Mobilna aplikacija za simuliranje ispita

Štajcer, Davor

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

Sveučilišni studij

MOBILNA APLIKACIJA ZA SIMULIRANJE ISPITA

Završni rad

Davor Štajcer

Osijek, 2020.

Sadržaj

1.UVOD

U današnje vrijeme teško je zamisliti život bez upotrebe mobilnog uređaja, a sa time i upotrebu mobilnih aplikacija, no nekada situacija nije bila kao što je danas. Tvrtka Android, osnovana 2003. godine, je nakon osnivanja krenula razvijati operacijski sustav koji bi mogao biti velika konkurencija tadašnjim popularnim operacijskim sustavima, no ubrzo je kupljena od strane tvrtke Google. Četiri godine kasnije se osniva grupacija Open Handset Alliance koja se zalaže za tzv. *open source* te predstavlja Android operacijski sustav, njihov prvi projekt. Prvi uređaj pokretan Androidom verzije 1.5 se predstavlja 2008. godine [1]. Ovaj rad se fokusira upravo na uređaje koji koriste jednu od verzija Android operacijskog sustava. U drugom poglavlju opisuje se alat korišten pri izradi aplikacija za uređaje koje koriste Android operacijski sustav, Android Studio. Treće poglavlje opisuje programski jezik Kotlin koji se koristi pri izradi aplikacija, opisuje se baza podataka u stvarnome vremenu Firebase koja je korištena za spremanje informacija pojedinog korisnika te za bilježenje postignutih rezultata. Nakon toga se opisuje razvoj aplikacije te se prolazi kroz određene dijelove koda koji se detaljno objašnjavaju.

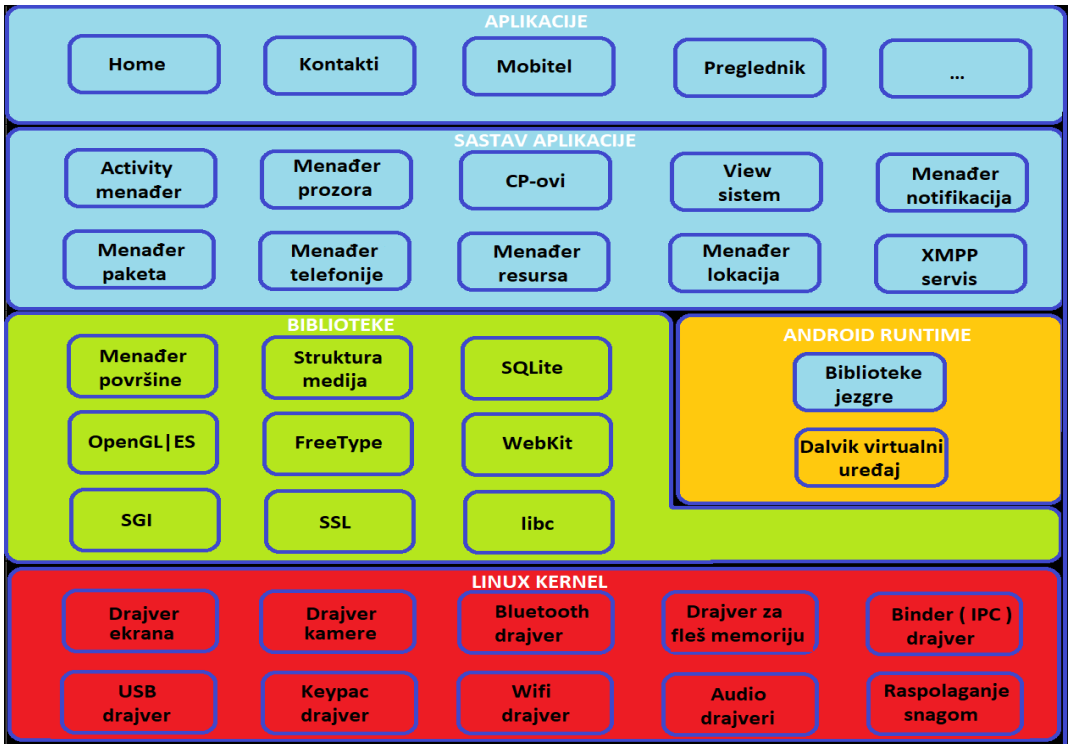
1.1 Zadatak rada

Cilj ovoga rada jest izrada mobilne aplikacije za operacijski sustav Android koja je namijenjena za studente prve godine Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek, ali i za fakultete slične njima ili za bilo koga tko jednostavno želi testirati svoje znanje u određenim kolegijima kao što su Matematika i Fizika. Aplikacija također treba pružati grafičku analizu ostvarenih rezultata te fleksibilan odabir broja pitanja i fleksibilan odabir vremena potrebnog za pisanje ispita.

2. PREGLED KORIŠTENIH TEHNOLOGIJA

2.1 Android operacijski sustav

Android operacijski sustav, osnovan prvobitno od tvrtke Android Inc., prodan je firmi Google 2005. godine. Zasniva se na modificiranoj jezgri Linux 2.6 koji ima modificirane upravljačke programe i modificirane biblioteke, a neke su i dodane, i to sve s ciljem da Android operacijski sustav radi što efikasnije na svim mobilnim uređajima koji ga podržavaju [3]. Android se pokreće pomoću biblioteka jezgre (engl. *Core Libraries*) te virtualnog uređaja zvanog Dalvik (DVM) koji pretvara Java kod (ovdje se koristi Kotlin, ali on je zasnivan na Javi) u DEX format [5]. DEX je zapakiran u tzv. aplikacijski paket koji ima nastavak *.apk* pomoću kojega se vrši instalacija Android aplikacije na mobilni uređaj. Trenutna arhitektura Android operacijskog sustava je prikazana na Slici 2.1. Linux jezgra osigurava upravljačke programe uređaja, upravlja memorijom sustava i procesima te održava funkcionalnost mreže. Sloj biblioteka je pisan u programskome jeziku Java i u njemu je locirana Android specifična biblioteka [4].

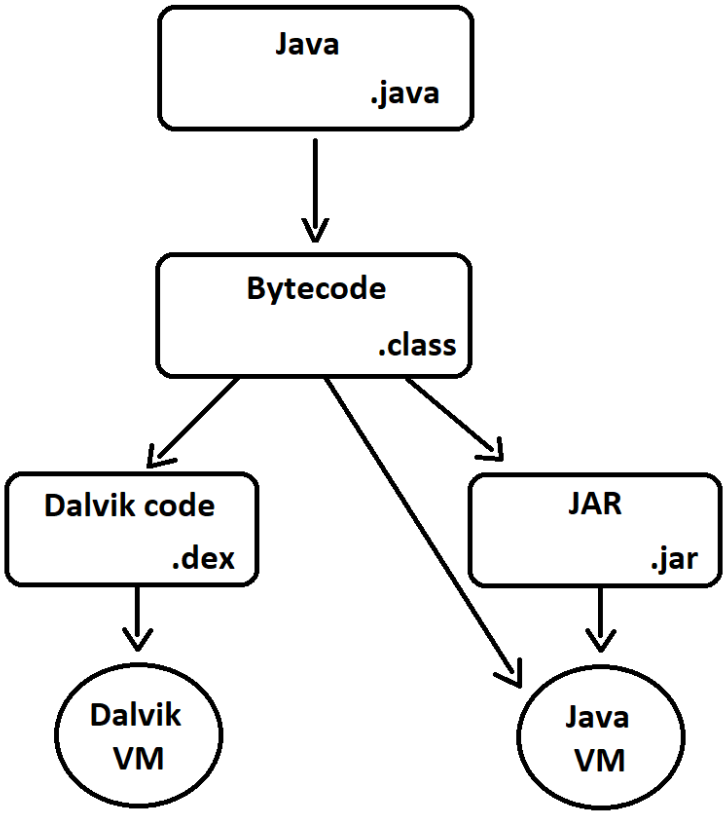


Slika 2.1. *Arhitektura Android operacijskog sustava*

Android *runtime* sloj sadrži već spomenuti Dalvik virtualni uređaj te biblioteke jezgre koje omogućavaju većinu funkcionalnosti Android operacijskog sustava. Android *runtime* sloj stoga omogućava pokretanje više aplikacija u isto vrijeme koristeći Dalvik virtualni uređaj. Aplikacijski sloj je najviši sloj u arhitekturi Android operacijskog sustava te sadržava sve aplikacije koje rade sa krajnjim korisnikom uz pomoć sloja aplikacijskog sučelja (engl. *Framework*) poput internet preglednika, imenika, kalendara, SMS usluge, Google Maps usluge te kamere. Sloj aplikacijskog sučelja sadrži usluge (engl. *Services*) i klase koje su potrebne pri razvoju Android aplikacije. U ovome sloju postoje tzv. menadžeri koji kontroliraju pristup podataka aplikaciji koja se razvija [3]. Sloj aplikacijskog sučelja sadrži *Activity Manager* koji kontrolira životni vijek aplikacije, *Resourse Manager* koji omogućava pristup dokumentima koji nisu programski kod kao što je grafika, *Package Manager* koji dohvaća podatke o instaliranim paketima na mobilnome uređaju, *Location Manager* koji šalje obavijesti i upozorenja kada krajnji korisnik napusti određenu geografsku lokaciju, *Window Manager* koji se koristi za kreiranje *layout* datoteka i *View* objekata, *Notification Manager* koji omogućava aplikacijama da prikazuju korisnički napravljena upozorenja te *Telephony Manager* koji se koristi za upravljanje postavkama mreže i za upravljanje svim informacijama koje su vezane za usluge na mobilnom uređaju [3].

2.1.1 Dalvik virtualni uređaj

Dalvik virtualni uređaj je napravljen i napisan od strane Dan Bornsteina uz doprinos Google zaposlenika i sve u sklopu Android platforme. Dobiva ime po ribarskome selu u Islandu. DVM je baziran na registrima i omogućava pokretanje više virtualnih uređaja odjednom te je optimiziran za zadovoljavanje potreba korištenja malih memorija. Često se DVM naziva *Java Virtual Machine*, ali set instrukcija (engl. *Bytecode)* pomoću kojih on funkcionira nije napisan u Javi nego je pisan sa svojim Dalvik instrukcijama te se pomoću *dx* alata Java klase koje koriste obični Java kompajler prebacuju u datoteku formata *.dex* (Slika 2.2.) [6]. Prednosti DVM-a prema [6] su obavještavanje završetka izvršavanja seta instrukcija, sve klase aplikacije se smještaju u jednu datoteku, čisto tekstualni podaci i druge konstante koje su duplicirane i korištene u više klasa se uključuju samo jednom u *.dex* datoteku zbog održavanja slobodnog prostora, spremanje memorije sa minimalnim brojem ponavljanja i mogućnost rada sa limitiranim resursima.



Slika 2.2. *Funkcionalnost DVM-a*

2.1.2. Upravljanje korištenjem energije

Upravljanje korištenjem energije (engl. *Power Managment*) je jedna od najbitnijih i neophodnih stavki u bilo kojem sistemu današnjice koji uključuje informacijske tehnologije. Razlog tome je neprestano povećanje zahtijevane snage današnjih kompjuterskih sustava. Kao posljedica toga, operacijski sustavi koji su bazirani na Linux jezgri posjeduju karakteristike za štednju energije, ali na trošak latencijskog ponašanja (engl. *Latency Behavior*). Primjeru karakteristika za štednju energije su prema [3] skaliranje napona, onemogućavanje memorije spremljene u tzv. predmemoriji (engl. Cache Memory), reduciranje dinamičke disipacije snage (engl. *Clock Gating*) te aktivacija stanja spavanja. Android operacijski sustav i njegove inačice posjeduju svoju infrastrukturu za upravljanje korištenjem energije. Takva infrastruktura je dizajnirana na pretpostavci da procesor ne bi trebao crpiti ikakvu količinu energije sve dok jedna od aplikacija ili usluga ne zahtjeva korištenje energije. Aplikacije i usluge šalju zahtjeve za

korištenje energije pomoću tzv. ključeva za buđenje (engl. *Wake Locks*) kroz domaće Linux biblioteke i Android aplikacijsko sučelje. U slučaju gdje nema aktivnih ključeva, Android operacijski sustav obustavlja rad procesora [3].

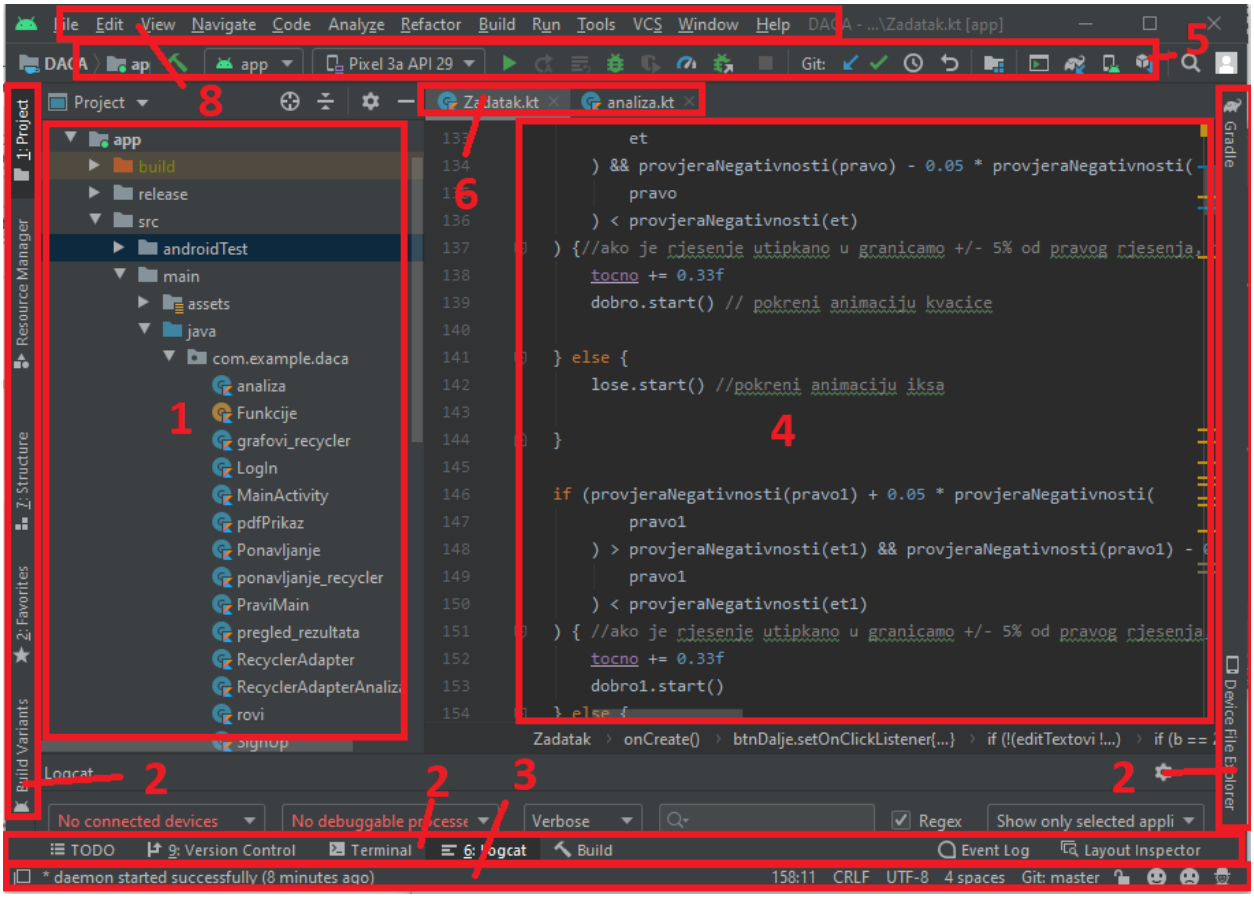
2.1.3. Aplikacije Android operacijskog sustava

Android aplikacije su smještene pomoću AAPT (engl. *Android Asset Packaging Tools*) u Android *.apk* paket. Google također osigurava ADT (engl. *Android Development Tools*) koji pojednostavljuje pretvorbu dokumenata iz *.class* formata u *.dex* format i stvara *.apk* datoteku. Prema [3] Android aplikacije općenito čine aktivnosti (stvaraju ekrane za korištenje krajnjim korisnicima, tj. predstavljaju ono što korisnik zapravo vidi), *intent* objekti (prebacuju mogućnost kontrole s jedne aktivnosti na drugu), pružatelji sadržaja (omogućuju razmjenu informacija između više aplikacija) te usluge (klase koje se izvršavaju u pozadini aplikacije).

2.2 Razvojno okruženje Android Studio

Android Studio je platforma za izradu mobilnih aplikacija na uređajima koji imaju instaliran Android operacijski sustav, tj. on je integrirano razvojno okruženje (engl. *Integrated Development Environment)* za razvijanje Android aplikacija. Baziran je na InteliJ IDEA razvojnom okruženju koji je pisan programskim jezikom Java te služi za proizvodnju kompjuterskih softvera [7]. Android aplikaciju pokreće već spomenuti virtualni uređaj Dalvik zajedno sa bibliotekama jezgre. Pri pokretanju svaka aplikacija posjeduje svoj Linux proces koji se pojavljuje kao instanca Dalvik virtualnog uređaja koji pretvara Java kod u *.dex* format i smješta ga u aplikacijski paket (nastavak *.apk*) pomoću kojega se aplikacija instalira na uređaju. Android operacijski sustav se prema [7] sastoji od više glavnih komponenti kao što su primatelji sadržaja koji se reagiraju na odašiljanja (npr. slaba baterija), servisi što su komponente koje nemaju vlastito sučelje već se pokreću u pozadini te imaju dva načina rada, mogu se pokretati unutar procesa gdje se vrti ostatak aplikacije (lokalni servis) te se mogu pokretati u zasebnom procesu odvojenom od ostatka aplikacije (udaljeni servis), nadalje posjeduje aktivnosti koje služe za prikazivanje korisničkog sučelja pomoću kojega je korisnik u interakciji s aplikacijom, *intent* objekte, što su zapravo poruke koje opisuju operacije koje se izvršavaju, tj. koje će se izvršiti (npr. prelazak iz jedne aktivnosti u drugu) te posjeduje još i davatelje sadržaja (engl. *Content Providers*) koji služe za razmjenu informacija između više različitih aplikacija. Na Slici 2.3. se vidi izgled Android Studio razvojnog okruženja uz primjer otvorenog projekta ovog rada. Izgled Android Studio

razvojnog okruženja prema [11] se može podijeliti na sedam dijelova: *Project Explorer,* gdje se mogu vidjeti sve datoteke u aplikaciji ili u cijelome projektu, nove datoteke su automatski dodane a programer po potrebi otvara klase, objekte, sučelja, aktivnosti i druge komponente na kojima se radi u Android Studio razvojnom okruženju (na slici broj 1), *Tool Buttons* koji se nalazi na krajevima sučelja Android Studio razvojnog okruženja te služi za otvaranje različitih manjih prozora koji prikazuju stvari poput programskih datoteka, strukture projekta, *logcat* gdje nam Android Studio šalje poruke o kodu na kojem se trenutno radi, itd.(na slici broj 2), *Status Bar* koji programeru prikazuje neke od informacija vezane za trenutno uređivanje koda, npr. pozicija miša (na slici broj 3), *Editor* gdje programer koji razvija aplikaciju u Android Studio razvojnom okruženju uređuje cijeli svoj kod otvaranjem datoteka za koje je odredio da su klase, objekti i slično, a otvaranjem datoteka za koje je odredio da su *layout* datoteke (*.xml*) *Editor* se transformira u okruženje za dizajniranje *View* objekata gdje se *drag and drop* tehnikom može namjestiti izgled pripadajućih *layout* datoteka a postoji i opciju odabira mijenjanja izgleda XML kodom (na slici broj 4), *Toolbar* gdje se nalaze ikone za brzo pokretanje pojedinih opcija u Android Studio razvojnom okruženju za koje se smatra da se najčešće upotrebljavaju što omogućava programeru brži (a time i bolji) razvoj koda same aplikacije (na slici broj 5), *Editor Tabs* koji prikazuje trenutno otvorene datoteke koje su podložne izmjenama ili datoteku koja se trenutno uređuje (na slici broj 6), te *Menu Bar* gdje programer koji razvija aplikaciju može doći do bilo koje opcije koja mu je potrebna**.** (na slici broj 8). Posebni dijelovi koji se nalaze u *Project Exploreru* (na slici broj 1) jesu dvije *build.gradle* datoteke (Slika 2.4. i 2.5.) [7]. Jedna od *build.gradle* datoteka prema [12] je na razini projekta i služi za definiranje *build* konfiguracije koji se primjenjuje na sve module u projektu. Druga od *build.gradle* datoteka je na razini aplikacije i locirana je u direktoriju *project/module,* ovdje se definira SDK verzija i definiraju se svi *dependency* segmenti koji služe za uključivanje vanjskih biblioteka pomoću naredbe *implementation* i ostalih modula biblioteka što omogućava olakšanu kontrolu nad vanjskim bibliotekama pri razvoju aplikacije. Android Studio kreira novu *build.gradle* skriptu dodavanjem novog modula u datoteku u kojoj nam se nalazi projekt [2]. Unutar bloka naredbom *applicationId* se definira ime paketa, a naredbama *targetSdkVersion* i *minSdkVersion* se određuje raspon uređaja koji mogu koristi aplikaciju.



Slika 2.3. *Sučelje Android Studio razvojnog okruženja*

```
apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-android-extensions'
apply plugin: 'com.google.gms.google-services'

android {
    compileSdkVersion 29
    buildToolsVersion "29.0.3"
    defaultConfig {
        applicationId "com.example.daca"
        minSdkVersion 23
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
        vectorDrawables.useSupportLibrary = true

    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
```

Slika 2.4. *Prvi dio build.gradle datoteke na razini aplikacije*

```
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = "1.8"
    }
}

dependencies {
    implementation 'com.github.barteksc:android-pdf-viewer:2.8.2'
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:1.3.72"
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.core:core-ktx:1.3.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    implementation 'com.google.firebase:firebase-auth:19.3.1'
    implementation 'com.google.firebase:firebase-core:17.4.3'
    implementation 'com.firebaseui:firebase-ui-auth:4.3.1'
    implementation 'com.google.android.material:material:1.1.0'
    implementation 'androidx.navigation:navigation-fragment-ktx:2.3.0'
    implementation 'androidx.navigation:navigation-ui-ktx:2.3.0'
    testImplementation 'junit:junit:4.12'
    implementation 'com.github.PhilJay:MPAndroidChart:v3.0.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
    implementation 'androidx.recyclerview:recyclerview:1.1.0'
    implementation 'com.google.firebase:firebase-database:19.3.1'
}
```

Slika 2.5. *Drugi dio build.gradle datoteke na razini aplikacije*

2.3 Kotlin programski jezik

Kotlin je programski jezik za razvoj aplikacija upotrebljivih na više platformi. Pokreće ga Java virtualni uređaj. Razvija ga tvrtka pod imenom JetBrains. Kotlin se prema [8] kao programski jezik u javnosti prvi puta pojavljuje 2012. godine, no prva stabilna verzija tog programskog jezika se javlja tek 2016. godine. Kada se govori o samoj primjeni programskog jezika Kotlin on je primjenjiv ondje gdje se primjenjuje i programski kod Java jer je Kotlin sam po sebi građen na Java programskom jeziku, tj. on je nastao kao pokušaj nadopune nedostataka programskog jezika Java. Ako se usredotoči na razvijanje Android aplikacija pomoću programskog jezika Kotlin, on naspram Java programskog jezika posjeduje više prednosti prema [8] kao što su smanjenje koda uz veću čitljivost svoga, a i koda od programera sa kojima programer zajedno piše kod, korištenje koda paralelno sa Java kodom, tj. ako programer ima kod već prethodno napisan u programskome jeziku Java ne mora prepravljati cijeli kod u Kotlin (Android ima i opciju da cijeli kod pretvara iz jednog jezika u drugi), podržavanje više platforma, tj. može se koristiti (osim za Android) i za internet aplikacije i aplikacije iOS operacijskog sustava, tj. iznimno olakšava prelazak sa jedne platforme na drugu. Nadalje, Kotlin posjeduje manje pogrešaka u kodu (što proizlazi direktno iz

toga što jednostavno treba manje koda za pisanje) i lagan je za naučiti (pogotovo ako osoba ima već prijašnjeg iskustva sa radom u Javi). Karakteristike programskog jezika Kotlin koje se razlikuju od Jave, a korištene su pri izradi Android aplikacije su *val* i *var* varijable (Slika 2.6.), razlika je u tome što je *var* promjenjiva varijabla, tj. vrijednost se kroz kod može mijenjati više puta, dok val predstavlja konstantnu varijablu i može biti inicijalizirana samo jedan put (nešto kao *final* varijabla u Javi), nadalje dolazi jako bitna stavka Kotlin programskog jezika a to su lambda varijable (Kod 2.1. i 2.2.) i one su funkcije koje mogu biti parametri u drugim funkcijama, a pri pozivu funkcije se na mjestu parametra vitičastim zagradama označava tijelo, tj. funkcionalnost same funkcije (u ovom primjeru je tijelo lambda funkcije napisano izvan zagrada funkcije koja ju poziva, to se može napraviti samo ako je lambda funkcija zadnji parametar u funkciji koja ga poziva), a sam parametar (ili parametri) koji se predaje lambda funkciji je označen kao varijabla iza znaka '->' (nazive tih varijabli se mogu mijenjati) [9]. Još jedna od prednosti Kotlina u razvoju Android aplikacija nad Javom je nekorištenje metode *findViewById()* za svaki *View* objekt *layout* datoteke korištene od strane određene klase (za *View* objekte koje nisu u pripadnoj *layout* datoteci kojeg koristi klasa se metoda treba izvršavati) što smanjuje vrijeme potrebno za rad aplikacije, a i oduzima par linija koda.

```
private val g = 9.81f
private var tocno: Float = 0f //varijabla za pracenje koliko je zadataka
private var b = 0 //sluzi za namjestavanje teksta na gumbu i da vidim kad
private var i = 0 //broji koji je zadatak na redu u funkciji setText()
var bla: Boolean = false //odlucuje je li rjesenje slika ili textView
lateinit var countDownTimer: CountDownTimer
```

Slika 2.6. *Varijable val i var*

```
class RecyclerAdapter(
private val context: Context,
private val list: ArrayList<String>,
                    private val itemClick : (String) ->Unit
) : RecyclerView.Adapter<RecyclerAdapter.ViewHolder>() {
```

Kod 2.1. *Definiranje lambda parametra*

```
val adapter = RecyclerAdapter(this,list){string ->
    val currentUser = FirebaseAuth.getInstance().currentUser!!.uid
FirebaseDatabase.getInstance().getReference("Users").child(currentUser).child("Sti
snuto")
.setValue(string)
    val intent = Intent(this,UnosPitanjaITrajanja::class.java)
    startActivity(intent)
```

Kod 2.2. *Pozivanje funkcije koja sadrži lambda parametar*

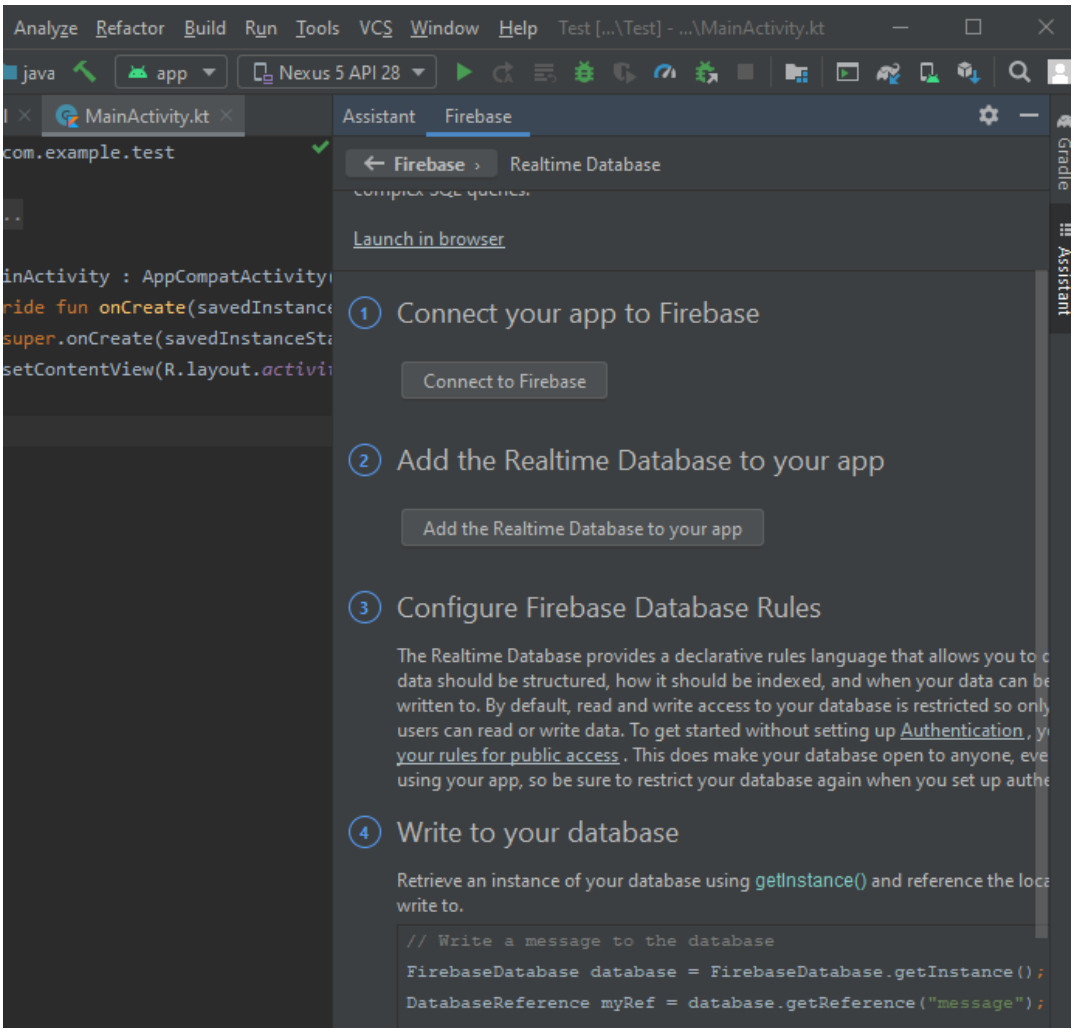
2.4 Firebase platforma za razvijanje aplikacija

Firebase je platforma za kreiranje iOS, Web i Android aplikacija uz mogućnost stvaranja baze podataka koja dodaje, briše i mijenja podatke u stvarnome vremenu, omogućava različite API-ove (*Application Programming Interface*, koji uzima zahtjeve od korisnika ili od računala i prosljeđuje ih serveru, i obrnuto), omogućava više načina pristupanja bazi podataka, itd. Osigurava SDK za iOS, Web, NodeJS, C++, Java Server te za Android [10]. Jedan od glavnih razloga zašto je Firebase korišten danas od određenih tvrtki koje se bave razvijanjem sličnih aplikacija jest rješavanje problema prijave i registracije korisnika, što je zapravo jako zahtjevno za napraviti i još zahtjevnije za održavati. Firebase platforma nudi jako jednostavnu implementaciju toga u Android Studio razvojno okruženje, što će biti opisano u narednim poglavljima rada. Primjena Firebase platforme u ovome radu je samo jedna od mnogih mogućnosti koje ona zapravo pruža, tj. u ovome projektu su korišteni segmenti Firebase platforme vezani za samu gradnju aplikacije, no mogu se također koristiti segmenti za poboljšanje kvalitete aplikacije kao što su poboljšanje grafike, testiranje i analiziranje rada aplikacije. Također se pružaju segmenti koji su više vezani za poboljšanje firmi kao što su analiziranje, razne vanjske biblioteke, posebna testiranja, dinamični linkovi i drugi.

3.RAZVOJ APLIKACIJE

3.1 Spajanje aplikacije sa bazom podataka

Prije kretanja sa razvojem aplikacije (većina aplikacija koristi podatke iz baze podataka) ona se mora spojiti sa bazom podataka u stvarnome vremenu i mora se uvesti mogućnost autentifikacije. Da bi se Android Studio mogao spojiti sa bazom podataka prvo se treba napraviti projekt na Firebase internet stranici. Nakon napravljenog novog projekta aplikacija se može ručno spajati sa Firebase projektom ili se može iskoristiti pogodnost Android Studio razvojnog okruženja. Pogodnost Android Studio razvojnog okruženja je u tome što ima već u sebi ugrađenu opciju spajanja Firebase platforme sa Android aplikacijom uz pojednostavljenu implementaciju potrebnih informacija.

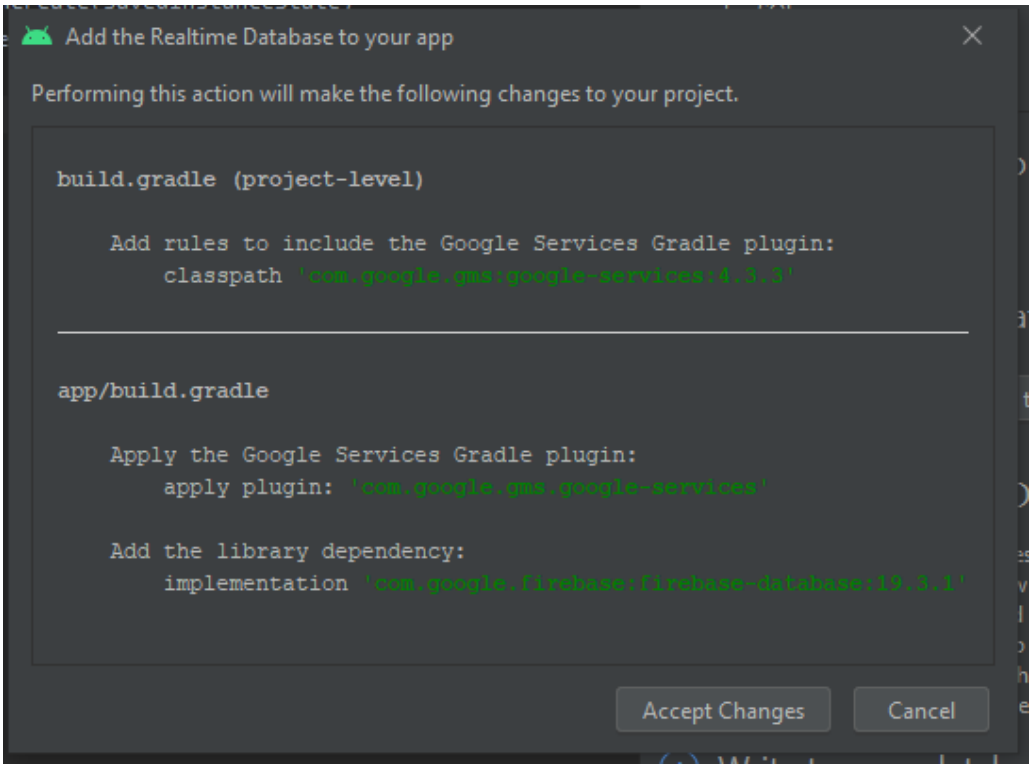


Slika 3.1. *Firebase izbornik u Android Studio razvojnom okruženju*

Prvi korak je se odabir opcije iz *Menu Bara* pod nazivom *Tools* gdje se klikom na opciju *Firebase* otvara sa desne strane izbornik sa svim mogućim karakteristikama Firebase platforme koje se mogu ugraditi u Android aplikaciju koja se razvija (Slika 3.1.). Redoslijed implementacija Firebase komponenata u projekt ne igra ulogu. Firebase se spaja sa aplikacijom klikom na gumb *Connect to Firebase* pod stavkom jedan. Nakon toga se pojavljuje prozor gdje programer dobiva opciju biranja već postojećeg projekta na Firebase internetskoj stranici, ali isto tako može sa tog mjesta stvoriti novi projekt u kojem se onda implementira pripadajuća Android aplikacija.

3.2 Baza podataka u stvarnome vremenu

Izvršeno je povezivanje Android aplikacije sa Firebase platformom. Slijedeći korak je dodavanje baze podataka u koju se podaci mogu dodavati, brisati i mijenjati u stvarnome vremenu, tj. *Realtime Database*. Ovo se izvršava odabirom gumba *Add the Realtime Database to your app* prikazanog na Slici 3.1. Ovim odabirom se otvara novi prozor prikazan na Slici 3.2.

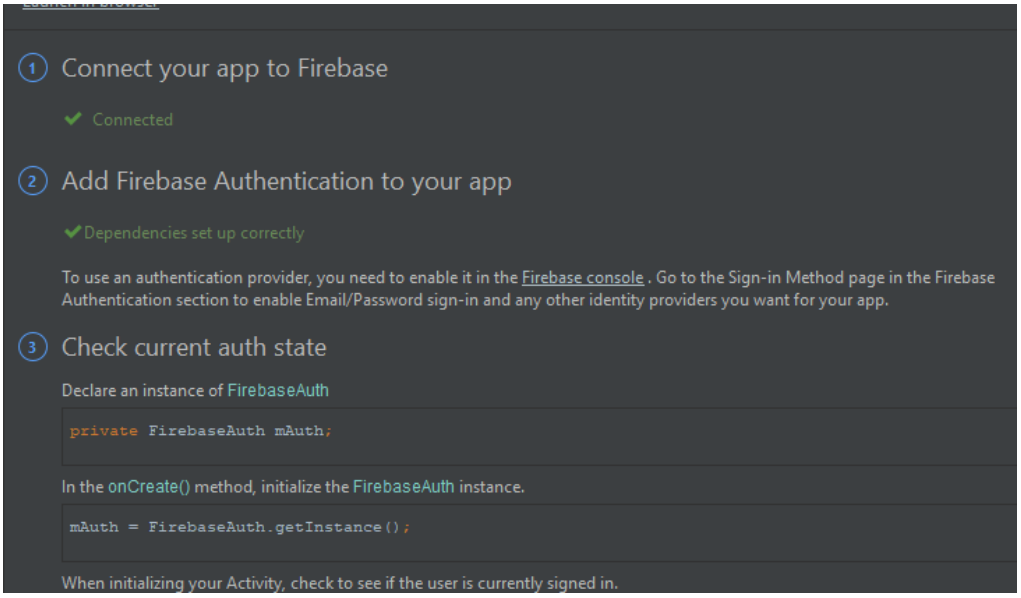


Slika 3.2. *Dodavanje baze podataka u stvarnome vremenu*

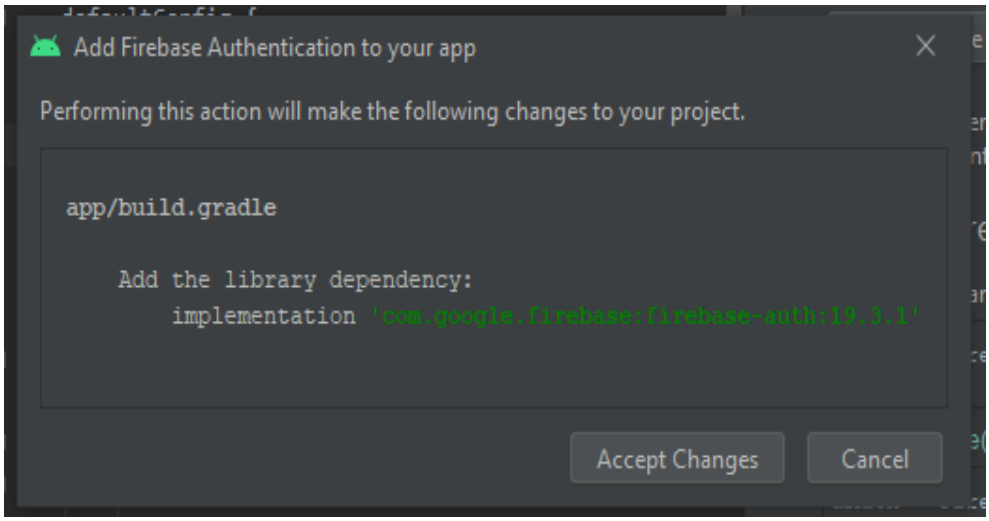
Sadržaj prikazan na Slici 3.2. prikazuje promjene koje Android Studio želi automatski unijeti u trenutni projekt radi realizacije baze podataka u stvarnome vremenu. Promjene se događaju upravo u *build.gradle* datotekama navedenim u poglavlju 2.2. Prvobitno se unose promjene u *build.gradle* datoteku koja je na razini projekta zbog unosa pravila za unošenje *Google Service Gradle* vanjske datoteke. U *build.gradle* datoteku na razini aplikacije modula se također automatski dodaje biblioteka za bazu podataka u stvarnome vremenu naredbom *implementation*. Postupak koji je upravo opisan se mogao napraviti ručno dodavajući linije koda na Slici 3.2. u odgovarajuće *build.gradle* datoteke.

3.3 Autentifikacija

Izvršeno je povezivanje novostvorenog projekta na Firebase internetskoj stranici i dodana je baza podatka u stvarnome vremenu. Slijedeći korak je dodavanje Firebase autentifikacije u Android aplikaciju odabirom gumba *Add Firebase Authentication to your app* pod stavkom broj dva u Firebase izborniku autentifikacije. Stavka se preskače jer je Android aplikacija već prethodno povezana sa Firebase platformom. Pojavljuje se novi prozor gdje se mogu vidjeti promjene koje se događaju u Android aplikaciji pri implementaciji Firebase autentifikacije. Ovoga puta promjene se događaju samo u *build.gradle* datoteci na razini aplikacije gdje se implementira biblioteka za Firebase autentifikaciju.



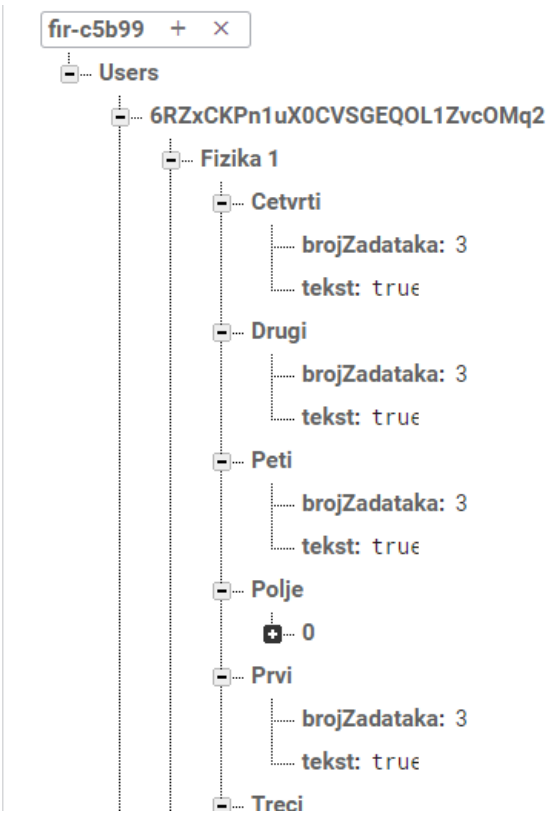
Slika 3.3. *Firebase izbornik autentifikacije*



Slika 3.4. *Dodavanje Firebase autentifikacije*

3.4 Postavljanje podataka u bazu podataka

Podaci se u Firebase bazu podataka u stvarnome vremenu zapisuju u JSON (engl. *JavaScript Object Notation*) formatu i posjeduju nastavak *.json*. JSON je tekstualno baziran otvoreni standard te je dizajniran za razmjenu podataka koja je čitljiva programeru. Ovakav format je stvorio Douglas Crockford prema [13] i jedna od glavnih primjena mu je prijenos podataka između mrežne aplikacije i servera. Bitna karakteristika JSON standarda jest što ga moderni programski jezici mogu primjenjivati (pa tako i u Android Studio razvojnom okruženju koji koristi Javu ili Kotlin) te je ne neovisan o jeziku u kojem se piše [13]. Sama sintaksa JSON tekstualnog standarda je kao jedan podskup JavaScript sintakse. Na Slici 3.5. se može vidjeti baza podataka u stvarnome vremenu od projekta koji je povezan sa ovom Android aplikacijom. Uspoređujući Sliku 3.5. sa Slikom 3.6. postiže se dobar uvid u funkcionalnost sintakse JSON tekstualnog standarda. Sintaksa se prema [13] sastoji od 4 pravila : 1. podaci su predstavljeni u ime/vrijednost parovima, 2. vitičaste zagrade sadrže objekte i iza svakog imena slijedi ‘:’ znak, 3. ime/vrijednost parovi su odvojeni zarezima, 4. uglate zagrade sadrže polja kao što se vidi na Slici 3.6., a članovi polja su odvojeni zarezom. Podaci iz Firebase baze podataka se dohvaćaju pomoću jedne od 3 metoda: *addValueEventListener()*, *addChildEventListener()* ili *addListenerForSingleValueEvent().*



Slika 3.5. *Baza podataka u stvarnome vremenu*


```
{
  "Users" : {
    "6RZxCKPn1uX0CVSGEQOL1ZvcOMq2" : {
      "Fizika 1" : {
        "Cetvrti" : {
          "brojZadataka" : 3,
          "tekst" : true
        },
        "Drugi" : {
          "brojZadataka" : 3,
          "tekst" : true
        },
        "Peti" : {
          "brojZadataka" : 3,
          "tekst" : true
        },
        "Polje" : [ {
          "x" : 0,
          "y" : 0
        } ],
        "Prvi" : {
          "brojZadataka" : 3,
          "tekst" : true
        },
        "Treci" : {
```


Slika 3.6. *JSON format baze podataka*

Prema [14] metoda *onCancelled()* prima argument tipa klase *DatabaseError* i ta metoda se aktivira ako *listener* nije uspio na serveru ili ako je maknut zbog zaštite podataka i pravila pristupa Firebase bazi podataka. Objekt klase *DatabaseError* sadrži opis pogreške koja se dogodila. Metoda *onDataChange()* sadrži kao argument objekt klase *DataSnapshot* koji sadrži podatke u tom trenutku na lokaciji reference na bazu zvanu *base*, a metoda se aktivira odmah i svaki puta kada se podaci sačuvani u tom *DataSnapshot* objektu promjene.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_zadatak)
    val switch = Funkcije.switch
    val a = intent.getStringExtra("Broj")!!.toInt() //broj zadataka koje je
korisnik odlucio pisati (izmedju 2 i 5)
    base.addValueEventListener(object : ValueEventListener {
        override fun onCancelled(p0: DatabaseError) {
        }

        override fun onDataChange(p0: DataSnapshot) {
            tvPredmet.text = p0.value.toString()

            setText()
            if (switch) {
                val time: Int =
intent.getStringExtra("Vrijeme")!!.toFloat().toInt()
                startTimer((time * 1000 * 60), a)
                tvTimer.visibility = View.VISIBLE
                countDownTimer.start()
            }


        }
    })
```

Kod 3.1. *Korištenje metode addValueEventListener()*

Na Kodu 3.2. se vidi primjena metode *addChildEventListener*(). Metoda kao argument prima objekt klase koja nasljeđuje sučelje *ChildEventListener,* a sa time i metodu *onCancelled()* koja prima argument tipa klase *DatabaseError.* Metoda *onCancelled()* se aktivira ako *listener* nije uspio na serveru ili ako je maknut zbog zaštite podataka i pravila pristupa Firebase bazi podataka. Nasljeđivanjem sučelja *ChildEventListener* klasa također dobiva metodu *onChildAdded()* koja se aktivira svaki puta kada se dodaje novo tzv. dijete na lokaciju na kojoj je *listener* stavljen. Metoda prima argument tipa *DataSnapshot* koji sadrži nepromjenjiv podatak na novoj lokaciji na kojoj je dijete kreirano te argument tipa *String?* (upitnik u Kotlinu znači da taj tip podatka može biti *null*,

tj. prazan) koji sadrži ime djeteta koji je po redu prije novokreiranog (ako je novokreirani prvi, vrijednost je *null*, zato je potreban upitnik kod tipa podatka).

```
private val base = FirebaseDatabase.getInstance().getReference("Users")
    .child(FirebaseAuth.getInstance().currentUser!!.uid).child("Stisnuto")

base.addChildEventListener(object : ChildEventListener {
    override fun onCancelled(error: DatabaseError) {
        TODO("Not yet implemented")
    }

    override fun onChildMoved(snapshot: DataSnapshot, previousChildName: String?)
{
        TODO("Not yet implemented")
    }

    override fun onChildChanged(snapshot: DataSnapshot, previousChildName:
String?) {
        TODO("Not yet implemented")
    }

    override fun onChildAdded(snapshot: DataSnapshot, previousChildName: String?)
{
        TODO("Not yet implemented")
    }

    override fun onChildRemoved(snapshot: DataSnapshot) {
        TODO("Not yet implemented")
    }

})
```

Kod 3.2. *Korištenje metode addChildEventListener()*

Treća metoda je *onChildChanged()* koja se aktivira svaki puta kada se podatak djeteta na lokaciji na kojoj je usmjeren *listener* promjeni. Ona prima argument tipa *DataSnapshot* koji sadrži nepromjenjiv podatak o djetetu na lokaciji gdje je vrijednost djeteta promijenjena te argument tipa klase *String?* koji sadrži ime djeteta koji je po redu prije novokreiranog. Četvrta metoda koja se nasljeđuje od sučelja je *onChildMoved()* koja se aktivira svaki puta kada se prioritet na djetetovoj lokaciji promjeni. Prioriteti djece se prema [15] postavljaju pri postavljanju vrijednosti korištenjem metode *setValue(data,priority*), a za promjenu prioriteta se koristi metoda *setPriority()*. Djeca se slažu prema prioritetima tako da djeca bez prioriteta idu prva, zatim djeca kojima je prioritet broj (idu od manjeg ka većem), djeca koji im je prioritet u tekstualnom obliku, tj. ako im je prioritet tipa klase *String*, dolaze zadnja. U slučaju da djeca imaju isti prioritet oni su

poredani po svojim ključevima na način da prvi dolaze numerički ključevi nakon čega slijede ostali. Metoda *onChildMoved()* sadrži argument tipa klase *DataSapshot* koji uzima nepromjenjiv podatak na lokaciji djeteta kojemu je prioritet promijenjen te argument tipa klase *String?* koji sadrži ime djeteta koji je po redu prije djeteta na čijoj je lokaciji prioritet promijenjen. Zadnja metoda korištena od sučelja *ChildEventListener* je metoda *onChildRremoved()* koja se aktivira svaki puta kada je dijete izbrisano sa lokacije na koju je *listener* dodan. Dodavajući metodu *addListenerForSingleValueEvent()* (Kod 3.3.) se može vidjeti da nasljeđuje iste metode kao i metoda *addValueEvenetListener()*. Razlika između njih dvoje jest u tome što metoda *addListenerForSingleValueEvent()* poziva metodu *onDataChange()* samo jednom iako se događaju više promjena podataka na serveru, dok *addValueEvenetListener()* poziva tu metodu pri svakoj promjeni.

```
private val base = FirebaseDatabase.getInstance().getReference("Users")
    .child(FirebaseAuth.getInstance().currentUser!!.uid).child("Stisnuto")

base.addListenerForSingleValueEvent(object : ValueEventListener{
    override fun onCancelled(error: DatabaseError) {
        TODO("Not yet implemented")
    }

    override fun onDataChange(snapshot: DataSnapshot) {
        TODO("Not yet implemented")
    }

})
```

Kod 3.3. *Korištenje metode addListenerForSingleValueEvent()*

3.5 Dohvaćanje podataka iz baze podataka

Preostalo je još razmotriti upisivanje podataka u bazu podataka u stvarnome vremenu. Ovo se postiže tako da se kreira referenca na određeni dio baze podataka (ili cijelu bazu). Na Kodu 3.4. je ta referenca spremljena u nepromjenjivu varijablu *baza*. Na primjeru sa Koda 3.4. je pokazano kako se postavlja podatak jednog djeteta pomoću metode *setValue()* koja jednostavno prima tip podatka koji se sprema u bazu podataka u stvarnome vremenu. No, ne moraju se podaci u Firebase bazu podataka unositi na način da se unosi jedno po jedno dijete kao na Kodu 3.4.

```
auth.createUserWithEmailAndPassword(etSignUpEmail.text.toString(),etSignUpPassword
.text.toString())
    .addOnCompleteListener(this) { task ->
        if (task.isSuccessful) {
            Toast.makeText(baseContext, "Registirani ste",
Toast.LENGTH_LONG).show()
            val intent = Intent(this,LogIn::class.java)
            val currentID = FirebaseAuth.getInstance().currentUser!!.uid
            val baza =
FirebaseDatabase.getInstance().getReference("Users").child(currentID)
            val Polje = mutableListOf(Entry(0f,0f))

            baza.child("email").setValue(etSignUpEmail.text.toString())
            baza.child("password").setValue(etSignUpPassword.text.toString())
            baza.child("Matematika").child("brojPokusaja").setValue(0)
            baza.child("Matematika 2").child("brojPokusaja").setValue(0)
            baza.child("Osnove elektrotehnike
1").child("brojPokusaja").setValue(0)
```

Kod 3.4. *Pisanje podataka u Firebase bazu podataka*

Drugi način je unos podataka pomoću klase (Kod 3.5.) čiji su parametri imena djece koja se unose (ili mijenjaju) u bazu, a metodi *setValue()* se kao argument predaje objekt takve klase (Kod 3.6.). Nazive djeteta (tj. parametara klase) Android Studio prikazuje na dosta praktičan način, tako da ih označava laganom sivom bojom, što se može vidjeti iz priloženog.

```
class Unos3(private val brojZadataka : Int,val t1 : String,val t2 :String, val t3
:  String,val tekst : Boolean)
```

Kod 3.5. Klasa za unos više podataka u bazu

```
//matematika1
baza.child("Matematika").child("Prvi").setValue(Unos3(3,"Odredite domenu zadane
funkcije.",
    "Odredite domenu zadane funkcije.","Odredite domenu zadane funkcije.",false))
```

Kod 3.6. *Unos više podataka u bazu odjednom*

3.5.1. Dohvaćanje korisnika

Primjer dohvaćanja reference korisnika se može vidjeti na Kodu 3.7. gdje se dohvaća ID korisnika (varijabla *uid* na Kodu 3.7.). Dohvaćanje ostalih podataka o korisniku kao što je *e-mail* ili lozinka izvršava se preko baze podataka u stvarnome vremenu.

```
val currentUser = FirebaseAuth.getInstance().currentUser!!.uid
FirebaseDatabase.getInstance().getReference("Users").child(currentUser).child("Sti
snuto").setValue(string)
```

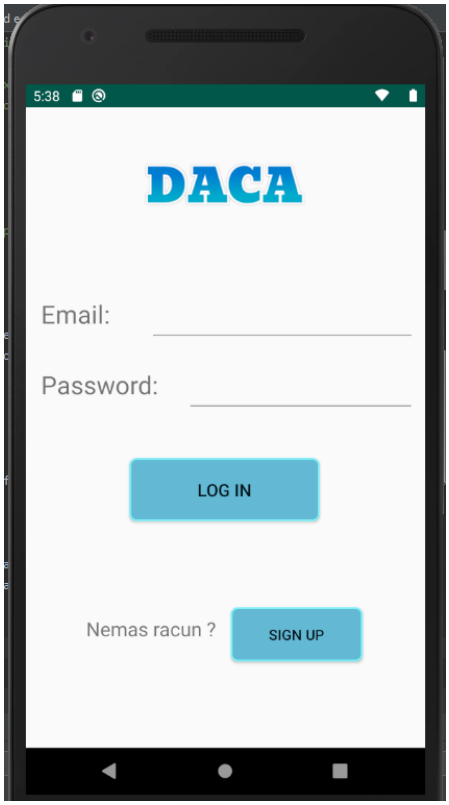
Kod 3.7. *Dohvaćanje trenutnog korisnika*

```
fun makeUser(){
    if(etSignUpEmail.text.isEmpty()){
        etSignUpEmail.error = "Unesite email adresu"
        etSignUpEmail.requestFocus()
        return
    }if(!Patterns.EMAIL_ADDRESS.matcher(etSignUpEmail.text.toString()).matches()){
        etSignUpEmail.error = "Unešena adresa nije valjana"
        etSignUpEmail.requestFocus()
        return
}if(etSignUpPassword.text.isEmpty()){
        etSignUpPassword.error ="Unesite sifru"
        etSignUpPassword.requestFocus()
        return}
auth.createUserWithEmailAndPassword(etSignUpEmail.text.toString(),etSignUpPassword
.text.toString()).addOnCompleteListener(this) { task ->
            if (task.isSuccessful) {
                Toast.makeText(baseContext, "Registirani ste",
Toast.LENGTH_LONG).show()
                val intent = Intent(this,LogIn::class.java)
                val currentID = FirebaseAuth.getInstance().currentUser!!.uid
                val baza =
FirebaseDatabase.getInstance().getReference("Users").child(currentID)
                val Polje = mutableListOf(Entry(0f,0f))
            //Unošenje podataka u bazu podataka poslije ovoga..
  startActivity(intent)
                finish()
            } else {
                Toast.makeText(baseContext, "Unešena adresa je već postojana ili
pokušajte kasnije.", Toast.LENGTH_SHORT).show()
            }
        }
}
```

Kod 3.8. *Kreiranje novog korisnika*

Prilikom stvaranja novog računa korisnika (engl. *Sign Up*) na Android aplikaciji podaci jedinstveni njemu potrebni za ulazak u aplikaciju (engl. *Log In*) se spremaju u Firebase bazu podataka. Način na spremanja se može vidjeti na Kodu 3.8. Što korisnik zapravo vidi pri prijavi i nakon prijave se može vidjeti na Slici 3.7. i Slici 3.8.

Slika 3.7. *Aktivnost nakon uspješne prijave*



Slika 3.8. *Aktivnost za prijavu*

3.6. Izrada kutka za ponavljanje gradiva

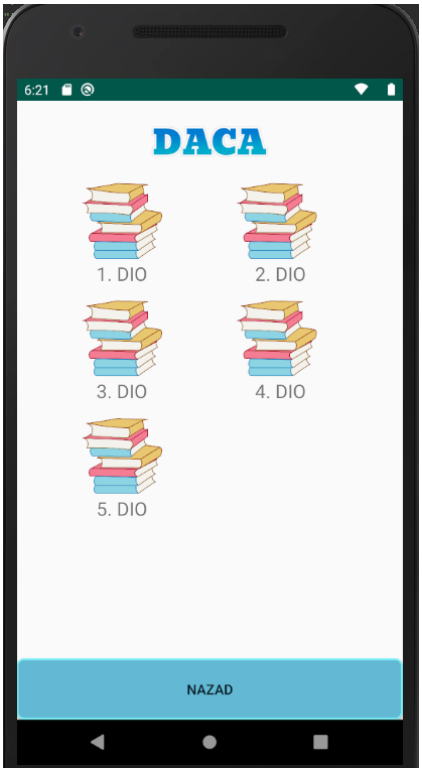
Ukoliko je korisnik suočen sa velikim problemima pri rješavanju zadataka, napravljen je kutak za ponavljanje koji sadrži sve materijale potrebne za rješavanje zadataka u *.pdf* obliku. Ova značajka aplikacije omogućena je dodavanjem vanjske biblioteke sa GitHub platforme. Dodavanje biblioteke je prikazano na Slici 3.9. gdje se naredbom *implementation* u *build.gradle* datoteku na razini aplikacije dodaje spomenuta biblioteka. Jedina metoda koja se koristi u programu iz ove biblioteke jest *pdfView.fromAsset(imePdfDatoteke).show()*.

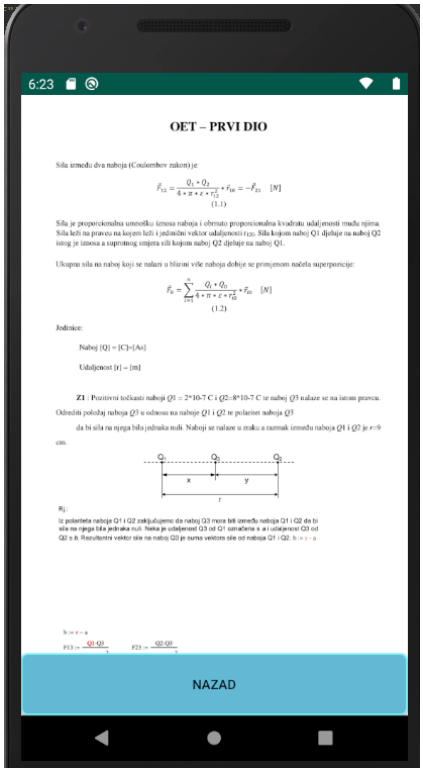
```
dependencies {
    implementation 'com.github.barteksc:android-pdf-viewer:2.8.2'
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:1.3.72"
```

Slika 3.9. *Implementiranje vanjske biblioteke za renderiranje datoteka sa .pdf ekstenzijom*

Korisnik pristupa ponavljanju gradiva klikom na sliku koja prikazuje knjige (Slika 3.7.) nakon čeka se otvara aktivnost koja izgleda identično poput aktivnosti na Slici 3.11. gdje korisnik

odabire predmet koji želi ponoviti/učiti. Odabirom željenog predmeta se korisniku otvara aktivnost sa cijelim gradivom kolegija podijeljenog na više cjelina predstavljenih pomoću slika knjiga (Slika 3.10.). Broj cjelina na koje je kolegij podijeljen ovisi o samome kolegiju. Odabirom cjeline za ponavljanje se otvara aktivnost koja prikazuje *.pdf* datoteku koja se može micati vertikalno po ekranu što se vidi na Slici 3.11.





Slika 3.10. *Izbor dijela gradiva za ponavljanje*

Slika 3.11. *Primjer dijela gradiva za ponavljanje*

3.7. Izrada izmjene zadataka za određeni predmet

Korisnik prvobitno odabire predmet kako je prikazano na Slici 3.12. Za kvalitetno rješavanje ispita korisnik odabirom predmeta ima opciju biranja broja zadataka na ispitu (od dva do pet) te može odabrati opciju vremenskog ograničenja ispita (od 25 do 90 minuta) što se može vidjeti na Slici 3.13. Cijeli ispit, osim unošenja upravo spomenutih parametara se odvija u jednoj aktivnosti. Na Kodu 3.9. je prikazan sadržaj *onCreate()* metode (bez *OnClickListenera*) i parametara koji pomažu pri prelaženju sa jednog zadatka na drugi, pri prikazivanju određenih *View* objekata, postavljanu teksta zadatka i druge funkcionalnosti. U metodi *onCreate()* se dohvaća iz Firebase baze podataka podatak o korisniku te ako je postojao ikakav tekst upisan u *editText* komponentu on se čisti te se postavlja redni broj zadatka koji se piše.

Slika 3.12. *Odabiranje zadatka*

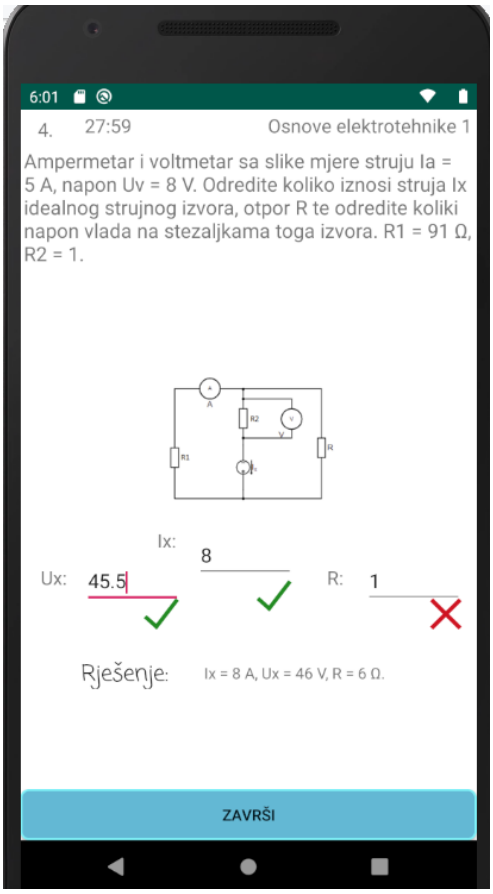
Slika 3.13. *Unos broja zadataka i vremena*

Svaki od mogućih pet zadataka ima nekoliko varijacija zadataka, tako npr. iz kolegija Osnove elektrotehnike 1 zadatak dva ima četiri varijacije na taj zadatak, tj. četiri različita zadatka

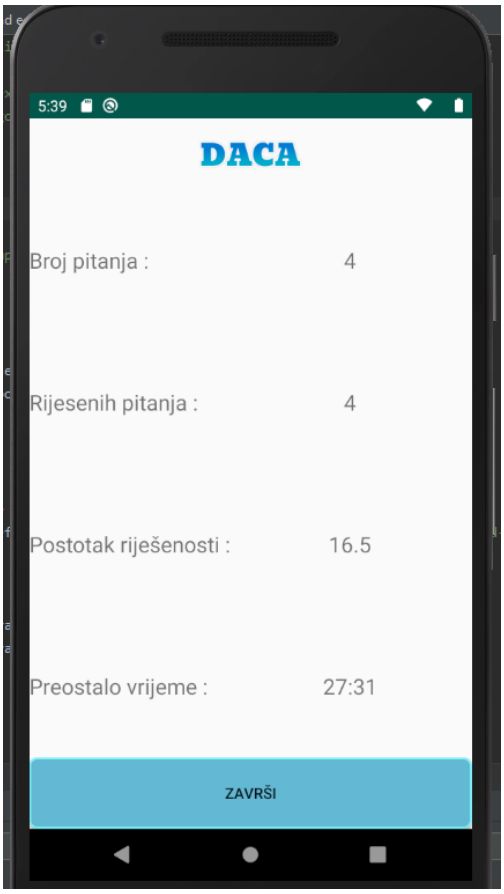
```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_zadatak)
    val switch = Funkcije.switch
    val a = intent.getStringExtra("Broj")!!.toInt() //broj zadataka koje je
    base.addValueEventListener(object : ValueEventListener {
        override fun onCancelled(p0: DatabaseError) {
        }
        override fun onDataChange(p0: DataSnapshot) {
            tvPredmet.text = p0.value.toString()
            setText()
            if (switch) {
                val time: Int =
intent.getStringExtra("Vrijeme")!!.toFloat().toInt()
                startTimer((time * 1000 * 60), a)
                tvTimer.visibility = View.VISIBLE
                countDownTimer.start()
            }
        }
    })
```

Kod 3.9. *Metoda onCreate() aktivnosti u kojem se mijenjaju zadatci*

koji mogu doći na mjesto drugog zadatka u ispitu. Funkcija *setText()* (Kod 3.10.) dohvaća koliko tih varijacija trenutni zadatak ima te nasumično pomoću funkcije *Random.nextInt()* bira jedan zadatak. Nakon što je zadatak odabran, korisniku se prikazuje aktivnost sa odabranim zadatkom. Primjer jednog takvog odabranog zadatka se može vidjeti na Slici 3.14. Tokom rješavanja zadatka sa primjera se vidi da korisnik ima prikaz preostalog vremena koje se mijenja svake sekunde, ima prikaz kolegija kojeg rješava, tekst zadatka, redni broj zadatka kojeg rješava te *EditText* objekte za unos rješenja. Nakon unosa rješenja i pritiska gumba za zaključavanje odgovora se korisniku prikazuje pravo rješenje i pokreću se animacije kvačice ili križića uz *EditText* ovisno je li odgovor točan ili nije. Nakon rješavanja svih zadataka ili nakon isteka vremena se korisniku prikazuju *View* objekti jedan po jedan sa animacijom (Slika 3.15.). Korisnik vidi broj pitanja koji je odabrao za rješavanje, broj pitanja koje je riješio (ako istekne vrijeme broj pitanja za rješavanje i riješenih pitanja neće biti isti), postotak riješenosti te preostalo vrijeme rješavanja zadataka.



Slika 3.14. *Primjer zadatka*



Slika 3.15. *Primjer kraja ispita*

```
private fun setText() {
    ivSlikaZadatka.visibility = View.INVISIBLE
    ivRjesenje.visibility = View.INVISIBLE
    tvRjesenjeSlika.visibility = View.INVISIBLE
    tvPravoRjesenje.visibility = View.INVISIBLE
    boxTocno.visibility = View.INVISIBLE
    boxNetocno.visibility = View.INVISIBLE
    tvPrikazRjesenja.visibility = View.INVISIBLE
    etRjesenje2.text.clear()
    etRjesenje1.text.clear()
    etRjesenje.text.clear()
    kvacica.visibility = View.INVISIBLE
    kvacica1.visibility = View.INVISIBLE
    kvacica2.visibility = View.INVISIBLE
    krizic.visibility = View.INVISIBLE
    krizic1.visibility = View.INVISIBLE
    krizic2.visibility = View.INVISIBLE
    i++
    tvBrojZadatka.text = getString(R.string.tv_brojZadatka, i.toString())
    val stisnuto = tvPredmet.text.toString()
    when (i) {
        1 -> zadatak = "Prvi"
        2 -> zadatak = "Drugi"
        3 -> zadatak = "Treci"
        4 -> zadatak = "Cetvrti"
        5 -> zadatak = "Peti"}
    val baza = FirebaseDatabase.getInstance().getReference("Users").

child(FirebaseAuth.getInstance().currentUser!!.uid).child(stisnuto).child(zadatak)
baza.addListenerForSingleValueEvent(object : ValueEventListener {
 override fun onDataChange(p0: DataSnapshot) {
val gornjaGranica = p0.child("brojZadataka").value.toString().toInt()
val random = Random.nextInt(1, gornjaGranica + 1) // random biranje jednog
var t = ""
when (random) {
                        1 -> t = "t1"
                        2 -> t = "t2"
                        3 -> t = "t3"
                        4 -> t = "t4"
                        5 -> t = "t5"
                    }
            tvZadatak.text = p0.child(t).value.toString()
            when (stisnuto) {
                "Matematika" -> matematika(zadatak, t, p0)
                "Matematika 2" -> matematika2(zadatak, t)
                "Osnove elektrotehnike 1" -> osnoveElektrotehnike1(zadatak,t)
                "Fizika 1" -> fizika1(zadatak,t)
                "Osnove elektrotehnike 2" -> osnoveElektrotehnike2(zadatak,t)
                "Fizika 2" -> fizika2(zadatak,t)
            }
            Funkcije.tekst = p0.child("tekst").value.toString().toBoolean()
            bla = Funkcije.tekst
        }
    })
}
```

Kod 3.10. *Funkcija setText()*

Na osnovu odabranog predmeta poziva se funkcija oblika *imepredmeta()* koja kao parametar prima broj zadatka koji se rješava i broj nasumično odabrane varijacije zadatka (iznimka je funkcija *matematika()* koja dodatno prima *DataSnapshot* odabranog zadatka). Takva funkcija (Kod 3.11.) poziva odgovarajuću funkciju varijacije zadatka (npr. *oetDrugiPrvi()* sa Koda 3.12.). Pomoću parametra *editTekstovi* govori se programu koliko parametara rješenja ima, a samim time i koliko *EditText* objekata treba korisniku prikazati na tom zadatku.

```
private fun  osnoveElektrotehnike1(zadatak: String, t : String){
    if(zadatak == "Prvi"){
        when (t) {
             "t1" -> oetPrviPrvi()
             "t2" -> oetPrviPDrugi()
             "t3" -> oetPrviTreci()
             "t4" -> oetPrviCetvrti()
        }
    }
    if(zadatak == "Drugi"){
       when (t) {
            "t1" -> oetDrugiPrvi()
            "t2" -> oetDrugiDrugi()
            "t3" -> oetDrugiTreci()
            "t4" -> oetDrugiCetvrti()
       }
    }
    if(zadatak == "Treci"){
       when(t){
            "t1" -> oetTreciPrvi()
            "t2" -> oetTreciDrugi()
            "t3" -> oetTreciTreci()
            "t4" -> oetTreciCetvrti()
        }
    }
    if(zadatak == "Cetvrti"){
        when(t){
             "t1" -> oetCetvrtiPrvi()
             "t2" -> oetCetvrtiDrugi()
             "t3" -> oetCetvrtiTreci()
        }
    }
    if(zadatak == "Peti"){
        when(t) {
             "t1" -> oetPetiPrvi()
             "t2" -> oetPetiDrugi()
             "t3" -> oetPetiTreci()
        }
    }
    tvZadatak.visibility = View.VISIBLE
    progressBarZadatak.visibility = View.INVISIBLE
}
```

Kod 3.11. *Funkcija predmeta Osnove elektrotehnike 1*

Funkcija varijacije zadatka (Kod 3.12.) postavlja pripadajući tekst zadatka i nasumično pomoću funkcije *Random()* odabire vrijednosti parametara zadatka u rasponima prilagođenim za te parametre, postavlja tekst rješenja zadatka, određuje vidljivost *View* objekata u aktivnosti, zapisuje točna rješenja u Firebase bazu podataka, postavlja sliku zadatka (ako postoji), sliku rješenja (ako postoji) te pomoću parametra *editTekstovi* govori programu koliko parametara rješenja ima a samim time i koliko *EditText* objekata treba korisniku prikazati na tom zadatku.

```
private fun oetDrugiPrvi(){
    val c1 : Float =  Random.nextInt( 1,10).toFloat()
    val c2 : Float =  Random.nextInt( 1,10).toFloat()
    val c3 : Float =  Random.nextInt( 1,10).toFloat()
    val e1 : Float =  Random.nextInt(6,15).toFloat()
    val e2 : Float =  Random.nextInt(15,25).toFloat()
    tvZadatak.text = getString(R.string.oet1DrugiPrviZadatak,c1,c2,c3,e1,e2)
    val c21 = c1 / 1000000
    val c22 = c2 / 1000000
    val c23 = c3 / 1000000

    val u2 : Float= (e1*c21 - e2*c23)/(c21 + c22 +c23)
    val u1 = e1 - u2
    val u3 = e2 + u2
    ivSlikaZadatka.visibility = View.VISIBLE
    ivSlikaZadatka.setImageResource(R.drawable.oet1_drugi_zad1)
    tvPrikazRjesenja.text  = getString(R.string.oet1DrugiPrviRjesenje,u1,u2,u3)

    editTextovi = 3
    val baza = FirebaseDatabase.getInstance().getReference("Users")
        .child(FirebaseAuth.getInstance().currentUser!!.uid)
        .child("Osnove elektrotehnike 1").child("Drugi")
    baza.child("Rjesenje").setValue(u1)
    baza.child("Rjesenje1").setValue(u2)
    baza.child("Rjesenje2").setValue(u3)
    tvParametar.text = getString(R.string.oet210)
    tvParametar1.text = getString(R.string.oet211)
    tvParametar2.text = getString(R.string.oet212)
    tvParametar.visibility = View.VISIBLE
    tvParametar1.visibility = View.VISIBLE
    tvParametar2.visibility = View.VISIBLE
    etRjesenje.visibility = View.VISIBLE
    etRjesenje1.visibility = View.VISIBLE
    etRjesenje2.visibility = View.VISIBLE

}
```

Kod 3.12. *Funkcija varijacije zadatka oetDrugiPrvi()*

Do sada je obrađen dio koda koji je odgovoran za postavljanje teksta zadatka, postavljanje *EditText* objekata i *TextView* objekata za upisivanje rješenja, za prikazivanje vremena, imena kolegija i rednog broja zadatka. Preostao je dio koda koji mijenja tekst gumba (iz "PROVJERI RJESENJE" u "IDUĆI ZADATAK" ili pak u "ZAVRŠI"), dio koda koji određuje trenutak prikazivanja *View* objekata odgovornih za prikaz rješenja i vrijeme prikazivanja *check box* objekata za točno i netočno (ako ih treba prikazati), dio koda koji određuje je li rješenje tekstualno ili je slika, dio koda koji odlučuje jesu li rješenja utipkana u *EditText* objekte točna ili netočna (ako nema *check box* objekata) i paralelno sa time pokreće animacije ovisno je li rješenje točno ili netočno, te dio koda koji zapisuje ostvarene rezultate u Firebase bazu podataka. Sve je ovo napravljeno u jednom *onClickListeneru*. Funkcionalnost će biti objašnjena na primjeru prelaženja sa prvog zadatka na drugi zadatak.

```
btnDalje.setOnClickListener {

    if (!(editTextovi != 0 && provjeraEditTekstova(editTextovi))) {
    kvacica.visibility = View.VISIBLE
        kvacica1.visibility = View.VISIBLE
        kvacica2.visibility = View.VISIBLE
        krizic.visibility = View.VISIBLE
        krizic1.visibility = View.VISIBLE
        krizic2.visibility = View.VISIBLE
        val dobro = kvacica.drawable as AnimatedVectorDrawable
    val dobro1 = kvacica1.drawable as AnimatedVectorDrawable
    val dobro2 = kvacica2.drawable as AnimatedVectorDrawable
    val lose = krizic.drawable as AnimatedVectorDrawable
    val lose1 = krizic1.drawable as AnimatedVectorDrawable
    val lose2 = krizic2.drawable as AnimatedVectorDrawable
    dobro.reset()
    dobro1.reset()
    dobro2.reset()
    lose.reset()
    lose1.reset()
    lose2.reset()
    b++

    // ako je oznaceno i tocno i netocno
    if (boxTocno.isChecked && boxNetocno.isChecked && editTextovi == 0) {
        boxTocno.error = "Ili mene označi"
        boxNetocno.error = "ili mene."
        boxTocno.requestFocus()
        boxNetocno.requestFocus()
        b--
    }
```

Kod 3.13. *Uvod u onClickListener*

Prvo se određuju varijable *a*, koju je korisnik odredio u aktivnosti prije ove i predstavlja ukupan broj zadataka na ispitu, te varijabla *b* s pomoću koje se broji koliko je puta stisnut gumb. Prvobitno stanje varijable *b* je nula, a stanje varijable *a* se pretpostavlja da je dva. Klikom na gumb se povećava vrijednost varijable *b* za jedan, te ako su svi *View* objekti dobro popunjeni ostaje na toj vrijednosti (Kod 3.13.). Ako je ostatak dijeljenja sa dva različito od nula (*b* = 1, tj. različito je od nule), onda se pokreće dio koda koji je odgovoran za prikazivanje pravog rješenja korisniku. Ovdje se prvo provjerava kojeg je tipa rješenje (tekstualno ili slika) te se prikazuju određeni *View* objekti ovisno o tipu rješenja. Ovisnost o varijabli *editTextovi* prikazuju se *check box* objekti za točno i netočno ili se provjeravaju rezultati. Uzima se primjer funkcija varijacije zadatka *oetDrugiPrvi().* Varijabla *editTextovi* je postavljena na vrijednost tri (!= 0), stoga se izvršava kod gdje se dohvaćaju prava rješenja iz Firebase baze podataka i uspoređuju se sa rješenjima koje je korisnik upisao u *EditText* objekte. Ukoliko je rješenje koje je korisnik upisao u granicama odstupanja od pet posto, rješenje se uzima kao točno te se varijabla *točno* povećava za 0.33 (pošto zadatak ima tri varijable rješenja).

Korisnik ponovo pritišće gumb. Pretpostavlja se da je korisnik sve dobro ispunio, te se varijabla *b* povećava za jedan (*b* = 2). Varijabla *a* ima vrijednost dva te zahtjev da se aktivira kod kada su sva pitanja odgovorena nije ispunjen (*b* == 2**a* – 1) i zahtjev da se izvršava kod za provjeru točnosti zadnjeg zadataka nije ispunjen (*b* == 2**a*), stoga se izvršava kod na Kodu 3.14. jer je

```
} else if (b % 2 == 0) {
    if (((!boxTocno.isChecked && !boxNetocno.isChecked)||(boxTocno.isChecked &&
boxNetocno.isChecked)) && editTextovi == 0) {
        boxTocno.requestFocus()
        boxNetocno.requestFocus()
        b--
    } else {
        btnDalje.text = getString(R.string.btn_zadatak_ProvjeriRjesenje)
        if (boxTocno.isChecked && editTextovi == 0) {
            tocno++
        }
        if (editTextovi == 0) {
            boxNetocno.isChecked = false
            boxTocno.isChecked = false
        }
        setText()
    }
}
```

Kod 3.14. *Prelazak na slijedeći zadatak*

ostatak dijeljenja sa dva jednak nuli. Ukoliko je korisnik prikladno ispunio sve *EditText* objekte, mijenja se tekst gumba na "PROVJERI RJESENJE" i poziva se funkcija *setText()* koja opet postavlja tekst zadatka i ostale funkcionalnosti već prethodno objašnjene. Nadalje, ponovnim pritiskom gumba se povećava vrijednost varijable *b* za jedan (*b* = 3). Pretpostavlja se da je korisnik do sada sve dobro unosio. Sada se ispunjava uvjet za izvršavanje koda za provjeru točnosti zadnjeg zadataka (*b* == 2**a* -1) te se izvršava taj dio koda. Ovdje se dohvaćaju prava rješenja iz Firebase baze podataka i uspoređuju se sa rješenjima koje je korisnik prethodno upisao u *EditText* objekte. Ukoliko je rješenje koje je korisnik upisao u granicama odstupanja od pet posto, rješenje se uzima kao točno te se varijabla *točno* povećava za 0.33. Tekst gumba se postavlja na "ZAVRŠI". Nakon zadnjeg pritiska gumba (pretpostavlja se da je korisnik dobro upisao podatke u *View* objekte) mijenja se iznos varijable *brojPokusaja* koja je jedna od djece u bazi podataka u stvarnome vremenu, tj. on se povećava za jedan. Nadalje, računa se postotak točne riješenosti i kao takav se zapisuje kao jedan od argumenata instance klase *Entry* koja se sprema u polje jedne dimenzije *Entry* objekata te se nakon toga polje zapisuje u bazu podataka i prelazi se na drugu aktivnost.

3.8. Izrada grafičke analize

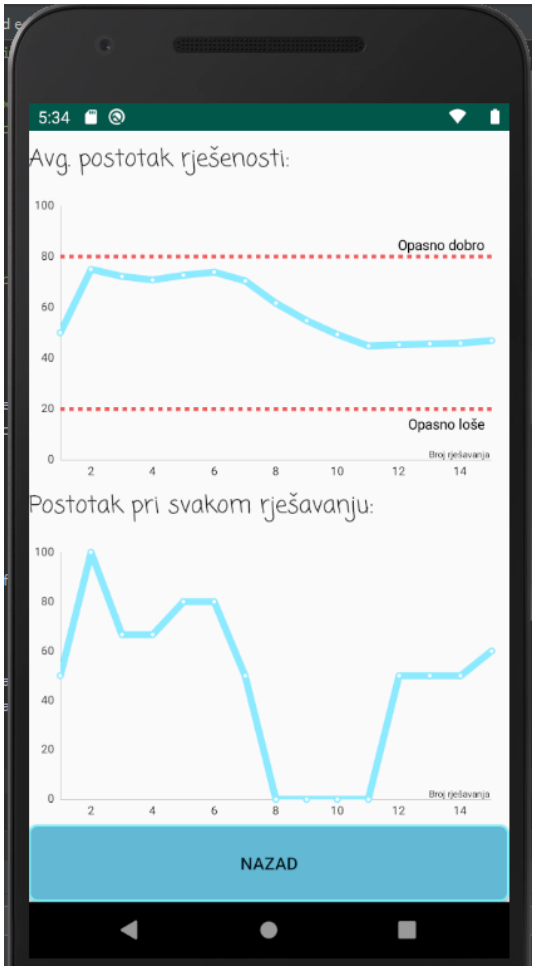
Izrada grafičke analize uključuje dodavanje biblioteka u *dependency* dio *build.gradle* datoteke, što znači mijenjanje *build.gradle* datoteke na razini aplikacije. U *build.gradle* datoteci na razini projekta dodaje se repozitorij prema Slici 3.16.

Slika 3.16. *Dodavanje LineChart objekta*

```
<com.github.mikephil.charting.charts.LineChart
    android:id="@+id/lineChart"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:visibility="invisible"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Slika 3.17. *Implementacija LineChart objekta*

Dodani repozitorij je sa GitHub platforme i implementacija njega je prikazana na Slici 3.17. Sve što je još potrebno za analizu jesu podaci iz Firebase baze podataka. Iz baze podataka u stvarnome vremenu se dohvaćaju članovi iz polja *Entry* objekata (u Firebase bazi podataka je to vrijednost djeteta *Polje*) jedan po jedan. Nakon dohvaćanja člana iz polja, zapravo se dohvaća njegova *y* vrijednost, tj. postotak riješenosti, i *x* vrijednost, koji označava broj pokušaja. Za jednu grafičku analizu potrebna je identična lista *Entry* objekata za zapis jer jedan od grafova zapisuje postotak riješenosti kod svakog broja rješavanja. Zadaća drugog grafa jest prikazivanje prosječnog postotka riješenosti nakon svakog pokušaja. Da bi se to realiziralo pravi se tzv. privremenu listu. Privremena lista se prvobitno popunjava sa uzetim *Entry* objektom. Ako je $x = 0$, tj. ako se zahtijeva prvi par iz liste, u privremenu listu se zapisuje *Entry(x,0)* pošto se pri povlačenju liste iz baze mora imat barem jedan član u listi. Pri povlačenju *Entry* objekta kojemu je $x = 1$, tj. broj pokušaja jednak jedinici, privremenoj listi se pomiče član na indexu 0 i na index 0 se stavlja *Entry*(*x,postotak*) (*postotak* je varijabla koja poprima vrijednost točnog postotka rješavanja kod povlačenja podatka iz Firebase baze podataka sa $x = 1$). Poslije prvog dodavanja, *postotak* se postavlja na nula i svaki se puta prelazi kroz cijelu privremenu listu te se postotci zbrajaju i dijele sa brojem članova privremene liste radi dobivanja prosječnog postotka rješavanja do toga broja pokušaja rješavanja. Provjerava se jesu li vrijednosti polja *yValues* i *xValues* prazne, ako jesu korisniku se pokazuje obavijest koja govori da do sada nema podataka o rješavanju ispita za odabrani predmet i usmjerava ga se na drugu aktivnost. S druge strane, ako nisu prazne, te vrijednosti se postavljaju na grafove prema Slici 3.18.



Slika 3.18. *Primjer grafičke analize*

4. ZAKLJUČAK

U završnom radu se proučavaju i opisuju korištene tehnologije kao što je Android operacijski sustav, Android Studio razvojno okruženje za razvoj aplikacija koje rade na Andorid operacijskom sustavu, Kotlin programski jezik s pomoću kojega se piše u Android Studio razvojnom okruženju te NoSQL baza podataka Firebase koja radi sa podacima u stvarnome vremenu. Programskim jezikom Kotlin se opisuje dohvaćanje podataka iz Firebase baze podataka, tj. trenutak dohvaćanja i samo dohvaćanje. Isto vrijedi i za autentifikaciju i spremanje podataka. Pomoću Kotlina se opisuje što se događa sa određenim komponentama, tj. *View* objektima ekrana nakon interakcije korisnika s istim, te se opisuje inicijalno stanje ekrana. Zaključno se može reći da se pomoću programskog jezika Kotlin (kao i kod upotrebe drugih jezika) proizvodi dinamičnost aplikacije, tj. njena sposobnost da se mijenja, da se išta događa. S druge strane Android Studio razvojno okruženje pruža objekte koje služe za stvaranje korisničkog sučelja i omogućava organizaciju i implementaciju istih pomoću XML koda te njihovu olakšanu realizaciju, pruža već prethodno definirane objekte koje predstavljaju kompleksnije realizacije prikaza *View* objekata (kao što je *Recycler View)* te pruža prethodno definirane metode za manipulaciju kreiranja i brisanja *View* objekata i njihovog crtanja na ekranu. Firebase se uvodi kao vanjska biblioteka u Android Studio i ima laganu implementaciju. Iskorištava se mogućnost lagane autentifikacije pomoću Firebase baze podataka i olakšana manipulacija podacima, što bi bez Firebase baze podataka bilo mnogo kompleksnije (pogotovo dio sa autentifikacijom, gdje dosta firmi koje u projektima implementiraju svoj sustav autentifikacije nailaze na dosta problema i gubitka vremena). Nakon izrade koda aplikacije i razrade iste se može primijetiti kroz poteškoće razvijanja Android aplikacije da se treba jako paziti na dohvaćanje podataka iz Firebase baze podataka i koje metode za dohvaćanje se trebaju upotrebljavati te gdje ćemo ih pozicionirati u kodu i što se u njih može staviti. Odgovor na tu nesigurnost proizlazi iz ovog projekta gdje se vidi da je najbolje staviti sve podatke bitne za sinkronizaciju u jednu od odabranih metoda za dohvaćanje podataka. Rezultat rada jest da dobivanje aplikacije pomoću koje korisnik može procijeniti svoje znanje u određenim kolegijima i pripremiti se za ispit i to se postiglo kombinacijom gore navedenih tehnologija. Aplikacija je jednostavna za korištenje i nema problema za zakašnjenjem prikaza korisničkog sučelja (što je jedan od glavnih zahtjeva modernih aplikacija), osim ako korisnik nema uspostavljenu konekciju sa internetom pri prijavi, no i tada se korisniku daje do znanje da se nešto krivo događa.

LITERATURA

[1] J., Kedveš, „Razvoj mobilnih aplikacija“, Travanj 2020. [Mrežno]. Dostupno na: http://mrkve.etfos.hr/pred/ozm/si/sem09.pdf

[2] G., Igaly, „Razvoj aplikacija za operacijski sustav android: Diplomski rad“, Travanj 2020. [Mrežno]. Dostupno na: https://zir.nsk.hr/islandora/object/pmf:3243/preview

[3] A., Shibly, „Android Operating System: Architecture, Security Challenges and Solutions“, Travanj 2020. [Mrežno]. Dostupno na: https://www.researchgate.net/publication/299394606_Android_Operating_System_Architecture_Security_Challenges_and_Solutions

[4] A., Harrison, J.Saxton, „Android Architecture“, Svibanj 2020. [Mrežno]. Dostupno na: http://meseec.ce.rit.edu/551-projects/fall2015/1-3.pdf

[5] I., Škorić, „Osnove izrade aplikacija za operativni sustav Android“, Svibanj 2020. [Mrežno]. Dostupno na: http://www.mathos.unios.hr/~mdjumic/uploads/diplomski/%C5%A0KO03.pdf

[6] A., Yadav, A., Vats, A., Nagpal, A., Yadav, „Indian Journal of Engineering, Volume 1: Dalvik – Vitual Machine, Discovery Journals“, Sibanj 2020. [Mrežno]. Dostupno na: http://www.discoveryjournals.org/engineering/current_issue/2012/A22.pdf

[7] „Meet Android Studio“, Svibanj 2020. [Mrežno]. Dostupno na: https://developer.android.com/studio/intro

[8] D., Griffiths, D., Griffiths, „Head First Kotiln“, O'Reilly Media, Inc., 2018.

[9] „Kotlin Language Documentation“, Lipanj 2020. [Mrežno]. Dostupno na: https://kotlinlang.org/docs/kotlin-docs.pdf

[10] „Learning firebase“, Travanj 2020. [Mrežno]. Dostupno na: https://riptutorial.com/Download/firebase.pdf

[11] „Parts of the Project’s UI“, Lipanj 2020. [Mrežno]. Dostupno na: https://developer.android.com/studio/intro/index.html

[12] „Configure your build:Overview“, Lipanj 2020. [Mrežno]. Dostupno na: https://www.geeksforgeeks.org/android-build-gradle

[13] „JSON:javascript object notation“, Srpanj 2020. [Mrežno]. Dostupno na: https://www.tutorialspoint.com/json/json_tutorial.pdf

[14] „Firabase database:ValueEventListener“, Srpanj 2020. [Mrežno]. Dostupno na: https://firebase.google.com/docs/reference/android/com/google/firebase/database/ValueEventListener

[15] „Firebase database:DatabaseReference“, Srpanj 2020. [Mrežno]. Dostupno na: https://firebase.google.com/docs/reference/android/com/google/firebase/database/DatabaseReference#setPriority(java.lang.Object)

SAŽETAK

Ovaj završni rad se bazira na izradi aplikacije na Android operacijskom sustavu pisane u programskom jeziku Kotlin koja služi za simulaciju ispita kolegija sa prve godine Fakulteta Elektrotehnike, Računarstva i Informacijskih tehnologija Osijek. Android aplikacija je povezana sa NoSQL bazom podataka Firebase koja zapisuje podatke baze u JSON formatu, tj. datoteci koja ima nastavak *.json*. Kroz rad je objašnjena funkcionalnost prelazaka sa jednog zadatka na drugi i bilježenje postignutih rezultata u Firebase bazu podataka. Također je obrađena izvedba grafičke analize postignutih rezultata pojedinih kolegija

Ključne riječi: Android, Kotlin, Realtime Database, JSON

ABSTRACT

This final paper is based on the making of an application on the Android operative system and it is written in the Kotlin programming language. Its purpose is to simulate exams from subjects on the first year of study on Faculty of Electrical Engineering, Computer Science and Information Technology Osijek. Android application is connected with the NoSQL database Firebase that stores data in a JSON format, in other words, it stores it in a file with the *.json* extension. Trough the work the functionality of going from one task in an exam to another and putting the data in the Realtime Database is explained. Furthermore, the implementation of graphical analysis for each subject is explained.

Key words: Android, Kotlin, Realtime Database, JSON

ŽIVOTOPIS

Davor Štajcer rođen je 26. Prosinca 1998. Godine u Zagrebu. Osnovnu školu je upisao I završio u Slavonskom Brodu. Nakon završetka osnovne škole upisuje srednju Tehničku školu u Slavonskom Brodu, smjer elektrotehničar, koju završava 2017. godine te u istoj godini upisuje preddiplomski studij elektrotehnike na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.

_____

Davor Štajcer