ANDROID KVIZ ZNANJA

SMOLČIĆ, DRAGAN

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijske tehnologije

DRAGAN SMOLČIĆ

Z A V R Š N I R A D

ANDROID KVIZ ZNANJA

Split, prosinac 2018.

SVEUČILIŠTE U SPLITU
SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijske tehnologije

Predmet: Objektno programiranje

Z A V R Š N I R A D

Kandidat: Dragan Smolčić

Naslov rada: Android kviz znanja

Mentor: Ljiljana Despalatović

Split, prosinac 2018.

Sadržaj

SAŽETAK

Android kviz znanja

Ideja ovog rada je opisati izradu Android aplikacije za kviz. Aplikacija omogućava korisniku da odabere proizvoljan broj kviz pitanja na koja odgovara izabirući jedan od ponuđenih 2 ili 4 odgovora (ovisno o vrsti pitanja). Nakon posljednjeg odgovorenog pitanja, aplikacija korisniku ispisuje broj točnih i postotak točnih odgovora. Broj i postotak za određenog korisnika se sprema u bazu podataka. Kod sljedećeg igranja istog korsnika, na zadnjem ekranu mu se ispisuje rezultat trenutnog kviza i top lista korisnika.

Tehnologije koje ovaj rad demonstrira su Android Studio, Firebase, te klijent-server model. Android Studio je službeni Googleov IDE za izradu Android mobilnih aplikacija i koristi Java programski jezik. Firebase je razvojna platforma koja pomaže u izradi mobilnih aplikacija, u ovom radu se koristi za autentikaciju korisnika (registracija i prijava), te za pohranu korisničkih podataka (rezultati kvizova). Na posljetku, klijent-server struktura se koristi u vidu servera koji s online JSON baze izvlači kviz pitanja i odgovore te ih prosljeđuje klijentu (naša Android aplikacija).

SUMMARY

Android trivia quiz

The idea behind this thesis is to describe the making of a quiz application for Android. The app lets users choose a random number of quiz questions, which are the answered choosing one of 2 or 4 answers (depending on the type of question). After the final answer, the app writes out the number and the percentage of the correct answers. The number and the percentage for a user is stored in a database. The next time the user plays the quiz, the end screen will show the current score and the top list of all the users.

The technologies demanstrated in this thesis are Android Studio, Firebase, and the client-server model. Android Studio is the Google's official IDE for making Android apps and it uses Java programming language. Firebase is the development platform that helps with app development, in this thesis it is used for user authentication (registration and login) and user data storage (quiz results). Lastly, the client-server structure is used in the way of the server which pulls quiz questions and answers from an online JSON database and sends it to the client (our Android application).

1. Uvod

Mobilna aplikacija (*engl. mobile app*) je kompjuterski program koji je napravljen sa svrhom da se koristi na pametnim telefonima, tabletima ili pametnim satovima. Od web aplikacija ih razlikuje to što se web aplikacijama pristupa preko browsera, a mobilne aplikacije se pokreću izravno. Prve mobilne aplikacije se pojavljuju 2008. godine, a do danas su postale najrasprostranjeniji način pristupanja internetu putem mobitela. Najčešće ih se instalira putem app storea (Apple) ili Google Playa (Android). Aplikacije se na Android sustave također mogu instalirati manualno, putem APK (Android application package) datoteka. Dva trenutno dominantna operativna sustava za pametne telefone su Appleov iOS i Googleov Android. Za izradu ove aplikacije izabrao sam operativni sustav Android.

Aplikacija koja će biti opisana u radu je aplikacija za kviz znanja. Korisnik može napraviti račun koristeći svoju email adresu i željenu lozinku, koji se spremaju u Firebase bazu podataka. Korisnik može odabrati određen broj pitanja na koja će odgovarati, a na kraju mu se ispisuje broj točnih odgovora i top lista korisnika.

U drugom poglavlju opisan je operativni sustav Android i struktura samog sustava. Treće poglavlje opisuje strukturu Android aplikacije u općenitom smislu, s naglaskom na dijelove aplikacije koje nalazimo u ovom radu. Četvrto poglavlje opisuje tehnologije korištene za izradu same aplikacije, kao i tehnologije korištene za spajanje aplikacije s bazom podataka. Peto poglavlje opisuje izradu same aplikacije, pobliže objašnjava pojedine aktivnosti i klase, kao i spajanje s Firebaseom. U šestom poglavlju nalazi se zaključak.

2. Android OS

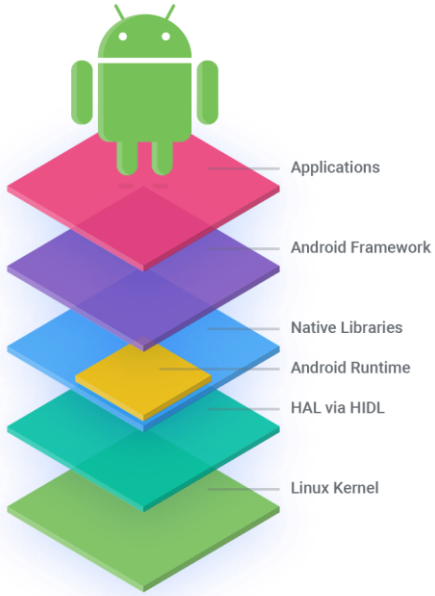
Android [1] je operativni sustav za mobilne uređaje, tablete, pametne satove i pametne televizore, razvio ga je Google. Sustav je baziran na modificiranoj verziji Linux kernela. Inicijalno ga je 2003. razvijala kompanija Android inc., koju je 2005. kupio Google. Android je javnosti inicirano predstavljen 2007., a prva komercijalna verzija je puštena na tržište 2008. Od tada smo vidjeli mnogobrojne verzije sustava, a trenutna verzija, 9 Pie, je izdana u kolovozu 2018.

Najveće prednost Androida naspram drugih sustava je što je potpuno besplatan, a kôd mu je otvoren. Jezgra Android sustava nosi ime Android Open Source Project (AOSP) i izdana je pod Apache licencom. To je naročito zanimljivo s razvojne strane zbog toga što svatko može preuzeti njegov kôd, koristiti ga, modificirati i prilagođavati. Ovo je također jedan od glavnih razloga zašto većina vodećih proizvođača pametnih uređaja (Huawei, Sony, Samsung, HTC....) koristi upravo Android.

Cilj Androida, kao projekta otvorenog kôda, je izbjeći takozvani prekid veze, gdje bi jedna kompanija mogla ograničiti druge i spriječiti inovacije tako što bi ograničila korištenje nekog dijela softvera. Kao što svatko može sudjelovati i dati svoj doprinos u kreiranju AOSP-a, tako svatko može sudjelovati i stvaranju i modificiranju dokumentacije za isti. Svačiji doprinos je dobrodošao i potiče se od strane Androida.

2.1 Arhitektura Android OS-a

Android je softverski stog (*eng. software stack*) sačinjen od 5 glavnih cjelina, zamišljen kako bi radio na više vrsta različitih uređaja. [2]



Slika 1. Stog operativnog sustava Android

2.1.1 Linux jezgra (*eng kernel*)

Osnova Android platforme je Linux kernel. Kernel je jezgra operativnog sustava i on upravlja procesorom, memorijom, perifernim uređajima i zadužen je za prevođenje zahtijeva perifernih uređaja u komande razumljive procesoru. Jedan od primjera je Android Runtime (ART), koji koristi Linux kernel za funkcionalnosti poput upravljanja memorijom. Još jedna prednost Linux kernela su ključna sigurnosna svojstva.

2.1.2 Razina apstrakcije (*eng Hardware Abstraction Layer (HAL)*)

Razina apstrakcije u računalima je logička sekcija kôda koja služi kao veza između fizičkog hardvera računala i njegovog softvera. Glavna zadaća sloja apstrakcije je da „sakrije“ različite hardverske arhitekture računalu i predstavi ih kroz uniformno sučelje koje računalo može razumjeti. Taj sloj postoji kako bi se izbjegla potreba za modificiranjem jezgre računala za svaku ulazno izlaznu komponentu koju će ono koristiti. Počevši s Android verzijom 8.00 pa nadalje, uvedene su dva sloja apstrakcije koje koriste svi noviji Android uređaji, to su takozvani

Binderized HALs i Passthrough HALs. Svaka nova verzija Adroida mora podržavati jedu od ovih apstrakcija.

2.1.3 Native Biblioteke

Native biblioteke Android sustava pisane su u C i C++ programskim jezicima. Nativni kôd je onaj kôd koji je moguće direktno prevesti u naredbe razumljive procesoru, što je slučaj s C i C++ jezicima, a nije slučaj s Javom, koja mora koristiti posrednike, poput virtualnog stroja (eng. *Java virtual machine*). Nekad je poželjno da dijelovi kôda Android aplikacije budu u C/C++ jeziku, jer to dovodi do bržeg izvođenja. Tako da na raspolaganju imamo biblioteke napisane u tim jezicima koje po potrebi možemo iskoristiti. Za korištenje nativnih biblioteka koristimo NDK (eng. *Native Development Kit*).

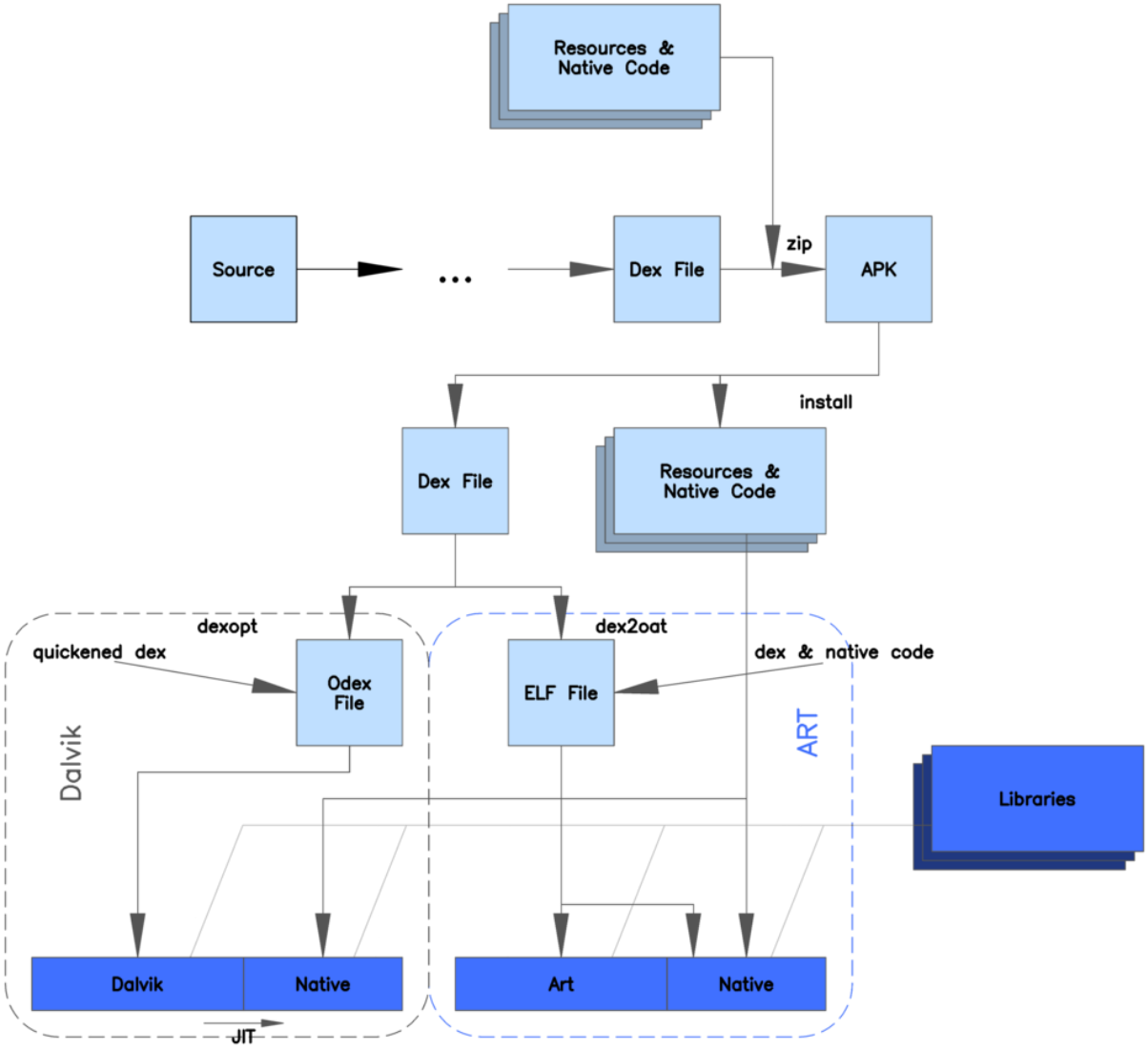
Prednosti korištenja NDA-a su sljedeće:

- Poboljšanje performansi.
- Mogućnost korištenja C/C++ biblioteka.
- Mogućnost razvijanja višeplatformskih aplikacija (Android i iOS).
- Mogućnost low-level programiranja (Upravljanje hardverskim komponentama i slično).

2.1.4 Android okruženje (eng. *Android runtime*)

Na istom sloju s native bibliotekama nalazimo i Android biblioteke. Android biblioteke na ovom sloju pisane su u Java programskom jeziku, te ih je potrebno prevoditi kako bi se mogle izvoditi na procesoru. Runtime koji se trenutno koristi na Android uređajima je ART, kao zamjena za ranije korišteni Dalvik. ART izvodi prevođenje byte kôda aplikacije u native instrukcije. Dalvik je radio tako da prevodi male segmente kôda (takozvano „just in time“ prevođenje), dok ART prevodi čitave aplikacije prije njihovog izvođenja, prilikom instalacije (takozvani „ahead of time“ tip prevođenja). Tako se poboljšala sveukupna učinkovitost rada,

ubrzalo se izvođenje i smanjila potrošnja baterije. Android 7.0 Nougat je uveo JIT prevoditelj koji radi usporedno s ART-om i pomaže mu poboljšati performanse.



Slika 2. Usporedba rada Dalvik i ART kompajlera

Kao što je vidljivo iz slike, najveća novost ART-a je što su odex datoteke zamijenjene ELF (Executable and Linkable Format) datotekama, koje mogu biti izvršavane u native kôdu, dok je odex datoteke bilo potrebno prevoditi putem JIT-a.

2.1.5 Okvir (eng. *framework*)

U ovom sloju nalaze se skupovi API-ja koji omogućavaju brzu i jednostavnu izradu aplikacija za Android. Sloj je razvijen u Java programskom jeziku i to je sloj koji koriste aplikacije u Android operativnom sustavu.

Svaka Android aplikacija sastoji se od:

• Aktivnosti (eng. *activity*) - To su programi s kojima korisnik vrši interakciju.

• Usluga (eng. *services*) - To su programi koji se izvršavaju u pozadini ili vrše nekakvu funkciju za drugu aplikaciju.

• Prijemnika za emitiranje (eng. *broadcast receiver*) - Programi koji hvataju informacije bitne za aplikaciju.

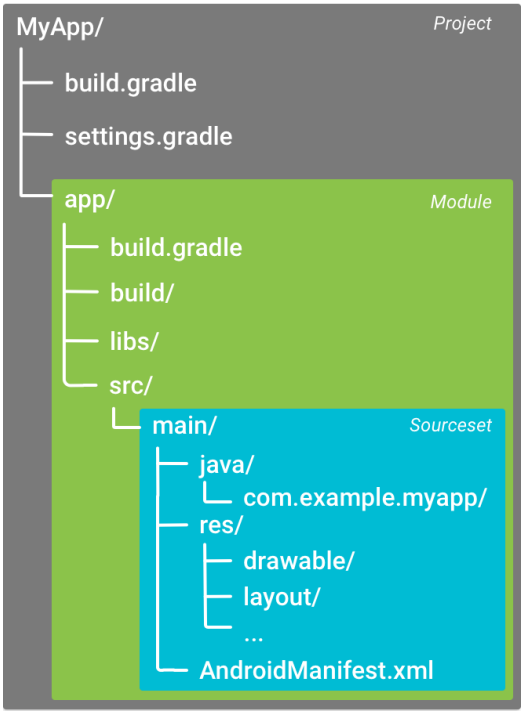
2.1.6 Aplikacije

Konačno, u ovom sloju nalaze se korisničke aplikacije. To su aplikacije poput onih koje dolaze sa samim uređajem, npr. kalendar, sat, foto aparat, galerija slika, kao i one koje možemo instalirati s Google Play trgovine (eng. *Google Play Store*). Na istom sloju se nalaze i aplikacije koje i samo možemo napisati, poput one koja se obrađuje u ovom radu. Aplikacije na ovom sloju su dostupne korisniku, ali isto tako i programeru. Na primjer, ako želimo iz naše aplikacije obaviti telefonski poziv, ne trebamo isprogramirati aplikaciju za pozivanje, možemo iskoristiti aplikaciju koja već postoji na uređaju i obaviti poziv koristeći nju.

3. Struktura Android aplikacije

Android PacKage (APK) je format instalacijske datoteke operativnog sustava Android. Da bi se napravila APK datoteka, program se prvo kompajlira, zatim se njegovi dijelovi spremaju u jednu datoteku. APK datoteka sadrži sav programski kôd, AndroidManifest.xml datoteku, resurse i certifikate potrebne za rad aplikacije. APK je arhivni tip datoteke, poput ZIP formata. APK datoteke mogu imati proizvoljno ime, jedino je bitno da završavaju s „.apk“. APK datoteke mogu se instalirati direktno na uređaj ili preko službene Google Play trgovine.

Android projekt ima definiranu strukturu po kojoj se slažu datoteke i treba sadržavati neke standardne datoteke da bi funkcionirao kako treba.



Slika 3. Struktura Android projekta

Osim nekih sitnih intervencija u project build.gradle i app build.gradle, većina stvari koje trebamo napraviti za ovakav projekt nalazi se u source direktoriju projekta, pa ćemo se u ovom dijelu rada bazirati na najbitnije datoteke source direktorija.

3.1 AndroidManifest.xml

Svaki Android app projekt mora sadržavati AndroidManifest.xml datoteku. Ona mora biti nazvana točno tako i mora se nalaziti u source direktoriju projekta. Manifest opisuje esencijalne informacije o našem projektu alatima Android builda, Android operativnom sustavu i Google Playu.

Između ostalog, bitne informacije koje manifest mora sadržavati su sljedeće:

- Ime app paketa. To služi alatima za izradu builda da lociraju pojedine segmente kôda kad buildaju projekt. Kod „pakiranja“ aplikacije, ti alati mijenjaju ovu vrijednost s identifikacijskim brojem aplikacije koji se nalazi u Gradle build datotekama. To služi kao unikatna identifikacija aplikacije u sustavu i na Google Playu.
- Sve komponente aplikacije. Ovo uključuje sve aktivnosti, usluge, pružatelje sadržaja i primatelje namjera.
- Dopuštenja koja aplikacija treba kako bi pristupila zaštićenim dijelovima sustava ili drugima aplikacijama.
- Hardverske i softverske zahtjeve aplikacije, što utječe na to na koje uređaje možemo instalirati aplikaciju.

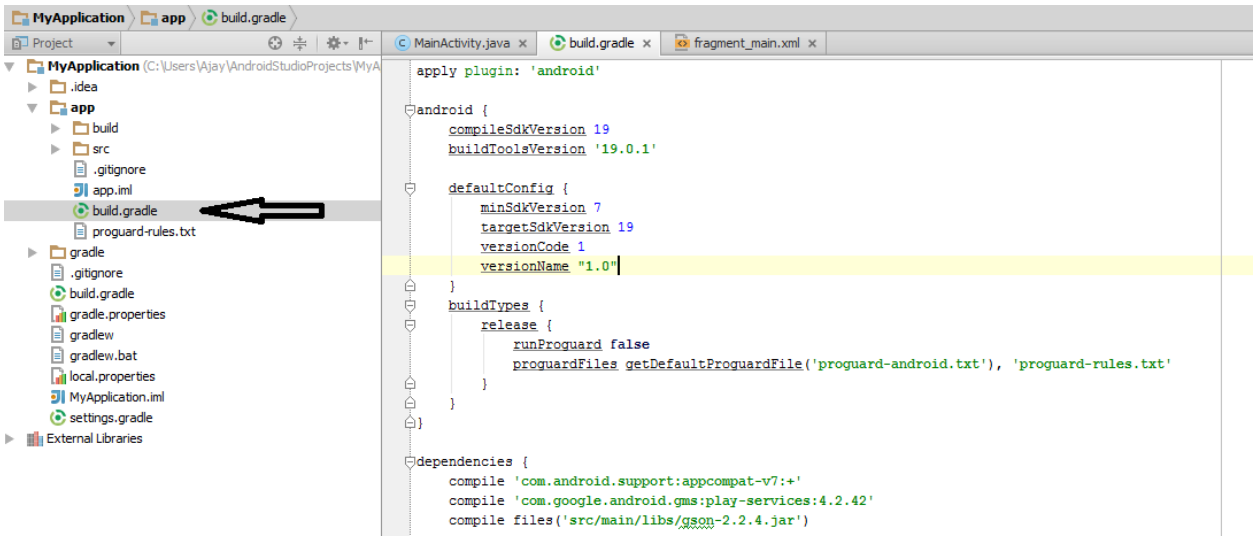
Kada za izradu aplikacije koristimo Android Studio, on nam automatski generira Android Manifest.xml datoteku i većinu osnovnih elemenata manifesta.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.homeandlearn.ken.simpleactivityexample">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="SimpleActivityExample"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".SecondActivity"
            android:parentActivityName=".MainActivity">
        </activity>
    </application>

</manifest>
```

Slika 4. Primjer AndroidManifest.xml datoteke

3.2 Gradle

Gradle je build sistem koji je uključen u Android Studio. Kad kreiramo novi projekt unutar Android Studia, Gradle build skripte se automatski kreiraju unutar našeg projekta. Klikom na „Run“ unutar našeg projekta, Android Studio pokreće odgovarajuće Gradle zadaće koje onda buildaju našu aplikaciju. Gradle je alat koji u našem projektu naš kôd prevodi u već spominjane DEX ili ELF datoteke, koje se onda u native kôdu izvode na procesoru. Gradle također obavlja sinkronizaciju s vanjskim bazama podataka koje projekt koristi (npr. Firebase). Unutar Gradle kôda je potrebno implementirati potrebne usluge koje omogućavaju spajanje našeg projekta s bazom. Nakon svake promjene unutar Gradle kôda, potrebno je naš projekt sinkronizirati da bi ispravno radio.

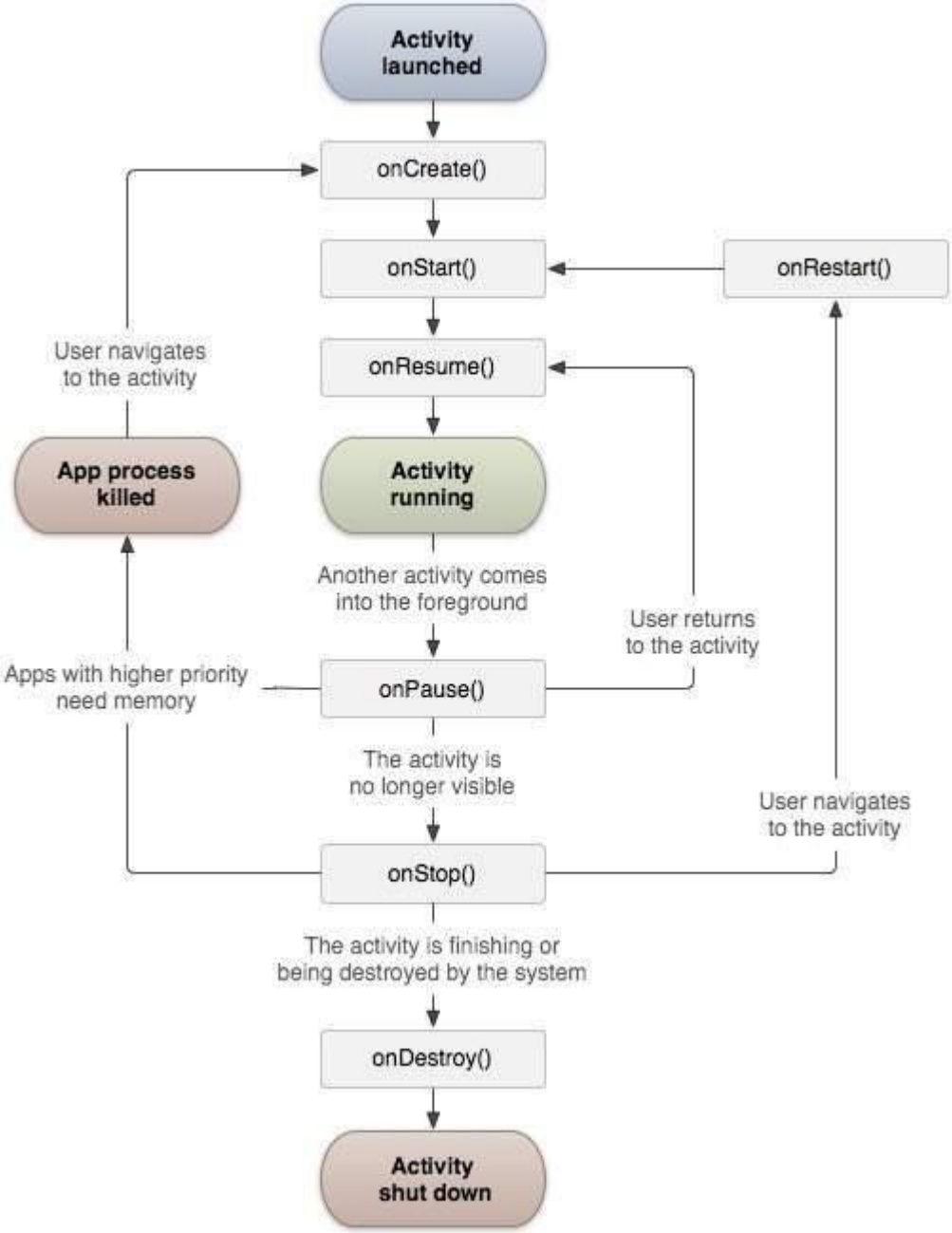


Slika 5. Primjer Gradle datoteke unutar Android Studija

3.3 Aktivnost (eng. *activity*)

Aktivnost u Android projektu predstavlja jedan ekran u našoj aplikaciji. Java.Android klasa je podklasa klase ContextThemeWrapper. Na isti način kako u C/C++ ili Javi postoji *main()* funkcija kojom program započinje, tako kod Android aktivnosti postoji *onCreate()* callback metoda koja se prva izvršava kod svakog otvaranja aktivnosti. Osim metode *onCreate()*, postoji 6 drugih metoda pomoću kojih možemo izvršiti kompletan životni ciklus jedne aktivnosti [3].

- **onCreate()** - Prva callback metoda, izvršava se kad pokrenemo aktivnost.
- **onStart()** - Ova metoda se poziva kad aktivnost postane vidljiva korisniku.
- **onResume()** - Ova metoda se poziva kad korisnik počne interakciju s metodom.
- **onPause()** - Ne prima nikakav unos od strane korisnika, ona se poziva kad želimo. pauzirati trenutnu aktivnost i pokrenuti neku drugu.
- **onStop()** - Poziva se kad aktivnost više nije vidljiva.
- **onDestroy()** - Poziva se netom prije nego je aktivnost uništena.
- **onRestart()** - Poziva se kad se aktivnost ponovno pokrene nakon što je bila zaustavljena.



Slika 6. Životni ciklus Android aktivnosti

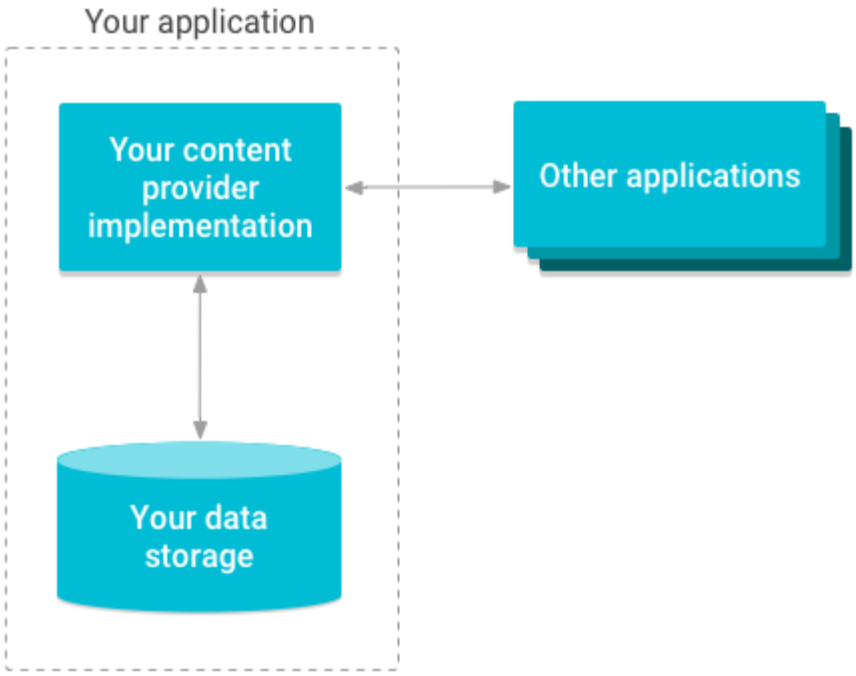
3.4 Usluga (eng. *service*)

Usluga je komponenta koja se pokreće u pozadini i nema direktnu interakciju s korisnikom aplikacije. Budući da usluga nema sučelje, ona nije povezana sa životnim ciklusom

aktivnosti. Usluge koristimo za potencijalno duge operacije (skidanje datoteka s interneta, procesuiranje podataka...). Usluge imaju veći prioritet nego neaktivne ili nevidljive aktivnosti, stoga postoji mala vjerojatnost da će Android zaustaviti njihovo izvršavanje. Usluge je moguće podesiti tako da se ponovno aktiviraju kad se oslobodi dovoljno resursa, ako ih sustav zaustavi. Također je moguće podesiti da usluge imaju jednak prioritet kao i aktivnosti, u kojem slučaju postavljamo nekakve vidljive notifikacije. To se često koristi kod usluga koje prikazuju videozapise ili glazbu.

3.5 Pružatelj sadržaja (eng. *content provider*)

Pružatelj sadržaja omogućava našoj aplikaciji da pristupa podacima koje je sama spremila, podacima koje je spremila neka druga aplikacija i daje nam način komunikacije s drugim aplikacijama. Korištenje pružatelja sadržaja nam daje mnoge prednosti, najvažnija od kojih je ta da nam omogućava da na siguran način iz jedne aplikacije pristupamo drugoj i mijenjamo njene podatke. Ako naša aplikacija koristi neku bazu podataka kojoj samo ona ima pristup i ne želimo dopustiti drugim aplikacijama da pristupaju našoj bazi, uspostavljanjem pružatelja sadržaja možemo omogućiti drugoj aplikaciji da posredno promjeni podatke koje mi dopustimo.



Slika 7. Prikaz komunikacije između naše aplikacije i druge aplikacije putem pružatelja usluga

3.6 Primatelj namjera (eng. *broadcast receiver*)

Primatelj namjera je komponenta koja odgovara na obavijesti koje su odaslane cijelom operativnom sustavu. Obavijesti mogu doći iz operativnog sustava, poput obavijesti da je razina baterije niska, da se neki prozor zatvorio, da je uhvaćena slika ekrana... Aplikacije također mogu odaslati obavijesti, recimo da je neki proces skidanja podataka gotov i da ih mogu koristiti. Primatelj namjera je implementiran kao potklasa klase BroadcastReceiver, a svako odašiljanje namjera se vrši pomoću objekta klase Intent.

3.7 Namjera (eng. *intent*)

Namjera je apstraktni opis operacije koja se treba obaviti. Možemo je koristiti sa *startActivity*, ako želimo pokrenuti neku aktivnost ili *broadcastIntent*, ako želimo odašiljati

obavijest svakom primatelju namjera i *Context.startService(Intent)* ili *Context.bindService(Intent, ServiceConnection, int)* ako želimo komunicirati s nekom uslugom. Najznačajnija primjena namjera je startanje aktivnosti. Dvije primarne informacije jedne namjere su [4]:

- **Akcija –** Akcija koju namjera treba izvršiti (npr. ACTION_VIEW, ACTION_EDIT, ACTION_MAIN).
- **Podaci –** Podaci s kojima namjera raspolaže (npr. osobni podatak neke osobe iz baze podataka).

4. Tehnologije korištene u izradi aplikacije

U ovom poglavlju ćemo se pobliže pozabaviti konkretnim tehnologijama koje su korištene i izradi same aplikacije. Aplikacija je izrađena u Android Studiju koristeći programski jezik Java. Korisnička registracija i prijava, kao i baza podataka u koju se spremaju korisnički rezultati, implementirana je koristeći Firebase alate. Kako zamjena za server služi nam Firebase, a klijent strana programirana je u Android Studiju.

4.1 Programski jezik Java

Java je objektno orijentiran programski jezik originalno objavljen u studenom 1995. godine, proizvela ga je tvrtka Sun Microsystems. Jezik je izrađen s namjerom da se može izvoditi na što više platformi, bez potrebe da se rekompajlira. Java programi se prevode u bytecode koji može biti pokrenut uz pomoć Java virtualnog stroja, bez obzira na arhitekturu procesora. Jezik koristi jednostavnu sintaksu i odlikuje ga velik broj bogatih biblioteka koje uvelike olakšavaju programiranje. Java je trenutno jedan od najpopularnijih programskih jezika, pogotovo za klijent-server web aplikacije.

4.2 Firebase platforma

Firebase [5] je razvojna platforma koja se koristi za razvijanje mobilnih i web aplikacija. Razvijena je od strane tvrtke Firebase, Inc 2001. godine, a kupio ju je Google 2014. godine. Trenutno raspolaže s 18 proizvoda koje koristi 1.5 milijuna aplikacija. Firebase nudi funkcionalnosti poput analitike, baza podataka, slanja poruka, analize bugova, što nam nudi način za brzu reakciju i rješavanje problema i brz pristup korisnicima. Firebase ima Googleovu pozadinu i objedinjuje više različitih usluga pod jednim krovom, što nam omogućuje lakšu integraciju i komunikaciju između različitih vrsta usluga.

Neki od najzanimljivijih alata, u našem kontekstu, su:

- **Firebase Auth** - Firebase auth usluga koja nam omogućuje autentikaciju korisnika koristeći samo klijent stranu kôda. Podržava i implementaciju za login

preko većine najvećih društvenih platformi (Facebook, GitHub, Twitter, Google...).

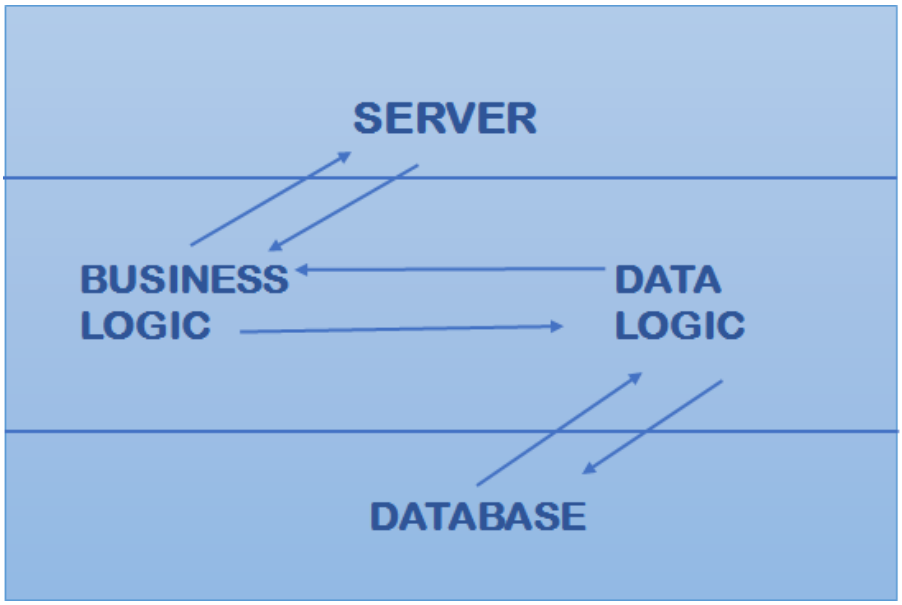
- **Realtime Database** – Realtime database je usluga koja nam omogućava korištenje baze podataka koju možemo ažurirati u realnom vremenu, iz kôda naše aplikacije. Firebase nudi biblioteke koje omogućavaju integraciju s Android, iOS, JavaScript, Java, Objective-C, Swift i Node.js aplikacijama. Programeri koji koriste ovu bazu imaju mogućnost osigurati svoje podatke postavljajući sigurnosna pravila za svoju bazu na serverskoj strani.

4.3 Klijent-server arhitektura

Klijent server struktura je raspodjela programa tako da razdvaja zadaće pružatelja usluge (server) i onoga koji traži uslugu (klijent). Server radi tako da vrti program koji dijeli svoje resurse s klijentom. Klijent se u određenom trenutku spaja na server i čita podatke koje server odašilje. Dakle, klijent je taj koji inicira komunikaciju, a server ima otvoren kanal i čeka da se klijent spoji.

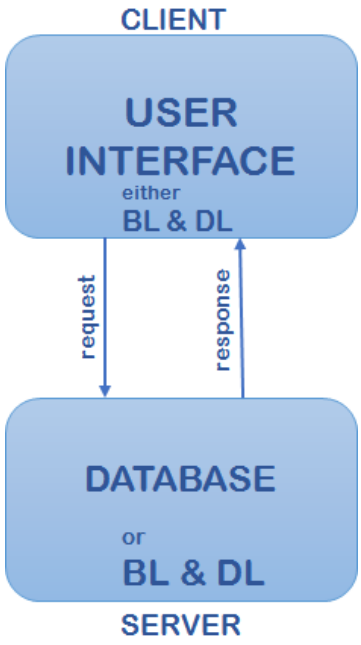
Vrste klijent-server arhitekture [6]:

- **Jednoslojna arhitektura –** U ovakvom tipu strukture korisničko sučelje, logika i pristup bazi su implementirani na istom računalu. Ovakav vid implementacije je najjeftiniji, ali nije efikasan, jer dovodi do nepotrebnog ponavljanja istih zadaća.



Slika 8. Jednoslojna klijent-server arhitektura

- **Dvoslojna arhitektura –** Kod ovog tipa arhitekture, korisničko sučelje je na klijent strani, a baza podataka na serverskoj strani. Logika može biti na klijent ili server strani, ali mora biti jedno od toga. Nije dobro da dio logike programa bude na jednom, a dio na drugom. Ako je logika na klijenti, onda govorimo o arhitekturi tankog servere, u suprotnom o arhitekturi tankog klijenta.



Slika 9. Dvoslojna klijent-server arhitektura

- **Troslojna arhitektura –** Kod ovakvog tipa arhitekture koristi se posrednik. Zahtjev klijenta prvo ide kroz posrednika, zatim server odgovor prvo šalje posredniku, a posrednik ga prosljeđuje klijentu. Kod ovakvog sustava sva logika nalazi se na posredniku, dok je na klijentu samo korisničko sučelje, a na serveru baza podataka. Ovakav tip arhitekture koristi se za velik broj korisnika, zbog toga što poboljšava brzinu izvođenja. Nedostatak je što je skup.

Slika 10. Troslojna klijent-server arhitektura

4.4 Android Studio

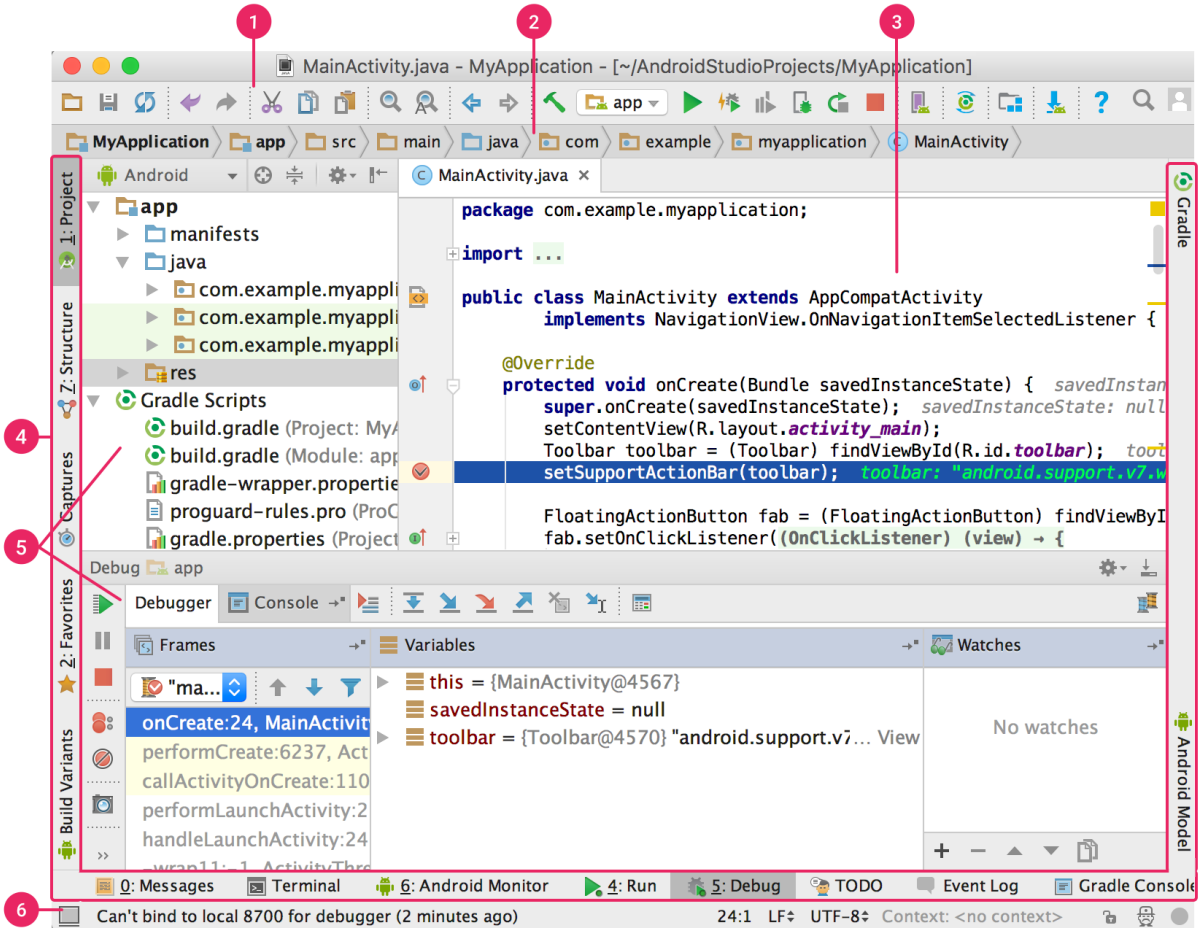
Android studio je službeno integrirano razvojno okruženje za Android operativni sustav. Napravljen je kao zamjena za Eclipse Android Development Tools koji je prije toga bio glavni IDE za nativni Android razvoj. Prvi stabilni build Android Studija izdan je u prosincu 2014, počevši s verzijom 1.0. Trenutna verzija, 3.2.1. izdana je u listopadu 2018.

U najnovijoj verziji, dostupno je sljedeće:

- Podrška za buildanje projekata bazirana na Gradleu.
- Refaktoriranje kôda na način specifičan za Android.
- Lint alati namijenjeni za hvatanje grešaka.
- Integracija ProGuard alata za optimizaciju kôda.
- Čarobnjak baziran na predlošcima za kreiranje izgleda Android aplikacija.
- Bogat editor layouta koji omogućava *drag and drop* postavljanje GUI elemenata.

- Podrška za kreiranje Android Wear aplikacija (verzija Androida za pametne satove).
- Ugrađena podrška za Google Cloud Platform koja omogućuje integraciju za Firebase Cloud Messaging i Google App Engine.
- Android Virtual Device emulator koji nam omogućava da pokrećemo i otklanjamo neispravnosti naše Android aplikacije.

Android Studio nam omogućava jednostavan pregled strukture Android projekta. S lijeve strane programa nalazi se prozor preko kojeg možemo pristupati svim razinama našeg projekta, a sama struktura se automatski kreira kad kreiramo novi projekt. Ostatak glavnog prozora Android Studija podijeljen je u nekoliko logičkih cjelina prikazanih na sljedećoj slici.



Slika 11. Prikaz glavnog prozora unutar Android Studija

1. Ova alatna traka omogućava nam pristup mnogim funkcijama, uključujući pokretanje i debagiranje našeg projekta.
2. Navigacijska traka koja nam pomaže da se krećemo kroz različite razine našeg projekta. Pruža kompaktniji pogled na strukturu projekta od onog u projektnom prozoru.
3. Prozor za pisanje i mijenjanje našeg kôda.
4. Traka za upravljanje prozorima koja nam omogućava da sakrijemo i otkrijemo određene prozore.
5. Prozor za alate koji nam omogućava pristup određenim zadaćama, poput upravljanja projektom, kontrole verzije, traženje...
6. Statusna traka prikazuje status našeg projekta, poruke, upozorenja...

4.5 JSON datoteke

JSON datoteka je datoteka koja sprema jednostavne strukture podataka i objekte u JavaScript Notation formatu. JSON datoteke su tekstualne, ne zauzimaju mnogo memorije, čitljive ljudima i mogu se kreirati koristeći programe za pisanje teksta (poput Notepada). Originalno zamišljen kao podskup JavaScripta, danas je format neovisan o jeziku. Originalno je proizveden iz želje da se ostvari mogućnost *real-time* komunikacije između servera i browsera bez potrebe za instaliranjem Flash plugina ili Java appleta, koje su se koristile u 2000-tim godinama.

Tipovi podataka koje JSON podržava su sljedeći:

- Number – Brojevni format koji podržava decimalne brojeve
- String – Sekvenca 0 ili više *Unicode* znakova. Stringovi se označavaju tako da se okruže duplim navodnicima.
- Boolean – Ima vrijednost *true* ili *false*.

- Array – Poredana lista 0 ili više vrijednosti, svako od kojih može biti bilo koji tip podatka. *Array* se označava uglatim zagradama, a elementi unutar Arraya se razdvajaju zarezom.
- Object - Neporedan skup podataka koji se sastoje od imena i vrijednosti. Ime podatka mora biti string. Objekti se označavaju vitičastim zagradama, a elementi se razdvajaju zarezom.
- Null – Prazna vrijednost, označava se s *null*

JSON datoteke imaju nastavak „.json“

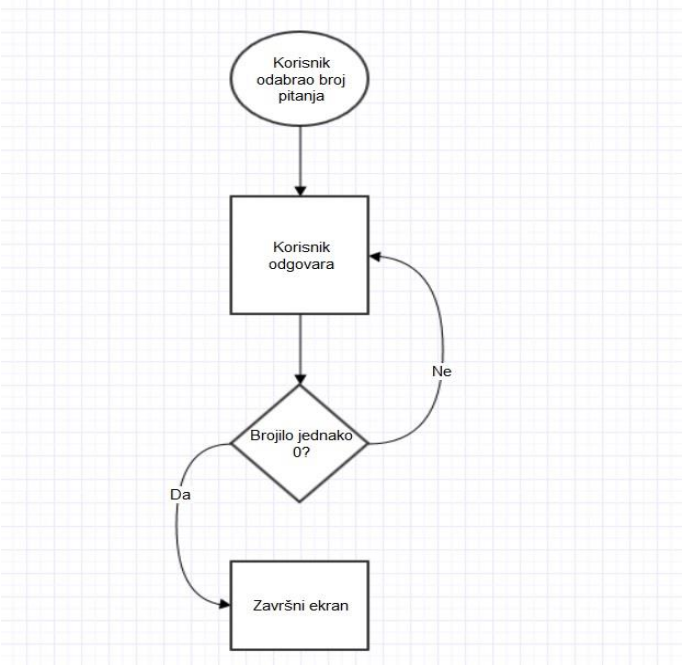
Slika 12. Primjer izgleda JSON datoteke

5. Opis praktičnog rada

Tema ovog rada je izrada Android aplikacije za kviz. Ideja aplikacije je omogućiti korisniku da odabere nasumičan broj pitanja na koja će odgovarati. Korisnik odgovara izabirući jedan od ponuđenih 2 ili 4 odgovora, ovisno o tipu pitanja. Nakon što odgovori na sva pitanja, aplikacija ispisuje koliko je od toga točnih, te ispisuje ukupan postotak točnih odgovora. Također, aplikacija ispisuje listu top 5 natjecatelja, poredanih po postotku točnih odgovora (od najvećeg prema najmanjem). Imajući to u vidu, bilo je potrebno implementirati registraciju i prijavu korisnika, kao i bazu podataka u kojoj se čuvaju korisnički podaci. Također, bilo je potrebno implementirati vezu s udaljenim serverom (preko stranice opentdb.com), s kojega aplikacija dovlači pitanja i odgovore u obliku JSON datoteke.

5.1 Osnovni tijek rada

Osnovni tijek rada aplikacije odvija se tako da korisnik odabire jedan od ponuđenih brojeva pitanja (10 i 15) ili unosi proizvoljan broj u polje. Kviz zatim započinje i korisniku se prikaže onoliki broj pitanja koliki je odabrao, a odabir se posprema u brojilo. Nakon svakog odgovorenog pitanja, brojilu se oduzima 1. Nakon što je zadnje pitanje odgovoreno (brojilo = 0), aplikacija otvara završni ekran, koji ispisuje broj točno odgovorenih pitanja, ukupan broj pitanja, te ispisuje listu top 5 korisnika, ovisno o njihovom ukupnom rezultatu (postotak točno odgovorenih pitanja).

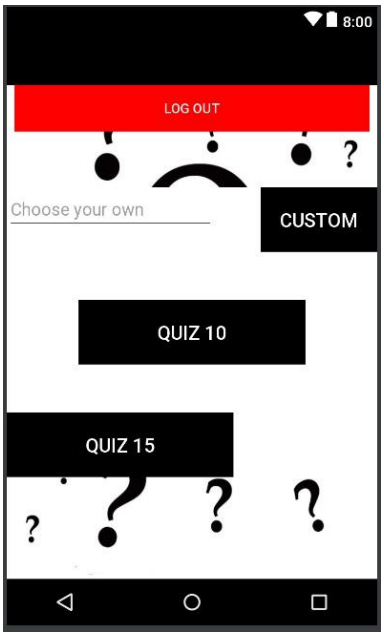


Slika 13. Osnovni tijek rada aplikacije

5.2 Osnovne aktivnosti i klase

5.2.1 MainActivity

MainActivity je naša glavna aktivnost, na nju „skačemo“ nakon što korisnik odradi registraciju/prijavu, kao i u slučaju da smo uključili aplikaciju kad je korisnik već ulogiran. Aktivnost se sastoji od 2 dugmeta, startQuiz2 i startQuiz3, pomoću kojih korisnik može odabrati 2 brza kviza, kviz od 10 pitanja ili kviz od 15 pitanja. Osim toga, aktivnost se sastoji od još jednog dugmeta, customButton, i editText elementa, kombinacijom kojih korisnih može sam odabrati kviz s proizvoljnim brojem pitanja. Također, ovdje se nalazi o LogOut dugme, pomoću kojeg se korisnik može odlogirati.



Slika 14. MainActivity

Elementi se dodaju tako da se prvo kreiraju u layout editoru. Tu se postave njihovi atributi (boja pozadine, boja teksta, veličina teksta...), a nakon toga se njihovo ponašanje programira. Ponašanje dugmeta se izvodi tako da se napravi jedna void metoda u kojoj se opiše ponašanje kakvo želimo. Nakon toga se u layout editoru zada da se ta metoda poziva kod određenog događaja (recimo `OnClick`, u slučaju dugmeta).

```
<Button
    android:id="@+id/customButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#000000"
    android:onClick="OpenQuestionOneCustom"
    android:padding="20dp"
    android:text="Custom"
    android:textColor="#FFFFFF"
    android:textSize="20sp" />
```

Ispis 1. Kôd za `customButton` iz layout editora.

Primjer implementacije ponašanja dugmeta ćemo objasniti na primjeru `customButton` elementa. Iz Ispisa 1 je vidljivo da je `onClick` događaja povezan s

metodom `OpenQuestionOneCustom`, dakle ona će se pozvati kod svakog klika na taj element.

Kako bismo mogli pristupati GUI elementima iz našeg kôda, potrebno je za svaki od tih elemenata napraviti odgovarajuću varijablu, te je povezati s GUI elementom koji joj pripada.

```
EditText editText = (EditText) findViewById(R.id.editText);
```

Ispis 2. Primjer kreiranja i inicijalizacije EditText varijable

U gornjem primjeru vidimo kreaciju varijable `editText` tipa *EditText*. `findViewById(R.id.editText)` nam služi kako bi naš objekt povezali s GUI dugmetom u layout editoru, koji smo također nazvali `editText`.

```
String tempValue = "";
tempValue = editText.getText().toString();
QuestionOne.NumberOfQuestions=Integer.parseInt(tempValue);
NumberOfQuestions=Integer.parseInt(tempValue);
NumberOfQuestionsUnchanging=NumberOfQuestions;
```

Ispis 3. Dohvaćanje broja unesenog u `editText` polje

Varijabla `tempValue`, tipa String, služi nam za spremanje vrijednosti unesene u polje `editText`. Zatim varijablu konvertiramo u integer te je spremamo u tri različite varijable, `NumberOfQuestions` u aktivnosti `QuestionOne`, te *`NumberOfQuestions`* i `NumberOfQuestionsUnchanging` u trenutnoj aktivnosti.

```
Intent OpenQuestionOne = new Intent(this, QuestionOne.class);
startActivity(OpenQuestionOne);
```

Ispis 4. Otvaranje aktivnosti `QuestionOne`

Na kraju, potrebno je da otvorimo aktivnost `QuestionOne`, u kojoj se ispisuju pitanja i odgovori. Prvo pravimo varijablu `OpenQuestionOne`, tipa *Intent*. Prosljeđivanjem objekta `OpenQuestionOne` u metodu `startActivity` pokrećemo željenu aktivnost.

5.2.2 QuestionOne

Aktivnost `QuestionOne` sastoji se od jednog *TextView* elementa, u kojem se ispisuju kviz pitanja, i 4 dugmeta, na kojima se ispisuju ponuđeni odgovori, tako da je na početku bilo potrebno kreirati četiri objekta tipa *Button* i jedan tipa *TextView*.

```
        public void OpenQuestionOne (View view)
 {
     if (NumberOfQuestions!=0) {
         Intent OpenQuestionOne = new Intent(this, QuestionOne.class);
         startActivity(OpenQuestionOne);
         NumberOfQuestions--;
     }

     if (NumberOfQuestions==0) {
         Intent OpenEndScreen = new Intent(this, EndScreen.class);
         startActivity(OpenEndScreen);

     }


 }
```

Ispis 5. Metoda `OpenQuestionOne`

Metoda `OpenQuestionOne` treba se izvršiti onoliko puta koliki je broj pitanja koji je korisnik odabrao. U tu svrhu koristimo brojač `NumberOfQuestions` koji smo postavili u aktivnosti `MainActivity`. Sve dok je brojač veći od 0 metoda će na otvaranje pozivati aktivnost `QuestionOne`, istovremeno umanjujući brojač za 1. Onog trenutka kad brojač bude jednak 0, metoda će otvoriti aktivnost `EndScreen`.

```
public void RateAnswer1 (View view)
{
        if (Answer1.getText().toString()==CorrectAnswer){

            MainActivity.score ++;


        }
```

```
}


public void AnswerOneClicked (View view){

    RateAnswer1(null);
    //OpenQuestionTwo(null);
    cnt = cnt + 1;
    OpenQuestionOne(null);

}
```

Ispis 6. Metode `RateAnswer1` i `AnswerOneClicked`

Svaki od 4 dugmeta na kojima se nalaze odgovori ima implementirane ovakve dvije metode. `AnswerOneClicked` se poziva u slučaju da je kliknuto dugme `Answer1`. Ta metoda poziva `RateAnswer1`, koja uspoređuje ponuđeni odgovor i točni odgovor i uvećava brojilo `score`, iz aktivnosti `MainActivity`, u slučaju da se odgovori podudaraju. Zatim se poziva prije opisana `QpenQuestionOne` metoda.

```
LoadQuestions process = new LoadQuestions();
process.execute();
```

Ispis 7. Kreiranje objekta `LoadQuestions` i njegovo izvršavanje

U primjeru vidimo kreaciju objekta tipa `LoadQuestions`, što je klasa koja služi da ispisuje pitanja u *TextView* i odgovore na dugmad.

5.2.3 LoadQuestions

`LoadQuestions` je klasa koja služi da dohvati string u koji je spremljena JSON datoteka s pitanjima i odgovorima, da ga parsira na stringove, te stringove postavi kao vrijednosti odgovarajućih *TextView* i *Button* elemenata u aktivnosti QuestionOne.

```
String [] question = new String [50];
String [] correct_answer = new String [50];
String incorrect_answer = "";
String [][] incorrect_answers = new String[50][4];
int incorrect_anwers_counter;
```

Ispis 8. Varijable `LoadQuestions` klase

- ***question*** **–** Niz stringova u koji će biti spremljeno svako od 50 pitanja iz JSON datoteke.
- ***correct_answer*** **–** Niz stringova u koji će biti spremljen svaki od 50 točnih odgovora na pitanja.
- ***incorrect_answer*** **–** String varijabla koja nam služi za spremiti točan odgovor na pitanje na kojem smo trenutno.
- ***incorrect_answers*** **–** Dvostruki niz stringova u koji ćemo spremati sve netočne odgovore za svako od 50 pitanja.
- ***incorrect_answers_counter*** **–** Brojač koji nam služi za izbrojati broj netočnih odgovora za pojedino pitanje, pošto broj odgovora može biti različit.

```
try {
    JSONObject JO = new JSONObject(MainActivity.data);

    JSONArray JA = JO.getJSONArray("results");
   for (int i = 0; i < JA.length(); i++)
   {

       JO = (JSONObject) JA.get(i);


       question[i] = String.valueOf(JO.get("question"));

   }

    for (int j = 0; j < JA.length(); j++)
    {

        JO = (JSONObject) JA.get(j);


        correct_answer[j] = String.valueOf(JO.get("correct_answer"));

    }
```

Ispis 9. Petlje pomoću kojih punimo nizove pitanja i točnih odgovora

`JO` je objekt tipa `JSONbject` u koji spremamo string `data` iz aktivnosti `MainActivity`. U varijablu `data` je prethodno spremljen string s JSON datotekom. `JA` je objekt tipa *JSONArray* u koji spremamo sve objekte pod kategorijom `results`, što će biti

naših 50 pitanja i odgovora. Prva `for` petlja u svakoj iteraciji prolazi kroz jedan objekt i sprema ga u `JO`. Zatim, vrijednost pod `question`, za taj objekt, sprema se u `question` niz. Na isti način druga petlja sprema sve točne odgovore u niz `correct_answer`.

```
for (int o = 0; o < JA.length(); o++) {

    JO = (JSONObject) JA.get(o);
    JSONArray JAIA = JO.getJSONArray("incorrect_answers");
    for (int k = 0; k < JAIA.length(); k++) {



        incorrect_answer = String.valueOf(JAIA.get(k));


        incorrect_answers[o][k] = incorrect_answer;


    }
}
```

Ispis 10. Dvostruka `for` petlja koja puni netočne odgovore

Vanjska `for` petlja prolazi kroz 50 objekata, svakog pojedinačno sprema u `JSONArray` po imenu `JAIA`. Unutarnja petlja za svaki od tih objekata, svaki od odgovora posprema u dvostruki niz `incorrect_answers`.

Obrađujući aktivnost `QuestionOne` vidjeli smo da metoda `AnswerOneClicked` uvećava brojač `cnt` za 1, pri svakom izvršavanju. Isto vrijedi za metode za svako od ostatka dogmadi za odgovore. Taj brojač koristimo u klasi `LoadQuestions` kako bismo znali na kojem pitanju se nalazimo.

```
        while (incorrect_answers[QuestionOne.cnt][k]!=null) {


    incorrect_anwers_counter ++;

    k++;
}
```

Ispis 11. Brojač netočnih odgovora

`QuestionOne.cnt` ovdje predstavlja redni broj pitanja na kojem smo trenutno, a `k` predstavlja redni broj netočnih odgovora za to pitanje. Zadatak ove petlje je izbrojati broj netočnih odgovora kako bismo znali koliko odgovora za ovo pitanje sveukupno ima.

```
if (incorrect_anwers_counter ==1) {
    process2.TwoAnswers(correct_answer[QuestionOne.cnt],
incorrect_answers[QuestionOne.cnt][0]);

    String [] Answer12 = new String[2];
    Answer12 = process2.TwoAnswers(correct_answer[QuestionOne.cnt],
incorrect_answers[QuestionOne.cnt][0]);


    QuestionOne.Answer1.setText(Answer12[0]);
    QuestionOne.Answer2.setText(Answer12[1]);
    QuestionOne.Answer3.setVisibility(View.INVISIBLE);
    QuestionOne.Answer4.setVisibility(View.INVISIBLE);
    incorrect_anwers_counter = 0;

}
```

Ispis 12. Postavljanje odgovora za slučaj da je broj netočnih pitanja jednak 1

Na ovom primjeru vidimo kako koristimo informaciju o broju netočnih odgovora. Ovo je slučaj kada je jedan netočan odgovor. Ako tom broju pribrojimo jedan točan odgovor, sveukupan broj odgovora je dva. `Answer12` je niz tipa String u koji pospremamo dva dostupna odgovora. `Proces2` je objekt klase `LoadAnswers`. Problem do kojeg je došlo u ovom trenutku kod izrade aplikacije je taj što su u JSON datoteci odgovori uvijek poredani istim redoslijedom, tako da je točan odgovor uvijek na prvom mjestu, nakon čega slijede netočni odgovori. To bi značilo da bi u našoj aplikaciji prvo dugme uvijek nosilo točan odgovor, što očigledno nije dobro rješenje. Tome sam doskočio upravo klasom `LoadAnswers`, koja služi da promiješa odgovore i vrati ih u obliku niza Stringova. Nakon što su odgovori pospremljeni u `Answer12`, preostaje još da ih postavimo kao vrijednost teksta za prvo i drugo dugme, dok treće i četvrto činimo nevidljivima. U slučaju da se radi o 3 netočna odgovora, treće i četvrto dugme ostaju vidljivi, te se treći i četvrti odgovor prokazuju na njima.

5.2.4 LoadAnswers

LoadAnswers je klasa koja ima dva konstruktora, jedan koji prima četiri stringa i drugi koji prima 2 stringa. Konstruktori vraćaju jednostruki niz stringova. Zadatak konstruktora je da primljene stringove promiješaju i vrate ih nasumičnim redoslijedom.

```
public String[] FourAnswers (String One, String Two, String Three, String Four)
{

        String [] Answers = new String[4];

        Answers[0] = One;
        Answers[1] = Two;
        Answers[2] = Three;
        Answers[3] = Four;



    List myList = Arrays.asList(Answers);
    //System.out.println("\nOriginal list: " + myList);

    Collections.shuffle(myList);
   // System.out.println("First shuffled list: " + myList);

    Collections.shuffle(myList);
    //System.out.println("Second shuffled list: " + myList);

    // CONVERTING LIST BACK TO ARRAY
    Object str[] = myList.toArray();

    Answers[0] = str[0].toString();
    Answers[1] = str[1].toString();
    Answers[2] = str[2].toString();
    Answers[3] = str[3].toString();

    return Answers;



}
```

Ispis 13. Konstruktor `FourAnswers`

Konstruktor iz ovog primjera prima četiri stringa: `One, Two, Three, Four`. Stringove prsprema u niz `Answers`. *Answers* konvertiramo, pospremamo u listu `myList`, koju dvaput promješamo koristeći `Collections.shuffle` metodu. Listu konvertiramo natrag u niz `str`, koji zatim konvertiramo i pospremamo u pojedinačno u svako od četiri polja niza `Answers`. `Answers` na posljetku vraćamo kao povratnu vrijednost našeg konstruktora.

5.2.5 EndScreen

`EndScreen` aktivnost predstavlja zadnji ekran aplikacije. Ovaj ekran se otvara nakon što korisnik odgovori na zadnje pitanje. Na ekranu se ispisuje korisnikov rezultat (broj odgovorenih pitanja i broj točnih odgovora) i lista igrača s najboljim rezultatom (postotkom točnih odgovora), poredana od najboljeg k najgorem. Aktivnost se sastoji od dva *TextViewa* u koje ispisujemo rezultat i postotak te *ListViewa* koji prikazuje listu najboljih igrača.

```
finalScoreText.setText ("You have answered " + Integer.toString(MainActivity.score)
+ "\n questions correctly out of " + MainActivity.NumberOfQuestionsUnchanging +
".");
finalPercentageText.setText ("You have answered " + Integer.toString(percentage) +
"\n% of the questions correctly.");
```

Ispis 14. Ispis korisničkog rezultata na ekran

Gornji ispis pokazuje ispisivanje korisnikovog rezultata u *TextView* elemente `finalScoreText` i `finalPercentageText`. U slučaju prvog vuče `score` iz `MainActivity` aktivnosti, te `NumberOfQuestionsUnchanging` iz iste aktivnosti. U slučaju `finalPercentageText` varijable, povratnu vrijednost objekta `percentage` (objekt klase `Percentage`, koja računa postotke zadanih brojeva) pretvara u string te ispisuje. U ovoj aktivnosti se također odvija čitanje iz Firebase baze podataka, te unos podataka u bazu. O tome više u sljedećim poglavljima.

5.3 Povezivanje s Firebaseom

Kako bi se naša aplikacija mogla služiti Firebaseom, potrebno je prije svega napraviti određene modifikacije Gradlea. Naime, kako bi Android Studio znao pravilno izbuildati naš projekt, potrebno je u Gradle datotekama specificirati biblioteke iz kojih će povlačiti potrebne podatke.

```
    dependencies {
        classpath 'com.android.tools.build:gradle:3.0.1'
        classpath 'com.google.gms:google-services:4.0.1'


        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}
```

Ispis 15. Dodavanje ovisnosti u Project Gradle datodeku

U Projekt Gradleu povezali smo naš prijekt za sa Googlov servis.

```
implementation 'com.google.firebase:firebase-auth:9.2.0'
implementation 'com.google.firebase:firebase-database:9.2.0'
```

Ispis 16. Dodavanje ovisnosti u App Gradle datoteku

U Gradle datoteci App nivoa dodali smo ovisnosti za biblioteke za autentikaciju i bazu podataka.

Nakon što su ovisnosti dodane u Gradle, može se prionuti radu s Firebaseom.

5.3.1 Registracija i LogIn

Za registraciju novih korisnika sam napravio aktivnost `Register`. Za prijavu korisnika koristimo `LogIn` aktivnost. Prva zadaća obje aktivnosti je da provjeri je li korisnik već ulogiran, pošto u tom slučaju ne želimo da se ova aktivnost prikaže, nego je potrebno da odmah ode na `MainActivity`.

```
private FirebaseAuth firebaseAuth;

firebaseAuth = FirebaseAuth.getInstance();
if (firebaseAuth.getCurrentUser() != null){
    finish();
    startActivity(new Intent(getApplicationContext(), MainActivity.class));

}
```

Ispis 17. Kreacija i upotreba `FirebaseAuth` objekta

`firebaseAuth` je objekt klase `FirebaseAuth`. Pomoću ovog objekta vršimo operacije s Firebase bazom korisnika. U ispisu 17 vidimo primjer kreacije i upotrebe jednog takvog objekta. `FirebaseAuth.getInstance()` nam vraća instancu trenutnog korisnika koju smo spremili u objekt *`firebaseAuth`*. Metoda *`firebaseAuth.getCurrentUser`* će vratiti rezultat različit od `null` u trenutku kad je traženi korisnik već ulogiran, u kojem slučaju završavamo s ovom aktivnošću i otvaramo `MainActivity`.

Na ovoj aktivnosti nalaze se dva tekstualna polja, jedno za unos e-mail adrese, drugo za unos željene lozinke, te dugme za registraciju.

```
firebaseAuth.createUserWithEmailAndPassword(email,
password).addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
    @Override
    public void onComplete(@NonNull Task<AuthResult> task) {
        if (task.isSuccessful())
        {
          //Registration is successful
            Toast.makeText(Register.this, "Registered successfully",
Toast.LENGTH_SHORT).show();
            Intent OpenMainActivity = new Intent(getApplicationContext(),
MainActivity.class);
            startActivity(OpenMainActivity);

        }
        else
        {
            Toast.makeText(Register.this, "Could not register. Please try again.",
Toast.LENGTH_SHORT).show();

        }
    }
});
```

Ispis 18. Registracija novog korisnika putem *FirebaseAuth* objekta

`CreateUserWithEmailAndPassword` je metoda `FirebaseAuth` klase koja izvršava registraciju korisnika, spremajući njegov e-mail i lozinku u korisničku bazu podataka. Metoda `isSuccessful()` provjerava uspješnost registracije (bit će neuspješna u slučaju da već postoji korisnik s istim e-mailom), a unutar tijela funkcije ispisujemo poruku o uspjehu ili neuspjehu registracije. U slučaju da je registracija uspješna, otvaramo `MainActivity`.

Slika 15. Primjer registriranog korisnika u Firbase Authentication Bazi. Vidimo korisnikov e-mail i šifriranu lozinku.

Na sličan način obavljamo LogIn. Koristimo `firebaseAuth.signInWithEmailAndPassword` metodu.

5.3.2 Pohrana podataka

Aktivnost `EndScreen` ispisuje rezultat korisniku, ali isti taj rezultat treba pohraniti u Firebase bazu podataka. Isto tako treba iščitati rezultate drugih korisnika i ispisati ih na ekran. Upisivanje u bazu se vrši tako da prvo kreiramo klasu koja u sebi ima varijable tipa podataka koje želimo unijeti u bazu. Zatim objekt te klase proslijedimo kao ulazni parametar `setValue` metode objekta klase `DatabaseReference`. Klasa koju sam kreirao za tu svrhu zove se `UserInformation`.

```
public int allTimePercentageNum = 0;
public int allTimePercentageNumInverted = 1;
public String allTimePercentage = "0";
public String lastScore = "0";
public String allTimeScore = "0";
public String allTimeNumberOfQuestions = "1";
public String userEmail;
```

Ispis 19. Varijable klase `UserInformation`.

```
firebaseAuth = FirebaseAuth.getInstance();

FirebaseUser uuser = firebaseAuth.getCurrentUser();



if (userInfoPublic != null) {
    UserInformation userInformation = new UserInformation(MainActivity.score,
MainActivity.NumberOfQuestionsUnchanging,
Integer.parseInt(userInfoPublic.allTimeScore),
Integer.parseInt(userInfoPublic.allTimeNumberOfQuestions), uuser.getEmail());

    databaseReference.child(uuser.getUid()).setValue(userInformation);
}
```

```
else
{
    UserInformation userInformation = new UserInformation(MainActivity.score,
MainActivity.score, 0 , 0, uuser.getEmail());
    databaseReference.child(uuser.getUid()).setValue(userInformation);
}
```

Ispis 20. Upis podataka u Firebase bazu.

```
        public void onDataChange(DataSnapshot dataSnapshot) {
    UserInformation temp = dataSnapshot.getValue(UserInformation.class);
    userInfoPublic = temp;
    Percentage percentageTemp = new Percentage();

}
```

Ispis 21. Provjera trenutnog stanja baze.

Ispis 19, 20 i 21 prikazuju redoslijed izvođenja operacija s Firebase bazom podataka u `EndScreen` aktivnosti.

Ispis 21 odvija se u `OnCreate` metodi i izvršit će se prvi. `DataSnapshot` nalazi se unutar `AddValueEventListenera` koji se poziva kod svake promjene u povezanoj bazi. Osim toga, pozvat će se jednom kod prvog izvršavanja kôda, bez obzira na promjene u bazi. Svojstvo da se poziva kod prvog izvođenja koristimo ovdje. `DataSnapshot` „snima“ trenutno stanje baze podataka. Objekt `temp` koji ovdje vidimo je objekt klase `UserInformation`. Pomoću `getValue()` metode objekta `dataSnapshot` dohvaćamo vrijednost u tipu `UserInformation` i spremamo je u objekt `temp`. Objekt `userInfoPublic` je također objekt klase `UserInformation` koji je inicijaliziran na početku. U njega spremamo vrijednost objekta `temp`.

Kôd prikazan u Ispisu 20 izvršava se nakon toga. Objekt `userInfoPublic` sad koristimo kako bismo provjerili stanje naše baze. U slučaju da je ovo prvi put da unosimo podatke za našeg korisnika (baza je prazna), vrijednost će biti `null`, u suprotnom će biti različita od `null`. Pomoću uvjeta `if` se uvjeravamo da se u oba slučaja izvrši željeni dio kôda.

5.3.3 Čitanje podataka

Zadnja stvar koju treba napraviti je pročitati podatke od ostalih korisnika u bazi i ispisati ih na `listView`. Da bismo to napravili prvo je potrebno kreirati jedan pomoćni *layout*. U našem slučaju, to je `list_layout`. Pomoćni layout sastoji se od dva *TextView* elementa, jedan za korisnički email, drugi za bodove. U ova dva polja unosit će se ti podaci za jednog korisnika, taj postupak ponovit će se za svakog korisnika, te će svaki pojedinačno biti unesen na *ListView* element koji se nalazi u `EndScreen` aktivnosti.

Zatim je potrebno kreirati novu klasu koja će implementirati unos podataka u listu. U našem slučaju to je klasa `UserList`.

```
databaseReference = FirebaseDatabase.getInstance().getReference();

Query query =
FirebaseDatabase.getInstance().getReference().orderByChild("allTimePercentageNumInv
erted");
query.addListenerForSingleValueEvent(new ValueEventListener() {

    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {

        userList.clear();



        for (DataSnapshot userSnapshot : dataSnapshot.getChildren()){
            UserInformation user = userSnapshot.getValue(UserInformation.class);

            userList.add(user);
        }



        UserList adapter = new UserList(EndScreen.this, userList);
        listViewUserScores.setAdapter(adapter);
    }



    @Override
    public void onCancelled(DatabaseError databaseError) {

    }
});
```

Ispis 22. Čitanje podataka iz baze i unos na listu

Da bismo dobili našu listu željenih korisnika, trebamo na bazi izvršiti `query`. U `databaseReference` u ovom slučaju kao referencu uzimamo cijelu bazu, tako da će „djeca“ naše reference biti pojedini korisnici. Metoda `orderByChild()` prima ime stupca po kojem želimo sortirati, te po tom elementu sortira bazu. Ovdje je namjera da sortira po korisničkom rezultatu, i to od najvećeg prema najmanjem. Ovdje sam naišao na jedan nedostatak Firebasea. Naime, nije moguće rezultate `queryja` poredati silazno, rezultati se uvijek sortiraju uzlazno. Stoga, bilo je potrebno naći okolno rješenje. Problemu sam doskočio tako da sam u klasu `UserList` dodao varijablu `allTimePercentageNumInverted`, u koju se sprema vrijednost varijable `allTimePercentageNum` (ukupni rezultat korisnika) pomnožena s -1. Redanjem rezultata po toj varijabli dobivamo željeni redoslijed korisnika.

6. Zaključak

Namjera ovog rada bila je demonstrirati izradu Android mobilne aplikacije u Android Studio razvojnom okruženju. Demonstrirana je i implementacija različitih tehnologija u Android aplikaciju, poput spajanja s Firebase platformom i korištenja njenih usluga i korištenja vanjske JSON datoteke za dohvaćanje podataka. Objašnjena je arhitektura Android sustava te kratak opis svih korištenih tehnologija.

Razvoj aplikacije bio je uvelike olakšan jako dobrim razvojnim okruženjem koje pruža Android Studio. Android Studio se sam brine za slaganje strukture projekta, nudi nam već napravljene potrebne xml datoteke (poput AndroidManufest.xml) i time znatno ubrzava razvoj aplikacije čineći repetitivne zadaće umjesto nas. Također, velika pomoć su jako dobri i brzi emulatori koje Android studio nudi, čime izbjegavamo potrebu da aplikaciju stalno pokrećemo na pravom uređaju, što je također jako dobrodošla ušteda vremena.

Firebase paleta usluga se također pokazala kao pun pogodak. Firebase umjesto nas obavlja čitav serverski dio autentikacije korisnika, što nas ostavlja s potrebom da implementiramo samo klijent dio tog odnosa. Zamjerka Firebaseu je to što u radu s bazom ima štur i ograničen izbor operacija, pa je bilo potrebe tražiti nekakva okolna rješenja za dobiti rezultate koji su bili potrebni. No, sitnice poput toga ni izbliza ne zasjenjuju sve prednosti koje ova platforma nudi.

Nameće se zaključak da je kombinacija Android Studija i Firebasea jako dobar i dinamičan razvojni duo koji nudi velike prednosti u vidu uštede vremena, korisnih alata i lakog otklanjanja grešaka.

7. Literatura

[1] Android OS,
datum zadnjeg pristupa: 20.11.2018
[2] Android arhitektura,
datum zadnjeg pristupa: 18.11.2018
[3] Activity,
datum zadnjeg pristupa: 20.11.2018
[4] Intent,
datum zadnjeg pristupa: 20.11.2018
[5] Firebase,
datum zadnjeg pristupa: 25.11.2018
[6] Klijent-server arhitektura, https://msatechnosoft.in/blog/tech-blogs/types-of-client-server-architecture
datum zadnjeg pristupa: 21.11.2018