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

Sveučilišni studij

ANDROID APLIKACIJA ZA PRAĆENJE PROMJENE UZORAKA KRVNOG TLAKA DOBIVENIH 24-SATNIM NOŠENJEM HOLTERA

Završni rad

Anamaria Franjić

Osijek, 2024.

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

Obrazac Z1P: Obrazac za ocjenu završnog rada na sveučilišnom prijediplomskom studiju

Ocjena završnog rada na sveučilišnom prijediplomskom studiju

Ime i prezime pristupnika:	Anamaria Franjić
Studij, smjer:	Sveučilišni prijediplomski studij Elektrotehnika i informacijska tehnologija
Mat. br. pristupnika, god.	4909, 21.09.2020.
JMBAG:	0165088236
Mentor:	izv. prof. dr. sc. Ivica Lukić
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Android aplikacija za praćenje promjene uzoraka krvnog tlaka dobivenih 24-satnim nošenjem holtera
Znanstvena grana završnog rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak završnog rada:	Aplikacija omogućuje unos osobnih podataka gdje je obavezno navesti ime, prezime, datum rođenja i godinu. Nakon prijave aplikacija bi tražila unos oznake i imena samog holtera koji će osoba nositi naredna 24 sata. Prilikom pokretanja holtera, korisnik bi unutar aplikacije morao navesti početno vrijeme kad bi krenulo mjerenje jer holter mjeri 24h. Holter bi aplikaciji nakon svakog mjerenja slao podatke tj. uzorke tlaka koje bi aplikacija spremala u tablicu kako bi korisniku bilo lakše pratiti promjene. Budući da za svaku dobnu skupinu postoje neka odstupanja od normalnog tlaka, aplikacija bi nakon svakog mjerenja ovisno o

Datum prijedloga ocjene završnog rada od strane mentora:	28.06.2024.
Prijedlog ocjene završnog rada od strane mentora:	Izvrstan (5)
Datum potvrde ocjene završnog rada od strane Odbora:	15.07.2024.
Ocjena završnog rada nakon obrane:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije završnog rada čime je pristupnik završio sveučilišni prijediplomski studij:	15.07.2024.

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

IZJAVA O IZVORNOSTI RADA

Osijek, 15.07.2024.

Ime i prezime Pristupnika:	Anamaria Franjić
Studij:	Sveučilišni prijediplomski studij Elektrotehnika i informacijska tehnologija
Mat. br. Pristupnika, godina upisa:	4909, 21.09.2020.
Turnitin podudaranje [%]:	5

Ovom izjavom izjavljujem da je rad pod nazivom**: Android aplikacija za praćenje promjene uzoraka krvnog tlaka dobivenih 24-satnim nošenjem holtera**

izrađen pod vodstvom mentora izv. prof. dr. sc. Ivica Lukić

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

Sadržaj

1.UVOD

U današnje vrijeme, gotovo milijarda ljudi u svijetu boluje od hipertenzije ili visokog krvnog tlaka. Hipertenzija je predstavljena kao tiha bolest koju simbolizira omjer dviju vrijednosti. Gornja vrijednost označava sistolički tlak ili arterijski tlak kada se srce steže dok s druge strane, donja vrijednost označava dijastolički tlak ili arterijski tlak kada se srce opušta. Vrijednost koja se smatra 'normalno izmjerenom vrijednosti tlaka' je upravo vrijednost 120/80 milimetara stupca žive (mmHg). Najveći problem s kojim se danas susreću oboljeli od hipertenzije je nedovoljna informiranost o tome koje organe bolest zahvaća i koja su potencijalna rješenja kako bi se smanjio utjecaj jačine bolesti. Uzrok nedovoljnoj informiranosti je sama bolest, koja unatoč simptomima koji su uobičajeni i slični nekim drugim bolestima ne mora godinama pokazivati nikakve simptome. Upravo zbog te svoje 'nenametljivosti' bolest uzrokuje probleme u kasnijim godinama života, stvarajući oštećenja na srcu, mozgu i ostalim organima koji su povezani arterijama. Neki od simptoma koji mogu poslati jasnu poruku da s tijelom nešto nije uredu su: jaka glavobolja uz prisutnost osjećaja pritiska u glavi, učestali umor, osjećaj prekomjernog znojenja uz prisutnost naglog lupanja srca, neočekivano krvarenje iz nosa, oslabljeni vid i mučnina. Nažalost, zbog prevelike zastupljenosti ovih simptoma kod drugih bolesti, osobe s hipertenzijom nisu ni svjesne da su izložene nastanku kardiovaskularnih bolesti, posebice bolesti srca i moždanog udara. Hipertenziju nije moguće odrediti jednim mjerenjem tlaka, već upravo suprotno. To je bolest čiji se napredak mora pratiti svakodnevnim mjerenjima, kako bi na vrijeme uspjeli otkloniti rizik od neželjenih opasnih ishoda.

Android aplikacija razvijena u ovom radu predstavlja komercijalno rješenje korisnicima uz brz pristup i jednostavno korisničko sučelje. Cilj ove aplikacije je olakšati i omogućiti korisniku detaljno praćenje mjerenja tlaka i informiranje o problemima koji mogu nastati ukoliko tlak prekorači dozvoljenu granicu koja je prethodno definirana u tekstu, a iznosi 120/80 mmHg. Ideja aplikacije temelji se na tome da korisnik prilikom ulaska u aplikaciju prvo mora obaviti prijavu ili/i registraciju pomoću koje ostaje zabilježen u bazi podataka. Pored toga što mora obaviti prijavu, korisnik ulaskom u aplikaciju ostvaruje mogućnost pohranjivanja svojih podataka mjerenja tlaka u listu koja mu daje zorni prikaz promjene stanja njegovog tlaka. Osim toga, aplikacija će korisniku omogućiti uvid u grafički prikaz promjene stanja njegovog tlaka na temelju mjerenja koje je korisnik prethodno unio i pohranio unutar liste. Uz sve to, aplikacija pruža korisniku potrebne informacije poput : „Što je hipertenzija?“, „Što je hipotenzija?“, „Koje namirnice su prikladne za snižavanje krvnog tlaka, a koje su prikladne za povišenje krvnog tlaka?“ i

druge koje su usko vezane promjene životnih navika. Realiziranje ideje ovog završnog rada podijeljeno je kroz poglavlja. Prije svega, na samom početku potrebno je upoznati okolinu s postojećim rješenjima i alatima koja pružaju slične funkcionalnosti kao ova Android aplikacija. Uz to važno je napomenuti i objasniti primjenu tehnologija koje su se uvelike koristile pri izradi aplikacije krenuvši od programskog koda koji je pisan programskim jezikom Kotlinom pa sve do Firebase baze podataka unutar koje su pohranjeni svi podatci korisnika i njegovih mjerenja. Naposljetku, struktura i funkcionalnosti aplikacije predstavlja glavno poglavlje unutar kojeg su objašnjeni izgledi pojedinog zaslona aplikacije kao i njegova funkcionalnost.

1.1.Zadatak završnog rada

Zadatak ovog završnog rada predstavlja realizaciju android aplikacije uz pružanje potencijalnog rješenja koje bi korisnike uvelike osvijestilo o problemima koji mogu nastati ukoliko zanemare činjenicu da im tijelo šalje signale o postojećim promjenama koje mogu biti životno opasne. Korisniku je potrebno omogućiti registraciju i prijavu. Uz to, prijavljenom korisniku pruža se mogućnost unosa i pohrane mjerenja tlaka te uvid u grafikon koji mu daje 'slikoviti' prikaz promjene njegova tlaka. Također, prijavljeni korisnik ima pristup novostima vezanima uz krvni tlak.

2. PREGLED POSTOJEĆIH RJEŠENJA

U nastavku je unutar ovog poglavlja opisana i prikazana nekolicina postojećih rješenja koja pružaju istu ili sličnu funkcionalnost. Jedan dio rješenja fokusira se na praćenje promjena cjelokupnog rada organizma, dok se drugi dio rješenja fokusira isključivo na praćenje promjene tlaka.

2.1. Heart Rate & Pulse Monitor -Pulse App

Heart Rate Monitor je mobilna aplikacija koja korisniku omogućuje mjerenje pulsa i otkucaja srca jednostavnim pritiskom jagodice prsta na kameru ili senzor koji se nalazi na stražnjoj strani mobitela (Sl.2.1.) . Ideja aplikacije zasniva se na praćenju mjerenja koja se obavljaju nekoliko puta dnevno te implementaciji tih mjerenja na grafikonu kako bi se korisniku osigurala potrebna analiza zdravstvenog stanja. Ovakav tip aplikacije nije preporučeno koristiti za hitne slučajeve s obzirom da ova aplikacija pruža samo okvirni pregled stanja pulsa i otkucaja srca [1].

Slika 2.1. Izgled aplikacije *Heart Rate & Pulse Monitor*

2.2. Heart Monitor: Measure BP & HR

Heart Monitor je mobilna aplikacija koja pomoću tehnologije skeniranja lica korisniku pruža mogućnost mjerenja znakova zdravlja srca poput gornjeg i donjeg tlaka, otkucaja srca i drugih (Sl.2.2.) . Korisnik prilikom novih mjerenja mora gledati usmjereno prema kameri kako bi kamera očitala vrijednosti pulsa, tlaka, disanja. Uz to, aplikacija također nudi vodič za korisnike unutar kojeg se nalaze razne informacije vezane uz promjene načina života te ozbiljnosti koje nosi hipertenzija [2].

Slika 2.2. Izgled korisničkog sučelja aplikacije *Heart Monitor: Measure BP & HR*

2.3. Blood Pressure Tracker

Blood Pressure Tracker je aplikacija koja korisniku omogućuje bilježenje i praćenje promjena krvnog tlaka sukladno kalendaru. Uz unos tlaka pruža korisniku mogućnost unosa pulsa, glukoze, kisika i težine. Korisnik prilikom unosa svojih mjerenja odabire datum prema kojem će mjerenja ostati zabilježena u aplikaciji. Unosom mjerenja, korisnik dobiva prikaz promjene svojih mjerenja u grafikonu uz sažetak svih mjerenja prikazanih unutar tablice (Sl.2.3.) . Grafikoni daju detaljan osvrt prema unesenim mjerenjima, prikazuju nalaze li se mjerenja većinski u stanju hipertenzije ili hipotenzije, također pružaju uvid u položaj pulsa ovisno prelazi li normalne granice ili ne prelazi [3].

Slika 2.3. Izgled korisničkog sučelja aplikacije *Blood Pressure Tracker*

2.4. MyBP – Blood Pressure App

Aplikacija *MyBP* je Android aplikacija namijenjena korisnicima koji imaju problem s krvnim tlakom. Ova aplikacija korisnicima osigurava jednostavan način praćenja promjena vezanih uz krvni tlak. Zbog svog bogatog korisničkog sučelja, aplikacija nudi veliki broj mogućnosti. Počevši od unosa mjerenja tlaka uz mogućnost prikaza povijesti očitanja tijekom vremena pa sve do podsjetnika vezanih uz životni stil i upravljanje planovima koji su usko vezani uz zdraviji način života. Aplikacija zbog svojih algoritama za prepoznavanje uzoraka pruža korisnicima personalizirane obavijesti i podsjetnike vezane za lijekove (Sl.2.4.) . Uz sve to, na kraju aplikacije se nalazi odjeljak u kojem su smješteni članci, videozapisi i ostali savjeti koji mogu biti od koristi u borbi protiv hipertenzije [4].

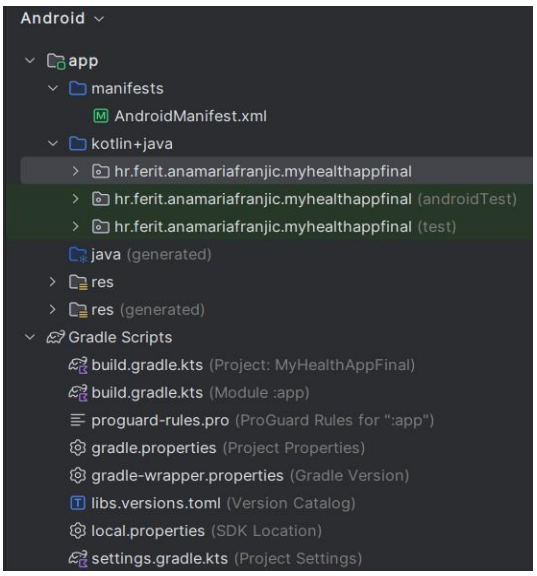
Slika 2.4. Izgled korisničkog sučelja aplikacije *MyBP – Blood Pressure App*

3.OPIS I PREGLED KORIŠTENIH TEHNOLOGIJA

Ovo poglavlje u nastavku nudi osvrt na teorijski dio rješavanja problematike ovog završnog rada uključujući navođenje svih dodatnih funkcionalnosti koje su korištene unutar programskog koda radi uspješnog kreiranja aplikacije. Uz to, detaljno je opisana svrha same aplikacije. Korisničko sučelje aplikacija dizajnirano je pomoću XML datoteka dok je kod pisan programskim jezikom Kotlin. Aplikacija je zajedno s programskim kodom implementirana u integriranom razvojnom okruženju Android Studio. Podatci koji se unose u aplikaciji spremaju se u *Realtime Database* i *Authentication* bazu podataka definiranu u *Googleovom Firebaseu*. Svrha ove Android aplikacije namijenjene korisnicima, temelji se na pružanju uvida u podatke te unosu i izmjeni podataka. Korisnik bi nakon registracije i prijave imao jedinstven pristup aplikaciji na način da bi mu prilikom registracije i definiranja korisničkog imena bila dodijeljena jedinstvena oznaka. Svakom idućom prijavom s prethodno definiranim korisničkim imenom i ispravnom zaporkom, korisnik biva prepoznat u sustavu jer je već registriran u bazi pod jedinstvenom oznakom. Nakon prijave korisniku se pruža mogućnost samostalnog unosa vrijednosti tlaka i spremanja istih u listu kako bi dobio sve potrebne informacije o promjeni tlaka tijekom vremena. Budući da je korisnik prijavljen pod jedinstvenom oznakom, njegovo unošenje mjerenja tlaka će također biti spremljeno pod tom jedinstvenom oznakom i predstavljat će podčvor baze.

3.1.Android Studio

Android Studio IDE (engl. *Integrated Development Environment*) predstavlja službeno integrirano razvojno okruženje za razvijanje Android aplikacija za Android operacijski sustav. Temelj Android Studio razvojnog alata počiva na *IntelliJ IDEA*; integriranom razvojnom okruženju koje ima naglasak na Javu, a uz to omogućuje automatsko dovršavanje koda što uvelike pomaže kod pisanja programerskog koda. Uz automatsko dovršavanje koda, Android Studio podržava više jezika odnosno pruža podršku za programske jezike od kojih su najpoznatiji: Java, C++ i Kotlin. Također Android Studio sadrži poboljšani emulator koji programeru omogućuje pokretanje aplikacije bez potrebe spajanja fizičkog uređaja. Emulator se u Android Studiu ponaša kao stvarni Android uređaj jer provodi emulaciju svih funkcija koje i stvarni fizički uređaj provodi. Nadalje, projekti su u Android Studiu strukturirani prema modulima, tri su najvažnija: prvi su moduli Android aplikacije koji sadrže mapu *manifest* (*Android manifest.xml* datoteka), mapu Java (sadrži izvorni kod Kotlin i Java) i mapu *res* (sadrži datoteke koje su vezane uz uređenje korisničkog sučelja), drugi su knjižnični moduli i treći su *Google App Engine* moduli [5].



Slika 3.1. Isječak modula iz Android Studia

3.2.Programski jezik Kotlin

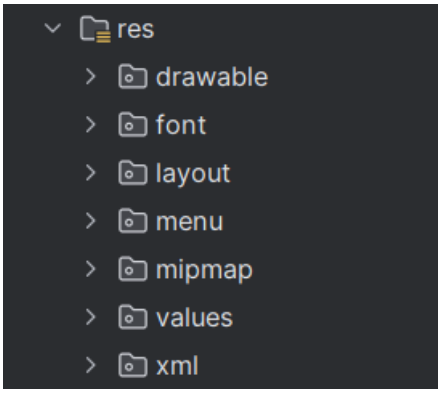
Programski jezik Kotlin, dobio je ime prema otoku Kotlinu koji se nalazi u blizini ruskog grada Saint Petersburga. Treći je *Googleov* programski jezik dok su preostala dva još Java i C++. U lipnju 2011. godine tvrtka JetBrains objavila je projekt Kotlin. Nedugo nakon toga u veljači 2012. godine projekt Kotlin biva predstavljen javnosti kao jezik otvorenog koda (engl. *open-source*) . Prva verzija programskog jezika Kotlina, Kotlin v1.0, objavljena je 2016. godine. Tri godine nakon objave prve verzije, Kotlin postaje najzastupljenijim službenim standardom za razvoj Android aplikacija. Prisutan je na brojnim platformama uključujući *Windows* i *Linux*. Dizajniran u potpunosti za rad s Javom. Za razliku od Jave, Kotlin ne zahtjeva točka-zarez znak na kraju reda već je dovoljan samo prelazak u novi red. Uz to sadrži statičke tipove koji zahtijevaju prethodnu definiciju tipa svake varijable. Varijable mogu biti *val* i *var*, gdje je *val* varijabla samo za čitanje, a *var* predstavlja promjenjiv tip varijable (Sl.3.2.) .

```
class Cures(
    var image : Int,
    var title : String,
    var descriptions : String
)
```

Slika 3.2. Prikaz deklariranja varijabli pomoću programskog jezika Kotlin

3.3.XML

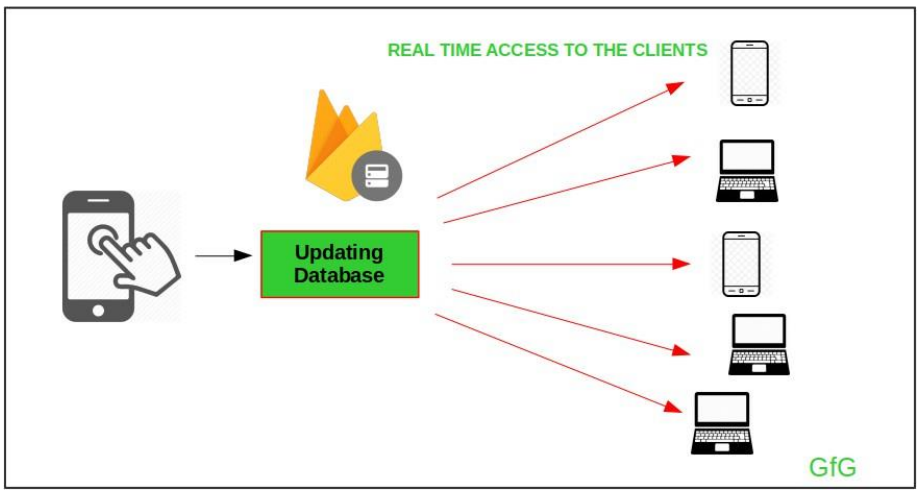
XML (engl. *eXtensible Markup Language*) je jezik koji koristi oznake za definiranje podataka. XML se u Android Studiu koristi prilikom dizajniranja korisničkog sučelja. Omogućuje uporabu proširivih oznaka, što znači da XML oznake nisu prethodno definirane niti ograničene već ih je moguće proizvoljno kreirati. Potpuno je neovisan za pohranu, prijenos i dijeljenje podataka. Prilikom kreiranja Android aplikacije, u *res* mapi nalaze se podmape ili poddirektoriji unutar kojih developeri aplikacije mogu prilagoditi izgled zaslona prema vlastitim željama. Unutar podmape *drawable* moguće je spremiti sve slike koje se koriste pri kreiranju izgleda samog zaslona. Uz slike, moguće je dodati i ikone koje se koriste na primjer prilikom kreiranja *navigation drawera*. Nadalje, u podmapi *layout* nalaze se svi izgledi pojedinih zaslona aplikacije. Podmapa *values* služi za pohranu vrijednosti *stringova*, boja i tema. Moguće je dodati još podmapa, na primjer za font ili *navmenu* koji se koristi za prikaz stavki na navigacijskom podizborniku u aplikaciji (Sl.3.3.) .



Slika 3.3. Isječak mape *res* iz Android Studia

3.4.Firebase – Realtime Database, Authentication

Firebase Realtime Database je *NoSQL* baza podataka koja omogućuje trenutnu sinkronizaciju podataka u stvarnom vremenu. Sinkronizacija i pohranjivanje podataka odvija se između korisnika i uređaja gdje upravo *Realtime Database* predstavlja JSON (engl. *JavaScript Object Notation*) objekt koji uvelike doprinosi upravljanju podatcima u stvarnom vremenu svojom jednostavnošću obrade i prijenosa podataka između korisnika i klijenta. Osim toga, JSON omogućuje programerima brzu interpretaciju podataka čime upravljanje podatcima u stvarnom vremenu postaje kvalitetnije. *Realtime Database* pruža korisnicima *online* i *offline* način rada. Ukoliko korisnik u određenom trenutku izgubi vezu s internetom, podatci koje je unio svejedno se spremaju jer se za potrebe spremanja koristi lokalna predmemorija. Nakon ponovnog pristupa internetu podatci se automatski sinkroniziraju s bazom. Korisnicima je dopušteno korištenje sigurnosnih pravila kojima oni određuju tko će imati pravo pristupa bazi podataka i na koji način. Nadalje, *Realtime Database* omogućuje korisnicima pristup s bilo kojeg uređaja, neovisno bio to mobilni uređaj ili desktop uređaj. Baza podataka koja je korištena pri izradi funkcionalnosti ove Android aplikacije je *Firebase Realtime Database* i *Firebase Authentication*. S obzirom da aplikacija zahtjeva registraciju korisnika i prijavu korisnika te uz to spremanje vrijednosti mjerenja tlaka u listu potrebno je uz *Firebase Realtime Database* koristiti i *Firebase Authentication*. *Firebase Realtime Database* i *Firebase Authentication* međusobno integriraju koristeći podatke iz različitih izvora neovisno gdje se podatci fizički nalaze, osiguravajući na taj način korisniku koji se uspješno registrira i prijavi u aplikaciju jedinstven pristup svojim podatcima koje unosi u listu. Podatci se unutar ove, a i svake druge Android aplikacije koja je povezana s *Realtime Databaseom* spremaju u obliku JSON objekta. Organizacija strukture podataka ove Android aplikacije raspoređena je u čvorove i podčvorove. Čvor s jedinstvenom oznakom koja predstavlja ključ čvora, nastat će nakon što se funkcija push().key upotrijebi točnije pozove nad objektom *DatabaseReference.* Dok se s druge strane podčvor stvara dodavanjem novih mjerenja nakon što se korisnik s jedinstvenom oznakom prijavi u aplikaciju. Svako novo ažuriranje baze podataka korisnici primaju u kratkom vremenu zbog trenutne sinkronizacije podataka koja se koristi u stvarnom vremenu (Sl.3.4.) [7].



Slika 3.4. Prikaz trenutne sinkronizacije podataka u stvarnom vremenu

Izvor: https://www.geeksforgeeks.org/firebase-realtime-database-with-operations-in-android-with-examples/

3.5.Repozitorij i biblioteke

Repozitorij (engl. *repositories*) se nalazi u *settings.gradle.kts* (*Project settings*) i predstavlja mjesto s kojeg *Gradle* koristi sve potrebne biblioteke (engl. *dependencies*) koje su od velike važnosti za kreiranje projekta. Bibliotekama se znatno olakšava i ubrzava pisanje programskog koda jer upravo te biblioteke predstavljaju prethodno definirani kod koji je moguće koristiti u vlastitoj Android aplikaciji. Prednost biblioteka počiva upravo na njihovoj velikoj funkcionalnosti. Najvažnija funkcionalnost biblioteka ponovna je uporaba prethodno napisanog i testiranog programskog koda čime se znatno štedi vrijeme. Ukoliko želimo povezivanje na bazu ili želimo pronaći trenutnog korisnika u bazi, potrebno je koristiti već definirane odgovarajuće biblioteke za *Firebase*. Ako želimo koristiti pozdravni zaslon (engl. *Splash Screen*) kao što je to slučaj ove Android aplikacije, potrebno je koristiti odgovarajuću biblioteku za pozdravni zaslon. Unutar ove aplikacije korištene su brojne biblioteke. Osim biblioteka, promjene je trebalo napraviti i unutar repozitorija. Korištenje grafičkog prikaza (engl. *BarChart*) zahtijevalo je promjenu *maven* repozitorija (engl. *maven repository*), ali i dodavanje vanjske biblioteke vezane za grafički prikaz. Promjenom maven repozitorija točnije dodavanjem JitPack repozitorija programeru se prilikom pisanja programerskog koda omogućava jednostavno uključivanje GitHub repozitorija u njegov projekt (Sl.3.6.) . Sukladno promjeni repozitorija potrebno je napraviti promjene u bibliotekama. JitPack repozitorij ove Android aplikacije zahtjeva uvođenje GitHub poveznice željenog projekta (Sl.3.5.) [8].

```
implementation(libs.androidx.core.splashscreen)
implementation(libs.circleimageview)
implementation(libs.androidx.fragment.ktx)
implementation (libs.androidx.activity.ktx)
implementation(libs.mpandroidchart.vv310)
implementation(libs.firebase.auth)
implementation(libs.androidx.recyclerview.selection)
```

Slika 3.5. Isječak *BarChart* biblioteke iz *build.gradlea*

```
maven { url = uri( path: "https://jitpack.io") }
```

Slika 3.6. Dodavanje JitPack repozitorija datoteci za izgradnju

4.STRUKTURA I FUNKCIONALNOSTI APLIKACIJE

Struktura aplikacije sastoji se od svih dijelova Android Studioa koji su bili korišteni prilikom izrade ove Android aplikacije. U nastavku su detaljno opisane sve aktivnosti i svi fragmenti koji se nalaze unutar ove Android aplikacije.

Osim navođenja, definiranja i pojašnjavanja aktivnosti i fragmenata, unutar ovog poglavlja pobliže su objašnjene i važne funkcionalnosti koje omogućuju spremanje podataka na bazu.

4.1 Aktivnost pozdravnog zaslona (engl. *SplashScreenActivity*)

Prvi zaslon koji se prikaže prilikom pokretanja aplikacije je upravo pozdravni zaslon. Pozdravni zaslon predstavlja zaslon koji se prikazuje određeno vrijeme. Kako bi pozdravni zaslon uopće bilo moguće dodati u Android aplikaciju potrebno je prvo dodati biblioteku koja odgovara aktivnosti pozdravnog zaslona. Ukoliko želimo da se aktivnost dulje prikazuje na zaslonu, potrebno je unutar metode Handler.PostDelayed(), koja omogućuje odgodu izvršenja programskog koda za određeni vremenski period promijeniti vrijeme prikazivanja na primjer s 2000ms na 5000ms. Važno je napomenuti da se vrijeme prikazivanja zaslona iskazuje u milisekundama. Zaslon ove aktivnosti je dizajniran na način da se prilikom pokretanja aplikacije prikaže logo aplikacije uz naziv aplikacije (Sl.4.1.) . Aplikacija nosi naziv „*Smart BPCA*“.

Slika 4.1. Aktivnost pozdravnog zaslona (engl. *SplashScreenActivity*)

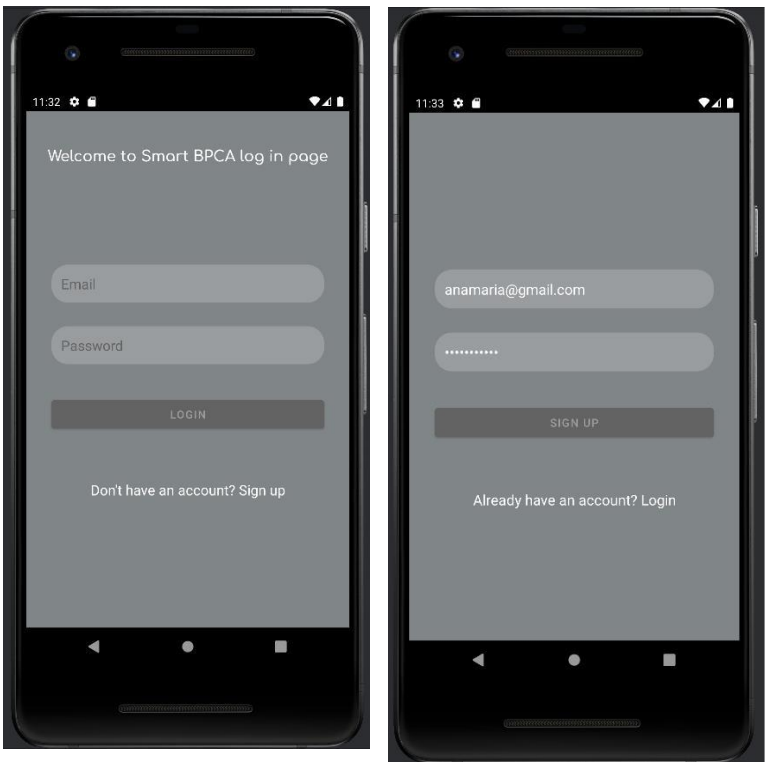
Prema programskom kodu 4.1. unutar metode odgode programskog koda, Handler.postDelayed() pojavljuje se Intent. Glavna uloga Intenta unutar Android Studioa temelji se na ostvarivanju komunikacije između različitih dijelova aplikacije. U ovom slučaju Intent pokreće iduću aktivnost, aktivnost prijave (engl. *LogInActivity*) , nakon što istekne 10000ms.

```
Handler().postDelayed({
    val intent = Intent( packageContext: this, LogInActivity::class.java)
    startActivity(intent)
    finish()
}, delayMillis: 10000)
```

Programski kod 4.1. Pokretanje aktivnosti pozdravnog zaslona

4.2. Aktivnost registracije (engl. *SignUpActivity*) i Aktivnost prijave (engl. *LogInActivity*)

Nakon što aktivnost pozdravnog zaslona (engl. *SplashScreenActivity*) istekne, zaslon koji se prvi prikazuje je upravo zaslon prijave koji predstavlja aktivnost prijave (engl. *LogInActivity*). Na zaslonu prijave nalaze se dva polja za unos podataka, točnije za unos adrese elektroničke pošte i zaporke. Oba polja unutar XML *layouta* definirana su kao *EditText* s odgovarajućom *id* oznakom kako bi ih bilo lakše pozvati u glavnom dijelu aktivnosti prilikom kreiranja funkcije za prijavu. Uz dva polja unosa podataka nalazi se i gumb koji uz funkcionalnost prijelaza na novu aktivnost sadrži također i funkcionalnost pripreme podataka, funkcionalnost slanja podataka te funkcionalnost provjere odgovora poslanog s baze podataka. Osim toga na zaslonu se nalazi i mogućnost kreiranja novog računa ukoliko korisnik već ne posjeduje račun (Sl.4.2.) . Upravo ta mogućnost kreiranja novog računa, korisnika vodi pomoću Intenta na aktivnost registracije (engl. *SignUpActivity*). Prilikom registriranja, novi korisnik će unijeti svoju adresu elektroničke pošte i odgovarajuću zaporku pomoću kojih će u budućnosti obavljati prijavu u aplikaciju. Nakon uspješne registracije, korisnik će u bazi podataka biti spremljen pod jedinstvenom oznakom. Također, kada se korisnik prvi put prijavljuje u aplikaciju nakon uspješne registracije, prvo se u bazi podataka provjerava postoji li korisnik koji je registriran pod tom jedinstvenom oznakom. Ukoliko postoji, prijava je uspješna te ga se preusmjerava na zaslon dobrodošlice.

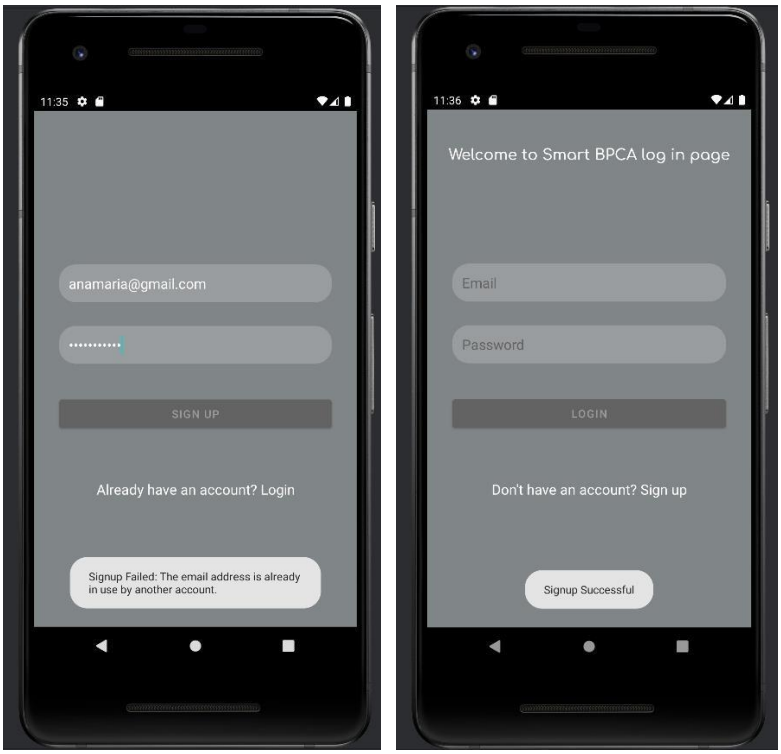


Slika 4.2 Prikaz izgleda korisničkog sučelja aktivnosti prijave (engl. *LogInActivity*) i aktivnosti registracije (engl. *SignUpActivity*)

```
val authUser = firebaseAuth.currentUser
if (authUser != null) {
    val id = authUser.uid
    val user = UserInformation(id, email, password)
    databaseReference.child(id).setValue(user)
        .addOnCompleteListener { task ->
            if (task.isSuccessful) {
                Toast.makeText( context: this@SignUpActivity,  text: "Signup Successful", Toast.LENGTH_SHORT).show()
                startActivity(
                    Intent( packageContext: this@SignUpActivity, LogInActivity::class.java))
                finish()
            } else {
                Toast.makeText( context: this@SignUpActivity,  text: "User already exists", Toast.LENGTH_SHORT).show()
            }
        }

}
```

Programski kod 4.2. Isječak koda iz funkcije registracije unutar aktivnosti registracije (engl. *SignUpActivity*)

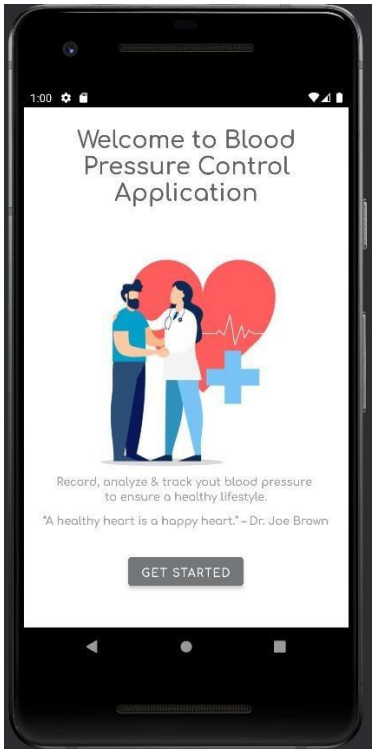
Prema programskom kodu 4.2. potrebno je prvo dohvatiti trenutno registriranog korisnika pomoću svojstva instance klase *Firebase Authentication*, svojstva currentUser. Ako authUser nije null odnosno ako je dohvaćen trenutno registrirani korisnik sukladno tome authUser.uid dohvaća jedinstvenu oznaku tog trenutnog korisnika. Uid predstavlja jedinstvenu oznaku korisnika koja mu se dodjeljuje nakon uspješne registracije u bazi. Korisnik će u bazu biti spremljen kao objekt tipa UserInformation koji sadrži jedinstvenu oznaku, adresu elektroničke pošte i zaporku. Objekt user sprema se pod čvor imena id te se nakon provjere ispravnosti registracije pomoću Intenta prelazi na aktivnost prijave (engl. *LogInActivity*) s naglaskom da je osoba uspješno registrirana.
Pokuša li se korisnik registrirati adresom elektroničke pošte koja već postoji u bazi pod jedinstvenom oznakom, na zaslonu se prikazuje obavijest kao na slici 4.3. (lijeva slika). Ali, ako se korisnik registrira s odgovarajućom adresom elektroničke pošte i zaporkom koje nisu prethodno registrirane u bazi, na zaslonu se prikazuje obavijest kao na slici 4.3. (desna slika).



Slika 4.3. Neuspješna i uspješna registracija

4.3. Aktivnost dobrodošlice (engl. *WelcomeActivity*)

Nakon uspješne prijave korisniku se otvara zaslon dobrodošlice (engl. *WelcomeActivity*). Dizajn zaslona osmišljen je tako da korisnik nakon prijave bude upućen da će mu unutar aplikacije biti omogućeno praćenje, zapis i analiza njegovog krvnog tlaka. Jedina funkcionalnost koja se nalazi na ovom zaslonu je gumb Get started. Taj gumb korisnika vodi na iduću aktivnost pomoću Intenta.



Slika 4.4. Prikaz aktivnosti dobrodošlice (engl. *WelcomeActivity*)

4.4. Aktivnost zamjene fragmenata (engl. *ChangeFragmentActivity*)

Iduća aktivnost točnije aktivnost zamjene fragmenata (engl. *ChangeFragmentActivity*) unutar aplikacije sadrži *Bottom Navigation Drawer* i *Frame Layout* pomoću kojeg je moguće na osnovi ikona koje su definirane za svaki fragment unutar *Bottom Navigation Drawera*, mijenjati fragmente po potrebi. Potrebno je koristiti metodu *setOnItemSelectedListener{}* koja postavlja slušatelja (engl. *Listener*) za određene stavke u *Bottom Navigation Draweru*. Osim toga potrebno je unutar te metode postaviti programski kod (Programski kod 4.3.) koji će omogućiti svakom itemIdu da trenutni fragment zamjeni novim fragmentom. Prilikom zamjene fragmenata potrebno je koristiti *supportFragmentManager* (Programski kod 4.4.) pomoću kojeg na poziv *beginTransaction()* započinje zamjena fragmenata. Uz to, *supportFragmentManager* predstavlja

instancu *FragmentManager*a koja se koristi prilikom rada s fragmentima unutar Android aplikacije.

```
binding.bottomNavigationView.setOnItemSelectedListener {


    when(it.itemId){
        R.id.icNews -> replaceFragment(NewsFragment())
        R.id.icAnalytics -> replaceFragment(AnalyticsFragment())
        R.id.icMeasurement -> replaceFragment(NewMeasurementFragment())
        R.id.icProfile -> replaceFragment(ProfileFragment())
        R.id.icSettings -> replaceFragment(SettingsFragment())

        else ->{

        }
    }
    true


}
```

Programski kod 4.3. Zamjena fragmenata prema itemIdu unutar *Bottom Navigation Drawera* pomoću funkcije *replaceFragment()*

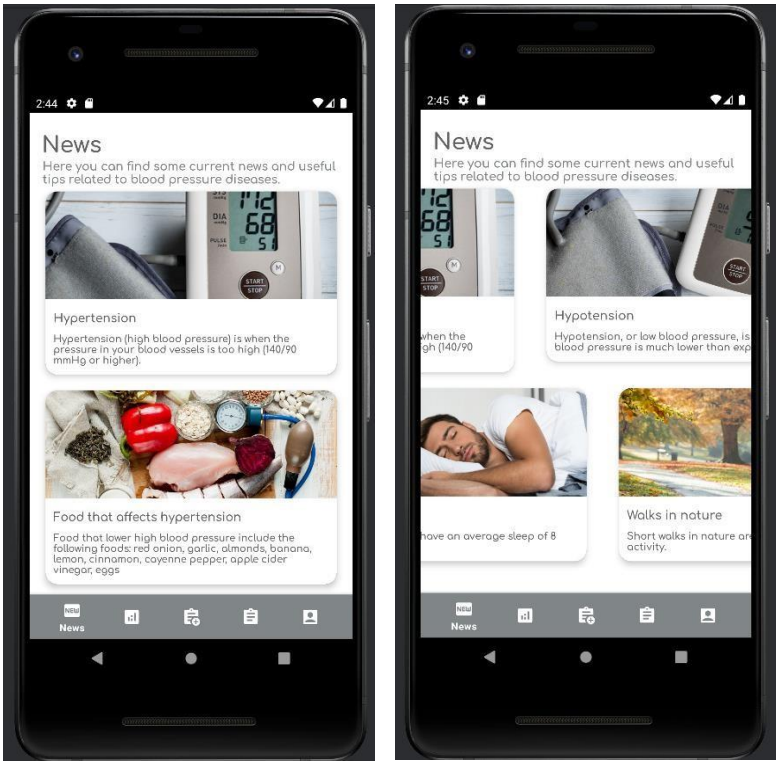
```
private fun replaceFragment(fragment: Fragment){

    val fragmentManager = supportFragmentManager
    val fragmentTransaction = fragmentManager.beginTransaction()
    fragmentTransaction.replace(R.id.fragment_container, fragment)
    fragmentTransaction.commit()
}
```

Programski kod 4.4. Funkcija *replaceFragment()* koja mijenja trenutni fragment s novim fragmentom

4.5. Fragment s novostima (engl. *NewsFragment*)

Fragment s novostima u aplikaciji služi kao obavještajni zaslon. Korisnik nakon prijave i zaslona dobrodošlice pomoću Intenta dolazi na fragment s novostima. Novosti su unutar fragmenta podijeljene u dvije skupine. Obje skupine su informativnog tipa. Prva skupina novosti upoznaje korisnika s pojmovima hipertenzije i hipotenzije. Druga skupina korisnika savjetuje o tome kako bi trebao prilagoditi prehranu ukoliko boluje od hipertenzije ili hipotenzije. Osim prehrane, korisnika se savjetuje i o tjelesnim aktivnostima koje je poželjno uvesti u svakodnevnu rutinu života ukoliko je korisnik spreman poboljšati svoje zdravlje. Novosti se nalaze unutar dva *Horizontal Recycler Viewa* koji omogućuju prikaz sadržaja horizontalno s lijeva na desno. Svaki

Recycler View mora imati i adapter koji služi za upravljanje podatcima, odnosno bavi se njihovim prikazom i položajem. Budući da aplikacija koristi dva adaptera, svaki od njih će se nalaziti u zasebnoj klasi. Svaki adapter klase od dva adaptera koja su korištena pri izradi ove Android aplikacije, nasljeđuje klasu *RecyclerView.Adapter* te samim time automatski implementira prethodno definirane metode. Tri su glavne metode, a to su: *onCreateViewHolder()* metoda koja kreira novi *ViewHolder* objekt , *onBindViewHolder()* metoda koja unutar *Recycler Viewa* povezuje podatke trenutne stavke, *getItemCount()* metoda koja vraća konačan broj stavki.



Slika 4.5. Dva *Horizontal Recycler Viewa*

Programski kod 4.5. predstavlja glavni dio fragmenta novosti (engl. *NewsFragment*). Unutar metode *onViewCreated()* prvo se inicijaliziraju *recyclerViewi*. Pristup istima ostvaruje se preko jedinstvene oznake id koja je prethodno definirana unutar XML *layouta*. Osim inicijalizacije *recyclerViewa*, potrebno je postaviti upravitelja rasporeda (engl. *LayoutManager*) točnije *LinearLayoutManager* koji će ukupan sadržaj usmjeriti horizontalno. Nadalje, uz inicijalizaciju *recyclerViewa* i postavljanje upravitelja rasporeda također je potrebno postaviti i adapter koji će upravljati podatcima. Funkcije *dataInitialize()* i *dataList()* unutar sebe sadržavaju sav potreban sadržaj koji će biti prikazan na zaslonu. Pod sadržajem se misli na slike (liste slika), naslove (liste naslova), podnaslove ili opise naslova. Obje funkcije se pozivaju unutar metode onViewCreated.

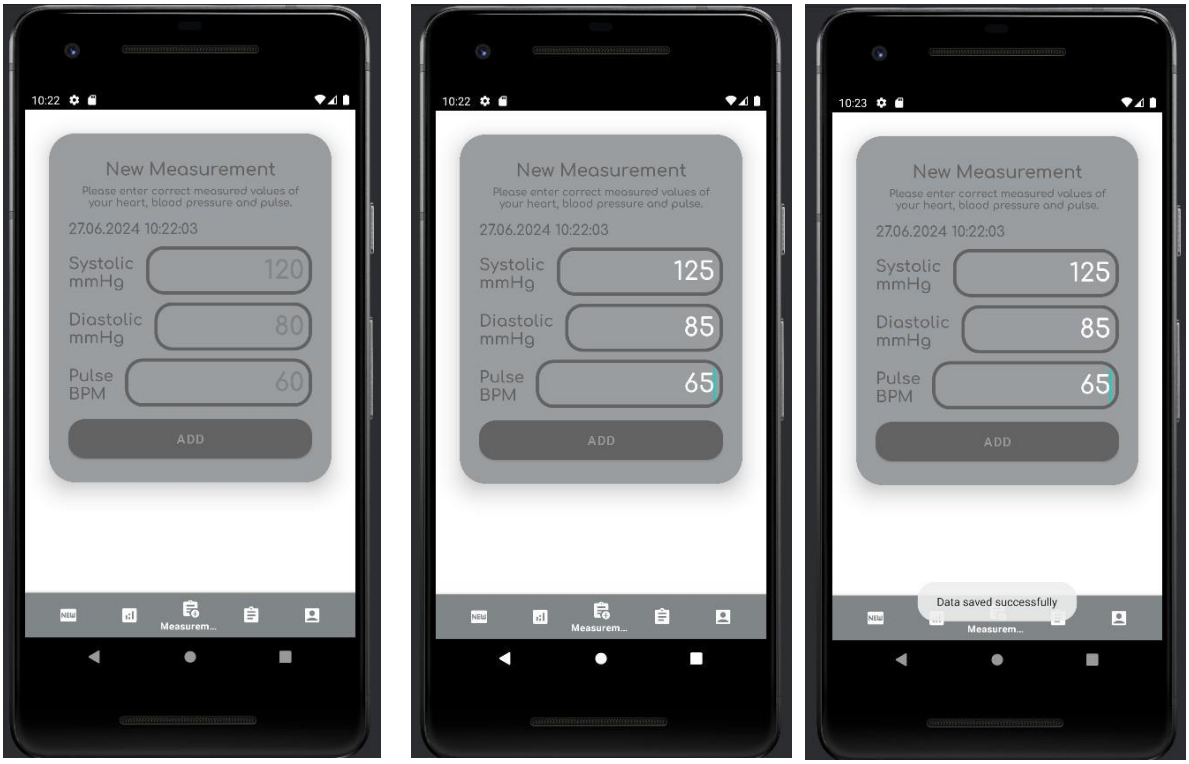
```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    dataInitialize()
    dataList()

    recyclerView = view.findViewById(R.id.recyclerView)
    recyclerView1 = view.findViewById(R.id.recyclerView1)

    recyclerView.layoutManager = LinearLayoutManager(requireContext(), LinearLayoutManager.HORIZONTAL,  reverseLayout: false)
    recyclerView1.layoutManager = LinearLayoutManager(requireContext(), LinearLayoutManager.HORIZONTAL,  reverseLayout: false)

    adapter = HorizontalRecyclerView(diseaseArrayList)
    adapter1 = HorizontalRecyclerView1(curesArrayList)

    recyclerView.adapter = adapter
    recyclerView1.adapter = adapter1

}
```

Programski kod 4.5. Inicijalizacija *recyclerViewa*, postavljanje *LayoutManagera* i adaptera

4.6. Fragment za unos novih mjerenja (engl. *NewMeasurementFragment*)

Nakon što se korisnik uspješno prijavi u aplikaciju, dobiva jedinstven pristup aplikaciji budući da je prilikom registracije dobio jedinstvenu oznaku prema kojoj se prepoznaje u bazi. S obzirom da je aplikacija namijenjena kontroliranju tlaka i pulsa uz to korisnik može unutar fragmenta za unos novih mjerenja unijeti i iznose svojih tlakova. Pod pojmom tlakovi misli se na sistolički tlak, dijastolički tlak i puls. Prilikom pokretanja fragmenta za unos novih mjerenja na zaslonu se prikazuju polja koja korisniku omogućuju unos mjerenja. Povrh polja nalazi se vrijeme, koje označava trenutno vrijeme kada korisnik unosi mjerenja. Pomoću *SimpleDateFormat* određen je format datuma. Unutar polja s desne strane nalaze se *hint* brojevi koji simboliziraju 'normalan tjelesni tlak' čija je vrijednost 120/80 mmHg i puls pri vrijednosti od 60 otkucaja u minuti. Osim polja za unos podataka, na zaslonu se nalazi i gumb Dodaj. Pritiskom na gumb, podatci koje je korisnik unio spremaju se pod jedinstveni ključ u *Realtime Database* (Sl.4.6.) . Korisnik je dakle prilikom registracije dobio svoj jedinstveni ključ ili jedinstvenu oznaku, a sad unošenjem novih mjerenja svako mjerenje dobiva svoj jedinstveni ključ kako bi s njime bilo lakše upravljati. Tako spremljeni podatci prikazuju se u listi na idućem fragmentu. Podatci kojima korisnik upravlja unutar ovog fragmenta predstavljeni su kao model. Model je u slučaju ovog fragmenta, klasa koja u sebi sadrži jedinstvene oznake svih tlakova i pulsa te jedinstvenu oznaku mjerenja. Ako se uzme u obzir da se ovi podatci unose u bazu unutar jednog fragmenta, prikazuju u listi unutar drugog fragmenta te se također grafički prikazuju unutar trećeg fragmenta. Iz toga proizlazi da sva tri fragmenta predstavljaju pogled (engl. *View*). Pogled zapravo predstavlja sve radnje nad podatcima. Kad se govori o radnjama nad podatcima i ako uzmemo u obzir ovu aplikaciju onda su pogledi ove Android aplikacije odgovorni za unos podataka, prikaz podataka u listi i prikaz podataka na grafikonima. Klikom na gumb Dodaj, pogled reagira na taj unos i prosljeđuje ga kontroleru. Naposljetku, kontroler (engl. *Controller*) je odgovoran za kontrolu

međusobnog odnosa unesenih podataka i radnjama koje se odvijaju nad tim podatcima. Od modela uzima sve potrebne podatke kako bi ih ažurirao, dok od pogleda koji je reagirao na korisnikov klik gumba dobiva zahtjev prema kojem treba odrediti koji podatci trebaju biti obrađeni. Model, pogled i kontroler predstavljaju strukturu *MVC* (engl. *Model-View-Controller*) arhitekture koja znatno pomaže u podjeli obveza i odgovornosti unutar Android aplikacije.



Slika 4.6. Prikaz unosa podataka unutar *NewMeasurementFragmenta*

```
saveIt.setOnClickListener {
    val user = getCurrentUser()
    if(user != null ){

        val id = user.uid
        val systolicPressure = systolicPressureInfo.text.toString().toIntOrNull()?. run {this} ?: 0
        val diastolicPressure = diastolicPressureInfo.text.toString().toIntOrNull()?. run {this} ?: 0
        val pulsNumb = pulsInfo.text.toString().toIntOrNull()?. run {this} ?: 0
        val time = time.text.toString()

        val newEntry = DataPressure(id, time, systolicPressure, diastolicPressure, pulsNumb)
        saveEnteredDataToFirebase(newEntry)
    }
    else{
        Toast.makeText(requireContext(), text: "Ne pronalazimo korisnika", Toast.LENGTH_SHORT).show()
    }
}
```

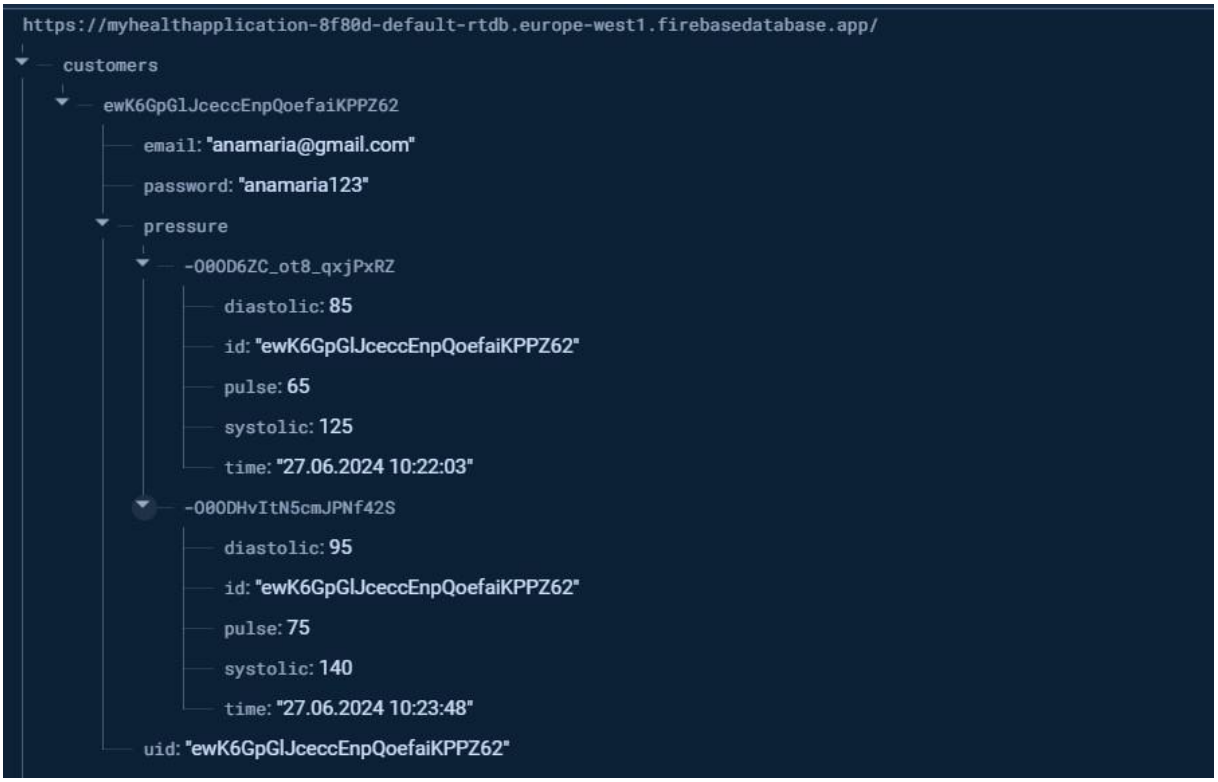
Programski kod 4.6. Metoda dodavanja novog mjerenja u bazu podataka

Prema programskom kodu 4.6. prilikom dodavanja novog mjerenja, prvo je potrebno dohvatiti trenutno prijavljenog korisnika i provjeriti je li različit od null. Ukoliko je korisnik uspješno pronađen kreira se novi objekt tipa *DataPressure* koji sadrži jedinstvenu oznaku trenutno prijavljenog korisnika, ali isto tako i jedinstvenu oznaku mjerenja. Osim toga sadrži vrijednosti vremena, sistoličkog tlaka, dijastoličkog tlaka i pulsa koje će bit spremljene u bazu podataka pod jedinstvenom oznakom koja se dodjeljuje isključivo tome mjerenju.

```
private fun saveEnteredDataToFirebase(pressure: DataPressure){
    val currentUser = getCurrentUser()
    val idOfUser = currentUser?.uid ?: return
    val keyId = database.child( pathString: "customers").child(idOfUser).child( pathString: "pressure").push().key
    if(keyId == null){
        Toast.makeText(requireContext(), text: "Something is wrong with saving data", Toast.LENGTH_SHORT).show()
        return
    }
    database.child( pathString: "customers").child(idOfUser).child( pathString: "pressure").child(keyId).setValue(pressure)
        .addOnCompleteListener { task ->
                if (task.isSuccessful) {
                    Toast.makeText(requireContext(), text: "Data saved successfully", Toast.LENGTH_SHORT).show()
                } else {
                    Toast.makeText(requireContext(), text: "Something is wrong with saving data", Toast.LENGTH_SHORT).show()
                }
        }
}
```

Programski kod 4.7. Funkcija spremanja mjerenja pod novostvorenim ključem

Prema programskom kodu 4.7. funkcija *saveEnteredDataToFirebase()* sprema nova mjerenja u *Realtime Database*. Prilikom spremanja podataka u bazu potrebno je provjeriti nalazi li se registrirani korisnik u bazi. Ukoliko je korisnik pronađen, postavlja se *database* referenca na glavni čvor baze odnosno čvor *customers* te se preko *push()* metode kreira nova jedinstvena oznaka unutar podčvora *pressure*. Nadalje, jedinstvena oznaka sprema se pod varijablu *keyId*. U konačnici, postoji korisnik koji je spremljen pod *idOfUser* oznakom unutar čvora *customers*. Korisniku je omogućeno spremanje mjerenja u bazu podataka gdje svako mjerenje dobiva svoju jedinstvenu oznaku *keyId*. Mjerenja će biti spremljena i prepoznata u bazi pomoću jedinstvene oznake unutar podčvor *pressure* (Sl.4.7.) .



Slika 4.7. Prikaz mjerenja spremljenih u bazi podataka unutar podčvora *pressure*

4.7. Fragment izlistanja mjerenja (engl. *ListFragment*)

S obzirom da je korisnik uspješno unio podatke, ti isti podatci sad će biti prikazani unutar fragmenta izlistanja. Kako bi podatci bili prikazani u vertikalnom rasporedu s naznakom da ih je moguće dizajnirati na vlastiti način za potrebe dizajna korisničkog sučelja koristi se *RecyclerView*. Budući da se podatci nakon unošenja, automatski spremaju i prikazuju na zaslonu fragmenta, za njihov prikaz i raspored potrebno je definirati adapter i upravitelja rasporeda korisničkog sučelja (engl. *Layout Manager*). Prethodno je spomenuto da adapter klase nasljeđuje *RecyclerView.adapter* te unutar klase implementira metode koje su odgovorne za prikaz stavki, povezivanje stavki i vraćanje ukupnog broja stavki. Da bi podatci bili prikazani unutar fragmenta izlistanja nakon novog dodavanja mjerenja, potrebno je koristiti *setFragmentResultListener* kojemu je zadatak postaviti slušatelja na fragment koji prima unesena mjerenja. Pri završetku obrade primljenih mjerenja, adapter određuje na koji će način *RecyclerView* biti ažuriran. Ideja ovog fragmenta je da on učitava mjerenja iz baze podataka i prikazuje ih u *RecyclerViewu*. Dakle, potrebno je pomoću svojstva *Firebase Authentication*a dohvatiti trenutnog korisnika, provjeriti postojanost njegove registracije pomoću svojstva *uid* i tek nakon toga pristupiti mjerenjima koja se nalaze unutar njegove jedinstvene oznake. Prema programskom kodu 4.8.

funkcija *onDataChange* prije dodavanja novih mjerenja mora „očistiti“ listu kako bi se osiguralo da su unesena mjerenja jedina koja se nalaze unutar liste. Nakon toga prelazi se na provjeru svakog podčvora koji se nalazi unutar čvora *customers*. Ako se ustanovi da je novi objekt *DataPressure* različit od null, dodaje ga se u listu *pressure*. Nakon uspješnog dodavanja objekta u listu, šalje se obavijest adapteru da je došlo do promjene. Adapter će pristigle promjene proslijediti *RecyclerViewu* koji će se potom ažurirati i prikazati podatke na zaslonu.

```
override fun onDataChange(snapshot: DataSnapshot) {
    pressure.clear()
    for (dataSnapshot in snapshot.children) {
        val newAdd = dataSnapshot.getValue(DataPressure::class.java)
        if (newAdd != null) {
            pressure.add(newAdd)
        }
    }
    pressureAdapter.notifyDataSetChanged()

}
```

Programski kod 4.8. Isječak iz metode dohvaćanja mjerenja iz baze

```
private fun getTime(): String{
    val date = SimpleDateFormat( pattern: "dd.MM.yyyy HH:mm:ss", Locale.getDefault())
    return date.format(Date())
}
```

Programski kod 4.9. Metoda koja omogućuje prikaz vremena (engl. *getTime*())

Prema programskom kodu 4.9. korisnik dobiva mogućnost da svakim novim unošenjem mjerenja, vrijeme koje je prikazano na zaslonu tad kao trenutno vrijeme, bude spremljeno u listi uz mjerenja koja su u tom trenutku unesena. Varijabla *date* dobiva vrijednost formatiranog zapisa vremena zbog poziva *SimpleDateFormat*.

Slika 4.8. Prikaz fragmenta izlistanja nakon unošenja mjerenja

Prema slici 4.8. fragment izlistanja korisniku omogućuje zorni prikaz svih mjerenja koja su unesena u bazu podataka. Osim unesenih mjerenja tlaka, korisnik ima pregled i o vremenu unošenja mjerenja. Samim time, korisniku je omogućeno kontinuirano praćenje promjena.

4.8. Fragment grafičkog prikaza (engl. *AnalyticsFragment*)

Osim što korisnik aplikacije ima pristup listi mjerenja koja mu olakšava snalaženje u promjenama tlaka. Korisniku je također omogućen i grafički prikaz unesenih mjerenja. Grafikoni predstavljaju slikoviti prikaz promjene tlaka. Unutar ovog fragmenta, nalaze se tri grafikona. Svaki grafikon predstavlja vrijednosti pojedinog tlaka ili pulsa. Grafikoni su dizajnirani tako da svaki grafikon pokazuje svoje vrijednosti, na primjer : grafikon sistoličkog tlaka prikazuje samo vrijednosti sistoličkog tlaka, grafikon dijastoličkog tlaka prikazuje vrijednosti samo dijastoličkog tlaka te na kraju grafikon pulsa koji prikazuje vrijednosti samo pulsa. Sva tri grafa su vertikalnog stupčastog oblika. Uz to, korisnik ukoliko želi, kad otvori fragment s grafikonima može svaku vrijednost posebno očitati na način da jagodicom prsta „pritisne“ određeni stupac grafikona nakon čega će se pojaviti oblačić iznad stupca unutar kojeg piše vrijednost tlaka. Budući da se vrijednosti dohvaćaju s baze, ponavlja se postupak dodavanja mjerenja u grafikon tako što se prvo provjerava korisnikova jedinstvena oznaka, točnije provjerava se je li korisnik prijavljen i registriran u bazi. Ukoliko je uspješno registriran, pomoću jedinstvene oznake moguće je pristupiti vrijednostima

podataka tog korisnika u bazi. Ponavlja se postupak dohvaćanja podataka iz baze kao kod *ListFragmenta*. Prema programskom kodu 4.10. unutar metode *onDataChange* prije svega prazni se lista, kako bi se osiguralo da su nova mjerenja jedina koja su u listi. Nadalje, iterira se za svaki podčvor unutar *customers* i na svakom podčvoru se primjenjuje deserializacija ukoliko je to moguće. Odnosno dolazi do stvaranja objekta klase *DataPressure* . Ako je objekt različit od *null*, dodaje se u listu. Nakon dodavanja objekata u listu, poziva se funkcija *showAllChartsTogether*() unutar koje se nalaze funkcije grafičkog prikaza za svaki tlak zasebno.

```
private fun loadDataOnChart(){
    val thisUser = firebaseAuth.currentUser
    if(thisUser != null){
        val userId = thisUser.uid
        database.child( pathString: "customers").child(userId).child( pathString: "pressure")
            .addListenerForSingleValueEvent(object : ValueEventListener{
                override fun onDataChange(snapshot: DataSnapshot) {
                    pressure.clear()
                    for(dataSnapshot in snapshot.children){
                        val newMeasure = dataSnapshot.getValue(DataPressure::class.java)
                        if (newMeasure != null){
                            pressure.add(newMeasure)
                        }
                    }
                    Log.d( tag: "AllChartFragment",  msg: "Data loaded: $pressure")
                    showAllChartsTogether()
                }
                override fun onCancelled(error: DatabaseError) {
                    Toast.makeText(requireContext(),  text: "Failed to load data", Toast.LENGTH_SHORT).show()
                }
            })
    }
}
```

Programski kod 4.10. Isječak koda za dohvaćanje podataka korisnika iz baze podataka

Funkcija *showAllChartsTogether*() sadrži tri druge funkcije (Sl.4.9.) , kod je za sve tri funkcije jednak te je zato u nastavku objašnjen grafički prikaz samo jednog tlaka. Prema programskom kodu 4.11. funkcija *barChartSystolic*() kreira praznu listu imena *listOfSystolic* čiji su podatci ujedno i *BarEntry* objekti. Nakon toga, prolazi se kroz sve indekse unutar liste *pressure* nakon čega se njihova pozicija dodjeljuje varijabli new. Drugi dio koda stvara novi objekt koji se dodaje u listu i predstavlja jedan stupac unutar grafikona, *i.toFloat*() je ujedno i X koordinata unutar grafikona dok je *new.systolic.toFloat*() Y koordinata grafikona koja prikazuje vrijednosti. Osim što su objekti kreirani, potrebno je cijeloj listi dodijeliti naziv ili labelu (engl. *label*) kako bi korisniku bilo jasnije koji graf predstavlja koje vrijednosti. Uz naziv, dodane su i boje kao i markeri. Upravo zbog markera koji su dio *CustomLabelMarker*a korisnik može u bilo kojem trenutku pritiskom jagodice prsta na određeni objekt *BarEntrya* očitati vrijednost tlaka.

```
private fun barChartSystolic() {
    val listOfSystolic = ArrayList<BarEntry>()
    for (i in pressure.indices) {
        val new = pressure[i]
        Log.d( tag: "SystolicChartFragment",  msg: "Adding entry: ${new.systolic}")
        listOfSystolic.add(
            BarEntry(
                i.toFloat(),
                new.systolic.toFloat()
            )
        )
    }
    val barDataSet = BarDataSet(listOfSystolic,  label: "Systolic Data")
    barDataSet.color = Color.parseColor( colorString: "#999DA0")

    val data = BarData(barDataSet)
    systolicChart.data = data
    val markerView = CustomLabelMarker(requireContext(), R.layout.marker_view)
    systolicChart.marker = markerView
    systolicChart.invalidate()
}
```

Programski kod 4.11. Funkcija grafičkog prikaza sistoličkog tlaka

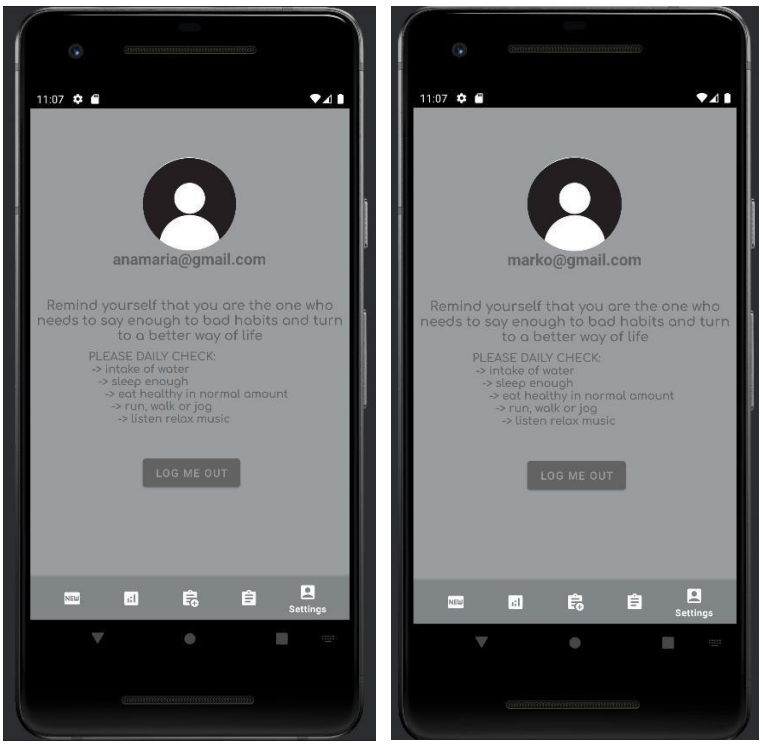
Slika 4.9. Funkcija *showAllChartsTogether()* na zaslonu

4.9. Fragment odjave s profila (engl. *SettingsFragment*)

Fragment odjave, kreiran je iz razloga što se korisniku želi omogućiti sigurnost pohrane podataka. Nakon prijave korisnik ostaje pribilježen u bazi kao i u samoj aplikaciji. Dok se ne odjavi, njegovi podatci su dostupni svima koji imaju pristup njegovom mobitelu. Kako bi se spriječila zlouporaba podataka ili stvaranje određenih problema kojima se korisnika dovodi u neugodnu situaciju, aplikacija na fragmentu odjave nudi mogućnost gumba odjave (engl. Log out). Prema programskom kodu 4.12. pritiskom na gumb, korisnika se prebacuje na *LogInActivity* i automatski ga se odjavljuje iz aplikacije. Postupak odjave moguć je korištenjem metode *Firebase Authentication* objekta, a to je metoda *signOut*() . Preusmjeravanje korisnika na *LogInActivity* ostvaruje se Intentom . Osim odjave, na zaslonu je omogućen prikaz adrese elektroničke pošte (Sl.4.10.) svakog korisnika koji je trenutno prijavljen u aplikaciji. Potrebno je proći kroz cijelu bazu registriranih korisnika i ukoliko određeni korisnik postoji, tada se on pretvara ukoliko je to moguće, u objekt *UserInformation* klase te mu se tekst adrese elektroničke pošte sprema kao atribut.

```
private fun logOut(){
    firebaseAuth.signOut()
    val intent = Intent(requireContext(), LogInActivity::class.java)
    startActivity(intent)
    requireActivity().finish()
}
```

Programski kod 4.12. Funkcija odjave (engl. *logOut*())



Slika 4.10. Prikaz spremanja adrese elektroničke pošte trenutno prijavljenog korisnika

5.ZAKLJUČAK

Danas je pristup rješavanju problema ovakve tematike vrlo otvoren. Korisnicima su dostupne razne verzije sličnih ili podjednako korisnih aplikacija zdravstvene namjene. Bez obzira na dostupnost, među ljudima i dalje postoji određen udio nesigurnosti i nepouzdanja vezanih uz aplikacije kad je u pitanju kontrola zdravlja i promjene načina života. Cilj ovog završnog rada bio je upravo otkloniti te nesigurnosti kod korisnika na način da im se pruži kvalitetno, jasno i jednostavno korisničko sučelje. Uz mogućnost prijave i registracije korisnici ostaju zabilježeni u bazi podataka, čime im se omogućuje personalizirani pristup. Personaliziranim pristupom samo oni imaju pristup aplikaciji što znači da nitko drugi ne može upravljati njihovim podatcima niti ih mijenjati. Promjene je moguće pratiti redovitim unošenjem mjerenja. Svako mjerenje ostaje prikazano na zaslonu aplikacije u obliku liste i grafikona uz točno određen datum i vrijeme kad je provedeno. Kako korisnici ne bi morali napuštati aplikaciju ukoliko požele istražiti što znači imati tlak veći od normalnog ili obrnuto, kreiran je zaslon koji im omogućuje pregled svih bitnih informacija i savjeta vezanih uz krvni tlak i stanja u kojima se tijelo nalazi ukoliko se prijeđu dozvoljene granice.

Tehnologije korištene za izradu ovog završnog rada su *Firebase Console*, točnije *Firebase Realtime Database* za pohranu podataka mjerenja u bazu i *Firebase Authentication* za spremanje podataka korisnika u bazu. Osim toga, korišteni su programski jezik *Kotlin* za pisanje programskog koda i Android Studio za implementaciju koda kao i pokretanje virtualnog uređaja na kojemu je jasno vidljiv dizajn korisničkog sučelja. Aplikacija je namijenjena isključivo u svrhe praćenja i kontrole mjerenja tlaka.

LITERATURA

[1] Google Play : https://play.google.com/store/apps/details?id=heartratemonitor.heartrate.pulse.pulseapp&hl=en, pristup: lipanj 2024.

[2] Google Play : https://play.google.com/store/apps/details?id=ai.mxlabs.heartmonitor&hl=en , pristup: lipanj 2024.

[3] Google Play : https://play.google.com/store/apps/details?id=com.aadhk.lite.bptracker&hl=en, pristup: lipanj 2024.

[4] Google Play : https://play.google.com/store/apps/details?id=com.mybp.app&hl=en, pristup: lipanj 2024.

[5] GeekForGeeks (Android Studio) : https://www.geeksforgeeks.org/overview-of-android-studio/, pristup: lipanj 2024.

[6] GeeksForGeeks (Kotlin) : https://www.geeksforgeeks.org/kotlin-programming-language/, pristup: lipanj 2024.

[7] Firebase Console : https://firebase.google.com/ , pristup: lipanj 2024.

[8] Developers (repositories) : https://developer.android.com/build/remote-repositories, pristup: lipanj 2024.

[9] Ivan Malčić, Medicinska naklada d.o.o. : „Plućna Hipertenzija“, pristup: lipanj 2024.

[10] Dawn Griffiths, David Griffiths : „Um caruje Kotlin“, pristup: lipanj 2024.

[11] Dr. Boon Lim : „Zdravo Srce: Kako zaštititi i sačuvati najvažniji organ u tijelu“, pristup: lipanj 2024.

[12] Rick Boyer : „Android 9 Development Cookbook“, pristup: lipanj 2024.

[13] Raul Portales : „Mastering Android Game Development“, pristup: lipanj 2024.

[14] Sheldon G. Sheps : „Mayo Clinic o visokom krvnom tlaku“, pristup: lipanj 2024.

SAŽETAK

Ovaj završni rad usmjeren je se na rješavanje problema neinformiranosti društva o posljedicama koje nosi visoki krvni tlak. Kao rješenje tog problema, kreirana je ova Android aplikacija. Korisnicima omogućuje praćenje promjene tlaka i pulsa u svako doba dana. Također, pri samom ulasku u aplikaciju korisnici se upoznaju s osnovnom teorijom vezanom za pojam 'hipertenzije' i 'hipotenzije' na način da im se pružaju savjeti i informacije što činiti ukoliko primijete nepravilnosti vezane uz tlak.

Aplikacija je kreirana uz pomoć aktivnosti i fragmenata gdje svaka aktivnost ima određenu funkcionalnost kao i fragmenti. Aktivnosti prijave i registracije kao i fragment unosa novih mjerenja zahtijevali su povezivanje na bazu podataka, *Firebase Realtime Database* i *Firebase Authentication*. Jezik kojim je pisan programski kod je *Kotlin*.

Ključne riječi: aplikacija, Firebase, hipertenzija, hipotenzija, Kotlin, tlak

ABSTRACT

Android application for monitoring changes in blood pressure samples obtained by wearing a holter 24 hours

This paper focuses on solving the problem of society's lack of awarness about the consequences that high blood pressure carries. As a solution to that problem, this Android application has been created. Application allows users to monitor changes of their blood pressure and heart rate at any time of the day. Also, upon entering the application, users are introduced to the basic theory related to the term 'hypertension' and 'hypotension' by being offered with advice and information on what to do if they notice irregularities related to pressure changes.

The application was created by combining activities and fragments where both activities and fragments have their own functionalities. Both, Register Activity and Log In Activity demand connecting to the Firebase Authentication while New Measurement Fragment demand connecting to Realtime Database. The name of programming language that was used for coding is *Kotlin*.

Keywords: application, *Firebase*, hypertension, hypotension, *Kotlin*, pressure

ŽIVOTOPIS

Anamaria Franjić rođena je 11.06.2001. godine u Đakovu. Po završetku vrtića, polazi u osnovnu školu, Josipa Antuna Ćolnića Sjever također u Đakovu. Godine 2020. završila je prirodoslovno-matematičku gimnaziju u Đakovu te se upisuje na preddiplomski sveučilišni studij Elektrotehnike, smjer Komunikacije i Informatika na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek.

Potpis autora