Mobilna aplikacija za informiranje o željenim lokacijama i događajima

Kekelić, Stjepan

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

Sveučilišni studij

MOBILNA APLIKACIJA ZA INFORMIRANJE O ŽELJENIM LOKACIJAMA I DOGAĐAJIMA

Diplomski rad

Stjepan Kekelić

Osijek, 2023.

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

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 01.09.2023.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Stjepan Kekelić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1210R, 07.10.2021.
OIB studenta:	61048515161
Mentor:	prof. dr. sc. Krešimir Nenadić
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	doc. dr. sc. Tomislav Galba
Član Povjerenstva 1:	prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 2:	Robert Šojo, mag. ing. comp.
Naslov diplomskog rada:	Mobilna aplikacija za informiranje o željenim lokacijama i događajima
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	Kratko opisati način kako korisnici mogu odabrati događaje i lokacije vezane za društveni život a koji su njima zanimljivi i žele ih pratiti preko aplikacije. Potrebno je izraditi i opisati postupak izrade mobilne aplikacije koja će imati različite profile korisnika. Moderator treba imati mogućnost postavljanja događaja, lokacije događaja i relevantnih informacija o samom događaju. Registrirani korisnik može se pretplatiti na njemu zanimljive događaje i pratiti putem aplikacije objave vezane za taj događaj. Prikaz lokacije događaja rješiti prikazom na mapi. Tema rezervirana: Stjepan Kekelić
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: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	01.09.2023.

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, 14.09.2023.

Ime i prezime studenta:	Stjepan Kekelić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1210R, 07.10.2021.
Turnitin podudaranje [%]:	3

Ovom izjavom izjavljujem da je rad pod nazivom**: Mobilna aplikacija za informiranje o željenim lokacijama i događajima**

izrađen pod vodstvom mentora prof. dr. sc. Krešimir Nenadić

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

Svaki se korisnik interneta barem jednom našao u situaciji da pretražuje podatke po internetu u nadi kako bi pronašao željenu lokaciju u sredini u kojoj živi ili u koju je privremeno došao. Među traženim lokacijama često se pronalaze restorani koji su jeftiniji ili koji nude tradicionalnu hranu, hoteli koji imaju dobar smještaj, kafići koji će prenositi utakmicu ili koji imaju organizirane događaje, pekarnice koje rade kasno u noć, mjesta u sredini na kojima se održavaju sportski turniri, koncerti i ostala javna okupljanja. Većinom se u takvim situacijama prvo pretražuju web stranice određenih lokacija kako bi se provjerilo pružaju li željenu uslugu. Druga opcija u takvim situacijama je korištenje *Google Maps* aplikacije [1], gdje se na karti dobiva popis željenih mjesta te se tada lakše može pristupiti web stranici i podatcima nekog mjesta. Postoje i alternativna rješenja u takvim situacijama, no manje su popularna.

Cilj ovog diplomskog rada je pružiti korisnicima aplikaciju koja ima mogućnost pretraživanja željenih lokacija na sličan način, kao što je *Google Maps* aplikacija ili putem interneta ali da korisnici budu više uključeni kroz razne objave i razgovore koji su omogućeni za tu specifičnu lokaciju. Aplikacijom upravljaju moderatori, kojima je zadaća odobriti ili odbiti zahtjeve za novu lokaciju gdje moraju voditi računa o ispravnosti podataka lokacije i da prava osoba bude administrator lokacije. Administrator lokacije je osoba kojoj je odobren zahtjev za novu lokaciju, a ona ima sve ovlasti uređivanja podataka o lokaciji, kreira i briše objave i upravlja razgovorom unutar lokacije. Ostali korisnici imaju mogućnost pregledavanja razgovora unutar lokacije, pisanja novih poruka, pregledavanja objava, ali i mogućnost kreiranja nove objave. Svaki se korisnik može pretplatiti na određenu lokaciju kako bi dobivao obavijesti o novim događajima. Takav način funkcioniranja aplikacije trebao bi pružiti puno više informacija od različitih korisnika za određenu lokaciju jer korisnici imaju mogućnost grupne komunikacije unutar pojedine lokacije i mogućnost kreiranja objava koje mogu biti događaji ili informativnog karaktera. Tako lokacije ostaju u središtu pozornosti, a velika zadaća je prepuštena ljudima, odnosno korisnicima aplikacije.

U drugom poglavlju ovog rada opisana su već gotova rješenja za ovu problematiku, odnosno rješenja koja rješavaju barem dio problema koji je opisan u ovom radu. Neke na sličan, a neke na malo drugačiji način, no svaka od tih aplikacija ima nekoliko razlika u odnosu s rješenjem koje se opisuje u ovom radu. U trećem poglavlju opisan je razvoj aplikacije i korištene tehnologije za izradu aplikacije. Redom su opisane tehnologije kao što je Firebase, Google Maps SDK, Navigacija, Hilt, *Composable*, Kotlin korutine, *ViewModel* i *Repository*. Zatim, u četvrtom

poglavlju prikazana je struktura aplikacije, odnosno izgled sučelja aplikacije i opisan je rad aplikacije za svaki zaslon unutar nje. Rad završava petim poglavljem, gdje je dan kratak zaključak.

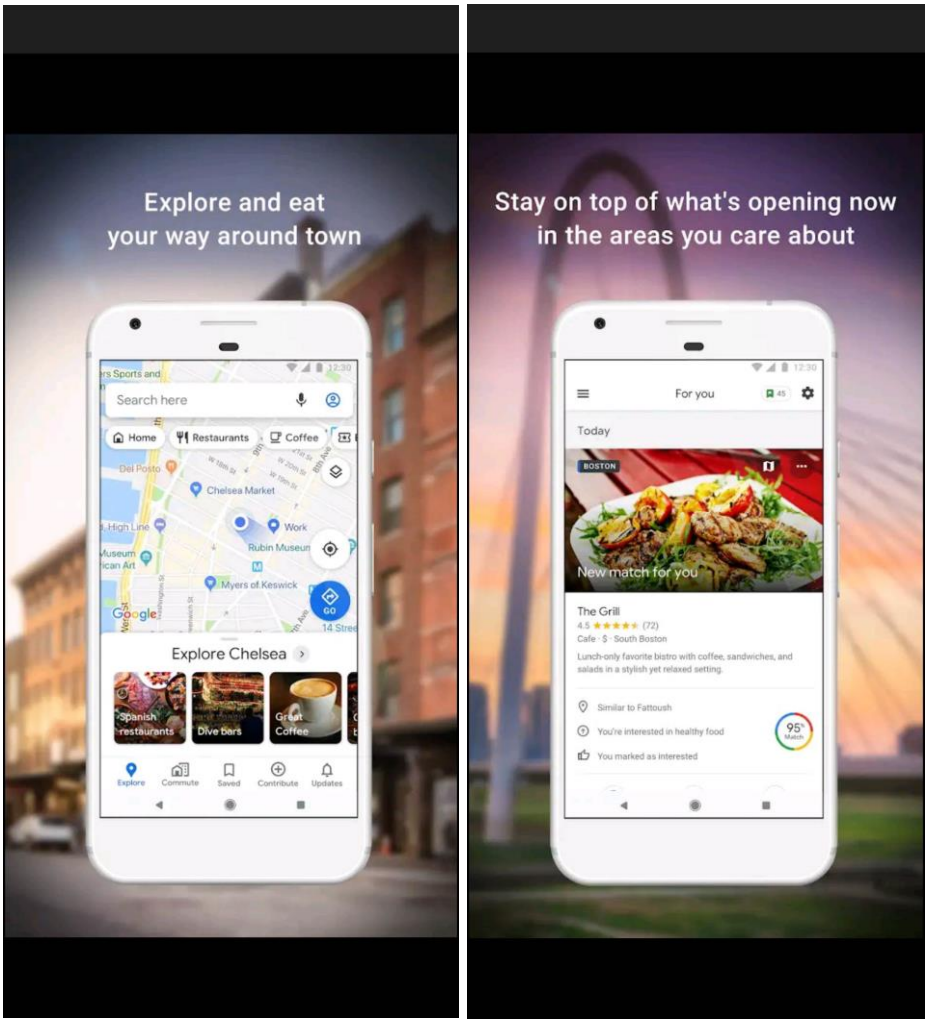
2. PREGLED POSTOJEĆIH RJEŠENJA

Postoje brojne aplikacije koje omogućuju informiranje korisnika o željenim lokacijama i događajima, od aplikacija koje nude pregled mjesta u korisnikovoj blizini gdje se mogu odabrati različite kategorije mjesta (restorani, kafići, trgovine, muzeji, banke i tako dalje) do aplikacije koje nude mogućnost pregleda događaja u korisnikovoj blizini ili istraživanja turističkih atrakcija. Većina tih aplikacija koristi servise za dohvaćanje lokacija i samo ih prikazuju, neke na bolji i jednostavniji način, a neke na lošiji, no korisnik treba odučiti koju će aplikaciju koristiti.

Najpoznatija aplikacija koja omogućuje pregled mjesta je *Google Maps* koju većina korisnika upotrebljava u svakodnevnim aktivnostima, ali postoje i druge aplikacije. Neke od njih jednostavno koriste usluge *Places API* (engl. *Application Programming Interface*) gdje se mogu dobiti sve kategorije i lokacije kao i u navedenoj *Google Maps* aplikaciji. Korištenjem te usluge stvaranje novih mjesta je otežano i ovisno o *Google Maps* aplikaciji. Iz tog razloga prvo treba odlučiti što se aplikacijom želi postići i omogućiti korisnicima. U ovom radu nije korištena navedena usluga jer je cilj na kreiranju mjesta kojima će moći upravljati korisnici i kreirati događaje za ta mjesta, ali detaljniji postupak rada aplikacije je opisan u sljedećim poglavljima. U idućim potpoglavljima navedeno je 5 sličnih postojećih rješenja koje bi donekle mogle zamijeniti aplikaciju izrađenu u ovom radu.

2.1. Google Maps

Prvi primjer sličnih rješenja jedna je od najpoznatijih aplikacija općenito koja ima preko 10 milijardi preuzimanja na *Google Play* trgovini, a to je *Google Maps*. Navedena aplikacija služi za razne svrhe: služi kao navigacija do odredišne lokacije, služi za pregledavanje rasporeda vožnje javnih prijevoza i omogućuje *Street View* pregled ulica pomoću slika od 360 stupnjeva. Moguće je istražiti koja se sve mjesta nalaze u blizini korisnika, ali i u drugim područjima svijeta korištenjem karte na kojoj se prikazuju mjesta.

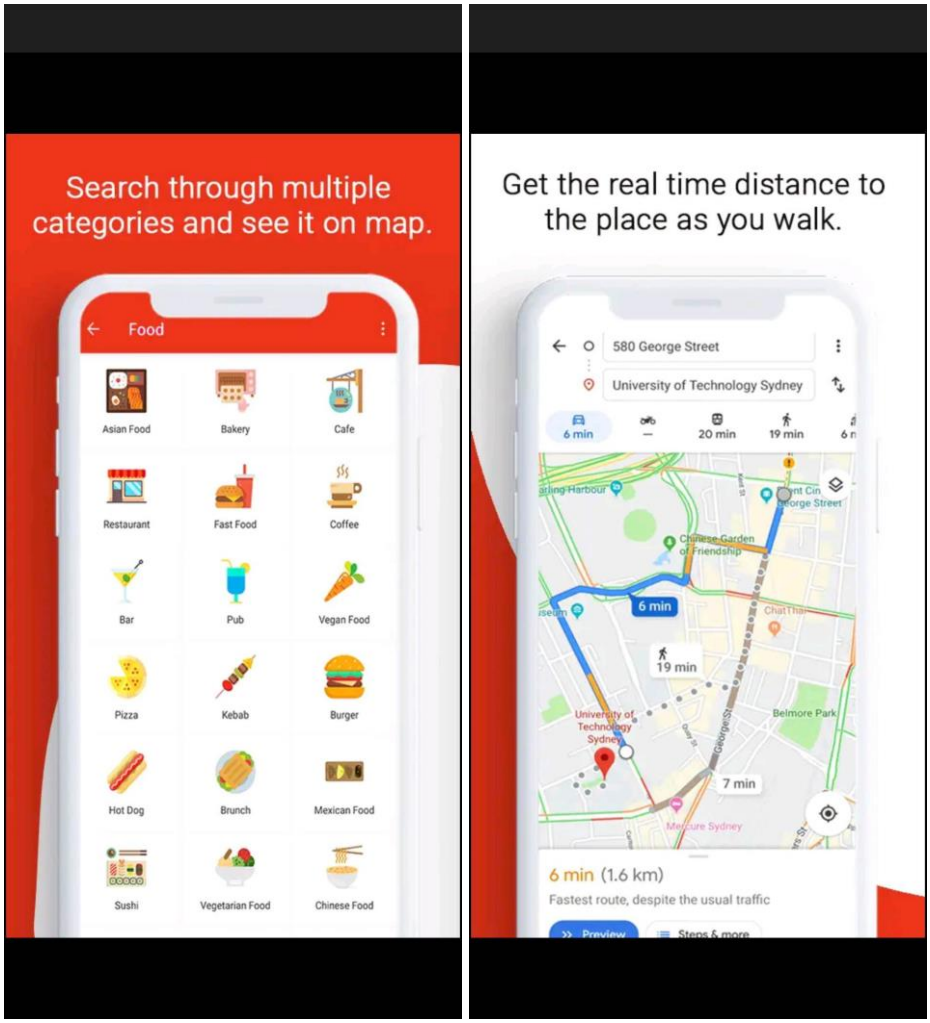


Slika 2.1. Prikaz zaslona aplikacije *Google Maps*, preuzeto s https://play.google.com/store/apps/details?id=com.google.android.apps.maps

Na slici 2.1. vidi se karta na kojoj je označen korisnik i sva javna mjesta oko njega. Korisnik ima mogućnost pritiska na marker koji označava pojedino mjesto te će tada dobiti više informacija o tome mjestu, kao što je: opis mjesta, slike, recenzije drugih korisnika i još mnogo toga. Svaki korisnik ima mogućnost napraviti mjesto gdje se trebaju unijeti potrebni detalji o mjestu kao što su naziv mjesta, adresa, kategorija i tako dalje. Ono što ova aplikacija nema, to je upravljanje događajima. Primarna svrha aplikacije je pružanje detalja o mjestima ali ne i o događajima koji se tamo događaju. Korisnik sam mora pomoću poveznica na web stranice koje se nalaze u detaljima mjesta doći do takvih podataka.

2.2. Near Me: Find Places Around Me

Sljedeći primjer aplikacije koja omogućuje jednostavno i brzo istraživanje mjesta u blizini korisnika je aplikacija *Near Me: Find Places Around Me* [2].

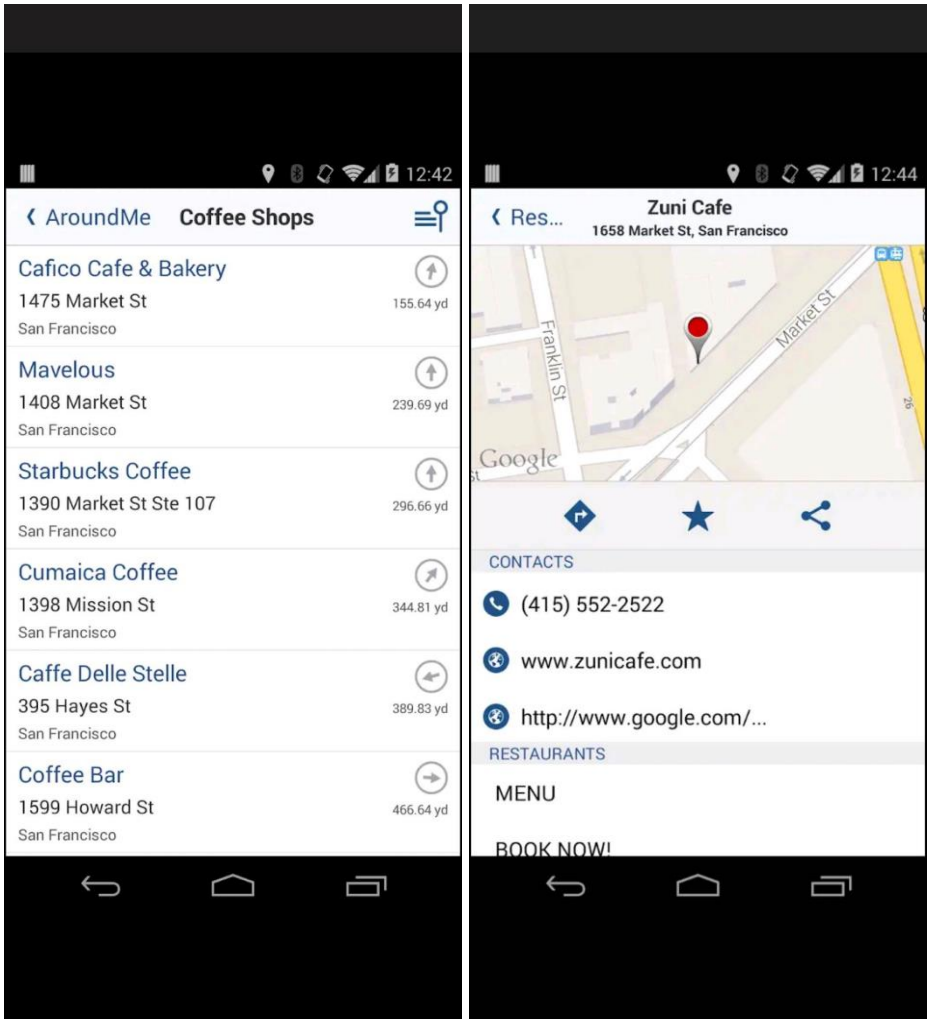


Slika 2.2. Prikaz zaslona aplikacije *Near Me: Find Places Around Me*, preuzeto s https://play.google.com/store/apps/details?id=com.tweakersoft.aroundme

Na slici 2.2. prikazan je osnovni izgled aplikacije, gdje se na lijevoj slici odabere kategorija i zatim kao što je vidljivo na desnoj slici otvara se već spomenuta aplikacija *Google Maps* koja pokazuje gdje se ta mjesta nalaze. Ova aplikacije je prvenstveno navedena zbog svoje jednostavnosti i lakoće korištenja, iako takvih aplikacija ima jako puno koje koriste usluge *Places API*-a kako bi dohvatile dostupna mjesta u blizini i zatim ih prikazali korisniku. U takvim aplikacijama nema mogućnosti kreiranja novih mjesta, osim ako se to ne napravi pomoću *Google Maps* aplikacije.

2.3. AroundMe

Treća aplikacija, vrlo slična prethodno opisanoj aplikaciji je nešto popularnija od nje, a to je aplikacija *AroundMe* [3].

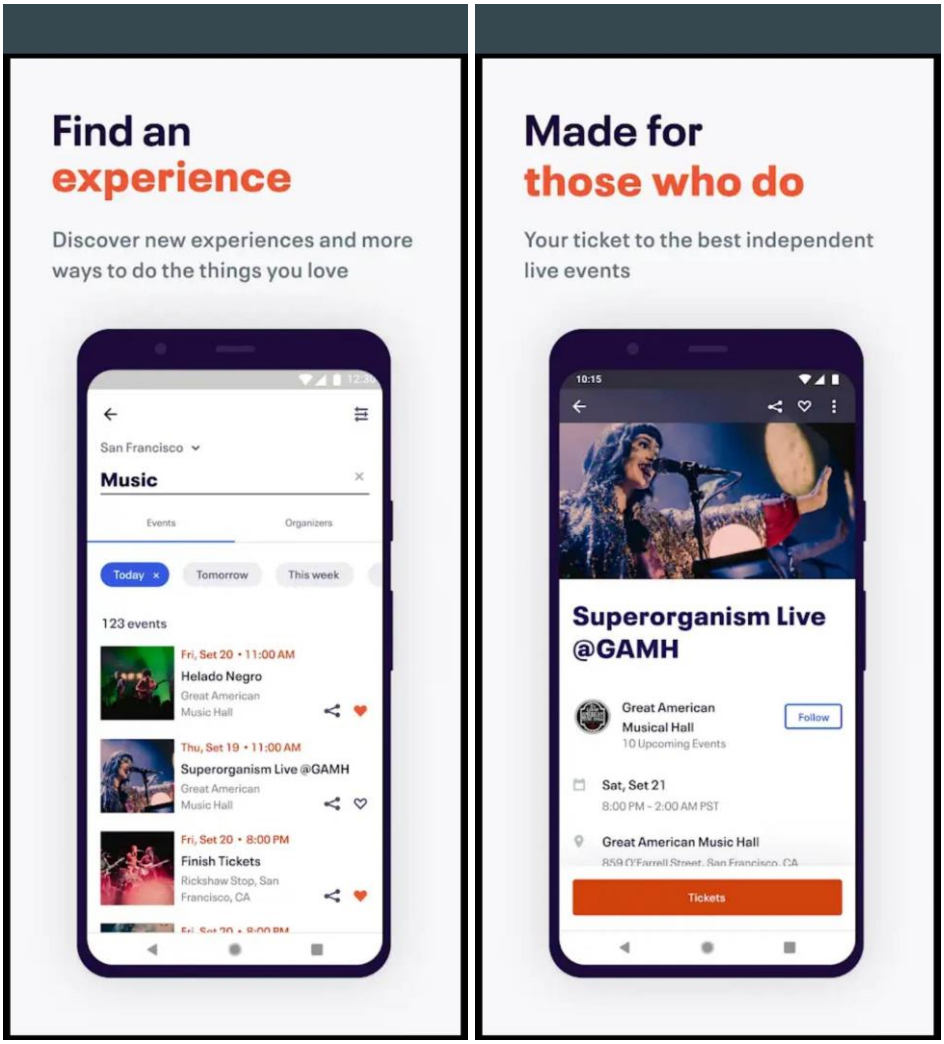


Slika 2.3. Prikaz zaslona aplikacije *AroundMe*, preuzeto s https://play.google.com/store/apps/details?id=com.tweakersoft.aroundme

Za razliku od prošle aplikacije, ova aplikacije je nešto više popularnija u odnosu na prethodnu, barem po broju preuzimanja na *Google Play* trgovini. Na slici 2.3. prikazan je izgled aplikacije gdje su na lijevoj slici prikazana sva mjesta u blizini, nakon što je korisnik odabrao kategoriju kafića. Za razliku od prošle aplikacije, ova aplikacija pruža pregled detaljnog opisa mjesta i omogućava pregled tog mjesta u *Google Maps* aplikaciji. Mjesta koja se prikazuju u ovoj aplikacije dohvaćaju se pomoću servisa *Places API* kao i kod prethodne aplikacije. Može se uočiti da niti kod ove aplikacije nema podrške za događaje koji se događaju na mjestima koje će aplikacija prikazati.

2.4. Eventbrite – Discover events

Sljedeća aplikacija je nešto drugačija u odnosu na prethodne, a to je aplikacija *Eventbrite - Discover events* u kojoj je cilj na prikazivanju događaja u okruženju korisnika [4].

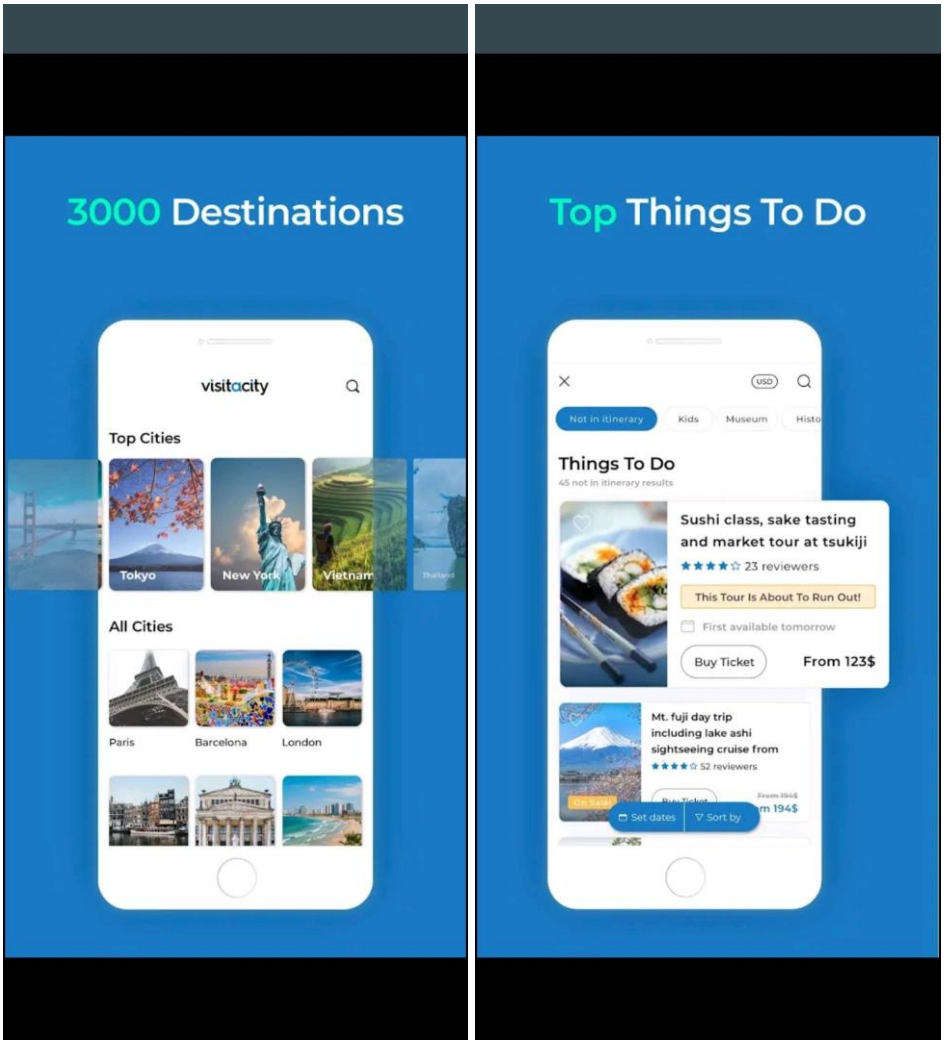


Slika 2.4. Prikaz zaslona aplikacije Eventbrite – Discover events, preuzeto s https://play.google.com/store/apps/details?id=com.eventbrite.attendee

Kao što je vidljivo iz naziva aplikacije ovdje nije naglasak na prikazivanju mjesta već događaja. Na slici 2.4. lijevo prikazana je lista sa svim glazbenim događajima koji se održavaju danas u San Franciscu. Vrlo jednostavna je i intuitivna za korištenje, a osim toga pruža i mogućnost kupnje karte za određene događaje što se može vidjeti na desnoj slici 2.4. Kao nedostatak ove aplikacije može se istaknuti nedostatak podrške za prikaz događaja na karti ili barem prikaz udaljenosti korisnika do događaja kako bi korisnik odmah uočio je li to njemu blizu ili daleko.

2.5. Visit A City

Posljednja opisana aplikacija je *Visit A City* koja služi za istraživanje gradova i prikaz atrakcija koje turisti mogu posjetiti [5].



Slika 2.5. Prikaz zaslona aplikacije *Visit A City*, preuzeto s https://play.google.com/store/apps/details?id=com.visitacity.visitacityapp&hl=hr

Ova aplikacija omogućuje pregled nekoliko tisuća najpoznatijih turističkih gradova i pregled mjesta i atrakcija koje se tamo nalaze kao što je to vidljivo na slici 2.5. Za svaku destinaciju u odabranom gradu može se vidjeti opis, gdje se ona nalazi na karti, pa čak i koje destinacije se nalaze u blizini odabrane. Aplikacija je odlična za turiste i ljude koji su se tek doselili u neki grad i žele ga istražiti, ali za ljude koji žive u tom gradu i za istraživanje mjesta i budućih događaja nije previše korisna. Uostalom aplikacija nema podršku za kreiranjem novih destinacija tako da ako autori aplikacije nisu upoznati s istom ona neće biti prikazana u aplikaciji.

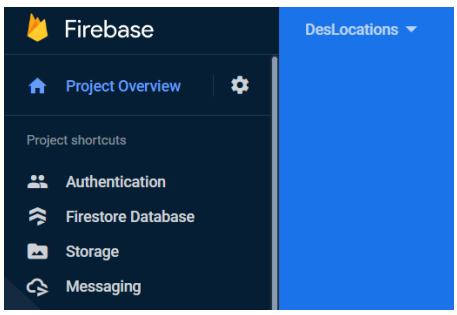
3. RAZVOJ APLIKACIJE I KORIŠTENE TEHNOLOGIJE

U ovom poglavlju prikazan je razvoj aplikacije zajedno s opisom korištenih tehnologija. Za izradu aplikacije korišten je Android Studio s verzijom Flamingo koja ima potpunu podršku za kreiranje Android aplikacija pomoću Jetpack Compose razvojnog okvira. Jetpack Compose je korišten za izradu aplikacije jer je prvenstveno novija tehnologija i ima brojne prednosti u odnosu na stariji način izrade Android aplikacija pomoću View razvojnog okvira. Jetpack Compose je potpuno drugačiji razvojni okvir u odnosu na View jer ne razdvaja izgled od funkcionalnosti. U sljedećim potpoglavljima detaljnije su opisane komponente i funkcionalnosti kako je Jetpack Compose korišten za izradu aplikacije. Kao programski jezik za izradu aplikacije korišten je Kotlin jer je Jetpack Compose izgrađen pomoću njega, a korištenje nekog drugog programskog jezika trenutno nije moguća opcija.

U sljedećim potpoglavljima detaljnije će biti objašnjeno kako su usluge Firebasea i Google Maps korištene za izradu aplikacije. Nadalje, opisano je kako je složena navigacija u aplikaciji i koja joj je uloga. Zatim je opisan Hilt te nakon toga *Composable* kao osnovni element Jetpack Compose razvojnog okvira. Nakon toga su opisane Kotlin korutine i zatim *ViewModel* i *Repository* klase koje su korištene kao dio *MVVM* (engl. *Model-View-ViewModel*) obrasca dizajna softvera.

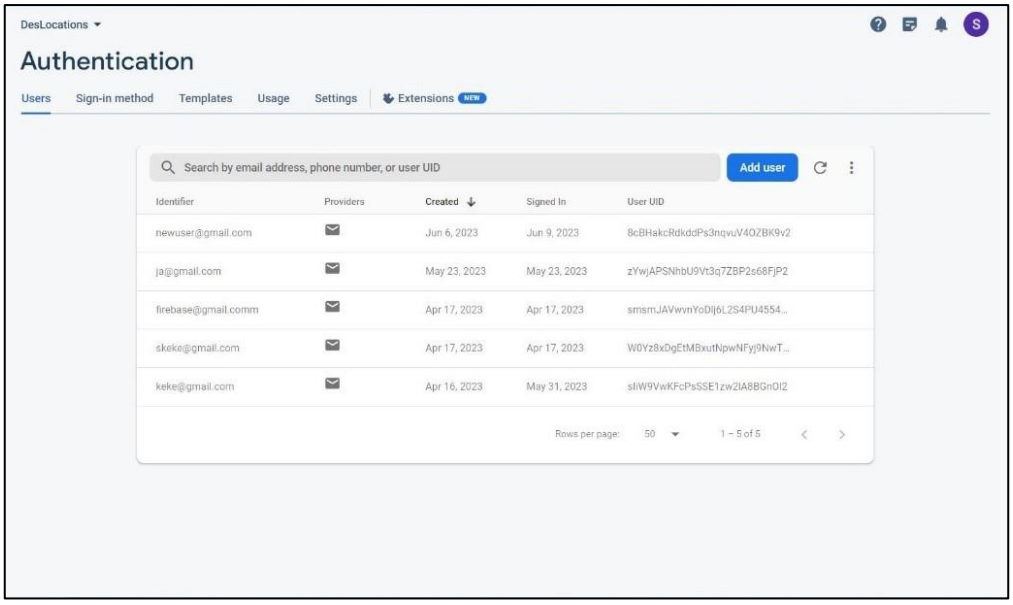
3.1. Firebase

Firebase je platforma koja olakšava izradu i razvoj aplikacija pružajući brojne mogućnosti i usluge. Milijuni tvrtki diljem svijeta koriste usluge Firebasea, što ga čini jako popularnim [6]. Brojne su usluge koje pruža Firebase, pa je za izradu aplikacije korišteno nekoliko koje su prikazane na slici 3.1., a to su *Authentication, Firestore Database, Storage* i *Cloud Messaging*.



Slika 3.1. Prikaz korištenih Firebase usluga

Prva korištena Firebase usluga je *Authentication*, koja je korištena kako bi se korisnik u aplikaciji mogao registrirati i prijaviti. Registracija, odnosno prijava nije potrebna svaki put prilikom ulaska u aplikaciju već se ona pamti. Kada je korisnik već prijavljen u sustav, tada će mu se prikazati početni zaslon aplikacije, u suprotnom, prikazat će se zaslon za prijavu, odnosno zaslon za registraciju novog korisnika. Prilikom kreiranja *Authentication* usluge potrebno je odabrati način prijave u sustav, a za potrebe ove aplikacije odabran je način prijave pomoću adrese elektroničke pošte i zaporke. Moguće je odabrati još brojne druge načine za prijavu kao što je Google ili Facebook autentikacija. Na slici 3.2. prikazan je početni zaslon *Authentication* usluge nakon što su se neki korisnici registrirali, a iz sigurnosnih razloga nisu prikazane stvarne adrese elektroničke pošte. Kao što se na slici vidi moguće je pristupiti adresi elektroničke pošte, jedinstvenom broju korisnika i datumu kreiranja i prijave, a zaporka nije izložena za pregled nikome tako da korisnici mogu uživati potpunu privatnost.



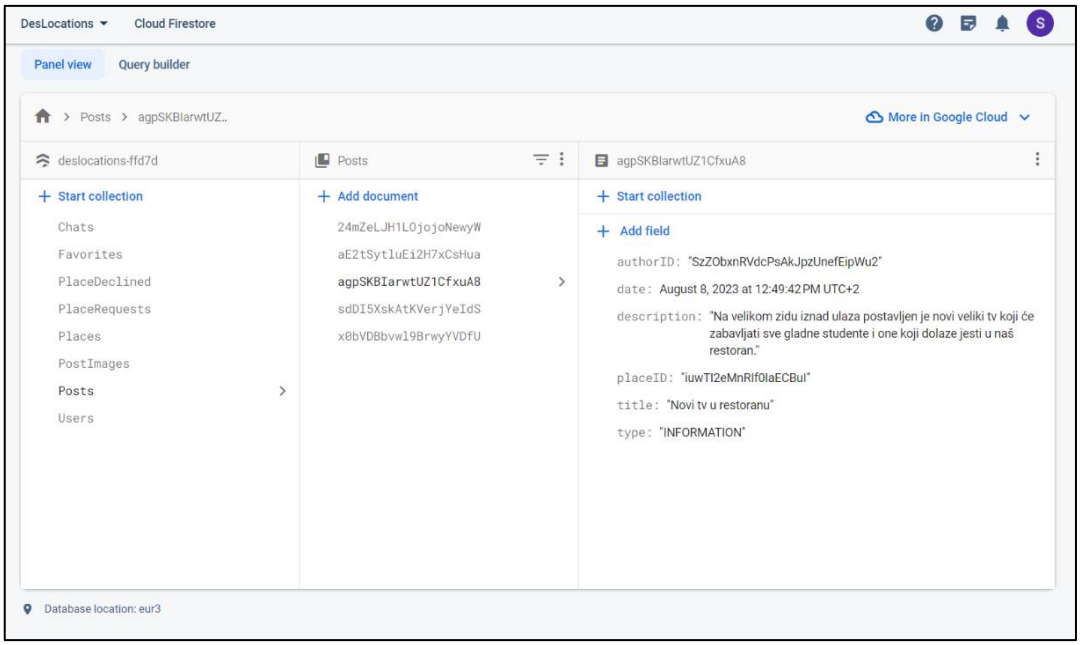
Slika 3.2. Prikaz registriranih korisnika pomoću *Authentication* usluge

Sljedeća korištena Firebase usluga je *Firestore Database*. To je fleksibilna i skalabilna baza podataka za mobilni, web i poslužiteljski razvoj aplikacija, a ujedno održava sinkronizaciju podataka u klijentskim aplikacijama tijekom stvarnog vremena [7]. Za potrebe aplikacije u *Firestore Database* spremaju se svi podatci nužni za rad aplikacije, a u to spadaju:

- detalji korisnika,
- detalji o mjestima koji se prikazuju korisnicima,

- detalji mjesta za koje korisnik želi postati administrator, a moderator aplikacije još nije prihvatio,
- detalji mjesta koja je moderator aplikacije odbio, a korisnik još nije obrisao,
- detalje objava (engl. *post*) za pojedina mjesta
- detalje slika koje su uključene u pojedinu objavu
- detalji mjesta koja su omiljena pojedinim korisnicima
- detalji razgovora za svako pojedino mjesto

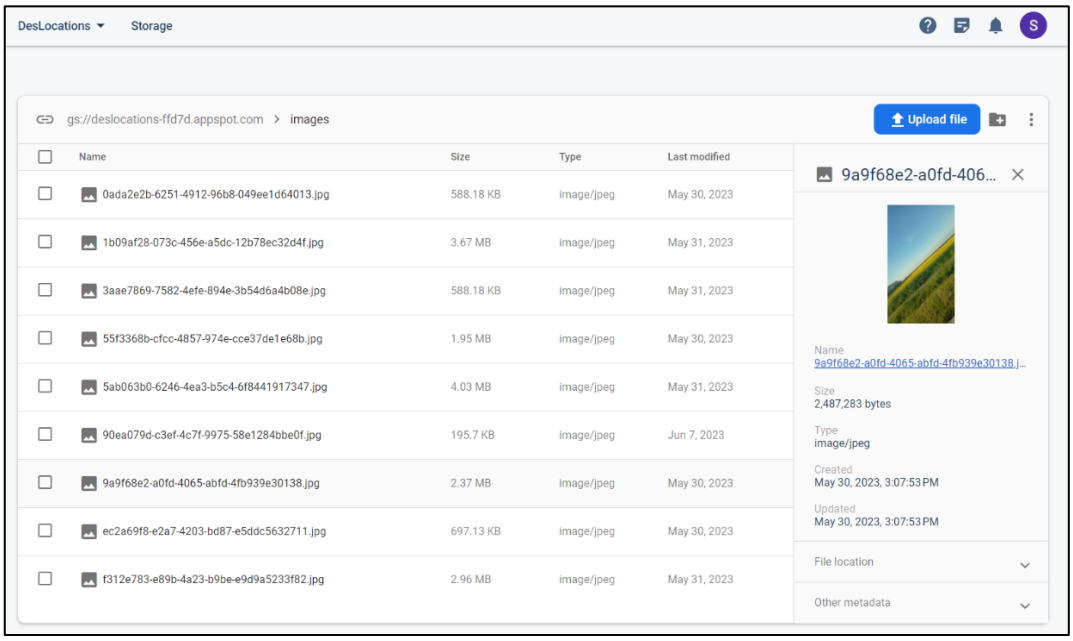
Sve to je potrebno kako bi aplikacija radila, a na slici 3.3. prikazani su ti podatci kako se oni spremaju u NoSQL (engl. *No Structured Query Language*) modelu podataka pomoću dokumenata koji sadrže polja i kolekcije koje *Firestore Database* koristi. Tako na primjer za potrebe objava jednog mjesta spremaju se podatci: *ID* autora, datum kreiranja objave, opis objave, *ID* mjesta za koje je objava napravljena, naslov objave i vrsta objave. No osim toga, jedna objava može sadržavati nijednu ili nekoliko slika, a slike nije moguće spremati u *Firestore Database*. Iz tog razloga napravljena je nova kolekcija koja ima naziv *PostImages* gdje se sprema *ID* objave i *URL* na kojemu je spremljena slika korištenjem *Storage* usluge.



Slika 3.3. Prikaz spremanja podataka pomoću *Firestore Database* usluge

Firebase usluga *Storage* je izgrađena za programere aplikacija koji trebaju pohranjivati i posluživati sadržaj koji generiraju korisnici, poput slika, zvuka i videa [8]. U svrhu ove aplikacije se pomoću navedene usluge pohranjuju slike. Na slici 3.4. prikazan je popis spremljenih slika

pomoću *Storage* usluge gdje se može vidjeti: ime, veličina, vrsta slike, datum kreiranja slike, a moguće je vidjeti i spremljenu sliku. No ti svi podatci nisu toliko važni kao što je važan podatak na kojoj lokaciji je spremljena slika jer se ta informacija koristi prilikom dohvaćanja slike kako bi ju korisnici mogli vidjeti u aplikaciji.



Slika 3.4. Prikaz spremanja slika pomoću *Storage* usluge

Posljednja korištena Firebase usluga je *Cloud Messaging* usluga koja služi za pouzdano slanje poruka bez ikakvih troškova. Može se obavijestiti klijentska aplikacija da su novi podatci dostupni za sinkronizaciju ili se mogu slati obavijesti kako bi se potaknuo ponovni angažman korisnika [9]. Za potrebe ove aplikacije odabrana je *Cloud Messaging* usluga kako bi se korisnicima omogućilo slanje obavijesti kada je nova objava određene lokacije dostupna u bazi podataka. Tako će korisnik koji je pretplaćen na određenu lokaciju imati mogućnost primanja obavijesti za svaku novu objavu koja se napravi unutar te lokacije. Kako bi korisnik primio obavijest, aplikacija ne mora biti aktivna, već se obavijest prikazuje i na zaključanom zaslonu.

3.2. Google Maps SDK

Za potrebe ove aplikacije korišten je Google Maps SDK (engl. *Software Development Kit*) kako bi se omogućio prikaz mjesta na karti koje su napravili drugi korisnici. Osim toga karta je korištena prilikom kreiranja novog mjesta kako bi korisnik označio lokaciju. No prije početka

njegovog korištenja u aplikaciji je potrebno napraviti novi projekt na *Google Cloud Platform*, omogućiti Google Maps SDK te u njemu kreirati API ključ koji je zatim potrebno dodati u projekt aplikacije. Nakon toga Google karta se jednostavno može koristiti u Jetpack Compose razvojnom okviru. Prikaz korištenja komponente *GoogleMap* vidljiv je na slici 3.5. U ovom slučaju komponenta *GoogleMap* služi za prikaz odabrane lokacije na karti ako korisnik pretražuje mjesta pomoću liste lokacija. Karta se jednostavno koristi navođenjem imena te komponente, a ostali parametri u konstruktoru *GoogleMap* komponente navedeni su kako bi se karta bolje prikazala. Osim *GoogleMap* komponente na slici je prikazana i komponenta *MarkerInfoWindow* koja će nacrtati marker na karti kako bi korisnik mogao vidjeti tu lokaciju. U ovom slučaju uvijek će biti prikazan samo jedan *Marker*, ali lako se može napraviti da komponenta *GoogleMap* prikazuje više markera što je i napravljeno prilikom prikaza mjesta u korisničkom okruženju.

```
GoogleMap(
    modifier = Modifier.fillMaxWidth(),
    properties = properties,
    cameraPositionState = cameraPositionState,
    uiSettings = uiSettings,
    onMapLoaded = {...}
) {
    MarkerInfoWindow(
        state = markerState,
        icon = bitmapDescriptorFactory(context, Category(category).iconResourceId),
        onInfoWindowClick = { it: Marker
            it.hideInfoWindow()
        },
        content = { it: Marker
            CustomInfoWindow(
                title = placeName,
                category = category,
                categoryName = categoryName,
                address = address,
                distance = distance
            )
        }
    )
```

Slika 3.5. Prikaz komponente *GoogleMap*

3.3. Navigacija

Unutar Jetpack Compose razvojnog okvira postoji navigacijska komponenta koja pomaže u implementiranju navigacije pomoću jednostavnih klikova na gumb do složenijih primjera kao što su aplikacijske trake i navigacijske ladice [10]. Kako bi se navigacija mogla koristiti prvo ju je potrebno uključiti unutar *build.gradle* datoteke i nakon sinkroniziranja moguće je upotrijebiti *NavHost* komponentu i *NavController* objekt koji su od iznimne važnosti unutar navigacije. Također je za navigaciju važna komponenta unutar aplikacije gdje se navode sva odredišta koje

korisnik može posjetiti. Za potrebe navedenog, napravljena je klasa *Screen* koja je u ovom slučaju *sealed* klasa koja posjeduje fiksni skup mogućih stanja. *Sealed* klasa se koristi za definiranje mogućih stanja klase koja ne moraju biti istog tipa. Unutar klase Screen stanja su predstavljena objektima koji imaju naziv odredišta na koji vode unutar navigacije. Slika 3.6. upravo predstavlja tu klasu gdje se vide sva odredišta koje pomoću navigacije korisnik može posjetiti unutar aplikacije. Treba naglasiti da je isto moguće napraviti i pomoću *enum* klasa koje Kotlin programski jezik pruža ili pomoću *XML* datoteke, ali predloženo rješenje izgleda elegantnije.

```
sealed class Screen(val route: String) {
    object SignIn: Screen( route: "Sign in")

    object SignUp: Screen( route: "Sign up")

    object Map: Screen( route: "Map")

    object LocationsList: Screen( route: "Location list")

    object YourLocations: Screen( route: "Locations")

    object MakeRequest: Screen( route: "Make request")

    object Requests: Screen( route: "Requests")

    object About: Screen( route: "About")

    object AccountDetails: Screen( route: "Account details")

    object PlaceDetails: Screen( route: "Place details")

    object MakePost: Screen( route: "Make post")

    object PostDetails: Screen( route: "Post details")

    object Chat: Screen( route: "Chat")
}
```

Slika 3.6. Prikaz svih odredišta unutar navigacije

Sljedeća važna komponenta unutar navigacije je već spomenuta *NavHost* komponenta. *NavHost* je zapravo prazan spremnik koji služi samo za prikaz navigacijskog grafa aplikacije. Aplikacija je složena tako da je svaki zaslon prikazan pomoću *Composable* komponenti koje sadrže druge manje *Composable* komponente, a ti zasloni su upravo sadržani unutar *NavHost* komponente. Slika 3.7. prikazuje navigacijski graf aplikacije koji sadrži *NavHost* komponentu gdje se može vidjeti kako je složen zaslon za prijavu. Unutar konstruktora zaslona za prijavu vidi se način kako mu se predaju metoda za navigaciju do početnog zaslona i metoda za navigaciju do zaslona za registraciju. Navigacija do početnog zaslona je nešto drugačija jer je potrebno maknuti prethodna odredišta sa stoga (engl. *stack*) kojeg navigacija koristi da bi pamtila kako se je korisnik kretao unutar aplikacije. Ako korisnik pritisne tipku za natrag na početnom zaslonu, neće ga vratiti na zaslon za prijavu već bi trebao izaći iz aplikacije. Kod metode za navigaciju do zaslona za prijavu, brisanje stoga se ne koristi jer ako korisnik pritisne tipku za natrag unutar zaslona za

registraciju treba ga vratiti na zaslon za prijavu. Opisane metode za navigaciju ne koriste nikakve argumente, no i to je moguće postići. Tako jedan zaslon može poslati drugom nekakav podatak kojeg taj drugi zaslon može iskoristiti, a to je jako važno i u aplikaciji se često koristi. Na primjer, tijekom odabira mjesta trenutni zaslon treba poslati identifikacijski broj odabranog mjesta zaslonu za prikaz detalja o mjestu kako bi se znalo koje mjesto treba dohvatiti iz baze podataka i prikazati.

Slika 3.7. Prikaz *NavHost* komponente unutar navigacijskog grafa

Posljednja važna komponenta unutar navigacije je *NavController* objekt kojeg se može vidjeti i na slici 3.7. jer upravo taj objekt služi za upravljanje navigacijom aplikacije unutar *NavHost* komponente. *NavController* radi tako da mijenja odredišni sadržaj unutar *NavHost* komponente kako se korisnik kreće po aplikaciji. Unutar aplikacije *NavController* je kreiran u *MainActivity* klasi, jedinoj klasi aktivnosti koja se koristi unutar ove aplikacije. Kreira se pozivom metode *rememberNavController()* koja će vratiti *NavController* objekt te će biti proslijeđen u komponentu navigacijskog grafa kako bi se mogao dalje koristiti.

3.4. Hilt

Hilt je moćan alat koji pomaže pri ubrizgavanju ovisnosti. No za početak treba shvatiti što je to ovisnost. Kada klasa *A* treba inicijalizirati klasu *B* kako bi klasa *A* postala korisna zove se ovisnost. To povećava usku povezanost klasa i dovodi do dupliciranja koda. Tada se

ubrizgavanjem ovisnosti smanjuje uska povezanost između klasa [11]. Upravo zato se u ovoj aplikaciji koristi Dagger Hilt. Postoji nekoliko pravila kojih se treba pridržavati prilikom njegovog korištenja. Prvo pravilo je da aplikacija mora imati aplikacijsku klasu koja ima anotaciju *HiltAndroidApp* što je prikazano na slici 3.8. Nakon što je to postavljeno, Hilt se može koristiti i može pružati ovisnosti drugim Android klasama koje imaju anotaciju *AndroidEntryPoint*, a u slučaju ove aplikacije to je *MainActivity* klasa što je prikazano na slici 3.9.

```
@HiltAndroidApp
class DesLocationsApp : Application()
```

Slika 3.8. Prikaz aplikacijske klase s *HiltAndroidApp* anotacijom

```
@AndroidEntryPoint
class MainActivity : ComponentActivity() {
```

Slika 3.9. Prikaz *MainActivity* klase s *AndroidEntryPoint* anotacijom

Jedan od lakših primjera za shvatiti funkcionalnost Hilta je kada se koristi *Composable* komponenta zaslona, u ovom slučaju neka to bude *MapScreen*. Ta komponenta treba inicijalizirati *ViewModel* klasu kao što je to prikazano slikom 3.10. i 3.11. Tada će se *ViewModel* klasa označiti anotacijom *HiltViewModel*, a u konstruktoru *MapScreen* komponente će se pozvati Hilt metoda naziva *hiltViewModel()* koja će odraditi posao inicijaliziranja tako da će se smanjiti uska povezanost tih klasa a sve funkcionalnosti ostaju iste.

```
@HiltViewModel
class MapViewModel @Inject constructor(
    private val placesRepository: PlacesRepository
) : ViewModel()  {
```

Slika 3.10. Prikaz *ViewModel* klase s anotacijom *HiltViewModel*

```
@Composable
fun MapScreen(
    viewModel: MapViewModel = hiltViewModel(),
    navigateToPlaceDetailsScreen: (String) -> Unit
    ) {
```

Slika 3.11. Prikaz korištenja *hiltViewModel()* metode unutar *MapScreen* komponente

Drugi primjer, odnosno malo složenije korištenje Hilta je tijekom ubrizgavanja konstruktora kada se tip ne može ubaciti kao što je napravljeno u prvom primjeru. Ne može se napraviti ubrizgavanje, ako je tip sučelje ili se ne posjeduje taj tip ili su klase vanjske biblioteke. Tada se Hiltu pružaju obvezujuće informacije pomoću Hilt modula koji se anotira s *Module* [12]. Na slici 3.10. u konstruktor se ubacuje tip sučelja *PlacesRepository* čija je svrha više opisana u poglavlju 3.8. U ovom slučaju tip je sučelje te se tada trebaju pružiti dodatne informacije pomoću modula što je prikazano slikom 3.12. Ako se bolje promotri slika, uvidjet će se da osim *Module* anotacije ima i *InstallIn* anotaciju jer je potrebno navesti u kojim klasama će se ovaj modul koristiti, a koristit će se u *ViewModel* klasama. Nakon *Provides* anotacije unutar navedenog sučelja navode se dodatne informacije:

- Povratni tip funkcije koji govori Hiltu na koje se sučelje odnosi funkcija i koje instance pruža
- Parametar funkcije govori Hiltu koju implementaciju treba pružiti

U primjeru je prikazan povratni tip *PlacesRepository* s instancom *PlacesRepositoryImpl* klase čiji je parametar *FirebaseFirestore* objekt, a to je vanjska klasa podržana od Firebasea.

```
@Module
@InstallIn(ViewModelComponent::class)
class AppModule {

    @Provides
    fun providePlacesRepository(): PlacesRepository = PlacesRepositoryImpl(
        db = Firebase.firestore
    )
```

Slika 3.12. Prikaz dijela klase modula s anotacijom *Module*

3.5. Composable

Već je nekoliko puta spomenut Jetpack Compose razvojni okvir, no nigdje nije opisano kako je izgrađeno korisničko sučelje. Zato su ovdje detaljnije opisane *Composable* funkcije koje služe pri opisivanju korisničkog sučelja. Funkcije se označavaju anotacijom *Composable* što govori kompajleru da je funkcija namijenjena pretvaranju podataka u korisničko sučelje [13]. Unutar takvih *Composable* funkcija moguće je koristiti definirane *Composable* funkcije kao što su *Text(), Button(), TextField(), Row(), Column()* i još mnoge druge. No osim tih definiranih *Composable* funkcija moguće je koristiti i vlastite napisane *Composable* funkcije, a poželjno je da one bude što manje i da imaju određenu zadaću pri prikazivanju korisničkog sučelja. Na slici 3.13. prikaza je

Composable funkcija za prikaz zaslona o detaljima objave. Navedena *Composable* funkcija ima zadaću dohvatiti podatke o detalju odabrane objave te kada su detalji spremni poziva drugu *Composable* funkciju naziva *PostDetailsContent*. *PostDetailsContent* funkcija će postaviti detalje o objavi u korisničko sučelje. Dok se čeka dohvaćanje podataka o objavi postavlja se kružni indikator napretka kako bi korisnik uočio da se podatci dohvaćaju. Takav način funkcioniranja aplikacije postignut je pomoću Kotlin korutina (eng. *coroutines*) što je objašnjeno u idućem poglavlju 3.6.

```
fun PostDetailsScreen(
    viewModel: PostDetailsViewModel = hiltViewModel(),
    postID: String,
    navigateToPlaceDetailsScreen: (String) -> Unit
) {
    val context : Context = LocalContext.current
    var isInitGetPostDone : Boolean by rememberSaveable {...}
    LaunchedEffect(key1 = isInitGetPostDone) {...}

    when (val getPostResponse : Response<PostResponse> = viewModel.getPostResponse) {
        is Response.Loading -> {
            LaunchedEffect(Unit) {...}
            Scaffold {...}
        }
        is Response.Success -> getPostResponse.apply { this: Response.Success<PostResponse>
            LaunchedEffect( key1: true) {...}
            PostDetailsContent(
                post = data,
                canDelete = { authorID ->
                    viewModel.currentUser?.let { it: FirebaseUser
                        it.uid == authorID || it.uid == viewModel.adminID
                    } ?: false
                },
                deletePost = { post -> viewModel.deletePost(post) },
                isAdminPost = { authorID -> authorID == viewModel.adminID }
            )
        }
```

Slika 3.13. Prikaz *Composable* funkcije za prikaz detalja o objavi

Na slici 3.14. Prikazan je dio koda kako se postavlja dio sadržaja za određenu objavu. Prikazana je *LazyColumn* komponenta za prikaz sadržaja koji je zapravo zamjena za *ReciclerView* komponentu unutar *View* razvojnog okvira, ali ovdje je to puno jednostavnije za implementirati. Koristi se kada treba postaviti veliki broj stavki vertikalno, ali da se samo stvaraju i postavljaju stavke koje korisnik trenutno može vidjeti na zaslonu [14]. Postoje i druge slične komponente kao što je *LazyRow* komponenta koja postavlja stavke horizontalno i nešto složenija *LazyGrid* komponenta, a sve one rade na sličan način recikliranjem stavki i tako poboljšavaju performanse aplikacije.

Slika 3.14. Prikaz postavljanja dijela sadržaja odabrane objave

U tijelu prikazane *LazyColumn* komponente na slici 3.14. vidljiv je dio prve stavke koji prikazuje naslov objave, a još služi za prikaz opisa, autora i datuma objave. Osim prve stavke, LazyColumn komponenta u ovom slučaju prikazuje i listu slika koja se može sastojati od nijedne do nekoliko, pa je to potrebno prikazati pomoću navedene *LazyColumn* komponente, a prikaz slika objave unutar te komponente vidljiv je na slici 3.15. Kako bi *LazyColumn* komponenta ispravno radila potrebno joj je predati listu stavki. Komponenti je predana lista *URI*-a odnosno lista adresa na kojoj su slike spremljene u *Storage* bazi podataka. Prikaz stavke, kako je vidljivo na navedenoj slici obavljeno je pomoću *Box* komponente koja radi sličan posao kao i *Column* i *Row*, to jest da raspoređuje elemente unutar sebe. U ovom primjeru, radi se o jednom elementu, a to je komponenta *AsyncImage* koju je moguće koristiti nakon što se u projekt uključi vanjska biblioteka *coil*. *AsyncImage* je komponenta koja olakšava prikazivanje slika preko mreže tako da joj se preda *URI* na kojemu je spremljena slika.

```
itemsIndexed(items = post.imageUris!!) { this: LazyItemScope  _, uri ->
    Box(
        modifier = Modifier
            .animateItemPlacement()
            .fillMaxWidth()
            .padding(bottom = 5.dp),
        contentAlignment = Alignment.Center
    ) { this: BoxScope
        Card(
            modifier = Modifier.width(280.dp),
            border = BorderStroke(1.dp, MaterialTheme.colorScheme.outline)
        ) { this: ColumnScope
            AsyncImage(
                model = uri,
                contentDescription = stringResource(id = "Post Image"),
                contentScale = ContentScale.FillBounds,
            )
        }
    }
}
```

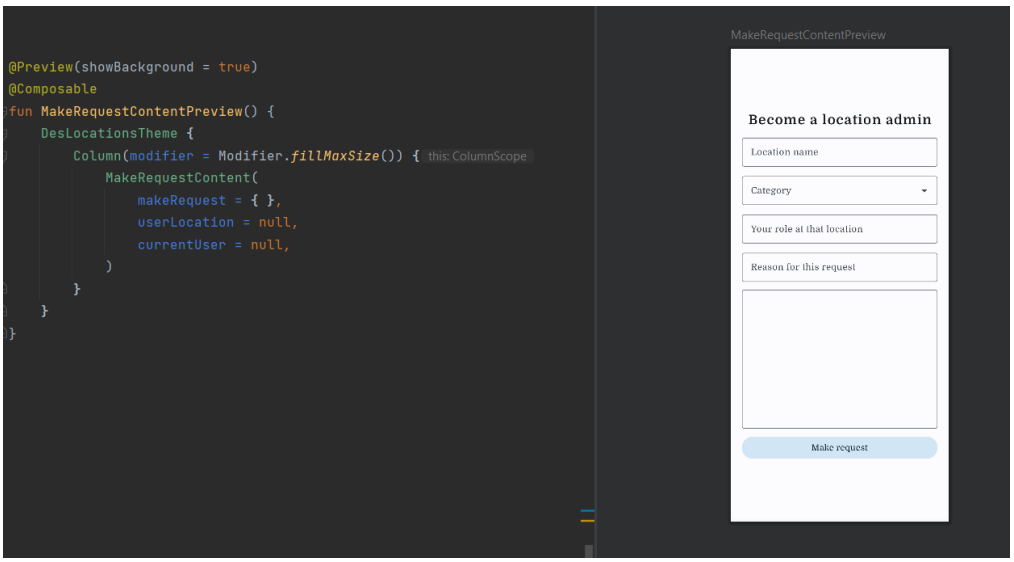
Slika 3.15. Prikaz stavki koje služe za prikaz slika unutar *LazyColumn* komponente

Postoje još brojni primjeri i načini rada s *Composable* funkcijama, jer ima jako puno prostora za programere koji mogu to iskoristiti kako bi njihov kod bio što bolji. Treba istaknuti rad s *Dialog* komponentama. To su komponente koje se prikazuju na sloju iznad glavnog sadržaja aplikacije kako bi obavijestili korisnika i većinom očekuju korisnikovu interakciju. Na slici 3.16. prikazan je način rada s komponentom *Dialog* koja služi za brisanje objave koju može napraviti administrator mjesta ili autor objave. Na prikazanoj slici može se vidjeti varijabla o kojoj ovisi prikaz *Dialog* komponente, ako je ona „*true*“ postavit će se *Dialog* komponenta, a kada je „*false*“ maknut će se, odnosno neće biti prikazana.

```
var isDeletePostDialogShown : Boolean by rememberSaveable {
    mutableStateOf( value: false)
}

if (isDeletePostDialogShown) {
    post?.let { it: PostResponse
        DeletePostDialog(
            onDismiss = { isDeletePostDialogShown = false },
            postTitle = post.title!!,
            deletePost = { deletePost(post) }
        )
    }
}
```

Slika 3.16. Prikaz rada s komponentom *Dialog*

Pregled korisničkog sučelja moguće je napraviti na dva načina. Prvi način je pomoću stvarnog mobilnog uređaja ili pomoću emulatora. Drugi način je pomoću *Preview* anotacije koja označava da se napisani kod treba prikazati u pregledu dizajna. To omogućava ažuriranje korisničkog sučelja uživo kako se napisani kod mijenja [15]. Na slici 3.17. prikazan je jedan primjer *Preview* funkcije koja omogućava brzi i jednostavni prikaz sadržaja. U ovom slučaju to je *Preview* funkcija za prikaz zaslona prilikom kreiranja zahtjeva za novo mjesto. U praznom prostoru iznad gumba za kreiranje zahtjeva sadržana je karta koja služi za označavanje lokacije na kojoj se nalazi to mjesto. *Preview* funkcija nema mogućnost prikazati kartu jer je komponenta za kartu uključena kroz vanjsku biblioteka i to otežava njezin prikaz na ovako novoj tehnologiji, pa je prikazan samo prazan prostor.



Slika 3.17. Prikaz rada s *Preview* funkcijama

3.6. Kotlin korutine

Kotlin korutine su standardno rješenje za asinkrono programiranje, a uključuje neke od značajki kao što su [16]:

- Mogu se pokrenuti mnoge korutine na jednoj niti zahvaljujući podršci za suspenziju, što ne blokira nit na kojoj se korutina izvršava
- Manje curenja memorije
- Ugrađena podrška za otkazivanje
- Jetpack integracija

Uzimajući u obzir navedene značajke može se zaključiti da su korutine odlično rješenje za asinkrono obavljanje posla kao što su slanje i dohvaćanje podataka s mreže, odnosno za slanje i dohvaćanje podataka iz Firebase baze podataka. Za rad s Kotlin korutinama prvo je napravljena *Response* klasa, a u ovom slučaju to je *sealed* klasa što je vidljivo na slici 3.18. *Sealed* klasa je opisana nešto ranije, ali unutar ove *Response sealed* klase postoje tri opcije različitih tipova. Prva opcija je *Loading* objekt koji se koristi kada se podatci dohvaćaju ili kada se podatci šalju na server. Druga opcija je *Success data* klasa koja služi za označavanje uspješnog slanja ili dohvaćanja podataka, a u sebi može sadržavati određene podatke. Zadnja opcija je *Failure data* klasa koja služi za označavanje greške koja se može pojaviti prilikom dohvaćanja ili slanja podataka na server, a u sebi može sadržavati iznimku, odnosno poruku zašto je došlo do određene greške.

```
sealed class Response<out T> {

    object Loading : Response<Nothing>()

    data class Success<out T>(
        val data: T?
    ) : Response<T>()

    data class Failure(
        val e: Exception,
        val errorCode: String = ""
    ) : Response<Nothing>()

}
```

Slika 3.18. Prikaz *Response* klase

Korištenje *Response* klase vidljivo je na slici 3.19. gdje se unutar LocationsList*ViewModel* klase koristi funkcija za dohvaćanje mjesta u krugu korisnika. Navedena funkcija radi unutar *CoroutineScope*a pozivom funkcije *launch* koja kreira korutine tako da se unutar tijela te funkcije posao dohvaćanja lokacija neće blokirati. Na početku se varijabla *getPlacesResponse* postavlja na vrijednost *Loading* kako bi se označilo da se posao dohvaćanja lokacija trenutno izvodi. Zatim se poziva metoda unutar *placesRepository* objekta koja će obaviti posao dohvaćanja mjesta. Detaljnija funkcionalnost *ViewModel* i *Repository* klasa prikazana je u poglavljima 3.7. i 3.8.

```
@HiltViewModel
class LocationsListViewModel @Inject constructor(
    private val placesRepository: PlacesRepository
) : ViewModel() {

    var getPlacesResponse : Response<List<Place>>
            by mutableStateOf<Response<List<Place>>>(Response.Success( data: null))
        private set

    fun getPlaces(location: Location?) = viewModelScope.launch { this: CoroutineScope
        getPlacesResponse = Response.Loading
        location?.let { getPlacesResponse = placesRepository.getPlaces(location) }

    }
}
```

Slika 3.19. Prikaz *ViewModel* klase koja koristi *Response* klasu

Funkcije za dohvaćanje lokacija unutar *Repository* klase prikazane su na slici 3.20. koja na prvu izgleda kompliciranо, no ono što je važno u okviru korutina je nekoliko detalja. Prvo, da je funkcija označena sa *suspend*. Zatim povratni tip funkcije je *Response<List<Places>>*, to jest povratni tip je već opisani objekt *Response* koji može sadržavati podatke s listom mjesta. I na kraju, funkcija za dohvaćanje, odnosno ona koju se želi postaviti kao ne blokirajuća označena je s *await()*. Kada su mjesta uspješno dohvaćena funkcija vraća opciju *Response* kao *Success* s listom tih mjesta, a u slučaju iznimke vraća se opcija *Failure* s iznimkom koja je prouzročila grešku.

```
override suspend fun getPlaces(location: Location): Response<List<Place>> {
    return try {
        val places = ArrayList<Place>()
        db.collection( collectionPath: "Places")
            .get().await().map { it: QueryDocumentSnapshot!
                val place : Place = it.toObject(Place::class.java)
                place.id = it.id
                val results = FloatArray( size: 1)
                Location.distanceBetween(
                    location.latitude,
                    location.longitude,
                    place.latitude!!.toDouble(),
                    place.longitude!!.toDouble(),
                    results
                )
                place.distance = "%.3f".format(Locale.ENGLISH, ...args: results[0] / 1000).toFloat()
                places.add(place) ^map
            }
        places.sortBy { it.distance }
        Response.Success(places)
    } catch (e: Exception) {
        Response.Failure(e)
    }
}
```

Slika 3.20. Prikaz funkcije za dohvaćanje lokacija unutar *Repository* klase

I za kraj rada s korutinama preostaje još upotrijebiti *Response* klasu unutar *Composable* funkcije, odnosno na osnovu trenutnog stanja varijable *getPlacesResponse* prikazati određeni sadržaj. Kada je ta varijabla jednaka objektu *Loading* tada se prikazuje kružni indikator napretka,

a kada je ta varijabla *Success* prikazat će se podatci o lokacijama na korisničko sučelje i u slučaju kada je ta varijabla *Failure* prikazat će grešku na korisničko sučelje. Takav način funkcioniranja već je prikazan na slici 3.13. samo u drugačijoj ulozi, prilikom dohvaćanja objave.

3.7. ViewModel

Već se je nekoliko puta u ovom tekstu pojavio *ViewModel* no nigdje nije opisan njegov rad i svrha u ovoj aplikaciji. Zato je ovo poglavlje posvećeno upravo *ViewModel* klasama i njihovoj svrsi. *ViewModel* je jedan od dijela *MVVM* obrasca dizajna softvera koji služi kao poveznica između *View* dijela i *Model* dijela. *View* dio je već opisani dio koji se sastoji od *Composable* funkcija koje služe za prikaz korisničkog sučelja, a *Model* dio se odnosi na *Repository* klase koje su opisane u idućem poglavlju.

ViewModel je zapravo AndroidX API koji olakšava programerima očuvati podatke tijekom promjena konfiguracije, a u isto vrijeme služi kao mjesto gdje se može čuvati poslovna logika. Tako na primjer, kada korisnik napravi interakciju s korisničkim sučeljem, odnosno pritisne gumb u aplikaciji, ta namjera se može proslijediti *ViewModel* klasi kako bi obavila posao i emitirala novo stanje [17].

```
@HiltViewModel
class MakeRequestViewModel @Inject constructor(
    private val authRepo: AuthRepository,
    private val placesRepo: PlacesRepository
) : ViewModel() {

    var userLocation : Location? by mutableStateOf<Location?>( value: null)

    var makeRequestResponse : Response<Boolean>
            by mutableStateOf<Response<Boolean>>(Response.Success( data: false))
        private set

    val currentUser: FirebaseUser?
        get() = authRepo.currentUser

    fun makeRequest(placeRequest: PlaceRequest) = viewModelScope.launch { this: CoroutineScope
        makeRequestResponse = Response.Loading
        makeRequestResponse = placesRepo.makeRequest(placeRequest)
    }

}
```

Slika 3.21. Prikaz *ViewModel* klase

Za potrebe aplikacije svaka *Composable* funkcija za prikaz zaslona ima svoju *ViewModel* klasu, no nema potrebe da ih se sve prikazuje jer rade na sličan način. Tako je na slici 3.21.

prikazana MakeRequest*ViewModel* klasa koja služi za kreiranje zahtjeva za novo mjesto. U njoj se spremaju svojstva *userLocation* i *makeRequestResponse* kako bi se njihovo stanje očuvalo tijekom promjene konfiguracije kada je korisniku prikazano sučelje za kreiranje zahtjeva za novo mjesto. Ovo nije jedini način kako bi se podatci očuvali tijekom promjene konfiguracije, jer se unutar *Composable* funkcija koristi *rememberSaveable* funkcija koja će automatski spremati stanja korisničkog sučelja. Druga opisana svrha *ViewModel* klase je očuvanje poslovne logike i to je prikazano na slici 3.21. u dvjema funkcijama koje komuniciraju s *Repository* klasama, jedna služi za autentikaciju korisnika, a druga za rad s lokacijama.

3.8. Repository

Repository je klasa čija je svrha pružiti API za pristup podatcima iz različitih izvora kao što su: REST (engl. *Representational State Transfer*) API, lokalna baza podataka ili udaljena baza podataka. Nakon što se pristupi podatcima, ta klasa prosljeđuje podatke ostatku aplikacije, a ostale komponente koje koriste te podatke ne znaju odakle su ti podatci stigli [18]. U slučaju ove aplikacije može se reći kako je *Repository* posrednik između *ViewModel* klase i izvora podataka, odnosno Firebase baze podataka.

```
interface AuthRepository {

    val currentUser: FirebaseUser?
    suspend fun signIn(user: User): Response<Boolean>
    suspend fun signUp(user: User): Response<Boolean>
    fun signOut()
    fun getAuthState(viewModelScope: CoroutineScope): StateFlow<Boolean>
    suspend fun isModerator(): Boolean
    suspend fun getUserDetails(): Response<UserResponse>
    suspend fun sendPasswordResetEmail(email: String): Response<Boolean>
    suspend fun changeAccountDetails(user: User): Response<Boolean>

}
```

Slika 3.22. Prikaz sučelja *Repository* klase za autentifikaciju

Svaki *Repository* ima sučelje i implementaciju, a u aplikaciji se koristi nekoliko *Repository* klasa, svaka za određeni posao i podatke. Tako je na slici 3.22. prikazano sučelje *Repository* klase za autentifikaciju gdje su navedene metode za: prijavu korisnika, registraciju korisnika, odjavu korisnika, metoda za dohvaćanje trenutnog prijavljenog korisnika, metoda za dohvaćanje je li trenutno prijavljeni korisnik moderator aplikacije, metoda za dohvaćanje podataka o korisniku,

metoda za slanje elektroničke pošte za promjenu zaporke i metoda za promjenu korisničkih podataka. U aplikaciji su još sadržane *Repository* klase za rad s lokacijama, objavama korisnika, za dohvaćanje i spremanje slika od objava i za rad s razgovorima unutar pojedine lokacije.

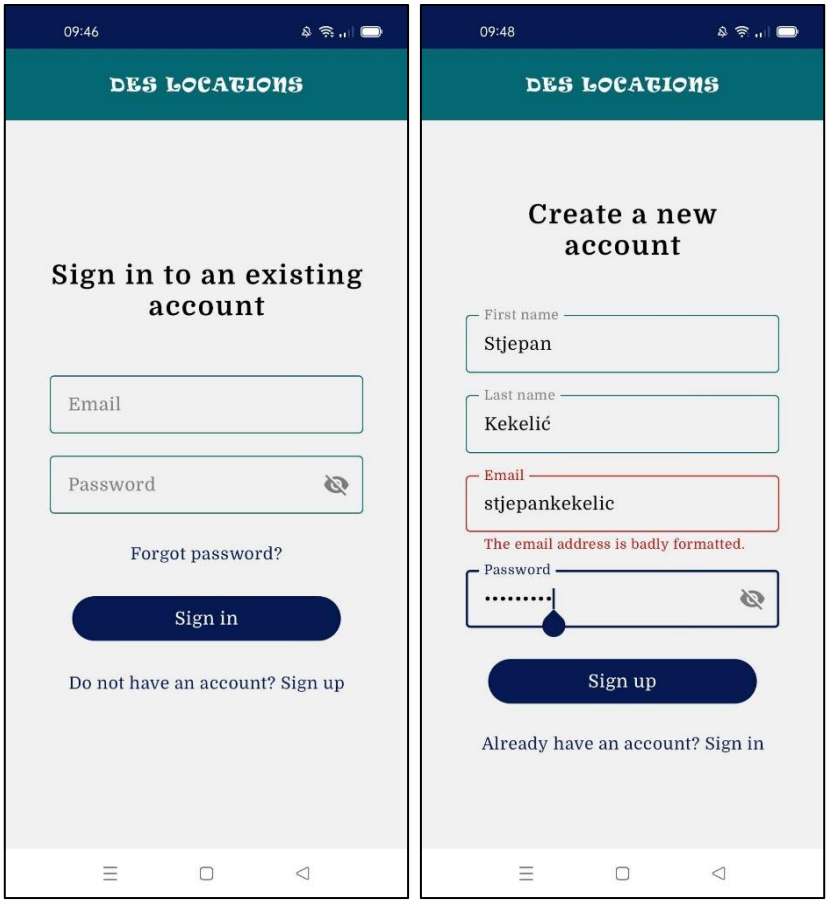
```
@Singleton
class AuthRepositoryImpl @Inject constructor(
    private val auth: FirebaseAuth,
    private val db: FirebaseFirestore
) : AuthRepository {

    override val currentUser: FirebaseUser?
        get() = auth.currentUser

    override suspend fun signIn(user: User): Response<Boolean> {
        return try {
            auth.signInWithEmailAndPassword(user.email, user.password).await()
            Response.Success( data: true)
        } catch (e: FirebaseAuthException) {
            Response.Failure(e, e.errorCode)
        } catch (e: Exception) {
            Response.Failure(e)
        }
    }
```

Slika 3.23. Prikaz implementacije Repository klase za autentifikaciju

Svako *Repository* sučelje ima svoju implementaciju, tako je na slici 3.23. prikazan dio implementacije *Repository* klase za autentikaciju gdje se može vidjeti metoda za prijavu korisnika. *Repository* komunicira s *Authentication* uslugom kako bi prijavio korisnika u aplikaciju. Za slučaj registracije korisnika *Repository* komunicira i s *Authentication* uslugom ali i s *Firestore* uslugom kako bi spremio podatke o imenu i prezimenu registriranog korisnika. Ovo je jedan od jednostavnijih primjera kako *Repository* komunicira s udaljenim uslugama skrivajući tako informacije odakle podatci sve dolaze, a *ViewModel* klasama predaju tražene podatke kako bi ih *ViewModel* dao na korištenje *Composable* funkcijama.

4. STRUKTURA APLIKACIJE

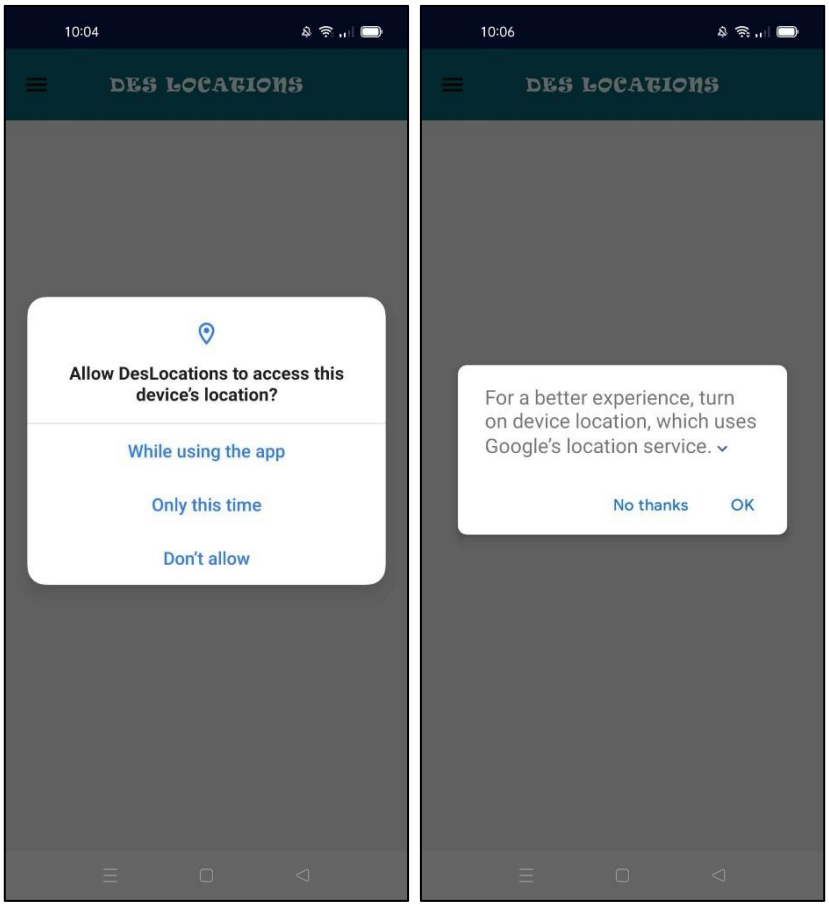
U ovom poglavlju opisana je struktura aplikacije, odnosno način rada korisničkog sučelja i njegov izgled. Opisan je rad pojedinog zaslona aplikacije, a animacije rada aplikacije nisu prikazane jer bi bilo potrebno postaviti puno više slika, a i tada bi bilo teško doživjeti potpuni izgled animacije. Zasloni su prikazivani redom, od početnog zaslona za prijavu i registraciju, pa sve do zaslona koji služi za prikaz informacija o aplikaciji.



Slika 4.1. Zaslona za prijavu i registraciju korisnika

Nakon instaliranja aplikacije, kada korisnik prvi puta ulazi u aplikaciju prikazat će mu se zaslon za prijavu koji je vidljiv na slici 4.1. kao i zaslon za registraciju novog korisnika. Korisnik se prijavljuje pomoću adrese elektroničke pošte i zaporke. Kada korisnik nema napravljen račun potrebno je pritisnuti na tekst ispod gumba za prijavu koji će ga odvesti do zaslona za registraciju. Prilikom registracije korisnik navodi još dodatne podatke o imenu i prezimenu. Kod upisa zaporke, zaporka će biti skrivena pa će biti prikazane zvjezdice umjesto znakova, no korisnik ju može vidjeti pritiskom na prekriženo oko. Ako korisnik unese neispravne podatke prikazat će se prikladna poruka, kao što je to vidljivo na zaslonu za registraciju novog korisnika, gdje je korisnik unio

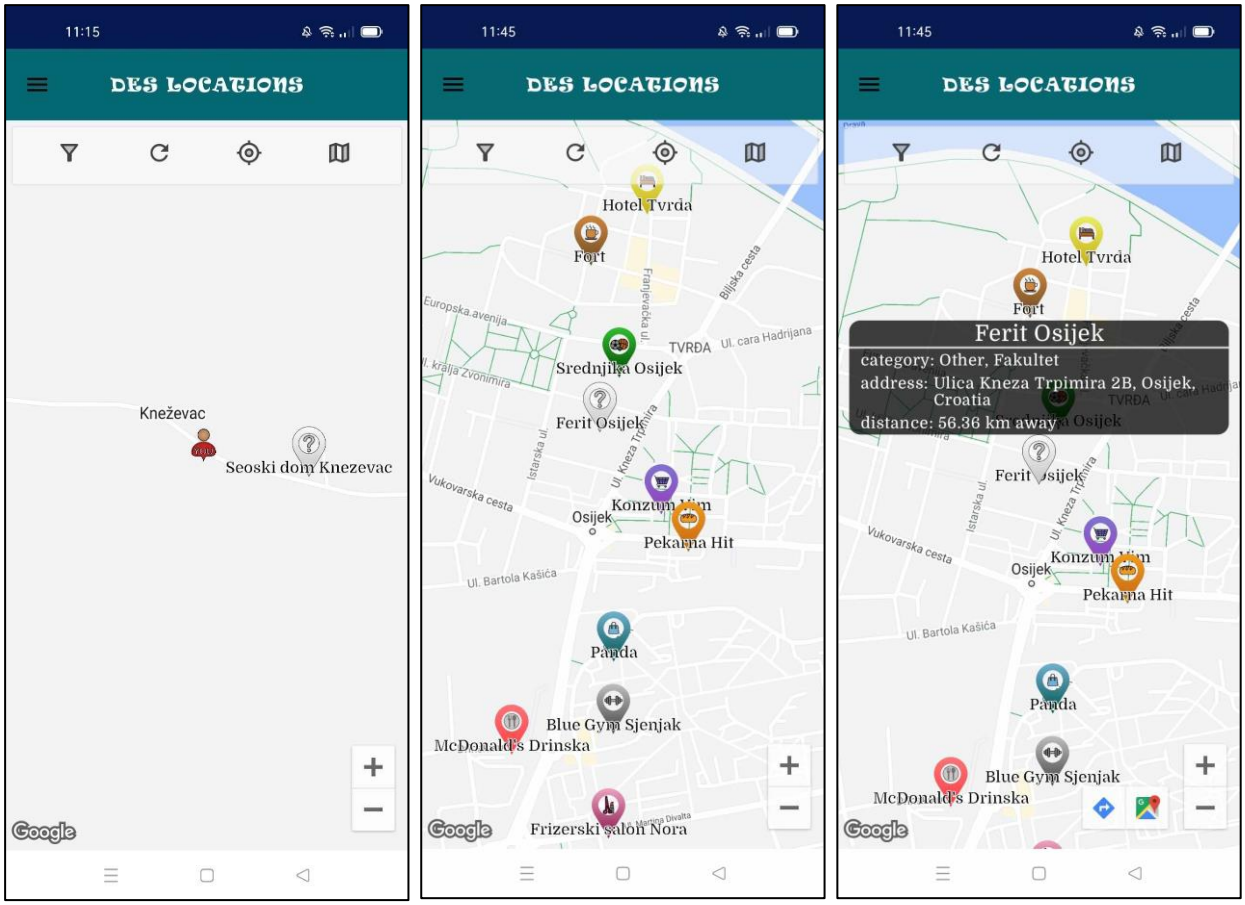
neispravnu adresu elektroničke pošte. Ako se korisnik uspješno prijavi ili registrira prikazat će se zaslon s prikazom lokacija na karti.



Slika 4.2. Prikaz dijaloškog okvira za dopuštenje korištenja i za uključivanje lokacijskih usluga na uređaju

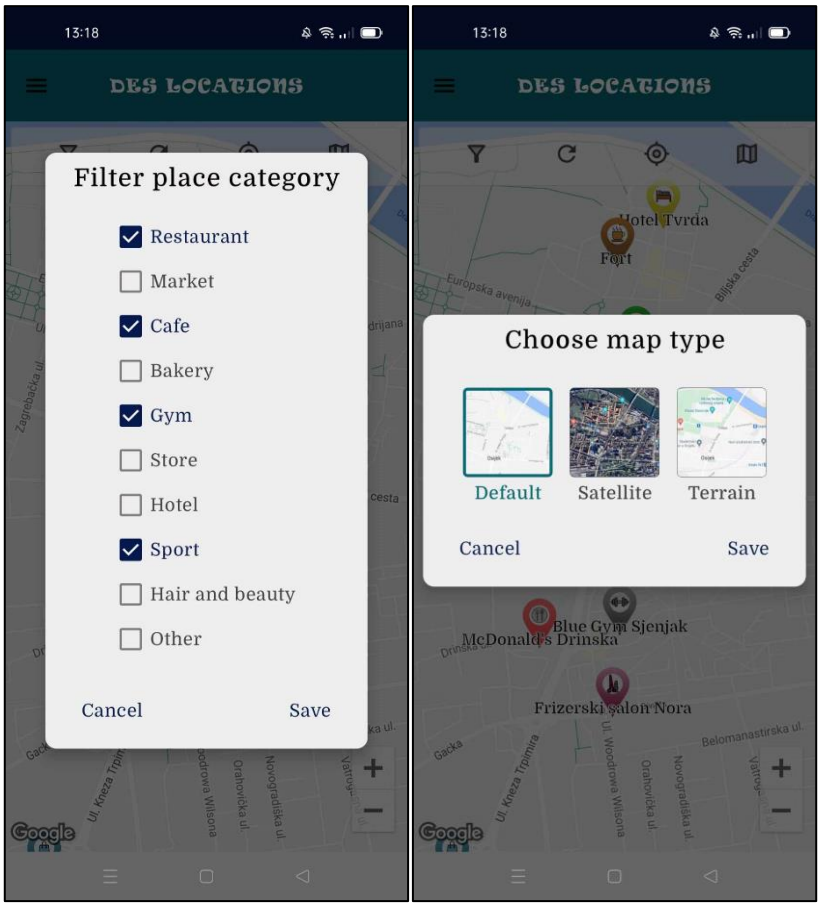
Prije nego što se prikaže karta s već zadanim lokacijama, aplikacija od korisnika traži dopuštenje za korištenje lokacijske usluge što je prikazano na slici 4.2. lijevo. Kada korisnik dopusti korištenje lokacijske usluge samo ovaj put, svaki idući put kada ulazi u aplikaciju prikazivat će se isti dijaloški okvir. Dopusti li korištenje lokacijske usluge uvijek, tada korisnik više neće vidjeti taj dijaloški okvir. Kada na uređaju nisu uključene usluge za lokaciju prikazat će se dijaloški okvir na slici 4.2. desno koji upozorava korisnika da su lokacijske usluge na uređaju isključene te da ih je potrebno uključiti kako bi ih aplikacija mogla koristiti. Korisnik ima mogućnost odustajanja i pritiska gumba „*No thanks*“ i tada lokacija uređaja neće biti dostupna korisniku, a ako pritisne gumb „*OK*“ lokacijska usluga na uređaju će se automatski uključiti. Kada korisnik već ima uključene lokacijske usluge tada taj dijaloški okvir neće biti prikazan. U slučaju da korisnik odbije korištenje ili uključivanje lokacijskih usluga tada će se prikazati tekst s objašnjenjem da su lokacijske usluge potrebne za korištenje aplikacije. Korisnik će moći pritiskom

na gumb dati dopuštenje lokaciji ili uključiti usluge lokacije kako bi nastavio s korištenjem aplikacije. Nakon uspješnog odobravanja i uključivanja lokacijskih usluga prikazuje se karta s lokacijama.



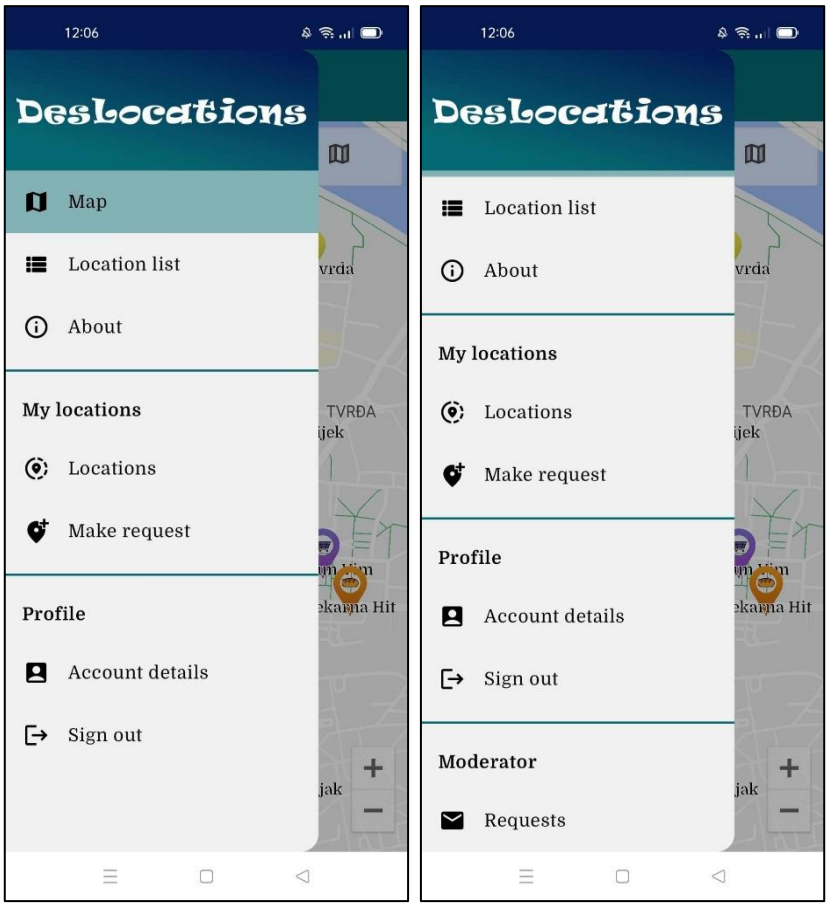
Slika 4.3. Zaslona s lokacijama na karti

Kada se prikaže karta izvrši se animacija tako da se približi na lokaciju korisnika. Korisnička lokacija je tada na sredini i prikazana je markerom čovjeka što je vidljivo na slici 4.3. lijevo. Na slici su također prikazani markeri s lokacijama koje su korisnici aplikacije napravili. Svako mjesto ima svoju kategoriju tako da je ovisno o kategoriji mjesta prikazan drugačiji marker. Svaka kategorija mjesta ima odgovarajući marker s odgovarajućom bojom i ikonicom u sredini markera. Osim kategorije lokacije, na karti se uz marker prikazuje i tekst s nazivom mjesta. Kada korisnik pritisne na marker, prikazat će se dodatni prozor iznad njega s informacijama o nazivu lokacije, kategoriji lokacije, adresi te lokacije i s udaljenosti korisnika do te lokacije što je prikazano slikom 4.3. desno. Kada korisnik pritisne na otvoreni prozor iznad markera, odvest će ga na zaslon koji prikazuje detalje te lokacije.



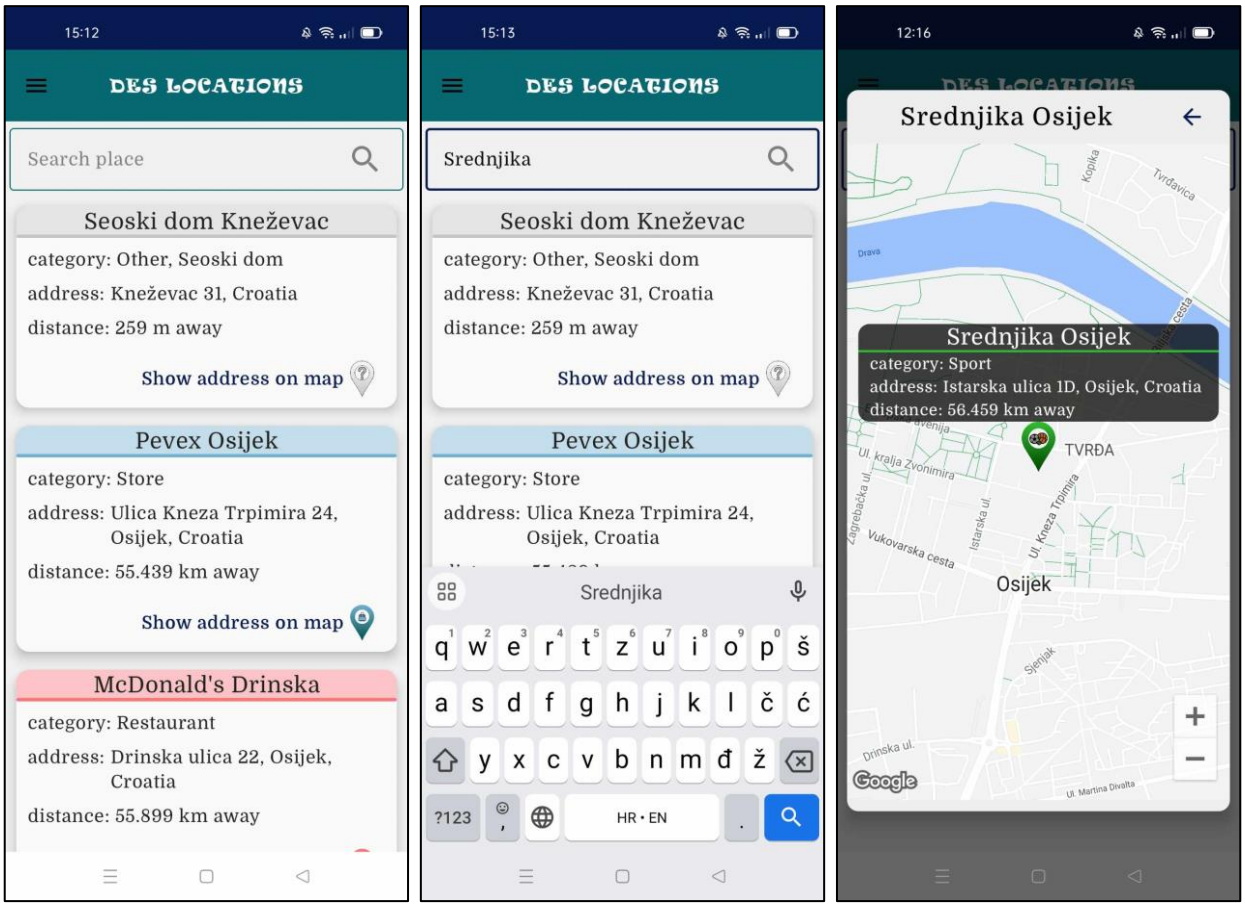
Slika 4.4. Prikaz opcija unutar zaslona s lokacijama na karti

Na vrhu zaslona za prikaz lokacija na karti vidljive su četiri ikonice, odnosno četiri gumba koja služe kako bi olakšali pretraživanje lokacija na karti. Pritiskom na ikonicu koja predstavlja filter otvori će se dijaloški okvir kao na slici 4.4. lijevo gdje korisnik ima mogućnost odabrati koje kategorije lokacija želi vidjeti na karti. Druga ikonica u nizu predstavlja osvježavanje karte. Kada korisnik pritisne na nju karta će se iznova prikazati. Treća ikonica služi kako bi se napravila animacija i pomakla karta gdje će u središtu biti lokacija korisnika, a lokacija korisnika je prikazana posebnim markerom na karti. Zadnja ikonica u nizu služi kako bi korisnik mogao odabrati drugačiji tip karte i pritiskom na nju otvorit će se dijaloški okvir kao na slici 4.4. desno. Korisnik može odabrati jednu od tri tipa karti, a to su normalna, odnosno već zadana karta, karta satelita i karta s prikazom terena. Pritiskom na tipku „*Save*“ karta će promijeniti svoj tip. Korisnik ima i drugu opciju za prikaz lokacija u krugu korisnika, a to je zaslon za prikaz s listom lokacija. Da bi korisnik došao do tog zaslona prvo je potrebno otvoriti navigacijsku ladicu pritiskom na gumb u gornjoj aplikacijskoj traci nakon čega se otvara navigacijska ladica.



Slika 4.5. Navigacijska ladica

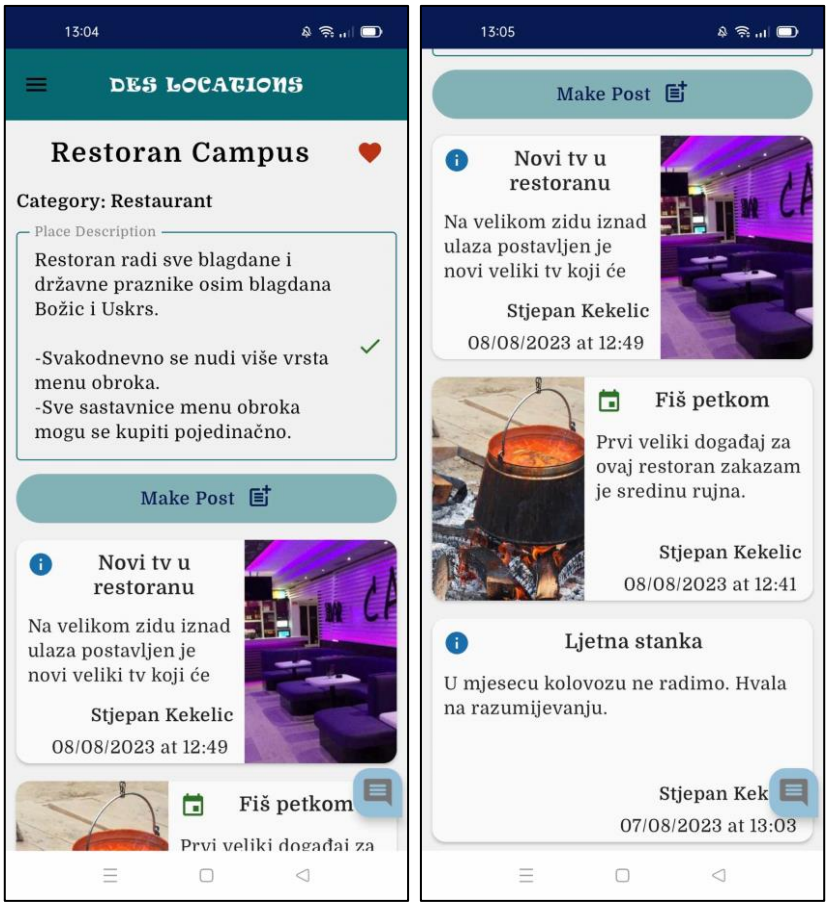
Kada se navigacijska ladica otvori korisnik može odabrati bilo koju opciju unutar nje što će ga odvesti na taj zaslon. To je vrlo jednostavan način kako napraviti navigaciju s puno zaslona, a da korisniku bude jednostavno i dostupno. Slika 4.5. lijevo prikazuje navigacijsku ladicu kada prijavljeni korisnik nije moderator, dok desna slika prikazuje navigacijsku ladicu kada je prijavljeni korisnik moderator. U desnoj slici kada se pomakne lista s opcijama prema gore, na dnu će se prikazati opcija za moderatore što omogućuje pristup zaslonu koji prikazuje zahtjeve za nova mjesta. Trenutni zaslon na slici 4.5. je karta s prikazom lokacija i zato je ta opcija u navigacijskoj ladici drugačije označena. Navigacijska ladica osim opcija za prikaz drugih zaslona ima i opciju za odjavu korisnika. Pritiskom na opciju odjave, korisnik će biti odjavljen i prikazat će se zaslon za prijavu, tako da kada drugi put bude ulazio u aplikaciju morat će se prijaviti.



Slika 4.6. Zaslon za prikaz liste lokacija i dijaloški okvir za prikaz lokacije na karti

Ako je korisnik u navigacijskoj ladici pritisnuo opciju za prikaz liste lokacija, bit će prikazan zaslon kao na slici 4.6. lijevo. Korisniku će biti prikazana mjesta u njegovom krugu od najbližih do najudaljenijih. Svaka stavka unutar liste prikazuje jedno mjesto gdje su prikazane informacije o imenu mjesta, kategoriji, adresi i udaljenosti do tog mjesta. Prikazan je i dodatan gumb koji služi kako bi se adresa tog mjesta prikazala na karti, odnosno da korisnik točno vidi gdje se nalazi ta lokacije. Nakon što korisnik pritisne navedeni gumb, prikazat će se karta u dijaloškom okviru kao na slici 4.6. desno, a pritiskom gumba za natrag korisnik se vraća na poziciju u listi lokacija gdje je i bio. Dodatno, korisnik ima mogućnost pretraživanja mjesta kao što se vidi na slici 4.6. u sredini. Upisivanjem teksta u polje za pretraživanje korisnik može pokrenuti pretraživanje pomoću tipke na tipkovnici ili pomoću gumna unutar polja za pretraživanje. Pretraživanje se izvodi pomoću naziva i kategorije lokacije tako da korisnik može unijet ili naziv lokacije ili kategoriju lokacije. U slučaju ne pronalaska niti jednog mjesta korisniku će biti vidljiv tekst da niti jedno mjesto nije pronađeno. U svakom trenutku korisnik može osvježiti listu lokacija pomicanjem liste prema dolje

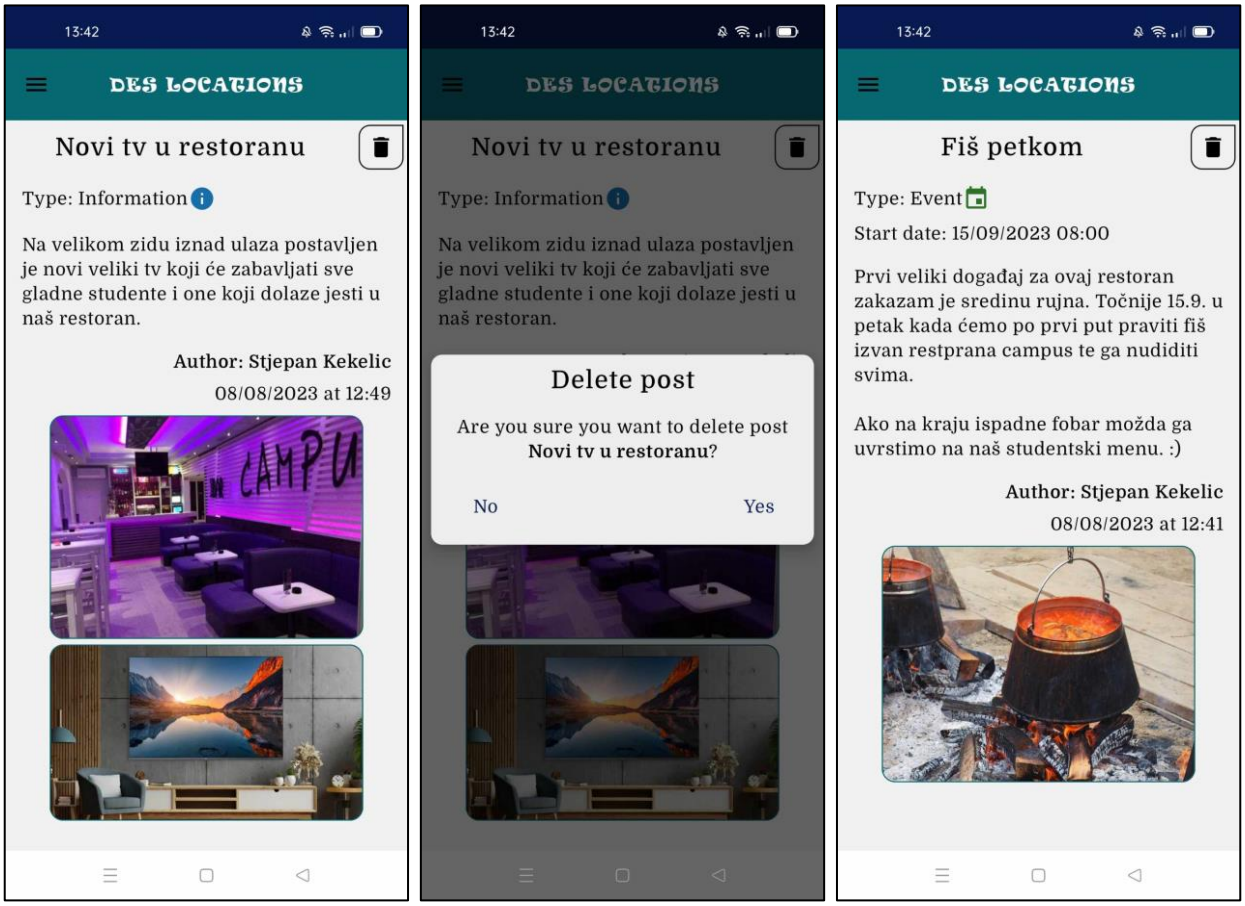
gdje će se pojaviti kružni indikator. Cijele stavke u listi imaju mogućnost pritiska, tako da ako korisnik pritisne na njih otvorit će se novi zaslon za prikaz detalja o tom mjestu.



Slika 4.7. Zaslon za prikaz detalja o mjestu

Kada je korisnik odabrao neko mjesto tada će se prikazati zaslon kao na slici 4.7. lijevo gdje se prikazuju detalji tog mjesta. Na vrhu zaslona je prikazan naziv mjesta a pored toga je prikazana ikona srca, odnosno gumb koji služi kako bi korisnik označio da mu se mjesto sviđa i tada će moći primati obavijesti o novim objavama tog mjesta. Trenutno prikazano mjesto se sviđa korisniku i zato je prikazano crveno srce. U suprotnom, prikazano bi bilo samo srce s tamno plavim obrubom. Ispod naziva mjesta je prikazana kategorija i jedan okvir za uređivanje. Trenutno prijavljeni korisnik je administrator tog mjesta i on jedini ima mogućnost uređivanja opisa tog mjesta pa je iz tog razloga njemu prikazan okvir za uređivanje, a svima ostalima bit će prikazan običan tekst. Nakon što korisnik izmjeni tekst bit će mu prikazana žuta kvačica koja označava da tekst nije spremljen, te pritiskom na nju tekst se sprema i kvačica postaje zelena. Svaki korisnik ima mogućnost kreiranja objave na tom mjestu. Ako administrator lokacije kreira objavu, njegovo ime bit će prikazano podebljano kao što je vidljivo na slici 4.7. Svaka objava može, a i ne mora imati

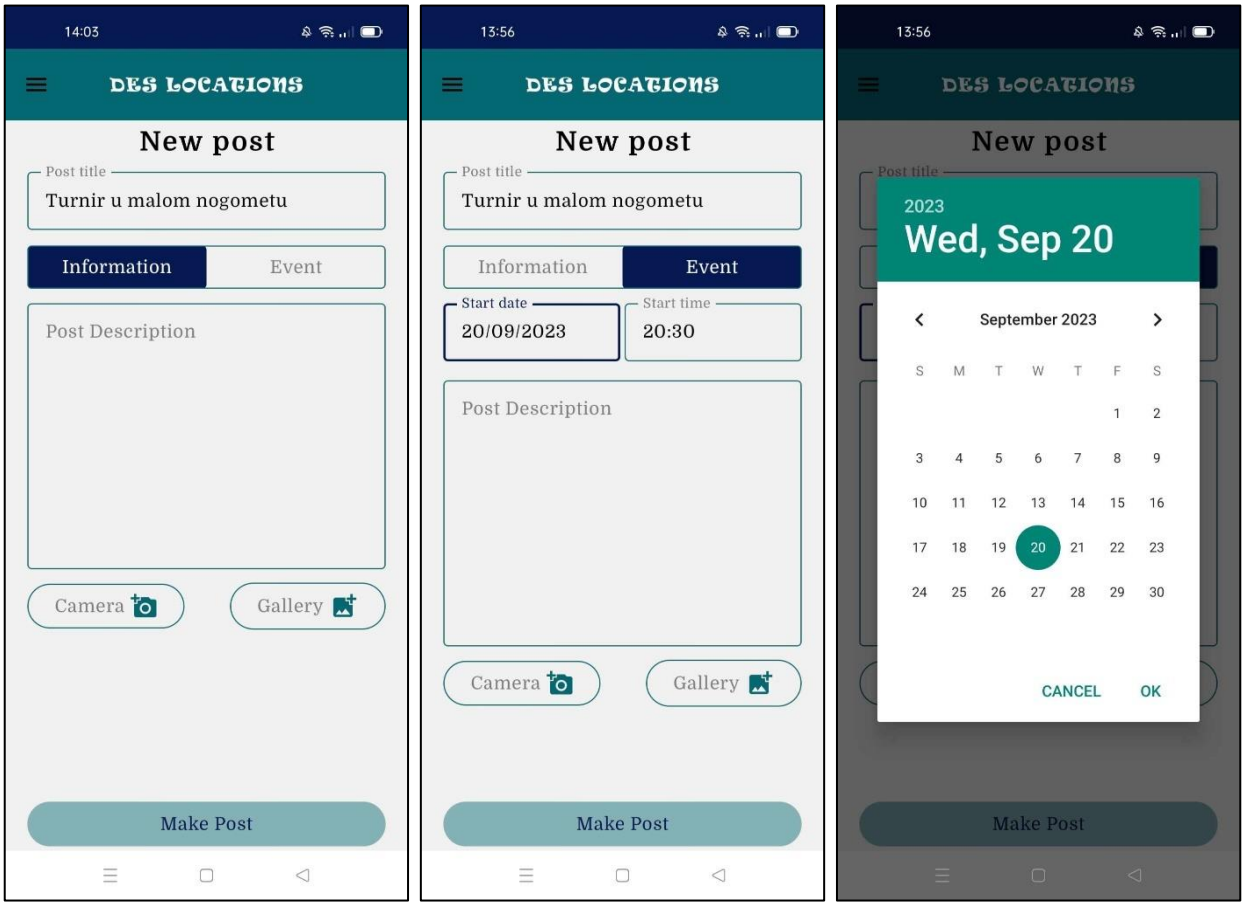
slike. Ako je autor objave postavio slike, prva slika u toj objavi bit će prikazana u listi objava s naslovom objave i skraćenim opisom. Slika može biti postavljena lijevo i desno kao što se vidi na slici 4.7. desno, a to se radi slučajnim odabirom kako bi pretraživanje objava bilo zanimljivo. Svaka objava može biti informativnog karaktera ili može biti događaj. Ako je objava nekakva informacija, tada će biti prikazana plava ikona, a ako je objava događaj bit će prikazana zelena ikona. Objave se razlikuju osim po vizualnim ikonama i po tome što se kod kreiranja događaja mora unijeti datum i vrijeme početka događaja. Posljednja opcija unutar detalja o mjestu je gumb za razgovor nakon čijeg pritiska će se otvoriti zaslon s prikazom razgovora s drugim korisnicima tog mjesta.



Slika 4.8. Zaslon za prikaz objave i dijaloški okvir za brisanje objave

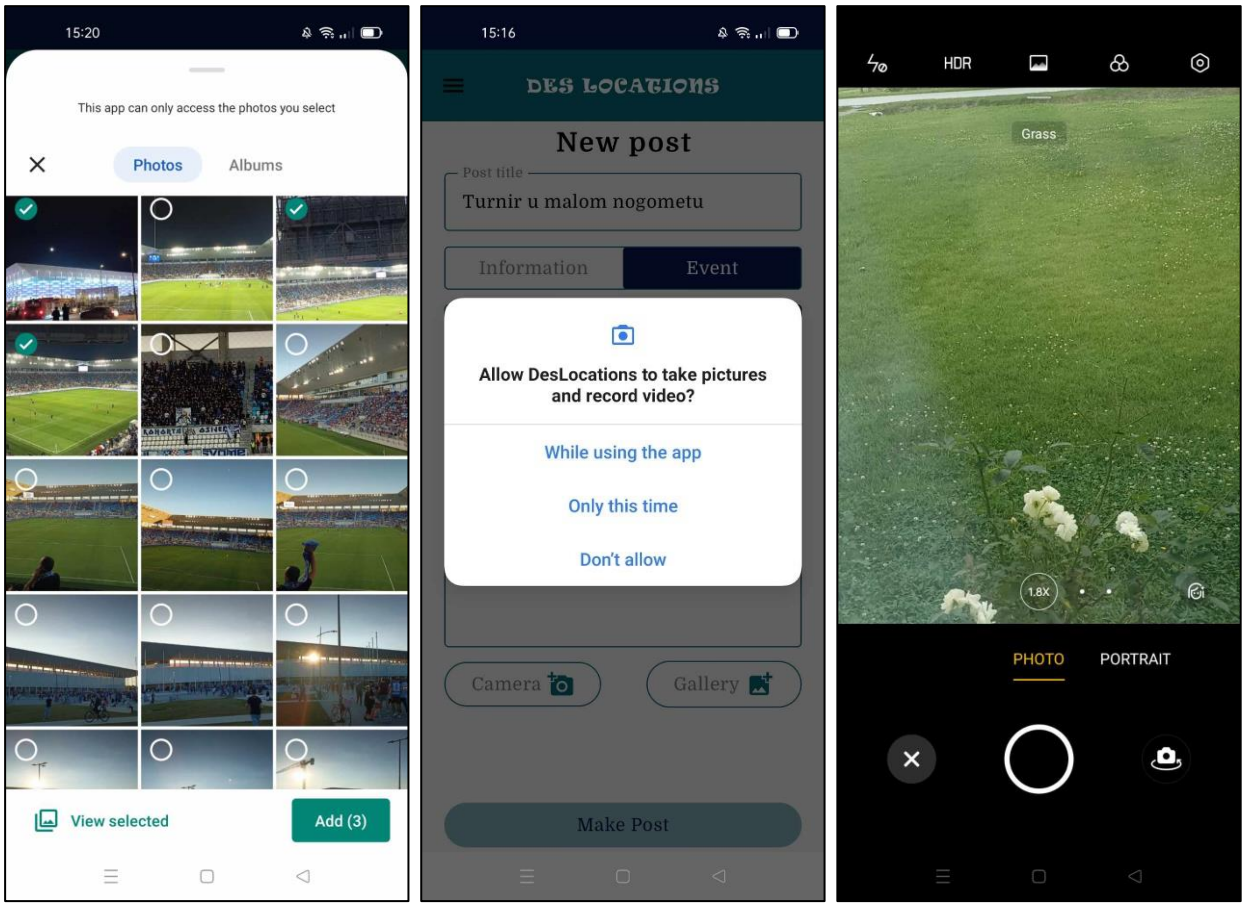
Kada korisnik pritisne na pojedinu objavu unutar liste objava prikazat će se zaslon s detaljima te objave kao što je prikazano slikom 4.8. Prikazani su detalji kao što je naziv objave, vrsta objave, cijeli tekst objave, autor objave, datum objave i lista slika tako da je cijeli zaslon moguće listati do posljednje slike. Na slici 4.8. prikazane su dvije vrste objave, informacija i događaj. Trenutno prijavljeni korisnik ima mogućnost brisanja objave, a obrisati neku objavu može administrator

lokacije i autor te objave. Desno na vrhu zaslona, pored naslova objave prikazan gumb za brisanje. Pritiskom na njega otvara se dijaloški okvir kao na slici 4.8. u sredini gdje se pritiskom na „*Yes*“ briše objava. U ovoj situaciji, dijaloški okvir se prikazuje iz razloga kako korisnik ne bi slučajno pritisnuo gumb za brisanje što bi dovelo do neželjene radnje, pa iz tog razloga treba potvrditi brisanje.



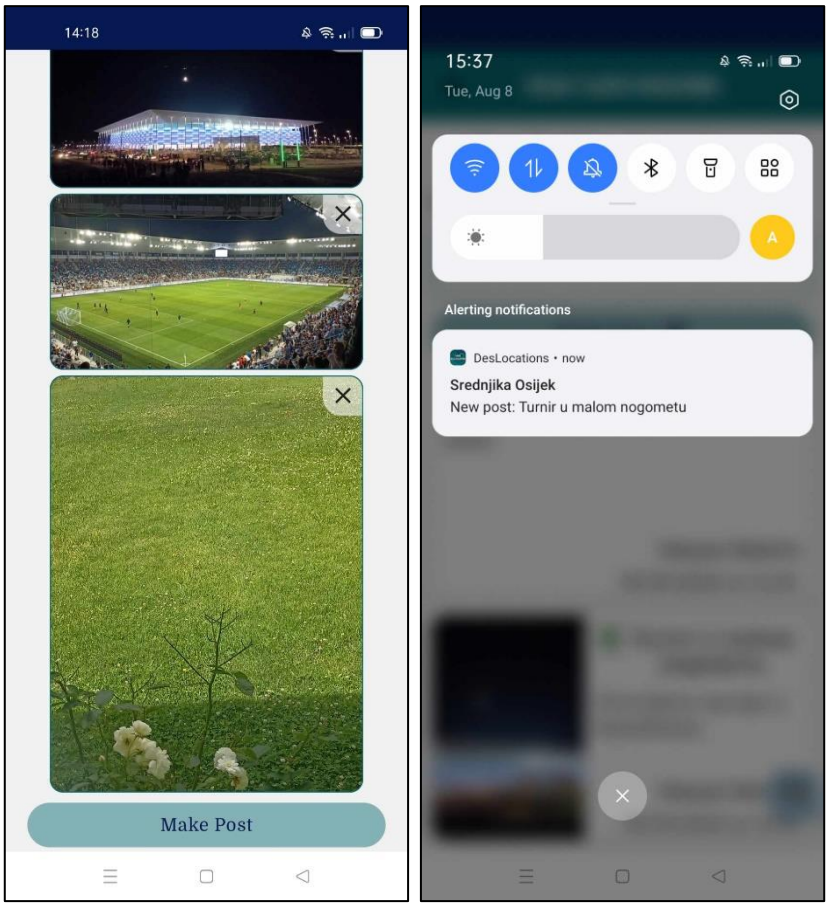
Slika 4.9. Zaslon za kreiranje nove objave i prikaz odabira datuma

Kada korisnik odluči napraviti objavu unutar nekog mjesta tada mu se prikazuje zaslon kao na slici 4.9. gdje je potrebno unijeti naslov, vrstu objave i tekst objave. Postavljanje slika nije obavezno jer nije nužno da neki korisnik koji kreira objavu želi s tom objavom prikazati slike, ali naravno da je uvijek ljepše postaviti i slike jer slika govori puno više od teksta u većini slučajeva. Ako se korisnik odluči umjesto informacije napraviti događaj, tada kao na slici 4.9. u sredini, mora unijeti podatke o datumu i vremenu početka događaja. Unos datuma i vremena početka događaja obavlja se pomoću dijaloškog okvira kao na slici 4.9. desno. Za dodavanje slika unutar objave korisnik ima mogućnost dodati sliku pomoću kamere ili iz galerije.



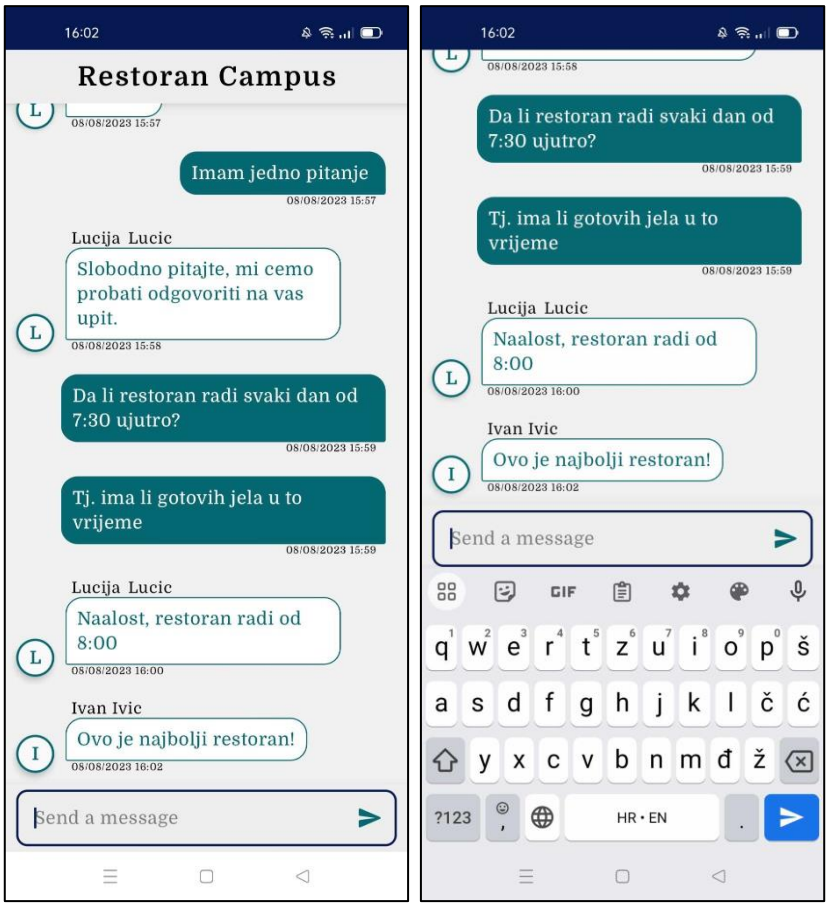
Slika 4.10. Prikaz korištenja galerije i kamere te prikaz dijaloškog okvira za dopuštanje korištenja kamere

Pritiskom na desni gumb koji simbolizira galeriju (vidljivo na slici 4.9.) otvara se galerija koja je prikazana na slici 4.10. lijevo. Korisnik može odabrati nekoliko slika i pritisnuti gumb za dodavanje te će se slike dodati ispod gumba za kameru i galeriju u listu koja se može pomicati. Korisnik može nekoliko puta otvarati galeriju ili kameru kako bi dodao slike te će se one uvijek postaviti na vrh. Na istom zaslonu kada korisnik umjesto pritiska na gumb za otvaranje galerije, pritisne gumb za otvaranje kamere, prvo će biti prikazan dijaloški okvir u kojemu mora dopustiti aplikaciji da se koristi kamerom što je prikazano slikom 4.10. u sredini. Ako korisnik ne dopusti, neće moći koristiti kameru. Ako korisnik dopusti korištenje kamere, otvorit će se kamera na uređaju te će je moći koristiti kako bi napravio trenutnu sliku s prednjom ili zadnjom kamerom. Kada korisnik napravi sliku i pritisne na gumb za dodavanje slike unutar kamere, slika će biti poslana aplikaciji i bit će spremljena kao i slike koje se dodaju iz galerije.



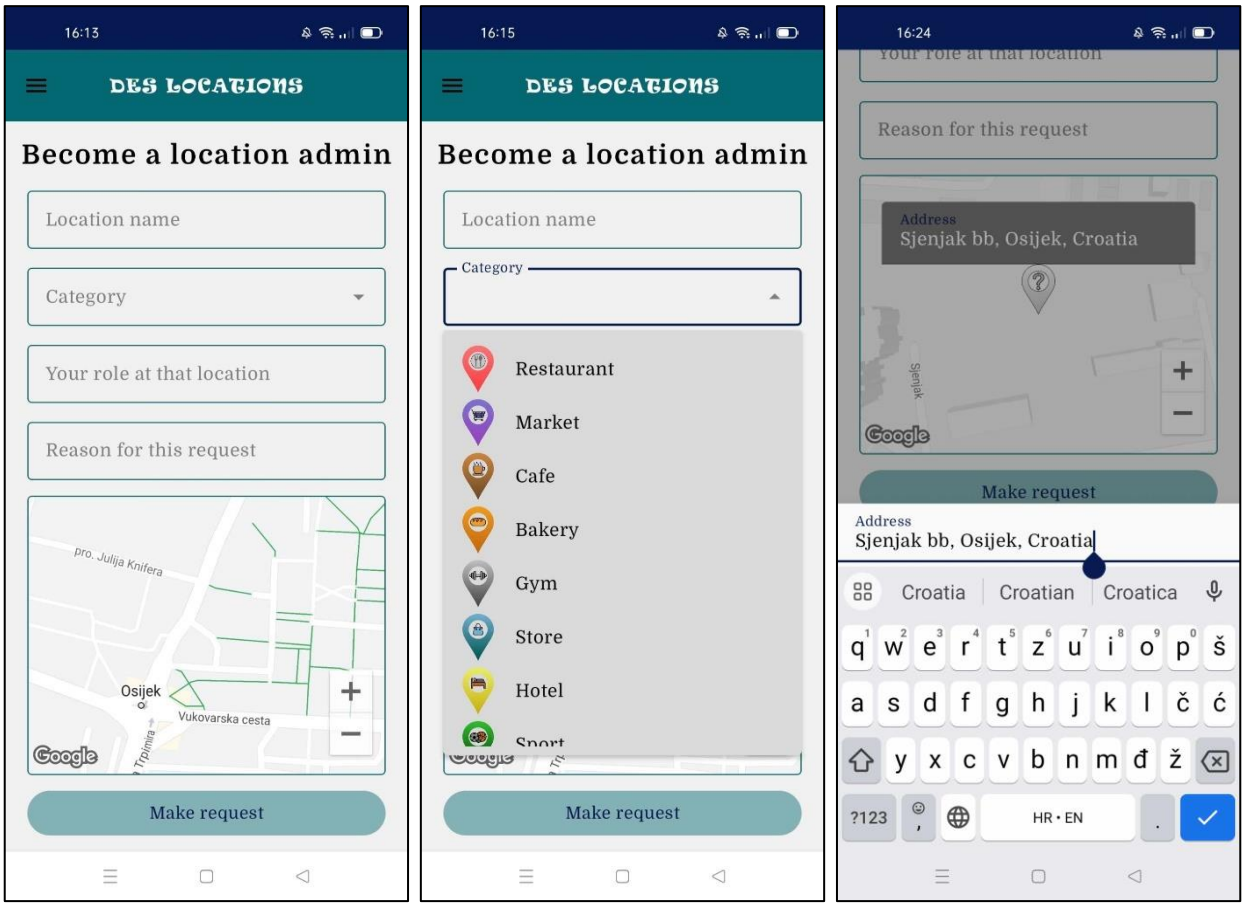
Slika 4.11. Prikaz dodanih slika s kamerom i galerijom te prikaz obavijesti o novoj objavi

Nakon što je korisnik dodao slike pomoću kamere ili iz galerije slike će biti prikazane u listi redom kako ih je korisnik dodavao što je vidljivo na slici 4.11. lijevo. Slike mogu biti različitih dimenzija te će se one pravilno formirati kako bi vizualno bilo što bolje. Svaka postavljena slika može biti obrisana pritiskom na gumb u desnom gornjem uglu slike. Prilikom brisanja slike bit će napravljena animacija uklanjanja slike sa zaslona, a druge slike u listi će se pomaknuti za jedno mjesto. Kada je korisnik završio s kreiranjem objave i unio je sve potrebne podatke, objava će se moći kreirati, u suprotnom će biti označen tekstualni okvir kojeg treba ispuniti kako bi se napravila objava. Kada se objava kreira, svi korisnici koji su pretplaćeni na lokaciju u kojoj je objava napravljena dobivaju obavijest na mobitel kao što se vidi na slici 4.11. desno. U obavijesti su sadržani podatci o nazivu lokacije unutar koje je napravljena objava i o naslovu objave. Pritiskom na obavijest korisnik može odmah doći do zaslona s prikazom detalja o obavijesti, a korisnik pri tome ne mora imati aktivnu aplikaciju.



Slika 4.12. Zaslon za prikaz razgovora unutar mjesta

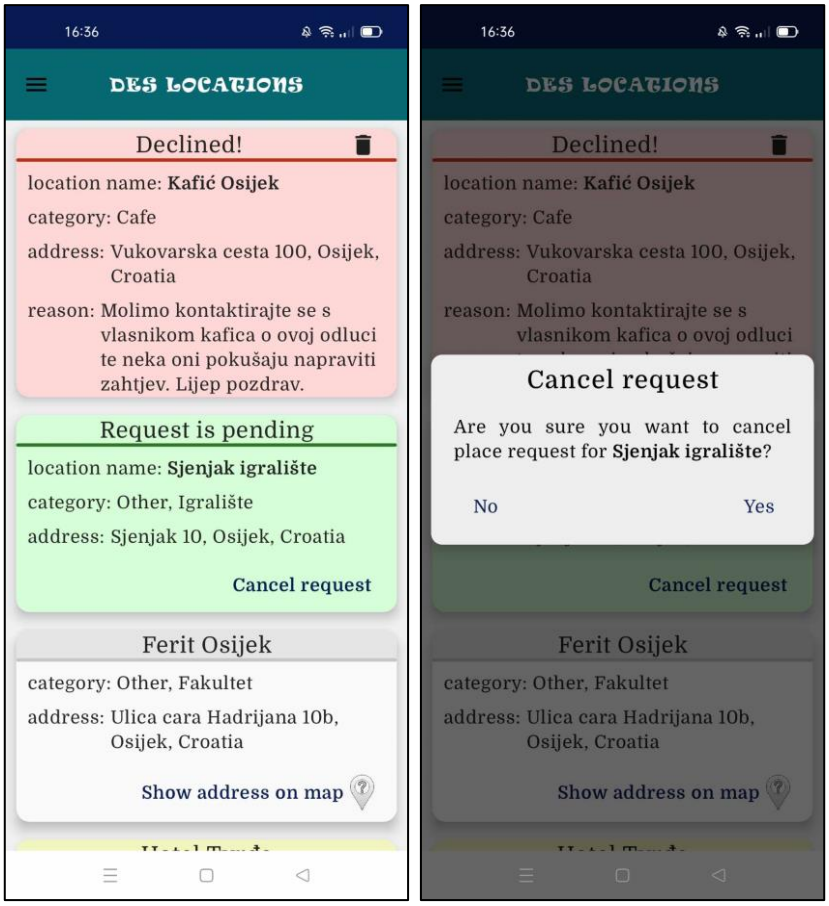
Posljednja mogućnost unutar prikaza detalja o mjestu je prikaz razgovora, a zaslon razgovora prikazan je na slici 4.12. Svi korisnici koji pregledavaju neko mjesto mogu poslati poruku, bilo to pitanje ili informacija kako bi drugi korisnici to mogli vidjeti. Unutar razgovora mogu se slati tekstualne poruke. Korisnik jednostavno može napisati tekst i pritisnuti gumb za slanje poruke što je omogućeno na tipkovnici ili unutar tekstualnog okvira za slanje poruke što se vidi na slici 4.12. desno. Drugi korisnici tada mogu vidjeti poslanu poruku, pa čak će je mogu odmah vidjeti i korisnici koji su trenutno u razgovoru jer se radi o podatcima uživo i njihov zaslon će se osvježiti s novom porukom. Na lijevoj strani zaslona prikazane su poruke drugih korisnika, dok su na desnoj strani prikazane poruke prijavljenog korisnika. Tako da će poruka koju je neki korisnik poslao njemu biti prikazana drugačije u odnosu na poruke drugih korisnika. Prikaz poruke se sastoji od jednostavne ikone s početnim slovom imena autora poruke, zatim od imena i prezimena autora, od teksta poruke i od datuma i vremena kada je poruka poslana. Poruke se u listi prikazuju od najstarijih do najnovijih, tako da će najnovije poruke biti prikazane na dnu, a korisnik može pomicati listu te tako pročitati i starije poruke.



Slika 4.13. Zaslon za kreiranje zahtjeva za novo mjesto

Slikom 4.13. prikazan je zaslon koji služi da korisnik može kreirati zahtjev za novo mjesto koje bi se prikazivalo drugim korisnicima. Svi korisnici imaju mogućnost kreiranja zahtjeva ali to ne znači da će njihov zahtjev biti odobren. Jedino moderatori aplikacije imaju mogućnost odobriti zahtjev za neko mjesto. Kako bi korisnik kreirao zahtjev mora unijeti podatke o nazivu mjesta i kategoriji mjesta koju odabire kroz padajući izbornik prikazan na slici 4.13. u sredini. U padajućem izborniku na slici prikazan je dio kategorija, a korisnik može pomicati padajući izbornik te tako pogledati sve kategorije. Ako korisnik odabere kategoriju naziva ostalo, tada će mu biti ponuđen dodatni tekstualni okvir kako bi unio naziv te kategorije. Korisnik također mora unijeti podatke o njegovoj ulozi na tom mjestu i razlogu zašto želi napraviti to mjesto što će poslužiti moderatoru aplikacije kako bi odlučio da li odobriti zahtjev ili odbiti. Također, korisnik mora još označiti na karti gdje se to mjesto nalazi, a karta je u potpunosti funkcionalna tako da se može pomicati, približavati i udaljavati. Korisnik može nekoliko puta označavati mjesto na karti ali uvijek će biti prikazano samo jedno mjesto i to krajnje mjesto će biti uzeto kao lokacija tog mjesta. Ako se na karti prikaže kriva adresa ili nepotpuna adresa, korisnik ju može urediti pritiskom na tekstualni

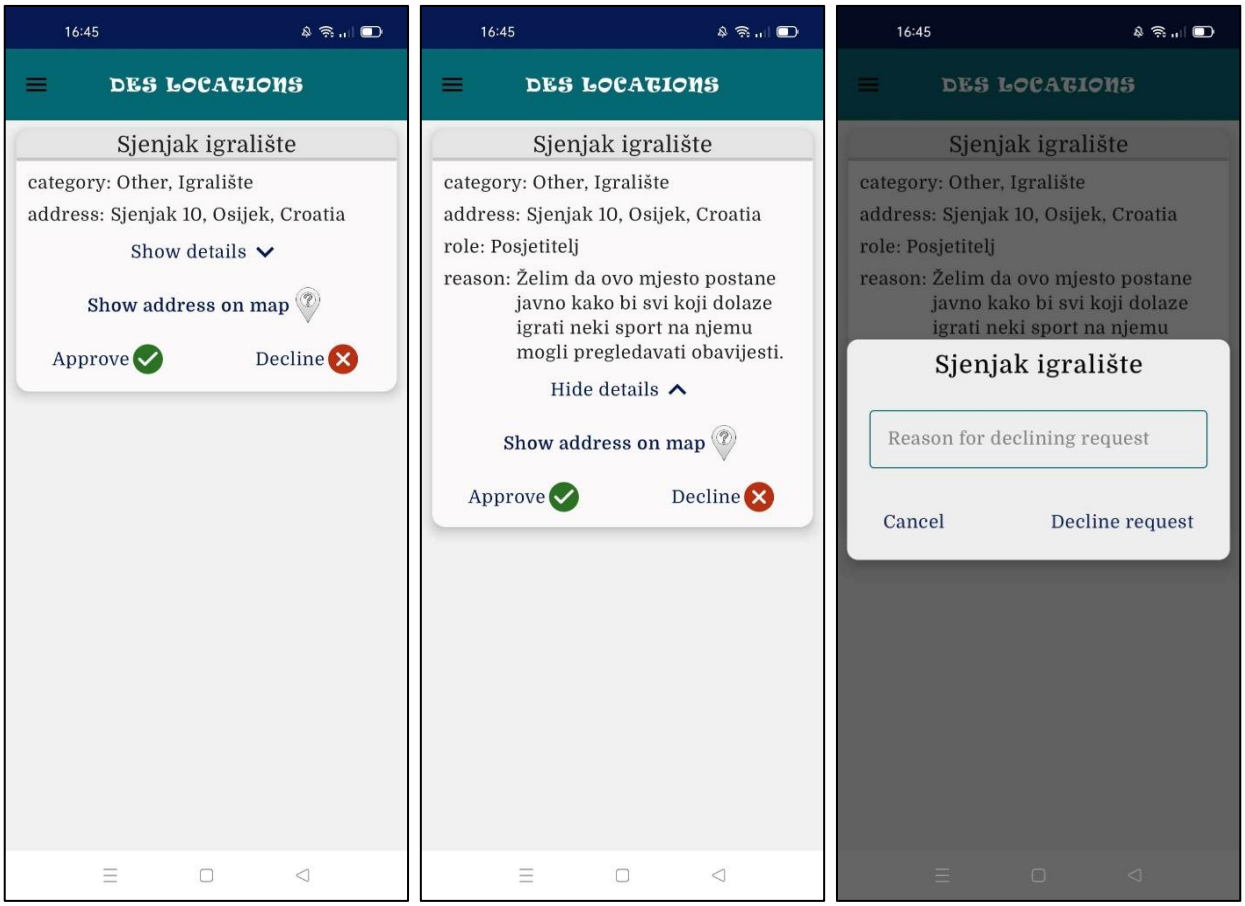
okvir za uređivanje adrese iznad markera kao što to slika 4.13. desno prikazuje. Na kraju, korisnik pritiskom na gumb kreira zahtjev za novo mjesto, a ako nešto nije dobro ispunjeno bit će označeno i upotpunjeno s prikladnom porukom.



Slika 4.14. Zaslon za prikaz lokacija trenutno prijavljenog korisnika i dijaloški okvir za otkazivanje zahtjeva

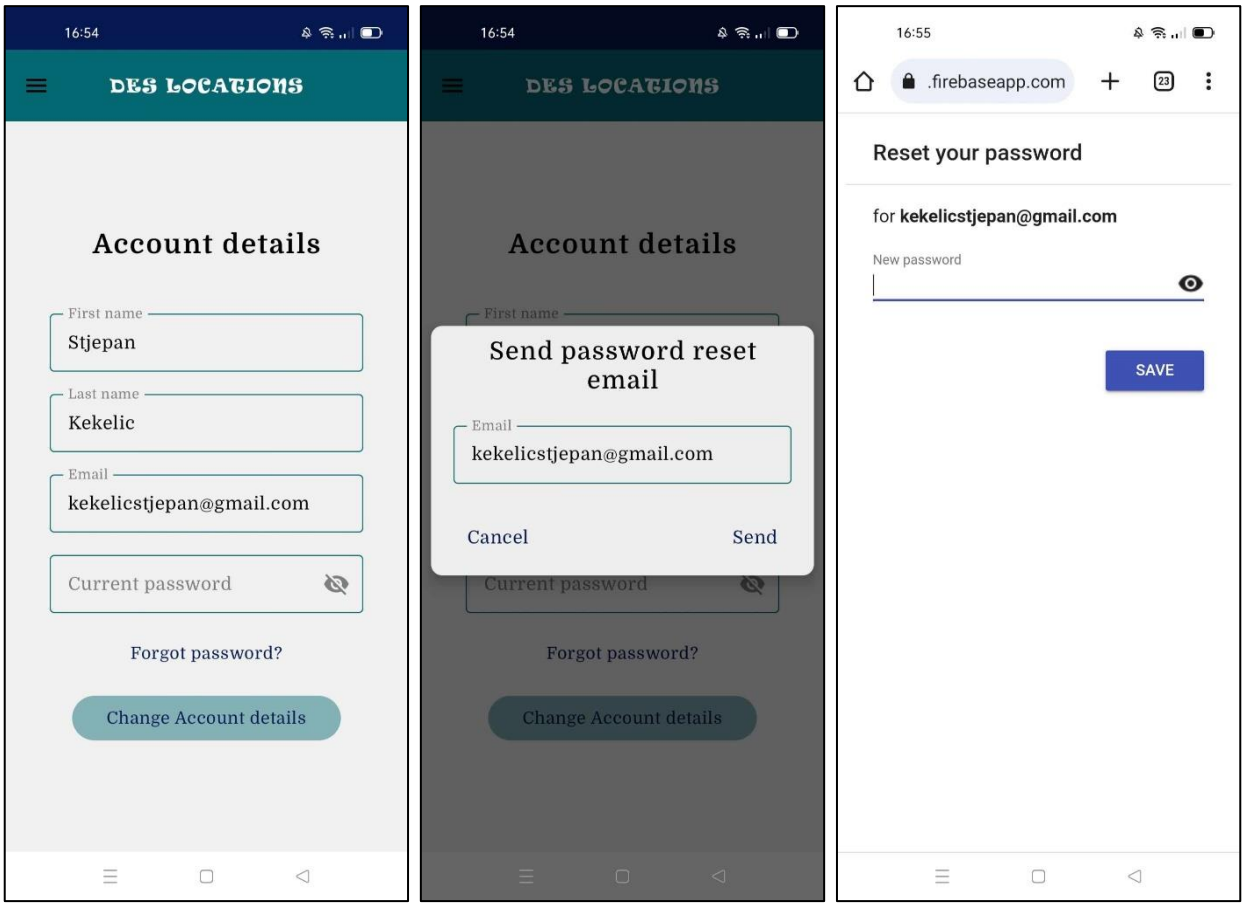
Pomoću navigacijske ladice korisnik može doći do zaslona za prikaz njegovih lokacija kao i zahtjeva koje je napravio što je prikazano na slici 4.14. lijevo. Unutar tog zaslona korisniku se prikazuju zahtjevi koji su odbijeni, zahtjevi koji čekaju odobrenje i zahtjevi koji su odobreni, odnosno mjesta gdje je trenutno prijavljeni korisnik administrator. Stavka unutar liste koja prikazuje odbijene zahtjeve označena je crveno što signalizira korisniku da nešto nije dobro. Unutar te stavke korisnik može vidjeti informacije koje je dao za to mjesto, ali može vidjeti i razlog zašto je to mjesto odbijeno, a mora ga upisati moderator koji odbija taj zahtjev. Korisnik ima mogućnost brisanja te stavke pritiskom na gumb u desnom gornjem uglu te stavke. Druga prikazana stavka označava zahtjeve koji trenutno čekaju odobrenje od moderatora te je prikazana zelenom bojom. Korisnik ima mogućnost otkazivanja tog zahtjeva gdje pritiskom na gumb za

otkazivanje mora to isto potvrditi u dijaloškom okviru koji je prikazan na slici 4.14. desno. Zadnja prikazana stavka je odobreni zahtjev odnosno mjesto gdje je taj korisnik sada administrator.



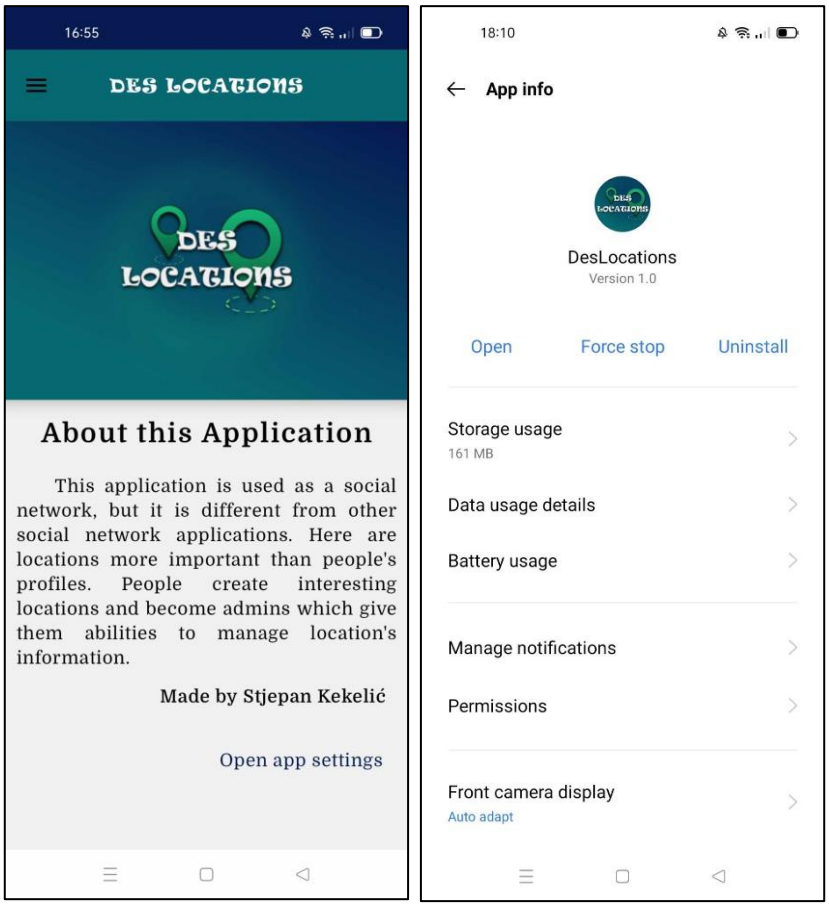
Slika 4.15. Zaslon za prikaz zahtjeva za novo mjesto i dijaloški okvir za upis razloga za odbijanje zahtjeva

Samo moderatori aplikacije mogu vidjeti zaslon za prikaz zahtjeva za novo mjesto što je prikazano na slici 4.15. Moderator može vidjeti sve informacije nekog zahtjeva te odlučiti želi li odobriti ili odbiti taj zahtjev. Pritiskom na gumb „*Show details*“ moderator može vidjeti više detalja o toj lokaciji. Ako se moderator odluči odbiti zahtjev, onda će morati upisati razlog zašto je zahtjev odbijen unutar dijaloškog okvira prikazanog na slici 4.15. desno. Tada će korisnik koji je napravio taj zahtjev dobiti informaciju zašto mu je zahtjev odbijen. Ako moderator odobri zahtjev, korisnik koji je napravio zahtjev postaje administrator tog mjesta i dalje se sam mora brinuti o pisanju informacija i objava za tu lokaciju.



Slika 4.16. Zaslon za prikaz korisničkih podataka i prikaz promjene zaporke

Unutar navigacijske ladice, kada korisnik odabere opciju za prikaz detalja korisničkog računa, prikazat će se zaslon kao na slici 4.16. lijevo. Tu je prikazano ime, prezime i adresa elektroničke pošte koju korisnik može promijeniti nakon što unese ispravnu trenutnu zaporku i pritisne gumb za promjenu korisničkih detalja. Ako je korisnik zaboravio zaporku, ima mogućnost da ju promijeni, ali ne unutar aplikacije već putem elektroničke pošte. Nakon što korisnik pritisne gumb da je zaboravio zaporku, otvara se dijaloški okvir kao na slici 4.16. u sredini. Trenutno prijavljena korisnička adresa bit će postavljena unutar tekstualnog okvira na koju će biti poslana elektronička pošta za promjenu zaporke. Nakon što korisnik pritisne tipku za slanje, za nekoliko sekundi stići će mu elektronička pošta s poveznicom na kojoj može promijeniti zaporku. Kada se poveznica otvori, bit će prikazan zaslon kao na slici 4.16. desno gdje korisnik treba unijeti novu zaporku.



Slika 4.17. Zaslon za prikaz informacija o aplikaciji i opcije aplikacije

Zadnji zaslon unutar aplikacije je zaslon za prikaz informacija o aplikaciji, prikazan slikom 4.17. lijevo do kojeg je moguće doći pomoću navigacijske ladice. To je jednostavan zaslon koji prikazuje logo aplikacije koji je samostalno napravljen pomoću Adobe XD alata za dizajniranje web i mobilnih aplikacija [19]. Unutar navedenog alata napravljeni su i markeri koji služe za različite kategorije lokacija. Osim prikaza loga aplikacije, prikazan je i kratak tekst s opisom čemu služi aplikacija. Na dnu je prikazano ime autora i dodatni gumb kako bi korisnik mogao direktno otvoriti opcije aplikacije, ako ga zanima nešto više o njoj, prikazano slikom 4.17. desno.

Svi navedeni zasloni prikazani su na engleskom jeziku, a dostupan je i hrvatski jezik, odnosno onaj dio koji korisnici nisu unosili jer je taj dio moguće unijeti na bilo kojem jeziku. Ako korisnik ima postavljen hrvatski jezik unutar opcija mobilnog uređaja bit će mu postavljeni zasloni na hrvatskom, u suprotnom bit će prikazan engleski jezik. Također zasloni su prikazani pomoću svijetle teme na mobilnom uređaju, a dostupna je i tamna tema. Ako korisnik ima uključen tamni način rada mobilnog uređaja bit će prikazani tamniji zasloni s drugačijim bojama. Također, aplikaciju je moguće funkcionalno koristiti i pomoću rotiranog ekrana na mobilnom uređaju.

5. ZAKLJUČAK

Napredak čovječanstva sve je brži, a ljudi sve teže drže korak s tim napretkom. Tako su pametni telefoni postali neophodni za većinu ljudi koja nastoji držati korak s napretkom. Pomoću pametnih telefona ljudi traže brojne informacije koje im pomažu u svakodnevnom životu, većinom preko društvenih mreža i raznih aplikacija. Često te informacije koje korisnici traže su informacije o lokacijama koje žele posjetiti ili vidjeti što se na njima događa pri tome često budu iznenađeni stvarnoj situaciji koja je na toj lokaciji.

Mobilna aplikacija koja je izrađena za ovaj diplomski rad nudi jedno od rješenja kako omogućiti korisnicima pretraživanje lokacija, a pri tome da informacije koje pronalaze na određenoj lokaciji budu što točnije i po mogućnosti od različitih korisnika. Korisnici koji pretražuju mjesta unutar aplikacije mogu ujedno promovirati i vlastite. Korisnici koji su posjetili neku od lokacija ili su u kontaktu mogu ostaviti korisne informacije drugim korisnicima.

Rezultat rada aplikacije je zadovoljavajući. Lako se može pretražiti neka lokacija, vidjeti informacije o njoj i ostaviti informaciju drugim korisnicima kroz objavu ili u razgovoru lokacije. Trenutno, moderator aplikacije ne može biti bilo tko te je to za sada samo kreator aplikacije te on odobrava zahtjeve za nova mjesta. Proces odobravanja ili odbijanja zahtjeva je dosta brz tako da moderator ne troši previše vremena na to. Takvim načinom odlučivanja dobiva se jako puno na ispravnosti informacija za neku lokaciju. Kada je zahtjev odobren, administrator lokacije se treba brinuti o lokaciji i da lokacija pruža točne informacije. Lako se može primijetiti ako je neka lokacija loša jer će tada drugi korisnici reagirati s lošijim objavama i porukama ili ih neće uopće biti. Stvaranje objava predstavlja važan alat za informiranje drugih korisnika. Objave mogu biti događaji ili informacije, odnosno baš ono što je potrebno za jednu lokaciju. Dodavanje slika u objavi je jako dobro jer će tada korisnici osim teksta, vidjeti što se događa na lokaciji. Razgovor unutar lokacije pruža jednostavan način kako korisnici mogu postavljati pitanja, odgovarati, dati pohvale ili kritike te vidjeti što drugi korisnici pišu.

Izrađena aplikacija ima preko desetak zaslona, brojne opcije, različite korisnike što je naravno otežalo testiranje aplikacije koja je dovedena do kvalitetne razine. Postoji još prostora za unaprjeđenje aplikacije gdje se mogu dodati još brojne značajke koje bi pomogle korisnicima i olakšale rad s aplikacijom. Ova aplikacija može se smatrati tek prvom verzijom, a ako se pokaže zanimljivom za velik broj ljudi lako se može nadograđivati.

LITERATURA

[1] „Google Karte, Aplikacije na Google Playu“. https://play.google.com/store/apps/details?id=com.google.android.apps.maps&hl=hr (pristupljeno 19. lipanj 2023.).

[2] „Near Me: Find Places Around Me, Aplikacije na Google Playu“. https://play.google.com/store/apps/details?id=com.inventorinc.nearforme&hl=hr (pristupljeno 19. lipanj 2023.).

[3] „AroundMe, Aplikacije na Google Playu“. https://play.google.com/store/apps/details?id=com.tweakersoft.aroundme&hl=hr (pristupljeno 19. lipanj 2023.).

[4] „Eventbrite – Discover events, Aplikacije na Google Playu“. https://play.google.com/store/apps/details?id=com.eventbrite.attendee&hl=hr (pristupljeno 19. lipanj 2023.).

[5] „Visit A City, Aplikacije na Google Playu“. https://play.google.com/store/apps/details?id=com.visitacity.visitacityapp&hl=hr (pristupljeno 19. lipanj 2023.).

[6] „Firebase“. https://firebase.google.com/ (pristupljeno 21. lipanj 2023.).

[7] „Firestore“, *Firebase*. https://firebase.google.com/docs/firestore (pristupljeno 21. lipanj 2023.).

[8] „Cloud Storage for Firebase“, *Firebase*. https://firebase.google.com/docs/storage (pristupljeno 21. lipanj 2023.).

[9] „Firebase Cloud Messaging“. https://firebase.google.com/docs/cloud-messaging (pristupljeno 07. kolovoz 2023.).

[10] „Navigation“, *Android Developers*. https://developer.android.com/guide/navigation (pristupljeno 24. lipanj 2023.).

[11] A. Pathak, „Dependency Injection — Dagger hilt in Android“, *Medium*, 23. svibanj 2023. https://medium.com/@myofficework000/dependency-injection-dagger-hilt-in-android-14a7f03050e8 (pristupljeno 24. lipanj 2023.).

[12] „Dependency injection with Hilt“, *Android Developers*. https://developer.android.com/training/dependency-injection/hilt-android (pristupljeno 24. lipanj 2023.).

[13] C. Arriola, „Composable Functions“, *Android Developers*, 22. rujan 2022. https://medium.com/androiddevelopers/composable-functions-a505ab20b523 (pristupljeno 25. lipanj 2023.).

[14] „Lists and grids | Jetpack Compose“, *Android Developers*. https://developer.android.com/jetpack/compose/lists (pristupljeno 25. lipanj 2023.).

[15] „Preview your UI with Composable previews | Jetpack Compose“, *Android Developers*. https://developer.android.com/jetpack/compose/tooling/previews (pristupljeno 25. lipanj 2023.).

[16] „Kotlin coroutines on Android“, *Android Developers*. https://developer.android.com/kotlin/coroutines (pristupljeno 26. lipanj 2023.).

[17] „Why you need ViewModels and why you don’t“, *Composables – Jetpack Compose components to build your next idea to life blazing fast.*, 13. svibanj 2023. https://www.composables.com/tutorials/viewmodels-in-jetpack-compose (pristupljeno 26. lipanj 2023.).

[18] „Repository in Android’s MVVM architecture – Digital Solutions Consulting GmbH“. https://digital-solutions.consulting/uncategorized/repository-in-androids-mvvm-architecture/ (pristupljeno 26. lipanj 2023.).

[19] „Get started with Adobe XD“. https://helpx.adobe.com/xd/get-started.html (pristupljeno 08. kolovoz 2023.).

SAŽETAK

Na početku ovog diplomskog rada prikazani su načini pretraživanja željenih lokacija te je opisano drugačije rješenje koje se razrađuje u ovom radu. Mobilna aplikacija koja rješava navedeni problem ima za zadatak uključiti što više korisnika na određenu lokaciju kako bi oni informirali druge korisnike o događajima na toj lokaciji. Glavni cilj aplikacije je pružiti brz i jednostavan način pretraživanja željenih lokacija putem karte ili pomoću liste te za svaku lokaciju pružiti važne informacije i događaje. Aplikacija se može smatrati kao društvena mreža, ali naglasak nije na ljudima već na lokacijama koje stvaraju korisnici. Korištenjem Firebase usluga omogućeno je spremanje i dohvaćanje podataka o korisnicima i lokacijama, spremanje slika te obavještavanje korisnika o novim objavama unutar pretplaćene lokacije. Aplikacija je kreirana pomoću Android Studio razvojnog okruženja i Jetpack Compose razvojnog okvira korištenjem Kotlin programskog jezika. U završetku ovog rada predstavljene su mogućnosti aplikacije i njeno korištenje koje može olakšati pronalazak željene lokacije i promoviranje vlastite.

Ključne riječi: Android, Firebase, Jetpack Compose, karta, lokacije

ABSTRACT

Mobile Application for information about desired locations and events

At the beginning of this thesis, the methods of searching for the desired locations are presented and a different solution which is elaborate in this paper is described. The mobile application that solves the mentioned problem has the task to involve as many users as possible in a certain location so that they can inform other users about events at that location. The main goal of the application is to provide a quick and simple way to search for the desired locations via a map or using a list, and to provide essential information and events for each location. The application can be considered as a social network, but the emphasis is on user-created locations rather then people. Using Firebase services allows the storage and retrieval of data about users and locations, image storage and notifying users about new posts within subscribed location. The application has been created using the Android Studio development environment and the Jetpack Compose framework using Kotlin programming language. At the end of this paper, the possibilities of the application and its use which can facilitate finding the desired location and promoting your own are presented.

Key words: Android, Firebase, Jetpack Compose, locations, map

ŽIVOTOPIS

Stjepan Kekelić je rođen 25. srpnja 1999. godine u Požegi. Osnovnu školu je pohađao u Osnovnoj školi Stjepana Radića Čaglin te nakon osam razreda odličnog uspjeha upisuje Gimnaziju u Požegi, smjer opća gimnazija. Tijekom srednjoškolskog razdoblja sudjeluje na natjecanjima iz matematike i informatike i aktivno se bavi nogometom koji mu je danas najomiljeniji hobi. Godine 2018. upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, preddiplomski sveučilišni studij, smjer računarstvo. Nakon završenog preddiplomskog studija, 2021. godine upisuje diplomski studij, smjer programsko inženjerstvo gdje se je najviše zainteresirao za izradu Android mobilnih aplikacija pomoću View i Jetpack Compose razvojnog okvira. Krajem 2022. godine pohađa stručnu praksu u tvrtki EM2.d.o.o izrađujući mobilnu aplikaciju pomoću React Native razvojnog okvira i REST servis pomoću Spring Boot razvojnog okvira.

_____

Potpis autora