Mobilna aplikacija za pružanje pomoći prilikom ekoloških katastrofa

Sokol, Matija

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I

INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

MOBILNA APLIKACIJA ZA PRUŽANJE POMOĆI PRILIKOM EKOLOŠKIH KATASTROFA

Diplomski rad

Matija Sokol

Osijek, 2021.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 25.08.2021.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime studenta:	Matija Sokol
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1088R, 06.10.2019.
OIB studenta:	20324109596
Mentor:	Izv. prof. dr. sc. Ivica Lukić
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Mirko Köhler
Član Povjerenstva 1:	Izv. prof. dr. sc. Ivica Lukić
Član Povjerenstva 2:	Doc.dr.sc. Zdravko Krpić
Naslov diplomskog rada:	Mobilna aplikacija za pružanje pomoći prilikom ekoloških katastrofa
Znanstvena grana rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak diplomskog rada:	U okviru ovog diplomskog rada potrebno je izraditi mobilnu aplikaciju za Android operacijski sustav koristeći MVVM arhitekturalni obrazac koja će služiti za povezivanje unesrećenih osoba i ljudi koji nude pomoć. Aplikacija bi, koristeći lokacijske usluge, trebala preporučivati korisnicima pomoć prema trenutnoj lokaciji. Kroz grafičko sučelje potrebno je omogućiti prikaz karte s lokacijama od značaja te interakciju trenutno prijavljenog korisnika s oglasima na način da se korisnik može prijaviti za određeni posao.
Prijedlog ocjene pismenog dijela ispita (diplomskog 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: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	25.08.2021.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 11.09.2021.

Ime i prezime studenta:	Matija Sokol
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1088R, 06.10.2019.
Turnitin podudaranje [%]:	11%

Ovom izjavom izjavljujem da je rad pod nazivom**: Mobilna aplikacija za pružanje pomoći prilikom ekoloških katastrofa**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Ivica Lukić

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:

SADRŽAJ

1. UVOD

U današnje vrijeme svjedoci smo sve više i više prirodnih katastrofa. Kako tehnologija napreduje, tako se na tržištu pojavljuju nova i poprilično uspješna programska rješenja koja pomažu u raznim nevoljama koje mogu utjecati na svakodnevni život pojedinaca. Jedno od takvih rješenja je i aplikacija koje je izrađena kao tema ovog diplomskog rada.

Cilj ovoga rada bio je izraditi mobilnu aplikaciju koja će omogućiti jednostavniji način komunikacije između unesrećenih osoba i pružatelja pomoći. Pod tim se podrazumijeva da se pružatelji usluga mogu javiti na oglase unesrećenih osoba te se prijaviti za određeni posao. Također, korisnici mogu samoinicijativno objaviti oglase za pružanje pomoći.

U izradi aplikacije korišten je programski jezik Kotlin te razvojno okruženje (engl. *development environment*) Android Studio koje je izgrađeno na JetBrains-ovom IntelliJ IDEA programskom alatu.

U drugom poglavlju će se obraditi različita trenutna rješenja sa sličnom tematikom, dok će u trećem poglavlju biti opisane korištene tehnologije prilikom izrade programskog rješenja. Četvrto poglavlje sadrži opis implementiranog programskog rješenja. Također, dan je uvid u samu strukturu projekta. U zadnjem poglavlju dat će se kratak zaključak i osvrt na sam rad te koja je bila namjera prilikom izrade aplikacije.

1.1. Zadatak diplomskog rada

U okviru ovog diplomskog rada potrebno je izraditi mobilnu aplikaciju za Android operacijski sustav koristeći MVVM arhitekturalni obrazac koja će služiti za povezivanje unesrećenih osoba i ljudi koji nude pomoć. Aplikacija bi, koristeći lokacijske usluge, trebala preporučivati korisnicima pomoć prema trenutnoj lokaciji. Kroz grafičko sučelje potrebno je omogućiti prikaz karte s lokacijama od značaja te interakciju trenutno prijavljenog korisnika s oglasima na način da se korisnik može prijaviti za određeni posao.

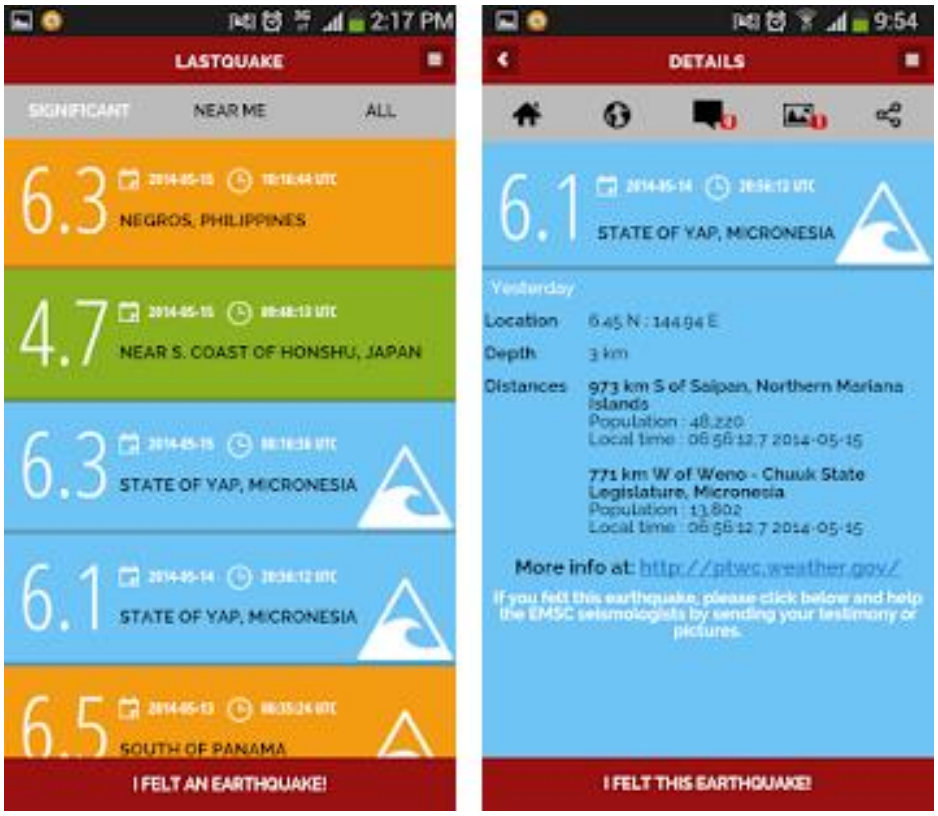
2. TRENUTNA RJEŠENJA

Pretraživajući internet, pronađeno je puno aplikacija slične tematike kao i aplikacija napravljena u svrhu izrade diplomskog rada. Neke od njih bit će opisane u sljedećim potpoglavljima.

2.1. Last Quake

Last Quake [6] službena je mobilna aplikacija Euromediteranskog seizmološkog centra (engl. *European-Mediterranean Seismological Centre*). Besplatna je i služi za obavješćivanje korisnika tijekom potresa. Ima više od milijun korisnika te je izuzetno popularna u Hrvatskoj i okolici, osobito nakon prethodnih nepogoda.

Aplikacija radi u realnom vremenu te je izvrsna za prikupljanje i razmjenu iskustava s ostalim korisnicima. Također, pruža korisne savjete kako bi ublažila posljedice potresa. Korisnicima je omogućeno dodavanje fotografija i videozapisa kako bi podijelili svoje informacije s ostalim korisnicima. Izgled aplikacije prikazan je na slici 2.1.



Slika 2.1. Izgled aplikacije Last Quake [7]

2.2. FEMA

FEMA (engl. *Federal Emergency Management Agency*) [8] je mobilna aplikacija namijenjena korisnicima unutar Sjedinjenih Američkih Država. Potpuno je besplatna te omogućuje primanje obavijesti za maksimalno pet prethodno odabranih lokacija. Za razliku od prethodne, ova aplikacija nudi obavijesti ne samo za potrese, već i za požare, snježne oluje, tornada, vulkanske erupcije i slično.

Koristeći aplikaciju, korisnici mogu kontaktirati ostale korisnike u slučaju potrebe za istim. Aplikacija predlaže uputstva za ponašanje prije, tijekom i nakon ekoloških nepogoda. Također, ista nudi lociranje skloništa za hitne slučajeve u blizini. Na slici 2.2. vidljivi su zasloni FEMA aplikacije.

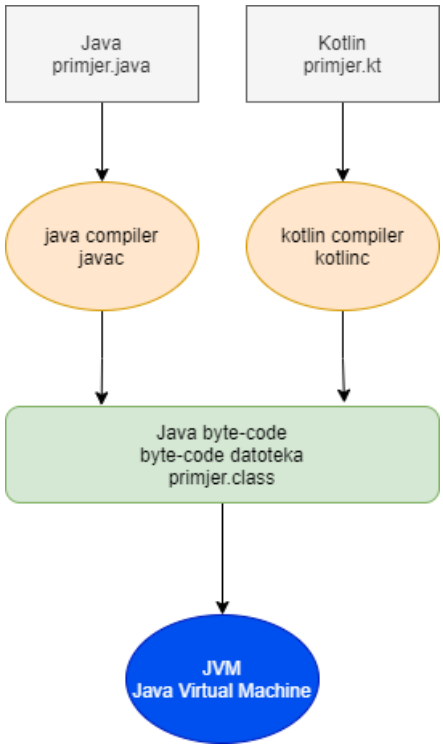
Slika 2.2. Izgled aplikacije FEMA [9]

3. KORIŠTENE TEHNOLOGIJE

Kao što je napisano u uvodu, prilikom izrade diplomskog rada korišten je Kotlin programski jezik te Android Studio razvojno okruženje. Navedeno razvojno okruženje je potpuno besplatno za korištenje. Kotlin je poprilično nov i moderan jezik s kojim se studenti mogu susresti na kolegiju Razvoj mobilnih aplikacija. Naravno, za kompleksnija i kvalitetnija rješenja potrebno je samostalno proučiti dodatnu literaturu i upoznati se s naprednim funkcionalnostima koje taj jezik pruža.

3.1. Programski jezik Kotlin

Kotlin [1] je višeplatformski, statički tipiziran, objektno orijentiran programski jezik razvijen od tvrtke JetBrains. Budući da koristi JVM (engl. *Java Virtual Machine*) u pozadini, Kotlin je dizajniran tako da može potpuno surađivati s programskim jezikom Javom. To znači da u istom projektu možemo imati datoteke koje sadrže Java (.java ekstenzija) i Kotlin kod (.kt ekstenzija). Na slici 3.1. prikazan je način kompilacije (engl. *compilation*) Java i Kotlin koda.



Slika 3.1. Proces kompilacije Java i Kotlin koda

Jedna od posebnosti Kotlina su ekstenzijske (engl. *extension*) funkcije. One nam omogućuju dodavanje novih funkcija na već postojeće klase koje inače ne bi mogli mijenjati. Izlistanje koda 3.1. prikazuje primjer i korištenje jedne takve funkcije.

```
fun Int.isEven(): Boolean {
    return this % 2 == 0
}

fun main() {
    val myNumber = 2
    println(myNumber.isEven()) // prints true
}
```

Izlistanje koda 3.1. Primjer ekstenzijske funkcije

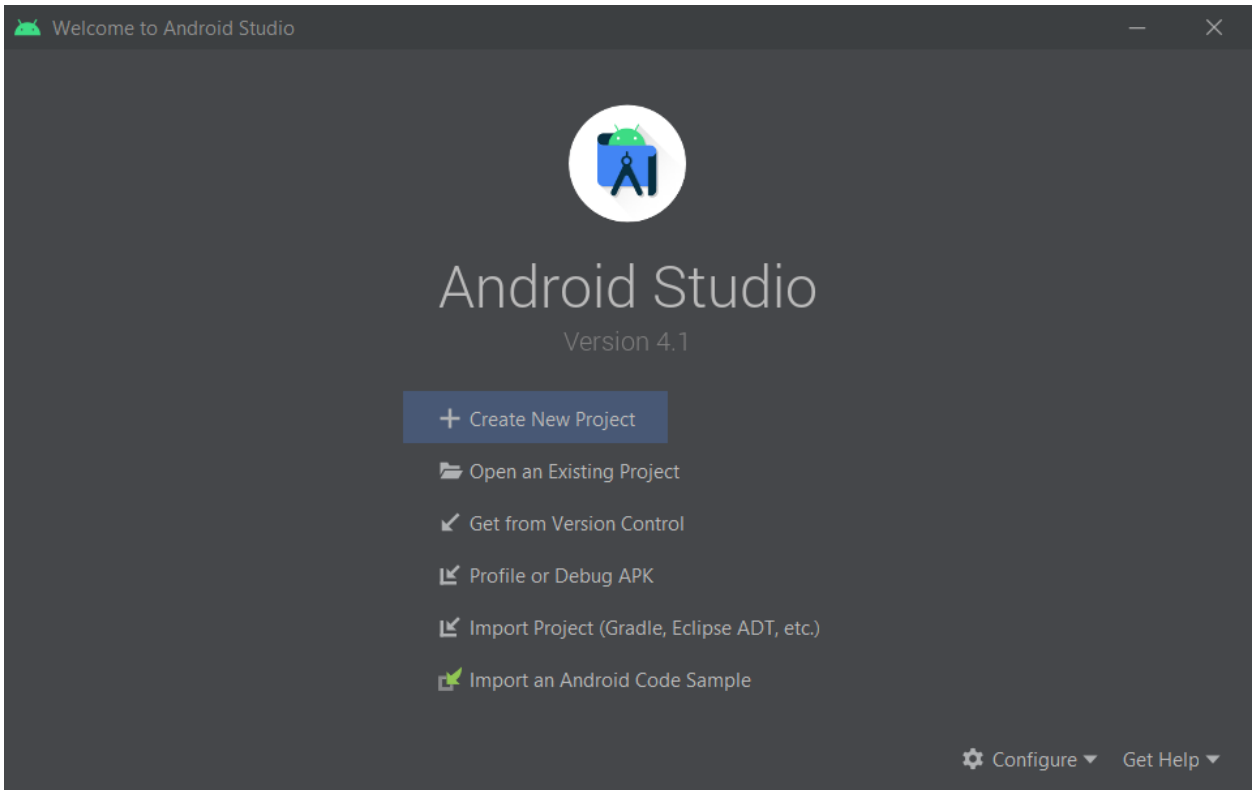
Naglo povećanje korištenja Kotlina kreće 2019. godine kada je Google objavio da je isti prvi izbor za izradu nativnih mobilnih aplikacija za Android operacijski sustav. Do tada je Java bila preferirani jezik za ovu vrstu razvoja aplikacija. Iako je Kotlin broj jedan, još uvijek postoje aplikacije pisane u Java programskoj jeziku, ali u sve manjem broju. Tomu pridonosi i sam Google koji sve svoje nove biblioteke (engl. *libraries*) izrađuje u Kotlinu.

3.2. Android Studio

Razvojno okruženje koje se koristi za razvoj Android aplikacija je Android Studio [2]. Iako nije jedini izbor, zasigurno je najpopularniji izbor za navedenu platformu. Riječ je o integriranom razvojnom okruženju (engl. *integrated development environment - IDE*) zasnovanom na IntelliJ IDEA platformi koje nudi niz razvojnih alata. Neki od njih su:

- uređivač teksta
- sustav za prevođenje (engl. *build*)
- emulator
- integracija s alatima za verzioniranje koda
- alati za statičku analizu koda i dr.

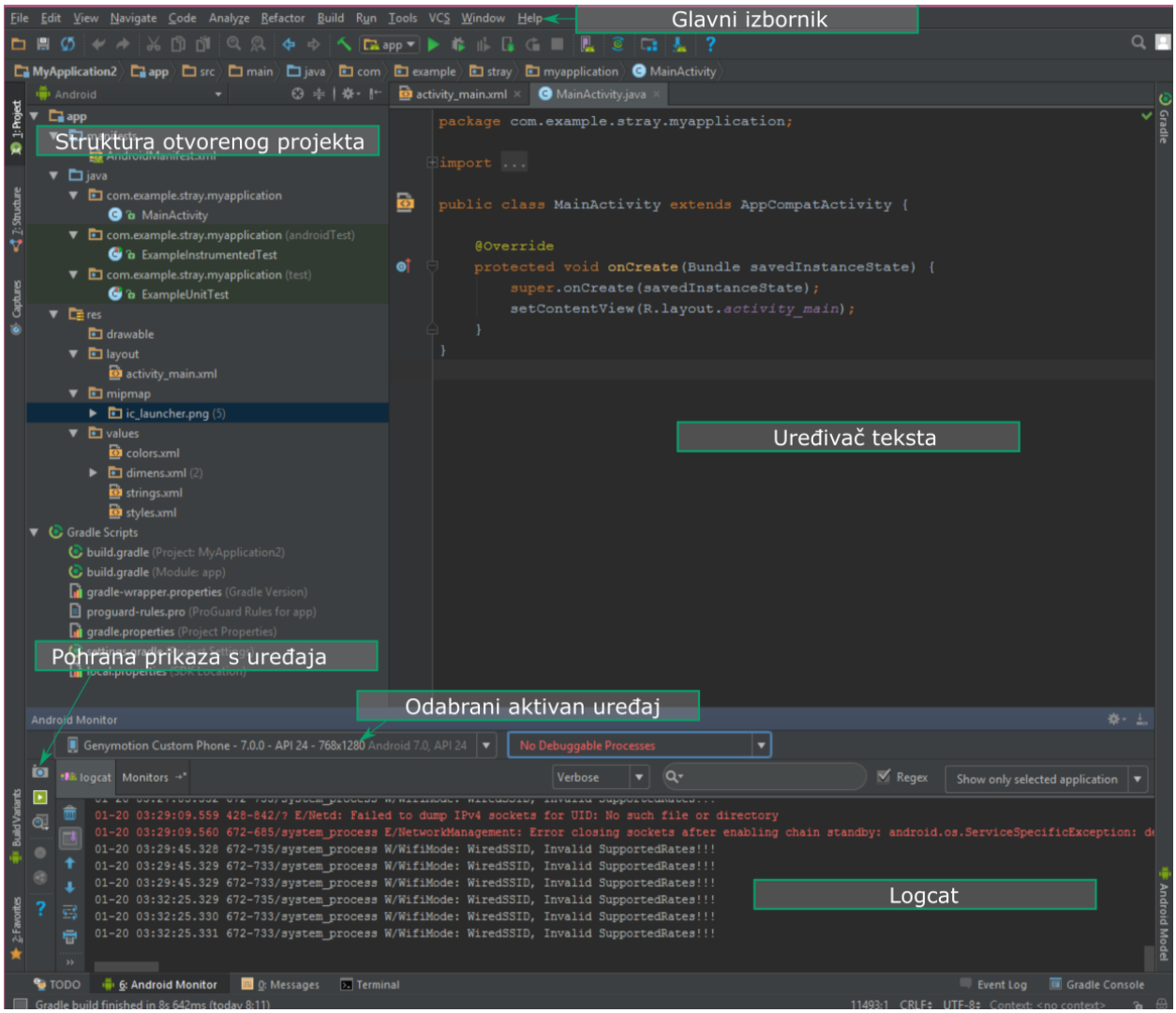
Prilikom razvijanja aplikacija za Android platformu potrebna su dva razvojna paketa (engl. *development kit*), a to su: JDK (engl. *Java Development Kit*) i Android SDK (engl. *Software Development Kit*). Android SDK se automatski može preuzeti prilikom instalacije Android Studia dok se JDK preuzima zasebno. Slika 3.2. prikazuje početni zaslon navedenog razvojnog okruženja.



Slika 3.2. Početni zaslon

Za pokretanje Android aplikacija potrebno je koristiti Android uređaj ili kreirati i pokrenuti virtualni Android uređaj (engl. *Android virtual device - AVD*). Za kreiranje virtualnog uređaja koristi se upravitelj virtualnim uređajima (engl. *Virtual device manager*). Kroz navedeni upravitelj dostupno je na desetke različitih uređaja koji se mogu koristiti. Također, korisnik može instalirati proizvoljnu verziju Android OS-a na isti. Moguće je koristiti i drugačiji emulator koji nije vezan za Android Studio. Jedan od njih je Genymotion koji je besplatan za osobnu uporabu, a oslanja se na VirtualBox alat za virtualizaciju računalnih resursa.

Još jedna od zanimljivih mogućnosti koje nudi Android Studio je olakšan pristup korištenja GIT-a. Kroz razvojno okruženje omogućeno je korištenje GIT naredbi bez pisanja istih unutar terminala. Ovo uvelike pomaže na većim projektima gdje postoji puno grana (engl. *branches*) te je često potrebno izmjenjivati više njih unutar kratkog perioda. Uz to, Android Studio nam nudi lako praćenje novih promjena u programskom kodu nakon izvršenja izmjena (engl. *commit*). Na slici 3.3. nalazi se prikaz otvorenog projekta unutar razvojnog okruženja.



Slika 3.3. Izgled projekta [2]

3.3. Firebase

Firebase [3] je mobilna i web platforma koja pruža programerima velik broj alata i servisa kako bi im pomogla razviti što kvalitetnije i robusnije aplikacije. Osnovana je 2011. godine kao startup nazvan Envolve. Zamisao je bila omogućiti integraciju usluge *online* razmjene poruka u stvarnom vremenu. 2012. nastaje Firebase kao zasebna tvrtka koja je pružala *Backend-as-a-Service* funkcionalnosti. Konačno, 2014. Google kupuje Firebase te je od tada postao neizostavan alat pri razvijanju aplikacija.

Korištenje Firebasea može se podijeliti u tri kategorije [4][5]:

- Izrada aplikacija
- Poboljšanje kvalitete aplikacije
- Rast poduzeća

U tablici 3.1. objašnjene su navedene kategorije.

Tablica 3.1. Firebase mogućnosti i kategorije

IZRADA APLIKACIJA	Cloud Firestore	Fleksibilna, skalabilna baza podataka za razvijane mobilnih i web aplikacija. NoSQL baza podataka gdje jedan redak tablice predstavlja dokument.
	ML Kit	SDK za mobilne aplikacije koji pruža mogućnost strojnog učenja. Lagan za korištenje početnicima, a pruža napredne funkcionalnosti za iskusnije korisnike.
	Cloud Functions	Omogućuje automatsko pokretanje pozadinskog (engl. *backend*) koda u odgovorima na HTTP zahtjeve.
	Authentication	Pruža biblioteke za autentifikaciju korisnika u aplikaciji. Omogućuje upotrebu raznih načina autentifikacije (lozinka, broj telefona, razne društvene mreže).
	Hosting	Alat za postavljanje web aplikacija na server.
	Cloud Storage	Omogućuje jednostavan rad s datotekama (postavljanje i preuzimanje). Podržava velik broj tipova datoteka: slike, audiozapisi, videozapisi itd.
	Real-time Database	NoSQL baza podataka za spremanje i sinkronizaciju podataka između korisnika u stvarnom vremenu.
POBOLJŠANJE KVALITETE APLIKACIJE	Crashlytics	Jednostavan alat za prijavljivanje grešaka u aplikacijama koji radi u stvarnom vremenu. Pomaže pri praćenju, kategoriziranju te popravljanju problema koji utječu na aplikaciju.
	Performance Monitoring	Servis koji pomaže pri određivanju karakteristika performansi. Omogućuje prikupljanje podataka performansi te pregled i analizu istih.
	Test Labs	Alat koji pruža velik broj mobilnih uređaja koji pomažu pri testiranju aplikacije. Omogućen je za Android i iOS operacijski sustav.
RAST PODUZEĆ	In-App Messaging	Omogućuje slanje poruka korisnicima aplikacije.
	Google Analytics	Pomaže pri razumijevanju kako krajnji korisnik upravlja aplikacijom iz čega se mogu donijeti važne poslovne odluke.

	Predictions	Koristi strojno učenje nad analitičkim podacima kako bi predvidio ponašanje korisnika.
	A/B Testing	Pomaže pri optimiziranju aplikacije kako bi ista bila jednostavna za pokretanje, analizu itd.
	Cloud Messaging (FCM)	Pruža pouzdanu i baterijski učinkovitu vezu između servera i uređaja koja omogućuje slanje i primanje obavijesti.
	Remote Config	Omogućuje izdavanje ažuriranja krajnjim korisnicima. Na primjer, koristeći ovaj alat, moguće je promijeniti boju zaslona aplikacije.
	Dynamic Links	Korisniku je klikom na poveznicu omogućeno direktno pokretanje aplikacije (ukoliko ona postoji na uređaju).
	App Indexing	Omogućuje pokretanje aplikacije preko Google Searcha. Ako korisnik ima instaliranu aplikaciju, mogu pokrenuti istu i doći izravno do sadržaja koji su tražili.

Prilikom izrade aplikacije korištene su sljedeće Firebase funkcionalnosti:

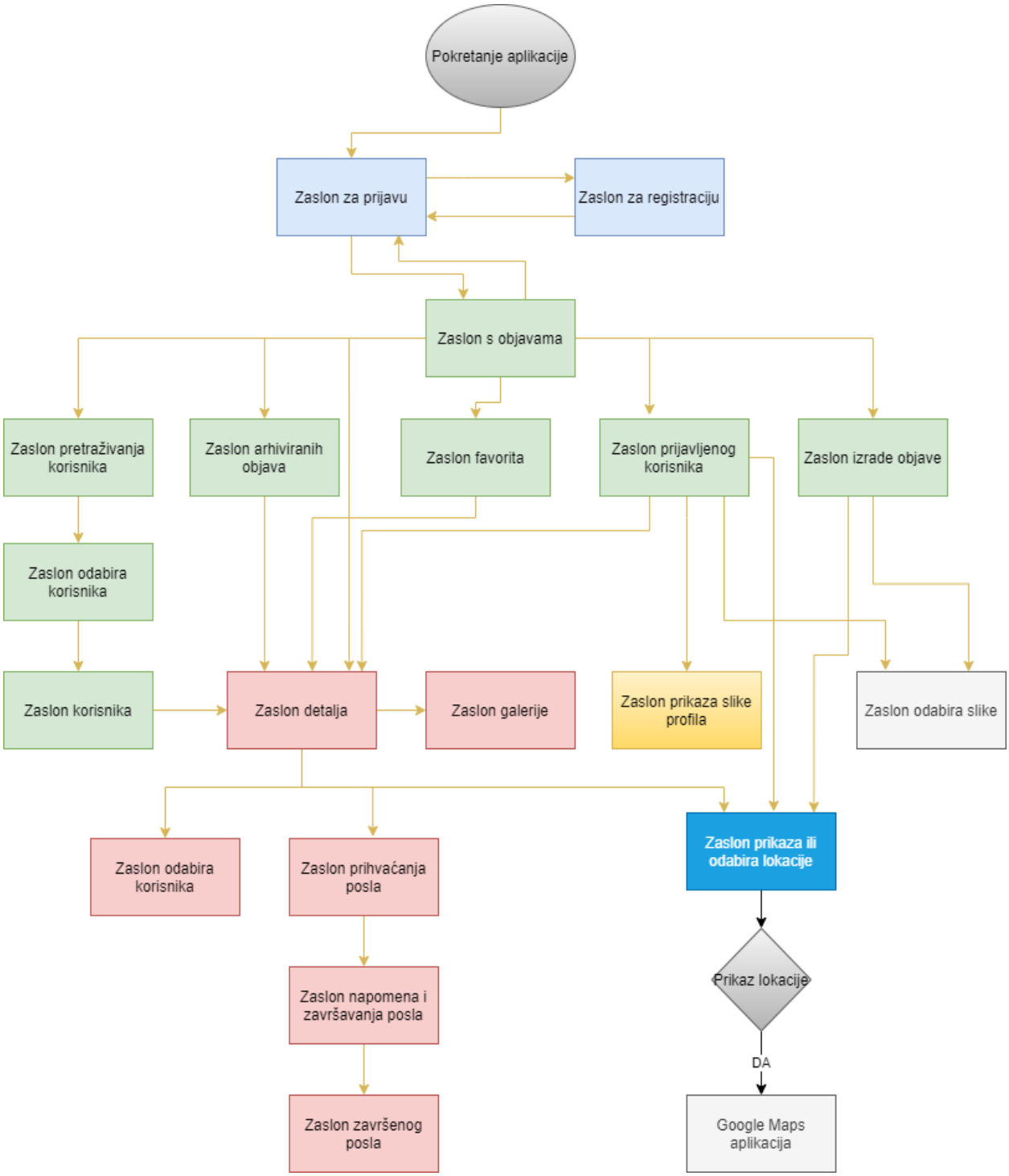
- Authentication – mogućnost prijave i registracije u sustav
- Cloud Firestore – korištenje baze podataka u stvarnom vremenu
- Cloud Storage – spremanje slika
- Cloud Messaging – slanje i primanje obavijesti

4. PROGRAMSKO RJEŠENJE

U daljnjem tekstu bit će opisano što se i kako odvija u implementiranom programskom rješenju. Cijela aplikacija se sastoji od pet aktivnosti (engl. *activity)*, dok se unutar većine aktivnosti nalazi nekoliko *fragmenata*, od kojih je svaki zadužen za određenu funkcionalnost. Budući da je aplikacija izrađena koristeći MVVM arhitekturalni obrazac (engl. *architectural pattern*), *fragmenti* su zaduženi samo za prikaz podataka, dok se poslovna logika (engl. *business logic*) odvija u *ViewModelima* i repozitorijima (engl. *repositories*). Više o tome će biti riječ u sljedećim potpoglavljima.

4.1. Dijagram tijeka

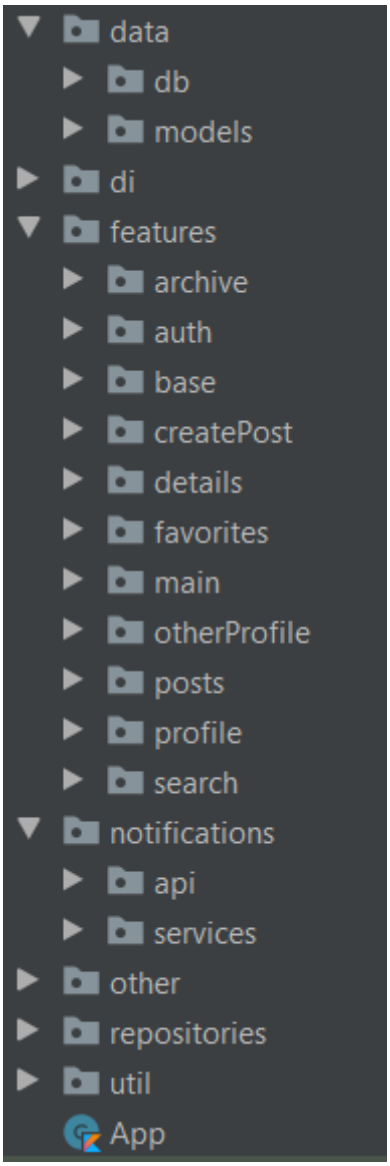
Dijagram tijeka (engl. *flow chart*) je grafički prikaz algoritma [10]. Pomoćno je sredstvo koje je neovisno o programskom jeziku i računalu te vizualizira zadatak ili problem te on postaje pregledniji i razumljiviji. Sastoji se od niza geometrijskih likova, od kojih svaki ima svoje značenje. Na primjer, ovalni lik predstavlja početak, kraj ili prekid programa, dok romb predstavlja uvjetno grananje. Oni su spojeni usmjerenim crtama te tako povezani čine cjelinu. Na slici 4.1. prikazan je dijagram tijeka izrađene aplikacije. Zasloni istih aktivnosti su, radi preglednosti, označeni istim bojama.



Slika 4.1. Dijagram tijeka aplikacije

4.2. Struktura projekta

Svaki veći i ozbiljniji projekt se nakon nekog vremena poprilično „zatrpa“ s datotekama klasa i ostalim dijelovima. Za bolju organizaciju strukture projekta, Android Studio omogućuje grupiranje srodnih datoteka unutar paketa (engl. *packages*). To su zapravo direktoriji što možemo i vidjeti ako otvorimo odgovarajući direktorij projekta na disku.



Slika 4.2. Struktura projekta

Na slici 4.2. vidljiva je struktura projekta izrađene aplikacije. Aplikacija je podijeljena u nekoliko paketa dok neki od njih sadrže i podpakete. Glavni paketi su:

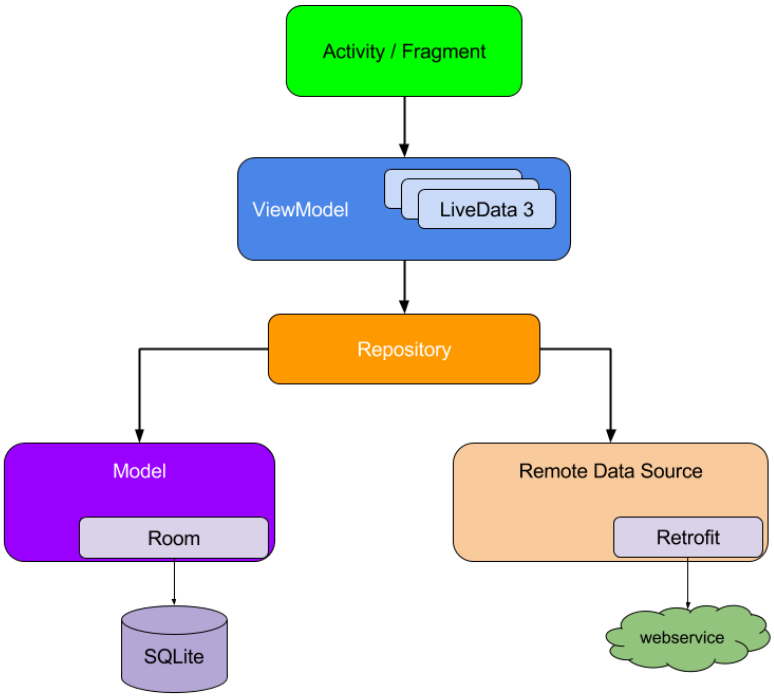
- *data* – sadrži *POJO* (engl. *Plain old Java object*) klase te datoteke za rad s lokalnom bazom podataka

- *di* – sadrži module za ubrizgavanje ovisnosti (engl. *dependency injection*) koristeći Dagger Hilt
- *features* – sadrži zaslone aplikacije, pripadajuće *ViewModele*, *Adaptere* i slično
- *notifications* – sadrži datoteke za rad s obavijestima (engl. *push notifications*)
- *other* – sadrži datoteku s konstantama
- *repositories* – sadrži repozitorije
- *util* – sadrži datoteke s ekstenzijskim i običnim funkcijama te ostale pomoćne klase koje služe za obradu podataka

4.3. Korištene tehnologije

4.3.1. MVVM

Kao što je već ranije napisano, aplikacija je izrađena koristeći MVVM (engl. *Model-View-ViewModel*) arhitekturalni obrazac. U današnje vrijeme većina novih aplikacija izrađuje se koristeći navedeni obrazac. Tome je pridonio i sam Google koji je u zadnjih par godina izdao niz biblioteka koje olakšavaju implementaciju MVVM obrasca. Na slici 4.3. prikazana je shema istog.



Slika 4.3. MVVM shema [11]

Na shemi *activity/fragment* predstavlja pogled (engl. *view*), *ViewModel* je samoobjašnjavajuć dok repozitorij, koji u ovome slučaju koristi lokalne i podatke s vanjskog izvora, predstavlja model jer je zadužen za obradu podataka. Na shemi se jasno vidi na koji su način komponente povezane. Prilikom interakcije korisnika s aplikacijom, metode koje se pozivaju nalaze se unutar komponente pogleda. Ona poziva odgovarajuće metode u *ViewModelu* dok on prosljeđuje poziv repozitoriju. Podaci koje repozitorij dobije vraćaju se obrnutim redoslijedom do pogleda te se isti prikazuju na zaslonu.

4.3.2. Kotlin korutine

Prilikom izrade mobilnih aplikacija važno je pružiti što bolje korisničko iskustvo (engl. *user experience - UX*). Jedna od najčešćih grešaka prilikom implementacije programskog rješenja je obavljanje velikih i teških poslova na glavnoj niti (engl. *main thread*), kao što su *API* pozivi na udaljeni server, rad s datotekama, bazom podataka i slično. Glavna nit je zadužena za prikaz podataka na zaslonu i nebi trebala obavljati teške poslove jer se na taj način zaslon ne može osvježavati što uzrokuje zastajkivanje aplikacije i loše korisničko iskustvo. Svaki takav događaj može potencijalno utjecati na korisnika da prestane s uporabom aplikacije.

Da bi se ovaj problem riješio, potrebno je prebaciti posao na pozadinsku nit. Android sustav omogućuje uporabu *Thread* klase za ovakve slučajeve, međutim u velikim projektima ona i nije baš preporučljiva. Na primjer, ako želimo dohvatiti podatke s udaljenog servera, proslijediti ih drugoj funkciji i odraditi dodatan posao, potrebno je koristiti *callback* pristup. On se temelji na tome da se određena funkcija pozove tek kada se posao završi. Time smo postigli asinkron rad aplikacije jer se ostali dio programa ne blokira dok se čeka rezultat *callbacka*. Ukoliko unutar *callbacka* imamo još funkcija koje se oslanjaju na *callback* pristup kod postane poprilično nerazumljiv. Ovaj slučaj kada imamo ugnježđene *callbackove* unutar drugih *callbackova* zove se *callback* pakao (engl. *callback hell*).

Kako bi doskočili ovome problemu, programeri se oslanjaju na vanjske biblioteke poput RxJave i Kotlin korutina (engl. *Kotlin coroutines*). RxJava [12][13] nam na jednostavan način omogućuje prebacivanje između niti što nam uvelike olakšava obradu podataka i prikazivanje istih na zaslonu. Kao što je vidljivo na izlistanju koda 4.1., RxJava se oslanja na veliki niz operatora zaduženih za transformaciju podataka poput: *map, flatmap, filter, doOnSuccess, doOnError, doFinally* i još puno toga. Iako puno pomaže u radu s velikom količinom podataka, važno je za napomenuti da RxJava zahtjeva duboko poznavanje iste te treba biti oprezan pri radu s njom.

```
private fun getList(): Disposable {
    return taskRepository.getTaskList()
            .firstOrError()
            .toObservable()
            .flatMapIterable { it }
            .map { it.id }
            .observeOn(Schedulers.io())
            .subscribeOn(AndroidSchedulers.mainThread())
            .subscribe({
               // consume data
            }, {
               // print error message
            })
}
```

Izlistanje koda 4.1. Primjer korištenja *RxJave*

Za razliku od RxJave, Kotlin korutine [14][15], koje su korištene u ovome projektu, ne oslanjaju se na pristup rada s operatorima već omogućuju pisanje asinkronog koda na sinkroni način. Korutine se mogu usporediti s nitima, ali troše puno manje memorije i resursa. Da bi se neka funkcija pozvala unutar korutine mora sadržavati *suspend* ključnu riječ. Ona nam omogućuje da se ta funkcija može pauzirati sve dok ne obavi posao za koji je zadužena. Iako funkcija pauzira s izvođenjem, ona ne blokira nit na kojoj je pozvana te tako ne utječe na izvođenje ostatka programa.

```
suspend fun getAllPosts() = withContext(Dispatchers.IO) {
    val postsDto = postsCollection
        .get()
        .await()
        .toObjects(PostDto::class.java)
    val posts = postMapper.toPosts(postsDto)
    postDao.insert(posts)
}
```

Izlistanje koda 4.2. Primjer korištenja Kotlin korutina

Prema izlistanju koda 4.2. vidljiva je upotreba Kotlin korutina. Potrebno je obratiti pažnju na *withContext* naredbu. Ona je zadužena za prebacivanje posla na odgovarajuću nit na kojoj će korutina biti izvedena. Kako se u navedenom primjeru dohvaćaju podaci s udaljenog servera te se isti spremaju u lokalnu bazu, korišten je *IO dispatcher* koji prebacuje posao na nit rezerviranu za ulazne i izlazne operacije.

Iz navedenog primjera vidljivo je da Kotlin korutine poprilično pojednostavljuju implementaciju složenijih operacija.

4.3.3. Dagger Hilt

Dagger Hilt [16][17] je biblioteka za ubrizgavanje ovisnosti za Android mobilne aplikacije koja smanjuje količinu nepotrebnog (engl. *boilerplate*) koda prilikom ručnog korištenja ubrizgavanja ovisnosti. Izgrađen je na temeljima *Daggera* te tako koristi sve dobre osobine ovog vrlo popularnog alata, kao što su, na primjer, ispravnost prilikom kompajliranja (engl. *compile*) i dobre performanse prilikom izvođenja. Također, Android Studio pruža podršku koju Dagger nudi. Da bi se Hilt uopće mogao koristiti u projektu, potrebno je dodati *@HiltAndroidApp* anotaciju unutar aplikacijske klase. Anotacije nam omogućuju pisanje pojednostavljenog koda te automatski generiraju potreban kod za nas. Hilt omogućuje ubrizgavanje objekata za različite komponente aplikacije. Ukoliko želimo jednu instancu klase unutar cijele aplikacije, potrebno je modul, koji je zadužen za stvaranje objekata, označiti s anotacijom *@InstallIn(SingletonComponent::class)*. Ostale komponente, kao i njihov životni ciklus, prikazane su tablicom 4.1.

Tablica 4.1. Životni vijek Dagger Hilt komponenti

Komponenta	Stvorena	Uništena
SingletonComponent	Application#onCreate()	Application#onDestroy()
ActivityRetainedComponent	Activity#onCreate()	Activity#onDestroy()
ViewModelComponent	ViewModel created	ViewModel destroyed
ActivityComponent	Activity#onCreate()	Activity#onDestroy()
FragmentComponent	Fragment#onAttach()	Fragment#onDestroy()
ViewComponent	View#super()	View destroyed
ViewWithFragmentComponent	View#super()	View destroyed
ServiceComponent	Service#onCreate()	Service#onDestroy()

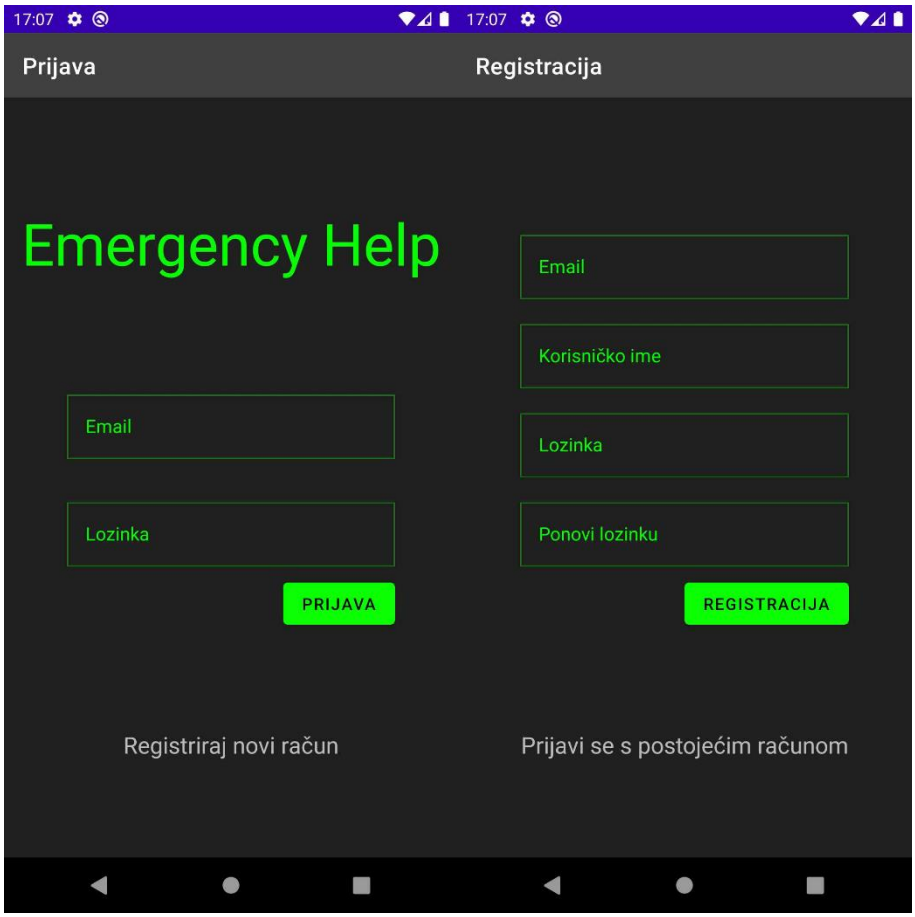
Iz imena komponenti se da iščitati kada se koja treba koristiti. Ukoliko želimo instancu klase koja treba biti „živa“ sve dok je *fragment*, unutar koje je ta instanca aktivna, „živ“, koristit ćemo *FragmentComponent* komponentu.

4.4. Zasloni za prijavu i registraciju

Kako bi korisnici bili u mogućnosti izraditi korisnički račun, potrebno je omogućiti formu za registraciju i prijavu, kao što je prikazano na slici 4.4 i 4.5. Budući da se da potrebe spremanja podataka koristi Firebase, korištena je i Firebaseova autentifikacija. Ona nudi cijeli niz mogućnosti

prijave. Za potrebe ove aplikacije korištena je najjednostavnija prijava, a to je pomoću emaila i lozinke. Neke od ostalih mogućih načina prijave su pomoću:

- Facebooka
- Googlea
- Microsofta
- Githuba
- Twittera itd.



Slika 4.4 i 4.5. Zasloni za prijavu i registraciju

Koristeći Firebase, proces autentifikacije je vrlo jednostavan. Kao što je prikazano na izlistanju koda 4.3., potrebno je pozvati funkciju za registraciju koja prima email i lozinku. Nakon toga se u tablicu s korisnicima dodaje novi entitet s generiranim ID-jem. Tablica korisnici, osim ID-ja i korisničkog imena, sadrži putanju do slike profila te lokaciju korisnika.

```
suspend fun register(
    email: String,
    username: String,
    password: String
) = withContext(Dispatchers.IO) {
```

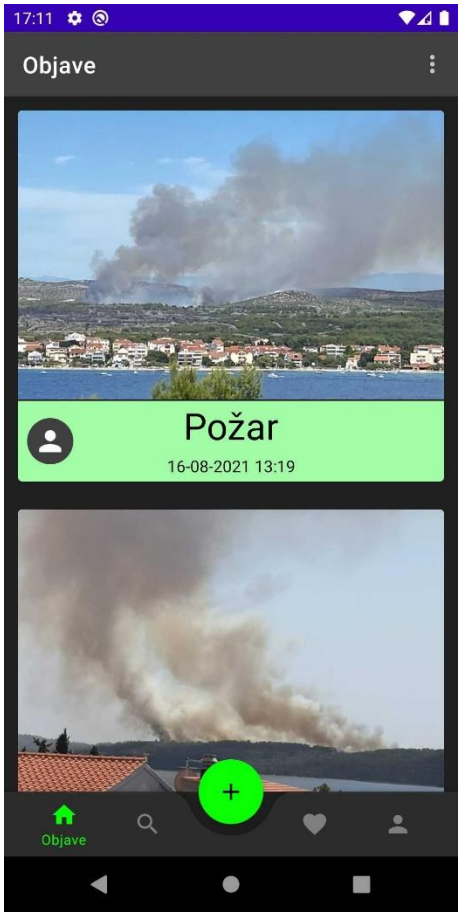
```
    safeCall {
        val result = auth.createUserWithEmailAndPassword(email, password).await()
        val uid = result.user?.uid!!
        val user = User(uid, username)
        users.document(uid).set(user).await()
        Resource.Success(result)
    }
}
```

Izlistanje koda 4.3. Metoda za registraciju

Zaslon za prijavu prati istu logiku kao i zaslon za registraciju. Nakon što korisnik unese podatke, koristeći Firebase autentifikaciju pravi se upit na bazu podataka te, ukoliko korisnik postoji, prelazi se na sljedeći zaslon.

4.5. Zaslon za prikaz objava

Zaslon za prikaz objava je namijenjen kako bi korisnici vidjeli sve aktivne poslove. Kao što je vidljivo na slici 4.6., objava se sastoji od slike te detalja vezanih za tu objavu.



Slika 4.6. Zaslon s objavama

Klikom na ikonu srca objava se sprema među favorite te se ista može vidjeti na posebnom zaslonu gdje su prikazani samo favoriti, dok klik na objavu vodi na zaslon s detaljima objave.

Aplikacija koristi Firebaseovu mogućnost osluškivanja promjena kolekcije (engl. *collection*) objava, to jest tablicu u bazi podataka koja sadrži objave. Kako prikazana lista uzima podatke spremljene u lokalnoj bazi, tako je potrebno na svaku promjenu kolekcije ažurirati tablicu u lokalnoj bazi. Budući da osluškivač, čija je implementacija prikazana izlistanjem koda 4.4., na bilo kakvu promjenu vraća cijelu kolekciju, potrebno je pronaći razliku s lokalnom listom te ažurirati samo one retke koji su se promijenili. Na ovaj način je omogućena automatska promjena liste i detalja objave na bilo kakvu promjenu s bilo kojeg uređaja.

```
fun collectData() = callbackFlow {
    val subscription = posts.addSnapshotListener { value, error ->
        error?.let {
            offer(Resource.Error(it))
            cancel(it.message.toString())
        } ?: {
            value?.let { querySnapshot ->
                offer(Resource.Success(querySnapshot))
            }
        }.invoke()
    }

    awaitClose { subscription.remove() }
}
```

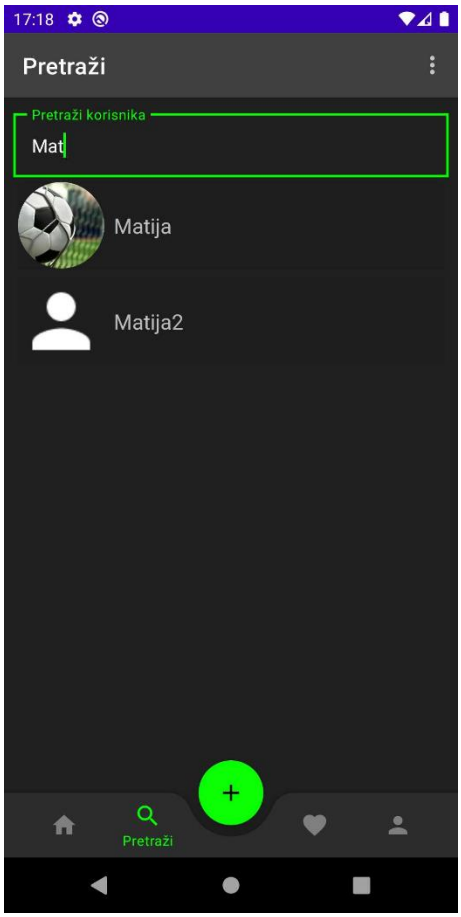
Izlistanje koda 4.4. Implementacija osluškivača kolekcije objava

Lista je implementirana koristeći *RecyclerView* komponentu. Za razliku od *ListViewa* koji je korišten u prošlosti, *RecyclerView* [18] stvara onoliko *view holdera* koliko je potrebno da se popuni zaslon te da se pripremi za pomicanje kroz listu. Nakon što je stvorio onoliko *view holdera* koliko mu treba, za sve iduće, u ovome slučaju objave, koriste se prethodno napravljeni *view holderi* te na njima postavlja potrebne podatke, to jest sliku, naslov i vrijeme. Budući da koristi pristup recikliranja, po tome je i dobio ime.

Zasloni arhive i favorita prate sličan princip kao i ovaj zaslon. Kada se obavi posao, objava namijenjena za taj posao briše se iz liste s početnog zaslona te se prebacuje u arhivu. Unutar svih zaslona koji sadrže listu objava omogućena je promjena redoslijeda elemenata liste i to na temelju vremena izrade objave i udaljenosti korisnika u odnosu na lokaciju određenu prilikom izrade objave.

4.6. Zaslon za pretraživanje korisnika

Zaslon za pretraživanje korisnika nudi mogućnost pretrage korisnika prema korisničkom imenu. Ovo je poprilično jednostavan zaslon koji služi za pregled objava pojedinog korisnika.

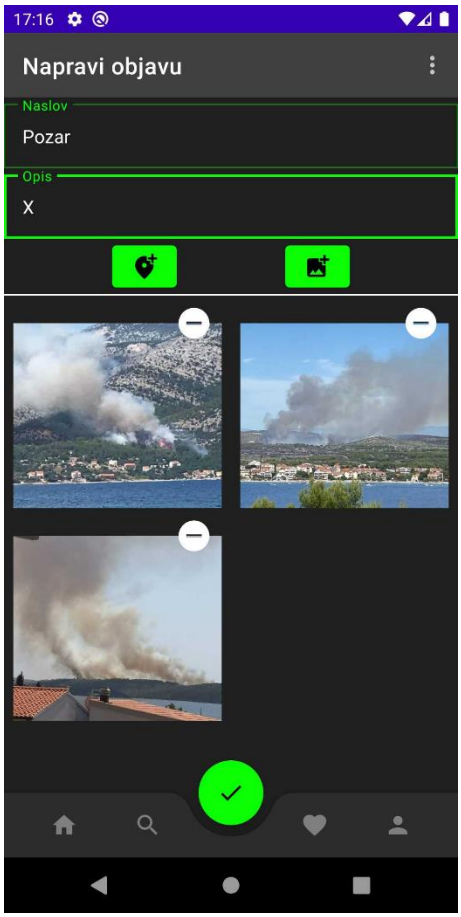


Slika 4.7. Prikaz zaslona za pretraživanje korisnika

Prema slici 4.7. vidljivo je da se željeni korisnik pretražuje prema korisničkom imenu. Odabirom korisnika iz liste, otvara se zaslon s objavama toga korisnika.

4.7. Zaslon za izradu objave

Zaslon za izradu objave, koji je vidljiv na slici 4.8., zahtjeva od korisnika unos naslova i opisa objave te odabir jedne ili više slika. Također, postoji opcija odabira lokacije kako bi ostali korisnici mogli vidjeti lokaciju posla za koji se prijavljuju. Ukoliko prijavljeni korisnik nema svoju fiksnu lokaciju (koja se može postaviti na zaslonu profila), onda je ova opcija obavezna. U slučaju da korisnik ima svoju lokaciju, prilikom izrade objave uzet će se u obzir ta lokacija, a ako se ipak odabere lokacija u zaslonu izrade objave ona će imati veći prioritet nego korisnikova fiksna lokacija.



Slika 4.8. Zaslon za izradu objave

Prilikom odabira lokacije postoje dvije opcije:

- trenutna lokacija
- odabir lokacije na karti

Prva opcija koristeći *FusedLocationProviderClient* [19] vraća koordinate trenutne lokacije. U nekim situacijama je moguće da ova metoda ne radi. Ukoliko je korisnik ugasio lokacijske usluge na uređaju nemoguće je dobiti koordinate trenutne pozicije čak iako ih je prije toga uspješno dohvatio. Gašenje lokacije na uređaju ne samo da onemogućuje trenutno pozicioniranje, već i briše prijašnje uspješne rezultate. Također, ako korisnik nema instalirane Google Play servise ne može koristiti navedenu funkcionalnost. Ukoliko korisnik ne može dohvatiti lokaciju koristeći ovu opciju potrebno je odabrati drugu opciju.

Druga opcija podrazumijeva implementaciju Maps SDK [20]. Kao i prva opcija, potrebno je imati instalirane Google Play servise. Međutim, postoji još jedna bitna stavka. Za korištenje ove funkcionalnosti potrebno je imati *API* ključ koji se može besplatno dobiti prateći dokumentaciju.

Nakon uspješne implementacije na zaslonu uređaja je interaktivna karta te se odabirom točke na karti pokazuje marker koji označuje mjesto od značaja.

Prilikom odabira slika potrebno je pokrenuti galeriju, odnosno upravitelj datotekama kako bi odabrali iste. Za pokretanje aktivnosti, kako unutar aplikacije, tako i za ostale aplikacije potrebno je koristiti mehanizam koji se naziva namjera (engl. *intent*). [1] Navedeno nam omogućuje stvaranje objekta koji enkapsulira zahtjev, odnosno skup informacija čiji su osnovni elementi akcija (opća radnja koju je potrebno izvršiti, engl. *action*) i podaci nad kojima se akcija izvršava (engl. *data*). Razlikuje se dvije vrste namjera: implicitna i eksplicitna. Eksplicitna definira ciljanu komponentu u samom objektu, dok implicitna traži od sustava da procjeni komponente temeljeno na podacima koje objekt sadrži. Na primjer, ako želimo pokrenuti sljedeću aktivnost unutar aplikacije, koristit ćemo eksplicitnu, a ako želimo pokrenuti vanjsku aplikaciju koristit ćemo implicitnu namjeru. Izlistanje koda 4.4. prikazuje jedan od načina kako pokrenuti vanjsku aplikaciju te obraditi rezultate dobivene od te aplikacije. U navedenom primjeru instancira se objekt korištenjem anonimne klase (engl. *anonymous class*) koja nasljeđuje *ActivityResultContract*. To je generička (engl. *generic*) klasa s dva parametra koji služe za definiranje tipova podataka s kojima će se raditi. Da bi koristili navedenu klasu, moramo prepisati (engl. *override*) dvije metode:

- *createIntent()* – služi za izradu objekta namjere te se na temelju njega pokreće odgovarajuća aktivnost/aplikacija
- *parseResult()* – služi za obradu podataka pristiglih iz otvorene aktivnosti/aplikacije (u ovome slučaju dobivamo *URI* (engl. *Uniform Resource Identifier*) odabrane slike ili više njih)

```
private val imagesResultContract = object : ActivityResultContract<Unit,
List<Uri?>?>() {
    override fun createIntent(context: Context, input: Unit?): Intent {
        return Intent(Intent.ACTION_GET_CONTENT).apply {
            type = "image/*"
            putExtra(Intent.EXTRA_ALLOW_MULTIPLE, true)
            action = Intent.ACTION_GET_CONTENT
        }
    }

    override fun parseResult(resultCode: Int, data: Intent?): List<Uri?>? {
        if (resultCode == RESULT_OK) {
            val images = mutableListOf<Uri?>()
            data?.clipData?.let { clipData ->
                val count = clipData.itemCount
                for (i in 0 until count) {
                    images.add(clipData.getItemAt(i).uri)
                }
```

```
            } ?: {
                data?.data?.let { data ->
                    images.add(data)
                }
            }.invoke()
            return images
        }
        return null
    }
}
```

Izlistanje koda 4.4. Pokretanje zaslona za odabir slika i dohvaćanje rezultata

4.8. Zaslon profila

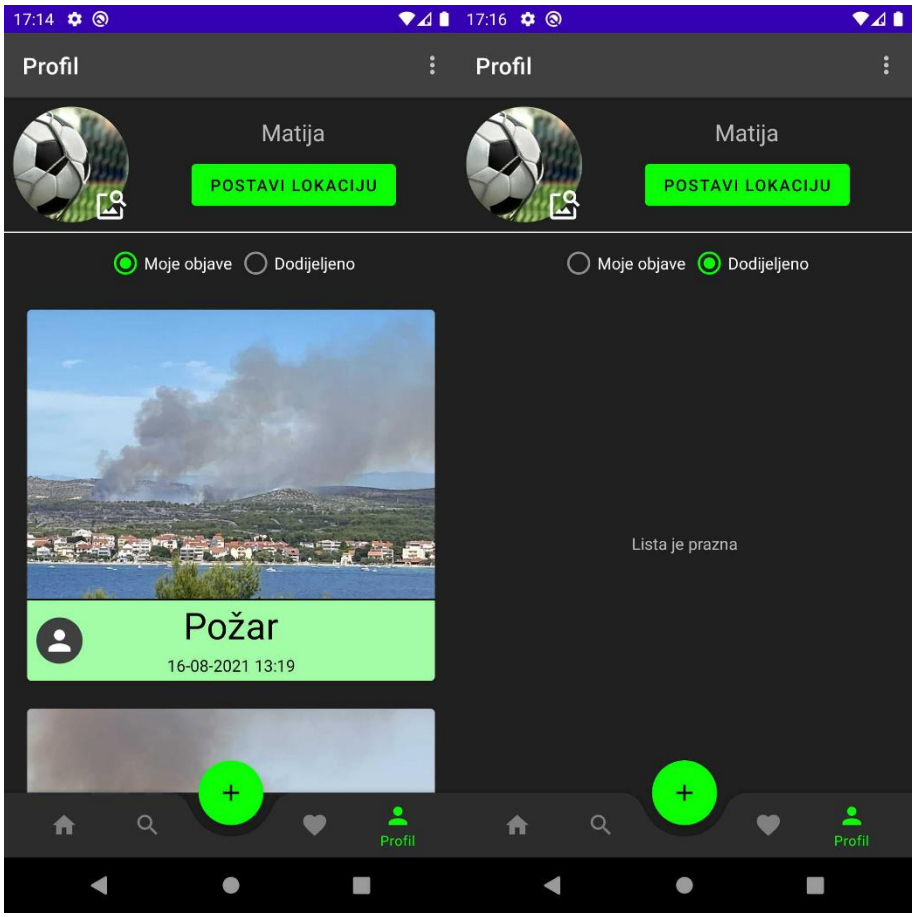
Zaslon profila omogućuje pregled informacija vlasnika korisničkog računa. To uključuje:

- sliku profila
- korisničko ime
- lokaciju korisnika
- listu izrađenih objava
- listu dodijeljenih poslova

Postavljanje slike profila radi na principu kao i odabir slike prilikom izrade objave. Pritiskom na ikonu za odabir slike (u donjem desnom kutu slike profila) otvara se upravitelj datoteka te se odabire slika. Pritiskom na samu sliku profila ista se otvara u cijelom zaslonu (engl. *fullscreen*). Unutar toga zaslona, sliku je moguće približiti (engl. *zoom in*) i udaljiti (engl. *zoom out*).

Postavljanje lokacije za korisnika nije obavezno, iako je poželjno. Kako ne bismo morali svaki put prilikom izrade objave dohvaćati lokaciju, korisnik je u mogućnosti postaviti fiksnu lokaciju koja će biti korištena u daljnjem radu. Više o samom postupku dohvaćanja lokacije je već objašnjeno u prethodnom potpoglavlju.

Slike 4.9. i 4.10. prikazuju zaslon profila. Korisnik može birati između dvije ponuđene opcije za prikaz vlastitih objava ili dodijeljenih poslova.



Slika 4.9. i 4.10. Zaslon profila

Prilikom odabira kategorije „Moje objave“ korisnik je u mogućnosti koristiti funkcionalnost pomicanja stavke unutar liste lijevo-desno (engl. *swipe*) u svrhu brisanja objave. To je moguće implementirati koristeći *ItemTouchHelper* objekt. On ne samo da omogućuje pomicanje stavke lijevo-desno, već je moguće koristiti i funkcionalnost mijenjanja redoslijeda elemenata u listi, to jest pomicanje gore-dolje. Ukoliko se korisnik nakon brisanja objave odluči da ipak ne želi obrisati istu, omogućena je opcija poništavanja radnje (engl. *undo*). Nakon brisanja se pri dnu zaslona pokazuje interaktivna obavijest te je pritiskom na istu moguće vratiti element u listu. Za što efikasniju implementaciju poništavanja radnje, poželjno je da se objava u trenutku brisanja ne briše na udaljenom serveru, već samo lokalno. Ukoliko korisnik želi vratiti objavu, potrebno je istu dodati u lokalnu bazu podataka. Tek nakon što se obavijest nakon nekoliko sekundi makne sa zaslona sigurni smo da smijemo obrisati objavu i na serveru. Navedena implementacija je prikazana izlistanjem koda 4.5.

```
private val itemTouchHelper = ItemTouchHelper(SwipeCallback { position ->
    postAdapter.getItemByPosition(position)?.let {
        viewModel.deletePostLocally(it)
        snackbar(requireContext().getString(R.string.post_deleted), actionMessage =
requireContext().getString(R.string.undo),
```

```
            action = {
                viewModel.postDeleted = false
                viewModel.insertPost(it)
                (binding.rvPosts.layoutManager as
LinearLayoutManager).scrollToPosition(position)
            }, onDismiss = {
                if (viewModel.postDeleted) viewModel.deletePost(it)
                viewModel.postDeleted = true
            })
    }
})
```

Izlistanje koda 4.5. Inicijalizacija ItemTouchHelper objekta te logika brisanja objave

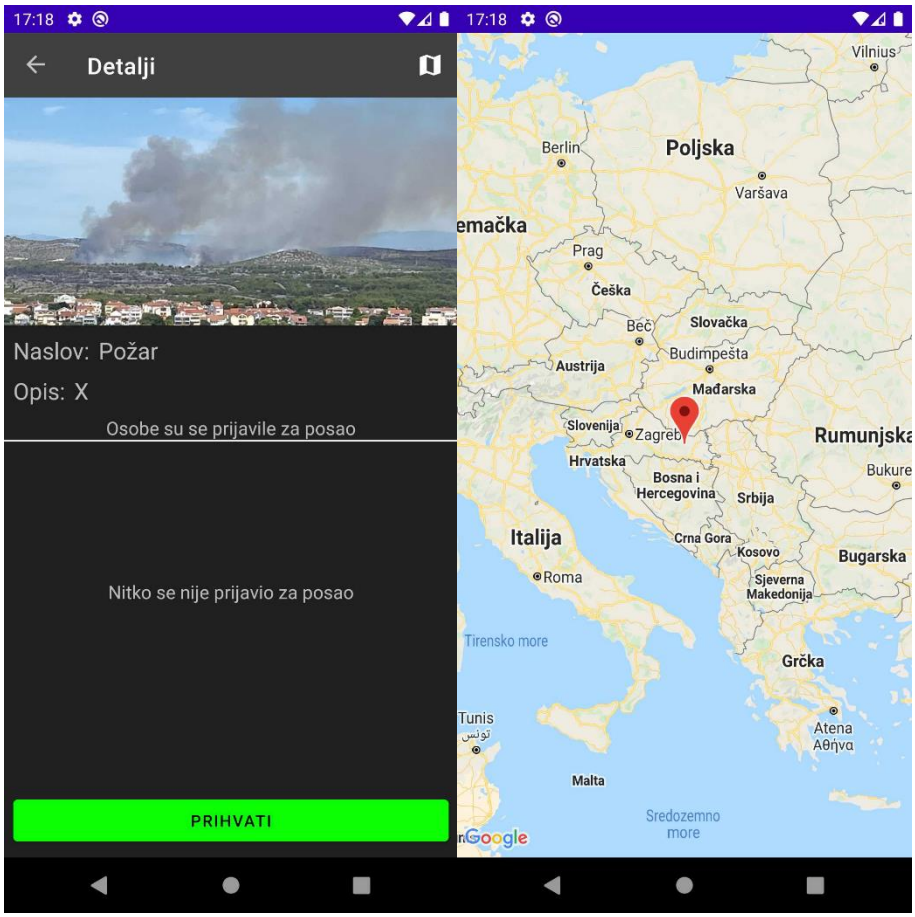
4.9. Zaslon detalja

Glavna svrha, odnosno problem koji ova aplikacija rješava odvija se upravo unutar zaslona detalja. Ovdje se korisnici mogu prijavljivati za poslove, ostavljati napomene, pregledavati detalje i slično. Sve promjene, to jest interakcije ostalih korisnika automatski su vidljive na svim uređajima bez potrebe ručnog osvježavanja. Navedeno je omogućeno uz već navedeni i objašnjeni osluškivač promjena kolekcije unutar baze podataka.

Kako bi ostali korisnici znali za lokaciju posla za koji se prijavljuju, pritiskom na ikonu u gornjem desnom kutu otvara se zaslon s kartom. Ulaskom u isti te dvostrukim pritiskom na označenu lokaciju otvara se Google Maps aplikacija unutar koje je moguće koristiti sve njezine funkcionalnosti, uključujući navigaciju. Zaslon s kartom, kao i inicijalni zaslon detalja nalazi se na slikama 4.11. i 4.12.

Također, pri vrhu zaslona se nalazi lista fotografija priloženih prilikom izrade objave. Pritiskom na jednu od fotografija, ista se otvara u cijelom zaslonu unutar kojega je moguće pregledavati i ostale pomicanjem lijevo-desno.

Nakon što korisnik napravi objavu te ju ostali korisnici vide isti su u mogućnosti prijaviti se za posao. Autor objave u bilo kojem trenutku može vidjeti prijavljene osobe te odabrati jednu za odrađivanje posla. Također, i ostali korisnici mogu vidjeti osobe prijavljene za posao, ali oni nemaju pravo odabira.



Slika 4.11. i 4.12. Zaslon detalja i karte

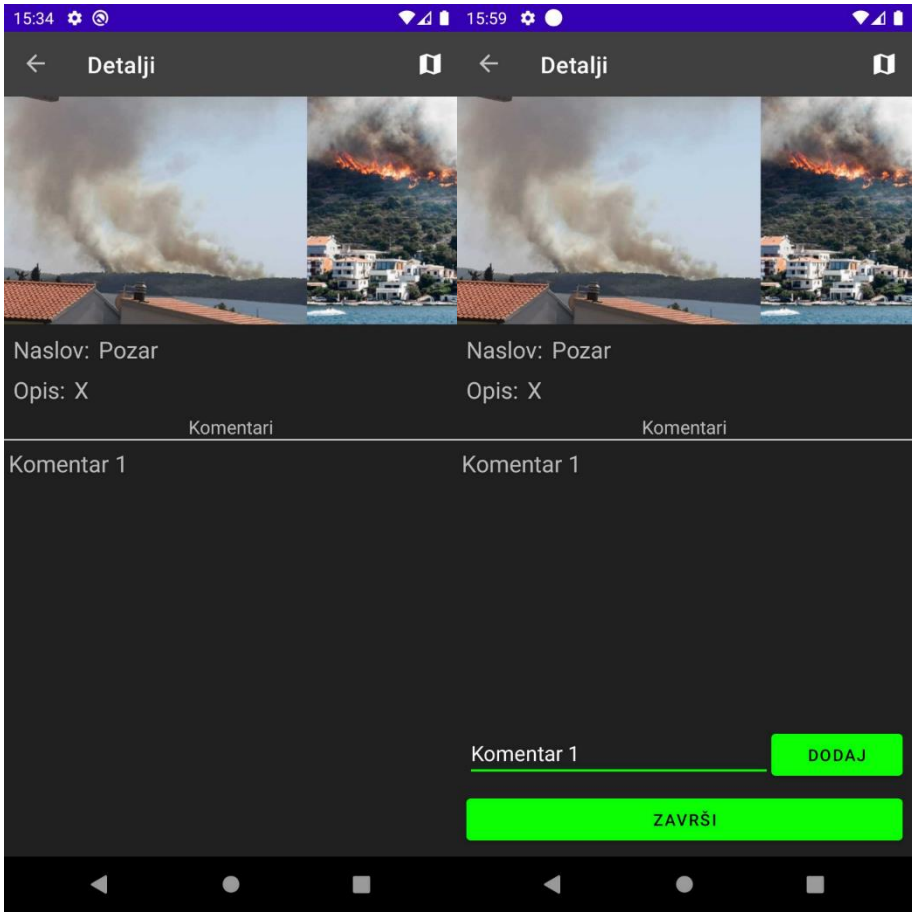
Nakon što autor objave prihvati korisnika za posao, dodijeljeni korisnik automatski prima obavijest o poslu na svoj uređaj. Pritiskom na obavijest otvaraju se detalji posla. Ova funkcionalnost omogućena je koristeći *Firebase Cloud Messaging*. Prilikom prijave korisnika u aplikaciju, isti dobiva odgovarajući token koji je jedinstven i služi za identifikaciju korisnika prilikom slanja obavijesti. Za iniciranje slanja obavijesti potrebno je napraviti *POST* zahtjev (engl. *request*) na *API* zadužen za slanje, odnosno primanje obavijesti. Svaki zahtjev u svom tijelu (engl. *body*) sadrži token primatelja te podatke koji se trebaju prikazati u obavijesti. Podaci sadrže i identifikator objave kako bi se ista mogla otvoriti nakon klika na obavijest.

Prilikom korištenja Firebasea za implementaciju slanja i primanja obavijesti potrebno je biti oprezan jer postoje dvije vrste obavijesti:

- *Notification (display) message* – obavijest se okida samo ako je aplikacija u prvom planu (engl. *foreground*)
- *Data message* – obavijest se okida nevezano je li aplikacija u prvom planu

Iako slične, svaku od navedenih obavijesti treba posebno uzeti u obzir kako bi u bilo kojem trenutku korisnik mogao primiti obavijest. Također, bitna stavka je da se i podaci poslani prilikom *API* poziva drugačije primaju ovisno o vrsti obavijesti.

Nakon što je korisnik prihvaćen za posao prikazuje se zaslon s mogućnošću objavljivanja napomena, to jest komentara vezanih za posao koji služe kako bi se poboljšala komunikacija. Komentari su vidljivi samo autoru objave te osobi prijavljenoj za taj posao. Na slici 4.13. i 4.14. nalaze se zasloni s komentarima. Mogućnosti unutar aplikacije se razlikuju ovisno o tome je li korisnik autor objave ili izvođač radova.



Slika 4.13. i 4.14. Zaslon s komentarima

Nakon što korisnik završi s poslom, pritiskom na tipku *Završi* završava se proces odrađivanja posla. Također, autor objave prima obavijest da je posao gotov.

4.10. Nedostaci aplikacije i mogućnosti za nadogradnju

Kao i svaka aplikacija i ova ima svoje mane i nedostatke. U ovome projektu postoji prostora za napredak, ali kako postoji rok za završetak istog, nije bilo mogućnosti za provedbu nadogradnje. Aplikacija, u ovim okvirima, sadrži sve prvobitno zamišljene funkcionalnosti.

Od samih funkcionalnosti, aplikacija bi mogla dodati opciju slanja i primanja poruka između korisnika. Trenutno, aplikacija podržava objavljivanje komentara unutar prihvaćenog posla, ali to je dosta „siromašan“ oblik komunikacije. Uz to, implementacija audio i video poziva bi pomogla u još boljoj i kvalitetnijoj komunikaciji korisnika.

Također, trenutno se unutar liste objava ne prikazuje slika profila autora objave, već je umjesto nje prikazana ikona osobe. Trenutno implementirano rješenje ne podržava jednostavan način dohvaćanja slike te je za rješenje ovoga problema potrebno refaktorirati dijelove koda zadužene za preuzimanje i prikaz objava. Svakako je plan riješiti ovaj problem prilikom izrade sljedeće inačice aplikacije.

Za korištenje aplikacije u komercijalne svrhe bilo bi dobro napraviti vlastito poslužiteljsko rješenje (engl. *backend*). Iako se aplikacija trenutno dosta oslanja na funkcionalnosti Firebasea, vlastiti poslužitelj pruža puno veće mogućnosti, od prilagođenih upita na bazu podataka do sigurnosnih značajki.

5. ZAKLJUČAK

U ovome diplomskom radu obrađena je implementacija mobilne aplikacije za Android operacijski sustav čija je svrha olakšati komunikaciju unesrećenih osoba i pružatelja pomoći prilikom ekoloških katastrofa. Također, izrada aplikacija vođena je korištenjem modernih tehnologija poput Dagger Hilta, Kotlin korutina te je cjelokupan projekt napravljen u skladu s MVVM arhitekturalnim obrascem koji omogućuje lakše testiranje koda i održavanje istog.

Ovaj projekt nastao je iz ideje kako bi unesrećeni ljudi lakše došli do pomoći te kako bi se pomagači lakše rasporedili na ponuđene poslove. Sama ideja dobivena je nakon teških potresa koji su pogodili Hrvatsku. Budući da je postojao veliki interes za pomaganjem nad potresom razorenim područjem, ovakva bi aplikacija uvelike pomogla u boljem i efektivnijem djelovanju te pružanju pomoći.

Iako aplikacija ne pruža sve mogućnosti kao primjeri navedeni u radu, zasigurno može biti od koristi. Također, daljnji rad i usavršavanje znanja u izradi nativnih mobilnih aplikacija će uvelike pomoći da ova aplikacija bude što kvalitetnija i dostupnija svima kojima ovakva pomoć bude potrebna.

LITERATURA

[1] Kotlin programski jezik, dostupno na: https://www.infoworld.com/article/3224868/what-is-kotlin-the-java-alternative-explained.html (15.7.2021.)

[2] Razvoj mobilnih aplikacija – predložak za laboratorijske vježbe

[3] Firebase, dostupno na: https://en.wikipedia.org/wiki/Firebase (17.7.2021.)

[4] Firebase, dostupno na: https://ignitevisibility.com/everything-need-know-google-firebase/ (17.7.2021.)

[5] Firebase, dostupno na: https://medium.com/codingurukul/introduction-to-firebase-f9f6ccc8a785 (17.7.2021.)

[6] Last Quake, dostupno na: https://play.google.com/store/apps/details?id=org.emsc_csem.lastquake (18.7.2021.)

[7] Last Quake zasloni, dostupno na: https://cdn.apkmonk.com/images/org.emsc_csem.lastquake.png (18.7.2021.)

[8] FEMA, dostupno na: https://play.google.com/store/apps/details?id=gov.fema.mobile.android&hl=en (18.7.2021.)

[9] FEMA zaslon, dostupno na: https://www.dhs.gov/sites/default/files/images/opa/15_0610_phones.png (18.7.2021.)

[10] Dijagram tijeka, dostupno na: https://sites.google.com/site/sandasutalo/oosnove-programiranja/pomocni-postupci-pri-programiranju/dijagram-tijeka (20.7.2021.)

[11] MVVM shema, dostupno na: https://developer.android.com/topic/libraries/architecture/images/final-architecture.png (21.7.2021.)

[12] RxJava, dostupno na: https://github.com/ReactiveX/RxJava (21.7.2021.)

[13] T. Nield, Learning RxJava - Build concurrent, maintainable, and responsive Java in less time, Packt Publishing Ltd., Birmingham, 2017

[14] Kotlin coroutines, dostupno na: https://kotlinlang.org/docs/coroutines-overview.html#tutorials (21.7.2021.)

[15] F. Babic, N. Srivastava, Kotlin Coroutines by Tutorials (1st Edition), Razeware LLC., Virginia US, 2018

[16] Dagger Hilt, dostupno na: https://developer.android.com/training/dependency-injection/hilt-android (22.7.2021.)

[17] Dagger Hilt, dostupno na: https://dagger.dev/hilt/ (22.7.2021.)

[18] RecyclerView, dostupno na: https://medium.com/geekculture/everything-you-should-know-to-create-a-recyclerview-3defdb660a2f (23.7.2021.)

[19] FusedLocationProviderClient, dostupno na: https://developer.android.com/training/location/retrieve-current (24.7.2021.)

[20] Maps SDK, dostupno na: https://developers.google.com/maps/documentation/android-sdk/start (24.7.2021.)

SAŽETAK

Cilj ovog diplomskog rada bio je izraditi mobilnu aplikaciju za Android operacijski sustav koristeći MVVM arhitekturalni obrazac koja omogućuje objavu, pregled i prijavu na poslove namijenjene za pružanje pomoći prilikom ekoloških katastrofa. Također, omogućena je komunikacija korisnika kroz dodijeljene poslove. Prilikom izrade aplikacije korišteno je razvojno okruženje Android Studio te programski jezik Kotlin. Aplikacija se sastoji od više aktivnosti unutar kojih su smješteni *fragmenti* od kojih je svaki zadužen za određenu funkcionalnost. Pri pokretanju aplikacije, nakon što se korisnik prijavi u sustav, prikazani su svi aktivni poslovi na koje se može prijaviti. Kroz jednostavno korisničko sučelje, korisnik se lako može prebacivati između različitih zaslona kao što su zaslon profila, zaslon izrade objave te zaslon pretraživanja korisnika. Odabirom određenog posla korisniku se otvara zaslon detalja te je u mogućnosti prijaviti se na isti. Unutar toga zaslona vidljive su sve informacije vezane za posao, uključujući galeriju slika, prikaz lokacije na karti i drugo. Nakon odobrene prijave za posao korisnik može ostavljati napomene koje su vidljive samo njemu i autoru objave. Svi su podaci sinkronizirani tako da je komunikacija između korisnika, odnosno različitih uređaja omogućena u stvarnom vremenu.

Ključne riječi:

Android, ekološke katastrofe, MVVM, pomoć, poslovi,

ABSTRACT

Mobile application to provide assistance in the event of environmental disasters

This paper aimed to create a mobile application for Android operating system using MVVM architecture pattern, which allows you to publish, review and apply for jobs intended to provide assistance in the event of environmental disasters. Also, user communication is enabled through assigned jobs. Android Studio development environment and programming language Kotlin were used to design the application. The application consists of several activities whose fragments are in charge of a certain functionality. When launching the application, upon logging in to the system, all active applicable jobs will be displayed to the user. Through a simple user interface, the user can easily switch between different screens such as a profile screen, post creation screen and user search screen. Selecting a specific job opens a detail screen where the user can apply to. Inside this screen, all job-related information, including an image gallery, a location view on a map and others, are displayed. After the approved job application, the user can publish notes that are visible only to him and the notes' author. All data are synchronized so that communication between users or different devices is enabled in real time.

Keywords:

Android, environmental disasters, help, jobs, MVVM

ŽIVOTOPIS

Matija Sokol rođen je 3. rujna 1997. godine u Našicama. Osnovnu školu pohađao je i završio u Našicama u školi kralja Tomislava u razdoblju od 2004. do 2012. godine. Zatim upisuje prirodoslovno matematičku gimnaziju u Srednjoj školi Isidora Kršnjavoga u Našicama koju pohađa od 2012. do 2016. godine. Nakon završene srednje škole upisuje preddiplomski studij računarstvo na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku 2016. godine. Tijekom treće godine preddiplomskog studija pohađa i uspješno završava Android Dev Academy u organizaciji Osijek Software City-a. Od rujna 2020. radi kao student-razvojni programer u poduzeću GDi d.o.o.

Vlastoručni potpis:

_____

Matija Sokol

PRILOZI

Na CD-u:

1. Mobilna aplikacija za pružanje pomoći prilikom ekoloških katastrofa.docx
2. Mobilna aplikacija za pružanje pomoći prilikom ekoloških katastrofa.pdf
3. Programski kod