setOnItemClickListener(ParkingZonesFragment.this);
} catch (JSONException e) {
    e.printStackTrace();
    Toast.makeText(getActivity(), "Error: " + e.toString(),
Toast.LENGTH_SHORT).show();
}

```

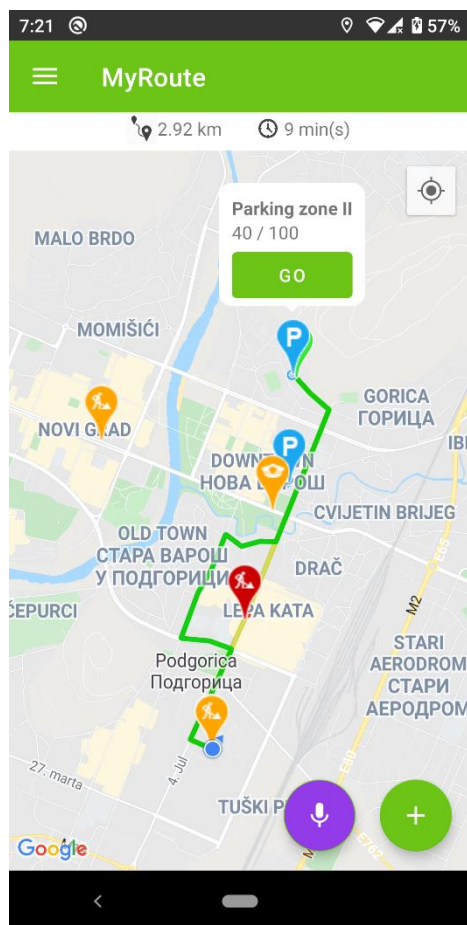
Slika 4.40. Dio programskog koda funkcije za preuzimanje podataka o parking zonama iz baze podataka

Zatim se ova lista predaje adapteru (engl. adapter) za parking zone. Ovaj adapter se koristi kao konekcija, odnosno most između interfejsa komponenti i podataka. Kada adapter preuzme podatke on ih šalje odgovarajućoj komponenti, u ovom slučaju elementima u šablonu koji smo pripremili za prikaz parking zona.

Pored svake parking zone, kao što smo već pomenuli, nalazi se dugme za pronalaženje optimalne saobraćajne rute. Ovo dugme omogućava korisniku da iz liste pređe na fragment glavne mape sa prikazom optimalne saobraćajne rute do odabrane parking zone.

Kako bi postigli ovu funkcionalnost potrebno je kreirati funkciju koja ažurira odgovarajuću promjenljivu u MVVM arhitekturi, odnosno klasi koja sadrži definisanu strukturu za praćenje podataka uživo – *SharedViewModel.class*. Ova klasa takođe preuzima selektovanu parking zonu od strane korisnika i ažurira drugu odgovarajuću promjenljivu u MVVM arhitekturi.

Ovaj pristup obezbeđuje kontrolu nad funkcijama za pronalaženje optimalne saobraćajne rute. Potom se prelazi na fragment glavne mape i navedene promjenljive se koriste u istom fragmentu za provjeru da li je korisnik odabrao parking zonu sa liste parking zona i ako jeste da preko već obrazloženog procesa za pronalaženje optimalne saobraćajne rute prikaze rezultate na mapi. Korisnici takođe mogu da pronađu optimalnu saobraćajnu rutu i preko ikonice za parking zone na mapi tako što klikom na ikonicu dobijaju podatke o odabranoj parking zoni i dugme za pronalaženje optimalne saobraćajne rute do iste (Slika 4.41.).

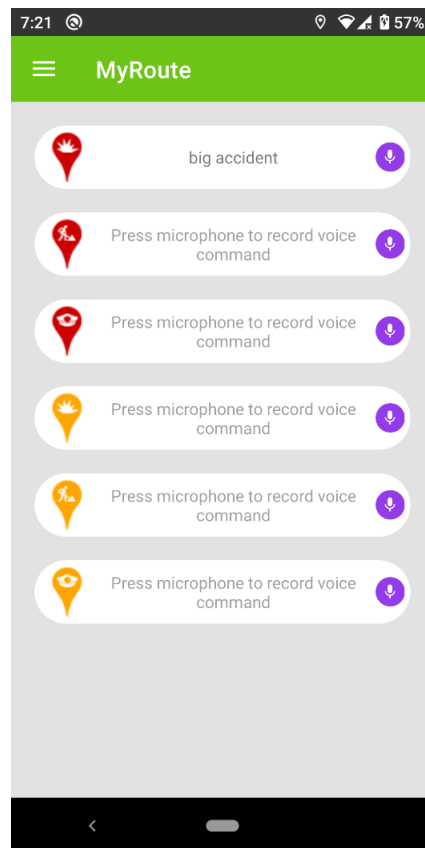


Slika 4.41. Prikaz optimalne saobraćajne rute do odabrane parking zone

## 4.5. Glasovne komande

Fragment za glasovne komande je osmišljen sa ciljem da pruži korisnicima Android mobilne aplikacije mogućnost korišćena glasovnih komandi za postavljanje markera koji predstavljaju nezgode, radove ili prisustvo policijskih patrola u saobraćaju. Ove komande korisnici mogu da registruju i sačuvaju u fragmentu glasovnih komandi. Kako bi uspješno registrovali i sačuvali glasovne komande koristimo ugrađenu Android funkciju prepoznavanja govora. Uz pomoć ove funkcije preuzimamo govor korisnika i pretvaramo ga u tekstualni podatak. Ova funkcija se naziva *startActivityForResult()* i koristi *ACTION\_RECOGNIZE\_SPEECH* radnju (engl. action). Ovaj pristup pokreće aktivnost prepoznavanja govora, a potom rezultate obrađujemo u *onActivityResult()*.

Korisnici mobilne aplikacije na interfesju glasovnih komandi mogu da prepoznaju ikonice koje se koriste na mapi, kao i dugme za registraciju i čuvanje glasovnih komandi koje se nalaze na desnoj strani svake ikonice (Slika 4.42.).



Slika 4.42. Interfejs fragmenta za glasovne komande

U XML fajlu ovog fragmenta definišemo linearni prikaz sa vertikalnom orijentacijom koji će sadržati sve ikonice koje se postavljaju na mapi, prikaz glasovne komande u tekstualnom obliku i dugme za registraciju i čuvanje glasovne komande. U ovom linearnom prikazu definišemo onoliko linearnih prikaza koliko ima ikonica, ali u horizontalnoj orijentaciji. Ovi prikazi sadrže dva elementa za slike i jedan elemenat za prikaz teksta, odnosno glasovne komande. Programski kod ovog interfejsa je prikazan na sljedećoj slici (Slika 4.43.):

```
<LinearLayout
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:orientation="vertical"
  android:padding="@dimen/global_padding_20">
  <LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_marginBottom="@dimen/global_padding_20"
    android:weightSum="3"
    android:background="@drawable/rectangle"
    android:padding="@dimen/global_padding_5">
    <ImageView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:src="@drawable/marker_red_accident"
      android:layout_weight="0"/>
    <TextView
      android:id="@+id/fragment_voice_commands_red_accident_out"
      android:layout_width="wrap_content"
      android:layout_height="match_parent"
      android:hint="@string/press_mic"
      android:gravity="center"
      android:layout_weight="3"/>
    <ImageView
      android:id="@+id/fragment_voice_commands_red_accident_in"
      android:layout_width="25dp"
      android:layout_height="25dp"
      android:src="@drawable/ic_mic_on"
      android:background="@drawable/circle_purple"
      android:layout_weight="0"
      android:layout_gravity="center"
      android:padding="@dimen/global_padding_5"
      android:onClick="onClick"/>
  </LinearLayout>
  ...
</LinearLayout>
```

Slika 4.43. Dio XML programskog koda fragmenata za glasovne komande

U klasi fragmenta za glasovne komande prvo definišemo sve potrebne XML elemente kako bi ih koristili, kao i potrebne promjenljive. Zatim kreiramo funkciju koja će registrovati selektovano dugme i na osnovu toga dodijeliti odgovarajući jedinstveni broj kojim se identifikuje selektovana ikonica, odnosno dugme za registraciju i čuvanje glasovne komande (Slika 4.44.).

```
int n = 0;
switch (view.getId()) {
    case (R.id.fragment_voice_commands_red_accident_in):
        n = 0;
        break;
    case (R.id.fragment_voice_commands_red_works_in):
        n = 1;
        break;
    case (R.id.fragment_voice_commands_red_police_in):
        n = 2;
        break;
    case (R.id.fragment_voice_commands_orange_accident_in):
        n = 3;
        break;
    case (R.id.fragment_voice_commands_orange_works_in):
        n = 4;
        break;
    case (R.id.fragment_voice_commands_orange_police_in):
        n = 5;
        break;
}
```

Slika 4.44. Dio programskog koda funkcije za registraciju selektovanog dugmeta – dodjeljivanje jedinstvenog broja

Sljedeći korak je da kreiramo intent (engl. intent) preko kojeg pokrećemo ugrađenu Android funkciju za prepoznavanje govora. Ovim pristupom kada korisnik odabere ikonicu kojoj želi dodijeliti glasovnu komandu pokreće se funkcija *startActivityForResult()* putem koje se preuzima govor (Slika 4.45.).

```
Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);

intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);

intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, Locale.getDefault());

if (intent.resolveActivity(getActivity().getPackageManager()) != null) {
    startActivityForResult(intent, n);
} else {
```

```

    Toast.makeText(getActivity(), "Your device does not support speech input.",
    Toast.LENGTH_SHORT).show();
}

```

Slika 4.45. Dio programskog koda funkcije za registraciju selektovanog dugmeta – kreiranje i pokretanje funkcije za preuzimanje govora

Potom se u funkciji *onActivityResult()* preuzeti podaci obrađuju, odnosno provjerava se koje je dugme selektovano preko jedinstvenog broja koji smo dodijelili pri selekciji. Zatim preuzima govor u tekstualnom obliku i preko dvije funkcije se čuva i prikazuje u fragmentu za glasovne komande (Slika 4.46.). Ove dvije funkcije obezbjeđuju da korisnici mobilne aplikacije imaju jasan pregled sačuvanih glasovnih komandi koji se prikazuju na interfejsu u tekstualnom obliku iz sačuvanih fajlova koji ostaju u mobilnom uređaju. U slučaju da korisnik želi da promijeni glasovnu komandu ponavlja ovaj proces pri čemu se nova glasovna komanda čuva umjesto već postojeće i time briše prethodnu.

```

ArrayList<String> result;
switch (requestCode) {
    case 0:
        result = data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
        saveText(result.get(0), 0);
        showText(0);
        break;
    case 1:
        result = data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
        saveText(result.get(0), 1);
        showText(1);
        break;
    case 2:
        result = data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
        saveText(result.get(0), 2);
        showText(2);
        break;
    case 3:
        result = data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
        saveText(result.get(0), 3);
        showText(3);
        break;
    case 4:
        result = data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
        saveText(result.get(0), 4);
        showText(4);
        break;
    case 5:

```

```

result = data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
saveText(result.get(0), 5);
showText(5);
break;
}

```

Slika 4.46. Sadržaj funkcije *onActivityResult()*

Ove glasovne komande se zatim koriste u fragmentu glavne mape gdje korisnici mobilne aplikacije imaju plutajuće dugme za pokretanje glasovnih komandi (Slika 4.17.). Kada korisnik selektuje ovo dugme pokreće sličan proces za preuzimanje govora i pretvaranje istog u tekstualni oblik. Ovaj tekstualni podatak se zatim koristi u funkciji koja služi za upoređivanje istog sa postojećim glasovnim komandama (Slika 4.47.).

```

String file_name = null;
for (int i = 0; i < 6; i++) {
    switch (i) {
        case 0:
            file_name = "voice_red_accident";
            break;
        case 1:
            file_name = "voice_red_works";
            break;
        case 2:
            file_name = "voice_red_police";
            break;
        case 3:
            file_name = "voice_orange_accident";
            break;
        case 4:
            file_name = "voice_orange_works";
            break;
        case 5:
            file_name = "voice_orange_police";
            break;
    }
    ...
}

```

Slika 4.47. Dio programskog koda za upoređivanje glasovnih komandi sa novo kreiranim tekstualnim podatkom

Kada se ustanovi koja je glasovna komanda zadata onda se na mapi postavlja odgovarajući marker koji predstavlja nezgodu, radove ili prisustvo policijskih patrola u saobraćaju. U suprotnom korisnik dobija poruku da nema glasovne komande koju je zadao.

Korisnik u ovom slučaju može da provjeri listu glasovnih komandi i ustanovi koju glasovnu komandu da koristi ili da registruje novu. Ovaj pristup omogućava korisnicima da unose dešavanja u saobraćaju sa minimalnim ručnim korišćenjem mobilnog uređaja.



## 5. Rezultati i diskusije

U ovom poglavlju razmatramo rezultate postignute kroz simulaciju, kao i rezultate postignute od strane anonimne grupe koja je koristila predloženo rješenje. Kako rješenje predstavljeno u ovom istraživanju ima za cilj da pruži korisnicima benefite u saobraćaju, kao što je pronalaženje optimalne saobraćajne rute i prikaz informacija o stanju u saobraćaju pomoću grafičkih prikaza (markera), takođe ima za posljedicu mogućnost da korisnicima umanja vrijeme vožnje motornih vozila i time umanja emitovanje štetnih čestica/gasova. Rezultati koji su postignuti u simulaciji i anonimnoj grupi ukazuju na korisne benefite razvijenog rješenja. Benefiti u oba slučaja pokazuju da su korisnici pomoću podataka u saobraćaju koji su predstavljeni na mobilnoj aplikaciji mogli da izbjegnu prepreke koje bi u uspurotnom rezultirale dužom upotrebom vozila i tako negativno uticalo na kvalitet vazduha, a samim tim i zdravlje ljudi.

Prikupljeni podaci simulacije koji uključuje 100 zahtjeva za pronalaženje optimalne saobraćajne rute ukazuju da korisnici mogu doći do informacija brzo i efikasno, kao i da pronađu optimalnu saobraćajnu rutu do željene destinacije. Kroz simulaciju korisnici unose željenu destinaciju i predloženi algoritam pronalazi optimalnu saobraćajnu rutu pri čemu izbjegava sve poznate prepreke u saobraćaju. Broj prepreka je varirao od 3 do 10 između trenutne lokacije mobilnog uređaja i željene destinacije, odnosno pri pronalaženju optimalne saobraćajne rutre. Algoritam koristi informacije koje su unešene u bazi podataka sa pretpostavkom da pružaju najbliži vid realnih situacija i kao rezultat algoritam predlaže optimalnu ili alternativnu saobraćajnu rutu. Ukupan broj unešenih informacija u simulaciji koje predstavljaju prepreku u saobraćaju je 80 i one uključuju nezgode, radove ili policijske patrole. Simulacija je ustanovila da korisnici ovog rješenja postižu željene ciljeve, odnosno efikasnost u saobraćaju. Efikasnost u saobraćaju se odnosi na umanjeno vrijeme potrebno za korisnika da putuje do željene destinacije motornim vozilom pri čemu dolazi i do manje potrošnje goriva kao i emitovanje štetnih čestica/gasova.

Rezultati anonimne grupe koja je koristila predloženo rješenje u periodu od 30 dana je ukazalo da korisnici dolaze do željenih informacija brzo i efikasno. Potrebno je naglasiti da je anonimnu grupu činjelo 10 korisnika i da su svi koristili veb platformu i Android mobilnu

aplikaciju. Prikupljeni podaci ove grupe ukazuju na benefite korišćenja ovog rješenja, kao i na pojedine segmente koji su korišćenji u većoj ili manjoj mjeri.

Tabela 5.1. Rezultati prikupljeni od strane anonimne grupe

	Mapa (od lokacije mobilnog uređaja do unešene destinacije korisnika)	Planiranje ruta	Parking zone	Glasovne komande	Postavljanje markera
1 dan	36	14	18	40	7
2 dan	42	12	12	37	12
3 dan	39	4	8	26	8
4 dan	26	0	12	12	3
5 dan	27	0	13	3	0
6 dan	22	2	16	0	0
7 dan	26	0	17	0	0
8 dan	29	6	12	2	0
9 dan	31	8	9	0	1
10 dan	27	2	6	0	0
11 dan	21	0	10	0	0
12 dan	24	0	11	0	4
13 dan	23	0	10	0	0
14 dan	22	0	9	0	0
15 dan	27	3	6	1	0
16 dan	31	5	8	0	0
17 dan	28	1	7	0	3
18 dan	25	0	12	0	0
19 dan	23	0	11	0	0
20 dan	22	0	15	0	2
21 dan	25	1	12	0	0
22 dan	31	4	10	0	0
23 dan	28	2	9	0	0
24 dan	27	0	8	0	1

25 dan	22	0	16	0	0
26 dan	23	0	12	0	0
27 dan	27	1	14	0	0
28 dan	23	0	13	0	0
29 dan	29	2	9	0	1
30 dan	35	1	12	0	3

Kako je srž ovog rješenja pronalaženje optimalne saobraćajne rute pri čemu predloženi algoritam uzima u obzir prikupljene podatke od korisnika, korisnici anonimne grupe su bili zadovoljni sa rezultatima. Glavni mehanizam pronalaženja optimalne saobraćajne rute od lokacije uređaja korisnika do željene destinacije je najviše korišten i samim time je postignut željeni cilj, odnosno efikasnost u saobraćaju. Svi korisnici anonimne grupe su koristili optimalne saobraćajne rute, kao i alternativne u nekoliko navrata. Drugi mehanizam koji je najviše korišten je pronalaženje parking zona, kao i pronalaženje optimalne saobraćajne rute do istih. Treći mehanizam koji je najviše korišćen je mehanizam glasovnih komandi, gdje su korisnici tokom vožnje mogli da glasovnom komandom označe prepreku u saobraćaju na mapi. Zanimljiv dio rezultata je sistem nagrađivanja, korisnici anonimne grupe su bili zainteresovani da prikupe poene kako bi dobili određene profilne slike. Ovaj mehanizam nagrađivanja se pokazao kao motivator za korisnike da dijele informacije u saobraćaju i to preko glasovnih komandi što ih ne ometa tokom vožnje. Mehanizam planiranje optimalnih saobraćajnih ruta između više lokacija je manje korišteno nego što se očekivalo. Korisnici anonimne grupe, koju su sačinjavali deset korisnika i koji su iz Podgorice, nijesu imali potrebu da koriste mehanizam planiranja optimalnih ruta između više lokacija.

Navedeni rezultati predloženog rješenja imaju za cilj da pruže korisnicima platformu pomoću koje će postignuti benefite u saobraćaju kao što je umanjeno potrebno vrijeme vožnje motornih vozila, a samim time i troškove goriva. Kao krajnji rezultat dolazi do smanjenja emitovanih štetnih čestica/gasova koji nastaju pri korištenju motornih vozila.

## 6. Zaključak

Ovo istraživanje ima za cilj da korišćenjem veb platforme i Android mobilne aplikacije pruži inovativno rješenje koje će korisnicima omogućiti postizanje efikasnog saobraćaja. Efikasnost u saobraćaju se odnosi na umanjeno vrijeme potrebno za korisnika aplikacije da putuje do željene destinacije motornim vozilom pri čemu dolazi i do manje potrošnje goriva kao i emitovanje štetnih čestica/gasova što za posljedicu ima pozitivan uticaj na životnu sredinu, odnosno na smanjenje zagađivanja vazduha.

Implementacijom predloženog rješenja za pronalaženje optimalne saobraćajne rute pruža se korisnicima mogućnost za lakšu navigaciju kroz saobraćaj, a posebno turistima. Benefite u saobraćaju koje korisnici mogu da očekuju je pronalaženje optimalne saobraćajne rute sa najmanjim vremenskim intervalom i/ili najkraćim rastojanjem između dvije ili više lokacija, a samim time korisnicima je trošak goriva umanjen.

Ovo rješenje pruža veb platformu i Android mobilnu aplikaciju.

Veb platforma je informativnog karaktera i sadrži mapu sa markerima koji predstavljaju određene informacije u saobraćaju. Takođe sadrži markere i za parking zone pomoću kojih se korisnici mogu informisati koliko ima slobodnih mjesta, kao i kapacitet iste. Ovaj segment rješenja ima potencijala da se nadogradi sa elementima korisničkog profila i kao takav da koristi funkcije slične kao na mobilnoj aplikaciji koje bi bile razvijane za veb platformu.

Android mobilna aplikacija omogućava korisnicima brojne funkcionalnosti. Pored glavne funkcije pronalaženja optimalne saobraćajne rute pruža postavljanje markera na mapi koji predstavljaju nezgode, radove ili policijske patrole, glasovne komande, parking zone, planiranje ruta između više lokacija i sistem nagrađivanja.

Funkcija glasovnih komandi se pokazala izuzetno korisnom iz razloga što u toku vožnje korisnici aplikacije mogu sa predefinisanim komandama postaviti odgovarajući marker na mapi. U ovom dijelu se može nadograditi segment glasovnih upozorenja. Korisnicima aplikacije, dok prate optimalnu saobraćaju rutu koju su dobili kao rezultat algoritma, glasovno upozorenje bi se aktiviralo kada se približe jednom od markera na mapi. Na ovaj način korisnici bi tokom vožnje bili upozoreni na dešavanja duž dobijene optimalne saobraćajne rute.

Parking zone pružaju korisnicima informacije o parking mogućnostima na definisanim parking zonama na mapi. Ovaj segment je korisnicima od značaja iz razloga što pored pronalaženja optimalne saobraćajne rute dobro je i planirati mjesto za parkiranje motornog vozila.

Planiranje ruta između više lokacija je poseban segment ovog rješenja koji pruža korisnicima optimalnu saobraćajnu rutu i do 5 (pet) lokacija. Posebno turistima je od značaja jer unaprijed mogu da planiraju dnevne aktivnosti, odnosno 5 (pet) željenih destinacija nakon čega mogu da ponove proces. Ovaj segment aplikacije se može proširiti i to na više lokacija.

Sistem nagrađivanja je interesantan segment ovog rješenja i korisnici bi mogli da prikupljaju poene za određene aktivnosti tokom korišćenja mobilne aplikacije. Kroz analizu postignutih rezultata simulacije, a posebno anonimne grupe, ovaj segment rješenja se pokazao kao motivacioni pokretač za korisnike anonimne grupe iz razloga što su prikupljali poene kako bi dobili određene profilne slike. Ovaj segment rješenja se može nadograditi sa raznim ponudama i popustima gdje bi korisnici koji dijele mjerodavne informacije o saobraćaju mogli da koriste poene u korisne svrhe.

Predloženo rješenje sa funkcionalnostima veb platforme i Android mobilne aplikacije je temelj za dalje istraživanje, nadogradnju i razvoj, zaključno sa širom primjenom istog.

## 7. Literatura

- [1] Pușcașiu, A. Fanca, H. Válean, “Tracking and Localization System using Android Mobile Phones”, Automation, Quality and Testing, Robotics (AQTR), 2016 IEEE International Conference on 19-21 May 2016.
- [2] N. Ahmadullah, S. Islam, T. Ahmed, „RouteFinder: Real-time Optimum Vehicle Routing using Mobile Phone Network”, TENCON 2015 - 2015 IEEE Region 10 Conference, 1-4 Nov. 2015.
- [3] A. Pinandito, M.C. Saputra, R.S. Perdana, „RouteBoxer Library for Google Android Platform”, Wireless and Mobile (APWiMob), 2016 IEEE Asia Pacific Conference on 13-15 Sept. 2016.
- [4] C.R. García, S. Candela, J. Ginory, A. Quesada-Arencibia, F. Alayón, „On Route Travel Assistant for Public Transport based on Android Technology”, Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on 4-6 July 2012.
- [5] Ing-Chau Chang, Hung-Ta Tai, Feng-Han Yeh, Dung-Lin Hsieh, and Siao-Hui Chang, „A VANET-Based A\* Route Planning Algorithm for Travelling Time- and Energy-Efficient GPS Navigation App“, Department of Computer Science and Information Engineering, National Changhua University of Education, Changhua 500, Taiwan, Article first published online July 18, 2013, issue published July 1, 2013.
- [6] Itsik Hefez, Yaron Kanza, Roy Levin, „TARSIUS: A System for Traffic-Aware Route Search under Conditions of Uncertainty“, GIS '11 Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Chicago, Illinois – November 01 - 04, 2011
- [7] Jeffrey JOSEPH, Roshan GAJANAN PATIL, Skanda Kumar KAIPU NARAHARI, Yogish DIDAGI, Jyotsna BAPAT, „Wireless Sensor Network Based Smart Parking System”, International Institute Of Information Technology - Bangalore, 26/c Electronics City Hosur Road, Bangalore – 560100, India, Received: 21 September 2013, Accepted: 22 November 2013, Published: 31 January 2014
- [8] Adil Hilmani, Abderrahim Maizate, Larbi Hassouni, „Designing and Managing a Smart Parking System Using Wireless Sensor Networks“, RITM-ESTC/CED-ENSEM, University

Hassan II, Km7, El jadida Street, B.P. 8012, Oasis, Casablanca 8118, Morocco, Received: 10 April 2018, Accepted: 31 May 2018, Published: 6 June 2018

[9] Omyonga Kevin, Kasamani Bernard Shibwabo, „The Application of Real-Time Voice Recognition to Control Critical Mobile Device Operations“, 1Faculty of Information Technology, Strathmore University, Nairobi, Kenya, International Journal of Research Studies in Science, Engineering and Technology Volume 2, Issue 7, July 2015

[10] Kristin Siu, Mark O. Riedl, „Reward Systems in Human Computation Games“, CHI PLAY '16 Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play, October 16 - 19, 2016

[11] Muhammad Saqib Jamil, Muhammad Atif Jamil, Anam Mazhar, Ahsan Ikram, Abdullah Ahmed, Usman Munawar, „Smart Environment Monitoring System by employing Wireless Sensor Networks on Vehicles For Pollution Free Smart Cities“, Humanitarian Technology: Science, Systems and Global Impact 2015, HumTech2015

[12] Anthony DeBarros, „Practical SQL, 2nd Edition: A Beginner's Guide to Storytelling with Data“, Januar 25, 2022

[13] Alan Beaulieu, “Learning SQL: Generate, Manipulate, and Retrieve Data”, May 17, 2006

[14] Jon Duckett, “HTML & CSS: Design and Build Web Sites”, October 25, 2011

[15] David Flanagan, “JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language” (7th edition), June 23, 2020

[16] Jon Duckett, “PHP & MySQL: Server-side Web Development”, January 12, 2022

[17] Alan Mycroft & Mario Fusco “Java 8 in Action: Lambdas, Streams, and Functional-style Programming”, August 28, 2014

[18] Herbert Schildt, “Java: A Beginner's Guide, Ninth Edition”, January 7, 2022