Mobilna aplikacija za ponavljanje gramatike i vokabulara u stranom jeziku struke

Kovač, Jakob

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Stručni studij

MOBILNA APLIKACIJA ZA PONAVLJANJE GRAMATIKE I VOKABULARA U STRANOM JEZIKU STRUKE

Završni rad

Jakob Kovač

Osijek, 2019.

Sadržaj

1. UVOD

Brzi razvoj tehnologije doveo je do novih metoda učenja jezika. U današnje vrijeme skoro svaka osoba posjeduje neku vrstu pametnog uređaja pa nije čudno da se korištenje tehnologije u svrhu educiranja povećava iz dana u dan. Jedna od mnogobrojnih metoda za učenje novih jezika je učenje jezika potpomognuto mobilnim tehnologijama, odnosno MALL (engl. *mobile-assisted language learning*). Ova metoda temelji se na korištenju mobilnih uređaja kako bi studenti i učenici imali konstantan pristup materijalima za učenje novih jezika te kako bi se nastavnicima olakšalo podučavanje. Predviđa se da će mobilne aplikacije za učenje jezika potpuno zamijeniti udžbenike u edukaciji [1]. Jedna od glavnih prednosti ove metode je mobilnost koja omogućava pristup materijalima na bilo kojem mjestu i u bilo koje vrijeme uz uvjet da postoji pristup mobilnoj mreži te pruža sve veću fleksibilnost i učinkovitost nego računalno potpomognuto učenje jezika, odnosno CALL (engl. *computer-assisted language learning*) [2].

Cilj ovog završnog rada je izraditi mobilnu aplikaciju za Android koja će se koristiti za provjeru znanja studenata. Svrha aplikacije je da nastavnik može ispitati znanje svojih studenata kroz kvizove kako bi prepoznao područja u kojima su studenti slabiji te kako bi u skladu s rezultatima mogao prilagoditi svoja predavanja. Aplikacija će rezultate studenata i statistiku kvizova spremati u bazu podataka te će se nastavniku rezultati i statistika prikazivati na zaslonu mobilnog uređaja.

Na početku ovog rada u poglavlju „Pregled postojećih rješenja“ dan je kratak osvrt na već postojeće aplikacije koje izvršavaju sličnu ulogu te tehnologije koje su se koristile za izradu aplikacije u poglavlju „Razvojni alati“. Nakon toga u poglavlju „Funkcionalni zahtjevi za mobilnu aplikaciju“ dan je opis funkcionalnih zahtjeva koje aplikacija mora imati. Sljedeće poglavlje „Programsko rješenje mobilne aplikacije“ prikazat će kako su programska rješenja implementirana i na koji način aplikacija funkcionira. Na kraju rada u poglavlju „Prikaz korištenja i ispitivanje rada mobilne aplikacije“ nalaze se upute za korištenje aplikacije.

.

2. PREGLED POSTOJEĆIH RJEŠENJA

2.1 Kahoot!

Kahoot! je platforma za učenje koja se temelji na igrama, odnosno kvizovima s više izbora. Često se koristi u svrhu edukacije u školama i na fakultetima. Ideja za razvijanje platforme pojavila se 2006. godine. Cilj platforme je, koristeći resurse učionice, stvoriti kviz gdje će nastavnik imati ulogu voditelja kviza a učenici ulogu natjecatelja u tom kvizu. Eksperimenti provedeni u ranim fazama razvoja aplikacije pokazali su da je platforma relativno jednostavna za korištenje, a zabilježeni su i bolji rezultati u učenju te su predavanja postala zabavnija, što je povećalo motiviranost učenika i studenata za predavanja. Platforma Kahoot! omogućava kreiranje kvizova od strane korisnika. Platformi se može pristupiti preko internet preglednika, mobitela ili preko same aplikacije. Kahoot! je službeno pokrenut 2013. godine [3].

Platforma Kahoot! zamišljena je da se koristi za socijalno učenje. Učenici bi se okupili oko zajedničkog zaslona, najčešće projektora. Platforma je namjerno dizajnirana na način da sudionici moraju često dizati pogled sa svojih uređaja [4]. Svi sudionici međusobno se povežu u sobu koristeći PIN koji se prikaže na zajedničkom zaslonu te koriste mobitel ili tablet za odgovaranje na pitanja koja su najčešće generirana od strane nastavnika. Pitanja se prikazuju na zajedničkom zaslonu zajedno s ponuđenim odgovorima. Nakon svakog pitanja bodovi se dodjeljuju sudionicima te se na zajedničkom zaslonu prikazuje trenutni poredak. Kada kviz završi, nastavnik može pogledati statistiku kviza kako bi identificirao kritična područja za učenike.

Nastavnici u školama i na fakultetima koriste platformu Kahoot! tijekom učenja u svrhu procjene znanja svojih učenika a zatim svoja predavanja i nastavne aktivnosti prilagodili kako bi poboljšali znanje učenika u područjima koja su identificirali kao slabije usvojena.

2.2 Socrative

Socrative je on-line aplikacija za izradu interaktivnih kvizova. Za razliku od Kahoota! koji dozvoljava samo stvaranje kvizova s pitanjima za koje je ponuđeno više odgovora, Socrative omogućuje kreiranje više različitih tipova zadataka, poput točnih i netočnih tvrdnji, dopunjavanja riječi, i sl., a kreirani kviz moguće je spremiti u PDF datoteku i isprintati. Jedan od glavnih nedostataka aplikacije Socrative je taj što se poredak sudionika ne može vidjeti nakon svakog odgovorenog pitanja niti na kraju samog kviza već se sudionici moraju sami fizički prijaviti čim završe s kvizom.

Aplikacija koristi bazu podataka u oblaku računala koja funkcionira u stvarnom vremenu. Razvijena je 2010. godine od strane nekolicine studenata s bostonskog sveučilišta. Trenutno je

registrirano više od 350 tisuća nastavnika a zabilježeno je da je u jednom trenutku oko 6,2 milijuna učenika i studenata koristilo aplikaciju te da je postavljeno više od 120 milijuna pitanja [5].

2.3 Osvrt na postojeća rješenja

Na tržištu trenutno postoji veliki broj različitih aplikacija za edukaciju, ali prethodno spomenute aplikacije su najpopularnije. Platforma Kahoot! dizajnirana je tako da bude prilagođena korisniku te je vrlo lagana za korištenje i postavljanje. Kvizovi se mogu kreirati lagano te se čak mogu i podijeliti na društvenim mrežama. Konstantan prikaz rezultata na zaslonu i način bodovanja dodatno motivira studente da se više potrude oko rješavanja pitanja. Mogućnost dodavanja slika i video isječaka u kvizove čini rješavanje kvizova manje monotonim. Velika prednost platforme je ta što se u kvizu može sudjelovati preko mobitela ili računala koristeći internetske preglednike. Velika mana platforme je ta što korisnici ne mogu vidjeti točna rješenja pitanja nakon odgovaranja koja donekle smanjuje količinu povratnih informacija koje studenti dobiju nakon rješavanja.

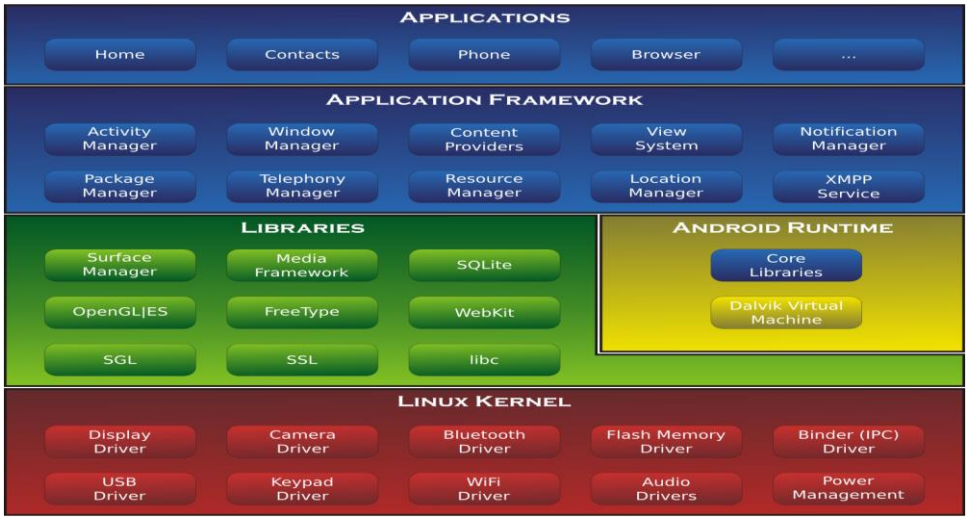
Prednost Socrativa je ta što za razliku od Kahoota! omogućava dodavanje više različitih tipova zadataka u kvizove što čini rješavanje kvizova raznolikijim. Mogućnost slobodnog kretanja po pitanjima je dosta velika prednost jer bude slučajeva kada korisnici nisu sigurni koji bi odgovor odabrali u trenutku kada se nađu na pitanju pa im to omogućava da se kasnije vrate na pitanje ako se nečega sjete. Mogućnost stvaranja posebnih kvizova koji će korisnicima prikazati točan odgovor te prikazati im objašnjenje za to pitanje je jako dobra funkcionalnost iz perspektive osobe koja rješava kviz.

3. OPERACIJSKI SUSTAV ANDROID

Android je mobilni operacijski sustav koji je razvio Google. On se temelji na modificiranoj Linux jezgri i drugim programima otvorenoga koda. Sustav je primarno dizajniran za uređaje sa zaslonom na dodir poput tableta i mobitela, ali varijacije sustava koriste se u televizorima, automobilima i pametnim satovima. Najnovija stabilna verzija android sustava je Android 9 *Pie* sustav koji je objavljen u kolovozu 2018. godine.

Android sustav prvo je razvijala tvrtka Android Inc. koju je 2005. godine kupio Google. Operacijski sustav otkriven je javnosti 2007. godine, a prvi uređaj koji je koristio Android stavljen je na tržište u rujnu 2008. godine. Operacijski sustav primarno se temelji na C/C++ programskim jezicima, dok se većina aplikacija piše u Java programskom jeziku korištenjem SDK (engl. *software development kit*).

Najniži sloj sustava je upravo Linuxova jezgra koja sadrži sve potrebne upravljačke programe (engl. *drivers*). U sloju iznad jezgre nalaze se biblioteke koje su napisane u C/C++ jeziku te izvršno okruženje (engl. *runtime enviroment*) u kojemu se aplikacije izvršavaju. Izvršno okruženje sastoji se od dvije glave komponente. Prva komponenta sadrži jezgrene biblioteke Java programskog jezika. Druga je virtualni stroj Dalvik koji pokreće aplikacije. Iznad biblioteka i izvršnog okruženja nalazi se aplikacijski okvir koji se sastoji od niza mehanizama koji olakšavaju pisanje programa za Android sustav. On omogućuje korištenje API-ja (engl. *application programming interface*). API je sučelje preko kojega aplikacije od jezgre sustava mogu zatražiti određene resurse i procese. Na samom vrhu arhitekture nalaze se aplikacije koje su vidljive krajnjem korisniku [6].



Slika 3.1. Arhitektura operacijskog sustava Android [6]

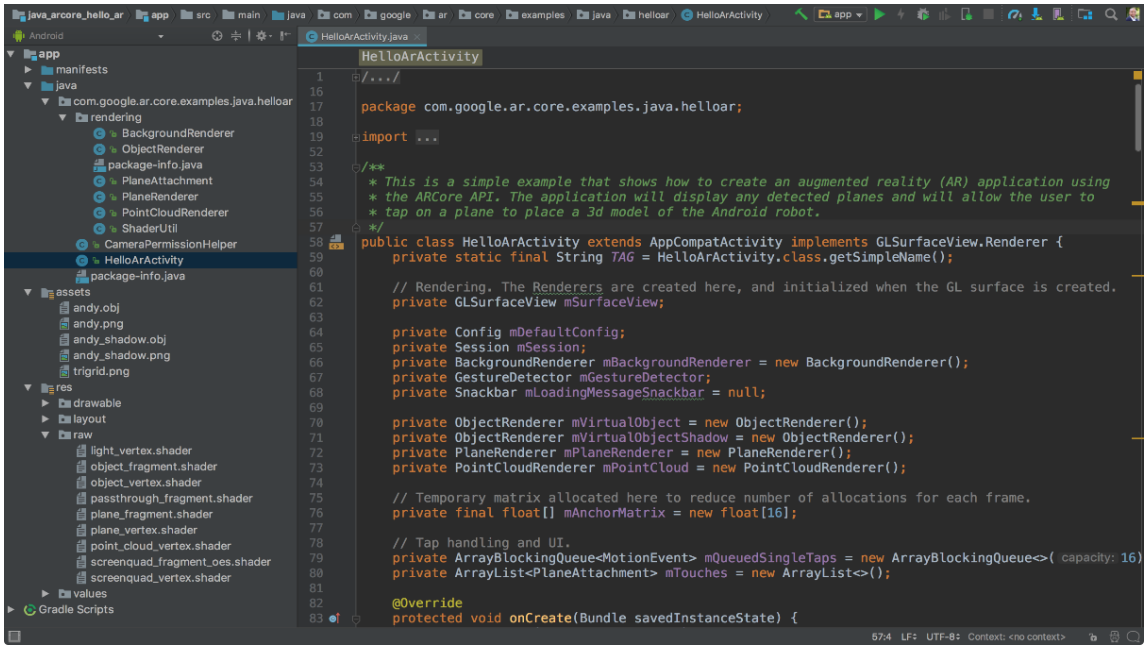
4. RAZVOJNI ALATI

Sljedeća potpoglavlja daju kratak prikaz nekih razvojnih alata koji su se koristili za izradu mobilne aplikacije, točnije razvojna okolina Android studio, Java programski jezik koji će se koristiti za programiranje aplikacije te Firebase baza podataka.

4.1 Android studio

Android studio službena je razvojna okolina za aplikacije namijenjene operacijskom sustavu Android. Najavljen je u svibnju 2013. godine, a prva stabilna verzija izašla je u prosincu 2014. godine. Razvojna okolina Android studio zamijenio je do tada službenu Eclipse razvojnu okolinu. On sadrži ugrađeni virtualni emulator pomoću kojeg se aplikacije mogu testirati na virtualnom uređaju, podržava *drag and drop* način rada kod kreiranja korisničkih sučelja, sadrži ugrađenu podršku za rad s Google *cloud* platformom, potpora je za aplikacije namijenjene Android Wear operacijskom sustavu, predloške za česte Android dizajne i komponente te Lint alat koji se koristi za analiziranje izvornog koda te označavanje programskih grešaka.

Razvojna okolina Android studio temelji se na integriranom razvojnom okruženju Java programskog jezika pod nazivom IntelliJ IDEA koju je razvila tvrtka JetBrains. Neka od svojstava tog programskog okruženja su sljedeća: pomoć pri programiranju, ugrađeni alati koji podržavaju distribuirane sustave za verzioniranje poput GITa, Mercuirala, te rad s bazama poput Oraclea, Microsoftovog SQL Servera, MySql, itd. te ekosustav za dodatke (engl. *plugins*) koji omogućava dodatne specifične funkcije [7].



Slika 4.1. Izgled sučelja Android studija

4.2 Programski jezik Java

Java je objektno-orijentirani programski jezik koji se temelji na klasama, iako nije čisti objektno-orijentirani jezik jer sadrži neke primitivne tipove varijabli te mu je sintaksa jezika vrlo slična jezicima C/C++. Jezik je dizajniran tako da ima što manji broj implementacijskih ovisnosti, a njegova je svrha programerima omogućiti da jednom napišu kod i da se on onda može izvršavati na svim platformama koje podržavaju Javu bez ponovne kompilacije koda. Sve aplikacije pisane Java jezikom izvršavaju se preko Java virtualnog uređaja, što omogućava neovisnost o specifikacijama računala.

Java programski jezik osmislio je James Gosling, koji je tada radio za tvrtku Sun Microsystems koju je 2010. godine kupila tvrtka Oracle. Prva verzija jezika puštena je u javnost 1995. godine kao ključna komponenta tvrtkine Java platforme. Od 2018. godine Java je najpopularniji i najčešće korišteni programski jezik. Trenutna verzija Java jezika je verzija 12 koja je puštena u javnost u ožujku 2019. godine [8].

4.3 Firebase

Firebase je razvojna platforma namijenjena mobilnim i web aplikacijama. Platformu je 2011. godine osnovala tvrtka Firebase Inc. koju je 2014. godine kupila tvrtka Google. Od listopada 2018. godine registrirano je 18 različitih usluga koje pruža Firebase platforma te je zabilježeno da 1.5 milijuna aplikacija koristi Firebase na neki način [9], [10].

Od 18 različitih usluga koje pruža Firebase platforma, za izradu aplikacije koristit će se Firebase baza podataka koja funkcionira u stvarnom vremenu. Prednosti korištenja baze podataka koja radi u stvarnom vremenu su mogućnost sinkronizacije između korisnika te spremanje podataka na *cloud*. Podaci koji se spreme na *cloud* u istom trenutku postaju dostupni svim korisnicima. Svi podaci koji se spreme na Firebase bazu podataka osigurani su provođenjem niza sigurnosnih pravila sa serverske strane.

5. FUNKCIONALNI ZAHTJEVI ZA MOBILNU APLIKACIJU

Ideja je stvoriti funkcionalnu aplikaciju kombinirajući prednosti već postojećih aplikacija poput Kahoota! i Socrativa. Aplikacija će se koristiti u svrhu provjeravanja znanja studenata te će se zbog svoje velike rasprostranjenosti pokretati na operacijskim sustavima Android.

Aplikacija će sadržavati sljedeće funkcionalnosti:

- registracija i prijava korisnika
- rješavanje kviza i provjera rješenja
- administratorska prava
- dodavanje i brisanje pitanja
- određivanje parametara i stvaranje kvizova
- otvaranje i zatvaranje kviza
- promjena uloga korisnika od strane administratora
- ispis postotka riješenosti kviza korisnika.

5.1 Registracija i prijava korisnika

Prilikom pokretanja aplikacije, korisniku se na zaslonu prikazuje forma za prijavljivanje. Ako korisnik nema račun za mobilnu aplikaciju, na početnom zaslonu treba odabrati opciju za registraciju. Kada korisnik odabere opciju za registraciju, otvara se nova aktivnost. Aktivnost za registraciju sadržava četiri polja koja se moraju ispuniti: e-mail, korisničko ime, lozinka te ponovljena lozinka. Registracija je potrebna kako bi se kasnije rezultati kvizova koje korisnik riješi mogli spremiti na *cloud* bazu podataka koristeći njegovo korisničko ime. Kada su sva polja popunjena, pritiskom na gumb izvršava se proces registracije te se svi podaci spremaju uz pomoć usluge Firebase Auth. S obzirom da će se korisnikovo ime koristiti za prikazivanje rezultata kviza, radi lakšeg identificiranja preporuča se da se za korisničko ime koristi ime i prezime korisnika. Prije same registracije, provjeravaju se sva polja kako bi se osiguralo da je sve pravilno uneseno. Korisniku se prikazuje poruka ako neka od polja nisu pravilno ispunjena.

Ako je korisnik već registriran, prije samog korištenja aplikacije korisnik će se morati prijaviti koristeći svoju e-mail adresu i lozinku. Kada korisnik unese e-mail adresu i lozinku, aplikacija provjerava je li korisnik registriran. Ako je korisnik registriran i svi podaci su pravilno uneseni, omogućava mu se pristup aplikaciji. Za registraciju i prijavu potrebno je imati pristup internetu.

5.2 Rješavanje kviza i provjera rješenja

Nakon procesa registracije i prijave, korisnika se dovodi do glavnog izbornika. Kada korisnik odabere gumb *Start*, pokreće se nova aktivnost. Ova aktivnost dohvaća sve kvizove iz baze podataka te ih korisniku prikazuje na zaslon. Korisnik može odabrati bilo koji od ponuđenih kvizova klikom na njih. Kako bi korisnik mogao početi rješavati kviz, on prvo mora biti otvoren. Kada korisnik odabere neki od ponuđenih kvizova, aplikacija iz baze podataka dohvaća njezin status. Ako je kviz otvoren, korisniku se omogućava pristup kvizu te ga se dovodi u novu aktivnost gdje započinje s rješavanjem kviza. U protivnom, korisniku se prikazuje poruka da je kviz zatvoren te da samo administrator može dopustiti pristup kvizu.

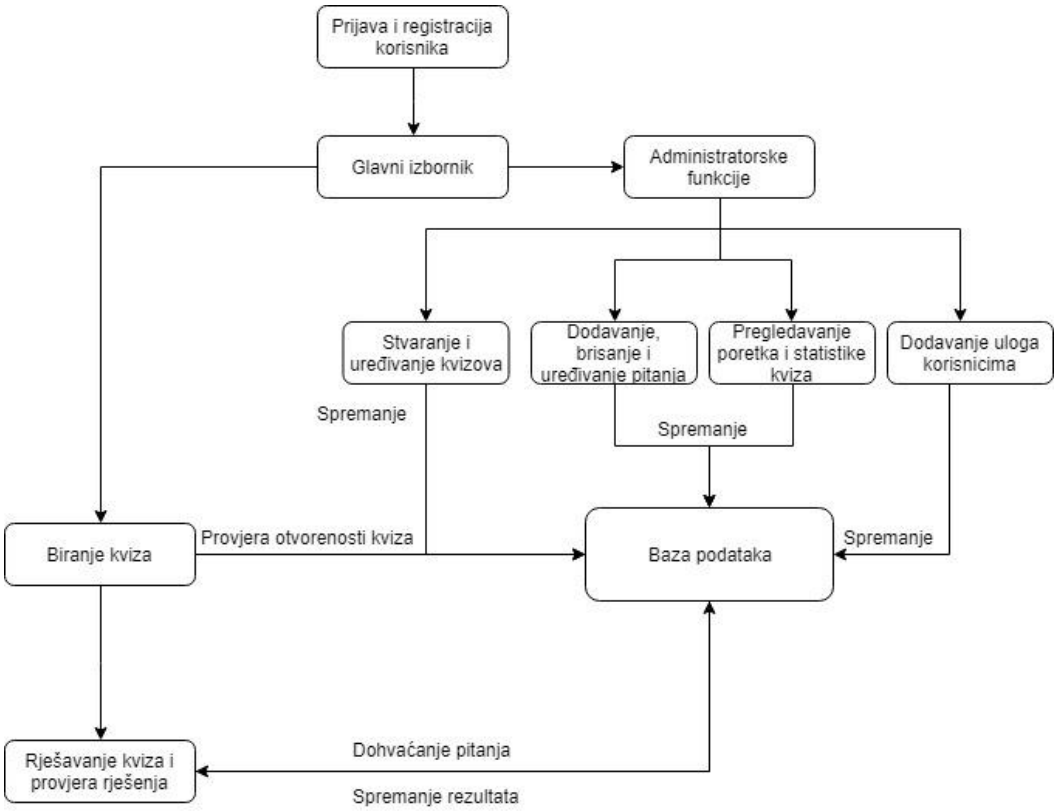
Tijekom rješavanja kviza, korisnik će se moći slobodno kretati od pitanja do pitanja, a svaki njegov odabir bit će zapamćen. Svaki će kviz koristiti vremenski brojač koji određuje koliko vremena korisnik ima za rješavanje prije nego što se kviz prekine i korisnikovi odabiri obrade. Administrator određuje koliko će dugo kviz trajati prilikom kreiranja kviza. Korisnik može ranije završiti rješavanje kviza odabirom gumba ili može sačekati da vremenski brojač istekne. Nakon završetka kviza, korisnikovi odabiri obrađuju se te se na zaslonu ispisuje njegov rezultat. Njegov rezultat također se sprema na bazu podataka zajedno sa statistikom kviza. Statistika kviza prati postotak korisnika koji su točno odgovorili na pojedino pitanje. Korisnik u ovom trenutku ima dvije opcije: pregledati odgovore ili izaći iz kviza. Ako odluči izaći iz kviza, korisnika se vraća na glavni izbornik. Ako odluči pregledati odgovore, korisnika se vraća na pitanja s tim da je odabir odgovora onemogućen te nema više vremenskog brojača.

5.3 Administratorska prava

Jednom će korisniku, u ovom slučaju nastavniku, biti dodijeljena posebna uloga pomoću koje će korisniku biti omogućen pristup administratorskim funkcionalnostima aplikacije. Nakon prijavljivanja u aplikaciju dolazi se do glavnog izbornika. Ako korisnik ima pristup administratorskim funkcijama, njemu će, za razliku od običnih korisnika, biti vidljiv gumb za pristup administratorskim funkcijama. Klikom na taj gumb pokreće se nova aktivnost koja dovodi do glavnog izbornika administratorskih funkcija. Korisnik će tamo imati pristup različitim aktivnostima pomoću kojih će moći upravljati kvizovima, rezultatima te samim korisnicima.

Klikom na gumb *Add Questions* pokreće se nova aktivnost koja omogućava dodavanje pitanja. Potrebno je ispuniti pet različitih polja za dodavanje pitanja: pitanje, točan odgovor te tri opcije koje će se ponuditi zajedno s točnim odgovorom. Na kraju je potrebno iz padajućeg izbornika odabrati kviz u koji se pitanje želi spremiti.

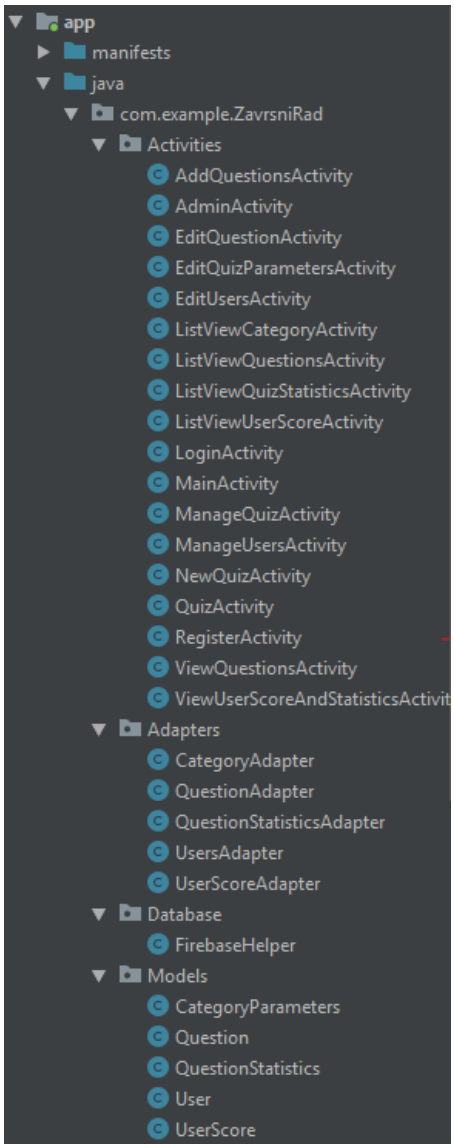
Klikom na gumb *Manage Quiz* pokreće se aktivnost koja omogućava otvaranje i zatvaranje kvizova te uređivanje vremenskog trajanja postojećeg kviza. Klikom na gumb *Create Quiz* pokreće se aktivnost koja omogućava stvaranje novih kvizova. Za stvaranje novog kviza potrebno je odrediti naziv kviza i vremensko trajanje kviza. Klikom na gumb *View Questions* pokreće se aktivnost koja omogućava pregled, uređivanje i brisanje pitanja koja su već dodana za odabrani kviz. Klikom na gumb *Scores and Statistics* pokreće se aktivnost koja omogućava pregled poretka korisnika te statistike kviza za odabrani datum. Klikom na gumb *Manage Users* pokreće se aktivnost koja omogućava pregled registriranih korisnika te dodjelu novih uloga.



Slika 5.1. Dijagram toka mobilne aplikacije

6. PROGRAMSKO RJEŠENJE MOBILNE APLIKACIJE

Sama aplikacija sastoji se od ukupno dvadeset devet različitih Java klasa te dvadeset tri različitih XML datoteka koje definiraju izgled sučelja. Klase su podijeljene u četiri različita paketa. *Activities* paket sastoji se od osamnaest klasa koje sadrže funkcionalnosti za aktivnosti aplikacije. *Adapters* paket sadrži pet različitih klasa koje se koriste kao posebni adapteri za prikazivanje podataka na zaslonu. *Database* paket sadrži samo jednu klasu koja izvršava sve potrebne operacije nad Firebase bazom podataka. *Models* paket sadrži pet klasa koje predstavljaju modele podataka koji se koriste za spremanje i dohvaćanje podataka s baze podataka Firebase. Pojašnjenje i pregled funkcionalnosti svih klasa i izgledi sučelja bit će prikazani u odgovarajućim potpoglavljima.



Slika 6.1. Struktura projekta i prikaz svih Java klasa

6.1 Struktura baze podataka Firebase

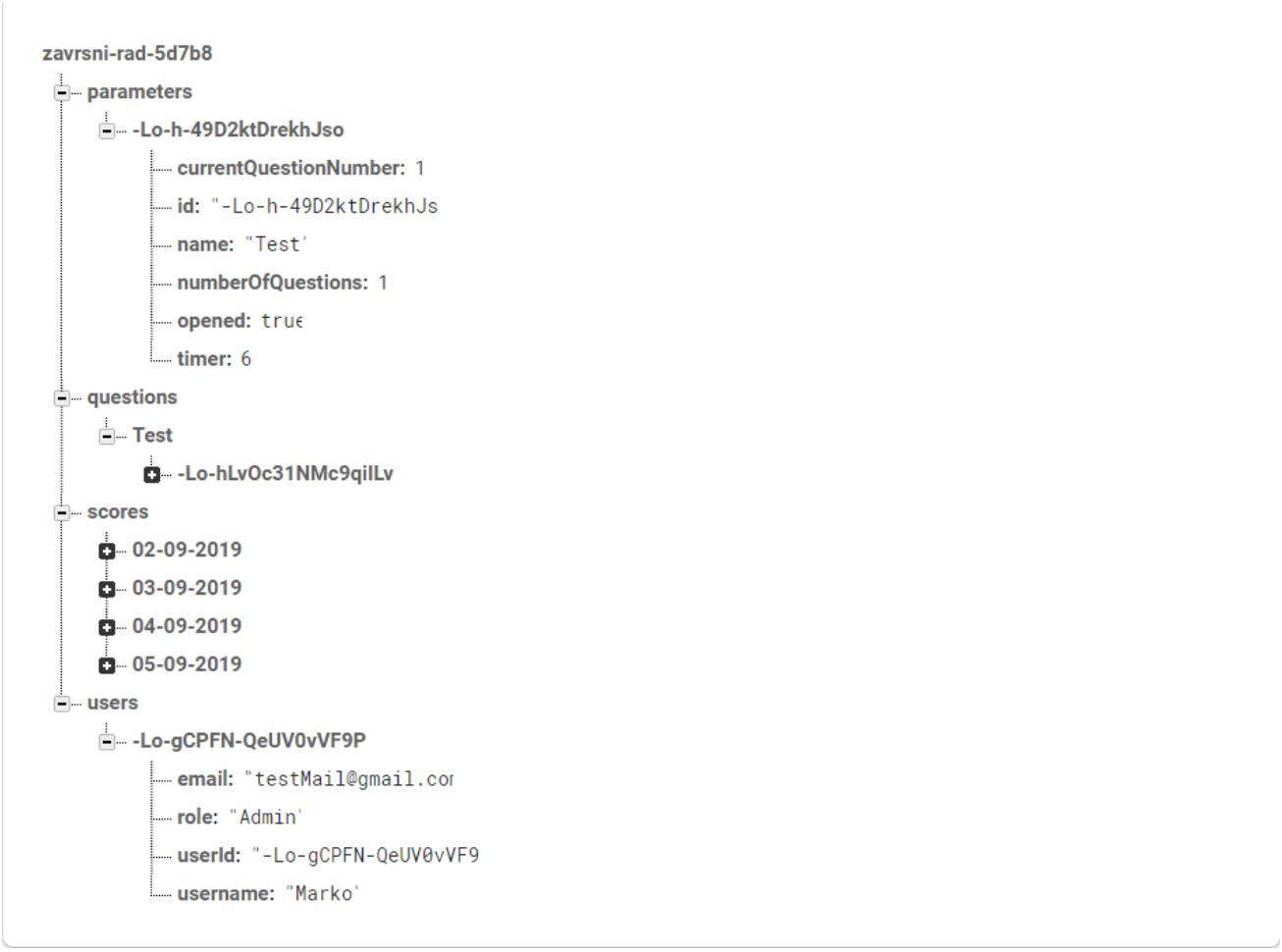
Baza podataka Firebase strukturirana je prema slici 6.2.

Parameters čvor sadržava sve stvorene kvizove te bitne informacije vezane uz te kvizove (kao što je ukupan broj pitanja koja su dodana u bazu, naziv kviza te vrijeme koje korisnik ima za rješavanje kviza kada ga pokrene).

Questions čvor sadrži sva pitanja koja su vezana uz kvizove. Ovaj čvor podijeljen je na podčvorove koji sadrže nazive stvorenih kvizova a u njima se spremaju pitanja za pojedini kviz.

Scores čvor sadrži sve poretke korisnika kada riješe kviz a podijeljen je na podčvorove koje čine datumi kada su kvizovi rješavani.

Users čvor sadrži sve bitne podatke o registriranim korisnicima poput njihovih e-mail adresa, korisničkih imena te njihovih uloga.



Slika 6.2. Prikaz strukture baze podataka Firebase

6.2 Registracija i prijava korisnika

Ova aplikacija koristi već ugrađeni Firebaseov sustav za registraciju i autentifikaciju korisnika. S obzirom da će svakom korisniku biti dodijeljena uloga koja će određivati funkcionalnosti koje on ima a Firebasov sustav za autentifikaciju ne dopušta spremanje nikakvih drugih podataka osim e-mail adrese i lozinke, prilikom registracije podaci o korisniku spremat će se u poseban čvor u bazi podataka u stvarnom vremenu. Postupak registracije korisnika izvršava se u aktivnosti *RegisterActivity*. XML datoteka definira izgled sučelja koje se sastoji od četiri *EditText* elemenata te jednog gumba. Gumb ima definirano *onClick* svojstvo koje poziva *register()* metodu kada korisnik klikne na gumb. Prilikom pokretanja aktivnosti, automatski se instanciraju novi objekti klasa *FirebaseHelper* i *FirebaseAuth*. Nakon što su sva polja ispunjena i korisnik klikne na gumb za registraciju poziva se *register()* funkcija koja prvo provjerava ima li uređaj pristup internetskoj mreži. Ako nema pristupa mreži, proces registracije se prekida te se korisniku prikazuje poruka koristeći *Toast* da je za registraciju potreban pristup internetu. Nakon provjere ima li uređaj pristup internetskoj mreži, provjerava se jesu li sva polja pravilno ispunjena. Firebaseov autentifikacijski server zahtijeva lozinku od minimalno 6 znakova tako da se i ovaj uvjet provjerava prije nego što se zahtjev pošalje serveru. Korisnik se zatim registrira koristeći funkciju *createUserWithEmailAndPassword()* koja je već ugrađena u klasu *FirebaseAuth* te kao ulazne parametre prima unesenu e-mail adresu i lozinku. Nakon uspješne pohrane podataka na autentifikacijski server, poziva se metoda *registerUser()* klase *FirebaseHelper* koja sprema korisničko ime i e-mail adresu u bazu podataka u stvarnom vremenu te se korisnik pozivanjem funkcije *startLoginActivity()* prebacuje na aktivnost za prijavljivanje.

```
public void register(final View v) {
    final String username = this.username.getText().toString();
    String password = this.password.getText().toString();
    String repeatPassword = this.repeatPassword.getText().toString();
    final String email = this.email.getText().toString().toLowerCase();

    if (!(checkFields(password, repeatPassword, username))) {
        toastMessage("Fill out all the required fields.");
    } else if(!(validateEmail(email))) {
        toastMessage("E-mail is either empty or badly formatted.");
    } else if (!(matchPasswords(password, repeatPassword))) {
        toastMessage("The passwords do not match.");
    } else if (password.length() < 6) {
        toastMessage("Password has to be at least 6 characters long.");
    } else if (!(connectivityManager.getNetworkInfo(ConnectivityManager.TYPE_MOBILE).getState() == NetworkInfo.State.CONNECTED
            || connectivityManager.getNetworkInfo(ConnectivityManager.TYPE_WIFI).getState() == NetworkInfo.State.CONNECTED)) {
        toastMessage("Activate your internet connection in order to register.");
    } else {
        firebaseAuth.createUserWithEmailAndPassword(email, password).addOnCompleteListener((task) -> {
                if (task.isSuccessful()) {
                    toastMessage("Registration successful");
                    firebase.registerUser(username, email);
                    startLoginActivity(v);
                } else {
                    String error = task.getException().getMessage();
                    toastMessage(error);
                }
        });
    }
}
```

Slika 6.3. Programski kod metode register()

Ukoliko korisnik već ima registriran račun za aplikaciju, potrebno je samo da se korisnik prijavi koristeći e-mail adresu i lozinku. Proces prijave u aplikaciju odvija se u aktivnosti *LoginActivity*. Sučelje koje korisnik vidi sastoji se od dva *EditText* elemenata te jednog gumba s definiranim *onClick* svojstvom koje poziva funkciju *login()*. Pri pokretanju aktivnosti poziva se funkcija *setupUI()* unutar *onCreate()* metode koja instancira sve objekte klasa *FirebaseHelper*, *FirebaseAuth* i *PreferencesManager* te pronalazi sve elemente koji se nalaze na sučelju koristeći funkciju *findViewById()*. Nakon što korisnik upiše e-mail adresu i lozinku s kojom se želi prijaviti, klikom na gumb poziva se već spomenuta funkcija koja dohvaća upisane vrijednosti iz elemenata. Nakon dohvaćenih vrijednosti, ponovno se vrši provjera ima li uređaj pristup internetskoj mreži te se provjerava je li e-mail adresa dobro formatirana. Ako uređaj nema pristup internetu ili e-mail adresa nije dobro formatirana, proces prijave se prekida te se korisnika obavještava putem *Toast* poruke. Ako su svi podaci pravilno uneseni, korisnikovi podaci se provjeravaju pomoću ugrađene funkcije *signInWithEmailAndPassword()* klase *FirebaseAuth*. Kada se korisnik uspješno prijavi, dohvaćaju se njegovi podaci iz baze podataka u stvarnom vremenu koristeći objekt klase *FirebaseHelper*. Nakon što se podaci dohvate, korisnikovi podaci poput korisničkog imena i e-mail adrese spremaju se pomoću *SharedPreferences* te se korisnika prebacuje u glavni izbornik.

Glavni izbornik nalazi se u aktivnosti *MainActivity*. Početkom aktivnosti dohvaćaju se podaci o prijavljenom korisniku koji su postavljeni u aktivnosti *LoginActivity* pomoću *SharedPreferences* te se korisnika pozdravlja *Toast* porukom. Koristeći dohvaćeni e-mail, ponovno se dohvaćaju podaci iz baze podataka u realnom vremenu. Podaci se ponovno dohvaćaju u slučaju da je korisnikova uloga promijenjena dok je on bio prijavljen u aplikaciju pa povratkom u glavni izbornik dobiva mogućnost pristupa administratorskim funkcijama bez potrebe ponovnog prijavljivanja. Pozivanjem funkcije *checkUser()* provjerava se korisnikova uloga dohvaćena iz baze podataka. Za dohvaćanje korisnikove uloge koristi se *getRole()* metoda koja se nalazi unutar klase *User*. Prema zadanome, gumb pomoću kojega se pristupa administratorskim funkcijama je sakriven te postaje vidljiv jedino u slučaju ako korisnikova uloga iz baze podataka odgovara ulozi *Admin*. Boolean varijabla *isMessageShown* koristi se za provjeravanje je li korisnika pozdravila poruka kada se dođe do glavnog izbornika prvi puta te prema zadanome ima neistinitu vrijednost. Kada korisnika pozdravi poruka, ova varijabla mijenja svoju vrijednost u istinito.

```
public void setupUI(){
    pref = this.getSharedPreferences( name: "prefs", MODE_PRIVATE);
    editor = pref.edit();
    email = pref.getString( s: "email", s1: null);
    isMessageShown = pref.getBoolean( s: "isMessageShown", b: true);
    admin = findViewById(R.id.btnAdmin);
    admin.setVisibility(View.INVISIBLE);
    firebase = new FirebaseHelper( reference: "users");
    user = firebase.getUserDetails(email, new FirebaseHelper.DataStatus() {
        @Override
        public void questionIsLoaded(List<Question> list) {

        }

        @Override
        public void categoryListIsLoaded(List<CategoryParameters> list) {

        }

        @Override
        public void userIsLoaded(User user) { checkUser(); }

        @Override
        public void categoryIsLoaded(CategoryParameters category) {

        }
    });
}

public void checkUser(){
    if(user.getRole().equals("Admin")){
        admin.setVisibility(View.VISIBLE);
    }
    if(!(isMessageShown)){
        toastMessage("You are now signed in. Welcome " + user.getUsername() + ".");
        editor.putBoolean( s: "isMessageShown", b: true);
        editor.commit();
    }
}
```

Slika 6.4. Programski kod aktivnosti MainActivity koja dohvaća podatke o korisniku i provjerava njegovu ulogu

6.3 Odabir i rješavanje kviza

Odabir kviza za rješavanje se odvija u aktivnosti *ListViewCategoryActivity*. Pri pokretanju aktivnosti dohvaćaju se svi postojeći kvizovi koji se nalaze u bazi podataka i spremaju se u listu objekata *categoryList* koristeći klasu *FirebaseHelper*. Metoda koja dohvaća listu objekata iz baze kao ulazni parametar prima sučelje. Sučelje se koristi kako bi se zaobišla asinkrona priroda baze podataka Firebase te osigurava da se s podacima ništa ne radi dok oni nisu dohvaćeni iz baze i učitani. Kada se svi podaci dohvate i učitaju, koristeći posebno programirani adapter *CategoryAdapter* podaci se prikazuju na zaslonu, točnije *ListView* elementu zaslona. Na adapter se postavlja osluškivač koji se aktivira svaki puta kada korisnik klikne na jedan od ponuđenih kvizova na zaslonu te on poziva funkciju *checkStatus()*. Ovoj se funkciji prosljeđuje cjelobrojni parametar *position* koji odgovara indeksu na kojemu se odabrani objekt nalazi u listi *categoryList*. Zatim se iz kliknutog objekta dohvaća njegov jedinstveni primarni ključ pozivajući metodu *getId()*. Ovaj primarni ključ koristi se kako bi se objekt lakše pronašao u bazi podataka te se njegova vrijednost ponovno dohvaća. Ako je kviz otvoren, korisnika se prebacuje u novu aktivnost

te se odabrani objekt kviza prosljeđuje slijedećoj aktivnosti pomoću *Intenta*. U protivnom, korisniku se prikazuje poruka koja ga obavještava da je kviz zatvoren te da mu je pristup zabranjen.

```
public void populateListView() {
    adapter = new CategoryAdapter( context: this, R.layout.list_item_category, (ArrayList<CategoryParameters>) categoryList);
    listView.setAdapter(adapter);
    listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> adapterView, View view, int position, long l) {
            checkStatus(position);
        }
    });
}

public void checkStatus(int position){
    String id = categoryList.get(position).getId();
    firebase.checkStatus(id, new FirebaseHelper.DataStatus() {
        @Override
        public void questionIsLoaded(List<Question> list) {

        }

        @Override
        public void categoryListIsLoaded(List<CategoryParameters> list) {

        }

        @Override
        public void userIsLoaded(User user) {

        }

        @Override
        public void categoryIsLoaded(CategoryParameters categoryParameters) {
            if(categoryParameters.isOpened()){
                startQuiz(categoryParameters);
            } else {
                toastMessage("Quiz is currently closed. Quiz first must be opened by an admin.");
            }
        }
    });
}
```

Slika 6.5. Programski kod koji dohvaća objekt kviza iz baze podataka i provjerava status kviza

Kada korisnik dobije pristup kvizu, prebacuje ga se u aktivnost *QuizActivity*. Pri pokretanju aktivnosti dohvaća se objekt klase *CategoryParameter* koji je proslijeđen koristeći *Intent*. Ovaj dohvaćeni objekt sadrži bitne informacije poput naziva kviza te vremenskog trajanja kviza. Iz *SharedPreferencesa* dohvaća se korisničko ime koje će se koristiti za spremanje njegovih rezultata. Pomoću klase *FirebaseHelper* dohvaćaju se sva pitanja vezana za kviz. Naziv kviza dohvaćen je iz proslijeđenog objekta koji se koristi za pronalaženje odgovarajućih pitanja u bazi. Nakon što su sva pitanja dohvaćena iz baze podataka postavlja se vrijednost cjelobrojne varijable *questionsLength* koja predstavlja količinu pitanja u listi pitanja. Uz pomoć ove varijable onemogućava se korisniku da slučajno pristupi indeksu liste koji ne postoji te se koristi za vizualni prikaz količine pitanja. Zatim se poziva metoda *shuffleQuestionAnswers()* koja pomiješa ponuđene odgovore za svako pitanje čime se osigurava da se točan odgovor ne nalazi na istom mjestu u sučelju. Samo sučelje sastoji se od elementa *RadioGroup* koji u sebi sadržava četiri različita elementa *RadioButton*, tri elementa *TextView* na kojemu se prikazuje pitanje, preostalo vrijeme za

rješavanje i trenutno pitanje u listi te dva gumba koja se koriste za kretanje naprijed i nazad po pitanjima. Na dnu sučelja nalazi se jedan gumb pomoću kojega se kviz može završiti ranije.

```
public void shuffleQuestionAnswers(List<Question> list){
    ArrayList<String> answers = new ArrayList<>();
    for(Question currentQuestion : list){
        answers.add(currentQuestion.getOption1());
        answers.add(currentQuestion.getOption2());
        answers.add(currentQuestion.getOption3());
        answers.add(currentQuestion.getOption4());

        Collections.shuffle(answers);

        currentQuestion.setOption1(answers.get(0));
        currentQuestion.setOption2(answers.get(1));
        currentQuestion.setOption3(answers.get(2));
        currentQuestion.setOption4(answers.get(3));
    }

    questionList = list;
    questionsLength = questionList.size();
}
```

Slika 6.6. Programski kod funkcija shuffleQuestionAnswers()

Prema slici 6.7. vidljivo je da klasa *Question* sadrži deset različitih varijabli koje se spremaju u njene objekte. String *id* sadržava vrijednost primarnog ključa u bazi podataka Firebase. Sljedećih šest varijabli odnosi se na spremanje podataka vezano za samo pitanje te se njihove vrijednosti postavljaju pri dodavanju pitanja u kvizove. Prema zadanome, varijabla *option1* uvijek ima istu vrijednost kao varijabla *correctAnswer*. Sljedeće tri varijable, koje su odvojene, koriste se za pamćenje odabira korisnika tijekom rješavanje kviza. String varijabla *selectedOption* sadržava u sebi vrijednost odabranog odgovora. Cjelobrojna varijabla *radioBtnId* koristi se za pamćenje koji od elemenata *RadioButtona* sadrži tu vrijednost. Boolean varijabla *selected* koristi se kako bi se provjerilo je li korisnik odabrao neki odgovor. Ove zadnje tri varijable ne spremaju se u bazu podataka koristeći *@Ignore* anotaciju na njihove *get()* metode.

```
public class Question implements Serializable {

    private String id;
    private String question;
    private String option1;
    private String option2;
    private String option3;
    private String option4;
    private String correctAnswer;

    private String selectedOption;
    private int radioBtnId;
    private boolean selected = false;
```

Slika 6.7. Prikaz varijabli koje se spremaju u klasu Question

Cjelobrojna varijabla *counter* koristi se za praćenje trenutnog pitanja te se pri pokretanju aktivnosti njezina vrijednost postavlja na 0. Ova varijabla mijenja svoju vrijednost ovisno o tome kako se korisnik kreće po pitanjima. Svaki put kada korisnik odabere jedan od odgovora, njegov odabir sprema se korištenjem *setSelected()* metode koja u objekt trenutnog pitanja sprema String vrijednost *RadioButtona*, njegov ID te postavlja vrijednost boolean varijable *selected* na istinito (Slika 6.8.). String vrijednost odgovora koristi se kasnije pri računanju točnih i netočnih odgovora, a ID gumba potreban je kako bismo znali na kojem se gumbu nalazi odabrani odgovor te kako bismo ga mogli ponovno označiti kao odabranog ako se korisnik ponovno vrati na pitanje. Varijabla *selected* koristi se za provjeru te osigurava da ne bi došlo do pokušaja označavanja odabranog odgovora ako on ne postoji.

```
public void setSelected(View v){
    int radioBtnId = radioGroup.getCheckedRadioButtonId();
    RadioButton radioBtn = findViewById(radioBtnId);
    String selectedOption = radioBtn.getText().toString();
    questionList.get(counter).setRadioBtnId(radioBtnId);
    questionList.get(counter).setSelectedOption(selectedOption);
    questionList.get(counter).setSelected(true);
}
```

Slika 6.8. Programski kod funkcije setSelected()

Nakon što vrijeme istekne ili korisnik sam odluči završiti kviz, poziva se funkcija *calculateScoreAndStatistics()*. Ova funkcija obrađuje korisnikove odabire te ili stvara ili ažurira statistiku kviza ovisno o tome postoji li već statistika u bazi podataka za trenutni datum. Na početku funkcije, uz pomoć klase *FirebaseHelper*, dohvaća se lista objekata *QuestionStatistics*. Ova klasa u sebi sadržava tri parametra koja se koriste za praćenje statistike. String varijabla *question* u sebi sadrži pitanje za koje je vezana statistika. Cjelobrojne vrijednosti *totalNumberOfUsers* i *usersThatAnsweredCorrect* koriste se za praćenje i računanje postotka riješenosti pojedinog pitanja. Ako ne postoji statistika kviza za trenutni datum, lista će biti prazna.

Nakon što je lista dohvaćena, prolazi se kroz cijelu listu objekata klase *Question* uz korištenje foreach petlje. Petlja se ovdje dijeli na dva slučaja. Ako je pitanje točno odgovoreno, potrebno je inkrementirati obje cjelobrojne varijable za statistiku kviza. Ukoliko je lista prazna, prije inkrementiranja, potrebno je stvoriti nove objekte klase *QuestionStatistics* i dodati ih u listu koja će se kasnije spremiti u bazu podataka.

Ako je pitanje krivo odgovoreno, potrebno je samo inkrementirati vrijednost cjelobrojne varijable *totalNumberOfUsers*. Ako je dohvaćena lista za statistiku kviza prazna, potrebno je ponovno stvoriti nove objekte klase *QuestionStatistics* prije inkrementiranja.

Nakon što su svi odgovori obrađeni, korisniku se na zaslon ispisuje postotak riješenosti te broj pitanja koja su točno odgovorena. U pozadini se korisnikov rezultat i statistika kviza dodaje u bazu podataka kao što je prikazano na slici 6.9.

```
public void saveQuizStatistics(List<QuestionStatistics> list) {
    firebase = new FirebaseHelper( reference: "quiz statistics/" + formattedDate);
    firebase.saveQuizStatistics(list);
}

public void saveScore(int score){
    UserScore userScore = new UserScore();

    userScore.setScore(score);
    userScore.setUsername(username);

    firebase = new FirebaseHelper( reference: "scores/" + formattedDate);
    firebase.saveUserScore(userScore);
}
```

Slika 6.9. Programski kod funkcije saveQuizStatistics() i saveScore()

Korisnik će zatim moći odabrati da se vrati na pitanja i pogledati točna rješenja ili se vratiti na glavni izbornik. Ako korisnik odluči pogledati točna pitanja, poziva se metoda *showAnswers()* koja će onemogućiti ponovno biranje odgovora. Ukoliko je korisnik krivo odgovorio na pitanje, njegov će odabir biti prikazan crvenom bojom, dok će točan odgovor biti prikazan zelenom bojom. Ako je korisnik točno odgovorio na pitanje, njegov će odabir biti prikazan zelenom bojom. Slika 6.10. prikazuje programsko rješenje za prikazivanje odgovora.

```
public void showAnswers(View v){
    if(callOnce) {
        callOnce = false;
        toggleUI();
        disableRadioGroup();
    }
    updateUI();
    option1.setTextColor(Color.BLACK);
    option2.setTextColor(Color.BLACK);
    option3.setTextColor(Color.BLACK);
    option4.setTextColor(Color.BLACK);

    if(questionList.get(counter).getCorrectAnswer().equals(questionList.get(counter).getSelectedOption())) {
        selectedOption = findViewById(questionList.get(counter).getRadioBtnId());
        selectedOption.setTextColor(Color.GREEN);
    } else {
        if(questionList.get(counter).getRadioBtnId() != 0) {
            selectedOption = findViewById(questionList.get(counter).getRadioBtnId());
            selectedOption.setTextColor(Color.RED);
        }
        if(option1.getText().toString().equals(questionList.get(counter).getCorrectAnswer())){
            option1.setTextColor(Color.GREEN);
        } else if (option2.getText().toString().equals(questionList.get(counter).getCorrectAnswer())){
            option2.setTextColor(Color.GREEN);
        } else if (option3.getText().toString().equals(questionList.get(counter).getCorrectAnswer())){
            option3.setTextColor(Color.GREEN);
        } else {
            option4.setTextColor(Color.GREEN);
        }
    }

}
```

Slika 6.10. Programski kod funkcije showAnswers()

6.4 Administratorski dio

6.4.1 Dodavanje, pregled i brisanje pitanja

Za dodavanje pitanja koristi se aktivnost *AddQuestionsActivity*. Korisničko sučelje sastoji se od pet *EditText* elemenata koji se moraju ispuniti, gumba te jednog *spinner* elementa koji sadržava sve stvorene kategorije kvizova.

Prilikom pokretanja aktivnosti, iz baze podataka dohvaćaju se sve kategorije kvizova koje su trenutno stvorene uz pomoć klase *FirebaseHelper* te se kao rezultat dobiva lista objekata klase *Category*. Kada se dohvate sve kategorije kvizova, stvara se nova lista koja će sadržavati nazive kvizova u obliku Stringa. Ova se lista koristi za popunjavanje *spinnera* uz pomoć adaptera.

Nakon što korisnik popuni sva polja i odabere kategoriju kojoj želi dodati pitanje, klikom na gumb poziva se funkcija *addEntry()*. Ovaj funkcija dohvaća sve vrijednosti iz elemenata te provjerava je li lista kategorija prazna. Ako je lista kategorija prazna, to znači da nema niti jednog odabranog kviza te se spremanje pitanja prekida i korisnika se putem *Toast* poruke obavještava da nijedna kategorija kviza nije odabrana. Ako je kviz odabran, dohvaća se njegov naziv te se poziva funkcija *checkFields()* koja provjerava je li sve pravilno uneseno te vraća istinitu ili neistinitu vrijednost. Kada funkcija vrati istinitu vrijednost, instancira se novi objekt klase *FirebaseHelper* s novom referencom te se također stvara novi objekt klase *Question*. U novostvoreni objekt spremaju se unesene vrijednosti te se pitanje sprema u bazu podataka. Slika 6.11. prikazuje programsko rješenje za dodavanje pitanja u odabrani kviz.

```
public void AddEntry(View v){
    String question = this.question.getText().toString();
    String correctAnswer = this.correctAnswer.getText().toString();
    String answer2 = this.answer2.getText().toString();
    String answer3 = this.answer3.getText().toString();
    String answer4 = this.answer4.getText().toString();
    if(!(categoryList.isEmpty())) {
        String category = this.category.getSelectedItem().toString();
        if(checkFields(question, correctAnswer, answer2, answer3, answer4)) {
            FirebaseHelper firebase = new FirebaseHelper( reference: "questions/" + category);
            addQuestion(firebase, question, correctAnswer, answer2, answer3, answer4, category);
            toastMessage("Entry successfully added to database.");
            clearEditText();
        } else {
            toastMessage("Please fill out all the fields and select a valid category.");
        }
    } else {
        toastMessage("There is no category selected.");
    }
}
```

Slika 6.11. Programski kod funkcije addEntry()

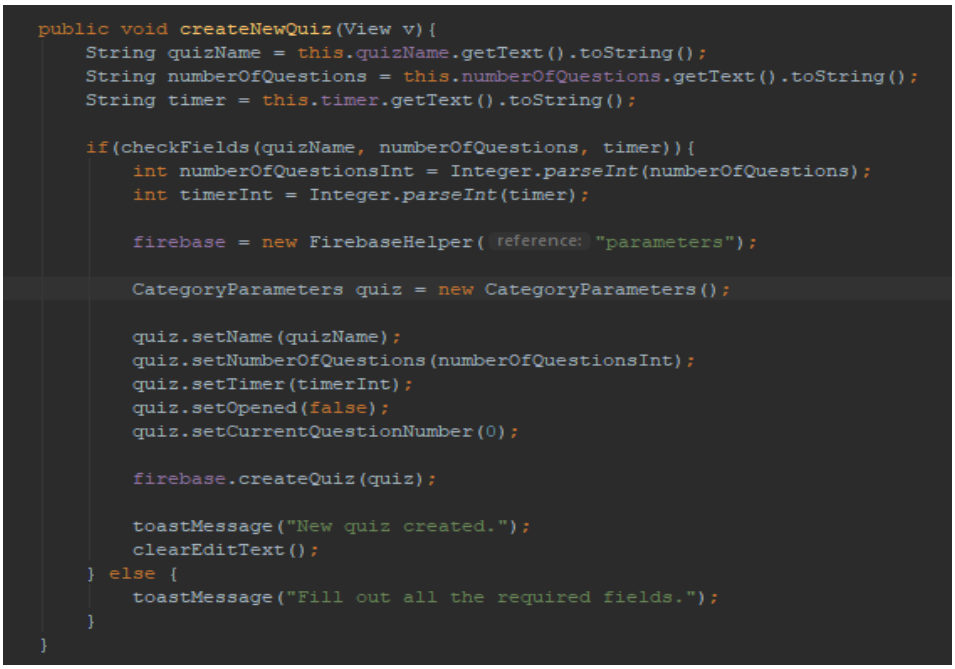
Za pregledavanje, brisanje i uređivanje pitanja koristi se aktivnost *ViewQuestionsActivity* koja se sastoji od *spinner* elementa i gumba. Pri pokretanju aktivnosti ponovno se iz baze podataka dohvaćaju sve stvorene kategorije te se *spinner* popunjava kada se podaci dohvate. Odabirom kategorije i pritiskom na gumb poziva se funkcija *startQuestionsView()* koja dohvaća ime kategorije te ju prosljeđuje *ListViewQuestionsActivity* aktivnosti koristeći *Intent*.

Pri pokretanju aktivnosti dohvaća se ime odabrane kategorije te se zatim dohvaćaju sva pitanja iz baze podataka. Nakon što su sva pitanja dohvaćena, preko objekta posebnog adaptera *QuestionAdapter* dohvaćena pitanja prikazuju se na *ListView* element. Na *ListView* elementu se zatim postavlja osluškivač (engl. *listener*) koji poziva funkciju *onItemClick()* svaki puta kada se klikne na neko pitanje. Ova funkcija dohvaća objekt pitanja na koji je korisnik kliknuo te pokreće aktivnost *EditQuestionsActivity* i prosljeđuje dohvaćeni objekt toj aktivnosti.

Ova se aktivnost koristi za uređivanje i brisanje pitanja. Sastoji se od pet *EditText* elemenata koji su automatski popunjeni vrijednostima iz pitanja te dva gumba - *Delete* i *Save*. Klikom na *Delete* gumb poziva se *deleteQuestion()* funkcija koja iz objekta pitanja dohvaća njegov ID u bazi te ga briše uz pomoć klase *FirebaseHelper*, dok se klikom na *Save* gumb poziva funkcija *updateQuestion()* koja dohvaća sve vrijednosti iz elemenata i sprema dohvaćene vrijednosti u bazu podataka u isti objekt.

6.4.2 Stvaranje, brisanje, uređivanje i otvaranje kvizova

Prije samog dodavanja pitanja u kategorije, prvo se moraju stvoriti kategorije kvizova. Za stvaranje kvizova koristi se aktivnost *NewQuizActivity*. Sučelje se sastoji od dva *EditText* elemenata koja se moraju popuniti nazivom novog kviza i vremenom za rješavanje kviza u minutama. Nakon popunjavanja elemenata i pritiskom na gumb poziva se funkcija *createNewQuiz()* koja prvo dohvaća sve unesene vrijednosti te provjerava je li sve pravilno uneseno pozivajući funkciju *checkFields()* koja vraća istinitu ili neistinitu vrijednost ovisno o tome je li sve pravilno uneseno. Zatim se stvara novi objekt klase *CategoryParameters* ako je sve pravilno uneseno te se unesene vrijednosti unose u objekt koristeći njegove *set()* metode. Nakon postavljanja vrijednosti u objekt, on se prosljeđuje metodi *createQuiz()* klase *FirebaseHelper* koja zatim sprema objekt u bazu podataka te se korisniku prikazuje toast poruka da je kategorija uspješno stvorena. Na slici 6.12. prikazan je programski kod koji dohvaća unesene vrijednosti te sprema kviz u bazu podataka.



Slika 6.12. Programski kod funkcije createNewQuiz()

Otvaranje, brisanje i uređivanje kvizova izvršava se u aktivnosti *ManageQuizActivity*. Sučelje ove aktivnosti se sastoji od *spinnera* koji sadržava sve stvorene kategorije koje se dohvaćaju iz baze podataka pri pokretanju aktivnosti te dva gumba - *Edit* i *Open/Close*. Prilikom odabira kategorije iz *spinnera* vrijednost *Open/Close* gumba mijenja se ovisno o tome je li kviz trenutno zatvoren ili otvoren. Pritiskom na taj gumb poziva se funkcija *toggleQuiz()* koja dohvaća ime odabrane kategorije te pretražuje listu objekata koja sadrži dohvaćene kategorije iz baze kako bi pronašla objekt s tim imenom (Slika 6.13.). Kada je objekt pronađen, postavlja se status kviza na otvoren ili zatvoren, ovisno o tome je li prije bio zatvoren ili otvoren, te se nova vrijednost šalje u bazu podataka preko klase *FirebaseHelper*. Pritiskom na gumb *Edit* poziva se funkcija *editExistingQuiz()* koja dohvaća kategoriju iz liste, prosljeđuje objekt koristeći *Intent* i pokreće aktivnost *EditQuizParametersActivity*.

Sučelje ove aktivnosti sastoji se od jednog *EditText* elementa te dva gumba, *Delete* i *Save*. Element *EditText* pri pokretanju aktivnosti već je popunjen trenutnom vrijednošću vremenskog trajanja kviza. U ovaj element uože se upisati nova vrijednost kviza te se klikom na gimb *Save* poziva funkcija updateCategory() koja postavlja novu vrijednost objekta i onda ga sprema u bazu podataka. Klikom na gumb *Delete* poziva se funkcija *deleteCategory()* koja dohvaća ID objekta u bazi te ga briše.

```
public void toggleQuiz(View v){
    String categoryName = spinner.getSelectedItem().toString();
    CategoryParameters currentCategory = new CategoryParameters();
    int index = 0;
    for(CategoryParameters category : categoryList){
        if(category.getName().equals(categoryName)){
            currentCategory = categoryList.get(index);
            break;
        } else {
            index++;
        }
    }

    if(currentCategory.isOpened()){
        currentCategory.setOpened(false);
        btn.setText("Open quiz");
    } else {
        currentCategory.setOpened(true);
        btn.setText("Close quiz");
    }

    firebase.updateCategory(currentCategory);
}
```

Slika 6.13. Programski kod funkcije toggleQuiz()

6.4.3 Upravljanje korisnicima, pregledavanja poretka i statistike

Svaki korisnik prema zadanome ima ulogu *User* koja ograničava funkcionalnosti kojima on ima pristup. S tom ulogom korisnik može samo rješavati kvizove. Korisnici s ulogom *Admin* imaju dodatne mogućnosti, pa čak i druge korisnike učiniti adminima. Dodjela uloga korisnicima izvršava se u aktivnosti *ManageUsersActivity* koja pri pokretanju dohvaća listu svih registriranih korisnika i prikazuje ih na zaslonu koristeći *ListView* element. Na zaslonu se prikazuju samo korisnička imena te njihove uloge, dok se privatni podaci poput e-mail adrese i lozinke ne prikazuju niti se registrirani korisnici mogu obrisati. *ListView* komponenta na sebi ima osluškivač koji se aktivira kada se klikne na jedan od ponuđenih korisnika te se onda pokreće aktivnost *EditUsersActivity*.

Ovo se sučelje sastoji od dva *TextView* elementa te jednog *spinner* elementa. *TextView* elementi prikazuju korisničko ime odabranog korisnika te njegovu trenutnu ulogu. Spinner element sadrži moguće uloge korisnika. Klikom na gumb *Save* poziva se *saveUserChanges*() funkcija koja prvo provjerava je li dodijeljena uloga različita od trenutne uloge korisnika te sprema novu ulogu korisnika u bazu podataka preko objekta klase *FirebaseHelper*.

Pregledavanje poretka korisnika i statistike kviza izvršava se u aktivnosti *ViewUserScoreAndStatisticsActivity*. Sučelje ove aktivnosti se sastoji od *spinner* elementa koji je popunjen datumima kada su kvizovi rješavani iz baze podataka koji se dohvaćaju prilikom pokretanja aktivnosti te dva gumba *User scores* i *Statistics*.

Kada je datum odabran, klikom na gumb *User scores* pokreće se nova aktivnost te se datum u obliku stringa prosljeđuje toj aktivnosti. Na temelju datuma se zatim korištenjem metode *getUserScores()* dohvaćaju svi rezultati korisnika koji se nalaze u tom čvoru u bazi te se sortiraju pomoću *orderBy()* funkcije koja ja ugrađena u Firebase. Funkcija kao ulazni parametar prima sučelje *UserScoreStatus*. Rezultat dohvaćanja je lista objekata klase *UserScore* koja je sortirana uzlazno. Koristeći *Collections.reverse()* funkciju lista se preokreće te dobivamo silazno sortiranu listu objekata te se podaci prikazuju na zaslon koristeći poseban adapter *UserScoreAdapter* (Slika 6.14.).

Klikom na gumb *Statistics* pokreće se aktivnost *ListViewQuizStatisticsActivity* koja iz baze podataka dohvaća statistiku kviza za odabrani datum te preko adaptera *QuestionStatisticsAdapter* prikazuje rezultate na zaslon.

```
public List<UserScore> getUserScores(final UserScoreStatus userScoreStatus){
    final List<UserScore> userScores = new ArrayList<>();

    Query query = firebase.orderByChild("score");
    query.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            for(DataSnapshot dataSnapshot1: dataSnapshot.getChildren()){
                UserScore userScore = dataSnapshot1.getValue(UserScore.class);
                userScores.add(userScore);
            }
            userScoreStatus.userScoresAreLoaded(userScores);
        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {

        }
    });

    return userScores;
}
```

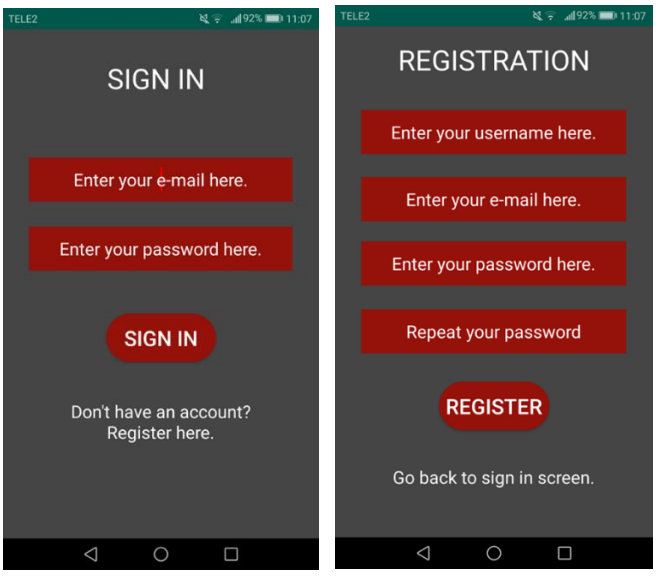
Slika 6.14. Programski kod funkcije getUserScores()

7. PRIKAZ KORIŠTENJA I ISPITIVANJE RADA MOBILNE APLIKACIJE

Ovo će poglavlje sadržavati prikaze sučelja mobilne aplikacije, kako pravilno koristiti sve funkcionalnosti koje ove aplikacija nudi te će pokriti sve moguće slučajeve korištenja aplikacije.

7.1 Početak aplikacije

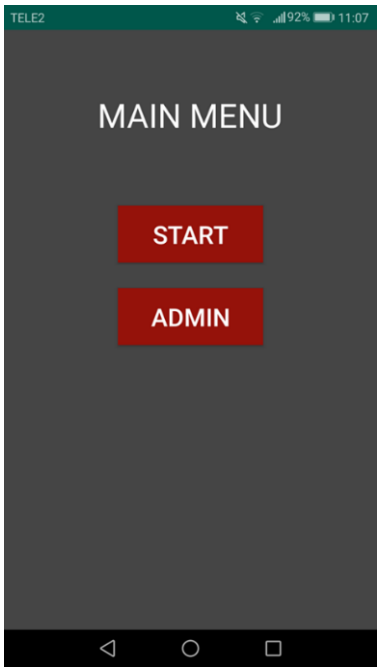
Prilikom pokretanja aplikacije korisniku se prikazuje početni zaslon koji se koristi za prijavu u aplikaciju. Korisnik se može prijaviti u aplikaciju ispunjavanjem polja za e-mail adresu i lozinku te pritiskom gumba *Login*. Ako korisnik nema registriran račun za aplikaciju, pritisak na tekst ispod gumba odvest će ga na zaslon za registraciju računa. Za registraciju računa potrebno je ispuniti odgovarajuća polja s korisničkim imenom, e-mail adresom, lozinkom te ponovljenom lozinkom. Lozinka mora imati najmanje šest znakova a e-mail adresa treba biti pravilno formatirana. Klikom na gumb *Register* izvršava se proces registracije te se korisnika automatski vraća na zaslon za prijavu gdje se može prijaviti sa svojim novim registriranim računom. Izgledi sučelja prikazani su na slici 7.1.



Slika 7.1. Prikazi sučelja za prijavu i registraciju korisnika

7.2 Glavni izbornik

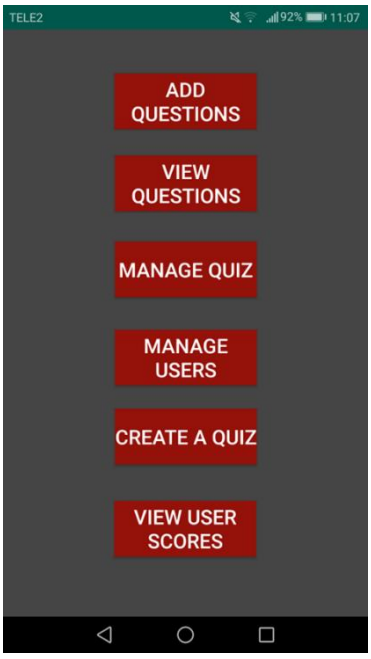
Nakon uspješne registracije i prijave korisnika dolazi se do glavnog izbornika. Ovdje se za običnog korisnika nalazi samo jedan gumb *Start* koji se koristi za odabiranje i rješavanje kviza dok korisnik s administratorskim pravima vidi gumb *Admin* koji ga dovodi do izbornika administratorskih funkcija, kao što se može vidjeti na slici 7.2.



Slika 7.2. Prikaz sučelja glavnog izbornika

7.3 Izbornik administratorskih funkcija

Za pristup administratorskom izborniku potrebno je odabrati gumb *Admin*. Korisnik tada dolazi do izbornika administratorskih funkcija, kao što je prikazano na slici 7.3. Ovdje korisnik vidi izbornik koji se sastoji od šest različitih gumbova koji se koriste za uređivanje i stvaranje kvizova te pregledavanja statistike kviza i poretka korisnika.

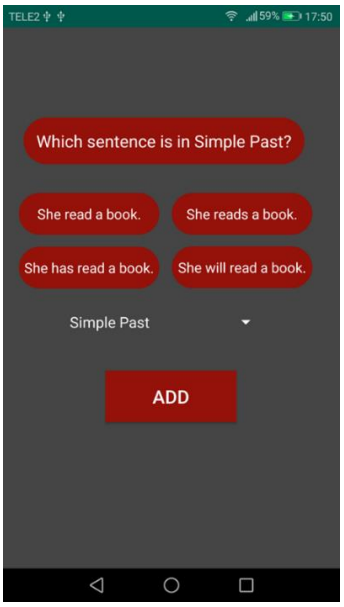


Slika 7.3. Prikaz sučelja administratorskog izbornika

Prvi korak u omogućavanju rješavanja kviza je stvaranje novog kviza. Klikom na gumb *Create a quiz* korisnika se dovodi u aktivnost gdje može stvoriti novi kviz. Ovdje treba popuniti dva polja: naziv kviza te vremensko trajanje kviza, kao što je prikazano na slici 7.4. Nakon ispunjavanja polja, klikom na gumb *Create quiz* stvara se novi kviz u bazi podataka.

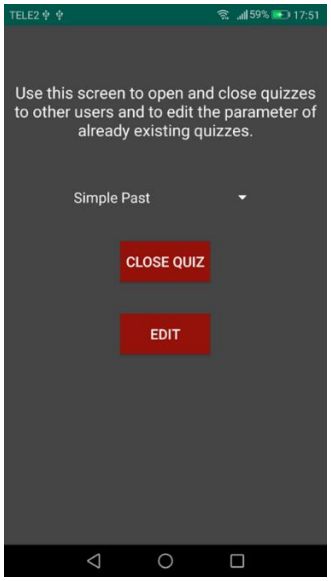
Slika 7.4. Prikaz sučelja aktivnosti za stvaranje kviza

Nakon što je novi kviz stvoren i dodan u bazu, potrebno je u njega dodati pitanja. Odabirom gumba *Add questions* iz administratorskog izbornika pokreće se aktivnost za dodavanje pitanja. Izgled sučelja prikazan je na slici 7.5. U poljima se nalaze natuknice koje korisniku govore kako ih pravilno ispuniti. Iz *spinner* elementa potrebno je odabrati kviz u koji se pitanje želi dodati. Nakon što su sva polja ispunjena i željeni kviz odabran, potrebno je odabrati gumb *Add* koji će zatim dodati pitanje u taj kviz te će se sva polja očistiti.



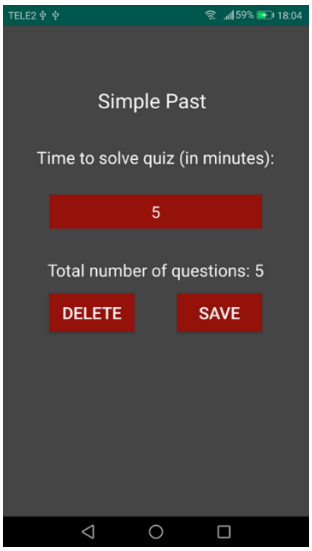
Slika 7.5. Primjer dodavanja pitanja u kategoriju Simple Past

Nakon što su sva pitanja dodana, potreban je još jedan korak prije nego što se on može početi rješavati. Svi novostvoreni kvizovi automatski su zatvoreni te ih je potrebno otvoriti kako bi se mogli rješavati. U administratorskom izborniku potrebno je odabrati gumb *Manage quiz*. Nakon odabira, korisnika se dovodi u aktivnost s definiranim sučeljem (Slika 7.6.). U ovoj je aktivnosti prvo potrebno odabrati kviz koji se želi uređivati. Prvi se gumb koristi za otvaranje i zatvaranje kvizova a njegova je vrijednost uvijek suprotna od statusa kviza. Ako je kviz otvoren, na gumbu će pisati *Close quiz* i obrnuto. Ako se odabere gumb *Edit*, započinje nova aktivnost.



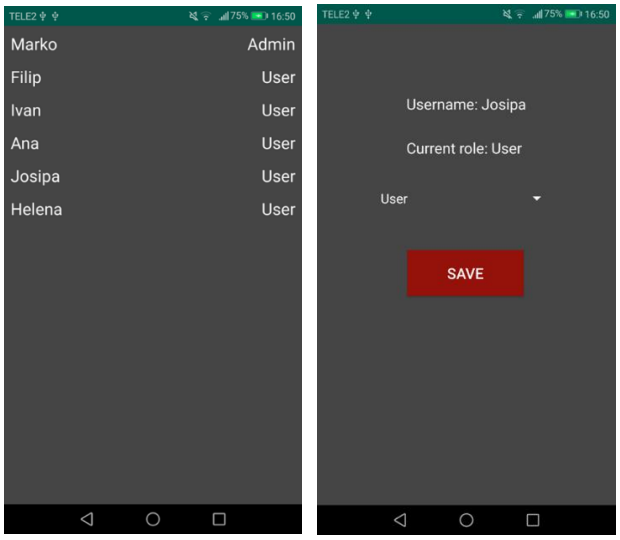
Slika 7.6. Prikaz sučelja aktivnosti za uređivanje kvizova

Kod ove nove aktivnosti može se promijeniti vremensko trajanje kviza te se može vidjeti trenutni broj pitanja koji se nalaze u kvizu. Sučelje za ovu aktivnost prikazano je na slici 7.7. Jednostavno se upiše novi broj u polje te se odabirom gumba *Save* vrijednost sprema. Ako se za odabrani kviz stisne gumb *Delete*, kviz će se obrisati iz baze podataka sa svim svojim pitanjima.



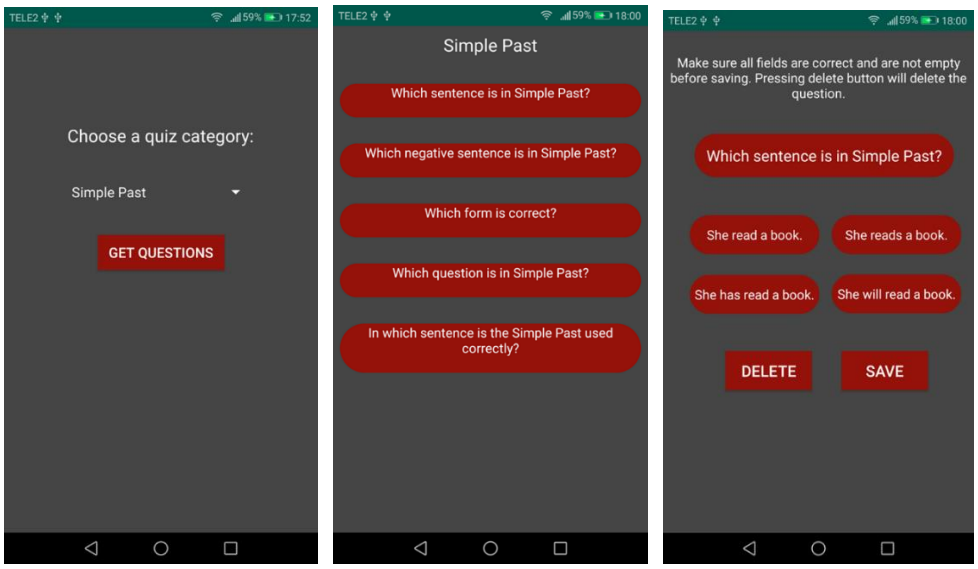
Slika 7.7. Prikaz sučelja za promjenu ili brisanje kviza

Odabirom gumba *Manage users* korisnik može drugim korisnicima dodijeliti uloge. Na zaslonu se ispisuju svi registrirani korisnici s njihovim korisničkim imenima i trenutnim ulogama. Klikom na bilo kojeg od korisnika započet će nova aktivnost koja prikazuje korisnikovu trenutnu ulogu te njegovo korisničko ime. Iz *spinner* elementa potrebno je odabrati ulogu koja se korisniku želi dodijeliti te se klikom na gumb *Save* mijenja korisnikova uloga (Slika 7.8).



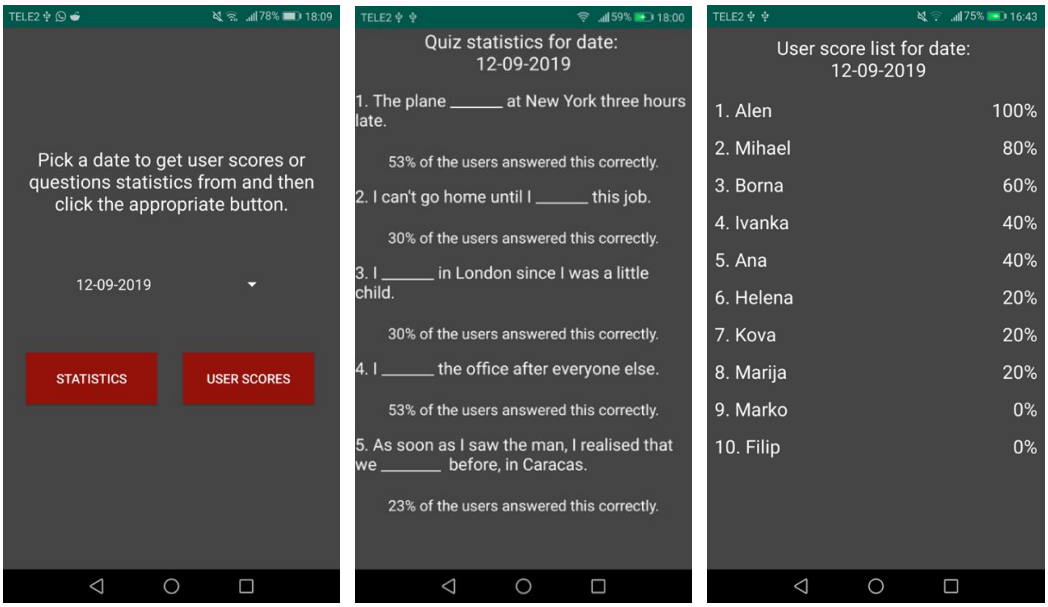
Slika 7.8. Prikaz sučelja za dodjelu uloga korisnicima

Za pregledavanje i uređivanje postojećih pitanja potrebno je u administratorskom izborniku odabrati gumb *View questions*. Ovaj se postupak sastoji od tri različite aktivnosti. U prvoj aktivnosti potrebno je odabrati kviz iz spinner elementa za koji se želi pogledati pitanja. Nakon što je odabran kviz, potrebno je kliknuti gumb *Get questions* te će se korisnika odvesti u novu aktivnost gdje će se prikazati sva pitanja koja se trenutno nalaze u tom kvizu. Ako korisnik želi urediti ili obrisati neka od pitanja, potrebno je klikom odabrati željeno pitanje. Zatim se korisnika dovodi do nove aktivnosti gdje se pitanje može urediti ponovnim ispunjavanjem polja te klikom na gumb *Save* ili se pitanje može obrisati klikom na gumb *Delete*. Na slici 7.9. prikazani su izgledi sučelja za pregled i uređivanje pitanja.



Slika 7.9. Prikaz sučelja aktivnosti za pregled i uređivanje pitanja

Poredak korisnika i statistika kvizova može se pregledati odabirom gumba *Scores and Statistics* u administratorskom izborniku. U ovoj aktivnosti potrebno je iz *spinner* elementa odabrati jedan od ponuđenih datuma. Nakon što je datum odabran, klikom na gumb *User scores* na zaslonu će se ispisati poredak i postotak riješenosti korisnika koji su na taj datum rješavali neki od kvizova. Klikom na gumb *Statistics* može se vidjeti koliko je korisnika točno odgovorilo na pojedino pitanje u postotcima. Slika 7.10. prikazuje izglede sučelja za pregledavanje statistike kviza i poretka korisnika.



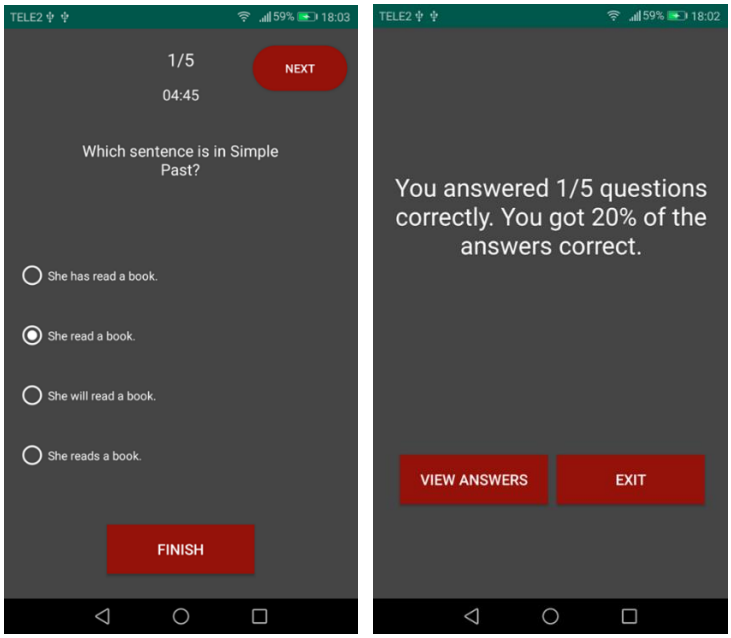
Slika 7.10. Prikaz sučelja aktivnosti za pregled statistike i poretka

7.4 Rješavanje kviza

Za rješavanje kviza potrebno je odmah nakon prijave u glavnom izborniku odabrati gumb *Start*. Na zaslonu se zatim pojavljuju svi kvizovi koji su trenutno stvoreni osim onih koji nemaju niti jedno pitanje dodano. Rješavanje kviza započinje klikom na bilo koji od kvizova pod uvjetom da su kvizovi otvoreni. Ako je kviz zatvoren, korisniku se ispisuje poruka prikazana na slici 7.11.

Slika 7.11. Prikaz poruke kada korisnik odabere zatvoreni kviz

Korisnik započinje s rješavanjem kviza ako mu je dopušten pristup kvizu. Sučelje se sastoji od četiri *RadioButton* elementa koji se koriste za odabir odgovora. Gumbovi *Previous* i *Next* koriste se za kretanje od jednog pitanja do drugog. Korisnikovi odabiri uvijek će biti zapamćeni ako korisnik odluči prebaciti se na neko od drugih pitanja. Pri vrhu zaslona korisnik može vidjeti na kojem se pitanju trenutno nalazi te je ispod toga prikazan vremenski brojač kako bi korisnik znao koliko mu je vremena preostalo za rješavanje kviza. Ponuđeni odgovori mogu se odabrati jednostavnim klikom na njih. Ako korisnik želi ranije završiti kviz, to može napraviti klikom na gumb *Finish*. Tada se korisnika prebacuje na drugačije sučelje koje će prikazati na koliko je točno pitanja odgovoreno te će se njegovi rezultati spremiti. Na ovom će sučelju prikazati dva nova gumba, *View answers* i *Exit*. Klikom na gumb *View answers* korisnika se ponovno vraća na pitanja s tim da će ovaj puta moći pogledati na koja je pitanja točno ili krivo odgovorio. U slučaju da je na pitanje krivo odgovoreno, korisnikov odabir bit će obojan crvenom bojom, dok će točan odgovor biti obojan zelenom bojom. Klikom na gumb *Exit* korisnika se vraća do glavnog izbornika. Na slici 7.12. prikazan je izgled sučelja tijekom i poslije rješavanja kviza.



Slika 7.12. Prikaz sučelja aktivnosti tijekom i poslije rješavanja kviza

8. ZAKLJUČAK

Na početku ovog rada dan je kratak uvod o prednosti korištenja tehnologije kao dodatnog alata u nastavi. Svrha ovoga rada je bila stvoriti mobilnu aplikaciju za operacijski sustav Android kojom će studenti moći testirati svoje znanje stranog jezika te omogućiti nastavnicima da temeljitije provjere usvojenost gradiva. U ovom radu dan je kratak osvrt na već postojeće aplikacije koje izvršavaju sličnu ulogu te su opisane tehnologije koje će se koristiti za izradu mobilne aplikacije. U radu su definirani funkcionalni zahtjevi koje aplikacije mora imati kako bi ispunila svoju svrhu. U glavnom dijelu rada detaljno su objašnjeni mehanizmi i programska rješenja pomoću kojih aplikacija izvršava svoje funkcionalnosti. Ispitivanjem rada mobilne aplikacije utvrđeno je da su svi funkcionalni zahtjevi uspješno ostvareni, a to su mogućnost pregledavanja rješenja kviza, sustav prijave i registracije, mogućnost dodavanje i uređivanja pitanja, stvaranje i uređivanje kvizova, sustav kontrole pristupa kvizu te spremanje i pregledavanja statistike kviza i poretka korisnika. Moguća unapređenja aplikacije su podržavanje dodavanja različitih tipova pitanja te proširivanje sustava za pohranu poretka i statistike kviza. Mobilna aplikacija dodatno će biti ispitana sa studentima u nastavi tehničkog engleskog jezika.

LITERATURA

[1] Z. Ali, M. A. I. M. Ghazali, Learning Technical Vocabulary through a Mobile App: English Language Teachers' Perspectives, International Journal of Language Education and Applied Linguistics (IJLEAL), Vol. 4, 81-91, April 2016.

[2] L. L. Liang, Exploring Language Learning with Mobile Technology: A Qualitative Content Analysis of Vocabulary Learning Apps for ESL Learners in Canada, Electronic Thesis and Dissertation Repository, October 2018.

[3] About Kahoot!, Kahoot!, https://kahoot.com/company/, pristupljeno: 30. lipnja 2019.

[4] Kahoot!, Inclusive Design, http://www.inclusivedesign.no/ict/kahoot-article172-261.html, pristupljeno: 30. lipnja 2019.

[5] Socrative, Edsurge, https://www.edsurge.com/product-reviews/socrative, pristupljeno: 30. lipnja 2019.

[6] Android operacijski sustav, Wikipedija, https://en.wikipedia.org/wiki/Android_(operating_system), pristupljeno: 23. lipnja 2019.

[7] Android studio, Wikipedija, https://en.wikipedia.org/wiki/Android_Studio, pristupljeno: 25. lipnja 2019.

[8] Programski jezik Java, Wikipedija, https://en.wikipedia.org/wiki/Java_(programming_language), pristupljeno: 23. lipnja 2019.

[9] J. T. Tamplin, Firebase is joining Google, Firebase, 2014, https://firebase.googleblog.com/2014/10/firebase-is-joining-google.html, pristupljeno: 24. lipnja 2019.

[10] Products, Firebase, https://firebase.google.com/products/, pristupljeno: 24. lipnja 2019.

SAŽETAK

Rezultat ovog završnoga rada je mobilna aplikacija za operacijski sustav Android koja omogućava provjeru znanja stranog jezika studenata. Korisnici ove aplikacije mogu odabrati posebno stvorene kategorije kvizova koje sadrže pitanja za rješavanje. Neki korisnici imaju administratorska prava koja im omogućuju stvaranje novih kvizova, dodavanje pitanja u njih te pregledavanje poretka korisnika nakon rješavanja i statistike kviza. Teorijski dio ovog rada daje kratak opis i povijest korištenih tehnologija. Nakon teorijskog dijela, postavljaju se funkcionalni zahtjevi za ovu aplikaciju. Praktičan dio ovoga rada predstavlja mehanizme i detaljno opisuje programski kod pomoću kojega se te funkcionalnosti ostvaruju. Aplikacija je razvijena u razvojnoj okolini Android studio, a kod je napisan programskim jezikom Java. Za bazu podataka korištena je Googleova baza podataka u realnom vremenu Firebase.

Ključne riječi: Android aplikacija, Firebase, kviz, poredak, statistika, strani jezik

ABSTRACT

Title: Mobile application for grammar and vocabulary revision in a language for specific purposes (LSP)

This paper resulted in a mobile application for the Android operating system which allows the testing of students' knowledge of a foreign language. The users of this application can choose specially created quiz categories that contain questions to be solved. Some users have been given administrator rights which allows them to create new quizzes, add questions to these quizzes and examine user ranking and quiz statistics. The theoretical part of this paper gives a short description and history of the technologies used. After the theoretical part, the paper defines user requirements for this application. The practical part of this paper presents mechanisms and describes the programming code which makes the functionalities work in detail. The application has been developed by using Android studio and the code is written in programming language Java. The database used is the Google's real-time database Firebase.

Keywords: Android application, Firebase, quiz, ranking, statistics, foreign language

ŽIVOTOPIS

Jakob Kovač je rođen 18.9.1996 godine u Đakovu. Od 2003. do 2011. godine pohađao je Osnovnu školu Josipa Antuna Čolnića u Đakovu. 2011. godine upisuje 1. gimnaziju Osijek u Osijeku. Od 2012. do 2015. pohađa gimnaziju Antuna Gustava Matoša te iste godine polaže ispit državne mature. Godine 2015. upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija na Sveučilištu Josipa Jurja Strossmayera u Osijeku, stručni studij Elektrotehnike, smjer Informatika.

PRILOZI

Prilog 1. Završni rad „Mobilna aplikacija za ponavljanje gramatike i vokabulara u stranom jeziku struke” u *.docx* formatu

Prilog 2. Završni rad „Mobilna aplikacija za ponavljanje gramatike i vokabulara u stranom jeziku struke” u *.pdf* formatu

Prilog 3. Izvorni kod programskog rješenja mobilne aplikacije