ANDROID APLIKACIJA ZA NAPREDNO PRAĆENJE I NADZOR KRETANJA OSOBA

Leskovar, Ivan

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

ANDROID APLIKACIJA ZA NAPREDNO PRAĆENJE I NADZOR KRETANJA OSOBA

Ivan Leskovar

Zagreb, veljača 2020.

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada".

U Zagrebu, 18.02.2020.

Predgovor

Zahvaljujem svome mentoru Aleksandru Radovanu, dipl. ing., na ukazanom povjerenju i slobodi koju mi je dao prilikom pisanja ovog rada.

Hvala zaručnici Valentini i cijeloj obitelji na pruženoj podršci tijekom studija.

Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi

Sažetak

Rad opisuje postupak izrade programskog rješenja za napredno lokacijsko praćenje i nadzor osoba primjenom programskog inženjerstva. Prije izrade praktičnog dijela rada definirani su funkcionalni i nefunkcionalni zahtjevi, model podataka i arhitektura rješenja. U praktičnom dijelu rada razvijeno je programsko rješenje uz prikaz i objašnjenje ključnih dijelova koda. Programsko rješenje sastoji se od dvije aplikacije, središnjeg poslužitelja i Android klijentske aplikacije. U nastavku rada navedene su tehnologije, biblioteke i programski alati koji su korišteni prilikom izrade praktičnog dijela te je opisan način pokretanja i korištenja aplikacije uz priložene slike grafičkog sučelja.

Programsko rješenje omogućava korisnicima Android mobilnih uređaja međusobno dijeljenje lokacija te lakše koordiniranje prilikom odlaska na zajednički put ili dogovaranja sastanka na određenom mjestu. Spomenuto programsko rješenje nudi mogućnost dijeljenja lokacijskih podataka s ostalim članovima grupe, pohrane i pristupa povijesnim lokacijskim podacima; time je korisniku omogućena i fleksibilnost korištenja aplikacije te analiza vlastitih kretanja u svrhu uštede vremena.

Ključne riječi: programsko rješenje, lokacijsko praćenje, Android klijentska aplikacija, nadzor

Summary

This final thesis describes developing process of a software solution for advanced location tracking and supervision of people using software engineering. Before practical solution was developed functional and non-functional requirements, data model and solution architecture were defined. In the practical part of this thesis, a software solution was developed along with its key parts which are shown and explained. Developed software solution consists of two separate applications; a central server and Android client application. The technologies, libraries, and software tools that were used to create the practical part of the thesis are mentioned and described along with launch instructions and use of the application supported with the graphical interface images.

Developed software solution allows Android mobile users location sharing and facilitates coordination when going on a joint trip or simply arranging a meeting at a specific location. It also offers the ability of sharing location data with other members of the group, storage and easy access to location dana historically; this way user can also utilize the application in order to analyse his movements and potentially save time.

Keywords: software solution, location tracking, Android client application, surveillance

Sadržaj

1. Uvod

Tijekom obavljanja dnevnih rituala, kao što je odlazak na posao, vožnja djece u dječji vrtić ili školu, sastanaka na drugom kraju grada ili odlaska na fakultet, puno vremena izgubi se u prometu. Ako se tome pridoda i vrijeme utrošeno na čekanje suputnika ili osobe s kojom je dogovoren sastanak, izgubi se velika količina vremena tijekom dana.

Cilj kojim se vodilo u ovom radu jest upravo smanjenje utrošenog vremena kako bi se došlo od točke A do točke B. Iz tog razloga, izrađen je prototip programskog rješenja koje će korisnicima olakšati geolokacijsko praćenje ostalih članova unutar zajedničke grupe. Cilj ovog programskog rješenja je implementirati sve funkcionalnosti koje su dovoljne za koordinaciju uobičajenih dnevnih aktivnosti, npr. roditelju će omogućiti praćenje djeteta do škole i dobivanje obavijesti o dolasku na odredište, kolegama koji dijele prijevoz do radnog mjesta pružati će informaciju o trenutnoj lokaciji osobe, a time i bolje usklađivanje i smanjenje vremena potrebnog do odlaska na posao. Povijesni podaci će korisniku omogućiti statistiku posjeta lokacijama, vremena potrebnog da s jedne lokacije dođe na drugu, a u svrhu što boljeg uvida u vlastita ili kretanja korisnika koji je s nama podijelio informacije u nekom vremenskom periodu.

Rješenje se sastoji od aplikacije na mobilnom uređaju, obzirom da je u današnje vrijeme, gotova svaka osoba vlasnik pametnog telefona, te aplikacije na centralnom poslužitelju čiji je zadatak omogućiti komunikaciju između mobilnih uređaja i pohrane povijesnih podataka.

U ovom radu definirani su korisnički zahtjevi i implementirane funkcionalnosti, opisana je arhitektura poslužitelja i Android korisničke aplikacije te je opisano određivanje geolokacije uređaja i predviđanje vremena dolaska na određenu lokaciju.

U izradi programskog rješenja korištena je baza podataka koja je detaljno opisana u četvrtom poglavlju. Slikom i opisom je predstavljen ER model te je pojedinačno opisan svaki entitet modela.

U poglavlju „Ključni dijelovi aplikacije“ prikazan je programski kod dok se u posljednjem dijelu rada nalaze upute za korištenje aplikacije. Radi lakšeg prikaza, korištene su slike koje prikazuju gdje se koja opcija u aplikaciji nalazi i na koji način se ista koristi.

2. Korisnički zahtjevi i implementirane funkcionalnosti aplikacije

U ovom poglavlju opisani su korisnici sustava izgrađenog u praktičnom dijelu rada. Opisano je što bi sustav trebao raditi u obliku funkcionalnih zahtjeva te su popisani nefunkcionalni zahtjevi, odnosno opis samog sustava.

2.1. Korisnici aplikacije

Aplikacija predviđa dva tipa korisnika:

- prijavljeni korisnik
- administrator grupe

Osnovna razlika u tipu korisnika odnosi se na prava koja korisniku omogućuju uređivanje grupe u kojoj je član. Sukladno tome, administrator grupe ima veća prava od korisnika. Više o tome bit će opisano u poglavlju koje slijedi.

2.2. Korisnički zahtjevi

Korisnički zahtjevi su kratke izjave koje opisuju što aplikacija mora raditi, odnosno opisuju funkcionalnosti ili značajke aplikacije. Ovi zahtjevi opisuju samo vanjsko ponašanje sustava, tj. ne bi trebali opisivati način implementacije i dizajn sustava (Krajcar, Belani, 2011).

Korisnički zahtjevi razlikuju se prema tipu korisnika:

- Prijavljeni korisnik
  - korisnik, prilikom prvog susreta s aplikacijom, ima mogućnost kreiranja korisničkog računa
  - za vrijeme korištenja aplikacije, korisnik mora biti prijavljen u aplikaciju
  - mogućnost kreiranja korisničkih grupa
  - pregled svih grupa u kojima je član
  - kreiranje zajedničkog puta ili točke sastanka unutar neke grupe
  - brisanje zajedničkog puta ili točke sastanka koje je korisnik sam kreirao
  - odabir kada se lokacijski podaci dijele

  - dijeljenje trenutne lokacije i posjećenih adresa
  - pohrana povijesnih podataka
  - pregled povijesnih podataka
  - uređivanje osobnih postavki
- Administrator grupe
  - uređivanje postavki grupe
  - dodavanje novih članova u grupu
  - micanje članova iz grupe
  - brisanje grupe
  - uređivanje i brisanje zajedničkog puta ili točke sastanka koje su ostali članovi grupe kreirali

Administrator grupe, uz sve svoje funkcionalnosti, poprima i funkcionalnosti prijavljenog korisnika.

2.3. Funkcionalni zahtjevi

Funkcionalni zahtjevi opisuju koje mogućnosti aplikacija mora nuditi, opisuju način na koji radi određeni dio aplikacije te kako aplikacija treba reagirati ovisno o tome koju je akciju korisnik napravio. Ovi zahtjevi mogu opisivati i kako se aplikacija ne treba ponašati (Krajcar, Belani, 2011).

U nastavku su opisani ključni funkcionalni zahtjevi implementirani u praktičnom dijelu rada:

- upravljanje podacima i bazom podataka
  - spremanje, pregledavanje, uređivanje i brisanje podataka o korisnicima, geolokacijskim podacima, korisničkim grupama, aktivnostima u korisničkim grupama
- prilikom registracije korisnika, potrebno je navesti ispravnu e-mail adresu, jedinstveno korisničko ime, ime korisnika te lozinku minimalne duljine od 6 znakova; slika profila se ne mora navesti
- kreiranje korisničke grupe zahtijeva unos naslova grupe te je potrebno pridružiti barem jednog korisnika
- brisanje grupe ne uvjetuje brisanje povijesnih geolokacijskih podataka
- brisanje korisnika podrazumijeva i brisanje povijesnih geolokacijskih podataka

- brisanje korisničke grupe podrazumijeva i brisanje svih aktivnosti vezanih uz tu grupu
- korisnik može napustiti određenu grupu
- prikupljanje geolokacijskih podataka minimalno podrazumijeva pohranjivanje podatka o geolokacijskoj koordinati, vremenu i podatak o korisniku, a ukoliko su dostupne pohranjuju se i informacije o adresi
- kod kreiranja sastanka potrebno je navesti:
  - naziv sastanka
  - datum i vrijeme
  - lokaciju odredišta (odabir na karti)
- kod kreiranja zajedničkog puta potrebno je navesti:
  - naziv
  - datum i vrijeme
  - lokaciju odredišta (odabir na karti)
  - početnu točku (korisnik)
- navođenje međustanica na zajedničkom putu je proizvoljno
- korisnik može birati kada će svoju lokaciju dijeliti, a kada ne

2.4. Nefunkcionalni zahtjevi

Za razliku od funkcionalnih zahtjeva, koji opisuju način na koji radi određeni dio aplikacije, zahtjevi vezani uz značajke aplikacije odnose se na sustav u cjelini. S obzirom da se odnose na cijelu aplikaciju, puno su kritičniji od funkcionalnih zahtjeva određenog modula. Ako pojedini dio aplikacije nije izveden, nesumnjivo će funkcionalnost aplikacije biti smanjena. No, ako neki od nefunkcionalnih zahtjeva nije zadovoljen, moglo bi se dogoditi da korištenje cijele aplikacije bude dovedeno u pitanje (Krajcar, Belani, 2011).

U nastavku su opisani ključni nefunkcionalni zahtjevi implementirani u praktičnom dijelu rada:

- Android mobilna aplikacija podrazumijeva stalnu dostupnost internetske veze kako bi mogla u realnom vremenu komunicirati sa serverom te prikazivati sve relevantne geolokacijske podatke o članovima grupe

- u slučaju nedostupnosti internet veze, prikupljeni podaci o trenutnoj lokaciji i posjećenim adresama se pohranjuju u lokalnu bazu podatka te će biti poslani na poslužitelj nakon što internet veza ponovno postane dostupna
- veći broj korisnika može istovremeno koristiti aplikaciju te raditi izmjene
- poslužitelj ne smije imati dužih zastoja u radu u jednom danu
- programsko rješenje mora biti napisano u Java programskom jeziku – i poslužitelj i Android mobilna aplikacija
- za komunikaciju između poslužitelja i klijenta mora se koristiti HTTPS protokol
- aplikacija se može koristiti u pozadini

3. Arhitektura i tehnologije

Prilikom izrade praktičnog dijela rada korišten je programski jezik Java. Sustav je podijeljen na dva dijela: poslužitelj i Android klijentska aplikacija. U izradi poslužitelja, uz programski jezik Java korištene su biblioteke otvorenog koda kako bi se ubrzao proces razvoja rješenja.

U sljedećim poglavljima detaljnije je opisana arhitektura poslužitelja i Android klijentske aplikacije te način na koji je ostvareno određivanje lokacije uređaja.

3.1. Arhitektura rješenja

Rješenje se sastoji od centralnog poslužitelja i klijentskih Android mobilnih aplikacija. Poslužitelj omogućuje pohranu i razmjenu podataka između klijentskih aplikacija. Komunikacija između njih odvija se putem nekoliko različitih web servisa, a podaci se prenose u JSON formatu podataka. Autorizacija i autentifikacija između klijenta i poslužitelja odvija se razmjenom JWT tokena koji sadrži i detaljne informacije o korisniku.

3.1.1. Android klijentska aplikacija

Android klijentska aplikacija napisana je u programskom jeziku Java. Minimalna verzija Android platforme je API verzija 26, odnosno Android 8 Oreo.

Aplikacija zahtijeva dozvolu za:

- pristup informacijama o lokaciji:
  - android.permission.ACCESS_FINE_LOCATION
  - android.permission.ACCESS_COARSE_LOCATION
- pristup internetu i dostupnosti mreže:
  - android.permission.INTERNET
  - android.permission.ACCESS_NETWORK_STATE
- rad aplikacije u pozadini:
  - android.permission.FOREGROUND_SERVICE

Za pohranu podataka se koristi SQLite relacijska baza podataka. Programski kod je organiziran u podatkovni sloj, sloj poslovne logike i prezentacijski sloj. Komunikacija

između poslužitelja i aplikacije odvija se na zasebnoj dretvi asinkronim načinom, kako se ne bi utjecalo na performanse korisničkog sučelja.

Aplikacija je razvijana u programskom okruženju „Android Studio“, a pokretanje i izvršavanje aplikacije testirano je u virtualnom uređaju koristeći Android Emulator te na osobnom Android mobilnom uređaju. Korištenje Android emulatora omogućuje veću kontrolu nad uređajem i interakciju sa senzorima (Horton, 2018). Mogućnost interakcije s GPS senzorom se pokazala vrlo korisnom za testiranje Android klijentske aplikacije.

3.1.2. Poslužitelj

Poslužiteljska aplikacija je napisana u programskom jeziku Java koristeći Spring Boot framework. Radi lakše obrade i pristupa podacima korišten je Hibernate ORM, a za samu pohranu podataka korištena je MySQL relacijska baza podataka verzije 8. Poslužiteljska aplikacija nema grafičko sučelje te se aplikacija koristi isključivo kroz REST web servise. U svrhu autentifikacije i autorizacije razmjenjuje se JWT token, koji osim podataka potrebnih za potvrdu identiteta sadrži i dodatne informacije o korisniku. Prilikom slanja zahtjeva prema poslužitelju, klijentske aplikacije moraju u zaglavlju HTTPS poruke navesti i valjani token. Iznimno, token nije potrebno navoditi kod pristupa web servisu za prijavu i registraciju korisnika. U praktičnom dijelu ovog rada razvijena je Android klijentska aplikacija koja koristi resurse poslužitelja kroz web servise, no bez dodatnih izmjena na poslužitelju može se razviti aplikacija pisana u nekom drugom programskom jeziku ili aplikacija koja može biti namijenjena nekom drugom mobilnom operacijskom sustavu.

U fazi razvoja i testiranja aplikacije korišten je programski alat IntelliJ IDEA, a za pristup bazi podataka korišten je programski alat DBeaver. Za pokretanje aplikacije korišten je i aplikacijski poslužitelj Apache Tomcat.

3.2. Geolokacija uređaja

Za određivanje lokacije uređaja koristi se funkcionalnost ugrađena u Android operacijski sustav koja omogućava precizno pozicioniranje uređaja u prostoru. Temeljem tih podataka i korištenjem Google lokacijskih servisa, određuju se put i vrijeme potrebno da bi se došlo do nekog odredišta te se ti podaci kasnije prikazuju na karti.

3.2.1. Praćenje kretanja u realnom vremenu

Ukoliko je uključeno dijeljenje geolokacijskih podataka, svaki puta kada Android uređaj zabilježi promjenu lokacije, ta se promjena obradi u aplikaciji. Iz trenutne lokacije uređaja uzima se geolokacijska koordinata te se koristeći Google Geocoding API iz koordinate određuju adrese u blizini te točke. Nakon toga se ti podaci šalju na poslužitelj te pohranjuju. Ti podaci su kasnije dostupni ostalim članovima unutar grupe za prikaz na karti ili određivanje puta i vremena dolaska do neke točke. Ostalim članovima grupe dostupan je samo zadnji podatak o lokaciji korisnika dok je samom korisniku omogućen pregled i brisanje svih podataka.

3.2.2. Predviđanje vremena dolaska na neko odredište

Prilikom određivanja vremena i puta kojim je najbolje ići, koristi se Google Routes API. Nakon što se, koristeći navedeni servis, izračuna put od početne točke do odredišta uz neke međustanice, dodatno se uračunava eventualno kašnjenje osoba koje trebaju stići na međustanicu. Put i vrijeme dolaska do određene međustanice se također izračunava koristeći navedeni servis, a eventualno kašnjenje se uračunava i za sve ostale stanice u nizu te i za samo odredište.

3.3. Zaštita osobnih podataka

Prilikom obrade i prikupljanja osjetljivih osobnih podataka poput imena i prezimena, e-mail adrese, datuma rođenja, podataka o kretanju korisnika i sl. potrebno je voditi računa o zaštiti podataka, prenošenju podataka na siguran način kroz mrežu te je potrebno korisniku omogućiti upravljanje vlastitim podacima.

Prilikom izrade praktičnog dijela rada podaci su spremljeni u bazu podataka koja nudi vlastite mehanizme ograničenja pristupa. Komunikacija između klijentskih aplikacija i poslužitelja ostvarena je kroz HTTPS protokol te su sami podaci dostupni isključivo osobama s kojima ih je korisnik podijelio.

U praktičnom dijelu ovog rada nije implementirano upravljanje korisničkim privolama, no prije omogućavanja javnog korištenja opisanog programskog rješenja potrebno je navedenu funkcionalnost implementirati.

„Privola ispitanika znači svako dobrovoljno, posebno, informirano i nedvosmisleno izražavanje želja ispitanika kojim on izjavom ili jasnom potvrdnom radnjom daje pristanak za obradu osobnih podataka koji se na njega odnose“ (https://www.privacy-regulation.eu/hr/4.htm, veljača 2020).

Traženje privole od korisnika mora biti jasno i nedvosmisleno naznačeno, također povlačenje privole mora biti jednako jednostavno kao i davanje privole (https://www.privacy-regulation.eu/hr/7.htm, veljača 2020).

Prilikom implementacije navedene funkcionalnosti, potrebno je korisniku omogućiti i preuzimanje i brisanje prikupljenih podataka.

4. Model podataka

Kako bi obrada i pregled povijesnih podataka, kao i razmjena podataka između različitih korisnika bila moguća, potrebno je osigurati pohranu podataka. Podaci se pohranjuju u relacijsku bazu podataka te su na taj način i modelirani.

Model podataka je u potpunosti implementiran na poslužitelju te se na njemu ostvaruje pohrana svih podataka. Android klijentska aplikacija tek po potrebi pohranjuje manji dio podataka.

ER model se prikazuje jednim ili više ER dijagrama. ER dijagrami se sastoje od sljedećih elemenata (Đambić, 2009):

- entitet
- skup entiteta
- odnos između entiteta
- atribut entiteta ili odnosa

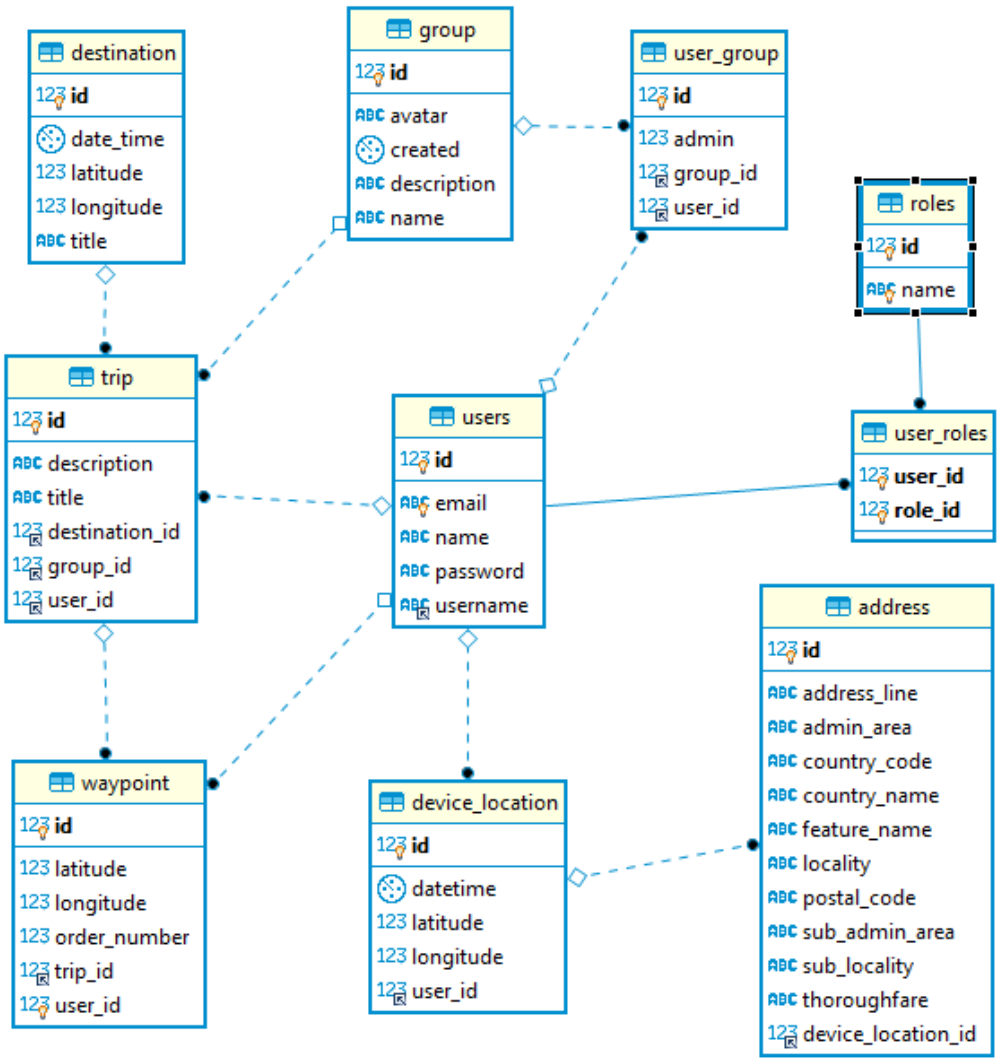
4.1. Poslužitelj

Kao što je navedeno ranije, model podataka je u potpunosti integriran na poslužitelju. Klijentske aplikacije šalju podatke prema poslužitelju koji ih pohranjuje i na zahtjev dostavlja ostalim klijentima.

U nastavku rada slikovno i tekstualno prikazani su model podataka, entiteti te veze između pojedinih entiteta unutar modela podataka.

4.1.1. ER model podataka

Na slici 4.1. prikazan je dijagram poslužiteljskog modela podataka. Navedeni dijagram preuzet je iz programskog alata DBeaver nakon što je baza podataka izgrađena. Slika sadrži sve entitete, tipove podataka od kojih su izgrađeni atributi i veze između samih entiteta.



Slika 4.1. ER model podataka poslužitelja

4.1.2. Opis modela podataka

Entitet *device_location* služi za pohranu povijesnih geolokacijskih podataka. Prilikom pohrane spremaju se podaci o koordinatama u prostoru, vrijeme te podaci o referenci na korisnika za kojeg su podaci pohranjeni. Svaki podatak o lokaciji može imati više podataka o adresi.

Podaci o adresama vezanim uz određenu geolokacijsku točku, pohranjuju se u entitetu *address*. Podatak o adresi ne mora uvijek biti dostupan, dok se za neku drugu lokaciju može imati nekoliko različitih adresa. Za svaku adresu bilježi se podatak o ulici, županiji, oznaci i

nazivu države, mjestu, općini, poštanskom broju te veza na podatak o koordinati uz koju je adresa vezana.

Za svakog korisnika u entitetu *users*, bilježi se puno ime i prezime, e-mail adresa, korisničko ime i lozinka. Uz navedeno, svaki korisnik može imati jednu i više aplikacijskih rola koje se pohranjuju u entitetu *roles* te su na korisnika vezane kroz entitet *user_roles*.

Entitet *groups* može se smatrati glavnim mjestom za razmjenu podataka između različitih članova aplikacije. Svaka grupa sadrži podatke o vremenu kada je kreirana, nazivu, opisu i tekstualnoj oznaci slike profila grupe. Entitet *users* u kojeg se pohranjuju korisnici, povezuje se s pripadajućim grupama kroz entitet *user_group*.

Entitet *trip* služi za pohranu neke točke sastanka ili zajedničkog puta na neku destinaciju. Bilježe se podaci o naslovu, opisu te kroz vezne podatke na entitete *destination*, podatak o krajnjoj točci puta, *user_id* podatak o početnoj osobi koja započinje put, *waypoint* podatak o međustanicama na nekom putu. Podatak o korisniku (*user_id*) može biti prazan.

Destination entitet služi za pohranu podataka o nekoj krajnjoj točki te se pritom bilježi podatak o geolokacijskoj koordinati i nazivu odredišta.

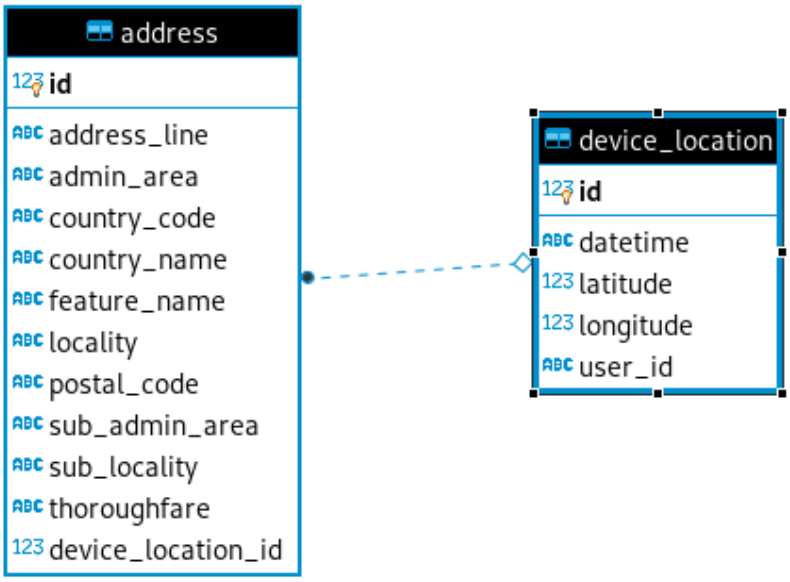
Definirani zajednički putevi mogu imati međustanice definirane entitetom *waypoint* koji je definiran geolokacijskom točkom te korisnikom uz kojeg je ta međustanica vezana.

4.2. Android klijentska aplikacija

Android klijentska aplikacija sadrži mali dio modela podataka zadužen za pohranu povijesnih podataka o lokaciji uređaja i posjećenim adresama. U prikazane entitete pohranjuju se isti podaci kao i kod modela podataka na poslužitelju.

4.2.1. ER model podataka

U nastavku je prikazan dijagram modela podataka koji se nalazi na Android klijentskoj aplikaciji. Sastoji se od dva entiteta identična onima na poslužitelju upravo zbog jednostavnosti razmjene podataka.



Slika 4.2. ER model podataka Android klijentske aplikacije

4.2.2. Opis modela podataka

Entitet *device_location* služi za pohranu povijesnih geolokacijskih podataka u slučaju da nije dostupna internet veza. Prilikom pohrane, spremaju se podaci o koordinatama u prostoru, vrijeme te podaci o referenci na korisnika za kojeg su podaci pohranjeni. Svaki podatak o lokaciji može imati više podataka o adresi.

Podaci o adresama vezanim uz određenu geolokacijsku točku pohranjuju se u entitetu *address*. Podatak o adresi ne mora uvijek biti dostupan, dok za neku drugu lokaciju može imati nekoliko različitih adresa. Za svaku adresu bilježe se podaci o ulici, županiji, oznaci i nazivu države, mjestu, općini, poštanskom broju te veza na podatak o koordinati uz koju je adresa vezana.

4.3. Opis entiteta

Entitet predstavlja apstrakciju elemenata poslovnih procesa koji se modeliraju. Može se reći da je entitet bilo što o čemu se može razmišljati i prikupljati informacije. Svaki entitet se mora na jedinstven način razlikovati od ostalih entiteta iz istog skupa. Iz tog razloga mora

postojati skup atributa čije vrijednosti jedinstveno određuju entitet. Ta svojstva zovu se karakteristična ili ključna svojstva (Đambić, 2009).

U nastavku su detaljno opisani entiteti:

- **Device_locations** - služi za pohranu povijesnih geolokacijskih koordinata
  - id - numerička jedinstvena oznaka lokacije
  - datetime - datumi i vrijeme kada je lokacijski podatak zabilježen
  - latitude - dio geolokacijske koordinate
  - longitude - dio geolokacijske koordinate
  - user_id - numerički podatak o vezi na pripadajućeg korisnika
- **Address** - služi za pohranu povijesnih informacija o adresi
  - id - numerička jedinstvena oznaka adrese
  - address_line - informacije o punoj adresi (grad, ulica, kućni broj, poštanski broj, država)
  - admin_area - naziv županije
  - country_code - oznaka države
  - country_name - naziv države
  - feature_name - kućni broj
  - locality - naziv mjesta
  - postal_code - poštanski broj mjesta
  - sub_admin_area - naziv općine
  - sub_locality - naziv kvarta, odnosno dijela nekog mjesta/grada
  - thoroughfare - naziv ceste
  - device_location_id - podatak o vezi na pripadajuću geolokacijsku koordinatu
- **Users** - služi za pohranu podataka o korisnicima aplikacije
  - id - numerička jedinstvena oznaka korisnika
  - e-mail - e-mail adresa korisnika
  - name - puno ime i prezime korisnika ili nadimak
  - password - kriptirani podatak o lozinki korisnika
  - username - korisničko ime za prijavu u aplikaciju

- **Roles** - služi za pohranu tipova aplikacijskog korisnika
  - id - numerička jedinstvena oznaka
  - name - naziv tipa aplikacijskog korisnika
- **User**_roles - veza između korisnika i tipa aplikacijskog korisnika
  - user_id - oznaka veze na korisnika
  - role_id - oznaka veze na tip aplikacijskog korisnika
- **Group** - služi za pohranu korisničkih grupa
  - id - numerička jedinstvena oznaka grupe
  - avatar - tekstualni zapis naziva slike profila grupe
  - created - datum i vrijeme kada je grupa kreirana
  - description - opis grupe
  - name - naziv grupe
- **User_group** - veza između korisnika i pripadajuće korisničke grupe
  - id - numerička jedinstvena oznaka
  - admin - oznaka je li korisnik u administrator u pripadajućoj grupi
  - user_id - oznaka veze na korisnika
  - group_id - oznaka veze s pripadajućom grupom
- **Trip** - opisuje zajednički korisnički put ili točku sastanka
  - id - numerička jedinstvena oznaka
  - description - opis
  - title - naziv
  - destination_id - oznaka veze na krajnju točku
  - group_id - oznaka veze na korisničku grupu
  - user_id - oznaka veze na korisnika koji pokreće put
- **Waypoint** - međustanice nekog zajedničkog puta
  - id - numerička jedinstvena oznaka
  - latitude - dio geolokacijske koordinate
  - longitude - dio geolokacijske koordinate
  - route_id - oznaka veze na neki put ili točku sastanka

  - user_id - oznaka veze na pripadajućeg korisnika
- **Destination** - opisuje krajnju točku nekog puta ili točku sastanka
  - id - numerička jedinstvena oznaka
  - title - naziv
  - latitude - dio geolokacijske koordinate
  - longitude - dio geolokacijske koordinate

4.4. Baza podataka

U fazi izrade poslužiteljske baze podataka korištena je MySQL relacijska baza podataka otvorenog koda, no kako je u implementaciji korišten i Hibernate ORM uz sitne izmjene u konfiguracijskoj datoteci, moguće je koristiti neku od sljedećih baza podataka:

- Oracle 11g, 11g RAC
- DB2 9.7 ili novija
- Microsoft SQL Server 2008
- Sybase ASE 15.5 (jConnect 6.0)
- MySQL 5.1, 5.5
- PostgreSQL 8.4, 9.1

Uz navedene, moguće je koristiti i još neke dostupne baze podataka koje su podržane u Hibernate ORM (https://docs.jboss.org/#all-updates, 2020).

Android klijentska aplikacija, za privremeno pohranjivanje lokacijskih podataka, koristi SQLite relacijsku bazu podataka, budući da je navedena baza podataka trenutno jedino prirodno podržana u Android operacijskom sustavu. U konfiguracijskoj datoteci potrebno je samo navesti parametre za povezivanje na bazu podataka. Prilikom inicijalnog pokretanja aplikacije, automatski će se kreirati sve potrebne tablice, veze, ključevi i indeksi, odnosno kompletna baza podataka potrebna za rad aplikacije. Kao zadnji korak kreiranja baze podataka, neke tablice će se popuniti s inicijalnim podacima potrebnim za rad sustava.

5. Ključni dijelovi aplikacije

U praktičnom dijelu ovog rada razvijene su dvije aplikacije koje zajedno čine programsko rješenje obrađeno u ovom radu. U dva poglavlja koja slijede, navedeni su ključni dijelovi aplikacija kao i primjeri programskog koda. Dijelovi programskog koda su i pojašnjeni.

5.1. Android klijentska aplikacija

Android programska aplikacija namijenjena je korisniku za direktno korištenje te je samim time, uz poslovnu logiku, razvijeno grafičko sučelje. U ovom poglavlju fokus je na ključnim dijelovima programskog koda koji implementira glavne dijelove poslovne logike dok će grafičko sučelje biti prikazano, u obliku slika ekrana, u šestom poglavlju.

Glavni zadatci Android klijentske aplikacije su:

- što preciznije odrediti trenutnu lokaciju
- lokacijske podatke proslijediti poslužitelju
- izračunati i prikazati informacije o vremenu i planiranom putu do željene lokacije

Određivanje trenutne lokacije je implementirano u klasi *BackgroundLocationService* koja nasljeđuje Android klasu *Service*. Navedeno omogućava određivanje lokacije i onda kad aplikacija nije u fokusu, odnosno u procesu koji se odvija u pozadini. Nakon što se pokrene zahtjev za određivanjem i slanjem lokacije, pokreće se pozadinski proces koji detektira svaki događaj (engl. *Event*) promjene lokacije te se ta promjena obrađuje dalje. Kako bi korisnik bio obaviješten o prikupljanju i dijeljenju lokacije, u notifikacijskom izborniku se prikaže poruka koja ga o tome obavještava. Klasa *BackgroundLocationService* sadrži veću količinu programskog koda, pa su zbog bolje preglednosti, manji dijelovi koda izbačeni.

```
public class BackgroundLocationService extends Service {
    private final LocationServiceBinder binder =
        new LocationServiceBinder();
    private static final String TAG =
        "BackgroundLocationService";
    private LocationListener mLocationListener;
    private LocationManager mLocationManager;
    private NotificationManager notificationManager;
    private final int LOCATION_INTERVAL = 500;
    private final int LOCATION_DISTANCE = 10;
```

```
    @Override
    public IBinder onBind(Intent intent) {
        return binder;
    }

    public class LocationServiceBinder extends Binder {
        public BackgroundLocationService getService() {
            return BackgroundLocationService.this;
        }
    }


    private class LocationListener implements
      android.location.LocationListener {
        private Location lastLocation = null;
        private final String TAG = "LocationListener";
        private Location mLastLocation;

        public LocationListener(String provider) {
            mLastLocation = new Location(provider);
        }

        @Override
        public void onLocationChanged(Location location) {
            mLastLocation = location;
            new LocationUpdater(
                    BackgroundLocationService.this,
                    location)
                    .execute();
        }
        …
    }
    …
}
```

Kôd 5.1. Programski kod pozadinskog servisa za određivanje trenutne lokacije

Klasa se sastoji od nekoliko privatnih atributa, metoda i dvije privatne klase. Klasa *LocationListener* implementira poslovnu logiku zaduženu za praćenje događaja o promjeni lokacije. Nakon što se događaj obradi i dostupna je trenutna lokacija, poziva se asinkroni

proces, zadužen za određivanje adrese i slanje podataka na poslužitelj, te je isti opisan u nastavku.

Za potrebe određivanja trenutne adrese iz prostornih koordinata (obrnuto geokodiranje) te slanje lokacijskih podataka poslužitelju, izrađena je klasa *LocationUpdater*. Definicija klase je prikazana u nastavku.

```
public class LocationUpdater extends AsyncTask {
    private Context context;
    private LocationPayload locationPayload;
    private Location location;
    private static int MAX_RESULTS = 3;

    public LocationUpdater(
            Context context,
            Location location
    ) {
        this.context = context;
        this.locationPayload = new LocationPayload();
        this.location = location;
    }

    @Override
    protected String doInBackground(Object object[]) {
        …
    }
}
```

Kôd 5.2. Definicija klase *LocationUpdater*

Klasa *LocationUpdater* nasljeđuje klasu *AsyncTask* kako bi se što jednostavnije u pozadinskom procesu, bez blokiranja glavne dretve aplikacije, obradilo lokacijske podatke te proslijedilo poslužitelju (https://developer.android.com/reference/android/os/AsyncTask, 2019).

Klasa ima definirana 3 privatna atributa:

- context – kontekst aplikacije
- locationPayload – jednostavna klasa koja opisuje strukturu podataka koji se šalje poslužitelju
- location – klasa koja opisuje trenutnu lokaciju uređaja

Uz atribute, klasa ima i metodu *doInBackground* u koju je smještena poslovna logika, te jednu brojčanu konstantu kojom se određuje maksimalan broj adresa za koje se želi da ih proces obrnutog kodiranja odredi.

Kôd metode *doInBackground* podijeljen je na dva glavna dijela, dio zaslužen za obrnuto geokodiranje i dio zadužen za slanje podataka poslužitelju. U programskom kodu 5.3. prikazana je implementacija logike za određivanje adrese iz trenutnih lokacijskih koordinata (Božić i sur., 2013).

```
locationPayload = new LocationPayload();
locationPayload.setLoacation(this.location);
Geocoder geocoder = new Geocoder(
        this.context,
        Locale.getDefault()
);
List<Address> addresses = new ArrayList<>();

try {
    addresses = geocoder.getFromLocation(
            this.location.getLatitude(),
            this.location.getLongitude(),
            MAX_RESULTS
    );
    for (Address address : addresses) {
        locationPayload.addAddress(new
AddressPayload(address));
    }
} catch (IOException e) {
    Log.d(Constants.DEBUG_TAG, e.getMessage());
    Log.d(Constants.DEBUG_TAG, "error getting addresess");
}
```

Kôd 5.3. Određivanje adrese iz lokacijskih koordinata.

Nakon što su poznati podaci o trenutnoj lokaciji (prostorne koordinate) i adresi, podaci se prosljeđuju poslužitelju koji dobivene podatke pohranjuje u bazu podataka i nakon toga su dostupni ostalim korisnicima na zahtjev.

```
try {
    JsonObjectRequest jsonObjectRequest = new
    JsonObjectRequest(
            Constants.LOCATION_URL,
```

```
                new JSONObject(locationPayload.toString()),
                new Response.Listener<JSONObject>() {
            @Override
            public void onResponse(JSONObject response) {
                Log.d(Constants.DEBUG_TAG, response.toString());
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                Log.d(Constants.DEBUG_TAG, " update error");
                error.printStackTrace();
            }
        }
        ) {
            @Override
            public Map<String, String> getHeaders() {
                HashMap<String, String> headers = new
                HashMap<>();
                headers.put(
                        Constants.AUTH_HEADER,
                        AuthManager
                          .getInstance(context)
                          .getAuthHeader()
                );
                return headers;
            }
        };
        RequestManager
                .getInstance(this.context)
                .addToRequestQueue(jsonObjectRequest);
    } catch (JSONException e) {
        e.printStackTrace();
    }
    return null;
```

Kôd 5.4. Slanje lokacijskih podataka poslužitelju

Prilikom izrade funkcionalnosti za slanje podataka prema poslužitelju, korištena je HTTP biblioteka Volley te je na istoj web stranici proučen način njezinog korištenja (https://developer.android.com/training/volley, 2019).

Poslovna logika za određivanje potrebnog vremena i puta kojim je potrebno ići da bi se stiglo do željenog cilja, implementirana je u klasi *DistanceTimeCalculator*. Podaci o zajedničkom putu i posljednjim lokacijama svih sudionika dobivenih od poslužitelja, predaju se klasi kroz konstruktor.

Budući da klasa sadrži veću količinu programskog koda kojeg nije potrebno izdvajati, u nastavku je prikazan i opisan središnji dio poslovne logike, odnosno metoda kojom se određuju parametri glavnog puta.

```
public void tripCalculator() {
    final DirectionsApiRequest request = DirectionsApi
            .newRequest(this.geoApiContext)
            .mode(TravelMode.DRIVING)
            .origin(this.origin)
            .destination(
                 getTrip().getDestination().getLocation()
            )
            .waypoints(getTrip().getWaypointArray())
            .departureTimeNow();

    try {
        DirectionsRoute[] routes = request.await().routes;
        this.setDirectionsRoute(routes[0]);
        long duration = 0;

        List<Waypoint> waypointList =
            this.trip.getWaypointList();
        int i = 0;
        for (DirectionsLeg leg : routes[0].legs) {
            duration = duration + leg.duration.inSeconds;
            if (i != routes[0].legs.length - 1) {
                waypointList.get(i).setDurationSec(duration);
            } else {
                this.trip
                  .getDestination().setDurationSec(duration);
            }
            i++;
        }
        this.trip.setWaypointList(waypointList);
    } catch (ApiException e) {
        Log.e(Constants.DEBUG_TAG, e.getMessage());
```

```
            e.printStackTrace();
        } catch (InterruptedException e) {
            Log.e(Constants.DEBUG_TAG, e.getMessage());
            e.printStackTrace();
        } catch (IOException e) {
            Log.e(Constants.DEBUG_TAG, e.getMessage());
            e.printStackTrace();
        }
        waypointCalculator();
    }
```

Kôd 5.5. Dio programskog koda klase *DistanceTimeCalculator* zadužen za izračun vremena i puta do odredišta

Implementacija metode *tripCalculator* navedena u prikazu programskog koda 5.5. opisana je u nastavku.

Svaki zajednički put definiran je jednostavnom klasom *Trip* koja se sastoji od polazišne točke, odredišne točke te može imati nula ili više međustanica definiranih klasom *Waypoint*.

Rezultat poziva *Google Directions API* servisa, su informacije o mogućim putevima i pripadajućim vremenima da bi se došlo do željenog cilja. Dio koda zadužen za definiranje parametara i pozivanje servisa definiran je u konačnoj varijabli *request*.

```
final DirectionsApiRequest request = DirectionsApi
        .newRequest(this.geoApiContext)
        .mode(TravelMode.DRIVING)
        .origin(this.origin)
        .destination(
             getTrip().getDestination().getLocation())
        .waypoints(getTrip().getWaypointArray())
        .departureTimeNow();
```

Kao način putovanja odabrana je „Vožnja“ (engl. *DRIVING*), početna točka (engl. *Origin*), odredište (engl. *Destination*) i međustanice (engl. *Waypoints*) ukoliko ih ima.

Nakon definicije poziva servisa, poziva se sam servis linijom koda:

```
DirectionsRoute[] routes = request.await().routes;
```

Nakon što se dobije povratna informacija od servisa, rezultati se obrađuju u sljedećem bloku koda:

```
DirectionsRoute[] routes = request.await().routes;
this.setDirectionsRoute(routes[0]);
```

```
long duration = 0;

List<Waypoint> waypointList = this.trip.getWaypointList();
int i = 0;
for (DirectionsLeg leg : routes[0].legs) {
    duration = duration + leg.duration.inSeconds;
    if (i != routes[0].legs.length - 1) {
        waypointList.get(i).setDurationSec(duration);
    } else {
        this.trip.getDestination().setDurationSec(duration);
    }
    i++;
}
this.trip.setWaypointList(waypointList);
```

Kao zadnji korak opisivane metode *tripCalculator* poziv je metode *waypointCalculator()*.

Metoda *waypointCalculator()* implementirana je na sličan način kao i prethodno opisana metoda uz razliku što *tripCalculator()* metoda određuje parametre glavnog puta, a ova metoda određuje parametre svi planiranih puteva pojedinih korisnika. Često glavni planirani put ne prolazi istom točkom u prostoru gdje se nalazi i sam korisnik, nego je određen nekom međustanicom do koje je potrebno doći. Upravo je metoda *waypointCalculator()* zadužena za izračun parametara puta od korisnikove zadnje lokacije do međustanice.

Nakon što se dobije odgovor za sve pozive servisa, iz rezultata se uzima najbrži put, te se izračunaju vremena za sve međustanice od početne točke do odredišta. Potom se ti podaci prikažu na karti.

Postupak se ponavlja svakih nekoliko sekundi. Željeno vrijeme osvježavanja je definirano u globalnim konstantama te se može lagano izmijeniti.

Primjeri korištenja i način implementacije su proučeni u Google Directions API dokumentaciji (https://developers.google.com/maps/documentation/directions/start, 2019).

5.2. Poslužitelj

Poslužiteljska aplikacija sastoji se od baze podataka, čiji model je već ranije opisan, i poslovne logike koja je zadužena za prihvat, obradu te pohranu podataka. Pohranjene podatke, na zahtjev klijentskih aplikacija, kroz web servise prosljeđuje korisnicima.

Glavni zadatak poslužitelja jest omogućiti Android klijentskim aplikacijama međusobnu komunikaciju pomoću web servisa. Uz navedeno, poslužitelj pohranjuje podatke u svrhu obrade statistike i pregleda povijesnih podataka. Ključni dijelovi poslužiteljske aplikacije su:

- entiteti - modeli
- kontroleri – web servisi
- repozitoriji – metode za pristup podacima u bazi podataka

Klase kojima se grade entiteti su jednostavne Java klase koje se proširuju s anotacijama kako bi ih usko vezali uz pripadajuće tablice u bazi podataka. U prikazu programskog koda 5.6. prikazana je implementacija entiteta *DeviceLocation.*

```
@Entity
@Table(name = "device_location")
public class DeviceLocation {
    private long id;
    private double longitude;
    private double latitude;
    private Date datetime;
    private Set<Address> addresses = new HashSet<>();
    private User user;

    public DeviceLocation() {
    }

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(unique = true, nullable = false)
    public long getId() {
        return id;
    }
    public void setId(long id) {
        this.id = id;
    }

    @NotNull
    public double getLongitude() {
        return longitude;
    }
    public void setLongitude(double longitude) {
```

```
        this.longitude = longitude;
    }

    @NotNull
    public double getLatitude() {
        return latitude;
    }
    public void setLatitude(double latitude) {
        this.latitude = latitude;
    }

    @NotNull
    public Date getDatetime() {
        return datetime;
    }
    public void setDatetime(Date datetime) {
        this.datetime = datetime;
    }

    @OneToMany(
         mappedBy = "deviceLocation",
         fetch = FetchType.LAZY
    )
    public Set<Address> getAddresses() {
        return addresses;
    }
    public void setAddresses(Set<Address> addresses) {
        this.addresses = addresses;
    }

    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(
        name = "user_id",
        nullable = false,
        unique = false
    )
    @JsonBackReference
    @OnDelete(action = OnDeleteAction.CASCADE)
    public User getUser() {
        return user;
    }
```

```
    public void setUser(User user) {
        this.user = user;
    }
}
```

Kôd 5.6. Implementacija modela koji opisuje lokacijske podatke

Java *persistance* i Hibernate ORM anotacije počinju sa simbolom @ te svaka ima svoju ulogu. Ključne anotacije prikazane u primjeru koda 5.6. su opisane u nastavku (Varanasi, Belida, 2015):

- @Entity - označava da je klasa entitet
- @Table - definira kojoj tablici u bazi podataka entitet pripada
- @Id - označava atribut koji je primarni ključ
- @GeneratedValue - definira atribut kao vrijednost koju se automatski generira
- @Column - definira parametre kolone kojoj pripada atribut
- @NotNull - koristi se u svrhu zabrane postavljanja null vrijednosti nekog atributa
- @OneToMany - označava jedan naprema više vezu
- @JoinColumn - definira kolonu koja se koristi kao strani ključ
- @OnDelete - definira što će se dogoditi s vezanim entitetom u slučaju brisanja

Podacima spremljenima u bazu podataka pristupa se kroz repozitorij sučelja. Nasljeđivanjem JPA sučelja i dodavanjem anotacija, implementirane su osnovne funkcionalnosti pristupa podacima u nekoj tablici kao što su dohvat svih podataka, dohvat po primarnom ključu, spremanje i brisanje. Sučelju se mogu definirati dodatne metode kako bi se proširio osnovni set funkcionalnosti.

U nastavku je primjer sučelja s dvije dodatne metode.

```
@Repository
public interface DeviceLocationRepository extends
JpaRepository<DeviceLocation, Long> {
    @Query("SELECT t " +
            "FROM DeviceLocation t " +
            "WHERE t.datetime >= ?1 " +
            "AND t.datetime <= ?2 " +
            "AND t.user.id = ?3"
    )
    List<DeviceLocation> findBetweenDatesByUserId(
            Instant dtStart,
            Instant dtEnd,
```

```
            Long userId
    );

    @Query("SELECT t " +
            "FROM DeviceLocation t " +
            "WHERE t.datetime >= ?1 " +
            "AND t.datetime <= ?2"
    )
    List<DeviceLocation> findBetweenDates(
            Instant dtStart,
            Instant dtEnd
    );
}
```

Kôd 5.7. Repozitorij sučelje s dodatnim metodama

Metoda *findBetweenDatesByUserId* služi za dohvaćanje svih lokacijskih podataka između dvije točke u vremenu, vezanih uz nekog korisnika, odnosno jedinstvenu oznaku korisnika.

Metoda *findBetweenDates* služi za dohvaćanje svih lokacijskih podataka između dvije točke u vremenu. Obje metode kao ulazne parametre primaju dva objekta tipa *Instant* kojima su određene točke u vremenu, a dodatno metoda *findBetweenDatesByUserId* prima i brojčani identifikator korisnika koji je tip podataka *Long*.

Komunikacija između poslužitelja i Android klijentskih aplikacija ostvaruje se kroz web servise u JSON formatu. Web servisi, odnosno njihove krajnje točke, definiraju se kontroler klasama. Metode za obradu podataka u kontroler klasama su izgrađene oko HTTP metoda GET, POST i DELETE.

Primjer implementacije kontroler klase koja obrađuje HTTP zahtjeve pristigle na putanju „*/location*“ nalazi se u nastavku.

```
@RestController
@RequestMapping(value = "/location")
public class DeviceLocationController {
  private DeviceLocationRepository deviceLocationRepository;
  private UserRepository userRepository;
  private AddressRepository addressRepository;

  @Autowired
  public DeviceLocationController(
         DeviceLocationRepository deviceLocationRepository,
```

```
        UserRepository userRepository,
        AddressRepository addressRepository
) {
  this.deviceLocationRepository = deviceLocationRepository;
  this.userRepository = userRepository;
  this.addressRepository = addressRepository;
}

@GetMapping()
public List<DeviceLocation> getDeviceLocations(
        @CurrentUser UserPrincipal currentUser,
        @RequestParam(value = "dtStart")
          Optional<Instant> dtStartParam,
        @RequestParam(value = "dtEnd")
          Optional<Instant> dtEndParam
) {
  Instant dtStart = dtStartParam.orElse(
          Instant.ofEpochSecond(0)
  );
  Instant dtEnd = dtEndParam.orElse(
          Instant.now()
  );

  User u1 = userRepository
          .findByUsername(currentUser.getUsername())
          .orElseThrow(()
                  -> new ResourceNotFoundException(
                          "User",
                          "username",
                          currentUser.getUsername())
                  );
  List<DeviceLocation> deviceLocations =
    deviceLocationRepository.findBetweenDatesByUserId(
              dtStart,
              dtEnd,
              u1.getId()
    );
  return deviceLocations;
}
@PostMapping()
public DeviceLocation postDeviceLocation(
```

```
        @RequestBody DeviceLocationPayload deviceLocationPayload,
        Principal currentUser
    ){
        User user = userRepository
                .findByUsername(currentUser.getName())
                .orElseThrow(()
                        -> new ResourceNotFoundException(
                                "User",
                                "username",
                                 currentUser.getName())
                        );
        DeviceLocation deviceLocation = new DeviceLocation();
        deviceLocation.setFromPayload(deviceLocationPayload);
        deviceLocation.setUser(user);
        deviceLocationRepository.save(deviceLocation);
        for (AddressPayload addressPayload :
              deviceLocationPayload.getAddresses()) {
                Address address = new Address();
                address.fromPayload(addressPayload);
                address.setDeviceLocation(deviceLocation);
                addressRepository.save(address);
                deviceLocation.getAddresses().add(address);
        }
        return deviceLocation;
    }
}
```

Kôd 5.8. Kontroler klasa s dva web servisa

Klasa se sastoji od tri privatna atributa kojima se definiraju i inicijaliziraju potrebni repozitoriji za pristup podacima i dvije metode kojima su definirane krajnje točke web servisa.

Repozitorijima *DeviceLocationRepository*, *UserRepository* i *AddressRepository* pristupa se podacima o lokacijama, korisnicima i adresama.

Metoda *getDeviceLocations* i anotacija *@GetMapping()* definiraju HTTP GET zahtjev na putanji „*/location*“ te primaju tri parametra, točke u vremenu i trenutno prijavljenog korisnika. Metoda dohvaća podatke iz baze podataka i u JSON obliku ih šalje kao odgovor na zahtjev. HTTP POST zahtjev, na istoj putanji, definiran je metodom *postDeviceLocation* i anotacijom *@PostMapping()* te prima dva parametra: parametar sa svim podacima

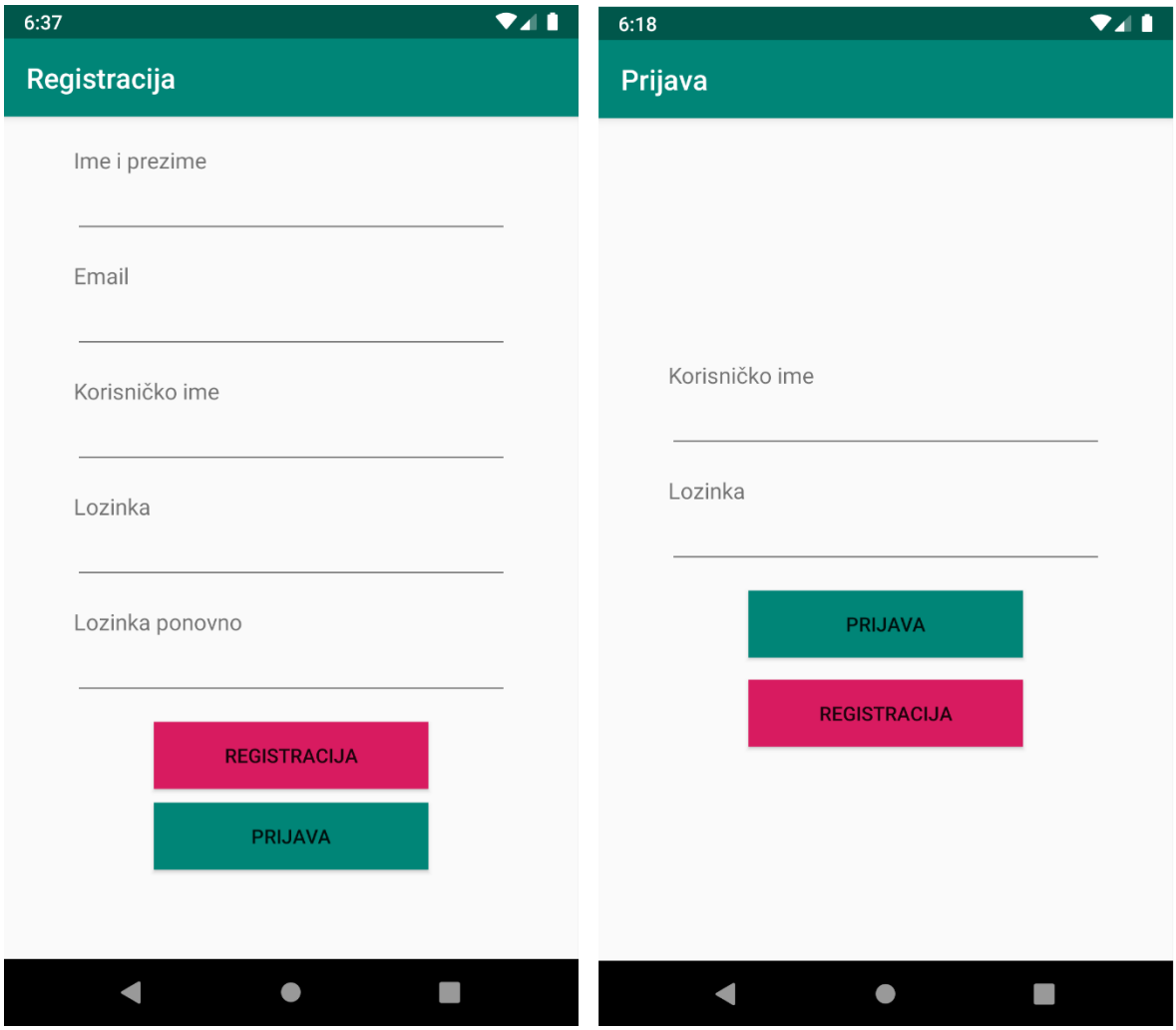
potrebnim za pohranu podatka o lokaciji uređaja i trenutnom prijavljenom korisniku uz koji su lokacijski podaci vezani.

6. Korištenje aplikacije

U nastavku rada, nalaze se upute za korištenje aplikacije s korisničkog stajališta. Uporaba aplikacije iz korisničke perspektive odnosi se na upute o inicijalnom pokretanju aplikacije, opisu navigacijskog izbornika i ponuđenih opcija, kreiranju i administraciji grupa i praćenju osoba unutar grupe. Radi lakšeg prikaza, korištene su slike koje prikazuju gdje se koja opcija u aplikaciji nalazi i na koji način se ista koristi.

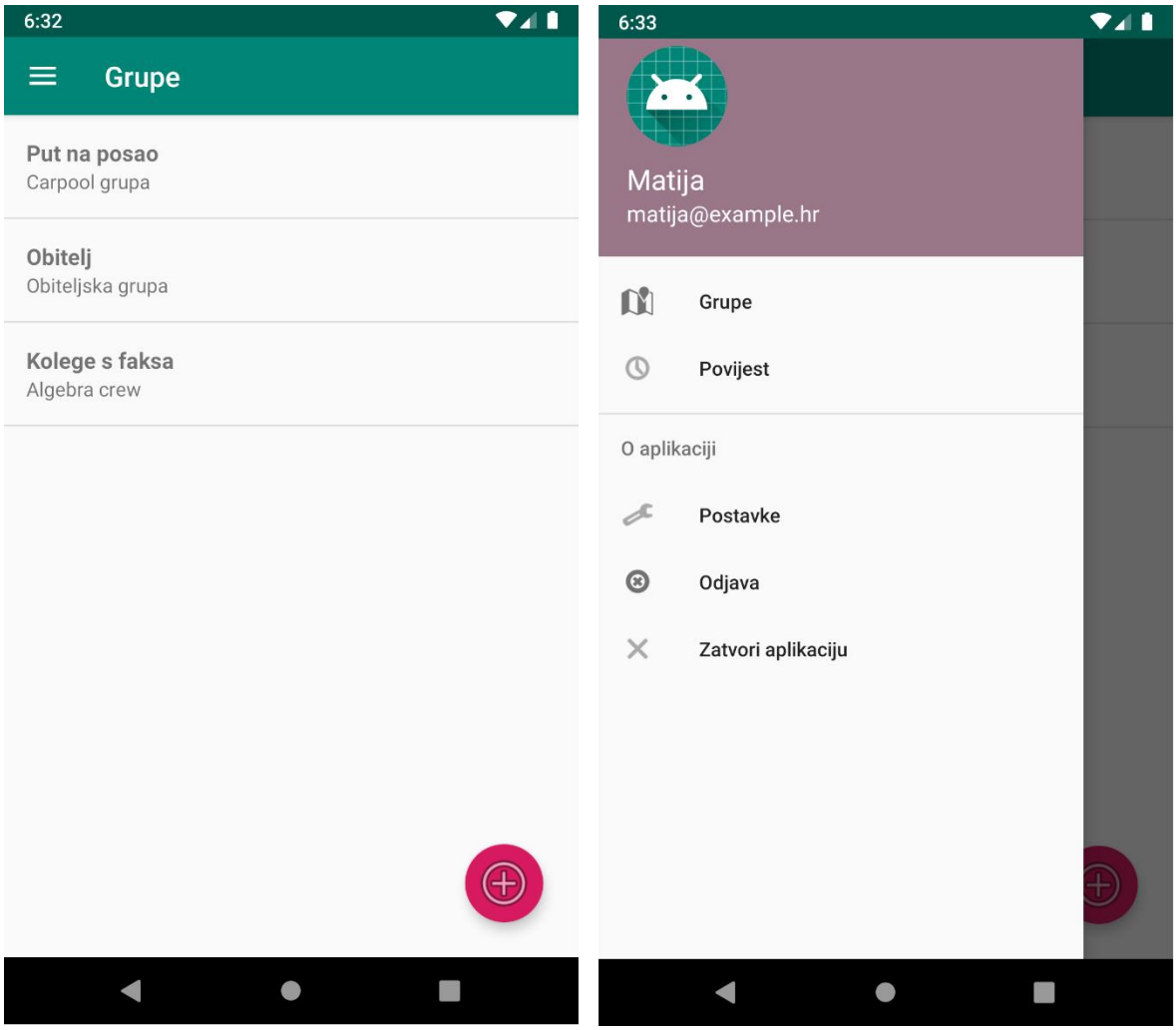
6.1. Inicijalno pokretanje aplikacije

Nakon instalacije, prilikom prvog pokretanja aplikacije, anonimnom korisniku nudi se mogućnost prijave te, u slučaju da već ima postojeći korisnički račun, nakon upisa korisničkog imena i lozinke može pristupiti korištenju aplikacije. Ukoliko korisnik još nema račun, može odabrati poveznicu na formu za kreiranje korisničkog računa te se nakon kreiranja računa, može vratiti na formu za prijavu. Bez registracije, odnosno prijave u aplikaciju, korisnik ne može koristiti aplikaciju.



Slika 6.1. Forme za registraciju i prijavu

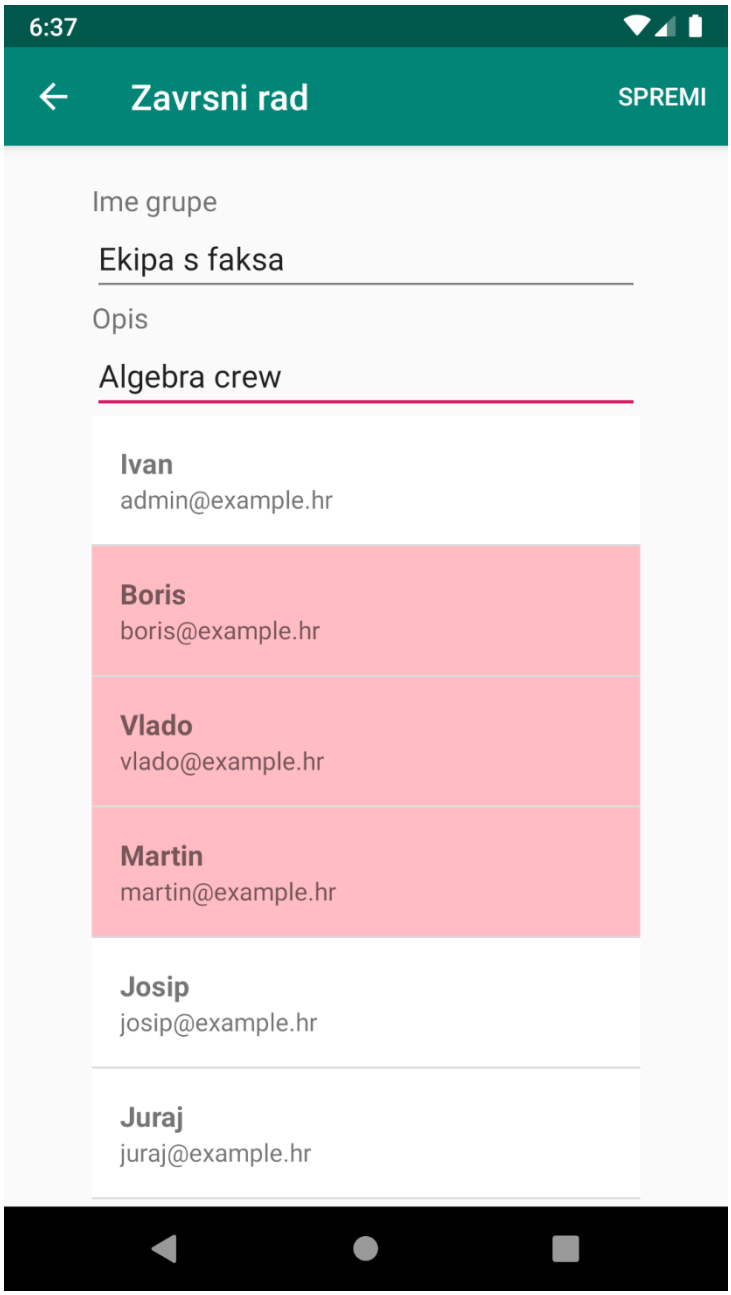
Nakon uspješne prijave, korisniku se prikazuje početni ekran s popisom pripadajućih grupa kao što je prikazano na slici 6.2. Odabirom poveznice u gornjem lijevom kutu ekrana (simbol s tri horizontalne crte), korisniku se otvara navigacijski izbornik. U naslovu izbornika prikazani su podaci o trenutno prijavljenom korisniku: profilna slika, ime i e-mail adresa. Koristeći navigacijski izbornik, korisnik može pristupiti ostalim dijelovima aplikacije. U slučaju da korisnik želi izaći iz aplikacije ili se privremeno odjaviti, može odabrati opcije „Zatvori aplikaciju“ ili „Odjava“. Postavkama unutar aplikacije može pristupiti putem poveznice „Postavke“. Pregled povijesne statistike može pregledati koristeći poveznicu „Povijest“. Izgled navigacijskog izbornika je prikazan na slici 6.2.



Slika 6.2. Početni ekran s popisom grupa i navigacijski izbornik

6.2. Kreiranje i administracija grupa

Na početnom ekranu, korisniku se prikazuje popis svih korisničkih grupa u kojima je član. Da bi pristupio sadržaju neke grupe, dovoljno je odabrati željenu grupu na popisu laganim dodirom. U desnom donjem kutu ekrana smješten je gumb za kreiranje nove grupe. Pritiskom na navedeni gumb, otvara se forma za kreiranje nove grupe. Forma za kreiranje grupe s popunjenim podacima prikazana je na slici 6.3.



Slika 6.3. Forma za kreiranje nove grupe

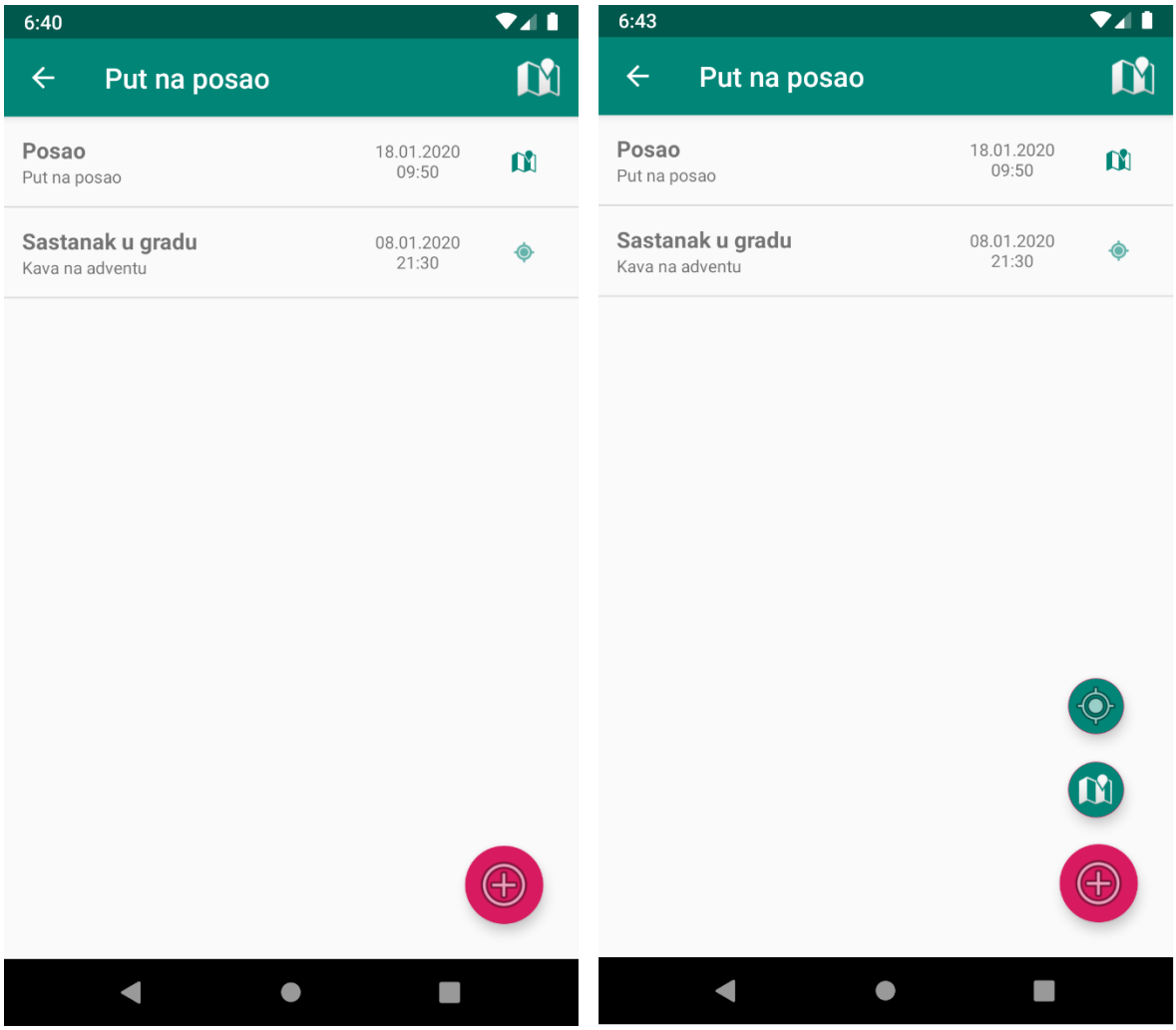
Prilikom kreiranja grupe potrebno je navesti ime grupe te njen kratak opis. Potom je, s popisa korisnika aplikacije, potrebno odabrati članove grupe koju korisnik želi kreirati. Navedeno se može napraviti na način da se korisnika kojeg se pridružuje grupi označi kratkim dodirom. Nakon što neki korisnik bude označen, na popisu poprima blago crvenu pozadinu. U slučaju odustajanja od dodavanja već označenog korisnika, dovoljno je još jednim kratkim dodirom maknuti odabir te će se pritom pozadina ponovno vratiti u bijelu boju. Po popunjavanju svih polja u formi, grupa se može kreirati odabirom gumba u desnom gornjem kutu. Da bi se grupa uspješno kreirala, potrebno je ispuniti sva polja te odabrati barem jednog korisnika. Korisnik koji kreira grupu automatski se dodaje te ujedno postaje administrator grupe. U

bilo kojem trenutku može se odustati od kreiranja grupe odabirom strelice u lijevom gornjem kutu ekrana. Aplikacija će se vratiti na popis postojećih grupa te će se svi popunjeni podaci izgubiti.

Dugim pritiskom na željenu grupu, na ekranu za popis grupa, korisniku se otvara forma za uređivanje grupe (ukoliko je korisnik administrator odabrane grupe). Forma je identična kao i forma za kreiranje grupe, no polja unutar forme će već biti ispunjena podacima grupe koja se uređuje. Ovdje se nudi mogućnost promjene naslova grupe te dodavanje ili micanje određenog člana iz grupe. Pritiskom na gumb „Spremi“, izmjene se pohranjuju. Odabirom gumba za povratak, eventualne izmjene će biti izgubljene ukoliko nisu pohranjene.

6.3. Praćenje osoba unutar grupe

Odabirom željene grupe, korisniku se prikaže popis svih aktivnosti u grupi kao što je prikazano na slici 6.4. Za svaku aktivnost naveden je naziv, opis, datum kada je potrebno biti na odredištu te simbol koji prikazuje o kojoj vrsti aktivnosti je riječ (put ili sastanak).



Slika 6.4. Prikaz sadržaja grupe

Aktivnosti se dijele na dvije vrste: sastanak i put. Sastanak se sastoji od odredišne točke i vremena kada počinje. Sastanak služi za istovremeno nalaženje više ljudi na nekom mjestu, a svatko na sastanak dolazi neovisno o drugome. Sastanak je dostupan svim članovima grupe, te će njihov put biti prikazan na karti ukoliko u tom trenutku odluče dijeliti svoju lokaciju. Put služi za zajedničko putovanje na neko odredište. Sastoji se od početne točke (najčešće korisnik koji je vozač i sl.), nekoliko usputnih stanica (točke na kojima treba pokupiti nekog člana grupe) te odredišta na koje svi članovi stižu.

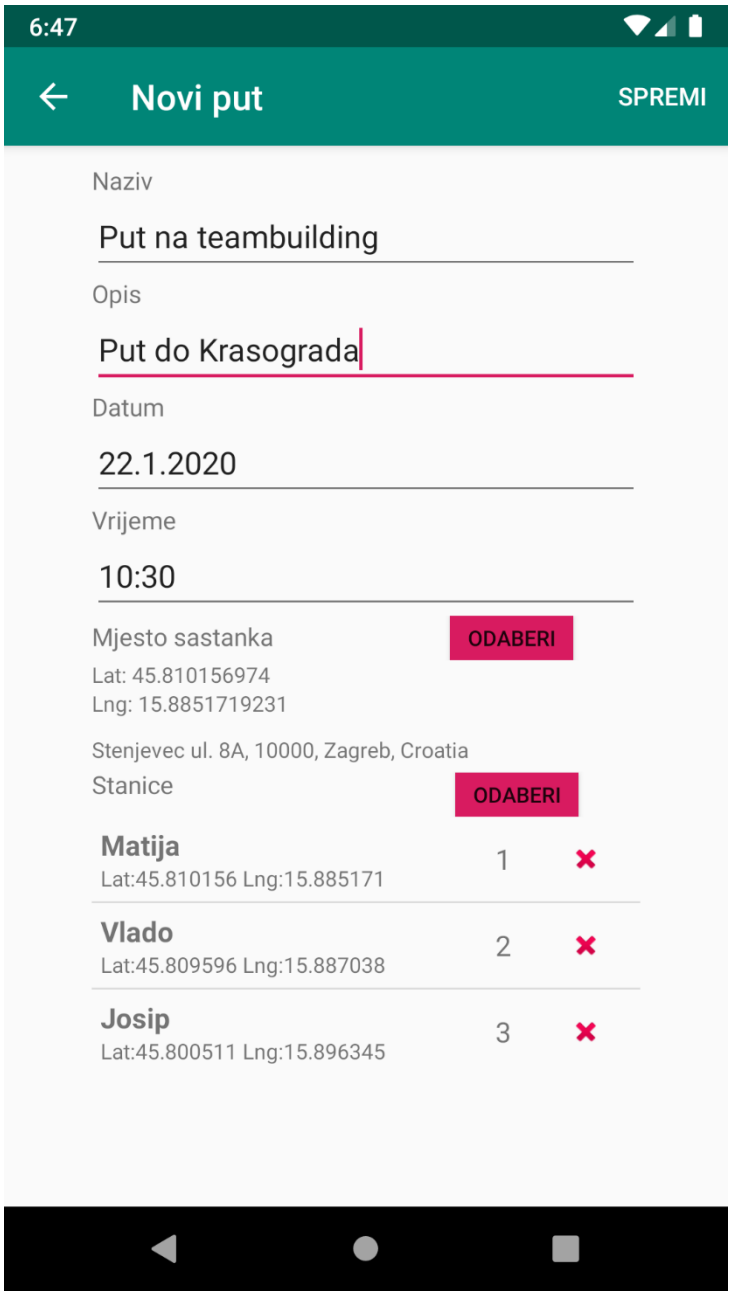
Dodirom gumba u donjem desnom kutu ekrana, prikazuju se dvije opcije koje omogućuju kreiranje novog puta ili novog sastanka. Opcije su prikazane na slici 6.4.

Prilikom kreiranja novog sastanka potrebno je navesti naziv i opis sastanka. Iz izbornika datuma i izbornika vremena potrebno je odabrati vrijeme održavanja sastanka. Kao zadnji korak, potrebno je na karti (dodirom gumba „ODABERI“) odabrati lokaciju sastanka. Pritiskom na gumb „SPREMI“ u gornjem desnom kutu ekrana kreira se novi sastanak. Da

bi se kreirao sastanak, potrebno je popuniti sva polja. Od kreiranja novog sastanka moguće je odustati u bilo kojem trenutku, odabirom strelice za povratak u gornjem lijevom kutu ekrana te se vratiti na popis aktivnosti grupe.

Kao i prilikom kreiranja sastanka, kod kreiranja novog zajedničkog puta potrebno je navesti naziv i opis te iz izbornika datuma i vremena odabrati željeni termin dolaska na odredište. Potrebno je odabrati lokaciju odredišta – gumb „DODAJ“ u retku „Odredište“. Kako bi se kreirao put, potrebno je popuniti sva do sada opisana polja. U nastavku se mogu dodati međustanice, no navedeno je opcionalno. Stanice se dodaju pritiskom gumba „DODAJ“ u retku „Stanice“. Pritom se odabire lokacija na karti te pripadajući korisnik s popisa korisnika grupe. Na dnu forme prikazuje se popis odabranih stanica te njihov redoslijed. Određena stanica može se ukloniti pritiskom na simbol „x“. Osoba koja kreira put je ujedno i vlasnik puta, odnosno vozač ili početna točka. Kao i u slučaju prethodnih formi, kreiranje puta vrši se odabirom opcije u gornjem desnom kutu, a u slučaju odustajanja odabire se strelica u gornjem lijevom dijelu ekrana.

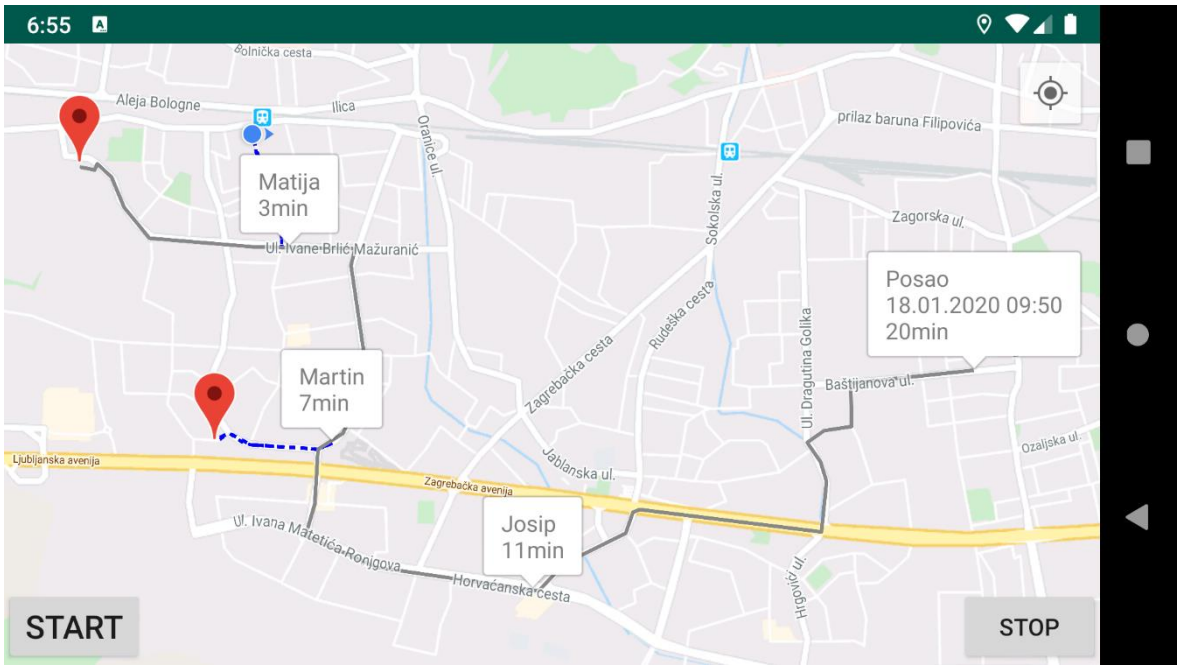
Popunjena forma za kreiranje novog puta je prikazana slici 6.5.



Slika 6.5. Forma za kreiranje zajedničkog puta

Prilikom odabira zajedničkog puta na popisu aktivnosti grupe, otvara se karta koja prikazuje trenutni status i lokaciju svih članova puta. Sivom ispunjenom linijom ucrtana je planiran put kojim će se vozač kretati. Svaka međustanica označena je pravokutnikom s imenom osobe koja se nalazi na toj stanici te maksimalnim vremenom kada će obje osobe biti na toj točci. Sudionici puta označeni su crvenim markerom na karti koji označava njihovu zadnje poznatu lokaciju. Svi članovi grupe, osim vozača, imaju ucrtan put plavom isprekidanom linijom do točke na kojoj će ih vozač pokupiti. Vrijeme iskazano na međustanici izračunava se tako da se u obzir uzme vrijeme koje je potrebno da vozač stigne na međustanicu i vrijeme potrebno članu koji je vezan uz tu stanicu. Prikazat će se vrijeme one osobe kojoj treba

najviše vremena da stigne do te točke. Npr. ako vozaču treba pet minuta do točke, a osobi koju će pokupiti na toj točci, treba sedam minuta, prikazati će se vrijeme od sedam minuta. Pri tome se u obzir uzima i kašnjenje koje u ovom slučaju iznosi dvije minute. Vremenu iskazanom na svim ostalim međustanicama do odredišta uračunati će se vrijeme kašnjenja. Navedeni postupak primijenit će se za svaku međustanicu. U slučaju da posljednja lokacija osobe nije poznata, ni planirani put ni sama lokacija osobe neće biti prikazana na karti, a vrijeme prikazano na međustanici će biti ono za koliko će vozač biti na toj stanici.



Slika 6.6. Prikaz zajedničkog puta

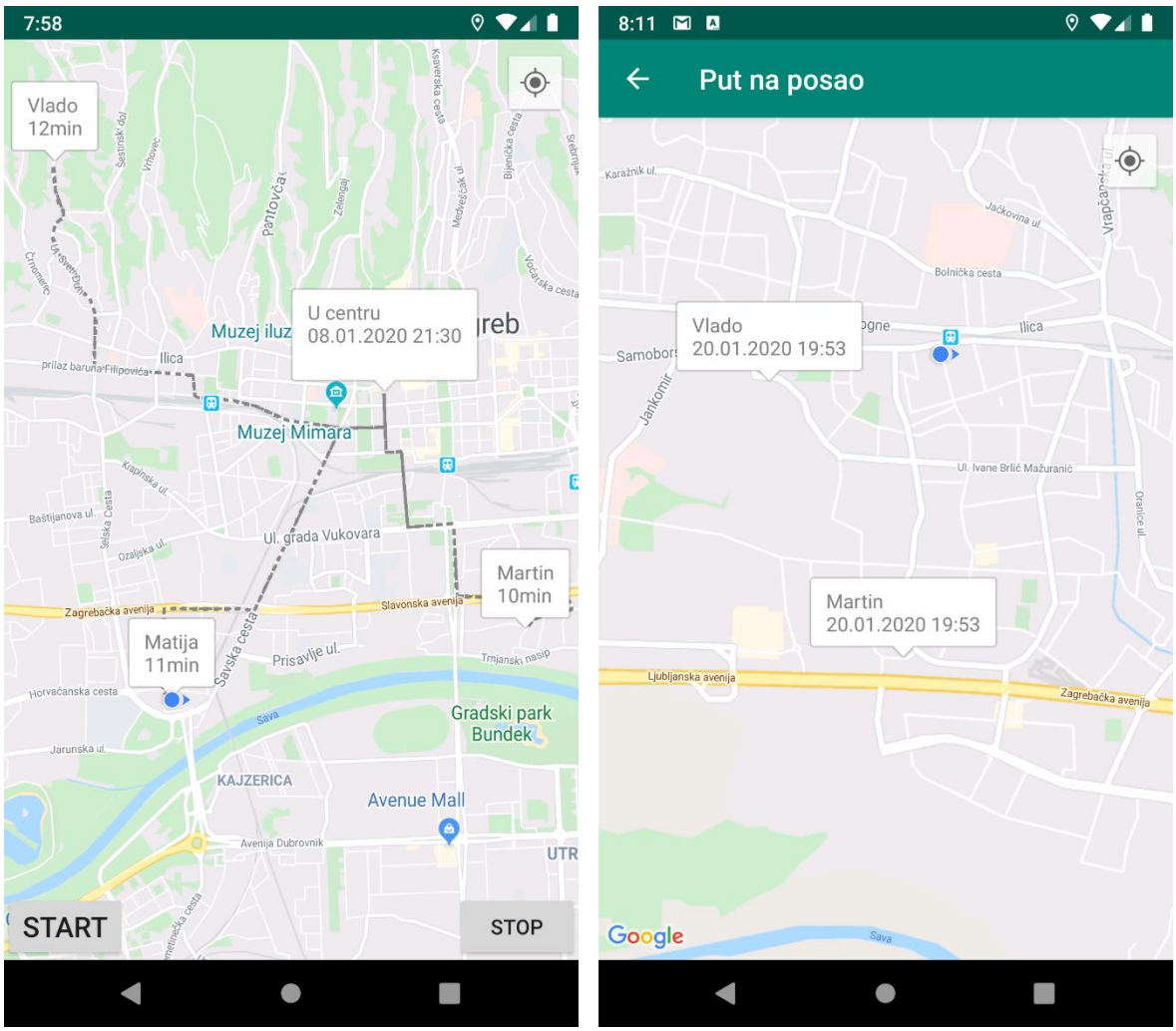
Primjer takvog puta vidimo na slici 6.6. Na slici se može primijetiti da Josipova lokacija nije poznata te ujedno njegova lokacija i put nisu ucrtani do stanice na kojoj je predviđeno da ga vozač pokupi. Također, Matijina lokacija nije prikazana crvenim markerom, nego plavim krugom i trokutom koji prikazuje smjer kretanja. Matija je korisnik aplikacije u kojem je slika ekrana kreirana. Korisnicima se na njihovom uređaju na taj način prikazuje njihova lokacija.

Na dnu ekrana s lijeve i desne strane nalaze se gumbi „START“ i „STOP“. Njima se može kontrolirati kada se lokacija korisnika dijeli i šalje prema poslužitelju kako bi bila vidljiva ostalim članovima grupe.

Ako se na popisu aktivnosti jedne grupe odabere aktivnost tipa *sastanak,* otvara se karta s lokacijom odredišta. U kvadratu s tekstom naveden je naziv sastanka i vrijeme. Na karti se također vide zadnje lokacije svih članova grupe koji u tom trenutku dijele svoju lokaciju. Uz

njihovu lokaciju, isprekidanom sivom linijom prikazan je i najbolji put do mjesta sastanka te vrijeme potrebno da se stigne na željeno mjesto sastanka. Kao i u slučaju prikaza zajedničkog puta, korisnik gumbima na dnu ekrana može kontrolirati kada dijeli svoju lokaciju s ostalim članovima grupe.

Ukoliko korisnik želi pregledati kartu s geolokacijskim podacima ostalih članova, to može napraviti klikom na simbol karte u gornjem desnom uglu ekrana. Nakon učitavanja karte, u kvadratima s tekstom prikazane su poznate lokacije ostalih članova grupe te vrijeme kada je podatak o lokaciji zabilježen. Na sljedećoj slici nalazi se prikaz karte s posljednjim poznatim lokacijama članova grupe te primjer prikaza sastanka.



Slika 6.7. Prikaz sastanka i karte s posljednjim poznatim lokacijama članova

6.4. Pregled povijesnih podataka

U glavnom izborniku na početnom ekranu, ponuđena je opcija „Povijest“. Funkcionalnost nije implementirana u praktičnom dijelu rada, no navedena opcija će prikazivati povijesne lokacijske podatke i statistiku prevaljenih putovanja.

Povijesni podaci moći će se pregledati prema:

- vremenskom periodu (jedan mjesec, tri mjeseca, šest mjeseci, jedna godina)
- zajedničkom putu kao vrsti aktivnosti unutar grupe

Povijesni podaci će korisniku omogućiti statistiku posjeta lokacijama, vremena potrebnog da s jedne lokacije dođu na drugu, a u svrhu što boljeg uvida u vlastita ili kretanja korisnika koji je podijelio informacije u nekom vremenskom periodu.

Prilikom pregleda statistike nekog od zajedničkih puteva, korisnik će imati mogućnost pregleda najkraćeg, najdužeg i prosječnog vremena trajanja puta.

Pregled i analiza povijesnih podataka, omogućit će korisniku bolju procjenu vremena potrebnog za određeni put, te dolazak do željene destinacije u najkraćem mogućem roku.

Zaključak

Za potrebe ovog završnog rada razvijeno je programsko rješenje za napredno praćenje i nadzor osoba. Rješenje se sastoji od poslužiteljske i klijentske aplikacije. Rad je pokazao primjer razvoja programskog rješenja primjenom programskog inženjerstva.

Na početku rada objašnjene su potrebe za ovakvom vrstom rješenja te su definirani korisnici, raspisani funkcionalni i nefunkcionalni zahtjevi. Prema tome je razrađen i razvijen model podataka koji je u potpunosti implementiran u poslužiteljskoj aplikaciji. Definirane su arhitektura i tehnologije koje su korištene u izradi aplikacija.

Razvijena je Android korisnička aplikacija kojom je korisnicima, uz pomoć poslužitelja, omogućeno dijeljenje vlastite lokacije kao i pregled lokacija i kretanja ostalih korisnika unutar zatvorene grupe. Ključni dijelovi programskog koda navedeni su i opisani te je opisano korištenje aplikacije uz priložene slike korisničkog sučelja.

Arhitektura programskog rješenja omogućava razvoj novih mobilnih klijenata poput aplikacije za iOS mobilnu platformu bez potrebe za izmjenama na postojećem poslužitelju.

Proučavanjem korištene literature i web stranica te dokumentacijama korištenih servisa, usvojeno je znanje o različitim tehnologijama, javno dostupnim bibliotekama i Google geolokacijskim servisima. Navedeno je, osim razvoja ovog rješenja dovelo i do novih ideja i funkcionalnosti koje bi još više mogle koristiti korisnicima, poput integracije s društvenim mrežama u svrhu lakšeg kreiranja korisničkog računa i povezivanja s vlastitim kontaktima.

Popis kratica

API	*application programming interface*	sučelje za programiranje aplikacija
ER	entity–relationship	veze entiteta
HTTP	hyper text transfer protocol	web protokol
HTTPS	hyper text transfer protocol secure	kriptirani web protokol
JPA	Java Persistence API	API za upravljanje podacima
JSON	JavaScript object notation	format podataka
JWT	JSON web token	web token u JSON formatu
ORM	Object-relational mapping	relacijsko mapiranje objekata

Popis slika

Popis kôdova

Literatura

[1] KRAJCAR M., BELANI H. *Projektni razvoj aplikacija.* Zagreb: Algebra, 2011.

[2] BOŽIĆ, J., KADEŽABEK, T., TOMIĆ, F., KAŠTELAN, T. *Izrada aplikacija za mobilne uređaje*. Zagreb: Algebra, 2013.

[3] HORTON, J. *Android Programming for Beginners.* Second Edition: Packt Publishing, 2018.

[4] VARANASI, B., BELIDA, S. *Spring REST.* Apress, 2015.

[5] ĐAMBIĆ, G. *Oblikovanje baza podataka.* Zagreb: Algebra, 2009.

[6] Hibernate JavaDoc, https://docs.jboss.org/hibernate/orm/3.5/javadocs/org/hibernate/dialect/package-summary.html, prosinac 2019.

[7] Android Developers, https://developer.android.com, siječanj 2020.

[8] Google Developers, https://developers.google.com, prosinac 2019.

[9] https://www.privacy-regulation.eu/hr/index.htm, veljača 2020.

ALGEBRA
VISOKO UČILIŠTE

ANDROID APLIKACIJA ZA NAPREDNO PRAĆENJE I NADZOR KRETANJA OSOBA

Pristupnik: Ivan Leskovar, 0036439014

Mentor: dipl. ing. Aleksander Radovan