JAVA APLIKACIJA EASYLANGUAGE

Beran, Ivan

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij informacijske tehnologije

IVAN BERAN

Z A V R Š N I R A D

JAVA APLIKACIJA EASYLANGUAGE

Split, rujan 2019.

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: Ivan Beran

Naslov rada: Java aplikacija EasyLanguage

Mentor: Ljiljana Despalatović, predavač

Split, rujan 2019.

Sadržaj

Sažetak

Cilj ovog završnog rada je opisati proces izrade i korištenja Java aplikacije EasyLanguage koja je namijenjena za korištenje u procesu učenja stranog jezika korisnika manje dobi. Rad je temeljen na aplikaciji namijenjenoj pametnim telefonima. Aplikacija prije samog korištenja zahtijeva registriranje putem emaila. Registracijom se kreira profil korisnika koji služi za korištenje aplikacije. Korisnici se dijele na učenike te učitelje. Sama aplikacija ime više segmenata; kviza koji se sam sastoji od dijela u kojem se nalazi lekcija nakon koje slijede pitanja vezana za to područje, igre *Memory* u kojoj korisnici nasumično odabrane sličice moraju upariti s odgovarajućom riječi, te od dijela u kojem učitelj može osobno postaviti zadatke i odgovore koje učenici moraju riješiti.

Ključne riječi: aktivnost, Android, aplikacija, Java, kôd.

Summary

Java application EasyLanguage

The goal of this thesis is to describe the process of creating and using the EasyLanguage Java application that is intended for use in the foreign language learning process by users of younger age. This thesis is based on application designed for use on smartphones. Before you can use the application, it requires registration via email. Registration creates a user profile which is used to login into the app. Users are divided into students and teachers. This application can be divided into multiple segments; a quiz consists of the lessons followed by the questions related to that lesson, a *Memory* game in which users need to match up randomly selected graphic pictures with the appropriate word, and the part where the teacher can personally set the homework which consists of questions and answers that the students have to fill in.

Keywords: activity, Android, application, code, Java.

1. Uvod

Današnje društvo teži sve većoj i većoj digitalizaciji. Na svakom kutu možemo vidjeti pametne uređaje koji nam ulaze u sve pore društva. Pametni mobiteli, pametni satovi, pametni televizori, pametni hladnjaci, pametne kuće su samo neki od pametnih uređaja s kojima imamo kontakt svaki dan. Nešto što se nekad smatralo znanstvenom fantastikom danas je stvarnost. Sve te pametne uređaje možemo skupiti pod jedan pojam Internet stvari (IOT eng. *Internet of things*). IOT nam omogućuje da možemo jednostavno kontrolirati svoj hladnjak, pametni televizor ili perilicu rublja pomoću pametnog telefona. Međutim, IOT je moguć samo zbog platforme na kojoj svi ti uređaji mogu raditi i komunicirati. Tu vodeću ulogu ima Android. Ako pogledamo tržište, možemo primijetiti da većina pametnih uređaja radi na Googleovom operativnom sustavu Android. Prema istraživanjima provedenim od strane Statcountera i Statista Android drži oko 76% ukupnog tržišta. Ovaj rad se temelji na ideji korištenja pametnih mobitela u svakodnevnom učenju. On bi trebao dati procesu učenja jednu modernu alternativu koja bi koristila učiteljima i učenicima.

U drugom poglavlju se objašnjava izgled i korištenje aplikacije sa stajališta korisnika. Svrha toga je detaljno proći kroz sve aktivnosti aplikacije što daje uvid u samu namjenu aplikacije i služi kao uputa. Treće poglavlje se fokusira na opis programskog razvoja aplikacije. Tu se objašnjavaju tehnologije odabrane za razvoj aplikacije te uvid u programski razvoj aplikacije. Programski razvoj zahtijeva poznavanje programskog jezika Java, razvojnog okruženja tehnologije Android Studio, korištenja baze podataka, u ovom slučaju Firebase *database*, te XML (eng. *Extensible Markup Language*) opisnog jezika. Aplikacija je izrađena u razvojnom okruženju Android Studio verzije 3.3.1, a testiranje aplikacije je izvođeno na Android Emulatoru Huawei p20.

2. Opis aplikacije s korisničke strane

Android aplikacija EasyLanguage namijenjena je djeci manje dobi kao dodatak prilikom učenja stranog jezika u školi. Aplikacija omogućuje učenje pojedinih dijelova stranog jezika kroz neke od implementacija kao što su učenje glagolskih vremena uz ugrađeni kviz podijeljen na vremena i proširivanje vokabulara uz igru *Memory*, te daje profesorima mogućnost postavljanja zadaća koje učenici mogu rješavati. Dio aplikacije koji je osmišljen kao kviz se sastoji od lekcija o glagolskim vremenima, te je podijeljen na tri cjeline. Prilikom odabira cjeline, slijedi dio teorije koji je popraćen samim zadatcima vezanim za tu teoriju.

U igri *Memory* od korisnika se traži da poveže određenu sliku s odgovarajućom riječi koja predstavlja tu sliku, te se pamti pet najboljih rezultata, što može služiti kao kompetitivni poticaj i dati više smisla repetitivnosti. Zadnji dio aplikacije je drugačiji ovisno koristi li aplikaciju učenik ili profesor. Korisnik prijavljen kao profesor ima mogućnost postavljanja domaćih zadaća na aplikaciju, uvida u postavljenu zadaću te uvid u rezultate učenika, dok se učeniku prikazuju samo pitanja koja je profesor postavio i ima mogućnost odgovaranja na ista.

2.1. Opis aplikacije

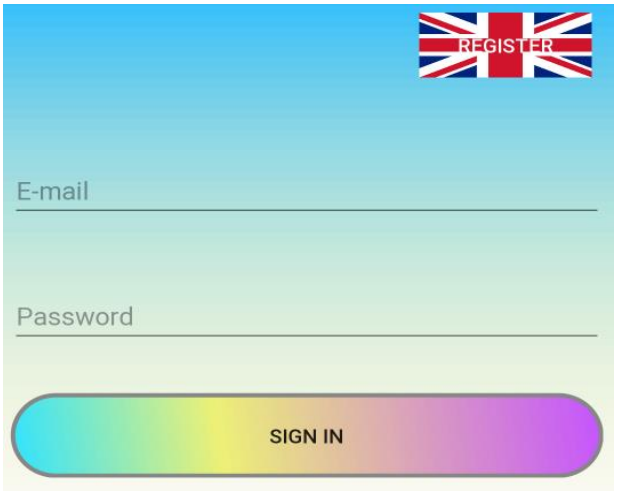
Aplikacija EasyLanguage se sastoji od nekoliko glavnih dijelova:

- Registracija korisnika
- Pravljenje profila
- Prijava korisnika
- Lekcije iz glagolskih vremena te provjera znanja
- Igra *Memory* za proširivanje vokabulara
- Domaće zadaće

2.1.1. Registracija korisnika

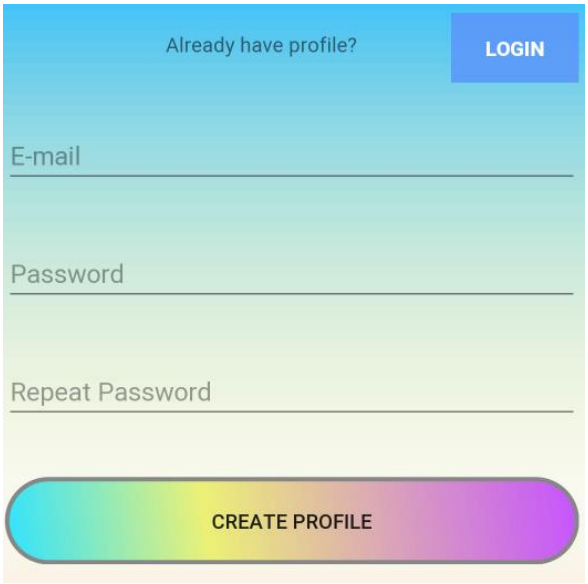
Prilikom pokretanja aplikacije, korisnik će se naći na aktivnosti za prijavljivanje koja je prikazana na slici 1. Ako korisnik nije prijavljen, to je aktivnost koja će se uvijek prva pokazati. Prijaviti se mogu samo korisnici koji su se prethodno registrirali te imaju profil. Ako korisnik

nema profil, na ovoj aktivnosti postoji tipka u gornjem desnom kutu koji ih vodi na aktivnost za registraciju.



Slika 1: Tipka za registraciju

Na aktivnosti za registraciju koja je prikazana na slici 2, od korisnika se zahtijeva adresa električne pošte koja nije već korištena za pravljenje profila, te lozinka koja mora imati više od osam znakova i mora je ponoviti dva puta.

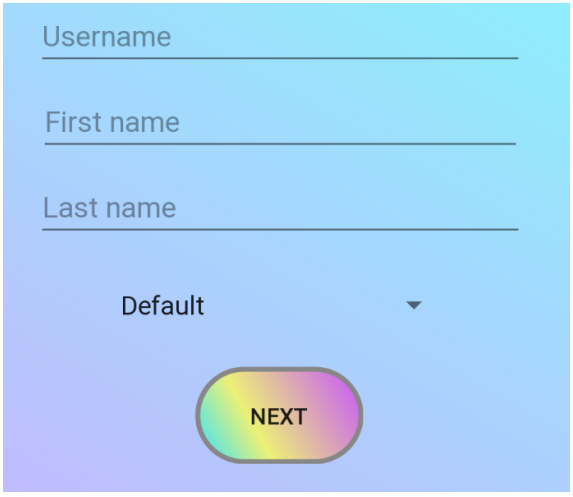


Slika 2: Aktivnost za registraciju

Nakon uspješno unesene adrese elektroničke pošte te lozinke, klikom na `create profile` tipku, korisnika se preusmjerava na aktivnost za popunjavanje detalja profila.

2.1.2. Pravljenje profila

Slika 3 prikazuje aktivnosti za pravljenje profila. Od korisnika se zahtijeva da se unese ime, prezime, korisničko ime te spol. Nakon unosa i odabira, klikom na tipku `next` se korisniku izrađuje profil, te ga se automatski prijavljuje u aplikaciju i dovodi u glavni izbornik.



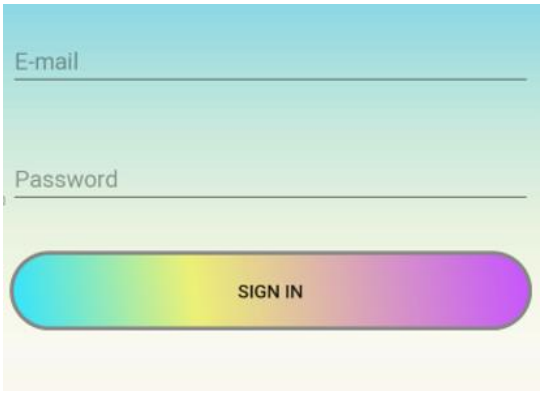
Slika 3: Aktivnost za uređenje detalja profila

Također u glavnom izborniku pritiskom tipke `profile` korisnika se dovodi na aktivnost u kojoj može vidjeti detalje profila (slika 4). Tu mu se pruža mogućnost uvida u detalje i izmjene nekih stavki korisničkog profila te uvid u broj riješenih zadataka.

Slika 4: Prikaz aktivnosti profila

2.1.3. Prijava korisnika

Ako korisnik već ima registriran profil, on se unosom svoje adrese elektronske pošte i lozinke može automatski prijaviti u aplikaciju. To radi na aktivnosti za prijavljivanje (slika 5).

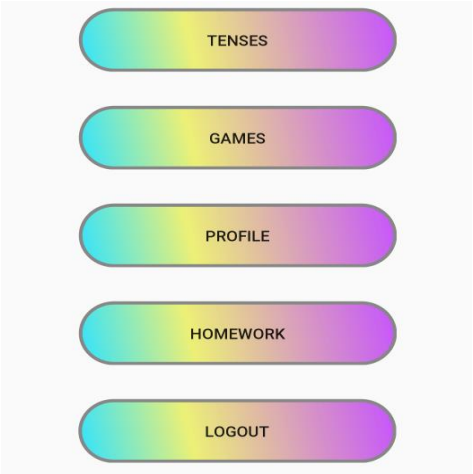


Slika 5: Aktivnost za prijavljivanje

Nakon unosa adrese elektronske pošte i lozinke te klikom na tipku `Sign in` korisnika se prebacuje na glavni izbornik.

2.1.4. Izbornik

U glavnom izborniku (slika 6) se nalaze tipke koji vode do svih funkcionalnosti koje ova aplikacija pruža.



Slika 6: Glavni izbornik

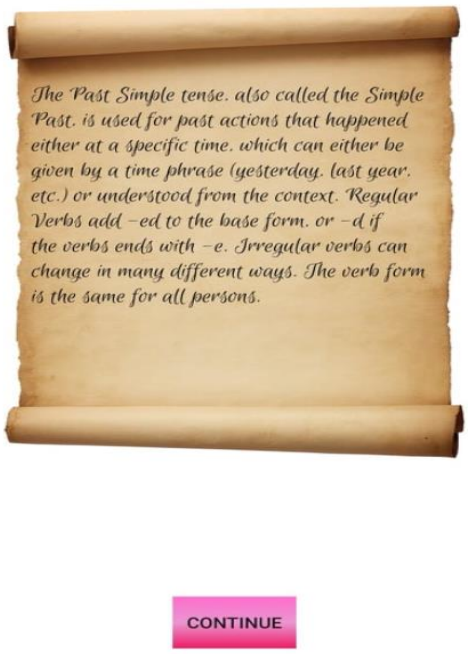
Tipke koji se tu nalaze su: `tenses` koja vodi na aktivnost za učenje glagolskih vremena, tipka `games` koja vodi na aktivnost za `memory` i listu najboljih rezultata, već spomenuti `profile`, tipka `homework` koja ovisno da li je korisnik prijavljen kao učitelj ili učenik vodi na aktivnost koja je predviđena za prijavljenog korisnika i `logout` koja odjavljuje korisnika iz trenutne sesije. Pozadina ove aktivnosti mijenja boju ovisno o tome da li je korisnik muško ili žensko.

2.1.5. Lekcije iz glagolskih vremena te provjera znanja

Pritiskom na tipku `tenses` korisnika se dovodi na aktivnost za odabir glagolskog vremena (slika 7). Oni su podijeljeni na tri dijela: prošlost, sadašnjost i budućnost.

Slika 7: Aktivnosti `tenses`

U ovoj aktivnosti predložene slike služe kao tipke za odabir. Nakon odabira glagolskog vremena, korisniku se vodi na aktivnost `lesson` i prezentira mu se kratka animirana lekcija iz odabranog glagolskog vremena. Na slici 8 je prikazan izgled jedne takve animacije.

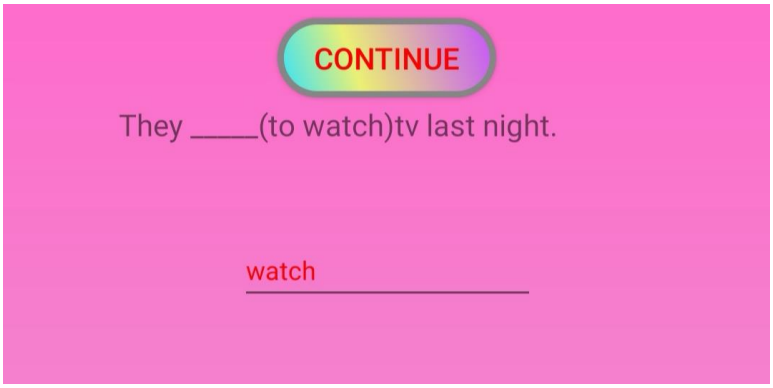


Slika 8: Aktivnost `lesson`

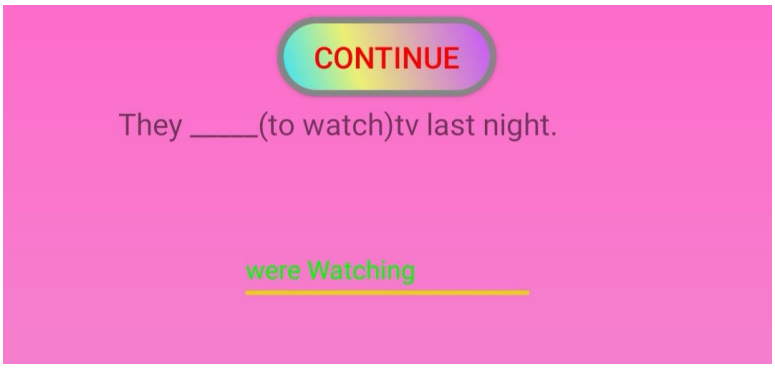
Pritiskom na tipku `continue` korisniku se prikazuju pitanja vezana za tu lekciju. Od korisnika se traži da točno odgovor na pitanja tako da odgovor upiše u pripadajuću kućicu (slika 9).

Slika 9: Prikaz aktivnosti *Quiz*

Nakon što korisnik upiše svoje odgovore pritiskom na tipku *continue* kućicu promijene boju shodno tome da li su odgovori točni (slika 11) ili ne (slika 10).

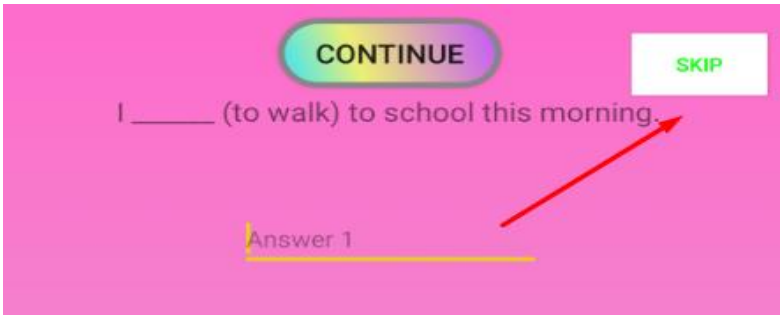


Slika 10: Prikaz pogrešnog odgovora



Slika 11: Prikaz ispravnog odgovora

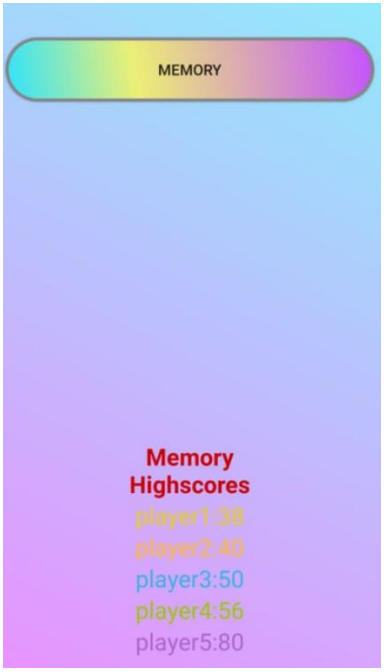
Korisnik ne može napredovati dok svi odgovori ne budu točni. Ako je korisnik već jednom riješio neku lekciju u gornjem desnom kutu se pojavi tipka `next` koja omogućuje preskakanje pitanja (slika 12). Broj riješenih pitanja korisnik može provjeriti na aktivnosti `profile`.



Slika 12: Prikaz tipke za preskakanje pitanja

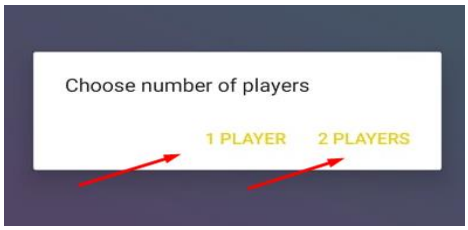
2.1.6. Igra *Memory* za proširivanje vokabulara

Pritiskom tipke `games` u glavnom izborniku, korisnika se dovodi na aktivnost (slika 13) na kojoj se nalaze tipka za pokretanje igre *Memory* te lista najboljih rezultata postignutih do sada.



Slika 13: Prikaz aktivnosti *Games*

Prilikom odabira igre *Memory*, korisniku je prikazan prozorčić (slika 14) u kojem bira da li želi igru igrati sam ili u paru.

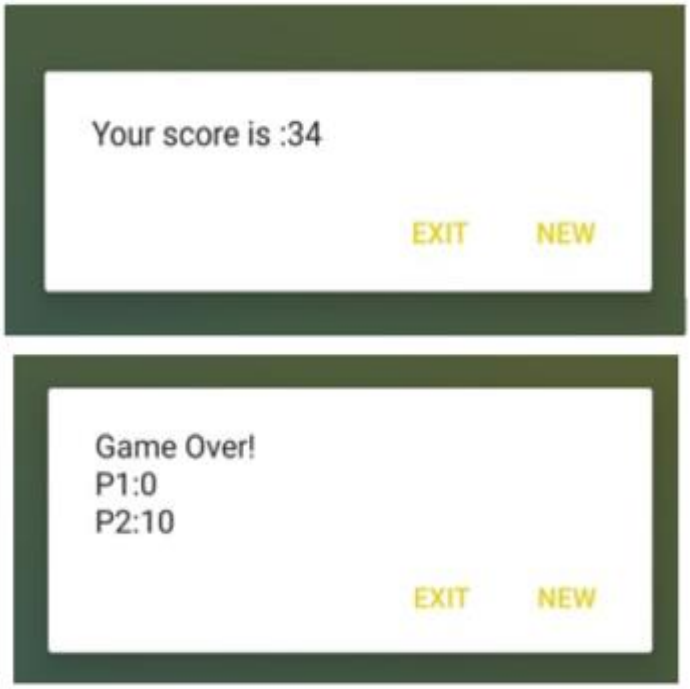


Slika 14: Odabir igrača

Ako korisnik odabere igru za jednog igrača, igra se odvija tako da igrač igra sam, a broje mu se koraci koji su bili potrebni da pronađe sve parove. Prilikom odabira igre za dva igrača, igrači se izmjenjuju svaki put kada onaj drugi ne uspije naći par.

Sama igra se sastoji od toga da je igračima, odnosno igraču prezentirana matrica veličine četiri puta pet, gdje svaki kvadratić predstavlja tipku. Kada korisnik pritisne jednu od tipki umjesto nje se prikaže jedna od slučajno izabranih slika koja predstavlja neki pojam ili riječ koja opisuje pojam, a konačni cilj je upariti sve riječi s odgovarajućim slikama.

Po završetku igre korisniku je predstavljen prozorčić u kojem piše njegov rezultat i mogućnost da pokrene novu igru, ali s različitim slikama i rasporedom samih ili se vrati na izbornik (slika 15). Ukoliko je igra igrana s dva igrača u prozorčiću će pisati pobjednik (slika 15).



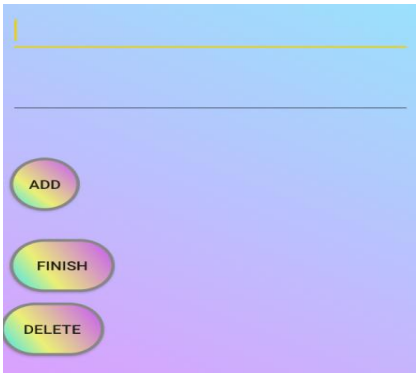
Slika 15: Prozor na kraju igre

2.1.7. Domaće zadaće

Pritiskom tipke `Homework` na glavnom izborniku, korisnik se može naći na dvije različite aktivnosti, ovisno o tome da li je korisnik prijavljen kao učenik ili učitelj.

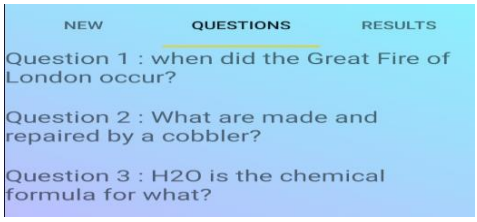
Ako je korisnik prijavljen kao učenik aktivnost mu se sastoji od pitanja koje je učitelj napisao. Iza svakog pitanja se nalazi linija na koju korisnik piše odgovor. Nakon što je korisnik odgovorio na sva pitanja i želi predati zadaću, to čini tako što pritisne tipku `continue` u gornjem desnom kutu. Važno je napomenuti da svaki korisnik ima pravo samo jednom rješavati jednu zadaću.

Nadalje, ako je korisnik prijavljen kao učitelj aktivnost mu se sastoji od tri različita prozora kojima može navigirati tipkama `new`, `questions` i `results` pri samome vrhu. U prozoru `New` se nalaze dvije linije te tri tipke (slika 16). Prva linija služi za pisanje pitanja dok druga služi za pisanje odgovora. Nakon upisanog pitanja i odgovora, pritiskom na tipku `add` to pitanje i odgovor se spremaju u memoriju i korisniku se nudi mogućnost upisa sljedećeg pitanja. Nakon što je korisnik upisao željeni broj pitanja i odgovora, pritiskom tipke `finish` on to prijenosi na bazu, što omogućuje učenicima rješavanje zadaće.



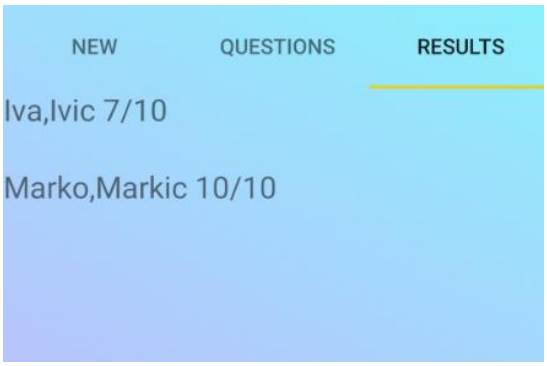
Slika 16: Aktivnost za dodavanje zadaće

U drugom prozoru korisnik ima uvid u trenutna pitanja postavljena na bazi (Slika 17).



Slika 17: Uvid u trenutna pitanja

Ako već postoji zadaća u tom prozoru, prilikom kreiranja nove zadaće potrebno je prvo pritisnuti tipku `delete` u prvom prozoru kako bi počistili bazu prije dodavanja nove zadaće. U trećem prozoru se nalaze imena i rezultati riješenih zadaća (slika 18).



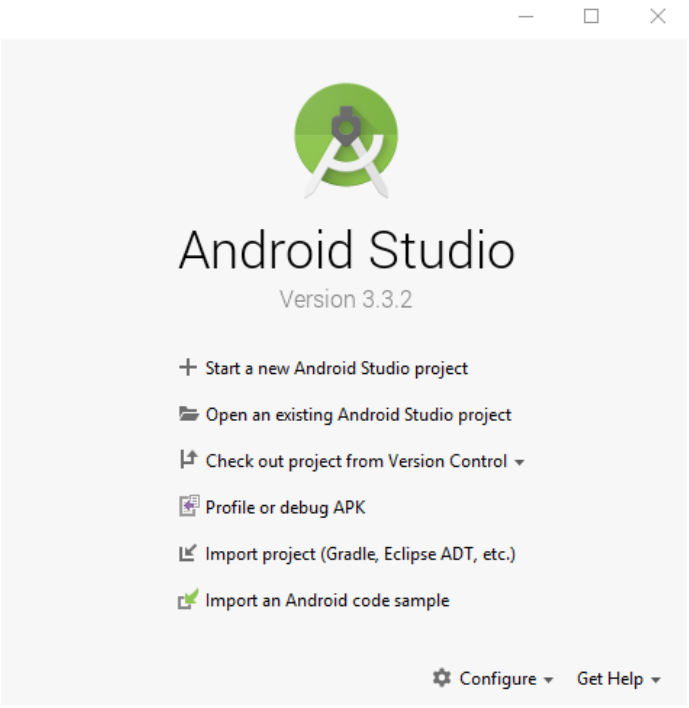
Slika 18: Uvid u rezultate zadaće

3. Izrada android aplikacije

3.1. Priprema aplikacije

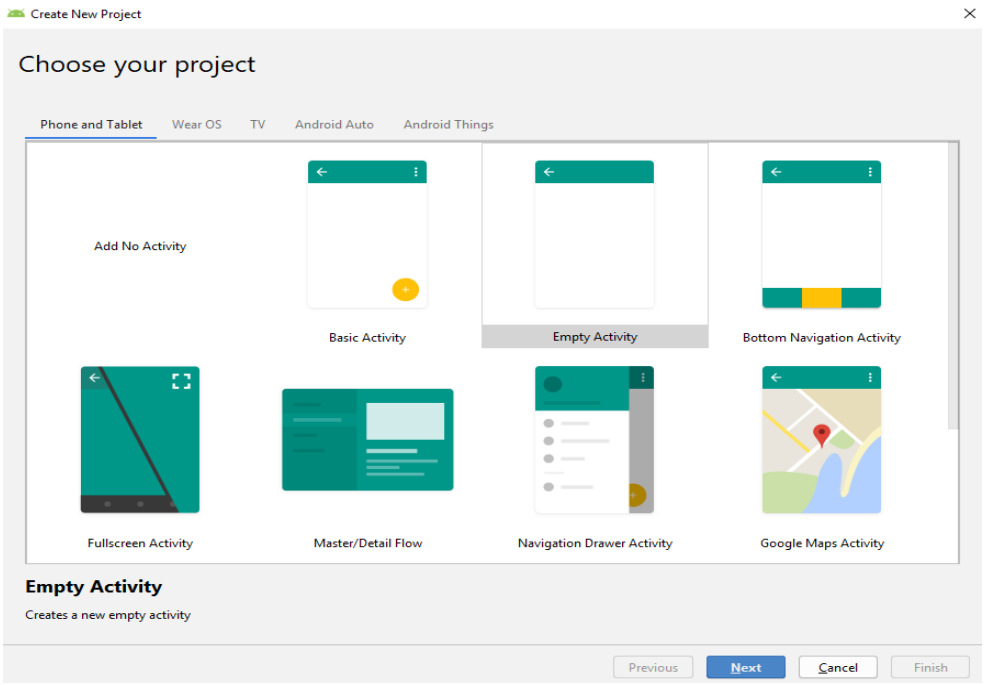
Integrirano razvojno okruženje, skraćeno IDE (Integrated Development Environment), korišteno za izradu ove aplikacije je Android Studio verzije 3.3.2. Android Studio je službeno integrirano razvojno okruženje za Android operacijski sustav. Android Studio pruža integrirani uređivač teksta, biblioteka, prevoditelja, testne platforme i programa za otkrivanje grešaka (engl. debugger). Izgrađen je na temelju softvera IntelliJ IDEA. Operativni sustav Android (OS) je operativni sustav otvorenog kôda (eng. *open source*) za mobilne telefone temeljen na Linuxu. Android pokreće telefone, satove, televizije i sve više sudjeluje u IOT. Kao i sustav na kojem je temeljen, otvorenost kôda ovaj operativni sustav čini popularnim jer programerima omogućuje slobodu u izradi aplikacija.

Prilikom pokretanja programa korisniku je predstavljen prozor s opcijama prikazanim na slici 19.



Slika 19: Početni prozor Android Studija

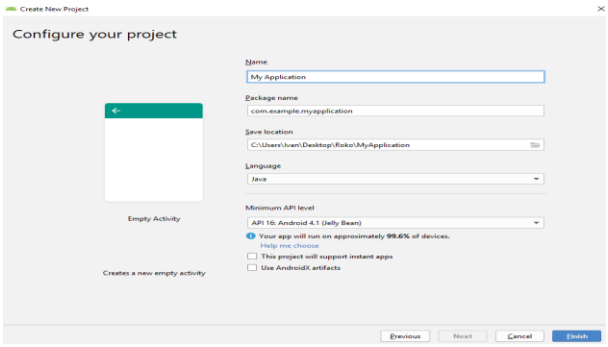
Klikom na tipku za kreiranje novog projekta, korisnika se dovodi na prozor u kojem mora odrediti neke osnovne detalje projekta kojeg kreira. Kao što je prikazano na slici 20 prvo mora odrediti na kojim uređajima želi pokretati svoju aplikaciju, te odrediti kako će mu glavna aktivnost izgledati, iako se sami izgled kasnije može promijeniti.



Slika 20: Odabir izgleda projekta

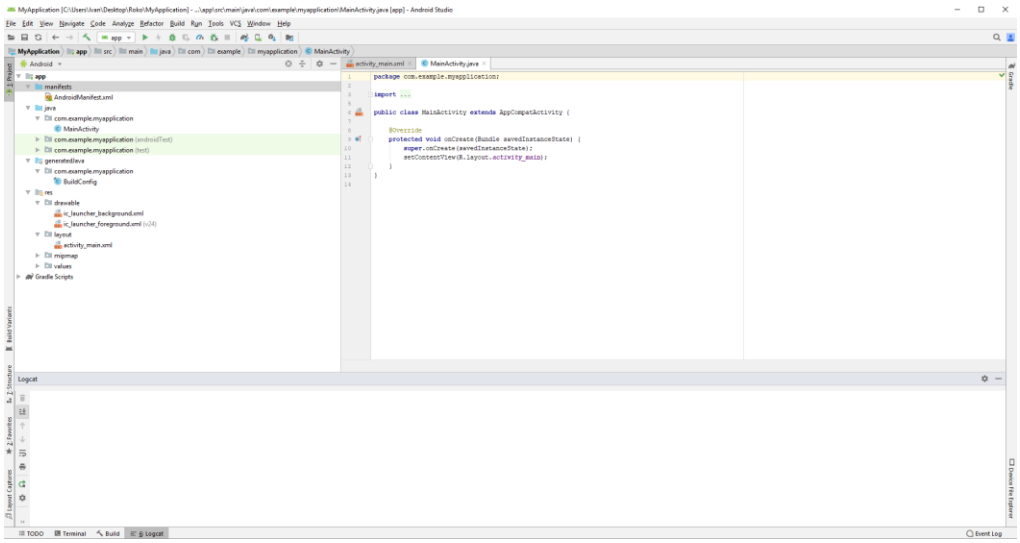
Nakon odabira od korisnika se zahtijeva da svojoj aplikaciji da ime (slika 21), odredi jezik u kojem će aplikacija biti pisana i minimalnu razinu sučelja za programiranje aplikacija (API *Application Programming Interface*). API je skup raznih biblioteka koje daje pristup svim komponentama strojne opreme, te određuje kompatibilnost aplikacije s uređajima na kojima može biti instalirana.

Novije verzije pružaju više mogućnosti prilikom programiranja te nude neka bolja programska rješenja, ali kao nedostatak zahtijevaju novije uređaje koji su kompatibilni s tom razinom i samim time smanjuju opseg uređaja koji mogu pokrenuti aplikaciju. Programski jezici koji se mogu koristiti u Android Studiju su Kotlin i Java.



Slika 21: Konfiguriranje projekta

Prilikom kreiranja ove aplikacije korišten je API 16 i programski jezik Java, te je aplikacija namijenjena za mobitel i tablet. Klikom na tipku *Finish* kreira se projekt, te se otvara prozor (slika 22) u kojem korisnik može početi pisati kôd.



Slika 22: Prikaz praznog projekta

U tom prozoru pri samome vrhu se nalazi alatna traka koja ima nekoliko svrha kao što su pokretanje aplikacije, zaustavljanje, odabir virtualnog uređaja za pokretanje, SDK (*Software Development Kit*) upravitelj i slično.

SDK upravitelj nam služi za instalaciju i preuzimanje najnovijih verzija SDK-ova. Instaliranjem Android SDK dobivamo potrebne biblioteke potrebne za razvoj, debugger, emulator, dokumentaciju za razna sučelja, primjere izvornog kôda i slično. Prilikom svakog puta kada Google izda novu verziju Androida, objavljuje se i odgovarajući SDK. Kako bi mogli pisali

programe s najnovijim značajkama, programeri moraju preuzeti i instalirati SDK za svaku verziju za određeni telefon.

S lijeve strane možemo vidjeti popis svih direktorija koji se koriste u aplikaciji. Nama najbitniji su `Android manifest`, `java`, `Res` te `Gradle Scripts`. `Android Manifest`, koji se vidi na ispisu 1, sastoji se od metapodatka za datoteke projekta. Tu su zapisane sve aktivnosti koje aplikacija ima i time možemo odrediti koja će biti prva aktivnost koja se prikaže prilikom pokretanja aplikacije. Također se nalaze i dozvole koje su joj potrebne za rad aplikacije, npr. dozvole za pristup internetu, dozvole za čitanje/pisanje u datoteku.

```
<application
...
    android:label="@string/app_name"
  ...
    <activity android:name=".HomeworkFS"></activity>
    <activity android:name=".Homework" />
    <activity android:name=".Quiz" />
    <activity android:name=".Lesson" />
    <activity android:name=".Profile" />
    <activity android:name=".Games" />
    <activity android:name=".Memory" />
    <activity android:name=".Tenses" />
    <activity android:name=".MainActivity" />
    <activity android:name=".ProfileDetails" />
    <activity android:name=".Registration" />
    <activity android:name=".Login">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <meta-data
        android:name="preloaded_fonts"
        android:resource="@array/preloaded_fonts" />
</application>
```

Ispis 1: Android Manifest

Direktorij `java` sadrži sve klase koje su napravljene tijekom izrade projekta. `Res` direktorij ima više dijelova i može se ručno izmijeniti, ali uobičajeno je da se unutar poddirektorija `layout` nalaze sučelja pojedinih aktivnosti, `drawable` služi za spremanje slika koje se koriste u aplikaciji i `values` unutar kojeg se definiraju boje, fontovi i slično.

U ovome projektu je kreiran i dodatni direktorij `Anim` u kojemu su spremane animacije koje su korištene prilikom tranzicija. Direktorij `Gradle Scripts` je najbitniji zbog dvije važne datoteke, a to su `build.gradle-ImeAplikacije` te `build.gradle-app` . Direktorij `build.gradle-ImeAplikacije` je root datoteka projekta i podatci u njoj se odnose na sve izolirane module projekta koji se može sastojati od više njih. U `build.gradle-app` datoteci, prikazanoj na ispisu 2, nalaze se postavke samo jednog modula projekta, u ovomu slučaju moje aplikacije. Tu se navodi SDK koji aplikacija koristi, te se implementiraju ovisnosti kojima aplikacija treba imati pristup jer ih koristi u svom radu.

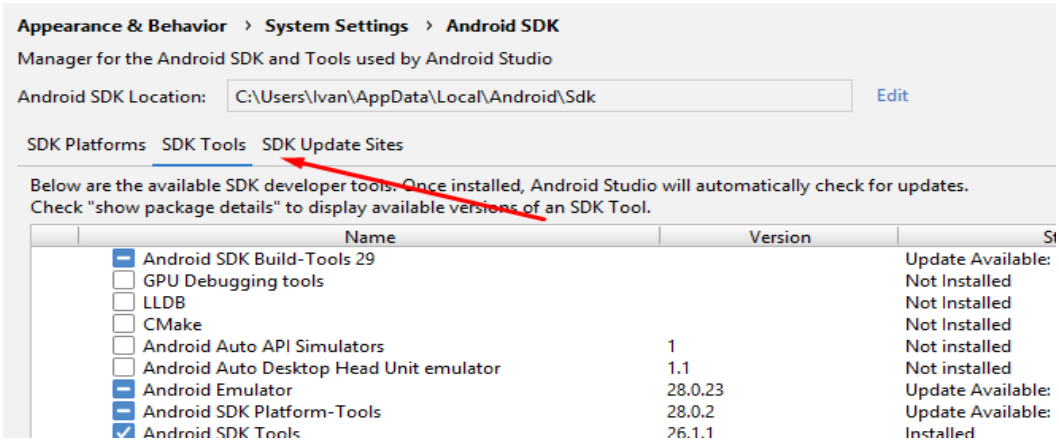
U ovome projektu dodane su podrške Android biblioteke `appcompat`, Googleov Firebase i Authentication. Skup biblioteka `appcompat` se koristi kako bi se omogućilo da aplikacije razvijene s novijim verzijama API-ja mogu raditi sa starijim verzijama. Firebase se koristi za upravljanje bazom podataka. Authentication se koristi za registriranje i prijavljivanje korisnika.

```
Android                                                                    {
    compileSdkVersion                                                      27
    defaultConfig                                                          {
        applicationId                                          "com.example.zavrsni2"
        minSdkVersion                                                      16
        targetSdkVersion                                                   27
        versionCode                                                        1
        versionName                                                        "1.0"
        testInstrumentationRunner    "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes                                                             {
        release                                                            {
            minifyEnabled                                                  false
            proguardFiles    getDefaultProguardFile('proguard-android-optimize.txt'),
```

```
            'proguard-rules.pro'
        }
    }
}
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:27.1.1'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    implementation 'com.google.firebase:firebase-database:16.1.0'
    implementation 'com.google.firebase:firebase-core:16.0.8'
    implementation 'com.android.support:cardview-v7:27.1.1'
    implementation 'com.android.support:design:27.1.1'
    implementation 'com.google.firebase:firebase-auth:16.2.0'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-
core:3.0.2'
}
```

Ispis 2: `build.gradle` datoteka

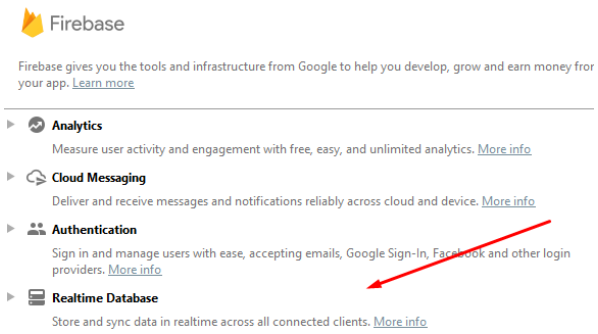
Osim instalacije samog Android Studija, potrebno je instalirati i potrebne Android SDK-ove. Upravljanje svim SDK-ovima nam pruža SDK upravitelj prikazan na slici 23. Tu se mogu vidjeti svi trenutno instalirani SDK-ovi, trenutno stanje instaliranih SDK-ova te jednostavno preuzimanje željenih.



Slika 23: Prikaz alatne trake i SDK Managera

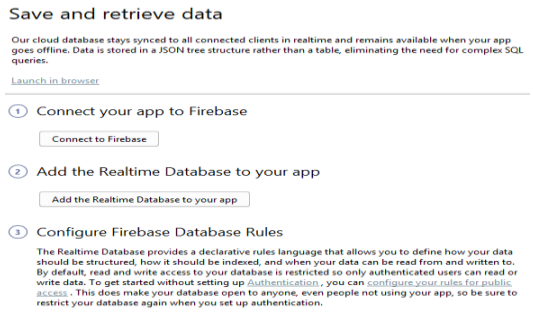
3.2. Baza podataka

S obzirom na to da aplikacija koristi Firebase kao bazu podataka za spremanje podataka, preporučeno ju je odmah povezati s aplikacijom. Za ovu aplikaciju odabran je Firebase jer je jako jednostavan i besplatan. Jedina mana je to što nije relacijska baza podataka, što predstavlja problem jer prilikom mijenjanja korisničkih podataka na jednom mjestu u bazi potrebno je promijeniti te podatke i na drugim mjestima gdje se koriste. Android Studio pruža jako jednostavan način povezivanja aplikacije s bazom. Jedino što je potrebno imati za kreiranje baze je Google račun. Odabirom izbornika *Tools*, a potom Firebase u alatnoj traci, na desnoj strani ekrana će se pojaviti prozor prikazan na slici 24.



Slika 24: Odabir baze podataka

U ovoj aplikaciji korišten je *Realtime Database*, pa pritiskom tipke *Realtime Database* otvara se novi prozor koji detaljno opisuje i vodi korisnika kroz korake potrebne za povezivanje aplikacije s bazom prikazano na slici 25.



Slika 25: Povezivanje Firebasea

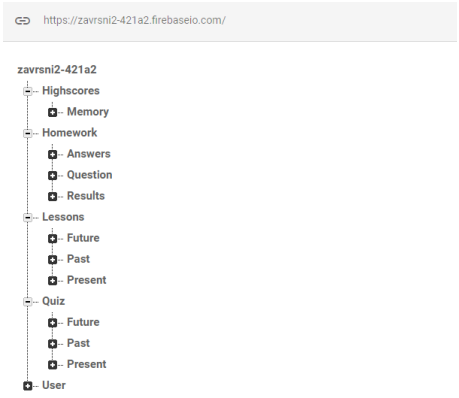
Nakon što se povežu baza i aplikacija, na stranici Firebasea je moguć uvid u trenutni projekt povezan s bazom. Tu se može napraviti nova baza koja će biti korištena za spremanje podataka iz aplikacije. Nadalje, u tom sučelju se mogu postaviti pravila tko smije koristiti bazu podataka za upisivanje, a tko za dohvaćanje podataka iz baze. To je vidljivo na ispisu 3. Za potrebe ove aplikacije postavljena su pravila da svi imaju pravo na upis podataka i dohvaćanje.

```
 /* Visit https://firebase.google.com/docs/database/security to learn more
about security rules. */
  „rules“: {
    „.read“: true,
    „.write“: true}}
```

Ispis 3: Pravila pristupa baze podataka

Ako već postoji željena baza podataka, nju se može postaviti tako da je se spremi u JSON *(JavaScript Object Notation)* formatu i postavi pritiskom tipke *import* JSON.

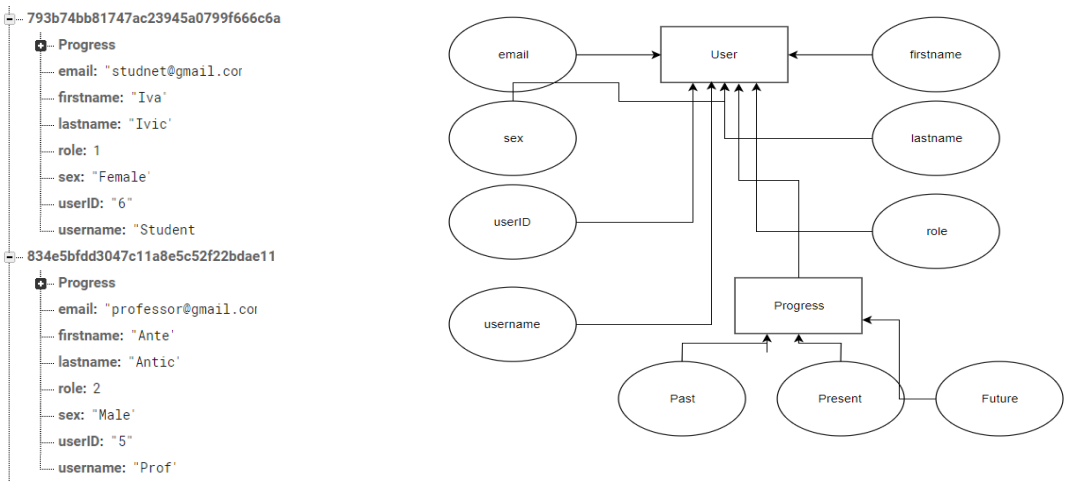
Baza podataka ove aplikacije se sastoji od nekoliko primarnih čvorova: `highscores`, `homework`, `lessons`, `quiz` i `user`. Izgled stabla je vidljiv na slici 26. Svaki od njih se grana na još nekoliko čvorova.



Slika 26: Izgled baze korištene u ovom projektu

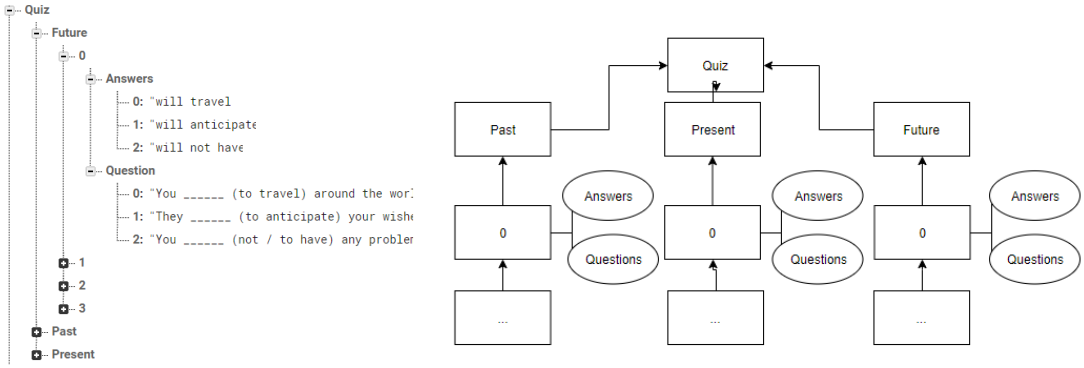
Unutar čvora `User` se nalaze svi korisnici gdje svaki korisnik ima svoj čvor. Na slici 27 se može vidjeti primjer dva korisnika. Jedan je učenik, a drugi učitelj. Osim što korisnička imena daju

do znanja tko je tko, razlika se također vidi u elementu `Role`. Ako je `Role` postavljen na dva to predstavlja učitelja, a jedan predstavlja učenika. Time je omogućeno drugačije ponašanje nekih aktivnosti ovismo o tome tko je prijavljen.



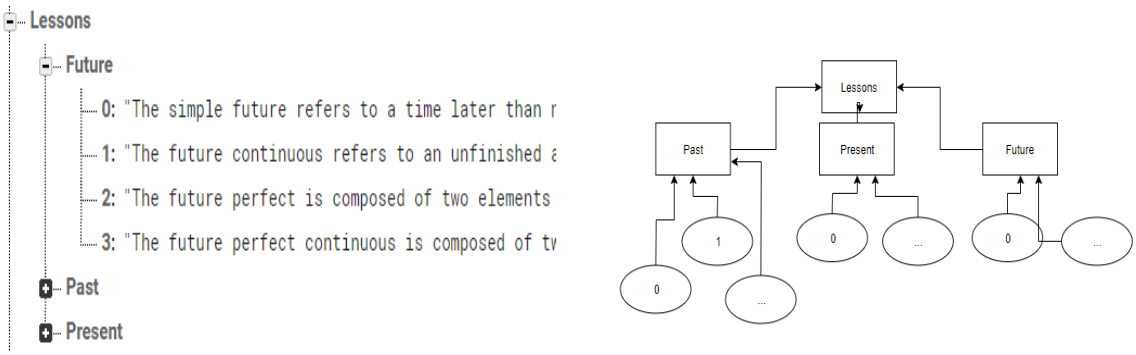
Slika 27: Primjer dva korisnika

Unutar čvora `Quiz` se nalaze pitanja i odgovori koji su potrebi za kviz koji slijedi iza svake lekcije (slika 28). Pitanja i odgovori su podijeljeni po glagolskim vremenima. Ključevi svih pitanja i odgovora kreću od nula kako bi se `for` petljom moglo pristupiti pitanjima i odgovorima.



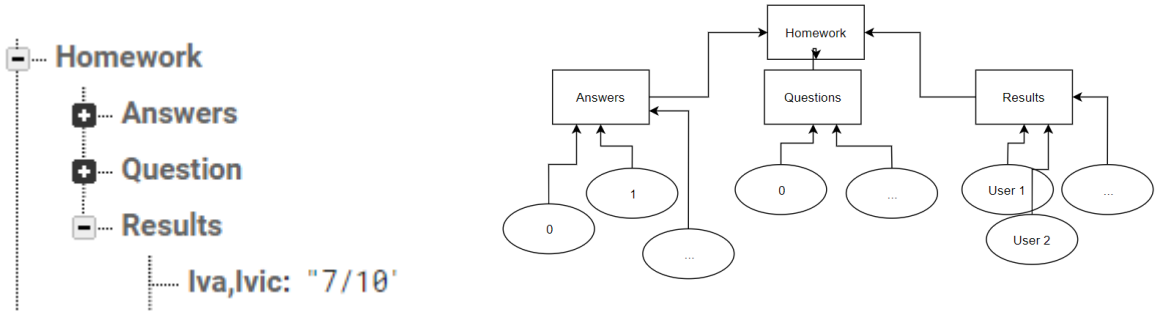
Slika 28: Izgled čvora `quiz`

Unutar čvora `lessons` (slika 29) nalazi se samo tekst koji se ispisuje na lekciji za svako glagolsko vrijeme. Podijeljen je jednako kao što su i pitanja i odgovori.



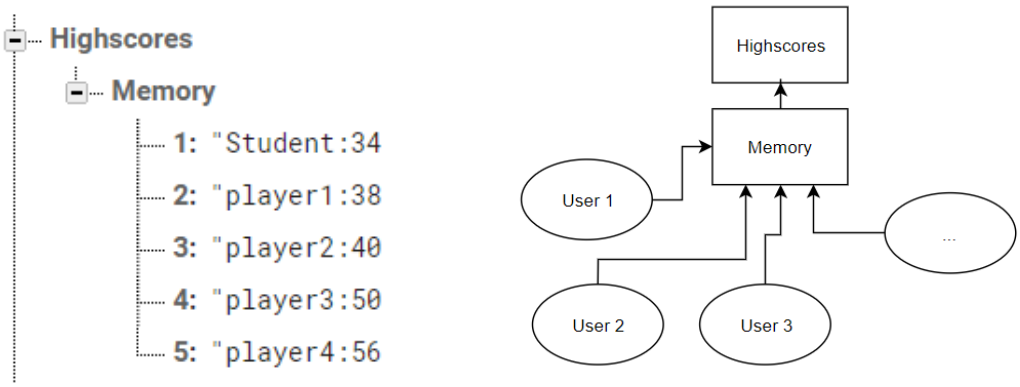
Slika 29: Izgled čvora `lessons`

Iz čvora `homework` (slika 30) se granaju još tri čvora. Prvi je `answers` u kojeg se spremaju odgovori na pitanja koja se nalaze u čvoru `question`. Drugi čvor `results` u kojem se spremaju rezultati koje učenici postignu prilikom rješavanja zadaće. Prilikom spremanja kao ključ se koristi ime i prezime učenika, a vrijednost ključa je postignuti rezultat.



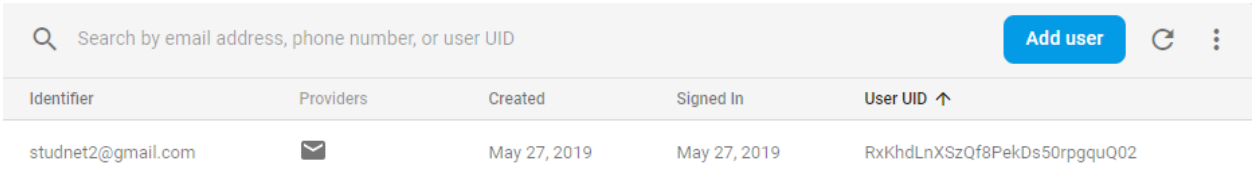
Slika 30: Izgled čvora `homework`

Posljednji čvor je `highscores`. Kao što se vidi na slici 31, u njega se spremaju najbolji postignuti rezultati u igri *Memory*. Tu su kao ključevi korišteni brojevi jer to olakšava rangiranje pri samom upisu u bazu.



Slika 31: Izgled čvora `highscores`

Još jedna jako korisna funkcionalnost koju pruža Firebase je Firebase *Authentication*. Firebase *Authentication* pruža ovjeru korištenjem elektronske pošte i lozinke, broja telefona, raznih društvenih mreža poput Facebooka, Twittera, Googlea i sličnih. S obzirom na to da ova aplikacija zahtijeva registriranje i logiranje, ti zahtjevi su implementirani koristeći adresu elektroničke pošte i lozinku. Prilikom registracije od korisnika se zahtijeva jedinstvena adresa elektroničke pošte koja nije već korištena i lozinka od najmanje osam znakova. Prilikom procesa registracije korisnički podatci se upisuju u Firebase te se lozinka zaštićuje funkcijom `Bcrypt`. Funkcija `Bcrypt` korištenjem soli i kriptografskih jednosmjernih funkcija povećava sigurnost lozinke i time štiti korisnički profil od hakiranja. Na slici 32 je prikazan izgled jedne adrese spremljene u Firebase i podataka koji se prate.



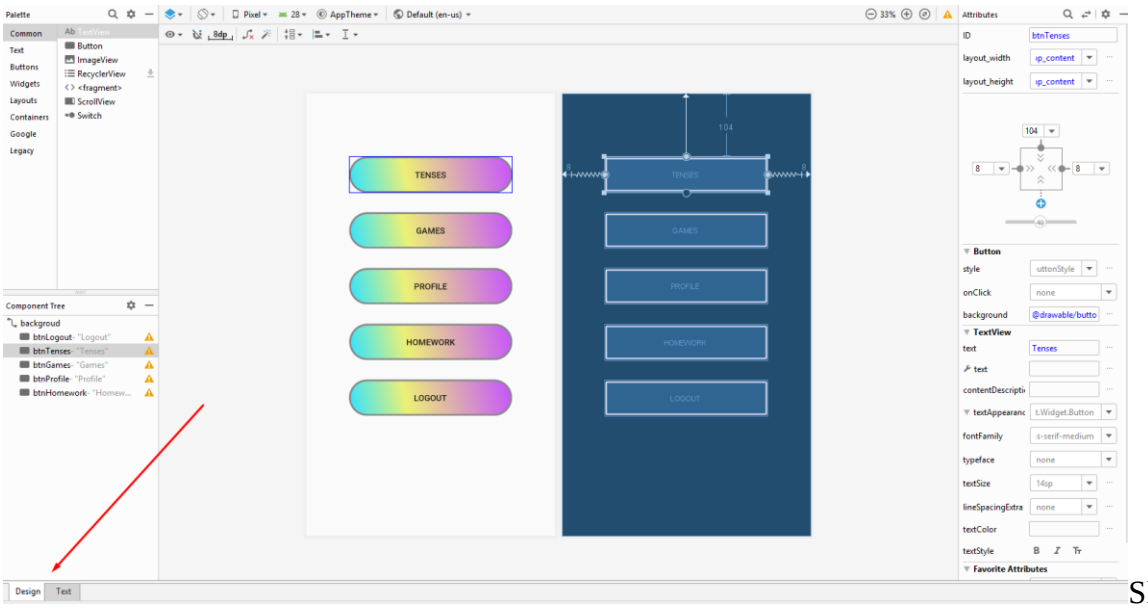
Slika 32: Adresa spremljena u Firebase

3.3. Implementacija

3.3.1. Sučelje

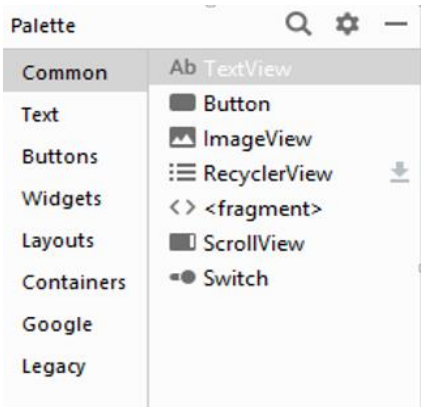
Kao što je već spomenuto u opisu aplikacije s korisničke strane, aplikacija se sastoji od nekoliko aktivnosti koje su međusobno povezane. Svaka od tih aktivnosti ima svoju .XML

datoteku spremljenu u `Layout` direktoriju koja aplikaciji služi za generiranje izgleda zaslona te aktivnosti. Prilikom kreiranja nove aktivnosti, automatski se generira i zadana .XML datoteka ovisno o tome koji je izgled aktivnosti izabran. Kôd u .XML datoteci se može uređivati na dva načina; u *design* i *text* načinu rada.



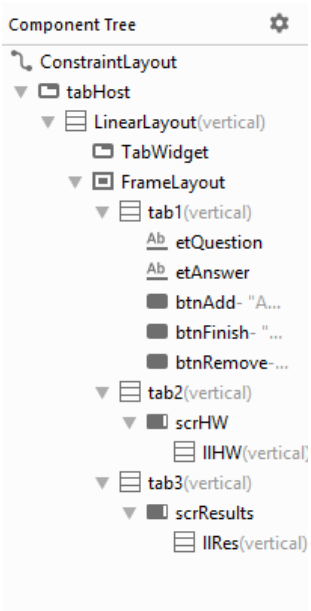
Slika 33: Korisničko sučelje Android Studija

Ukoliko je odabran *design* način rada prikazan na slici 33, s lijeve strane će se pojaviti alatna traka koja je vidljiva na slici 34, u kojoj se nalaze svi mogući elementi koji se mogu dodati u trenutnu aktivnost poput tipki, prostora za slike, prostora za tekst, linija za unošenje teksta i sličnog. To se radi tako da se pokazivačem pritisne na željeni element i samo ga se odvuče na željeno mjesto na aktivnosti.



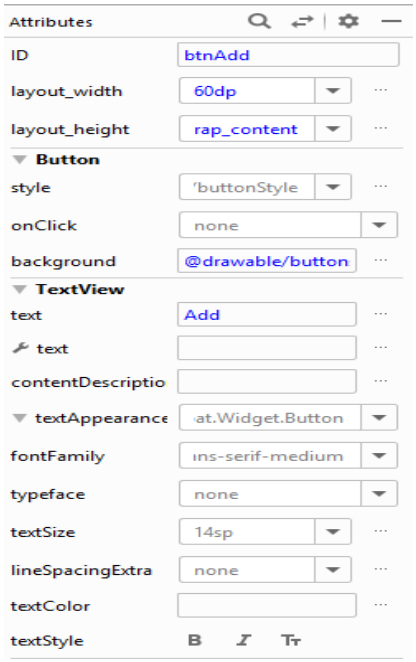
Slika 34: Alatna traka s elementima

Ispod alatne trake se nalazi popis svih elemenata koji se trenutno nalaze u aktivnosti, što je vidljivo na slici 35. Oni su poredani hijerarhijski, a svi elementi koji se dodaju nalaze se ispod primarnog rasporeda kojeg smo izabrali prilikom kreiranja aktivnosti.



Slika 35: Hijerarhijski popis elementa

S desne strane se nalazi traka prikazana na slici 36. U njoj se nalaze sva svojstva elementa kojeg trenutno pregledajmo. Tu se mogu upisati svojstva kao što su ime elementa koje se kasnije koristi za dohvaćanje, veličina elementa, zadana vrijednost elementa, natuknice koju će element prikazivati i slično.



Slika 36: Detalji odabranog elementa

Ako pak izaberemo *Text* način rada, onda se pred korisnikom otvori prozor u kojeg je potrebno pisati XML kôd za svaki pojedini element koji želimo dodati. To zahtjeva više znanja i iskustva, ali daje pregledniji uvid u detalje svakog elementa. Na ispisu 4 je prikazan primjer elementa `TextView`. Desno od tog prozora se nalazi dio u kojem se može vidjeti vizualni pregled aktivnosti i tako se može pratiti da li sve napisano odgovara kreativnim željama.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    <TextView
        android:id="@+id/textView8"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="TextView"
        tools:layout_editor_absoluteX="176dp"
        tools:layout_editor_absoluteY="155dp" />
</android.support.constraint.ConstraintLayout>
```

Ispis 4: *Text* prikaz elementa

Također postoji i mogućnost da se dinamički dodaju elementi u aktivnost ukoliko postoji potreba za time. Primjer implementacije toga vidljiv je na ispisu 5. U aktivnosti za zadaću je

korišten takav način kreiranja elementa jer se prilikom pokretanja aktivnosti ne zna unaprijed koliko pitanja će biti, pa zato prilikom pokretanja te aktivnosti moraju se prebrojati pitanja i kreirati toliko elemenata za ispis teksta.

```
@Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
    for(long I =0;i<number;i++) {
        TextView tv1 = new TextView(HomeworkFS.this);
        tv1.setText("Question "+ I +" :
"+dataSnapshot.child("Homework").child("Question").child(String.valueOf(String.value
Of(i))).getValue().toString());
        scroll.addView(tv1);
    }
}
```

Ispis 5: Dinamičko kreiranje elementa

3.3.2. Programski dio

Nakon uređenja izgleda aktivnosti u XML datoteci potrebno je napisati kôd u Java direktoriju kojim se upravlja tim elementima. Ukoliko se želi upravljati s elementima u XML datoteci, u java datoteci je potrebno kreirati objekte koji odgovaraju istoj klasi kojom se želi upravljati. To se radi tako da se kreiranom objektu neke klase pridruži referenca koju vrati metoda `findViewbyId` i pretvori (eng. *cast*) u željeni objekt, prikazano na ispisu 6. U funkciju `FindViewById` šalje se ime traženog elementa te ona pregleda sadržaj postavljen funkcijom `setConentView`.

```
setContentView(R.layout.activity_homework);
final EditText etQuetion = (EditText) findViewById(R.id.etQuestion);
final EditText etAnswer = (EditText) findViewById(R.id.etAnswer);
final ArrayList<String> Question = new ArrayList<String>();
final ArrayList<String> Answers = new ArrayList<String>();
```

Ispis 6: Kreiranje objekata

Kako je navedeno u `AndroidManifestu` aktivnost koja se prva pokrene prilikom pokretanja aplikacije je `Login` aktivnost. Pri pokretanju te aktivnosti provjerava se da li je korisnik

prijavljen tako što se kreira objekt klase `FirebaseAuth` koji služi da bi se mogle koristiti metode klase `Firebase Authenticationa` kao što su prijavljivanje, registriranje korisnika te dohvaćanje trenutnog korisnika. Ako postoji prijavljeni korisnik, korisnika vodimo na aktivnost za glavni izbornik, što je vidljivo na ispisu 7.

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

Ispis 7: Provjera da li je korisnik prijavljen

Ukoliko korisnik nije već prijavljen ostaje na ovoj aktivnosti i kako bi se prijavio mora upisati svoju adresu i lozinku. Nakon upisa i pritiska tipke `Login` poziva se funkcija `userLogin()` prikazana na ispisu 8.

```
private void userLogin() {
    if (TextUtils.isEmpty(Username)) {
        Toast.makeText(this, "Enter E-mail", Toast.LENGTH_SHORT).show();
        return;
    }
    if (TextUtils.isEmpty(Password)) {
        Toast.makeText(this, "Enter password", Toast.LENGTH_SHORT).show();
        return;
    }
    firebaseAuth.signInWithEmailAndPassword(Username, Password)
            .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    progressDialog.dismiss();
                    if (task.isSuccessful()) {
                        finish();
                        startActivity(new Intent(getApplicationContext(),
        MainActivity.class));
                        slideInTransition();
                    }
```

Ispis 8: Prijavljivanje korisnika

U njoj se dohvaćaju podatci koje je korisnik upisao i šalju metodi `signInWIthEMailAndPassword(Username, Password)`, koja provjerava ispravnost adrese i lozinke, te ako je upisana postojeća adresa i točna lozinka, pokreće aktivnost glavnoga izbornika. Ako dođe do neke greške prilikom prijavljivanja to se prikaže u `else` dijelu petlje tako što uhvatimo iznimku (ispis 9).

```
else {
    String errorCode = ((FirebaseAuthException) task.getException()).getErrorCode();
    switch (errorCode) {
        case "ERROR_INVALID_EMAIL":
            Toast.makeText(Login.this, "The email address is badly formatted.",
Toast.LENGTH_LONG).show();
            etUsernameLogin.setError("The email address is badly formatted.");
            etUsernameLogin.requestFocus();
            break;
    }
```

Ispis 9: Dohvaćanje greške prilikom prijavljivanja

Na sličnom principu funkcionira i aktivnost za registraciju koja je vidljiva na ispisu 10. Razlikuje se u tome što ima nekoliko provjera više prilikom upisa adrese elektronske pošte i lozinke. Klikom na tipku `register` provjeravaju se korisnikovi uneseni podatci. Zahtijeva se da unesena adresa odgovara izgledu prave adrese elektronske pošte, da se unese adresa koja nije već korištena za pravljenje profila, lozinka koja mora biti duljine 8 ili više znakova te potvrda te iste kako ne bi došlo do greške prilikom odabira lozinke. Nakon uspješno odrađene provjere, poziva se metoda `createUserWithEmailAndPassword(Email, Password)` u koju se prosljeđuju podatci koje je korisnik unio i time se kreira korisnikov profil na `Firebase Authenticationu`.

```
if (TextUtils.isEmpty(Email)) {
    Toast.makeText(this, "Enter E-mail", Toast.LENGTH_SHORT).show();
    edUsername.setError("Enter E-mail");
    edUsername.requestFocus();
    return;
} else {
    progressDialog.setMessage("Making your profile...");
    progressDialog.show();
    firebaseAuth.createUserWithEmailAndPassword(Email, Password)
```

```
            .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    progressDialog.dismiss();
                    if (task.isSuccessful()) {
                        finish();
                        startActivity(new Intent(getApplicationContext(),
ProfileDetails.class));
                    } else {
                        String errorCode = ((FirebaseAuthException)
        task.getException()).getErrorCode();
                        switch (errorCode) {

                            case "ERROR_INVALID_EMAIL":
                                Toast.makeText(Registration.this, "The email address
is badly formatted.", Toast.LENGTH_LONG).show();
                                edUsername.setError("The email address is badly
formatted.");
                                edUsername.requestFocus();
                                break;
                        }
                    }
                }
            });
```

Ispis 10: Provjera podataka prilikom kreiranja profila i kreiranje profila

Nakon toga korisnika se preusmjerava na aktivnost na kojoj mora unijeti neke osnovne informacije o sebi. Kako bi pohrana podataka korisnikovog profila u bazu podataka bila jednostavnija, napravljena je klasu `User` (ispis 11) koja se sastoji od svih potrebnih informacija, konstruktora te metoda za postavljanje (eng. *Setter*) i dohvaćanje (eng. *Getter*) vrijednosti.

```
public class User {
    private String username;
    private String firstname;
    private String lastname;
    private String sex;
    private String userID;
    private String email;
    private Integer role;
    private Integer score;
```

```
    public User(){
    }
    public User(String username, String firstname, String lastname, String sex) {
        ...
    }
    public User(String username, String firstname, String lastname, String sex,
String userID, String email, Integer progress, Integer score) {
        this.username = username;
        this.firstname = firstname;
        ...
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
}
...
```

Ispis 11: Primjer klase `User`

Prilikom spremanja podataka u bazu, prvo se kreira objekt klase `DatabaseReference` te se poziva njegova metoda. Taj objekt pokazuje na točno određenu lokaciju unutar baze. Nakon njegove inicijalizacije mogu se koristiti njegove metode za čitanje i pisanje podataka u bazu. Koristeći metodu `child` kreira se putanju unutar baze. Vrijednost čvora u bazi postavlja se tako da mu se dodijeli vrijednost dobivena unošenjem korisnikove adrese u kriptografski jednosmjernu funkciju, te se unutar njega pomoću metode `setValue` i klase `User` postavljaju sve vrijednosti koje je korisnik unio, što je prikazano na ispisu 12.

```
DatabaseReference reff;
reff=FirebaseDatabase.getInstance().getReference().child("user");
reff.child(Username).setValue(user);
```

Ispis 12: Kreiranje objekta klase `user`

Funkcija za glavni izbornik se sastoji od tipki koje vode do drugih aktivnosti. Prilikom kreiranja funkcionalnosti tipki potrebno je nadjačati (eng. *override*) metodu `OnClick` klase `View`, što je vidljivo na ispisu 13. To je moguće zato što je klasa `View` roditeljska klasa klase

`Button` pa klasa `Button` nasljeđuje tu funkciju. Nadjačavanjem omogućimo otvaranje željene aktivnosti pritiskom na tipku.

```
btnProfile.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Animation anim = AnimationUtils.loadAnimation(MainActivity.this,
R.anim.scale2);
        view.startAnimation(anim);
        Handler handler = new Handler();
        handler.postDelayed(new Runnable() {
            @Override
            public void run() {
                startActivity(new Intent(getApplicationContext(), Profile.class));
                slideInTransition();
            }
        }, 800);}});
```

Ispis 13: Kreiranje funkcionalnosti tipke i postavljanje animacije

U kôdu ispisa 13 se može vidjeti i da se unutar metode `onClick` kreira objekt klase `Animation`, koji koristi metodu `loadAnimation` klase `AnimationUtils` koja za povratnu vrijednost ima objekt klase `Animation`. Metodom `startAnimation` se poziva ta animacija spremljena u objektu `anim`. Ova animacija se odnosi na samu tipku `Button`. Kako bi se sama animacija vidjela moramo kreirat objekt klase `Handler` koji će odgoditi prijelaz na drugu aktivnost za određeno vrijeme. Tu postoji još jedna funkcija `slideInTransition()` čiji kôd je prikazan na ispisu 14. Ona je također animacija, ali se ona odnosi na samu tranziciju između aktivnosti.

```
protected void slideInTransition() {
    overridePendingTransition(R.anim.slide_in_right, R.anim.fade_back);
}
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <translate android:fromXDelta="100%p" android:toXDelta="0"
        android:duration="750"/>
</set>
```

Ispis 14: Primjer kôda animacije

U aktivnosti `tenses`, korisnik ima izbor koje glagolsko vrijeme želi rješavati. Na ispisu 15 možemo vidjeti da se korisnikov odabir šalje u sljedeću aktivnost `lesson` koristeći metodu `putExtra` koja prima dvije varijable. Prva varijabla predstavlja ključ koji koristimo za dohvatiti drugu varijablu koja je vrijednost. Aktivnosti `lesson` i `quiz` služe kao predlošci koji se mijenjaju ovisno u korisničkom odabiru.

```
Intent intent = new Intent(getApplicationContext(), Lesson.class);
intent.putExtra("time", "Past");
startActivity(intent);
```

Ispis 15: Pokretanje aktivnosti

Prilikom otvaranja aktivnosti `lesson` prvo se dohvaćaju poslane vrijednosti preko ključeva time i `id` (ispis 16). Prilikom prvog pokretanja aktivnosti `lesson` ključ `id` neće biti poslan pa zbog toga ima zadanu vrijednost 0, a ona se koristi za koordiniranje između pojedinih lekcija i zadataka koje korisnik rješava. Ona se svaki put šalje iz aktivnosti `lessona` u aktivnost `quiz` i obrnuto, ali samo prilikom slanja iz `quiz` u `lesson` vrijednost se povećava za jedan (ispis 17).

```
LessonId = getIntent().getIntExtra("id", 0);
timeid = getIntent().getStringExtra("time");
```

Ispis 16: Dohvaćanje vrijednosti

```
intent.putExtra("id", quizID + 1);
intent.putExtra("time", timeid);
```

Ispis 17: Postavljanje vrijednosti

Za dohvaćanje podataka iz baze podataka koristimo sučelje (eng. *Interface*) `ValueEventListener`. Ono ima dvije metode koje moramo nadjačati a to su `OnDataChange` i `OnCancelled`. Metodi `onDataChange` šaljemo objekt klase `DataSnapshot`, jer svako čitanje podataka iz baze podataka vraća objekt klase `DataSnapshot`. `DataSnapshot` se koristi samo prilikom dohvaćanja podataka iz baze, za upisivanje, mijenjanje i brisanje se koristi već navedeni `DatabaseReference`. Nakon dohvaćanja teksta iz baze on se ispisuje na ekran, slovo po slovo kreirajući animaciju pisanja teksta. Cijeli proces je vidljiv na ispisu 18.

```
final String Text = dataSnapshot.child(LessonId.toString()).getValue().toString();
Thread thread = new Thread() {
            @Override
            public void run() {
                try { for (nti  = 0; i < Text.length(); i++) {
                        Thread.sleep(65);
                        runOnUiThread(new Runnable() {
                            @Override
                            public void run() {
                                tvLesson.setText(Text.substring(0, i));
                            }});}}}};thread.start();
```

Ispis 18: Ispis teksta iz baze

U aktivnosti `quiz`, na isti način se dohvaćaju podatci iz baze samo se ovoga puta dohvaćaju pitanja i odgovori. Pitanja se postavljaju u kućice za tekst, a odgovori u niz `answers`. Pritiskom tipke `continue`, provjerava se ispravnost odgovora funkcijom `checkanswers`() koja prima točan odgovor, korisnikov odgovor i poziciju kućice. Ukoliko je odgovor točan mijenja se boja teksta u zeleno i omogućuje se nastavak rješavanja kviza. Ukoliko je odgovor netočan, polja koja su netočna mijenjaju boju teksta u crvenu i korisniku se onemogućuje nastavak dok ne odgovori točno na pitanja. Prikaz kôda je vidljiv na ispisu 19.

```
 public void onClick(View view) {
            String A[]=new String[3];
            for(int x=0;x<3;x++) {
                A[x]= (etA[x].getText().toString().trim());
            }
            for(int x=0;x<3;x++) {
                checkanswers(Answers[x], A[x],x);
            }
        }
    });
}
private boolean checkanswers(String rightA, String userA, Integer nu){
    if (rightA.equalsIgnoreCase(userA)) {
        etA[nu]. setTextColor (Color.GREEN);
    } else {
        etA[nu]. setTextColor (Color.RED);
        btnContinue.setTextColor(Color.RED);
```

```
    }
    return true;
}
```

Ispis 19: Dohvaćanje odgovora i provjera

Ako su sva tri odgovora točna, poziva se funkcija `checknext()` koja je prikazana na ispisu 20. Ona provjerava da li je korisnik došao do kraja lekcija odabranog glagolskog vremena, te ako je vraća ga se na glavni izbornik, a inače ga se šalje ponovo na aktivnost `lessons` samo s povećanim `quiz ID`-om.

```
private void checknext() {
    if (timeid.equals("Past")) {
        if (quizID.toString().equals("5")) {
            Intent intent2 = new Intent(getApplicationContext(),
MainActivity.class);
            finish();
            startActivity(intent2);
        }
}
```

Ispis 20: Provjera kraja kviza

Ukupan broj riješenih lekcija korisnik može vidjeti u aktivnosti `profile`. Tamo se osim korisničkih podataka, iz baze podataka dohvaća i broj riješenih lekcija. Taj broj služi i za omogućavanje preskoka neke lekcije ukoliko je već riješena. Na aktivnosti `profile` korisniku je također omogućeno i mijenjanje nekih detalja profila kao što su ime, prezime i korisničko ime.

S obzirom na to da baza nije relacijska, prilikom mijenjanja korisničkih podataka koji se koriste još negdje u aplikaciji, ti podatci se moraju promijeniti i drugim mjestima gdje se koriste, kao na primjer u rezultatima zadaće. Rezultat zadaće se sprema tako da ključ bude korisničko ime i prezime odijeljeno imenom, a vrijednost predstavlja postignuti uspjeh iz zadaće.

Ime se ažurira tako da se pozove funkcija `updatename()` prikazana na ispisu 21. Zatim se u bazi traži ključ koji odgovara starom imenu i prezimenu korisnika poslanih u funkciju. Ukoliko se nađe ključ koji odgovara tim vrijednostima, korisnikov rezultat se sprema u trenutnu

varijablu stari čvor se briše i kreira novi čija vrijednost ključa odgovara izmijenjenim vrijednostima imena korisnika, a vrijednost spremljena u trenutnu varijablu ostaje ista.

```
public void updatename(final String oldn1,final String oldn2,final String
newn1,final String newn2) {
    final DatabaseReference reff3 = FirebaseDatabase.getInstance().getReference();
    reff3.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            if (dataSnapshot.hasChild("Homework")) {
                if (dataSnapshot.child("Homework").hasChild("Results")) {
                    for (DataSnapshot child :
        dataSnapshot.child("Homework").child("Results").getChildren()) {
                        String key = child.getKey();
                        String Value =
        dataSnapshot.child("Homework").child("Results").child(key).getValue().toStr
        ing();
                        String temp = oldn1 + "," + oldn2;
                        Toast.makeText(getApplicationContext(), key+"  "+temp,
        Toast.LENGTH_SHORT).show();
                        if (key.equals(temp)) {
reff3.child("Homework").child("Results").child(key).removeValue();
                            reff3.child("Homework").child("Results").child(newn1 +
        "," + newn2).setValue(Value);
                            reff3.removeEventListener(this);
                        }
                    }
                }
        }
}
```

Ispis 21: Funkcija `updatename`

Aktivnosti `homework` i `homeworkFS` se pokreću pritiskom na tipku `homework`, a aktivnost koja se otvara ovisi o `id`-u korisnika. Razliku vidimo na ispisu 22.

```
if(Role.equals(1)){
            startActivity(new Intent(getApplicationContext(), HomeworkFS.class));
            slideInTransition();
    }else if(Role.equals(2)){
    {startActivity(new Intent(getApplicationContext(), Homework.class));
```

```
            slideInTransition();
        }}
```

Ispis 22: Provjera varijable `role` aktivnog korisnika

Prilikom pokretanja aktivnosti `homeworkFS`, provjerava se u bazi da li je student već rješavao tu zadaću i ako je, zabranjuje mu se pristup toj aktivnosti i vraća ga se na glavni izbornik. To je prikazano na ispisu 23.

```
public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
if(dataSnapshot.child("Homework").child("Results").hasChild(firstname+","+lastname))
{     finish();
}
```

Ispis 23: Provjera da li je korisnik već rješavao zadaću

Ukoliko je to prvi put da korisnik rješava zadaću, korisniku će se na aktivnosti pojaviti sva pitanja poredana redoslijedom kao u bazi te se iza svakog pitanja nalazi linija na koju korisnik može unijeti svoj odgovor. Prilikom dohvaćanja pitanja iz baze, dohvatit će se i odgovori, koji se spremaju u listu `Answers` i kasnije služe za provjeru ispravnosti korisnikovih odgovora. Nakon pritiska tipke `finish` dohvatit će se upisani korisnikovi odgovori, usporedit s ispravnima i rezultat će se upisati u bazu, što je vidljivo na ispisu 24.

```
public void onClick(View view) {
        int n=Integer.valueOf(String.valueOf(number));
        for(int i =0;i<n;i++) {
         if(etA.get(i).getText().toString().trim().equalsIgnoreCase(Answers.get(i)))
{
                correct = correct + 1;
            }
        }
reff.child("Homework").child("Results").child(firstname+","+lastname).setValue(corre
         ct+"/"+number);
        finish();
        startActivity(new Intent(getApplicationContext(),MainActivity.class));
    }
});
```

Ispis 24: Provjera ispravnih odgovora i upis rezultata u bazu

S korisničke strane učitelja aktivnost homework se sastoji od tri kartice. Za izradu kartica korištena je TabHost klasa vidljiva na ispisu 25.

```
 TabHost.setup();
TabHost.TabSpec spec = TabHost.newTabSpec("Tab One");
spec.setContent(R.id.tab1);
spec.setIndicator("New");
TabHost.addTab(spec);
...
```

Ispis 25: Kreiranje TabHosta

Prva kartica služi za izradu zadaće i spremanje u bazu, druga služi za uvid u trenutna pitanja postavljena u bazi a treća za pregled rezultata učenika koji su riješili zadaću. Prilikom pisanja zadataka za zadaću, korisnik mora unijeti pitanje i odgovor na to pitanje nakon čega pritiskom tipke add, to pitanje i odgovor se spremaju u listu question i answers, a linije za upis se očiste. Nakon što je korisnik unio sva pitanja klikom na tipku finish ta pitanja se postavljaju u bazu. To je prikazano na ispisu 26. Nije dopušteno dodavati ukoliko nije uneseno niti jedno pitanje.

```
if (Question.isEmpty()) {
    Toast.makeText(Homework.this, "Enter at least one question",
Toast.LENGTH_LONG).show();

} else {
    reff = FirebaseDatabase.getInstance().getReference();
    for (int i = 0; i < Question.size(); i++) {
        reff.child("Homework").child("Question").child(String.valueOf(i)).setValue(
        Question.get(i));

        reff.child("Homework").child("Answers").child(String.valueOf(i)).setValue(A
        nswers.get(i));
    }
```

Ispis 26: Unos pitanja za zadaću

Ako je već postoji neka zadaća u bazi, potrebno je tipkom delete izbrisati staru zadaću kako ne bi došlo do nekih preklapanja. Kôd je prikazan na ispisu 27.

```
reff = FirebaseDatabase.getInstance().getReference();
reff.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        if (dataSnapshot.hasChild("Homework")) {
            reff.child("Homework").removeValue();
            reff.removeEventListener(this); }}
```

Ispis 27: Brisanje stare zadaće

U drugoj kartici pitanja se dohvaćaju na isti način kao i kod učenika. S obzirom na to da je za spremanje rezultata u bazu kao ključ korišteno ime i prezime korisnika, onda se u trećoj kartici moraju dohvatiti ti ključevi kako bi se podatci mogli prikazati. To se radi pomoću for petlje koja dohvati sve čvorove putanje koju smo joj poslali. To se vidi na ispisu 28.

```
if (dataSnapshot.child("Homework").hasChild("Results")) {
    for (DataSnapshot child :
dataSnapshot.child("Homework").child("Results").getChildren()) {
        String key = child.getKey();
        String Value =
dataSnapshot.child("Homework").child("Results").child(key).getValue().toString();
        TextView tv2 = new TextView(Homework.this);
        tv2.setText(key + " " + Value+"\n");
        tv2.setTextSize(20);
        Results.addView(tv2);
```

Ispis 28: Dohvaćanje rezultata

`Games` aktivnost služi za odabir igre i uvid u najbolje rezultate. Rezultati su upisani u bazi i dohvaća ih se brojevima koji služe i za rangiranje liste. Na njoj se nalazi i tipka za pokretanje igre *Memory*. Prilikom pritiska tipke, korisnika se pita da odabere broj igrača za igru koji može biti jedan ili dva. Njegov odabir se šalje u sljedeću aktivnost putem metode `putExtra`. Prilikom pokretanja aktivnosti *Memory* dohvaća se korisnikov odabir i skrivaju se određeni dijelovi aktivnosti ovisno o tome koliko igraća igra. Ako igra jedan igrač pokazuje se brojač njegovih koraka, a ako igraju dva igrača prikazuju se pojedini bodovi koje su igrači skupili. S obzirom na to da se pokreće ista aktivnost za jednog i dva igrača potrebno je sakriti određene dijelove aktivnosti ovisno o korisnikovom odabiru, što je prikazano na ispisu 29.

```
if (sessionId == 1) {
    tvp1.setVisibility(View.INVISIBLE);
    tvp2.setVisibility(View.INVISIBLE);
}
else {
    tvCounter.setVisibility(View.INVISIBLE);
}
```

Ispis 29: Skrivanje dijela aktivnosti

Poslije toga se kreira matrica `img[][]` i koristeći funkciju `getIdentifier`, koja preko `id`-a pretražuje `Res` direktorij, se uparuje s pojedinim mjestima definiranim u .XML datoteci. U niz `cardsArray` smo spremili niz brojeva koji ćemo koristit kasnije za uspoređivanje karata. Taj niz izmiješamo koristeći metodu `shuffle`, što nam omogućuje različito pozicioniranje karata u matrici. Također imamo i dva cjelobrojna (eng. *integer*) niza koja koristimo za spremanje identifikacijskih brojeva slika koje nasumično odaberemo. Pozivom funkcije `cardsResources()` punimo ta dva niza adresama nasumicno odabranim slikama i slika teksta koji predstavljaju te slike, što je prikazano na ispisu 30.

```
Integer[] size =generaterandom();
for (int i = 0; i < 10; i++) {
    filename = "im" + size[i];
    int id = getResources().getIdentifier(filename, "drawable",
Memory2.this.getPackageName());
    imagearray1[i] = id;
    filename = "im" + size[i] + "i";
    id = getResources().getIdentifier(filename, "drawable",
Memory2.this.getPackageName());
    imagearray2[i] = id;
}
```

Ispis 30: Kreiranje liste karata

Klikom na neki od mjesta na matrici na ekranu se poziva funkcija `clicked()` koja prima objekt klase `view`, odnosno tu sliku koja je pritisnuta i `tag` te slike koji je postavljen u XML-u. Pomoću funkcije `showfront(ImageView iv, int card)` na pritisnuto mjesto se postavlja određena slika koju je potrebno upariti. To se radi tako da se prolazi po cijelom nizu `cardsArray` i pomoću vrijednosti `card` pronalazi tražena vrijednost. Tako osiguravamo da se svaka slika pojavi samo na jednom mjestu u matrici. Nakon što smo pritisnuli dvije slike

moramo kreirati `Handler` koji će odužiti vrijeme provjere za sekundu jer inače korisnik ne bi vidio koja je druga slika pritisnuta. Za to vrijeme potrebno je onemogućiti pritiskanje ijedne druge kartice. Taj proces je prikazan u ispisu 31.

```
for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 4; j++) {
            img[i][j].setEnabled(false);
        }
    }
    Handler handler = new Handler();
    handler.postDelayed(new Runnable() {
        @Override
        public void run() {
            equal();
        }
    }, 1000);
}
```

Ispis 31: Kreiranje matrice koja se koristi za igranje

Unutar objekta `Handler` pozivamo funkciju `equal()` koja provjerava da li odabrane slike odgovaraju jedna drugoj. Unutar funkcije `equal()` se poziva funkcija `remove(Integer clicked)` koja prima pritisnutu vrijednost te nakon provjere postavlja slike na `Invisibile` ukoliko ih je korisnik uspješno upario, što je vidljivo na ispisu 32.

```
if (firstcard == secondcard) {
    remove(clickedfirst);
    remove(clickedsecond);
private void remove(Integer clicked) {
    int ct = 0;
    for (int d = 0; d < 5; d++) {
        for (int o = 0; o < 4; o++) {
            if (clicked == ct) {
                img[d][o].setVisibility(View.INVISIBLE);
            }
            ct++;
        }
    }
}
```

Ispis 32: Provjera da li su izabrane odgovarajuće slike

Funkcija `check()` provjerava da li su sve slike postavljene na `Invisible` i ukoliko jesu to označava kraj igre. To je prikazano na ispisu 33. Nakon kraja igre na ekran se ispisuje pobjednik ukoliko se igra igrala u dvoje ili konačni rezultati koji se upisuje u bazu ako je u pet najboljih osvojenih rezultata.

```
private boolean check() {
    boolean flag = true;
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 4; j++) {
            if (!(img[i][j].getVisibility() == View.INVISIBLE)) {
                flag = false;
            }
        }
    }
    return flag;
```

Ispis 33: Provjera kraja

Ukoliko provjera bude bezuspješna, odnosno korisnik krivo upari slike, na trenutno prikazane slike se ponovo postavlja slika pozadine kako bi sakrili pravu sliku. Nakon toga se ponovo omogućuje korištenje svih slika, što je prikazano na ispisu 34.

```
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 4; j++) {
        img[i][j].setImageResource(R.drawable.back1);
    }
}
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 4; j++) {
        img[i][j].setEnabled(true);
    }
}
```

Ispis 34: Kreiranje matrice koja se koristi za igranje

4. Zaključak

U ovom radu objašnjena je uporaba i proces izrade Java aplikacije EasyLangugae. Temelj ovoga rada je aplikacija za pametne telefone EasyLanguage. Zamisao koja je potaknula kreiranje ovog rada je činjenica da se učenje u školama u posljednje vrijeme jako modernizira. Ovakva aplikacija je zamišljena kao dodatak u nastavnom programu, što može pružiti jednu zabavnu prijenosnu alternativu učenju.

Uporaba aplikacije se pokazala poprilično jednostavnom i intuitivnom zbog načina na koji su kreirana grafička sučelja s kojima korisnik ima interakciju. Korisnicima se najviše svidio dio s igrom *Memory* jer su se tu mogli međusobno natjecati. Kreiranje osnovnog grafičkog sučelja se pokazalo poprilično jednostavnim zadatkom jer integrirano razvojno okruženje Android Studio pruža sve potrebne alata za izradu istog. Pravi izazov se pokazalo kodiranje pojedinih dijelova aplikacije te njihovo povezivanje u jednu kohezivnu cjelinu.

Odabir Androida kao operacijskog sustava korištenog za kreiranje ovog završnog rada je potaknuto time što je Android sveprisutan u današnjem svijetu i postotak tržišta kojeg pokriva svake godine raste. Takve stavke su vjerojatno plod činjenice da je Android operacijski sustav otvorenog kôda te da sve kompanije i korisnici koji ga žele koristiti mogu podešavati operacijski sustav vlastitim željama.

Tijekom procesa izrade same aplikacije provedeno je mnogo vremena istražujući dokumentaciju pojedinih klasa Android operativnog sistema, osmišljena su pametna rješenja problema na koje se nailazilo i time je znatno prošireno znanje studenta te su dobivene vještine u samostalnom radu. S obzirom na to da je ovo prvi projekt ovog opsega na kojem je rađeno, te da je projekt temeljen na vlastitoj ideji, jasno je da aplikacija može imati nekih mana ali daljnjim testiranjem i nadogradnjom otvara mjesto za poboljšanje. Jedna od ideja za nadogradnju koja se sama nametnula je proširenje aplikacije na druge jezike, a uz daljnji razvoj moguće je i na druge predmete.

5. Literatura

[1]https://developer.android.com/about (posjećeno 06.05.2019)

[2]https://web.math.pmf.unizg.hr/~karaga/android/android_skripta.pdf (posjećeno 10.05.2019)

[3]https://developer.android.com/guide/components/activities/intro-activities.html (posjećeno 08.05.2019)

[4]https://developer.android.com/guide/topics/manifest/manifest-intro.html (posjećeno 08.05.2019)

[5]https://developer.android.com/reference/android/app/Activity.html (posjećeno 08.05.2019)

[6]https://developer.android.com/studio/releases/sdk-tools (posjećeno 12.05.2019)

[7]https://developer.android.com/reference/android/widget/Button (posjećeno 08.05.2019)

[8]https://developer.android.com/reference/android/view/View.html#generateViewId%28%29 (posjećeno 14.05.2019)

[9]http://www.dre.vanderbilt.edu/~schmidt/android/android-4.0/out/target/common/docs/doc-comment-check/guide/appendix/api-levels.html (posjećeno 07.05.2019)

[10]https://firebase.google.com/docs/reference/android/com/google/firebase/database/ValueEventListener (posjećeno 10.05.2019)

[11]https://firebase.google.com/docs/reference/android/com/google/firebase/database/DatabaseReference.html (posjećeno 10.05.2019)

[12]https://developers.google.com/android/guides/overview (posjećeno 09.05.2019)

[13]https://hackernoon.com/how-android-is-used-in-iot-85c3e560d369 (posjećeno 07.06.2019)

[14]https://arxiv.org/ftp/arxiv/papers/1207/1207.0203.pdf (posjećeno 07.06.2019)

[15] https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/ (posjećeno 01.09.2018)

[16] https://gs.statcounter.com/os-market-share/mobile/worldwide (posjećeno 01.09.2019)