Mobilna Android aplikacija za preporučivanje aktivnosti u prirodi

Krpes, Filip

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

Sveučilišni preddiplomski studij računarstva

MOBILNA ANDROID APLIKACIJA ZA PREPORUČIVANJE AKTIVNOSTI U PRIRODI

Završni rad

Filip Krpes

Osijek, 2023.

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

Obrazac Z1P: Obrazac za ocjenu završnog rada na sveučilišnom prijediplomskom studiju

Ocjena završnog rada na sveučilišnom prijediplomskom studiju

Ime i prezime pristupnika:	Filip Krpes
Studij, smjer:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. pristupnika, god.	R4227, 25.07.2018.
JMBAG:	0165078815
Mentor:	prof. dr. sc. Goran Martinović
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Mobilna Android aplikacija za preporučivanje aktivnosti u prirodi
Znanstvena grana završnog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rada:	U teorijskom dijelu završnog rada potrebno je opisati i analizirati zahtjeve i izazove pri planiranju posjeta lokacijama koje omogućuju sportske, rekreacijske, zabavne i slične aktivnosti u prirodi. Uzimajući u obzir mogućnosti sličnih postojećih rješenja, treba definirati zahtjeve, te predložiti model i arhitekturu mobilne Android aplikacije koja omogućuje definiranje profila korisnika i njegovih sklonosti aktivnostima u prirodi, opis mogućnosti koje pruža pojedina lokacija, dodavanje novih obilježja i fotografija s lokacije, te sustav preporuka lokacije na temelju mogućih aktivnosti, vremenskih prilika i ocjena od strane korisnika mobilne

Datum prijedloga ocjene završnog rada od strane mentora:	22.09.2023.
Prijedlog ocjene završnog rada od strane mentora:	Dobar (3)
Datum potvrde ocjene završnog rada od strane Odbora:	28.09.2023.
Ocjena završnog rada nakon obrane:	Dobar (3)
Datum potvrde mentora o predaji konačne verzije završnog rada čime je pristupnik završio sveučilišni prijediplomski studij:	01.10.2024.

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

IZJAVA O IZVORNOSTI RADA

Osijek, 01.10.2024.

Ime i prezime Pristupnika:	Filip Krpes
Studij:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	R4227, 25.07.2018.
Turnitin podudaranje [%]:	11

Ovom izjavom izjavljujem da je rad pod nazivom**: Mobilna Android aplikacija za preporučivanje aktivnosti u prirodi**

izrađen pod vodstvom mentora prof. dr. sc. Goran Martinović

i sumentora

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

Potpis pristupnika:

SADRŽAJ

1. UVOD 1

1.1. Zadatak završnog rada 2

2. PROBLEMATIKA ODABIRA AKTIVNOSTI U PRIRODI I OSNOVE SUSTAVA PREPORUKA S PREGLEDOM STANJA U PODRUČJU 3

2.1. Sustavi preporuka 3

2.1.1. Podatkovni objekti sustava preporuka 3

2.1.2. Podjela sustava za preporučavanje i tehnike preporučivanja 4

2.1.3. Izazovi u planiranju sustava preporuka 5

2.2. Preporuka aktivnosti u prirodi 6

2.3. Pregled stanja u području i analiza sličnih rješenja 6

2.3.1. Sustavi preporuke aktivnosti na temelju vremenskih uvjeta 6

2.3.2. Sustavi preporuke aktivnosti na temelju geolokacije 7

2.3.3. Sustavi preporuke aktivnosti na temelju zajednice 8

3. ANALIZA ZAHTJEVA I MODELIRANJE MOBILNE APLIKACIJE 9

3.1. Analiza zahtjeva za mobilnu aplikaciju 9

3.1.1. Funkcijski zahtjevi 9

3.1.2. Nefunkcijski zahtjevi 9

3.2. Model mobilne aplikacije 10

3.2.1. Algoritam preporučivanja 11

4. PROGRAMSKO RJEŠENJE MOBILNE APLIKACIJE 14

4.1. Korištene programske tehnologije, jezici i razvojne okoline 14

4.1.1. Kotlin 14

4.1.2. Android Studio 14

4.1.3. Fused Location Provider API 15

4.1.4. Open Weather API 15

4.1.5. Places API 16

4.1.6. XML 17

4.1.7. Shared Preferences 17
4.1.8. Baza podataka Room 17
4.2. Programsko rješenje na korisničkoj strani 17
4.2.1. Implementacija korisničkog sučelja 17
4.2.2. Prilagodba korisničkog sučelja 23
4.3. Programsko rješenje na poslužiteljevoj strani 23
4.3.1. Dohvaćanje lokacije korisnika i trenutnog meteorološkog stanja 24
4.3.2. Spremanje i dohvaćanje korisnikovih preferenci 27
4.3.3. Preporuka aktivnosti 29
4.3.4. Pohrana i dohvaćanje korisničkih ocjena 38
5. PRIKAZ KORIŠTENJA I ANALIZA RADA MOBILNE APLIKACIJE 44
5.1. Prikaz korištenja mobilne aplikacije 44
5.2. Ispitivanje mobilne aplikacije i analiza rezultata 47
5.2.1. Ispitni slučaj 1 47
5.2.2. Ispitni slučaj 2 48
5.2.3. Ispitni slučaj 3 50
5.3. Analiza rezultata 52
ZAKLJUČAK 53
LITERATURA 54
SAŽETAK 56
ABSTRACT 57
PRILOZI 58

1. UVOD

Aktivnosti u prirodi, bile one rekreativne, sportske ili zabavne su bitan čimbenik u ljudskom fizičkom i mentalnom zdravlju. Ovo se posebno očituje za vrijeme pandemije Covid-19, koja je početkom 2020. godine iduće dvije godine prisiljavala ljude da izbjegavaju socijalne kontakte i ostaju u svojim domovima. Takva vrsta izolacije je negativno utjecala na mentalno zdravlje mnogih ljudi, posebno onih skupina podložnih takvim problemima kao što su mladi ljudi i studenti. Socijalna izolacija nije jedini čimbenik mentalnih problema već i nedovoljan boravak u prirodi može biti štetan. Planiranje ovakvih aktivnosti nije uvijek lako, jer različiti ljudi imaju različite zahtjeve za aktivnosti kojima bih se željeli baviti. Je su li to fizička blizina mjestu stanovanja, koliko sama aktivnost košta ili je li neka aktivnost zadovoljava standarde kvalitete pojedine osobe. Pronaći odgovarajuću aktivnost nije uvijek lako i zbog činjenice da neke usluge pri preporučivanju aktivnosti ne uzimaju u obzir korisnika već daju prioritet vlasniku oglasa koji je dao više novca za oglašavanje svojeg oglasa.

Iz navedenih razloga je osmišljena aplikacija za preporučivanje aktivnosti u prirodi. Aplikacija treba korisniku ubrzati i olakšati pronalazak aktivnosti u prirodi ovisno o njegovim sklonostima opisanima u njegovom korisničkom profilu, te prema korisnikovoj lokaciji i vremenskim prilikama. Aplikacija će također imati mogućnost uz postojeća obilježja pojedinih lokacija dodati i nova obilježja i slike.

U drugom poglavlju opisuju se zahtjevi i izazovi pri planiranju posjeta lokacijama koje omogućuju sportske, zabavne, rekreativne ili slične aktivnosti u prirodi. Nadalje, opisuju se osnove sustava za preporučivanje kao i primjeri postojećih rješenja. Treće poglavlje predstavlja model i arhitekturu mobilne aplikacije, kao i korištene postupke za njeno ostvarivanje. Četvrto poglavlje daje opis programskih alata korištenih prilikom izrade, te opis programskog koda mobilne aplikacije. Peto poglavlje sadrži primjere primjene aplikacije, njeno ispitivanje i analizu rezultata.

1.1. Zadatak završnog rada

U teorijskom dijelu završnog rada potrebno je opisati i analizirati zahtjeve i izazove pri planiranju posjeta lokacijama koje omogućuju sportske, rekreacijske, zabavne i slične aktivnosti u prirodi. Uzimajući u obzir mogućnosti sličnih postojećih rješenja, treba definirati zahtjeve, te predložiti model i arhitekturu mobilne Android aplikacije koja omogućuje definiranje profila korisnika i njegovih sklonosti aktivnostima u prirodi, opis mogućnosti koje pruža pojedina lokacija, dodavanje novih obilježja i fotografija s lokacije, te sustav preporuka lokacije na temelju mogućih aktivnosti, vremenskih prilika i ocjena od strane korisnika mobilne aplikacije. Nadalje, treba opisati potrebne programske tehnologije, jezike i razvojne okvire. U praktičnom dijelu rada, treba razviti mobilnu aplikaciju s bazom podataka na temelju predloženog modela. Mobilnu aplikaciju potrebno je ispitati i analizirati za različite profile korisnika i različite mogućnosti lokacija za provođenje aktivnosti.

2. PROBLEMATIKA ODABIRA AKTIVNOSTI U PRIRODI I OSNOVE SUSTAVA PREPORUKA S PREGLEDOM STANJA U PODRUČJU

U ovom poglavlju definira se što je sustav za preporuke i opisuje se način rada i svojstva različitih sustava za preporučivanje. Objašnjavaju se najčešći izazovi pri izradi sustava za preporučivanje. Opisuje se kakav je to sustav za preporuku aktivnosti u prirodi koji će se koristiti u izradi aplikacije. Te se promatra stanje u području i analiziraju slična rješenja.

2.1. Sustavi preporuka

Sustavi za preporuke su programski alati i tehnike koje pružaju prijedloge za predmete koji bi korisniku mogli biti od koristi. Ovakve preporuke se mogu upotrijebiti kako bih se olakšao proces dobivanja odluke je li koje vijesti čitati ili koju knjigu kupiti [2]. Drugim riječima sustavi za preporuke pokušavaju filtrirati informacije bitnije korisniku, na ovaj način pokušava predvidjeti moguće preferencije korisnika [3]. „Predmet" je opći pojam za ono što sustav preporučuje korisniku. Sustavi za preporuke se najčešće fokusiraju na specifičnu vrstu predmeta i sukladno tome dizajn, grafičko sučelje i način dobivanja preporuke takvih sustava su prilagođeni dobiti efikasan i koristan prijedlog za taj specifičan tip predmeta. Sustavi preporuke su namijenjeni korisnicima koji nemaju dovoljno iskustva ili sposobnosti da ocijene veći broj predmeta koji pruža neki servis.

2.1.1. Podatkovni objekti sustava preporuka

Postoje tri vrste općih podatkovnih objekata koje koristi sustav preporuka: predmet, korisnik i transakcija. Predmet je podatkovni objekt koji se preporučuje korisniku. Predmeti mogu općenito biti opisani svojom složenosti, vrijednosti i korisnosti korisniku. Korisnost predmeta može ovisiti o drugim (kontekstualnim) varijablama kao što su korisnikovo znanje nekog područja. Vrijednost može biti pozitivna ili negativna ovisno o tome je li korisniku potreban takav predmet. Svaki predmet ima svoj trošak bio on kognitivan koji se pojavljuje kada korisnik traži željeni predmet ili sami monetarni trošak predmeta. Ako je predmet korisniku zadovoljavajuć i potreban njegova vrijednost će nadvladati trošak predmeta. Predmeti mogu biti vremenski osjetljivi tj. vrijednost im opada s vremenom, kao što bih bio slučaj s vijestima. Sustav preporuka koristi svojstva i značajke predmeta kako bih odredio korisnost tog predmeta za kupca.

Korisnik je podatkovni objekt koji predstavlja podatke korisnika i time njegove ciljeve i karakteristike. Korisničkim podacima stvara se korisnički profil tj. model korisnika. Podaci koji se koriste u korisničkom modelu ovisi o tehnici preporučivanja. Neki od najčešće korištenih načina

modeliranja korisnika su:

- Kao skup ocjena koje je korisnik pridijelio pojedinim predmetima (suradničko filtriranje).
- Kao skup osobnih podataka korisnika kao što su dob, spol, zanimanje i obrazovanje (demografski sustav preporuka).
- Kao skup ponašanja korisnika kao npr. koje web stranice posjećuju.

Transakcije su podatkovni objekti koji predstavljaju interakcije između korisnika i sustava preporuka. Ovaj objekt pohranjuje bitne podatke koji se generiraju dok korisnik upotrebljava sustav preporuka.Ti podaci se mogu dalje koristiti u algoritmu koji generira preporuke u sustavu. Transakcije također mogu sadržavati povratnu informaciju koja može biti eksplicitna ili implicitna. Eksplicitne povratne informacije su dobivene traženjem korisnikovog mišljenja o pojedinom predmetu. Ovo može biti u obliku ocijene (1-5), binarnog ocjenjivanja (Sviđa mi se, ne sviđa mi se), unarnog ocjenjivanja (korisnik je pregledavao ili kupio predmet) ili neke druge vrste. Implicitne povratne informacije se dobivaju praćenjem korisnikovog ponašanja.

2.1.2. Podjela sustava za preporučavanje i tehnike preporučivanja

Osnova svih tehnika preporučivanja i ono što im je zajedničkome jest njihova glavna funkcionalnost. Kako bih se implementirala glavna funkcionalnost tj. određivanje koji su predmeti interesantni za korisnika, sustav preporuka mora „predvidjeti" da je predmet vrijedan preporučiti. Kako bih ovo postigao sustav moram moći pretpostaviti korisnost bar nekoliko predmeta ili barem moći usporediti korisnost nekih predmeta i tada procijeniti na temelju usporedbe koji predmet preporučiti.

Osnovna podjela sustava za preporučivanje je na sustave s personaliziranim preporukama i na one bez njih. Sustavi koji se temelje na personaliziranim preporukama često svoje preporuke prikazuju u obliku poredanih lista. Predmeti s liste su poredani s obzirom na korisnikove podatke ili prijašnje transakcije. Sustavi s nepersonaliziranim preporukama daju preporuke ovisno o prosječnim povratnim informacijama drugi korisnika. Ovakve preporuke je jednostavnije stvoriti ali s obzirom na to da preporuke ne ovise o pojedinom korisniku svi korisnici će imati istu preporuku.

„Klasična" raspodjela sustava preporuka dijeli ih na šest kategorija ovisno o informacijama koje koriste za preporuku predmeta:

- Na temelju sadržaja - Sustav uči preporučivati predmete koji su slični onima koji su se ranije svidjeli korisnik. Sličnost predmeta je izračunat ovisno o svojstvima uspoređenih predmeta.
- Suradničko filtriranje – Preporučuje korisniku predmete koje su drugi korisnici sa sličnim ukusima ranije pozitivno ocijenili. Sličnost ukusa se računa na temelju sličnosti povijesti ocijenjivanja predmeta korisnika.
- Demografski – Preporučuje predmete ovisno o demografskom profilu korisnika. Predpostavlja se da bih se trebale različite preporuke generirati ovisno o različitim demografskim skupinama.
- Na temelju znanja korisnika – Preporučuje predmete ovisno o znanju korisnika u specifičnom području.
- Na temelju zajednice – Preporučuje predmete ovisno o preferencama korisnikovih prijatelja.
- Hibridni sustav preporuka – Kombinacija prijašnjih tehnika. Kombinacijom dvije tehnike ispravljamo nedostatke prve sa svojstvima druge.

2.1.3. Izazovi u planiranju sustava preporuka

Kvaliteta sustava za preporučivanje ovisi o količini podataka kojim ima pristup kao i njihovoj kvaliteti. Izazovi u planiranju su različiti i mnogobrojni. Kvaliteta podataka ovisi o korisniku i nije uvijek osigurana. Ako je kvaliteta osigurana i dalje postoje etičke problemi ovisno o tome kako korisnik želi da se koriste njegovi podaci. Potrebno je stvoriti ravnotežu između potreba korisnika i poslužitelja. Slijede najčešći problemi pri planiranju sustava za preporučivanje

Problem hladnog starta se može klasificirati u 2 kategorije: hladni start novog korisnika i hladni start novog predmeta. Hladni start kod predmeta nastaje kada predmet nema dovoljno prethodnih ocjena za taj predmet. Hladni start kod korisnika nastaje kada korisnik tek počne koristiti sustav preporuka ili sustav nema informacija o tom korisniku. Problem hladnog starta se može zaobići stvaranjem korisničkog profila s preferencama korisnika pri prvom korištenju ili korištenjem suradničkog filtriranja.

Porast broja korisnika i predmeta u sustavu samom sustavu je potrebno više resursa kako bi nastavio davati najtočnije moguće preporuke korisnicima. Najviše se resursa troši na određivanje korisnika sa sličnim ukusima i na određivanje sličnosti između svojstava predmeta. Problem skalabilnosti se najčešće pojavljuje kod suradničkog filtriranja. Kako bi se dobile najtočnije

preporuke potrebno je dati veliku količinu podatak sustavu za preporuke. Ovo može kršiti prava privatnosti korisnika ili pružati povećanu opasnost od krađe podataka ako podatkovni sustav nije dovoljno osiguran. Problem vezan uz privatnost je posebno problem kod sustava koji koriste demografski način preporučivanja. Privatnost i sigurnost korisnika uvijek mora biti na prvom mjestu. Svi korišteni podaci moraju biti dobro zaštićeni od ne ovlaštenog pristupa i potrebna je potpuna transparencija koji se podaci koriste i zašto.

Svrha sustava za preporuku je preporuka predmeta koji ne samo odgovaraju korisniku već i pružaju neku novu ideju ili iznenađenje korisniku kojih se on sam možda ne bi sjetio. Pre specijalizirani sustav preporučuje mali skup previše predvidivih stvari. Ovo je jedan od najčešćih problema koji se pojavljuju kod sustava za preporuku na temelju sadržaja. Primjer ovoga bih bila platforma za pregled videa koja konstantno preporučuje istih nekoliko videa.

2.2. Preporuka aktivnosti u prirodi

Preporuka aktivnosti u prirodi ovisi o tri čimbenika: hidro-meteorološki uvjeti, korisnikove preference i geolokacija. Najznačajniji čimbenik pri odabiru željene aktivnosti su korisnikove preference koje se biraju prvim korištenjem aplikacije stvaranjem korisničkog računa. Korisnikove preference utječu na aktivnosti koje su prikazane dok fizička udaljenost od aktivnosti utječe na to koje specifične lokacije će biti prikazane. Hidro-meteorološki uvjeti samo utječu na dostupnost pojedine aktivnosti tj. služe kao upozorenje da nije prikladno vrijeme za pojedinu aktivnost.

2.3. Pregled stanja u području i analiza sličnih rješenja

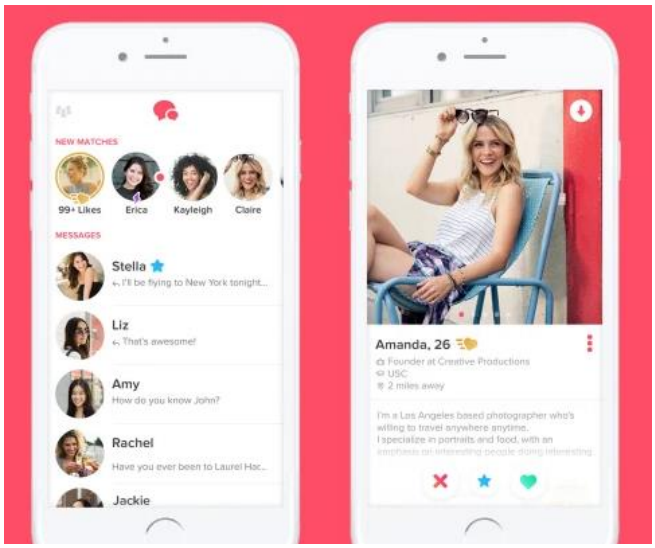
2.3.1. Sustavi preporuke aktivnosti na temelju vremenskih uvjeta

Clima Cell Weather Assistant je mobilna aplikacija koja uz normalne funkcionalnosti aplikacije za uvid vremenske prognoze korisniku pomaže u odabiru savršenog trenutka sa savršenim uvjetima za željenu aktivnost u prirodi. Korisnik može izabrati željene aktivnosti i aplikacija će prikazati najbolje vrijeme za tu aktivnost ovisno o vremenskim uvjetima kao što su vlaga, brzina vjetar, uv zračenje, vidljivost i sl. Iako aplikacija završnog rada i ova aplikacija oboje imaju korisnički profil i primjenjuju sustav preporučivanja koji ovisi o vremenskim uvjetima, ova aplikacija prati detaljno vrijeme i preporučuje koliko je ono prikladno aktivnosti dok aplikacija završnog rada samo prati je li vrijeme ne prikladno pojedinoj aktivnosti. Slika 2.1. prikazuje izgled aplikacije Clima Cell Weather Assistant.

Slika 2.1. Prikaz aplikacije Clima Cell Weather Assistant

2.3.2. Sustavi preporuke aktivnosti na temelju geolokacije

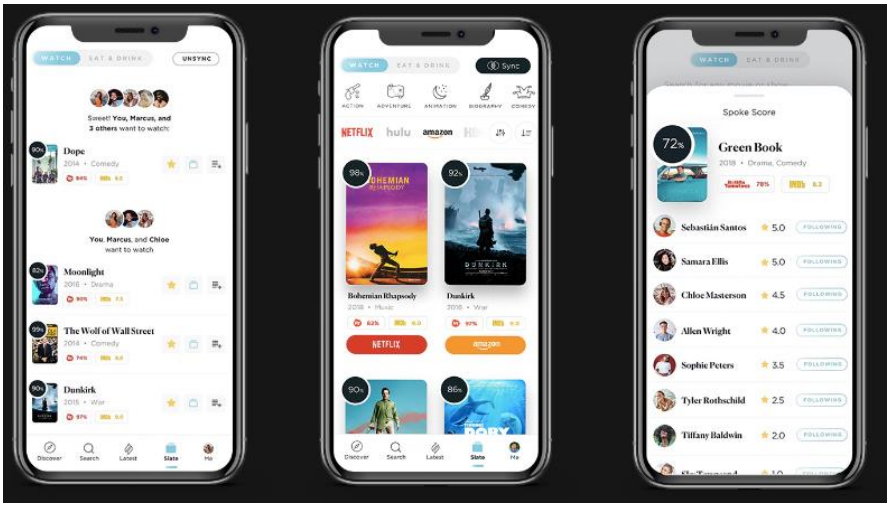
Tinder je popularna aplikacija za upoznavanje za mlade. Korisnik stvara vlastiti korisnički račun s opisom i slikom. Aplikacija preporučuje druge korisnike na zadanoj udaljenosti i na temelju eksplicitne povratne veze informacija korisnik bira sebi interesantne osobe. Ako dva korisnika potvrde jedan za drugog da su zainteresirani omogućuje se mogućnost slanja poruka. Slika 2.2. prikazuje izgled aplikacije Tinder.



Slika 2.2. Izgled aplikacije Tinder

2.3.3. Sustavi preporuke aktivnosti na temelju zajednice

The spoke je aplikacija za I-Phone i I-Pad uređaje koja služi za preporučivanje raznih serija, filmova i restorana. Korisnik može odabrati između više različitih kategorija kao npr. „podcijenjeni filmovi" i samostalno ocjenjivati ih. Oni koji su bili pozitivno ocjenjeni mogu biti pohranjeni za kasnije. Ovisno o korisnikovim ocijenama filmova ili restorana aplikacija može dalje preporučivati nove slične aktivnosti. Aplikacija također ima mogućnost sinkronizacije s prijateljima gdje njihovi prosječni odabiri utječu na korisnikove daljnje preporuke. Slika 2.3. prikazuje izgled aplikacije The Spoke.



Slika 2.3. Izgled aplikacije The Spoke

3. ANALIZA ZAHTJEVA I MODELIRANJE MOBILNE APLIKACIJE

U ovom poglavlju opisat će se funkcijski i ne funkcijski zahtjevi pri modeliranju mobilne aplikacije. Također će se opisati model mobilne aplikacije i algoritam preporučivanja.

3.1. Analiza zahtjeva za mobilnu aplikaciju

U ovom potpoglavlju opisani su funkcijski i ne funkcijski zahtjevi po kojima je dizajnirana aplikacija.

3.1.1. Funkcijski zahtjevi

Funkcijski zahtjevi su značajke ili funkcionalnosti mobilne aplikacije koje opisuju njen način rada kao i način na koji će je korisnik upotrebljavati. Funkcijski zahtjevi se određuju prije izrade same aplikacije kako bi bilo točno definirano ponašanje i svrha namjene aplikacije. Oni ovise o potrebama korisnika, kao i zahtjevima samog projekta.

Funkcijski zahtjevi ove aplikacije su:

- Učitavanje korisnikovih podataka i njihovo pohranjivanje
- Učitavanje korisnikove trenutne lokacije kao i vremenskih uvjeta na toj lokaciji
- Omogućiti korisniku davanje ocjena i slika pojedinim lokacijama
- Preporuka aktivnosti korisniku na temelju vremenskih uvjeta, ocijene lokacije i korisnikovih preferenci.

3.1.2. Nefunkcijski zahtjevi

Nefunkcijski zahtjevi aplikacije su svojstva koja definiraju potrebnu razinu kvalitete aplikacije tj. njenih pozitivnih atributa. Njima su definirani okvirni načini uz pomoć kojih je moguće implementirati pojedino ponašanje aplikacije na najprimjereniji način ovisno o danim zahtjevima. Oni su također definirani unaprijed ovisno o potrebama korisnika i omogućuju bolje korisničko iskustvo, manji trošak izrade, mogućnost nadogradnje ili održavanja aplikacije i sl. U ovom radu nefunkcijski zahtjevi su određeni od strane autora.

Upotrebljivost tj. lakoća korištenja aplikacije jedno je od naj bitnijih svojstava svake aplikacije. Aplikacija mora imati jednostavno korisničko sučelje bez ne potrebnih dodataka kako ne bi zbunila korisnika. Također aplikacija bi trebala biti na što više rasprostranjenom jeziku kako bi što više ljudi moglo koristiti aplikaciju. Ova aplikacija je izrađena s obzirom na unaprijed navedena načela.

Sigurnost aplikacije se primarno odnosi na sigurnost korisnikovih podataka. Korisnik za korištenje aplikacije mora samo dati minimum podataka, svoje ime te definirati svoje preferirane aktivnosti. A ako korisnik želi pohraniti dodatne podatke, oni će biti osigurani od strane poslužitelja programskih alata koje aplikacija koristi. Također je bitno napomenuti kako korisnikova lokacija neće biti zatražena ako on to nije odobrio. Aplikacija ne pohranjuje povijest korisnikovih lokacija. Programski alati koji se koriste u svrhu dohvaćanja lokacije i vremenskih uvjeta tj. Places API i Open Weather API su sigurni i korisnik vj. već koristi neku od drugih usluga tih tvrtki što može povećati osjećaj sigurnosti kod korisnika.

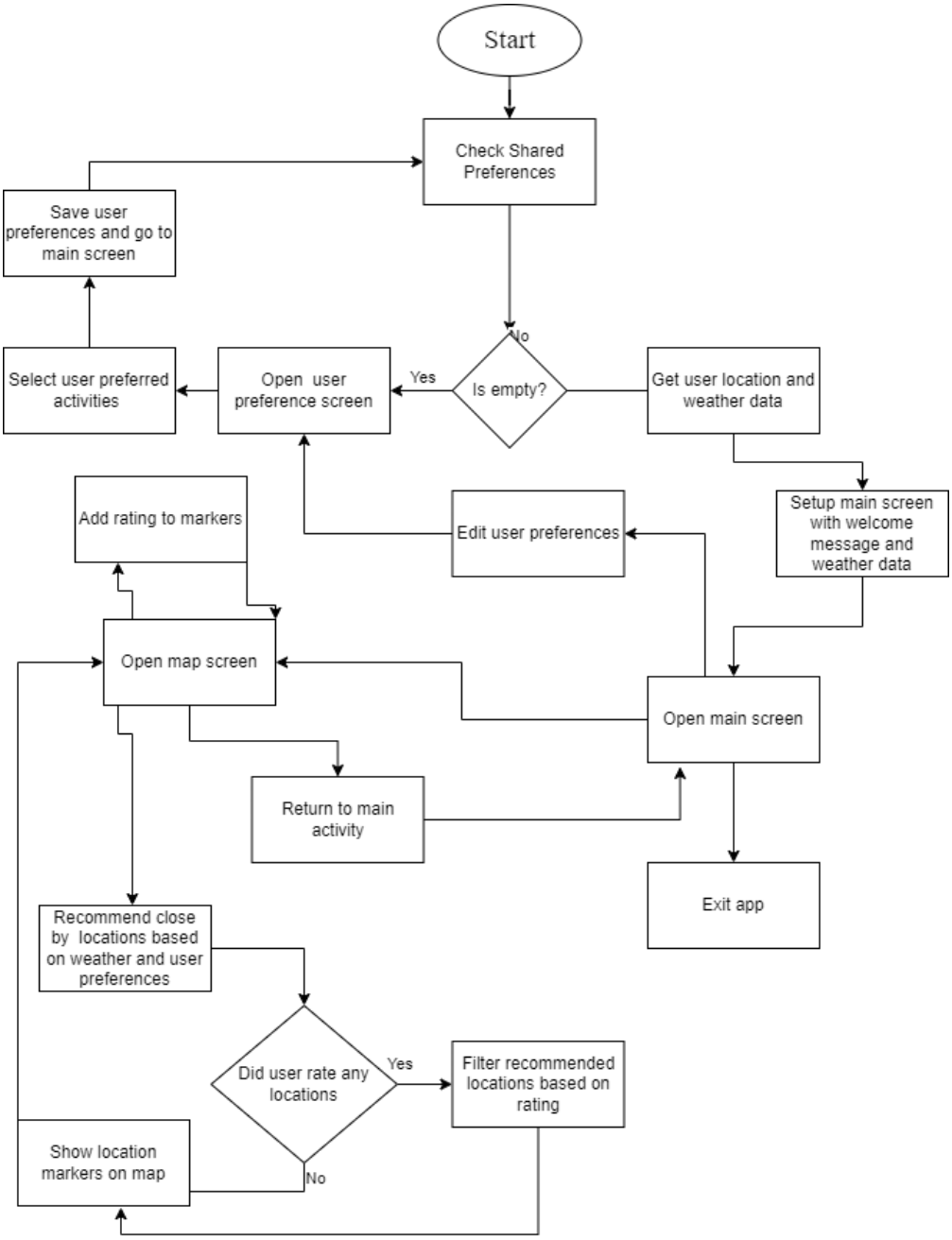
Skalabilnost se odnosi na sposobnost aplikacije da rukuje sa sve više i više korisničkih podataka, korisnika i sl. Dodavanjem baze podataka može se značajno smanjiti vrijeme dohvaćanja korisnikovih podataka kao i dovoljni volumen za spremanje tih podataka. Vrlo je jednostavno podesiti koje se aktivnosti mogu preporučiti korisniku. Ova aplikacija preporučuje aktivnosti vezane za prirodu, sport i zabavu u prirodi, ali dodavanjem dodatnih kategorija moguće je povećati potencijalan broj korisnika.

Performanse se odnose na brzinu kojom aplikacija izvršava svoje funkcionalnosti. Nije prihvatljivo dopustiti da korisnik čeka više od nekoliko sekundi, ovo bi kvarilo korisničko iskustvo i potaklo korisnika upotrebiti drugu aplikaciju. Brzina ove aplikacije može ovisiti o brzini dohvaćanja lokacije ako se korisnik nalazi u zgradi, ali to je generalno zanemarivo jer je brzina odziva i dalje unutar nekoliko sekundi.

3.2. Model mobilne aplikacije

Model aplikacije je prikazan kroz dijagram tijeka koji prikazuje tijek rada aplikacije. Dijagram tijeka prikazan je na slici 3.1. Uključivanjem aplikacije prvo se provjerava postoji li podatak o korisnikovim preferiranim aktivnostima tj. je li korisnik otvara aplikaciju prvi puta. Ako je to slučaj otvara se zaslon gdje korisnik može ispuniti svoje ime i odabrati vrste aktivnosti koje mu odgovaraju. Nakon spremanja preferenca korisnik se vrača na glavni zaslon gdje se prvo dohvaća korisnikova lokacija i vremenski uvjeti na toj lokaciji nakon čega ima izbor želi li promijeniti svoje postavke aktivnosti, izaći iz aplikacije ili dobiti preporuku aktivnosti. Ako odluči dobiti preporuku otvorit će se zaslon s kartom. Na karti će se pojaviti markeri koji predstavljaju preporučene lokacije. Klikom na markere moguće je vidjeti svojstva lokacije te se pojavljuju dva gumba koja korisnika vode do aplikacije za karte Google Maps kako bi našli put do lokacije ili do stranice s podacima o lokaciji u web pregledniku. Klikom na svojstva lokacije korisnik dobiva prozor za ocjenjivanje lokacije. Markeri će biti obojeni po korisnikovoj ocjeni ako je korisnik ocijenio neke

od lokacija. tj. loše ocijenjene lokacije bit će prikazane zeleno, a dobro ocijenjene će biti prikazane žutom bojom. Neutralno ocijenjene i ne ocijenjene lokacije su crvene boje. Lokacije se mogu ocijeniti odmah nakon prve preporuke.



Slika 3.1. Dijagram tijeka rada mobilne aplikacije

3.2.1. Algoritam preporučivanja

Temeljni zadatak aplikacije koju ovaj rad opisuje je preporuka aktivnosti. Ovo se u aplikaciji postiže korištenjem više parametara kao što su geolokacija korisnika, trenutni vremenski uvjeti na toj geolokaciji, korisnikove preference koje je unio u aplikaciju pri početku korištenja kao i korisnikove ocjene pojedinih lokacija nakon što ih je posjetio. Preporuke koje korisnik dobiva su

ubiti lokacije prikazane na karti. A moguće kategorije aktivnosti su tipovi lokacija u Places API-u tvrtke Google. Korišteni Google Maps i Google Place API kao i ostale korištene tehnologije i njihova primjena će biti opisani u daljnjim poglavljima. Glavni dio sustava za preporuke je korisnikov odabir voljenih aktivnosti. Ovdje korisnik može birati između četiri kategorije koje su cjeline od više sličnih aktivnosti. Kao unaprijed spomenuto jer se za prikaz kategorija aktivnosti koristi Place API postoji ograničen broj tipova aktivnosti koje sustav može preporučiti. Prema [19] postoji oko devedeset šest različitih tipova lokacija prema dokumentaciji Place API-a. S obzirom na to da je svrha ovog rada izraditi aplikaciju koja preporučuje aktivnosti vezane uz prirodu, rekreaciju i ima za zadataka čovjeka motivirati napustiti dom kako bi sudjelovao u njemu informativnim, zabavnim i oplemenjujućim aktivnostima samo šesnaest tipova je odabrano za preporučivanje. Tih šesnaest tipova lokacija je dalje spojeno u četiri cjeline, sport, kultura, životinje i kampiranje. Ovo je dovoljno kako bih se opisao rad aplikacije, ali lako je moguće dodati nove kategorije tj. tipove aktivnosti jer je aplikacija skalabilna po tom pitanju.

Idući parametar po kojem se vrši preporučivanje je geolokacija. Nakon što se je korisnika pitalo za dopuštenje dohvaća se geolokacija korisnika putem servisa FusedLocationProvider. Korisnikova lokacija se tada koristi kako bih se dohvatile trenutni hidro-meteorološki uvjeti na toj lokaciji. Ovi uvjeti se dohvaćaju od servisa Open Weather API tvrtke Open Weather. Primljeni podaci o vremenu pružaju mnogobrojne informacije o trenutnim vremenskim uvjetima no sustav preporuka se obazire samo na *weather ID*. Kao opisano u dokumentaciji [8] *weather ID* dijeli trenutno stanje vremenskih uvjeta na devet grubih kategorija po kojima je moguće procijeniti stanje vremena. Ovo će poslužiti aplikaciji kako bi zaključila jesu li vani trenutno dobri ili loši vremenski uvjeti. U loše vremenske uvjete bi spadala kiša, grmljavinsko nevrijeme, magla, snijeg i sl.

Preporuke na temelju geolokacije se vrše pomoću već unaprijed dohvaćene geolokacije kao i korisnikom odabrana udaljenost od lokacije. Korisnik ima mogućnost prije dobivanja preporuke odabrati koliko daleko želi da preporučene aktivnosti budu od njegove trenutne lokacije u kilometrima. Nakon što su određeni početni parametri korisnikove geolokacije, njegove željene udaljenosti od preporučenih lokacija, vremenskih uvjeta i korisnikovim odabranim kategorijama moguće je dati preporuku.

Sustav za preporuku prvo provjerava koje cjeline je korisnik odabrao te stvara listu svih tipova aktivnosti koja pripadaju pojedinoj odabranoj cjelini. Ako korisnik nije odabrao niti jednu cjelinu, ova lista će se sastojati od općenito zabavnih aktivnosti odabranih od strane autora. Cilj ovih općih

aktivnosti je naći aktivnosti koje su općenito smatrane zabavnim ali nemaju pre velike zahtjeve kao što su nedostupnost ili potrebna oprema kako bih se neko njima bavio. Nakon toga sustav provjerava koji su trenutni vremenski uvjeti. Ako su loši vremenski uvjeti i korisnik nije odabrao niti jednu cjelinu, korisniku će se preporučiti sve aktivnosti koje se mogu raditi po lošim vremenskim uvjetima. Ako je korisnik odabrao jednu ili više skupina, provjerava se koje od aktivnosti s korisnikove liste se mogu odrađivati po lošem vremenu i samo one su preporučene. Ako niti jedna od korisnikovih aktivnosti nije primjerena za odrađivanje tijekom lošeg vremena korisniku se preporučuje opća lista svih aktivnosti koje se mogu odrađivati po lošem vremenu. Lokacije koje se preporučuju za vrijeme loših vremenskih uvjeta su rekreativne aktivnosti koje ne ovise o vremenu kao što su odlazak u teretanu ili kuglanje kao i aktivnosti koje potiču kasniji izlazak u prirodu tj. bavljenje sportom. To su lokacije kao muzeji i trgovine za knjige gdje korisnik može steći interes u pojedine aktivnosti ili naučiti više o njima. U ovu kategoriju spadaju i trgovine s opremom kao što su trgovine obućom ili trgovine biciklima kako bi se korisnik mogao opremiti za aktivnosti u prirodi.

Nakon što je korisnik dobio svoju preporuku i možda posjetio neku od lokacija, korisnik može ostaviti svoju ocjenu za pojedinu lokaciju i dati komentar. Oboje će pri sljedećoj preporuci pisati u opisu lokacije. Ako je korisnik pozitivno ocijenio lokaciju marker na karti te lokacije bit će žute boje. Ako lokacija nije ocijenjena ili je ocijenjena neutralno imat će uobičajenu crvenu boju. A ako je korisnik loše ocijenio lokaciju marker na karti će imati zelenu boju. Crvena boja je odabrana kao neutralna jer je to standardna boja markera koje koristi Google Maps i to je boja s kojom je većina korisnika upoznata. Žuta je izabrana jer se povezuje s osjećajem sreće, a zelena je odabrana jer se može asocira s osjećajem gađenja ili bolesti. Bilo bi prikladnije odabrati sivu, crnu ili neku sličnu boju no one nisu podržane u skupu nijansi koje pruža Google Maps.

4. PROGRAMSKO RJEŠENJE MOBILNE APLIKACIJE

U ovom poglavlju opisani su programski alati, programski jezici i razvojna okolina koji su se koristili u izradi aplikacije. Objašnjena je implementacija korisničkog sučelja i njegove prilagodbe tijekom rada aplikacije. Opisana je implementacija načina pohrane podataka u aplikaciji, dohvaćanje korisnikove lokacije i vremenskih uvjeta. Također je opisana implementacija preporuke aktivnosti korisniku.

4.1. Korištene programske tehnologije, jezici i razvojne okoline

U ovom potpoglavlju opisani su programski jezici i okolina u kojima je pisana aplikacija. Također su opisane tehnologije i programski alati kojima su izvedene funkcionalnosti aplikacije.

4.1.1. Kotlin

Prema [11] Kotlin je programski jezik koji je razvila tvrtka JetBrains 2010. Kotlin izvorni kod je od njegovog početka bio slobodno dostupan i besplatan. Kotlin pruža mogućnosti objektno orijentiranog kao i funkcijskog programiranja. Može se koristiti za izradu aplikacija na strani poslužitelja i na strani korisnika za web ili za android. Temelji se na programskom jeziku Java. Cilj iza Kotlina je bio napraviti novi sažetiji programski jezik koji usavršuje neke od Javinih nedostataka. Kotlin je u potpunosti interoperabilan s Javom te mnoga razvojna okruženja pružaju pretvornike iz jednog u drugi jezik. Zbog navedenih prednosti popularnost Kotlina kao programskog jezika je rasla značajno zadnjih nekoliko godina posebno u području mobilnog razvoja te je iz tog razloga ova aplikacija pisana u Kotlinu.

4.1.2. Android Studio

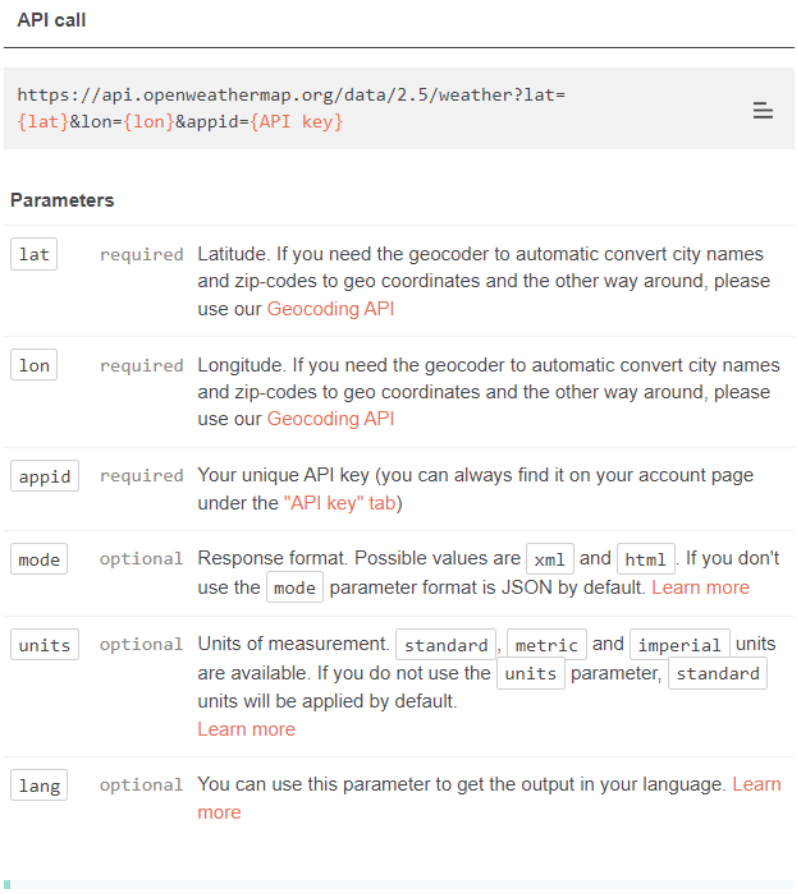
Prema [12] Android Studio je službeno integrirano razvojno okruženje za android aplikacije. Temelji se na uređivaču koda IntelliJ IDEA. Pruža mnoge pogodnosti kao što su nadopunjavanje koda, savjetovanja pri refaktoriranju koda, robusne alate za testiranje kao što su LogCat. Android Studio emulator simulira stvarne mobilne uređaje na računalu tako da je moguće testirati aplikacije na njima. Moguće je spojiti i stvarni mobilni uređaj na računalo i preko njega testirati aplikacije pomoću Android Studia ali potrebno je prvo uključiti tzv. developer mode na mobilnom uređaju. Android Studio podržava Javu, Kotlin i C++ što ga čini svestranim i efikasnim okruženjem za razvoj aplikacija.

4.1.3. Fused Location Provider API

Prema [13] Fused Location Provider je lokacijski API u Google Play servisima. On omogućuje jednostavan i energetski efikasan način pristupu trenutnoj lokaciji uređaja. Pomoću Fused Location Provider moguće je odrediti željenu kvalitetu signala i on će pokušati iskoristiti onu mrežu koja daje najbolje rezultate npr. GPS ili WiFI. Također pruža mogućnost periodičkih osvježavanja trenutne lokacije korisnika. U radu se koristio samo u najosnovnijem načinu rada tj. get Last Known Location.

4.1.4. Open Weather API

Open Weather API je besplatni *online* servis koji pruža podatke o trenutnoj vremenskoj prognozi u više od 200 000 gradova u svijetu. Open Weather skuplja podatke o vremenu od različitih izvora kao što su radari, sateliti, vremenske postaje i sl. Podaci o vremenu se zatražuju pomoću API poziva. Na slici 4.1. vidi se primjer takvog poziva kao i njegovi parametri. Obvezno je poslati podatke o lokaciji za koju se želi saznati vrijeme, kao i API ključ koji je potreban radi identifikacije korisnika. Zatraženi podaci o vremenu mogu biti izdani u HTML, XML ili JSON formatu. Na slici 4.2. se vidi primjer u JSON formatu.



Slika 4.1. Izgleda poziva prema Open Weather API i parametri poziva

JSON

JSON format API response example


```
{
  "coord": {
    "lon": 10.99,
    "lat": 44.34
  },
  "weather": [
    {
      "id": 501,
      "main": "Rain",
      "description": "moderate rain",
      "icon": "10d"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 298.48,
    "feels_like": 298.74,
    "temp_min": 297.56,
    "temp_max": 300.05,
    "pressure": 1015,
    "humidity": 64,
    "sea_level": 1015,
    "grnd_level": 933
  },
```


Slika 4.2. Odgovor API poziva, povratna JSON datoteka

4.1.5. Places API

Places API je servis koji omogućuje pristup informacijama o traženim lokacijama putem HTTP zahtjeva. Places API vraća formatirane podatke o lokacijama i slike poslovanja, geografskih lokacija ili čestih točaka interesa. Places API omogućava rezultate pretraživanja od različitih vrsta korisničkih zahtjeva, omogućava automatsko dovršavanje riječi tijekom pretraživanja ili dodavanje slika visoke kvalitete lokacijama u korisničkoj aplikaciji. Places API prihvaća standardne URL zahtjeve s određenom krajnjom točkom usluge, kao što je /place ili /photo. Informacije o lokaciji se vračaju u JSON ili XML formatu ovisno o korisnikovom zahtjevu. Na kraju zahtjeva je potreban API ključ. Na slici 4.3 vidi se izgled API poziva prema Places API.

```
https://maps.googleapis.com/maps/api/place/details/json
  ?place_id=ChIJrTLr-GyuEmsRBfy61i59si0
  &fields=address_components
  &key=YOUR_API_KEY
```

Slika 4.3. Izgleda poziva prema Places API

4.1.6. XML

XML je opisni jezik koji se koristi za strukturiranje i spremanje podataka tako da je lako čitljiv čovjeku i računalu. XML definira način kodiranja dokumenata pomoću tagova. S obzirom na to da nisu definirane točne oznake tagova korisnik može definirati svoje. Ovakav univerzalan način kodiranja omogućuje korištenje između više različitih operacijskih sustava ili programskih jezika. XML se najčešće koristi za prijenos podataka ili pohranu podataka. U ovom radu XML je korišten kao opisni jezik za grafičke elemente aplikacije.

4.1.7. Shared Preferences

Shared Preferences je mehanizam za pohranu i dohvaćanje male količine osnovnih tipova podataka kao što su *int, string*, itd. Najčešće se koristi za pohranu korisnikovih preferenci, postavki i sl. Podaci se pohranjuju kao parovi ključ vrijednost. Potreban je specifičan ključ kako bi se dohvatila određena vrijednost. U ovom radu ovaj mehanizam se koristio za pohranu korisnikovih preferenci kao i njegovog imena. Preference su skup *string* elemenata tako da je bilo potrebno sve preference spojiti u jedan *string* odvojen zarezima kako bi mogle biti pohranjene.

4.1.8. Baza podataka Room

Room je lokalna relacijska baza podataka za android. Temelji se na SQLite-u i koristi se pohranu podataka i očuvanje postojanja podataka van rada aplikacije. Room ima tri glavne komponente : klasa baze podataka koja sadrži bazu podataka, podatkovni entiteti koji predstavljaju tablice u bazi podataka i *Data Access object* tzv. DAO koji pruža metode kojima aplikacija može slati SQL upite bazi podataka . Klasa baze podataka pruža aplikaciji instancu objekta za pristup podacima DAO i njime aplikacija može pristupiti podatku u obliku instance podatkovnih entiteta. Room može samo pohranjivati osnovne tipove podataka, svi ostali tipovi moraju biti pretvoreni u podržane tipove. Room pruža mogućnost korištenja korutina čime omogućuje asinkrono izvođenje instrukcija. Ova baza podatak se koristi u aplikaciji kako bi pohranjivala korisnikove ocjene lokacija.

4.2. Programsko rješenje na korisničkoj strani

4.2.1. Implementacija korisničkog sučelja

U razvojnom okruženju Android Studio svaki zaslon tj. svaka aktivnost ima svoju XML datoteku koja opisuje raspored elemenata te aktivnosti. Unutar android studija postoje dva načina uređivati korisničko sučelje korištenjem grafičkog sučelja XML datoteke ili direktnim unosom tagova u XML datoteku. Korištenjem grafičkog sučelja moguće je povlačiti elemente na mjesto kuda

moraju ići i to je mnogo brže nego pisati u XML datoteku. Unatoč tome postavljanje parametara pojedinih elemenata je lakše raditi unutar datoteke. Sve promjene u XML datoteci se trenutno ažuriraju na grafičkom sučelju i obratno.

Prvi element u datoteci je *layout*, on određuje kako će se ostali elementi unutar njega rasporediti kada se pokrene aplikacija. U aplikaciji je korišten *constraint layout* čija implementacija je prikazana na slici 4.4. *Constraint layout* nalaže da svaki element unutar njega mora biti vezan za drugi element ili za rub ekrana. Te veze imaju konstantnu dužinu koja je zabilježena u svojstvima pojedinih elemenata. Svojstva elementa mogu biti opisni izgledu samog elementa ili mogu biti funkcionalni. Svaki element bi trebao imati svoj vlastiti identifikacijski kod u svojstvima tj. *id* vrijednost kako bi mu se moglo pristupiti iz programskog dijela aktivnosti. Tako je moguće mijenjati svojstva korisničkog sučelja iz programskog djela aktivnosti koristeći funkciju *findViewById*. U ovoj funkciji definira se tip elementa grafičkog sučelja i njegov identifikator i tako se može pristupiti njegovim svojstvima kao tekst unutar elementa. *FindViewById* također može definirati ponašanje elementa npr. Što se dogodi kada je element pritisnut. Pri stvaranju novih grafičkih elemenata android studio generira samostalno osnovne parametre elementa tako da nije potrebno svaki put pisati isti kod.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
```

Slika 4.4. Prikaz layout elementa u XML datoteci

Na slici 4.5. može se vidjeti implementaciju *text view* elementa koji se nalazi na glavnom zaslonu. Koristi se za prikaz trenutne temperature korisniku. *FindviewById* funkcijom se pristupa parametru *text* i onda se mijenja na željeni vrijednost. U aplikaciji *text view* služi za davanje obavijesti o vremenu i lokaciji korisniku kao i komunikacija s korisnikom u smislu objašnjavanje što mora napraviti.

```
<TextView
    android:id="@+id/tempShow"
    android:layout_width="146dp"
    android:layout_height="22dp"
    android:layout_marginBottom="20dp"
    android:text=""
    android:textAlignment="center"
    app:layout_constraintBottom_toTopOf="@+id/weatherShow"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.528"
    app:layout_constraintStart_toStartOf="parent" />
```

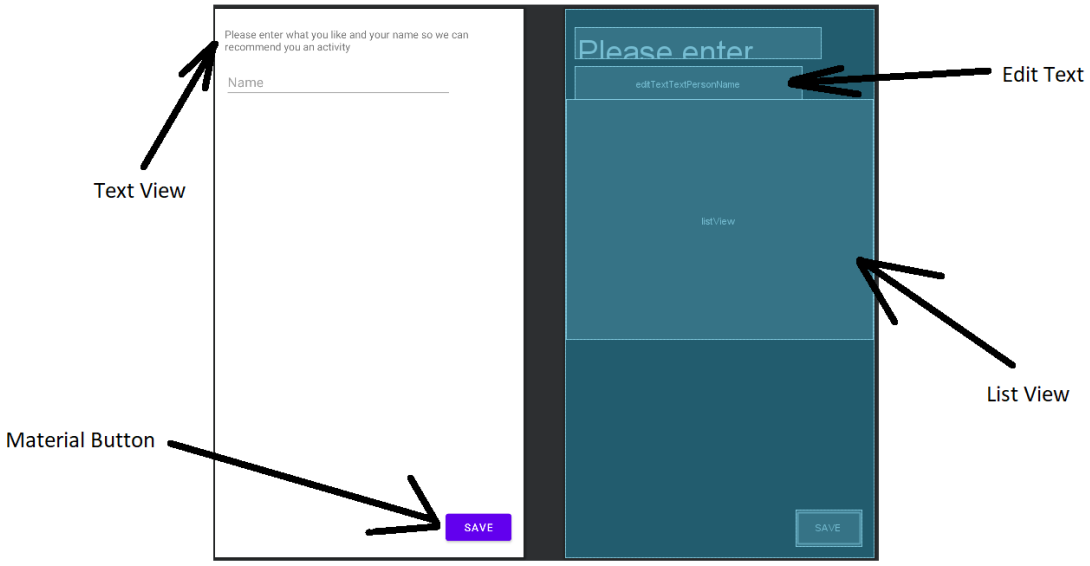
Slika 4.5. Prikaz implementacije textView elementa

Korištene su dvije vrste gumba *material button* i *image button*. Oboje imaju sličnu primjenu, omogućuju korisniku direktno komunicirati s aplikacijom. Razlika je u tome što je moguće pozadinu image *button*-a promijeniti u sliku a *material button* samo u boju. Na slici 4.7. se vidi glavna aktivnost. Ovdje su korišteni jedan *material button* koji kada pritisnut korisniku daje preporuke aktivnosti i jedan *image button* za povratak na aktivnost za odabir kategorija. *Image button* samim svojim izgledom asocira korisnika na njegovu ulogu u ovom slučaju odlazak na postavke aplikacije. Stoga pruža intuitivan način korištenja aplikacije.

Na glavnoj aktivnosti je također korišten *seek bar*. Ovo je jednostavni element koji korisniku služi za unos cijelog broja u granicama koji je programer odredio. U aplikaciji *seek bar* određuje u kojem radiusu od trenutne lokacije korisnika će mu biti preporučene aktivnosti. *Image view* se koristi za prikaz slika. Na glavnom zaslonu se je koristio za prikaz slike koja predstavlja trenutno vrijeme na lokaciji korisnika.

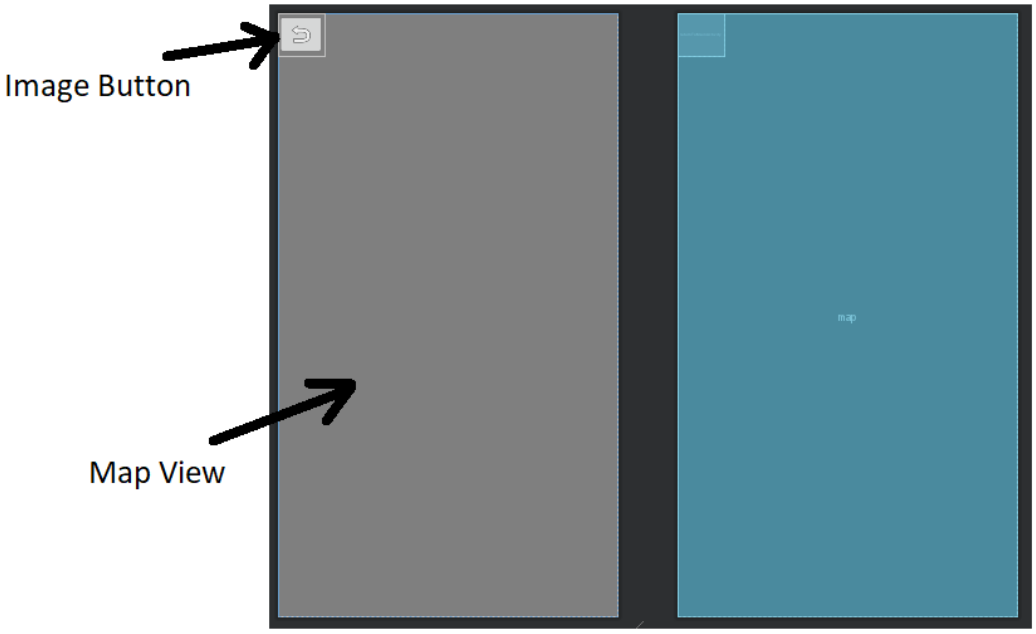
Slika 4.6. Prikaz XML rasporeda aktivnosti MainActivity

Na slici 4.7 prikazan je raspored elemenata na zaslonu za odabir kategorija aktivnosti. *Edit text* se ovdje koristi za unos podatak u aplikaciju, specifično korisnikovo ime pri prvom pokretanju aplikacije. Potrebno je pripaziti da korisnik zna što se očekuje od njega tj. da zna što treba unijeti. Zbog toga postoji svojstvo *hint* koje sivim slovima korisniku pokaže što bi trebao napisati. Sljedeći korišteni element je *list view*. Sastoji se od manjih stavki. Dizajn tih manjih osnovnih stavki je zasebno definiran u njihovoj XML datoteci. *List view* omogućuje prikaz koji se može pomicati unutar okvira *list view*-a, što znači da kada popis stavki premaši raspoloživi prostor na zaslonu, korisnik može pomicati gore i dolje kako bi vidio više stavki. Ovo je korišteno u aplikaciji za prikaz aktivnosti od kojih je korisnik mogao birati željene. *CheckBox* element ima ulogu spremati svoje stanje i mijenjati ga kada korisnik klikne na njega iz označenog u ne označeno i obratno. Korišten je kao oznaka za odabranu kategoriju u pojedinim stavkama *list view*-a.



Slika 4.7. Prikaz XML rasporeda aktivnosti UserProfileActivity

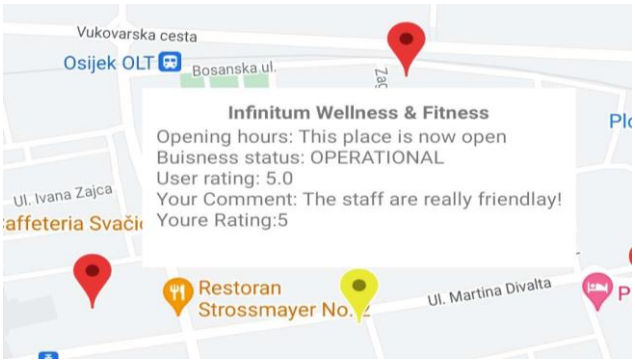
Slika 4.8. prikazuje aktivnost u kojoj se prikazuje karta i njene značajke. Najbitniji korišteni element je *MapView*. Uz pomoć ovog elementa se prikazuje karta. Moguće je fokusirati na kartu ili postavljati markere na željene lokacije. Postavljanjem tih markera se korisniku preporučuju aktivnosti.



Slika 4.8. Prikaz XML rasporeda aktivnosti GoogleMapsActivity

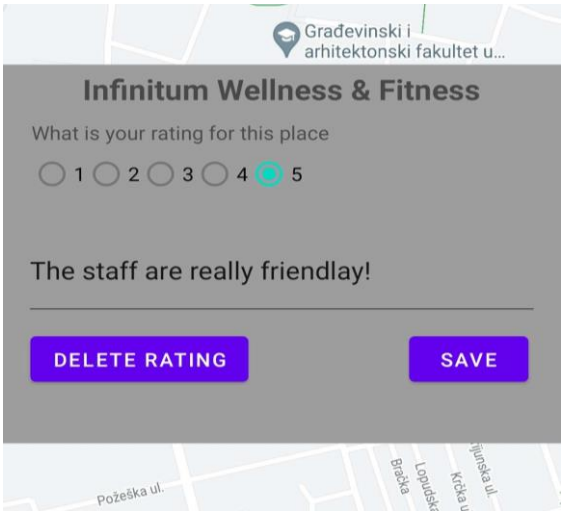
Markeri se koriste za oznaku lokacija na karti. Mogu im se podesiti naslov, opis i ikona. Moguće je napraviti prilagođenu novu verziju markera ili samo promijeniti boju. Promjena boje je korištena za prikaz korisnikove ocijene lokacije. Klikom na marker dobiva se *InfoWindow* element u kojemu

piše opis samog markera kao i ocjena i komentar korisnika ako postoji. Na slici 4.9. prikazan je marker i njegov pripadajući informacijski prozor. Za ovaj rad izrađen je prilagođeni informacijski prozor jer standardni nije bio prikladan za prikaz svih informacija o markeru.



Slika 4.9. Prikaz informacijskog prozora markera

Prozor za ocjenjivanje je skočni prozor koji se otvara kada korisnik klikne na informacijski prozor. Prozor za ocjenjivanje prikazan je na slici 4.10. Informacijski prozor markera i prozor ocjenjivanja lokacije se oboje gase pritiskom van njihovih okvira. Sadržaj oba prozora ovisi o markeru kojem pripadaju. Kako bi se ostvarila funkcionalnost prozora za ocjenjivanje koristi se *radio group* element. Ovo je složeni element koji se sastoji od više *radio button* elemenata. *Radio button* elementi su po funkcionalnosti slični *check box*-u, ali samo jedan od *radio button* elemenata može biti pritisnut odjednom. Ovaj element omogućuje dobiti točno jedan određeni ulaz od korisnika. Ovo je korišteno za korisnikov odabir ocijene lokacije.



Slika 4.10. Prikaz prozora za ocjenjivanje

4.2.2. Prilagodba korisničkog sučelja

Na slici 4.11. vidi se funkciju *setWeatherImage*. Njena uloga je mijenjanje slike na glavnom zaslonu. Funkcija provjerava je li dohvaćen identifikator vremena tj. je li prazna varijabla koja ga sadrži. Ako nije provjerava je li identifikator unutar raspona brojeva koji predstavlja pojedinu vremensku priliku. Ovisno u kojem je rasponu bit će prikazana druga slika na glavni ekran. Metodom *findViewById* i identifikatorom mijenja se koja slika se prikazuje na *image view* elementu.

```
private fun setWeatherImage(){
    if(weatherid.isNotBlank())
    {


        if(weatherid.toInt()>=200&&weatherid.toInt()<240)
            findViewById<ImageView>(R.id.weatherShow).setImageResource(R.drawable.thunderstorm)

        if(weatherid.toInt()>=700&&weatherid.toInt()<800)
            findViewById<ImageView>(R.id.weatherShow).setImageResource(R.drawable.fog)

        if(weatherid.toInt()>=300&&weatherid.toInt()<330)
            findViewById<ImageView>(R.id.weatherShow).setImageResource(R.drawable.drizzle)
```

Slika 4.11. Prilagodba imageView elementa vremenskim uvjetima

Još jedan dio korisničkog sučelja koji se prilagođuje tijekom rada aplikacije je *text view* koji prikazuje pozdravnu poruku prilikom otvaranja glavne aktivnosti. Poruka je definirana unaprijed i naknadno se kao argument dodaje ime koje je korisnik unio. Poruka se pokazuje samo ako je korisnik unio svoje ime i prikazana je na slici 4.12.

```
if (username != "") {
    findViewById<TextView>(R.id.welcomeTxT).text =
        "Welcome " + username + ",\nready for some new activities?"
```

Slika 4.12. Prilagodba poruke na početnom zaslonu

4.3. Programsko rješenje na poslužiteljevoj strani

U ovom potpoglavlju bit će opisana implementacija programskih rješenja na strani poslužitelja korištenih u aplikaciji.

4.3.1. Dohvaćanje lokacije korisnika i trenutnog meteorološkog stanja

Pri svakom otvaranju glavnog zaslona aplikacija provjerava korisnikovu lokaciju kao i vremenske uvjete na toj lokaciji. Ti se podaci prikazuju na korisničkom sučelju zajedno s porukom korisniku. Dohvaćanje lokacije obavlja funkcija *fetchLocation*, dok dohvaćanje vremenskih uvjeta obavlja unutarnja klasa *weatherTask*. Dohvaćanje lokacije i vremena prikazano je na slici 4.13. Funkcije za dohvaćanje lokacije i vremena se odvijaju asinkrono tj. događaju se istodobno. Metoda za dohvaćanje vremena ovisi o podacima koje dohvaća *fetchLocation* i stoga je bitno da *fetchLocation* funkcija završi s dohvaćanjem podataka prije nego što se krene dohvaćati vrijeme. Ovo se postiže korištenjem korutina i funkcije *delay*. Korutine su alati za upravljanje paralelnim operacijama unutar Android okruženja. Funkcijom *GlobalScope.launch* poziva se nova korutina u kojoj je moguće koristiti funkciju *delay* koja će nakon što se pokrene *fetchLocation* onemogućiti pokretanje ostatka kod dok ne prođe zadanih 500 milisekundi. Sljedeće se izvodi *fetchLocation.*

```
if (username != "") {
    findViewById<TextView>(R.id.welcomeTxT).text =
        "Welcome " + username + ",\nready for some new activities?"

    GlobalScope.launch(Dispatchers.Main) {  this: CoroutineScope
        fetchLocation()
        delay( timeMillis: 500)
        Log.d( tag: "OOOOOOOOOO",  msg: "ASYNC:" + userLocation.latitude)
        if (userLocation != defaultLocation) {
            var adress = geocoder.getFromLocation(userLocation.latitude, userLocation.longitude,  maxResults: 1)
            var d = adress?.get(0)?.locality!!.lowercase() + "," + adress?.get(0)?.countryCode!!.lowercase()

            weatherTask(d).execute()
            delay( timeMillis: 500)
            Log.d( tag: "OOOOOOOOOO", msg: "weatherid:"+weatherid)
            setWeatherImage()
        }
    }
}
else{
    startActivity(intent)
}
```

Slika 4.13. Prikaz dohvaćanja vremenski uvjeta i lokacije korisnika

Kako bi se dohvatila korisnikova lokacija unutar funkcije *fetchLocation* koristi se *FusedLocationProviderClient*. Aplikacija prvo provjerava unutar manifest datoteke je li definirano da aplikacija ima dozvolu pristupiti finoj i gruboj lokaciji korisnika. Fina lokacija omogućuje pristup preciznoj lokaciji uređaja i dobiva se putem GPS poslužitelja. Gruba lokacija omogućuje dohvaćanje manje precizne lokacije preko lokalnog rutera. Ako dozvola nije pronađena pita se korisnika za dozvolu dohvaćanja lokacije. Nakon što se dobije dozvola koristi se *FusedLocationProvider* da bi se dobila korisnikova zadnja lokacija i pohranjuje se kao tip podatka

Location. Zbog oznake Task dohvaćanje lokacije će se odvijati paralelno s drugim funkcijama. Stoga se postavlja slušatelj koji prati je li funkcija uspješno dohvatila lokaciju korisnika. Nakon što je lokacija uspješno dohvaćena podatak tipa *Location* se pretvara u *LatLng*. Uzimaju se latituda i longituda iz *location* objekta i pohranjuju u globalnu varijablu *userLocation* kao *LatLng* tip podatka. Dohvaćanje korisnikove lokacije prikazano je na slici 4.14.

```
private fun fetchLocation(){

    var tempLatLng = LatLng( latitude: 0.0, longitude: 0.0)
    if (ActivityCompat.checkSelfPermission( context: this,
            android.Manifest.permission.ACCESS_COARSE_LOCATION
        ) != PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission(
            context: this,
            android.Manifest.permission.ACCESS_FINE_LOCATION
        ) != PackageManager.PERMISSION_GRANTED
    ) {

        ActivityCompat.requestPermissions(
            activity: this,
            arrayOf(
                android.Manifest.permission.ACCESS_FINE_LOCATION,
                android.Manifest.permission.ACCESS_COARSE_LOCATION
            ), requestCode: 101)

    }
    val task : Task<Location> = fusedLocationProviderClient.lastLocation

    task.addOnSuccessListener {  it: Location!
        if(it != null){
            var tempLat = it.latitude
            var tempLng =it.longitude
            tempLatLng = LatLng(tempLat,tempLng)
            userLocation=tempLatLng

            Log.d( tag: "OOOOOOOOOO",  msg: "LOCATION GOT successfully")
        }
        else{
        Log.d( tag: "OOOOOOOOOO",  msg: "FAILED TO GET LOCATION")
        }            }

}
```

Slika 4.14. Prikaz funkcije fetchLocation

Nakon toga unutar glavne aktivnosti se provjerava je li lokacija dohvaćena tako da se uspoređuje s početnom vrijednosti koja je nula. Ako je uspješno dohvaćena koristi se funkcija *geocoder*. Ona prima koordinate latitude i longitude i vrača podatke o lokaciji na tim koordinatam. *WeatherTask* kao parametar prima *string* koji sadrži naziv lokacije i kod države. Stoga se od dobivenih podataka stvara ulazni parametri te se pokreće funkcija za dohvaćanje vremenskih uvjeta.

Pozivom *excecute* funkcije na *weatherTask* objektu poziv se funkcija *doInBackground* čija implementacija je prikazana na slici 4.15. Ona prima korisnikovu lokaciju u obliku tipa *String* jer taj podatak umeće u *string* koji će služiti kao zahtjev Open Weather poslužitelju. Također je potrebno dodati API ključ kako bi poslužitelj znao tko šalje zahtjev i kakve podatke ima pravo dohvatiti. Ako dohvaćanje podataka bude ne uspješno moglo bi doći do pogreške u radu aplikacije i ona se može srušiti. Stoga se koristi *try-catch* blok kako bi se osiguralo da u tom slučaju programer može odlučiti što će se dogoditi. U ovom slučaju samo se vrijednost odgovora postavlja na *null* vrijednost.

```
override fun doInBackground(vararg p0: String?): String? {
    var response: String?
    try {
        response = URL( spec: "https://api.openweathermap.org/data/2.5/weather?q=" +
                "$ldata&units=metric&&APPID=$defaultkey").readText(Charsets.UTF_8)

    } catch (e: Exception) {
        response = null
    }
    return response
}
```

Slika 4.15. Prikaz slanja zahtjeva Open Weather API poslužitelju

OnPostExcecute metoda čeka da se podaci dohvate i pohranjuje ih u JSON formatu. Iz pojedinih polja dobivenog JSON tipa podatka se dobivaju identifikator trenutnih meteoroloških uvjeta, temperatura i naziv lokacije. Te informacije se formatiraju u pravilan oblik za prikaz na glavnom ekranu i postavljaju se u *text view* elemente. Dok se vremenski identifikator sprema u varijablu *weatherid* koja će se kasnije koristiti u algoritmu za preporuku aktivnosti. Implementacija *OnPostExecute* metode je prikazano na slici 4.16.

Nakon izvršavanja *OnPostExecute* metode ponovno se poziva *delay* funkcija jer je potrebno osigurati da *setWeatherImage* klasa ne počne s izvršavanjem prije nego li podaci o vremenu budu

dohvaćeni. Zatim se postavlja slika na glavnom zaslonu ovisno o identifikatoru vremena *weather id.*

```
override fun onPostExecute(result: String?) {
    super.onPostExecute(result)
    try {
        val jsonObj = JSONObject(result)
        val main = jsonObj.getJSONObject( name: "main")
        val sys = jsonObj.getJSONObject( name: "sys")
        val wind = jsonObj.getJSONObject( name: "wind")
        val weather = jsonObj.getJSONArray( name: "weather").getJSONObject( index: 0)
        val temp = main.getString( name: "temp") + "°C"
        val address = jsonObj.getString( name: "name") + ", " + sys.getString( name: "country")
        weatherid=weather.getInt( name: "id").toString()
        findViewById<TextView>(R.id.locationShow).text=address
        findViewById<TextView>(R.id.tempShow).text=temp

    } catch (e: Exception) {
```

Slika 4.16. Prikaz dohvač anja meteoroloških uvjeta

4.3.2. Spremanje i dohvaćanje korisnikovih preferenci

Pri pokretanju glavne aktivnosti aplikacija provjerava je li korisnik ranije unio svoje ime tj. ima li spremljene preference. Ako nema vjerojatno je prvi put upalio aplikaciju te se pali *UserProfileActivity* gdje korisnik unosi ime i bira iz kojih kategorija želi dobiti preporučene aktivnosti. Nakon što se otvori aktivnost za odabir preferenca korisnik mora popuniti polje za ime i odabrati kategorije koje mu odgovaraju. Ako ne unese ime pritiskom na gumb za spremanje dobit će obavijest da unese ime. U suprotnom stvara se objekt tipa *StringBuilder*. Zatim se prolazi kroz svaku stavku objekta *dataModel*. Unutar svake stavke *dataModel* se nalazi naziv kategorije i oznaka je li odabrana ili ne. U *stringBuilderobjekt* se dodaju svi nazivi kategorija koje su odabrane u *list view* te se odvajaju zarezom. Zatim se poziva editor *SharedPreference* objekta koji je zadužen za pohranu podataka u objekt shared preferences. Unose se korisnikovo ime i odabir u *SharedPreference* svaki sa svojim identifikacijskim ključem. Potom se korisnika vraća na glavni zaslon. Na slici 4.17. prikazana je pohrana korisnikovih podataka.

```
findViewById<Button>(R.id.button2).setOnClickListener(){ it: View!

    if(findViewById<EditText>(R.id.editTextTextPersonName).text.isEmpty())
    {
        Toast.makeText( context: this, text: "Pleace enter a name to continue",Toast.LENGTH_SHORT).show()

    }
    else {
        var stringBuilder = StringBuilder()

        for (item in dataModel!!) {
            if (item.checked)
                stringBuilder.append(item.name).append(",")
        }


        prefEditor.apply { this: SharedPreferences.Editor!
            putString("name", etName.text.toString())
            putString("preferences", stringBuilder.toString())
            apply()
        }

            Toast.makeText( context: this,  text: "Preferences are saved", Toast.LENGTH_SHORT).show()


            startActivity(intent)

    }
}
```

Slika 4.17. Spremanje korisnikovih preferenci u SharedPreferences

Nakon povratka u glavnu aktivnost dohvaća se User Preference pod imenom *userPref* i učitava se ime koje je korisnik ranije odabrao kao što je prikazano na slici 4.18.

```
val sharedPreferences: SharedPreferences = getSharedPreferences( name: "userPref", Context.MODE_PRIVATE)
var username = sharedPreferences.getString("name", "").toString()
```

Slika 4.18. Učitavanje korisnikovog imena iz SharedPrefer

Ako se korisnik odluči predomisliti i odabrati druge aktivnosti po povratku u aktivnost odabira aktivnosti učitava se korisnikov prijašnji odabir. Prvo se stvara objekt *SharedPreference* koji dohvaća korisnikove podatke koji su spremljeni pod nazivom *userPref*. Zatim se učitavaju korisnikovo ime i odabiri kategorija iz *userPref*. Provjerava se je li korisnikovo ime spremljeno. Ako je pristupa se listi koja sadrži nazive svih kategorija koje su odabrane i osvježava prikaz tih kategorija unutar *list view* elementa na korisničkom sučelju metodom *notifyDataSetChanged*. Sada je korisnikov prvi odabir i podaci u aktivnosti sinkronizirani i korisnik može birati nove aktivnosti. Ovaj postupak je prikazan na slici 4.19.

```
override fun onStart() {
    super.onStart()
    var uName = ""
    var uPref = ""

    val sharedPreferences: SharedPreferences =getSharedPreferences( name: "userPref", Context.MODE_PRIVATE)

    uName=sharedPreferences.getString("name", "").toString()
    uPref=sharedPreferences.getString("preferences", "").toString()
    if(uName.isNotBlank())
    {
        findViewById<EditText>(R.id.editTextTextPersonName).setText(uName)
        for (item in dataModel!!)
        {
            if (uPref.contains(item.name.toString()))
            {
                item.checked=!item.checked
                adapter.notifyDataSetChanged()
            }
        }
    }
```

Slika 4.19. Sinkronizacija korisnikovih prijašnjih odabira i aktivnosti

4.3.3. Preporuka aktivnosti

Korisnikovim pritiskom na gumb za preporuku aktivnosti prvo se provjerava je li su svi potrebni podaci za preporuku aktivnosti već dohvaćeni. Provjerava se jesu li identifikator vremena i korisnikova lokacija još na početnoj vrijednosti. Ako nisu znači da su uspješno dohvaćeni. Također se provjerava je li prikazana poruka pozdrava korisniku. Ako je prikazana znači da je korisnik dobio mogućnost birati kategorije aktivnosti jer se poruka ne prikazuje ako nije prije uneseno ime u *UserProfileActivity*. Postavljaju se dohvaćeni podaci o vremenu, lokaciji, i radius u kojem će se prikazivati aktivnosti oko korisnika u objekt *intent*. Tada se poziva *GoogleMapsActivity* koristeći taj *intent* objekt. Dodavanje parametara u *intent* prikazano je na slici 4.20.

```
    if (findViewById<TextView>(R.id.welcomeTxT).text.isNotBlank()&&weatherid
            .isNotBlank()&&userLocation!=defaultLocation)
    {
        intentM.putExtra( name: "radius",seekBarProgress)
        intentM.putExtra( name: "LAT",userLocation.latitude)
    intentM.putExtra( name: "LON",userLocation.longitude)
    intentM.putExtra( name: "weatherID", weatherid.toInt())
    Log.d( tag: "OOOOOOOOOO", msg: "CHANGE ACTIVITY TO MAPS "+"lat:"+
            userLocation.latitude+" Lon:"+userLocation.longitude+" Weatherid:"+weatherid)
    startActivity(intentM)}
    else
    {
        Toast.makeText( context: this,  text: "Location and Weather " +
                "data has not been loaded corretly", Toast.LENGTH_SHORT).show()
    }
}
findViewById<ImageButton>(R.id.imageButton).setOnClickListener(){  it: View!

    startActivity(intent)
}
```

Slika 4.20. Dodavanje parametara u intent

Nakon što se otvori *GoogleMapsActivity* prvo se učitavaju korisnikove odabrane kategorije aktivnosti iz *SharedPreference*-a te zatim se razdvajaju pojedine kategorije na mjestu zareza i pohranjuju se u novu listu kao zasebni elementi. Učitavanje korisnikovih odabranih kategorija prikazano je na slici 4.21.

```
val sharedPreferences: SharedPreferences = getSharedPreferences( name: "userPref", Context.MODE_PRIVATE)
var userPreferences=sharedPreferences.getString("preferences", "").toString()
val stringBuilder = StringBuilder(userPreferences)
userSelectedPref=stringBuilder.split( ...delimiters: ",")
```

Slika 4.21. Učitavanje korisnikovih odabira

Zatim se poziva funkcija *GetLocationPermission* kako bih se još jednom provjerilo da aplikacija ima pravo pristupiti korisnikovoj lokaciji. Funkcija *GetLocationPermission* je prikazana na slici 4.22. Aplikacija je već dostavila korisnikovu lokaciju putem *intent* mehanizma. Unatoč tome Google Maps može za pojedine funkcionalnosti samostalno pokušati dohvatiti korisnikovu lokaciju. Ako se ovo ne napravi može doći do rušenja aplikacije.

```
private fun GetLocationPermission() {
    Log.d( tag: "OOOOOOOOOO",  msg: "GETTING LOCATION PERMISSION")
    if (ActivityCompat.checkSelfPermission( context: this,
            android.Manifest.permission.ACCESS_COARSE_LOCATION
        ) == PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission(
            context: this,
            android.Manifest.permission.ACCESS_FINE_LOCATION
        ) == PackageManager.PERMISSION_GRANTED){
        locationPermission=true
        Log.d( tag: "OOOOOOOOOO",  msg: "GETTING LOCATION PERMISSION permission there")
        initMap()
    }
    else{
        ActivityCompat.requestPermissions(
            activity: this,
            arrayOf(
                android.Manifest.permission.ACCESS_FINE_LOCATION,
                android.Manifest.permission.ACCESS_COARSE_LOCATION
            ), requestCode: 101)
        Log.d( tag: "OOOOOOOOOO",  msg: "GETTING LOCATION PERMISSION not there")
    }
}
```

Slika 4.22. Prikaz funkcije GetLocationPermission

Ako su prava pristupu korisnikovoj lokaciji ponovno potvrđena poziva se funkcija za inicijalizaciju karte. Unutar pomoćnog fragmenta se učitava karta koristeći pozadinski proces kao što je prikazano na slici 4.23.

```
private fun initMap(){
    Log.d( tag: "OOOOOOOOOO",  msg: "INITIALIZING MAP")
    val mapFragment = supportFragmentManager
        .findFragmentById(R.id.map) as SupportMapFragment
    mapFragment.getMapAsync( callback: this)
}
```

Slika 4.23. Prikaz funkcije initMap

Nakon što je karta uspješno učitana automatski se poziva funkcija *onMapReady*. Jer se funkcija za učitavanje karte događa u pozadini potrebna je ova funkcija kako bi osigurala da sve što je nalazi unutar nje tek izvrši nakon što je karta inicijalizirana. Nakon što je karta učitana dohvaća se korisnikova lokacija iz *intent*-a. Također se uključuje opcija koja pokazuje korisnikovu poziciju

na karti u pravom vremenu te se zumira na korisnikovu lokaciju. Sljedeće se stvara lista svih konkretnih tipova aktivnosti koje će korisniku biti preporučene. Funkcija *getRecommendation* generira konkretnu preporuku za korisnika, a kao parametre uzima listu svih tipova aktivnosti koji se mogu raditi unatoč lošem vremenu, listu općenito zabavnih tipova aktivnosti, listu tipova aktivnosti koje je korisnik odabrao i identifikator vremena *weatherID*. Lista korisnikovih tipova aktivnosti se prvo mora generirati stoga se poziva funkcija *getUserSelection*. Svaka kategorija koju je korisnik mogao odabrati sadrži nekoliko konkretnih tipova aktivnosti. Prolazi se kroz korisnikov odabir i dodaju se svi tipovi aktivnosti za pojedinu odabranu kategoriju u novu listu koja se vraća kao ulaz funkciji *getRecommendation*. Na slici 4.24 prikazana je implementacija *getUser* selection funkcije.

```
private fun getUserSelection(userSelected:List<String>):List<String>{
    var recomendedActivitys = listOf<String>()

    val sports  = listOf("bowling_alley","stadium","gym","bicycle_store")
    val culture = listOf("museum","tourist_attraction")
    val animals = listOf("zoo","pet_store","aquarium")
    val camping = listOf("rv_park","campground")

    for (item in userSelected)
    {
        if(item=="Sports")
            recomendedActivitys+=sports
        if(item=="Culture")
            recomendedActivitys+=culture
        if(item=="Animals")
            recomendedActivitys+=animals
        if(item=="Camping")
            recomendedActivitys+=camping
    }
    return recomendedActivitys
}
```

Slika 4.24. Prikaz funkcije getUserSelection

Funkcije za generiranje tipova aktivnosti koje su općenito zabavne i prigodne za loše vremenske uvjete prikazane su na slici 4.25. Obje funkcije imaju točno zadane tipove aktivnosti koje vraćaju u obliku liste.

```
private fun getBadWeatherActivities():List<String>{
    val badWeatherActivities = listOf("aquarium","bicycle_store","bowling_alley",
        "gym","museum","book_store","pet_store","shoe_store","travel_agency")
    return badWeatherActivities
}

private fun getGenericActivities():List<String>{
    val nothingSelected = listOf("zoo","museum","park","tourist_attraction","aquarium","gym")
    return nothingSelected
}
```

Slika 4.25. Prikaz funkcija getBadWeatherActivities i getGenericActivities

Funkcija za preporuke prvo provjerava je li lista s korisnikovim odabirima prazna. Ako je korisnik nije napravio odabir. Tada se provjerava kakvi su vremenski uvjeti. Ako je dobro vrijeme lista općenito zabavnih tipova aktivnosti će se pohraniti u *tempRecommedations* varijablu kao privremena preporuka. Ako nije dobro vrijeme lista s vremenski ne ovisnim tipovima aktivnostima će biti spremljena kao privremena preporuka. Ako lista korisnikovih odabira nije prazna ona će pohranjena kao privremena preporuka. U sljedećem koraku ako je loše vrijeme funkcija će provjeravati koji tipovi aktivnosti u listi privremenih preporuka *tempRecommendations* se mogu raditi po lošem vremenu. Ako takvih nema izlaz iz funkcije će biti skup svih tipova aktivnosti koje su neovisne o lošem vremenu. Ako je dobro vrijeme izlaz funkcije će biti lista pohranjena u varijabli *tempRecommendation*. Na slici 4.26. prikazana je implementacija funkcije *getRecommendation*.

```
private fun getRecommendation(userSelectedActivities :List<String>, badWeatherActivities : List<String>,
                              genericActivities : List<String>, weather : Int):List<String>{
    Log.d( tag: "OOOOOOOOOO", msg: "INPUT Recommendations: " +userSelectedActivities.toString())
    var tempRecommendation : List<String> = listOfNotNull()
    var recommendation : List<String> = listOfNotNull()
    if(userSelectedActivities.isEmpty()) {
        if(weather>=200&&weather<800) {
            tempRecommendation = badWeatherActivities
            Toast.makeText( context: this,
                text: "The weather might be bad but you can still plan ahead for your next adventure",
                    Toast.LENGTH_LONG).show()
        }
            else
            tempRecommendation=genericActivities
            Toast.makeText( context: this,
                text: "Here are some generaly fun activities", Toast.LENGTH_SHORT).show()
    }
        else
        tempRecommendation=userSelectedActivities

    if(weather >=200 && weather < 800)
    {
        for(item in tempRecommendation)
        {
            if(badWeatherActivities.contains(item))
                recommendation+=item
        }
        if (recommendation.isEmpty())
            recommendation=badWeatherActivities
        Toast.makeText( context: this,
            text: "The weather might be bad but you can still plan ahead for your next adventure",
                Toast.LENGTH_LONG).show()
    }
    else
        recommendation=tempRecommendation
    Log.d( tag: "OOOOOOOOOO", msg: "OUTPUT Recommendations: " +recommendation.toString())
    Log.d( tag: "OOOOOOOOOO", msg: "WeatherID: " +weather.toString())
    return recommendation
}
```

Slika 4.26. Prikaz funkcije getRecommendation

Kada je lista konkretnih tipova lokacija koje se preporučuju korisniku generirana poziva se funkcija *setUpRetrofitCall* za svaku od tipova lokacije. Poziv funkcije *setUpRetrofitCall* prikazan je na slici 4.27. *SetUpRetrofitCall* za svaki element liste tipova aktivnosti stvara API poziv prema Place API-u. Specifično se poziva *nearby_Search* opcija koja kao odgovor daje lokacije oko korisnika u određenom radiusu ovisno o zadanom tipu lokacije. Komunikacija s Place API-om se vrši klijentom Retrofit. Definirano je sučelje *PlaceApiService* koje opisuje način komunikacije s Place API-om. Sučelje sadrži krajnji parametar API poziva u ovom slučaju „*nearbysearch/json?*”, sadrži parametre koji definiraju upit prema poslužitelju i get metodu kojom se dohvaćaju lokacije. Na slici 4.28 prikazana je implementacije sučelja *PlaceApiService.*

```
var temp = getRecommendation(getUserSelection(userSelectedPref),getBadWeatherActivities(),getGenericActivities(), weatherId)
for(item in temp){
setUpRetrofitCall(mMap,item)
}
```

Slika 4.27. Prikaz poziva funkcije setUpRetrofitCall

```
interface PlaceApiService {
    @GET("nearbysearch/json?")
    fun getPlaces(
    @Query("location") latLng: String,
    @Query("radius") radius: String,
    @Query("type") type: String,
    @Query("key") key: String
    ): Call<PlacesApiData>
}
```

Slika 4.28. Prikaz sučelja PlaceApiService

Kako bi se koristio Retrofit potrebno je napraviti klasu *PlacesRetrofit* koja stvara objekt tipa retrofit i spaja se sa sučeljem. *PlacesRetrofit* sadrži osnovnu URL adresu koja je potrebna za izradu API poziva. Unutar *PlacesRetrofit* definiran je Gson pretvarač koji povratni podatak iz API poziva pretvara u željeni oblik. U aplikaciji povratni podatak se pretvara u podatak tipa *PlacesApiData.* Na slici 4.29. prikazana je klasa *PlacesRetrofit, a* na slici 4.30. prikazan je izgled podatkovne klase *PlacesApiData*. Kako bi se podaci mogli koristiti pretvaraju se u podatkovnu klasu. Struktura podatkovne klase ovisi o strukturi odgovora na API poziv.

```
class PlacesRetrofit {
    private val baseUrl ="https://maps.googleapis.com/maps/api/place/"
fun getApiInterface(): PlaceApiService {
    val retrofit: Retrofit = Retrofit.Builder()
        .baseUrl(baseUrl)
        .addConverterFactory(GsonConverterFactory.create()).build()
    val service: PlaceApiService = retrofit.create<PlaceApiService>(PlaceApiService::class.java)
return service
}
}
```

Slika 4.29. Prikaz klase PlacesRetrofit

```
data class PlacesApiData(
    var html_attributions: Array<String>,
    var results : Array<Result>,
    var status: String
    ):java.io.Serializable

data class Result(
    var business_status: String? = null,
    var geometry: Geometry? = null,
    var icon: String? = null,
    var icon_background_color: String? = null,
    var icon_mask_base_uri: String? = null,
    var name: String? = null,
    var opening_hours: OpeningHours? = null,
    var photos: List<ContactsContract.CommonDataKinds.Photo>? = null,
    var place_id: String? = null,
    var plus_code: PlusCode? = null,
    var price_level: Int? = null,
    var rating: Double? = null,
    var reference: String? = null,
    var scope: String? = null,
    var types: List<String>? = null,
    var user_ratings_total: Int? = null,
    var vicinity: String? = null
)
```

Slika 4.30. Prikaz dijela podatkovne klase PlacesApiData

Na slici 4.31. može se vidjeti kako funkcija stvara poziv prema Place API. Prvo se inicijalizira objekt tipa *PlaceRetrofit*. Zatim se stvara call objekt koji će sadržati povratne informacije od API poziva koji su u ovom slučaju *PlacesApiData*. *PlacesRetrofit* inicijalizira Retrofit klijent i pretvarač Gson. Zatim se poziva *getPlaces* metoda koja je definiran u sučelju i prosljeđuju joj se parametri. Nakon što je poziv napravljen funkcija *onResponse* čeka odgovor od poslužitelja. Ako je poziv uspješan pretvarač Gson dobivene podatke će parsirati u zadanu podatkovnu klasu.

```
private fun setUpRetrofitCall(map: GoogleMap, activity : String){
    val placesRetrofit = PlacesRetrofit()
    val call : Call<PlacesApiData> = placesRetrofit.getApiInterface().getPlaces(
        latLng: lat.toString()+","+lon.toString(),range.toString(),activity,MAPS_API_KEY)
    call.enqueue(object : Callback<PlacesApiData> {
        override fun onResponse(call: Call<PlacesApiData>, response: Response<PlacesApiData>) {
            if(response.isSuccessful){
                val places = response.body()
                var marker = MarkerOptions()
                if (places != null) {
                    for(item in places.results) {
                        if(places.results.isNotEmpty()) {
                            var posLat = item.geometry?.location?.lat.toString()
                            var posLng = item.geometry?.location?.lng.toString()
                            var openingHours: String
```

Slika 4.31. Prikaz dijela funkcije SetUpRetrofitCall

Nakon što su podaci o lokacijama uspješno dohvaćeni stvara se marker objekt za svaku lokaciju. API call za jedan tip lokacije vrča više lokacija stog se prolazi petljom kroz sve lokacije istog tipa i za svaku se posebno definira marker. Kako bi se marker postavio na kartu potrebno mu je odrediti koordinate. Na slici 4.32. prikazan je postupak pridruživanja vrijednosti lokacije markeru. Za marker također je potrebno definirati titulu koja je u ovom slučaju naziv lokacije i *snippet* koji predstavlja dodatne informacije o markeru tj. lokaciji. Dodatno moguće je i promijeniti ikonu marker.

```
val places = response.body()
var marker = MarkerOptions()
if (places != null) {
    for(item in places.results) {
        if(places.results.isNotEmpty()) {
            var posLat = item.geometry?.location?.lat.toString()
            var posLng = item.geometry?.location?.lng.toString()
            var openingHours: String
            if (item.opening_hours?.open_now == true)
            {
                openingHours="This place is now open"
            }
            else{
                openingHours="This place is currently closed"
            }

            var snippet = "Opening hours: "+openingHours+"\n"+
                          "Buisness status: "+item.business_status+"\n"+
                          "User rating: "+item.rating.toString()
            var pos = LatLng(posLat.toDouble(),posLng.toDouble())
            Log.d( tag: "OOOOOOOOOO", msg: "MarkerNAME LAT LONG"+item.name+": "+item.rating)

            marker.position(pos)
            marker.title(item.name)
            marker.snippet(snippet)
            marker.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_RED))
```

Slika 4.32. Prikaz stvaranja markera unutar setUpRetrofitCall klase

Ikoni markera je moguće u potpunosti promijeniti oblik ili samo boju markera. Na slici 4.33. prikazan je postupak promjene boje marker ovisno o korisnikovoj ocjeni lokacije. Nakon što je marker definiran funkcijom *addMarker* dodaje se opisani marker na kartu.

```
if(item.name==marker.title&&(item.rating==4||item.rating==5))
    marker.icon(BitmapDescriptorFactory.
    defaultMarker(BitmapDescriptorFactory.HUE_YELLOW))
else if(item.name==marker.title&&(item.rating==1||item.rating==2))
    marker.icon(BitmapDescriptorFactory.
    defaultMarker(BitmapDescriptorFactory.HUE_GREEN))
```

Slika 4.33. Prikaz promjene boje markera ovisno o ocjeni korisnika

4.3.4. Pohrana i dohvaćanje korisničkih ocjena

Kao u prethodnom poglavlju rečeno potrebno je minimalno tri komponente kako bi se implementirala baza podataka Room. Prvi je entitet. Svaki entitet predstavlja tablicu unutar baze

podataka tj. svaka instanca entiteta predstavlja red u tablici. Kako bi se stvorio entitet potrebno je klasi dati oznaku *@Entity* iznad definicije klase i definirati naziv tablice .Također je potrebno odrediti primarni ključ oznakom *@PrimaryKey* iznad jednog od podatkovnih članova klase. U ovoj bazi bilo je potrebno postaviti *Autogenerate na false* za primarni ključ. Ovo je bilo potrebno jer primarni ključ ove tablice je ime lokacije i on ne smije biti slučajno generiran već odgovarati pojedinoj lokaciji. Prikaz implementacije entitet klase prikazan je na slici 4.34.

```
@Entity(tableName = "ratings_table")
data class favoritePlaceRating(
    @PrimaryKey(autoGenerate = false)
    var name : String = "",
    var rating : Int =0,
    var coment : String = ""
)
```

Slika 4.34. Prikaz entity klase favoritePlaceRating

Druga komponenta je baza podataka. Implementacija klase baze podataka prikazana je na slici 4.35. Ova komponenta je odgovorna za stvaranje i inicijaliziranje baze podataka. Sadrži metodu s *@Database* oznakom koja definira listu entitet klasa koje su dio baze podataka. *Database* klasa slijedi oblikovni obrazac singleton. To znači da ne smije postojati više od jedne instanca klase. Metoda *getInstance* provjerava je li postoji instanca klase i ako ne postoji stvara ju. Funkcijom *ratingsDao* može se pristupiti *data access* objektu.

```
@Database(entities = [favoritePlaceRating::class], version = 1, exportSchema = false)
abstract class RatingsDatabase : RoomDatabase(){
    abstract fun ratingsDao() :ratingsDao

    companion object{

        @Volatile
        private var INSTANCE: RatingsDatabase? = null
        fun getInstance(context: Context):RatingsDatabase{
            synchronized( lock: this){
                var instance = INSTANCE
                if(instance == null) {
                    instance = Room.databaseBuilder(
                        context.applicationContext,
                        RatingsDatabase::class.java,
                         name: "ratings_database1"
                    ).build()
                    INSTANCE = instance
                }
                return instance
                }
            }
        }
    }
```

Slika 4.35. Prikaz klase baze podataka RatingsDatabase

Zadnja obvezna komponenta je DAO – objekt za pristup podacima. Dao je glavni način pristupa podacima. Dao je sučelje koje ima @Dao oznaku iznad definicije. Unutar njega se nalaze sve metode pristupa podacima. Dao sadrži osnovne operacije baza podataka kao brisanje i dodavanje objekata ali je i moguće pisati SQL upite bazi podataka. Implementacija DAO klase prikazana je na slici 4.36.

```
@Dao
interface ratingsDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insert(rating:favoritePlaceRating)
    @Update
    suspend fun update(rating:favoritePlaceRating)
    @Delete
    suspend fun delete(rating:favoritePlaceRating)
    @Query("SELECT * FROM ratings_table WHERE name = :placeName")
    suspend fun get(placeName:String): favoritePlaceRating
    @Query("SELECT * FROM ratings_table ORDER BY name DESC")
    fun getAll(): LiveData<List<favoritePlaceRating>>
}
```

Slika 4.36. Prikaz dao klase ratingsDao

Prema [21] Repozitorij je klasa koja apstrahira pristup do više izvora podataka. Repozitorij nije nužna komponenta ali je preporučen radi odvajanja koda i arhitekture. Repozitorij koristi dao kako bi obavljao svoje funkcije. Implementacija klase repozitorij nalazi se na slici 4.37.

```
class RatingsRepository (private val dao : ratingsDao){
    val readAllData: LiveData<List<favoritePlaceRating>> = dao.getAll()
  suspend  fun addRating(rating: favoritePlaceRating){
        dao.insert(rating)
    }

    suspend fun updateRating(rating: favoritePlaceRating){
        dao.update(rating)
    }
    suspend fun deleteRating(rating : favoritePlaceRating){
        dao.delete(rating)
    }
    suspend fun getRating(name : String):favoritePlaceRating{
         var rating = dao.get(name)
         return rating
     }
}
```

Slika 4.37. Prikaz klase RatingRepository

ViewModel omogućuje komunikaciju između elemenata korisničkog sučelja i repozitorija. Uloga *ViewModel*-a je pružiti potrebne podatke elementima korisničkog sučelja. Implementacija *ViewModel*-a se nalazi na slici 4.38.

```
class RatingsViewModel(application: Application) : AndroidViewModel(application){
    val readAllData: LiveData<List<favoritePlaceRating>>
    private val repository : RatingsRepository
    private val _ratingLiveData = MutableLiveData<favoritePlaceRating>()
    init
    {
        val dao = RatingsDatabase.getInstance(application).ratingsDao()
        repository = RatingsRepository(dao)
        readAllData = repository.readAllData
    }
    fun addRating(rating:favoritePlaceRating){
        viewModelScope.launch (Dispatchers.IO){ this: CoroutineScope
        repository.addRating(rating)
        }
    }
    fun getRating(name : String){
        viewModelScope.launch(Dispatchers.IO) { this: CoroutineScope
        val rating = repository.getRating(name)
        _ratingLiveData.postValue(rating)
        }
    }
    fun updateRating(rating:favoritePlaceRating){
        viewModelScope.launch(Dispatchers.IO) { this: CoroutineScope
        repository.updateRating(rating)
        }
    }
    fun deleteRating(rating: favoritePlaceRating){
        viewModelScope.launch(Dispatchers.IO) { this: CoroutineScope
       repository.deleteRating(rating)
        }
    }
}
```

Slika 4.38. Prikaz klase RatingsViewModel

U aplikaciji se Room *database* koristi na sljedeći način. Prvo je potrebno inicijalizirati *ViewModel* čime se inicijalizira i DAO. Time imamo pristup bazi podataka. Unutar aplikacije Room se koristi za pohranu korisnikovih ocjena lokaciji. Svaki puta kada je *GoogleMapsActivity* pozvan nakon inicijalizacije *ViewModel*-a poziva se funkcija *getRating*. Ona pokreće pozadinsku funkciju koja čita sve podatke iz baze. Postavlja se *listener* objekt koji prati kada će podaci biti spremni i tada se pohranjuju u globalnu varijablu *placeRatings*. Ova varijabla će se koristiti kako bi se provjerilo je li marker koji se upravo stvara od lokacije kojoj je korisnik već dao ocjenu. Provjerava se je li trenutna titula markera ista kao ime u bazi i ako je provjerava se koju je ocjenu korisnik dao lokaciji i po tome se određuje boja markera. Ako je lokacija ocijenjena bit će prikazana i korisnikova ocjena i komentar u informacijskoj kutiji markera. Stvaranje adaptiranog markera prikazano je na slici 4.39.

```
    marker.position(pos)
    marker.title(item.name)
    marker.snippet(snippet)
    marker.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_RED))
    for(item in placeRatings)
    {
        if(item.name==marker.title&&(item.rating==4||item.rating==5))
            marker.icon(BitmapDescriptorFactory.
            defaultMarker(BitmapDescriptorFactory.HUE_YELLOW))
        else if(item.name==marker.title&&(item.rating==1||item.rating==2))
            marker.icon(BitmapDescriptorFactory.
            defaultMarker(BitmapDescriptorFactory.HUE_GREEN))

        if(item.name==marker.title)
        {
            var sb = StringBuilder( str: snippet+"\nYour Comment: "+item.coment+
                    "\nYoure Rating:"+item.rating.toString())
            marker.snippet(sb.toString())
        }
    }
map.addMarker(marker)}
```

Slika 4.39. Prikaz stvaranja adaptiranog markera ovisno o korisnkovoj ocjeni

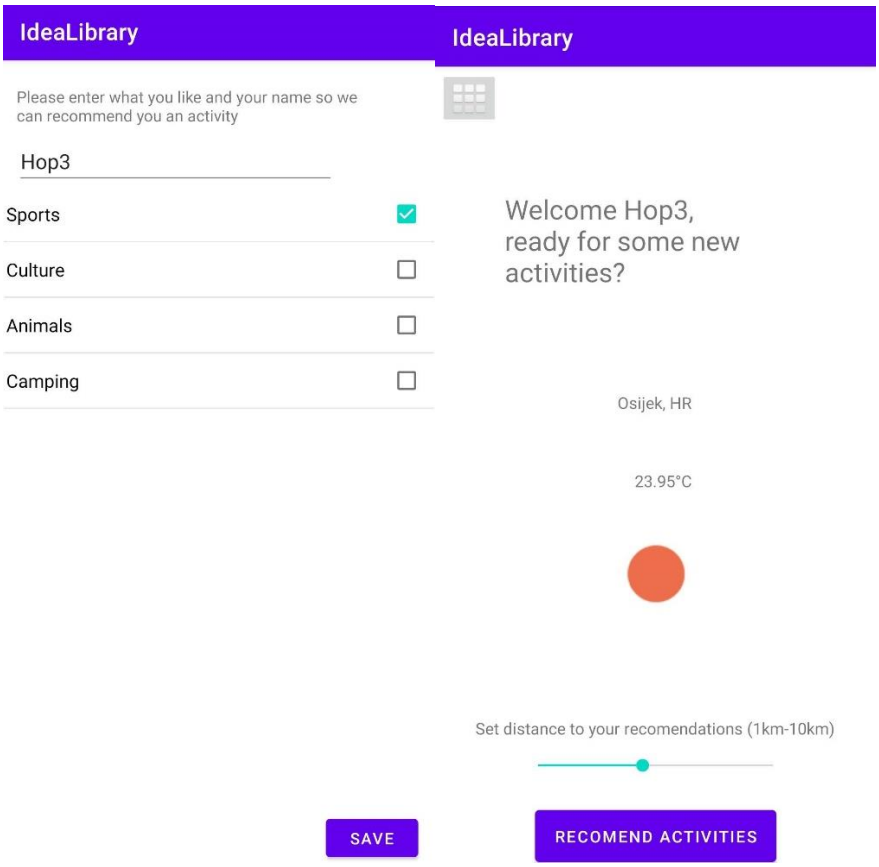
Osim za prikaz ocijenjenih lokacija dohvaćeni podaci iz baze se također koriste kako bi se elementi korisničkog sučelja ažurirali.

5. PRIKAZ KORIŠTENJA I ANALIZA RADA MOBILNE APLIKACIJE

U ovom poglavlju prikazat će se upotrijebe mobilne aplikacije. Bit će opisane njene mogućnosti i funkcionalnosti. Također će biti testirane neke od funkcionalnosti aplikacije i analizirani rezultati testova.

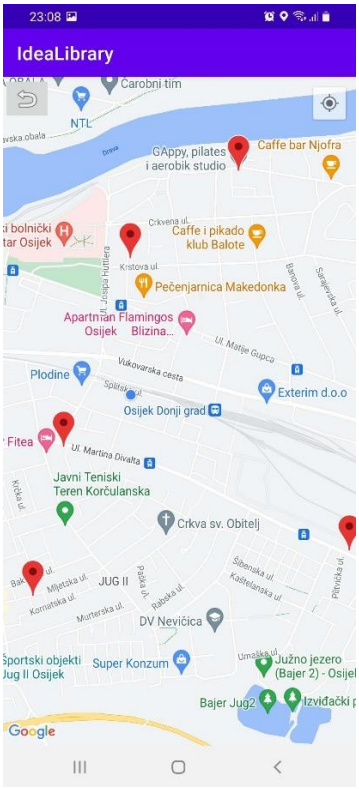
5.1. Prikaz korištenja mobilne aplikacije

Prvim pokretanjem aplikacije korisnik se dovodi u *UserProfileActivity* prikazano na slici 5.1. gdje se od korisnika traži unos imena i odabir kategorija koje želi. Korisnik nije dužan odabrati niti jednu od kategorija ali mora unesti ime ako želi nastaviti. Nakon što korisnik unese ime dovodi se do *MainActivity* koji je glavni zaslon ove aplikacije. Glavni zaslon je prikazan na slici 5.1. Ovdje se prikazuje korisnikova lokacija kao i trenutna temperatura zraka i sličica koja opisuje meteorološko stanje na toj lokaciji. Korisnik se može odlučiti vratiti nazad na prošlu aktivnost i odabrati druge kategorije. Korisnik može odabrati u kojem radiusu želi dobivati preporuke aktivnosti i kada je zadovoljan sa svojim odabirom može dobiti preporuku klikom na gumb *RecommendActivities*.



Slika 5.1. Prikaz izgleda aktivnosti UserProfileActivity i MainActivity

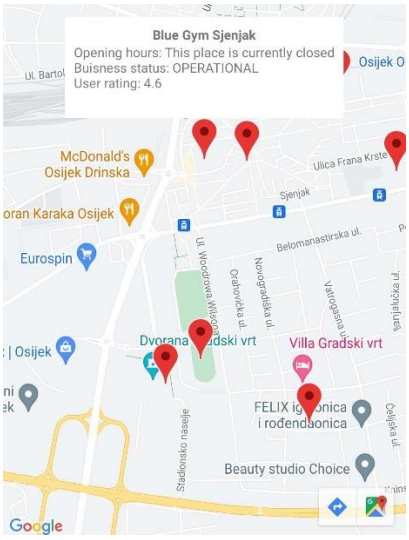
Tada dolazi do *GoogleMapsActivity* gdje se učitava karta. Ovdje se korisniku preporučuju aktivnosti ovisno odabiru kategorija na *UserProfileActivity*, ovisno o korisnikovoj geolokaciji, vremenskim uvjetima na toj geolokaciji i korisnikovom ocjenom lokacije ako postoji. Aktivnosti se preporučuju prikazivanjem markera na karti oko korisnika u zadanom radiusu. *GoogleMapsActivity* je prikazan na slici 5.2. Ako korisnik nije odabrao niti jednu kategoriju aktivnosti dobit će ponuđene aktivnosti koje su općenito zabavne, ne zahtijevaju mnogo opreme i koje nisu rijetke. Ako je loše vrijeme preporuke će biti prilagođene tako da se prikazuju samo one aktivnosti koje se mogu izvoditi tijekom loših vremenski uvjeta.



Slika 5.2. Prikaz izgleda aktivnosti GoogleMapsActivity

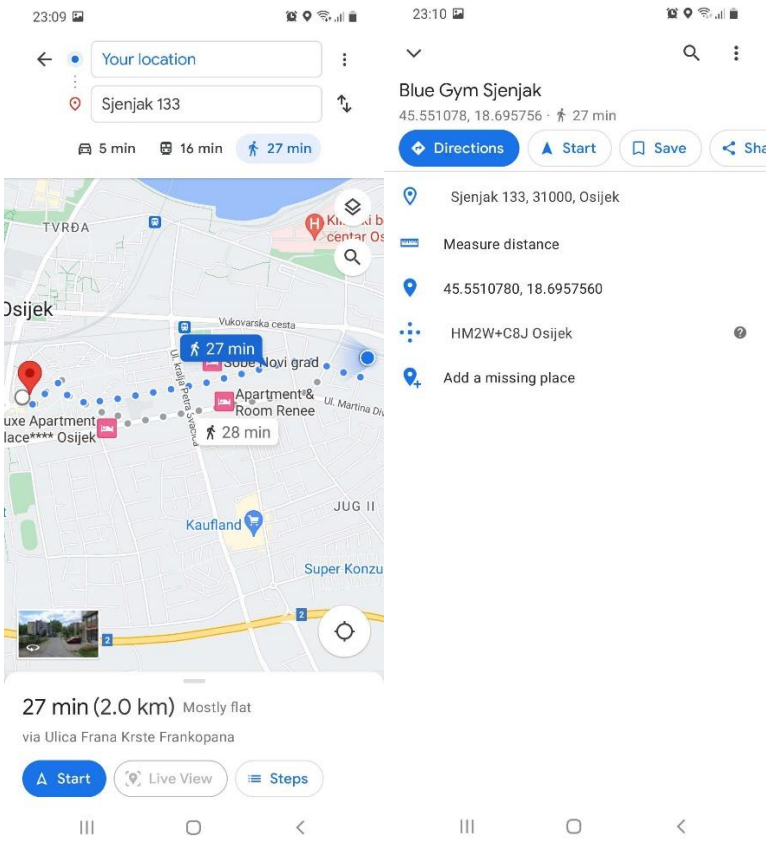
Pritiskom na marker korisniku se otvara informacijski prozor u kojem pišu informacije o kliknutom markeru. Korisnik ovdje može vidjeti informacije o statusu lokacije npr. je li

privremeno zatvorena, je li se renovira i sl. Korisnici može vidjet je li lokacija trenutno otvorena i može vidjeti ocjenu drugih korisnika. Informacijski prozor prikazan je na slici 5.3.



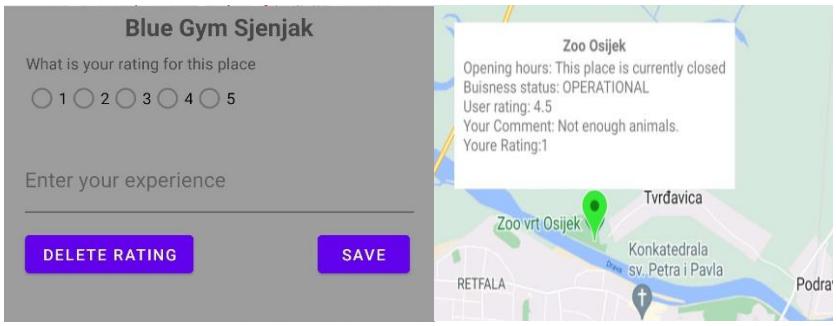
Slika 5.3. Prikaz informacijskog prozora

Klikom na marker korisnik također dobiva opciju dobiti preporučen put do lokacije klikom na donji lijevi gumb ili dobit više informacija klikom na donji desni gumb. Ove funkcionalnosti su prikazane na slici 5.4.



Slika 5.4. Prikaz funkcionalnosti putokaza i dodatnih informacija o lokaciji

Klikom na informacijski prozor otvara se ocjenjivački prozor za taj marker gdje korisnik može ostaviti ocjenu i komentar za tu lokaciju. Idući puta kada korisnik bude tražio preporuku moći će vidjeti svoj komentar i ocjenu u informacijskom prozoru. Ako je korisnik promijenio mišljenje može obrisati ocjenu za lokaciju u prozoru za ocjenjivanje. Ako je korisnik dao lošu ocjenu tj dva ili jedan marker te lokacije će biti obojen zeleno. Ako da neutralnu ocjenu tri marker će ostati crven, a ako da ocjenu četiri ili pet marker će imati žutu boju. Prikaz promjene boje markera i informacijskog prozora nakon ocjenjivanja je prikazano na slici 5.5. Ako korisnik se želi vratiti nazad u pojedinu aktivnost i promijeniti kategorije može to napraviti pritiskom na strelicu u gornjem lijevom kutu.



Slika 5.5. Prikaz ocjenjicaćkog prozora i informacijskog prozora

5.2. Ispitivanje mobilne aplikacije i analiza rezultata

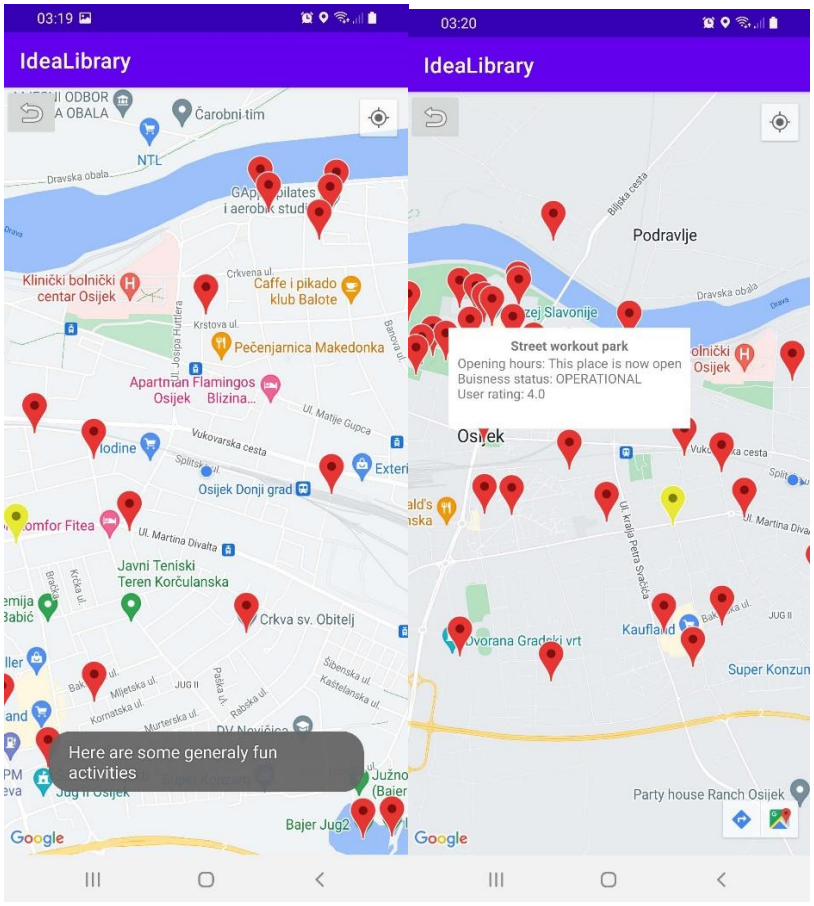
5.2.1. Ispitni slučaj 1

U ovom ispitnom slučaju se provjerava je li će korisniku biti preporučene opće zabavne aktivnosti ako korisnik ne odabere niti jednu od kategorija u *UserProfileActivity*. Vremenski uvjeti će za ovo ispitivanje biti stabilni. Postavke prije prvog ispitivanja su prikazane na slici 5.6.

Slika 5.6. Prikaz ulaznih parametara za prvo ispitivanje

Po preporučenim aktivnostima i prikazanoj poruci može se vidjeti kako je aplikacija pravilno preporučila općenitu skupinu aktivnosti kao što je bilo očekivano. Lista opće zabavnih aktivnosti

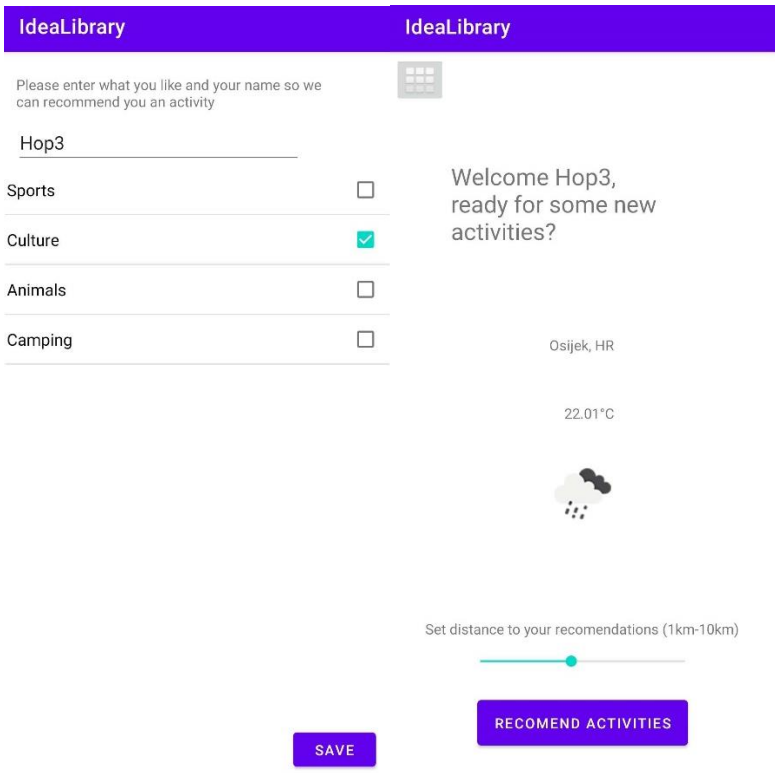
je birana tako da su se tražile najčešće aktivnosti tj. lokacije koje ne zahtijevaju specifičnu opremu. Po količini markera na karti može se reći da je to postignuto. Na slici 5.7. Prikazana je preporuka sustava za ovaj ispitni slučaj.



Slika 5.7. Prikaz preporuke aktivnosti za prvi ispitni slučaj

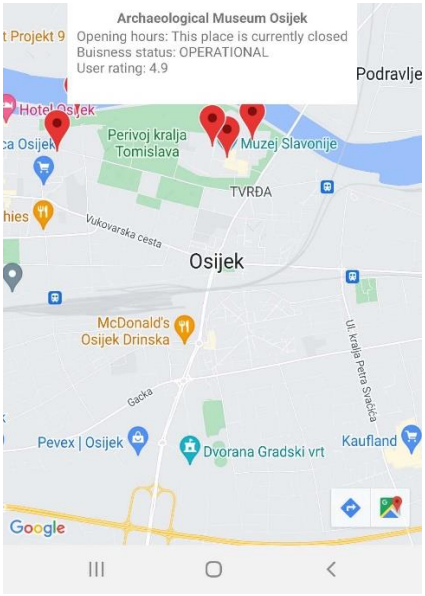
5.2.2. Ispitni slučaj 2

U ovom ispitnom slučaju ispituje se hoće li algoritam preporučivanja prikazati točne tipove lokacija ovisno o vremenskim uvjetima. Ako su nepogodni vremenski uvjeti algoritam preporučivanja bi trebao preporučivati samo one aktivnosti koje su prikladne tom vremenu od onih koje je korisnik odabrao. Za ovo ispitivanje je odabrana samo kategorija kultura jer ona sadrži jednu aktivnost koja je ovisna o vremenskim uvjetima tj. turističke znamenitosti i jednu koja je neovisna o njima tj. muzeji. Početni parametri za drugo ispitivanje su prikazani na slici 5.8.



Slika 5.8. Prikaz početnih parametara za drugo ispitivanje

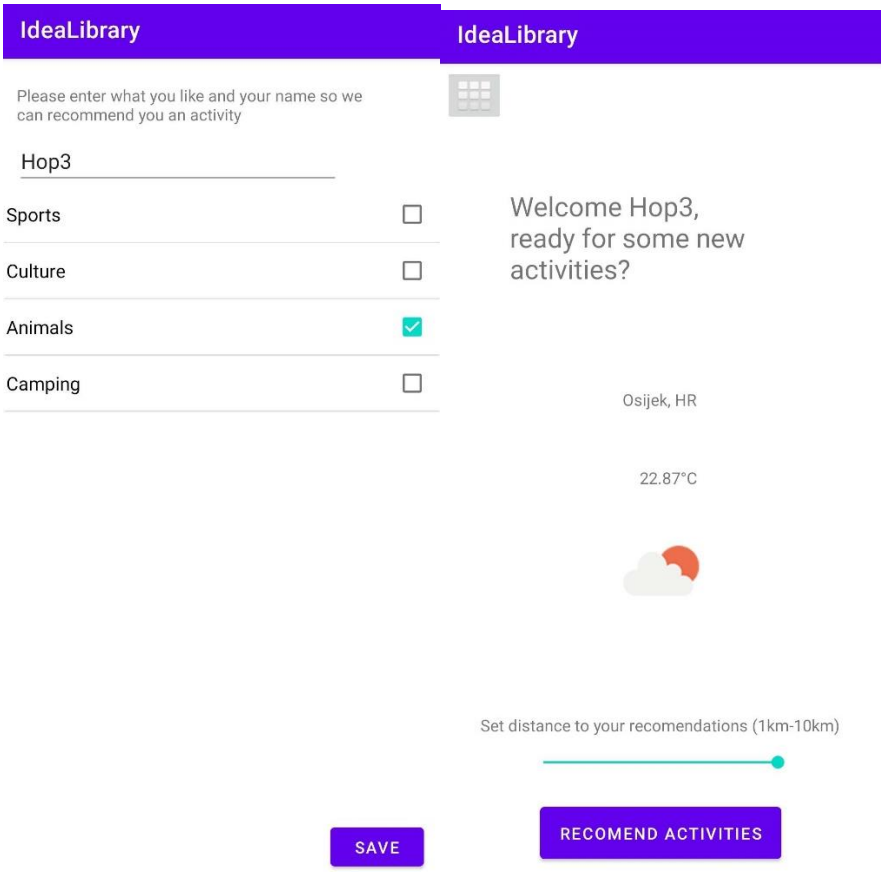
Algoritam je preporučio samo lokacije tipa muzej. S obzirom na to da je to jedina vrsta lokacije koju je korisnik odabrao a da nije ovisna o vremenu možemo zaključiti da je aplikacija pravilno preporučila aktivnost za vrijeme ne pogodnih vremenskih uvjeta. Rezultati ovog ispitivanja prikazani su na slici 5.9.



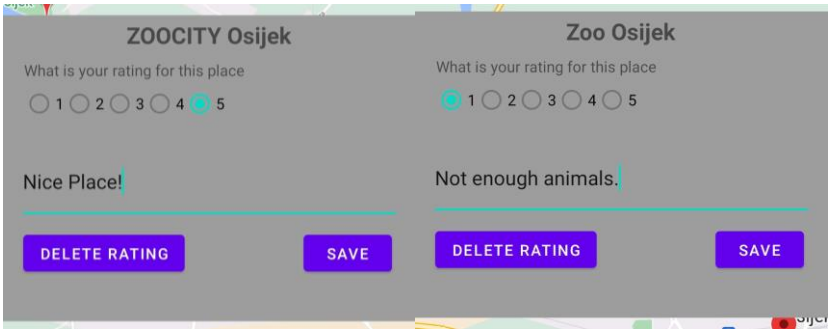
Slika 5.9. Prikaz rezultata drugog ispitivanja

5.2.3. Ispitni slučaj 3

U ovom ispitnom slučaju se provjerava ispravnost unosa komentara i ocjene korisnika kao i njihovo dohvaćanje. Korisnik je odabrao kategoriju životinje i vremenski uvjeti su dobri. Korisnikov odabir i vremenske uvjete možemo vidjeti na slici 5.10. Korisnik je dao pozitivnu ocjenu i komentar jednoj lokaciji, a loš komentar i ocjenu drugoj lokaciji. Ocijenjene lokacije su prikazane na slici 5.11. Pretpostavlja se da će pri idućoj preporuci aktivnost ocijenjene lokacije imati drugu boju markera. Također se pretpostavlja da će korisnikov komentar i ocjena biti dodani u informacijski prozor.

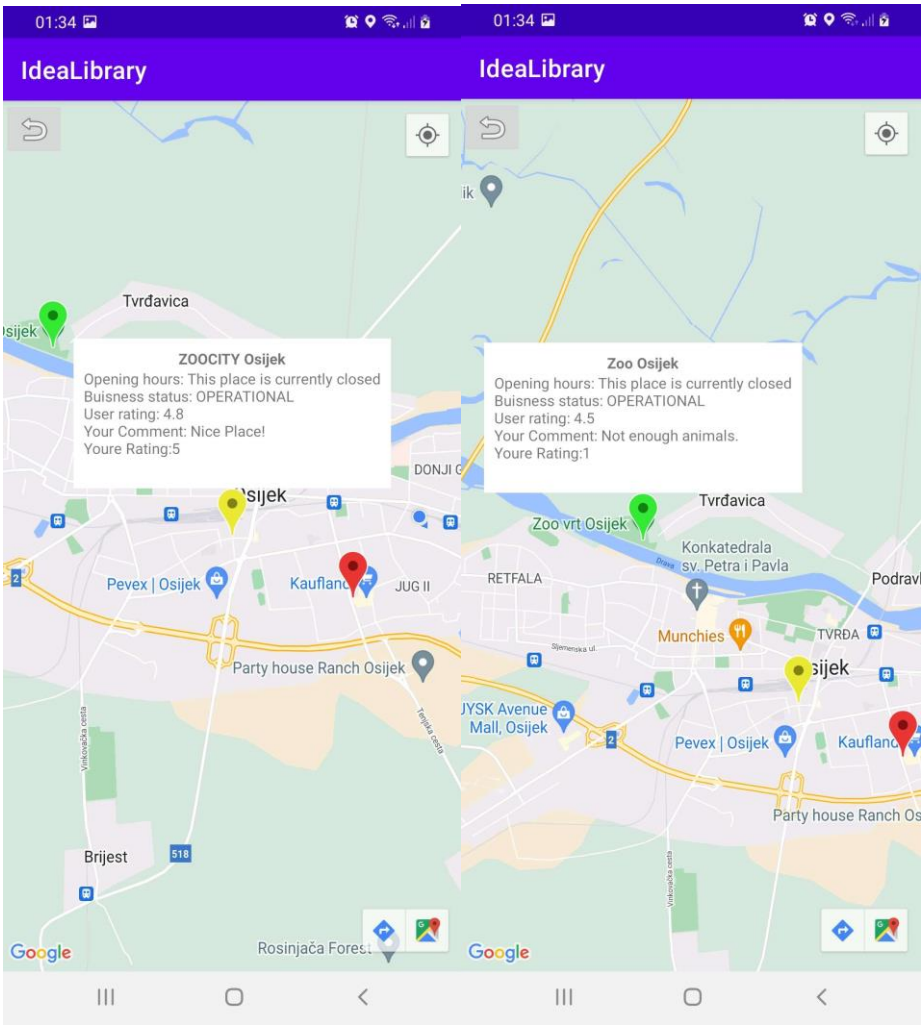


Slika 5.10. Prikaz odabrabih kategorija i vremenskih uvjeta prije trećeg ispitivanja



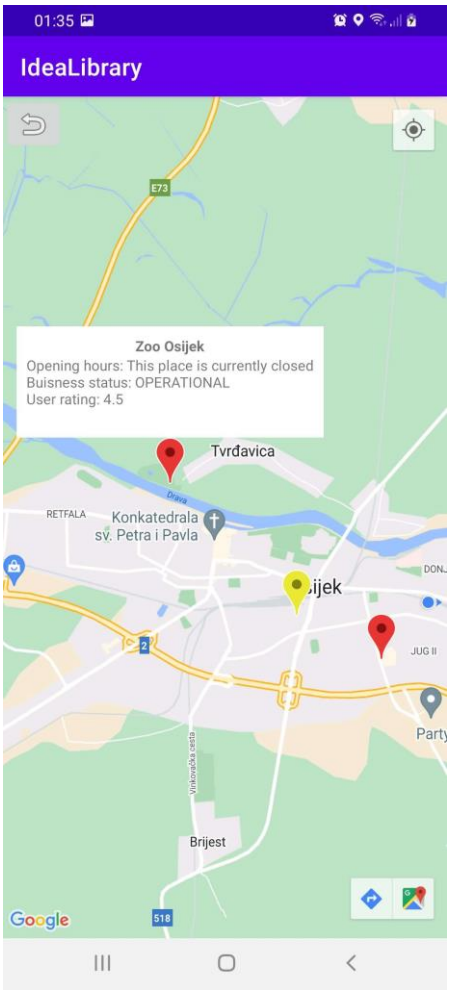
Slika 5.11. Prikaz ocjenjivanja lokacija

Na slici 5.12. vidi se kako se je boja markera loše ocijenjene lokacije promijenio u zeleno a boja dobro ocijenjene lokacije u žuto. Također se vidi kako su korisnikov komentar i ocjena upisani u informacijski prozor lokacije. Sada je potrebno vidjeti pritiskom na gumb za brisanje ocjene promjene na markeru i informacijskoj kutiji također nestanu.



Slika 5.12. Prikaz promjene boje markera nakon ocjenjivanja lokacije

Na slici 5.13. vidi se kako je marker ponovno vraćen u početnu boju nakon brisanja ocjene. Korisnikove informacije su se također obrisale iz informacijske kutije. Može se zaključiti da se komentari i ocjene korisnika ispravno pohranjuju i dohvaćaju.



Slika 5.13. Prikaz markera nakon brisanja ocjene

5.3. Analiza rezultata

U prva dva ispitna slučaja ispitan je algoritam preporučivanja aplikacije. U oba slučaja aplikacija se je ponašala prema očekivanjima tj. kako je definirano u trećem poglavlju. U trećem ispitnom slučaju provjeravalo se pohranjivanje u bazu podataka i dohvaćanje korisnikovih ocjena iz nje. Aplikacija je pravilno prikazala boju markera ocijenjenih lokacija i pravilno dodala korisnikove komentare u informacijski prozor. Brisanjem ocjena nakon ponovnog zatraživanja preporuke lokacije marker i informacijska kutija se vračaju u prvobitno stanje. Može se zaključiti da aplikacija pravilno pristupa bazi podataka. Aplikacija pravilno dohvaća vremenske uvjete i korisnikovu lokaciju te ih s manjim odgodama prikazuje korisniku. Stoga se zaključuje kako su naveden funkcionalnosti ispravno implementirane.

ZAKLJUČAK

U ovom završnom radu razvijena je mobilna Android aplikacija za preporučivanje aktivnosti u prirodi. Njen cilj je omogućiti jednostavno preporučivanje rekreativnih, zabavnih ili obrazovnih aktivnosti u prirodi bez traženja korisnikovih osobnih podataka. To se postiže preporukom lokacija blizu korisnika uz pomoć Place API-ja. Preporuke se nadalje temelje na sklonostima korisnika, odnosno odabiru korisniku zanimljivih kategorija aktivnosti, te na vremenskim uvjetima, geolokaciji i ocjenama lokacije. Geolokacija korisnika dobiva se putem usluge FusedLocationProvider, a vremenski uvjeti putem OpenWeather API-ja. Aplikacija koristi dvije vrste stalne pohrane Shared Preferences za korisnikove odabire i bazu podataka Room za pohranjivanje korisnikovih ocjena lokacija. Ispitivanje aplikacije je provedeno kroz tri slučaja. U prvom slučaju provjereno je hoće li aplikacija preporučiti korisniku općenite aktivnosti ako korisnik ne napravi odabir kategorija aktivnosti. U drugom slučaju provjereno je hoće li u slučaju loših vremenskih uvjeta aplikacija preporučiti samo aktivnosti koje se mogu izvoditi po lošim vremenskim uvjetima. U oba slučaja aplikacija je pravilno preporučila aktivnosti korisniku po čemu se zaključuje da sustav za preporučivanje radi ispravno. U trećem slučaju provjereno je hoće li se korisnikove ocjene lokacija pravilno dohvaćati i pohranjivati u bazu podataka. Postavljenim markerima na karti su dodane ocjene i promatrano je hoće li se promijeniti izgled markera ovisno o ocjeni nakon ponovnog učitavanja karte. Nakon toga su ocjene obrisane i ponovno se učitalo kartu kako bi se vidjelo dali se je marker vratio u prvobitnu boju. Markeri su pravilno mijenjali boju ovisno o korisnikovoj ocjeni stoga je zaključeno da aplikacija pravilno pohranjuje i dohvaća podatke iz baze podataka. Postoje mnoge mogućnosti poboljšanja aplikacije, pa je moguće poboljšati sustav stvaranja preporuka, rad s većim brojem kategorija i druge.

LITERATURA

[1] L. R. Larsona, L. E. Mullenbach, M. H. Browning, A. Rigolon i sur., „Greenspace and park use associated with less emotional distress among college students in the United States during the COVID-19 pandemic", Environmental Research 204 part D, str. 1-3., Ožujak 2022.

[2] F. Ricci, L. Rokach, B. Shapira, „Introduction to Recommender Systems Handbook ", Springer, 2011

[3] Dr. S. Jain, A. Grover, P. S. Thakur, S. K. Choudhary, „Trends, Problems And Solutions of Recommender System", International Conference on Computing, Communication and Automation, str. 955-958., Indija, 2015.

[4] FusedLocationProviderClient, Google Play services, , dostupno na: https://developers.google.com/android/reference/com/google/android/gms/location/FusedLocationProviderClient#public-tasklocation-getcurrentlocation-int-priority,-cancellationtoken-token [3.9.2023.]

[5] How to Get Current Location in Android?, GeeksforGeeks, ažurirano 19.9.2021. , dostupno na: https://www.geeksforgeeks.org/how-to-get-current-location-in-android/ [3.9.2023.]

[6] Create CheckBox For Each Item in ListView in Android, GeeksforGeeks, ažurirano 14.2.2022., dostupno na: https://www.geeksforgeeks.org/create-checkbox-for-each-item-in-listview-in-android/ [3.9.2023.]

[7] Maps SDK for Android Quickstart, Google Maps Platform, dostupno na: https://developers.google.com/maps/documentation/android-sdk/start [5.9.2023.]

[8] Weather conditions , OpenWeather, dostupno na: https://openweathermap.org/weather-conditions [1.9.2023.]

[9] Weather API current weather dostupno na: https://openweathermap.org/current [1.9.2023.]

[10] Difference between functional and non-functional requirements, javatpoint, dostupno na: https://www.javatpoint.com/functional-vs-non-functional-requirements [7.9.2023.]

[11] B. M., Mitchell, What Is Kotlin and Why Use It In Programming?, Coding Dojo, dostupno na: https://www.codingdojo.com/blog/what-is-kotlin [7.9.2023.]

[12] Meet Android Studio, developer.android.com, dostupno na: https://developer.android.com/studio/intro [7.9.2023.]

[13] FusedLocationProviderAPI: Simple, battery-efficient location API for Android, developers.google.com, dostupno na: https://developers.google.com/location-context/fused-location-provider [7.9.2023.]

[14] Google Maps Platform: Overview, Google Maps Platform, dostupno na: https://developers.google.com/maps/documentation/places/web-service/overview [7.9.2023.]

[15] Save simple data with SharedPreferences, dostupno na: https://developer.android.com/training/data-storage/shared-preferences [7.9.2023.]

[16] Maps SDK for Android: Info Windows, Google Maps Platform, dostupno na: https://developers.google.com/maps/documentation/android-sdk/infowindows [8.9.2023.]

[17] Maps SDK for Android: Markers, Google Maps Platform, dostupno na: https://developers.google.com/maps/documentation/android-sdk/marker [8.9.2023.]

[18] Save data in a local database using room, dostupno na: https://developer.android.com/training/data-storage/room [8.9.2023.]

[19] Places API: Place Types, dostupno na: https://developers.google.com/maps/documentation/places/web-service/supported_types [7.9.2023.]

[20] C.G. , Gures, Basic Implementation of Room Database With Repository and ViewModel | Android Jetpack, medium.com, dostupno na: https://medium.com/swlh/basic-implementation-of-room-database-with-repository-and-viewmodel-android-jetpack-8945b364d322 [9.9.2023.]

[21] 10.1: Room, LiveData, and ViewModel, dostupno na: https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/unit-4-saving-user-data/lesson-10-storing-data-with-room/10-1-c-room-livedata-viewmodel/10-1-c-room-livedata-viewmodel.html#repository [10.9.2023.]

SAŽETAK

U ovom završnom radu izrađena je mobilna aplikacija za preporuku sportskih rekreacijskih i sličnih aktivnosti u prirodi. Aplikacija je izrađena u razvojnoj okolini Android Studio pomoću programskog jezika Kotlin i opisnog jezika XML. Aplikacija preporučuje aktivnosti na temelju kategorija koje korisnik bira pri početku korištenja aplikacije, korisnikovoj geolokaciji, vremenskim uvjetima na toj geolokaciji, udaljenosti od preporučene aktivnosti te ocjeni pojedine lokacije. Za dohvaćanje geolokacije koristi se FusedLocationProvider, a za dohvaćanje vremenskih uvjeta OpenWeather API. Koristeći Retrofit klijent šalje se poziv Place API-ju koji vraća skup lokacija oko korisnika. Te lokacije se prikazuju korisniku u obliku markera na karti. Korisnik može uvidjeti opis lokacije, dobit putokaz do nje ili ju ocijeniti kako bi pri sljedećem preporučivanju korisnik lakše mogao razlikovati između dobro ocijenjenih i loše ocijenjenih ovisno o boji. Provedenim ispitivanjima aplikacije zaključeno je da aplikacija točno pohranjuje i dohvaća podatke iz baze podataka i na očekivan način preporučuje aktivnosti.

Ključne riječi: aktivnosti u prirodi, Android, mobilna aplikacija, sustav preporučivanja.

ABSTRACT

In this final project, a mobile application for recommending sports, recreational, and similar outdoor activities has been developed. The application was created in the Android Studio development environment using the Kotlin programming language and the XML descriptive language. The application recommends activities based on categories selected by the user when they first use the application, the user's geolocation, weather conditions at that geolocation, the distance from the recommended activity, and the rating of each location. The FusedLocationProvider is used to retrieve the geolocation, and the OpenWeather API is used to retrieve weather conditions. Using the Retrofit client, a call is made to the Place API, which returns a set of locations around the user. These locations are displayed to the user in the form of markers on a map. The user can view the location's description, get directions to it, or rate it for easier differentiation between well-rated and poorly rated locations in future recommendations, depending on the marker color. The conducted tests of the application have concluded that the application accurately stores and retrieves data from the database and recommends activities in the expected manner.

Keywords: outdoor activities, Android, mobile application, recommendation system

PRILOZI

Prilog 1: Završni rad u docx i pdf formatu

Prilog 2: Projektni kod tj. Projekt mobilne Android aplikacije