Android aplikacija za vođenje stambene zgrade

Milošević, Antonio

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

Sveučilišni diplomski studij Računarstvo

ANDROID APLIKACIJA ZA UPRAVLJANJE STAMBENOM ZGRADOM

Diplomski rad

Antonio Milošević

Osijek, 2024.

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

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 15.02.2024.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Antonio Milošević
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1145R, 13.10.2020.
OIB studenta:	67833186579
Mentor:	izv. prof. dr. sc. Mirko Köhler
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Ivica Lukić
Član Povjerenstva 1:	izv. prof. dr. sc. Mirko Köhler
Član Povjerenstva 2:	Miljenko Švarcmajer, mag. ing. comp.
Naslov diplomskog rada:	Android aplikacija za vođenje stambene zgrade
Znanstvena grana diplomskog rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak diplomskog rada:	Zauzeto za studenta Antonio Milošević. Potrebno je napraviti aplikaciju za Android operacijski sustav koja će omogućiti vođenje više stambene zgrade. Administrator zgrade dodaje korisnike (suvlasnike) u aplikaciju. Aplikacija pruža korisnicima međusobnu komunikaciju, objavu slika, prijedloga i primjedbi. U aplikaciji se nalaze i osnovne informacije o zgradi kao što su kućni red i važniji datumi za održavanje zgrade. Može se voditi evidencija o sastancima stanara i omogućiti virtualni sastanak s mogućnošću glasanja o pojedinim prijedlozima.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	15.02.2024.

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, 20.03.2024.

Ime i prezime studenta:	Antonio Milošević
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1145R, 13.10.2020.
Turnitin podudaranje [%]:	15

Ovom izjavom izjavljujem da je rad pod nazivom**: Android aplikacija za vođenje stambene zgrade**

izrađen pod vodstvom mentora izv. prof. dr. sc. Mirko Köhler

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

Kako bi bili u korak s današnjim tehnologijama i kako bi se razmjena informacija obavljala što je moguće lakše i brže, tako svakoga dana na tržište dolaze nove mobilne aplikacije koje nam omogućuju brže, bolje i lakše komuniciranje te razmjenu informacija. Trenutno doba u kojemu se nalazimo je doba u kojemu svatko posjeduje mobilni uređaj i baš iz tog razloga se mobilne aplikacije u velikom obujmu mogu iskoristiti za poboljšanje i pojednostavljenje razmjene informacija.

Aplikacija koju ovaj diplomski rad opisuje omogućuje brzu i laku razmjenu informacija, omogućuje jednostavnije planiranje dana kao i bolju organizaciju vremena, sve to jer su informacije na dohvat ruke, unutar same aplikacije.

U drugom poglavlju dana su trenutna i aktualna rješenja koja se koriste za razmjenu informacija i komunikaciju stanara unutar zgrade. Treće poglavlje donosi opis korištenih alata i tehnologija kako bi aplikacija mogla biti uspješno izrađena. Četvrto poglavlje objašnjava strukturiranje i arhitekturu same aplikacije unutar Android Studio razvojnog okruženja. Peto poglavlje prikazuje izgled aplikacije i koda kako bi ona mogla ispravno funkcionirati.

1.1. Zadatak diplomskog rada

Zadatak ovog diplomskog rada je napraviti mobilnu aplikaciju za Android uređaje čija je funkcionalnost vođenje stambene zgrade. Administrator zgrade dodaje korisnike u aplikaciju, te aplikacija omogućuje međusobnu komunikaciju korisnika, objavu prijedloga i primjedbi. Unutar same aplikacije se nalaze i osnovne informacije o stambenoj zgradi kao što su kućni red i važniji datumi za održavanje zgrade. Mogu se voditi sastanci stanara.

2. TRENUTNA RJEŠENJA

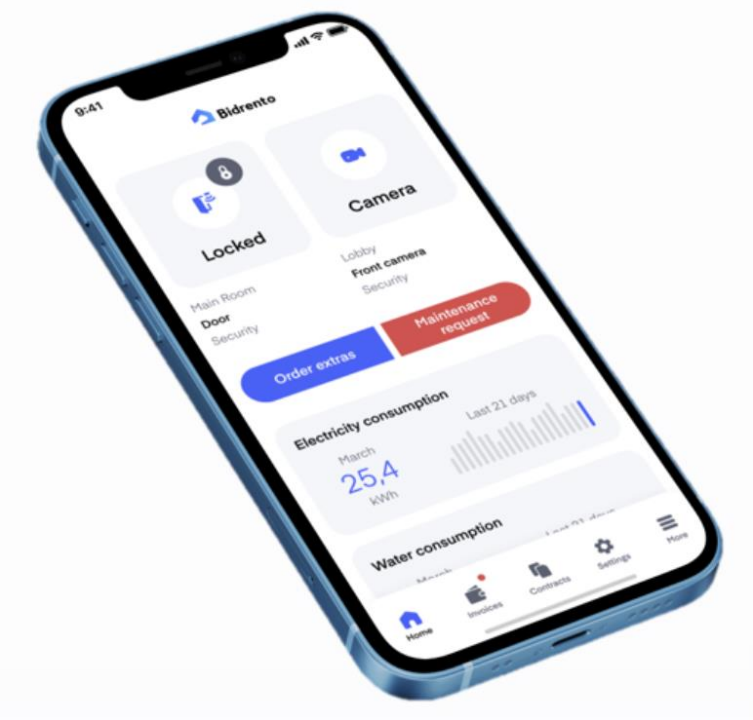
U ovom poglavlju je dan pregled aktualnih rješenja. To su najčešće aplikacije za razmjenu poruka ili aplikacije društvenih mreža, ali postoje i ciljane aplikacije za ovaj problem.

2.1. Pregled aplikacija

Kako bi se ostvarila komunikacija među stanarima i olakšala razmjena informacija, trenutno je najčešće rješenje to da se prave Viber ili WhatsApp grupe unutar kojih se nalaze stanari zajedno s predstavnikom. Ovo rješenje omogućuje komunikaciju i razmjenu informacija, ali isto tako se stanari mogu lako izgubiti u svim prijašnjim porukama. Može doći do toga da se ne zna jesu li koje prijavljene štete popravljene ili nisu. Zato je dobro imati specijalizirano rješenje za konkretan problem.

Postoje i društvene mreže u kojima su gore navedeni problemi riješeni, ali one se naplaćuju i ne nude rješenja koja su tražena u zadatku diplomskog rada.

Aplikacija koja se nalazi na tržištu je „Bidrento“ i omogućuje komunikaciju za stanare i stanodavce te je stvorena kako bi osmislila upravljanje nekretninama za najam i učinila je mnogo jednostavnijim. „Bidrento“ je aplikacija za stanare koja rješava najčešće izazove korištenja s kojima se stanodavci svakodnevno suočavaju, a to su režije, fakture, ugovori o najmu, zahtjevi za održavanjem, razmjenom informacija i ostalim. Korisnik može pratiti svoju potrošnju vode, struje i ostalih režija kako bi ta potrošnja bila što optimalnija.



Slika 1.1. *Izgled „Bidrento“ aplikacije [1]*

3. OPIS KORIŠTENIH ALATA I TEHNOLOGIJA

Ovo poglavlje će sadržavati opis tehnologija koje su korištene pri izradi aplikacije.

3.1. Android operacijski sustav

Otvoreni operacijski sustav za mobilne uređaje, kao što su pametni telefoni i tableti, američke tvrtke Google Inc., temeljen je na Linux jezgri. Android je isprva razvijala tvrtka Android Inc., koju je Google kupio 2005. godine. Android je prvi puta predstavljen javnosti 2007. godine, a prvi uređaji za Android operacijski sustav su se pojavili u rujnu 2008. godine [9].

Također, Google je razvio sustav za pametne televizore, zvan Android TV. Isto tako, za automobile koji se zove Android Auto. Osim toga, postoji i Wear OS koji se može ugraditi u uređaje poput pametnih satova.

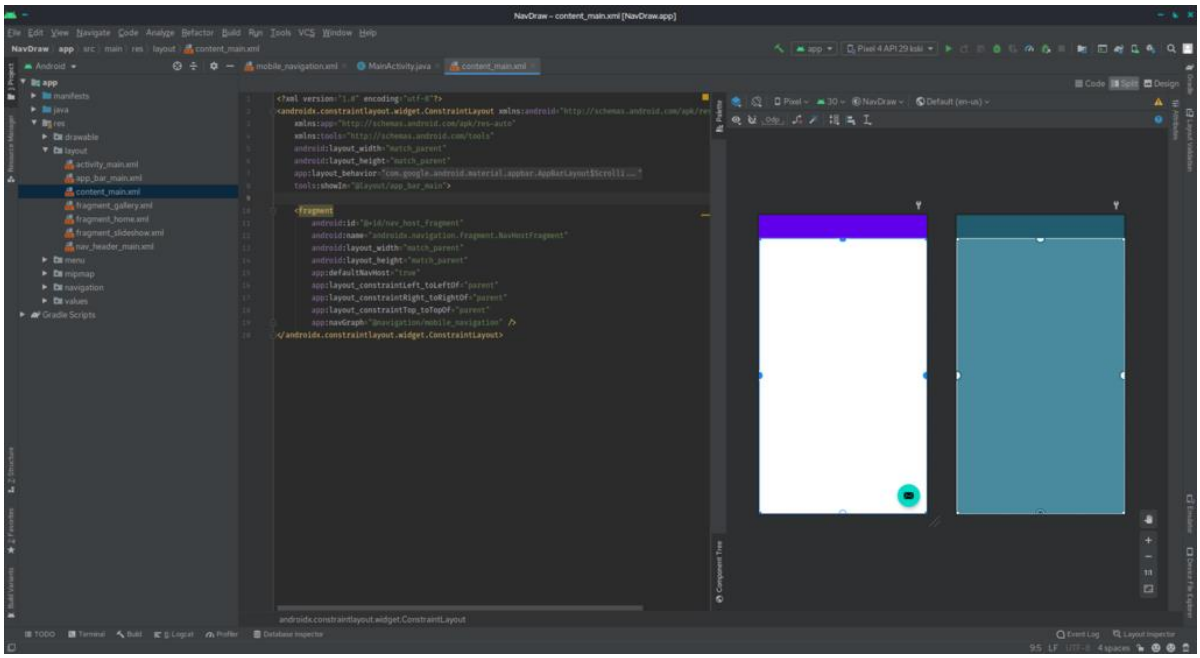
Android operacijski sustav, odnosno sama platforma je prilagođena za korištenje na uređajima s većim zaslonima, odnosno na pametnim telefonima, jer su stariji mobilni uređaji uglavnom imali manje zaslone. Novije generacije mobilnih uređaja koriste 2D ili 3D grafičku knjižicu koja je temeljena na OpenGL ES 2.0 specifikacijama, knjižica koja je pisana u C++ programskom jeziku radi boljeg manipuliranja memorijom i resursima [9].

Arhitektura Android operacijskog sustava je zasnovana na Linux 2.6 jezgri koja je pisana u C i C++ programskim jezicima, to su programski jezici niske razine, odnosno jezici koji pružaju jako malo ili nimalo apstrakcije mikroprocesora.

Android aplikacije se mogu pisati u programskom jeziku Java, novije aplikacije se većinom pišu uz pomoć programskog jezika Kotlin. Kako bi izrada Android aplikacije bila što jednostavnija, koristi se Android programsko razvojno sučelje, odnosno Android Software Development Kit (SDK).

3.2. Android Studio

Android Studio je službeno integrirano razvojno okruženje za Googleov operacijski sustav Android, izgrađen je na JetBrainsovom IntelliJ IDEA softveru i dizajnirano posebno za razvoj Android aplikacija.



Slika 3.1. *Android Studio korisničko sučelje [2]*

Android Studio je najavljen 16. svibnja 2013. godine na Google I/O konferenciji. Bio je u fazi pregleda ranog pristupa počevši od verzije 0.1 u svibnju 2013., a zatim je ušao u beta fazu počevši od verzije 0.8 koja je objavljena u lipnju 2014. godine. Dana 7. svibnja 2019. godine Kotlin je zamijenio Javu kao Googleov preferirani jezik za razvoj Android aplikacija. Java je još uvijek podržana kao i C++.

Integrirano razvojno okruženje Android Studio podržava iste programske jezike kao i IntelliJ. Jednom kada je aplikacija pokrenuta, ona može biti postavljena na Google Play Store kako bi šira javnost mogla koristiti napravljenu aplikaciju.

Android Studio razvojno okruženje se može pokretati na Windows, macOS i Linux operacijskim sustavima.

3.3 Kotlin

Kotlin je višeplatformski, statički tipiziran programski jezik opće namjene. Kotlin je dizajniran za potpunu interakciju s Javom. Voditelj razvoja Andrey Breslav je rekao da je Kotlin dizajniran da bude objektno orijentiran jezik industrijske snage i bolji jezik od Jave, ali da i dalje bude u potpunosti interoperabilan s Javom, omogućujući tvrtkama postupnu migraciju s Jave na Kotlin [3].

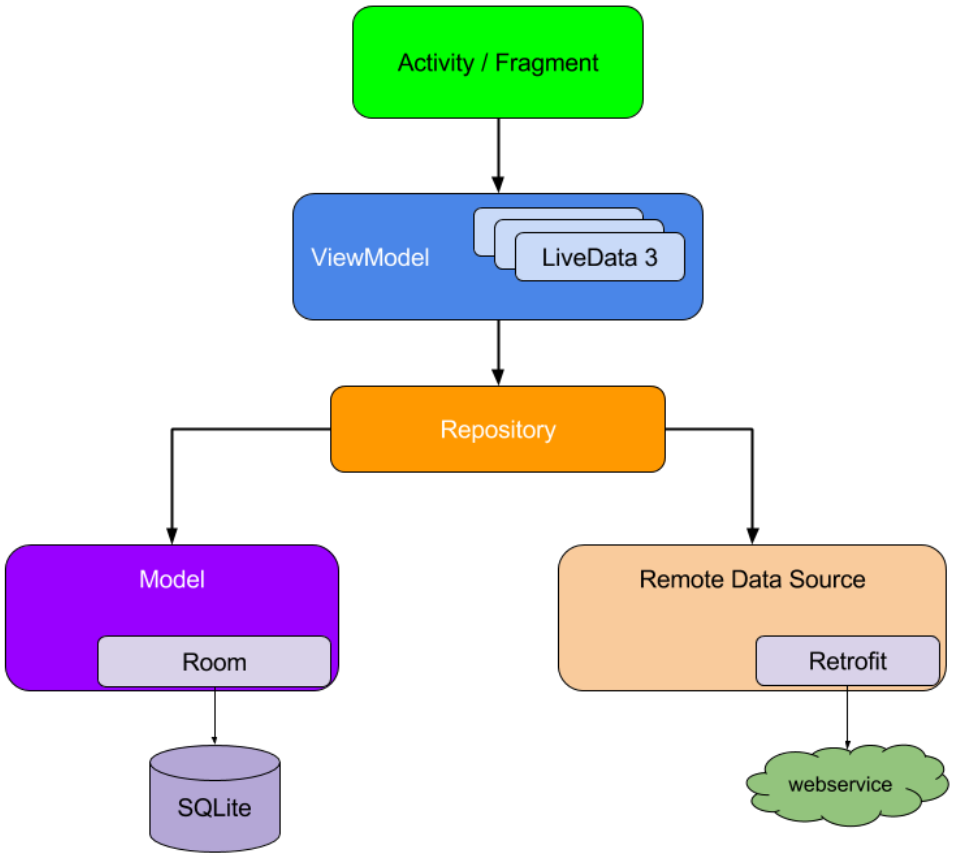
Slika 3.2. *Kotlin Logo [3]*

Točke sa zarezom nisu obavezne na kraju linije kôda, u većini slučajeva je dovoljan novi redak da prevoditelj zaključi da je naredba završila. Kotlin deklaracije varijabli i popisi parametara imaju tip podatka iza naziva varijable koji se odvaja s dvotočkom.

Svojstva u Kotlinu mogu biti samo za čitanje, deklarirane ključnom riječi „val“ ili promjenjive, deklarirane ključnom riječi „var“.

3.4. MVVM arhitekturni obrazac

MVVM (Model-View-ViewModel) je arhitekturni obrazac koji razdvaja aplikaciju na više komponenti tako da svaka komponenta ima svoje specifične odgovornosti. Pri korištenju MVVM obrasca programski kôd je razdvojen na tri dijela: View, ViewModel i Model. Ova arhitektura je preporučena od strane Googlea kao jedan od najboljih načina strukture kôda Android aplikacija.



Slika 3.3. *MVVM arhitektura [4]*

Osnovne karakteristike MVVM obrasca su te da se UI komponente drže podalje od poslovne logike, dakle sva logika se piše u ViewModel. Poslovna logika se drži podalje od operacija vezanih za bazu podataka. Preporučeni način za komunikaciju između dva sloja je takozvani obrazac Promatrač.

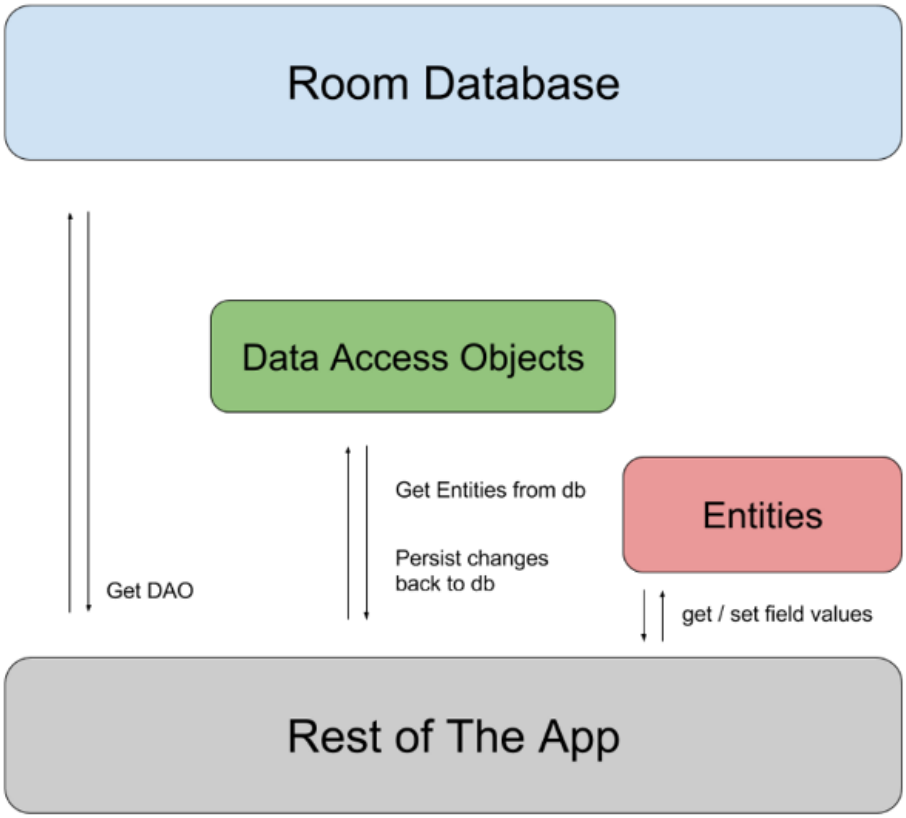
3.5. Kotlin Coroutines

Coroutine su oblikovni obrazac istodobnosti koji se može koristiti na Android sustavu za pojednostavljenje kôda koji se izvršava asinkrono. Coroutine su dodane u Kotlin u verziji 1.3 i temelje se na utvrđenim konceptima iz drugih jezika [12].

Na Android sustavu Coroutine pomažu u upravljanju dugotrajnim zadacima koji bi inače mogli blokirati glavnu nit i uzrokovati da vaša aplikacija ne reagira. Više od 50 % profesionalnih programera koji koriste Coroutine je prijavilo povećanu produktivnost [11].

3.6. Room

Room je biblioteka koja pruža sloj apstrakcije preko SQLitea kako bi se omogućio robusniji pristup bazi podataka uz iskorištavanja pune snage SQLitea. [10] Najčešće se koristi za lokalno spremanje podataka i spremanje veće količine strukturiranih podataka kojima se može pristupiti bez internetske veze. Google preporučuje korištenje Room baze podataka jer uklanja veliku većinu dupliciranog kôda potrebnog za interakciju sa SQLite bazom podataka.



Slika 3.4. *Room struktura [5]*

3.7. Koin

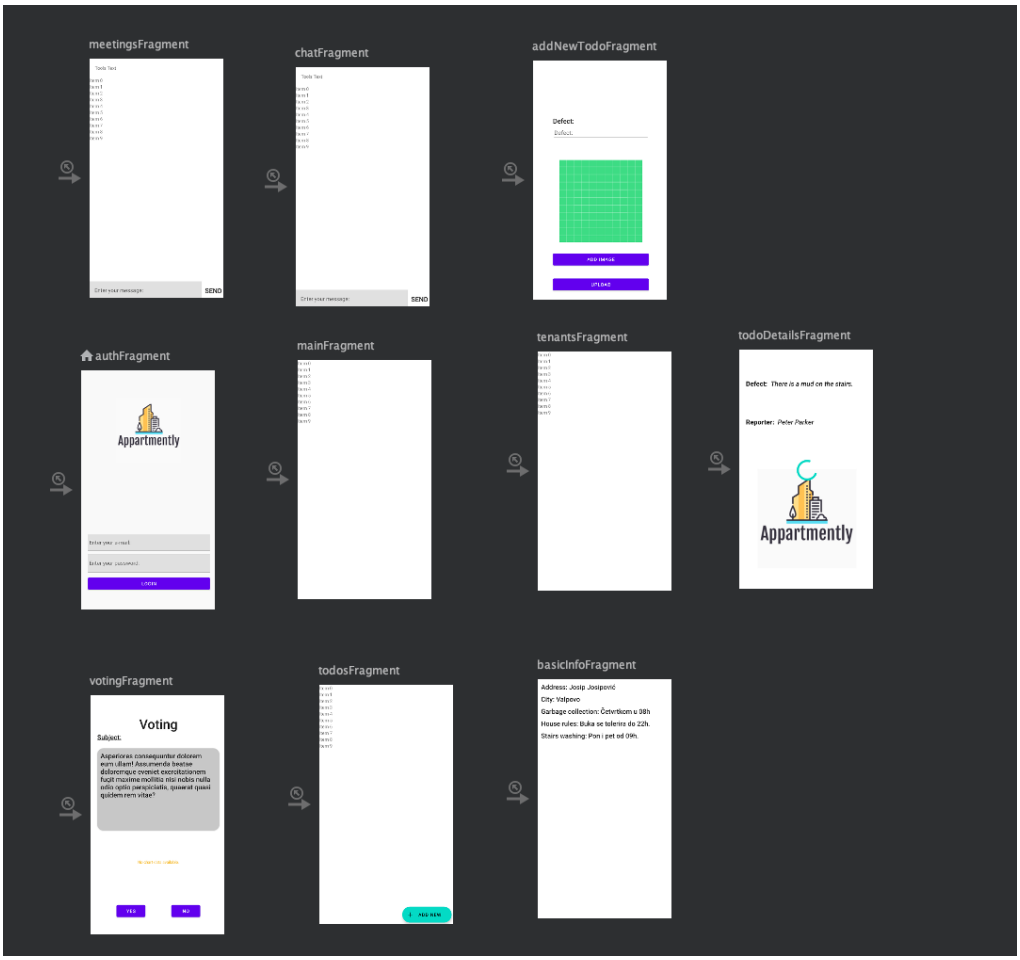
Koin je biblioteka koja se koristi za ubrizgavanje ovisnosti. Biblioteka je napisana u Kotlinu, posjeduje vrlo opsežnu dokumentaciju i poprilično je intuitivna za korištenje [13].

Ubrizgavanje ovisnosti je tehnika koja se široko koristi u programiranju i dobro je prikladna za razvoj Android aplikacija. Slijedeći postojeća načela, postavljate temelje za dobru arhitekturu aplikacije. Implementiranje ubrizgavanja ovisnosti nam pruža ponovnu upotrebu kôda, jednostavnost refaktoriranja i testiranja.

3.8. Navigation Component

Navigation Component biblioteka nam pomaže te olakšava implementaciju navigacijskog grafa unutar aplikacije, isto tako pruža mogućnost uvida u vizualni prikaz navigacije aplikacije [14].

Neke od prednosti koje Navigation Component pruža prilikom korištenja su da se fragmentima rukuje uz pomoć transakcija, rukovanje s akcijama prema naprijed i nazad, pruža resurse koji služe za animaciju i tranzicije, olakšava implementaciju i rukovanje dubokim povezivanjem i sigurni argumenti [15].



Slika 3.5. *Navigacijski graf*

Navigation Component biblioteka se sastoji od navigacijskog grafa, navigacijskog kontejnera i navigacijskog kontrolera. Navigacijski graf je komponenta koja sadrži sve informacije koje su vezane uz navigaciju na jednom mjestu. Navigacijski kontejner je prazan kontejner koji prikazuje destinacije s navigacijskog grafa i navigacijski kontroler omogućuje zamjenu destinacija dok se korisnik kreće kroz aplikaciju.

3.9. Firebase

Firebase Inc. Je skup pozadinskih usluga računarstva u oblaku i platformi za razvoj aplikacije koje pruža Google. U njemu se nalaze baze podataka, usluge, autentifikacija i integracija za razne aplikacije koje uključuju Android, iOS, JavaScript, Node.js, Java, Unity, PHP i C++.

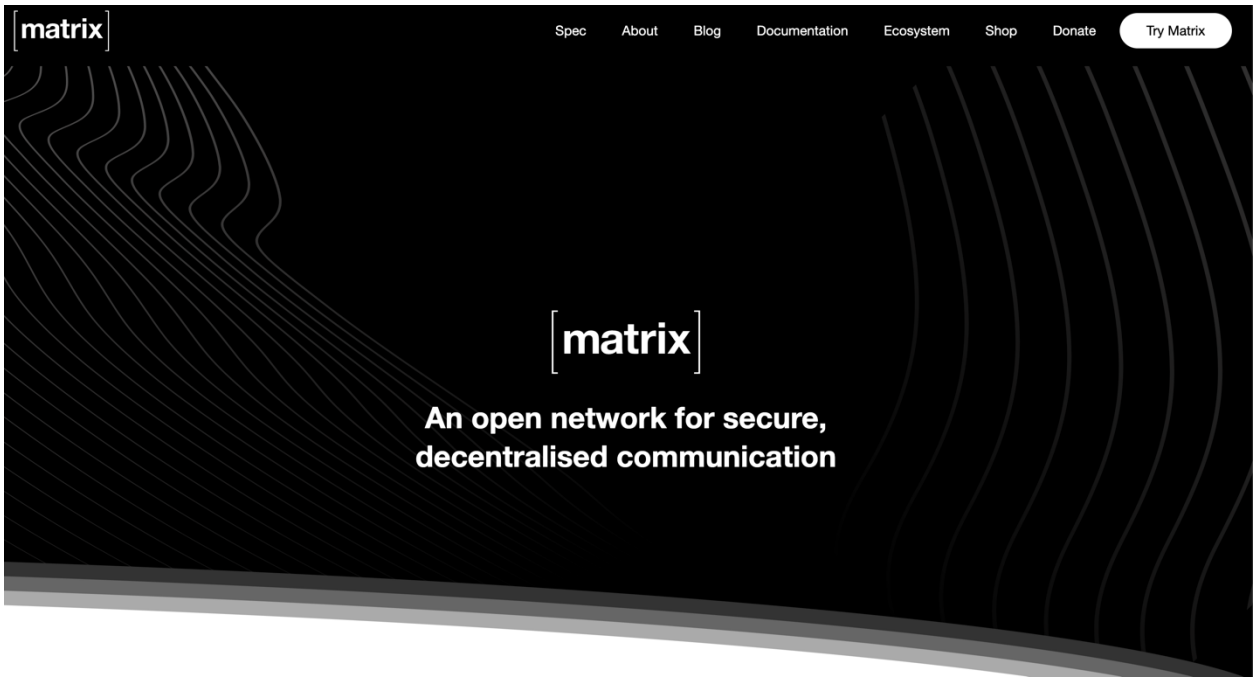
Firebase se razvio iz Envolvea, prethodnog startupa koji su osnovali James Tamplin i Andrew Lee 2011. godine. Envolve je razvojnim programerima pružao API koji omogućuje integraciju funkcionalnosti online razgovora na njihovoj web stranici. Nakon što su lansirali u javnost chat, Tamplin i Lee su otkrili da se ono koristi za prosljeđivanje podataka aplikacijama koje nisu poruke unutar aplikacija za razgovore. Programeri su koristili Envolve za sinkronizaciju podataka aplikacija kao što je stanje igre u stvarnom vremenu među svojim korisnicima.

Slika 3.6. *Firebase Logo [6]*

Kao zasebna tvrtka se Firebase lansirala u javnost u travnju 2012. godine. Njihov prvi proizvod je bila Firebase baza podataka u stvarnom vremenu (Firebase Realtime Database), API koji sinkronizira podatke iOS, Android i Web aplikacija u stvarnom vremenu i pohranjuje ih u Firebaseov oblak.

3.10. Matrix.org

Matrix SDK je otvoreni protokol za decentraliziranu i sigurnu komunikaciju. Omogućuje potpunu kontrolu nad korisničkom komunikacijom te osigurava sigurnost i privatnost. Glavni motiv je taj da komunikacija treba biti dostupna svima kao besplatna i otvorena, neopterećena, standardna i globalna mreža.

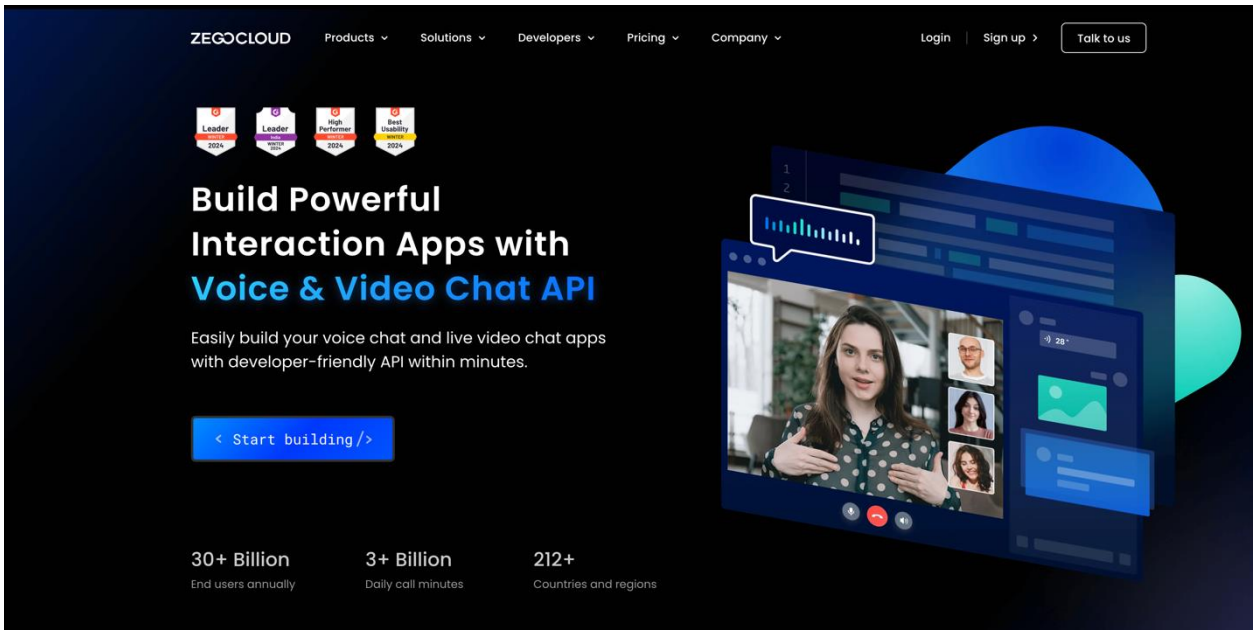


Slika 3.7. *matrix.org početna stranica [7]*

Zaklada Matrix.org postoji kako bi djelovala kao neutralni skrbnik Matrixa i njegovala ga što je učinkovitije moguće kao jedinstveni nefragmentirani standard za veću dobrobit cijelog ekosustava, ne donoseći koristi niti privilegirajući bilo kojeg pojedinačnog korisnika ili podskup korisnika.

3.11. ZEGO Cloud

Zego Cloud je audio i video komunikacijska platforma u stvarnom vremenu koja programerima omogućuje izradu live streaming aplikacija. Uz Zego Cloud programeri mogu kreirati interaktivne aplikacije koje korisnicima omogućuju dijeljenje video i audio sadržaja u stvarnom vremenu.



Slika 3.8. *Početna stranica zegocloud.com [8]*

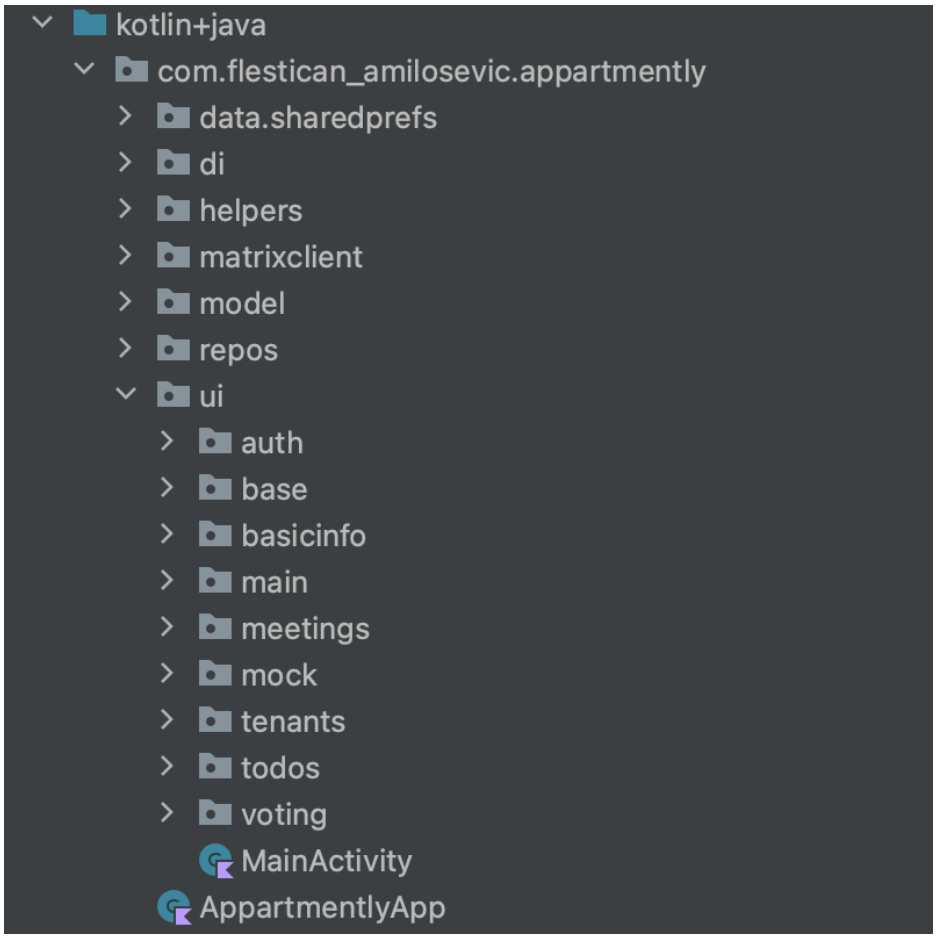
Zego Cloud revolucionira virtualni angažman s uslugama interakcije u stvarnom vremenu za komunikaciju u oblaku bez muke, povezivanje poduzeća, timova, kupaca i korisnika. Biblioteka je napisana u programskom jeziku Java i može se nesmetano koristiti unutar Android projekta koji je pisan u programskom jeziku Kotlin.

4. STRUKTURA I ARHITEKTURA PROJEKTA

Nastavak rada će obuhvaćati opis i prikaz strukture i arhitekture projekta, odnosno strukturu paketa i što se u pojedinima nalazi, te općenita arhitektura.

4.1. Struktura

Kako zahtjevi i kompleksnost značajki koje se nalaze unutar samih aplikacija rastu, to isto tako može dovesti do otežanog snalaženja unutar samog projekta. Razvojno okruženje Android Studio pruža mogućnost kreiranja paketa za raspoređivanje određenih klasa odnosno datoteka koje su međusobno povezane. Paketi predstavljaju direktorije unutar kojih su spremljene datoteke. Zbog lakšeg snalaženja i održavanja projekta, paketi unutar Android projekta bi uvijek trebali biti dobro organizirani.



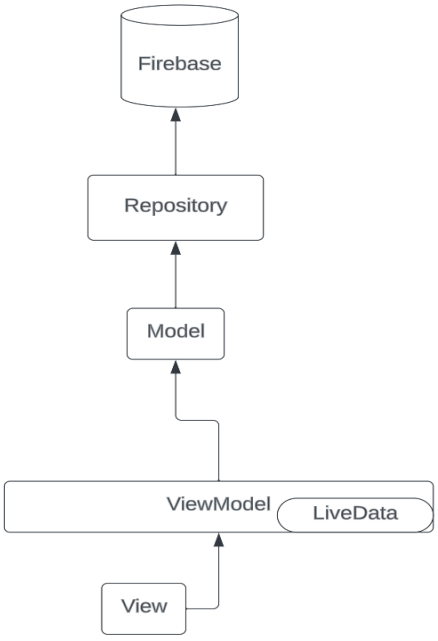
Slika 4.1. *Struktura paketa*

Sa slike 4.1. vidimo kako su paketi raspoređeni u nekoliko grupa, razdvojeno je korisničko sučelje od podatkovnog dijela kako bi se što lakše snašli u istom.

- data.sharedprefs – sadrži „SharedPrefsManager“ klasu za pohranu podataka
- di – sadrži module koji se koriste za ubrizgavanje ovisnosti
- helpers – sadrži sve pomoćne datoteke kao što su konstante, globalni pružatelj dispečera za korutine i ostale pomoćne datoteke
- matrixclient – sadrži implementaciju matrix klijenta
- model – sadrži poslovne modele
- repos – sadrži repozitorije za unos i pohranu
- ui – sadrži sve datoteke i pakete koji su vezani za korisničko sučelje

4.2. Arhitektura

Za izradu aplikacije je korištena MVVM arhitektura, odnosno arhitekturni obrazac. Jedan od trenutno najpopularnijih obrazaca današnjice je sigurno MVVM zbog toga što razdvaja poslovnu logiku od korisničkog sučelja, odnosno na korisničkom sučelju se potrebne stvari samo prikazuju, dok se poslovna logika izvršava u pozadini, odnosno unutar ViewModel-a.



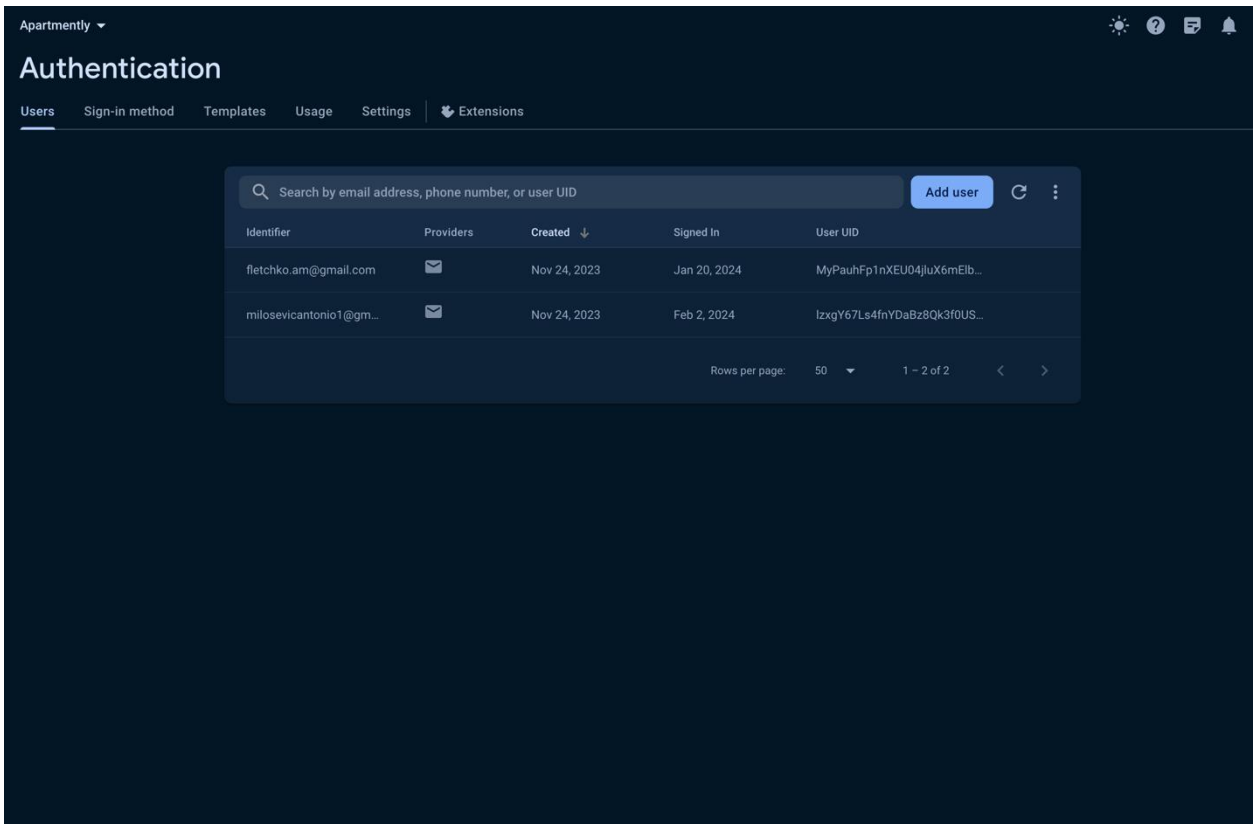
Slika 4.2. *Arhitektura [4]*

5. PROGRAMSKO RJEŠENJE I DIZAJN APLIKACIJE

U nastavku će biti objašnjeno kako aplikacija funkcionira, što se sve trebalo isprogramirati kako bi aplikacija radila onako kako je u početku zamišljeno.

5.1. Kreiranje i prijava korisnika

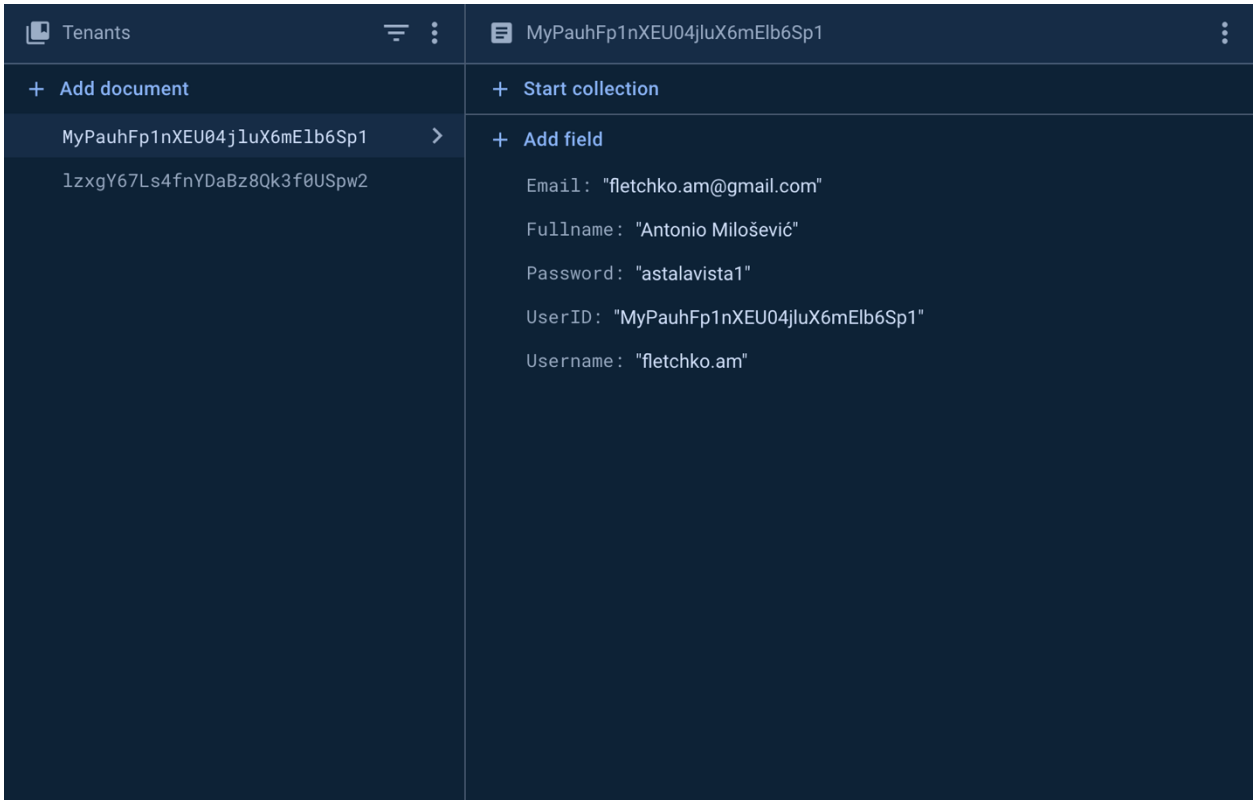
Kako bi se pojedini korisnik mogao prijaviti i ući u aplikaciju, prije toga mu administrator treba kreirati korisnički račun na Firebase konzoli. Od osnovnih informacija koje su potrebne, za početak treba samo e-mail adresa korisnika i lozinka koju bi kreirani korisnik htio imati.



Slika 5.1. *Firebase Authentication konzola*

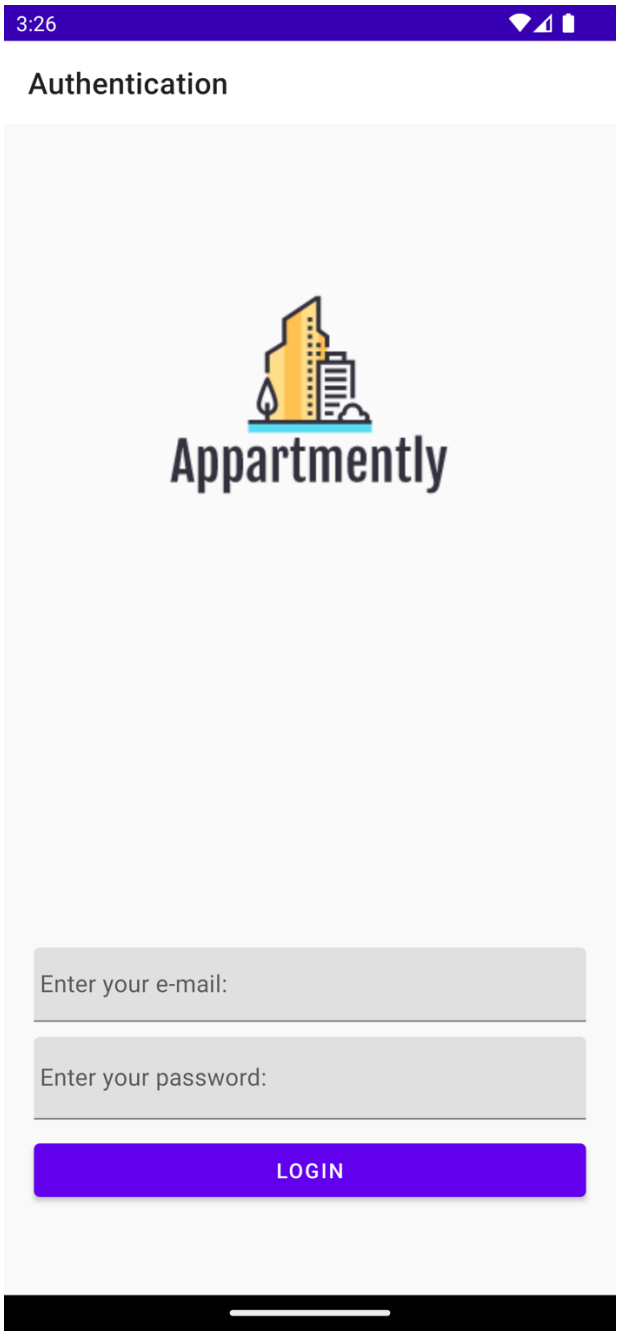
Nakon što je na Firebase konzoli u Authentication dijelu kreiran novi korisnik s e-mail adresom i lozinkom, korisnik se može s gore navedenim podacima prijaviti u aplikaciju. Nadalje, kako bi za svakog kreiranog korisnika imali osnovne informacije, njegov e-mail, ime, prezime,

lozinku, jedinstvenu oznaku i korisničko ime, u nastavku ćemo prikazati sučelje za dodavanje gore navedenih informacija.



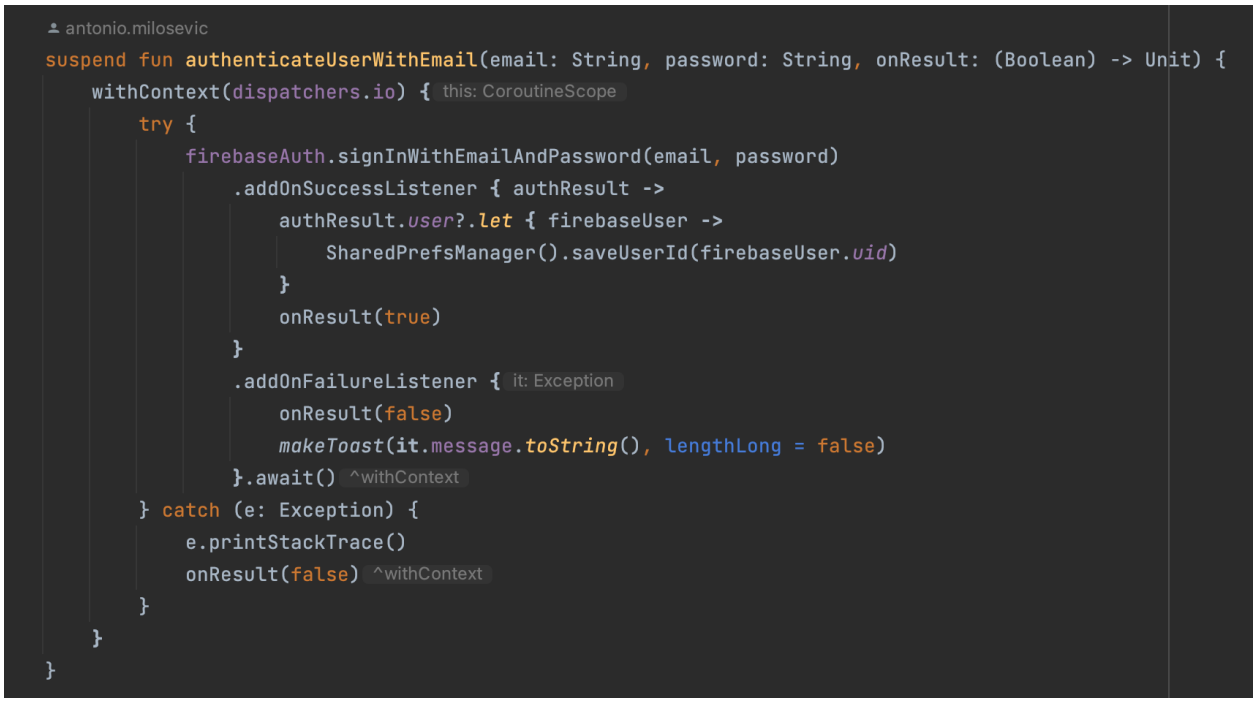
Slika 5.2. *Firebase Firestore dokumenti*

Kada otvorimo aplikaciju, prvi zaslon na koji nailazimo je „AuthFragment“ u kojemu se na vrhu nalazi logo same aplikacije, ispod su dva „EditText“ polja u koja upisujemo svoj e-mail i lozinku. Slika 5.3. prikazuje izgled početnog zaslona.



Slika 5.3. *Izgled „AuthFragment“ početnog zaslona*

Nakon što smo upisali svoj e-mail i lozinku koja nam je dodijeljena ili koju smo sami izabrali, klikom na „LOGIN“ gumb se pokreću naredbe za prijavu korisnika. U nastavku na slici 5.4. možemo vidjeti kod koju to i omogućuje.



Slika 5.4. *Programski kod prijave korisnika u aplikaciju*

U slučaju da prijava korisnika ne uspije, ispisuje se poruka na zaslon aplikacije koja dolazi s Firebase servera. Ako prijava uspije, korisnik se navodi na sljedeći zaslon unutar aplikacije. Kako se glavna nit za korisničko sučelje ne bi blokirala, poziv „authenticateUserWithEmail“ funkcije se izvršava na pozadinskoj „io“ niti koja služi za upisivanje i preuzimanje podataka.

Prilikom svakog novog ulaska u aplikaciju provjerava se je li se korisnik već ranije bio prijavio, tako da se automatski navodi na sljedeći zaslon bez potrebe za ponovnim upisivanjem e-mail adrese i lozinke. Na slici 5.5. možemo vidjeti funkciju koja provjerava je li korisnik već prijavljen.

```
antonio.milosevic
private fun isUserAlreadySignedIn() {
    viewModelScope.launch { this: CoroutineScope
        _isUserSignedIn.value = authRepository.getCurrentUser()?.let { it: FirebaseUser
            initMatrixClient(
                SharedPrefsManager().getUserEmail()!!.substringBefore( delimiter: '@'),
                SharedPrefsManager().getUserPassword()!!
            )
            SharedPrefsManager().saveUserId(it.uid)
            true ^let
        } ?: false
    }
}
```

Slika 5.5. *Programski kod za provjeru prijave korisnika*

Kako bi se mogle pratiti promjene koje se događaju u pozadini, unutar samog „AuthFragment“ zaslona se prati „isUserSignedIn“ svojstvo kako bi se znalo treba li navoditi korisnika na sljedeći zaslon ili ne. Slika 5.6. prikazuje promatranje „isUserSignedIn“ svojstva unutar „AuthFragment“ zaslona.

```
antonio.milosevic
private fun initObservers() {
    authViewModel.isUserSignedIn.observe(viewLifecycleOwner) { isUserSignedIn ->
        if(isUserSignedIn) {
            navigateToMainFragment()
        }
    }
}
```

Slika 5.6. *Programski kod za promatranje „isUserSignedIn“ svojstva*

Navigiranje prema „MainFragment“ zaslonu se izvodi preko „NavController“ klase koja nam služi za navigaciju kroz zaslone aplikacije, u nastavku se može vidjeti kako funkcija izgleda.

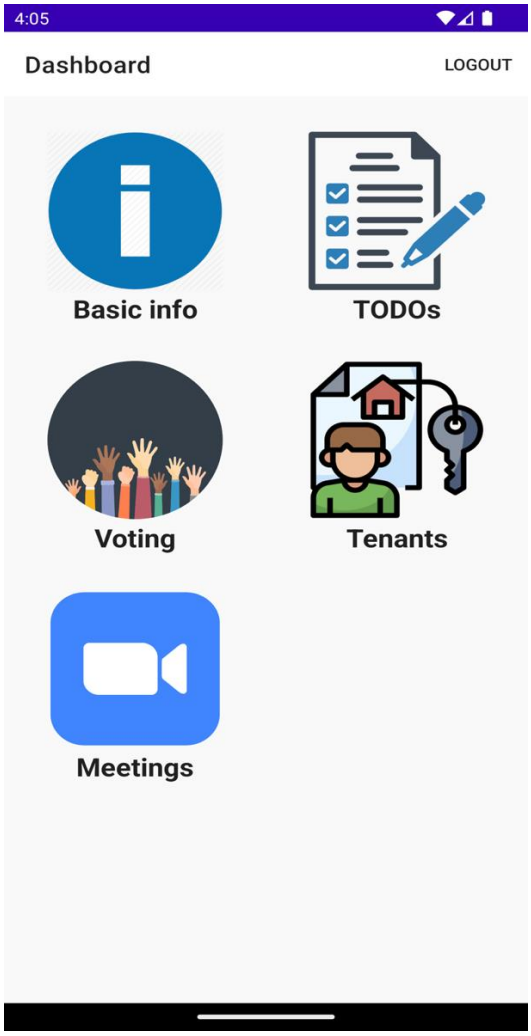
```
antonio.milosevic
private fun navigateToMainFragment() {
    findNavController().navigate(AuthFragmentDirections.toMainFragment())
}
```

Slika 5.7. *Programski kod za navigaciju u „MainFragment“ zaslon*

Gore navedena funkcija je privatna jer nema potreba da se ista može pozivati izvan same klase, odnosno „AuthFragment“ zaslona.

5.2. Zaslon oglasne ploče

Ukoliko se korisnik uspješno prijavi u aplikaciju, slijedi „MainFragment“ zaslon na kojemu korisnik može odabrati koje informacije želi pregledati, jesu li to osnovne informacije o zgradi, stvari koje treba popraviti, aktualno glasanje, čavrljanje s drugima ili ući u grupni sastanak. U nastavku, na slici 5.8., se može vidjeti dizajn gore spomenutog zaslona.



Slika 5.8. *Izgled „MainFragment“ zaslona*

U slučaju da se trenutno prijavljeni korisnik želi odjaviti iz aplikacije, može kliknuti na „LOGOUT“ tekst u gornjem desnom kutu koji će automatski registrirati klik i pozvati funkciju koja će odjaviti korisnika. Na slici 5.9. možemo vidjeti primjer funkcije koja odjavljuje korisnika i na koji način dolazi do toga da se „logout“ funkcija pozove.

```
override val menuProvider: MenuProvider by lazy {
    object : MenuProvider {
        override fun onCreateMenu(menu: Menu, menuInflater: MenuInflater) =
            menuInflater.inflate(R.menu.main_menu, menu)

        override fun onMenuItemSelected(menuItem: MenuItem) = when (menuItem.itemId) {
            R.id.action_logout -> {
                viewModel.logout()
                true
            }
            else -> false
        }

    }
}
```

Slika 5.9. *Prikaz registracije klika na LOGOUT*

Konkretnu implementaciju „logout“ funkcije možemo vidjeti na slici 5.10. Funkcija se poziva u „MainFragmentViewModel“ klasi kojoj je kroz konstruktor ubrizgana ovisnost o „AuthRepository“ klasi koja sadrži poziv na vanjski servis, u ovom slučaju Firebase, gdje se odjava zapravo i izvršava.

```
antonio.milosevic
fun logout() {
    authRepository.logout()
    logout.value = Unit
}
```

Slika 5.10. *Prikaz „logout“ funkcije*

Funkcija za odjavu ne vraća ništa, ali kako bi mogli registrirati da je odjava izvršena, postavljeno je „SingleLiveEvent“ svojstvo koje je tipa „Unit“. „SingleLiveEvent“ je obrazac koji

se koristi u Android okruženju uz „LiveData“. Namijenjeno je upravljanju događajima unutar same aplikacije i omogućuje sigurnost izvedbe unutar višenitnog načina rada.

Nakon što postavimo „logout“ vrijednost na „Unit“, unutar „MainFragment“ zaslona promatramo „logout“ svojstvo i kada se ta promjena dogodi, pokreće se navođenje aplikacije na početni zaslon, odnosno na „AuthFragment“. U nastavku možemo vidjeti izgled promatranja „logout“ svojstva i navigaciju na početni zaslon.

```
antonio.milosevic
private fun initObserver() {
    viewModel.logout.observe(viewLifecycleOwner) {
        findNavController().navigate(MainFragmentDirections.toAuthFragment())
    }
}
```

Slika 5.11. *Promatranje „logout“ i navigacija na početni zaslon*

Glavne stavke na zaslonu su smještene na sredini zaslona i raspoređene su uz pomoć „RecyclerView“ elementa. Kako bi se podaci mogli ispravno povezati, potrebno je bilo napraviti „MainAdapter“ koji prima listu podataka koji će se prikazati na zaslonu. Za organiziranje elemenata unutar „RecyclerView“ komponente mogu se koristiti razni „LayoutManager“, u ovom konkretnom slučaju je korišten „GridLayourManager“.

```
antonio.milosevic
private fun setupRecyclerView() {
    with(binding.rvItems) { this: RecyclerView
        adapter = mainAdapter
        layoutManager = GridLayoutManager(requireContext(), spanCount: 2)
    }
}
```

Slika 5.12. *Postavljanje „RecyclerView“ komponente*

Adapter koji prilagođava podatke za „RecyclerView“ može registrirati klik na bilo koju stavku koja se nalazi unutar liste.

```
antonio.milosevic
inner class RecyclerDashboardViewHolder(private val binding: MainDashboardItemBinding) : RecyclerViewHolder(binding.root) {
    antonio.milosevic
    fun bindDashboard(dashboard: Dashboard) {
        with(binding) { this: MainDashboardItemBinding
            tvTitle.text = dashboard.title
            ivIcon.setImageDrawable(dashboardIcon(root.context, dashboard))
            root.setOnClickListener { it: View!
                onDashboardClickListener?.invoke(dashboard)
            }
        }
    }
}
```

Slika 5.13. *Postavljanje podataka u adapteru i registracija klika*

Nakon što se klikne bilo koja od stavki, klik se registrira i anonimna funkcija vraća podatak koji je tipa „Dashboard“ koji se nakon toga u „MainFragment“ zaslonu provjerava i navodi korisnika na zaslon ovisno o tome koja je stavka kliknuta. U nastavku možemo vidjeti registraciju i provjeru kliknute stavke.

```
antonio.milosevic *
private fun initClickListeners() {
    mainAdapter.setOnDashboardClickListener { dashboard ->
        when (dashboard) {
            Dashboard.TODOS -> {
                findNavController().navigate(MainFragmentDirections.toTodosFragment())
            }
            Dashboard.TENANTS -> {
                findNavController().navigate(MainFragmentDirections.toTenantsFragment())
            }
            Dashboard.MEETINGS -> {
                findNavController().navigate(MainFragmentDirections.toMeetingsFragment())
            }
            Dashboard.VOTING -> {
                findNavController().navigate(MainFragmentDirections.toVotingFragment())
            }
            Dashboard.BASIC_INFO -> {
                findNavController().navigate(MainFragmentDirections.toBasicInfoFragment())
            }
        }
    }
}
```

Slika 5.14. *Registracija i provjera klika*

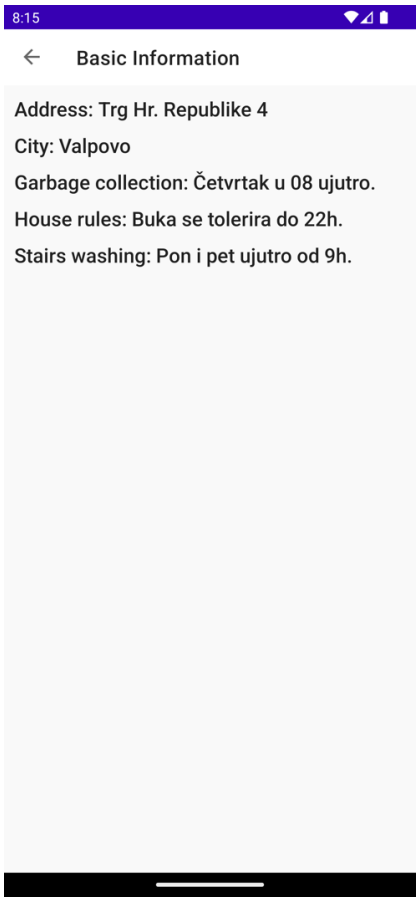
5.3. Zaslon osnovnih informacija

Kako bi korisnik bio naveden na zaslon osnovnih informacija, treba kliknuti na „Basic Info“ stavku na „MainFragment“ zaslonu.

```
Dashboard.BASIC_INFO -> {
    findNavController().navigate(MainFragmentDirections.toBasicInfoFragment())
}
```

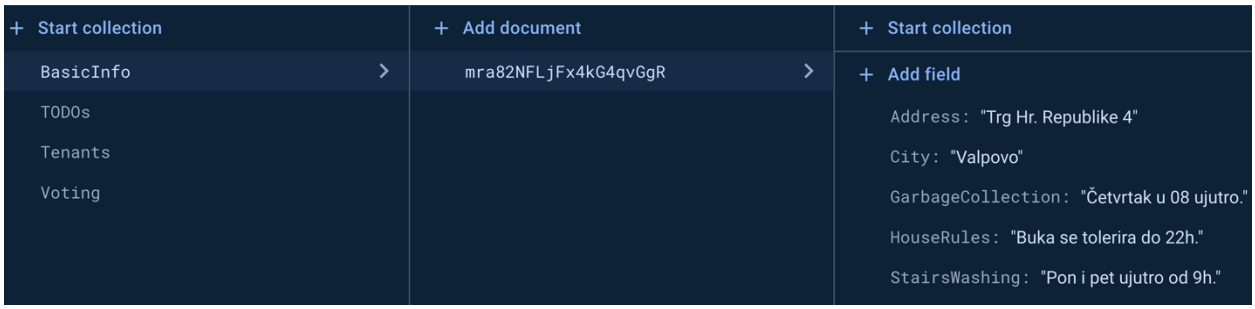
Slika 5.15. *Navođenje korisnika na „BasicInfoFragment“ zaslon*

Nakon što korisnik uđe u zaslon osnovnih informacija, prikazat će mu se zapis o tome na kojoj točno adresi se stambena zgrada nalazi, u kojemu gradu, kojim danima se odvozi smeće i u koje vrijeme, koja su pravila kuće, kao što je tolerancija na buku te ostalo i zadnje će se korisniku prikazati kojim danima se čisti stubište i u koje vrijeme. Na slici 5.16. možemo vidjeti izgled „BasicInfoFragment“ zaslona s gore navedenim zapisima.



Slika 5.16. *„BasicInfoFragment“ zaslon*

Na „BasicInfoFragment“ zaslonu se ne može ništa dodavati niti brisati jer se podaci povlače sa servera, odnosno dolaze s „Firebase“ servisa gdje su podaci spremljeni u kolekciji pod nazivom „BasicInfo“. U nastavku je izgled kolekcije iz koje se povlače podaci za prikaz na „BasicInfoFragment“ zaslonu.



Slika 5.17. *Izgled kolekcije „BasicInfo“*

Kako bi se podaci uspješno dohvatili, potrebno je pri inicijalizaciji zaslona pokrenuti proces koji će gore prikazane podatke i dohvatiti. U nastavku možemo vidjeti funkciju „retrieveBasicInfo“ koja ne prima nikakve parametre i poziva se pri inicijalizaciji unutar „BasicInfoFragment“ zaslona.

```
antonio.milosevic
private fun retrieveBasicInfo() {
    viewModelScope.launch { this: CoroutineScope
        firestoreRepository.getBasicInfo().addOnSuccessListener { it: QuerySnapshot!
            _basicInfo.value = it.documents.first().data.let { data ->
                requireNotNull(data)
                BasicInfo(
                    address = data[ADDRESS] as String,
                    city = data[CITY] as String,
                    garbageCollection = data[GARBAGE_COLLECTION] as String,
                    houseRules = data[HOUSE_RULES] as String,
                    stairsWashing = data[STAIRS_WASHING] as String
                ) ^let
            }
        }.addOnFailureListener { it: Exception
            it.printStackTrace()
        }.await()
    }
}
```

Slika 5.18. *Funkcija „retrieveBasicInfo“*

Funkcija prilikom dohvaćanja podataka unutar „LiveData“ objekta pohranjuje instancu „BasicInfo“ klase unutar koje se nalaze podaci koji su potrebni kako bi se na „BasicInfoFragment“ zaslonu prikazali ispravni podaci. Nakon što su se podaci uspješno pohranili unutar „_basicInfo“, istu varijablu promatramo unutar „BasicInfoFragment“ zaslona, na slici 5.19. možemo vidjeti kod koji prikazuje upisivanje podataka unutar tekstualnih polja.

```
antonio.milosevic
private fun initObserver() {
    viewModel.basicInfo.observe(viewLifecycleOwner) { it: BasicInfo!
        with(binding) { this: FragmentBasicInfoBinding
            tvAddressValue.text = it.address
            tvCityValue.text = it.city
            tvGarbageCollectionValue.text = it.garbageCollection
            tvHouseRulesValue.text = it.houseRules
            tvStairsWashingValue.text = it.stairsWashing
        }
    }
}
```

Slika 5.19. *Promatranje „basicInfo“ i upisivanje podataka unutar tekstualnih polja*

5.4. Zaslon potrebnih popravaka

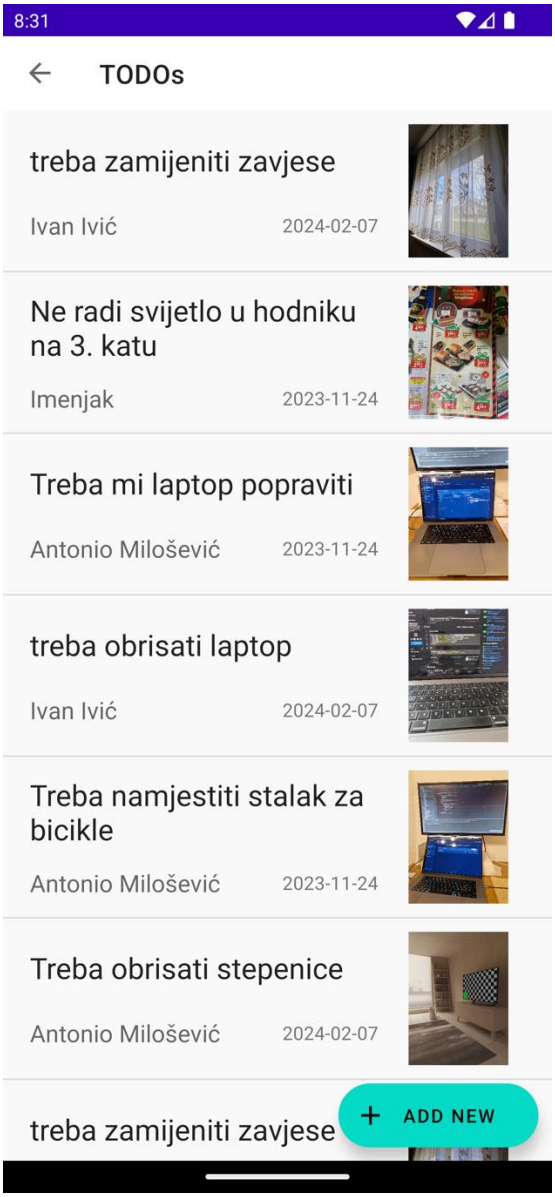
Kako bi mogli uspješno otvoriti zaslon koji nam prikazuje potrebne popravke, trebamo se vratiti na „MainFragment“, odnosno na glavni zaslon na kojemu se nalazi izbornik za odabir željenih zaslona i potrebno je kliknuti na stavku na kojoj piše „TODOs“.

```
Dashboard.TODOS -> {
    findNavController().navigate(MainFragmentDirections.toTodosFragment())
}
```

Slika 5.20. *Navođenje korisnika na „TodosFragment“ zaslon*

Nakon što korisnik otvori „TodosFragment“ zaslon, automatski će se poslati zahtjev na „Firebase“ servis kako bi se dodali svi oni potrebni popravci s pripadajućim fotografijama,

imenom i prezimenom tko je objavio željeni popravak, te kada je objava poslana. Slika 5.21. prikazuje izgled „TodosFragment“ zaslona.

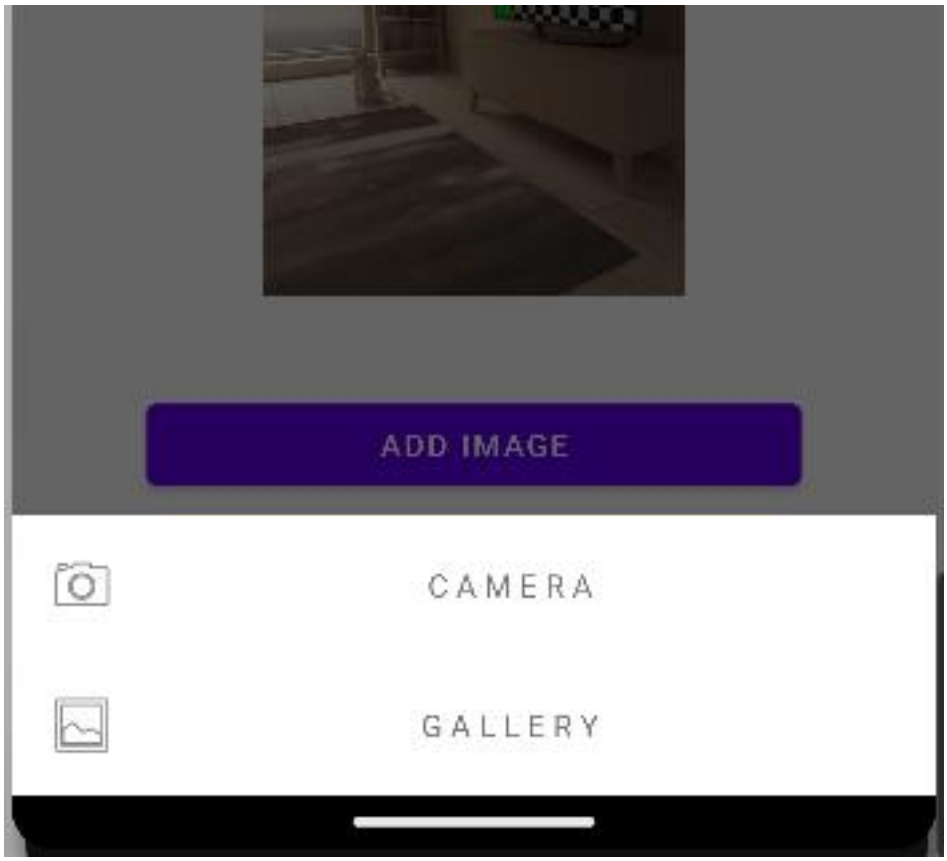


Slika 5.21. *Prikaz „TodosFragment“ početnog zaslona*

U slučaju da korisnik želi dodati novi zahtjev za potrebnim popravkom, potrebno je kliknuti na „Add New“ gumb koji se nalazi na dnu početnog zaslona, može se vidjeti na slici 5.21. Klik na „Add New“ gumb otvara novi zaslon koji se zove „AddNewTodoFragment“.

Slika 5.22. *Prikaz „AddNewTodoFragment“ zaslona*

Kao što se može vidjeti iz slike 5.22. korisnik može, uz opis, postaviti i fotografiju te ju priložiti uz zahtjev za željenim popravkom. Fotografije koje idu uz prilog mogu biti odabrane iz galerije mobilnog uređaja ili fotografirane kamerom samog uređaja.



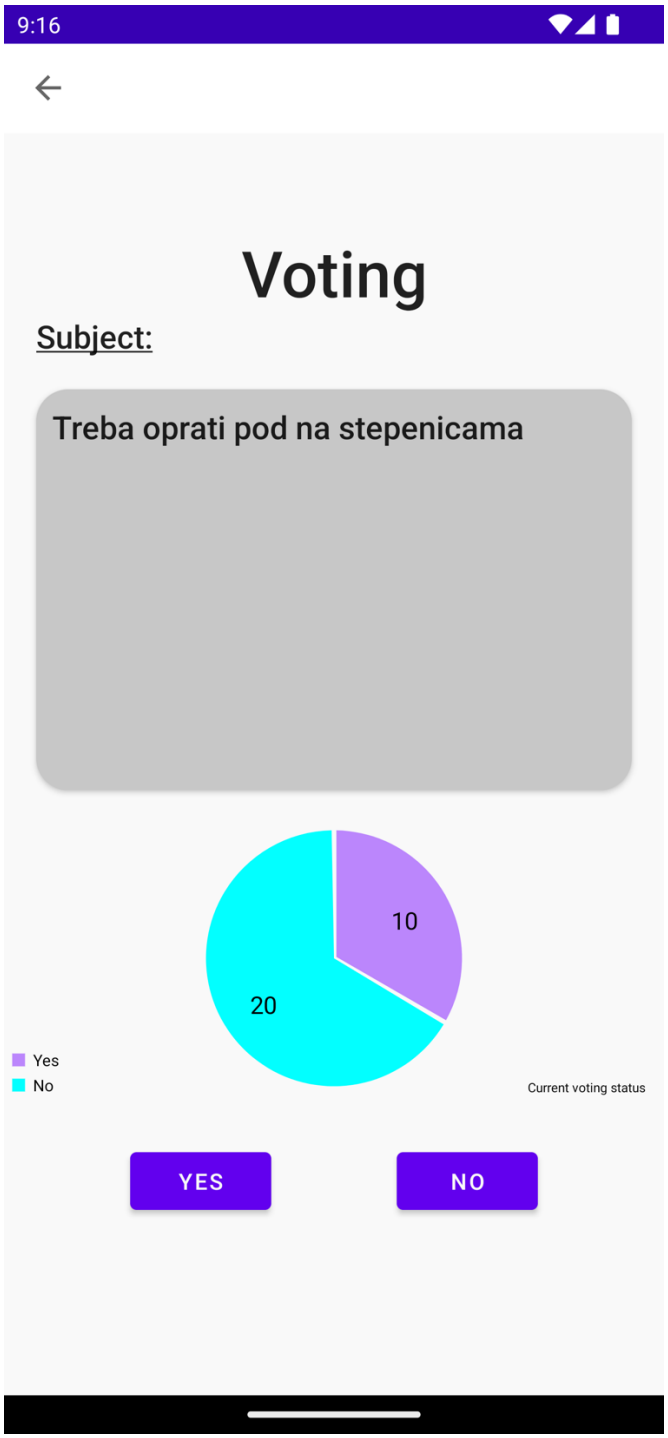
Slika 5.23. *Mogući izbor za prilog fotografije*

Kada je korisnik završio s opisom i prilogom fotografije uz željeni zahtjev za popravak, treba kliknuti na „Upload“ gumb koji pokreće zahtjev za slanjem zahtjeva na „Firebase“ servis, kada se zahtjev pošalje, aplikacija automatski vraća korisnika na prethodni zaslon koji prikazuje listu svih prijašnje dodanih zahtjeva. U nastavku možemo vidjeti kako izgleda funkcija za podnošenje zahtjeva za popravkom.

Slika 5.24. *Funkcija za slanje novog zahtjeva za popravak*

5.5. Zaslon glasanja

Ukoliko korisnik želi dati svoj glas za aktualno glasanje, to može napraviti tako da na „MainFragment“ zaslonu klikne na stavku pod imenom „Voting“ koja će ga nakon toga navesti na „VotingFragment“ zaslon unutar kojega se nalazi subjekt glasanja, odnosno za što to konkretno korisnik daje svoj glas te se prikazuje trenutno stanje glasanja, odnosno koliko je ljudi glasalo za „YES“ a koliko za „NO“.



Slika 5.25. *Izgled „VotingFragment“ zaslona*

Prikaz trenutnog stanja glasanja se izvodi uz pomoć kružnog grafikona, odnosno „MPAndroidChart“ komponente. Kako bi se podaci ispravno prikazali, potrebno je proslijediti trenutne i ažurirane vrijednosti kružnom grafikonu i ponovno inicijalizirati samu komponentu. U nastavku možemo vidjeti funkcije koje omogućuju prikaz kružnog grafikona.

Slika 5.26. *Funkcije za inicijalizaciju kružnog grafikona*

Prilikom klika na „YES“ ili „NO“ gumbe, šalje se zahtjev na „Firebase“ servis kako bi se ažuriralo trenutno stanje glasanja. Uvijek je otvorena samo jedna tema za glasanje i kada se ono izglasa, odnosno kada svi stanari daju svoj glas, ono se zatvara i odluka je donesena.

```
antonio.milosevic
fun updateVoting(didVoteYes: Boolean, context: Context) {
    viewModelScope.launch { this: CoroutineScope
        firestoreRepository.getVoting().addOnSuccessListener { it: QuerySnapshot!
            if (didVoteYes) {
                val yesVotesIncremented = overallVoting.first.inc()
                it.documents.first().data?.set(YES_VOTES, yesVotesIncremented.toString())
                didVote.value = Pair(yesVotesIncremented, overallVoting.second)
                makeToast("Successfully voted YES.", lengthLong = true)
            } else {
                val noVotesIncremented = overallVoting.second.inc()
                it.documents.first().data?.set(NO_VOTES, noVotesIncremented.toString())
                didVote.value = Pair(overallVoting.first, noVotesIncremented)
                makeToast("Successfully voted NO.", lengthLong = true)
            }
        }.addOnFailureListener { it: Exception
            it.printStackTrace()
        }.await()
    }
}
```

Slika 5.27. *Funkcija za ažuriranje stanja glasanja*

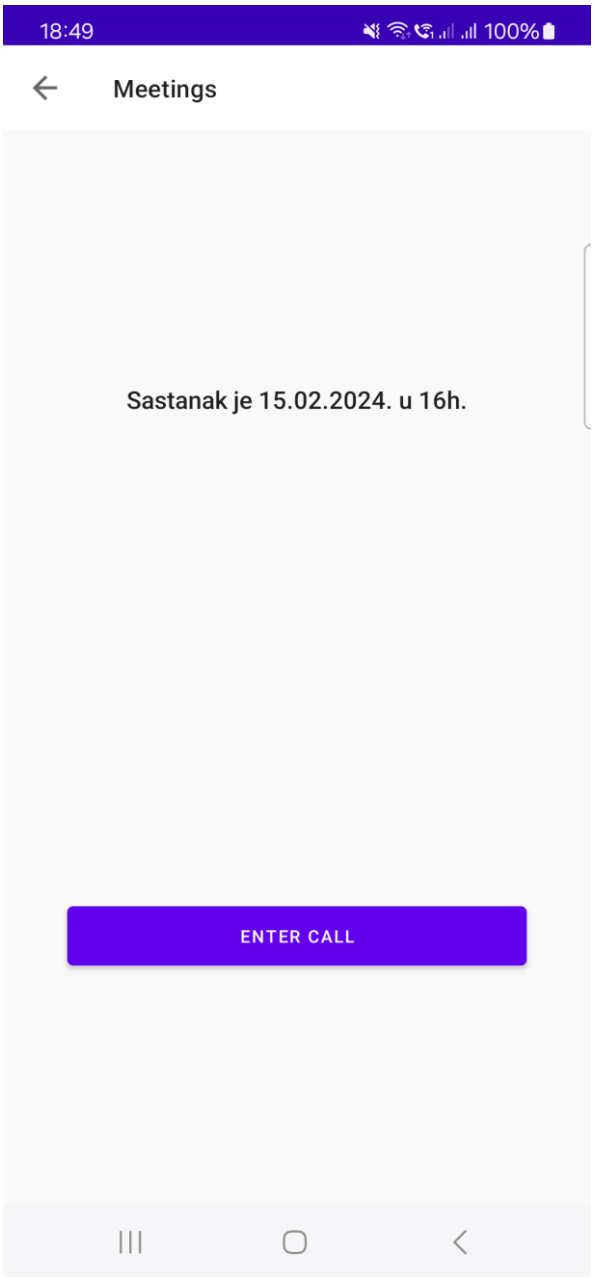
5.6. Zaslon sastanka

Kako bi se smanjila sveopća zbrka koja se može pojaviti unutar same zgrade, predstavnik stanara može objaviti datum i vrijeme kada će se održati sastanak svih stanara. Stanari trebaju otvoriti „MeetingsFragment“ zaslon koji mogu pronaći na glavnom zaslonu „MainFragment“ pod stavkom „Meetings“.

```
Dashboard.MEETINGS -> {
    findNavController().navigate(MainFragmentDirections.toMeetingsFragment())
}
```

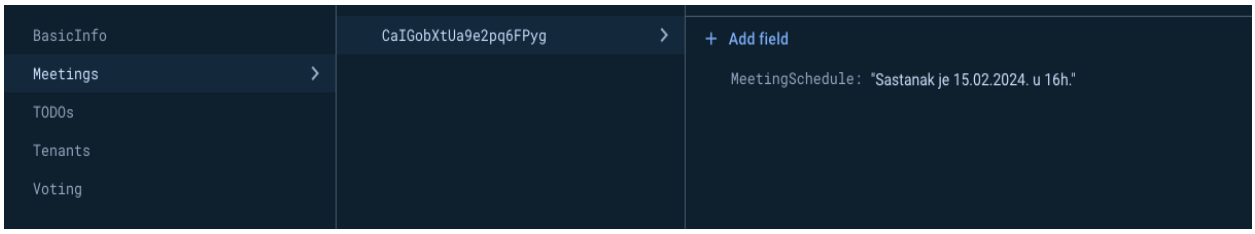
Slika 5.28. *Navođenje korisnika na „MeetingsFragment“ zaslon*

Čim korisnik otvori „MeetingsFragment“ zaslon, pri vrhu samog zaslona će mu se pojaviti trenutna obavijest koju je postavio predstavnik stanara, odnosno kada će biti idući sastanak svih stanara zgrade. Odmah ispod obavijesti je gumb „Enter Call“ koji omogućuje spajanje na virtualni sastanak.



Slika 5.29. *Prikaz „MeetingsFragment“ početnog zaslona*

Način na koji predstavnik stanara može postaviti obavijest kada će se održati sastanak je vrlo jednostavan, treba na „Firebase“ poslužitelju unutar kolekcije „Meetings“ promijeniti vrijednost stavke „MeetingSchedule“ i spremiti promjene.



Slika 5.30. *Prikaz kolekcije „Meetings“ na „Firebase“ servisu*

Nakon što predstavnik stanara promijeni trenutnu obavijest o vremenu i datumu sljedećeg sastanka, sve što se u aplikaciji treba napraviti jest jednostavno ući u „MeetingsFragment“ zaslon kao što je već objašnjeno. Kako bi se podaci uspješno preuzeli s „Firebase“ servisa, potrebno je prilikom inicijalizacije „MeetingsFragment“ zaslona u „init“ funkciju „MeetingsFragmentViewModel“ klase pozvati funkciju „getMeetingSchedule“ koja nam vraća vrijednost koju je predstavnik stanara postavio, odnosno datum i vrijeme kada će se sastanak održati.

```
antonio.milosevic *
class MeetingsFragmentViewModel(
    private val firestoreRepository: FirestoreRepository,
) : ViewModel() {

    val meetingSchedule = SingleLiveEvent<String>()

    antonio.milosevic *
    init {
        getMeetingSchedule()
    }

    antonio.milosevic *
    private fun getMeetingSchedule() {
        viewModelScope.launch { this: CoroutineScope
            firestoreRepository.getMeetingSchedule().addOnSuccessListener { it: QuerySnapshot!
                meetingSchedule.value = it.documents.first().data.let { data ->
                    requireNotNull(data)
                    data[MEETING_SCHEDULE]as String ^let
                }
            }.addOnFailureListener { it: Exception
                it.printStackTrace()
            }.await()
        }
    }
}
```

Slika 5.31. *Prikaz „MeetingsFragmentViewModel“ klase*

```
antonio.milosevic *
private fun initObserver() {
    viewModel.meetingSchedule.observe(viewLifecycleOwner) { meetingSchedule ->
        binding.tvMeetingSchedule.text = meetingSchedule
    }
}
```

Slika 5.32. *Promatranje „meetingSchedule“ unutar „MeetingsFragment“ zaslona*

Kako bi se „meetingSchedule“ vrijednost uspješno pohranila unutar tekstualnog okvira koji se nalazi na „MeetingsFragment“ zaslonu, potrebno je promatrati „meetingSchedule“ svojstvo i kada se promjena dogodi, vrijednost koja je dohvaćena će se upisati unutar „tvMeetingSchedule“ tekstualnog okvira.

Nadalje, u slučaju da se korisnik želi pridružiti virtualnom sastanku, sve što treba napraviti jest kliknuti na „Enter Call“ gumb koji se nalazi na donjoj polovici „MeetingsFragment“ zaslona. Kada korisnik klikne na gore spomenuti gumb, izvršava se funkcija koja je prikazana u nastavku, na slici 5.33.

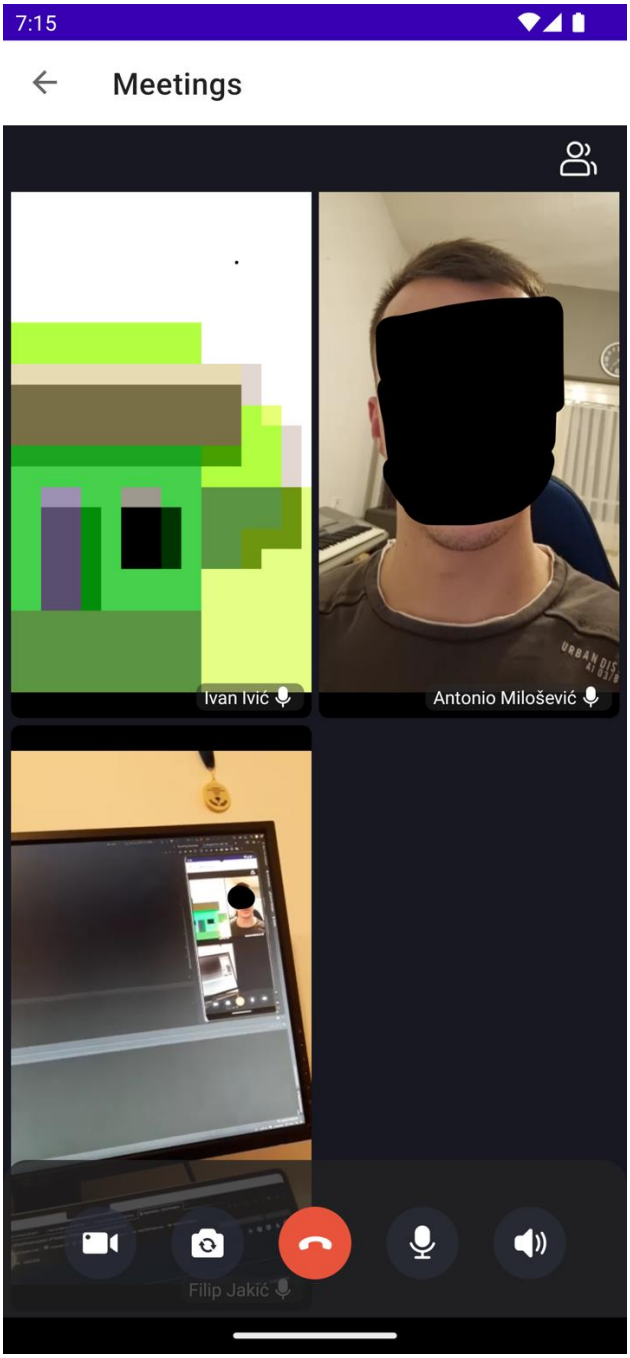
```
antonio.milosevic *
private fun setOnClickListener() {
    binding.btnVideoCall.setOnClickListener { it: View!
        val zegoFragment = ZegoUIKitPrebuiltCallFragment.newInstance(
            APP_ID,
            APP_SIGN,
            SharedPrefsManager().getUserId()!!,
            SharedPrefsManager().getUserFullName()!!,
            TENNANTS_MEETING,
            zegoConfig
        )

        binding.btnVideoCall.visibility = View.GONE

        parentFragmentManager.beginTransaction()
            .replace(
                binding.container.id,
                zegoFragment
            ).commitNow()
    }
}
```

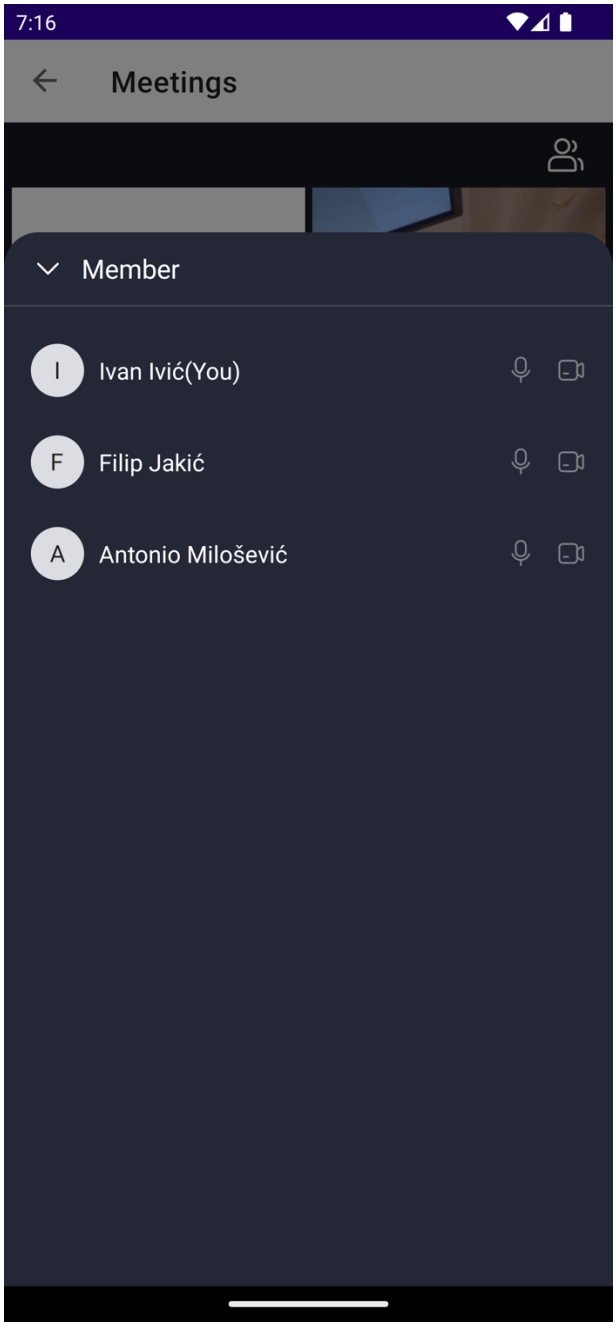
Slika 5.33. *Funkcija za pokretanje virtualnog sastanka*

Kada se korisnici pridruže virtualnom sastanku, mogu upaliti kamere ili ih ostaviti ugašene, mogu biti utišani ili pričati kako bi ih ostali sudionici sastanka čuli. Svaki sudionik se nalazi unutar kvadrata gdje u desnom donjem uglu piše korisnikovo ime.



Slika 5.34. *Prikaz zaslona virtualnog sastanka*

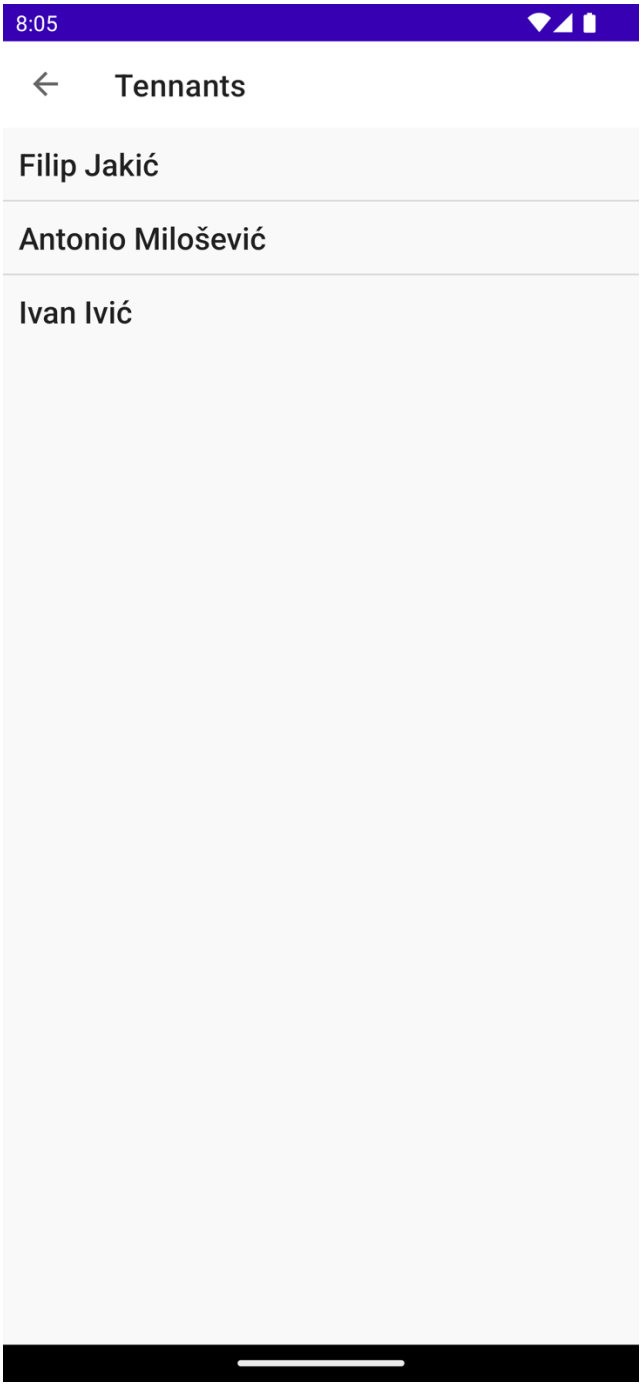
Isto tako, kako bi predstavnik stanara mogao vidjeti tko se sve nalazi unutar samog virtualnog sastanka, pritiskom na gumb u desnom gornjem kutu može vidjeti sve trenutno prisutne korisnike. Na slici 5.35. možemo vidjeti listu prisutnik stanara na virtualnom sastanku.



Slika 5.35. *Prikaz liste prisutnih stanara na sastanku*

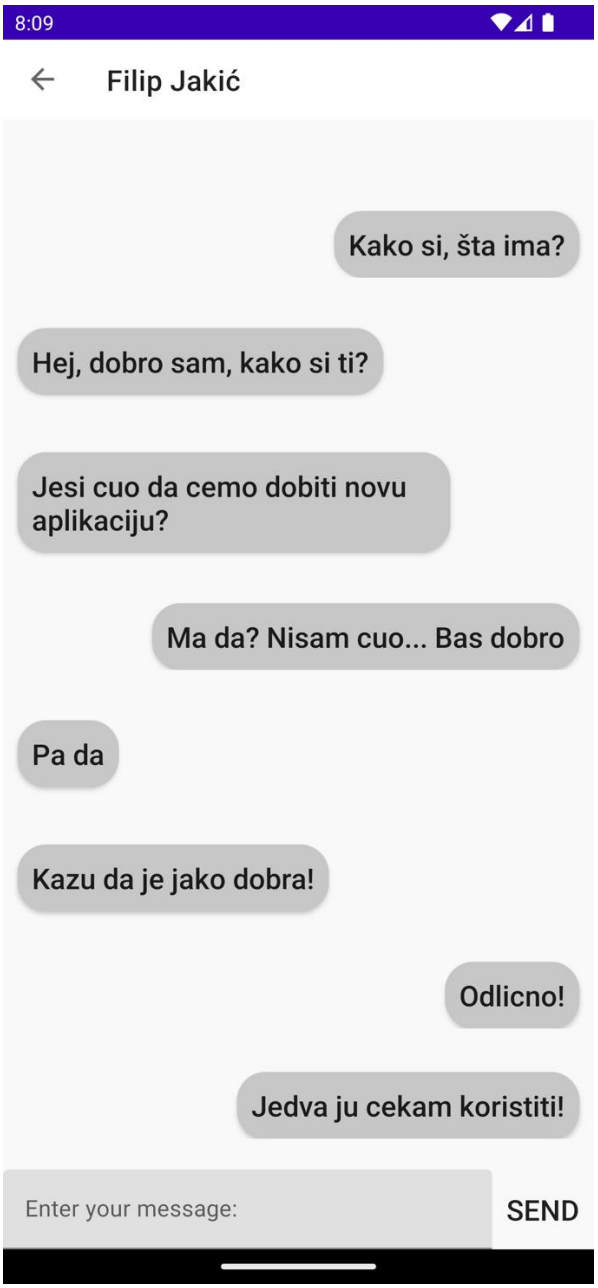
5.7. Zaslon svih stanara i čavrljanje

Želi li korisnik pogledati listu svih stanara koji žive u stambenoj zgradi i poslati poruku jednome od njih kako bi ga obavijestio o bitnim informacijama, može na „MainFragment“ zaslonu kliknuti na „Tennants“ stavku kako bi ga aplikacija navela na „TenantsFragment“ zaslon u kojemu može napraviti sve gore navedeno.



Slika 5.36. *Izgled „TennantsFragment“ zaslona*

Kako bi korisnik otvorio čavrljanje s bilo kojim stanarom iz liste, potrebno je samo da klikne na ime i prezime te će s time pokrenuti funkcije za provjeravanje postoji li već soba s tim korisnikom i nakon toga otvaranje „ChatFragment“ zaslona. U nastavku možemo vidjeti kako izgleda „ChatFragment“ zaslon s porukama.



Slika 5.37. *Izgled „ChatFragment“ zaslona*

Prilikom inicijalizacije „ChatFragment“ zaslona poziva se funkcija „joinRoom“ unutar „ChatFragmentViewModel“ klase. Maloprije spomenuta funkcija služi za ulazak u sobu za

čavrljanje i pretplaćuje korisnika na svaku novu poruku koja bude poslana unutar same sobe. Tako da se lista svih poruka u sobi može automatski ažurirati kako poruke budu stizale.

```
antonio.milosevic *
init {
    joinRoom()
}

antonio.milosevic *
private fun joinRoom() {
    viewModelScope.launch { this: CoroutineScope
        clientImplementation.joinRoomById(RoomId(chatRoomId))
            .onSuccess { roomId ->
                clientImplementation.getLastTimelineEvents(roomId).collectLatest { roomEvents ->
                    _messagesList.value = roomEvents.reversed().map { timelineEvent ->
                        var actualMessage = ""
                        timelineEvent.content?.onSuccess { it: RoomEventContent
                            when (it) {
                                is RoomMessageEventContent.TextBased.Text -> {
                                    actualMessage = it.body
                                }

                                else -> {}
                            }
                        }
                        val senderID = timelineEvent.event.sender.localpart
                        Message(
                            userID = senderID,
                            message = actualMessage
                        ) ^map
                    }
                }
            }.onFailure { it: Throwable
            it.printStackTrace()
        }
    }
}
```

Slika 5.38. *Prikaz „joinRoom“ funkcije*

Funkcija „joinRoom“ kao povratni tip ima „RoomId“ koji se nakon ulaska u sobu može iskoristiti za pretplaćivanje na „getLastTimelineEvents“ funkciju koja za povratni tip ima „Flow“ liste od „TimelineEvent“ koji se može strukturirati u poruku, odnosno poruke koje će kasnije biti prikazane na „ChatFragment“ zaslonu. Sve funkcije koje su vezane za čavrljanje se pozivaju preko „ClientImplementation“ klase koja sadrži konkretnu implementaciju i spajanje na „Matrix“ server.

Slika 5.39. *Prikaz „ClientImplementation“ klase*

Prikaz svih poruka unutar „ChatFragment“ zaslona se izvršava uz pomoć „RecyclerView“ komponente i „ChatAdapter“ klase. Svaki „RecyclerView“ ima svoj adapter koji prilagođava podatke kako bi se oni mogli ispravno prikazati, odnosno upravo onako kako je to korisnik i zamislio. Na slici 5.40. možemo vidjeti kako se postavlja adapter i „LayoutManager“ za „RecyclerView“ komponentu.

```
with(binding) { this: FragmentChatBinding
    rvChat.adapter = chatAdapter
    rvChat.layoutManager = LinearLayoutManager(requireContext())
}
```

Slika 5.40. *Postavljanje „RecyclerView“ komponente*

Poruke se prikazuju na način da se „Message“ klasa, koja ima „userID“ i „message“ svojstva, predaje „ChatAdapter“ klasi koja nakon predavanja napravi ostatak posla, odnosno postavlja poruke na pripadajuće strane zaslona, ovisno o tome tko je poslao poruku.

```
antonio.milosevic
override fun getItemViewType(position: Int) =
    if (differ.currentList[position].userID == myUserID) {
        MY_CHAT_ITEM_VIEW_TYPE
    } else {
        OTHER_CHAT_ITEM_VIEW_TYPE
    }

antonio.milosevic
override fun onBindViewHolder(holder: ChatViewHolder, position: Int) = when(holder) {
    is MyViewHolder -> holder.bindSth(differ.currentList[position])
    is OtherViewHolder -> holder.bindOtherMessages(differ.currentList[position])
}

new *
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int) = when(viewType) {
    MY_CHAT_ITEM_VIEW_TYPE -> MyViewHolder(
        MyChatItemBinding.inflate(
            LayoutInflater.from(parent.context),
            parent,
            attachToParent: false
        )
    )
    OTHER_CHAT_ITEM_VIEW_TYPE -> OtherViewHolder(
        OtherChatItemBinding.inflate(
            LayoutInflater.from(parent.context),
            parent,
            attachToParent: false
        )
    )
    else -> throw Exception("Unhandled view type: $viewType")
}

antonio.milosevic
sealed class ChatViewHolder(root: View) : RecyclerView.ViewHolder(root)

antonio.milosevic
inner class MyViewHolder(val binding: MyChatItemBinding) : ChatViewHolder(binding.root) {
    antonio.milosevic
    fun bindSth(message: Message) {
        binding.tvMessageValue.text = message.message
    }
}

antonio.milosevic
inner class OtherViewHolder(val binding: OtherChatItemBinding) : ChatViewHolder(binding.root) {
    antonio.milosevic
    fun bindOtherMessages(message: Message) {
        binding.tvMessageValue.text = message.message
    }
}
```

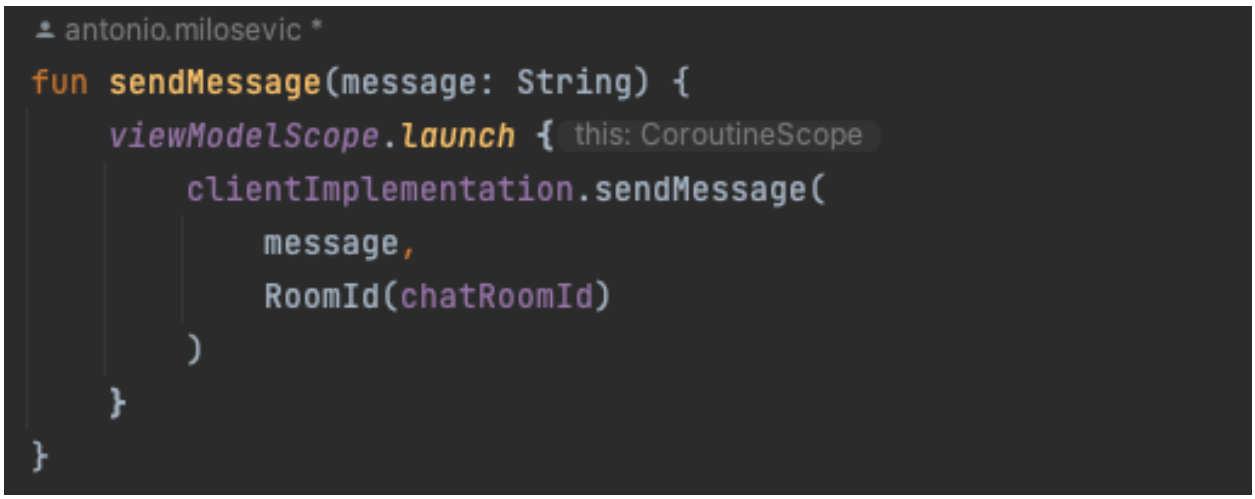
Slika 5.41. *Prikaz „ChatAdapter“ klase*

Nakon što se poruke pošalju „ChatAdapter“ klasi, potrebno je postaviti zaslon na zadnje dodanu poruku kako bi se ona mogla cijela vidjeti, to se mora napraviti ručno zato što „RecyclerView“ nema ugrađeno automatsko povlačenje zaslona na zadnje vidljivu stavku. Funkcija u nastavku, na slici 5.42. prikazuje algoritam za automatsko povlačenje zaslona na zadnje vidljivu stavku.

Slika 5.42. *Funkcija za povlačenje „RecyclerView“ komponente na zadnju vidljivu stavku*

Korisnik koji želi poslati poruku drugom korisniku s kojim je u sobi za čavrljanje treba kliknuti na tekstualni okvir na kojemu piše „Enter your message here:“. Nakon što je kliknuo tekstualni okvir, prikazat će mu se tipkovnica mobilnog uređaja, te nakon što upiše željenu poruku treba kliknuti na „SEND“ gumb koji se nalazi u donjem desnom kutu. Klikom na „SEND“ gumb, pokreće se funkcija za slanje poruke u sobu za čavrljanje preko servera. Isto tako na klik gumba se briše poruka koja se nalazila unutar tekstualnog okvira za unos poruke. U nastavku se nalazi funkcija koja šalje poruku u sobu za čavrljanje i logika za brisanje poslane poruke iz tekstualnog okvira.

Slika 5.43. *Prikaz dijela koda za pokretanje funkcije za slanje poruke, logiku za brisanje poslane poruke i povlačenje zaslona na zadnju poruku*



Slika 5.44. *Prikaz „sendMessage“ funkcije*

6. ZAKLJUČAK

Ovaj diplomski rad prikazuje implementaciju Android aplikacije koja omogućuje međusobnu komunikaciju stanara stambene zgrade, brzu i laku razmjenu korisnih informacija od predstavnika stanara prema podstanarima i obrnuto, također olakšava organizaciju vlastitog vremena jer obavještava stanare o datumima i vremenima kada se odvozi smeće i kada se čisti stubište. Isto tako omogućuje ubrzan proces popravka željenih stvari, bilo da je nešto malo do većih oštećenja, korisnici mogu opisati što treba popraviti i uz to priložiti sliku.

Kao i dobar dio javnih aplikacija koje su dostupne na „Google Play Trgovini“, aplikacija za vođenje stambene zgrade, koju ovaj diplomski rad opisuje, također ima prostora za proširenja. Neka od mogućih proširenja su da se napravi poseban modul aplikacije koji će služiti majstorima da imaju uvid u zahtjeve koji su bili poslani kroz ovu aplikaciju, što treba popraviti i koji su prioriteti, je li nešto hitno ili ne. Isto tako, ako bi broj korisnika rastao, za ovu aplikaciju bi bilo poželjno imati vlastiti server s proizvoljnim rješenjima vezano za razmjenu informacija, obradu zahtjeva za popravak i ostalo.

Gore navedena unaprjeđenja bi pridonijela učinkovitosti i praktičnosti kako bi se korisnici mogli služiti aplikacijom još jednostavnije. Također, daljnje nadograđivanje bi omogućilo kontinuirani razvoj aplikacije i daljnje istraživanje na spomenutom području gdje bi se vrlo vjerojatno pronalazila rješenja i za neke druge probleme koji su danas aktualni.

LITERATURA

[1] Bidrento, Tenant App for tenant engagement and communication, dostupno na: https://bidrento.com/tenant-app/ (pristupljeno siječanj 2024.)

[2] Android Studio, Wikipedia, dostupno na: https://en.wikipedia.org/wiki/Android_Studio (pristupljeno siječanj 2024.)

[3] Kotlin (programming language), Wikipedia, dostupno na: https://en.wikipedia.org/wiki/Kotlin_(programming_language) (pristupljeno siječanj 2024.)

[4] Model-view-viewmodel, Wikipedia, dostupno na: https://en.wikipedia.org/wiki/Model–view–viewmodel (pristupljeno siječanj 2024.)

[5] Save dana in local database using Room, Android Developers, dostupno na: https://developer.android.com/training/data-storage/room (pristupljeno siječanj 2024.)

[6] Firebase, Wikipedia, dostupno na: https://en.wikipedia.org/wiki/Firebase (pristupljeno siječanj 2024.)

[7] Matrix SDK, dostupno na: https://matrix.org (pristupljeno siječanj 2024.)

[8] ZEGO Cloud, dostupno na: https://www.zegocloud.com (pristupljeno siječanj 2024.)

[9] Android (operating system), Wikipedia, dostupno na: https://en.wikipedia.org/wiki/Android_(operating_system) (pristupljeno siječanj 2024.)

[10] Room Database with Kotlin, dostupno na: https://medium.com/huawei-developers/room-database-with-kotlin-mvvm-architecture-477c3ad3c264 (pristupljeno siječanj 2024.)

[11] Coroutines | Kotlin Documentation, dostupno na: https://kotlinlang.org/docs/coroutines-overview.html (pristupljeno siječanj 2024.)

[12] Kotlin coroutines on Android, dostupno na: https://developer.android.com/kotlin/coroutines (pristupljeno siječanj 2024.)

[13] Koin – The pragmatic Kotlin & Kotlin Multiplatform Dependency Injection framework, dostupno na: https://insert-koin.io (pristupljeno siječanj 2024.)

[14] Navigation | Android Developers, dostupno na: https://developer.android.com/guide/navigation (pristupljeno siječanj 2024.)

[15] Migrate to the Navigation component, dostupno na: https://developer.android.com/guide/navigation/migrate (pristupljeno siječanj 2024.)

SAŽETAK

U ovom diplomskom radu je razrađena, osmišljena te na kraju i programski napravljena Android aplikacija za vođenje stambene zgrade. Aplikacija je napravljena unutar Android Studio razvojnog okruženja uz pomoć programskog jezika Kotlin. XML opisni jezik je korišten za dizajniranje izgleda pojedinog zaslona. Ova aplikacija omogućuje korisniku prijavu u aplikaciju, pregled osnovnih informacija o stambenoj zgradi, pregled i postavljanje novih zahtjeva za popravak, moguće je uz opis zahtjeva priložiti i fotografiju koja se može odabrati iz galerije mobilnog uređaja ili fotografirati s fotoaparatom mobilnog uređaja. Osim toga, stanari mogu glasati za određene promjene unutar same zgrade, svaki stanar ima pravo glasa. Također, omogućena je i video komunikacija putem virtualnih sastanaka, te i pisana komunikacija u obliku čavrljanja. Svrha ove aplikacije je omogućiti korisnicima lakšu i bržu razmjenu korisnih informacija te bolje planiranje dana i općenito vremena jer korisnici imaju sve bitne informacije pri ruci uz aplikaciju.

Ključne riječi: Android, Android Studio, Firebase, Kotlin, MVVM, XML

ABSTRACT

Title: Application for managing a residental building

In this master thesis, an Android application for managing a residental building was elaborated, designed and finally programmed. The application was created within the Android Studio development environment with the help of the Kotlin programming language. The XML description language was used to design the layout of each screen. This application allows user to log into the application, view basic information about the residental building, view and set new repair requests, it is possible to attach a photo from the gallery of the mobile device or take a photo with the camera of the mobile device. In addition, tenants can vote for certain changes within the building itself, each tenant has the right to vote. Also, video communication through virtual meetings and written communication in the form of chats are enabled as well. The purpose of this application is to enable users to exchange useful information more easily and quickly, and better plan their day and time in general because users have all the essential information at hand with the application.

Keywords: Android, Android Studio, Firebase, Kotlin, MVVM, XML

ŽIVOTOPIS

Antonio Milošević je rođen u Osijeku 30. 3. 1997. godine. Trenutno živi u mjestu Valpovo nedaleko Osijeka. Pohađao je Osnovnu školu Matije Petra Katančića u Valpovu. Nakon osnovne škole upisuje Srednju školu u Valpovu, smjer Elektrotehničar. Nakon završene srednje škole upisuje preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek te nakon završetka preddiplomskog studija upisuje diplomski studij računarstva, smjer programsko inženjerstvo. Uz studij je odradio i stručnu praksu kao Android developer, nakon odrađene prakse se zaposlio i trenutno radi kao Android developer te se nastavlja i dalje razvijati u tom smjeru.