Android aplikacija za traženje i pružanje usluga prijevoza

Novinc, Antonio

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

Sveučilišni studij

ANDROID APLIKACIJA ZA TRAŽENJE I PRUŽANJE USLUGA PRIJEVOZA

Diplomski rad

Antonio Novinc

Osijek, 2022.

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

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 12.12.2022.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Antonio Novinc
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1148R, 13.10.2020.
OIB studenta:	09246502784
Mentor:	Izv. prof. dr. sc. Ivica Lukić
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv.prof.dr.sc. Zdravko Krpić
Član Povjerenstva 1:	Izv. prof. dr. sc. Ivica Lukić
Član Povjerenstva 2:	Izv. prof. dr. sc. Mirko Köhler
Naslov diplomskog rada:	Android aplikacija za traženje i pružanje usluga prijevoza
Znanstvena grana diplomskog rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak diplomskog rada:	Izraditi Android aplikaciju u Kotlin programskom jeziku. Glavna funkcionalnost je omogućiti korisniku opciju davanja usluge prijevoza i prikaz ponuđenih usluga prijevoza. Osim toga, omogućiti korisniku registraciju i prijavu, filtriranje usluga po različitim parametrima, prikaz adresa na karti i mogućnost poziva iz aplikacije. Rezervirano za: Antonio Novinc
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: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	12.12.2022.

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

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 18.12.2022.

Ime i prezime studenta:	Antonio Novinc
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1148R, 13.10.2020.
Turnitin podudaranje [%]:	7

Ovom izjavom izjavljujem da je rad pod nazivom**: Android aplikacija za traženje i pružanje usluga prijevoza**

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

Današnje doba može se okarakterizirati kao doba digitalizacije. Gotovo je svaka usluga digitalizirana ili je u procesu digitalizacije. Osim toga, velika većina svjetske populacije koristi pametne mobilne uređaje. Po nekim istraživanjima čak oko 70 % svih ljudi posjeduje pametne mobilne uređaje. Također, istraživanja pokazuju da u Hrvatskoj trenutno ima više mobilnih uređaja nego stanovnika. Stoga ni ne čudi da su aktivnosti kao što su kupovina odjeće, odlazak u banku, narudžba hrane, rezerviranje hotela i slično zamijenjene mobilnim aplikacijama koje nam omogućavaju te usluge, pri čemu znatno smanjuju vrijeme utrošeno za obavljanje pojedinih zadataka. Primjerice, vrijeme čekanja u redu u banci, odlaska u trgovine po odjeću ili bespotrebnog zvanja i pisanja e-maila određenom hotelu za slobodan smještaj. Mobilne aplikacije u samo par klikova rješavaju navedene probleme i usluge čine jednostavnijim za korisnika, ali i pružatelja usluga.

Svjedoci smo krize koja je započela još COVID-19 pandemijom i kojoj se ne nazire kraj. Kriza se posebno odrazila na cijene goriva, što je dovelo do toga da su autoprijevoznici jednostavno povisili svoje cijene prijevoza, ukinuli ponude kao što su studentski popusti, pa čak i ukinuli određene linije. Potaknuti time, ljudi se sve češće okreću drugim načinima prijevoza. Jedan od načina koji je sve popularniji jest nuđenje usluga prijevoza preko društvenih mreža, kako bi se spojilo ugodno s korisnim. Korisnici kojima je potrebna usluga prijevoza jeftinije i brže dođu do određene lokacije, a korisnici koji nude prijevoz nadoknade troškove goriva naknadom koju naplate za usluge prijevoza.

Zbog svega navedenog, razvila se ideja o Android aplikaciji kao idealnom rješenju ovog problema. Vodeći se ovom problematikom, cilj je diplomskog rada prikazati mogućnosti i pozitivne implikacije digitalizacije usluge prijevoza za korisnike, neovisno o tome je li riječ o korisnicima koji nude ili traže prijevoz.

1.1. Zadatak diplomskog rada

Zadatak je diplomskog rada realizirati mobilnu Android aplikaciju. Glavna je funkcionalnost omogućiti korisniku opciju davanja usluge prijevoza i prikaz ponuđenih usluga prijevoza. Osim toga, mobilna aplikacija omogućit će korisniku registraciju i prijavu, filtriranje usluga po različitim parametrima, prikaz adresa na karti i mogućnost poziva iz aplikacije.

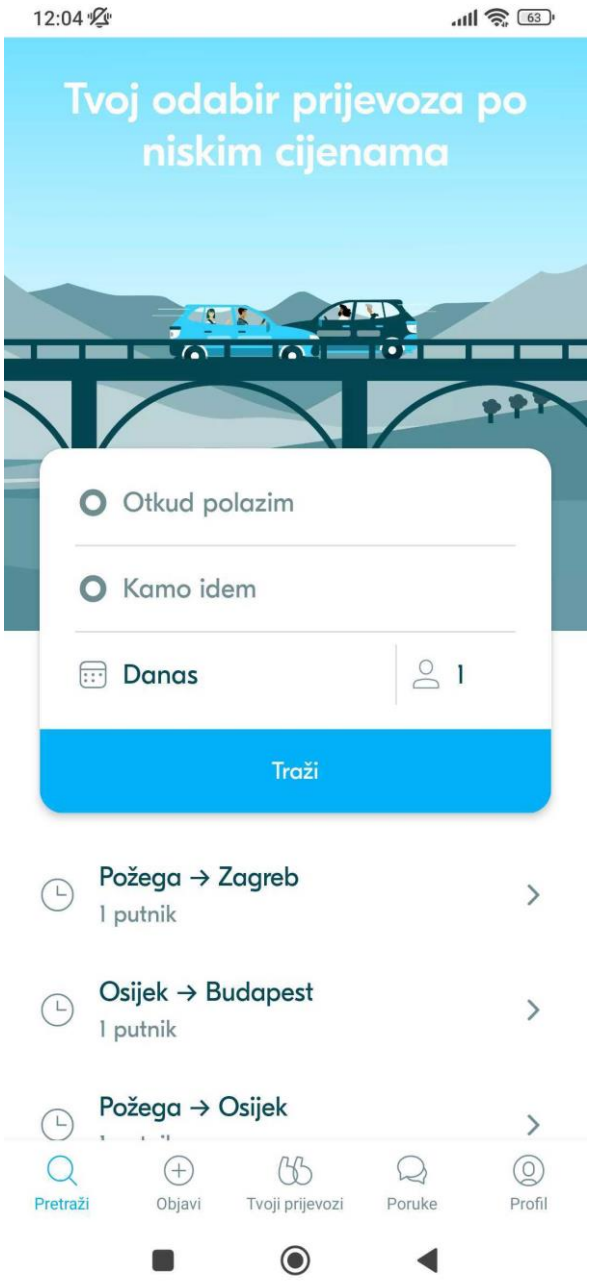
2. PREGLED PODRUČJA TEME

Studenti koji ne studiraju u mjestu stanovanja primorani su putovati kući javnim prijevozom ili drugim alternativama. S obzirom na to da je COVID-19 pandemija izazvala i povećanje cijene javnog prijevoza, studenti su prepušteni drugim, alternativnim načinima putovanja. Stoga, na društvenim mrežama nerijetko se stvaraju grupe gdje se nude i traže usluge prijevoza.

S obzirom na vrijeme u kojem živimo, vrijeme digitalizacije i tehnologije, pojavila se mogućnost za unaprjeđivanje ovakve vrste komunikacije, nuđenja i traženja usluge prijevoza do odredišta. Ovakav način mogao bi obuhvatiti i ljude koji možda nemaju određene društvene mreže. Zbog funkcionalnosti koje aplikacija nudi, korisnici mogu brzo i jednostavno naći potrebnu uslugu, ali isto tako i ponuditi takvu vrstu usluge. Također, korisnici mogu jednostavno pronaći adresu početne i krajnje točke puta, filtrirati usluge prema parametrima koji zadovoljavaju njihove potrebe i slično. Ovo su samo neke od prednosti koje aplikacija pruža te je daljnjim razvojem i usavršavanjem moguće ostvariti i druge prednosti.

2.1. Pregled sličnih rješenja

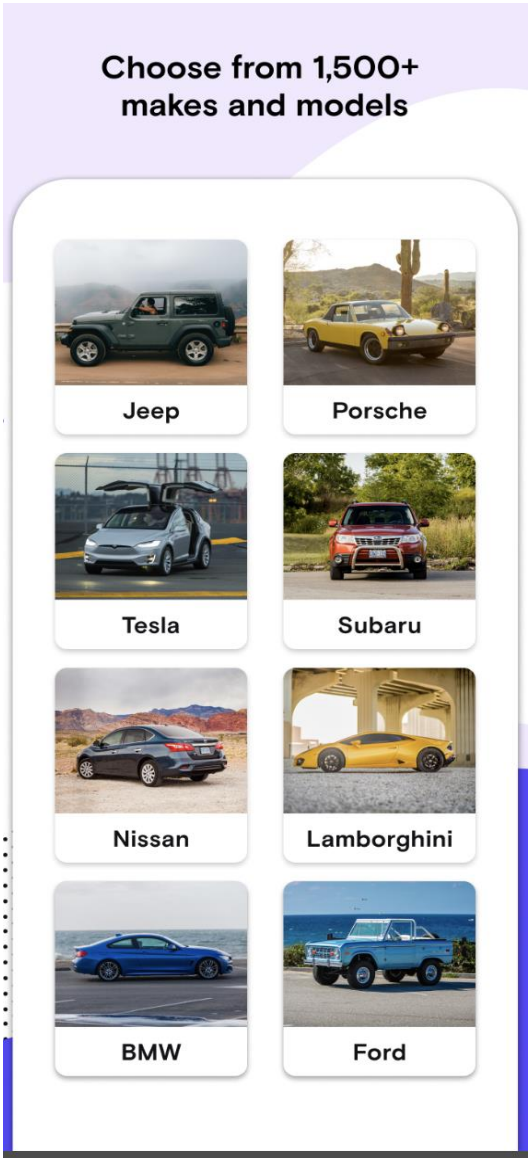
Najpopularnija aplikacija sa sličnim zahtjevima je vjerojatno *BlaBlaCar* [1]. Koliko je aplikacija popularna govori i podatak da je preuzeta čak preko 50 milijuna puta za Android uređaje s *Google Play*. *BlaBlaCar* pruža razne funkcionalnosti. Korisnik ima mogućnost prijave i registracije, traženje usluge i pružanje usluge prijevoza. Osim toga postoji opcija komunikacije putem slanja i primanja poruka unutar aplikacije. Također nakon što je korisnik našao prihvatljivu uslugu i povezao se s drugim korisnikom ili je povezao nekog drugog korisnika, aplikacija nudi mogućnost da se ocijeni vozač ili putnik što daje drugim korisnicima mogućnost kako bi lakše odlučili čiju uslugu će koristiti.



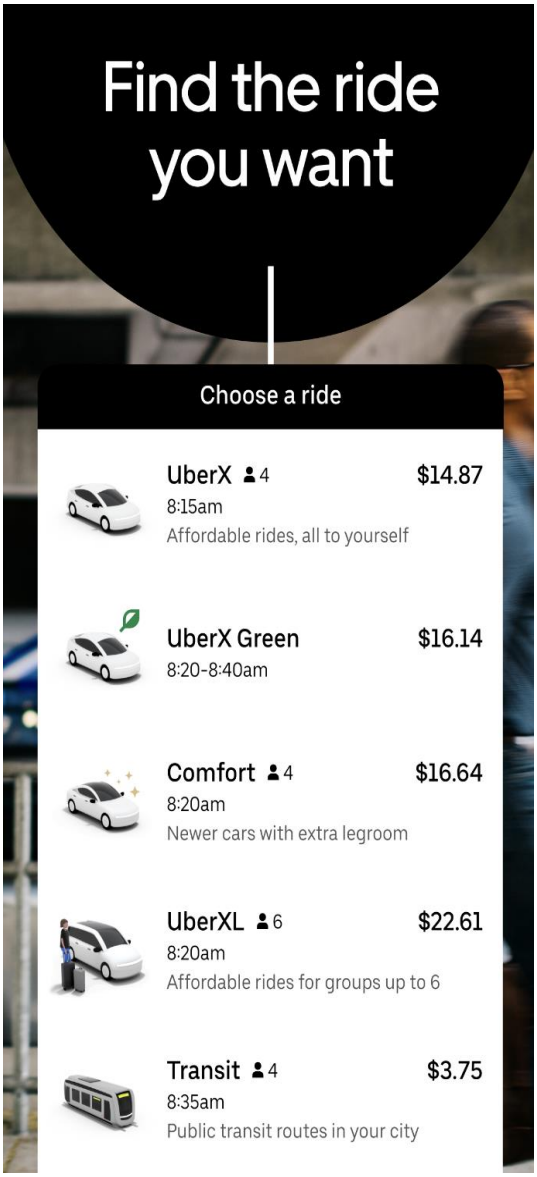
Slika 2.1. *Sučelje aplikacije BlaBlaCar*

Što se tiče stranog tržišta vjerojatno najpopularnija aplikacija sa sličnim funkcionalnostima je *Uber*. Najveća razlika je u tome što *Uber* aplikacija omogućava samo traženje prijevoza gdje zapravo korisnici nemaju mogućnost ponuditi prijevoz. Nakon registracije, korisnik unosi početnu i krajnju lokaciju svoje vožnje i nakon toga najbliži slobodni vozač ga kontaktira i dolazi s prijevozom na početnu lokacija. Također aplikacija omogućava ocjenjivanje cjelokupne vožnje i vozača nakon završetka vožnje. Postoje i druge alternative *BlaBlaCar-u*, ali nisu toliko popularne

kao *BlaBlaCar*. Neke od drugih opcija su *Turo* aplikacija gdje korisnik može ponuditi svoj automobil na korištenje drugim korisnicima odnosno ustupiti u najam svoj automobil.



Slika 2.2. *Turo aplikacija [2]*



Slika 2.3. *Uber aplikacija [3]*

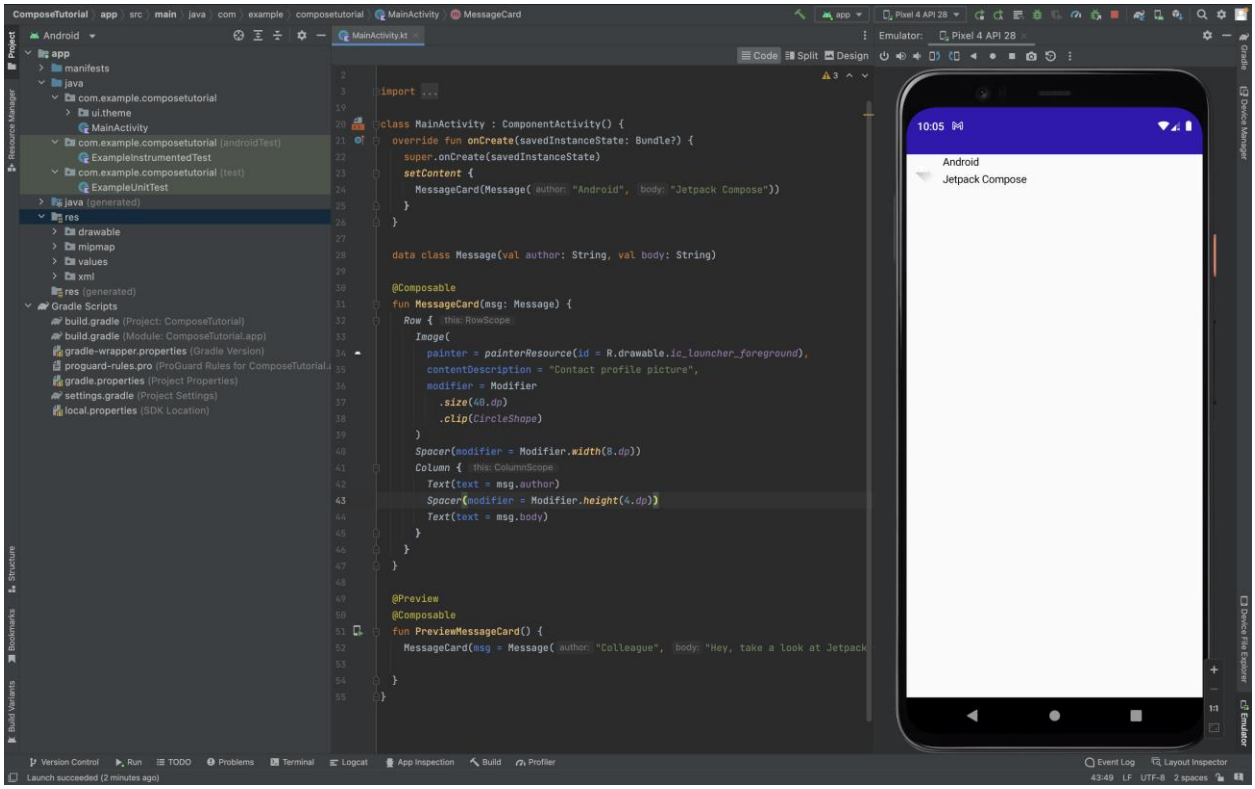
3. OPIS PRIMIJENJENIH TEHNOLOGIJA I ALATA

U ovom poglavlju bit će opisane tehnologije i alati korišteni pri izradi aplikacije..

3.1. Android Studio

Android Studio [4] je službeno razvojno okruženje razvijeno od *Googlea* koje je namijenjeno za razvoj i izradu Android mobilnih aplikacija za mobilne uređaje, televizore, tablete i pametne satove. Zasnovan je na *IntelliJ IDEA* platformi. Android Studio nudi razne alate i mogućnosti koje ubrzavaju i olakšavaju razvoj mobilnih aplikacija kao što su [5] :

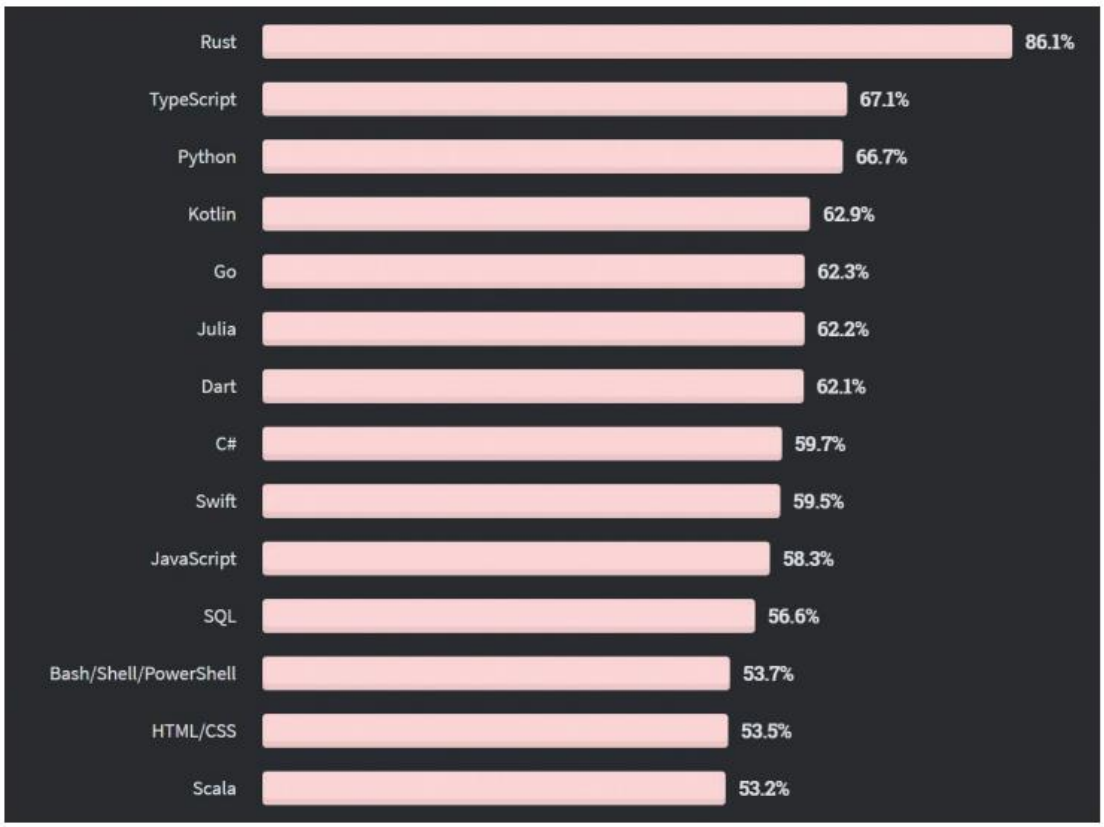
- emulator za pokretanje i *debuggiranje* aplikacija
- alati za testiranje
- integraciju s alatima za verzioniranje koda
- inteligentni uređivač koda
- alati za analiziranje performansi



Slika 3.1. *Sučelje Android Studia*

3.2. Kotlin

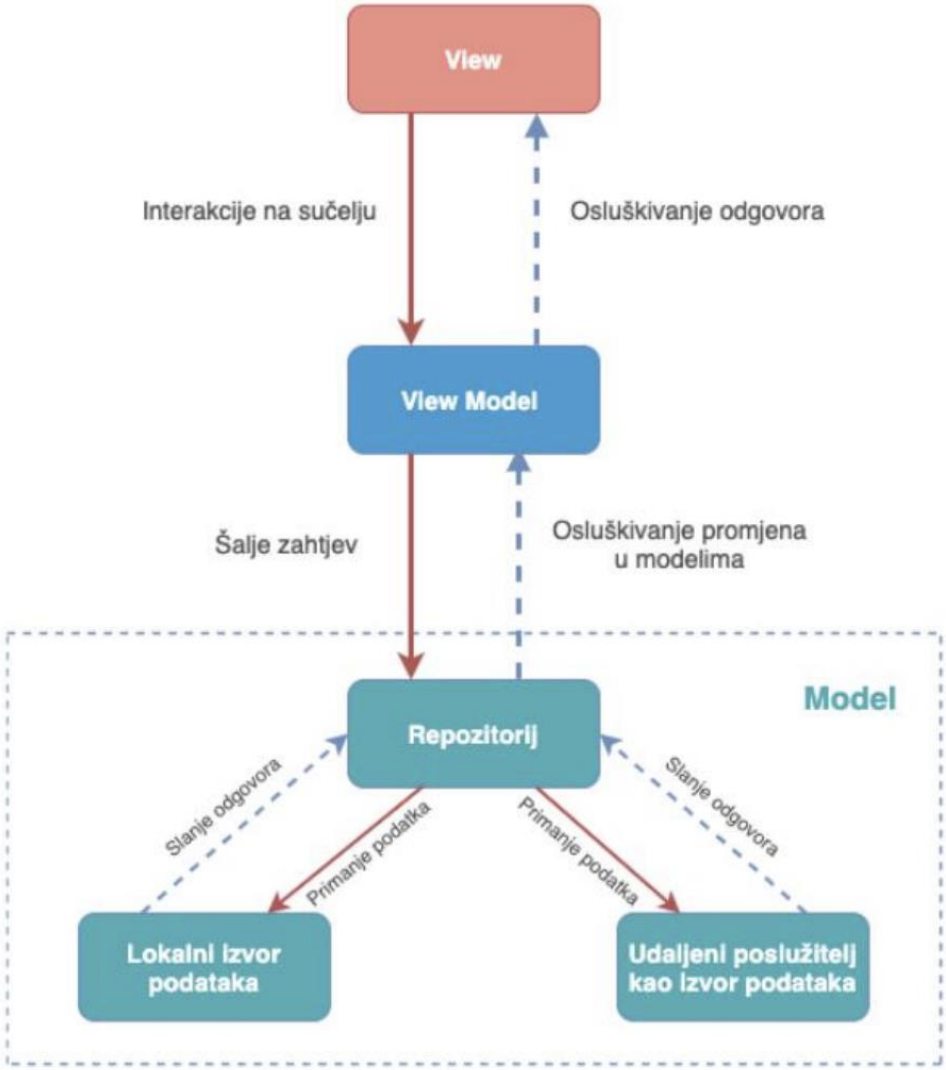
Kotlin je programski jezik koji u potpunosti kompatibilan s Android sustavom. Razvila ga je *JetBrains* tvrtka i danas je službeno proglašen kao programski jezik koji se koristi za razvoj mobilnih Android aplikacija. Nasljedio je Javu koja je prije bila prvi izbor prilikom razvoja Android mobilnih aplikacija te tako još uvijek postoje aplikacije koje su pisane u Javi, ali to se svakodnevno smanjuje jer i sam *Google* nastoji sve svoje nove biblioteke pisati u Kotlinu. Omogućuje manje pisanog koda s većom čitljivošću *(*engl. *readability)* [6]. Manje koda dovodi do lakšeg testiranja i smanjenog broja pogrešaka. U potpunosti je integriran u Android Studio te omogućuje istovremeno korištenje s Java-om bez potrebe prebacivanja koda u Kotlin. Također, sama zajednica i podrška oko programskog jezika je velika i širi se svakodnevno što doprinosi daljnjem razvoju. Primjerice, prema *Google*-u više od 60% od 1000 najboljih ocijenjenih aplikacija u *Google Play* koristi Kotlin [7].



Slika 3.2. *Lista omiljenih programskih jezika prema StackOverflow anketi 2020.godine* [8]

3.3. MVVM arhitekturalni obrazac

Svaki programer želi svoj projekt što bolje strukturirati i imati što „čišći kod" *(*engl. *clean code).* Kako bi postigli navedeno programeri koriste arhitekturalne obrasce, jedan on takvih je i MVVM arhitekturalni obrazac *(*engl. *Model – ViewModel – View)* [9]. MVVM je obrazac koji je preporučen od *Google-a* te omogućuje izradu skalabilne aplikacije koja je jednostavna za testiranje i lako razumljiva drugim programerima. Navedeni arhitekturalni obrazac omogućuje odvajanje logike korisničkog sučelja od poslovne logike držeći se pri tome načela jedinstvene odgovornosti (engl. *Single Responsibility Principle*) [10]. Na slici 3.3. je prikazana shema MVVM arhitekturalnog obrasca i kako komponente međusobno komuniciraju.



Slika 3.3. *Shema MVVM arhitekturalnog obrasca*

View prikazuje ažurirane i svježe podatke koje prima od *ViewModel-a*. On promatra odnosno *observe*-a podatke *ViewModel-a* te prateći promjene jednostavno prikazuje nove podatke na korisničko sučelje. Uglavnom *View* predstavlja fragment ili aktivnost (engl. *activity*) unutar Android aplikacije [11].

ViewModel je posrednički sloj između *Model-a* i *View-a* koji sadrži poslovnu logiku te pomoću *LiveData-e* pruža podatke koje *View* onda može promatrati te tako omogućuje nove i ažurne podatke za korisničko sučelje. Jedno od važnih svojstva ove implementacije je to da *ViewModel* ne bi trebao imati nikakvu referencu na *View* odnosno ne bi trebao biti svjestan *View-a* s kojim je u komunikaciji [11].

Model je najdublja komponenta koja sprema podatke i omogućuje ViewModel-u da dohvati stanje. Podatke može dohvaćati s udaljenog poslužitelja ili iz lokalne baze podataka. Uglavnom se koristi obrazac spremišta (engl. *repository pattern*) iako to ne mora biti pravilo. *Model* kao takav ne zna za *ViewModel* i *View* [11].

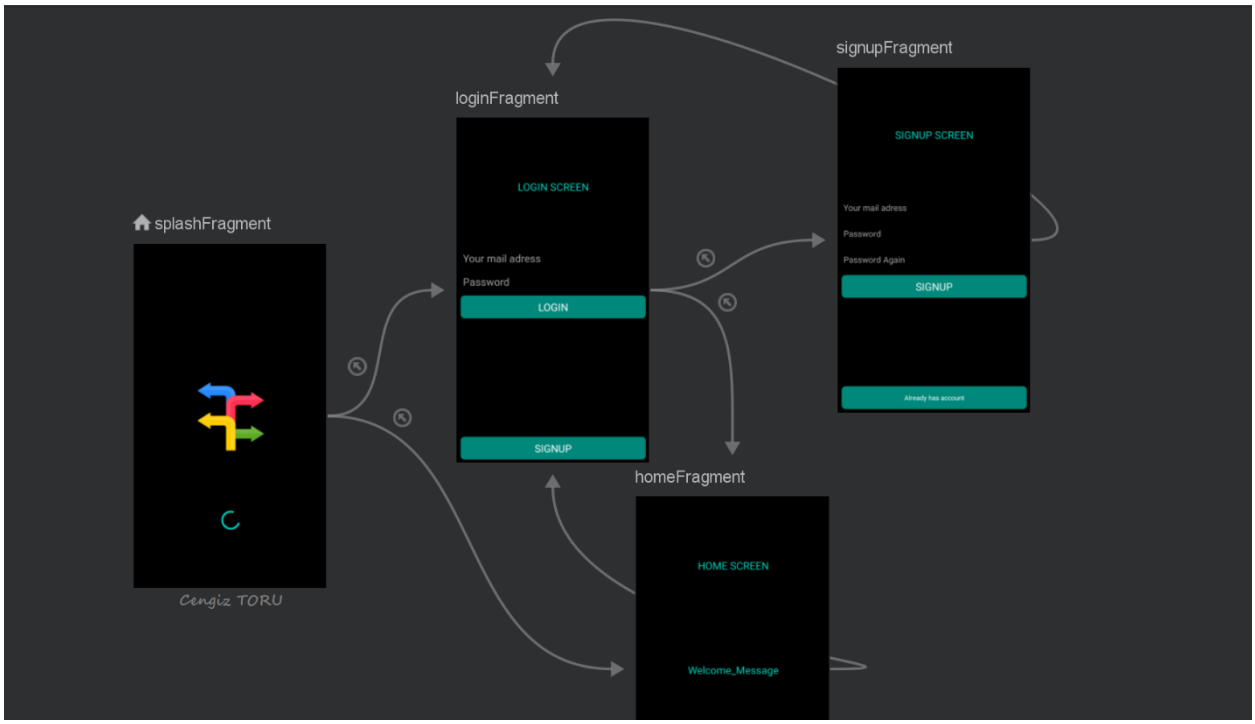
3.4. *Navigation Component*

Navigation Component je biblioteka koja pojednostavljuje implementaciju navigacije te nudi vizualni prikaz same navigacije unutar aplikacije. Pojam navigacije unutar aplikacije podrazumijeva sve interakcije koje korisniku omogućuju navigaciju do različitih sadržaja aplikacije [12].

Glavne prednosti *Navigation Componenta* su:

- pojednostavljuje obrasce navigacije
- dobro rukuje s akcijama nazad što znači da korisnik odmah zna kojem je zaslonu bio prije
- automatizira transakciju fragmenata
- *Safe Args* [13] - sigurni prijenos podataka između destinacija
- jednostavno rukovanje dubokim povezivanjem (engl. *deep linking*)
- centralizira i vizualizira navigaciju

Glavne komponente *Navigation Component*-a su: navigacijski graf (engl. *navGraph*), kontroler navigacije (engl. *navController*) i kontejner navigacije (engl. *navHost*). Navigacijski graf je XML resurs koji sadrži sve informacije o navigaciji na jednom centraliziranom mjestu. To podrazumijeva sve destinacije, akcije od početne do krajnje destinacije te informacije o argumentima koje si šalju pojedinom akcijom. Kontejner navigacije ili *navHost* predstavlja prazni kontejner koji prikazuje destinacije iz navigacijskog grafa. Kontroler navigacije je komponenta koja upravlja destinacijama odnosno on je zadužen za pravilno prikazivanje i zamjenjivanje destinacija koje će se prikazivati na kontejneru navigacije [12].



Slika 3.4. *Izgled navigacijskog grafa*

```
<fragment
  android:id="@+id/navigation_host_fragment"
  android:name="androidx.navigation.fragment.NavHostFragment"
  app:defaultNavHost="true"
  app:layout_constraintTop_toTopOf="parent"
  app:navGraph="@navigation/nav_graph"
  android:layout_width="match_parent"
  android:layout_height="match_parent" />
```

Slika 3.5. *Kontejner navigacije (engl. navHost)*

3.5. *Coroutines*

Paralelno izvođenje zadataka unutar aplikacije je vrlo važno i danas je postalo sastavni dio modernih aplikacija. Kotlin je iz tog razloga uveo *Coroutine* koje omogućavaju izvođenje većeg broja zadataka i funkcija bez da izvođenja utječu jedno na drugo odnosno bez međusobnog blokiranja. Čak preko 50 % programera tvrde da su uvidjeli veću optimizaciju i produktivnost nakon što su krenuli koristiti *Coroutine*. Često uz sebe imaju i pridjev ultralagan, što se odnosi na mogućnost pokretanja velikog broja funkcija na jednoj niti (engl. *thread*) i mogućnost zaustavljanja (engl. *suspend*) koje ne blokira cijelu niti na kojoj se funkcija pokreće. Također smanjuje curenje memorije (engl. *memory leaks*). Osim *suspend* mogućnosti, sadrže i mogućnost otkazivanja (engl. *cancel*) koje je omogućeno kroz samu hijerarhiju pokrenutih *Corutine-a* [14].

Najčešće su korištene za mrežne zahtjeve (engl. *network request*), pozive za baze podataka, učitavanje s diska i slično. Kako bi korisniku omogućili ugodno iskustvo korištenja aplikacije, takvi pozivi i akcije se obavljaju izvan glavne niti (engl. *main thread*) kako se ne bi blokiralo korisničko sučelje. Drugi način za izvesti navedene pozive je pomoću povratnih poziva, ali oni često znaju stvarati duplicirani kod, previše ovise jedan o drugome te dolazi do ugnježđivanja i nepreglednog koda [14].

3.6. *Dependency Injection (Hilt)*

U svijetu programskog inženjerstva postoji pojam *SOLID* koji predstavlja pet obrazaca koji služe kako bi objektno orijentirano programiranje učinili održivijim, razumljivijim i jednostavnijim. S obzirom da programski jezik Kotlin i sam spada u grupu objektno-orijentiranih jezika ta pravila vrijede i za njega. *Dependency injection* predstavlja mehanizam kojim se uklanja

međusobna ovisnost između objekata tako što se odmah kroz konstruktor klase ugrađuje objekt klase koji je potreban te tako smanjuje međusobnu ovisnost . Kako današnje aplikacije postaju sve veće i kompleksnije, međusobna ovisnost između objekata postaje teška za nadograđivanje, testiranje i održavanje, ali dovodi i do puno dupliciranog koda što svakako nije dobra praksa. Kao jednostavan primjer toga može se uzeti primjerice klasu *Računalo* koja je ovisna o klasi *Procesor*. Ako bi se instancirao objekt klase *Procesor* unutar klase *Računalo* došlo bi do velike međupovezanosti, izmjenom zahtjeva moralo bi se mijenjati kod na svim mjestima gdje se navedeno koristi. Ubacivanjem instance objekta *Procesor* unutar konstruktora *Računalo* klase postiže se *dependency injection.*

Android pruža razne biblioteke koje olakšavaju uvođenje *dependency injection-a* u projekt te zamjenjuju ručno uvođenje za svaku klasu. Jedna od takvih biblioteka je i *Hilt* koju je razvio *Google*. Izgrađen je na vrhu druge popularne biblioteke *Dagger* kako bi ubrzao vrijeme kompajliranja, pospješio performanse tijekom izvođenja i povećao skalabilnost. Android Studio pruža podršku za *Hilt* [15]. Pomoću ove biblioteke postoji mogućnost da se na jednom mjestu unutar projekta definiraju pravila za generiranje objekata te *Hilt* nakon toga u pozadini obavlja posao generiranja. *Hilt* funkcionira tako da pruža spremnike za svaku Android klasu unutar projekta i automatski vodi računa o životnom ciklusu. Karakteristika za *Hilt* je također i korištenje anotacija koji smanjuju dio koda tako što automatski generiraju potrebne stvari za korištenje. Na slici 3.5. prikazan je način na koji je korišten *Hilt* za generiranje *Firebase* modula unutar projekta.

```
@Module
@InstallIn(SingletonComponent::class)
object FirebaseModule {

  @Provides
  @Singleton
  fun provideFirebaseAuthentication() = FirebaseAuth.getInstance()

  @Provides
  @Singleton
  fun provideFirestore() = FirebaseFirestore.getInstance()

  @Provides
  @Singleton
  fun provideFirebaseStorage() = FirebaseStorage.getInstance()

  @Provides
  @Singleton
  fun provideAuthenticationRepository(
    firebaseAuth: FirebaseAuth, firestore: FirebaseFirestore, firebaseStorage:
    FirebaseStorage, sharedPrefs: SharedPrefs
  ):
    AuthenticationInteractor {
    return AuthenticationInteractor(firebaseAuth, firestore, firebaseStorage, sharedPrefs)
  }

}
```

Slika 3.5. *Firebase modul za dependency injection*

3.7. *Firebase*

Firebase je platforma koja pruža niz raznih alata i servisa prilikom izrade raznih projekata. Nastala je 2011. godine pod vlasništvom tvrtke *Envolve*, ali 2014. godine ga kupuje *Google* pod čijim vlasništvom su i danas. *Firebase* pruža svoje usluge za sve vrste aplikacija i projekata, od Androida, iOS-a, Javascripta, PHP-a pa nadalje. Također sadrži vrlo dobru dokumentaciju te olakšava integriranje *Firebase* alata u projekte [16].

Usluge i servise *Firebase-a* može se podijeliti u tri kategorije:

- Razvoj aplikacija
- Unaprjeđenje kvalitete aplikacije
- Rast poduzeća i aplikacije

U tablici 3.1. su prikazani pojedini alati i kategorije [17].

Tablica 3.1. *Firebase alati*

RAZVOJ APLIKACIJA	Authentication	Omogućuje razne načine autentifikacije unutar aplikacije(lozinka, email, telefonski broj, Facebook, Google i dr.).
	ML Kit	Mobilni SDK koji nudi mogućnost strojnog učenja za Android i iOS platforme-
	Cloud Functions	Omogućuje automatsko pokretanje pozadinskog koda (engl backend) te je kod pohranjen u Googleovom oblaku.
	Cloud Firestore	Fleksibilna i skalabilna NoSQL baza podataka u kojoj jedan red tablice predstavlja jedan dokument u bazi podataka.
	Hosting	Servis za postavljanje web aplikacija na server.
	Real-time database	NoSQL baza koja omogućava spremanje i sinkronizaciju korisnika u stvarnom vremenu.
	Cloud Storage	Alat za postavljanje i preuzimanje datoteka koji podržava veliki broj različitih tipova podataka(fotografije, videozapisi i sl.).
UNAPRJEĐENJE KVALITETE APLIKACIJE	Crashlytics	Alat koji služi da prijavljivanje grešaka u aplikaciji tijekom rada aplikacije. Vrlo koristan alat za otkrivanje grešaka te popravljanje istih.
	Performance Monitoring	Servis koju pomaže pri analiziranju performansi mobilnih aplikacija.
	Test Labs	Omogućuje infrastrukturu unutar oblaka s velikim brojem mobilnih uređaja za testiranje mobilnih

		aplikacija.
RAST PODUZEĆA I APLIKACIJA	In-App Messaging	Omogućuje međusobno slanje poruka između korisnika aplikacije
	Google Analytics	Alat koji pomažu analizirati podatke i ponašanja korisnika tijekom korištenja aplikacije. Pomoću ovog alata može se doći do važnih zaključaka i podataka koji mogu utjecati na određene poslovne odluke.
	Predictions	Omogućuje korištenje strojnog učenja zajedno s analitikom kako bi se predvidjela ponašanja korisnika aplikacije.
	A/B Testing	Pruža alate za lakše testiranje promjena unutar aplikacije.
	Remote Config	Omogućuje promjene aplikacije koja je već u funkciji bez potrebe za razvijanjem i objavljivanjem nove verzije aplikacije.
	Cloud Messaging (FCM)	Servis koji omogućuje sigurnu komunikaciju između uređaja i servera u obliku poruka ili notifikacija.
	Dynamic Links	Alat koji omogućuje jednostavno navigiranje na sadržaje izvan aplikacije pomoću poveznica.
	App Indexing	Pruža mogućnost pokretanja aplikacije preko Google tražilice odnosno ako korisnik ima instaliranu aplikaciju istu može pokrenuti preko traženog sadržaja na tražilici

U izradi ove aplikacije korišteni su:

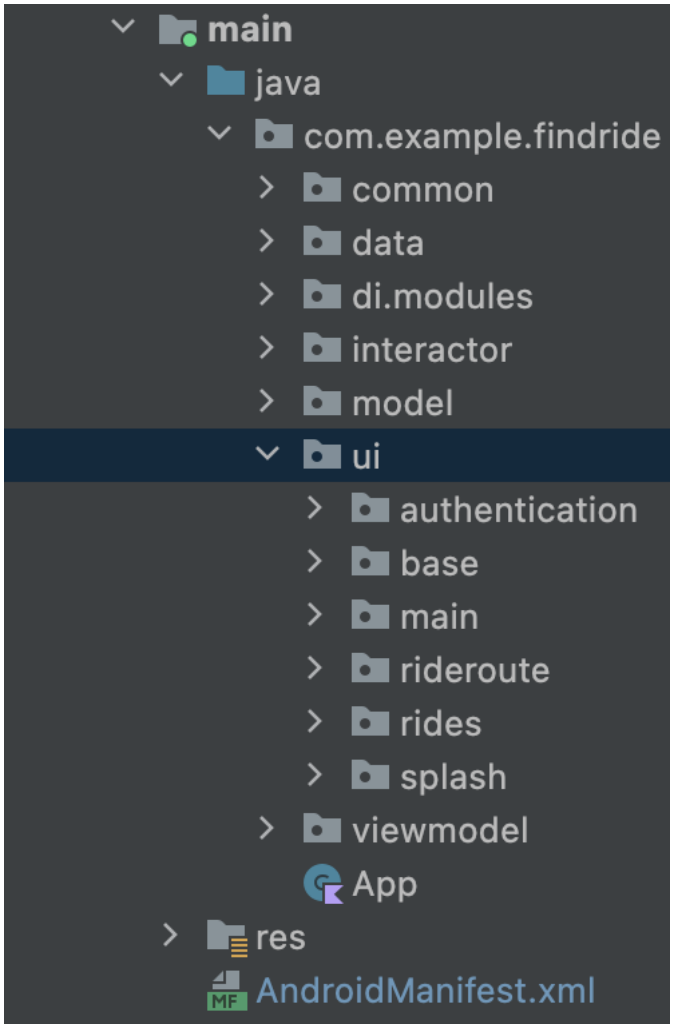
- *Firebase Authentication* – registracija i prijava korisnika unutar aplikacije
- *Cloud Firestore* – baza podataka u stvarnom vremenu
- *Cloud Storage* – spremanje profilnih slika korisnika

4. STRUKTURA I ARHITEKTURA SUSTAVA

U nastavku rada bit će opisani struktura projekta, dijagram toka aplikacije i arhitektura aplikacije.

4.1. Struktura projekta

Projekti i aplikacije danas često postaju sve veće i kompleksnije što može dovesti do zbunjenosti, ali i zatrpanosti datotekama unutar projekta.. Android Studio pruža pakete (engl. *Package)* za razvrstavanje određenih datoteka. Oni predstavljaju zapravo određene direktorije unutar kojeg se spremaju i razvrstavaju datoteke projekta. Android aplikacije bi uvijek trebale biti dobro organizirane što se tiče strukture datoteka jer olakšavaju razumijevanje i čitanje samog projekta te čini kod „čišćim" i lakšim za održavanje.



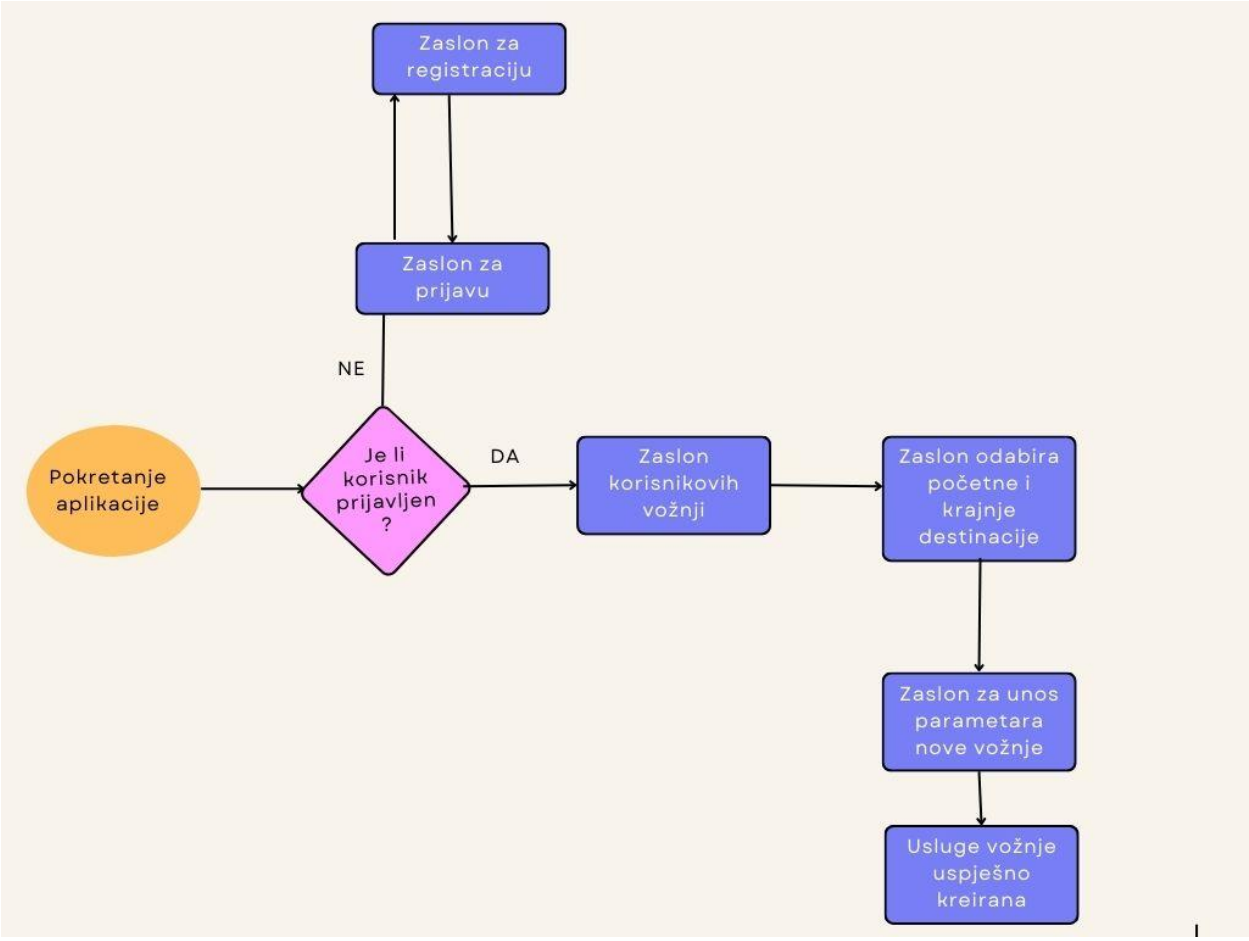
Slika 4.1. *Struktura projekta*

Na slici 4.1. vidljivo da je projekt strukturiran u nekoliko paketa. Glavni paketi su:

- common – sadrži konstante, pomoćne funkcije i pomoćne klase
- data – sadrži podatkovne klase
- di.modules – sadrži module za korištenje i ubrizgavanje ovisnosti (engl. *dependency injection*)
- interactor – sadrži sve datoteke koje se koriste za slanje zahtjeva na server
- model – sadrži sve modele poslovne logike
- ui – sadrži sve datoteke i klase koje su vezane za korisničko sučelje te sadrži dodatne package gdje svaki naziv govori o kojem se zaslonu korisničkog sučelja radi
- viewmodel – sadrži sve *ViewModel* klase koji se koriste unutar projekta

4.2. Dijagram toka

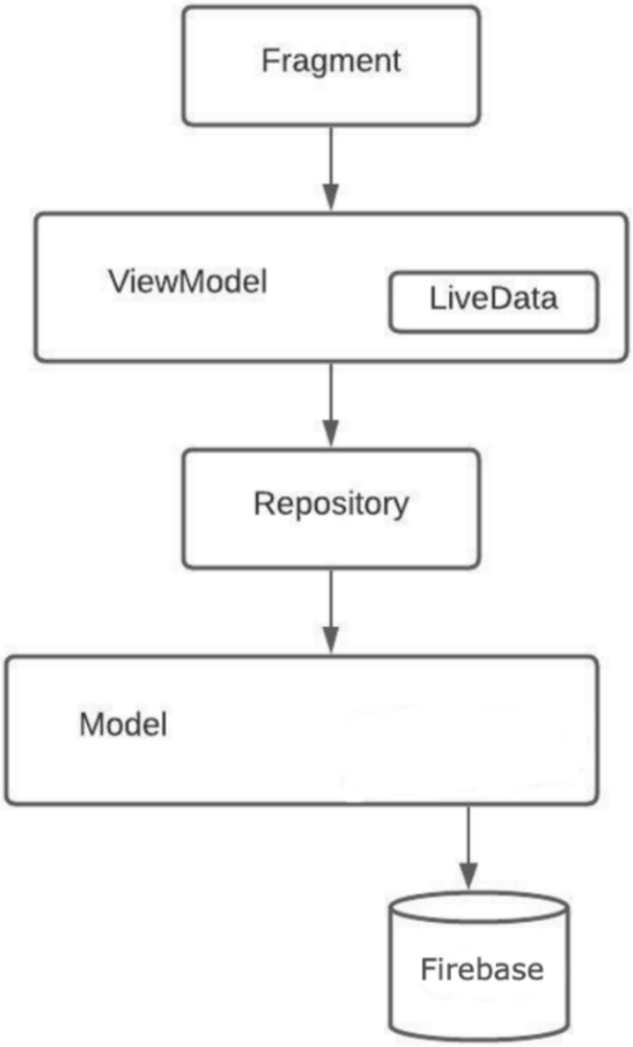
Dijagram toka (engl.*flowchart*) predstavlja grafički prikaz programa koji pojednostavljuje i vizualno prikazuje tijek aplikacije što čini problem ili zadatak programa jednostavnijim za shvatiti i razumjeti [18]. Dijagram toka se može koristiti neovisno o programskom jeziku. Za prikaz se koriste razni geometrijski oblici i likovi od koji svaki ima svoju svrhu. Tako primjerice elipsa predstavlja početak, kraj ili prekid programa dok na primjer romb predstavlja uvjetno grananje. Međusobno su povezani linijama koje predstavljaju tok odnosno prijelaz s jedne komponente na drugu. Linije i oblici zajedno čine smislen cjelinu. Na slici 4.2. prikazan je pojednostavljeni dijagram toka za kreiranje nove vožnje.



Slika 4.2. *Dijagram toka za kreiranje nove vožnje*

4.3. Arhitektura aplikacije

Kao što je ranije rečeno, za izradu aplikacije korišten je MVVM arhitekturalni obrazac. Može se reći da je to trenutno najpopularniji obrazac te velika većina aplikacije izrađena ovakvim načinom. Također, na slici 4.3. je vidljivo da svaka komponenta arhitekture aplikacije ovisi o samo jednoj drugoj komponenti. Fragment ili aktivnost su ovisni o samo o *ViewModel-u*, *ViewModel* je ovisan o *repository-u* ili više njih dok je *repository* ovisan o udaljenom poslužitelju ili bazi podataka u ovom slučaju o Firebase bazi podataka. Ovakav pristup poboljšava strukturu aplikacije, omogućava lakše testiranje i održavanje, ali i olakšava primjerice ako bi drugi razvojni programeri nastavili raditi na ovakvom projektu gdje bi se bez problema mogli snaći i nastaviti razvijati.



Slika 4.3. *Arhitektura aplikacije*

5. PROGRAMSKO RJEŠENJE I IZGLED APLIKACIJE

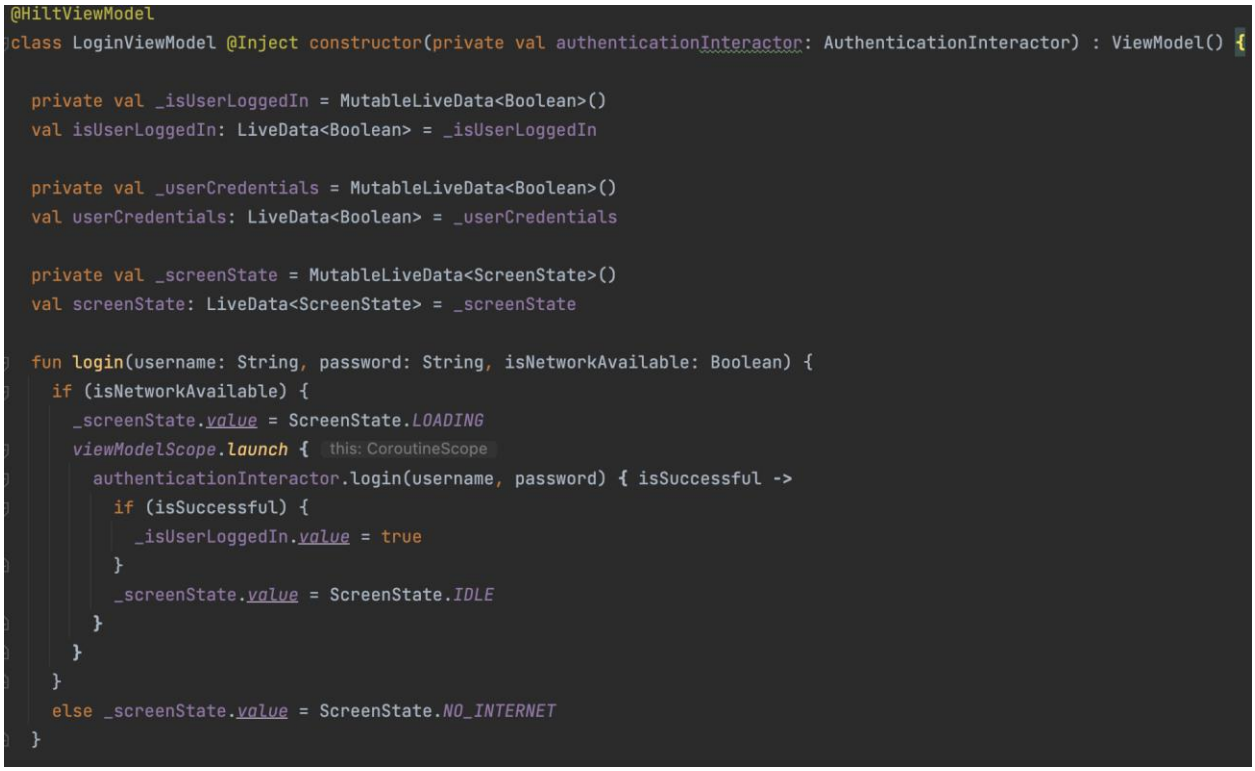
U idućem poglavlju će detaljnije biti objašnjeni način rada aplikacije i glavne programske funkcije kroz alate i tehnologije koje su prethodno navedene u ovom radu.

5.1. Prijava i registracija korisnika

Kako bi se ispunio zahtjev da se korisniku omogući registracija i prijava potrebno je bilo osmisliti samu formu autentifikacije. Kao što je već ranije napomenuto, korišten je *Firebase Authentication* za prijavu i registraciju. Ako se korisnik uspješno registrirao, *Firebase Firestore* sprema u svoju kolekciju *users* dokument s posebnim nazivom za svakog korisnika. F*irebase Authentication* pruža razne načine autentifikacije dok je u ovom slučaju korišten način s email adresom i lozinkom. Kako aplikacija ima mogućnost postavljanja profilne slike korisnika prilikom registracije, korišten je i *Firebase Storage* gdje se spremaju slike i s tog mjesta se povlače ako je registracija uspješna te spremaju u kolekciju unutar *Firebase Firestore-a*. Ako korisnik ne odabere odnosno ne postavi novu sliku profila bit će mu postavljena slika profila koja je zadana od strane aplikacije. Na slici 5.1. je vidljiva funkcija koja se poziva prilikom prijavljivanja korisnika.

```
suspend fun login(username: String, password: String, onResult: (Boolean) -> Unit) {
  withContext(Dispatchers.IO) {  this: CoroutineScope
    try {
      firebaseAuth.signInWithEmailAndPassword(username, password).addOnCompleteListener {  it: Task<AuthResult!>
        if (it.isSuccessful) {
          sharedPrefs.saveUser(firebaseAuth.uid.toString())
          onResult(it.isSuccessful)
          saveUserId()
        }
      }
        .addOnFailureListener {  it: Exception
          onResult(false)
          makeToast(it.message.toString())
        }  ^withContext
    } catch (e: Exception) {
      onResult(false)
      makeToast(e.message.toString())  ^withContext
    }
  }
}
```

Slika 5.1. *Funckija za prijavljivanje korisnika*



```
@HiltViewModel
class LoginViewModel @Inject constructor(private val authenticationInteractor: AuthenticationInteractor) : ViewModel() {

  private val _isUserLoggedIn = MutableLiveData<Boolean>()
  val isUserLoggedIn: LiveData<Boolean> = _isUserLoggedIn

  private val _userCredentials = MutableLiveData<Boolean>()
  val userCredentials: LiveData<Boolean> = _userCredentials

  private val _screenState = MutableLiveData<ScreenState>()
  val screenState: LiveData<ScreenState> = _screenState

  fun login(username: String, password: String, isNetworkAvailable: Boolean) {
    if (isNetworkAvailable) {
      _screenState.value = ScreenState.LOADING
      viewModelScope.launch {  this: CoroutineScope
        authenticationInteractor.login(username, password) { isSuccessful ->
          if (isSuccessful) {
            _isUserLoggedIn.value = true
          }
          _screenState.value = ScreenState.IDLE
        }
      }
    }
    else _screenState.value = ScreenState.NO_INTERNET
  }
```

Slika 5.2. *LoginViewModel*

Kao što je vidljivo funkcija *login* na 5.1. slici ima *suspend* ispred svog naziva. Taj naziv je vezan uz već spomenute *coroutine*. *Suspend* funkcija označava da je funkciju moguće pauzirati i nastaviti kasnije što omogućava čekanje dugotrajnih operacija i zapravo završetak izvršavanja *suspend* funkcija bez blokiranja. Također *withContext(Dispatcher.IO)* [19] označava da će se navedena funkcija izvesti na *IO* niti odnosno neće se izvesti na glavnoj niti što također omogućava izbjegavanje blokiranja te pospješuje performanse aplikacije. Pomoću instance objekta *FirebaseAuth* pozivamo funkciju za prijavu postojećih korisnika koja prima samo korisničku mail adresu i lozinka. Pomoću *dependency injection-a* uvodimo *authenticationInteractor* unutar konstruktora *ViewModel-a* te unutar *viewModelScope* poziva se funkciju za logiranje. Takav način olakšava posao jer se ne mora voditi računa o životnoj aktivnosti *coroutine* te ne postoji strah da će ona ostati negdje u pozadini ako se *ViewModel* uništi već program sam vodi o tome računa. Ako je prijavljivanje korisnika uspješno, mijenja se vrijednost *_isUserLoggedIn* varijable te u fragmentu prateći promjenu vrijednosti te varijable ažurira se korisničko sučelje odnosno korisnik se navigira na idući zaslon. Može se reći da je to ogledni primjer već navedenog MVVM arhitekturalnog obrasca Na slici 5.3. je prikazano promatranje u fragment klasi.

```
viewModel.isUserLoggedIn.observe(viewLifecycleOwner) {  it: Boolean!
  if (it) navigateToMainScreen()
}
```

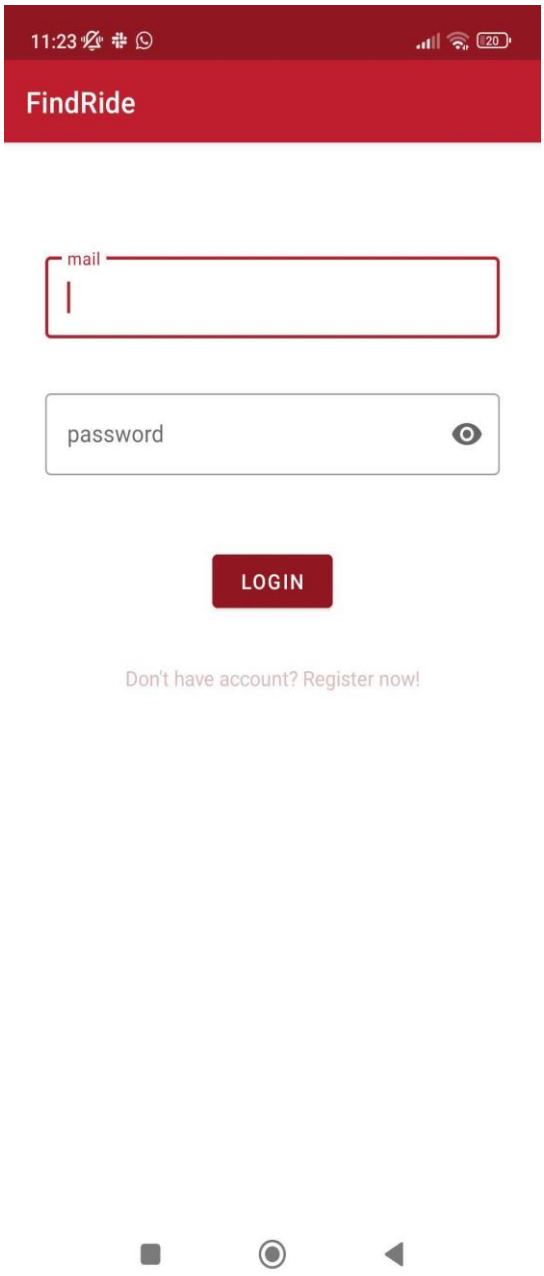
Slika 5.3. *Praćenje i ažuriranje korisničkog sučelja na temelju LiveDate*

Registracija je omogućena na isti način, samo što se poziva funkcija *createUserWithEmailAndPassword()*. Važno je napomenuti da navedena funkcija u pozadini obavlja razne provjere kao što su: ima li lozinka dovoljan broj znakova, postoji li korisnik s takvom emal adresom već registriran u sustav, je li mail adresa ispravna i mnogi drugi što uvelike olakšava implementaciju razvojnom programeru gdje on sam ne mora vršiti svaku provjeru pri pozivu funkcije. Poziv navedene funkcije prikazan je na slici 5.4. dok je logika oko ažuriranja korisničkog sučelja ista kao i kod prijavljivanja korisnika gdje na promjenu određene vrijednosti *LiveData-e* se ažurira korisničko sučelja odnosno u ovom slučaju korisnika se navigira na drugi zaslon.

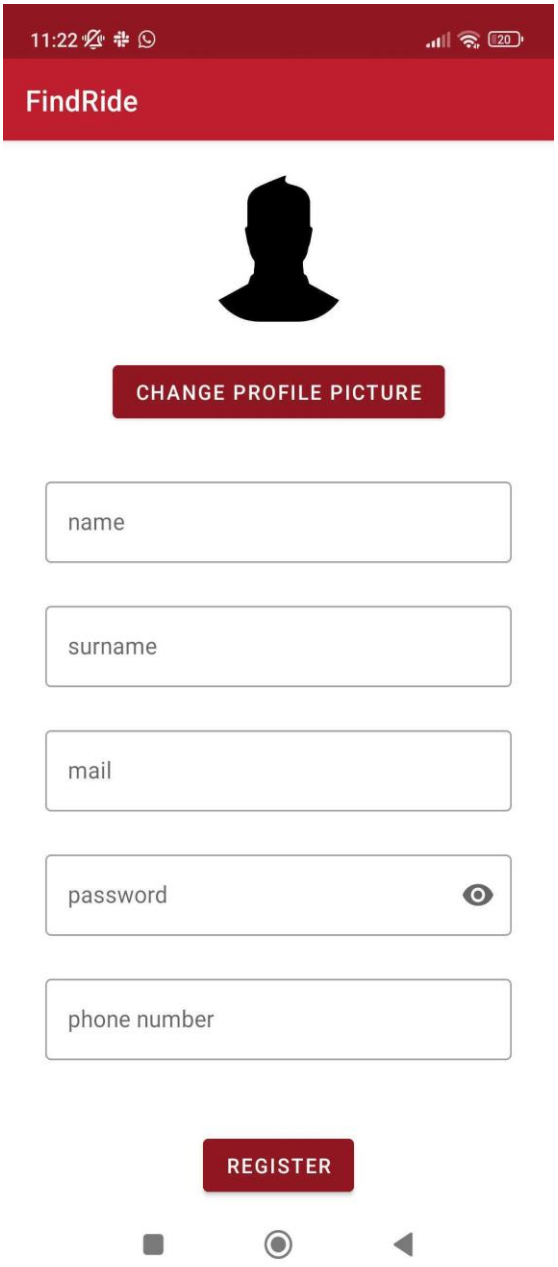
```
firebaseAuth.createUserWithEmailAndPassword(username, password).addOnCompleteListener {  it: Task<AuthResult!>
```

Slika 5.4. *Poziv funkcije za registraciju korisnika*

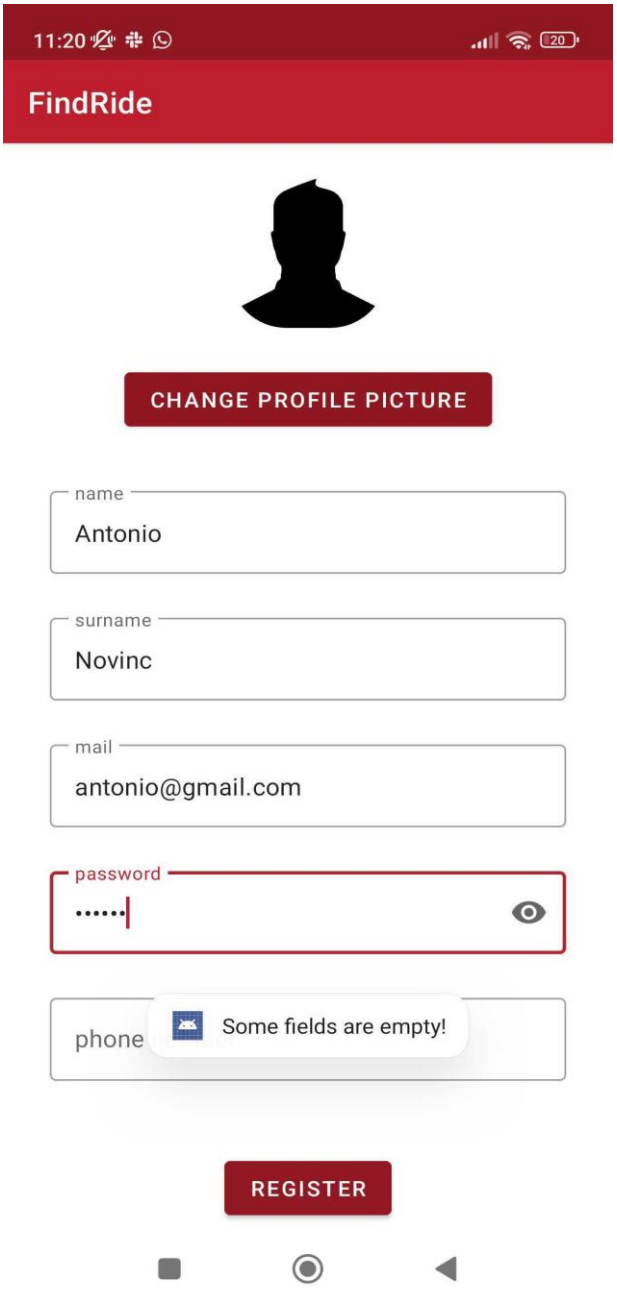
Prilikom registracije i prijave od korisnika se zahtjeva da ispuni sva navedena polja dok kao što je već navedeno postoje razne provjere kako bi autentifikacija bila uspješna koje *Firebase Authentication* provodi. Ako je korisnik primjerice unio lozinku s premalim brojem znakova ili je obavio bilo koji drugi krivi unos aplikacija će ga obavijestiti u obliku *Toast* poruke što je napravio pogrešno. Na slikama 5.5. i 5.6. je prikazan izgled zaslona za prijavu i registraciju korisnika dok je na slikama 5.7. prikazan izgled nepotpunog unosa podataka s praznim poljima te na slici 5.8. je prikazan pogrešan unos, u ovom slučaju lozinka sadrži premalo znakova.



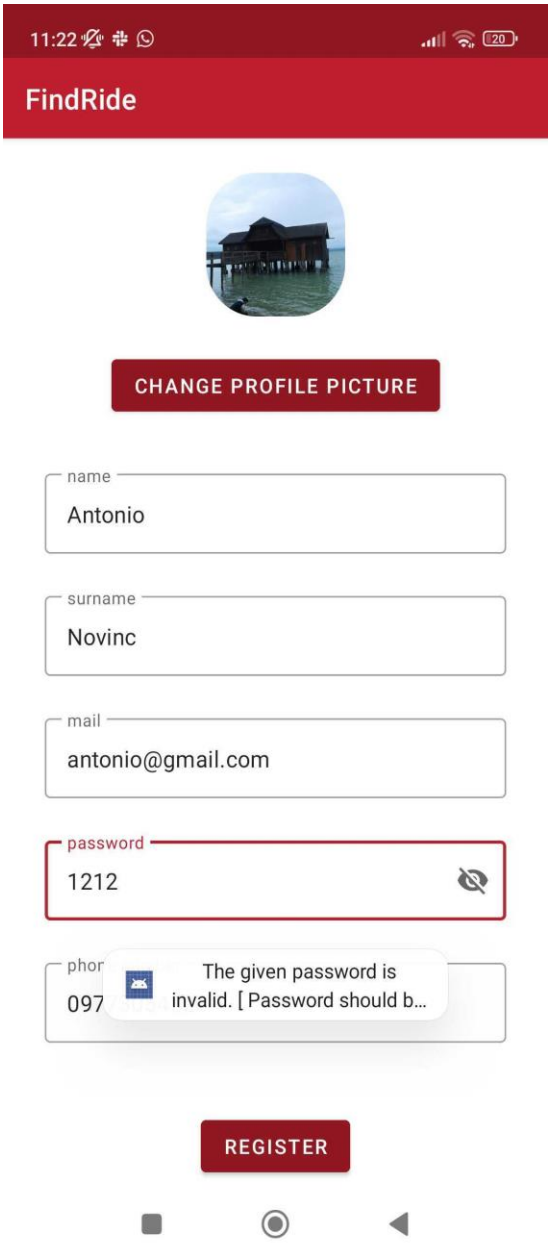
Slika 5.5. *Zaslon za prijavu korisnika*



Slika 5.6. *Zaslon za registraciju korisnika*



Slika 5.7. *Prikaz nepotpunog unosa podataka*



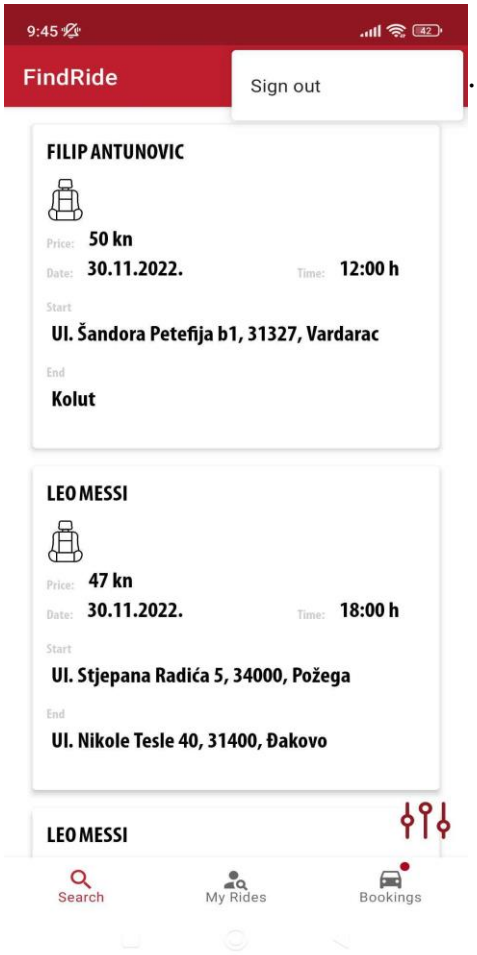
Slika 5.8. *Prikaz unosa prekratke lozinke*

Ako je korisnik izašao iz aplikacije, a ostao je prijavljen odnosno nije se odjavio, aplikacija prilikom novog pokretanja ne zahtijeva ponovno prijavljivanje već odmah vodi korisnika na glavni zaslon aplikacije. Navedeno je postignuto pomoću *SharedPreferences* gdje se prilikom prijave korisnika sprema ID korisnika koji se onda prilikom *splash* zaslona provjera te na temelju toga se postavlja početni zaslon aplikacije. *SharedPreferences* [20] omogućava spremanje primitivnih tipova podataka gdje se određenoj vrijednosti pridodaje određeni ključ pomoću kojeg se onda

dohvaćaju vrijednosti. *Splash* zaslon je prvi zaslon koji se prikaže korisniku nakon pokretanja aplikacije obično dok se još aplikacija učitava. Prilikom prijavljivanja korisnika *SharedPreferences* koristi ključ *user* gdje sprema ID korisnika. Ako se korisnik odjavio sprema se prazan *String* te prema tome upravlja koji će zaslon biti prikazan, ako pomoću ključa se dohvati prazan *String* početni zaslon je zaslon za prijavu, a u suprotnom početni zaslon je glavni zaslon aplikacije.

5.2. Traženje i pružanje usluge prijevoza

Glavna funkcionalnost aplikacije je jednostavno traženje prihvatljive usluge prijevoza te nuđenje iste. Nakon što se korisnik uspješno prijavi ili registrira aplikacija ga navigira do glavnog zaslona. Glavni zaslon sastoji se od donje navigacije, kontejnera unutar kojeg se izmjenjuju zasloni te *menu-a* unutar gornje alatne trake koji sadrži opciju za odjavu korisnika. Ako se korisnik odjavi aplikacija ga navigira na zaslon za prijavu. Na slici 5.9. prikazan je glavni zaslon.

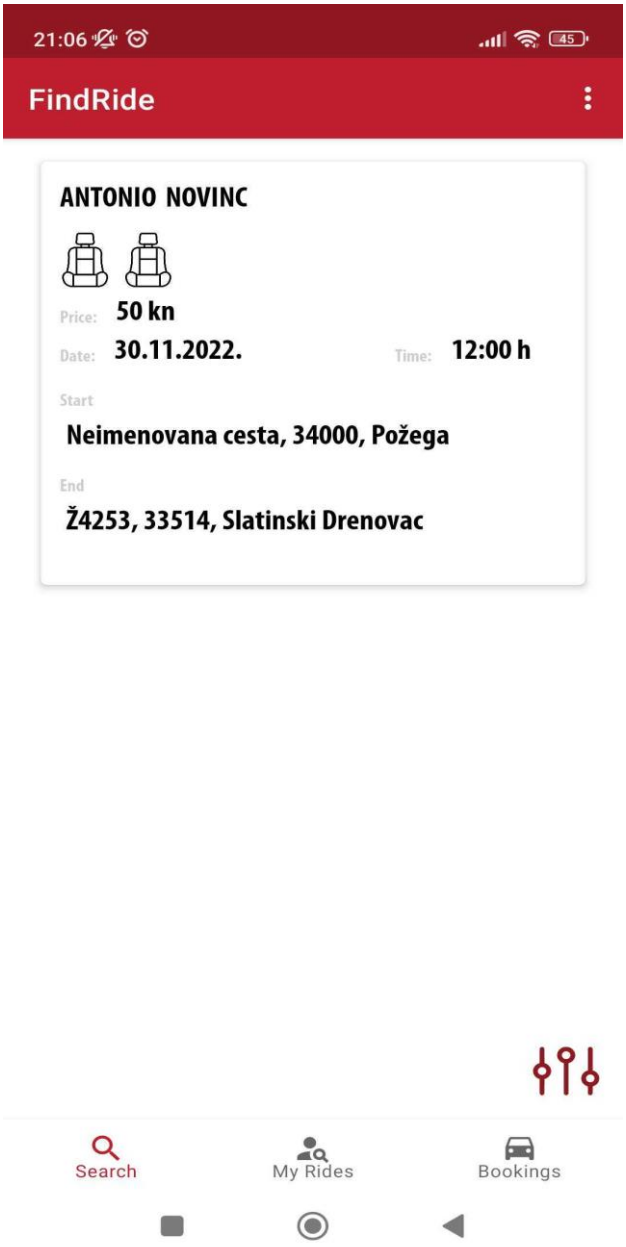


Slika 5.9. *Glavni zaslon aplikacije*

Pomoću donje navigacije moguće je izmjenjivati tri zaslona unutar kontejnera, a to su: zaslon koji sadrži sve vožnje koje je moguće rezervirati (*Search*), zaslon s vožnjama korisnika (*My Rides*) i zaslon koji prikazuje rezervirane vožnje i vožnje na koje se očekuje korisnikov odgovor (*Bookings)*. Svaka vožnja je prikazana karticom koja sadrži osnovne informacije kao što su:

- autor vožnje
- broj slobodnih mjesta koja su prikazana vizualno pomoću slike sjedala
- datum vožnje
- vrijeme vožnje
- cijena vožnje
- početna lokacija vožnje
- krajnja lokacija vožnje

Prilikom prikaza svih vožnji vodilo se računa o tome da prikazuje samo aktivne vožnje odnosno one vožnje zakazane s datumom koji tek treba doći. Na slici 5.10. prikazan je izgled kartice dostupne vožnje dok je na slici 5.11. prikazana pomoćna funkcija pomoću koje se prikazuju samo aktivne vožnje. Funkcija radi tako da uzima dan, mjesec i godinu vožnje te uspoređuje s trenutnim datumom te ako je cjelokupni datum već prošao vožnja se uklanja iz liste. Na slici 5.12. prikazana je funkcija koja dohvaća sve vožnje drugih autora. Funkcija dohvaća sve dokumente iz rides kolekcije te ako je ID autora jednak ID-u prijavljenog korisnika onda se dokument uklanja iz liste vožnji. Funkcija za dohvaćanje vožnji prijavljenog korisnika funkcionira identično samo što ako je ID autora vožnje jednak ID-u prijavljenog korisnika onda se vožnja dodaje u listu.



Slika 5.10. *Izgled kartice dostupne vožnje*

```
fun isDateValid(day: Int, month: Int, year: Int): Boolean {
  val _day = Calendar.getInstance().get(Calendar.DAY_OF_MONTH)
  val _month = Calendar.getInstance().get(Calendar.MONTH) + 1
  val _year = Calendar.getInstance().get(Calendar.YEAR)
  if (year < _year) return false
  if (month < _month) return false
  else if (year == _year && month == _month && day < _day) return false
  else return true
}
```

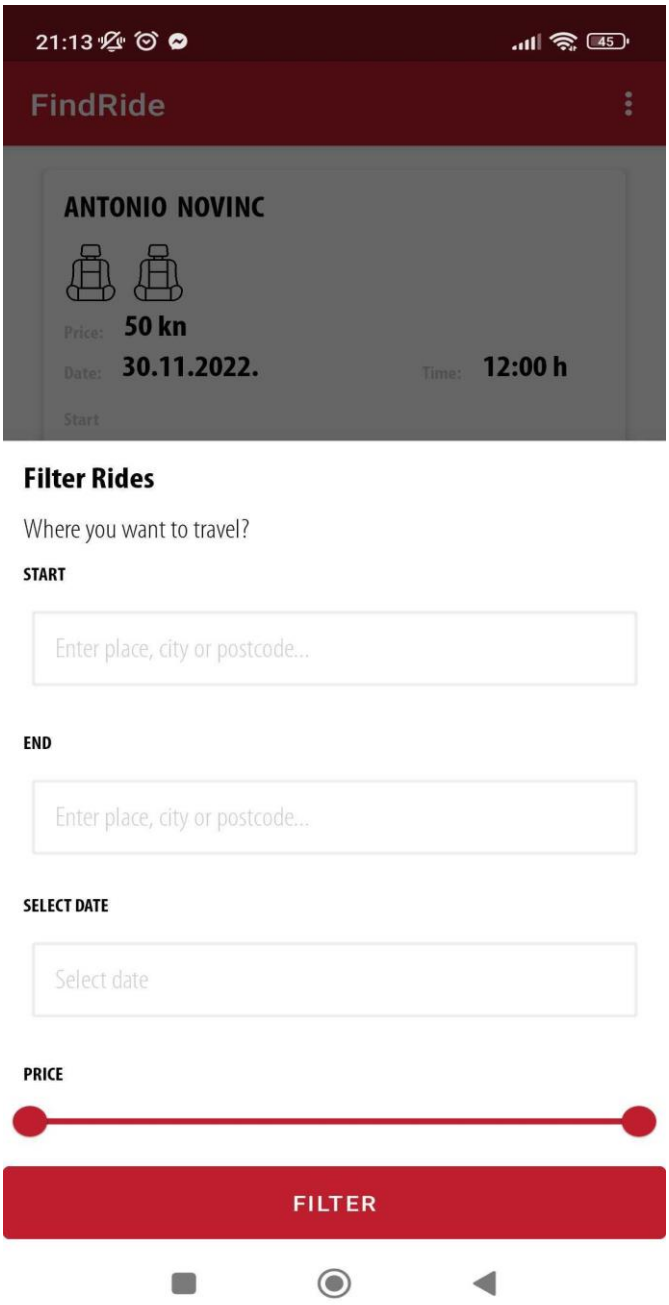
Slika 5.11. *Funkcija za provjeru valjanosti datuma*

```
override suspend fun getAllRides() = withContext(Dispatchers.IO) {  this: CoroutineScope
  suspendCoroutine<List<Ride>> { continuation ->
    firestore.collection(RIDES).get()
      .addOnFailureListener {  it: Exception
        continuation.resumeWithException(it)
      }
      .addOnSuccessListener { documents ->
        val rides = arrayListOf<Ride>()
        val _rides = documents.toObjects<Ride>()
        rides.addAll(_rides)
        rides.sortWith(compareBy({ it.year }, { it.month }, { it.day }, { it.time }))
        rides.removeIf { ride ->
          ride.authorId == sharedPreferences.getUserId()
        }
        rides.removeIf {  it: Ride
          !isDateValid(it.day, it.month, it.year)
        }
        continuation.resume(rides)
      }
  }
}
```

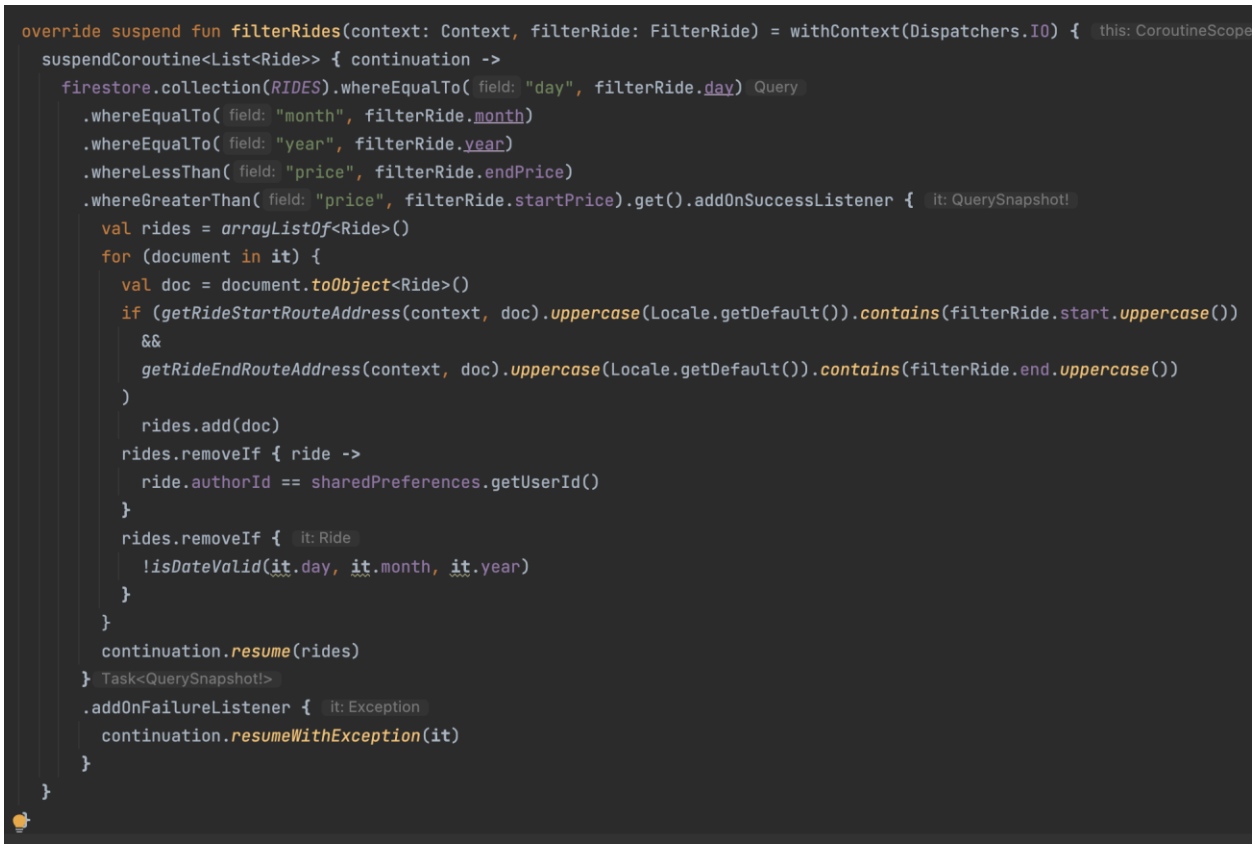
Slika 5.12 *Funkcija za dohvaćanje vožnji*

Listu svih dostupnih vožnji moguće je filtrirati po datumu, početnoj i krajnjoj adresi te rasponu cijene vožnje .Klikom na gumb u donjem desnom krugu zaslona otvara se dijalog s opcijama za filtriranje. Aplikacija funkcionira tako da je korisnik dužan unijeti jedino datum dok su ostale vrijednosti opcionalne. Također, početne i krajnje lokacije je moguće pretražiti po poštanskom

broju ili nazivu mjesta. Na slici 5.14. je prikazan dijalog za filtriranje vožnji dok je na slici 5.15. prikazana funkcija za filtriranje vožnji.



Slika 5.14. *Dijalog za filtriranje vožnji*

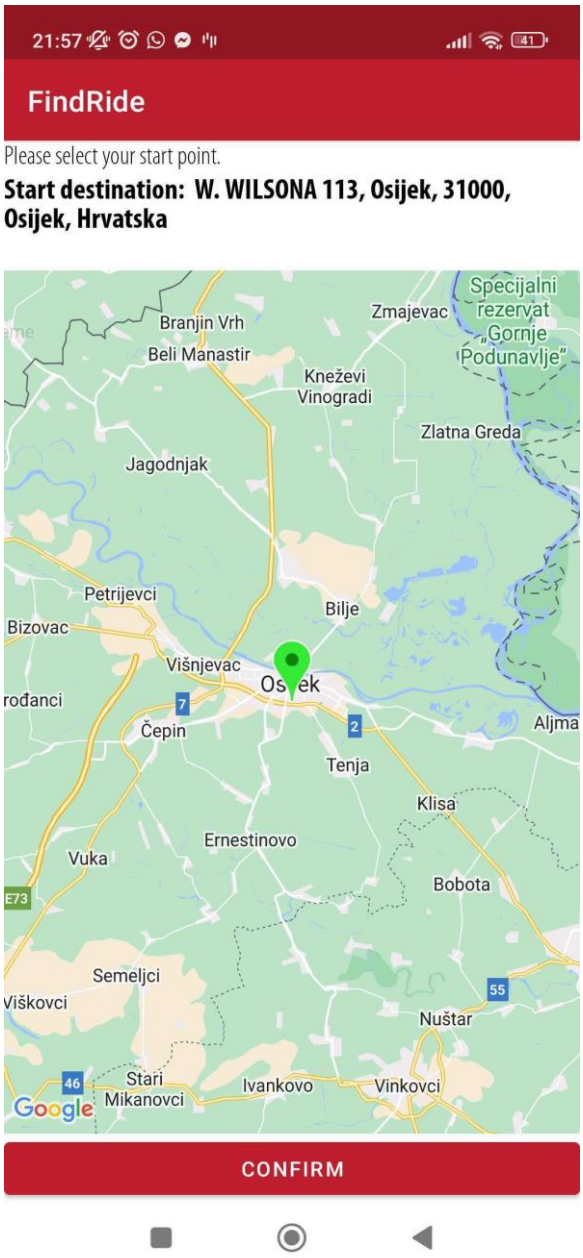


Slika 5.15. *Funkcija za filtriranje vožnji*

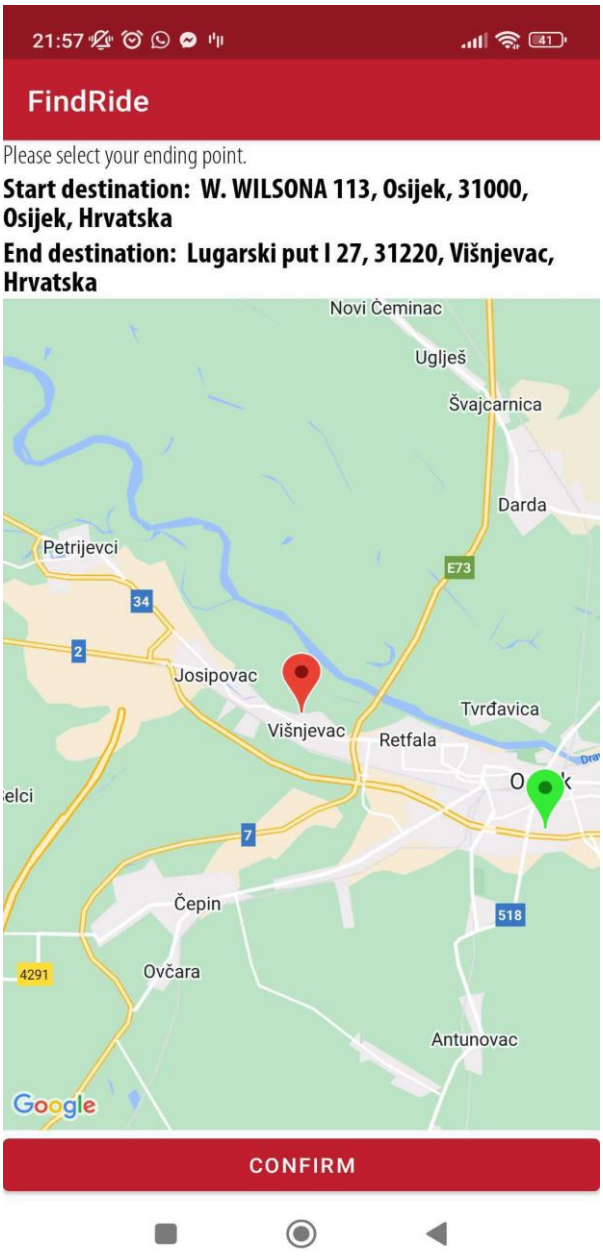
Funkcija prima objekt podatkovne klase *FilterRide* pomoću čijih parametara pretražuje *rides* kolekciju *Firebase Firestore-a* koja sadrži sve vožnje. Pristupa kolekciji i traži dokument čija su polja (engl. *field*) jednaka atributima predanog objekta, dohvaća dokument te ga pretvara u objekt *Ride* i sprema u listu. Također lokacije nisu osjetljive na velika i mala slova pa je svejedno kako korisnik unese naziv lokacije u dijalog za filtriranje.

Na slici 5.16. prikazan je drugi zaslon s donje navigacije pod nazivom *My Rides*. Drugi zaslon prikazuje samo korisnikove vožnje koje je on ponudio kao uslugu te su prikazane kao i sve vožnje prethodno opisane, pomoću kartice. Korisnikove vožnje se prikazuju identično kao i s funkcijom na slici 5.12. samo što prilikom dohvaćanja svih dokumenata iz kolekcije *rides* s *Firebase Firestore-a* uspoređuje se zadovoljava li se uvjet da je korisnikov ID jednak ID-u autora vožnje. S tog zaslona klikom na gumb korisnik može kreirati novu vožnju. Prilikom kreiranja nove vožnje korisnik je dužan unijeti početnu lokaciju vožnje te ga se navigira na kartu na kojoj je potrebno označiti početnu lokaciju. Nakon potvrde da je to početna adresa korisnika se navigira na drugi zaslon na kojem je potrebno označiti krajnju lokaciju vožnje. Na slikama 5.17. i 5.18. su prikazani zasloni za određivanje početne i krajnje lokacije prilikom kreiranja nove vožnje.

Slika 5.16. *Prikaz zaslona s korisničkim vožnjama*



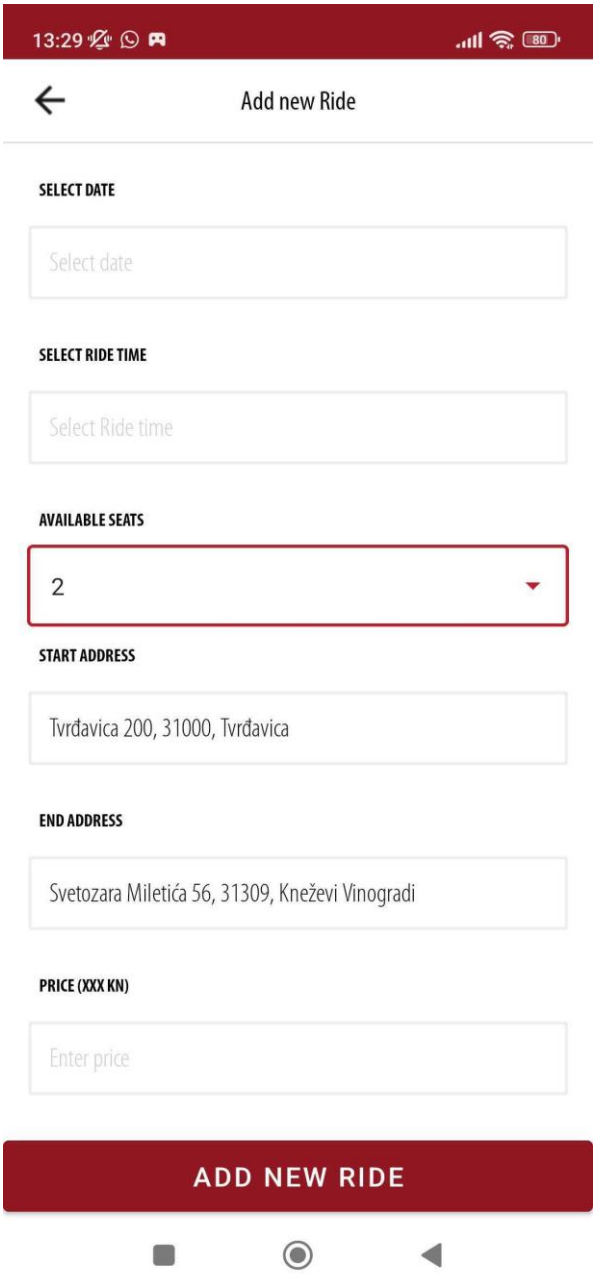
Slika 5.17. *Izbor početne lokacije vožnje*



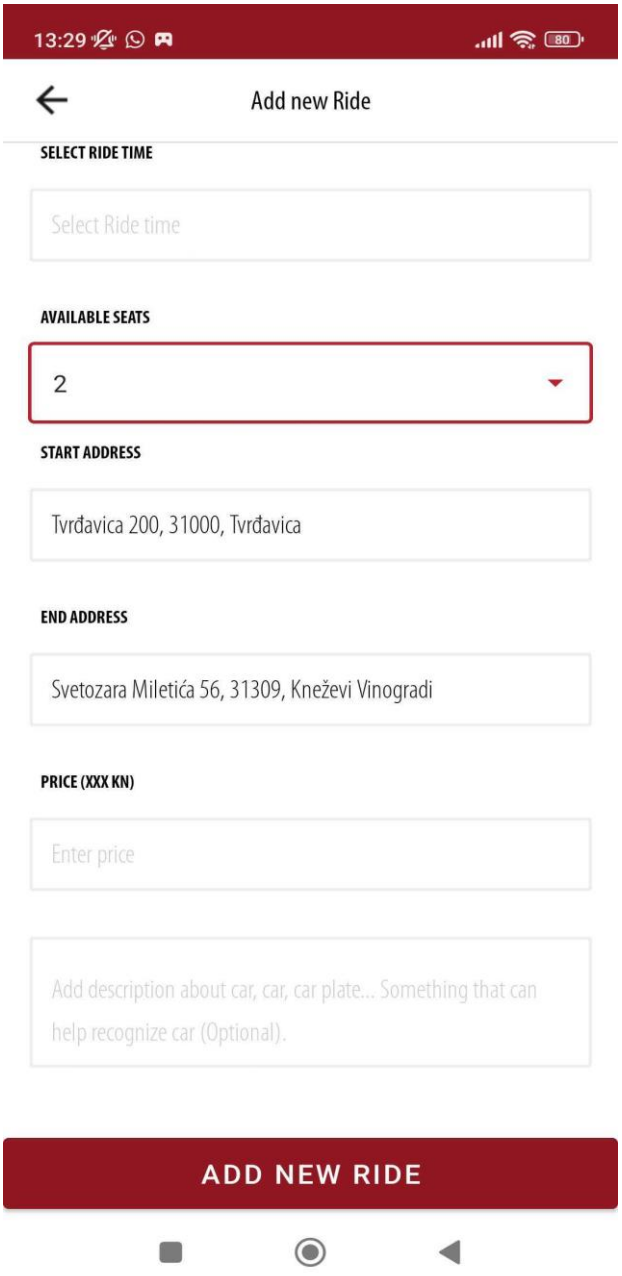
Slika 5.18. *Izbor krajnje lokacije vožnje*

Za postizanje navedenog korišten je *Google Maps API* koji radi na taj način da se označavanjem na karti dobivaju točne koordinate na karti te se pomoću pomoćnih funkcija enkodiraju u to da se dobije smislena adresa [21]. Prikaz karte je interaktivan odnosno omogućava korisniku kretanje po karti, približavanje te udaljavanje prikaza karte. Prilikom kreiranja i spremanja u *Firestore* bazu podataka spremaju se vrijednosti koordinata te se te vrijednosti ponovno enkodiraju u smislenu adresu prilikom dohvaćanja i prikazivanja na korisničko sučelje. Nakon što je korisnik odredio početnu i krajnju adresu aplikacija ga navigira na zaslon na kojem je potrebno unijeti datum

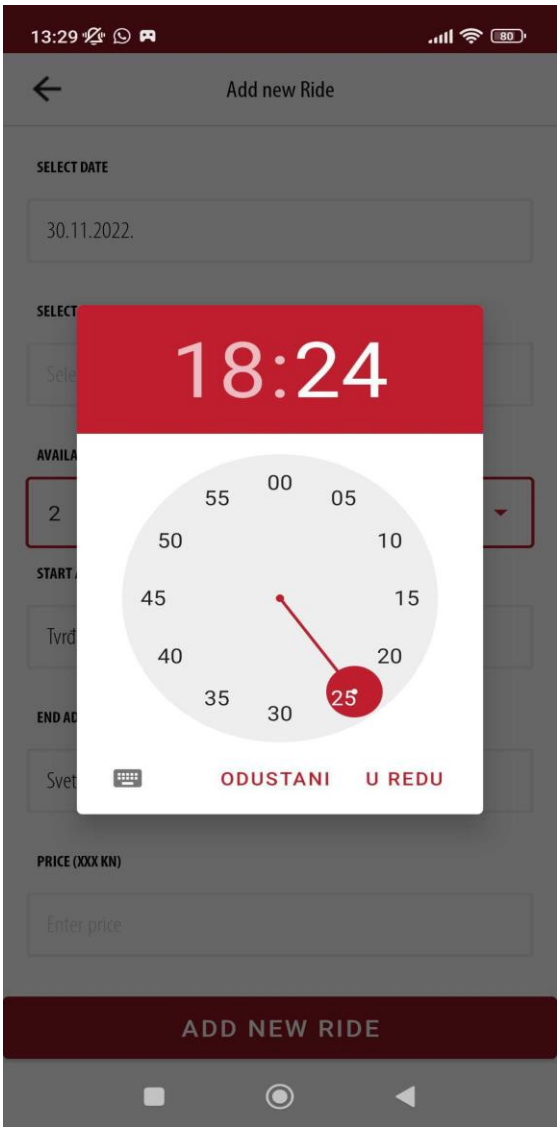
vožnje, vrijeme vožnje, broj slobodnih mjesta i cijenu vožnje. Od korisnika se traži da sve navedene informacije o vožnji mora unijeti inače nije moguće kreirati vožnju. Također, korisnik ima opcionalnu mogućnost unijeti dodatne informacije za koje smatra da su bitne za vožnju ili primjerice lakše snalaženje. Ideja je da korisnik može unijeti primjerice vrstu i tip vozila, broj registracijske oznake i slično.



Slika 5.19. *Izgled zaslona za kreiranje vožnje*



Slika 5.20. *Izgled zaslona za kreiranje vožnje*

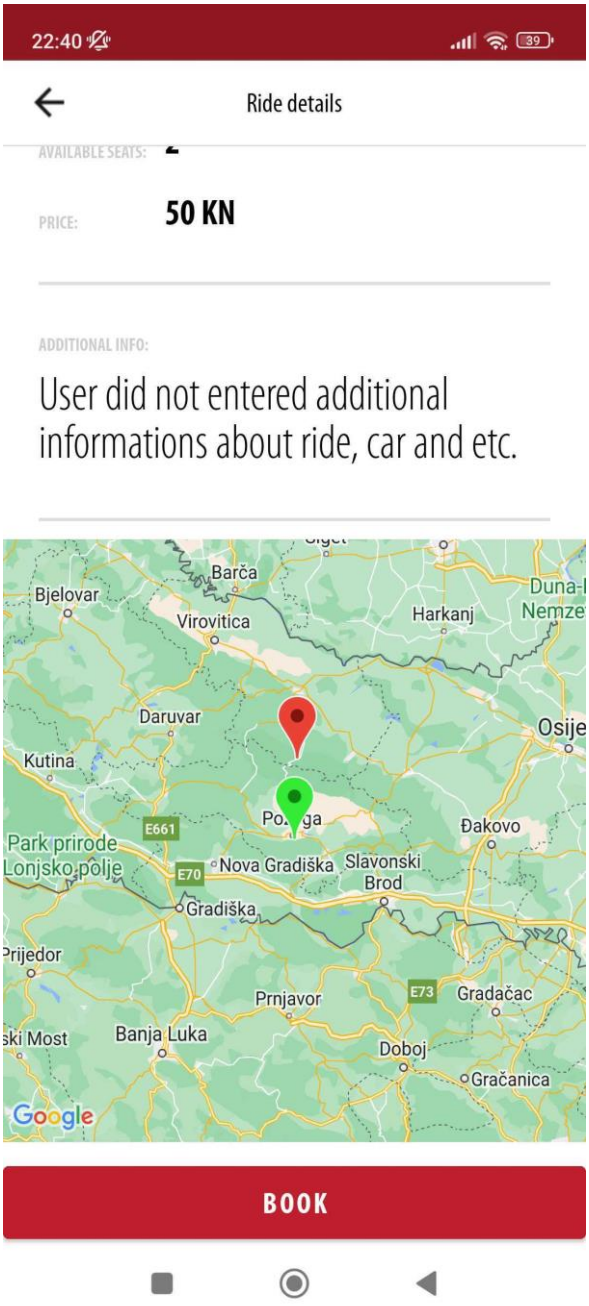


Slika 5.21. *Unos vremena vožnje*

Slika 5.22. *Unos datuma vožnje*

Kada je prijevoz kreiran vidljiv je kao kartica unutar liste prijevoza. Klikom na karticu prijevoza otvaraju se detalji vožnje gdje je vidljiva slika profila autora vožnje, ime i prezime autora, datum i vrijeme vožnje, cijena vožnje te dodatne informacije ako ih je autor vožnje uvrstio. Također na karti su prikazani početna i krajnja lokacija vožnje gdje je zelenom ikonom označena početna dok je crvenom označena krajnja lokacija. Na detalje vožnje moguće je doći klikom i na vlastitu vožnju te na vožnje drugih autora. Razlika je u tome što detalji vlastite ponuđene vožnje imaju mogućnost i brisanja vožnje te sadrže listu putnika kojima je odobrena vožnja dok klikom na detalje vožnje drugih autora imamo opciju za rezervaciju vožnje ako već nismo rezervirali tu istu vožnju. Na slikama 5.23. i 5.24. prikazan je izgled zaslona detalja vožnje.

Slika 5.23. *Detalji vožnje*



Slika 5.24. *Detalji vožnje*

Na slikama 5.25, 5.26, 5.27., 5.28. su prikazane funkcije pomoću kojih se prikazuju detalji vožnje. Prilikom klika na karticu pomoću već spomenutih *SafeArgs* se šalje ID vožnje gdje se onda pretražuje kolekcija *rides* i traži se dokument koji ima isti ID koji je poslan putem *SafeArgs*.

```
private val ridesAdapter by lazy {
  context?.let { RidesAdapter(it) { ride -> showRideDetails(ride) } }
}

private fun showRideDetails(ride: Ride) {
  val action = MainScreenDirections.actionMainScreenToRideDetailsFragment(ride.id,  fromWhichFragment: 1)
  findNavController().navigate(action)
}
```

Slika 5.25. *Slanje ID-a vožnje prilikom klika na karticu*

```
override suspend fun getRideDetails(id: String) = withContext(Dispatchers.IO) {  this: CoroutineScope
  suspendCoroutine<Ride> { continuation ->
    firestore.collection(RIDES).whereEqualTo(ID, id).get()
      .addOnFailureListener {  it: Exception
        continuation.resumeWithException(it)
      }
      .addOnSuccessListener { documents ->
        val _rides = documents.toObjects<Ride>()
        val ride = _rides[0]
        continuation.resume(ride)
      }
  }
}
```

Sliika 5.26. *Funkcija za dohvaćanje detalja vožnje*

```
fun getRide(id: String) {
  _screenState.value = ScreenState.LOADING
  viewModelScope.launch {  this: CoroutineScope
    _ride.postValue(ridesUseCaseImpl.getRideDetails(id))
    _screenState.value = ScreenState.IDLE
  }
}
```

Slika 5.27. *RideDetailsViewModel*

```
viewModel.getRide(args.rideId)
```

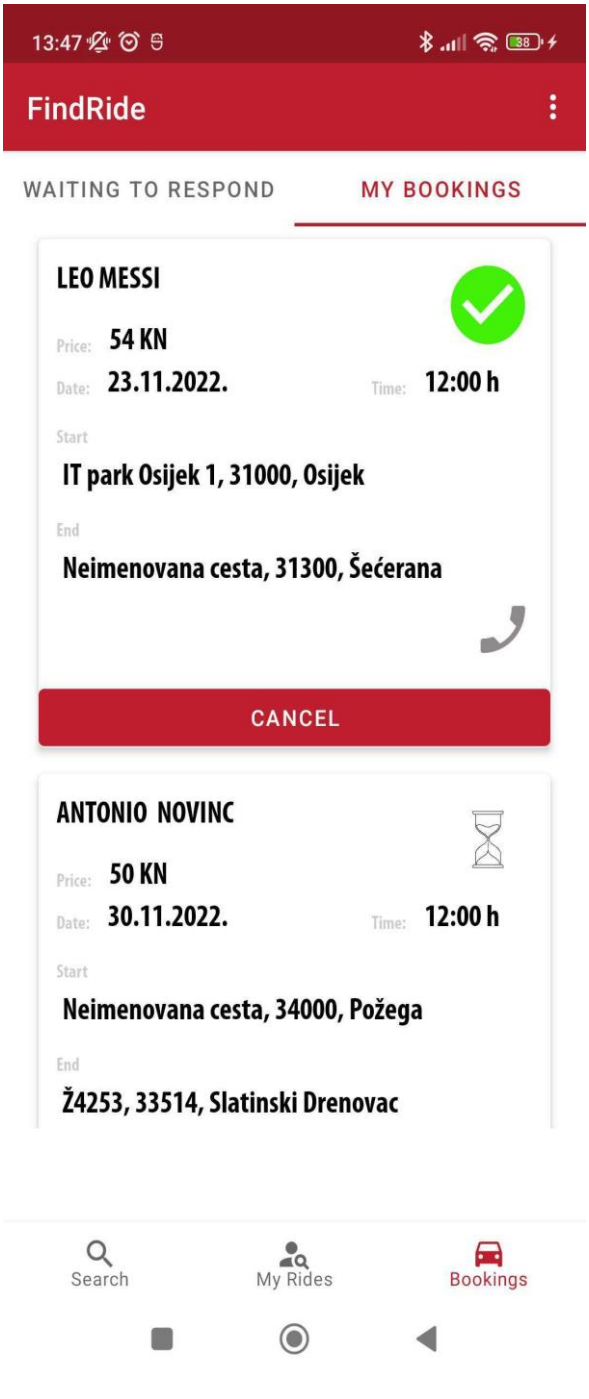
Slika 5.28. *Dohvaćanje SafeArgs*

5.3. Rezervacija vožnji

Zadnji zaslon na donjoj navigaciji sastoji se od dvije kartice (engl. *tabs*) gdje se moguće kretati kliznim pokretom prsta. Prva kartica unutar prikazuje rezervacije vožnje na koje korisnik mora odgovoriti odnosno rezervacije drugih korisnika za vožnje koje je trenutno prijavljeni korisnik kreirao. Druga kartica prikazuje vožnje koje je prijavljeni korisnik rezervirao kod drugih

korisnika. On prikazuje vožnje koje nisu odobrene, koje su odobrene i koje su još u procesu odobrenja. Odobrene i vožnje u procesu odobrenja korisnik može otkazati odnosno može otkazati njihove rezervacije. Ako su rezervacije vožnje prihvaćene korisnik može kontaktirati autora vožnje pozivom jednostavnim klikom na ikonu telefona koja vodi iz aplikacije na poziv korisnika. Kada korisnik ima neodgovorene zahtjeve za svoje vlastite vožnje pojavljuje se značka (engl. *badge*) na donjoj navigaciji koja ukazuje da postoje još neodgovorene vožnje. Na slici 5.29. je prikazana kartica s neodgovorenim zahtjevima na rezervaciju od drugih korisnika te je također vidljiva značka u donjem desnom kutu u donjoj navigaciji ukoliko postoje neodgovoreni zahtjevi. Na slici 5.30. su prikazane svi zahtjevi koje je korisnik poslao drugim korisnicima za rezervaciju njihovih vožnji. Također je vidljivo da su različitim slikama prikazana različita stanja pojedinih zahtjeva (odobrena rezervacija, odbijena rezervacija i neodgovorena rezervacija).

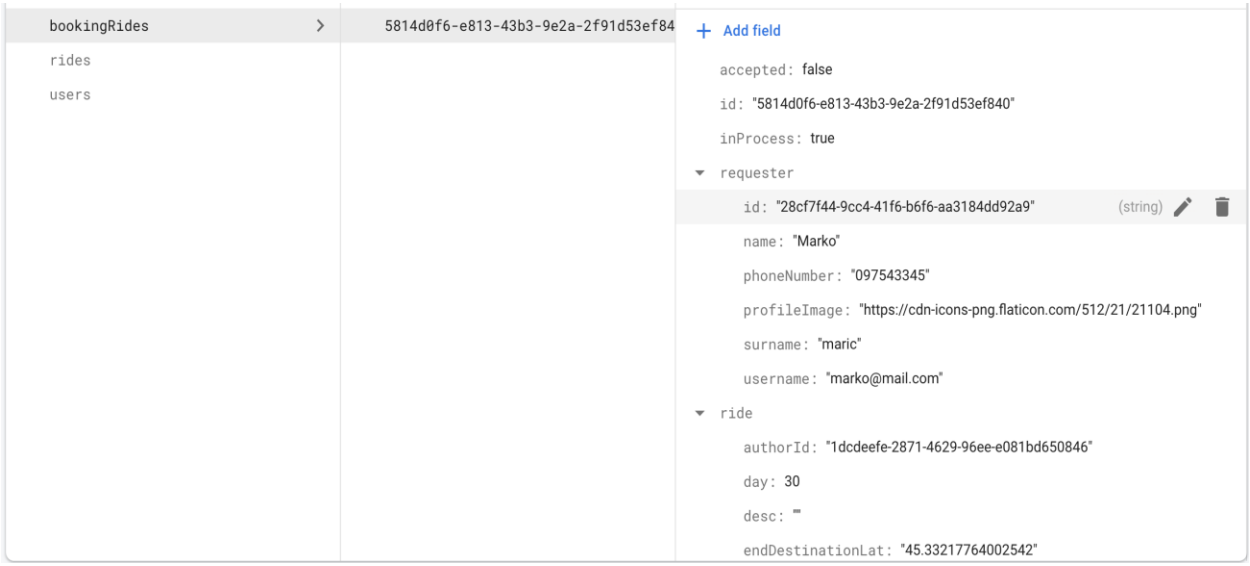
Slika 5.29. *Zaslon za odobravanje rezervacija*



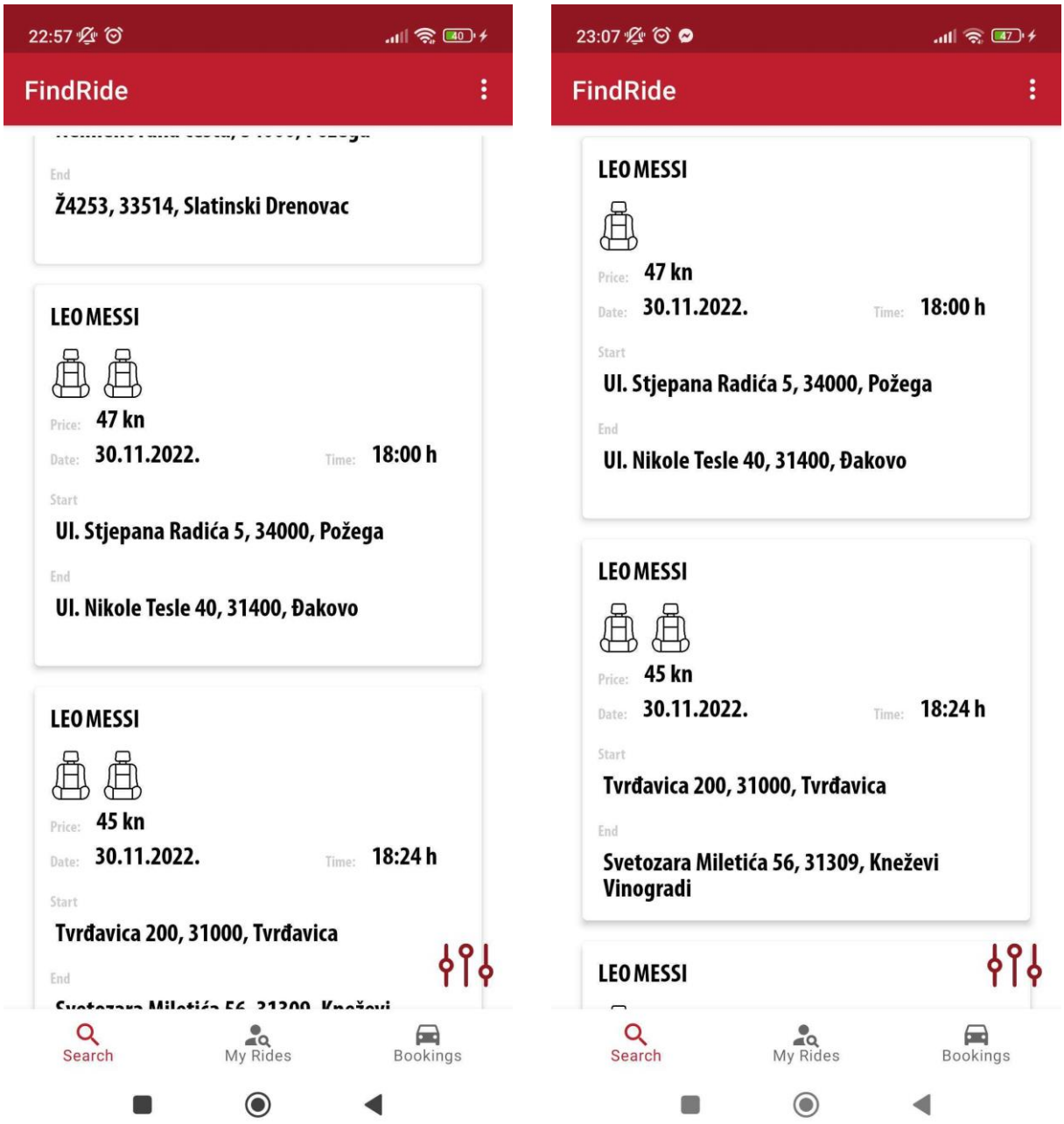
Slika 5.30. *Zaslon korisničkih rezervacija*

Proces rezervacije je vrlo jednostavan, korisnik klikom na karticu vožnje otvara detalje vožnje gdje ima opciju rezervacije klikom na gumb. Klikom na taj gumb pravimo rezervaciju te se autoru vožnje prikazuje neodgovoreni zahtjev rezervacije na zaslonu kao što je prikazano na slici. Kada se napravi rezervacija, odnosno zahtjev za rezervaciju automatski se rezervira jedno slobodno

sjedalo kako se ne bi otišlo u negativnu stranu i dopustilo više rezervacije nego što postoji slobodnih mjesta. Mjesto ostaje rezervirano sve dok korisnik ne odgovori na zahtjev za rezervaciju odnosno ako odbije zahtjev drugog korisnika rezervacije se uklanja te se zauzeto mjesto oslobađa dok potvrdom na rezervaciju mjesto ostaje zauzeto za tog korisnika. Kada je autor vožnje prihvatio zahtjev za rezervaciju od drugog korisnika, taj isti drugi korisnik se pojavljuje u listi putnika na zaslonu detalja vožnje. Također ako nema slobodnih mjesta, korisnik ne može rezervirati vožnju te nam aplikacija javlja da za navedenu vožnju nema više slobodnih mjesta. Kada korisnik napravi zahtjev za rezervaciju za vožnju odnosno klikne na gumb za rezervaciju kreira se novi dokument u kolekciji *bookingRides* na *Firebase Firestore-u*. On sadrži informacije o vožnji te podnositelju zahtjeva. Osim toga sadrži polja (engl. *fields*) pod nazivom *inProcces* i *accepted* što ukazuju je li autor vožnje odgovorio na zahtjev za rezervaciju i koji je status rezervacije odnosno je li autor vožnje odobrio ili odbio zahtjev. Pomoću navedenih polja lako u programskom rješenju se upravlja s podacima i poznat je status rezervacije te prema tome se zna na kojem zaslonu odnosno kartici se prikazuju. Na slici 5.31. prikazan je novokreirani dokument kada se napravi rezervacije vožnje. Također isti korisnik ne može više puta rezervirati istu vožnju te ima opciju za otkazati svoju rezervaciju gdje se također vodi računa o ažuriranju slobodnog broja mjesta ako je rezervacije otkazana.

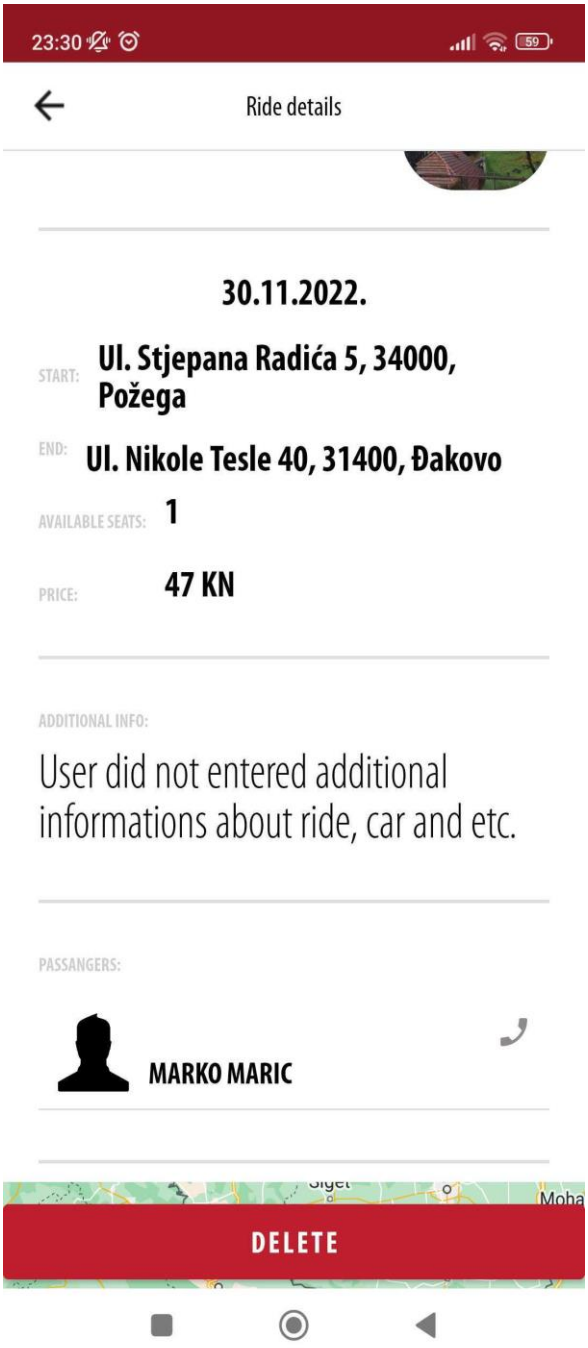


Slika 5.31. *Izgled dokumenta unutar bookingRides kolekcije*

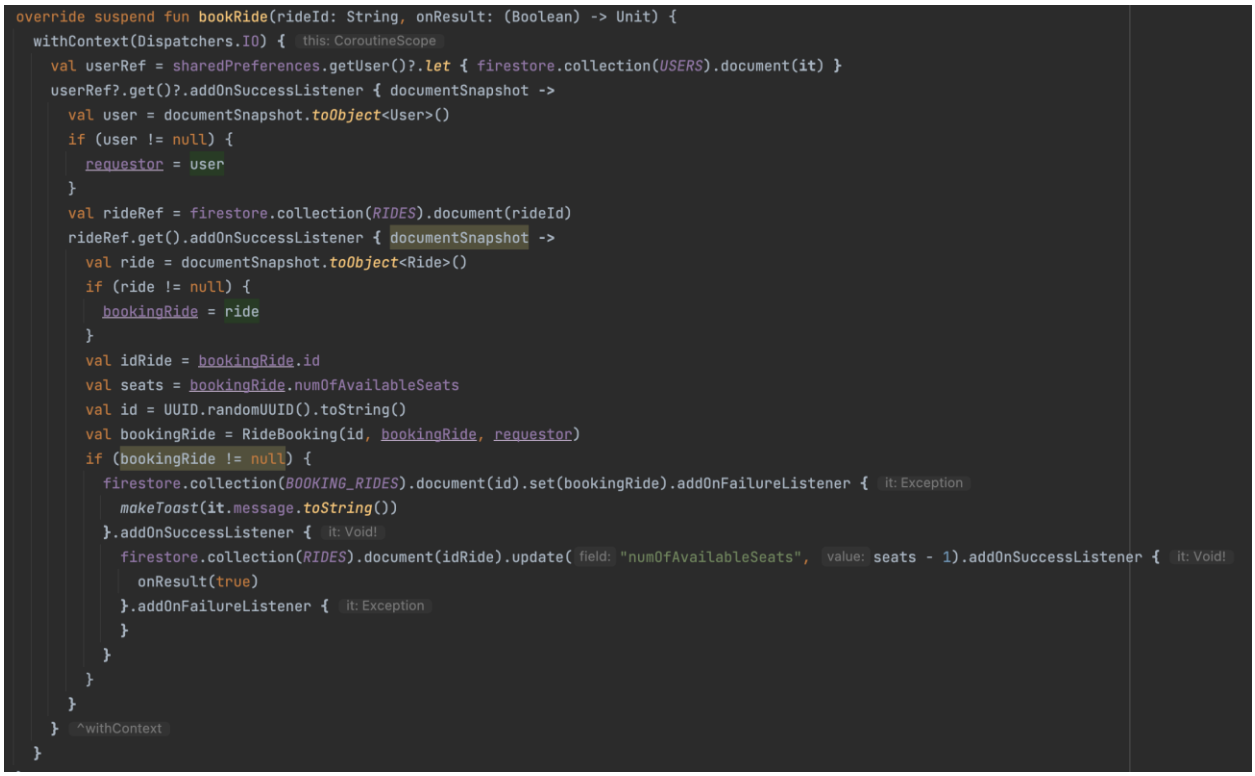


Slika 5.32. *Prikaz kartice (Leo Messi, 30.11.,47kn) prije i poslije rezervacije*

Slika 5.33. *Prikaz kartice kada nema slobodnih mjesta*



Slika 5.34. *Prikaz liste putnika unutar detalja vožnje*



```
override suspend fun bookRide(rideId: String, onResult: (Boolean) -> Unit) {
  withContext(Dispatchers.IO) { this: CoroutineScope
    val userRef = sharedPreferences.getUser()?.let { firestore.collection(USERS).document(it) }
    userRef?.get()?.addOnSuccessListener { documentSnapshot ->
      val user = documentSnapshot.toObject<User>()
      if (user != null) {
        requestor = user
      }
      val rideRef = firestore.collection(RIDES).document(rideId)
      rideRef.get().addOnSuccessListener { documentSnapshot ->
        val ride = documentSnapshot.toObject<Ride>()
        if (ride != null) {
          bookingRide = ride
        }
        val idRide = bookingRide.id
        val seats = bookingRide.numOfAvailableSeats
        val id = UUID.randomUUID().toString()
        val bookingRide = RideBooking(id, bookingRide, requestor)
        if (bookingRide != null) {
          firestore.collection(BOOKING_RIDES).document(id).set(bookingRide).addOnFailureListener { it: Exception
            makeToast(it.message.toString())
          }.addOnSuccessListener { it: Void!
            firestore.collection(RIDES).document(idRide).update( field: "numOfAvailableSeats", value: seats - 1).addOnSuccessListener { it: Void!
              onResult(true)
            }.addOnFailureListener { it: Exception
            }
          }
        }
      }
    } ^withContext
  }
```

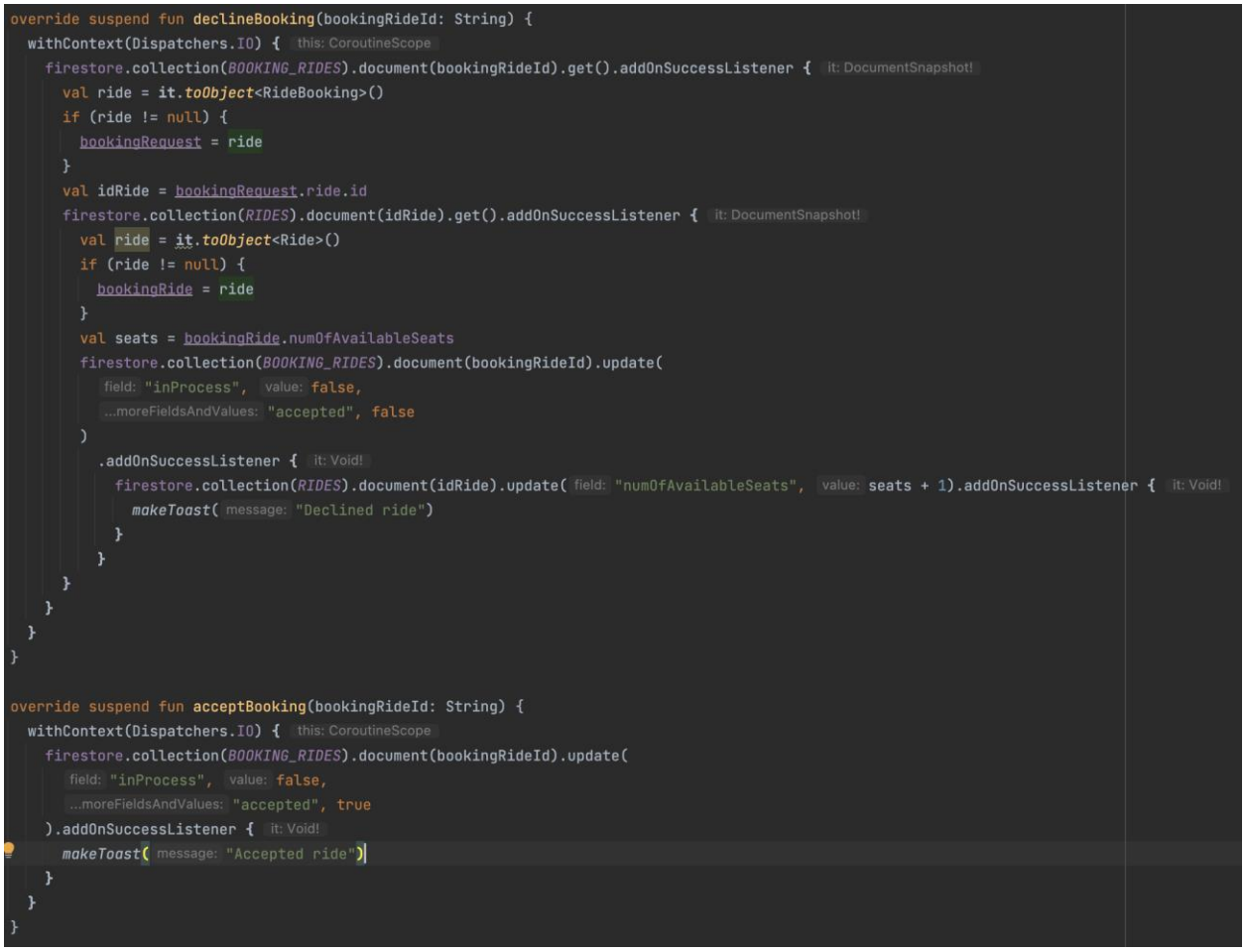
Slika 5.35. *Funkcija za rezerviranje vožnji*

Funkcija za rezerviranje vožnji pomoću ID vožnje pretražuje kolekciju *rides* te dohvaća vožnju s predanim ID te ažurira vožnju umanjujući broj slobodnih sjedala zbog rezervacije. Zatim pomoću ID korisnika pretražuje kolekciju *users* te dohvaća korisnika. Nakon toga s dohvaćenim korisnikom koji je podnositelj zahtjeva te s dohvaćenom vožnjom koja predstavlja vožnju koju se želi rezervirati stvara se novi dokument unutar kolekcije *bookingRides*.

```
override suspend fun getRequestedBookingRides() = withContext(Dispatchers.IO) {  this: CoroutineScope
  suspendCoroutine<List<RideBooking>> { continuation ->
    firestore.collection(BOOKING_RIDES).get().addOnFailureListener {  it: Exception
      continuation.resumeWithException(it)
    }.addOnSuccessListener { documents ->
      val rides = arrayListOf<RideBooking>()
      for (document in documents) {
        val doc = document.toObject<RideBooking>()
        if (doc != null) {
          bookingRide = doc.ride
          requestor = doc.requester
          val isInProcess = doc.inProcess
          if (sharedPreferences.getUserId() == bookingRide.authorId) {
            if (isInProcess) {
              rides.add(
                RideBooking(
                  document.get("id").toString(),
                  bookingRide,
                  requestor,
                  document.get("inProcess").toString().toBoolean(),
                  document.get("accepted").toString().toBoolean()
                )
              )
            }
          }
        }
      }
      rides.sortWith(compareBy({ it.ride.year }, { it.ride.month }, { it.ride.day }, { it.ride.time }))
      rides.removeIf {  it: RideBooking
        !isDateValid(it.ride.day, it.ride.month, it.ride.year)
      }
      continuation.resume(rides)
    }
  }
}
```

Slika 5.36. *Funkcija za dohvaćanje neodgovorenih zahtjeva za rezervacije*

Na slici 5.36. prikazana je funkcija koja dohvaća sve neodogovorene zahtjeve za rezervaciju vožnji čiji je trenutno prijavljeni korisnik autor. Funkcija dohvaća sve dokumente iz kolekcije *bookingRides* te provjerava ako je atribut inProcess istinit te ako je ID korisnika jednak ID-u autora vožnje. Na sličan način funkcioniraju i ostale funkcije za dohvaćanje rezervacija vožnji samo što pomoću atributa *inProcess* i *accepted* jednostavno možemo dohvatiti i prikazati rezervacije vožnje koje su potrebne.



```
override suspend fun declineBooking(bookingRideId: String) {
  withContext(Dispatchers.IO) {
    firestore.collection(BOOKING_RIDES).document(bookingRideId).get().addOnSuccessListener {
      val ride = it.toObject<RideBooking>()
      if (ride != null) {
        bookingRequest = ride
      }
      val idRide = bookingRequest.ride.id
      firestore.collection(RIDES).document(idRide).get().addOnSuccessListener {
        val ride = it.toObject<Ride>()
        if (ride != null) {
          bookingRide = ride
        }
        val seats = bookingRide.numOfAvailableSeats
        firestore.collection(BOOKING_RIDES).document(bookingRideId).update(
          field: "inProcess", value: false,
          ...moreFieldsAndValues: "accepted", false
        )
          .addOnSuccessListener {
            firestore.collection(RIDES).document(idRide).update( field: "numOfAvailableSeats", value: seats + 1).addOnSuccessListener {
              makeToast( message: "Declined ride")
            }
          }
      }
    }
  }
}

override suspend fun acceptBooking(bookingRideId: String) {
  withContext(Dispatchers.IO) {
    firestore.collection(BOOKING_RIDES).document(bookingRideId).update(
      field: "inProcess", value: false,
      ...moreFieldsAndValues: "accepted", true
    ).addOnSuccessListener {
      makeToast( message: "Accepted ride")
    }
  }
}
```

Slika 5.37. *Funkcije za odbijanje i prihvaćanje rezervacije*

Na slici 5.37. su prikazane funkcije za odbijanje i prihvaćanje rezervacije. Prilikom odbijanja rezervacije funkcija pomoću ID rezervacije vožnje pretražuje kolekciju *bookingRides*, pronalazi rezervaciju te ažurira dokument tako da atribute *inProcess* i *accepted* postavlja kao neistinit. Također, prilikom odbijanja rezervacije funkcija pomoću ID vožnje unutar rezervacije vožnje pronalazi ID unutar *rides* kolekcije te ažurira broj slobodnih sjedala vožnje tako da ih povećava jer je rezervacija odbijena. Funkcija za prihvaćanje rezervacije pomoću ID rezervacije vožnje ažurira atribute *inProcess* kao neistinit te *accepted* postavlja kao istinit.

6. ZAKLJUČAK

U ovom diplomskom radu uspješno je implementirana mobilna Android aplikacija za traženje i pružanje usluge prijevoza. U radu su opisane korištene tehnologije, alati i arhitektura aplikacije. Prikazan je izgled aplikacije i mogući slučajevi korištenja aplikacije. Osim izgleda opisane su i najvažnije funkcije i programska rješenja pojedinih zahtjeva aplikacija. Prilikom izrade aplikacije korištene su moderne tehnologije te je sama aplikacija i projekt izrađen po arhitekturalnim obrascima što omogućava jednostavno testiranje, proširenje i održavanje projekta.

Aplikacija je uspješno implementirana, ispunjeni su svi zadani zahtjevi te je kao takva spremna za korištenje. Naravno kao i velika većina aplikacija i ova aplikacija ima prostora za napredak. Jedno od mogućih proširenja bi bilo da se omogući oblik komunikacije između korisnika u obliku slanja i primanja poruka kroz aplikaciju. Također ako bi se aplikacija ikada krenula koristiti u komercijalne svrhe bilo bi dobro implementirati vlastitog poslužitelja jer iako je Firebase vrlo dobar, vlastiti poslužitelj bi pružio više mogućnosti s razine sigurnosti i slično. Korištenjem u komercijalne svrhe bi omogućilo primjerice i korištenje naprednijih *Google-ovih* API-ija koji zahtijevaju plaćanje kao na primjer prikazivanje rute na karti označavanjem cesta i puteva kojih korisnik mora proći od početne do krajnje lokacije i slično.

Iako postoje određene mane i mogućnosti unaprjeđenja aplikacije, kroz upotrebu i testiranje može se zaključiti da se aplikacija ponaša u skladu s očekivanjima i zahtjevima te je primjenjiva u realnim situacijama i pomaže korisnicima da ponude ili nađu uslugu prijevoza jednostavno i brzo što je i bio prvotni plan i zadatak diplomskog rada.

LITERATURA

[1] BlaBlaCar Google Play, dostupno na: https://play.google.com/store/apps/details?id=com.comuto&hl=en&gl=US (pristupljeno listopad 2022.)

[3] Uber Google Play, dostupno na: https://play.google.com/store/apps/details?id=com.ubercab (pristupljeno studeni 2022.)

[2] Turo Google Play, dostupno na: https://play.google.com/store/apps/details?id=com.relayrides.android.relayrides&hl=en&gl=US (pristupljeno studeni 2022.)

[4] Wikipedia, Android Studio, dostupno na: https://en.wikipedia.org/wiki/Android_Studio (pristupljeno listopad 2022.)

[5] Developers, Android Studio Overview, dostupno na: https://developer.android.com/studio/intro (pristupljeno listopad 2022.)

[6] Kotlin for Android, dostupno na: https://kotlinlang.org/docs/android-overview.html (pristupljeno listopad 2022.)

[7] A.Šimec, M.Filipec, Kotlin u odnosu na javu za razvoj na Jvm i Android platformi – Tehničko veleučilište u Zagrebu, 2021, dostupno na: https://hrcak.srce.hr/file/396776 (pristupljeno listopad 2022.)

[8] Stackoverflow 2020 Developer Survey, dostupno na: https://insights.stackoverflow.com/survey/2020 (pristupljeno listopad 2022.)

[9] GeeksForGeeks, MVVM (Model View ViewModel) Architecture Pattern in Android, dostupno na: https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/ (pristupljeno listopad 2022.)

[10] G.Shaha, Medium, Single Responsibiliity Principle, dostupno na: https://proandroiddev.com/single-responsibility-principle-f806c88f4003 (pristupljeno listopad 2022.)

[11] P.Kumar, Medium, Understanding MVVM Architecture in Android, dostupno na: https://medium.com/swlh/understanding-mvvm-architecture-in-android-aa66f7e1a70b (pristupljeno listopad 2022.)

[12] Developers, Navigation Component, dostupno na: https://developer.android.com/guide/navigation/navigation-getting-started (pristupljeno studeni 2022.)

[13] Developers, Pass data between destinations, dostupno na: https://developer.android.com/guide/navigation/navigation-pass-data (pristupljeno studeni 2022.)

[14] Coroutines, dostupno na: https://kotlinlang.org/docs/coroutines-overview.html (pristupljeno studeni 2022.)

[15] Developers, Dependency injection with Hilt, dostupuno na: https://developer.android.com/training/dependency-injection/hilt-android (pristupljeno studeni 2022.)

[16] D.Stevenson, What is Firebase? The complete story, abridged, dostupno na: https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0 (pristupljeno studeni 2022.)

[17] J.E. Lincoln, Everything you need to know about Google Firebase, dostupno na: https://ignitevisibility.com/everything-need-know-google-firebase/ (pristupljeno studeni 2022.)

[18] Wikipedia, Flowchart dostupno na : https://en.wikipedia.org/wiki/Flowchart (pristupljeno studeni 2022.)

[19] Developers, Improve app perfomance with Kotlin coroutines, dostupno na: https://developer.android.com/kotlin/coroutines/coroutines-adv (pristupljeno studeni 2022.)

[20] Developers, Save key-value data, dostupno na: https://developer.android.com/training/data-storage/shared-preferences (pristupljeno studeni 2022.)

[21] Google Maps Platform, Reverse Geocoding, dostupno na: https://developers.google.com/maps/documentation/javascript/geocoding#ReverseGeocoding (pristupljeno studeni 2022.)

SAŽETAK

Ovaj diplomski rad obrađuje temu izrade Android mobilne aplikacije koja nudi mogućnost traženja i pružanja usluge prijevoza između korisnika. Glavni zadatak je bio osmisliti način koji će korisnici imati mogućnost međusobno se dogovoriti za prijevoz. Omogućena je prijava i registracija korisnika pomoću Firebase Authentication-a. Nakon prijave ili registracije korisnik ima mogućnost rezervirati prijevoz koji mu najviše odgovara. Također ima mogućnost filtriranja svih ponuđenih prijevoza po raznim parametrima. Svaki korisnik može ponuditi uslugu prijevoza drugim korisnicama te na temelju zahtjeva za rezervaciju drugih korisnika, može odbiti ili prihvatiti pojedini zahtjev za rezervaciju. Svaka vožnja ima svoje detalje koji su prikazani korisniku kroz jednostavno sučelje aplikacije. Za izradu aplikacije korišten je Kotlin programski jezik.

Ključne riječi: Android, *Firebase*, Kotlin , MVVM, , prijevoz

ABSTRACT

Title: Application for searching and offering transport services

This master's thesis addresses the topic of creating an Android application for searching and offering transport services between users. Users can find an ideal ride for themselves. Users also can filter transport rides with different parameters. The application provides login and registration for users. Every user can offer their transport service. Depending on the ride reservation request, the author of a ride can accept or decline the request. Every transport ride has details that are shown on the app screen. For this master's thesis, Kotlin programming language is used.

Keywords: Android, Firebase, Kotlin, MVVM, transport

ŽIVOTOPIS

Antonio Novinc je rođen 23.7.1997. u Požegi. Živi u mjestu Vidovci blizu Požege. Pohađao je Osnovnu školu Antuna Kanižlića u Požegi. Nakon toga upisuje Opću gimnaziju u Požegi. 2016. godine upisuje preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek. Uspješno završava preddiplomski studij 2020. godine te upisuje diplomski studij računarstva, smjer programsko inženjerstvo. Tijekom studiranja počinje raditi u struci i trenutno se usavršava u području razvoja mobilnih aplikacija.

_____

Potpis autora