Android mobilna aplikacija za recenziranje objekata i događaja

Lukić, Ante

Undergraduate thesis / Završni rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:146580

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-19**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

Android mobilna aplikacija za recenziranje objekata i događaja

Završni rad

Ante Lukić

Osijek, 2022.

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

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 18.09.2022.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Ante Lukić
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	R 4387, 22.07.2019.
OIB Pristupnika:	25714789860
Mentor:	Izv. prof. dr. sc. Josip Balen
Sumentor:	,
Sumentor iz tvrtke:	
Naslov završnog rada:	Android mobilna aplikacija za recenziranje objekata i događaja
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rad:	U teorijskom dijelu rada potrebno je proučiti i opisati zahtjeve za razvoj aplikacija za recenziranje različitih tipova objekata (ugostiteljskih, hotelskih i sl.) i događaja (manifestacije, koncerti i sl.) i opisati tehnologije za izradu mobilnih aplikacija za Android platformu. U praktičnom dijelu rada potrebno je izraditi Android mobilnu aplikaciju u kojoj će se korisnik moći registrirati i upravljati svojim profilom te postaviti recenzije za objekte i događaje uz mogućnost postavljanja slika i poveznica. Također, potrebno je uz pomoć definiranja vlastitih interesa i praćenje lokacije učinkovito dijeliti
Prijedlog ocjene završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	18.09.2022.
Datum potvrde ocjene od strane Odbora:	21.09.2022.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 21.09.2022.

Ime i prezime studenta:	Ante Lukić
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R 4387, 22.07.2019.
Turnitin podudaranje [%]:	7

Ovom izjavom izjavljujem da je rad pod nazivom**: Android mobilna aplikacija za recenziranje objekata i događaja**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Josip Balen

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 studenta:

1.UVOD

Unazad desetak godina ljudi su sve više prihvaćali internetske stranice te su u njima nalazili zabavu. Internetske stranice su se drastično mijenjale i uvijek se novim korisničkim sučeljem pokušavala olakšati korisnikova interakcija. Najveći i neizbježivi problem je bio taj što je korisnik morao biti za nekom vrstom računala. Tomu je došao kraj izumom prenosivih pametnih telefona ili popularno nazvanih *smartphonea*.

Upravo nastankom pametnih uređaja ljudi su se riješili problema uvijek vezanja za računalo bilo ono stolno ili prijenosno. Milijuni informacija u džepu od hlača, vječna zanimljivost itd. To su samo neki od marketinških slogana za mobilne uređaje i svaki od njih je istinit. Cilj svake mobilne aplikacije je riješiti ili barem olakšati problem za korisnika. Osim toga cilj prilikom dizajniranja aplikacije je pokušati na sve moguće načine zadržati korisnika u aplikaciji. Dobar primjer toga je korištenje galerije ili kamere koja je dio aplikacije, a ne odlazak u galeriju ili kameru koja pripada mobitelu kako bi se dohvatila slika.

Izrada mobilne aplikacije zahtjeva razumijevanje nekoliko programskih jezika, ponašanje korisnika te smisao za dizajn. Dakle, potrebno je poznavati *Kotlin* ili *Java*, te *XML* za Android sustav i *Swift* za *iOS* sustav. Ponašanje korisnika se odnosi na ponašanje korisnika s mobitelom, odnosno na geste sustava. Osnovne geste sustava su klizanje u lijevo, desno, dolje ili gore. Iz tog razloga svaka uspješna aplikacija će za svoj dodatak gledati da je ta gesta najzastupljenija. Također, za razliku od *iOS* sustava, mobiteli s Android sustavom imaju navigacijsku traku s tri gumba. Najviše se koristi gumb *unazad*. Iz tog razloga Android aplikacije često ne sadrže gumb za vračanje unutar aplikacije. Smisao za dizajn je usko povezan s ponašanjem korisnika. Cilj dizajna je na što jednostavniji način predstaviti korisniku podatke i mogućnosti aplikacije.

Završni rad predstavlja aplikaciju koja rješava problem nedovoljnog poznavanja mjesta za izlaženje na piće, večeru ili zabavu unutar grada. Aplikacija ukomponira sva tri zahtjeva izrade mobilne aplikacije. Pisana je u *Kotlin* programskom jeziku, a sučelje je kreirano u *XML* programskom jeziku. Poštuje sva pravila dizajna koja su izdana od strane Googlea te koristi i inačice tih pravila iz biblioteka. Inačice poput *MaterialCardView, TextInputLayout* itd.

1.1. Zadatak završnog rada

Zadatak ovog završnog rada je objasniti tehnologije poput Android, *Firebase, Google Maps API* itd. Također, te tehnologije je potrebno implementirati u praktičnom dijelu završnog rada. U praktičnom dijelu je potrebno kreirati mobilnu aplikaciju koja omogućava dodavanje različitih tipova objekata (ugostiteljskih, hotelskih i sl.) i događaja (manifestacije, koncerti i sl.).

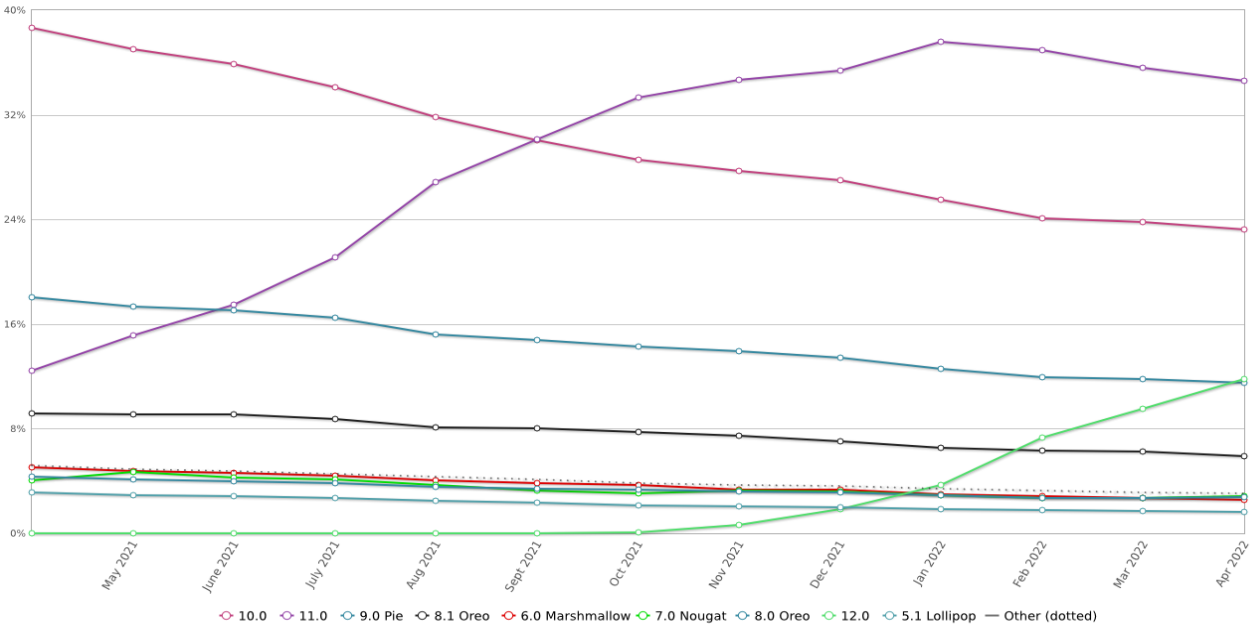
Aplikacija je izvedena pomoću Android Studio razvojnog okruženja (engl. IDE – *Integrated development environment*) i *Git* sustava za nadzor inačica verzije (engl. VCS – *Version Control System*).

2. OPERACIJISKI SUSTAV ANDROID

Android je najprošireniji operacijski sustav. Baziran je na *Linuxovoj* jezgri te se može implementirati u mobilne uređaje, tablete, televizore, satove, aute itd. Vlasnik Androida je Google i ono što Android sustav činiji drugačijim od ostalih je to što je otvorenog koda, odnosno dostupan je svima za nadogradnju i ispravljanje pogrešaka. Aplikacije za Android su se do 2019. Godine pisale u Java programskom jeziku, ali od 2019. Godine Google je proglasio *Kotlin* službenim programskim jezikom za Android.

2.1. Povijest Android operacijskog sustava

Android je operacijski sustav koji je nastao 2003. Godine od strane kompanije Android Inc. koju su osnovali Andy Rubin, Rich Miner i Nick Sears. Godinu dana kasnije promijenili su namjenu projekta te su se usmjerili na mobilne uređaje. Google je to prepoznao te je 2005. godine otkupio Android Inc. Također, Google je uveo nove inačice, odnosno napravio je Android tipom otvorenog koda kako bi omogućio svima doprinos u razvoju i za jezgru su izabrali Linux operacijski sustav koji je jedan od najvećih projekata otvorenog koda. Prvi mobilni uređaj koji je imao Android sustav bio je HTC Dream, odnosno T-Mobile G1, a izašao je 23, rujna 2008. [1].



Slika 2.1. Postotak tržišta korištenja Android verzija [2]

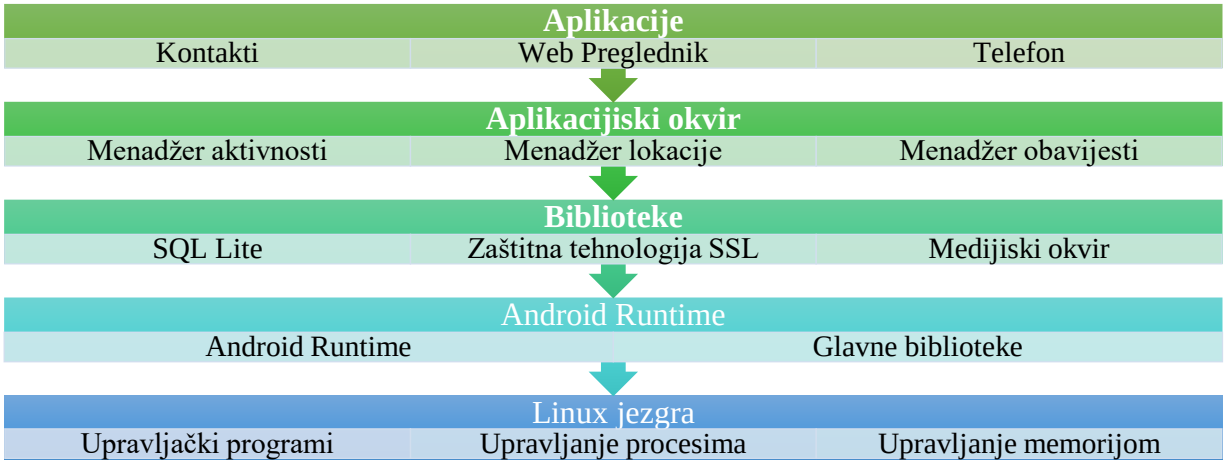
Do danas, Android je objavio 13 cjelokupnih verzija i sveukupno 32 API razine. Svaka verzija je uvela nove inačice bez kojih današnje korištenje mobilnih aplikacija ne bi bilo isto. 2010. godine izdana je prva verzija Android sustava po imenu Froyo. Svaka od verzija sadrži ime deserta po

abecednom redu, ali od verzije 10 pa nadalje Google je donio odluku kako će imena deserta biti samo interno ime dok će se nadalje verzije označavati brojkama. Na slici 2.1. je vidljiv postotak korištenih verzija Android sustava za mobilne uređaje i dlanovnike u razdoblju između travnja 2020. godine pa sve do travnja 2021. godine. Može se primijetiti iz grafa kako novije verzije vladaju tržištem, a kako je manji postotak starih verzija. Odnosno, izlaskom novih verzija sustava potiče proizvođače na implementiranje istih u svojim budućim pametnim uređajima.

2.2. Arhitektura Android sustava

Više razine u arhitekturi su zaslužne za interakciju s korisnikom dok su niže zaslužne za komunikaciju sa sklopovljem. Arhitektura Android sustava se može podijeliti na 5 dijelova:

- Aplikacijska razina
- Aplikacijski okvir
- Android radno okruženje
- Biblioteke
- Linux jezgra



Slika 2.2. Arhitektura Android sustava [3]

2.2. Aplikacijska razina

Najviši sloj koji odnosi se na korisničke aplikacije na Slici 2.2 su prikazane aplikacije poput kontakata, pretraživača itd., ali tu mogu pripadati sve aplikacije koje se mogu instalirati na uređaj. Ovaj sloj korisnik vidi i korisnik s njim komunicira. Jedan on ciljeva aplikacijskog sloja je na što bolji način olakšati korisniku interakciju s uređajem i prikriti mu kompleksnost koja se odvija pozadini.

2.2.1. Aplikacijski okvir

Aplikacijski okvir pruža servise više razine za aplikacije gdje spadaju menadžer lokacije, telefona, obavijesti itd. Ovaj sloj omogućava programerima da ih koriste i ponude njihove usluge u svojim aplikacijama [4]. Ova razina je vrlo važna Android programerima te je korištena i u ovom završnom radu. Neke od inačica ovog sloja korištene u ovom radu:

- Menadžer aktivnosti – donosi informacije i komunicira s aktivnostima, servisima i procesima
- Menadžer lokacije – omogućava periodično donošenje podataka o korisnikovoj lokaciji i njegov ulazak u zadani geografski prostor
- Menadžer obavijesti – brine se o stvaranju i uništavanju obavijesti.

2.2.2. Biblioteke

Biblioteke sadrže instrukcije koje omogućavaju korištenje svih glavnih Android funkcionalnosti. Pišu se u C i C++ programskim jezicima. Neke od biblioteka su:

- Menadžer površine – Biblioteka koja se brine o iscrtavanju na ekran i upravlja radom prozora
- Medijski okvir - omogućava snimanje i prikazivanje audio i video zapisa.
- SQL Lite - omogućava kreiranje i korištenje baze podataka. Također, prelaskom na Kotlin, Android je uveo jednu novu inačicu, a to je Room, odnosno lokalna baza podataka koja je zapisana na sloju iznad *SQL Litea*.
- OpenGL |ES i SGL – služi kao mehanizam za 2D i 3D grafiku
- FreeType - omogućava korištenje fontova
- Web Kit – je web-pretraživački mehanizam koji omogućava prikazivanje web stranica i njihovo učitavanje
- SSL – zaštitna tehnologija koja pruža kriptiranu poveznicu između servera i web preglednika

2.2.3. Android radno okruženje

Nalazi se na istom sloju kao i biblioteke. Jedan od najvažnijih slojeva zato što sadrži glavne biblioteke i Dalvik virtualni stroj (DVM) koja pruža platformu za pokretanje aplikacija. Također, ovaj sloj je zaslužan za pokretanje više aplikacija istovremeno, a glavne biblioteke omogućuju pokretanje aplikacija pisanih u Java ili Kotlin programskim jezicima.

2.2.4. Linux jezgra

Poziv sustava pruža sloj između sklopovlja i programskih procesa. Ovaj sloj ima tri glavna zadatka. Prvi, pruža apstraktno sučelje sklopovlja programima. U slučaju pisanja ili čitanja iz datoteke jer aplikacije ne vode brigu o tipu diska, medijskog sadržaja ili tipa datotečnog sustava na kojem datoteka obitava. Drugo, pozivi sustava osiguravaju sigurnost sustava i stabilnost. S jezgrom koja se ponaša kao posrednik između resursa i programa, jezgra može pružati pristup temeljen na dopuštenjima, korisnicima i drugim kriterijima. Naprimjer, ovakav način može spriječiti aplikacije od nepravilnog korištenja sklopovlja, krađe resursa drugim procesima ili bilo koje druge štete sustava. Treće, jedan sloj između programa i ostatka sustava omogućava pružanje virtualnog sustava za procese [5].

Ovaj sloj je srce Android arhitekture i on upravlja upravljačkim programima (engl. driver) za kameru, memoriju itd. Pruža sloj apstrakcije između programa i sklopovlja te je zaslužan za upravljanje procesima, alokaciju memorije, napajanje itd. Neki od dijelova koji pripadaju ovom sloju su:

- Sigurnost – Linux jezgra upravlja sigurnošću između aplikacija i sustava
- Upravljanje memorijom - služi za alociranje i oslobađanje memorije pokretanjem aplikacije
- Upravljanje procesima – upravlja pokrenutim aplikacijama kako bi svaki zadatak procesa se uspješno obavio
- Internetski stog - rješava uspostavljenju komunikaciju s internetskom mrežom
- Model upravljačkih programa - služi kako bi svi dijelovi sklopovlja uspješno komunicirali.

3. ANDROID RAZVOJNI ALATI

Razvojni alati su programi koji automatiziraju kreiranje i izvršavanje aplikacija iz izvornog koda. Stvaranje ili građenje aplikacije se sastoji od prevođenja, povezivanja i pakiranja koda u iskoristivu i izvršivu formu. U manjim projektima, programeri često izvršavaju građenje aplikacije manualno dok to nije praktično kod većih projekata gdje je teško pratiti što treba izgraditi, kojim redoslijedom i koje su ovisnosti u cjelokupnom procesu. Koristeći automatizirani alat ovaj proces postaje konzistentniji [6].

Ovaj cijeli postupak se najčešće automatizira kod velikih projekata na način da se sa svakom promjenom na nekom od sustava za praćenje verzija (najčešće Git) ponovno pokreće čitav postupak. Glavni koraci u ovom postupku su :

1. Prevođenje izvornog koda u strojni jezik , odnosno binarni kod
2. Pakiranje binarnog koda
3. Pokretanje napisanih testova
4. Isporuka ili izdavanje na izdavačke sustave (npr. Google Play Store, App Center...)

Android aplikacije se mogu razvijati na Windows, Linux i Mac operacijskim sustavima, a prilikom izdavanja aplikacije ona se omotava u program s ekstenzijom .apk.

3.1. Android programski razvojni alat

Android SDK sadrži biblioteke koje povezuju aplikacije s ključnim Android dodacima poput uspostave poziva (pozivanje i primanje poziva), GPS funkcionalnosti i dopisivanje. Ove biblioteke čine jezgru SDK-a [7].

Android SDK omogućava razvijanje Android aplikacija i moguće ih je razviti u terminalu, ali zbog kompleksnosti najčešće se primjenjuje razvojno okruženje za razvijanje Android aplikacija. Glavno razvojno okruženje za razvijanje aplikacija je Android Studio koji s *Gradle* razvojnim alatom može razviti i ispraviti (engl. debug) programski kod aplikacije. Također, bitno je navesti i neke od programskih alata a to su:

- Android programski razvojni alat za kreiranje (engl. Android SDK Build Tool) - Služi za pretvaranje programskog koda u strojni jezik, prevođenje aplikacija i pokretanje testova
- Android simulator (engl. Android Emulator) - virtualni uređaj koji simulira stvarni Android uređaj u našem sustavu. Dolazi u raznim konfiguracijama kako po veličini zaslona i rezoluciji tako i po uređajima (mobilni uređaji, tableti ili dlanovnici, satovi i TV uređaji)

- Android programski razvojni alat platforme – prikazuje programske pogreške u stvarnom vremenu. Najviše se koristi kod testiranja. Također, sadrži i Android ispravljački most (ADB) koji pomaže u komunikaciji s uređajem te instalira aplikaciju na virtualni ili fizički uređaj

Programski alati su od velike pomoći prilikom izrade aplikacije jer upravo oni omogućuju ispravak i uvid u izrađeni dio. Također, Android Studio omogućava uvid u potrošnju memorije, internetskih podataka i skladištenja uređaja i s time pruža podršku u optimizaciji aplikacije.

3.2. Gradle razvojni alat

Android razvojni sustav prevodi aplikacijske resurse i izvorni kod te ih prevodi u i paket s ekstenzijom *.apk* koja se može testirati, izdati, potpisati i distribuirati. Android Studio koristi *Gradle* kao napredni razvojni alat za automatizaciju i upravljanje razvojnim procesima, a istovremeno dozvoljava programerima definiranje fleksibilnih prilagođenih konfiguracija za stvaranje. Svaka konfiguracija može konfigurirati svoj zaseban set koda i resursa, a i dalje reciklirati, odnosno ponovno koristiti zajedničke dijelove koji su poznati svim verzijama aplikacije. Android programski dodatak za *Gradle* radi zajedno s razvojnim alatom kako bi omogućio procese i konfiguraciju postavki koji su zasebni za stvaranje i testiranje Android aplikacija [8].

Mogućnosti koje *Gradle* pruža u razvijanju aplikacije su:

- Tipovi razvijanja (engl. Build Types) - Omogućava definiranje ponašanje aplikacije u životnom ciklusu razvijanja, pa možemo odrediti ponašanje za razvijanje (engl. development), testiranje (engl. staging) i izdavanje (engl. release). Na primjer, *R8* i *ProGuard* će biti onemogućeni u svakom od ciklusa osim u izdavanju kako bi aplikacija ostala u izvornom kodu, odnosno spriječili bi sloj zaštite koji pruža *R8* i *ProGuard*
- Okusi – Predstavljaju različite verzije aplikacije. Recimo da aplikacija ima besplatnu i plaćenu verziju, ako plaćena verzija koristi jedinstveni dio koda onda se postave okusi gdje će taj dio koda pripadati samo okusu za plaćenu verziju.
- Varijante razvijanja – Spaja tipove razvijanja s okusima pa će svaki okus imati sve tipove razvijanja. Za svaku od varijanti se mogu uvesti dodatne ovisnosti ili dodati konstante u *BuildConfiguration* datoteci.

- Manifest ulazi – Omogućava dinamično dodavanje vrijednosti ovisnih o varijanti razvijanja. Naprimjer, ime aplikacije, minimalna verzija razvojnog alata, ciljana verzija razvojnog alata itd.
- Ovisnosti – Upravlja ovisnostima u projektu iz lokalnih i udaljenih repozitorija kako ih programer ne bi morao samostalno tražiti, preuzimati i integrirati
- Potpisivanje – Razvojni sustav automatski potpisuje aplikaciju koja je u procesu građenja tako što uzima već poznati certifikat i ključ. Potpisivanje nije automatsko ako je tip razvijanja izdavanje.
- Smanjivanje koda i resursa – Razvojni sustav koristi alat *R8* koji prilikom smanjivanja briše sve neiskorištene dijelove koda i resursa. Također, *R8* mijenja ime svim klasama i varijablama u programskom kodu dodajući im kodna imena (tipično su to par slova a, b, abb, fd itd.). Programski kod se mijenja u cijelom projektu osim u dokumentima koji su zaštićeni *ProGuard* pravilima. Ovim putem se aplikacija optimizira i zaštiti te je ovo preporučen korak u slučaju izdavanja aplikacije.
- Podrška više aplikacija – Razvojni sustav omogućava automatizirano stvaranje različitih apk programa gdje se svaki sastoji samo od programskog koda i resursa potrebnih različitu rezoluciju ili aplikacijsko binarno sučelje (ABI).

U jednoj aplikaciji postoji više gradle dokumenata te svaki ima u imenu *.gradle,* a to su:

- *Settings.gradle* - nalazi se u korijenu projekta. Definira postavke repozitorija na razini projekta i govori Gradle razvojnom okruženju koje module treba uzeti pri građenju aplikacije.
- *Build.gradle* (najviše razine) - nalazi se u korijenu projekta. Definira ovisnosti koje se primjenjuju na sve module unutar aplikacije. Sadrži blok "dodatci" s kojim definira Gradleove ovisnosti koje su također poznate svim modulima u aplikaciji. Također, sadrži i kod za posao koji je zadužen da očisti direktoriji za građenje aplikacije.
- *Build.gradle* (razina modula) - nalazi se u svakom direktoriju modula. Omogućava definiranje svih mogućnosti razvijanja aplikacija koje su navedene u prethodnom odlomku. Poželjno je razvijati aplikacije u više modula kako bi se ubrzalo vrijeme građenja aplikacije i poboljšala arhitektura aplikacije. Moduli se mogu razlikovati u samoj svrsi ali najčešće modul predstavlja zaseban dio aplikacije što uključuje: bazu podataka, mrežni dio aplikacije, značajka(engl. feature) aplikacije itd.

Gradle Wrapper dopušta izvršavanje *Gradle* razvoja na uređajima gdje *Gradle* nije instaliran. Samim time je vrlo koristan za kontinuiranu integraciju servera [9]. Zbog toga objavljivanje novih verzija aplikacije te testiranje prilikom svakog spremanja na sustav za kontrolu inačica je olakšan.

4.OSNOVNE ANDROID APLIKACIJSKE KOMPONENTE

Osnovne aplikacijske komponente su zapravo blokovi od kojih se sastoji aplikacija. Naravno, aplikacija ne mora sadržavati svaki od navedenih blokova sve ovisi o njezinoj namjeni. Svaka komponenta ili blok predstavlja ulaznu točku kroz koju sustav ili korisnik može pristupiti aplikaciji. Neke komponente su ovisne o drugima ali svaka komponenta ima svoj zaseban životni ciklus, odnosno pripadaju joj metode kreiranja i uništenja.

4 osnovne komponente su:

- Aktivnosti (engl. Activity)
- Usluge (engl. Services)
- Prijamnik emitiranja (engl. Broadcast reciever)
- Pružatelj sadržaja (engl. Content provider)

4.1. Aktivnosti

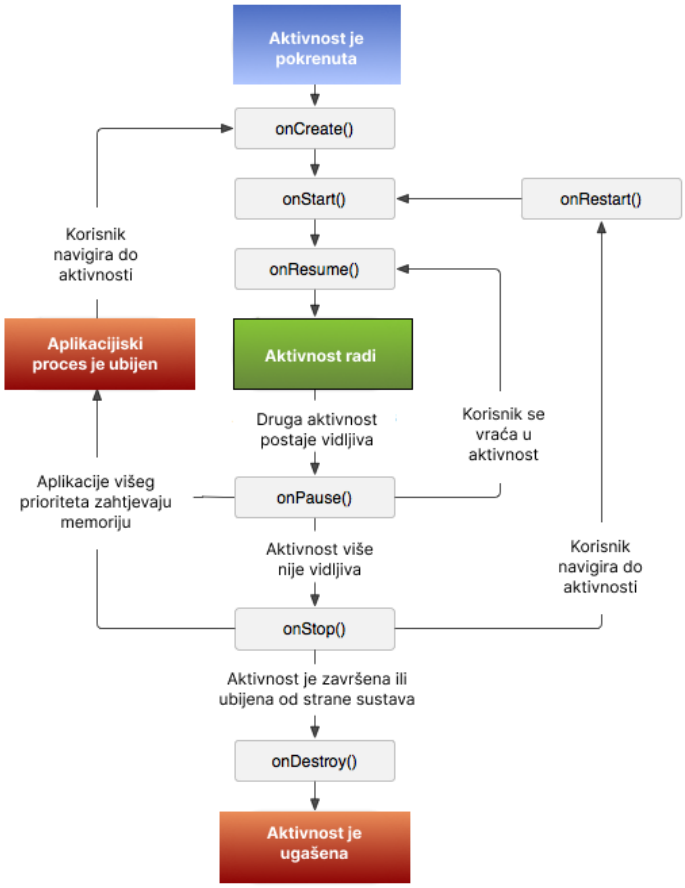
Aktivnost je ulazna točka za interakciju s korisnikom. Predstavlja jedan zaslon s korisničkim sučeljem [10]. Aplikacija se može sastojati od više aktivnosti i svaka predstavlja zasebnu cjelinu. Druge aplikacije mogu pristupati, odnosno ulaziti u aktivnosti aplikacije ako aplikacija dozvoli pristup. Čest primjer toga je ulazak u aplikaciju elektronske pošte iz druge aplikacije. Aktivnosti vode brigu o:

- Korisnikovim interakcijama (trenutnim prikazom na zaslonu) i o pozadinskim procesima koji održavaju trenutni zaslon
- Poznavanju prethodne procese koji sadrže vrijednosti zbog kojih bi se korisnik mogao vratiti
- Spremanju vrijednosti koje su prikazane na ekranu u slučaju uništenja aktivnosti zbog odlaska aplikacije u pozadinu.

Aktivnosti se smatraju teškima za aplikacijske resurse te je dobra praksa imati što manje aktivnosti moguće, a idealno je imati jednu aktivnost. Prilikom *ubijanja* aktivnosti (npr. rotacija zaslona) sve što je dio aplikacijskog *contexta (singleton klase*, pozadinski procesi, lokalne baze podataka itd.) će i dalje biti dostupno, a sve što je unutar aktivnosti ili fragmenta će biti uništeno. Iz tog razloga je dobra praksa svu poslovnu logiku staviti u viši sloj Android arhitekture [11]

Aktivnost kao i Fragment pruža korisničko sučelje korisniku. To korisničko sučelje se sastoji od pogleda (engl. View) ili složenica (engl. Composable) ovisno o načinu stvaranja, tj. Radi li se u

XML-u ili *Jetpack Composeu*. Pogled je objekt koji slika sliku na ekran i to nešto služi korisniku za interakciju. Ta slika može predstavljati gumb, tekstni okvir za upis ili tekstni okvir s tekstom, sliku, itd. Također, postoji i grupa pogleda, odnosno to je grupa koja drži ostale objekte pogleda kako bi definirala izgled sučelja. To su nevidljivi kontejneri za objekte pogleda koji definiraju kako će pogledi biti postavljeni [12]. Primjer jednoj je *RecyclerView* koji je također i korišten u aplikaciji.



Slika 4.1. Životni ciklus aktivnosti [13]

Čitav životni ciklus aktivnosti je definiran metodama aktivnosti. Svaka od ovih je poveznica koja se može prepisati da bi se odradio potreban posao prilikom promjene stanja aktivnosti [14]. Sa slike 4.1. možemo vidjeti metode koje se pozivaju unutar klase, a to su:

- *onCreate()* - Prva metoda kod prvog kreiranja aktivnosti, postavlja zaslon iz resursa
- *onStart()* - Poziva se na stvaranju aktivnosti ili u slučaju da je došlo do vraćanja iz smrti procesa aplikacije (smrt procesa se događa kada je aplikacija duže vrijeme u pozadini kako bi se sačuvali resursi)
- *onResume()* - Poziva se na stvaranju aktivnosti ili u slučaju da se aplikacija vraća, a otišla je u pozadinu, odnosno fokus je bio na nekom drugom elementu zaslona (u slučaju

podijeljenog zaslona aplikacija je vidljiva ali ako ima interakciju s drugom aplikacijom automatski aplikacija bez interakcije ide u pozadinu)

- *onPause()* - Poziva se svaki puta prilikom odlaska aplikacije u pozadinu.
- *onStop()* - Poziva se u slučaju da aplikacija više nije vidljiva
- *onDestroy()* - Poziva se ili završavanjem aplikacije zbog gašenja aplikacije ili zbog prelaska na drugu aktivnost

4.2. Usluge

Usluga je ulaz u aplikaciju koji drži istu pokrenutom u pozadini kako bi obavila dugotrajne zadatke ili udaljene procese [10]. Ona ne sadrži korisničko sučelje ali i dalje sadrži životni ciklus. U ovoj aplikaciji se može primijetiti usluga koja sluša Firebase server kako bi dohvatila podatke koji se odnose na uređaj u stvarnom vremenu. Također, usluga može poslužiti i za reprodukciju glazbe u pozadini. Razlikujemo dvije vrste usluga po životnom ciklusu:

- Startne usluge – usluga koju pokreće aplikacija i traje beskonačno, odnosno mora se zaustaviti sama ili pomoću druge komponente [15]. Primjer ovoga je sinkronizacija lokalne baze podataka s udaljenom bazom ili reprodukcija glazbe čak i kada korisnik nije u aplikaciji. Također, korisnik može i ne mora biti svjestan da startna usluga obavlja posao. U slučaju da korisnik treba znati da se odvija usluga dobra praksa je postaviti notifikaciju kada je korisnik izvan aplikacije. Najčešći primjer toga su aplikacije za reprodukciju glazbe. Dok usluge za koje korisnik nije svjestan omogućavaju sustavu više slobode i one mogu čak i dopustiti *ubijanje* usluge te ponovno pokretanje nakon nekog vremena kako bi se oslobodila RAM memorija na uređaju.
- Vezane usluge – pokrenute su zbog drugih aplikacija ili sustava koje ih žele iskoristiti. Tako recimo ako aplikacija A koristi uslugu aplikacije B sustav zna da mora održavati čitav proces aplikacije B i njezinu uslugu kako bi održao i aplikaciju A.

Usluge su vrlo fleksibilne te se zato pronalazi velika primjena poput slušanja obavijesti čuvari zaslona, pokretna pozadina itd.

4.3. Prijamnik emitiranja

Prijamnik emitiranja je komponenta koja omogućava sustavu da dostavi sustavu događaje koji se odvijaju izvan korisnikovog dometa, odnosno znanja [10]. Sustav može dostaviti događaje aplikaciji čak i kada ona nije pokrenuta. Taj slučaj također postoji u ovoj aplikaciji gdje zapravo ulazak u određeni prostor javlja aplikaciji kako je korisnik na određenoj lokaciji te aplikacija

obavještava korisnika o tome i kroz obavijest mu pruža ulaz u aplikaciju. Prijamnici ne sadrže korisničko sučelje ali mogu ponuditi statusnu traku obavijesti kako bi javili korisniku da se događaj odvio. Česta uporaba prijamnika je da pokrene uslugu koja je zakazana.

Aplikacije se mogu registrirati da primaju specifična emitiranja. Kada se emitiranje pošalje sustav automatski usmjerava emitiranja na aplikacije koje su se pretplatile da zaprime taj tip emitiranja [16].

4.4. Pružatelj sadržaja

Pružatelj sadržaja upravlja pristupom za centralni repozitoriji podataka. On je dio Android aplikacije i često pruža svoje korisničko sučelje za rad s podatcima. Također, pružatelj sadržaja je primarno namijenjen da se koristi od strane drugih aplikacija koje pristupaju pružatelju preko klijent objekta. Zajedno pružatelj i klijent nude konstantno, standardno sučelje podatcima, upravljaju komunikacijom i osiguravaju pristup podatcima [17].

Pružatelj sadržaja pruža dijeljeni set podataka aplikacije koji se može pohraniti u datotečni sustav, SQL bazu podataka, Internet ili bilo koji drugi način pohrane kojem aplikacija može pristupiti. Čest primjer korištenja pružatelja sadržaja je uvoz kontakata iz imenika ili slika iz galerije. S gledišta sustava pružatelj sustava predstavlja ulaz u aplikacijske podatke koji su identificirani po URI shemi. Aplikacija ima slobodu određivanja kako želi predstaviti podatke u samoj URI shemi. Takav način dopušta sustavu da dodjeljuje shemu bez potrebe da je aplikacija pokrenuta. U tom slučaju sustav je zadužen da aplikacija bude pokrenuta prilikom vračanja podataka. Prednost URI sheme je što osigurava sigurnosni model na način da aplikacija može pružiti na određeni dio podataka URI koji zahtjeva dopuštenje aplikacije koja mu želi pristupiti.

4.5. Namjera

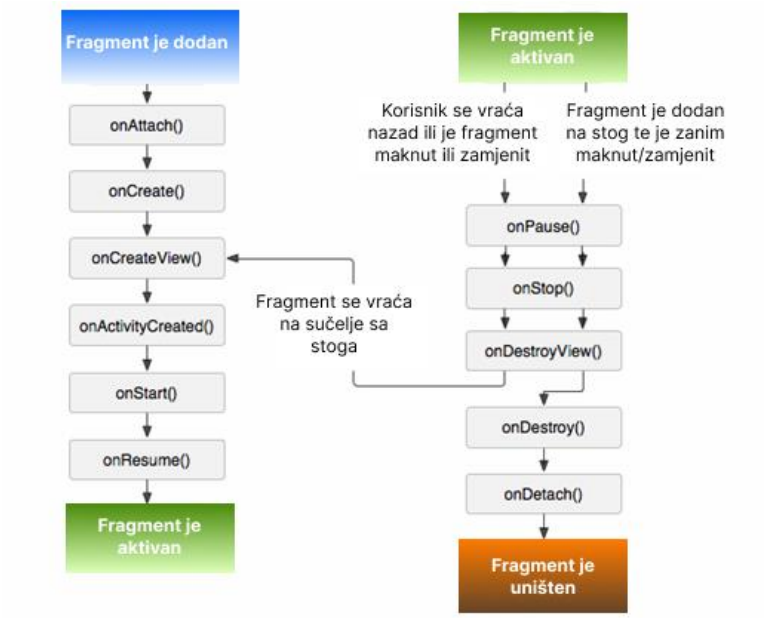
Namjera je apstraktni opis operacije koja se treba obaviti [18]. Iako ne pripada osnovnim komponentama namjera je jedan od ključnih elemenata aplikacije. Namjera pokreće aktivnosti, usluge i prijemnike emitiranja. Namjere su asinkrone poruke koje vežu zasebne komponente za vrijeme pokrenute aplikacije. Za usluge i aktivnosti namjera definira akciju koja će se obaviti i dodatno može dodati URI shemu od podataka na kojima će se obaviti određena akcija. S namjerom se mogu dohvaćati i pokretati druge aplikacije poput galerije, kamere, Internet preglednika itd.

Razlikujemo:

- Eksplicitne namjere - sadrže komponentu koja pruža komponentu koja će se pokrenuti. Često ne sadrže nikakve dodatne podatke te jednostavno predstavljaju narednu akciju u aplikaciji
- Implicitne namjere - ne sadrže komponentu, ali moraju sadržavati dovoljno informacija za sustav kako bi znao koja od mogućih komponenti je najbolja za pokretanje te namjere.

4.6. Fragmenti

Fragmenti predstavljaju reciklirajući dio aplikacijskog korisničkog sučelja. Definiraju i upravljaju svojim sučeljem, imaju svoj životni ciklus i mogu upravljati svojim ulaznim događajima. Ne mogu živjeti sami od sebe te moraju biti dio aktivnosti ili drugog fragmenta [19].



Slika 4.2. Životni ciklus fragmenta [19]

Kako su aktivnosti osnovna komponenta i kao što je prethodno rečeno preporučeno je u teoriji imati samo jednu aktivnost u cijeloj aplikaciji, tu nastupaju fragmenti. Fragmenti omogućavaju recikliranje (ponovno korištenje) programskog koda i omogućavaju modularnost. Jedna aktivnost može sadržavati jedan ili više fragmenata (najčešće jedan) te samim time olakšava posao u izradi kompleksnog korisničkog sučelja. Kako Android omogućava prilagođavanje i kreiranje vlastitih komponenti korisničkog sučelja često se dovodi u pitanje treba li koristiti fragment koji sadrži svoju kompleksnost ili koristiti prilagođene komponente. Odgovor je oboje. Ako se prati trenutna najbolja praksa u izradi aplikacija jedna aktivnost će prikazivati sve

fragmente koji će predstavljati sva sučelja dizajnirana u aplikaciji, a prilagođene komponente korisničkog sučelja će se nalaziti u fragmentima. Također, roditeljski ili fragment na aktivnosti može sadržavati jedan ili više fragmenata koji predstavljaju njegovu *djecu*. Najveća kompleksnost koju dodavanje fragmenata pruža je briga o još jednom životnom ciklusu ali ako se koristi *Jetpack Compose* dodavanje fragmenata se može u potpunosti izbjeći. Kao što se može primijetiti iz slike 4.2 može vidjeti životni ciklus jednog fragmenta se sastoji od nešto više metoda nego životni ciklus aktivnosti. Iako se čini kompleksnijim najbitnije je zapamtiti kako se kod fragmenta korisničko sučelje postavlja najranije u *onCreateView()* metodi.

5. APLIKACIJA

Aplikacija je zamišljena kao savjet (lat. Conseo), odnosno personalizirani pomoćnik korisniku u cilju pronalaska novih ljudi i lokacija. Omogućava korisniku da vidi sve dodane kafiće, restorane i događaje u udaljenosti koju on odabire. Također, čim korisnik priđe blizu nekog kafića, restorana ili događaja aplikacija ga obavještava o njegovoj blizini. Korisnik može ostavljati komentare na svakoj od navedenih lokacija te samim time dijeliti svoje mišljenje s drugima. Svaka od dodanih lokacija se može vidjeti na kartama i u slučaju dodatnih pitanja korisnicima je omogućen tekstualni razgovor između njih i kreatora navedene lokacije. Korisnik se u početku može prijaviti te omogućiti biometrijsko spremanje njegovih podataka, odnosno kako bi izbjegao pisanje svoje elektroničke pošte i zaporke omogućeno mu je spremanje i upisivanje za njega čim potvrdi svoj identitet pomoću otiska prsta ili lica. Ako korisnik želi kreirati račun mora navesti osobne podatke poput imena, spola i godina. Također, može ali i ne mora postaviti fotografiju koja je vidljiva korisnicima koji s njim budu razgovarali preko aplikacije. Svi podaci se spremaju na udaljenu bazu podataka *Firebase* koja vodi računa o očuvanju njihovog integriteta, i o pristupu istima. Dok Conseo daje udaljenost, odnosno mogućnost filtriranja događaja, kafića i restorana Trip Advisor prikazuje sve podatke i zanimljivosti o određenom mjestu. Također, prednost Conseoa nad Trip Adviserom je ta što Conseo prikazuje i događaje koji nisu javni, odnosno uključuje privatne zabave i tulume.

5.1. Priprema razvoja aplikacije

Kako bi se započela aplikacija prvobitno je bilo postaviti Android Studio i Firebase bazu podataka. Nakon toga je uslijedila priprema testne opreme, a u ovom slučaju to su bili fizički uređaji Xiaomi Redmi 9, Samsung Galaxy A12 i Huawei P20 Lite koji su bili povezani s Android Studiom preko USB ili bežične veze. Nakon postavljanja osnovnih komponenata uslijedilo je planiranje izvedbe same aplikacije, odnosno planiranje svih kolekcija u *Firebase* bazi podataka i arhitektura same aplikacije. Za registraciju i prijavu u aplikaciju se pobrinula *Firebase* provjera autentičnosti (engl. *Firebase Authentication*), za spremanje podataka *NoSQL Firestore* baza podataka i za spremanje slika *Firebase* skladištenje (engl. *Firebase Storage*). Sama aplikacija se bazira na arhitekturi MVVM (*Model – View – ViewModel*) koja je omogućila odvajanje programskog koda i povezanosti između komponenti. S MVVM arhitekturom aplikacija se odvojila na tri dijela, a to su korisničko sučelje, poslovna logika i podaci. Korisničko sučelje je vodilo brigu o prikazivanju podataka i navigaciji po aplikaciji, Poslovna logika je odvijala operacije na podacima gdje je zapravo pripremala podatke za korisničko sučelje ili slala podatke na udaljenu bazu ili skladište. Sloj podataka se sastoji od više dijelova

poput modela podataka i odabiru spremanja i vraćanja podataka. Također u sloj podataka pripadaju i dva zasebna modula koja su odvojena kako bi se strogo definirala arhitektura. Ta dva modula su baza podataka i reprezentacijsko stanje prijenosa internetskih podataka (engl. *REST API*). Zbog bolje preglednosti aplikacijski modul je podijeljen u zasebne pakete od kojih svaki predstavlja jednu inačicu, odnosno dodatak (engl. feature) aplikacije.

U *Model-View-ViewModel* arhitekturi *Model* je zaslužan za podatkovne modele s klijentske strane. Sastoji se od objekata čiji atributi i neke varijable sadrže podatke u memoriji Neki od tih podataka povezuju druge modele i kreiraju graf objekata koji se ponašaju kao model objekta. Zadatak *Viewa* je definirati strukturu korisničkog sučelja te se sastoji od statičkih i dinamičkih dijelova. Statički dijelovi su komponente i površina komponenti od kojih se Pogled sastoji. Dinamički dio su animacije i promjene stanja koje su definirane kao dio *Viewa*. Na kraju *ViewModel* je centralna točka MVVM aplikacije. On predstavlja primarnu odgovornost u pružanju podataka od sloja Model do sloja *View*.. Također omogućava korisniku interakciju s podatcima. Zaslužan je i za *enkapsulaciju* poslovne logike koja je potrebna *Viewu* [20].

5.2. Geofencing

Geofencing je dodatak koji omogućava aplikaciji da reagira na određeni događaj unutar aplikacije. To je moguće napraviti pomoću prijemnika emitiranja. U aplikaciji se događaj predstavlja kao ulazak u zonu ili zadržavanje u zoni gdje zona predstavlja radijus oko određene lokacije u kojoj se nalazi objavljeni događaj, kafić ili restoran.

```
fun getGeofencingRequest(): GeofencingRequest {
    return GeofencingRequest.Builder().apply {
        setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_DWELL or
GeofencingRequest.INITIAL_TRIGGER_ENTER)
        addGeofences(geofenceList.value!!)
    }.build()
}
```

Programski kod 5.1. Kreiranje zahtjeva za geofencing

Kao što je moguće vidjeti na programskom kodu 5.1. u funkciji *getGeofencingRequest* gdje mi vraćamo *geofencing* zahtjev postavlja se u graditelju zahtjeva da je okidač (engl. *trigger*) *INITAL_TRIGGER_DWELL* (korisnik se zadržava unutar radijusa *geofencinga*) ili *INITAL_TRIGGER_ENTER (*korisnik ulazi u radijus *geofencinga*). Svaki prijemnik emitiranja mora prepisati funkciju *onReceive* kao u programskom kodu 5.2. gdje zaprima *context* koji

predstavlja trenutno stanje sustava i namjera (engl. *Intent*), odnosno poruku unutar aplikacije koja kao što je ranije navedeno može sadržavati akciju i podatke.

```
override fun onReceive(context: Context?, intent: Intent?) {
    val geofencingEvent = intent?.let { GeofencingEvent.fromIntent(it) }
    if (geofencingEvent?.hasError() == true) {
        val errorMessage = GeofenceStatusCodes
            .getStatusCodeString(geofencingEvent.errorCode)
        Log.e(TAG, errorMessage)
        return
    }
    // Get the transition type.
    val geofenceTransition = geofencingEvent?.geofenceTransition
    // Test that the reported transition was of interest.
    if (geofenceTransition == Geofence.GEOFENCE_TRANSITION_ENTER ||
        geofenceTransition == Geofence.GEOFENCE_TRANSITION_DWELL
    ) {
        // Get the geofence that were triggered. A single event can trigger
        // multiple geofence.
        val triggeringGeofence = geofencingEvent.triggeringGeofences
        // Get the transition details as a String.
        val geofenceRequestID = triggeringGeofence.first()?.requestId
        if (geofenceRequestID != null && Firebase.auth.currentUser != null) {
            sendGeofenceEnteredNotification(geofenceRequestID)
        }
    } else {
        Log.e(TAG, "Geofence Transition Invalid $geofenceTransition")
    }
}
```

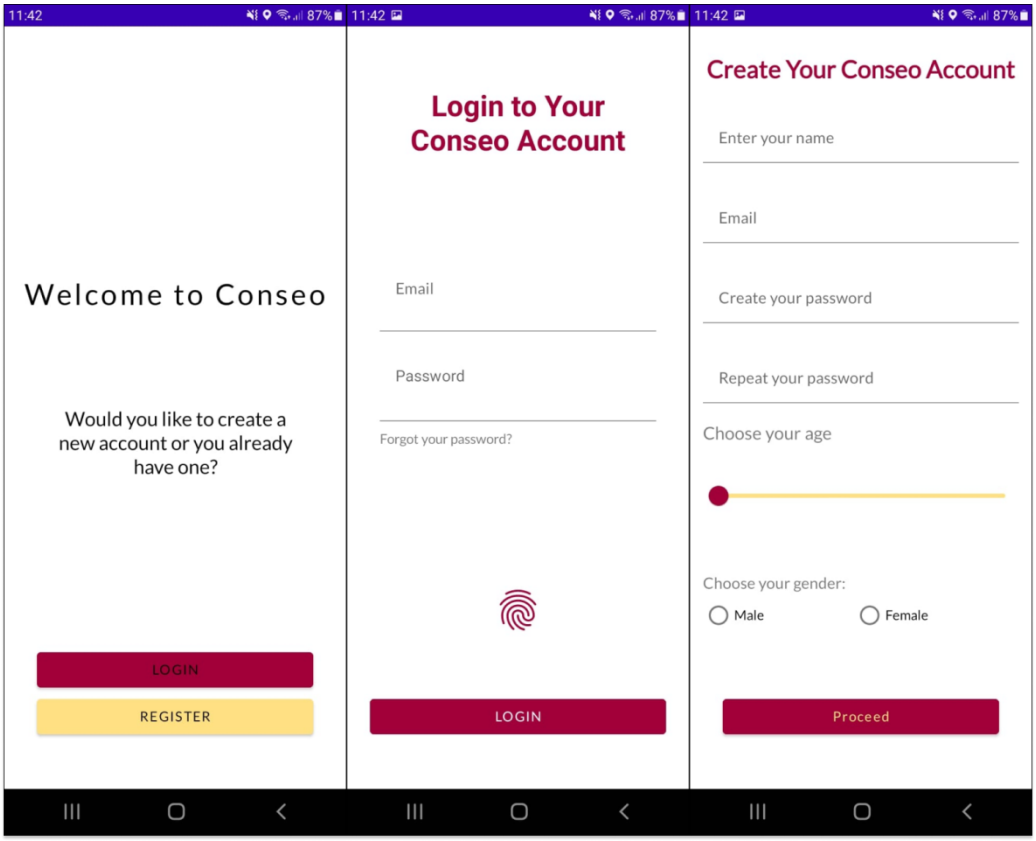
Programski kod 5.2. Geofencing prijem emitiranja

Prvobitno se stvara *geofencing* događaj pomoću ugrađene funkcije *GeofencingEvent.fromIntent(intent)* te se zatim provjerava ukoliko ima određenih grešaka kako bi se mogao prekinuti tijek izvođenja funkcije te javljanja da je došlo do pogreške. Zatim, bitno je dobiti informaciju o kakvoj tranziciji se radi kako ne bi prilikom korisnikovog izlaženja iz radijusa *geofencinga* javljali korisniku da je u blizini određene lokacije ako se udaljava od iste. Te ukoliko tranzicija zadovoljava zahtjeve pokušava se dobiti identifikacijski broj od ulaska događaja. Taj identifikacijski je ranije zadan da predstavlja identifikacijski broj od lokacije kako bi se mogla povezati zajedno s obavijesti koju korisnik dobiva prilikom okidanja događaja. Također, u obavijest se stvara pomoću namjera koje se okidaju prilikom pritiska na obavijest. Namjera sadrži dva dodatka podatka, a to su identifikacijski broj i ime lokacije kako bi ulaskom u aplikaciju bilo prepoznato koji fragment treba prikazati i s kojim podatcima. Kako bi se mogao koristiti *geofencing* potrebno je od korisnika zatražiti dopuštenja za dohvaćanje lokacije koje također treba navesti i u *Manifestu*. Ukoliko korisnik ne dozvoli lokaciju *geofencing* neće

reagirati jer neće dobiti informaciju o korisnikovoj lokaciji. Ukoliko se pokuša dobiti lokacija iako korisnik nije dao dopuštenje aplikacija se automatski ruši te izbacuje *RuntimeException.*

5.3. Prijava i registracija

Kako je Conseo zasebna aplikacija te ne pruža web aplikaciju jedina mogućnost registracije je putem aplikacije. Registraciju i prijavu uvelike vodi *Firebase* tehnologija sa svojom već ugrađenom poslovnom logikom te je potrebno pozvati tu istu tehnologiju i manualno spremiti korisnika u bazu podataka. Kao što je već napisano prilikom prijave uvedeno je inovativno rješenje, a to je da korisnik može spremiti svoje podatke za prijavu lokalno u mobitel gdje se ti isti podaci kriptiraju i prilikom biometrijske prijave gdje korisnik potvrđuje svoj identitet ti podaci se pružaju kako bi se ubrzalo vrijeme pisanja svojih podataka. Prilikom registracije korisnik pruža osnovne podatke te dodatno može odabrati sliku i odabrati koje obavijesti želi primati osim obavijesti dobivene poruke. Ukoliko je korisnik već prijavljen on se automatski poslije ulazne animacije prebacuje na glavni dio aplikacije, odnosno odmah može vidjeti dodane kafiće itd.



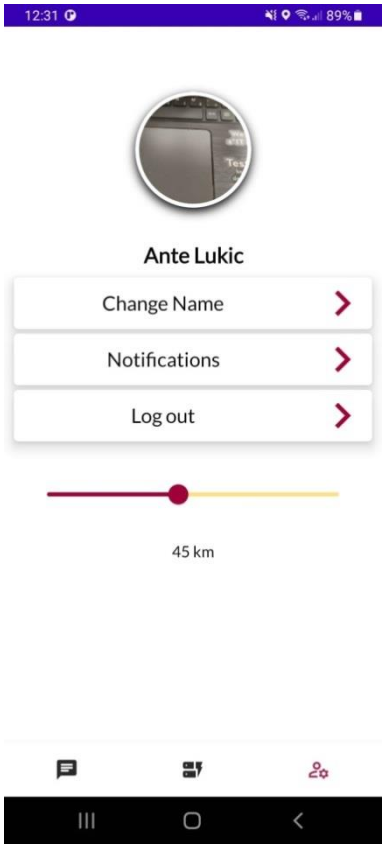
Slika 5.1. Izgled sučelja za prijavu i registraciju

Kao što se može vidjeti na slici 5.1. prvo sučelje koje se prikazuje omogućuje grananje puta aplikacije u dva dijela, a to su prijava i registracija. Na početnom sučelju slova se dodaju jedno

po jedno te prilikom dovršetka rečenice se podižu gore. Također, može se primijetiti kako je za primarnu i sekundarnu boju odabrana tamno crvena i žuta.

5.4. Postavke

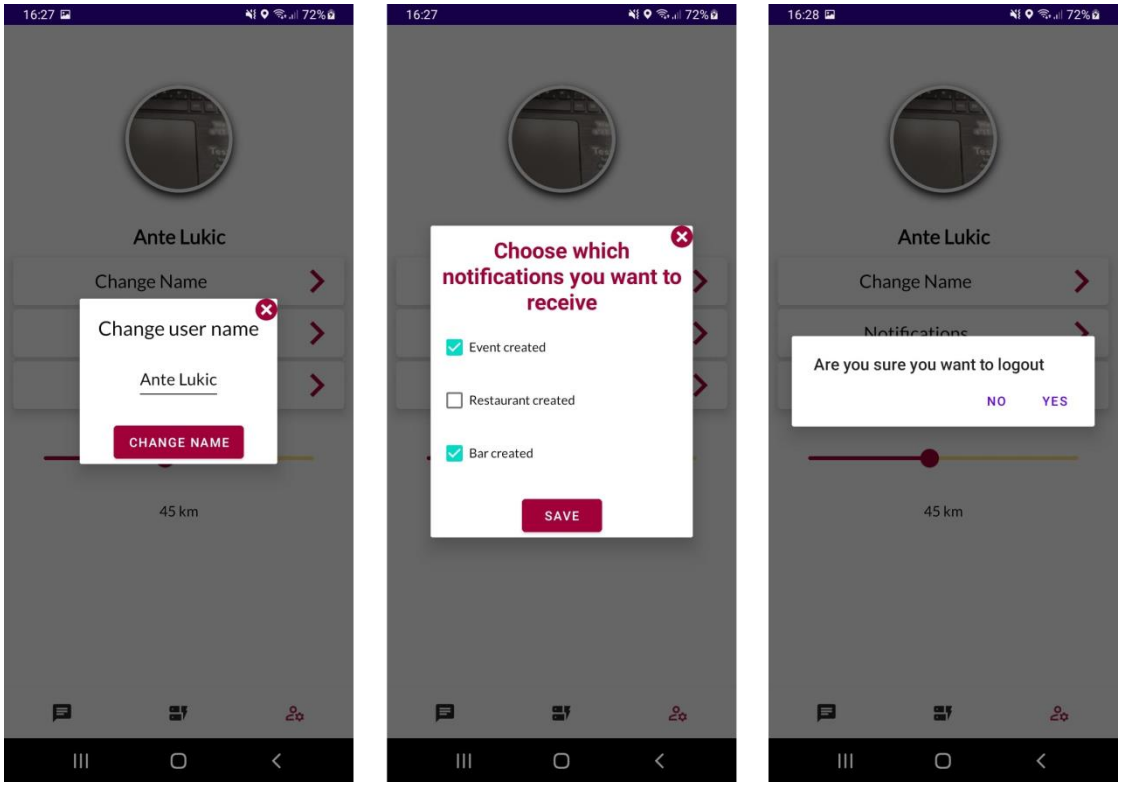
Kao što se može vidjeti iz slike 5.2 postavke pružaju nekolicinu mogućnosti poput promjene imena, odabira obavijesti koje će se prikazivati uz obavijesti razgovora, odjava te odabir maksimalne udaljenosti lokacija koje se prikazuju. Maksimalna udaljenost je 100 kilometara. Za svaku od kartica je korišten *MaterialCardView*. *MaterialCardView* pruža *Material* stilove za karticu te će prikazati zadani *Material* stil bez korištenja *style* zastavice [21]. Pritiskom na bilo koju od navedene tri kartice pojavljuje se *dialog* sa svojim korisničkim sučeljem. Za promjenu imena i obavijesti se pojavljuje *dialog* s prilagođenim korisničkim sučeljem dok se za odjavu koristi *dialog* tipa *AlertDialog*.



Slika 5.2. Izgled korisničkog sučelja za postavke

Dialog je mali prozor koji priupita korisnika da donese odluku ili unese dodatnu informaciju. *Dialog* ne popunjava ekran i uobičajeno se koristi za događaje koji zahtijevaju korisnike da izvedu određenu akciju prije nastavka. *Alert Dialog* je također dio *dialog* skupine te on može prikazati naslov, maksimalno tri gumba, listu elemenata za odabiranje ili prilagođeno sučelje [22]. Kao što se može vidjeti sa slike 5.3. izgled *Alert Dialoga* je drugačiji te je on ovisan o

svakom uređaju, odnosno na svakom uređaju izgleda drugačije dok *dialog* za obavijesti i promjenu imena je uvijek identičan.



Slika 5.3. Izgled dialoga za promjenu imena, obavijesti i odjavu

```
fun logoutDialog(){
    AlertDialog.Builder(requireContext())
        .setTitle("Are you sure you want to logout")
        .setPositiveButton("Yes") { _, _ ->
            settingsViewModel.logout()
            startActivity(Intent(requireContext(), MainActivity::class.java))
        }
        .setNegativeButton("No") {_, _ -> }
        .create()
        .show()
}
```

Programski kod 5.3. dialog za odjavu

Kao što se može vidjeti u programskom kodu 5.4. za povezivanje korisničkog sučelja i programskog koda se koristi *Data Binding* i njegova inačica *two-way binding* gdje se smanjuje statičan i nepotreban kod te se iz XML jezika mogu pozvati funkcije kao što vidimo u *onClick* medoti u XML. Također, da bi to bilo moguće potrebno je deklarirati varijablu (u ovom slučaju *view*) u XML-u pod *<data><data/>* dijelom kako bi se mogle pozvati metode i uzeti atributi koji pripadaju toj klasi (u ovom slučaju u *SettingsFragmentu)*. Manipuliranje obavijestima, odnosno omogućavanje zaprimanja samo određenih obavijesti je ostvareno na način da se obavijesti šalju vezane za temu.

```
<com.google.android.material.card.MaterialCardView
    android:id="@+id/Fragment_Settings_Logout"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="5dp"
    app:cardElevation="10dp"
    android:onClick="@{() -> view.logoutDialog()}"
    app:layout_constraintEnd_toEndOf="@id/Fragment_Settings_ChooseNotifications"
    app:layout_constraintStart_toStartOf="@id/Fragment_Settings_ChooseNotifications"
    app:layout_constraintTop_toBottomOf="@id/Fragment_Settings_ChooseNotifications">
```

Programski kod 5.4. XML kod kartice za odjavu

```
fun saveNotificationsChoice() = viewModelScope.launch(Dispatchers.IO) {
    if (eventsChecked.value == true) {
        FirebaseMessaging.getInstance().subscribeToTopic("events")
    } else {
        FirebaseMessaging.getInstance().unsubscribeFromTopic("events")
    }
    if (restaurantsChecked.value == true) {
        FirebaseMessaging.getInstance().subscribeToTopic("restaurants")
    } else {
        FirebaseMessaging.getInstance().unsubscribeFromTopic("restaurants")
    }
    if (barsChecked.value == true) {
        FirebaseMessaging.getInstance().subscribeToTopic("bars")
    } else {
        FirebaseMessaging.getInstance().unsubscribeFromTopic("bars")
    }


    appPrefs.putBoolean(key = AppPrefs.BARS_KEY, barsChecked.value ?: false)
    appPrefs.putBoolean(key = AppPrefs.EVENTS_KEY, eventsChecked.value ?: false)
    appPrefs.putBoolean(key = AppPrefs.RESTAURANTS_KEY, restaurantsChecked.value ?: false)
}
```

Programski kod 5.5. Metoda za stvaranje obavijesti

Dakle, prema programskom kodu 5.5. dok se u razgovorima obavijesti šalju na korisnikov token te tu obavijest zaprima mobitel s tim tokenom, kod slanja obavijesti na temu svi korisnici koji su pretplaćeni na temu dobivaju obavijest. Samim time korisnik odabirom u *dialogu* može se pretplatiti i obrisati pretplatu s teme, a kako bi svaki puta kada mu se prikazuje *dialog* bilo prikazano na što je već pretplaćen njegovi odabiri se spremaju u lokalnu datoteku pod ključem i *boolean* vrijednosti. Iako je pisanje u *Shared Preferences thread safe* i dalje unutar *coroutines* prelazimo na *Input/Output Thread* kako bi oslobodili *Main Thread* od dodatnog posla jer je on zadužen samo za *crtanje* po ekranu, tj. za brigu o korisničkom sučelju. Za upisivanje koristi *apply* funkcija, a ona je asinkrona te se neće odmah izvršiti

5.5 Razgovori (engl. Chats)

Kako bi se korisnici mogli dogovoriti oko mogućih nesporazuma aplikacija u sebi sadrži razgovore (eng. *Chats*). Svaki razgovor se može kreirati iz lokacije odnosno prilikom gledanja detalja lokacije. Rješenje za razgovore, odnosno kako prikazivati poruke s lijeve i desne strane je bio *RecyclerView* s dva *ViewHoldera*. Unutar *RecyclerViewa* se provjerava je li identifikacijski broj pošiljatelja poruke isti kao od korisnika prijavljenog u aplikaciju. Ukoliko je to znači da je korisnik poslao poruku, a u suprotnom predstavlja da je korisnik zaprimio poruku. Što se tiče pozadinske strane slanja poruka proces se odvija dva puta, odnosno prvobitno se spajaju dva identifikacijska broja korisnika gdje je prvi dio broja broj jednog korisnika, a drugi dio broj drugog. Na taj način se dobiva identifikacijski broj razgovora te se sprema u bazu. Poslije toga se uzima prvo broj drugog korisnika, a zatim broj prvog korisnika te se proces ponavlja. Na taj način oba korisnika vrlo lako mogu prepoznati da imaju otvoren razgovor.

Također, u adapteru je potrebno osim tri obavezne funkcije (*getItemCount, onBindViewHolder* i *onCreateViewHolder()*) potrebno je prepisati i funkciju *getItemViewType* koja provjerava je li poruka poslana ili primljena. Kako se ona poziva prije *onCreateViewHolder* to nam omogućava da u *onCreateViewHolderu* provjerimo tip te prema njemu stvorimo *ViewHolder* koji će se koristiti u *onBindViewHolderu.* Kako bi se omogućio razgovor s primanjem poruka u stvarnom vremenu u obzir su dolazile dvije opcije. Prva je bila dodavanje praćenja stanja određene kolekcije u bazi koja je odgovarala trenutno aktivnom razgovoru. To je omogućeno pomoću *Kotlinove* komponente koja se zove *Flow*. Ali zbog već odrađene infrastrukture i jednostavnosti u aplikaciji se dolazeća poruka automatski provjerava je li pripada razgovoru. Ako pripada, ona se dodaje u listu koja se predaje u adapter od *RecyclerViewa*. Na taj način se izbacila dodatna implementacija koda te se iskoristio već postojeći kod. Svaki korisnik prilikom slanja poruke šalje putem *REST API-a* na *Firebase* server koji te podatke preusmjerava na token koji je poslan zajedno s podatcima. Podaci koje korisnik šalje su samo osnovni dijelovi poruke poput identifikacijskog broja pošiljatelja, same poruke, tokena primatelja poruke te imena pošiljatelja kako bi se mogao prikazati u obavijesti. Metoda *onBindViewHolder* postavlja izgled *ViewHoldera* odnosno jednog elementa *Recycler Viewa* te se sami *ViewHolder* reciklira i time štedi resurse. Za svaki element metoda *onBindViewHolder* će biti pozvana tek kada sami element dođe na red za prikaz. Ona se poziva neovisno o tome je li element već jednom bio prikazan zato što se njegovim nestajanjem s ekrana većina njegovih podataka uništava te se čuva samo referenca u memoriji.

```
class MessageRecyclerAdapter(private val messages: ArrayList<MessageEntity>)
    : RecyclerView.Adapter<RecyclerView.ViewHolder>() {
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder {
        return if(viewType == ITEM_RECEIVE){
            ReceiveMessageViewHolder(
                ItemReceiveLayoutBinding.inflate(
                    LayoutInflater.from(parent.context),
                    parent,
                    false))
        } else{
            SentMessageViewHolder(
                ItemSendLayoutBinding.inflate(
                    LayoutInflater.from(parent.context),
                    parent,
                    false))
        }
    }
    override fun onBindViewHolder(holder: RecyclerView.ViewHolder, position: Int) {
        if(holder.javaClass == SentMessageViewHolder::class.java){
            val viewHolder = holder as SentMessageViewHolder
            viewHolder.sentMessage.text = messages[position].message
        }
        if(holder.javaClass == ReceiveMessageViewHolder::class.java){
            val viewHolder = holder as ReceiveMessageViewHolder
            viewHolder.receiveMessage.text = messages[position].message
        }
    }
    override fun getItemCount(): Int = messages.size
    override fun getItemViewType(position: Int): Int {
        val currentMessage = messages[position]
        return if(currentMessage.senderID == FirebaseAuth.getInstance().currentUser?.uid.toString()){
            ITEM_SENT
        } else {
            ITEM_RECEIVE
        }
    }
    inner class SentMessageViewHolder(binding: ItemSendLayoutBinding):
RecyclerView.ViewHolder(binding.root){
        val sentMessage = binding.ItemSendSentMessage
    }
    inner class ReceiveMessageViewHolder(binding: ItemReceiveLayoutBinding):
RecyclerView.ViewHolder(binding.root){
        val receiveMessage = binding.ItemRecieveMessage
    }
    private companion object{
        private const val ITEM_RECEIVE = 1
        private const val ITEM_SENT = 2
    }
}
```

Programski kod 5.6. RecyclerView Adapter za prikazivanje poruka

```
class FirebaseService: FirebaseMessagingService() {
    override fun onNewToken(token: String) {
        super.onNewToken(token)
        val prefs = getSharedPreferences(
            getString(R.string.token),
            Context.MODE_PRIVATE
        )
        prefs.edit().putString(
            getString(R.string.token_key), token
        ).apply()
    }
    override fun onMessageReceived(message: RemoteMessage) {
        super.onMessageReceived(message)
        remoteMessage.postValue(message)
        //SendNotificationUpdateChat
        sendNotification(message)
    }
    private fun sendNotification(message: RemoteMessage){
        val channelId = "ConseoChannel"
        val channel = NotificationChannel(
            channelId,
            "my_notification",
            NotificationManager.IMPORTANCE_HIGH
        )
        channel.enableLights(true)
        channel.lightColor = Color.GREEN
        channel.enableVibration(false)
        val intent = Intent(this, MainActivity::class.java)
        intent.putExtra("receiverID", message.data["senderID"])
        val pendingIntent =
            PendingIntent.getActivity(this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT or
PendingIntent.FLAG_IMMUTABLE)
        val mNotificationManager = getSystemService(Context.NOTIFICATION_SERVICE) as
NotificationManager
        val builder = NotificationCompat.Builder(this, channelId)
            .setSmallIcon(R.mipmap.ic_launcher)
            .setContentTitle(message.data["title"])
            .setContentText(message.data["message"])
            .setPriority(NotificationCompat.PRIORITY_MAX)
        builder.setContentIntent(pendingIntent).setAutoCancel(true)


        mNotificationManager.createNotificationChannel(channel)//Notice this
        mNotificationManager.notify(123, builder.build())
    }
    companion object{
        val remoteMessage = MutableLiveData<RemoteMessage>()
    }
}
```

Programski kod 5.7. Servis za primanje poruka

U programskom kodu 5.7. je prikazana programska klasa koja predstavlja uslugu za Firebase *Cloud Messaging*. To je usluga koja obavlja radnju i kada je aplikacija ugašena kako bi korisnik mogao biti obaviješten i kada je van aplikacije. Kako je ovo servis iz Firebase biblioteke ona vodi brigu o njegovom aktiviranju, odnosno ne mora se nigdje unutar koda navesti da se servis pokrene. Ovaj servis vodi brigu o generiranju te spremanju tokena u lokalnu datoteku, primanju poruke te slanju obavijesti korisnika. Za obavijest se može primijetiti kako nema navedeno sučelje, odnosno izgled obavijesti. To je zato što se koristi obavijest sustava te će ona imati zaseban izgled na svakom uređaju. Također, može se primijetiti kako klasa sadrži *companion object* s varijablom tipa *MutableLiveData* koja sadrži unutar sebe tip *RemoteMessage*. Dakle, *companion object* je ekvivalent *static* oznaci u Java ili C# programskom jeziku. Samim time ovoj varijabli se pristupa preko klase, a ne preko instance klase. Tip *MutableLiveData* je tip koji se može promatrati te će on promjenom svoje vrijednosti obavijestiti sve svoje promatrače. Dio promatranja promjene vrijednosti je ključan za održavanje razgovora u stvarnom vremenu. Baš ta inačica je omogućila da aplikacija prepozna kada treba reagirati te prikazati dolazeću poruku. *Firebase Cloud Messaging* (FCM) je platforma za dopisivanje koja omogućava sigurno slanje poruka bez troška. Koristeći FCM omogućeno je obavještavanje aplikacije da je nova elektronička pošta ili neki drugi podaci spremni za sinkronizaciju. Također, mogu se slati poruke obavijesti kako bi se potaknula ponovna interakcija korisnika [23].

5.6. Mjesta

Mjesta su glavni dodatak u aplikaciji zato što se iz njih mogu stvarati razgovori, a postavkama se može manipulirati s udaljenošću i samim time i brojem prikazanih mjesta. Također, mjesta omogućuju pregled detalja kojima kreator mjesta može pobliže opisati restoran kafić ili događaj. U detaljima mjesta je prikazan ime, opis i komentari s kojima se omogućava korisnicima ostavljanje recenzije. Povratnom informacijom pomoću komentara kreator mjesta dobiva povratnu informaciju isto kao i budući gosti tog mjesta. Samim time omogućavanjem razgovora kreator može sam u jedan na jedan razgovoru s korisnikom na još bolji način opisati korisniku detalje mjesta. Iznad kartice koja prikazuje mjesta postoji još jedna kartica koja prikazuje animaciju ovisno o vrsti mjesta. Tako će naprimjer za kafić prikazivati šalicu kave, za restoran će prikazivati tanjur, a za događaj će prikazivati disko kuglu. Te animacije su ostvarene pomoću vanjske biblioteke koja se naziva Lottie. S *Lottie* bibliotekom dobiva se novi dio koji je moguće dodati u korisničko sučelje, a zove se *LottieAnimationView*. U toj komponenti moguće je prikazati animaciju pomoću internetske poveznice ili pomoću imena JSON datoteke koja mora

biti spremljena unutar aplikacije. U ovoj aplikaciji se sve *Lottie* animacije prikazuju pomoću datoteka koje su spremljene u *assets* direktoriju.

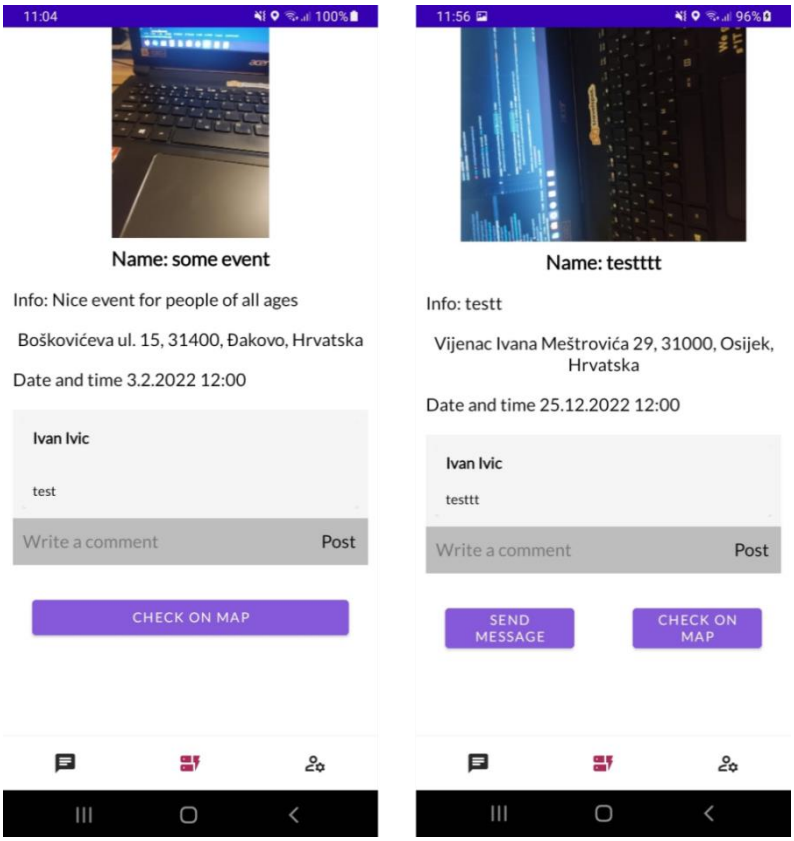
Slika 5.8. Izgled popisa mjesta

```
<com.airbnb.lottie.LottieAnimationView
    android:id="@+id/Item_ServiceAnimations_LottieAnimation"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:lottie_autoPlay="true"
    app:lottie_loop="true" />
```

Programski kod 5.8. XML kod komponente za prikazivanje animacije

Također, *Lottie* animacije omogućavaju automatsko pokretanje i ponovno pokretanje animacije prilikom završetka kao što je prikazano na programskom kodu 5.8.. Prikazivanje različitih

animacija je ostvareno isto kao i prikazivanje različitih tipova mjesta, pomoću *ViewPagera*. *ViewPager* je menadžer sučelja koji omogućava korisniku listanje stranica podataka u lijevom i desnom smjeru [24]. *ViewPager* se može uskladiti s *TabLayoutom* gdje se pritiskom na *tab* korisničko sučelje mijenja, odnosno *ViewPager* prebacuje trenutno prikazano sučelje.



Slika 5.9. Izgled sučelja za prikaz detalja mjesta

Na slici 5.9 se može primijetiti kako prikaz detalja mjesta sadrži dva različita izgleda. Izgled ovisi o korisniku. Na lijevom sučelju je korisnik ujedno i kreator mjesta te samim time mu je onemogućeno slanje poruka samom sebi, a na desnom sučelju se radi o korisniku koji nije kreator mjesta. Provjera se radi na način da baza objekata i manifestacija u sebi sadrži atribut ili polje koje se ispunjava identifikacijskim brojem kreatora. Samim time prilikom pristupa korisnika na fragment detalja automatski se radi provjera.

6. Zaključak

Unatoč tome što postoji mnoštvo članaka, artikala i aplikacija koje preporučuju mjesta za posjetiti, ne postoji niti jedan članak ili aplikacija bez vanjskog utjecaja na sadržaj. Odnosno, niti jedna aplikacija ili članak ne omogućava korisnicima da pišu svoja iskustva, ostavljaju komentare i sami dodaju objekte ili manifestacije. Samim time, u većim gradovima dolazi do problema gdje je mlada populacija prisiljena slijepo tražiti mjesta za druženje. Trenutačno se ovaj problem rješava putem verbalne komunikacije ili pomoću društvenih mreža između prijatelja. Trenutnim načinom rješavanja ovog problema osobe često znaju uzalud izgubiti vrijeme te dovesti se u neugodne situacije. Također, pojavom interneta i pametnih uređaja osobe se privikavaju na način života gdje imaju sve potrebne informacije na dlanu ruke te samim time ispitivanje stranaca o putu do lokacije ili o samoj lokaciji postaje sve rjeđe i rjeđe.

Ovim završnim radom kreiran je sustav sličan društvenim mrežama. S udaljenom bazom podataka i mogućnošću korištenja istodobno više uređaja mobilna aplikacija omogućava pregled i komunikaciju korisnika, objekata i lokacija u stvarnom vremenu.

Mobilna aplikacija omogućava registraciju i prijavu korisnika u sustav. Te dodavanje i novih objekata i manifestacija. Prikaz istih ovisi o udaljenosti koju korisnik postavi u postavkama aplikacije. Također, korisniku je dopuštena manipulacija s obavijestima, odnosno osim obavijesti za razgovore može odabrati za koji tip objekta ili manifestacije želi obavijest primiti. Aplikacija pruža i promjenu već postavljenog imena. Osim što korisnik može dodavati objekte i manifestacije, on također može i recenzirati iste putem komentara te samim time pružiti povratnu informaciju ostalim korisnicima. Svaki objekt i manifestacija uključuje osim slike i opis koji korisnik prilikom kreiranja može dodati. Kao dodatne funkcije dodano je pregled lokacije na mapi koristeći *Google Maps API* i kreiranje razgovora s kreatorom. Razgovor pruža direktnu interakciju između dva korisnika gdje se korisniku omogućava postavljanje upita te razjašnjavanje potencijalnih nejasnoća koje nisu navedene u opisu samog mjesta ili manifestacije. Slanjem poruke primatelj poruke je istovremeno obaviješten o primljenoj poruci te je ta mogućnost implementirana pomoću *Firebase Cloud Messaginga*. Korisnik je također obaviješten i ako uđe u radijus koji malog promjera što označava da je korisnik u blizini mjesta ili manifestacije te se na taj način promovira svaka dodana manifestacija ili objekt. Ta mogućnost je implementirana pomoću *Geofencinga*.

LITERATURA

[1]Elprocus, Android and Android versions, 29.5.2022., dostupno na: https://www.elprocus.com/what-is-android-introduction-features-applications/

[2]Stat counter, Android version market share, 29.5.2022., dostupno na: https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide

[3]Elinux, Android arhitecture, 29.5.2022., dostupno na: https://elinux.org/Android_Architecture

[4]Britannica, Android, 29.5.2022., dostupno na: https://www.britannica.com/technology/Android-operating-system

[5]Robert Love, Linux kernel development, third edition, 2010., Developers Library

[6]Geeks for Geeks, Android SDK, 29.5.2022., dostupno na: https://www.geeksforgeeks.org/android-sdk-and-its-components/

[7]Jerome (J.F) DiMarzio, Android A Programmer's guide, 2008., McGraw-Hill Companies

[8]Android Developers, Build, 30.5.2022., dostupno na: https://developer.android.com/studio/build

[9]TutorialsPoint, Gradle, 2020.

[10]Android developers, Components, 31.5.2022., dostupno na: https://developer.android.com/guide/components/fundamentals

[11]The Open University of Sri Lanka, Department of Electrical and Computer Engineering, 2017., Commonwealth of Learning

[12]Dave MacLean, Satya Komatineni, Android Fragments, 2014., Apress

[13]Android developers, Activity, 25.6.2022., dostupno na: https://developer.android.com/reference/android/app/Activity

[14]Android developers, Activity Lifecycle, 31.5.2022., dostupno na: https://developer.android.com/guide/components/images/activity_lifecycle.png

[15]Android developers, Services, 25.6.2022., dostupno na: https://developer.android.com/guide/components/services

[16]Android developers, Broadcast Receivers, 25.6.2022., dostupno na: https://developer.android.com/guide/components/broadcasts

[17]Android developers, Content Providers, 25.6.2022., dostupno na: https://developer.android.com/guide/topics/providers/content-provider-basics

[18]Android developers, Intent, 25.6.2022. dostupno na: https://developer.android.com/reference/android/content/Intent

[19]Medium, Fragments, 31.5.2022., dostupno na: https://medium.com/@mohdfarhanakram/android-fragment-introduction-9ecf132501da
[20]Tutorials point, MVVM 2018.
[21]Android developers, MaterialCardView, 25.6.2022., dostupno na: https://developer.android.com/reference/com/google/android/material/card/MaterialCardView
[22]Android developers, Dialogs, 25.6.2022., dostupno na: https://developer.android.com/guide/topics/ui/dialogs
[23]Firebase documentation, Firebase Cloud Messaging, 26.6.2022., dostupno na: https://firebase.google.com/docs/cloud-messaging
[24]Android developers, ViewPager, 26.6.2022., dostupno na: https://developer.android.com/reference/kotlin/androidx/viewpager/widget/ViewPager

SAŽETAK

Razvojem mobilnih uređaja i njihovih mogućnosti, te društvenih mreža ljudi se sve više privikavaju imati sve potrebne informacije na dlanu ruke. Primjeri ispitivanja stranaca o lokaciji ili detaljima lokacije su sve rjeđe i rjeđe te sve više postaje navika tražiti povratnu informaciju o mjestu na društvenim mrežama ili novinskim člancima. Upravo ovaj problem rješava ova mobilna aplikacija gdje zapravo korisnici imaju mogućnost listanja svih obližnjih objekata i manifestacija te pritom im je omogućeno gledanje i objavljivanje subjektivnih komentara. Također, u slučaju nejasnoća uvijek mogu postaviti pitanje kreatoru objekta ili manifestacije u dodatku razgovora koji omogućava komunikaciju dvaju korisnika u stvarnom vremenu. Za kreiranje aplikacije bilo je potrebno kreiranje čitavog sustava koji zadovoljava zahtjeve te se u konačnici odlučilo za tehnologiju Firebase koja rješava problem udaljene baze podataka i prijave i registracije korisnika. Također, zaslužan je i za obavještavanje korisnika o novo dodanim mjestima i dolazećim porukama. Kako se radi o aplikaciji za Android operacijski sustav programski jezik Kotlin je bio logično rješenje jer upravo on je primarni jezik za kreiranje aplikacija za Android. Kako bi se korisnicima mogla pružiti podrška s točnom lokacijom objekta ili manifestacije te pregled iste, za rješenje te inačice se koristi Google Maps API koji prikazuje kartu na koju su korisnici navikli.

Ključne riječi: Android, Android Arhitektura, Android Komponente, Geofencing, Gradle, Kotlin

ABSTRACT

Development of smartphones, their possibilities, and social network users are getting used to having all the essential information on the tip of their hands. Examples of strangers on the street asking about the location or details of a certain location are rear nowadays. Nowadays practice is reading about the location on social networks or in paper articles. Exactly this problem is what the mobile application is trying to solve. In the application, users can scroll through all nearby places. Also, they can read and post subjective comments. For any misunderstanding, they can always ask the creator questions in real-time chats. For creating the application it was required to firstly create a system that solves all requests that this problem has. In the end, the chosen technology was Firebase for remote database and user authentication. Also, Firebase was used to notify users about newly added places and receive messages. Because it is the application for the Android operating system, Kotlin programming language was the logic solution because it is the primary language for creating applications for Android. Providing users with exact locations and a preview, the Google Maps API was a solution because of his wide user and users habit.

Keywords: Android, Android Arhitecture, Android Components, Geofencing, Gradle, Kotlin

ŽIVOTOPIS

Ante Lukić rođen je 10. studenog 2000. godine u Osijeku. 2015. godine završava Osnovnu školu Vladimira Nazora u Đakovu, a srednju strukovnu školu u Đakovu, smjer Tehničar za mehatroniku završava 2019. te iste godine upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, smjer Računarstvo na preddiplomskom sveučilišnom studiju.

Ante Lukić

_____