ANDROID APLIKACIJA ZA PRAĆENJE ZDRAVIH NAVIKA

Lamza, Matej

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

Android aplikacija za praćenje zdravih navika

Matej Lamza

Zagreb, studeni 2019.

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada".

U Zagrebu, 7.11.2019.

Matej Lamza

Predgovor

Ovim putem bih se želio zahvaliti svojoj obitelji koja me podržala tijekom pisanja ovog rada, te svojem mentoru Aleksandera Radovanu na njegovom izdvojenom vremenu i savjetima koji su uvelike olakšali proces izrade ovog rada.

Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi

Sažetak

Ideja ovog rada je izraditi Android aplikaciju koja bi pomogla rekreativcima, ali i fitness trenerima da lakše i učinkovitije prate svoj ili napredak i zdrave navike svojih klijenata na jednome mjestu.

U razvojnom svijetu Android aplikacija nove tehnologije stižu na tržište gotovo svakih nekoliko mjeseci, stoga je potrebno da programer konstantno uči i razvija svoje znanje. Uz samu izradu rada jedan od ciljeva je upoznavanje s trenutno najnovijim tehnologijama na tržištu. Svaka dobro razvijena aplikacija bazirana je na nekoj od dostupnih arhitektura. Ova aplikacija razvijena je na MVVM arhitekturi koja će se objasniti dalje u radu. Bez obzira na kojoj su tehnologiji razvijene, sve aplikacije rade s nekom vrstom podataka. Za takve stvari u ovom radu predstaviti će se JavaRx biblioteka koja dohvaća i obrađuje podatke na pozadinskoj niti aplikacije. Kako aplikacija nudi mogućnost pregleda svih korisničkih podataka, ti podatci se moraju spremiti u nekom obliku baze podataka. Pohrana podataka unutar aplikacije je izvedena pomoću vanjskih biblioteka, koje se povezuju s lokalnom bazom podataka na uređaju te s Google Firebase oblakom. Naposljetku će se pogledati gotovo rješenje te će biti uspoređeno s već postojećim rješenjima na tržištu.

Ključne riječi: Android okruženje, MVVM arhitektura, JavaRx, oblak, baza podataka.

Summary

The idea of this work is to create mobile application that would help people in fitness world regardless of their level of experience to keep track of their healthy habbits or habbits of their clients in case of fitness coaches.

In the world of Android development new technologies are rapidly hitting the market, so in order to be a good Android developer, a programmer must constantly learn new things and expand their knowledge. One of the goals of making this application is introduction to new technologies in Android development. Every good application is built using some kind of architectural pattern. This application will use MVVM pattern as a architecture of choice, wich will be explained more detailed furuther into this paper. No matter what thenology we use to build our program on, the one thing that all programs have in common is that they use some kind of data. Here we will be introduced to JavaRx library wich retrives and process data on background thread of the application. As this application allows user to see information tha the previously stored, that information needs to be stored in some kind of database. Data storage in this application is achived by using thrid party libraries. At the end of this paper we will look at fully built solution and compare it to some of the existing ones on the market.

Sadržaj

1. Uvod

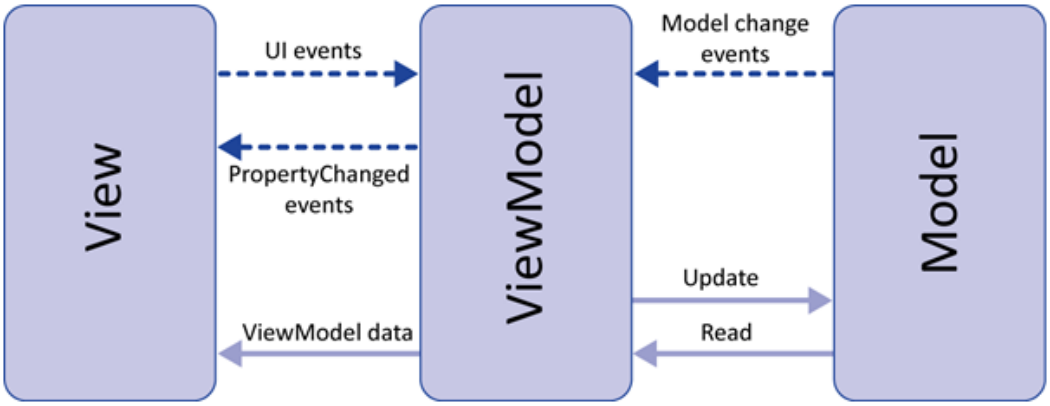
Glavna ideja ovog rada je upoznavanje s najnovijim tehnologijama koje se koriste u svijetu razvoja Android aplikacija, sva dobivena znanja biti će prikazana u obliku fitness aplikacije za praćenje zdravih navika korisnika.

Rad je napisan u Kotlin programskom jeziku te je temeljen na MVVM arhitekturi. U ovom radu predstaviti će se MVVM arhitektura te će se pobliže opisati ostale biblioteke koje su korištene u aplikaciji. Rad će započeti s predstavljanjem MVVM arhitekture te modela lokalne baze podataka kao i one na oblaku.

Nakon predstavljanja arhitekture i modela baze podataka, krenuti će se s opisivanjem registracije i prijave korisnika u aplikaciju, što ujedno označava i prvu funkcionalnost aplikacije. Iza toga slijedi spremanje korisničkih podataka u bazu podataka i oblak te njihovo dohvaćanje i obrađivanje na pozadinskoj niti aplikacije, kao i funkcionalnost brojanja koraka svakog korisnika. Naposljetku će se opisati dodatne dvije funkcionalnosti, kojima imaju pristup korisnici koji su odabrali ulogu trenera prilikom registracije, a to su kreiranje lista klijenata te prikazivanje i stvaranje događaja unutar integriranog kalendara. U zadnjem poglavlju rada usporediti će se izrađena aplikacija, s već postojećim programskim rješenjima te će se prikazati njihove prednosti i nedostatci.

2. Uvod u MVVM arhitekturu

[1]Arhitektura je temelj svakog programskog rješenja koja omogućava aplikaciji strukturu, skalabilnost i lakoću dugoročnog održavanja sustava. Svaki uzorak arhitekture (engl. *Architectural pattern*) ima svoje prednosti i nedostatke, stoga prije odabira uzorka arhitekture, potrebno je uzeti u obzir opseg te ciljeve rada. U ovom poglavlju govoriti će se o MVVM (Model View ViewModel) uzorku arhitekture, izdanog od strane Google-a koji se koristi u razvoju Android aplikacija [1]. Iz slike 1. može se vidjeti da je MVVM struktura vrlo slična MVP (Model View Presenter) arhitekturi, no za razliku od MVP-a, MVVM uvodi nove elemente kao što su *ViewModel* i *LiveData*. Glavni cilj MVVM arhitekture je odvajanje poslovne logike od korisničkog sučelja korištenjem „živih" podataka (engl. *LiveData*) i *ViewModel* komponenti. Kao što samo ime arhitekture nalaže, MVVM arhitektura se sastoji od 3 glavna elementa, a to su: *Model*, *ViewModel* i *View*. Model predstavlja podatke i poslovnu logiku aplikacije, koja će se izložiti putem promatrača (engl. *Observables*) ostalim komponentama. Zadatak *ViewModel*-a je da surađuje s modelom i priprema promatrače tako da mogu biti promatrani od strane pogleda (engl. *View*), koji tada prikazuje podatke korisniku. Prilikom pravilne implementacije MVVM arhitekture, *ViewModel* ne bi trebao znati koji *View* se pretplaćuje na njega, što nudi fleksibilnost prilikom ponovnog iskorištavanja *ViewModela* u nekom drugom kontekstu. Zadnja komponenta MVVM arhitekture je *View*, čiji je zadatak da promatra te da se pretplati na *ViewModel* kako bi dobio podatke i mogao ažurirati korisničko sučelje.



Slika 1. Prikaz MVVM arhitekture[1]

[1] https://proandroiddev.com/mvvm-architecture-viewmodel-and-livedata-part-1-604f50cda1

2.1. Primjena MVVM arhitekture u aplikaciji

U ovom dijelu će se prikazati MVVM arhitektura unutar aplikacije za praćenje zdravih navika korisnika te odvajanje sve tri komponente arhitekture (*Model*, *View*, *ViewModel*) na primjeru prijave korisnika u aplikaciju. Unutar aplikacije postoji *Model* korisnika (engl. *User*), koji se sastoji od korisničkih podataka potrebnih za prijavu, a to su e-mail i zaporka. *View* komponenta je zadužena za iscrtavanje korisničkog sučelja te slušanja korisničkih događaja, kao što su unos podataka, pritisak na gumbe itd. Na primjeru prijave korisnika, *View* komponenta prikazuje korisniku sučelje, koje zahtjeva od korisnika unos e-maila i zaporke, kako bi se mogao prijaviti u aplikaciju. *View* nije svjestan modela korisnika unutar aplikacije, već samo sprema podatke koje je korisnik unio putem *TextView widget-a*. *View* ima prisluškivač na gumbu, koji označuje da je korisnik gotov s unosom podataka te da je spreman za prijavu u aplikaciju. Nakon što je korisnik ispunio podatke koji se zahtijevaju od njega te pritisne gumb za prijavu, podatci se tada šalju u *ViewModel* koji provjerava ispravnost formata unesenih podataka. Nakon validacije podataka, *ViewModel* odlazi u repozitorij koji sadrži metodu prijave korisnika. *ViewModel* nakon dobivene povratne informacije od repozitorija, prosljeđuje dalje informacije *View* komponenti. Ukoliko je korisnik uspješno prijavljen u aplikaciju, *View* će ga preusmjeriti na početni zaslon aplikacije. Ako je korisnik krivo unio zaporku ili email, *View* će prikazati korisniku poruku s odgovarajućom pogreškom. Kod 1. prikazuje kako Model *User*-a izgleda. Kod 2. prikazuje inicijalizaciju *ViewModela* i pozivanje metode validacije korisničkih podataka. Kod 3. Prikazuje *ViewModel* klasu koja ima „žive“ podatke na koje se *View* pretplaćuje. Pomoću „živih“ podataka *ViewModel* osvježava podatke u *View* komponenti. *ViewModel* sadrži metodu koja zove repozitorij da obavi korisničku prijavu, čiji se rezultat šalje putem „živih“ podataka u *View*.

```
@Entity(tableName = "users")
open class User: Serializable {
    @PrimaryKey(autoGenerate = true)
    var id          : Int?              = null
    var uid         : String?           = ""
    var loginInfo   : LoginInfo?        = null
    @Embedded
    var basicInfo   : BasicInformation?  = null
}
```

Kod 1. Model korisnika unutar aplikacije

```
loginViewModel =
ViewModelProviders.of(this,loginVMFactory).get(LoginViewModel
::class.java)
if(loginViewModel!!.validateLoginFields(loginInfo)){
   loginViewModel!!.onlineUserLogin(loginInfo,this)}
```

Kod 2. Inicijalizacija *ViewModela* te poziv metoda za prijavu korisnika

```
val liveUser: MutableLiveData<User> = MutableLiveData()
fun onlineUserLogin(loginInfo: LoginInfo, context: Context){
 userRepo.signInUserFromOnlineDatabase(loginInfo, context)}
fun validateLoginFields(loginInfo: LoginInfo):Boolean{
        for(field in LoginInfo::class.memberProperties){
            if (field.get(loginInfo) == ""){
                Log.d("aaa","Fields must not be empty!")
                return false
            }
        }
        return true
    }
```

Kod 3. *ViewModel* klasa s metodama za prijavu i validaciju korisničkih podataka

3. Baza podataka unutar aplikacije

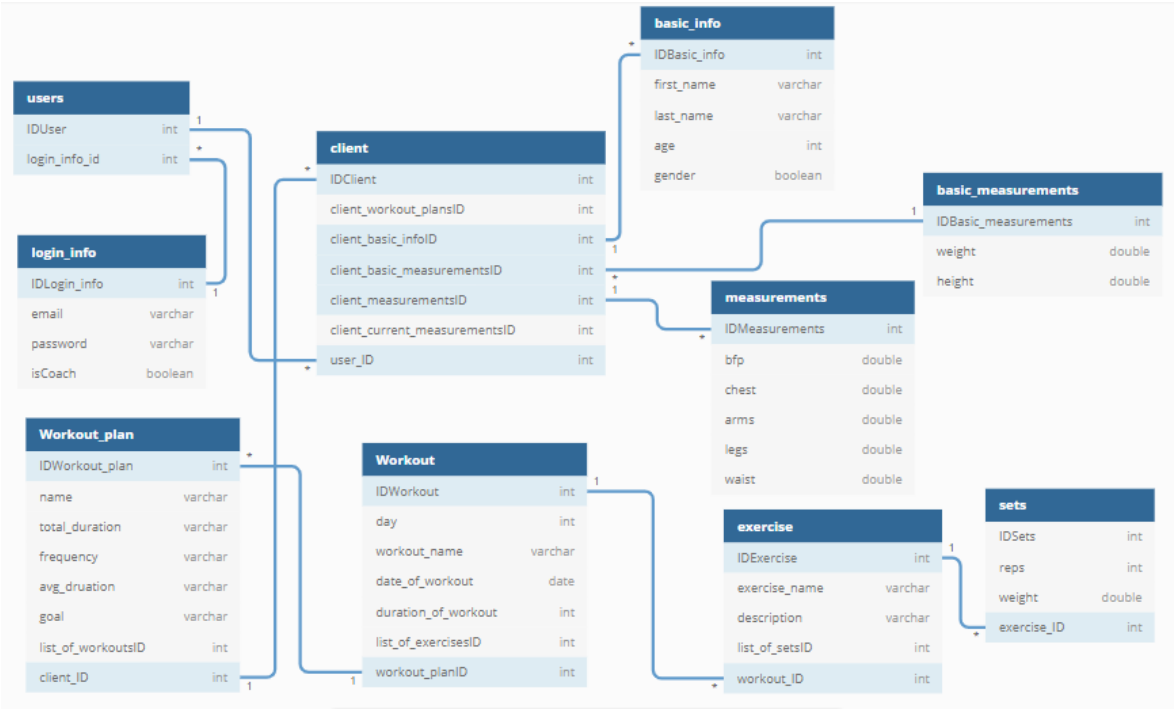
U ovom poglavlju definirati će se što je baza podataka te će se dati primjer korištenja baze podataka i njenih funkcija unutar aplikacije. Baza podataka[2] se može definirati kao kolekcija ili skup podataka koji su organizirani najčešće u tablice, no postoje i drugi oblici baza podataka koje ne koriste tablični sustav. Baze podataka za spremanje ili obavljanje operacija nad podatcima, koriste jezik za upravljanje bazom podataka, koji ovisi o bazi podataka u kojoj se radi. Najčešće je to SQL jezik. Aplikacija za praćenje zdravih navika omogućava korištenje aplikacije kada je korisnik spojen na internet, ali i kada je u izvan mrežnom načinu rada, što znači da se korisnički podatci spremaju se lokalno na sam uređaj korisnika te na oblak (engl. *Cloud*). U Android okruženju za spremanje podataka, koristi se SQLite baza podataka. SQLite je zapravo biblioteka izrađena u C programskom jeziku, koja implementira sve značajke prave SQL baze podataka [2]. Također, ugrađena je na sve pametne telefone i većinu aplikacija koje korisnici koriste na osobnim računalima. SQLite je stabilna, podržana međusobno između platformi te unazad kompatibilna biblioteka. U Android okruženju SQLite sprema podatke u tekstualne datoteke lokalno na uređaju. Nad podatcima unutar baze podataka mogu se izvršavati razni upiti i sortiranja podataka prema potrebama programera. Najčešće operacije koje se izvršavaju nad podatcima su CRUD operacije. CRUD je kratica koja se odnosi na *create* (stvori), *read* (pročitaj), *update* (osvježi) i *delete* (obriši). Aplikacija za praćenje zdravih navika korisnika koristi Room biblioteku za rad sa SQLite bazom podataka, koja će se objasniti dalje u radu.

3.1. Model lokalne baze podataka

Slika 2. predstavlja model lokalne baze podataka na uređaju. Svaki korisnik mora izraditi korisnički račun, kako bi dobio pristup aplikaciji. Korisnički podatci koji se koriste za prijavu, preciznije email i zaporka, spremaju se u tablicu *User*. Zatim tablica *User* povezuje se na tablicu *Coach* (trener) ili *Athlete*, ovisno o ulozi koju korisnik odabere prilikom registracije. Povezivanje tablica korisnika s tablicom trenera ili rekreativca, omogućava povezivanje korisničkih podataka za prijavu s ulogom koju je korisnik izabrao prilikom registracije, jedan zapis u tablici korisnika biti će povezan s točno jednom ulogom. Treneri

[2] https://searchsqlserver.techtarget.com/definition/database

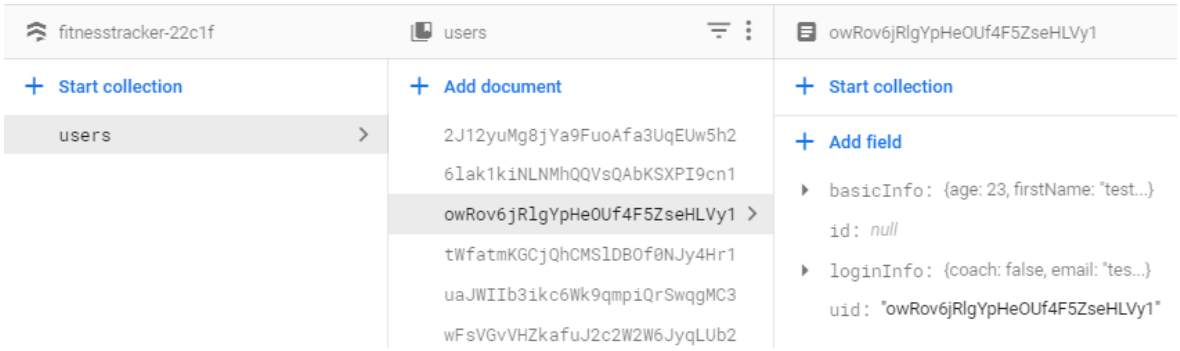
unutar aplikacije imaju mogućnost dodavanja svojih klijenata u listu zbog lakše preglednosti te također kreiranje događaja unutar kalendara, što im omogućava da pratiti svoje obaveze na jednome mjestu. Klijenti koje trener kreira spremaju se u tablicu *Clients*, koja sadrži planove treninga koje pojedini klijent prati, osnovne informacije kao što su ime, prezime, godine, spol te početna mjerenja i buduća mjerenja, kako bi se dobio uvid u napredak pojedinog klijenta. S aspekta treninga postoje tablice *WorkoutPlan*, *Workout*, *Exercise* te *Sets*. U *WorkoutPlan* tablicu sprema se jedan plan treninga, koji se sastoji od proizvoljnog broja treninga, svaki trening se sprema u tablicu *Workout* i sastoji se od više vježbi, koje imaju svoje setove. Set se može definirati kao jedna radna serija određene vježbe koju osoba izvede. U kontekstu aplikacije, jedan set sprema podatke o broju ponavljanja odrađene vježbe te s koliko je kilograma osoba izvela spomenuti broj ponavljanja. Kako Android okruženje nema ugrađenu vizualizaciju baze podataka i relacije među tablicama, ovaj model je izrađen u online alatu te je korišten samo kako bi se prikazao model lokalne baze podataka.



Slika 2. Model lokalne baze podataka

3.2. Model baze podataka u oblaku

Prije nego što se da primjer modela baze podataka u oblaku, potrebno je definirati što je oblak. Oblak (engl. *Cloud*) u smislu pohrane podataka, je usluga u kojoj se podatci pohranjuju na udaljene poslužitelje kojima se pristupa putem interneta. Ova aplikacija za pohranu podataka u oblaku koristi Google-ov Firestore oblak, kojeg će se objasniti u sljedećem poglavlju, za sada će se dati samo primjer modela baze. Firestore oblak radi na principu kolekcija i dokumenata, svaka kolekcija može imati proizvoljan broj dokumenata [3]. Iz slike 3. se može vidjeti kako kolekcija „*users*“ ima više dokumenata. Svaki dokument označava jednog korisnika, svaki korisnik u sebi ima podatke kao što su ime, prezime, spol, godine te informacije koje se koriste za prijavu. Svaki dokument je jedinstveni zapis u kolekciji, stoga Firestore prilikom zapisivanja novog podatka u bazu, kreira jedinstveni ključ pod nazivom *Unique Identifier* (UID), u obliku *stringa*, pomoću kojeg se kasnije dohvaćaju željeni podatci. Slika 3. prikazuje kako se spremaju korisnici i njihove informacije unutar oblaka.



Slika 3. Model baze podataka na oblaku

4. Registracija i rad s podatcima u oblaku

Proces registracije unutar Aplikacije za praćenje zdravih navika korisnika sastoji se od nekoliko koraka. U ovome dijelu će se proći kroz sve korake korisničke registracije te korisničku prijavu u aplikaciju, a koraci registracije biti će popraćeni primjerima iz aplikacije. Također objasniti će se Google Firestore oblak, koji se koristi za potvrdu korisničkih podataka i njihovo spremanje na oblak.

4.1. Google Firestore

Google Firestore je „baza podataka“ koja se nalazi na oblaku, s kojom se iOS, Android i web aplikacije mogu povezati izravno pomoću alata za razvoj *software*-a (engl. *Software Development Kit*) [3]. Klijentske biblioteke podržavaju sinkronizaciju podataka u stvarnom vremenu te izvan mrežnu podršku. Sigurnosne značajke su već integrirane unutar Firestore-a i Google Cloud Platforme (GCP). Google Firestore zahvaljujući svojem dizajnu, posve je skalabilan i može se koristit za aplikacije manjeg obujma, pa sve do većih projekata koji idu u produkciju. Firestore također nudi podršku prilikom obavljanja transakcija, ukoliko neku od operacija unutar transakcije nije moguće izvršiti, cijela transakcija će se poništiti. Podatci u Firestore oblaku se pohranjuju u obliku dokumenata koji sadrže polja za pohranu vrijednosti. Dokumenti se tada spremaju u kolekcije, koje služe kao spremnici u kojima se mogu organizirati dokumenti te izvršavati upiti nad njima. Dokumenti podržavaju razne tipove podataka, od jednostavnih kao što su brojevi i *stringovi*, pa sve do kompliciranih ugniježđenih objekata. Struktura podataka unutar Firestore oblaka slična je datotekama i dokumentima koji se koriste na osobnim računalima. Jedna od prednosti korištenja Firestorea je mogućnost dodavanja slušatelja (engl. *Listeners*) unutar aplikacije, koji mogu u stvarnom vremenu poslati samo nove promijene koje su se dogodile unutar dokumenta.

4.2. Registracija i potvrda korisničkih podataka

Firestore nudi razne načine korisničkog registriranja, kao što su registracija pomoću broja telefona, povezivanje računa Facebook-a, Twittera, Githuba, te putem email-a i zaporke koju ova aplikacija koristi. Prilikom registracije korisnik najprije unosi željeni e-mail i zaporku te odabire ulogu koju želi imati u aplikaciji. Aplikacija ima dvije različite uloge za korisnike,

jedna je uloga trenera koja će ponuditi korisniku da napravi listu osoba koje trenira te da prati njih individualni napredak na jednom mjestu i također kalendar s kojim može pratiti buduće termine treninga. Druga uloga je uloga običnog korisnika, odnosno individualca, koja korisniku pruža mogućnost da prati vlastite navike, kao što su broj koraka, odrađeni trening, trenutna tjelesna kilaža i mnoge druge stvari. Na slici 4. može se vidjeti registracijska forma unutar aplikacije.

Slika 4. Registracija korisnika unutar aplikacije

Aplikacija iako podržava izvan mrežni način rada, prilikom korisničke registracije korisnik mora imati pristup internetu kako bi Firestore biblioteka mogla poslati podatke na oblak te ih usporediti s postojećim korisnicima i vidjeti koristi li već netko tu email adresu. Ukoliko email adresa koju je korisnik unio već postoji, Firestore će pomoću slušatelja vidjeti da registracija nije uspjela, tada će aplikacija obavijestiti korisnika da se pokuša registrirati s drugom email adresom. Registracija korisnika unutar koda je poprilično jednostavna zahvaljujući Firestore biblioteci. Kao što se iz koda 4. može vidjeti, kreira se instanca Firestore *authenticator*, koji ima već ugrađenu metodu za kreiranje korisničkog računa, a koja kao argumente prima korisnički email i zaporku. Zatim se kreiraju slušatelji koji čekaju rezultat korisničke registracije te ukoliko je registracija uspjela, zove se metoda koja sprema korisničke informacije u Firestore bazu podataka.

```
fun registerUser(user: User){
      authInstance.createUserWithEmailAndPassword(
      user.loginInfo!!.email.toString(),
      user.loginInfo!!.password.toString())
      .addOnCompleteListener {
            if (it.isSuccessful)
                  {saveUserToOnlineDatabase(user)}
            }
      .addOnFailureListener
      { Log.d("aaa","Error: ${it.message}") }}
```

Kod 4. Registracija korisnika pomoću Firestore biblioteke

4.3. Prijava i dohvaćanje podataka iz oblaka

Na primjeru prijave korisnika vidjeti će se međusobna komunikacija između triju glavnih komponenti MVVM arhitekture te odvajanje briga svake komponente zasebno. Iz koda 6. može se vidjeti *View* komponenta koja korisniku iscrtava korisničko sučelje te uzima podatke unesene od strane korisnika i sprema ih u varijablu `loginInfo`. Također, može se vidjeti inicijalizacija *ViewModel* komponente, koju *View* zove kako bi izvršio validaciju korisničkih podataka te obavi prijavu korisnika. *View* komponenta također ima prisluškivače za „žive“ podatke koje *ViewModel* komponenta emitira. Prisluškivači su potrebni kako bi uhvatili emitiranje podataka koje *ViewModel* šalje ili osvježili već postojeće podatke. *ViewModel* s druge strane nema nikakve veze s *View* komponentom, već se u njemu drže funkcije za obradu podataka. U ovom slučaju to su funkcije za validaciju podataka i prijavu korisnika. *ViewModel* za prijavu korisnika zove drugu klasu `UserRepo,` koja u sebi ima instancu Firestore biblioteke i koja je zadužena za obradu korisničkih podataka. Da bi *ViewModel* mogao obavljati pozive prema repozitoriju, unutar konstruktora kroz injektiranje ovisnosti poslana mu je instanca repozitorija, pomoću koje onda može zvati metode kao što su prijava, registracija korisnika i slično. Firestore biblioteka u sebi ima već ugrađenu metodu prijave korisnika koja prima argumente ovisno o načinu prijave. Kao što se već spomenulo postoje razni načini prijave na Firestore, ali u ovoj aplikaciji koristi se email i zaporka. Iz koda 5. može se primijetiti sličnosti kao prilikom registracije korisnika, instanca klase `authInstance` ima metodu koja prima e-mail i zaporku korisnika, koji su ovdje u kodu umotani (engl. *Wrapped*) u model `LoginInfo`. Biblioteka tada šalje podatke na oblak i uspoređuje ih s postojećim korisnicima.

```
fun signInUserFromOnlineDatabase(loginInfo: LoginInfo, context: Context){
      this.userFoundListener = context as UserFoundListener
      authInstance.signInWithEmailAndPassword(loginInfo.email as
      String,loginInfo.password as String)
            .addOnSuccessListener
            {this.userFoundListener!!.isUserFound(true)}
            .addOnFailureListener
            {this.userFoundListener!!.isUserFound(false)}
            }
```

Kod 5. Firestore metoda za prijavu korisnika

```
loginViewModel =
ViewModelProviders.of(this,loginVMFactory).get(LoginViewModel::class.java
)
btn_SignUp.setOnClickListener {
            val intent = Intent(this, RegisterActivity::class.java)
            startActivity(intent)
 }
 btn_Login.setOnClickListener {
            val loginInfo = getLoginInfo()
            if (NetworkHelper.checkForInternetConnectivity(this)){
            if(loginViewModel!!.validateLoginFields(loginInfo)){
                  loginViewModel!!.onlineUserLogin(loginInfo,this)}}
            else{
            loginViewModel!!.getUserFromOfflineDatabase.
            (loginInfo.email!!,loginInfo.password!!)
            }
 }

loginViewModel!!.isUserFound.observe(this, Observer {
      it?.let {
               isSuccess ->
      if (isSuccess){
            Toast.makeText(this,"Login
      successfull",Toast.LENGTH_SHORT).show()}
      else{
            Toast.makeText(this,"Login Failed",
      Toast.LENGTH_SHORT).show()}
                        }
                  })
```

Kod 6. *View* komponenta za prijavu korisnika

5. Rad s lokalnim podatcima

U ovom dijelu će se objasniti Room biblioteka koja predstavlja sloj iznad SQLite lokalne baze podataka te će se dati praktični primjer korištenja Room biblioteke unutar aplikacije.

5.1. Room biblioteka

Aplikacija za rad sa SQLite bazom podataka koristi Room biblioteku, koja služi kao apstrakcijski sloj iznad SQLite baze za lakše korištenje objekata unutar aplikacije. Room biblioteka sastoji se od tri glavne komponente; Baza podataka (engl. *Database*), Entitet (engl. *Entity*) koji predstavlja individualnu tablicu unutar baze podataka te sloja pristupa podatcima (engl. *Data Access Layer*) koji sadrži metode za obradu i pristup podatcima [4]. Da bi Room biblioteka mogla prepoznati različite komponente u kodu, iznad svake klase potrebno je staviti anotaciju u obliku znaka „@“ i komponente koju ta klasa predstavlja. Za primjer iz koda može se vidjeti klasa *User*, koja iznad naziva klase ima anotaciju *@Entity*, što označava biblioteci da tretira tu klasu kao entitet, odnosno tablicu. Entitet također ima jedan argument, a to je naziv tablice (engl. *Table Name*). Naziv tablice će se koristiti prilikom kreiranja upita ili metoda nad tom tablicom. Kod 7. prikazuje klasu `UserDatabase` koja nasljeđuje `RoomDatabase` klasu i sadrži anotaciju *@Database*, što označava tu klasu kao bazu podataka. *Database* komponenta također ima argumente, a to su entiteti, odnosno tablice, koje se nalaze u bazi podataka te verziju baze. Također, u klasi postoji metoda koja dohvaća model pristupa podataka u kojem se nalaze metode za rad nad podatcima unutar baze podataka. Room biblioteka ne može spremati objekte u bazu, već joj je za to potreban pretvarač. Pretvarač ili *TypeConverter* omogućava Room biblioteci da pretvori objekt u tip podataka koje Room može spremiti u bazu podataka.

```
@Database(entities = arrayOf(User::class,CachedUser::class,
Client::class,WorkoutPlan::class),version = 1)
@TypeConverters(DataConverter::class)
abstract class UserDatabase: RoomDatabase() {
    abstract fun getUserDAO():UserDAO
}
```

Kod 7. *Database* komponenta u kodu

5.2. Primjena Room biblioteke unutar aplikacije

Kako bi Room biblioteka mogla obavljati svoj posao, potrebne su joj sve tri komponente unutar koda. Ukoliko programer izostavi jednu od komponenti, kod se neće moći izvršiti. Unutar aplikacije postoje nekoliko klasa koje su označeni kao entiteti, odnosno tablice, u koje se spremaju korisnički podatci. Kod 9. prikazuje sučelje (engl. *Interface*) sloja pristupa podatcima (DAO) koje sadrži sve metode koje se izvršavaju nad bazom podataka. DAO ima vlastite anotacije unutar svog sučelja, kao što su *Update*, *Insert*, *Delete* i *Query* koje omogućavaju obradu podataka u bazi. Anotacija *Query* predstavlja izvršavanje naredbe za dohvaćanje podataka, koja je sintaksom vrlo slična SQL-ovoj verziji *select* naredbe. U kodu 8. može se vidjeti jedan entitet u bazi, klasa `User` koja ima iznad anotaciju *Entity* kako bi ju Room biblioteka prepoznala. Unutar svake tablice u bazi podataka, postoji kolona koja je označena kao primarni ključ. Primarni ključ omogućuje da svaki zapis u tablici bude jedinstven. Room biblioteka ima mogućnost automatskog inkrementa primarnog ključa, što znači svaki puta kada dođe novi zapis unutar tablice, primarni ključ će automatski biti povećan.

```
@Entity(tableName = "users")
open class User: Serializable {
    @PrimaryKey(autoGenerate = true)
    var id          : Int?              = null
    var uid         : String?           = ""
    @Embedded
    var loginInfo   : LoginInfo?        = null
    @Embedded
    var basicInfo   : BasicInformation? = null
}
```

Kod 8. *Entity* komponenta Room biblioteke

```
@Dao
interface UserDAO {
      @Insert
      fun saveWorkoutPlanToOfflineDB(workoutPlan: WorkoutPlan)

      @Insert
      fun saveClientToOfflineDb(client: Client)

      @Insert
      fun saveUserToOfflineDatabase(user:User)

      @Update
      fun updateWorkoutPlan(workoutPlan: WorkoutPlan)

      @Query("UPDATE clients SET clientWorkoutPlans  = :mWorkoutPlans
      WHERE id = :mClientId")

      @Update
      fun updateClientsWorkoutPlans
      (mClientId:Int,mWorkoutPlans:ArrayList<WorkoutPlan>)

      @Update
      fun updateClient(client: Client)

      @Query("SELECT * FROM workoutPlan WHERE id = :idWorkoutPlan")
      fun getWorkoutPlanByID(idWorkoutPlan:Int):Single<WorkoutPlan>

      @Query("SELECT * FROM workoutPlan")
      fun getAllWorkoutPlansFromOfflineDB():Flowable<List<WorkoutPlan>>

      @Query("SELECT * FROM clients")
      fun getAllClientsFromOfflineDb(): Flowable<List<Client>>

      @Query("SELECT * FROM USERS WHERE email = :mEmail AND password =
      :mPassword")
      fun getUserFromOfflineDatabase(mEmail:String, mPassword:String):
      Single<User>}
```

Kod 9. DAO komponenta Room biblioteke

6. Rad s podatcima na pozadinskoj niti

U ovom dijelu će se objasniti što je to više nitni razvoj, dati će se primjer iz aplikacije te će se objasniti JavaRx biblioteka, pomoću koje je izveden rad na pozadinskoj niti unutar aplikacije. Više nitni razvoj omogućava da se zadatci unutar aplikacije podijele na manje dijelove te se izvrše paralelno umjesto sekvencijalno, gdje bi program čekao da se jedan zadatak izvrši da bi mogao započeti s drugim. Kako se zadatci brže obavljaju kada se izvršavaju na više niti, samim time i korisnik ima bolje iskustvo prilikom korištenja aplikacije. U Android okruženju aplikacije se zadano izvršavaju na jednoj niti koja se naziva UI (*User Interface*) nit, odnosno nit korisničkog sučelja. Nit korisničkog sučelja, zadužena je za prikazivanje korisničkog sučelja aplikacije te prisluškivanje događaja koji se odvijaju prilikom korištenja aplikacije. UI nit uvijek mora biti dostupna kako bi pratila promjene na ekranu te prikazivala korisniku odgovarajuće elemente korisničkog sučelja, stoga u slučaju ove aplikacije dohvaćanje podataka iz baze, izvršava se na zasebnoj niti.

6.1. JavaRx biblioteka

Reaktivno ili asinkrono programiranje[3] je programska paradigma koja je orijentirana prema tokovima podataka i širenja promjena, odnosno u reaktivnom programiranju protok podataka koje emitira jedna komponenta, proširit će se do drugih komponenti koje su se registrirale da primaju te podatke uz pomoć struktura koje su pružene od strane Rx biblioteka. JavaRx biblioteka sastoji se od tri glavne komponente [5]. *Observables,* odnosno tok podataka koji se šalje s jedne niti na drugu. Važno je naglasiti da *Observables* komponenta emitira podatke jednom u svom životnom ciklusu. Promatrači (engl. *Observers*) su druga komponenta JavaRx biblioteke koji uzimaju i koriste podatke koje *Observables* emitira. *Observers* se pretplaćuje na *Observables* korištenjem ugrađene metode *subscribeOn*(). *Schedulers* je posljednja komponenta JavaRx biblioteke, pomoću svojih ugrađenih funkcija, kao što su *observeOn*() ili *scheduleOn*(), govori ostalim komponentama na kojoj niti da izvršavaju svoj posao. Kako su zahtjevi aplikacije različiti, postoje i različiti tokovi podataka koje *Observables* može emitirati. *Observables* tipovi su sljedeći: *Flowable*, *Observable*, *Single*, *Maybe* i *Completable*. *Flowable* emitira nula ili n stavki koje mogu biti uspješne ili ne, te

[3] https://medium.com/@kevalpatel2106/what-is-reactive-programming-da37c1611382

podržava kontrolu brzine izvora, koji emitira podatke. *Observable* je sličan *Flowable*-u, samo ne podržava kontrolu brzine izvora. Dalje, postoji *Single* koji emitira samo jednu stavku ili vraća događaj koji nije uspio. *Maybe* šalje uspjeh sa stavkom, bez stavke ili s događajem koji nije uspio. Na kraju nalazi se *Completable*, koji emitira događaj koji je uspio ili emitira događaj koji nije uspio, nikada ne emitira stavke. Korištenjem JavaRx tipova podataka, podatci se mogu vrlo lako pretvarati iz jednog Rx tipa podatka u drugi. Tipove podataka i metode s kojima se mogu pretvarati u druge tipove podataka, mogu se vidjeti u tablici 1.

Tablica 1. Prikaz metoda za pretvaranje između Rx tipova

From / To	*Flowabe*	*Observable*	*Maybe*	*Single*	*Completable*
Flowable		*toObservable()*	*reduce(),* *elementAt(),* *firstElement(),* *singleElement()*	*Scan(),* *elementAt(),* *first(),firstOrError()* *last()/lastOrError()* *single()/singleOrErrr()* *all()/any()/count()*	*ignoreElements()*
Observable	*toFlowable*()	*toObservable*()	*reduce()* *elementAt()* *firstElement()* *lastElement()* *singleElement()*	*scan()* *elementAt()* *first()/firstOrError()* *last()/lastOrError()* *single()/singleOrError()* *all()/any()/count()*	*ignoreElements()*
Maybe	*toFlowable()*	*toObservable()*		*toSingle()* *sequenceEqual()*	*toCompletable()*
Single	*toFlowable()*	*toObservable()*	*toMaybe()*		*toCompletable()*
Completable	*toFlowable()*	*toObservable()*	*toMaybe()*	*toSingle()* *toSingleDefault()*	

6.2. Primjer korištenja JavaRx biblioteke

U Android operacijskom sustavu, glavna niti ili nit korisničkog sučelja uvijek mora biti slobodna kako bi korisniku mogla osvježavati elemente korisničkog sučelja i slušati korisničke događaje. Kako bi se podatci mogli dohvatiti ili obraditi na pozadinskoj niti, unutar aplikacije koristi se JavaRx biblioteka. U sljedećim primjerima će se pokazati koje tipove *Observablesa* aplikacija koristi te na koji način se radi s njima u kodu. Za prvi primjer uzet će se *Flowable* koji se koristi prilikom dohvaćanja klijenata iz baze podataka, što se može vidjeti iz koda 10. *View* radi poziv na *ViewModel* da dohvati sve klijente, kako bi ih mogao prikazati korisniku. No da bi *ViewModel* dohvatio sve klijente, mora otići do repozitorija koji pomoću upita nad bazom podataka dohvaća sve klijente i vraća ih nazad *ViewModelu,* u obliku *Flowable* liste. Podatci koje je *View* zatraži putuju i obrađuju se na pozadinskoj niti, dok u međuvremenu glava nit iscrtava korisničko sučelje te sluša daljnje korisničke događaje. U kodu 10. može se vidjeti kako se dohvaćaju podatci putem pozadinske niti, a *View* komponenta koja je pozvala tu metodu se pretplatila na glavnoj niti te čeka rezultat pozvane metode. Također, jedan od problema koji se može riješiti pomoću JavaRx biblioteke, jest osvježavanje podataka dok korisnik obavlja druge zadatke unutar aplikacije. Za takve stvari gdje je samo važna potvrda da li su podatci uspješno obrađeni ili ne, može se koristiti tip *Observable*-a koji se zove *Completable*. *Completable* govori je li događaj koji mu je zadan uspio ili nije. Za primjer *Completable*-a može se uzeti osvježavanje planova treninga unutar baze podataka. Svaki plan treninga sastoji se od naziva, trajanja, frekvencije treninga te željenog cilja. Ukoliko se korisnik odluči za promjenu bilo kojeg svojstva, tijek događanja bih bio isti kao i prilikom dohvaćanja liste klijenata, samo što *ViewModel* ne bi vraćao podatke, već bi obavijestio korisnika jesu li su podatci uspješno ažurirani. Još jedan primjer tipova *Observable*-a koje aplikacija koristi, je *Single*. *Single* omogućava dohvaćanje jedne stavke podataka. U aplikaciji *Single* se koristi za dohvaćanje korisnika prilikom njegovog prijavljivanja u aplikaciju. Ukoliko korisnik nema pristup internetu, u lokalnoj bazi podataka se nalaze njegovi korisnički podatci koji se dohvaćaju i uspoređuju s podatcima koje je korisnik unio prilikom prve prijave u aplikaciju. Budući da je u takvoj situaciji potreban samo jedan objekt korisnika iz baze podataka, za njegovo dohvaćanje koristi se *Single* komponenta.

```
fun getAllClientsFromOfflineDb(){
val disposable = userRepo.getAllClientsFromOfflineDb()
.subscribeOn(Schedulers.io())
.observeOn(AndroidSchedulers.mainThread())
.subscribe({
liveClientList.value = it
},{Log.d("aaa","ERROR: ${it.message}")
it.printStackTrace()
})disposables.add(disposable)}
```

Kod 10. Dohvaćanje klijenata iz baze podataka pomoću JavaRx biblioteke

7. Kalendar unutar aplikacije

Aplikacija za praćenje zdravih navika omogućuje osobnim trenerima da imaju podatke o svim klijentima na jednome mjestu, ali i da prate buduće treninge s klijentima putem integriranog kalendara. Aplikacija jednostavno prikazuje treneru koje obaveze ima danas, na kojoj lokaciji te s kojim klijentom. Ta funkcionalnost izvedena je kroz Google Calendar API (*Application Programming Interface*). Kada trener želi dodati novi događaj, aplikacija ga preusmjerava na Google Calendar, gdje ispunjava podatke o događaju te se taj događaj šalje u aplikaciju. U ovom poglavlju proći će se ukratko Google Calendar API, kako se s njime radi te će se proći kroz primjer korištenja unutar aplikacije.

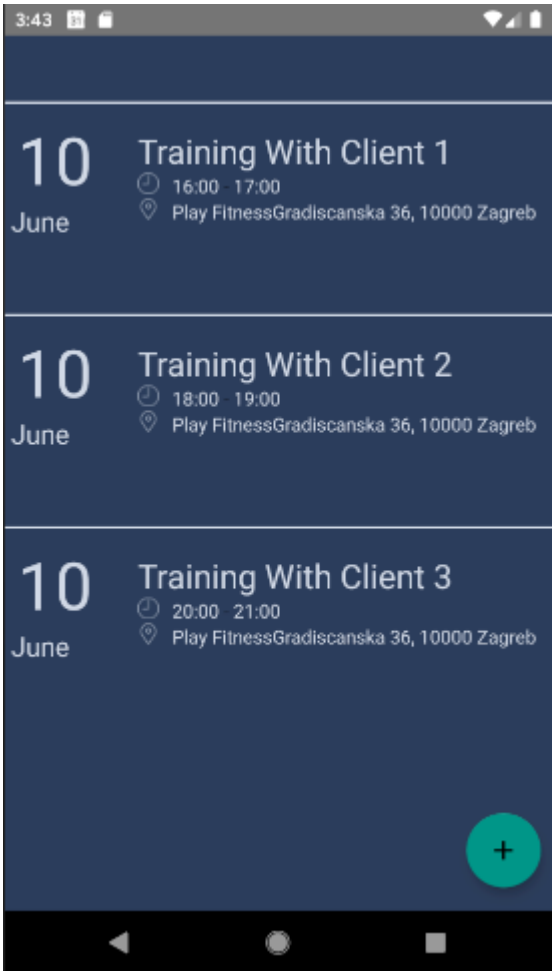
7.1. Google Calendar

Google Calendar dolazi instaliran na svim Android pametnim telefonima. Korištenjem Google API-a programer ima mogućnost integrirati Google Calendar u svoju web ili mobilnu aplikaciju kako bi mogao kreirati, prikazivati ili sinkronizirati već postojeće događaje u kalendaru i samim time olakšati korisniku tranziciju između naše aplikacije i kalendara [6]. Google Calendar osim što ga mnogi ljudi koriste, ima jako dobru podršku i može se koristiti u mnogim drugim okvirima programiranja izvan Androida te ga to čini savršenim odabirom za integraciju unutar ove aplikacije.

7.2. Kreiranje događaja

Google API omogućava kreiranje novih događaja na Googleovom Calendaru iz druge aplikacije. Kreiranje novih događaja može se riješiti na dva načina. Prvi način je da unutar aplikacije postoji forma koju će korisnik popuniti te će se ti podatci omotati u `event` objekt, koji će se proslijediti Google Calendar API-u te će naposljetku biti automatski spremljen u Google Calendar. Drugi način za kreiranje novih događaja je automatsko preusmjeravanje korisnika u Google Calendar, gdje će korisnik također ispuniti podatke vezane za događaj unutar samog Google Calendar okruženja te kada završi Google API će automatski obaviti obradu i spremanje podataka vezane za događaj. Aplikacija za praćenje zdravih navika korisnika koristi Google Calendar API samo kako bi prikazala korisniku koje događaje ima tog dana, stoga implementacija je izvršena kroz preusmjeravanje korisnika u Google

Calendar, koji sam obrađuje i sprema podatke. Razlog prebacivanja odgovornosti na Google Calendar je izbjegavanje spremanja potencijalno osjetljivih podataka u bazu aplikacije. Iz slike se može vidjeti kako su događaji prikazani korisniku. U programskom kodu može se vidjeti da je određen raspon u kojem se prikazuju korisnički događaji, no prije toga slijedi provjera je li korisnik dao aplikaciji dozvolu da čita podatke iz kalendara. Ukoliko aplikacija ima dozvolu za čitanje iz kalendara, zove se Google Calendar API, koji s metodom *getEvents*() dohvaća sve događaje te ih prikazuje korisniku kao na slici 5. Iz programskog koda 11. može se vidjeti da se dohvaćeni događaji filtriraju prema zadanom rasponu i uzimaju se podatci kao što su ime događaja, vrijeme početka, vrijeme završetka, dan u mjesecu i opis te se omotavaju u objekt, koji se prosljeđuje u listu kako bi korisniku imao uvid u sve događaje. Raspon događaja koji se uzimaju iz Google Calendara, je uvijek jedan dan, čime se postiže da korisnik svaki dan dobiva ažurirane podatke. Ukoliko korisnik želi vidjeti događaje koje su već prošli, to može učiniti odlaskom u Google Calendar aplikaciju gdje su svi događaji spremljeni.



Slika 5. Prikaz događaja kalendara unutar aplikacije

```
if(ContextCompat.checkSelfPermission(this.requireContext(),Manifest.permi
ssion.READ_CALENDAR) == PackageManager.PERMISSION_GRANTED){
calendarProvider.getEvents(1).list.forEach {
      if(it.dTStart >= startMillis && it.dTend <= endMillis ){
            var myEvent = CustomEvent()
            myEvent.eventName = it.title
            myEvent.eventHourStart = getHourFromMilis(it.dTStart)
            myEvent.eventHourEnd = getHourFromMilis(it.dTend)
            myEvent.dayOfMonth = getDayOfTheMonthFromMilis(it.dTStart)
            myEvent.description = it.description
            myEvent.location = it.eventLocation
            events.add(myEvent)}
            }}")
```

Kod 11. Dohvaćanje korisničkih događaja iz Google Calendara

8. Korisničke dozvole u Android okruženju

Unutar Android okruženja postoje korisničke dozvole koje štite privatnost korisnika. Svaka aplikacija koja pristupa korisničkim podatcima mora zatražiti dozvolu od korisnika [7]. Te dozvole najčešće se daju prilikom prvog pokretanja aplikacije ili prvog korištenja značajki koja pristupa korisničkim podatcima. Dozvole se prikazuju korisniku u skočnom prozoru, gdje aplikacija uz traženje dozvole, također može pružiti i objašnjenje korisniku zašto je potrebna ta dozvola, kako bi korisnik mogao donijeti odluku prilikom davanja dozvole. U Android sustavu postoje različite vrste dozvola, neke od njih se automatski prihvate od strane operacijskog sustava, dok s druge strane postoje dozvole koje aplikacija izričito mora tražiti od korisnika. Dozvole koje aplikacija mora zatražiti od korisnika, najčešće su vezane za pristupanje korisničkim podatcima. Jedan takav primjer dozvola može biti kada aplikacija traži od korisnika pristup kontaktima, to mogu biti aplikacije kao što su Viber[4] ili WhatsApp[5]. Postoje još i specifične sistemske dozvole koje zahtijevaju korisničku dozvolu, a to su kamera i internet. Android operacijski sustav zadano nema dozvole koje bi obavljale operacije koje bi utjecale na druge aplikacije, operacijski sustav ili samog korisnika. Ukoliko su aplikaciji potrebne dozvole koje ne narušavaju korisničku privatnost ili funkciju uređaja, takve dozvole biti će automatski odobrene od strane operacijskog sustava. Naime, prikazivanje korisniku dozvola koje aplikacija zahtjeva za rad, drugačije je ovisno o verziji Android operativnog sustava kojeg korisnik ima instaliranog. Prije verzije 6.0 korisnik je dodjeljivao dozvole aplikaciji prilikom instalacije, dok nakon verzije 6.0, korisničke dozvole se traže kada se značajke kojima trebaju te dozvole prvi puta pokrenu. Korisničke dozvole podijeljene su na tri razine – normalne, opasne i potpisne. Normalne korisničke dozvole odnose se na dozvole koje su potrebne aplikaciji da pristupi podatcima ili resursima koji su izvan dohvata aplikacije, ali ne predstavljaju rizik korisničkim podatcima. Kada je aplikaciji potreban pristup podatcima ili resursima koji uključuju korisničke podatke ili koje bi mogli utjecati na druge aplikacije, takve dozvole se smatraju opasnim dozvolama.

[4] https://www.viber.com
[5] https://www.whatsapp.com

8.1. Korisničke dozvole unutar aplikacije

Aplikacija za praćenje zdravih navika je usmjerena korisnicima koji imaju verziju Android operacijskog sustava veću od 6.0. Stoga aplikacija će prilikom pokretanja značajki tražiti dozvole. Neke od dozvola koje aplikacija koristi su: čitanje i pisanje u kalendar, dohvaćanje *wi-fi* te mrežnog stanja. U kodu 12. može se vidjeti kako izgleda traženje dozvole od korisnika. Najprije mora se provjeriti da li je korisnik već dao dozvolu, koristeći se klasom `ContextCompat` koja u sebi ima statičku metodu za provjeru korisničkih dozvola `checkSelfPermission`. Metoda `checkSelfPermission` prima dva argumenta, jedan je kontekst u kojem se traži dozvola, a drugi je naziv dozvole koja se traži. Kod 13. prikazuje Manifest koji sadrži sve dozvole koje aplikacija koristi. Manifest je XML datoteka koja sadrži sve informacije o aplikaciji, Android alatima i Google Play Store-u. Da bi se pristupilo određenoj dozvoli, potrebno je pozvati statičku klasu Manifest koja u sebi sadrži sve dozvole u Android okruženju.

```
if(ContextCompat.checkSelfPermission(this.requireContext(),
Manifest.permission.READ_CALENDAR) ==
PackageManager.PERMISSION_GRANTED)
```

Kod 12. Provjera dozvole za čitanje kalendara unutar koda

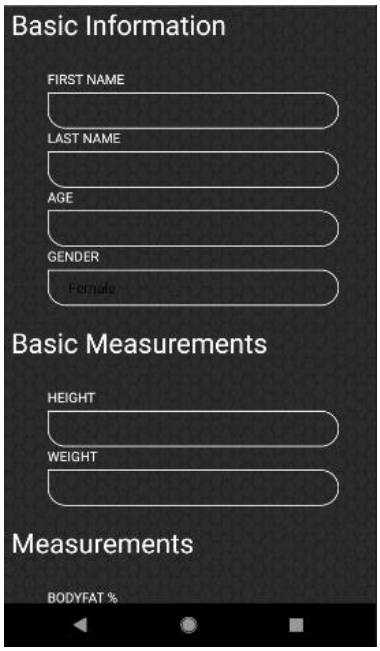
```
<uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission
android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission
android:name="android.permission.READ_CALENDAR"/>
    <uses-permission
android:name="android.permission.WRITE_CALENDAR"/>
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission
android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE"/>

    <uses-feature
android:name="android.hardware.sensor.stepcounter"
android:required="true"/>
    <uses-feature
android:name="android.hardware.sensor.stepdetector"
android:required="true"/>
```

Kod 13. Dozvole unutar manifesta koje aplikacija koristi

9. Praćenje napretka klijenta

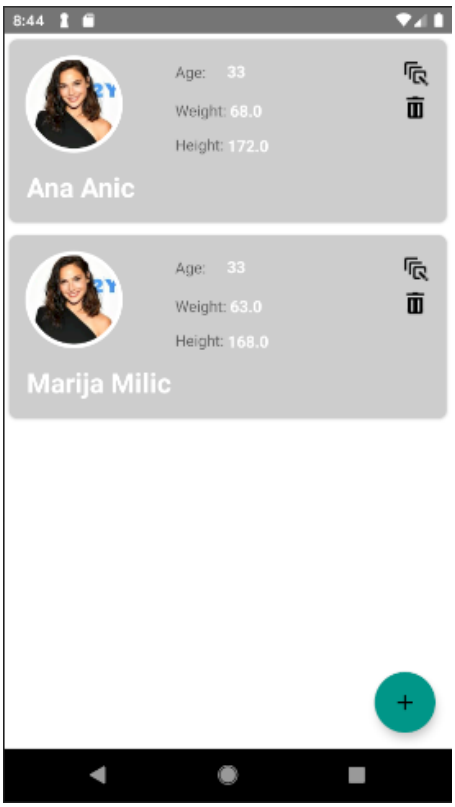
Aplikacija za praćenje zdravih navika korisnika omogućava korisnicima koji imaju ulogu trenera unutar aplikacije, da prate napredak svojih klijenata. Kako bi trener pratio napredak svakog klijenta, prvo je potrebno da ispuni podatke o klijentu, koji se tada dodaje u listu klijenata. Iz slike 6. može se vidjeti forma koju trener ispunjava prilikom dodavanja novog klijenta.



Slika 6. Forma za dodavanje novog klijenta

Nakon što je klijent uspješno dodan u listu klijenata, treneru se prikazuje u obliku kartice na koju može pritisnuti kako bi dobio detaljniji prikaz individualca. Na slici 7. može se vidjeti kako lista klijenata izgleda. Prilikom pritiska na karticu klijenta, trener dobiva detaljan uvid u sve informacije klijenta te njegovom posljednjem treningu. Također, prilikom pritiska na ikonu informacija, trener dobiva statistički pregled svih mjerenja

klijenta. Statistički pregled prikazan je pomoću kartica na kojima su prikazane trenutne mjere te razlika između prošlog i trenutnog mjerenja.



Slika 7. Prikaz kartica klijenata

9.1. Praćenje napretka klijenta unutar koda

U aplikaciji nakon što trener doda klijenta u listu, klijent se sprema u lokalnu bazu podataka te ukoliko korisnik ima pristup internetu, informacije o klijentu se šalju u oblak. Za spremanje informacija o klijentu u lokalnu bazu podataka, *View* `AddClientActivity` prvo uzima podatke koje je korisnik unio kroz formu za dodavanje klijenta i te podatke omotava u objekt koji se zatim prosljeđuje u *ViewModel* metodom `saveClientToOfflineDB`, što se može vidjeti u kodu 13. *ViewModel* tada odlazi do sloja pristupa podatcima (DAO), gdje se zove metoda koja proslijeđeni objekt sprema u bazu. Prilikom spremanja informacija o klijentu na oblak, procedura je vrlo slična kao i onoj kod spremanja u lokalnu bazu, samo prije nego što aplikacija pozove metodu za pohranu podataka na oblak provjerava da li korisnik ima pristup internetu. Svaki objekt klijenta unutar koda ima svojstva koja pohranjuju trenutna mjerenja, ali i prethodna mjerenja. Za

prikaz statističkih podataka klijenta unutar *View* komponente, aplikacija putem paketa (engl. *Bundle*) šalje odabranog klijenta u *View*, uzima podatke njegovih mjerenja i oduzima trenutna mjerenja s prethodnim mjerenjima, na temelju toga se prikazuje napredak klijenta.

```
class AddClientActivity: AppCompatActivity()  {
      @Inject
      lateinit var factory: AddClientVMFactory
      private var clientViewmodel:AddClientViewModel? = null

override fun onCreate(savedInstanceState: Bundle?) {
      AndroidInjection.inject(this)
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_add_client)
      clientViewmodel = ViewModelProviders.of(this,factory).
      get(AddClientViewModel::class.java)
}
add_btn_finish.setOnClickListener {
      val newClient = getNewClient()
      clientViewmodel!!.saveClientToOfflineDB(newClient)
      setResult(Activity.RESULT_OK)
      finish()
}
```

Kod 14. *View* komponenta poziva *ViewModel* za pohranu klijenta

10. Pregled postojećih rješenja

Trenutno na Google PlayStore-u postoje stotine različitih fitness aplikacija, svaka sa svojim osobnim značajkama. U ovome poglavlju predstaviti će se nekoliko najpopularnijih aplikacija, individualno ih se opisati te naposljetku usporediti aplikacije i njihove značajke s ovim radom.

10.1. Bodyspace aplikacija

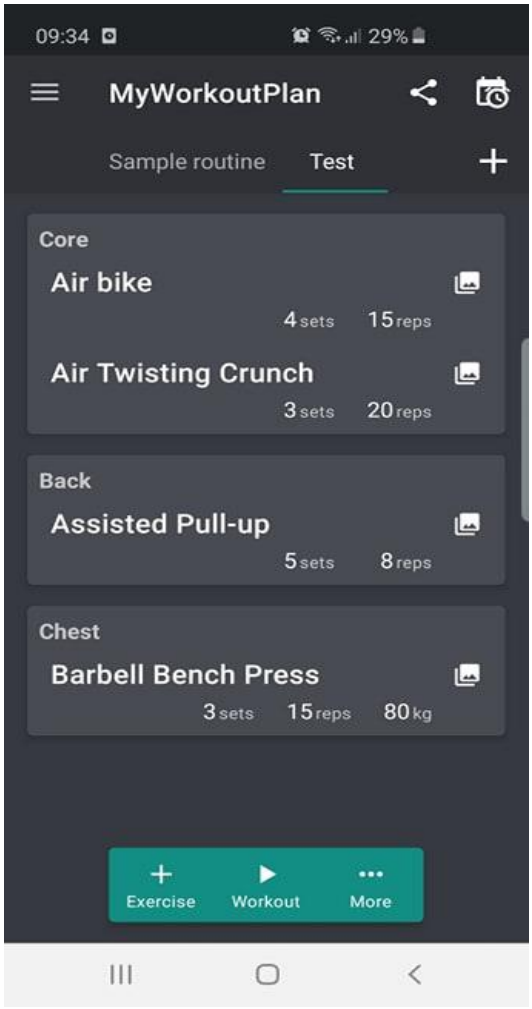
Bodyspace[6] aplikacija izrađena je u obliku fitness društvene mreže. Prilikom izrade korisničkog računa potrebno je odgovoriti na dvije vrste pitanja. Prvi niz pitanja usmjeren je prema konačnom cilju prilikom treniranja te vježbačkom iskustvu pojedinca. Naposljetku korisnik odgovara na osobna pitanja, kao što su spol i godine. Nakon prolaska faze kreiranja korisničkog računa, aplikacija korisniku predlaže već postojeće planove treninga koji se mogu pratiti unutar aplikacije. Ukoliko korisnik odluči da ne želi pratiti postojeće planove treninga, ima opciju da sam kreira vlastiti plan s vježbama koje izabere i vlastitim ciljem treninga. Jedna od glavnih značajki Bodyspace aplikacije je njihova velika baza podataka koja sadrži preko tisuću vježbi koje su vrlo detaljno opisane i popraćene video zapisom, gdje korisnik može vidjeti kako se pravilno izvodi odabrana vježba. Bodyspace svojim korisnicima nudi i opciju da pregledaju povijest treninga koje su odradili do sada, mogućnost kreiranja vlastitog predloška za trening te njegovog dijeljenja s ostalim korisnicima. Aplikacija također ima i element društvene mreže u sebi, pa tako korisnik može učitavati i postavljati vlastite slike, pretražiti i dodati druge osobe na listu prijatelja te postavljati statuse, koji su formatom vrlo slični onima na Facebook-u.

10.2. MyWorkoutPlan App aplikacija

MyWorkoutPlan App[7] je aplikacija koja uz svoje jednostavno korisničko sučelje, pruža korisniku uvid u vlastite treninge i napretke na jednome mjestu. Na slici 8. može se vidjeti početni zaslon aplikacije. Uz pomoć velike baze podataka vježbi, korisnik vrlo lako može kreirati željeni plan treninga u svega nekoliko klikova. Svaka vježba unutar aplikacije

[6] https://www.whatsapp.com
[7] https://play.google.com/store/apps/details?id=com.myworkoutplan.myworkoutplan&hl=en

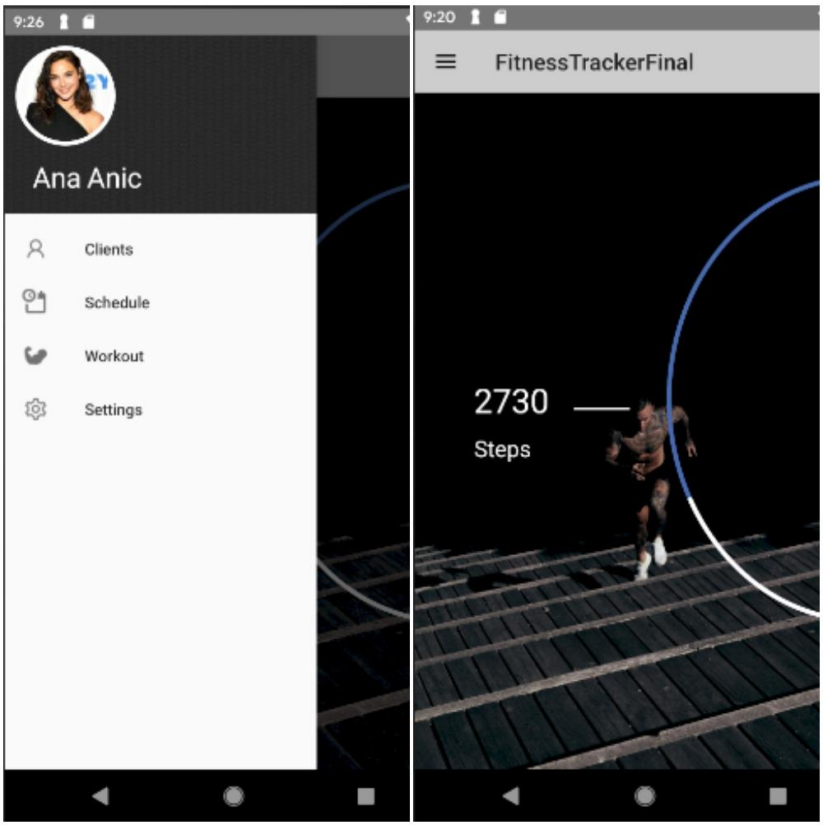
prikazuje koja se mišićna grupa radi tijekom izvođenja odabrane vježbe te je popraćena s kratkom animacijom koja daje korisniku uvid u kako ta vježba izgleda. Kako bi se korisnik što lakše i brže snašao prilikom izrade vlastitih treninga, aplikacija nudi opciju sortiranja svih vježbi prema željenoj mišićnoj skupini ili opremi s kojom vježbač trenutno raspolaže. Aplikacije ima ugrađeni sustav podsjetnika, gdje korisnik može kreirati podsjetnike na dnevnoj ili tjednoj bazi da bi što lakše pratio svoju željenu rutinu. Ukoliko korisnik ima želju otključati cijelu verziju MyWorkoutPlan aplikacije, za to mu potrebno izdvojiti 15 dolara kako bi mogao neograničeno koristiti aplikaciju i sve njene značajke. Ako se korisnik odluči platiti 15 dolara, otključati će mu se dodatne mogućnosti kao što su: sinkronizacija podataka na više uređaja, uvid u statistiku odrađenih treninga, spremanje i vraćanje planova treninga te svaka nova značajka koja dođe u aplikaciju, biti će potpuno besplatna.



Slika 8. Glavni zaslon MyWorkoutPlan aplikacije

10.3. Aplikacija za praćenje zdravih navika korisnika

Aplikacija za praćenje zdravih navika korisnika usmjerena je da pomogne rekreativcima, ali i osobnim trenerima da imaju sve podatke o napretku i obavljenim treninzima na jednome mjestu. Za korištenje aplikacije potrebno je izraditi korisnički račun. Prilikom registracije aplikacija od korisnika zahtjeva da odabere jednu od dvije uloge u aplikaciji, a to su uloga trenera ili rekreativca. Nakon odabira uloge, aplikacija od korisnika zahtjeva unos osobnih podataka, kao što su spol, godine i osnovna mjerenja potrebna za praćenje vlastitog napretka. Nakon potvrde korisničkih podataka, izrađuje se račun te je tada korisnik usmjeren na glavni zaslon aplikacije. Slika 9. prikazuje glavni zaslon aplikacije. Aplikacija bez obzira na ulogu odabranu u procesu registracije, pruža mogućnost praćenja svakodnevnih navika, kao što su broj koraka, vrste odrađenih vježbi na treningu, trajanje pojedinih treninga i praćenje vlastitih mjerenja. Aplikacija također korisniku nudi uvid u statističke podatke, gdje se mogu vidjeti promjene u odnosu na početna mjerenja. Korisnici koji su odabrali ulogu trenera imati će dodatne mogućnosti, koje će im omogućiti izradu liste klijenata te izradu planova treninga za individualne klijente i naposljetku kalendar događaja koji im omogućava da prate buduće termine treninga.



Slika 9. Glavni zaslon Aplikacije za praćenje zdravih navika korisnika

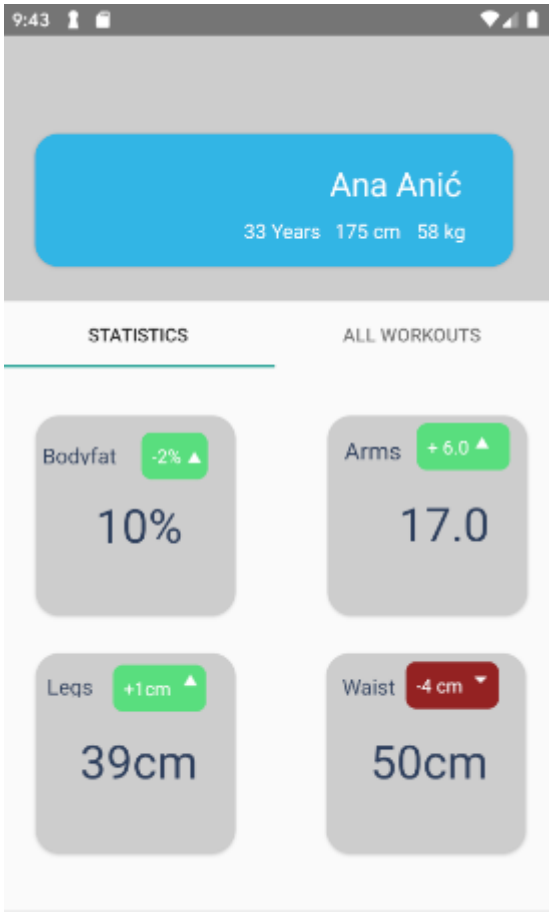
10.4. Usporedba postojećih rješenja

Svaka od tri navedene aplikacije ima svoju glavnu značajku iza sebe prema kojoj je cijela aplikacija usmjerena. Bodyspace aplikacija je jedina od tri navedene aplikacije koja ima elemente društvene mreže u sebi i to je čini jedinstvenom na tržištu. Korisnici se mogu međusobno povezati, dijeliti treninge, slike ili objave, što je u današnje vrijeme vrlo popularno. Bodyspace i MyWorkoutPlan App imaju velike baze podataka vježbi, što olakšava korisnicima izradu vlastitog plana treninga ili ljudima koji su novi u svijetu fitnessa da pravilno izvode pokrete i tako smanje rizike od povreda. Unatoč sličnoj bazi podataka, Bodyspace aplikacija uz video zapise vježbi, nudi i opširne komentare na svaku vježbu, tako da korisnik ne samo da zna kako izgleda pokret koji pokušava napraviti, već i da ima na umu česte pogreške prilikom izvođenja određenih vježbi. Stvar koja najviše ističe Bodyspace aplikaciju iz mnoštva fitness aplikacija je da pruža gotove planove treninga koji su izrađeni od poznatih stručnjaka u području fitnessa, što može pomoći korisnicima koji ne znaju sami izraditi vlastite planove treninga ili jednostavno ne znaju kojem programu vježbanja mogu vjerovati. Bodyspace aplikacija namijenjena je osobama različitih razina iskustva, korisnicima koji su novi u svijetu fitnessa omogućava praćenje već gotovih planova treninga, ali također nudi slobodu prilikom izrade treninga iskusnijim vježbačima.

MyWorkoutPlan App je aplikacija usmjerena na jednostavno korisničko sučelje, koje omogućava korisnicima lako kreiranje vlastitih planova treninga. Aplikacija također sadrži veliku bazu podataka vježbi od kojih korisnik može sortirati vježbe prema mišićnoj skupini, ali i opremi koju korisnik ima dostupnu. Glavni nedostatak baze podataka vježbi je manjak objašnjena kako se vježbe izvode, niti stvari na koje bi vježbač trebao pripaziti prilikom izvođenja pokreta. Aplikacija također nema gotove planove treninga, već korisniku samo omogućava da kreira vlastiti plan treninga i prati svoju tjelesnu težinu. Također, još jedna od nedostataka ove aplikacije je 15 dolara koje korisnik mora platiti kako bi otključao sve značajke. MyWorkoutPlan App namijenjen je osobama koje imaju iskustva u treniranju i koje žele imati sve odrađene treninge na jednome mjestu.

Naposljetku tu je Aplikacija za praćenje zdravih navika korisnika koja je kreirana da zadovolji potrebe osobnih trenera, ali i rekreativaca. Svojim jedinstvenim značajkama omogućuje osobnim trenerima da prate napredak svih svojih klijenata, što se može vidjeti na slici 10. Zahvaljujući integriranom kalendaru, aplikacija im također omogućava da kreiraju i pregledaju buduće događaje sa svojim klijentima. S druge strane rekreativcima ova

aplikacija pruža brojne mogućnosti praćenja svojih svakodnevnih sportskih navika, kroz brojač koraka do mogućnosti kreiranja, uređivanja i pregleda treninga te funkcionalnosti za unos vlastite tjelesne težine i drugih mjerenja, kako bi korisnik mogao pratiti vlastiti napredak.



Slika 10. Prikaz napretka klijenta

Zaključak

Programiranje mobilnih aplikacija iziskuje od programera da konstantno razvija svoje znanje i uči nove tehnologije. Svijet mobilnih aplikacija se razvija brže od bilo koje druge grane programiranja, stoga je potrebno da programer ostane informiran te da vlada znanjima iz trenutnih tehnologija.

Kroz ovaj rad naučio sam puno o važnosti arhitekture, i to ne samo prilikom razvijanja mobilnih aplikacija. Arhitektura omogućava aplikacijama lakše održavanje koda u budućnosti te također proširivanje aplikacije ukoliko postoji potreba za time. Također ovaj rad me naučio da je vrlo važno planirati unaprijed, planirati model baze podataka, modele, odnosno potencijalne klase unutar programa, odabir arhitekture te na kraju odabir tehnologija. U svijetu programiranja svaki problem se može riješiti na više načina, tako da treba dobro promisliti o pristupu prije rješavanja istoga. Prilikom izrade rada, najteži dio je bio rad s JavaRx bibliotekom te dohvaćanje i obrada podataka na pozadinskoj niti aplikacije. JavaRx je vrlo kompleksna biblioteka koja ima jako puno mogućnosti, čije ova aplikacija nije uspjela iskoristiti u potpunosti. Iako je rad s JavaRx bibliotekom bio najteži, također je bio i najzanimljiviji, smatram da obrada podataka i razvoj na više niti jako koristan za svakog programera.

Dobivena znanja prilikom izrade ovog rada su najvrjednija i smatram ih kao dobrom podlogom za daljnje razvijanje u smjeru izrade mobilnih aplikacija.

Popis kratica

API	*Application Programming Interface*	Aplikacijsko programsko sučelje
DAO	*Data Access Layer*	Sloj pristupa podatcima
SDK	*Software Development Kit*	Komplet za razvoj softvera
UI	*User Interface*	Korisničko sučelje
UID	*Unique Identifier*	Jedinstveni Identifikator

Popis slika

Popis tablica

Popis kôdova

Literatura

[1]. MVVM architecture, ViewModel and LiveData, https://proandroiddev.com/mvvm-architecture-viewmodel-and-livedata-part-1-604f50cda1

[2]. SQLite, https://www.sqlite.org/index.html

[3]. Cloud Firestore, https://firebase.google.com/docs/firestore/

[4]. Save data in a local database using Room, https://developer.android.com/training/data-storage/room/index.html

[5]. Using RxJava, https://www.vogella.com/tutorials/RxJava/article.html

[6]. Google Calendar API, https://developers.google.com/calendar

[7]. Permissions overview, https://developer.android.com/guide/topics/permissions/overview

ALGEBRA
VISOKO UČILIŠTE

NASLOV ZAVRŠNOG RADA

Pristupnik: Hrvoje Horvat, JMBAG

Mentor: prof. dr. sc. Dobar Voditelj