Android aplikacija za tržnicu

Haubrich, Kristijan

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

ANDROID APLIKACIJA ZA TRŽNICU

Završni rad

Kristijan Haubrich

Osijek, 2021.

SADRŽAJ

1. UVOD

Tema ovog završnog rada jest izrada android tržnice. Zadatak aplikacije je omogućiti korisniku registraciju preko elektronske pošte i zaporke, te nakon toga omogućiti korištenje raznih funkcionalnosti koje aplikacija nudi među kojima su najvažnije dodavanje proizvoda, kontaktiranje drugih korisnika te pretplaćivanje na druge korisnike. Aplikacija spaja dvije jako važne funkcionalnosti, a to su komunikacija korisnika preko poruka te mogućnost pretplaćivanja.

Tijekom izrade poslužila su znanja stečena iz kolegija „Razvoj programske podrške objektno orijentiranim načelima“ i „Osnove razvoja web i mobilnih aplikacija“. Aplikacija je namjenjena svim korisnicima koji žele promovirati svoje proizvode a nemaju platformu za oglašavanje (najčešće manji proizvođači i obrtnici) ili žele pretražiti proizvode i stupiti u kontakt s proizvođačima proizvoda za kojeg su zainteresirani. Za izvedbu aplikacije koristi se Firebase BaaS (engl. *Backend as a Service*), Java programski jezik, proširivi jezik za označavanje XML (engl. *Extensible Markup Language*) pomoću kojega se opisuju podaci, Retrofit API (engl. *Application Programming Interface*) za slanje HTTP (engl. *Hyper Text Transfer Protocol*) zahtjeva Firebase serveru, *RecyclerView* za prikaz elemenata liste te *Glide* API za prikaz slika u aplikaciji. Aplikacija je programirana unutar Android Studio integriranog razvojnog okruženja (engl. *IDE - Integrated Development Environment*). Struktura završnog rada sastoji se od pregleda sličnih rješenja, teorijske podloge, postupka izrade aplikacije, prikaz izgleda aplikacije te zaključak. U pregledu sličnih rješenja opisano je pet rješenja za kupnju i prodaju koja imaju sličnosti s aplikacijom koja se izrađuje. Teorijska podloga je kratki opis tehnologija korištenih u izradi aplikacije. Izrada same aplikacije predočava i objašnjava same funkcionalnosti aplikacije te kako su one izvedene. U prikazu izgleda aplikacije prikazuje se izgled same aplikacije na uređaju.

U trećem poglavlju objašnjena je teorija svih tehnologija koje su korištene u izradi aplikacije. Četvrto poglavlje opisuje način rada i postupak izrade same aplikacije. Prikaz aplikacije i izgled svih opisanih elemenata prikazuje peto poglavlje. U šestom poglavlju nalazi se zaključak o aplikaciji i cijelom projektu.

1.1. Zadatak završnog rada

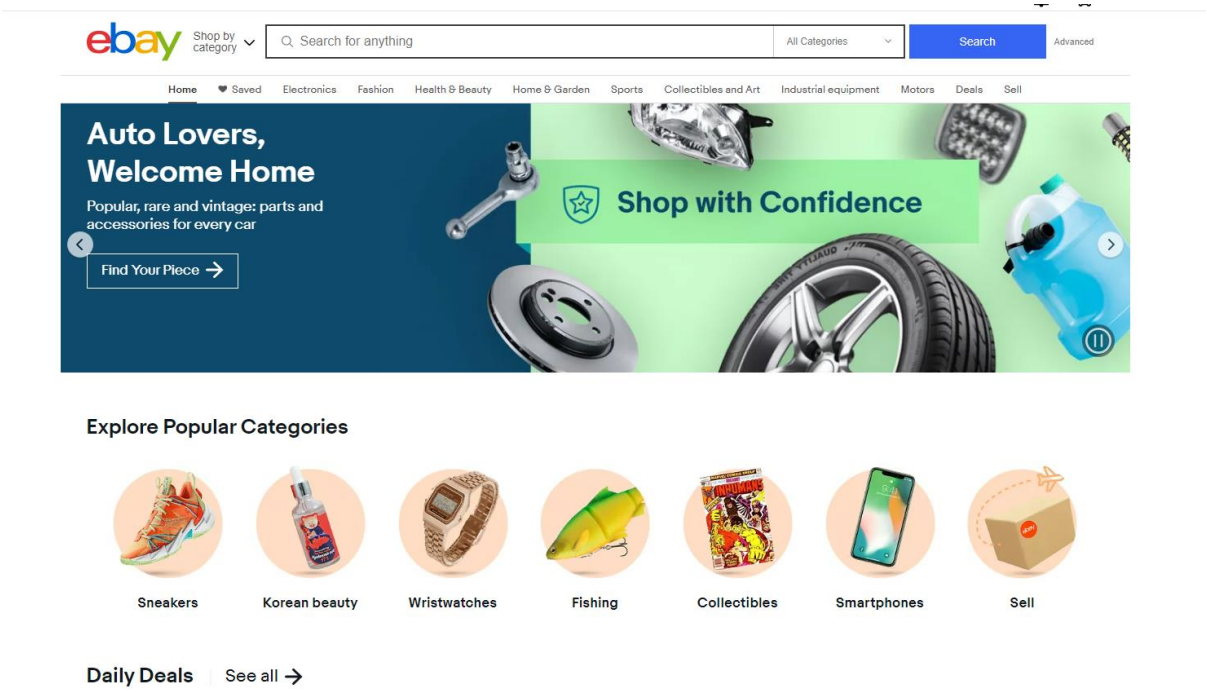
Kratko opisati način rada online tržnice. Izraditi Android aplikaciju koja treba koristiti publish/subscribe metodu pomoću koje će se kupci pretplaćivati na prodavače koji im se sviđaju te će moći zahtjevati od njih kupnju njihovih proizvoda koje su prodavači stavili u aplikaciju. Kupci će moći dobivati obavijesti o novom dodanom proizvodu. Povezivanje na bazu podataka će se raditi preko Firebase API te će se omogućiti još prijava (login) i registracijska logika preko koje će se prodavači i kupci prijavljivati u aplikaciju.

2. PREGLED SLIČNIH RJEŠENJA

Danas postoji jako puno različitih rješenja za prodaju i kupnju različitih proizvoda. Razvojem tehnologije razvila su se i rješenja koja ljudima štede novac i vrijeme.

2.1. eBay

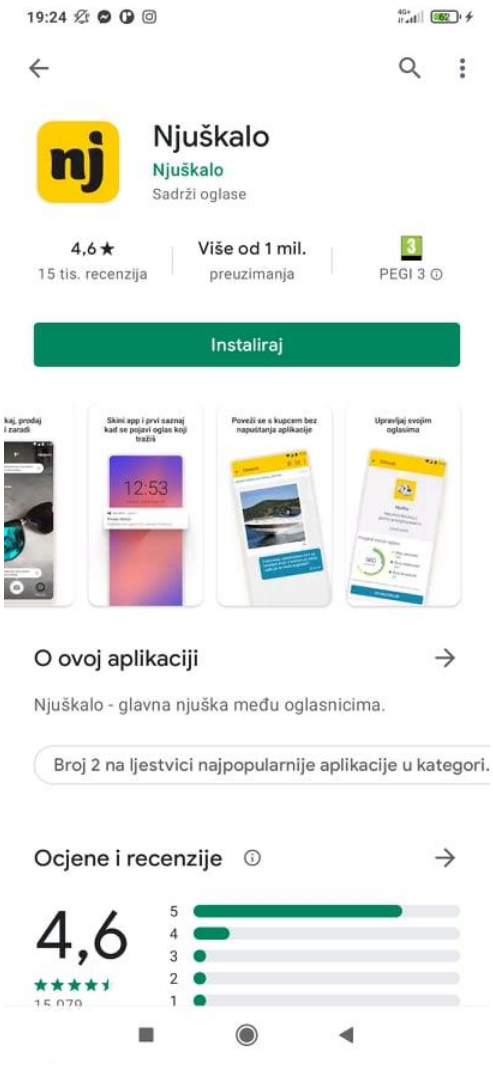
Među najpoznatijim rješenjima je *eBay* web stranica koja pruža online uslugu aukcijske kupnje i prodaje. Kupac koji ponudi najveću svotu novca pobjeđuje na aukciji i zaprima proizvod. Također postoji i mogućnost prodaje po jedinstvenoj cijeni ako prodavač ne želi staviti na aukciju svoj proizvod. Slično kao i u ovom projektu, korisnici mogu kupovati i prodavati proizvode koje oni žele.



Slika 2.1. Prikaz rješenja eBay

2.2. Njuškalo

Njuškalo je web rješenje koje nudi prodaju, kupnju ili najam različitih proizvoda, nekretnina i usluga. Jedno je od najpopularnijih rješenja koje koriste korisnici iz cijele Hrvatske i šire. Kao i u ovom projektu, korisnici mogu kupovati i prodavati svoje proizvode i druge usluge te komunicirati s drugim korisnicima.



Slika 2.2. Pregled rješenja Njuškalo na Trg Play-u

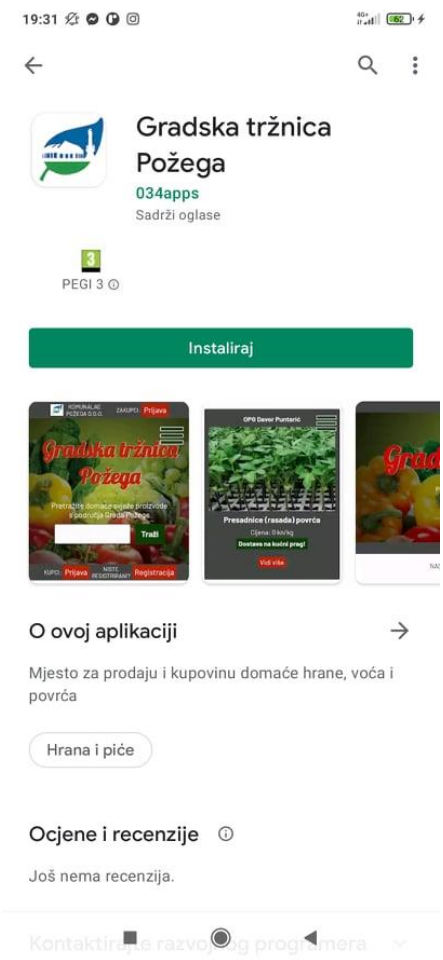
2.3. Crno Jaje

Crno Jaje je primjer hrvatskog portala za kupnju različitih proizvoda i usluga. Kupcu je često omogućen odabir izgleda proizvoda i/ili načina provedbe usluge što čini ovo rješenje jako fleksibilnim. Kao i u ovom projektu, korisnici mogu pregledavati i odabirati proizvode i usluge.

Slika 2.3. Pregled rješenja Crno Jaje

2.4. Gradska tržnica Požega

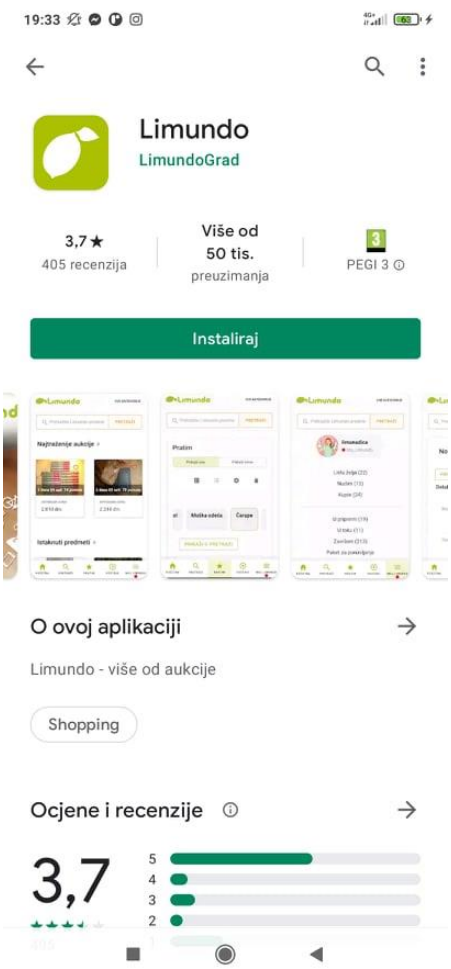
Gradska tržnica Požega je android rješenje koje nudi mogućnost kupnje i prodaje proizvoda na području grada Požege i tržnice koja se nalazi u spomenutom gradu. Kao i u ovom projektu, korisnici mogu odabirati proizvode i kupovati.



Slika 2.4. Pregled rješenja Gradska tržnica Požega na Trg Play-u

2.5. Limundo

Limundo je android rješenje pomoću kojega možete kupovati i prodavati proizvode koji su podijeljenji u nekoliko kategorija. Fleksiblino filtriranje i unošenje novih proizvoda su glavne karakteristike ovog rješenja. Kao i u ovom projektu, korisnici mogu kupovati i postavljati svoje proizvode te predstavlja najsličnije riješenje projekta.



Slika 2.2. Pregled rješenja Limundo na Trg Play-u

3. TEORIJSKA PODLOGA

U ovom poglavlju objašnjene su sve tehnologije koje će biti korištene u aplikaciji. Pored opisa tehnologija objašnjeni su i razlozi korištenja u aplikaciji te njihova uloga.

3.1. Firebase

Backend-as-a-service (BaaS) je model usluge u oblaku u kojem programeri povjeravaju sve aspekte u pozadini (engl. *backend*) web ili mobilne aplikacije tako da trebaju samo pisati i održavati sučelje. BaaS dobavljači nude unaprijed napisani softver za aktivnosti koje se odvijaju na poslužiteljima, kao što su provjera autentičnosti korisnika, upravljanje bazom podataka, daljinsko ažuriranje i obavijesti (za mobilne aplikacije), kao i pohrana u oblaku i usluge poslužitelja (engl. *hosting*). [1]

Firebase je upravo BaaS model usluge na oblaku. Programerima pruža razne alate i usluge koji im pomažu u razvoju kvalitetnih aplikacija, povećanju korisničke baze i ostvarivanju dobiti. Izgrađen je na Google-ovoj infrastrukturi. Firebase je kategoriziran kao program baze podataka NoSQL koji pohranjuje podatke u JSON-slične dokumente. [2]

Glavne značajke Firebase-a su autentifikacija (engl. *authentication*), baza podataka u stvarnom vremenu (engl. *realtime database*), usluge poslužitelja, mjesto za testiranje (engl. *test lab*) i obavijesti (engl. *notifications*). Autentifikacija podržava provjeru autentičnosti pomoću zaporki, telefonskih brojeva, Googlea, Facebooka, Twittera i mnogih drugih. Firebase autentifikacija može se koristiti za ručnu integraciju jednog ili više načina prijave u aplikaciju. Kod baze podataka u stvarnom vremenu podaci se sinkroniziraju na svim klijentima u stvarnom vremenu i ostaju dostupni čak i kada aplikacija nema pristup mreži (engl. *offline*). U mjestu za testiranje aplikacija je testirana na virtualnim i fizičkim uređajima koji se nalaze u Googleovim podatkovnim centrima. Obavijesti se mogu slati s Firebaseom bez dodatnog kodiranja. [2]

Unutar aplikacije koristi se i Firebase Cloud Messaging Service (FCM) za primanje obavijesti o novim proizvodima i porukama. Korištenjem FCM-a može se obavijestiti klijentsku aplikaciju da je nova e-pošta ili drugi podaci dostupni za sinkronizaciju. Mogu se poslati poruke obavijesti kako bi se potaknulo ponovno angažiranje i zadržavanje korisnika. Za slučajeve korištenja, poput razmjene trenutnih poruka, poruka može prenijeti korisni podatkovni teret (engl. *payload*) do 4000 bajtova u klijentsku aplikaciju. [3] Pored navedenih

značajki aplikacija još koristi Firebase Storage pomoću koje se spremaju sve slike korištene u aplikaciji na Firebase server.

3.2. Java programski jezik

Java je općenito, objektno orijentirani programski jezik, zasnovan na klasama, dizajniran za manje ovisnosti o implementaciji. To je računalna platforma za razvoj aplikacija. Java je stoga brza, sigurna i pouzdana. Široko se koristi za razvoj Java aplikacija na prijenosnim računalima, podatkovnim centrima, igraćim konzolama, znanstvenim superračunalima, mobitelima i slično. [4]

Java Platform je zbirka programa koji pomažu programerima učinkovito razvijati i pokretati programe za programiranje Java. Uključuje izvršni mehanizam, kompajler i skup knjižnica u njemu. To je skup računalnog softvera i specifikacija. [4]

Sintaksa Java-e slična je C ++, ali je strogo objektno orijentirani programski jezik. Na primjer, većina Java programa sadrži klase koje se koriste za definiranje objekata i metode koje se dodjeljuju pojedinim klasama. Java je također poznata po tome što je stroža od C++, što znači da varijable i funkcije moraju biti izričito definirane. To znači da izvorni kod Java-e može stvarati pogreške ili "iznimke" lakše od ostalih jezika, ali također ograničava i druge vrste pogrešaka koje mogu biti uzrokovane nedefiniranim varijablama ili neraspoređenim vrstama. Za razliku od Windows izvršnih datoteka (.EXE datoteka) ili Macintosh aplikacija (.APP datoteka), Java programe ne pokreće izravno operativni sustav. Umjesto toga, Java programe interpretira Java Virtual Machine ili JVM. To znači da su svi Java programi multiplatformni i mogu se izvoditi na različitim platformama, uključujući Macintosh, Windows i Unix računala. Međutim, JVM mora biti instaliran kako bi se Java programi uopće mogli izvoditi. Srećom, JVM je uključen kao dio Java Runtime Environment (JRE), koji je dostupan kao besplatno preuzimanje. [5]

3.3. XML proširivi jezik za označavanje

Proširivi jezik za označavanje (XML) koristi se za opisivanje podataka. Standard istoimenog naziva je fleksibilan način za stvaranje informacijskih formata i elektroničku razmjenu strukturiranih podataka putem javnog interneta, kao i putem korporativnih mreža. Jezik XML je sličan jeziku za označavanje hiperteksta (engl. HTML - *Hypertext Markup Language*). Oba

jezika sadrže simbole za označavanje koji opisuju sadržaj stranice ili datoteke. Kôd HTML-a opisuje sadržaj web stranice (uglavnom tekst i grafičke slike) samo u smislu načina na koji se prikazuje i komunicira s njima. [6]

Podaci XML-a poznati su kao samoopisivi ili samodefinirani, što znači da je struktura podataka ugrađena u podatke, pa kad podaci stignu nema potrebe za unaprijed izgrađenom strukturom za pohranu podataka; struktura se tako dinamički razumije unutar XML-a. Format XML-a može koristiti bilo koji pojedinac ili grupa pojedinaca ili tvrtki koji žele dosljedno dijeliti informacije. [6]

Osnovni blok XML dokumenta je element definiran oznakama. Element ima oznaku početka i završetka. Svi su elementi u XML dokumentu sadržani u najudaljenijem elementu poznatom kao korijenski element. Jezik XML također može podržavati ugniježđene elemente ili elemente unutar elemenata. Ova sposobnost omogućuje XML-u podržavanje hijerarhijske strukture. Imena elemenata opisuju sadržaj elementa, a struktura opisuje odnos između elemenata. [6]

XML dokument smatra se "dobro oblikovanim" (to jest, može ga pročitati i razumjeti XML rasčlanjivač sintakse (engl. *parser*)) ako je njegov format u skladu s XML specifikacijom, ako je pravilno označen i ako su elementi pravilno ugniježđeni. Jezik XML također podržava mogućnost definiranja atributa za elemente i opisivanja karakteristika elemenata u početnoj oznaci elementa. [6]

3.4. Android Studio

Android Studio službeno je integrirano razvojno okruženje (IDE) za razvoj Android aplikacija, temeljeno na IntelliJ IDEA. Povrh moćnog IntelliJ-ovog uređivača koda i alata za programere, Android Studio nudi još više značajki koje poboljšavaju produktivnost prilikom izrade Android aplikacija, kao što su:

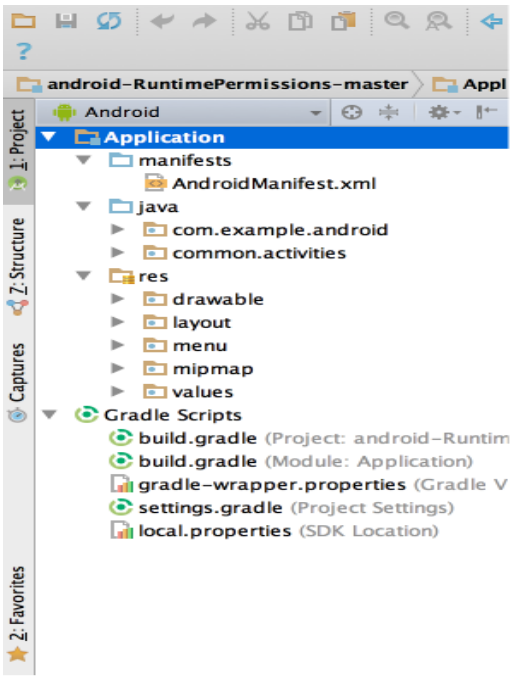
- Fleksibilni graditeljski sustav zasnovan na Gradle-u
- Brz i bogat značajkama emulator
- Objedinjeno okruženje u kojem se mogu razvijati aplikacije za sve Android uređaje
- Mogućnost primijene promjene koda i resursa u pokrenutoj aplikaciji bez ponovnog pokretanja aplikacije

- Predlošci koda i integracija GitHub-a pomažu u izgradnji uobičajenih značajki aplikacije i uvozu uzorka koda
- Opsežni alati i okviri za testiranje
- Lagani alati za prepoznavanje performansi, upotrebljivosti, kompatibilnosti verzija i drugih problema
- Podrška za C++ i NDK
- Ugrađena podrška za Google Cloud Platform, što olakšava integraciju Google Cloud Messaginga i App Enginea [7]

Svaki projekt u Android Studiju sadrži jedan ili više modula s datotekama izvornog koda i datotekama resursa. Vrste modula uključuju:

- Moduli Android aplikacija
- Knjižnični moduli
- Moduli Google App Engine [7]

Prema zadanim postavkama, Android Studio prikazuje projektne datoteke u prikazu Android projekta, kao što je prikazano na **Slici 3.1.** Ovaj je prikaz organiziran po modulima kako bi se omogućio brz pristup ključnim izvornim datotekama projekta. [7]



Sl. 3.1. Prikaz projektnih datoteka u Android Studiu [7]

Sve datoteke gradnje vidljive su na najvišoj razini u Gradle skriptama, a svaki modul aplikacije sadrži sljedeće mape:

- manifesti - sadrži datoteku AndroidManifest.xml.
- java - sadrži datoteke izvornog koda Java, uključujući testni kôd JUnit.
- res - sadrži sve resurse koji nisu kod, kao što su XML izgledi, nizovi korisničkog sučelja i bitmap slike. [7]

Kako bi se vidjela stvarna struktura datoteka projekta, s padajućeg izbornika *Projekt* treba odabrati *Project* (na slici 3.1. prikazuje se kao Android). Također može se prilagoditi prikaz projektnih datoteka kako bi se usredotočilo na određene aspekte razvoja aplikacije. [7]

3.5. Retrofit

Retrofit je API koji predstavlja vezu između korisnika i servera u obliku HTTP klijenta. Jedan je od proizvoda *Square Open Source* kompanije koja se bavi proizvodnjom sličnih kodova koji su besplatni za korištenje i skidanje (engl. *Open Source*). *Retrofit* olakšava slanje HTTP zahtjeva serveru te je važan dio aplikacije gdje se upravlja obavijestima o novim proizvodima i porukama. [8]

3.6. Recycler View

RecyclerView olakšava učinkovito prikazivanje velikih skupova podataka. Korisnik dostavlja podatake i određuje kako svaka stavka izgleda, a biblioteka *RecyclerView* dinamički stvara elemente kada su potrebni. Kao što naziv govori, *RecyclerView* reciklira te pojedinačne elemente. Kad se stavka pomakne s ekrana, *RecyclerView* ne uništava njezin prikaz. Umjesto toga, *RecyclerView* ponovno koristi prikaz novih stavki koje su pomaknute na zaslonu. Ova ponovna upotreba znatno poboljšava performanse, poboljšavajući odziv aplikacije i smanjujući potrošnju energije. [9]

3.7. Glide

Glide je brz i učinkovit okvir (engl. *framework*) za upravljanje medijima otvorenog koda i učitavanje slika za Android koji omotava dekodiranje medija, predmemoriranje memorije i diska (engl. *memory and disk caching*) te udruživanje resursa u jednostavno sučelje. [10]

4. POSTUPAK IZRADE ANDROID APLIKACIJE ANDROID TRŽNICE

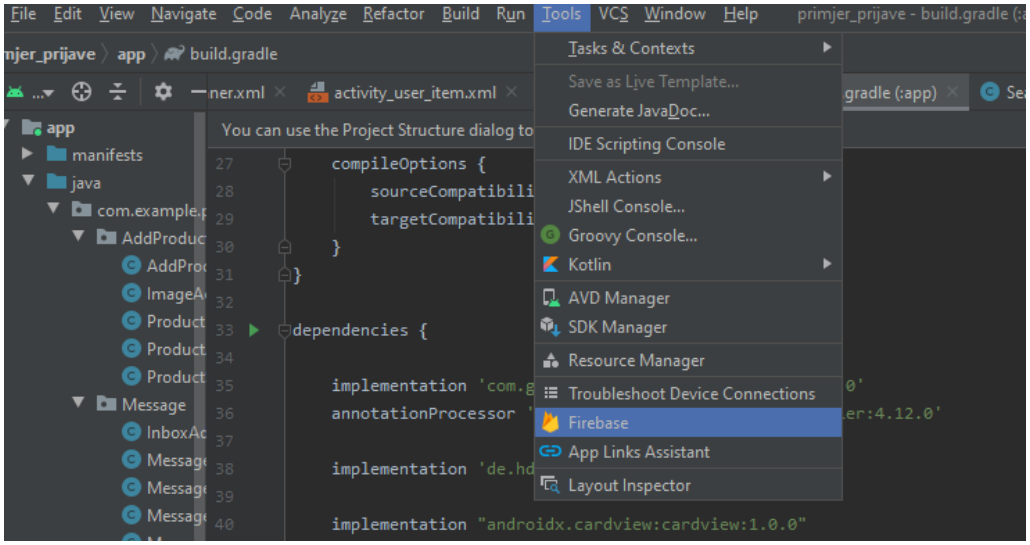
U postupku izrade prvo treba odrediti programske alate koji će se koristiti. Koristi se Firebase BaaS, stoga koriste se većinom Firebase alati za komuniciranje s bazom podataka. Nakon kreiranja aplikacije potrebno je prvo spojiti aplikaciju s bazom podataka na Firebase serveru. Bazu podataka u stvarnom vremenu odlučeno je raditi uz pomoć Firebase Realtime Database kompleta za razvoj softvera (engl. *SDK - Software Development Kit*) koji olakšava upravljanjem JSON objektima unutar baze na Firebase serveru. Od ostalih Firebase SDK-ova koristi se Firebase Authenthification SDK za registraciju korisnika i održavanje registriranih korisničkih profila, Firebase Cloud Messaging (FCM) SDK za upravljanje obavijestima o novim proizvodima i porukama te Firebase Storage SDK za upravljanje i spremanje slika na Firebase server.

Za slanje obavijesti uz FCM koristi se i *Retrofit* za lakše upravljanje i slanje HTTP zahtjeva koji se šalju Firebase serveru. *Glide* je korišten kod prikazivanja slika u aplikaciji. Za prikazivanje liste elemenata koristi se *RecyclerView*. Za prikazivanje profilnih slika korišten je oblik kružne slike s obrubom. To je omogućeno pomoću *Circle Image View* pogleda od *de.hdodenhof* [11] korisnika na platformi GitHub. Za prikaz elementa *RecyclerView*-a korišten je *CardView* pogled.

Zbog opsežnosti kodovi metoda i klasa korištenih u aplikaciji nisu prikazani u dokumentu. Programski kod cijelog projektnog zadatka nalazi se na optičkom mediju koji će biti priložen uz ovaj rad.

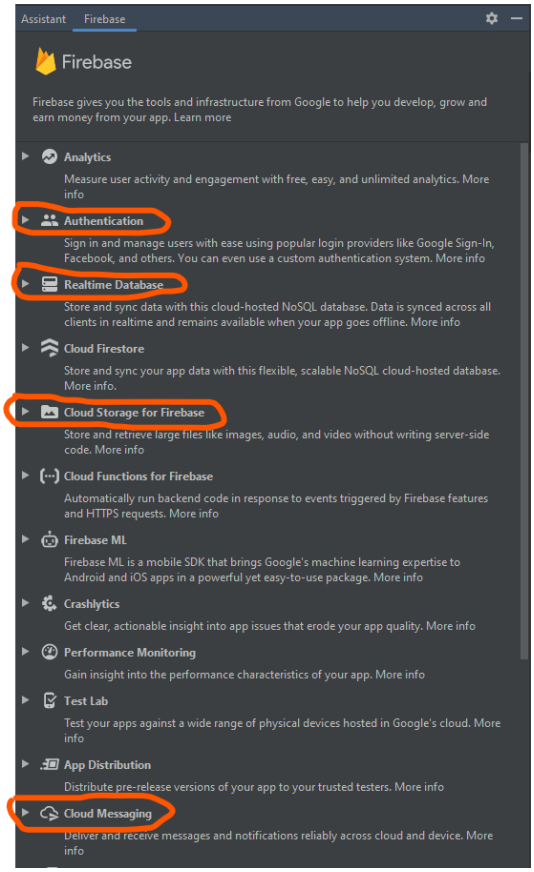
4.1. Povezivanje aplikacije s Firebase projektom

Za početak potrebno je povezati aplikaciju s Firebase projektom. U alatnoj traci Android Studia odabere se kartica *Tools* i nakon toga u padajućem izborniku odabere se *Firebase* opciju (**Slika 4.1.**).



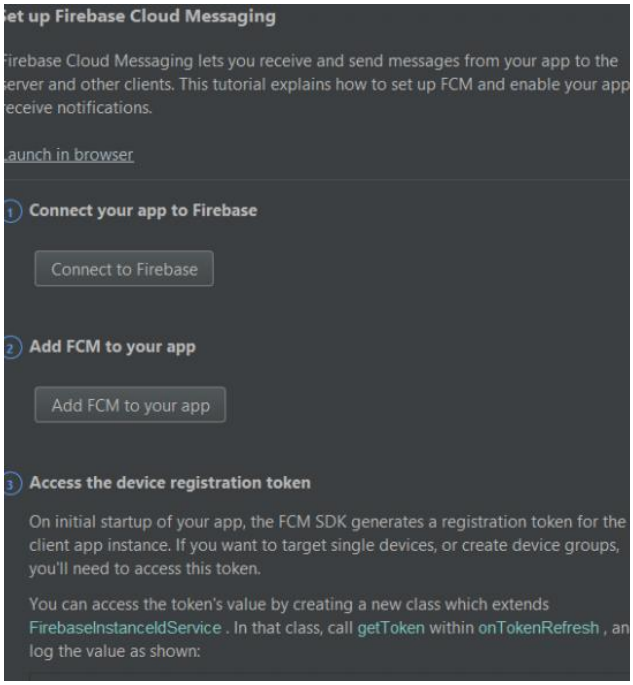
Slika 4.1. *Odabir Firebase alata*

Potom se otvori izbornik Firebase alata (**Slika 4.2.**) te se klikne na bilo koji SDK koji je potreban u projektu.

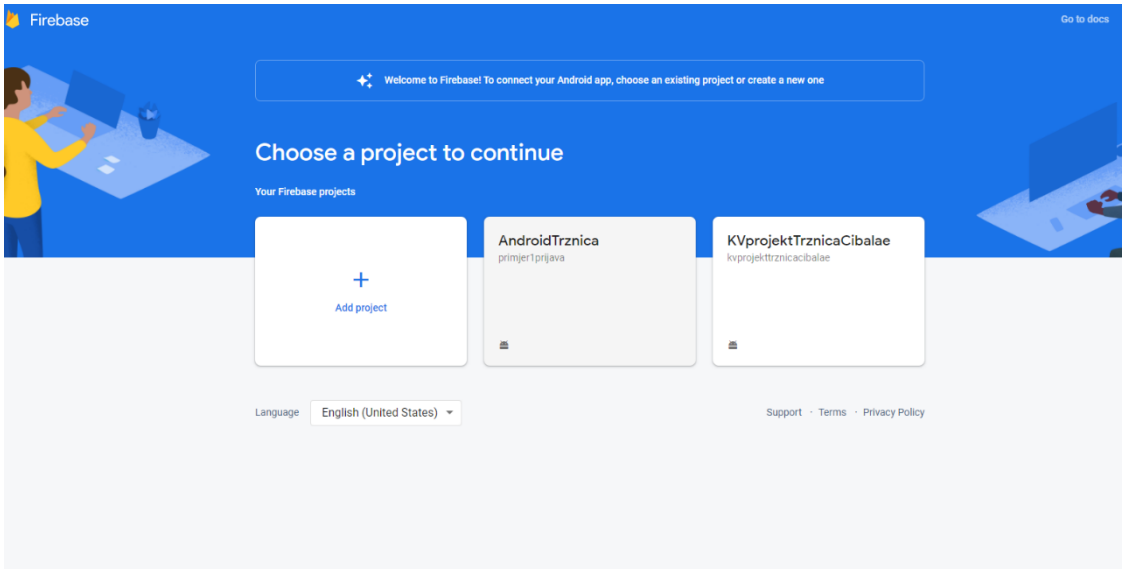


Slika 4.2. *Firebase alati*

Nakon klika na neki od potrebnih alata klikne se na *Connect To Firebase (***Slika 4.3.***)* te se potom obavi povezivanje aplikacije na željeni Firebase projekt jednostavnim odabirom projekta na Firebase konzoli (**Slika 4.4.**).



Slika 4.3. *Dodavanje FCM SDK-a*

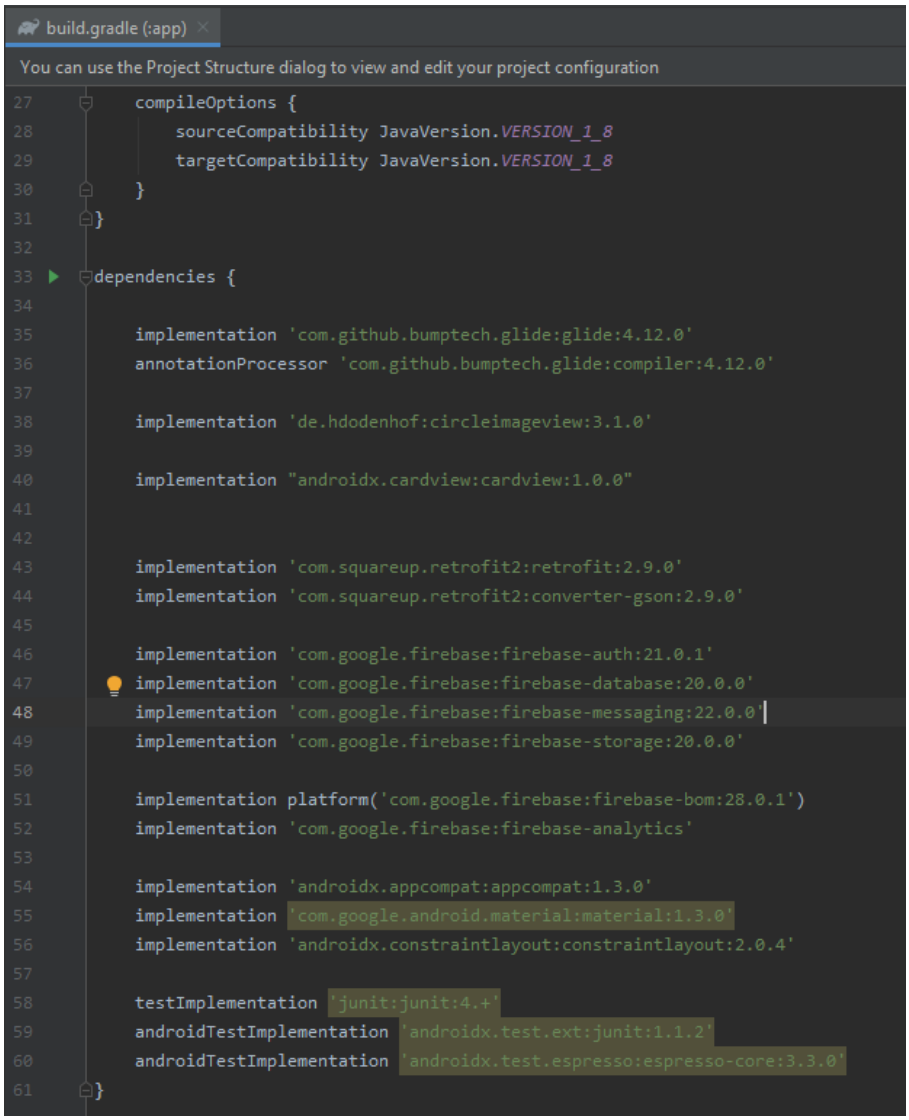


Slika 4.4. *Odabiranje Firebase projekta*

Nakon toga klikne se na drugi korak, a to je dodavanje odabranog SDK-a u aplikaciju (**Slika 4.3.** prikazuje između ostalog i gumb za dodavanje FCM SDK-a u aplikaciju). **Slika 4.2.**

prikazuje Firebase alate, a narančastom bojom su zaokruženi svi Firebase SDK-ovi koje treba dodati u aplikaciju.

Nakon dodavanja svih potrebnih Firebase SDK-ova i povezivanja aplikacije sa željenim Firebase projektom potrebno je dodati i ostale ovisnosti (engl. *dependencies*) za tehnologije koje se koriste (*Retrofit*, *RecyclerView*, *Glide*, *CardView* te *Circle Image View*) unutar gradle modula (**Slika 4.5.**).

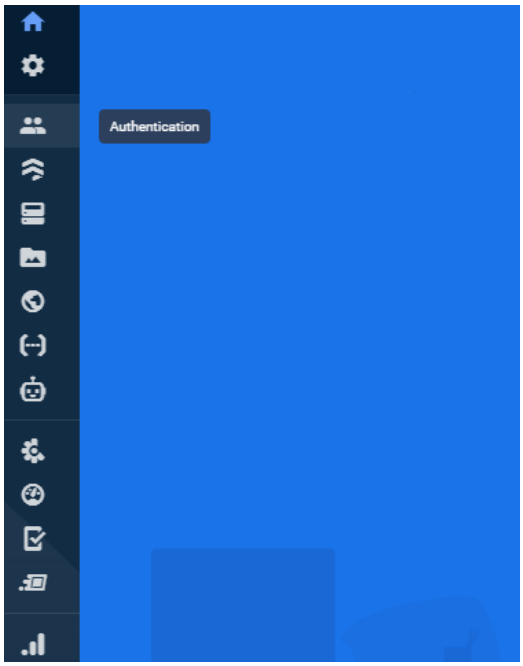


Slika 4.5. *Izlistanje koda za sve ovisnosti (dependencies) u build.gradle (:app) dokumentu*

4.2. Registracija i prijava u aplikaciju te početni izgled

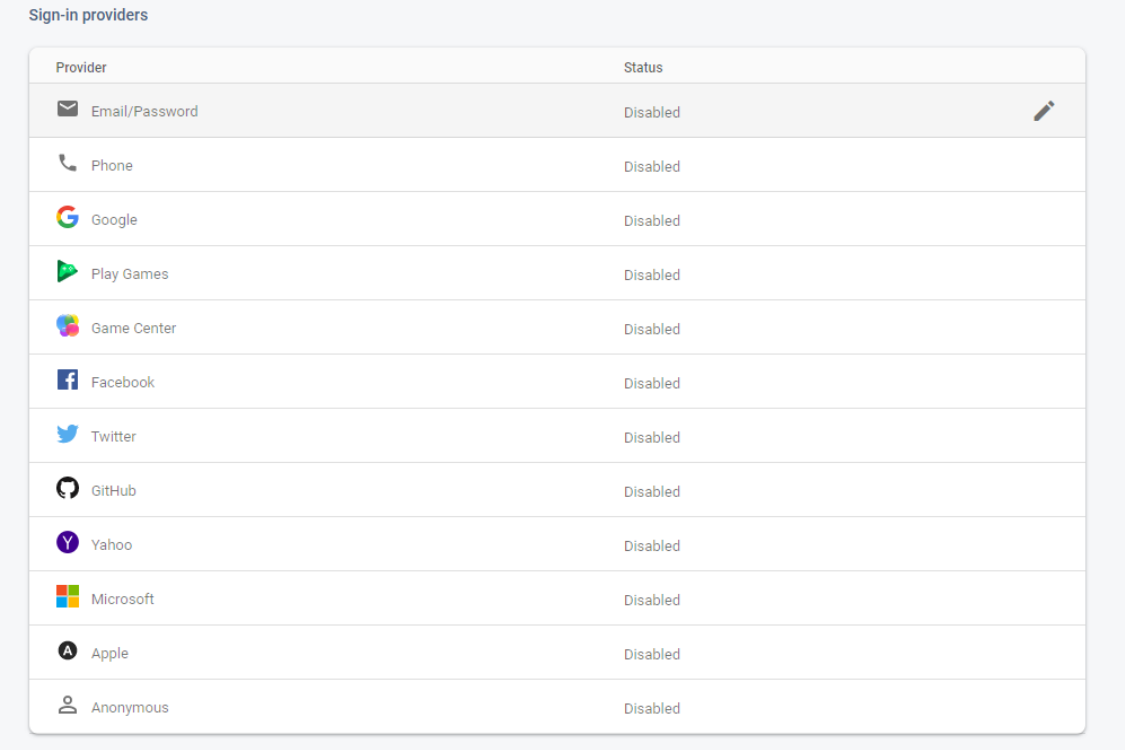
U ovom poglavlju opisane su četiri aktivnosti i jedan fragment odgovorni za uslugu prijave i registracije korisnika te početni izlged neposredno nakon prijave u aplikaciju. Registracija u

aplikaciju se izvodi uz pomoć adrese e-pošte korisnika i lozinke. Firebase pomaže u mnogim funkcionalnostima prijave i registracije. Kako bi se registracija korisnika preko Firebase-a ostvarila potrebno je omogućiti registraciju preko adrese e-pošte i zaporke unutar Firebase projekta. To se učini odlaskom na Firebase projekt s kojim je povezana aplikacija te se ode na karticu *Authentication* (**Slika 4.6.**).

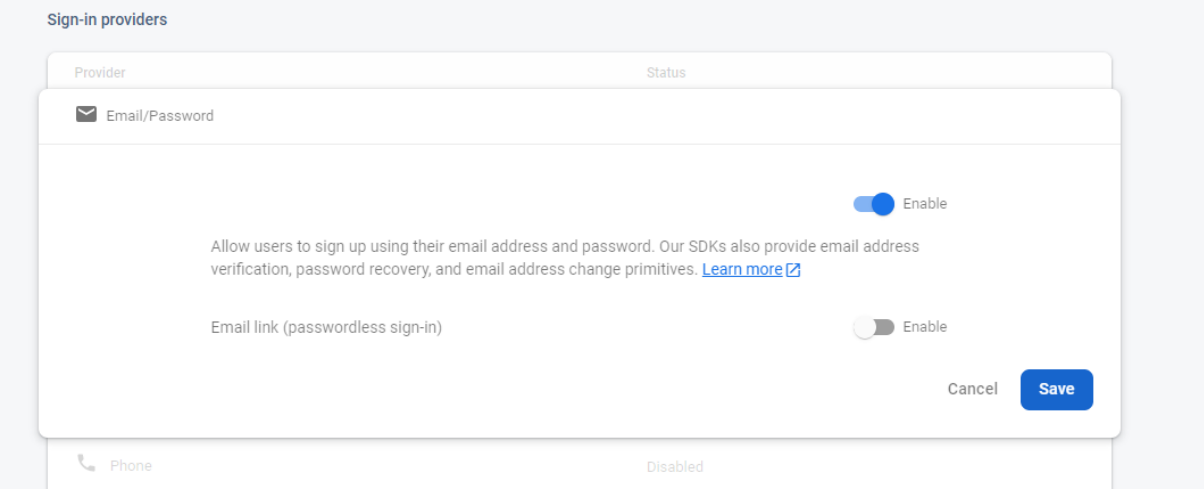


Slika 4.6. Odabir Authentication kartice u Firebase projektu

Nakon što je odabrana *Authentication* kartica potrebno je odabrati *Email/Password* način registracije (**Slika 4.7.**) i kliknuti na *Enable* te potom na *Save* (**Slika 4.8.**).



Slika 4.7. Odabir Email/Password opcije registracije



Slika 4.8. Odabir Enable opcije i Save gumba

Tako je aplikacija sada uspješno povezana s Firebase autentikacijom preko e-pošte i zaporke. Pomoću korištenja jednostavnih funkcija od Firebase Authentification SDK-a može se registrirati korisnika. Jedinstvenost korisnika u Firebase-u odražava se u e-pošti korisnika koja je za svakog posebna. Ipak, zbog ostalih potreba aplikacije mora se pripaziti i na jedinstvenost korisničkog imena pri registraciji korisnika te korištenja nedozvoljenih znakova

za naziv grane u bazi podataka ('.', '#', '$', '[', ili ']') unutar korisničkog imena. Kako se napravi registracijska logika i prijava bit će prikazano u poglavljima koji slijede.

4.2.1. MainActivity

Prvi prikaz nakon otvaranja aplikacije je prikaz *MainActivity* aktivnosti (eng. *activity*). Ova aktivnost je odgovorna za prijavljivanje korisnika kao i ponudu za odlazak na registraciju ili promijenu lozinke.

U ovom poglavlju bit će prikazane najbitnije funkcionalnosti (kao i kod preostalih potpoglavlja u poglavlju 4.) a prikaz samog izgleda *MainActivity* aktivnosti (kao i svih drugih aktivnosti) bit će prikazan u poglavlju 5.

Pritiskom na *TextView REGISTRACIJA* otvara se *RegisterActivity* aktivnost u kojoj se korisnik registrira. Pritiskom na *TextView Zaboravili ste lozinku?* otvara se *ResetPasswordActivity* aktivnost u kojem korisnik može zatražiti novu zaporku u slučaju da je zaboravio postojeću. Ako se pritisne na gumb *PRIJAVA* pokreće se metoda *userLogin* koja prijavljuje korisnika ako je već registriran i ako je pravilno popunio polja za upis adrese e-pošte i zaporke.

Ako je korisnik upisao sve pravilno za prijavu poziva se metoda *login* koja za argumente prima zaporku i adresu e-pošte koju je korisnik upisao. Metoda *login* zatim prijavljuje korisnka koristeći *signInWithEmailAndPassword* metodu.

Ako je korisnik potvrdio svoju adresu e-pošte, poziva se prvo metoda *getToken* koja dobavlja token od uređaja na kojem je korisnik i sprema ga u bazu na Firebase-u u podgranu grane *Tokens* koja ima naziv korisnikovog imena.

Zatim se poziva metoda *goToUserProfile* koja otvara novu aktivnost *UserProfileActivity* te sprema zaporku korisnika u bazu podataka u podgranu grane *Passwords* koja ima naziv korisnikovog imena. Ako korisnik nije potvrdio svoj adresu e-pošte, šalje mu se na e-poštu potvrda koju treba kliknuti kako bi adresa e-pošte bila potvrđena.

4.2.2. RegisterActivity

Ova aktivnost je odgovorna za registraciju novog korisnika i upozoravanje korisnika za bilo kakav oblik krivog unosa podataka.

Korisnik upisuje svoju adresu e-pošte, zaporku te korisničko ime. Ako korisnik stisne na gumb *REGISTRACIJA* poziva se metoda *registerUser* koja provjerava je li korisnik upisao dobre podatke, pa ako je, poziva metodu *register*. Važno je napomenti ako korisničko ime već postoji unutar baze, metoda *register* će dobiti prazan string za argument korisničkog imena.

Unutar *register* metode prvo se provjerava je li korisničko ime prazan string ili nije. Proces registracije se nastavlja ako korisničko ime nije prazan string i ne sadrži znakove '.', '#', '$', '[', ili ']'. Prvo se registrira korisnika u Firebase autentifikaciji pozivajući metodu *createUserWithEmailAndPassword*. Potom se adresa e-pošte i zaporka korisnika sprema u bazu podataka pod granom *Users*. Korisnici u *Users* grani se razlikuju po jedinstvenom stringu koji identificira svakog korisnika u Firebase bazi, a dobije se pozivanjem metode *getUid*. Zatim se pohranjuje u bazu token od uređaja kojeg korisnik koristi te se prebacuje na prikaz *MainActivity* aktivnosti.

U slučaju da korisnik pritisne gumb *NAZAD*, vraća se prikaz *MainActivity* aktivnosti.

4.2.3. ResetPasswordActivity

Ova aktivnost je odgovorna za slanje e-pošte pomoću kojeg može promijeniti postojeću zaporku na adresu e-pošte. Ako korisnik unese pravilno svoju adresu e-pošte i klikne na gumb *PROMIJENI LOZINKU* poslat će mu se poruka preko e-pošte za promjenu zaporke pomoću metode *resetPassword*. Na kraju se prikaz vraća na *MainActivity* aktivnost kao i kada bi se pritisnio gumb *NAZAD*.

4.2.4. UserProfileActivity

Ova aktivnost se prikazuje nakon svake uspiješne prijave korisnika. Na dnu sadrži navigaciju za kretanje po fragmentima, a iznad navigacije se nalazi spremnik (engl. *container*) za prikaz trenutnog fragmenta. U spremniku može biti prikazan jedan od četiri fragmenta: *HomeFragment*, *AddProductFragment*, *MessageFragment* ili *SearchFragment*. Kod stvaranja ove aktivnosti bitno je naglasiti da očekuje podatak o tipu fragmenta kojeg stvara. Ako je *type* varijabli pridružena vrijednost *'upload'* onda će se u spremniku prikazati *AddProductFragment*, a ako je pridružena vrijednost *'message'* prikazat će se *MessageFragment*. Inače će se uvijek prvo prikazati *HomeFragment*. Sve ove provjere odrađene su u metodi *setStartingFragment*.

Unutar navigacije nalaze se ikone za svaki fragment te ako se pritisne na neku od njih prikazat će se fragment koji je pridružen toj ikoni.

4.2.5. HomeFragment

Ovaj fragment se prikazuje unutar spremnika *UserProfileActivity* aktivnosti odmah nakon prijave ili ako se klikne na ikonu u navigaciji *UserProfileActivity* aktivnosti kojoj je pridružen ovaj fragment. *HomeFragment* fragment je početni prikaz nakon prijave. Fragment sadrži profilnu sliku korisnika, ime korisnika, adresu e-pošte korisnika, broj pretplanika pretplaćenih na korisnika, gumb *ODJAVA*, *RecyclerView* koji prikazuje proizvode korisnika, *SearchView* preko kojega se pretražuju proizvodi te gumb *IZBRIŠI RAČUN*. U početku se profilna slika prikaže ako korisnik ima postavljenu profilnu sliku pomoću metode *loadProfilePic* gdje je definiran slušatelj (eng. *listener*) za dohvaćanje profilne slike s Firebase-a.

U slučaju da korisnik nije dodao profilnu sliku prikazuje se zamjenska ikona (engl. *placeholder*) što je omogućeno preko *Glide*-a. Ako korisnik klikne na profilnu sliku, preusmjerava se na prikaz *ImageActivity* aktivnosti gdje korisnik može urediti svoju profilnu sliku. Sljedeća važna metoda koja se poziva pri stvaranju fragmenta je *checkSubscribersCount* koja provjerava broj pretplatnika pretplaćenih na korisnika.

Metoda prvo pronađe ime korisnika koji se prijavio i pokupi njegove podatke te ih prikaže u obliku imena korisnika i adrese e-pošte korisnika. Također se u tom dijelu dodijeli listener kreiran u metodi *loadProfilePic* nad granom *ProfilePics* u bazi podataka na Firebase-u. Slijedi provjera jesu li svi pretplatnici još uvijek u bazi. Ako neki korisnik nije više u bazi, znači da je izbrisao profil te se briše podgrana nazvana imenom obrisanog korisnika čiji je roditelj grana ime trenutnog korisnika. Grana trenutnog korisnika sadrži sve pretplatnike trenutnog korisnika, a ona sama je podgrana velike grane *Subscribers* u bazi podataka. Ako pretplatnik postoji, brojač za pretplatnike se inkrementira za jedan. Kad se prođe kroz sve pretplatnike, brojač za pretpltnike (u aplikaciji varijabla *subscribers_count*) se postavi kao broj pretplatnika trenutno prijavljenog korisnika.

Pri stvaranju fragmenta također se treba postaviti *RecyclerView* za proizvode korisnika. Taj posao odrađuje metoda *setRecycler*. Važno je napomenti da ta metoda postavlja prikaz te instancira listu koja će se koristiti za prikaz proizvoda.

Metoda *fetchData* puni instanciranu listu podacima koji se nalaze unutar grane nazvane imenom korisnika čija je roditeljska grana nazvana *Produts*. Metoda postavlja adapter *RecyclerView*-a naziva *ProductAdapter* (o *ProductAdapter*-u nešto više u poglavlju o proizvodima).

Također, postavlja listener na *SearchView* kako bi osluškivao je li korisnik upisao neki string. Ako je, poziva se metoda *search* koja stvara novi *RecyclerView* koji sadrži proizvode čije ime sadrži string kojeg je korisnik zadao. Svi proizvodi imaju mogućnost brisanja. Pritiskom na gumb *ODJAVA* korisnika se odjavljuje i prikazuje se *MainActivity* aktivnosti. Pritisnkom na gumb *IZBRIŠI RAČUN* prelazi se na prikaz *DeleteActivity* aktivnosti.

4.2.6. DeleteActivity

DeleteActivity je aktivnost odgovorna za brisanje korisničkog računa. Kako bi se korisnički račun uspješno obrisao potrebno je obrisati sve korisnikove podatke iz baze podataka zajedno sa svim slikama proizvoda i profilnom fotografijom. U aktivnosti se nalaze dva gumba: *IZBRIŠI RAČUN* i *NAZAD*. Funkcija gumba *NAZAD* jest da korisnika odvede nazad u *UserProfileActivity* aktivnost ako se ipak predomislio i odlučio sačuvati svoj račun. U slučaju da je korisnik ipak odlučio obrisati svoj račun pritisniti će na gumb *IZBRIŠI RAČUN*. Potom se aktivira metoda *deleteUser*. Prvi korak koju metoda odradi je brisanje korisnika iz Firebase autentifikacije. Nakon toga slijedi brisanje iz baze podataka pozivajući redom metode koje brišu korisnikove podatke iz raznih grana baze.

Metodom *deleteProfilePic* briše se URI (engl. *Uniform Resource Identifier*) korisnikove profilne slike ako ju je imao. Put granama baze podataka do URI-ja profilne slike je: *profilePics -> korisnikovo ime*. Prije nego što se obriše URI, koristi se da se pronađe slika na Firebase Storage-u te se prvo tamo obriše.

Metodom *deleteFromUsers* briše se adresa e-pošte i korisničko ime spremljeno u bazi podataka unutar *Users -> Uid korisnika* . Uid korisnika se dobije metodom Firebase Authentication-a naziva *getUid* .

Metodom *deleteFromMessages* brišu se sve poruke spremljene u bazi unutar *Messages->korisnikovo ime*.

Metodom *deleteFromSubscribers* brišu se podaci o svim korisnikovim pretplatnicima koji se nalaze unutar *Subscribers->korisnikovo ime*.

Metodom *deleteToken* briše se token uređaja kojeg korisnik koristi, a nalazi se unutar *Tokens->korisnikovo ime*.

Metodom *deleteFromProducts* brišu se svi proizvodi koje je korisnik imao u tom trenutku, a nalaze se unutar *Products->korisnikovo ime*. Kao i kod brisanja profilne slike, prvo se provjerava je li proizvod koji se briše imao sliku. Ako je, prvo se briše slika spremljena na Firebase Storage pa se tek onda brišu ostali podaci o proizvodu u bazi podataka.

Metodom *deletePassword* briše se zaporka koju je korisnik imao zapisanu unutar *Passwords->korisnikovo ime*.

Nakon što su obrisani svi podaci o korisničkom računu, korisnika se prebacuje na *MainActivity* aktivnost.

4.3. Pretraživanje drugih korisnika i prikazivanje njihovih profila

U ovom poglavlju objašnjeno je na koji način korisnik može pretražiti druge korisnike te kako izgleda aktivnosti koji prikazuje profil nekog odabranog korisnika. Objašnjena je također jedna POJO klasa te tehnika pretplaćivanja korisnika na nekog drugog korisnika.

POJO je kratica za Plain Old Java Object. To je obični Java objekt, koji nije vezan nikakvim posebnim ograničenjima osim onih koje nameće *Java Language Specification*. POJO se koriste za povećanje čitljivosti i ponovne upotrebljivosti programa. POJO-i su najviše prihvaćeni jer ih je lako napisati i razumjeti. [12] Pomoću POJO klase definira se određeni entitet koji se onda koristi u drugim dijelovima programa.

4.3.1. User

POJO klasa *User* sadrži u sebi dva privatna atributa: *username* i *email*. Oni predstavljaju korisničko ime i adresu e-pošte nekog korisnika. Može im se pristupiti preko *gettera,* a postaviti ih preko *settera*. Ova klasa je jako važna jer se pomoću nje uzimaju podaci o korisniku u *Users* grani baze podataka te se koriste u mnogo dijelova aplikacije.

4.3.2. UserAdapter

UserAdapter je adapter za *RecyclerView* koji prikazuje korisnike. Koristi listu tipa *User* za spremnik podataka. Kao i svaki drugi adapter za *RecyclerView*, i ovaj ima tri standardne metode: *onCreateViewHolder*, *onBindViewHolder* i *getItemCount*. Za prikaz jednog elementa *RecyclerView*-a koristi se XML kod *activity_user_item*. Unutar tog XML koda definirano je kako će se prikazivati profilna slika korisnika, broj pretplatnika te ime korisnika. U prikazu je još dodana i ikona za brisanje elementa, ali ona se koristi u drugom adapteru. U ovom adapteru ikona za brisanje elementa je postavljena na *INVISIBLE*, tj. ne prikazuje se. Elementi *activity_user_item* koda se pridružuju atributima *ViewHolder* klase koji se kasnije koriste kod prikaza elementa recycler-a.

Metoda *getItemCount* vraća broj elemenata u *ReycylerView*, a to čini tako da vrati broj elemenata liste koja se koristi kao spremnik.

Unutar *onCreateViewHolder* metode instancira se intent objekt koji će voditi u *ShowUserActivity* aktivnost te se potom postavlja listener na cijeli element koji osluškuje klik na element. U slučaju pritiska na element poziva se metoda *goToUserProfile*. Ova metoda prosljeđuje intentu instanciranom u metodi *onCreateViewHolder* dodatni podatak o korisnikovom imenu čiji je element odabran. Nakon toga pokreće prikaz *ShowUserActivity* aktivnosti.

Metoda *onBindViewHolder* postavlja izgled elementa. Postavlja ikonu za brisanje elementa na *INVISIBLE* te korisničko ime povlači iz spremnika (liste elemenata tipa *User*). Potom poziva dvije metode: *getSubscribersCount* i *setProfilePic*.

Metoda *getSubscribersCount* prebrojava broj pretplatnika korisnika u bazi podataka te postavlja broj pretplatnika u prikazu. Metoda *setProfilePic* provjerava ima li korisnik prikazan u elementu profilnu fotografiju. Ako ima postavlja ju u prikazu, a ako nema postavlja zamjensku ikonu na mjesto profilne fotografije.

4.3.3. SearchFragment

U ovom fragmentu korisnik ima mogućnost pregleda svih korisnika. Fragment ima *SearchView* preko kojeg može pretraživati korisnike te *RecyclerView* koji prikazuje korisnike. Ako korisnik klikne na nekog od korisnika prikazanih u *RecyclerView*, otvara se i prikazuje profil korisnika unutar *ShowUserActivity* aktivnosti.

Kao i u *HomeFragment* fragmentu, pomoću metode *setRecycler* postavlja se *RecyclerView*. Uz pomoć metode *fetchData* uzimaju se podaci o korisnicima i spremaju u listu korisnika koju *RecyclerView* koristi kao spremnik podataka. Jedina razlika između *RecyclerView*-a u ovom fragmentu i *RecyclerView*-a u *HomeFragment* fragmentu je što se sada prikazuju korisnici a ne proizvodi. Tako se umjesto grane *Products* pretražuje grana *Users* unutar baze podataka na Firebase-u. U prethodnom poglavlju opisan je način rada *UserAdapter* klase koju koristi ovaj *RecyclerView*. Bitno je navesti još metodu *search* koja filtrira korisnike s obzirom na string kojeg je korisnik upisao u *SearchView*.

4.3.4. ShowUserActivity

Ova aktivnost je odgovorna za prikaz profila drugih korisnika. Sadrži profilnu sliku korisnika, korisničko ime, broj pretplatnika, ikonu za otvaranje *MessageActivity* aktivnosti koja omogućava dopisivanje s korisnikom, dva gumba pomoću kojih se može pretplatiti, odnosno otkazati pretplata na korisnika, gumb za vraćanje na *UserProfileActivity* aktivnost te *RecyclerView* za proizvode korisnika.

Radi prikazivanja svih podataka željenog korisnika prvo se treba dohvatiti podatak o imenu korisnika kojeg se želi prikazati. Taj podatak se dobije od aktivnosti koja je aktivirala prikaz profila korisnika. Nakon toga treba postaviti prikaz imena korisnika, broja pretplatnika i profilne fotografije. Upravo to čini metoda *setUserData.*

Potom se uzima podatak o imenu korisnika i postavi ga u prikazu aktivnosti. Potom pod granom *Subscribers->korisnikovo ime* u bazi podataka pregledava broj korisnika i postavlja u prikazu broj pretplatnika. Zatim pregledava u bazi podataka granu *profilePics->korisnikovo ime* zbog provjere ima li korisnik profilnu fotografiju. Ako je ima postavlja je u prikazu, a ako ne postavlja zamjensku ikonu za profilnu fotografiju. Naposljetku, postavlja *RecyclerView* za proizvode korisnika. *RecyclerView* za proizvode korisnika je postavljen na isti način kao i u *HomeFragment* fragmentu. Jedina razlika je što se ovdje ne može brisati proizvode jer se pregledavaju proizvodi drugih korisnika. Stoga je u kodu jedina razlika što se koristi drugi adapter (*ProductShowUserAdapter*) od onog u *HomeFragment* fragmentu.

Osim *setUserData* metode, prilikom stvaranja aktivnosti poziva se još jedna važna metoda naziva *checkIfSubscribed*. Ova metoda prvo iz baze podataka dohvaća korisničko ime trenutnog korisnika. Nakon toga provjerava je li korisnik pretplaćen na korisnika čiji se profil

prikazuje. S obzirom na pretplatu, postavlja se prikaz gumbova za *PRETPLATI SE* i *OTKAŽI PRETPLATU*.

Ako korisnik klikne na gumb *PRETPLATI SE* izvodi se metoda *subscribe*. Ova metoda dodaje korisnika u bazi kao pretplatnika na korisnika čiji profil pregledava. Suprotno radi metoda *unsubcribe* pri klikanju gumba *OTKAŽI PRETPLATU*.

4.4. Proizvodi

Proizvodi su jedan od najbitnijih elemenata ove aplikacije. Pomoću proizvoda prikazujemo sve što jedan korisnik ima u ponudi. Korisnik može dodavati i brisati proizvode. Svi korisnici mogu pretraživati i pregledavati proizvode svih ostalih korisnika. Svaki proizvod treba imati svoje ime, opis te cijenu. Korisnik ima mogućnost i dodavanja slike proizvoda ako želi, ali nije obavezno. Za prikazivanje proizvoda koristimo dva *RecyclerView-a*. U *HomeFragment* fragmentu prikazane proizvode možemo brisati, dok u *ShowUserActivity* aktivnosti ne možemo. Proizvod (engl. *product*) ima svoju POJO klasu koja sadrži sve bitne atribute za prikazivanje u aplikaciji.

4.4.1. Product

Product je POJO klasa koja predstavlja proizvod. Sadrži *gettere* i *settere* za svaki od atributa. Klasa je korištena pri prikazvanju proizvoda u aplikaciji kao i za spremanje podataka o proizvodu u bazu podataka. Ako slika proizvoda postoji dodijeliti će se URL slike atributu *imageUri*, a u suprotnom dodijeliti će se string *'null'*.

4.4.2. AddProductFragment

Ovaj fragment je odgovoran za upisivanje podataka o imenu, opisu i cijeni novog proizvoda te prosljeđivanje na daljnju obradu *ImageActivity* aktivnosti. Sadrži polja za upis imena, opisa i cijene novog proizvoda te gumb *DODAJ*.

Klikom na gumb *DODAJ* poziva se metoda *next* koja provjerava ispravnost podataka i šalje ih *ImageActivity* aktivnosti na obradu.

4.4.3. ImageActivity

Ova aktivnost je zadužena za dodavanje slike proizvodu, prenošenje podataka o proizvodu na bazu zajedno sa slikom te brisanje i dodavanje profilne slike. Po odgovornostima je najveća aktivnost u aplikaciji. Sadrži ikonu za odabiranje slike s uređaja, gumb *DODAJ* , gumb *NAZAD* te ikonu za brisanje profilne slike koja je vidljiva samo kada se pozove prikaz aktivnosti iz *HomeFragment* fragmenta.

Klikne li korisnik na ikonu za odabiranje slike, pokrenit će se metoda *openFileChooser*. Pomoću te metode korisnik otvara pretraživač slika na svome mobitelu te odabire sliku koju želi odabrati. Metoda se jednako pokreće i za odabiranje slike profila i za odabiranje profilne slike. Kada korisnik odabere sliku pokreće se *OnActivityResult* metoda u kojoj dohvaćamo URI slike koju je korisnik odabrao. Slika se potom prikazuje u sredini aktivnosti.

Ako je aktivnost pokrenuta radi postavljanja profilne slike, kad korisnik klikne gumb *DALJE* pokreće se metoda koja sprema URI slike u bazu podataka pod granom *profilePics->korisnikovo ime,* a samu sliku sprema u Firebase Storage pod granu *profilePics->korisnikovo ime +“.“+ ekstenzija*.

U slučaju da korisnik dodaje sliku proizvoda i klikne gumb *DALJE,* pozvat će se metoda *addProduct* koja će provjeriti je li korisnik odabrao sliku proizvoda te poziva s obzirom na to metodu *add* kojoj za argument da je string *'null'* ako je korisnik nije odabrao sliku, a ako je predaje URI slike. Metoda *add* potom dodaje podatke o proizvodu u bazu podataka te šalje obavijesti svim pretplatnicima da je korisnik dodao novi proizvod. Obavijest pojedinačnom pretplatniku se šalje metodom *sendNotification.* Više o obavijestima u poglavlju o istima. Važno je napomenuti da se pri stvaranju aktivnosti poziva metoda *getSubscribersTokens* koja dohvaća sve tokene pretplatnika i sprema ih u listu *tokens*. Lista *tokens* se kasnije koristi pri slanju obavijesti.

Ako je korisnik u aktivnosti radi brisanja profilne fotografije i klikne na gumb *IZBRIŠI SLIKU*, poziva se metoda *deleteProfilePic* pomoću koje se briše trenutna korisnikova profilna fotografija iz baze podataka.

Ako korisnik klikne na gumb *NAZAD,* odvede ga na *UserProfileActvivity* aktivnost. U slučaju da korisnik odustane od dodavanja slike i odluči promijeniti neke informacije o porizvodu, gumb *NAZAD* ga odvodi preko *UserProfileActivity* aktivnosti u *AddProductFragment* fragment sa podacima koje je već napisao upisanim u polja. Kako

UserProfileActivity aktivnost prebaci korisnika na *AddProductFragment* fragment pogledajte u poglavlju *4.2.4. UserProfileActivity*.

4.4.4. ProductAdapter i ProductShowUserAdapter

ProductAdapter i *ProductShowUserAdapter* su dva adaptera pomoću kojih se prikazuju proizvodi u *HomeFragment* fragmentu i u *ShowUserActivity* aktivnosti. *HomeFragment* fragment koristi *ProductAdapter* dok *ShowUserActivity* aktivnost koristi *ProductShowUserAdapter*. Razlika između dva prikaza je ta što u *HomeFragment* fragmentu se mogu brisati proizvodi dok se u *ShowUserActivity* aktivnosti ne mogu. Razlog tome je što se u *HomeFragment* fragmentu prikazuju proizvodi trenutnog korisnika koji ima mogućnost brisanja svojih proizvoda, dok se u *ShowUserActivity* aktivnosti prikazuju proizvodi drugih korisnika koje ne bi smio uređivati.

ProductAdapter u metodi *onBindViewHolder* postavlja listenere na profilnu sliku, sliku proizvoda te na ikonu za brisanje proizvoda.

Također u toj metodi se postavlja prikaz opisa, cijene i imena elementa proizvoda. Pri stvaranju elmenta pregledava se atribut proizvoda zadužan za sliku proizvoda te se slika proizvoda namješta u prikazu. Profilna fotografija se namješta pomoću metoda *getProfilePic* i *setProfilePic*.

Ako korisnik klikne na profilnu sliku ili sliku proizvoda, slika se rotira za kut od 90°. Ako korisnik klikne na ikonu za brisanje proizvoda poziva se metoda *deleteProduct* koja briše proizvod iz baze, a samim time i iz *RecyclerView*-a.

ProductShowUserAdapter radi sve isto što i *ProductAdapter* osim što nema funkcionalnost brisanja proizvoda.

4.5. Poruke

Poruke su jedini način komunikacije među korisnicima. Služe za dogovor korisnika o kupnji porizvoda ili prenošenja drugih bitnih informacija. Korisnik unutar *MessageFragment* fragmenta vidi popis svih korisnika s kojima ima poruke. Da bi započeo komuikaciju s nekim korisnikom potrebno je unutar *ShowUserActivity* aktivnosti klikuti na ikonu za poruke. Zatim se otvori *MessageActivity* aktivnost koja pokupi podatke s kojim korisnikom se trenutni korisnik želi dopisivati. Korisnik zatim komunicira s drugim korisnikom preko

MessageActivity aktivnosti, a nakon prve poruke sugovornika se doda u popis ljudi s kojima se trenutni korisnik dopisivao unutar *MessageFragment*-a. Nakon svake poslane poruke sugovorniku dođe obavijest o zaprimljenoj novoj poruci. Poruka (engl. *message*) kao i proizvod ima svoju POJO klasu.

4.5.1. Message

Message je POJO klasa (Slika 3.49.) koja sadrži sve bitne podatke za jednu poruku. Atributu *type* je pridružen string konačne vrijednosti *'message'* koji je potreban za razlučivanje vrste obavijesti koja se prima sa strane primatelja tj. radi li se o obavijesti za poruku ili za novi proizvod. Atribut *messsageType* predstavlja tip poruke koju treba spremiti na bazu podataka. Ako je atributu pridružena vrijednost *'receive'*, korisniku će se prikazat poruka kao poruka koju prima. Ako je atributu pridružena vrijednost *'send'*, korisniku će se prikazat poruka kao poruka koju je poslao. Atribut *sender* predstavlja korisnika koji je poslao poruku, a atribut *receiver* primatelja poruke. Atriburt *id* služi kod soritranja poruka unutar *RecyclerView*-a za poruke. Atribut *body* predstavlja sadržaj poruke.

4.5.2. MessageFragment i InboxAdapter

Unutar *MessageFragment* fragmenta prikazani su svi korisnici s kojima trenutni korisnik ima barem jednu poruku (sugovornici). Prikaz sugovornika je omogućen pomoću *RecyclerView*-a koji koristi *InboxAdapter* kao svoj adapter.

InboxAdapter ima sve elementte prikaza kao i *UserAdapter*. Jedina razlika je što ovaj adapter sadrži i ikonu za brisanje poruka. Kada se klikne na nju poziva se metoda za brisanje elmenta nazvana *removeMessages* koja briše sve poruke sugovornika i korisnika unutar baze podataka pod granom *Messages->korisnikovo ime->sugovornikovo ime*. Ako se klikne na element poziva se pokretanje *MessageActivity* aktivnosti koji će prikazati sve poruke korisnika i sugovornika.

Kako bi *MessageFragment* fragment prikazao sugovornike koristi se metoda *setRecycler*. Ova metoda koristi *fetchData* metodu za kupljenje podataka s baze. Pored prikazivanja sugovornika, *MessageFragment* fragment ima i odgovornost za brisanje poruka ako je netko od sugovornika obrisao račun. Kada se izbriše račun sugovornika, poruke koje je on imao s korisnikom i dalje su vidljive korisniku. Metoda *fetchData* upravo riješava i taj problem pregledavajući je li sugovornik još u bazi te ako nije briše poruke koje je korisnik imao s njim

time brišući i element *RecyclerView*-a koji prikazuje dotičnog sugovornika. Ako je sugovornik u bazi dodaje ga u listu sugovornika. Tu je još i metoda *search* pomoću koje možemo filtrirati sugovornike u recycler-u koristeći *SearchView*.

4.5.3. MessageAdapter

MessageAdapter je adapter pomoću kojeg se prikazuju poruke unutar *RecyclerView*-a *MessageActivity* aktivnosti. Koristi *message_item* XML kod za prikazivanje jednog elementa. Jedan element prikaza sadrži dva podprikaza: prikaz za poruku koju je korisnik primio (*cardViewLeft*) i prikaz za poruku koju je korisnik poslao (*cardViewRight*). Unutar *onBindViewHolder* metode oba su prikaza sakrivena dok se ne provjeri koja je vrsta poruke.

Ako je poruka tipa *'receive'*, poruka se postavlja u prikaz *cardViewLeft* metodom *loadReceiverMesssage*. U suprotnom poruka se postavlja u prikaz *cardViewRight* metodom *loadSenderMessage*.

4.5.4 MessageActivity

MessageAcitivity aktivnost ima odgovornost prikazivanja poruka od korisnika i sugovornika. Sadrži ime sugovornika, *SearchView* za filtiriranje poruka, *RecyclerView* za prikaz poruka, polje za upis poruke te ikonu za slanje poruke napisane u polju.

Za prikaz poruka koristi se *RecyclerView* koji koristi *MessageAdapter*. Kao i sve aktivnosti koje koriste *RecyclerView*, i ova ima metode *setRecycler*, *fetchData* i *search*. Metoda *setRecycler* postavlja *RecyclerView*. Važno je primjetiti kako prva poruka *RecyclerView* je zadnja poruka unutar baze i mora biti pri dnu jer je to zadnja poruka poslana te kako fokus *RecyclerView*-a nakon dodane poruke mora također biti prema na dnu. Kako bi fokus ostao na dnu *RecyclerView*-a, unutar stvaranja *LinearLayoutManager*-a zadnji argument za fokus na dno je *true*.

Metoda *fetchData* prikuplja podatke o porukama iz baze i sprema ih u listu poruka. Metoda također prije prikazivanja poruka sortira poruke tako da zadnja poruka bude na dnu *RecyclerView*-a. Metoda *search* filtrira poruke preko *SearchView*-a koji radi na isti način kao i svi ostali *SearchView*-i dosada samo što su ovdje elementi pretraživanja poruke.

Ako korisnik klikne na ikonu za slanje poruke poziva se metoda *sendMessage*. Metoda prvo stvara objekt tipa *Message*. Potom se stvoreni objekt šalje sugovorniku u bazi kao poruka tipa '*receive'* , a korisniku kao poruka tipa *'send'*. Ako je korisnik sam sebi poslao poruku, poruka tipa *'send'* se samo jednom šalje korisniku. Na kraju se šalje obavijest sugovorniku o primljenoj poruci preko *sendMessageNotification* osim ako se korisnik ne dopisuje sam sa sobom.

4.6. Obavijesti

U ovom poglavlju objasnit će se princip rada obavijesti u aplikaciji. Postoje dvije vrste obavijesti: obavijesti o novoj poruci i obavijesti o novom proizvodu. Obavijesti o novoj poruci prima sugovornik kojem je poslana poruka od bilo kojeg korisnika. Obavijest o novom proizvodu dobivaju svi pretplatnici na korisnika koji je dodao novi proizvod.

Prva bitna stavka za obavijesti jest sučelje (engl. *interface*) koje se koristi kod slanja obavijesti. U aplikaciji to je sučelje *APIService* koje unutar sebe ima sadržana zaglavlja (engl. *headers*) i *POST* zahtjev za slanje podataka serveru. Zaglavlja sadrže podatak o tipu podataka koji se šalje serveru (JSON tip podatka) te podatak o ključu kojeg ima Firebase projekt s kojim je povezana aplikacija na dotičnom Firebase serveru. Pomoću metode *SendNotification* slati će se obavijesti na server.

POJO klasa *Data* sadrži sve podatke koji su potrebni za slanje jedne obavijesti serveru. Atribut *type* predstavlja tip obavijesti koji se šalje. Ovaj atribut je *'message'* ako se radi o obavijesti za poruku odnosno *'notification'* ako se radi o obavijesti za novi proizvod. Ostali atributi su: *title* koji predstavlja naslov obavijesti, *message* koji predstavlja sadržaj obavijesti, *sender* koji predstavlja korisnika od kojeg je došla obavijest te *receiver* koji predstavlja primatelja obavijesti.

POJO klasa *MyResponse* sadrži samo atribut tipa *int* i naziva *success* koji predstavlja kod koji server šalje kao odziv na poslani zahtjev.

POJO klasa *NotificationSender* sadrži atribut tipa *Data* naziva *data* koji predstavlja podatke koji se šalju serveru te atribut *to* koji predstavlja put do servera kojem se zahtjev šalje.

POJO klasa *Client* je *singleton* tip klase koji sadrži instancu klijenta preko kojega se komunicira sa serverom. Klijent je zapravo objekt tipa *Retrofit*.

4.6.1. MyFirebaseMessagingService

MyFirebaseMessagingService je klasa kojom se obrađuju podaci o obavijesti koji stignu primatelju te se kreira obavijest koja se onda prikaže primatelju. Bitno je napomenuti da sve obavijesti dolaze na uređaj preko tokena koji je jedinstven za svakog korisnika na Firebase serveru.

Sve podatke o obavijesti koju primatelj primi od servera mogu se dohvatiti u *onMessageReceived* metodi.

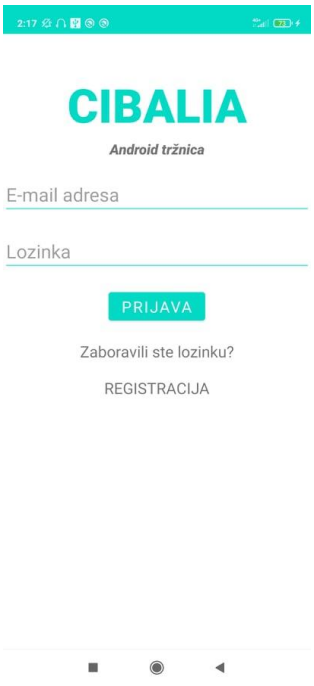
U toj metodi se podaci o obavijesti spremaju u za njih predviđene varijable te se provjerava kojeg je tipa obavijest. Ako je tip za obavijest o novom proizvodu poziva se metoda *pushNotification*, a ako je tip za obavijest o poruci onda se poziva metoda *pushMessageNotification.*

Btino je spomenuti da se obavijest o proizvodu ne može kliknuti dok se obavijest o poruci može. Ako se obavijest o poruci klikne otvori se *MessageActivity* aktivnost koji korisniku nudi mogućnost odgovora na poruku koju je zaprimio.

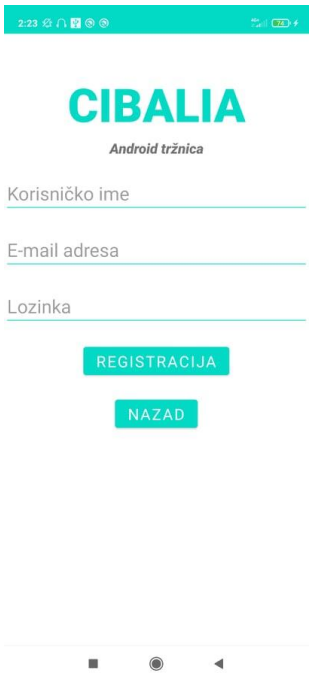
5. PRIKAZ ANDROID APLIKACIJE ANDROID TRŽNICE

U prikazu aplikacije pokazat će se izgled svih aktivnosti i fragmenata uz kratak opis. Najbolji prikaz može se dobiti ako se prati prirodan i očekivan put korištenja aplikacije jednog korisnika koji bi imao interes koristiti istu. Sve počinje od interesa. Korisnik je zainteresiran za korištenje aplikacije jer želi oglašavati svoje proizvode ili kontaktirati oglašivača o proizvodu za kojeg je zainteresiran. U najboljem slučaju korisnik ima oba interesa za aplikaciju.

Kad korisnik otvori aplikaciju prvo se otvara *MainActivity* aktivnost (**Slika 5.1.**). Korisnik da bi koristio aplikaciju prvo se mora registrirati. Registriranje se odrađuje u *RegisterActivity* aktivnosti (**Slika 5.2.**) koji se poziva klikom na gumb *REGISTRACIJA*.



Slika 5.1. MainActivity



Slika 5.2. RegisterActivity

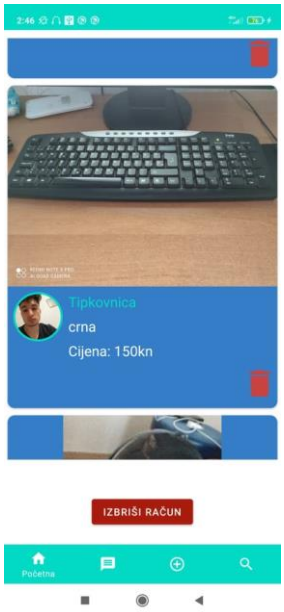
Unutar *RegisterActivity* aktivnosti korisnik popunjava podatke potrebene za registraciju te klikne na gumb *REGISTRACIJA*. Kada se to obavi potrebno je obaviti prvu prijavu unutar *MainActivity* aktivnosti. Kada se korisnik prvi puta prijavi mora otići na svoju adresu e-pošte i potvrditi je. Tek nakon toga prijava u aplikaciju će biti uspješna. U slučaju da korisnik zaboravi

lozinku uvijek može kliknuti na *Zaboravili ste lozinku?* što odvodi na *ResetPasswordActivity* (**Slika 5.3.**) aktivnost gdje će upisao svoju adresu e-pošte te na e-pošti koju dobije od Firebase servera promijeniti lozinku.

Slika 5.3. ResetPasswordActivity

Nakon uspješne prijave korisniku se otvori *HomeFragment* fragment (**Slika 5.4.**, **Slika 5.5.**) gdje može vidjeti svoje proizvode, profilnu fotografiju te podatke o korisničkom imenu, adresi e-pošte i broju pretplatnika. Ako korisnik u *HomeFragment* fragmentu klikne na gumb *ODJAVA* vraća se na *MainActivity* aktivnost te se opet mora prijaviti želi li pristup svom profilu. Klikne li na crveni koš za smeće unutar elementa proizvoda obrisati će taj proizvod.

Slika 5.4. HomeFragment (1)



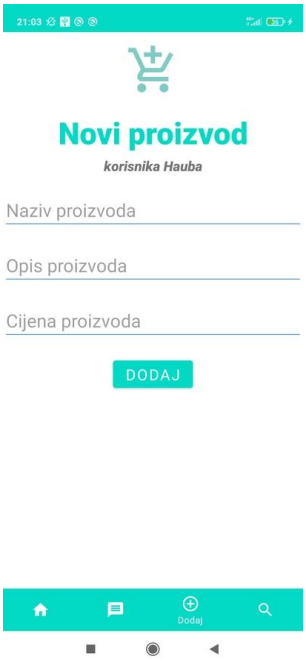
Slika 5.5. HomeFragment (2)

Klikne li korisnik na profilnu fotografiju odvesti će ga u *ImageActivity* aktivnost (**Slika 5.6.**, **Slika 5.7.**) za mijenjanje profilne fotografije. Kada je korisnik zadovoljan sa svojim odabirom slike mora pritisnuti gumb *DALJE*. U sluačaju da želi obrisati profilnu fotografiju korisnik mora pritisnuti gumb *IZBRIŠI SLIKU*.

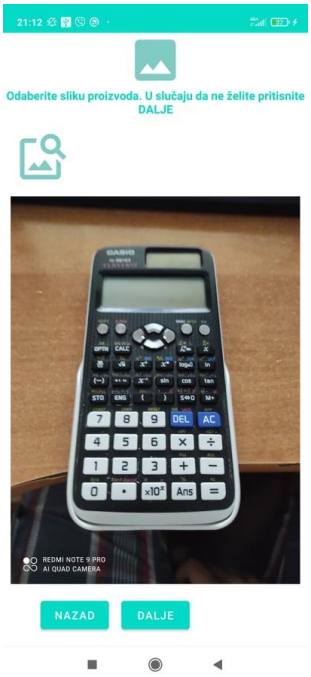
Slika 5.6. ImageActivity za uređivanje profilne fotografije

Slika 5.7. ImageActivity za uređivanje profilne fotografije nakon učitavanja slike

Nakon što korisnik uredi profilnu fotografiju možda se odluči dodati novi proizvod. Klikom na ikonu za dodavanje novog proizvoda u navigaciji otvara se *AddProductFragment* fragment (**Slika 5.8.**). Korisnik potom unosi podatke o proizvodu te mora kliknuti *DODAJ* ako želi nastaviti s dodavanjem novog proizvoda. Nakon klika na gumb *DODAJ* otvara se *ImageActivity* aktivnost (**Slika 5.9.**) gdje korisnik može dodati sliku proizvoda i završiti dodavanje proizvoda klikom na gumb *DALJE*.



Slika 5.8. AddProductFragment



Slika 5.9. Dodavanje slike proizvoda

Ako se korisnik odluči pregledati popis korisnika i vidjeti njihove profile mora kliknuti na ikonu za pretraživanje korisnika. Otvorit će se *SearchFragment* fragment (**Slika 5.10.**) u kojem korisnik može pretražiti korisnike i kliknuti na određenog korisnika da se otvori profil dotičnog. Profil se otvara u *ShowUserActivity* aktivnosti (**Slika 5.11.**) gdje se korisnik može pretplatiti na profil, pregledati proizvode kojeg vlasnik profila sadrži ili poslati poruku korisniku.

Slika 5.10. SearchFragment

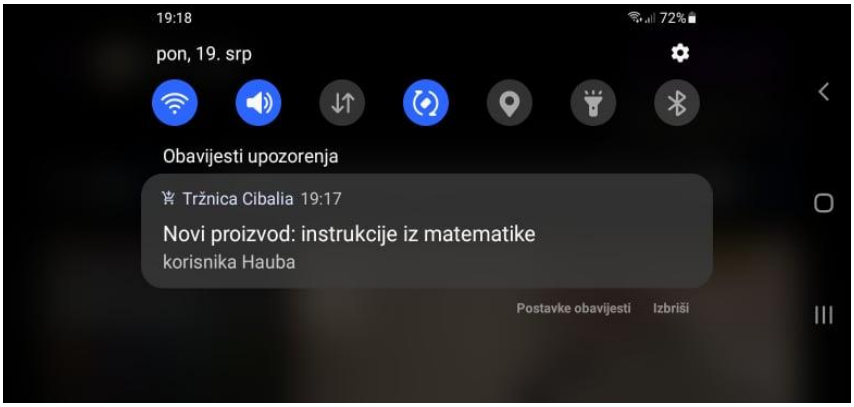
Slika 5.11. ShowUserActivity

Ako korisnik odluči poslati poruku vlasniku profila mora kliknuti na ikonu za slanje poruka čime pokreće *MessageActivity* aktivnost (**Slika 5.12.**). Unutar *MessageActivity* aktivnosti korisnik može komunicirati s vlasnikom profila (sugovornikom). Nakon što korisnik pošalje prvu poruku sugovorniku, unutar *MessageFragment* fragmenta (**Slika 5.13.**) će se pojaviti ime sugovornika. Ako klikne na njega otovriti će mu se opet *MessageFragment* fragment gdje može nastaviti komunikaciju sa sugovornikom. Također korisnik može izbrisati poruke sa sugovornikom ako klikne na ikonu za brisanje poruka unutar *MessageFragment* fragmenta.

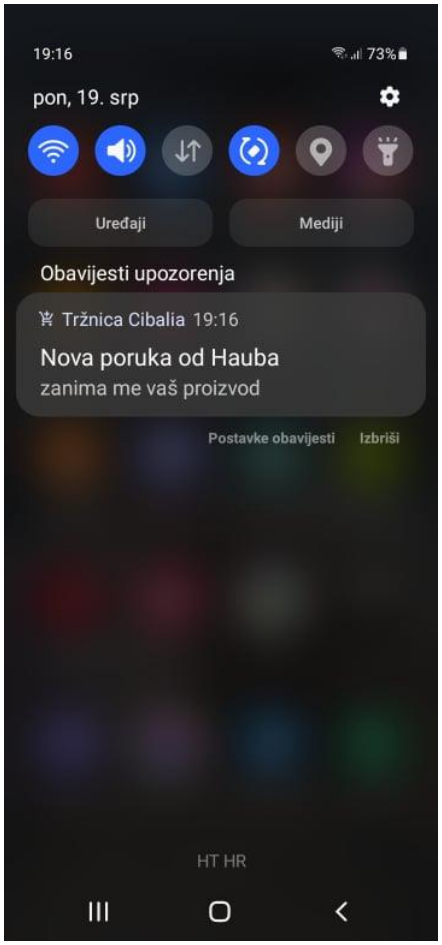
Slika 5.12. MessageActivity

Slika 5.13. MessageFragment

Jedna jako bitna stvar je i izgled obavijesti. U slučaju da korisnik dobije obavijest da je netko od njegovih pretplata dodao novi proizvod obavijest će izgledati kao što prikazuje **Slika 5.14.** Ako korisnik dobije novu poruku od korisnika prikazat ćemu se obavijest kao što prikazuje **Slika 5.15.**



Slika 5.14. Obavijest o novom proizvodu



Slika 5.15. Obavijest o novoj poruci

Na kraju ako korisnik želi obrisati svoj profil, mora kliknuti na gumb *IZBRIŠI RAČUN* (**Slika 5.5.**) unutar *HomeFragment* fragmenta. Otvorit će se *DeleteActivity* aktivnost (**Slika 5.16.**) gdje klikom na gumb *IZBRIŠI RAČUN* briše sve svoje podatke o računu u bazi podataka te se mora ponovo registrirati ako želi opet koristiti aplikaciju.

Slika 5.16. DeleteActivity

6. ZAKLJUČAK

Cilj ovog završnog rad bio je prikazati rad i prikaz aplikacije koja bi imala svrhu android tržnice. Aplikacija mora omogućiti komuniciranje među korisnicima, dodavanje proizvoda, pretplate te obavijesti. Da bi se aplikacija realizirala bilo je potrebno dobro proučiti način rada Firebase-a i Java programskog jezika.

Poruke i proizvodi čine srž ove aplikacije jer se preko njih odvijaju glavne funkcionalnosti aplikacije. Pretplata na korisnika je bila najkompliciranija logika koja se morala implementirati. Ipak najviše vremena je utrošeno u rad s obavijestima. Iako na prvu jednostavan koncept, zbog ponekad zbunjujuće i u dijelovima loše organizacije dokumentacije Firebase-a obavijesti su bile najteži dio aplikacije za napraviti. U pomoć pri realiziranju obavijesti puno je pomogao *Retrofit* API koji je komunicirao sa Firebase serverom i omogućio u konačnici elegantno rješenje za slanje obavijesti na server.

Aplikacija sadrži puno POJO klasa koje su uvelike olakšale spremanje i manipuliranje podatcima unutar baze podataka. *RecyclerView* se pokazao kao vrlo koristan alat za prikaz podataka s baze podataka. XML jezik se pokazao kao koristan alat za uređivanja prikaza aplikacije. *Glide* je bio odličan u pravilnom prikazivanju slika i ikona aplikacije te se pokazao kao vrlo elegantno rješenje.

Za kraj treba reći da je ova aplikacija namjenjena upravo malim proizvođačima, obrtnicima i poduzetnicima kojih u Hrvatskoj ima puno. Takva skupina korisnika nema novaca za velike reklame na televizijama ili drugim oglasima. Ova aplikacija nudi mogućnost brzog komuniciranja preko baze podataka u stvarnom vremenu te pretplata kojima korisnik dobiva potrebne informacije samo od željenih izvora i korisnika. Upravo zato mišljenja sam da bi ova aplikacija imala mjesto na tržištu ako se dorade određeni segmenti poput proizvoda, poruka i pretplata.

LITERATURA

[1] BaaS model usluge na oblaku, dostupno na: https://www.cloudflare.com/learning/serverless/glossary/backend-as-a-service-baas/ , pristupljeno 7. srpnja 2021.

[2] Firebase definicija, dostupno na: https://www.educative.io/edpresso/what-is-firebase , pristupljeno 7. srpnja 2021.

[3] Firebase Cloud Messaging Service (FCM), dostupno na: https://firebase.google.com/docs/cloud-messaging , pristupljeno 7. srpnja 2021.

[4] Java definicija, dostupno na: https://www.guru99.com/java-platform.html , pristupljeno 7. srpnja 2021.

[5] Java u usporedbi s drugim programskim jezicima, dostupno na: https://techterms.com/definition/java , pristupljeno 7. srpnja 2021.

[6] XML programski jezik, dostupno na: https://whatis.techtarget.com/definition/XML-Extensible-Markup-Language , pristupljeno 7. srpnja 2021.

[7] Android Studio, dostupno na: https://developer.android.com/studio/intro , pristupljeno 7. srpnja 2021.

[8] Retrofit, dostupno na: https://square.github.io/retrofit/ , pristupljeno 7. srpnja 2021.

[9] Recycler View, dostupno na: https://developer.android.com/guide/topics/ui/layout/recyclerview , pristupljeno 7. srpnja 2021.

[10] Glide, dostupno na: https://github.com/bumptech/glide , pristupljeno 7. srpnja 2021.

[11] Circle Image View, dostupno na: https://github.com/hdodenhof/CircleImageView , pristupljeno 7. srpnja 2021.

[12] POJO klasa, dostupno na: https://www.geeksforgeeks.org/pojo-vs-java-beans/ , pristupljeno 8. srpnja 2021.

SAŽETAK

Cilj ovog završnog rada bila je izrada aplikacije kao android tržnice. Za programiranje koristio se Java programski jezik. Za oblikovanje prikaza koristio se XML proširivi jezik za označavanje. Firebase BaaS se koristio za bazu podataka, registraciju u aplikaciju te za slanje obavijesti korisnicima. Napravljene su i ostvarene glavne funkcionalnosti aplikacije a to su: funkcionalnost registracije i prijave u aplikaciju, funkcionalnost komuniciranja korisnika, funkcionalnost dodavanja proizvoda te funkcionalnost pretplate na druge korisnike. U uvodu su se navele glavne zadaće aplikacije. U poglavlju o teorijskoj podlozi kratko su se opisale sve tehnologije korištene u aplikaciji. Zatim je objašnjen način rada aplikacije te prikaz iste. Na kraju je donesen zaključak u obliku kratkog opisa rada. Aplikacija je testirana i sve funkcionalnosti rade bez ikakvih problema ili poteškoća.

Ključne riječi: android, Firebase, Java, pretplata, tržnica

ABSTRACT

ANDROID APPLICATION FOR MARKETPLACE

The goal of this final work was to create an application as an android marketplace. The Java programming language was used for programming. An XML extensible markup language was used to format the display. Firebase BaaS was used for the database, registration in the application and for sending notifications to users. The main functionalities of the application were developed, namely: the functionality of registration and login to the application, the functionality of user communication, the functionality of adding products and the functionality of subscribing to other users. The main tasks of the application were stated in the introduction. The theoretical background chapter briefly described all the technologies used in the application. Then making of the application is explained and its display. Finally, a conclusion was reached in the form of a brief description of the work. The application has been tested and all functionalities work without any problems or difficulties.

Key words: android, Firebase, Java, marketplace, subscription

ŽIVOTOPIS

Kristijan Haubrich rođen je 19.9.1998. godine u Vinkovcima. 2005. godine počeo je osmogodišnje osnovnoškolsko obrazovanje u školi Josipa Kozarca u Vinkovcima. Nakon osam godina upisuje Tehničku školu Ruđera Boškovića Vinkovci, smjer Tehnička gimnazija. Godine 2016. nastupa u Međužupanijskom natjecanju učenika u obrazovnom sektoru Elektrotehnika i računalstvo u disciplini Osnove elektrotehnike i mjerenja u elektrotehnici. Maturu je pisao 2017. godine te upisao Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek, koji trenutno pohađa. Dobro se služi programskim jezicima kao java, C, C++ te zna koristiti bazu podataka. Posjeduje znanje engleskog jezika za čitanje i pisanje te se bavi sportom od rane dobi. Posjeduje vozačku dozvolu B kategorije.

Kristijan Haubrich