Aplikacija za učenje propisa u prometu

Blažević, Nikola

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

Nikola Blažević

APLIKACIJA ZA UČENJE PROPISA U PROMETU

Diplomski rad

Pula, Rujan 2020.

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

Nikola Blažević

APLIKACIJA ZA UČENJE PROPISA U PROMETU

Diplomski rad

JMBAG: 0303054285, redovni student

Studijski smjer: Informatika

Predmet: Mobilne aplikacije

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: doc. dr. sc. Siniša Sovilj

Pula, Rujan 2020.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani **Nikola Blažević**, kandidat za **magistra informatike** ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli,_____, 2020. godine

IZJAVA

o korištenju autorskog djela

Ja, **Nikola Blažević** dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom **aplikacija za učenje propisa u prometu** koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli,_____

Potpis

_____

SVEUČILIŠTE JURJA DOBRILE U PULI

FAKULTET INFORMATIKE

Pula, 11. prosinca 2019.

DIPLOMSKI ZADATAK

Pristupnik: **Blažević Nikola (0303054285)**
Studij: Sveučilišni diplomski studij Informatike

Naslov (hrv.): **Aplikacija za učenje propisa u prometu**

Naslov (eng.): Application for learning traffic regulations

Opis zadatka: Zadatak je izraditi aplikaciju za učenje prometnih znakova i propisa te unutar aplikacije napraviti mogućnost rješavanja ispita. Ispit bi trebao nalikovati ispitima autoškole (propisi, prva pomoć).
Istražiti mogućnosti povezivanja mobilne aplikacije s web servisom koji bi omogućio administriranje baze ispita i pitanja te osvježavanje mobilne aplikacije novim sadržajem.
Aplikaciju izraditi u Android Studio okruženju pomoću Java programskog jezika.

Zadatak uručen pristupniku: 18. prosinac 2019.

Rok za predaju rada: 1. rujan 2020.

Mentor:

_____

doc.dr.sc. Siniša Sovilj

Sadržaj

1. Uvod

Kako u hrvatskoj, tako i u svijetu, cestovni promet spada u jedan od najsloženijih i najkorištenijih sustava kojima se koriste osobe obučene za vožnju motornih vozila. Kako bi se osobe obučile vožnji motornih vozila moraju položiti sve ispite autoškole u koje spadaju: propisi, prva pomoć i praktični dio (vožnja motornog vozila). Svrha edukacije vozača je pružanje znanja, vještina i stavova potrebnih za sigurnost vozila i kao vozača i kao pješaka. Kada osoba koja je kandidat za vozača položi ispite u skladu sa zakonima (NN 141/2011) dobiva vozačku dozvolu i može koristiti motorna vozila položene kategorije.

Velik dio kandidata za vozače spada pod mlađi dio populacije stanovništva što znači da su dobro upoznati s mobilnim uređajima. To otvara mogućnost prebacivanja metode učenja s klasične metode gdje se koriste knjige na praktičniju metodu učenja s mobilne aplikacije. Ovaj diplomski rad se bavi razvojem i izradom aplikacije za učenje propisa u prometu. Kao što i sam naziv aplikacije implicira, aplikacija za učenje propisa u prometu je mobilna aplikacija koju osobe mogu koristiti za učenje gradiva propisa i prve pomoći. Razvijena aplikacija je kreirana u *android studio* razvojnom okruženju te koristi *Firebase API* servise za potrebe baze podataka kao što je korištenje gradiva potrebnog za kreaciju sadržaja aplikacije. Jedna od glavnih motivacija za izradu aplikacije za učenje propisa u prometu je manjak aplikacija za svrhu obučavanja vozača. Aplikacija nudi korisnicima učenje gradiva autoškole pomoću vježbi i ispita, te opcija gdje korisnik može čitati gradivo koje ga zanima ili želi učiti. Također aplikacija nudi korisniku uvid u rješenja vježbi i ispita gdje može vidjeti rezultate prijašnje riješenih vježbi i ispita. Također nudi detaljan pregled svakog riješenog ispita gdje korisnik može vidjeti svaki odgovor koji je dao na pitanje. Aplikacija nudi i administratorske opcije kojima administrator ima mogućnost vršenja izmjeni na bazu podataka.

Ovaj rad se sastoji od 7 poglavlja. Ta poglavlja su uvod, šest glavnih poglavlja i zaključak rada. Prvo poglavlje je poglavlje s uvodom. Drugo poglavlje sadrži informacije o ispitima iz autoškole, te kako ih polagati i proći, te što napraviti u slučaju da se ne prođu. Treće poglavlje opisuje *android* i *javu* gdje se opisuje *android studio* razvojno okruženje, kako

java funkcionira te opis android aplikacija. Četvrto poglavlje opisuje bazu podataka aplikacije za koju se koriste *Firebase* servisi. Peto poglavlje rada govori o motivaciji za izradu aplikacije. Šesto poglavlje koje je najveće poglavlje ovoga rada opisuje implementaciju i funkcionalnosti praktičnog dijela rada. Prvo su prikazani i opisani *UML use case* i *sequence* dijagrami. Potom se opisuje kako se izradila baza podataka te njezina struktura. Zatim se opisuje izrada aplikacije raspoređena na dijelove. Na kraju je sedmo poglavlje koje sažima prethodno navedeno te opisuje rezultate postignute aplikacijom.

2. Ispiti u autoškolama

U jedan od najsloženijih sustava na svijetu spada cestovni promet koji ima svrhu organiziranog kretanja u određenim vremenskim i prostornim uvjetima, tako da se prijevoz obavlja sigurno i racionalno. Kako bi se promet odvijao sigurno i racionalno treba se držati utvrđenih prometnih propisa i pravila. Ti propisi i pravila su utvrđeni Zakonom o sigurnosti prometa na cestama i ostalim pratećim propisima.

Kako bi osoba postala vozač, dužna je osposobiti se u autoškoli tako da nauči i položi cestovne propise i prvu pomoć te vozački ispit. Kada osoba položi sve prije navedeno dobiva vozačku dozvolu kojom zakonski ima dopuštenje upravljanja motornih vozila za kategoriju koju je polagao. Kako bi osoba mogla postati vozač mora imati najmanje 18 godina, a svaki vozač koji još nije navršio 24 godine spada u kategoriju mladih vozača. Postoje različite kategorije vozila koje su namijenjene kretanju po cesti i isto tako postoje različite kategorije vozačkih dozvoli.

2.1. Ispit iz propisa

Ispit iz propisa kandidat polaže na računalu, a može polagati i na odgovarajućem tiskanom obrascu, nakon što završi osposobljavanje. Ispit se obvezno polaže prije početka osposobljavanja upravljanja vozilom. Ispit iz propisa osoba je dužna položiti u roku od godine dana od dana završetka osposobljavanja. Osoba i koja ispit iz propisa ne položi u roku ili taj ispit ne položi niti nakon petog pokušaja u roku od godine dana od dana završetka osposobljavanja mora proći ponovno osposobljavanje (NN 141/2011).

Sastoji se od 38 pitanja koji mogu imati jedan ili više točnih odgovora. Svako pitanje ima određeni broj bodova (2,3,4 ili 7 bodova), a sveukupno ispit ima 120 bodova. Kako bi se ispit položio potrebno je skupiti najmanje 90% bodova tj. 108 bodova. U slučaju da se skupi najmanje 108 bodova, ali se krivo odgovori na pitanje koje se odnosi na propuštanje vozila na raskrižju osoba pada ispit te ga mora ponovo polagati. Kada se jednom ispit položi, osoba može početi osposobljavanje upravljanja vozilom.

2.2. Ispit iz prve pomoći

Prva pomoć je skup postupaka kojima se pomaže ozlijeđenoj ili iznenada oboljeloj osobi na mjestu događaja, prije dolaska hitne pomoći ili drugih kvalificiranih zdravstvenih djelatnika. Na mjestu nesreće prvu pomoć pruža osoba koja se zatekne na mjestu nesreće. Cilj prve pomoći je početak pomaganja i zbrinjavanja ozlijeđene ili naglo oboljele osobe tako da se osobi spasi život, pokušaju spriječiti komplikacije i invalidnosti i skraćivanje liječenja i oporavka (Rogić, 2012: 3).

Ispit iz prve pomoći polaže se individualno i obvezno prije pristupanja ispitu iz upravljanja vozila. Osobe koje već posjeduju važeću vozačku dozvolu nisu dužne polagati prvu pomoć. Isto tako nisu dužni polagati prvu pomoć ni doktori medicine i medicinske sestre i tehničari te primalje. Ispit se sastoji od teorijskog i praktičnog dijela koji se polaže istodobno (NN 141/2011).

2.3. Ispit iz upravljanja vozilom

Ispitu iz upravljanja vozilom pristupa kandidat koji je završio osposobljavanje prema propisanom programu i ako je navršio broj godina potrebnih za izdavanje vozačke dozvole za kategoriju za koju se osposobio, nakon što mu je to stručni voditelj autoškole u kojoj se osposobljavao potpisao i ovjerio u Knjižici kandidata za vozača. Osoba je obvezna položiti ispit iz upravljanja vozilom na javnoj cesti u roku koji ne može biti duži od 18 mjeseci od dana kada je položio ispit iz propisa (NN 141/2011).

Ispit se polaže individualno za pojedinu kategoriju sukladno programu vozačkog ispita. Dio ispita za pojedinu kategoriju vozila na javnoj cesti osoba može polagati ako je prethodno položila dio ispita, u kojem se provjerava izvođenje posebnih radnji vozilom na prometnom vježbalištu. Dio ispita za pojedinu kategoriju vozila u kojem se provjerava izvođenje posebnih radnji vozilom na prometnom vježbalištu osoba polaže samostalno bez nazočnosti instruktora vožnje u vozilu. Ovlašteni ispitivač njegovu vožnju prati i ocjenjuje izvan vozila, osim u slučajevima kada je to drugačije propisano. Tijekom provedbe ispita na javnoj cesti, u vozilu osim osobe koja polaže ispit i ovlaštenog

ispitivača može biti nazočan instruktor vožnje te ovlaštena osoba za nadzor. Osoba koja nije položila ispit na javnoj cesti nije obvezna ponovo polagati dio ispita u kojem se provjerava izvođenje posebnih radnji vozilom (NN 141/2011).

Ako je u vozilu, instruktor vožnje ne smije utjecati na način ispitivanja osobe koja polaže ispit, izbor sadržaja ispita niti na bilo koji način utjecati na provedbu ili ishod ispita. Kada postoji objektivna opasnost da će osoba izazvati prometnu nesreću ili materijalnu štetu, instruktor vožnje dužan je intervenirati (NN 141/2011).

Osobi koja je položila ispit, ovlašteni ispitivač mora priopćiti obrazloženje ocjene i upute za samostalnu i sigurnu vožnju te za odgovorno ponašanje u prometu, a koja nije, ovlašteni ispitivač mora priopćiti što je dobro učinio te greške učinjene tijekom vožnje, radi kojih nije položio ispit, a koje je za vrijeme provedbe ispita uočio i upisao u ispitni list. Osoba koja nije položila ispit je obavezna realizirati dodatno osposobljavanje u trajanju od najmanje tri nastavna sata (NN 141/2011).

3. Android i Java

Android Studio je službeno integrirano razvojno okruženje za razvoj aplikacija za Android pametne uređaje. Temelji se na IntelliJ IDEA, koje je Java integrirano razvojno okruženje softvera, a uključuje svoje alate za uređivanje koda i programere. Da bi podržao razvoj aplikacija u Android operativnom sustavu, Android Studio koristi *Gradle*[1] zasnovan sustav, emulator, predloške gotovih kodova i integraciju *Github*-a temeljenu na *Gradle-u*. Svaki projekt u Android studiju ima jedan ili više modaliteta s izvornim kodom i datotekama resursa. Ti modaliteti uključuju module aplikacija za Android, biblioteke i module Google App Enginea (Android Studio, 2020).

Android je sveobuhvatna platforma otvorenog koda namijenjena mobilnim uređajima. Cilj androida je ubrzati inovacije u mobilnim uređajima i ponuditi potrošačima bogatije, jeftinije i bolje mobilno iskustvo. Android to može učiniti. Kao takav, Android izvršava revoluciju u mobilnom svijetu (Gargenta, Nakamura, 2014:1).

Android Studio koristi značajku *Instant Push* za pokretanje promijenjenog koda i resursa u pokretanoj aplikaciji. Editor za kod pomaže programeru u pisanju koda i nudi kodove za dovršavanje, refrakciju i analizu koda. Aplikacije izgrađene u Android studiju sastavljaju se u APK formatu i predaju u Google Play Store. Softver je prvi put najavljen na Google I / O u svibnju 2013., a prva stabilna verzija objavljena je u prosincu 2014. Android Studio dostupan je za Mac, Windows i Linux radne platforme. Zamijenio je Eclipse Android razvojne alate (ADT) kao primarni IDE za razvoj aplikacija za Android. Android Studio i komplet za razvoj softvera (SDK) mogu se preuzeti izravno s Googlea (Android Studio, 2020).

3.1. Java

Java je programski jezik koji se koristi za proizvodnju softvera za više platformi. Kada programer napiše Java aplikaciju, sastavljeni kod radi na većini operativnih sustava (OS). Java svoj dio sintakse izvodi iz programskog jezika C i C ++ (Java, 2020).

[1] Sustav gradnje Android Studio temelji se na Gradle-u, a dodatak Android Gradle dodaje nekoliko značajki specifičnih za izgradnju Androidovih aplikacija.

Za razvoj Java programa potreban je Java razvojni komplet (eng. java development kit, JDK) koji obično uključuje kompilator, interpreter, generator dokumentacije i druge alate koji se koriste za izradu kompletne aplikacije (Java, 2020).

Vrijeme razvoja se ubrzava uporabom integriranog razvojnog okruženja. Integrirana razvojna okruženja olakšavaju GUI (eng. Graphical user interface) razvoj, koji uključuje gumbe, okvire za tekst, panele, okvire, scrollbarove i druge objekte pomoću *drag and drop*[2] poteza i *point nad click*[3] poteza (Java, 2020).

3.1.1. Java razvojni komplet (JDK)

Java razvojni komplet je razvojno okruženje softvera koje se koristi za razvoj Java aplikacija i apleta. To uključuje JRE[4], interpreter / loader (java), kompilator (javac), arhiver (jar), generator dokumentacije (javadoc) i druge alate potrebne za Java razvijanje. Za pokretanje Java aplikacija i appleta se preuzima JRE. No, za razvoj Java aplikacija i appleta i njihovo pokretanje potreban je JDK (Java Development Kit, 2020).

Java programeri u početku su predstavljeni s dva JDK alata, java i javac. Java izvorne datoteke su jednostavne tekstualne datoteke spremljene s nastavkom .java. Nakon pisanja i spremanja Java koda, javac kompilator se poziva da stvori .class datoteke. Nakon što se stvore datoteke .class, naredba 'java' može se koristiti za pokretanje java programa JDK. Postoje različiti JDK-ovi za razne platforme. Korisnici Mac računala trebaju drugačiji paket za razvoj softvera, koji uključuje prilagodbe nekih alata koji se nalaze u JDK (Java Development Kit, 2020).

3.1.2. Java virtualni stroj (eng. Java Virtual Machine, JVM)

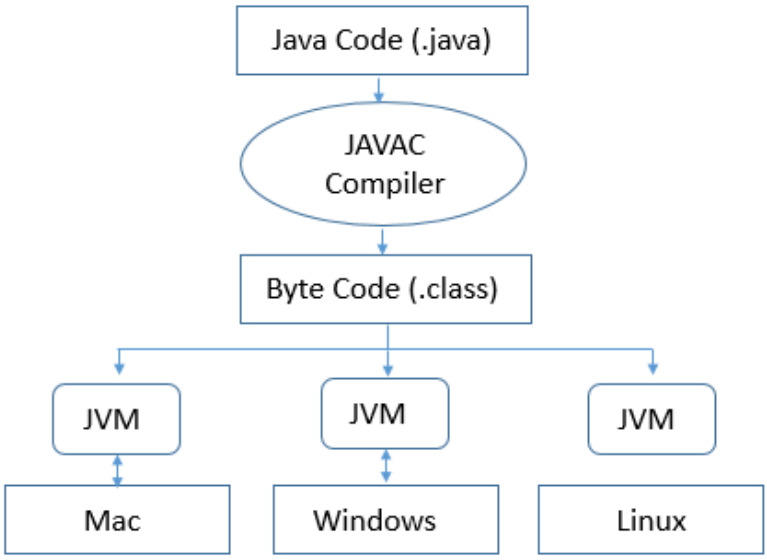
Java virtualni stroj specifikacija je koja pruža okruženje za vrijeme izvođenja u kojem se može izvršavati java *bytecode*[5]. Kao što naziv govori, JVM djeluje kao "virtualni" stroj ili procesor. Neovisnost platforme Java sastoji se većinom od Java virtualnog stroja.

[2] Povlačenje i ispuštanje omogućuje korisnicima premještanje podataka iz jednog View-a u drugi
[3] Označava sučelje u kojem korisnik pokaže i klikne kako bi pokrenuo funkciju.
[4] Java Runtime Environment (JRE) skup je softverskih alata za razvoj Java aplikacija.
[5] Bytecode je programski kod koji je preveden iz izvornog koda u niskorazinski kod dizajniran za softverski interpreter.

JVM ovo omogućuje jer je svjestan specifičnih duljina uputa i drugih posebnosti platforme. JVM obavlja sljedeće operacije (What is Java virtual machine?, 2020):

- Učitava se kod
- Provjerava kod
- Izvršava kod

U većini slučajeva, u ostalim programskim jezicima, kompilator proizvodi kod za određeni operativni sustav, ali Java kompilator proizvodi *Bytecode* samo za Java virtualni stroj. Kada pokrenete Java program, on se pokreće kao dretva[6] unutar JVM postupka. Zadatak JVM-a je da učita datoteke klase, učita kod, provjeri ga i izvrši. Kada se izdaje naredba kao što je Java, JVM učitava definiciju klase za tu klasu i poziva glavnu metodu te klase (What is Java virtual machine?, 2020).



Slika 1. Primjer JVM-a na različitim operacijskim sustavima

Izvor: http://net-informations.com/java/intro/jvm.htm

Zadatak JVM-a je omogućavanje pokretanja iste datoteke klase na bilo kojem drugom operacijskom sustavu. JVM uzima sastavljeni bajt-kod neutralni na platformi i tumači ga

[6] Dretve predstavljaju oblik paralelizacije na razini procesa. Svaki se proces može podijeliti u jedan broj dretvi koje se izvršavaju prividno paralelno, na isti način kao i procesi.

da pokreće strojni kod specifičan za platformu kao što je vidljivo na slici 1. Također se može sastaviti u izvorni kod pomoću JIT (pravodobni kompilator koji sastavlja i stavlja kod u predmemoriju, obično jednu metodu odjednom). Prema tome, kod JVM-a se kod, ako je potrebno, nalazi u pozivima operativnog sustava. Stoga se u JVM-u kod za dretve koji se neutralizira na platformi pretvara u kod za dretve koji je specifičan za platformu (What is Java virtual machine?, 2020).

3.2. Android aplikacije

Android aplikacija je softverska aplikacija koja radi na Android platformi. Budući da je Android platforma stvorena za mobilne uređaje, tipična Android aplikacija dizajnirana je za pametni telefon ili tablet-računalo na Android OS-u (Android App, 2020).

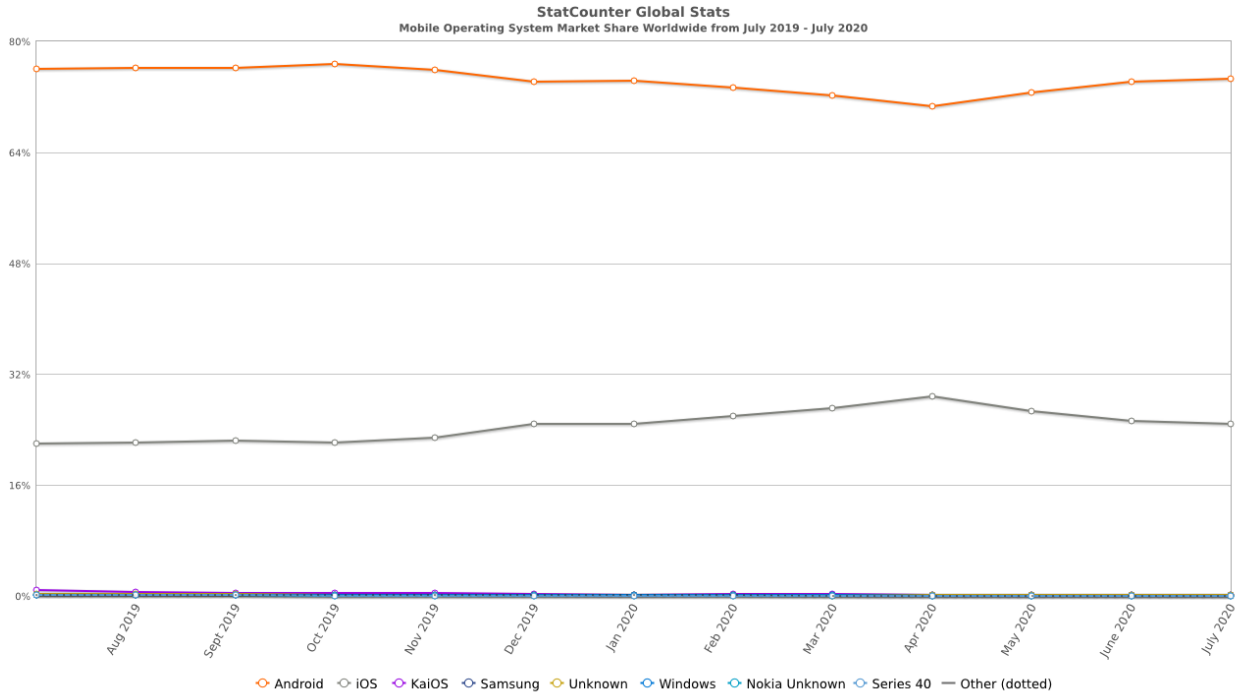
Iako Android programeri mogu omogućiti dostupnost aplikacija putem svojih web stranica, većina Android aplikacija učitava se i objavljuje na Android Marketu, internetskoj trgovini posvećenoj tim aplikacijama. Android Market sadrži i besplatne i cjenovne aplikacije. Aplikacije za Android napisane su u programskom jeziku Java i koriste Java *core* biblioteke. One se prvo sastavljaju u izvršnim datotekama kako bi se izvodile na virtualnom stroju. Programeri mogu preuzeti Android razvojni komplet (eng. software development kit, SDK) s Android web stranice. SDK uključuje alate, primjerak koda i relevantne dokumente za stvaranje Android aplikacija (Android App, 2020).

3.2.1. Mobilni operativni sustav

Mobilni operativni sustav je OS izgrađen isključivo za mobilne uređaje, poput pametnih telefona, tableta ili drugih ugrađenih mobilnih OS-a. Najpopularniji mobilni operativni sustavi su Android i iOS. Još neki od mobilnih OS sustava su KaiOS, Samsung, Windows phone i Nokia (Mobile Operating System, 2020).

Mobilni OS odgovoran je za prepoznavanje i definiranje značajki i funkcija mobilnih uređaja, uključujući tipkovnice, sinkronizaciju aplikacija, e-poštu i tekstualne poruke. Mobilni OS je sličan standardnom OS-u (poput Windows, Linuxa i Mac-a), ali je relativno jednostavan i lagan te prvenstveno upravlja bežičnim varijacijama lokalnih i širokopojasnih veza, mobilnom multimedijom i različitim metodama unosa podataka. Da

bi se prilagodio inherentnim okruženjima mobilnih uređaja, mobilni OS radi na ograničenim resursima koji naglašavaju komunikaciju, poput RAM memorije, pohrane i brzine procesora (Mobile Operating System, 2020).



Slika 2. Postotak korištenih mobilnih OS-a u svijetu za 2019. i 2020. godinu

Izvor: https://gs.statcounter.com/os-market-share/mobile/worldwide

Na slici 2 je vidljivo da je Android OS najviše korišteni operacijski sustav u svijetu te drži čak 74.6% tržišta, dok Apple-ov IOS drži 24.8% tržišta.

3.2.2. Android SDK (eng. software development kit)

Android SDK se koristi za razvoj aplikacija za Android platformu pomoću skupa razvojnih alata. Android SDK uključuje sljedeće (Android SDK, 2020):

- Potrebne biblioteke
- Debugger[7]
- Emulator

[7] Program za ispravljanje pogrešaka

- Odgovarajuću dokumentaciju za sučelja aplikativnog programa Android (API-i)
- Uzorak izvornog koda
- Vodiče za Android OS

Kada izlazi nova verzija Android OS-a izdaje se i odgovarajuća SDK verzija. Da bi se moglo pisati programe s najnovijim značajkama, programeri moraju preuzeti i instalirati SDK verziju za određeni telefon. Komponente Android SDK-a mogu se preuzeti zasebno. Dodaci treće strane također su dostupni za preuzimanje. SDK za pisanje android programa najčešće koristi metode za pisanje android programa pomoću integriranog razvojnog okruženja. Većina ovih integriranih razvojnih okruženja nudi grafičko sučelje koje programerima omogućuje brže izvršavanje razvojnih zadataka. Budući da su Android aplikacije napisane u Java kodu, korisnik bi trebao imati instaliran Java razvojni komplet (Android SDK, 2020).

3.3. XML (eng. Extensible Markup Language) u android aplikacijama

XML je Markup jezik sličan HTML-u[8] koji se koristi za opisivanje podataka. XML oznake nisu unaprijed definirane u XML-u. Moraju se definirati vlastite oznake. XML je sam po sebi dobro čitljiv i od strane ljudi i od strane strojeva. Također je skalabilan i jednostavan za razvijanje. U Androidu XML se koristi za dizajniranje izgleda, jer je XML lak za korištenje.U Androidu postoji više vrsta XML datoteka koje se koriste u nekoliko različitih svrha (XML in Android, 2020):

- Layout[9] datoteka
- Manifest datoteka
- String[10] datoteka
- Datoteka stilova
- Drawable[11] datoteka

[8] HTML je standardni markup jezik za dokumente dizajnirane za prikaz u web pregledniku
[9] Resurs izgleda definira arhitekturu za korisničko sučelje u aktivnosti ili komponentu korisničkog sučelja.
[10] Resurs String-a pruža tekstualne nizove za aplikaciju s dodatnim oblikovanjem i formatiranjem teksta.
[11] Resurs koji se može crtati opći je koncept grafike koja se može nacrtati na zaslon i koji se može dohvatiti pomoću API-ja ili primijeniti na drugi XML resurs s atributima

- Datoteka boja
- Dimenzijska datoteka

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:orientation="vertical" android:layout_width="match_parent"
 android:layout_height="match_parent">


 <Button
 android:id="@+id/buton1"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Button"/>

 <TextView
 android:id="@+id/textView1"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="sample Text"
 android:layout_marginTop="15dp"
 android:textSize="30dp"/>

 <RelativeLayout
 android:layout_width="match_parent"
 android:layout_height="match_parent">

 <EditText
 android:id="@+id/editTextName"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:hint="First Name"
 />

 <EditText
 android:id="@+id/editTextLastName"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:hint="Last Name"/>

 </RelativeLayout>
</LinearLayout>
```

Slika 3. Primjer XML datoteke

Izvor: https://abhiandroid.com/ui/xml

4. Firebase

Firebase je BaaS[12] platforma za stvaranje mobilnih i web aplikacija koju je razvio Google. Prvotno je to bila neovisna tvrtka osnovana 2011. godine. U 2014. Google je kupio platformu i sada je njihova vodeća platforma za razvoj aplikacija (What is Firestore?, 2020).

4.1. Cloud Firestore

Cloud *Firestore* je fleksibilna, skalabilna baza podataka za razvoj mobilnih, web i poslužiteljskih aplikacija od strane *Firebase*-a i Google Cloud Platforme. *Firestore* održava podatkovnu sinkronizaciju između klijentskih aplikacija putem slušatelja u stvarnom vremenu i nudi izvanmrežnu podršku za mobilne uređaje i web, tako da se mogu graditi responzivne aplikacije koje rade bez obzira na mrežno kašnjenje ili internetsku povezanost. Cloud *Firestore* također nudi besprijekornu integraciju s drugim *Firebase* i *Google Cloud Platform* proizvodima. Ključne mogućnosti *Firestore*-a su (Cloud Firestore, 2020):

- **Fleksibilnost** - *Cloud Firestore*-ov model podataka podržava fleksibilne, hijerarhijske strukture podataka. Podatci se mogu spremiti u dokumente organizirane u kolekcije. Dokumenti mogu sadržavati složene ugniježđene objekte uz dodatne podkolekcije.

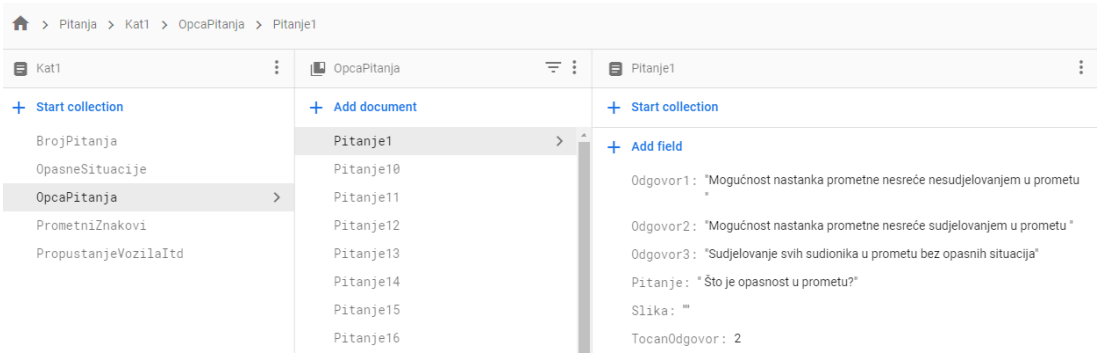
- **Izražajni upiti (eng. Query)** - U *Cloud Firestore*-u upiti se mogu koristiti za dohvaćanje pojedinačnih, određenih dokumenata ili za preuzimanje svih dokumenata u zbirci koji odgovaraju vašim parametrima upita. Vaši upiti mogu uključivati višestruke, lančane filtre i kombinirati filtriranje i razvrstavanje. Oni su također indeksirani prema zadanom, tako da je izvedba upita proporcionalna veličini vašeg skupa rezultata, a ne vašem skupu podataka.

[12] Backend as a Service (BaaS) je model usluge računalstva u oblaku koji služi kao srednji softver koji programerima pruža načine povezivanja njihovih web i mobilnih aplikacija s uslugama u oblaku putem API-ja i SDK-a.

- **Ažuriranja u stvarnom vremenu** - *Cloud Firestore* koristi sinkronizaciju podataka za ažuriranje podataka na bilo kojem povezanom uređaju. No također je dizajniran za učinkovito postavljanje jednostavnih, jednokratnih upita.

- **Offline podrška** - *Cloud Firestore* predmemorira podatke koje aplikacija aktivno koristi tako da aplikacija može pisati, čitati, slušati i raditi upite za podatke, čak i ako je uređaj izvan mreže. Kad se uređaj vrati na mrežu, *Cloud Firestore* sinkronizira sve lokalne promjene natrag u *Cloud Firestore*.

- **Dizajn za skalabilnost** - *Cloud Firestore* donosi najbolju od Google *Cloud Platform* infrastrukturi: automatsko kopiranje podataka u više regija, jake garancije dosljednosti, atomske *batch* operacije i stvarnu podršku za transakcije. *Cloud Firestore* je dizajniran da podnese najteža opterećenja baze podataka iz najvećih svjetskih aplikacija.



Slika 4. Primjer *Cloud Firestore* strukture podataka iz aplikacije

Na slici 4 je prikazana struktura podataka aplikacije za učenje propisa u prometu. Na slici je prikazana podkolekcija općih pitanja unutar koje se nalaze dokumenti s jedinstvenim ID-jevima. Svaki dokument je jedno pitanje koje se sastoji od pitanja, tri odgovora, slike i točnog odgovora.

4.2. Cloud Storage

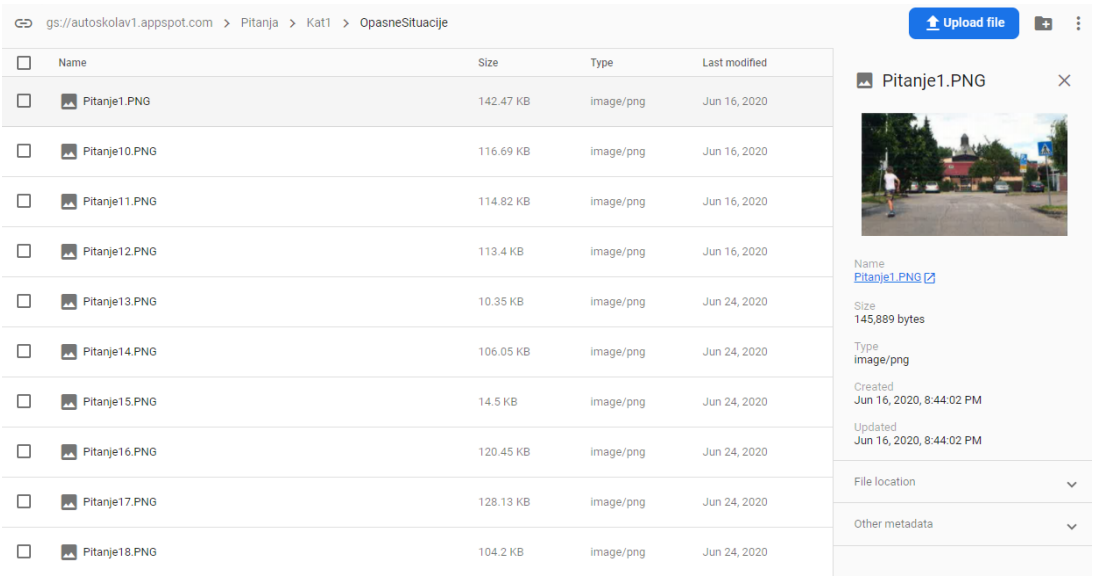
Firebase-ov *Cloud Storage* snažna je, jednostavna i isplativa usluga skladištenja objekata. *Firebase* SDK-ovi za *Cloud Storage* dodaju sigurnost na prijenos datoteka i preuzimanja za *Firebase* aplikacije, bez obzira na kvalitetu mreže. SDK-ovi se mogu koristiti za pohranu slika, audio, video ili drugog sadržaja koje stvara korisnik. Na poslužitelju se može koristiti Google *Cloud Storage* za pristup istim datotekama. Ključne mogućnosti *Cloud Storage*-a su (Cloud Storage, 2020):

- **Robusne operacije** - SDK-ovi Firebase-a za Cloud Storage izvršavaju učitavanja i preuzimanja bez obzira na kvalitetu mreže. Prijenosi i preuzimanja su robusni, što znači da se ponovo pokreću tamo gdje su stali, štedeći vrijeme korisnicima.

- **Snažna sigurnost** - SDK-ovi *Firebase*-a za *Cloud Storage* integriraju se s *Firebase* provjerom autentičnosti kako bi programerima omogućili jednostavnu i intuitivnu provjeru autentičnosti. Mogu se koristiti deklarativni sigurnosni modeli da bi omogućili pristup na temelju imena datoteke, veličine, vrste sadržaja i drugih metapodataka.

- **Visoka skalabilnost** - *Cloud Storage* za *Firebase* izrađen je za exabyte[13] razmjere kada aplikacija postane viralna. Lako se razvija od prototipa do proizvodnje koristeći istu infrastrukturu.

[13] 1 exabyte = 1 000 000 000 gigabytea



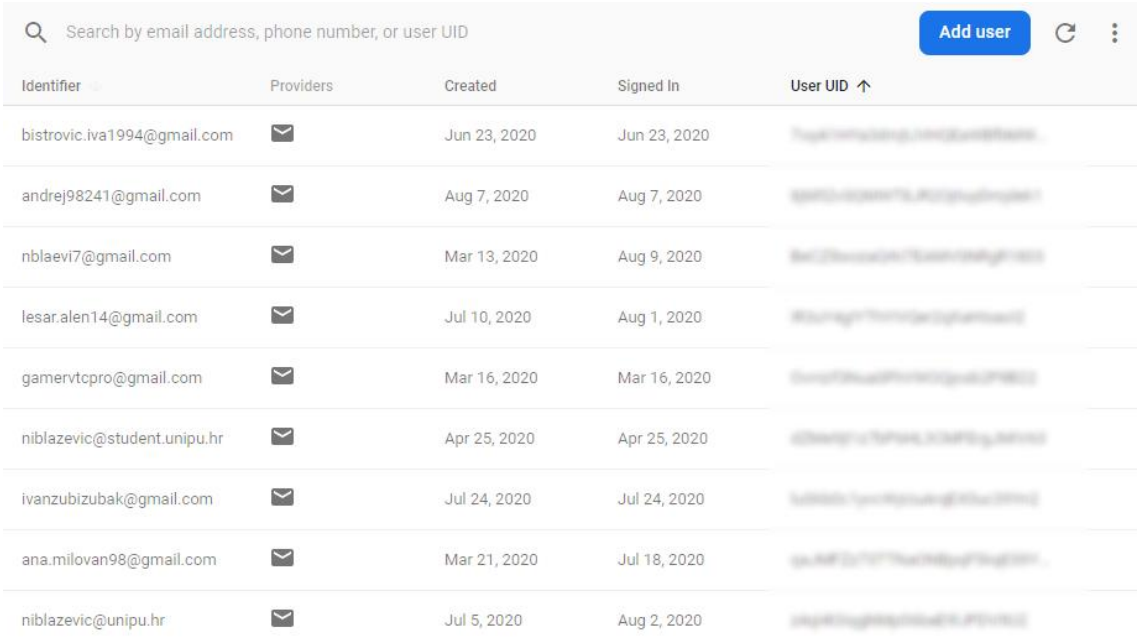
Slika 5. Primjer *Cloud Storage*-a popunjenog PNG slikama iz aplikacije

Slika 5 prikazuje *Cloud Storage* korišten u aplikaciji za učenje propisa u prometu. Unutar *Cloud Storage*-a se nalaze PNG slike koje su povezane s određenim pitanjima unutar *Cloud Firestore*-a pomoću *Access Tokena*[14] svake slike.

4.3. Firebase Authentication

Kako bi se korisnik registrirao u aplikaciju, prvo od korisnika dobivate informacije za autentifikaciju. Ove informacije mogu biti korisnička adresa e-pošte i zaporka ili *OAuth* token od davatelja identiteta. Zatim se proslijede ove informacije *Firebase* SDK-u za provjeru autentičnosti. Usluge pomoćnika provjerit će te informacije i vratiti klijentu odgovor. Nakon uspješne prijave se može pristupiti osnovnim podacima o profilu korisnika i može se kontrolirati pristup korisnika podacima pohranjenim u drugim *Firebase* proizvodima. Također se može koristiti i dostavljeni token provjere identiteta za provjeru identiteta korisnika u vlastitim *backend* uslugama (Firebase Authentication, 2020).

[14] Pristupni ključ za korištenje zaštićenih podataka



Slika 6.Primjer *Firebase Authenticationa* sa registriranim korisnicima iz aplikacije

Na slici 6 je prikazan *Firestore Authentication* od aplikacije za učenje propisa u prometu u kojemu su prikazani registrirani korisnici tj. e-mail, vrijeme kreiranja korisničkog računa, kada se korisnik ulogirao i njihov *UID*.

4.4. Sigurnosna pravila

Sigurnosna pravila *Firebase*-a stoje između podataka i zlonamjernih korisnika. Mogu se napisati jednostavna ili složena pravila koja štite podatke iz aplikacije do razine preciznosti koju zahtijeva specifična aplikacija. Sigurnosna pravila *Firebase*-a koriste proširive, fleksibilne jezike konfiguracije da bi definirali kojim podacima mogu korisnici pristupiti u *Realtime Database*-u, *Cloud Firestore*-u i *Cloud Storage*-u. Sigurnosna pravila za *Cloud Firestore* i *Firebase* sigurnosna pravila za *Cloud Storage* koriste jedinstveni jezik izgrađen za smještaj složenijih struktura specifičnih za pravila. Ključne mogućnosti *Firebase* sigurnosnih pravila su (Firebase Security Rules, 2020):

- **Fleksibilnost** - Pisanje prilagođenih pravila koja imaju smisla za strukturu i ponašanje određene aplikacije. Pravila koriste jezike koji omogućuju korištenje podataka kako bi se autorizirao pristup.

- **Granularnost** - Pravila mogu biti opširna ili jednostavna koliko god je potrebno.

- **Neovisna sigurnost** - Budući da su Pravila definirana izvan aplikacije (u *Firebase* konzoli ili *Firebase* CLI), klijenti nisu odgovorni za provođenje sigurnosti, *bugovi* ne ugrožavaju podatke te su podaci uvijek zaštićeni.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents/users {
    match /{document=**} {
      allow read, write, update, delete : if request.auth != null;
    }
  }
  match /databases/{database}/documents/Pitanja {
    match /{document=**} {
      allow read : if request.auth != null;
      allow write, update, delete : if request.auth.uid == [redacted];
    }
  }
}
```

Slika 7. Primjer sigurnosnih pravila za *Cloud Firestore* iz aplikacije

Slika 7 prikazuje sigurnosna pravila za *Cloud Firestore* za aplikaciju za učenje propisa u prometu. Vidljivo je da se za pristup podataka korisnik mora registrirati. Ako korisnik ima UID od administratora onda dobiva mogućnost operacija pisanja, ažuriranja i brisanja određenih dokumenata u *Cloud Firestore*-u. Također postoje i pravila sigurnosti za *Firebase Storage* koja nalikuju na prethodno navedena pravila.

4.5. Cloud Firestore API za reprezentativni prijenos stanja (eng. Representational state transfer, REST)

REST API je sučelje aplikacijskog programa koje koristi HTTP[15] zahtjeve za GET, PUT, POST i DELETE kako bi se obradili podatci. REST API zasnovan je na reprezentativnom prijenosu stanja, arhitektonskom stilu i pristupu komunikacijama koji se često koriste u razvoju web servisa. REST tehnologija općenito se preferira nad

[15] HTTP je protokol koji se koristi za prijenos podataka putem interneta.

robusnijom tehnologijom SOAP[16], jer REST koristi manju propusnost (eng. bandwidth), što je čini prikladnijom za učinkovito korištenje interneta (RESTful API, 2020).

REST API rastavlja transakciju kako bi stvorio niz malih modula. Svaki modul adresira određeni temeljni dio transakcije. Ova modularnost pruža programerima veliku fleksibilnost, ali programerima može biti teško dizajnirati svoj REST API od početka. REST API koristi postojeće HTTP metodologije definirane RFC 2616 protokolom. Oni koriste GET za pronalaženje resursa; PUT za promjenu stanja ili ažuriranje resursa, koji može biti objekt, datoteka ili blok; POST za stvaranje tog resursa; i DELETE da ga uklonite (RESTful API, 2020).

REST API može biti od pomoći za sljedeće slučajeve upotrebe (REST API, 2020):

- Pristup *Cloud Firestore*-u iz okruženja s ograničenim resursima, kao što je Internet stvari (IoT), gdje pokretanje kompletne biblioteke klijenata nije moguće.
- Automatizacija administriranja baze podataka ili dohvaćanje detaljnih metapodataka baze podataka.

Radi provjere autentičnosti, *Cloud Firestore* REST API prihvaća ili *Firebase authentication ID* token za provjeru identiteta ili token *Google Identity OAuth 2.0.* Nakon što se dobije ili token od *Firebase authetication ID*-a ili token od *Google Identity OAuth 2.0*, proslijedi ga se krajnjim točkama *Cloud Firestore*-a kao zaglavlje autorizacije postavljeno na „Bearer {Token}“ (REST API, 2020).

Sve krajnje točke REST API-ja postoje pod osnovnim URL-om „https://firestore.googleapis.com/v1/“. Za kreiranje puta do određenog dokumenta ili kolekcije treba unijeti ostatak URL-a kao npr. „projects/autoskolav1/databases/(default)/documents/Pitanja/“.

[16] Simple Object Access Protocol je specifikacija protokola za razmjenu poruka strukturiranih informacija u implementaciji web usluga u računalnim mrežama.

REST Resource: v1.projects.databases.documents

Methods	
batchGet	POST /v1/{database=projects/*/databases/*}/documents:batchGet Dohvatite više dokumenata.
batchWrite	POST /v1/{database=projects/*/databases/*}/documents:batchWrite Provedite brojne operacije pisanja.
beginTransaction	POST /v1/{database=projects/*/databases/*}/documents:beginTransaction Započnite novu transakciju.
commit	POST /v1/{database=projects/*/databases/*}/documents:commit Obavljajte transakcije, a izborno ažurirajte dokumente.
createDocument	POST /v1/{parent=projects/*/databases/*/documents/**}/{collectionId} Izradite novi dokument.
delete	DELETE /v1/{name=projects/*/databases/*/documents/*/**} Brisanje dokumenata.
get	GET /v1/{name=projects/*/databases/*/documents/*/**} Dohvatite jedan dokument.
list	GET /v1/{parent=projects/*/databases/*/documents/*/**}/{collectionId} Popis dokumenata.
listCollectionIds	POST /v1/{parent=projects/*/databases/*/documents}:listCollectionIds Navedite sve ID-ove zbirke pod dokumentom.
partitionQuery	POST /v1/{parent=projects/*/databases/*/documents}:partitionQuery Podjelite upit vraćanjem pokazivača particije koji se može koristiti za paralelno pokretanje upita.
patch	PATCH /v1/{document.name=projects/*/databases/*/documents/*/**} Ažurirajte ili umetnite dokumente.
rollback	POST /v1/{database=projects/*/databases/*}/documents:rollback Vratite transakciju.
runQuery	POST /v1/{parent=projects/*/databases/*/documents}:runQuery Pokrenite upit.

Slika 8. *Cloud Firestore* zahtjevi za REST API

Izvor: https://firebase.google.com/docs/firestore/reference/rest/

Na slici 8 su vidljive metode koji se koriste u *Cloud Firestore* REST API-u. U aplikaciji za učenje propisa u prometu se koriste GET, PATCH i DELETE metode.

4.5.1. Kodovi grešaka u Cloud Firestore REST API-u

Kada zahtjev na *Cloud Firestore* uspije, API *Cloud Firestore* vraća HTTP „200 OK“ kod statusa i tražene podatke. Kad zahtjev ne uspije, API *Cloud Firestore* vraća HTTP 4xx ili 5xx kod statusa i odgovor s informacijama o pogrešci. Ti kodovi su sljedeći (REST API, 2020):

- **ABORTED** - Zahtjev je u konfliktu s drugim zahtjevom.

- **ALREADY_EXISTS** - Zahtjev je pokušao stvoriti dokument koji već postoji.
- **DEADLINE_EXCEEDED** - *Cloud Firestore* poslužitelj koji obrađuje zahtjev prekoračio je vremenski rok.
- **FAILED_PRECONDITION** - Zahtjev nije ispunio jedan od potrebnih preduvjeta.
- **INTERNAL** - *Cloud Firestore* vratio je pogrešku.
- **INVALID_ARGUMENT** - Parametar zahtjeva uključuje nevažeću vrijednost.
- **NOT_FOUND** - Zahtjev je pokušao ažurirati dokument koji ne postoji.
- **PERMISSION_DENIED** - Korisnik nije ovlašten podnijeti ovaj zahtjev.
- **RESOURCE_EXHAUSTED** - Projekt je premašio ili svoju kvotu ili kapacitet regije ili više regija.
- **UNAUTHENTICATED** - Zahtjev nije uključivao valjane informacije za autentifikaciju.
- **UNAVAILABLE** - *Cloud Firestore* poslužitelj vratio je pogrešku.

5. Motivacija za izradu aplikacije

Slika 9. SWOT analiza aplikacije

Većina ljudi koja ide polagati vozački ispit u autoškole je mlada populacija stanovništva. Mlada populacija stanovništva je dobro upoznata s mobilnim uređajima i kako ih koristiti, ali većina ljudi koja polaže vozački ispit uči gradivo iz knjiga jer nema dovoljnu ponudu aplikacija s kojih bi mogli učiti. Knjige iz kojih se uči gradivo naravno nisu besplatne te je potrebno kupiti knjigu. Knjige trebaju imati alternativnu zamjenu kao što su mobilne aplikacije. Mobilna aplikacija može biti besplatna, ali to nije jedina

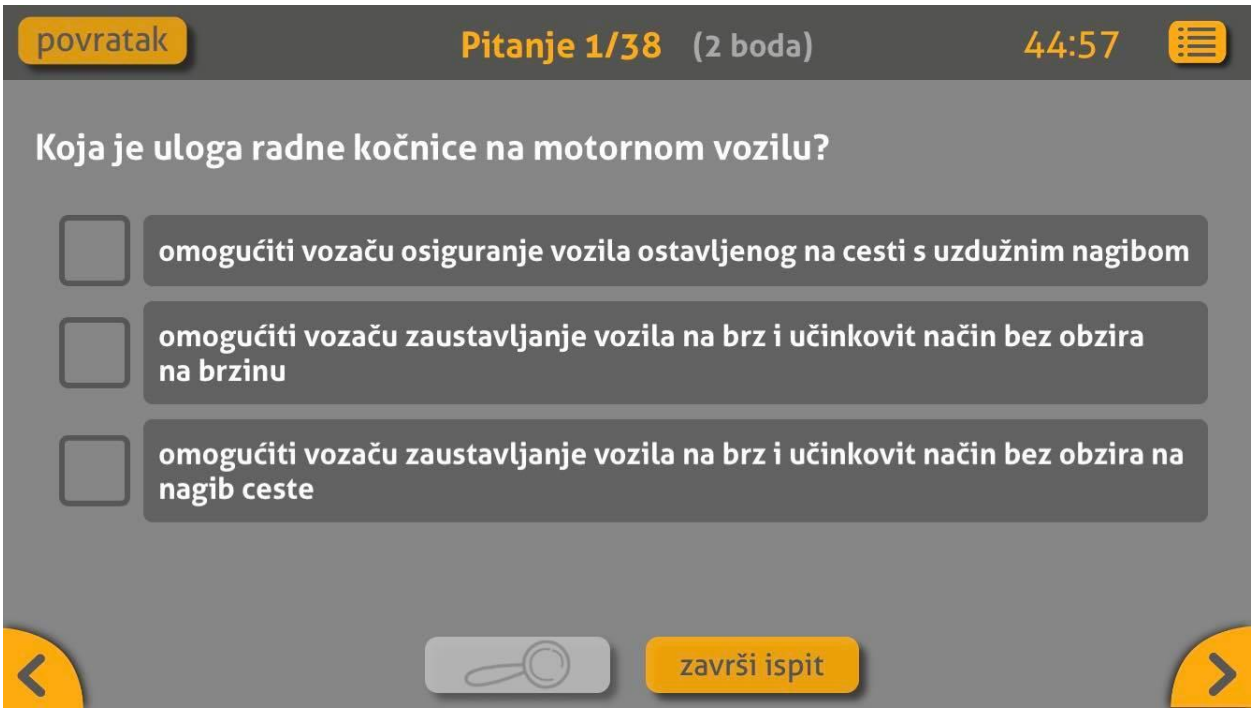
prednost. Kada god dođu promjene u zakonima, knjige se mora zamijeniti jer ne sadrže najnovije podatke. Naravno da i nova proizvodnja knjiga košta, ali isto tako se koristi još više papira pa zbog toga nije ni ekološki bolje rješenje učenje iz knjiga. Kod aplikacija kada dođe do zakonskih promjena, ne treba proizvoditi novu aplikaciju nego ju je dovoljno ažurirati, ako je uopće potrebno. Korisnik ima i puno više mogućnosti od aplikacije. Osim što može učiti na aplikaciji, može koristiti i druge funkcionalnosti aplikacije koje nisu dostupne s knjigama.

Na hrvatskom tržištu ima jako malo aplikacija za učenje prometnih propisa i prve pomoći. Aplikacije koje su ponuđene su uglavnom vrlo jednostavne izrade i nemaju puno funkcionalnosti, te su im pitanja u ispitima unaprijed određena. Od postojećih aplikacija se može izdvojiti „AUTOŠKOLA“ kao jedna od najkvalitetnijih aplikacija na tržištu. Dostupna je na android i IOS uređajima.

Slika 10. Glavni izbornik aplikacije „AUTOŠKOLA“

U aplikaciji „AUTOŠKOLA“ korisnici imaju veliku bazu pitanja na raspolaganju za učenje gradiva. Koristi 3D simulacije koje su najveća prednost ove aplikacije jer animiraju korisnika u virtualnom prostoru prikazujući prometne situacije u kojima bi se mogao naći.

Ispiti se generiraju iz baze podataka tako da korisnici neće naletjeti na ispit koji su prethodno rješavali.

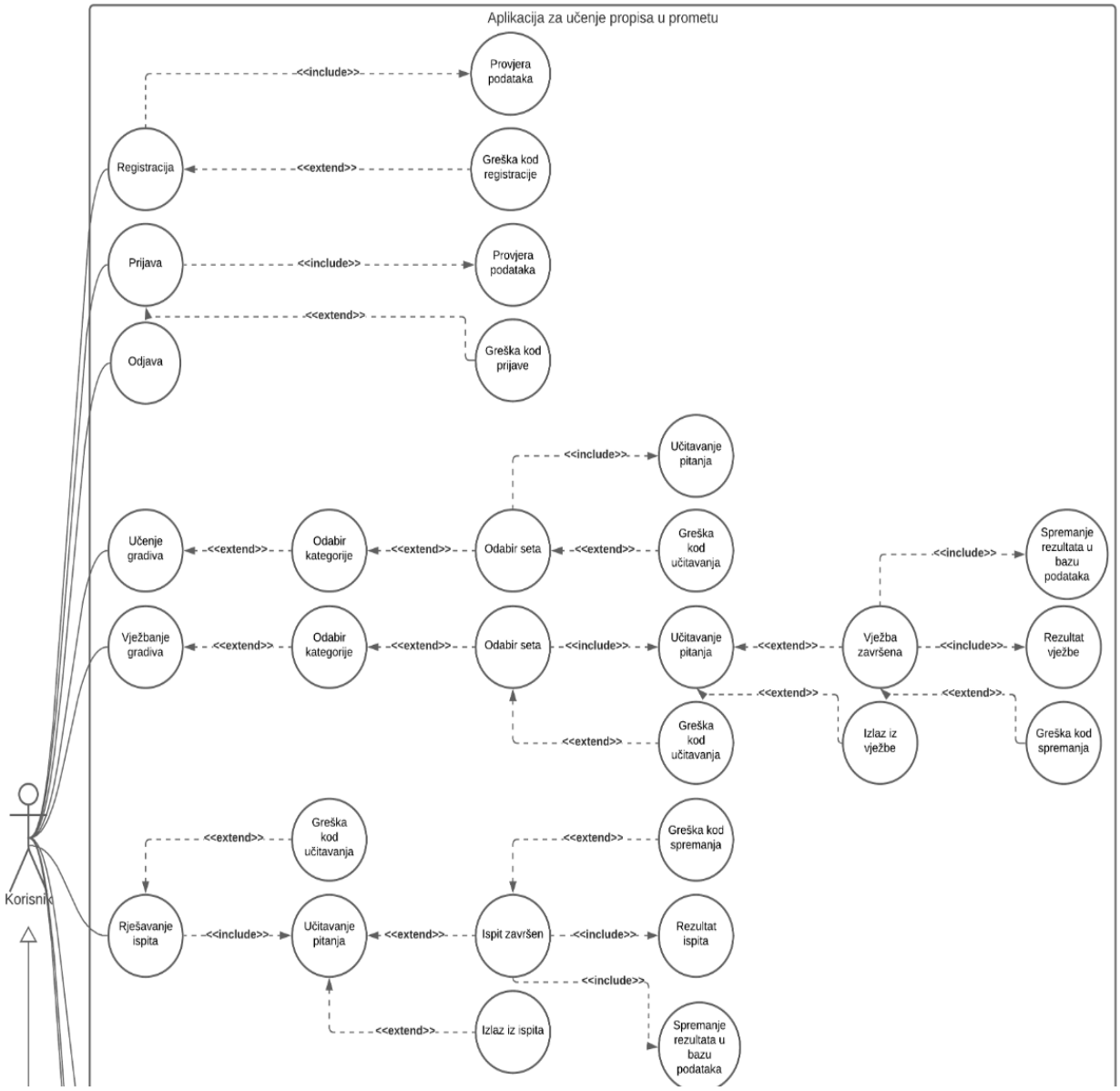


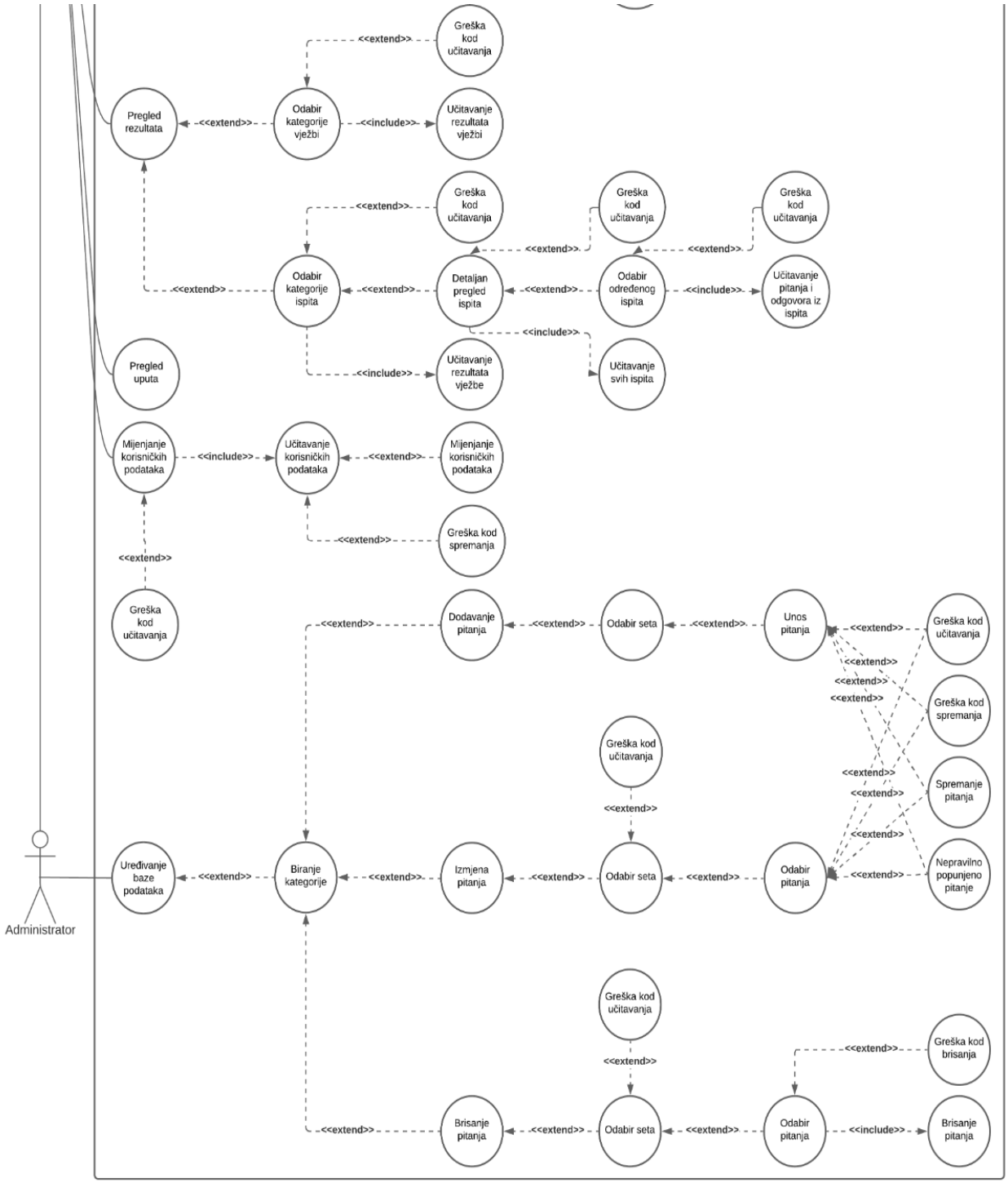
Slika 11. Prikaz ispita aplikacije „AUTOŠKOLA“

6. Implementacija aplikacije za učenje propisa u prometu

Prethodno navedene tehnologije su korištene za implementaciju aplikacije za učenje propisa u prometu. Rezultat korištenja navedenih tehnologija je mobilna aplikacija koja se koristi od strane korisnika, ali i od strane administratora koji može koristiti aplikaciju za ažuriranje baze pitanja.

6.1 UML dijagram slučajeva korištenja (eng. Use Case) i sekvencijalni(eng. Sequence) dijagrami





Slika 12. *Use case* dijagram aplikacije

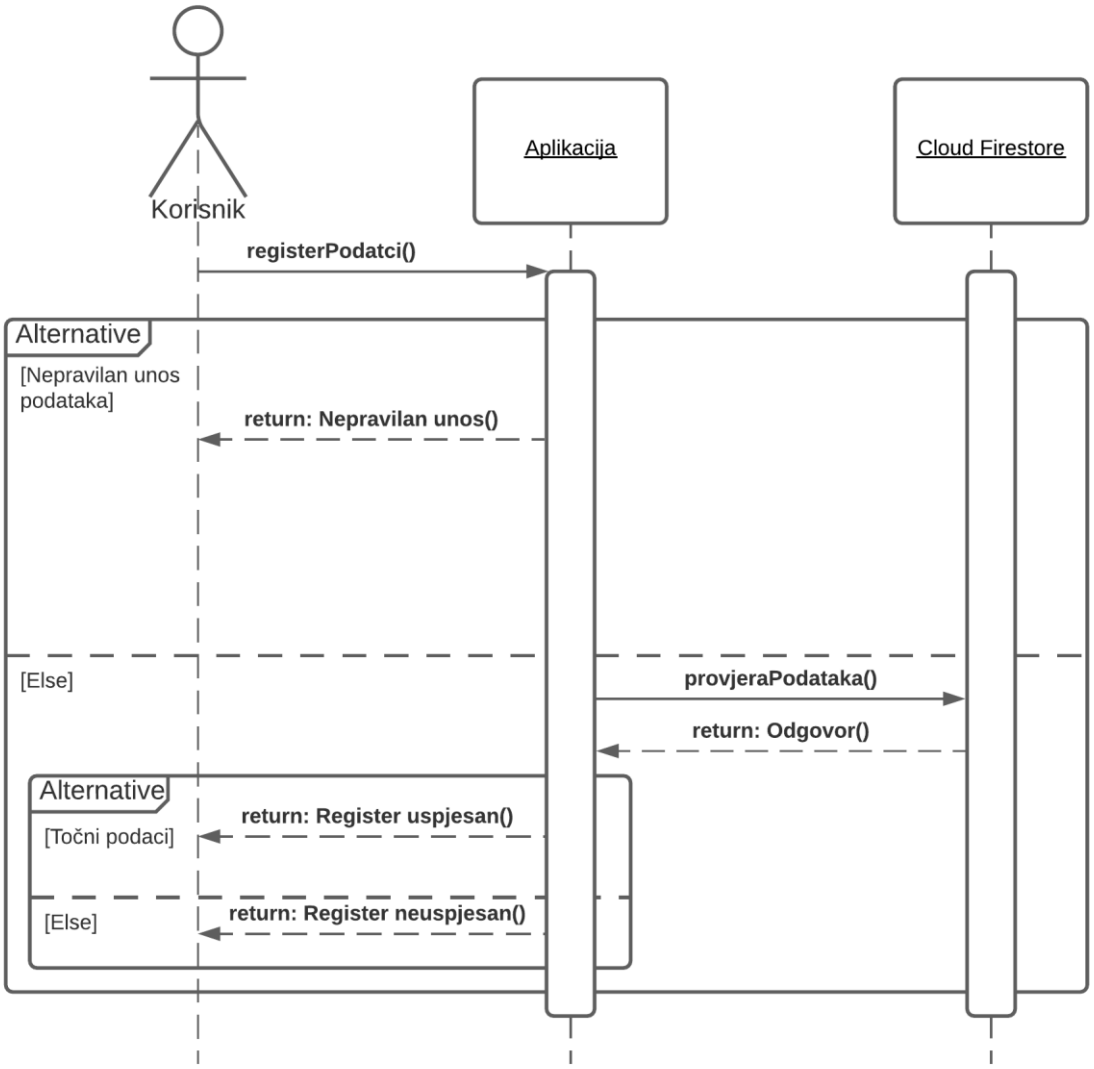
Slika 12 prikazuje *Use case*[17] dijagram aplikacije gdje je vidljivo da aplikaciju koriste korisnici i administrator koji je i korisnik. Pošto su korisnici i administrator sudionici

[17] Use case dijagram je najjednostavniji prikaz korisnikove interakcije sa sustavom koji prikazuje odnos između korisnika i različitih slučajeva uporabe u kojima je korisnik uključen.

oni se nalaze van granice sustava. Granice sustava su označene pravokutnikom izvan kojega se nalaze sudionici. Korisnik ima slučajeve korištenja koji su:

- Registracija – Korisnik kreira svoj korisnički račun
- Prijava – Korisnik se prijavljuje na postojeći korisnički račun
- Odjava – Korisnik se odjavljuje s korisničkog računa
- Učenje gradiva – Korisnik bira kategorije i setove gradiva koje želi učiti
- Vježbanje gradiva – Korisnik bira kategorije i setove pitanja koje želi vježbati
- Rješavanje ispita – Korisnik rješava ispit koji nasumično izvlači pitanja iz određenih kategorija i setova
- Pregled rezultata – Korisnik pregledava rezultate riješenih vježbi i ispita
- Pregled uputa – Korisnik pregledava upute za korištenje aplikacije
- Mijenjanje korisničkih podataka – Korisnik mijenja svoje ime, prezime i e-mail

Administrator uz slučajeve korištenja koje ima korisnik ima slučaj korištenja koji je uređivanje baze podataka. To znači da može dodavati, izmjenjivati i brisati pitanja iz određenih kategorija i setova pitanja.

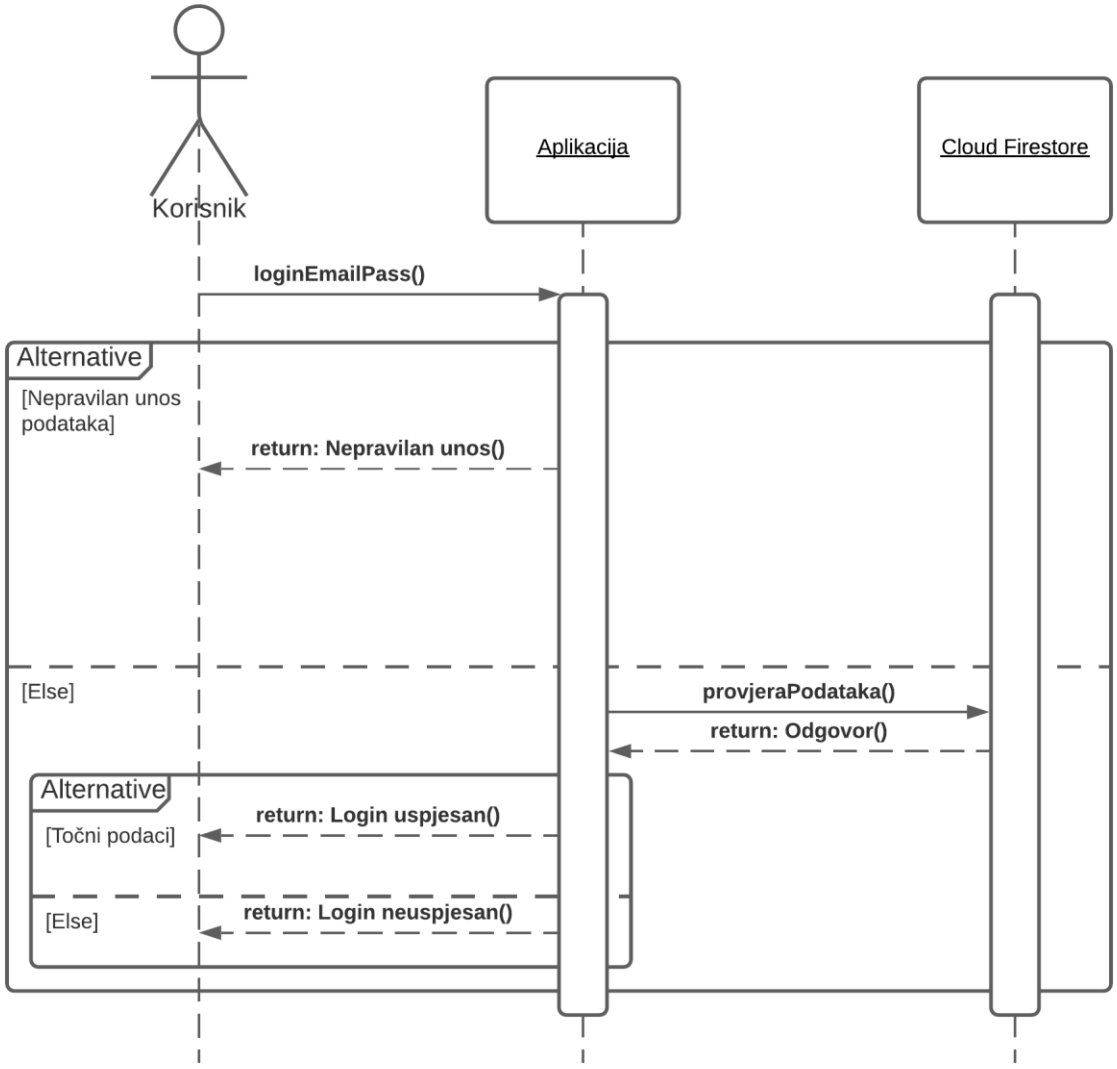


Slika 13. *Sequence* dijagram registracije

U *sequence*[18] dijagramu sa slike 13 korisnik komunicira s aplikacijom tako da upisuje svoje ime i prezime, e-mail, lozinku i potvrdu lozinke. Te podatke zatim šalje aplikaciji koja provjerava dali je svaki podatak pravilno unesen. Ako podatci nisu pravilno uneseni aplikacija šalje korisniku obavijest da nije dobro popunio podatke, a ako su podatci ispravni onda se šalju drugom objektu koji je *Cloud Firestore*. Ako se podatci unesu, *Cloud Firestore* šalje aplikaciji odgovor da je korisnik registriran, a aplikacija to prikaže

[18] Sequence dijagrami su dijagrami interakcije koji detaljno opisuju kako se operacije provode.

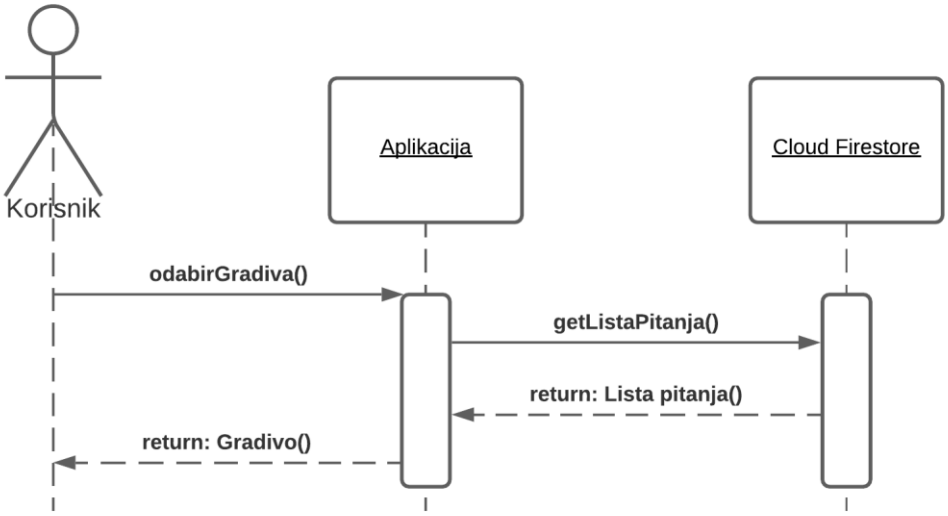
korisniku, a ako se ne unesu, *Cloud Firestore* šalje odgovor aplikaciji zašto korisnik nije registriran, a aplikacija to prikazuje korisniku.



Slika 14. *Sequence* dijagram prijave

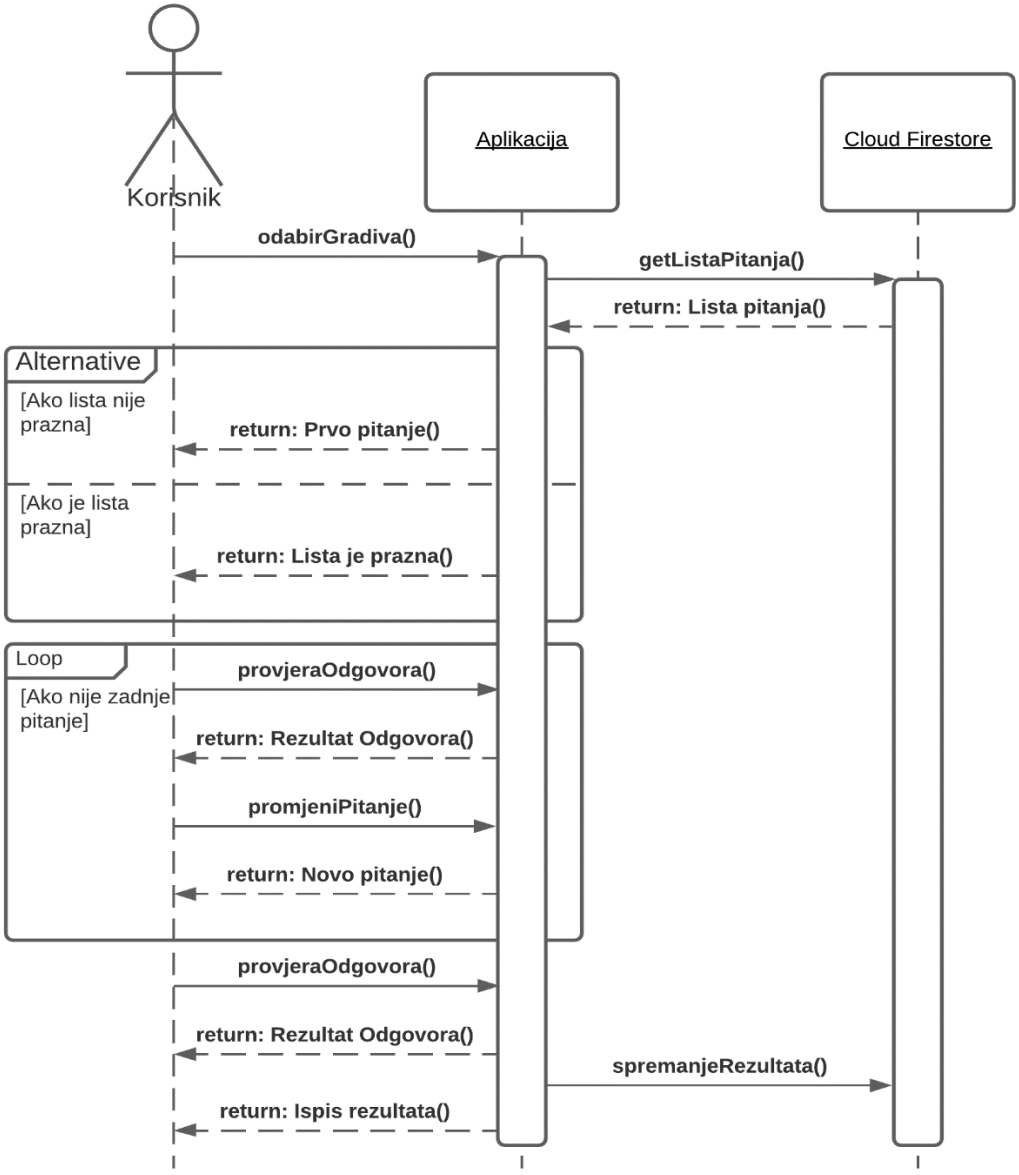
Slika 14 sadrži *Sequence* dijagram koji prikazuje kako korisnik komunicira s aplikacijom prilikom prijave u aplikaciju. Korisnik upisuje svoj e-mail i lozinku. TI podatci se provjeravaju na aplikaciji, te ako su neispravni, aplikacija šalje obavijest da podatci nisu dobro uneseni. Ako su podatci dobro uneseni, aplikacija ih šalje na *Cloud Firestore* gdje se provjeravaju. *Cloud Firestore* šalje aplikaciji odgovor. Ovisno o odgovoru kojeg je

zaprimila aplikacija, određuje se hoće li se korisnik prijaviti ili će dobiti obavijest da je upisao netočne podatke.



Slika 15. *Sequence* dijagram teorije

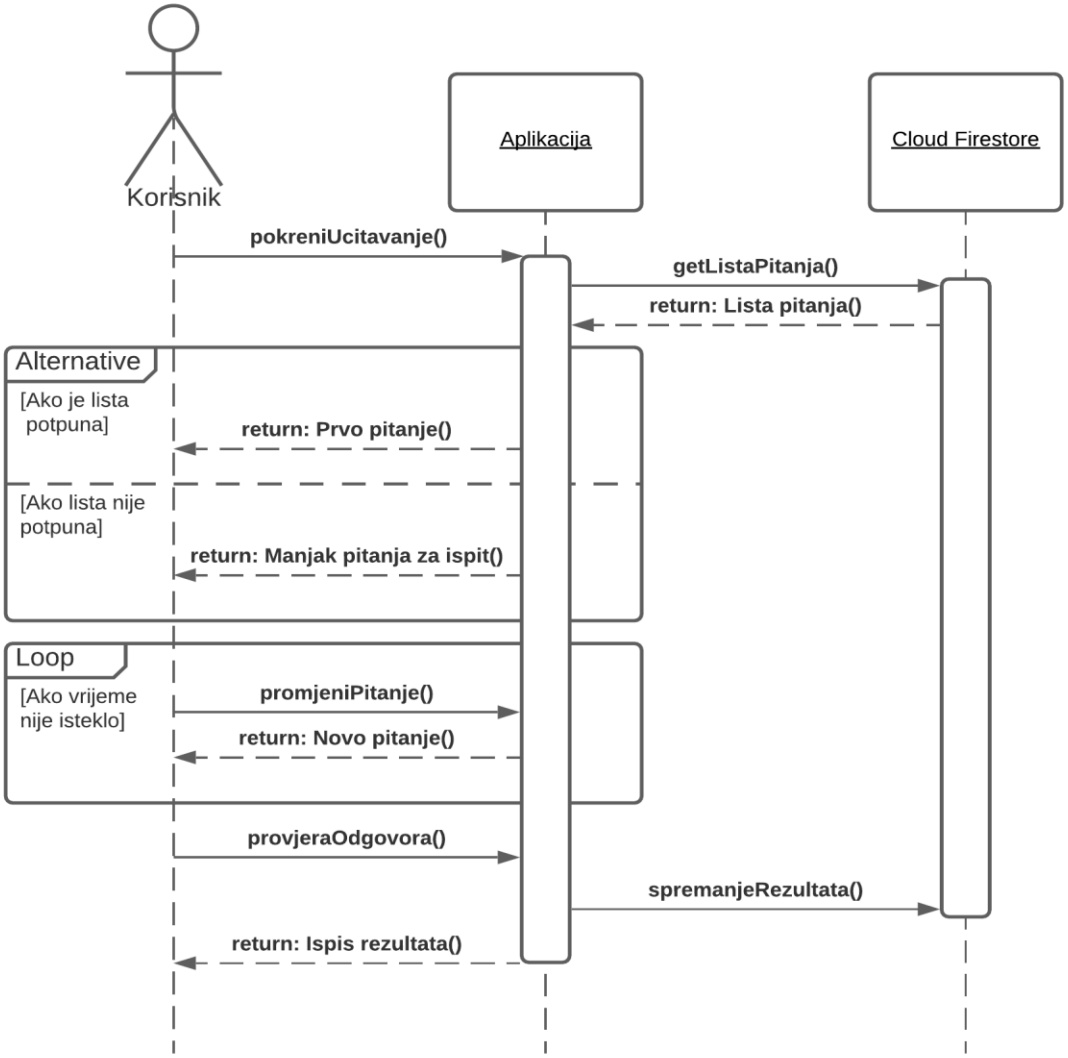
Na slici 15 *Sequence* dijagram prikazuje kako korisnik pomoću aplikacije i *Cloud Firestore*-a dolazi do gradiva koje želi pregledati. Korisnik prvo bira kategoriju, a zatim set gradiva koji želi pregledati. Kada sve odabere, aplikacija šalje upit na *Cloud Firestore*, a *Cloud Firestore* vraća kolekciju pitanja traženog gradiva. Aplikacija tada prikazuje traženo gradivo korisniku.



Slika 16. *Sequence* dijagram vježbe

Sequence dijagram sa slike 16 prikazuje kako korisnik rješava vježbe u aplikaciji. Korisnik prvo bira kategoriju i set gradiva. Aplikacija radi upit na temelju označenog gradiva i šalje ga na *Cloud Firestore*. *Cloud Firestore* šalje listu pitanja aplikaciji te ako lista pitanja nije prazna, aplikacija postavlja prvo pitanje korisniku. U slučaju da je lista prazna, aplikacija prikazuje obavijest korisniku da u bazi nema pitanja za to gradivo. Kada trenutno pitanje nije zadnje pitanje u listi pitanja, korisnik odgovara na pitanje i aplikacija daje odgovor na pitanje. Zatim korisnik traži promjenu pitanja, a aplikacija mu postavlja sljedeće pitanje iz

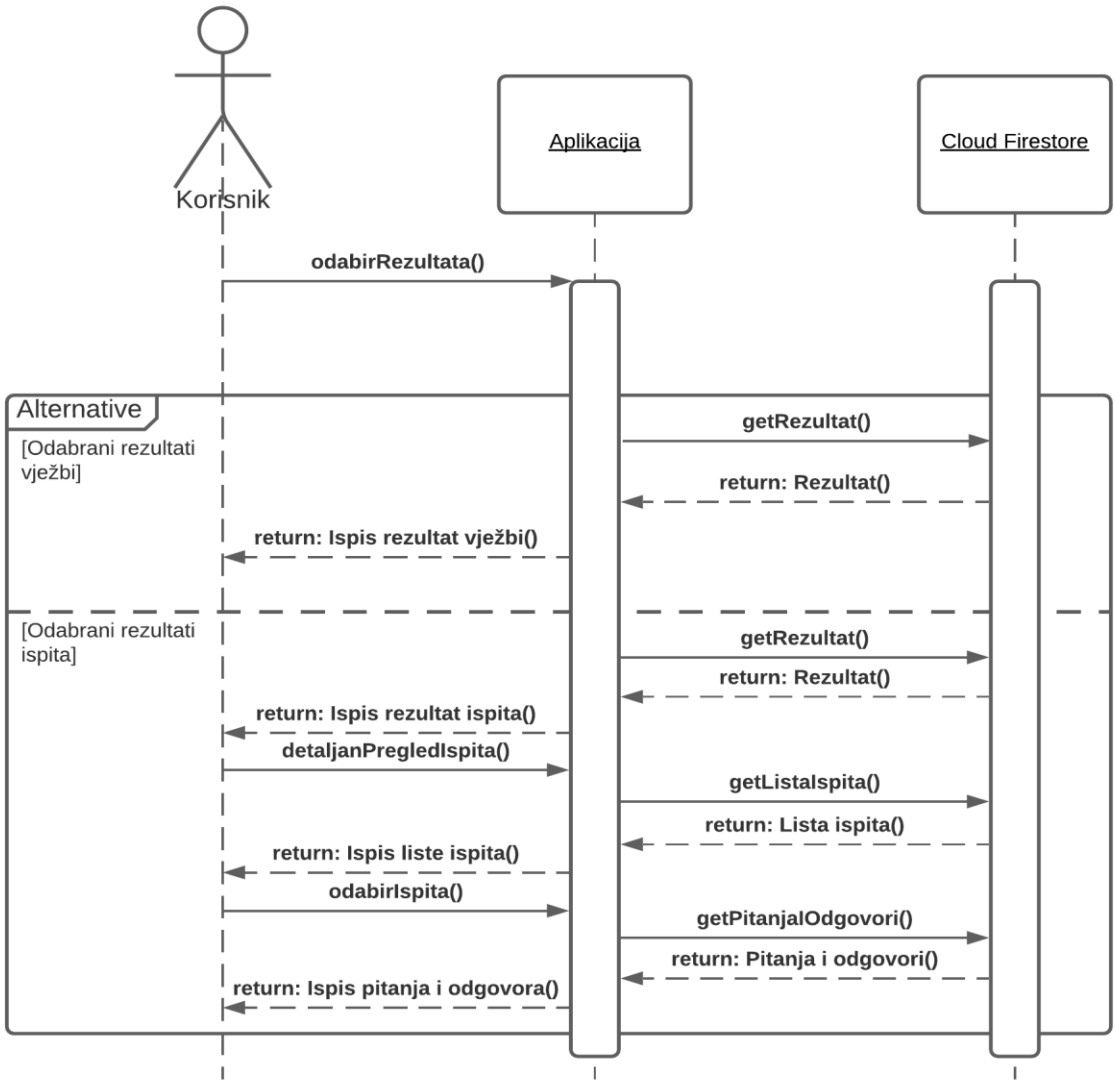
liste. Kada korisnik dođe do zadnjeg pitanja, odgovara na pitanje i aplikacija mu vraća odgovor na pitanje te sprema rezultat riješene vježbe u Cloud Firestore. Potom aplikacija ispisuje rezultat riješene vježbe.



Slika 17. *Sequence* dijagram ispita

Slika 17 prikazuje *sequence* dijagram ispita u kojemu korisnik rješava ispit u aplikaciji. Korisnik kada odabere rješavanje ispita, aplikacija šalje upit *Cloud Firestore*-u. Upit se kreira od nasumično odabranih pitanja iz određenih setova propisa. *Cloud Firestore* šalje listu s pitanjima aplikaciji. Aplikacija provjerava dali ima dovoljan broj pitanja u listi tj. dali je lista potpuna. Ako je lista pitanja potpuna, aplikacija postavlja prvo pitanje iz liste, te

ako lista nije potpuna, aplikacija prikazuje obavijest da se ispit ne može rješavati zbog manjka pitanja. Korisnik može prolaziti kroz sva pitanja više puta i odgovarati na njih, te im mijenjati odgovore dok god ima vremena. Vrijeme je određeno i započinje nakon što se prvo pitanje postavi. Kada vrijeme istekne ili korisnik sam završi ispit, aplikacija provjerava odgovore na pitanja. Kada provjeri pitanja, sprema rezultat riješenog ispita na *Cloud Firestore*. Aplikacija zatim ispiše korisniku rezultate riješenog ispita.

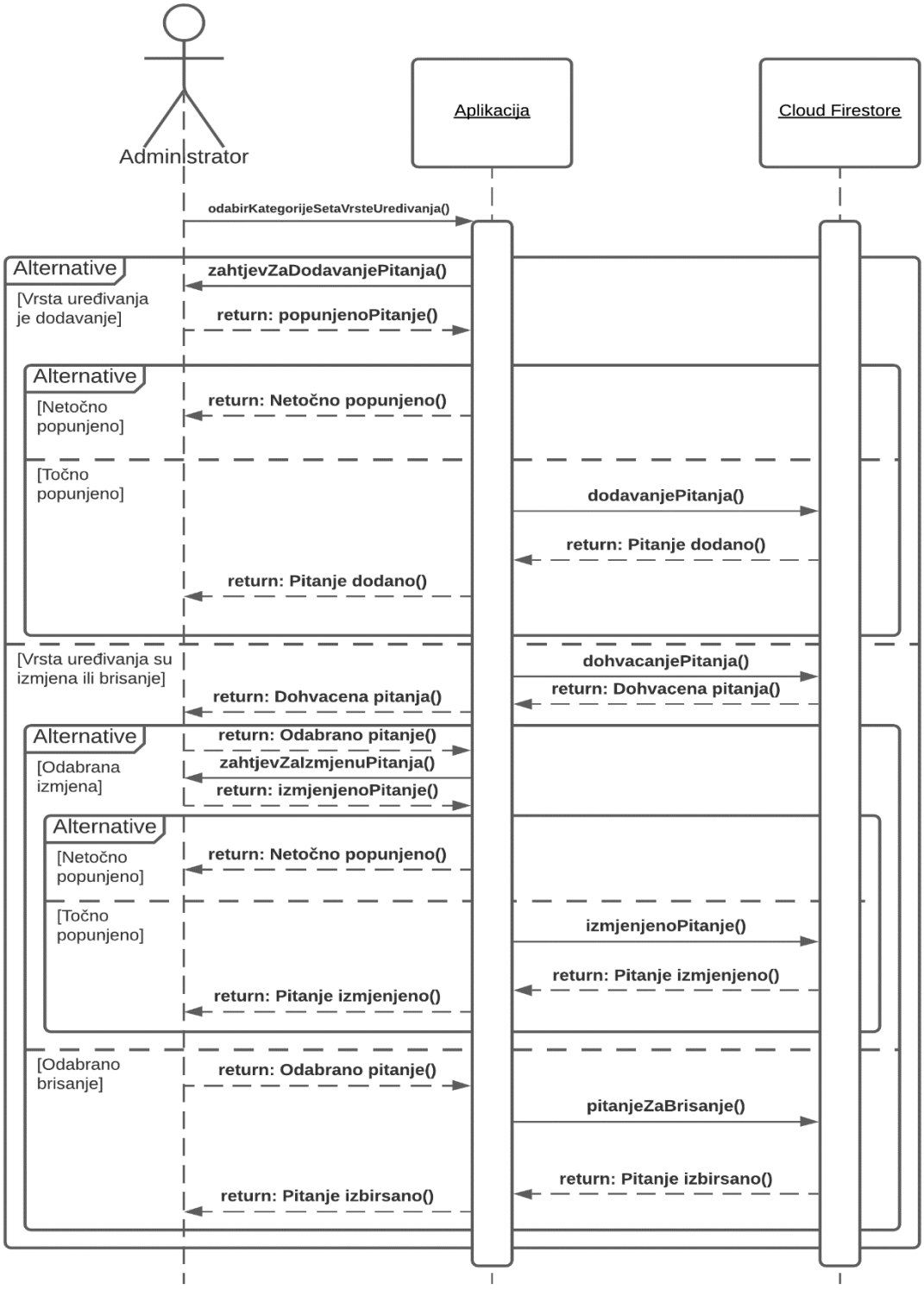


Slika 18. *Sequence* dijagram rezultata

Na *sequence* dijagramu sa slike 18 se prikazuje kako korisnik pregledava rezultate riješenih vježbi i ispita. Korisnik može birati koje rezultate želi vidjeti. Ako odabere

rezultate vježbi, aplikacija šalje upit na *Cloud Firestore* od kojeg traži rezultate određene vježbe. Kada dobije te rezultate, ispisuje ih korisniku. Ako korisnik odabere rezultate ispita, aplikacija šalje upit na *Cloud Firestore*. *Cloud Firestore* vraća rezultate ispita aplikaciji te ih aplikacija ispisuje korisniku. Kada korisnik odabere detaljan pregled ispita, aplikacija šalje upit na *Cloud Firestore*, a *Cloud Firestore* vraća kolekciju riješenih ispita nazad. Aplikacija ispisuje sve riješene ispite korisniku. Korisnik odabire ispit koji želi vidjeti detaljno, te kada ga odabere, aplikacija šalje upit *Cloud Firestore*-u. *Cloud Firestore* vraća od odabranog ispita kolekciju pitanja iz ispita i odgovore koje je korisnik dao kada je rješavao ispit. Aplikacija ispisuje tu kolekciju korisniku na pregled.

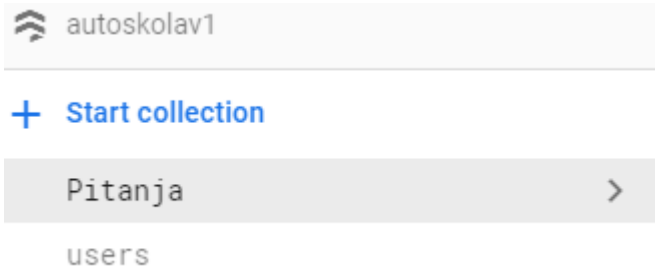
Sequence dijagram sa slike 19 prikazuje kako administrator radi izmjene nad bazom pitanja koja se nalazi na *Cloud Firestore*-u. Administrator odabire kategoriju set i vrstu uređivanja gradiva. Ako je vrsta uređivanja dodavanje novih pitanja, aplikacija korisniku otvara obrazac u koji upisuje novo pitanje. Ako administrator ne popuni pitanje u potpunosti, aplikacija mu šalje obavijest da pitanje nije popunjeno do kraja. Ako je pitanje pravilno popunjeno, aplikacija šalje popunjeno pitanje *Cloud Firestore*-u koji pohranjuje pitanje u već prije određenu kategoriju i set. Aplikacija zatim obavještava administratora da je pitanje dodano. Ako je vrsta uređivanja izmjena ili dodavanje pitanja, aplikacija šalje upit na *Cloud Firestore* koji vraća kolekciju pitanja od prije označene kategorije i seta aplikaciji. Administrator bira pitanje koje želi urediti. Ako je administrator izabrao izmjenu kao vrstu uređivanja, odabrano pitanje aplikacija učitava u obrazac i prikazuje administratoru. Nad pitanjem se mogu vršiti promjene te kada administrator izvrši promjene, aplikacija provjerava dali je pitanje pravilno popunjeno. Ako nije prikazuje obavijest administratoru da pitanje nije dobro popunjeno. Ako je pitanje pravilno popunjeno, aplikacija šalje izmijenjeno pitanje na *Cloud Firestore* gdje se stare vrijednosti pitanja izmjenjuju s novim vrijednostima. Ako je administrator izabrao brisanje za vrstu uređivanja, aplikacija *Cloud Firestore*-u šalje pitanje koje je označeno od strane administratora. *Cloud Firestore* briše pitanje, te aplikacija ispisuje administratoru obavijest da je pitanje izbrisano.



Slika 19. *Sequence* dijagram administratora

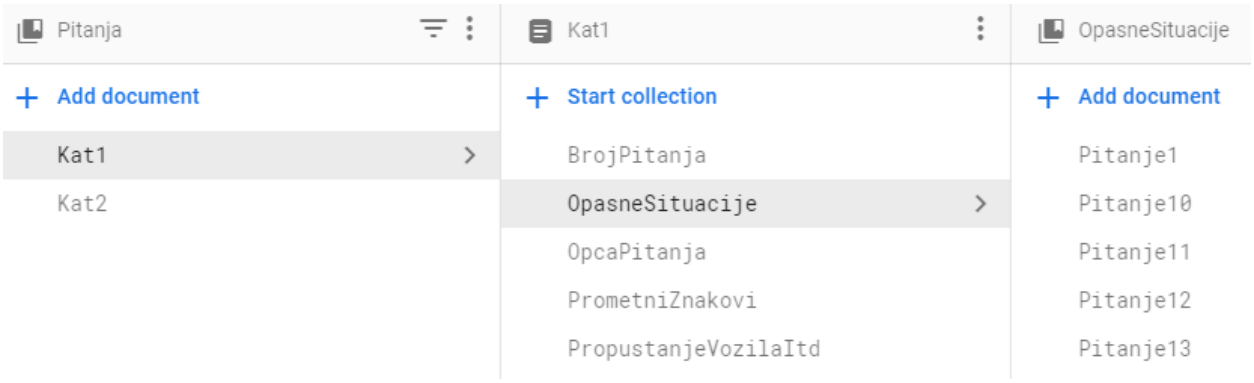
6.2. Izrada baze podataka

Za potrebe izrade baze podataka korišten je *Cloud Firestore*. Kao što je vidljivo na slici 20, podatci su raspoređeni na dvije glavne kolekcije *Pitanja* i *users*. Unutar kolekcije *Pitanja* nalaze se sva pitanja koja se koriste za učenje, vježbanje i ispite. Unutar kolekcije *users* su pohranjeni korisnici i podatci vezani za njihov korisnički račun.



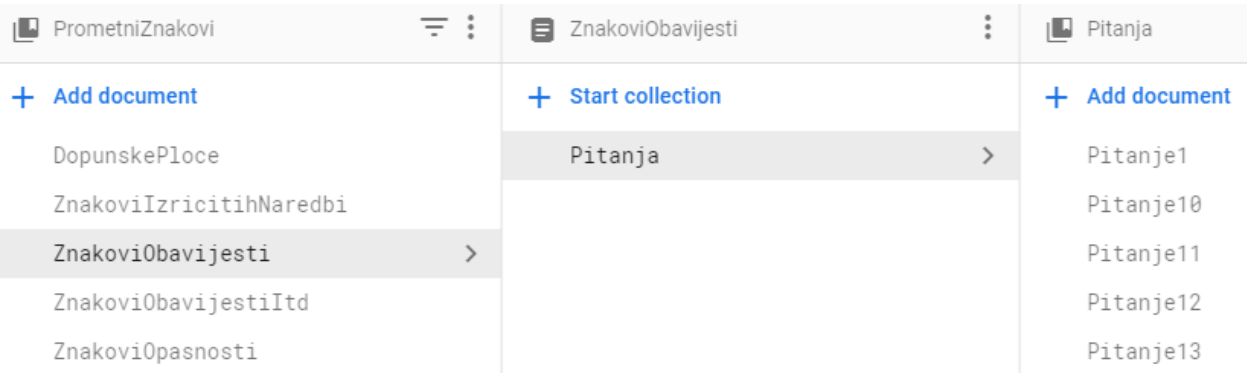
Slika 20. Slika glavnih kolekcija na *Cloud Firestore*-u

Slika 21. prikazuje da unutar kolekcije *Pitanja* se nalaze dokumenti *Kat1* i *Kat2*. *Kat1* je dokument unutar kojeg su spremljena pitanja za propise, a unutar *Kat2* se nalaze pitanja za prvu pomoć.



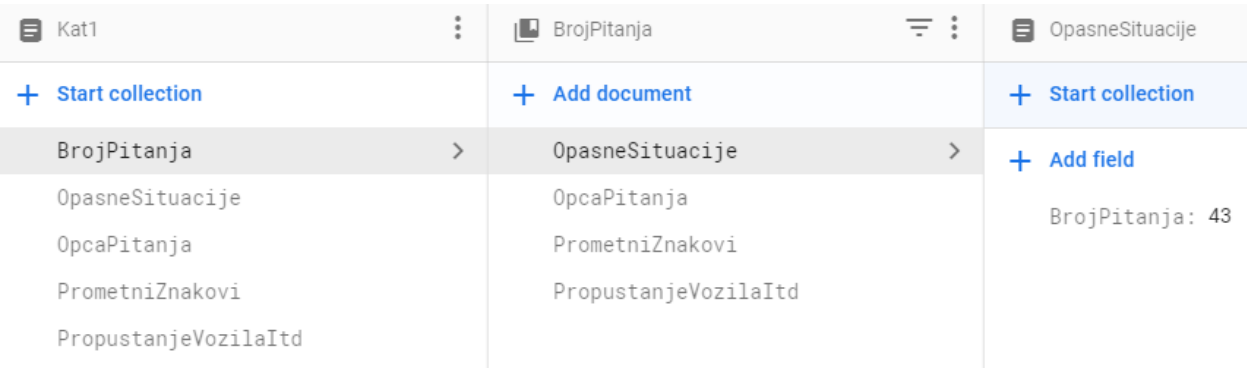
Slika 21. Struktura kolekcije *Pitanja*

Kat1 i *Kat2* sadrže kolekcije koji se mogu promatrati kao setovi unutar kojih su raspoređena pitanja. *Kat1* ima 4 seta pitanja, dok *Kat2* ima 3 seta pitanja. Na slici 22 se može vidjeti da kod *Kat1* se nalazi poseban set koji sadrži grupu podsetova, a taj set je *PrometniZnakovi*. *PrometniZnakovi* je set koji sadrži 5 podsetova koji su također popunjeni s raspoređenim pitanjima.



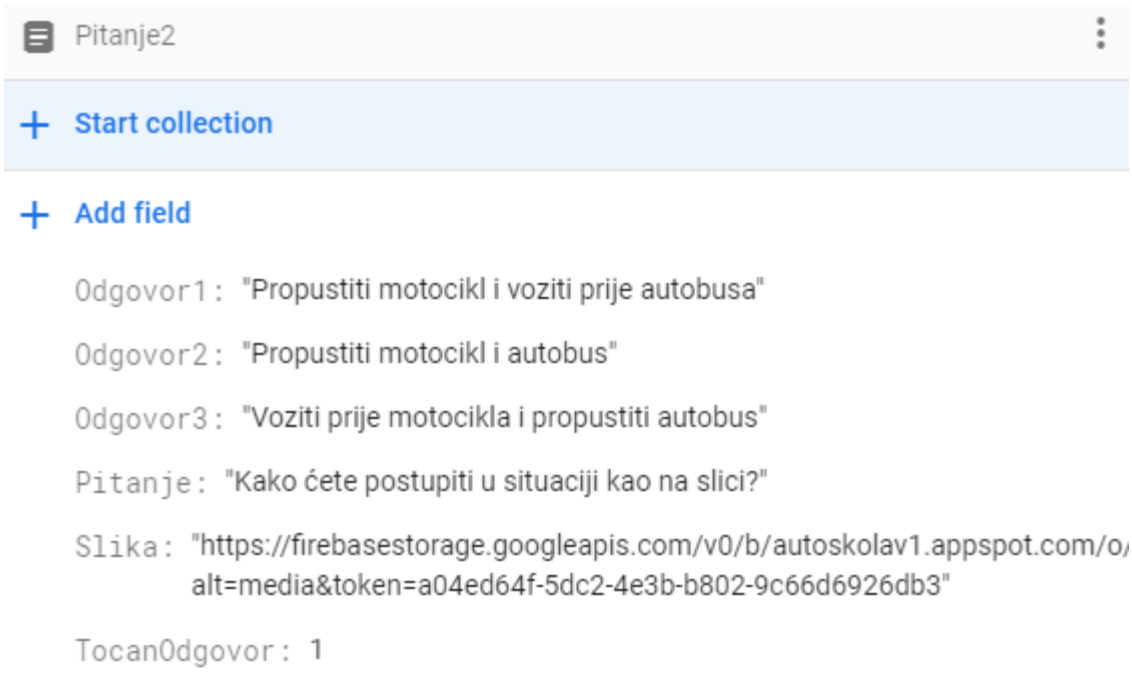
Slika 22. Struktura kolekcije *PrometniZnakovi*

Cloud Firestore ima ograničeni dnevni broj besplatnih čitanja, pisanja i brisanja nad podatcima, te zbog toga *Kat1* i *Kat2* sadrže dodatnu kolekciju *BrojPitanja* koja se koristi za optimizaciju i vidljiva je na slici 23. Kolekcija *BrojPitanja* sadrži setove koji se nalaze unutar *Kat1* i *Kat2*, te za svaki set sadrži broj pitanja koji naznačuje koliko svaki set unutar *Kat1* i *Kat2* ima pitanja.



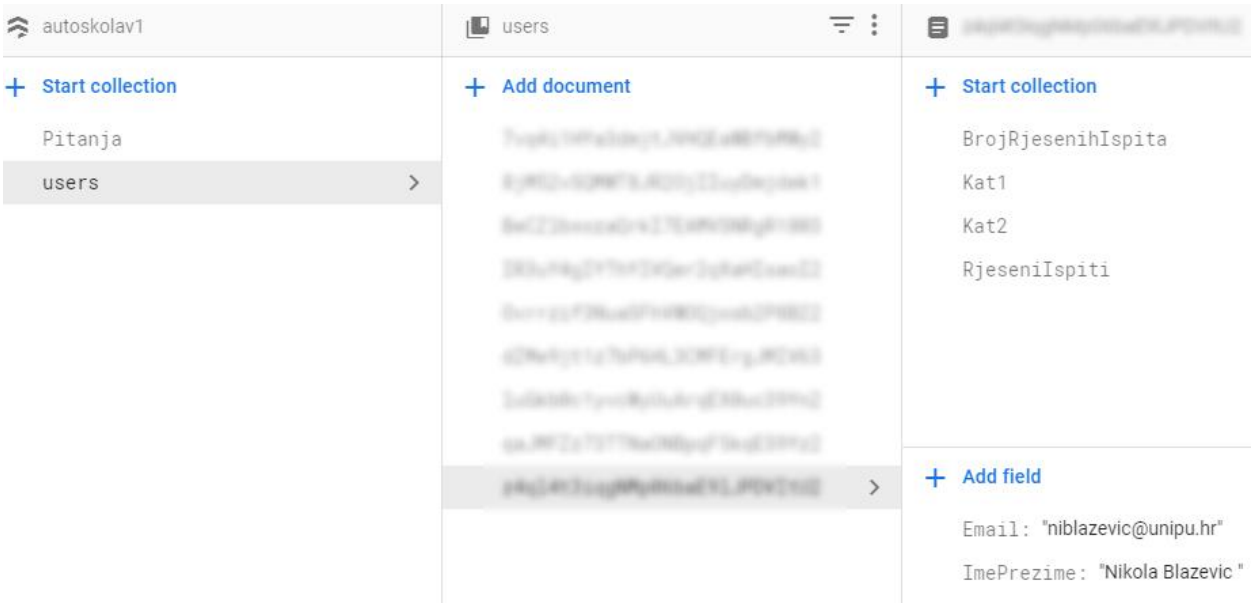
Slika 23. Struktura kolekcije *BrojPitanja*

Svi setovi sadrže dokumente pitanja koja imaju istu strukturu, a struktura je vidljiva na slici 24. Svaki dokument pitanja sadrži polje *Pitanje* koje sadrži tekst pitanja, polja *Odgovor1, Odgovor2, Odgovor3* koji sadrže tekst odgovora, polje *TocanOdgovor* koji sadrži broj jednog ili više točnih odgovora, te polje *Slika* koje sadrži vrijednost pristupnog tokena tj. linka od slike koja je određena za to pitanje. *Slika* je jedino polje unutar dokumenta pitanja koje može biti prazno, ali bez obzira na to što može biti prazno, mora postojati kao polje.



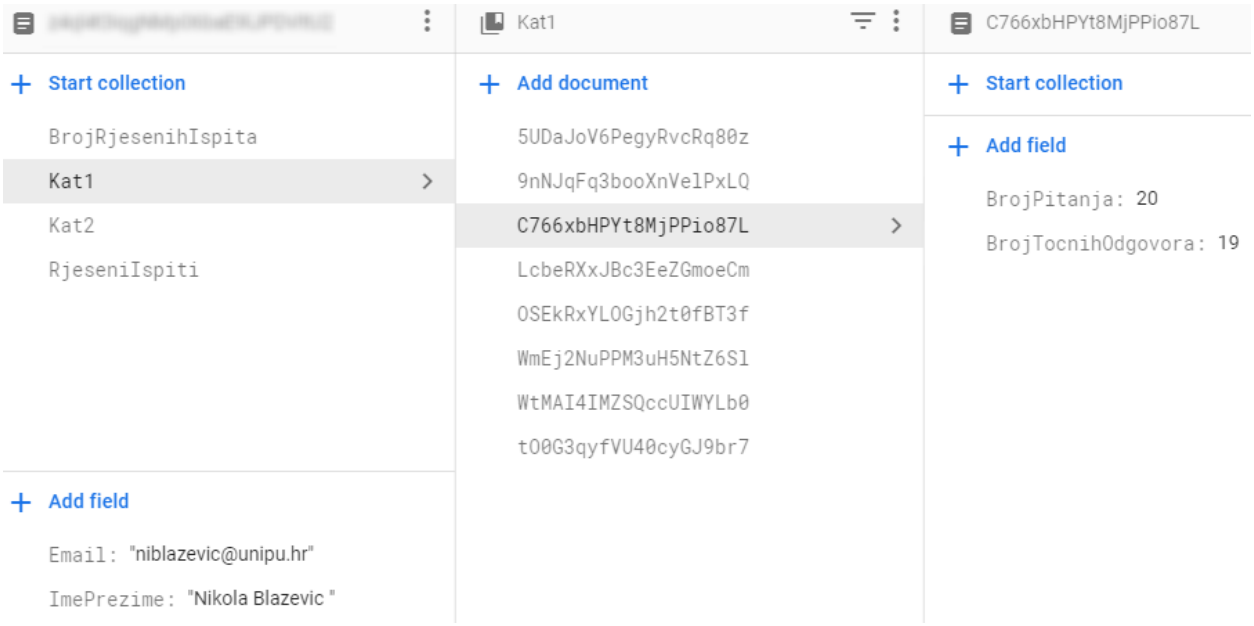
Slika 24. Struktura pitanja unutar setova

Na slici 25 je vidljiva struktura kolekcije *users*. Unutar te kolekcije se nalaze registrirani korisnici. Podatci o korisniku se nalaze unutar jednog dokumenta koji ima naziv korisnikovog UID-a.



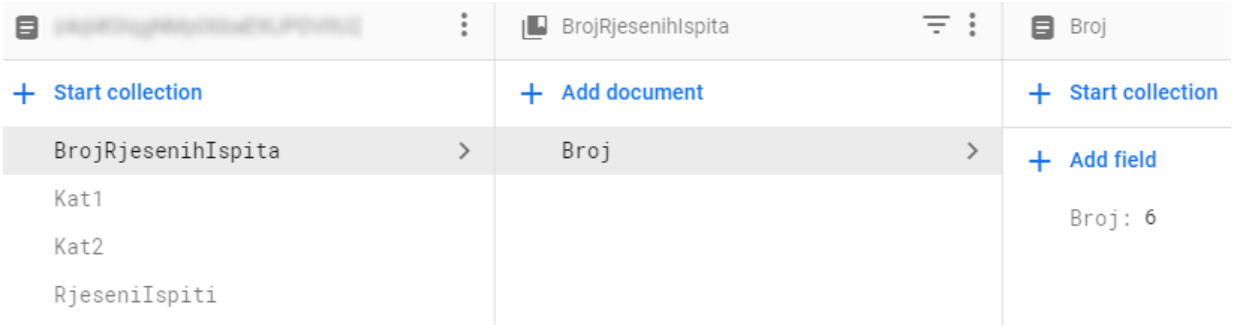
Slika 25. Struktura kolekcije *users*

Slika 26 prikazuje strukturu kolekcija *Kat1* i *Kat2* koje se kreiraju kada korisnik riješi vježbu po prvi put. Podatci o riješenim vježbama se nalaze unutar navedenih kolekcija. Svaka riješena vježba se sprema unutar odgovarajuće kategorije kao novi dokument s nasumično generiranim ID-om, te sadrži polje *BrojPitanja* koje označuje sveukupan broj pitanja iz riješene vježbe i polje *BrojTocnihOdgovora* koje označuje broj točno odgovorenih pitanja u riješenoj vježbi.



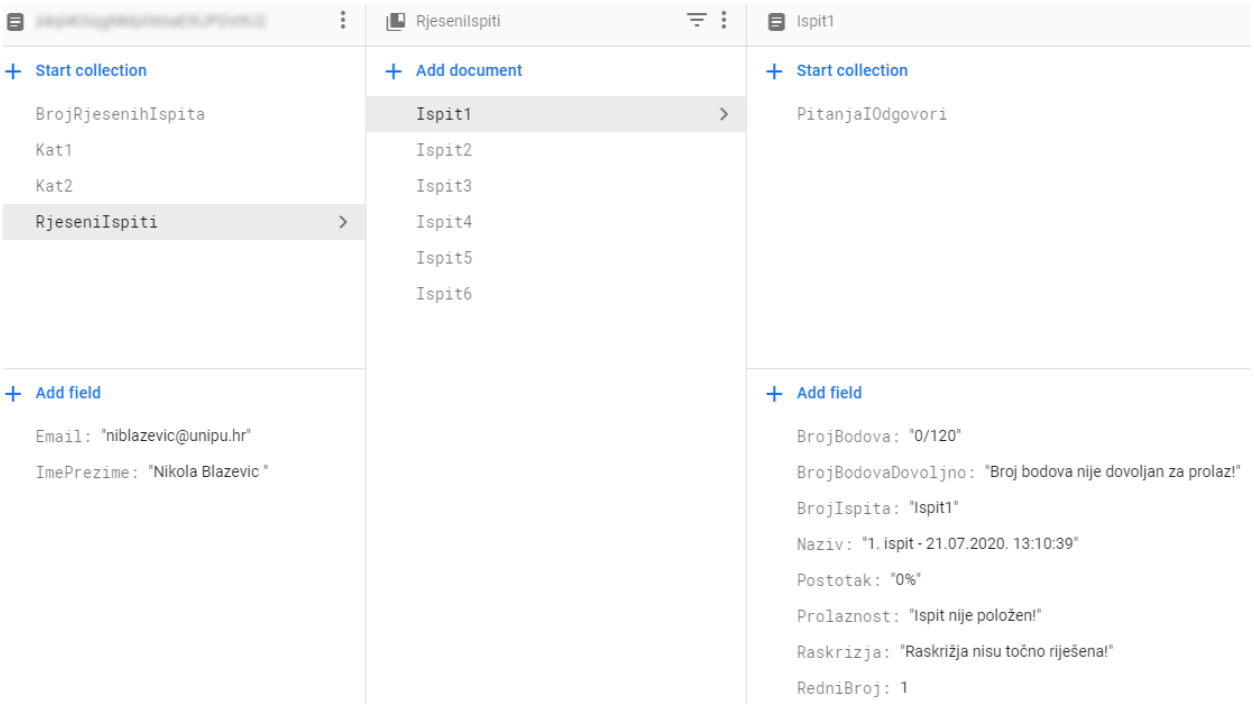
Slika 26. Struktura kolekcija *Kat1* i *Kat2*

Kada korisnik riješi ispit, kreira se kolekcija *BrojRjesenihIspita* kao što je vidljivo na slici 27. Unutar te kolekcije se kreira dokument pod nazivom *Broj*. Dokument *Broj* sadrži polje koje se također zove *Broj,* te se unutar tog polja ažurira broj riješenih ispita koje je korisnik riješio do sada.



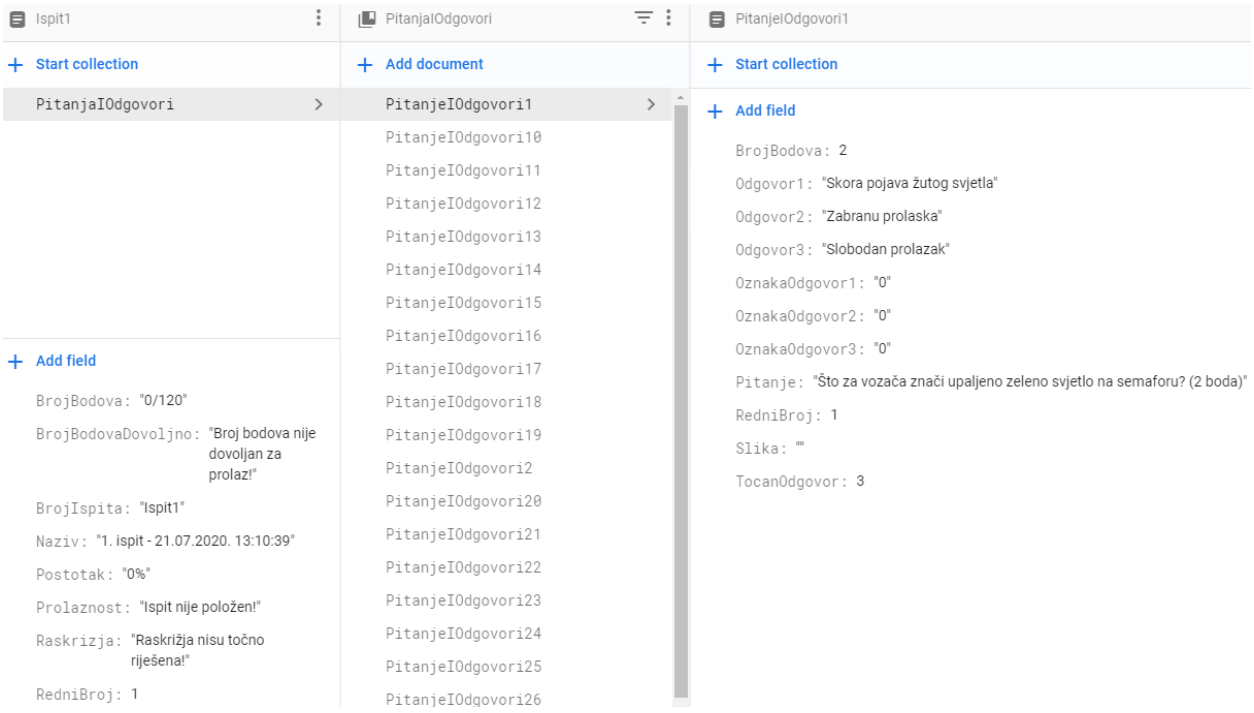
Slika 27. Struktura kolekcije *BrojRjesenihPitanja*

Nakon što korisnik riješi ispit i kreira se kolekcija *BrojRjesenihIspita*, također se kreira i kolekcija *RjeseniIspiti* čija se struktura može vidjeti na slici 28. Kada se riješi ispit kreira se dokument pod nazivom *Ispit* s rednim brojem ispita. Redni broj ispita se nabavlja iz kolekcije *BrojRjesenihIspita*. Svaki dokument ispita ima istu strukturu te sadrži polja koja opisuju kako je ispit riješen. Osim polja, svaki dokument ispita sadrži i kolekciju *PitanjaIOdgovori*.



Slika 28. Struktura kolekcije *RjeseniIspiti*

Na slici 29 je vidljiva struktura dokumenta *Ispit*. Kao i prije navedeno, dokument *Ispit* sadrži kolekciju *PitanjaIOdgovori.* Unutar te kolekcije se nalaze sva pitanja i odgovori s riješenog ispita. Pitanje i odgovor su spremljeni kao jedan dokument pod nazivom *PitanjeIOdgovor* s rednim brojem pitanja. Unutar dokumenta *PitanjeIOdgovor* se nalaze sva polja koja se nalaze unutar dokumenta *Pitanje* koje se nalazi u kategorijama i setovima u kolekciji *Pitanja*. Uz polja iz dokumenta *Pitanje,* dokument *PitanjeIOdgovor* također sadrži polja *OznakaOdgovor1, OznakaOdgovor2* i *OznakaOdgovor3* koja služe kako bi se znalo koje je odgovore korisnik označio na pitanje iz ispita.



Slika 29. Struktura dokumenta *Ispit*

6.3. Izrada mobilne aplikacije

Mobilna aplikacija se razvijala paralelno s bazom podataka. Sastoji se od 31 klase i 25 aktivnosti. Za potrebe izgleda i funkcionalnosti aplikacije su korištene ovisnosti navedene na slici 30.

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation("com.squareup.okhttp3:okhttp:4.8.0")
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    implementation 'com.google.firebase:firebase-database:19.3.1'
    implementation 'com.google.firebase:firebase-auth:19.3.2'
    implementation 'com.google.firebase:firebase-firestore:21.5.0'
    implementation 'com.firebaseui:firebase-ui-firestore:6.2.1'
    implementation 'androidx.legacy:legacy-support-v4:1.0.0'
    implementation 'com.google.android.material:material:1.1.0'
    implementation 'androidx.navigation:navigation-fragment:2.3.0'
    implementation 'androidx.navigation:navigation-ui:2.3.0'
    implementation 'androidx.lifecycle:lifecycle-extensions:2.2.0'
    implementation 'com.squareup.picasso:picasso:2.71828'
    implementation 'com.google.firebase:firebase-storage:19.1.1'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
}
```

Slika 30. *build.gradle* ovisnosti

6.3.1. SplashScreen aplikacije

Slika 31. *SplashScreen* aplikacije

Kada korisnik pokrene aplikaciju, prvo što se otvara je animirani *SplashScreen* zaslon u trajanju od 3 sekunde koji je vidljiv na slici 31. Na njemu se nalazi logo aplikacije koji se s vrha spušta prema sredini, te dva teksta s porukom koji idu od dna aplikacije prema sredini.

Slika 32 prikazuje programski rješenje prethodno opisane aktivnosti. Kada se aplikacija pokrene, pokušava se spojiti na bazu podataka kako bi dohvatila prijavljenog korisnika. Aplikacija učitava animacije i dodjeljuje ih. Kada animacija u trajanju od 3000 milisekundi završi, aktivnost se mijenja. Ako postoji prijavljeni korisnik, aplikacija pokreće novu aktivnost koja će odvesti korisnika u glavni izbornik aplikacije. U slučaju da nije postojao prijavljen korisnik, aplikacija otvara novu aktivnost gdje se korisnik može prijaviti u sustav. Kod otvaranja nove aktivnosti koristi se funkcija *makeSceneTransitionAnimation()* koja služi za kreiranje glatke animacije između aktivnosti gdje se slika i dva teksta skaliraju i mijenjaju poziciju na ekranu. Nova aktivnost se pokreće funkcijom *startActivity().*

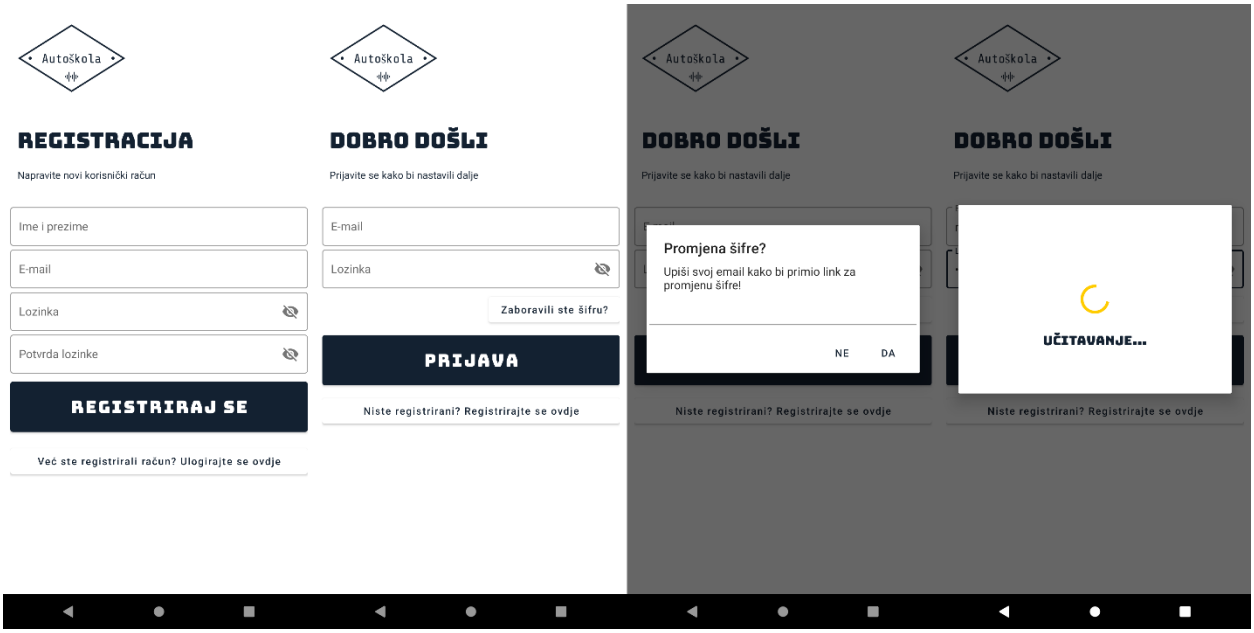
```
        fAuth = FirebaseAuth.getInstance();

        topAnim = AnimationUtils.loadAnimation( context: this, R.anim.top_animation);
        bottomAnim = AnimationUtils.loadAnimation( context: this, R.anim.bottom_animation);

        imgLogoSplashScreen = findViewById(R.id.imgLogoSplashScreen);
        txtVelikiSplashScreen = findViewById(R.id.txtVelikiSplashScreen);
        txtMaliSplashScreen = findViewById(R.id.txtMaliSplashScreen);

        imgLogoSplashScreen.setAnimation(topAnim);
        txtVelikiSplashScreen.setAnimation(bottomAnim);
        txtMaliSplashScreen.setAnimation(bottomAnim);

        new Handler().postDelayed(() -> {

                if(fAuth.getCurrentUser() != null){

                    Intent intent = new Intent( packageContext: SplashScreen.this, GlavniIzbornik.class);

                    Pair[] pairs = new Pair[3];
                    pairs[0] = new Pair<View, String>(imgLogoSplashScreen, "logo_image");
                    pairs[1] = new Pair<View, String>(txtVelikiSplashScreen, "logo_text");
                    pairs[2] = new Pair<View, String>(txtVelikiSplashScreen, "logo_desc");

                    ActivityOptions options = ActivityOptions.makeSceneTransitionAnimation( activity: SplashScreen.this, pairs);
                    startActivity(intent, options.toBundle());

                }else{
                    Intent intent = new Intent( packageContext: SplashScreen.this, Login.class);

                    Pair[] pairs = new Pair[3];
                    pairs[0] = new Pair<View, String>(imgLogoSplashScreen, "logo_image");
                    pairs[1] = new Pair<View, String>(txtVelikiSplashScreen, "logo_text");
                    pairs[2] = new Pair<View, String>(txtVelikiSplashScreen, "logo_desc");

                    ActivityOptions options = ActivityOptions.makeSceneTransitionAnimation( activity: SplashScreen.this, pairs);
                    startActivity(intent, options.toBundle());
                }

        }, delayMillis: 3000);
    }
```

Slika 32. Programski kod *SplashScreen.java* klase

6.3.2. Registracija i prijava aplikacije

Kao i prije navedeno, nakon što se završi *SplashScreen* aktivnost, a korisnik nije prijavljen, aplikacija pokreće novu aktivnost. Ta aktivnost je aktivnost za prijavu. Ako je korisnik registriran u sustav, treba unijeti svoj e-mail i lozinku kako bi se prijavio. U slučaju da korisnik zaboravi svoju lozinku ili ju želi promijeniti, može koristiti tipku s tekstom „Zaboravili ste šifru?“ koja nudi opciju mijenjanja šifre, tako što korisnik upiše svoj e-mail i zahtjev za promjenu šifre mu se pošalje na e-mail. Ako korisnik nema kreirani račun,

koristi tipku koja pokreće novu aktivnost. Ta nova aktivnost je aktivnost za registraciju i ona nudi mogućnost registracije novog korisnika. Za registraciju su potrebi podatci u koje spadaju: *Ime i prezime, e-mail, lozinka i potvrda lozinke*. U slučaju da korisnik ima već prethodno kreirani račun, može doći do aktivnosti za prijavu pomoću tipke na dnu aktivnosti za registraciju. Zaslone aktivnosti registracije i prijave se može vidjeti na slici 33.



Slika 33. Registracija i prijava aplikacije

Na slici 34 je vidljiv programski kod tipke „Prijava“ u *Login.java* klasi za prijavu koja ima postavljen *OnClickListener*[19] kako bi se programski kod pokrenuo pritiskom korisnika. Kada korisnik pritisne navedenu tipku, aplikacija provjerava dali su uneseni svi podaci te jesu li ispravno uneseni. Ako su svi podatci ispravno uneseni, aplikacija pokreće *dialog box*[20] koji se prikazuje na više mjesta u aplikaciji kada se nešto učitava te funkciju od *firebase-auth* ovisnosti. Ako je prijava uspješna, pokreće se aktivnost glavnoga izbornika i miče *dialog box*. U slučaju da je prijava neuspješna, aplikacija ispisuje obavijest da prijava nije uspjela i također miče *dialog box*. Na slici 33 je vidljiv zaslon aplikacije u trenutku kada se pokrene *dialog box* za učitavanje.

[19] Definicija sučelja za povratni poziv koji se poziva kada se klikne na određeni View.
[20] Dialog box je mali prozor koji od korisnika traži da donese odluku ili unese dodatne informacije.

```
btnUlogirajSe.setOnClickListener((v) → {

        final String email = Objects.requireNonNull(txtEmail.getText()).toString();
        String sifra = Objects.requireNonNull(txtSifra.getText()).toString();

        if(TextUtils.isEmpty(email)){
            txtEmail.setError("Email nije unesen!");
            return;
        }
        if(TextUtils.isEmpty(sifra)){
            txtSifra.setError("Sifra nije unesena!");
            return;
        }
        if(sifra.length() < 6){
            txtSifra.setError("Sifra mora imati barem 6 znakova!");
            return;
        }

        ucitavanje.pokreniUcitavanje();

        fAuth.signInWithEmailAndPassword(email,sifra).addOnCompleteListener((task) → {
                if(task.isSuccessful()) {
                    Toast.makeText( context: Login.this, text: "Ulogiravanje uspijesno!", Toast.LENGTH_LONG).show();
                    Intent intent = new Intent( packageContext: Login.this, GlavniIzbornik.class);

                    Pair[] pairs = new Pair[3];
                    pairs[0] = new Pair<View, String>(imgLogoLogin, "logo_image");
                    pairs[1] = new Pair<View, String>(txtNaslov, "logo_text");
                    pairs[2] = new Pair<View, String>(txtPodnaslov, "logo_desc");
                    ActivityOptions options = ActivityOptions.makeSceneTransitionAnimation( activity: Login.this, pairs);

                    ucitavanje.makniUcitavanje();
                    startActivity(intent, options.toBundle());
                    Login.this.finishAfterTransition();

                }else{
                    Toast.makeText( context: Login.this,  text: "Neuspjesno ulogiravanje!" + task.getException().getMessage(), Toast.LENGTH_LONG).show();
                    ucitavanje.makniUcitavanje();
                }
        });
});
```

Slika 34. Programski kod za prijavu

Kada korisnik pritisne tipku „Zaboravljena šifra?“ na aktivnosti prijave, aplikacija otvara *dialog box* koji je vidljiv na slici 33. On traži od korisnika da unese svoj e-mail kako bi mogao primiti poruku u kojoj je zahtjev za promjenu šifre. U slučaju da korisnik nije dobro unio e-mail, aplikacija mu ispisuje obavijest da zahtjev za promjenu šifre nije poslan. Ako korisnik upiše dobar e-mail, aplikacija ispisuje obavijest da je zahtjev za promjenu šifre poslan na e-mail. Programski rješenje za promjenu šifre je vidljiv na slici 35.

Slika 36 prikazuje programsko rješenje za registraciju korisnika. Kada korisnik pritisne tipku „Registriraj se“ na aktivnosti za registraciju, aplikacija koristi funkciju od *firebase-auth* ovisnosti za kreiranje korisnika. Prije nego što se pokrene navedena funkcija, aplikacija provjerava dali su podatci dobro uneseni, te ako su dobro uneseni onda se pokreće funkcija. Ako funkcija uspješno kreira korisnika, aplikacija ispiše obavijest da je korisnik kreiran, usuprot tome, aplikacija ispiše zašto korisnik nije registriran. Nakon što se korisnik registrira, otvara se aktivnost glavnog izbornika.

```
    btnZaboravljenaSifra.setOnClickListener((v) → {
            final EditText resetEmail = new EditText(v.getContext());
            AlertDialog.Builder dijalogZaPromjenuSifre = new AlertDialog.Builder(v.getContext());
            dijalogZaPromjenuSifre.setTitle("Promjena sifre?");
            dijalogZaPromjenuSifre.setMessage("Upisi svoj email kako bi primio link za promjenu sifre!");
            dijalogZaPromjenuSifre.setView(resetEmail);

            dijalogZaPromjenuSifre.setPositiveButton( text: "Da", (dialog, which) → {
                    String emailPromjena = resetEmail.getText().toString();
                    if(emailPromjena.length() < 1){
                        Toast.makeText( context: Login.this, text: "Niste unijeli e-mail!", Toast.LENGTH_SHORT).show();
                    }
                    else{
                        fAuth.sendPasswordResetEmail(emailPromjena).addOnSuccessListener((OnSuccessListener) (aVoid) → {
                                Toast.makeText( context: Login.this, text: "Link za promjenu lozinke vam je poslan na email!", Toast.LENGTH_SHORT).show();
                        }).addOnFailureListener((e) → {
                                Toast.makeText( context: Login.this,  text: "Link za promjenu sifre nije poslan!" + e.getMessage(), Toast.LENGTH_SHORT).show();
                        });
                    }
            });
            dijalogZaPromjenuSifre.setNegativeButton( text: "Ne", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                }
            });

            dijalogZaPromjenuSifre.create().show();
    });
}
```

Slika 35. Programski kod za zaboravljenu šifru

```
fAuth.createUserWithEmailAndPassword(email,sifra).addOnCompleteListener((task) → {
        if(task.isSuccessful()){

            Toast.makeText( context: Register.this, text: "Racun napravljen!", Toast.LENGTH_LONG).show();
            userID = fAuth.getCurrentUser().getUid();


            DocumentReference documentReference = fStore.collection( collectionPath: "users").document(userID);
            Map<String, Object> user = new HashMap<>();
            user.put("ImePrezime", ImePrezime);
            user.put("Email", email);
            documentReference.set(user).addOnSuccessListener(new OnSuccessListener<Void>() {
                @Override
                public void onSuccess(Void aVoid) {
                    Toast.makeText( context: Register.this,  text: "Korisnički profil je napravljen!", Toast.LENGTH_SHORT).show();
                }
            }).addOnFailureListener(new OnFailureListener() {
                @Override
                public void onFailure(@NonNull Exception e) {
                    Toast.makeText( context: Register.this,  text: "Greška: " + e.toString(), Toast.LENGTH_SHORT).show();
                }
            });

            Intent intent = new Intent(getApplicationContext(),GlavniIzbornik.class);

            Pair[] pairs = new Pair[3];
            pairs[0] = new Pair<View, String>(imgLogoRegister, "logo_image");
            pairs[1] = new Pair<View, String>(txtNaslovRegistracija, "logo_text");
            pairs[2] = new Pair<View, String>(txtPodnaslovRegistracija, "logo_desc");

            ActivityOptions options = ActivityOptions.makeSceneTransitionAnimation( activity: Register.this, pairs);
            ucitavanje.makniUcitavanje();
            startActivity(intent, options.toBundle());
        }else{
            Toast.makeText( context: Register.this,  text: "Neuspjela registracija! " + task.getException().getMessage(), Toast.LENGTH_LONG).show();
            ucitavanje.makniUcitavanje();
        }

});
```

Slika 36. Programski kod za registraciju

6.3.3. Glavni izbornik aplikacije

Nakon što se korisnik prijavi u sustav, dolazi do zaslona koji je vidljiv na slici 37. U glavnom izborniku korisnik ima izbor za daljnje aktivnosti na aplikaciji. Na zaslonu se nalazi 6 tipki i svaka ima svoju svrhu, a te svrhe su odlazak na aktivnosti za biranje teorije ili vježbi iz propisa ili prve pomoći, odlazak na aktivnost gdje se rješava ispit, te odlazak na aktivnost gdje se bira kategorija rezultata za pregled. Osim tih tipki postoje i tipke koje se nalaze unutar *navigationDrawer*[21] izbornika gdje se nalaze *itemi*[22] za drugu svrhu. Svrha tih *itema* su mijenjanje korisničkih podataka, čitanje uputa za korištenje aplikacije i *item* za odjavu. U slučaju da je prijavljen korisnik ujedno i administrator, ima dodatan *item* koji služi za administraciju baze podataka.

Slika 37. Glavni izbornik aplikacije

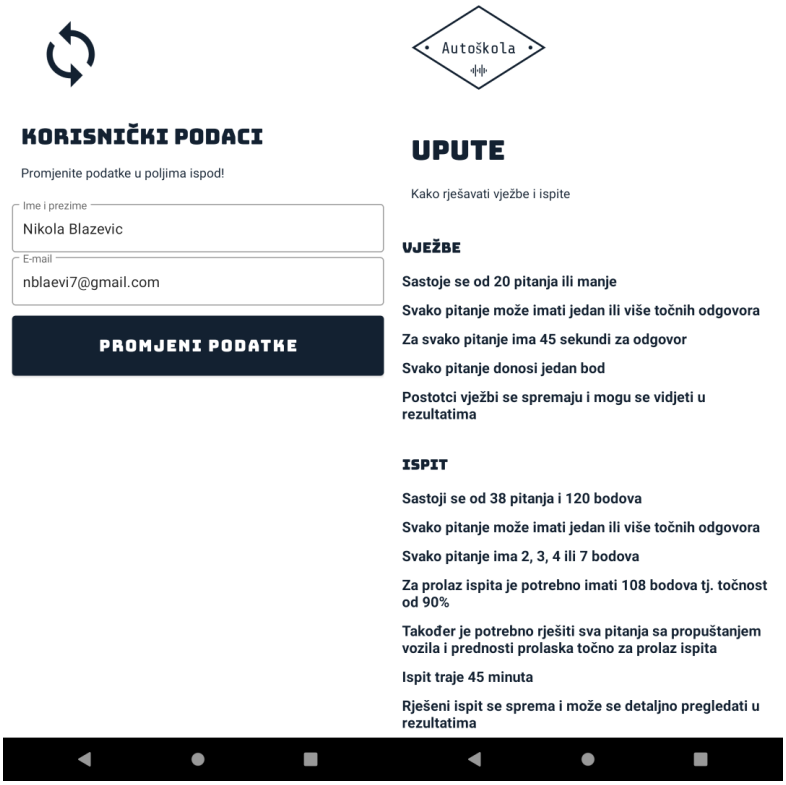
[21] NavigationDrawer pruža pristup odredištima i funkcijama aplikacije. Može biti trajno prikazan na zaslonu ili kontroliran ikonom navigacijskog izbornika.
[22] Svaki item opisuje svoje odredište pomoću tekstualne oznake i ikone.

Programski kod na slici 38 prikazuje kako će aplikacija postaviti *navigationDrawer* i što će napraviti u slučaju da korisnik pritisne jedan od ponuđenih *itema* unutar *navigationDrawera*. Svaki od prikazanih *itema* mijenja aktivnost u aplikaciji, ali jedino *item* „Odjava“ odjavljuje korisnika iz aplikacije te ga vraća na aktivnost za prijavu. *Item* „Administracija pitanja“ je poseban, jer kao i prije spomenuto, nije vidljiv običnim korisnicima nego samo administratoru.

```
setSupportActionBar(toolbar);
ActionBar actionBar = getSupportActionBar();
assert actionBar != null;
actionBar.setTitle("");

navigationView.bringToFront();
ActionBarDrawerToggle toggle = new ActionBarDrawerToggle( activity: this, drawerLayout, toolbar, "Open navigation drawer", "Close navigation drawer");
drawerLayout.addDrawerListener(toggle);
toggle.syncState();

Menu menu = navigationView.getMenu();
MenuItem admin = menu.findItem(R.id.navAdmin);

if (!Objects.equals(Objects.requireNonNull(fAuth.getCurrentUser()).getEmail(),  b: "nblaevi7@gmail.com")) {
    admin.setVisible(false);
} else {
    admin.setVisible(true);
}

navigationView.setNavigationItemSelectedListener(this);
headerView = navigationView.getHeaderView( index: 0);
txtHeaderImePrezime = headerView.findViewById(R.id.txtHeaderImePrezime);
txtHeaderEmail = headerView.findViewById(R.id.txtHeaderEmail);
@Override
public boolean onNavigationItemSelected(@NonNull MenuItem menuItem) {

    switch (menuItem.getItemId()){

        case R.id.navPromjenaPodataka:
            startActivity(new Intent( packageContext: GlavniIzbornik.this,PromjenaPodataka.class));
            break;

        case R.id.navUpute:
            startActivity(new Intent( packageContext: GlavniIzbornik.this, Upute.class));
            break;

        case R.id.navAdmin:
            startActivity(new Intent( packageContext: GlavniIzbornik.this, AdminBiranjeKategorije.class));
            break;

        case R.id.navOdjava:
            fAuth.signOut();
            Intent intent = new Intent( packageContext: GlavniIzbornik.this,Login.class);
            intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);

            Pair[] pairs = new Pair[3];
            pairs[0] = new Pair<View, String>(imgAuto, "logo_image");
            pairs[1] = new Pair<View, String>(txtNaslovGlavniIzbornik, "logo_text");
            pairs[2] = new Pair<View, String>(txtPodnaslovGlavniIzbornik, "logo_desc");

            ActivityOptions options = ActivityOptions.makeSceneTransitionAnimation( activity: GlavniIzbornik.this, pairs);
            startActivity(intent, options.toBundle());
            break;
    }

    drawerLayout.closeDrawer(GravityCompat.START);
    return true;
}
```

Slika 38. Programski kod *navigationDrawer* izbornika

6.3.4. Korisnički podaci i upute aplikacije



Slika 39. Korisnički podaci i upute u aplikaciji

Unutar aktivnosti korisničkih podataka korisnik ima mogućnost pregleda i promjene podataka. Ako se podatci zamjene te su dobrog formata, onda prilikom korištenja tipke „Promjeni podatke“ se mijenjaju korisnički podatci. Korisnik može promijeniti podatke jedino ako se je tek nedavno prijavljivao u sustav. Aktivnost od uputa služi za prikazivanje uputa kod rješavanja vježbi i ispita. Slika 39 prikazuje izgled od dvije navedene aktivnosti.

Unutar *PromjenaPodataka.java* klase su vidljive dvije funkcije na slici 40. Prva funkcija je *promjeniPodatke()* čija je svrha mijenjanje podataka. Kada korisnik zatraži promjenu podataka, prvo se provjerava dali su podatci dobro uneseni. Ako su podatci dobro uneseni i korisnik se nedavno prijavljivao u sustav, podatci će se promijeniti i odmah će biti vidljivi. Ako korisnik nije nedavno prijavljivan u sustav, *Firebase* mu ne dopušta promjenu podataka radi sigurnosnih razloga. Druga funkcija je *ucitavanjePodataka()* i ona se pokreće pri samom otvaranju aktivnosti od strane korisnika, te učitava korisničke

podatke s *Firebase*-a. Za učitavanje podataka s *Firebase*-a aplikacija uzima korisnički UID i koristi ga za kreiranje upita, te kada se upit izvrši, aplikacija koristi *documentSnapshot*[23] koji je *Firebase* vratio aplikaciji.

```
private void promjeniPodatke() {
    if(txtPromjeniImePrezime.getText().toString().isEmpty() || txtPromjeniEmail.getText().toString().isEmpty()){
        Toast.makeText( context: this,  text: "Sva polja moraju biti popunjena!", Toast.LENGTH_SHORT).show();
        return;
    }

    final String email = txtPromjeniEmail.getText().toString();
    user.updateEmail(email).addOnSuccessListener((OnSuccessListener) (aVoid) → {
            DocumentReference documentReference = fStore.collection( collectionPath: "users").document(user.getUid());
            Map<String, Object> map = new HashMap<>();
            map.put("Email", email);
            map.put("ImePrezime", txtPromjeniImePrezime.getText().toString());
            documentReference.update(map);

            Toast.makeText( context: PromjenaPodataka.this,  text: "Korisnik je ažuriran!", Toast.LENGTH_SHORT).show();

            Intent intent = new Intent(getApplicationContext(),GlavniIzbornik.class);
            intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
            startActivity(intent);
            overridePendingTransition(R.anim.slide_in_right,R.anim.slide_out_left);
    }).addOnFailureListener((e) → {
            Toast.makeText( context: PromjenaPodataka.this,  text: "Neuspjeno ažuriranje: " + e.getMessage(), Toast.LENGTH_SHORT).show();
    });
}

private void ucitavanjePodataka() {

    DocumentReference documentReference = fStore.collection( collectionPath: "users").document(user.getUid());
    documentReference.addSnapshotListener( activity: this, (documentSnapshot, e) → {
            assert documentSnapshot != null;
            txtPromjeniImePrezime.setText(documentSnapshot.getString( field: "ImePrezime"));
    });
    txtPromjeniEmail.setText(user.getEmail());
}
```

Slika 40. Programski kod iz *PromjenaPodataka.java* klase

6.3.5. Teorija propisi i prva pomoć i pitanja

Slika 41 prikazuje aktivnosti od sljedećih klasa: *TeorijaPropisi.java* , *TeorijaPropisiPrometniZnakovi.java, TeorijaPrvaPomoć.java, TeorijaPitanja.java*. Kada korisnik u glavnom izborniku odabere tipke od teorije propisa ili prve pomoći, aplikacija otvara aktivnost u kojoj korisnik može odabrati set koji želi učiti. Teorije propisa i prve pomoći imaju različite setove, ali unutar teorije propisa se nalazi poseban set „Prometni znakovi“. Pritiskom na taj set se otvara nova aktivnost koja nudi podsetove koji

[23] DocumentSnapshot sadrži podatke pročitane iz dokumenta u bazi podataka Cloud Firestore.

raspoređuju gradivo seta. Nakon što korisnik odabere set ili podset, aplikacija otvara aktivnost unutar koje se učitavaju određena pitanja s *Cloud Firestore*-a, te se učitavaju slike pitanja prema potrebi s *Cloud Storage*-a.

Slika 41. Izbornici teorije propisa, prve pomoći i pitanja u aplikaciji

Slika 42 prikazuje programsko rješenje za odabir setova „Opća Pitanja“ i „Prometni znakovi“ unutar kategorije propisa. Kada korisnik pritisne na set „Opća pitanja“, aplikacija prvo provjerava odakle je korisnik došao do trenutne aktivnosti. To radi pomoću *getIntent().getStringExtra()* funkcije gdje se može pronaći dodatan *String*[24]. U ovome slučaju korisnik dolazi iz glavnog izbornika što znači da ne postoji *String* koji označuje da se ušlo u aktivnost preko administracijskih opcija, te se stoga pokreće kod za promjenu aktivnosti. Navedena aktivnost je od *TeorijaPitanja.java* klase te se prije njenog pokretanja unutar *intenta* stavljaju *Stringovi* koji će se koristiti za kreiranje upita na Cloud Firestore. Ovo programsko rješenje se koristi na tipkama svih setova i podsetova osim na „Prometni znakovi“ setu. Također je na istoj slici vidljivo programsko rješenje za set „Prometni znakovi“. U ovom slučaju je korisnik također došao iz glavnog izbornika, te aplikacija pokreće aktivnost od klase *TeorijaPropisiPrometniZnakovi.java*, ali ne stavlja unutar *intenta Stringove*.

[24] String je niz znakova

```
btnOpcaPitanjaTeorija.setOnClickListener((v) → {

        if(getIntent().getStringExtra( name: "ADMIN")!=null){

            if(Objects.equals(getIntent().getStringExtra( name: "ADMIN"), b: "Dodavanje")){

                Intent intent = new Intent( packageContext: TeorijaPropisi.this, AdminPitanje.class);
                intent.putExtra( name: "KATEGORIJA", value: "Opća pitanja");
                intent.putExtra( name: "IDKATEGORIJE", value: "Kat1");
                intent.putExtra( name: "IDSETA", value: "OpcaPitanja");
                intent.putExtra( name: "IDDETALJNO", value: "0");
                intent.putExtra( name: "ADMIN", getIntent().getStringExtra( name: "ADMIN"));
                startActivity(intent);

            }else{

                Intent intent = new Intent( packageContext: TeorijaPropisi.this,EditiranjePitanja.class);
                intent.putExtra( name: "KATEGORIJA", value: "Opća pitanja");
                intent.putExtra( name: "IDKATEGORIJE", value: "Kat1");
                intent.putExtra( name: "IDSETA", value: "OpcaPitanja");
                intent.putExtra( name: "IDDETALJNO", value: "0");
                intent.putExtra( name: "ADMIN", getIntent().getStringExtra( name: "ADMIN"));
                startActivity(intent);
            }

        }else{
            Intent intent = new Intent( packageContext: TeorijaPropisi.this,TeorijaPitanja.class);
            intent.putExtra( name: "KATEGORIJA", value: "Opća pitanja");
            intent.putExtra( name: "IDKATEGORIJE", value: "Kat1");
            intent.putExtra( name: "IDSETA", value: "OpcaPitanja");
            intent.putExtra( name: "IDDETALJNO", value: "0");
            intent.putExtra( name: "LOGO", value: "logo_image_opca");

            Pair[] pairs = new Pair[1];
            pairs[0] = new Pair<View, String>(btnOpcaPitanjaTeorija, "logo_image_opca");

            ActivityOptions options = ActivityOptions.makeSceneTransitionAnimation( activity: TeorijaPropisi.this, pairs);
            startActivity(intent, options.toBundle());
        }
});

btnPrometniZnakoviTeorija.setOnClickListener((v) → {

        if(getIntent().getStringExtra( name: "ADMIN")!=null){
            Intent intent = new Intent( packageContext: TeorijaPropisi.this, TeorijaPropisiPrometniZnakovi.class);
            intent.putExtra( name: "ADMIN", getIntent().getStringExtra( name: "ADMIN"));
            startActivity(intent);
        }else{
            Intent intent = new Intent( packageContext: TeorijaPropisi.this, TeorijaPropisiPrometniZnakovi.class);
            startActivity(intent);
        }
});
```

Slika 42. Programski kod iz *TeorijaPropisi.java* klase

```
FireStoreListaPitanja = findViewById(R.id.FireStoreListaPitanja);
fStore = FirebaseFirestore.getInstance();

if(String.valueOf(getIntent().getStringExtra( name: "IDDETALJNO")).equals("0")){
    query = fStore.collection( collectionPath: "Pitanja").document(String.valueOf(getIntent()
            .getStringExtra( name: "IDKATEGORIJE"))).collection(String.valueOf(getIntent().getStringExtra( name: "IDSETA")));
}else{
    query = fStore.collection( collectionPath: "Pitanja").document(String.valueOf(getIntent()
            .getStringExtra( name: "IDKATEGORIJE"))).collection(String.valueOf(getIntent().getStringExtra( name: "IDSETA")))
            .document(String.valueOf(getIntent().getStringExtra( name: "IDDETALJNO"))).collection( collectionPath: "Pitanja");
}


FirestoreRecyclerOptions<Pitanje> options = new FirestoreRecyclerOptions.Builder<Pitanje>()
        .setQuery(query, Pitanje.class)
        .build();

adapter = new FirestoreRecyclerAdapter<Pitanje, PitanjeViewHolder>(options) {
    @NonNull
    @Override
    public PitanjeViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.lista_pitanje, parent,  attachToRoot: false);
        return new PitanjeViewHolder(view);
    }

    @Override
    protected void onBindViewHolder(@NonNull PitanjeViewHolder holder, int position, @NonNull Pitanje model) {
        holder.Pitanje.setText((position + 1) + ". " + model.getPitanje());
        holder.Odgovor1.setText(model.getOdgovor1());
        holder.Odgovor2.setText(model.getOdgovor2());
        holder.Odgovor3.setText(model.getOdgovor3());

        if(model.getOdgovor3().equals("0")){...}else{
            holder.Odgovor3.setVisibility(View.VISIBLE);
        }

        switch ((int) model.getTocanOdgovor()){...}

        if(!model.getSlika().equals("")){...}else{
            holder.Slika.setVisibility(View.GONE);
        }
    }
};

FireStoreListaPitanja.setLayoutManager(new LinearLayoutManager( context: this));
FireStoreListaPitanja.setAdapter(adapter);
```

Slika 43. Programski kod iz *TeorijaPitanja.java* klase

Programsko rješenje sa slike 43 prikazuje kako aplikacija kreira upit koji se koristi za učitavanje pitanja s *Cloud Firestore*-a pomoću *RecyclerView*[25] adaptera. Aplikacija prvo sastavlja *Cloud Firestore* upit pomoću prije navedenih *Stringova* stavljenih u *intent* u prethodnoj aktivnosti. Kada se upit sastavi on se prosljeđuje varijabli od ovisnosti

[25] RecyclerView naprednija je i fleksibilnija verzija ListViewa koji služi za prikaz liste podataka.

firebase-ui-firestore, a naziv funkcije je *FirestoreRecyclerOptions<>.* Adapter pomoću funkcije *FirestoreRecyclerOptions<>* i klasa *Pitanje* te *PitanjeViewHolder* (slika 44) kreira listu pitanja tako što svako pitanje stavlja u novi *layout* (slika 45) jedno ispod drugog. Adapter također koristi postojeće funkcije *onStop()* i *onStart()* koje su nadjačane.

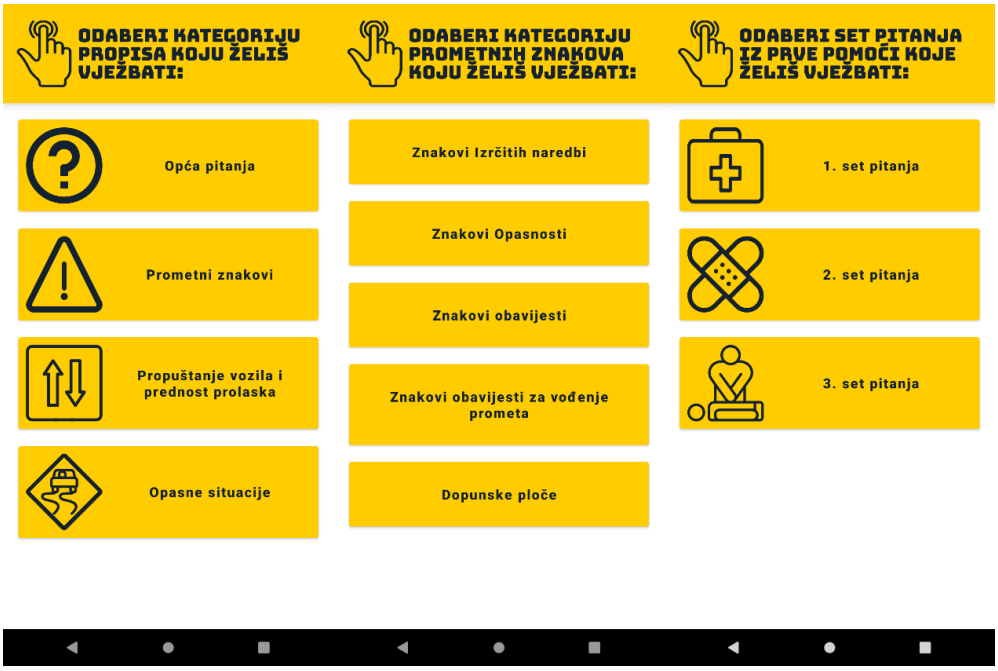
```
private class PitanjeViewHolder extends RecyclerView.ViewHolder{

    private TextView Pitanje, Odgovor1, Odgovor2, Odgovor3;
    private ImageView Slika;

    public PitanjeViewHolder(@NonNull View itemView) {
        super(itemView);

        Pitanje = itemView.findViewById(R.id.Pitanje);
        Odgovor1 = itemView.findViewById(R.id.Odgovor1);
        Odgovor2 = itemView.findViewById(R.id.Odgovor2);
        Odgovor3 = itemView.findViewById(R.id.Odgovor3);
        Slika = itemView.findViewById(R.id.Slika);
    }
}

@Override
protected void onStop() {
    super.onStop();
    adapter.stopListening();
}

@Override
protected void onStart() {
    super.onStart();
    adapter.startListening();
}
```

Slika 44. Programski kod iz TeorijaPitanja.java klase

Slika 45. lista_pitanje.xml

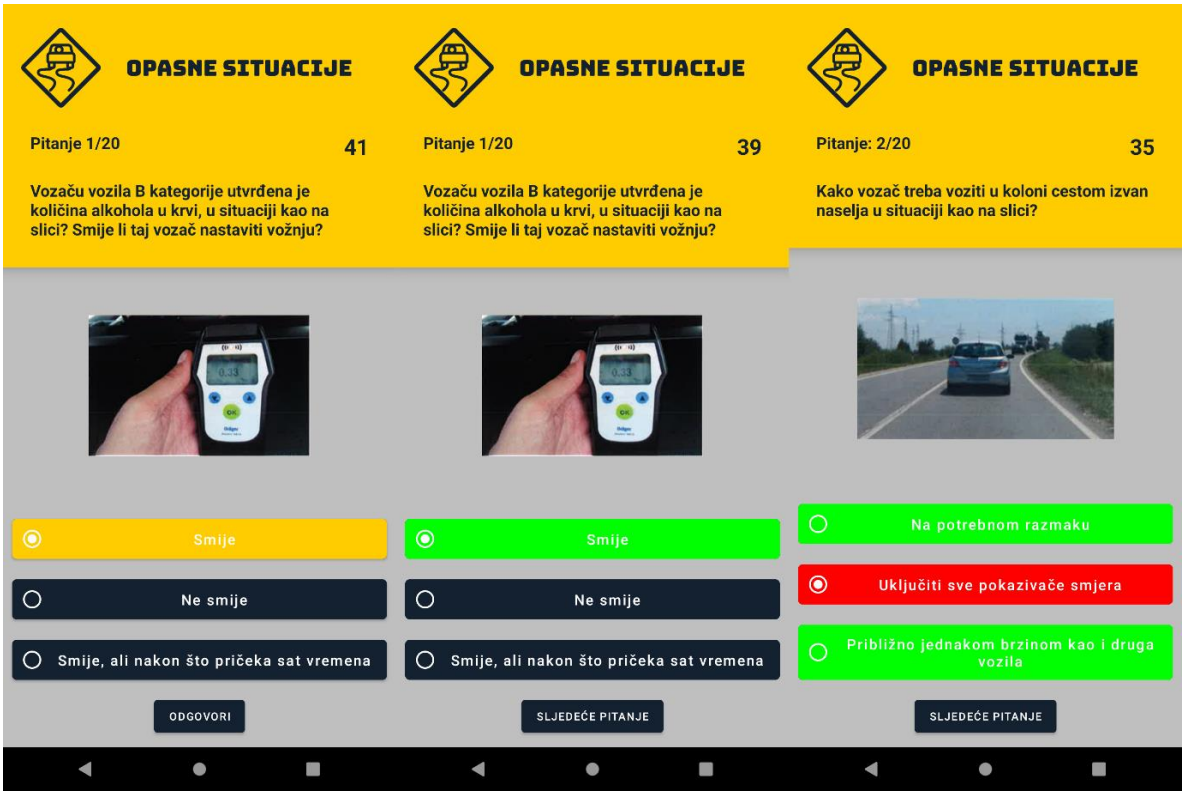
6.3.6. Vježbe propisi, prva pomoć i pitanja

Klase *Vjezbepropisi.java, VjezbePropisiPrometniZnakovi.java, VjezbePrvaPomoc.java* su gotovo identične kao i prije navedene klase za teoriju. Korisnik bira koji set ili podset želi vježbati.



Slika 46. Izbornici vježbe propisa i prva pomoć u aplikaciji

Kada korisnik odabere set ili podset, aplikacija učitava pitanja iz određenog seta u aktivnost od klase *VježbePitanja.java* vidljivu na slici 47. Unutar te aktivnosti se pokreće vrijeme nakon što se pitanje učita te se prikaže broj pitanja u vježbi i pitanje uz odgovore te slika ako pitanje ima sliku. Pritiskom na bilo koji odgovor, taj odgovor se označava i mijenja boju, a pritiskom na tipku „Odgovori“ se provjerava točnost korisnikovog odgovora i zaustavlja se vrijeme. U slučaju da je korisnikov odgovor točan, točni odgovori mijenjaju boju u zelenu, a netočni odgovori ne mijenjaju boju. Ako korisnik netočno odgovori, točni odgovori mijenjaju boju u zelenu, ali netočni odgovori mijenjaju boju u crvenu. Kada je korisnik pregledao rješenje pitanja koristi tipku „Sljedeće pitanje“, koje resetira vrijeme te ga pokreće kada se pitanja, odgovori i slika učitaju. Ako sljedeće pitanje nema sliku, slika se sakriva sve dok ne dođe pitanje koje ima sliku. Treći odgovor u pitanju može biti prazan kao i slika, stoga se isto skriva ako ga nema.



Slika 47. Vježbe u aplikaciji

Slika 48. Rezultati riješene vježbe u aplikaciji

Kada se vježba riješi, aplikacija otvara aktivnost gdje ispisuje broj točnih odgovora što je vidljivo na slici 48. Pritiskom na tipku „Izlaz“ korisnik se vraća u glavni izbornik.

Funkcije na slici 49 se koriste za kreiranje, rješavanje, ispravljanje i spremanje vježbi unutar klase *VjezbePitanja.java.*

```
private void getListaPitanja(){...}

private void setPitanje() {...}

private void startTimer() {...}

private void provjeraOdogovora(String OznakaOdgovor1, String OznakaOdgovor2, String OznakaOdgovor3){...}

private void promjeniPitanje() {...}

private void playAnim(final View view, final int value, final int viewBroj) {...}

private void oznacavanjeOdgovora(String oznaceniOdgovor, View view){...}
```

Slika 49. Funkcije *VjezbePitanja.java* klase

```
btnOdgovor1.setOnClickListener((v) -> { oznacavanjeOdgovora( oznaceniOdgovor: "1", v); });
btnOdgovor2.setOnClickListener((v) -> { oznacavanjeOdgovora( oznaceniOdgovor: "2", v); });
btnOdgovor3.setOnClickListener((v) -> { oznacavanjeOdgovora( oznaceniOdgovor: "3", v); });

btnOdgovori.setOnClickListener((v) -> {
        countDownTimer.cancel();
        if(odgovor == 0){
            btnOdgovori.setText("Sljedeće Pitanje");
            provjeraOdogovora(OznakaOdgovor1, OznakaOdgovor2, OznakaOdgovor3);
            odgovor = 1;
        }else{
            btnOdgovori.setText("Odgovori");
            oznacavanjeOdgovora( oznaceniOdgovor: "0", v);
            promjeniPitanje();
            odgovor = 0;
        }

});

getListaPitanja();
```

Slika 50. Programski kod za tipku „Odgovori“/„Sljedeće pitanje“ iz *VjezbePitanja.java* klase

U programskom rješenju sa slike 50 su postavljeni *onClickListeneri* na tipke za odgovore i na tipku „Odgovori“/„Sljedeće pitanje“. Kada korisnik pritisne na jedan od odgovora, poziva se funkcija *oznacavanjeOdgovora()*. Tipka „Odgovori“/„Sljedeće pitanje“ na pritisak mijenja svoj tekst, te poziva funkciju *provjeraOdgovora()* ili *oznacavanjeOdgovora()* zajedno sa funkcijom *promjeniPitanje()* ovisno o tome dali se na pitanje odgovara ili se pitanje mijenja.

```
ListaPitanja = new ArrayList<>();

if(String.valueOf(getIntent().getStringExtra( name: "IDDETALJNO")).equals("0")){

    DocumentReference documentReference = fStore.collection( collectionPath: "Pitanja") CollectionReference
            .document(String.valueOf(getIntent().getStringExtra( name: "IDKATEGORIJE"))) DocumentReference
            .collection( collectionPath: "BrojPitanja") CollectionReference
            .document(String.valueOf(getIntent().getStringExtra( name: "IDSETA")));
    documentReference.get().addOnCompleteListener((task) -> {
            if(task.isSuccessful()){
                velinicnaSeta = Objects.requireNonNull(Objects.requireNonNull(task.getResult()).getLong( field: "BrojPitanja")).intValue();

                if(velinicnaSeta < 20){...}else{
                    txtBrojPitanja.setText("Pitanje 1/20");
                    List<Integer> ListaBrojeva = new ArrayList<>();
                    Random random = new Random();
                    int temp;

                    for(int i = 0; i < 20; i++){
                        temp = random.nextInt(velinicnaSeta) +1;
                        if(ListaBrojeva.contains(temp)){
                            i--;
                        }else{
                            ListaBrojeva.add(temp);
                            DocumentReference documentReference1 = fStore.collection( collectionPath: "Pitanja") CollectionReference
                                    .document(String.valueOf(getIntent().getStringExtra( name: "IDKATEGORIJE"))) DocumentReference
                                    .collection(String.valueOf(getIntent().getStringExtra( name: "IDSETA"))) CollectionReference
                                    .document( documentPath: "Pitanje" + ListaBrojeva.get(i));
                            documentReference1.get().addOnCompleteListener((task) -> {
                                    if(task.isSuccessful()){
                                        DocumentSnapshot snapshot = task.getResult();
                                        assert snapshot != null;
                                        Pitanje pitanje = new Pitanje(snapshot.getString( field: "Pitanje"),
                                                snapshot.getString( field: "Odgovor1"),
                                                snapshot.getString( field: "Odgovor2"),
                                                snapshot.getString( field: "Odgovor3"),
                                                Objects.requireNonNull(snapshot.getLong( field: "TocanOdgovor")),
                                                snapshot.getString( field: "Slika"));
                                        ListaPitanja.add(pitanje);
                                        setPitanje();
                                    }
                            });
                        }
                    }
                }
            }
    });
```

Slika 51. Dio programskog koda funkcije *getListaPitanja()*

Funkcija *getListaPitanja()* služi za popunjavanje liste pitanja. Dio programskog rješenja je vidljiv na slici 51. Prvo se lista inicijalizira, te potom aplikacija kreira upit na sličan način kako se upit kreira unutar klase *TeorijaPitanja.java*. Upit se kreira na *documentReference*[26] funkciju te se potom koristi *get().addOnCompleteListener()* na istoj kako bi se došlo do broja pitanja od odabranog seta. Funkcija vraća nazad broj pitanja te

[26] DocumentReference se odnosi na mjesto dokumenta u bazi podataka Cloud Firestore i može se koristiti za pisanje, čitanje ili slušanje lokacije.

aplikacija ovisno o tom broju povlači pitanja iz seta s novim upitom. U slučaju da je broj pitanja 0, aplikacija ispisuje da je baza podataka prazna, a ako je broj pitanja u setu manji od 20, aplikacija povlači sva pitanja iz seta u listu. Kada set ima više od 20 pitanja, aplikacija nasumično izvlači 20 različitih brojeva koji ne mogu biti manji od 1 i veći od ukupnog broja pitanja u odabranom setu. Zatim radi upite pomoću *for* petlje da s *Cloud Firestore*-a učita nasumično odabrana pitanja i spremi ih u listu. Time se broj čitanja na *Cloud Firestore*-u optimizira, a vježba nije prevelika. Kada se pitanja učitaju u listu, poziva se funkcija *setPitanje().* Učitavanje pitanja iz setova i podsetova je slično, te se određuje s dodatnim *Stringom* „IDDETALJNO“.

```
txtBrojac.setText(String.valueOf(45));
txtTekstPitanja.setText(ListaPitanja.get(0).getPitanje());

btnOdgovor1.setText(ListaPitanja.get(0).getOdgovor1());
btnOdgovor2.setText(ListaPitanja.get(0).getOdgovor2());
if(ListaPitanja.get(redniBrojPitanja).getOdgovor3().equals("0")){
    btnOdgovor3.setVisibility(View.GONE);
}else{
    btnOdgovor3.setVisibility(View.VISIBLE);
    btnOdgovor3.setText(ListaPitanja.get(0).getOdgovor3());
}
redniBrojPitanja = 0;

if(!ListaPitanja.get(0).getSlika().equals("")){
    Picasso.get().load(ListaPitanja.get(0).getSlika()).into(imgPitanja, new com.squareup.picasso.Callback() {
        @Override
        public void onSuccess() {
            ucitavanje.makniUcitavanje();
            startTimer();
        }
        @Override
        public void onError(Exception e) {
        }
    });
}else{
    imgPitanja.setVisibility(View.GONE);
    if(provjeraTimera == 0){
        ucitavanje.makniUcitavanje();
        startTimer();
    }
    provjeraTimera++;
}
```

Slika 52. Programski kod funkcije *setPitanje()*

Programsko rješenje sa slike 52 prikazuje funkciju *setPitanje()* koja postavlja prvo pitanje na *layout* aktivnosti. Vrijeme se postavlja na 45 sekundi te se učitava sadržaj pitanja i odgovora na *layout* aktivnosti i inicijalizira se redni broj pitanja na 0. Također se učitava i

slika ako postoji pomoću ovisnosti *picasso:picasso,* koja je korištena za učitavanje svih slika u aplikaciji koje dolaze iz *Cloud Storage*-a. Nakon učitavanja slike ako postoji, miče se *dialog box* učitavanja koji je pokrenut na samom početku otvaranja aktivnosti i pokreće se vrijeme pomoću funkcije *startTimer().*

```
countDownTimer = new CountDownTimer( millisInFuture: 46000,  countDownInterval: 1000) {
    @Override
    public void onTick(long millisUntilFinished) {
        if(millisUntilFinished < 45000){
            txtBrojac.setText(String.valueOf(millisUntilFinished / 1000));
        }
    }

    @Override
    public void onFinish() {
        oznacavanjeOdgovora( oznaceniOdgovor: "0", btnOdgovori.getRootView());
        promjeniPitanje();
    }
};
countDownTimer.start();
```

Slika 53. Programski kod funkcije *startTimer()*

Unutar funkcije *startTimer()* sa slike 53 kreira se brojač koji je postavljen na 46 sekundi te interval od jedne sekunde. Svaki put kad prođe sekunda, vrijeme na brojaču se ažurira tako što se smanji za jednu sekundu. U slučaju da korisnik ne odgovori na pitanje, automatski se pokreće funkcija *oznacavanjeOdgovora(),* a zatim funkcija *promjeniPitanje().* Na kraju funkcije se pokreće brojač.

```
if(oznaceniOdgovor.equals("1")){
    if(OznakaOdgovor1.equals("0")){
        OznakaOdgovor1 = "1";
        ((Button)view).setBackgroundTintList(ColorStateList.valueOf(Color.parseColor( colorString: "#ffcc00")));
        int img = R.drawable.oznaceno_icon;
        btnOdgovor1.setCompoundDrawablesWithIntrinsicBounds(img,  top: 0,  right: 0,  bottom: 0);
    }
    else{
        OznakaOdgovor1 = "0";
        ((Button)view).setBackgroundTintList(ColorStateList.valueOf(Color.parseColor( colorString: "#132131")));
        int img = R.drawable.neoznaceno_icon;
        btnOdgovor1.setCompoundDrawablesWithIntrinsicBounds(img,  top: 0,  right: 0,  bottom: 0);
    }
}
```

Slika 54. Dio programskog koda funkcije *oznacavanjeOdgovora()*

Kao prije navedeno, kada korisnik pritisne tipke s odgovorima ili tipku „Sljedeće pitanje“, pokreće se funkcija *oznacavanjeOdgovora()* čiji je dio programskog rješenja vidljiv na slici 54. Funkcija prima 2 argumenta, a to su *String* argumenti s vrijednosti 0, 1, 2 ili 3 te *View*[27] pritisnute tipke. Dio programskog rješenja prikazuje da ako je vrijednost *String* argumenta 1, aplikacija mijenja vrijednost prvog odgovora. Ako prvi odgovor nije bio pritisnut prije toga ima vrijednost odgovora unutar varijable *OznakaOdgovor1* jednaku nuli. Tipka prvog odgovora se izmjenjuje tako što joj se mijenja boja i ikona s lijeve strane te varijabla *OznakaOdgovor1* dobiva vrijednost 1. Sljedeće put kada se pritisne prvi odgovor, aplikacija radi isti proces kao i prije, ali ovaj put *OznakaOdgovor1* dobiva vrijednost 0, te se boja i ikona sa lijeve strane vračaju na početne postavke, čime je prvi odgovor postao neoznačen. Isto programsko rješenje se koristi i za drugi i treći odgovor, ali ne i za pritisak na tipku „Sljedeće pitanje“. Pritiskom na spomenutu tipku funkcija prima *String* argument s vrijednosti 0. Time se svim *OznakaOdgovor* varijablama odgovora vraćaju boje na početne postavke što znači da su sva pitanja neoznačene boje.

Kada korisnik stisne tipku „Odgovori“, poziva se funkcija *provjeraOdgovora()* čiji je dio programskog rješenja vidljiv na slici 55. Navedena funkcija ima 3 *String* argumenta spomenutih ranije: *OznakaOdgovor1, OznakaOdgovor2* i *OznakaOdgovor3.* Pomoću argumenata aplikacija sastavlja varijablu *OznaceniOdgovor* čija vrijednost može biti 1, 12, 2, 13, 123, 23 i 3. Varijabla se uspoređuje s pitanjem iz liste pitanja pod rednim brojem zadnje prikazanog pitanja. Ako su varijabla *OznaceniOdgovor* i varijabla točnog odgovora pitanja iste, tipke označenih odgovora postanu zelene pomoću *switcha*[28] koji mijenja boju tipki koje su označene. Primjerice za vrijednost varijable 23, mijenja se boja drugog i trećeg odgovora. Nakon promjene nad tipkama povećava se vrijednost varijable *rezultatExtra* za 1 više koja je inicijalizirana na 0 na pokretanju trenutne aktivnosti. U ostatku programskog rješenja funkcije *provjeraOdgovora()* kada korisnik odgovori netočno, aplikacija mijenja tipke točnih odgovora u zeleno kao i kada korisnik odgovori točno, ali mijenja boje netočnih odgovora u crveno i ne povećava vrijednost varijable *rezultatExtra*.

[27] View je osnovni građevni blok sučelja na androidu.

[28] Switch omogućuje testiranje varijable na jednakost prema popisu vrijednosti. Svaka se vrijednost naziva slučaj, a varijabla koja se switcha je provjerena za svaki slučaj.

```
if(OznakaOdgovor1.equals("1")){
    OznaceniOdgovor = 1;
}

if(OznakaOdgovor2.equals("1")){
    if(OznaceniOdgovor == 1){
        OznaceniOdgovor = 12;
    }
    else{
        OznaceniOdgovor = 2;
    }
}

if(OznakaOdgovor3.equals("1")){
    if(OznaceniOdgovor == 1){
        OznaceniOdgovor = 13;
    }
    if(OznaceniOdgovor == 12){
        OznaceniOdgovor = 123;
    }
    if(OznaceniOdgovor == 2){
        OznaceniOdgovor = 23;
    }
    if(OznaceniOdgovor == 0){
        OznaceniOdgovor = 3;
    }
}

if(OznaceniOdgovor == ListaPitanja.get(redniBrojPitanja).getTocanOdgovor()) {

    switch (OznaceniOdgovor){
        case 1:
            btnOdgovor1.setBackgroundTintList(ColorStateList.valueOf(Color.GREEN));
            break;
        case 2:
            btnOdgovor2.setBackgroundTintList(ColorStateList.valueOf(Color.GREEN));
            break;
        case 3:
            btnOdgovor3.setBackgroundTintList(ColorStateList.valueOf(Color.GREEN));
            break;
        case 12:
            btnOdgovor1.setBackgroundTintList(ColorStateList.valueOf(Color.GREEN));
            btnOdgovor2.setBackgroundTintList(ColorStateList.valueOf(Color.GREEN));
            break;
        case 13:
            btnOdgovor1.setBackgroundTintList(ColorStateList.valueOf(Color.GREEN));
            btnOdgovor3.setBackgroundTintList(ColorStateList.valueOf(Color.GREEN));
            break;
        case 23:
            btnOdgovor2.setBackgroundTintList(ColorStateList.valueOf(Color.GREEN));
            btnOdgovor3.setBackgroundTintList(ColorStateList.valueOf(Color.GREEN));
            break;
        case 123:
            btnOdgovor1.setBackgroundTintList(ColorStateList.valueOf(Color.GREEN));
            btnOdgovor2.setBackgroundTintList(ColorStateList.valueOf(Color.GREEN));
            btnOdgovor3.setBackgroundTintList(ColorStateList.valueOf(Color.GREEN));
            break;
    }
    rezultatExtra++;
}
```

Slika 55. Dio programskog koda funkcije *provjeraOdgovora()*

```
ucitavanje.pokreniUcitavanje();
if (redniBrojPitanja < ListaPitanja.size() - 1){

    btnOdgovor1.setEnabled(true);
    btnOdgovor2.setEnabled(true);
    btnOdgovor3.setEnabled(true);

    redniBrojPitanja++;

    playAnim(txtTekstPitanja,  value: 0,  viewBroj: 0);
    playAnim(btnOdgovor1,  value: 0,  viewBroj: 1);
    playAnim(btnOdgovor2,  value: 0,  viewBroj: 2);
    if(ListaPitanja.get(redniBrojPitanja).getOdgovor3().equals("0")){
        btnOdgovor3.setVisibility(View.GONE);
    }else{
        btnOdgovor3.setVisibility(View.VISIBLE);
        playAnim(btnOdgovor3,  value: 0,  viewBroj: 3);
    }

    txtBrojPitanja.setText("Pitanje: " + (redniBrojPitanja+1) + "/" + ListaPitanja.size());

    txtBrojac.setText(String.valueOf(45));

    if(!ListaPitanja.get(redniBrojPitanja).getSlika().equals("")){...}else{
        imgPitanja.setVisibility(View.GONE);
        ucitavanje.makniUcitavanje();
        startTimer();
    }

    int img = R.drawable.neoznaceno_icon;
    btnOdgovor1.setCompoundDrawablesWithIntrinsicBounds(img,  top: 0,  right: 0,  bottom: 0);
    btnOdgovor2.setCompoundDrawablesWithIntrinsicBounds(img,  top: 0,  right: 0,  bottom: 0);
    btnOdgovor3.setCompoundDrawablesWithIntrinsicBounds(img,  top: 0,  right: 0,  bottom: 0);
}
```

Slika 56. Dio programskog koda funkcije *promjeniPitanje()*

Slika 56 prikazuje dio programskog rješenja funkcije za promjenu pitanja *promjeniPitanje()* koja se poziva pritiskom na tipku „Odgovori“ ili istekom vremena za odgovor na pitanje. Kada se funkcija pozove, prvo se pokreče *dialog box* za učitavanje, varijable *redniBrojPitanja* se povećava za 1, te se poziva funkcija *playAnim()* za svaki odgovor, ali kod trećeg odgovora se provjerava u listi pitanja dali na sljedećem pitanju postoji treće pitanje, te ako postoji tek onda se pokreće funkcija *playAnim()* za navedeni odgovor. U slučaju da ne postoji, treći odgovor se skriva. Mijenja se tekst rednog broja pitanja na *layoutu* aktivnosti, resetira se vrijeme, te se kao i kod trećeg odgovora provjerava postoji li slika u listi pitanja. Ako postoji slika se učitava, a ako ne postoji, slika

se skriva. Nakon učitavanja ili skrivanja slike, aplikacija miče *dialog box* i pokreće vrijeme. Aplikacija također mijenja ikone koje se nalaze lijevo na tipkama odgovora na početnu ikonu koja prikazuje odgovor kao neoznačen. Ostatak programskog rješenja funkcije *promjeniPitanje()* je kada se korisnik nalazi na zadnjem pitanju, sprema broj točnih odgovora i ukupan broj odgovora. Spremaju se na *Cloud Firestore* unutar korisnikovih rješenja u vježbu koju je korisnik rješavao. Nakon toga pokreće se nova aktivnost na kojoj se prikazuju rezultati kao na slici 48.

```
view.animate().alpha(value).scaleX(value).scaleY(value).setDuration(150)
        .setStartDelay(100)
        .setInterpolator(new DecelerateInterpolator()).setListener((AnimatorListenerAdapter) (animation) → {
        if(value == 0){
            switch (viewBroj){
                case 0:
                    ((TextView)view).setText(ListaPitanja.get(redniBrojPitanja).getPitanje());
                    break;

                case 1:
                    ((Button)view).setText(ListaPitanja.get(redniBrojPitanja).getOdgovor1());
                    break;

                case 2:
                    ((Button)view).setText(ListaPitanja.get(redniBrojPitanja).getOdgovor2());
                    break;

                case 3:
                    ((Button)view).setText(ListaPitanja.get(redniBrojPitanja).getOdgovor3());
                    break;
            }

            if(viewBroj != 0){
                ((Button)view).setBackgroundTintList(ColorStateList.valueOf(Color.parseColor( colorString: "#132131")));
            }
            playAnim(view,  value: 1, viewBroj);
        }
});
```

Slika 57. Programski kod funkcije *playAnim()*

Funkcija *playAnim()* sa slike 57 prikazuje programsko rješenje koje mijenja tekst pitanja i odgovora na *layoutu* akotivnosti, a poziva se unutar funkcije *promjeniPitanje().* Funkcija *playAnim()* ima 3 argumenta, a to su: *View* pitanja ili odgovora, *int*[29] varijablu s vrijednosti 0 ili 1, te *int* varijablu koja označava koji odgovor ili pitanje treba promijeniti funkcija. Ako

[29] Int se koristi za spremanje brojeva

je vrijednost drugog argumenta 0, argumentirani *View* će se s animacijom postepeno smanjiti na veličinu 0, što znači da će se sakriti. Nakon toga se pomoću *switcha* mijenja tekst *Viewa* na *layoutu* aktivnosti, te ako treći argument nije 0, promijenit će se boja *Viewa* na početne postavke. Potom se funkcija rekurzivno poziva s istim argumentima osim drugog argumenta koji ima vrijednost 1, što znači da će animacijom *View* postati postepeno vidljiv s novim tekstom.

```
@Override
public void onBackPressed(){

    DialogInterface.OnClickListener dialogClickListener = (dialog, which) -> {
            switch (which){
                case DialogInterface.BUTTON_POSITIVE:
                    countDownTimer.cancel();
                    VjezbePitanja.super.onBackPressed();
                    break;

                case DialogInterface.BUTTON_NEGATIVE:
                    break;
            }
    };
    AlertDialog.Builder builder = new AlertDialog.Builder( context: this);
    builder.setMessage("Jeste li sigurni da želite napustiti ovu vježbu?").setPositiveButton( text: "Da", dialogClickListener)
            .setNegativeButton( text: "Ne", dialogClickListener).show();
}
```

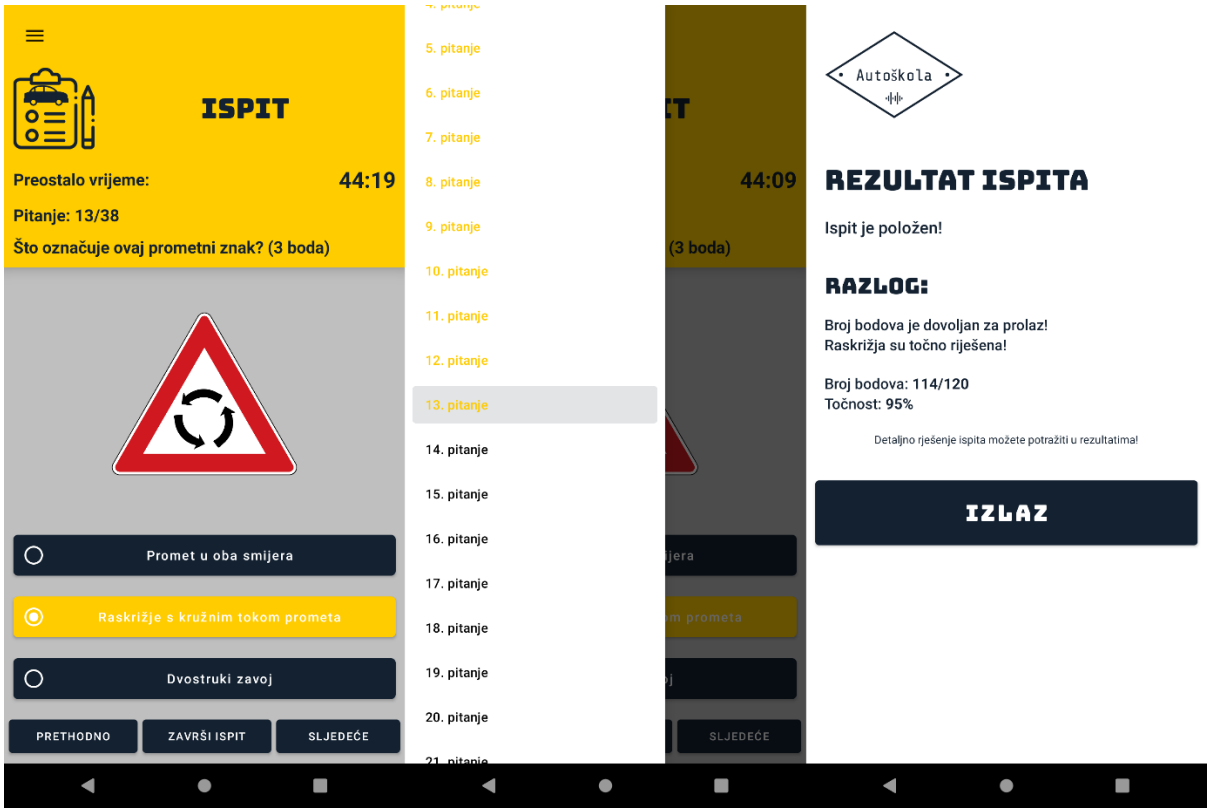
Slika 58. Programski kod nadjačane funkcije *onBackPressed()*

Na slici 58 prikazana je nadjačana funkcija *onBackPressed()* koja se pokreće kada korisnik pritisne tipku za vraćanje, koja je dio android OS sustava. Prilikom pritiska navedene tipke, kreira se *dialog box* s tekstom koji korisnika pita želi li napustiti vježbu, te nudi opcije „Da“ i „Ne“. Pritiskom na opciju „Da“, vrijeme se zaustavlja te se aplikacija vraća u prethodnu aktivnost, a pritiskom na opciju „Ne“, ne događa se ništa i vježba se nastavlja dalje.

6.3.7. Ispit aplikacije

Kada korisnik koji se nalazi na aktivnosti glavnog izbornika pritisne tipku „Ispit“, aplikacija otvara novu aktivnost od klase *Ispit.java* koja je vidljiva na slici 58. gdje se prvo učitavaju pitanja koja će biti u ispitu, te se pitanjima dodjeljuju bodovi. Postavlja se broj pitanja zajedno sa samim prvim pitanjem i odgovorima pitanja te slika ako postoji na *layout* aktivnosti, tipka za listanje pitanja unazad se poništava pošto se korisniku postavlja prvo pitanje. Kada se sve postavi, vrijeme ispita se pokreće u trajanju od 45 minuta te

korisnik može započeti rješavanje ispita. Korisnik lista pitanja pomoću tipki „Prethodno“ i „Sljedeće“ te pomoću *navigationDrawer* izbornika unutar kojih je lista svih pitanja. Unutar navedenog izbornika svako pitanje koje je korisnik već pregledao mijenja boju u žutu, dok pitanja koja nisu pregledana imaju boju početnih postavki. Ako je korisnik na zadnjem pitanju, poništava se tipka „Sljedeće“, a ako je na prvom pitanju poništava se tipka „Prethodno“. Kada korisnik odluči predati ispit na pregled, može pritisnuti tipku „Završi ispit“. Tada ga aplikacija pita dali je siguran da želi predati ispit te korisnik mora potvrditi predaju. Ako vrijeme istekne, automatski se ispit predaje na pregled. Kada se ispit pregleda korisnika se vodi na novu aktivnost od klase *ProlaznostIspit.java* koja je također vidljiva na slici 59 gdje se korisniku prikazuju informacije o prolaznosti riješenog ispita. Kada korisnik pregleda informacije, može pritisnuti tipku „Izlaz“ koja ga vodi nazad na glavni izbornik.



Slika 59. Ispit u aplikaciji

Programsko rješenje sa slike 60 prikazuje *onClickListenere* na tipkama za odgovore, te na tipkama za završavanje i listanje pitanja. Na *onClickListenerima* odgovora se nalazi

funkcija *oznacavanjeOdgovora()* kao i u klasi *VjezbePitanja.java*. Na *onClickListenerima* tipki za listanje pitanja se poništavaju tipke za listanje pitanja i završetak ispita, te se pokreče funkcija *promjeniPitanje()* unutar koje se navedene tipke ponovno omogućuju za korištenje, ali nakon što se pitanje učita. *OnClickListener* tipke za završetak ispita kreira *dialog box* za potvrdu predaje ispita. Pokretanjem aktivnosti ispita, prvo se pokreće funkcija *getListaPitanja().*

```
btnOdgovor1Ispit.setOnClickListener((v) → { oznacavanjeOdgovora( oznaceniOdgovor: "1"); });

btnOdgovor2Ispit.setOnClickListener((v) → { oznacavanjeOdgovora( oznaceniOdgovor: "2"); });

btnOdgovor3Ispit.setOnClickListener((v) → { oznacavanjeOdgovora( oznaceniOdgovor: "3"); });

btnPrethodnoPitanje.setOnClickListener((v) → {
        btnPrethodnoPitanje.setEnabled(false);
        btnSljedecePitanje.setEnabled(false);
        btnZavrsiIspit.setEnabled(false);
        promjeniPitanje( dugme: true);
});

btnSljedecePitanje.setOnClickListener((v) → {
        btnPrethodnoPitanje.setEnabled(false);
        btnSljedecePitanje.setEnabled(false);
        btnZavrsiIspit.setEnabled(false);
        promjeniPitanje( dugme: false);
});

btnZavrsiIspit.setOnClickListener((v) → {
        AlertDialog.Builder builder = new AlertDialog.Builder(v.getContext());
        builder.setMessage("Jeste li sigurni da želite završiti ispit?").setPositiveButton( text: "Da", dialogClickListener)
                .setNegativeButton( text: "Ne", dialogClickListener).show();
});

getListaPitanja();
```

Slika 60. Programski kod *Ispit.java* klase

Slika 61 prikazuje kreirane funkcije unutar *Ispit.java* klase koje služe za kreiranje, rješavanje, ispravljanje i spremanje ispita. Funkcije su uglavnom iste kao i u *VjezbePitanja.java* klasi uz dodatak par novih funkcija. Također se koristi funkcija *headerUcitavanje()* kao i u glavnom izborniku, za učitavanje korisničkih podataka s Cloud Firestore-a na *header* od *navigationDrawera* trenutne aktivnosti.Ispit.java ima dva *dialog boxa* koji su *onClickListeneri.* Jedan se koristi kod prije navedene tipke za završetak ispita, a drugi se koristi kao i kod *vjezbePitanja.java* klase, samo što se sami *onClickListener* nalazi van nadjačane klase *onBackPressed().*

```
private void getListaPitanja() {...}

private void setPitanje() {...}

private void promjeniPitanje(boolean dugme) {...}

private void playAnim(final View view, final int value, final int viewBroj) {...}

private void startTimer() {...}

private void oznacavanjeOdgovora(String oznaceniOdgovor) {...}

private void ucitavanjeOdgovora() {...}

private void zavrsetakIspita() {...}

DialogInterface.OnClickListener dialogClickListener = (dialog, which) -> {
        switch (which) {
            case DialogInterface.BUTTON_POSITIVE:
                provjeraOdgovora();
                zavrsetakIspita();
                break;

            case DialogInterface.BUTTON_NEGATIVE:
                break;
        }
};

DialogInterface.OnClickListener dialogClickListener2 = (dialog, which) -> {
        switch (which) {
            case DialogInterface.BUTTON_POSITIVE:
                countDownTimer.cancel();
                Intent intent = new Intent( packageContext: Ispit.this, GlavniIzbornik.class);
                startActivity(intent);
                overridePendingTransition(R.anim.slide_in_left, R.anim.slide_out_right);
                break;

            case DialogInterface.BUTTON_NEGATIVE:
                break;
        }
};

private void provjeraOdgovora() {...}

private void headerUcitavanje() {
    DocumentReference documentReference = fStore.collection( collectionPath: "users").document(fAuth.getCurrentUser().getUid());
    documentReference.addSnapshotListener( activity: this, (documentSnapshot, e) -> {
            txtHeaderImePrezime.setText(documentSnapshot != null ? documentSnapshot.getString( field: "ImePrezime") : null);
            txtHeaderEmail.setText(documentSnapshot != null ? documentSnapshot.getString( field: "Email") : null);
    });
}
```

Slika 61. Funkcije *Ispit.java* klase

Funkcija *getListaPitanja()* radi na gotovo identičan način kao i istoimena funkcija iz *VjezbePitanja.java* klase. Razlika je u tome da unutar *Ispit.java* klase funkcija nabavlja pitanja iz više setova. Uz to kreira i dodatnu listu s vrijednostima odgovora (slika 62) u

koju se pohranjuju početne vrijednosti odgovora na pitanju te bodovi za pitanje. Lista s vrijednostima odgovora je jednake veličine kao i lista s pitanjima, te primjerice deseto pitanje u listi pitanja ima desetu vrijednost odgovora u listi odgovora. Nakon učitavanja pitanja poziva se funkcija *setPitanje()* koja radi isto kao funkcija iz *VjezbePitanja()* klase, ali još poništava tipku „Prethodno“.

```
DocumentSnapshot snapshot = task.getResult();
if (finalI < 14) {
    Pitanje pitanje = new Pitanje( Pitanje: snapshot.getString( field: "Pitanje") + " (2 boda)",
            snapshot.getString( field: "Odgovor1"), snapshot.getString( field: "Odgovor2"),
            snapshot.getString( field: "Odgovor3"), snapshot.getLong( field: "TocanOdgovor"),
            snapshot.getString( field: "Slika"));
    ListaPitanja.add(pitanje);
    ListaOdgovora.add(new OznaceniOdgovoriIspit( oznakaOdgovor1: "0",
            oznakaOdgovor2: "0",
            oznakaOdgovor3: "0",
            brojBodova: 2));
```

Slika 62. Dio programskog koda funkcije *getListaPitanja()*

Funkcija *startTimer()* sa slike 63 se poziva unutar funkcije *setPitanje().* Kada se pozove, kreira se tajmer koji je postavljen na 45 minuta, uz interval od jedne sekunde, te kada vrijeme istekne pozivaju se funkcije *provjeraOdgovora()* i *zavrsetakIspita()*. Zatim se tajmer pokreče.

```
countDownTimer = new CountDownTimer( millisInFuture: 2700000,  countDownInterval: 1000) {
    @Override
    public void onTick(long millisUntilFinished) {
        txtVrijemeIspit.setText("" + String.format(FORMAT,
                TimeUnit.MILLISECONDS.toMinutes(millisUntilFinished) - TimeUnit.HOURS.toMinutes(
                        TimeUnit.MILLISECONDS.toHours(millisUntilFinished)),
                TimeUnit.MILLISECONDS.toSeconds(millisUntilFinished) - TimeUnit.MINUTES.toSeconds(
                        TimeUnit.MILLISECONDS.toMinutes(millisUntilFinished))));
    }
    @Override
    public void onFinish() {
        provjeraOdgovora();
        zavrsetakIspita();
    }
};
countDownTimer.start();
```

Slika 63. Programski kod funkcije *startTimer()*

Programsko rješenje funkcije *oznacavanjeOdgovora()* je slično istoimenoj funkciji iz *VjezbePitanja.java* klase (slika 64). Prima samo 1 umjesto 2 argumenta i umjesto da se

za svako pitanje resetiraju vrijednosti označenosti odgovora kao u *VjezbePitanja.java* kada korisnik promjeni pitanje, ovdje se vrijednosti označenosti odgovora pohranjuju u prije spomenutu listu vrijednosti odgovora tako da korisnik kada se vrati na pitanje gdje je već davao odgovore, može vidjeti koje je odgovore dao i izmijeniti ih.

```
if (oznaceniOdgovor.equals("1")) {
    if (ListaOdgovora.get(redniBrojPitanja).getOznakaOdgovor1().equals("0")) {
        ListaOdgovora.get(redniBrojPitanja).setOznakaOdgovor1("1");
        btnOdgovor1Ispit.setBackgroundTintList(ColorStateList.valueOf(Color.parseColor( colorString: "#ffcc00")));
        int img = R.drawable.oznaceno_icon;
        btnOdgovor1Ispit.setCompoundDrawablesWithIntrinsicBounds(img,  top: 0,  right: 0,  bottom: 0);
    } else {
        ListaOdgovora.get(redniBrojPitanja).setOznakaOdgovor1("0");
        btnOdgovor1Ispit.setBackgroundTintList(ColorStateList.valueOf(Color.parseColor( colorString: "#132131")));
        int img = R.drawable.neoznaceno_icon;
        btnOdgovor1Ispit.setCompoundDrawablesWithIntrinsicBounds(img,  top: 0,  right: 0,  bottom: 0);
    }
```

Slika 64. Dio programskog koda funkcije *oznacavanjeOdgovora()*

```
for (int i = 0; i < ListaOdgovora.size(); i++) {
    OznaceniOdgovor = 0;

    if (ListaOdgovora.get(i).getOznakaOdgovor1().equals("1")) {
        OznaceniOdgovor = 1;
    }
    if (ListaOdgovora.get(i).getOznakaOdgovor2().equals("1")) {
        if (OznaceniOdgovor == 1) {
            OznaceniOdgovor = 12;
        } else {
            OznaceniOdgovor = 2;
        }
    }
    if (ListaOdgovora.get(i).getOznakaOdgovor3().equals("1")) {
        if (OznaceniOdgovor == 1) {
            OznaceniOdgovor = 13;
        }
        if (OznaceniOdgovor == 12) {
            OznaceniOdgovor = 123;
        }
        if (OznaceniOdgovor == 2) {
            OznaceniOdgovor = 23;
        }
        if (OznaceniOdgovor == 0) {
            OznaceniOdgovor = 3;
        }
    }

    if (ListaPitanja.get(i).getTocanOdgovor() == OznaceniOdgovor) {
        BrojBodova = BrojBodova + ListaOdgovora.get(i).getBrojBodova();
        if (ListaOdgovora.get(i).getBrojBodova() == 7) {
            raskrizja++;
        }
    }
```

Slika 65. Dio programskog koda funkcije *provjeraOdgovora()*

Programsko rješenje sa slike 65 prikazuje dio funkcije *provjeraOdgovora().* Funkcija se poziva prilikom završetka ispita te je njena svrha izračun broja bodova riješenog ispita. Broj bodova se računa pomoću *for* petlje tako što se svakom pitanju odredi varijabla *OznaceniOdgovor* za koju se koristi lista vrijednosti odgovora. Ta varijabla se uspoređuje s varijablom točnog odgovora za pitanje koje se ispravlja, te ako su varijable iste, znači da je odgovor točan. Bodovi se spremaju u varijablu koja služi kao ukupan zbroj bodova. Ako se ispravlja pitanje koje ima 7 bodova, varijabla *raskrizja* raste za 1. Korisnik mora odgovoriti na sva 4 pitanja iz raskrižja točno što znači da varijabla *raskrizja* mora imati vrijednost 4 na kraju ispravka ispita kako bi korisnik mogao proći ispit. Uz sva točno riješena raskrižja, korisnik mora imati više od 107 bodova za prolaz. U ostatku programskog rješenja funkcije se kreiraju varijable koje će se koristiti unutar funkcije *zavrsetakIspita()* za postavljanje na *Cloud Firestore*.

Dio programskog rješenja od funkcije *promjeniPitanje()* vidljivog na slici 66 ima sličnosti s istoimenom funkcijom iz *VjezbePitanja.java* klase. Funkcija u trenutnoj aktivnosti ima 1 argument tipa *boolean*[30] za razliku od funkcije iz *VjezbePitanja.java* klase gdje funkcija nema argumente. *Boolean* argument se koristi u funkciji tako da određuje dali se treba korisniku učitati prethodno ili sljedeće pitanje u listi. Funkcija se poziva pomoću tipki za listanje pitanja „Prethodno“ i „Sljedeće“, te pomoću tipki koje se nude unutar *navigationDrawer* izbornika koji ima listu svih pitanja. Kada se funkcija pozove i argument je istinit, funkcija pokreče *dialog box* učitavanja te smanjuje varijablu rednog broja pitanja za 1. Mijenja se boja tipke pitanja koje se učitava na *navigationDraweru*, te se označava tipka pitanja na istom. Mijenja se tekst broja pitanja na *layoutu* od aktivnosti te se kao i u *VjezbePitanja.java* klasi poziva funkcija *playAnim()* na pitanje i prva dva odgovora i prema potrebi na treći odgovor i sliku. U slučaju da korisnik ide s drugog pitanja na prvo pomoću tipke „Prethodno“, pitanje će se promijeniti, ali nakon toga će se poništiti tipka „Prethodno“ kako ju korisnik više ne bi mogao koristiti. Tipka se ponovno omogućava kada korisnik prelista na sljedeće pitanje. U ostatku programskog rješenja funkcija na isti način mijenja pitanje, ali ga mijenja na sljedeće pitanje koje ne mora nužno biti za 1 veće. Razlog toga je što svaka tipka *navigationDrawera* ima poziv na funkciju, ali prije poziva mijenja

[30] Boolean vrijednost je vrijednost s dva izbora: istinitim ili lažnim.

varijablu rednog broja pitanja na broj pitanja koje predstavlja tipka. Ako funkcija dođe do zadnjeg pitanja u listi, poništava tipku „Sljedeće“ sve dok korisnik ne promijeni pitanje.

```
if (dugme) {
    if (redniBrojPitanja > 0) {
        ucitavanje.pokreniUcitavanje();
        redniBrojPitanja--;

        navigationView.getMenu().getItem(redniBrojPitanja).setChecked(true);
        navigationView.getMenu().getItem(redniBrojPitanja).setTitle(Html
                .fromHtml( source: "<font color='#ffcc00'>" + (redniBrojPitanja + 1) + ". pitanje</font>"));

        txtBrojPitanjaIspit.setText("Pitanje: " + (redniBrojPitanja + 1) + "/" + ListaPitanja.size());
        playAnim(txtTekstPitanjaIspit,  value: 0,  viewBroj: 0);
        playAnim(btnOdgovor1Ispit,  value: 0,  viewBroj: 1);
        playAnim(btnOdgovor2Ispit,  value: 0,  viewBroj: 2);
        if (ListaPitanja.get(redniBrojPitanja).getOdgovor3().equals("0")) {
            btnOdgovor3Ispit.setVisibility(View.GONE);
            btnOdgovor3Ispit.setEnabled(false);
        } else {
            btnOdgovor3Ispit.setEnabled(true);
            btnOdgovor3Ispit.setVisibility(View.VISIBLE);
            playAnim(btnOdgovor3Ispit,  value: 0,  viewBroj: 3);
        }

        if (!ListaPitanja.get(redniBrojPitanja).getSlika().equals("")) {

            imgSlikaIspit.setVisibility(View.VISIBLE);

            Picasso.get().load(ListaPitanja.get(redniBrojPitanja).getSlika()).into(imgSlikaIspit, new com.squareup.picasso.Callback() {
                @Override
                public void onSuccess() {
                    ucitavanje.makniUcitavanje();
                    btnPrethodnoPitanje.setEnabled(true);
                    btnSljedecePitanje.setEnabled(true);
                    btnZavrsiIspit.setEnabled(true);
                }

                @Override
                public void onError(Exception e) {

                }
            });
        } else {
            imgSlikaIspit.setVisibility(View.GONE);
            ucitavanje.makniUcitavanje();
            btnPrethodnoPitanje.setEnabled(true);
            btnSljedecePitanje.setEnabled(true);
            btnZavrsiIspit.setEnabled(true);
        }

        if (redniBrojPitanja == 0) {
            btnPrethodnoPitanje.setEnabled(false);
        }
    }
```

Slika 66. Dio programskog koda funkcije *promjeniPitanje()*

Funkcija *playAnim()* je gotovo identična kao i ista funkcija iz klase *VjezbePitanja()*. Razlika je što se u trenutnoj funkciji ne resetiraju boje tipki odgovora na početne postavke i koristi

se po završetku animacije *withEndAction*[31] koji služi za poziv funkcije *ucitavanjeOdgovora()*.

```
if (ListaOdgovora.get(redniBrojPitanja).getOznakaOdgovor1().equals("1")) {
    btnOdgovor1Ispit.setBackgroundTintList(ColorStateList.valueOf(Color.parseColor( colorString: "#ffcc00")));
    int img = R.drawable.oznaceno_icon;
    btnOdgovor1Ispit.setCompoundDrawablesWithIntrinsicBounds(img,  top: 0,  right: 0,  bottom: 0);
} else {
    btnOdgovor1Ispit.setBackgroundTintList(ColorStateList.valueOf(Color.parseColor( colorString: "#132131")));
    int img = R.drawable.neoznaceno_icon;
    btnOdgovor1Ispit.setCompoundDrawablesWithIntrinsicBounds(img,  top: 0,  right: 0,  bottom: 0);
}
```

Slika 67. Dio programskog koda funkcije *ucitavanjeOdgovora()*

```
if (!task.getResult().exists()) {
    Map<String, Integer> rezultatBroj = new HashMap<>();
    rezultatBroj.put("Broj", 1);
    documentReference.set(rezultatBroj);
} else {
    brojRjesenihIspita = task.getResult().getLong( field: "Broj").intValue() + 1;
}


Map<String, String> brojIspita = new HashMap<>();

DocumentReference documentReference1 = fStore.collection( collectionPath: "users")
        .document(userID).collection( collectionPath: "RjeseniIspiti")
        .document( documentPath: "Ispit" + brojRjesenihIspita);

brojIspita.put("Naziv", brojRjesenihIspita + ". ispit");
brojIspita.put("BrojIspita", "Ispit" + brojRjesenihIspita);
documentReference1.set(brojIspita);

Map<String, Object> rezultat = new HashMap<>();

for (int i = 0; i < 38; i++) {
    documentReference1 = fStore.collection( collectionPath: "users") CollectionReference
            .document(userID).collection( collectionPath: "RjeseniIspiti")
            .document( documentPath: "Ispit" + brojRjesenihIspita) DocumentReference
            .collection( collectionPath: "PitanjaIOdgovori") CollectionReference
            .document( documentPath: "PitanjeIOdgovori" + (i + 1));

    rezultat.put("Pitanje", ListaPitanja.get(i).getPitanje());
    rezultat.put("Odgovor1", ListaPitanja.get(i).getOdgovor1());
    rezultat.put("Odgovor2", ListaPitanja.get(i).getOdgovor2());
    rezultat.put("Odgovor3", ListaPitanja.get(i).getOdgovor3());
    rezultat.put("TocanOdgovor", ListaPitanja.get(i).getTocanOdgovor());
    rezultat.put("Slika", ListaPitanja.get(i).getSlika());
    rezultat.put("OznakaOdgovor1", ListaOdgovora.get(i).getOznakaOdgovor1());
    rezultat.put("OznakaOdgovor2", ListaOdgovora.get(i).getOznakaOdgovor2());
    rezultat.put("OznakaOdgovor3", ListaOdgovora.get(i).getOznakaOdgovor3());
    rezultat.put("BrojBodova", ListaOdgovora.get(i).getBrojBodova());
    rezultat.put("RedniBroj", i + 1);

    documentReference1.set(rezultat);
    rezultat.clear();
```

Slika 68. Dio programskog koda funkcije *zavrsetakIspita()*

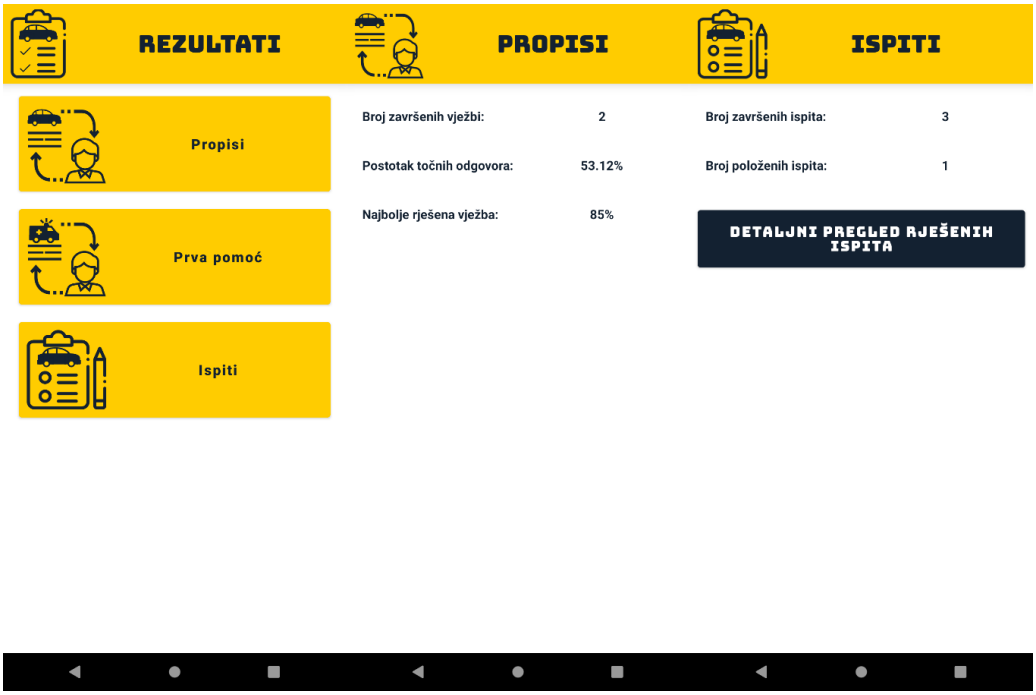
[31] Akcija koja se izvodi kad se animacija završi.

Slika 67 prikazuje dio programskog koda funkcije *ucitavanjeOdgovora()* koja se poziva unutar funkcije *playAnim()*. To znači da svaki puta kada se pitanje mijenja da se funkcija pozove. Funkcija radi na sličan način kao funkcija *oznacavanjePitanja()*, ali se koristi za učitavanje vrijednosti odgovora iz liste odgovora na *layout* trenutne aktivnosti. Svrha funkcije je učitavanje odgovora od prijašnje odgovorenog pitanja. Time korisnik može vidjeti koje odgovore je označio, te može promijeniti svoje odgovore.

Programsko rješenje sa slike 68 prikazuje funkciju *zavrsetakIspita()* koja se poziva pri završetku ispita. Svrha funkcije je spremanje rezultata riješenog ispita na *Cloud Firestore*. Prvo se na *Cloud Firestore*-u provjeri dali postoji datoteka s brojem riješenih ispita, te ako ne postoji kreira se. Zatim se ažurira broj ispita na *Cloud Firestore*-u. Kada se ažurira, funkcija pomoću *for* petlje kreira novu kolekciju riješenog ispita na *Cloud Firestore*-u i sprema riješena pitanja skupa s korisnikovim odgovorima jedno po jedno. U ostatku programskog rješenja funkcije se spremaju informacije o prolaznosti ispita na *Cloud Firestore*, te se otvara aktivnost od klase *ProlaznostIspit.java* gdje se korisniku prikazuju iste informacije.

6.3.8. Rezultati u aplikaciji

Kada korisnik odluči pregledati rezultate riješenih vježbi i ispita, pritišće tipku „Rezultati“ na aktivnosti od glavnog izbornika. Aplikacija otvara aktivnost (slika 69) gdje korisnik bira pregled rezultata određene kategorije ili ispita. Ako korisnik odabere jednu od kategorija vježbi, aplikacija mu ispisuje koliko je riješio vježbi iz odabrane kategorije, te postotak točnih odgovora kroz sve riješene odabrane vježbe. Uz to ispisuje i postotak točnih odgovora najbolje riješene vježbe. U slučaju da korisnik odabere pregled riješenih ispita, aplikacija ispisuje broj riješenih ispita, te broj položenih ispita. Kada korisnik riješi barem jedan ispit, prikazuje mu se tipka za detaljan pregled ispita. Pritiskom na tu tipku se otvaraju aktivnosti sa slike 70. Prvo se ponude svi riješeni ispiti koji su poredani prema datumu kada su riješeni, od najstarijeg datuma prema najranijem. Kada korisnik odabere ispit koji želi detaljno pregledati, aplikacija otvara aktivnost na kojoj se ispisuju sva pitanja iz ispita s označenim odgovorima od strane korisnika. Pitanja su označena zelenom bojom ako su točno odgovorena, crvenom ako su netočno odgovorena. Iznad pitanja se nalazi tipka za skrivanje i prikazivanje informacija o prolaznosti riješenog ispita.



Slika 69. Rezultati u aplikaciji

Slika 70. Detaljni rezultati riješenih ispita

Programsko rješenje sa slike 71 prikazuje klasu *DetaljnoRezultati.java* od aktivnosti gdje se prikazuju rezultati vježbi ili ispita, ovisno o odabiru korisnika na prijašnjoj aktivnosti. Klasa sadrži 3 funkcije od kojih se pokreće jedna ovisna o odabiru korisnika. Funkcije

učitavaju informacije o vježbama ili ispitima s *Cloud Firestore*-a te ih ispisuju na *layout* trenutne aktivnosti. Također se nalazi i *onClickListener* na tipki za detaljan pregled ispita. Navedena tipka se prikazuje u slučaju da korisnik pregledava riješene ispite, a ima barem jedan riješen ispit. Pritiskom na tipku se mijenja aktivnost.

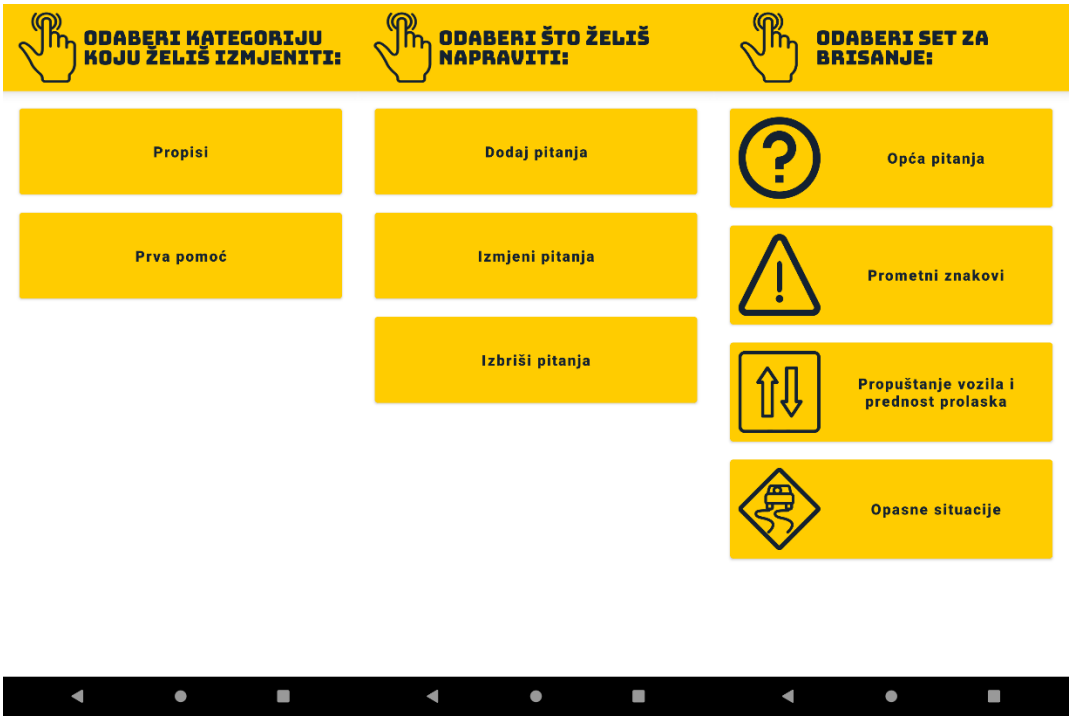
```
    btnIspitDetaljnoRezultati.setOnClickListener((v) → {
            Intent intent = new Intent( packageContext: DetaljnoRezultati.this, ListaIspita.class);
            startActivity(intent);
    });

    if(Objects.equals(getIntent().getStringExtra( name: "KATEGORIJA"), b: "Propisi")){
        imgSlikaDetaljnoRezultati.setImageResource(R.drawable.vjezba_propisi_icon);
        getPropisiRezultati();
    }

    if(Objects.equals(getIntent().getStringExtra( name: "KATEGORIJA"), b: "Prva pomoć")){
        imgSlikaDetaljnoRezultati.setImageResource(R.drawable.vjezba_prva_pomoc_icon);
        getPrvaPomocRezultati();
    }

    if(Objects.equals(getIntent().getStringExtra( name: "KATEGORIJA"), b: "Ispiti")){
        imgSlikaDetaljnoRezultati.setImageResource(R.drawable.ispit_propisi_icon);
        getIspitiRezultati();
    }
}

private void getPropisiRezultati() {...}

private void getPrvaPomocRezultati() {...}

private void getIspitiRezultati() {...}
```

Slika 71. Programski kod *DetljanoRezultati.java* klase

U detaljnom pregledu ispita u klasi *ListaIspita.java* se koristi *recyclerView* adapter za učitavanje svih riješenih ispita na *layout* aktivnosti. Svaki riješeni ispit je tipka sa *onClickListenerom* koji vodi na sljedeću aktivnost od klase *IspitiRezultati.java* na kojoj se nalazi još jedan *recyclerView* adapter. Navedeni adapter učitava pitanja i odgovore iz odabranog riješenog ispita na *layout* aktivnosti, te učitava informacije o prolaznosti ispita koje je moguće sakriti pomoću tipke za smanjivanje detalja o ispitu. Ista tipka služi i za prikazivanje informacija o prolaznosti ispita.

6.3.9. Administracijski dio aplikacije

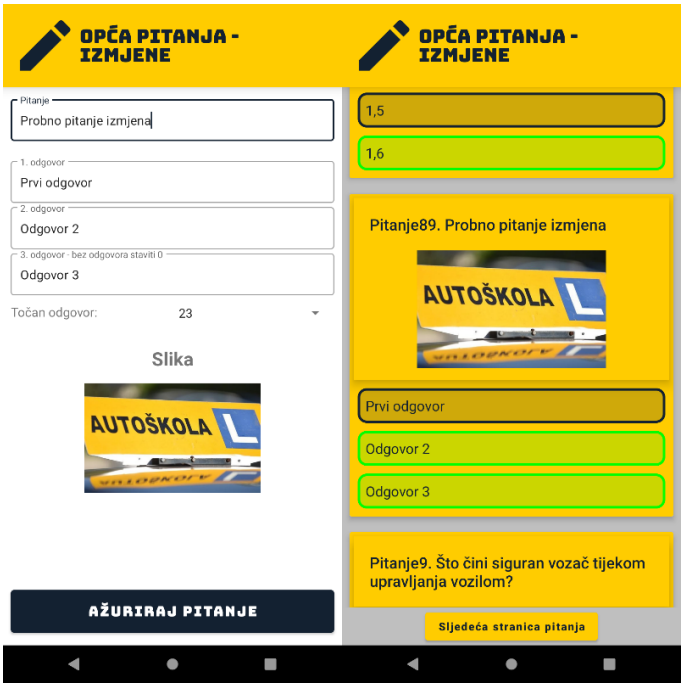
Administrator je korisnik koji ima dodatnu opciju u *navigationView-u* glavnog izbornika. Tom opcijom pristupa administracijskim izbornicima (slika 72) s kojima odabire kategoriju pitanja koju želi uređivati. Također odabire vrstu uređivanja i set pitanja koji želi urediti.



Slika 72. Administracijski izbornici u aplikaciji

U slučaju da administrator odabere opciju za dodavanje pitanja i odabere set u koji želi dodati pitanje, otvara se aktivnost (slika 73) u kojoj se nalazi upitnik za popunjavanje pitanja. Korisnik mora popuniti sva polja osim slike koja je opcionalna. Kada administrator popuni upitnik ispravno i stisne tipku za dodavanje pitanja, pitanje se sprema na *Cloud Firestore*, a aplikacija ispiše obavijest da je pitanje dodano i otvori aktivnost za izmjenu pitanja u slučaju da administrator hoće izmijeniti pitanja u odabranom setu. Pitanje je odmah vidljivo administratoru u *recyclerView* adapteru koji sadrži sva pitanja.

Slika 73. Dodavanje pitanja u aplikaciji

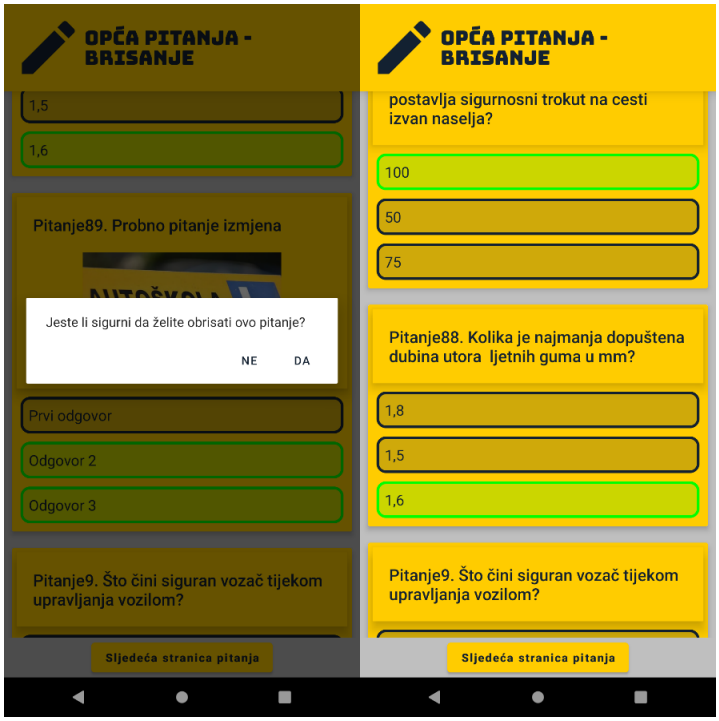


Slika 74. Izmjena pitanja u aplikaciji

Kada administrator odabere opciju izmjena i odabere set, otvara se aktivnost popunjena pitanjima odabranog seta koja se nalaze unutar *recyclerView* adaptera. Vidljivo je do 20

pitanja na aktivnosti, ali administrator za dolazak do ostalih pitanja može koristiti tipku za listanje stranica pitanja. Kada administrator odabere pitanje (slika 74), otvara se isti upitnik za popunjavanje pitanja kao i kod dodavanja pitanja, ali upitnik je popunjen s podatcima pitanja koje je odabrano za izmjenu. Administrator tada može izmijeniti pitanje, te kada je upitnik popunio ispravno, tipka za ažuriranje pitanja sprema ažurirano pitanje na *Cloud Firestore*. Administratoru dolazi obavijest da je pitanje ažurirano te je odmah vidljiva promjena na pitanju.

Ako administrator odabere opciju za brisanje pitanja i potom set pitanja iz kojeg želi brisati pitanja, otvara se aktivnost kao i kod izmjene pitanja. Aktivnost sadrži pitanja u *recyclerView* adapteru samo što administrator na pritisak na pitanje sada briše pitanja. Također se kroz pitanja lista s tipkom jer ih je do 20 na aktivnosti odjednom. Kada administrator pritisne na pitanje, pojavi se *dialog box* koji traži potvrdu za brisanje pitanja(slika 75) kako se ne bi slučajno brisala pitanja. Kada administrator potvrdi brisanje, aplikacija šalje odabrano pitanje koje želi obrisati na *Cloud Firestore*, te ispisuje obavijest da je pitanje obrisano, što je odmah vidljivo.



Slika 75. Brisanje pitanja u aplikaciji

Slika 76 prikazuje funkcije koje se koriste unutar *EditiranjePitanja.java* klase koja se koristi za odabir pitanja kako bi se izmijenila ili obrisala.

```
@Override
public void onPitanjeClickListener(final int position) {...}

private void ucitavanjeBroja(final String getName) {...}

private void updateanjeBroja(final String broj, final String getName) {...}

private void ucitavanje(final String token, boolean Delete, final String Ime, final String brojPitanja) {...}

private void premjestanjePodataka(final Pitanje pitanjeTemp, final String getName, final String brojPitanja) {...}

private void brisanjePitanja(final String brojPitanja, String Slika) {...}
```

Slika 76. Funkcije *EditiranjePitanja.java* klase

Funkcija *onPitanjeClickListener()* se poziva kada administrator pritisne na jedno od pitanja. Ako je administrator odabrao izmjenu pitanja, otvara se aktivnost gdje se nalazi upitnik za izmjenu pitanja, a u slučaju da je administrator odabrao brisanje pitanja u prethodnoj aktivnosti, pokreće se *dialog box* koji pita administratora za potvrdu brisanja pitanja.

Ako administrator potvrdi da želi obrisati pitanje poziva se funkcija *ucitavanjeBroja()* čiji je programski kod funkcije na slici 77. Funkcija prvo nabavlja *IdToken* administratora koji se koristi u REST api web servisu. Kada se *IdToken* dobavi sastavlja se *URL* koji se koristi za nabavu broja pitanja odabranog seta s *Cloud Firestore*-a. Na varijablu S*tring* „auth“ se stavlja *IdToken* administratora s nazivom „Bearer“, te se koristi za autorizaciju. Varijabla „client“ ovisnosti *okhttp3:okhttp* se koristi za rađenje zahtjeva na *Cloud Firestore*. Prije nego navedena varijabla radi zahtjev, kreira se varijabla za zahtjev u koju se stavlja u *header* varijabla „auth“, a potom *URL.* Kada se zahtjev uspješno izvrši, vraća se odgovor unutar kojeg se nalazi *JSON* datoteka s brojem pitanja odabranog seta. Zatim se zove funkcija *updateanjeBroja()* koja uzima nabavljeni broj kao argument uz *String* ID-a pitanja za brisanje.

```
fAuth.getCurrentUser().getIdToken( b: true).addOnCompleteListener((task) → {
        if (task.isSuccessful()) {

            String Kategorija = getIntent().getStringExtra( name: "IDKATEGORIJE");
            String IDseta = getIntent().getStringExtra( name: "IDSETA");

            String url = "https://firestore.googleapis.com/v1/projects/autoskolav1/databases/(default)/documents/Pitanja/"
                    + Kategorija + "/BrojPitanja/" + IDseta;
            if(!getIntent().getStringExtra( name: "IDDETALJNO").equals("0")){
                url = "https://firestore.googleapis.com/v1/projects/autoskolav1/databases/(default)/documents/Pitanja/"
                        + Kategorija + "/BrojPitanja/" + IDseta + "/Pitanja/" + getIntent().getStringExtra( name: "IDDETALJNO");
            }

            String auth = "Bearer " + task.getResult().getToken();


            OkHttpClient client = new OkHttpClient();

            final Request request = new Request.Builder().addHeader( name: "Authorization", auth).url(url).build();

            client.newCall(request).enqueue(new Callback() {
                @Override
                public void onFailure(@NotNull Call call, @NotNull IOException e) {
                    Toast.makeText( context: EditiranjePitanja.this, e.getMessage(), Toast.LENGTH_SHORT).show();
                }

                @Override
                public void onResponse(@NotNull Call call, @NotNull Response response) throws IOException {
                    if(response.isSuccessful()){
                        final String myResponse = Objects.requireNonNull(response.body()).string();

                        EditiranjePitanja.this.runOnUiThread(() → {
                                try {
                                    brojPitanja = new JSONObject(myResponse).getJSONObject("fields")
                                            .getJSONObject("BrojPitanja")
                                            .getString( name: "integerValue");
                                    updateanjeBroja(brojPitanja, getName);

                                }catch (JSONException err){
                                    Toast.makeText( context: EditiranjePitanja.this, err.getMessage(), Toast.LENGTH_SHORT).show();
                                }
                        });
                    }
                }
            });

        }
});
```

Slika 77. Programski kod funkcije *ucitavanjeBroja()*

Slika 78 prikazuje dio programskog rješenja iz funkcije *updateanjeBroja()* koja se poziva iz prethodno objašnjene funkcije. Unutar trenutne funkcije se kao i u prethodnoj kreira *URL*[32] i *String* „auth“, ali ovdje se argument broj prebacuje u *int* varijablu, te se kreira *String* varijabla „json“ s vrijednosti broja pitanja umanjene za 1. Zatim se kreira

[32] URL je referenca na web resurs koji određuje njegovo mjesto na računalnoj mreži i mehanizam za njegovo dohvaćanje.

requestBody varijabla koja uzima *String* „json“ i određuje mu format *JSON*[33] datoteke. Zatim se kreira zahtjev u koji se kao i u prethodnoj funkciji dodaju *header* i *URL,* ali i *patch* unutar kojega se stavlja *requestBody*. *Patch* je objašnjen na slici 7. Nakon što „client“ varijabla uspješno napravi zahtjev na *Cloud Firestore*, vraća se odgovor pomoću kojega se zna da je broj pitanja ažuriran.

```
final int temp = Integer.parseInt(broj);

String json = "{\n" +
        "  \"fields\": {\n" +
        "    \"BrojPitanja\": {\n" +
        "      \"integerValue\": \"" + (temp - 1) + "\"\n" +
        "    }\n" +
        "  }\n" +
        "}";

RequestBody requestBody = RequestBody.create(json, JSON);

OkHttpClient client = new OkHttpClient();

final Request request = new Request.Builder().addHeader( name: "Authorization", auth)
        .addHeader( name: "Accept",  value: "application/json")
        .addHeader( name: "Content-Type",  value: "application/json")
        .patch(requestBody).url(url).build();
```

Slika 78. Dio programskog koda funkcije *updateanjeBroja()*

Zatim se poziva funkcija *ucitavanje().* Ta funkcija ima dvije svrhe. Prva je da učita sva pitanja iz odabranog seta iz *JSON* datoteke na isti način kako se učitavao broj pitanja seta te postavi pitanja u *recyclerView* adapter i koristi dobiveni „nextPageToken“ iz *JSON* datoteke kako bi administrator mogao listati pitanja pomoću tipke za listanje. Druga svrha je učitavanje posljednjeg pitanja u odabranom setu, te pozivanje funkcije *premjestanjePodataka()* s argumentom koji sadrži učitano posljednje pitanje. Funkcija *premjestanjePodataka()* sastavlja *JSON* datoteku od argumenta koje je posljednje pitanje u setu, te radi zahtjev gdje *JSON* datoteku stavlja na pitanje koje je označeno za brisanje. Zatim se poziva funkcija *brisanjePitanja().*

[33] JSON je otvoreni standardni format datoteke i format razmjene podataka, koji koristi tekst čitljiv za pohranu i prijenos podataka koji se sastoje od parova atributa, vrijednosti i nizova podataka

```
Request request = new Request.Builder().addHeader( name: "Authorization", auth).addHeader( name: "Accept",  value: "application/json").delete().url(url).build();
client.newCall(request).enqueue(new Callback() {
    @Override
    public void onFailure(@NotNull Call call, @NotNull IOException e) {
        Toast.makeText( context: EditiranjePitanja.this, e.getMessage(), Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onResponse(@NotNull Call call, @NotNull Response response) throws IOException {
        if(response.isSuccessful()){
            EditiranjePitanja.this.runOnUiThread(() -> {
                    Toast.makeText( context: EditiranjePitanja.this,  text: "Uspjesno obrisano!", Toast.LENGTH_SHORT).show();
            });
        }
    }
});
```

Slika 79. Dio programskog koda funkcije *brisanjePitanja()*

Funkcija *brisanjePitanja()* sastavlja *URL* na posljednje pitanje u setu koje je premješteno na označeno pitanje za brisanje, te se kreiraju sve potrebne varijable za rađenje zahtjeva. Zahtjev za brisanje pitanja (slika 79) koristi *delete()* kao oznaku na zahtjevu da se sadržaj s *URL*-a treba obrisati. Kada se pitanje obriše vraća se odgovor te se ispisuje obavijest administratoru da je pitanje uspješno obrisano.

```
    if(Objects.equals(getIntent().getStringExtra( name: "ADMIN"),  b: "Izmjene")){...}

    btnPotvrdi.setOnClickListener((v) -> {
            if(Objects.equals(getIntent().getStringExtra( name: "ADMIN"),  b: "Izmjene")){
                izmjenaPitanja();
            }else{
                ucitavanjeBroja();
            }
    });
    imgSlikaAdmin.setOnClickListener((v) -> { oznaciSliku(); });
}

private void oznaciSliku() {...}

@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {...}

private void izmjenaPitanja() {...}

private void ucitavanjeBroja() {...}

private void updateanjeBroja(final String broj) {...}

private void dodavanjePitanja(final String brojPitanja) {...}
```

Slika 80. Funkcije *AdminPitanje.java* klase

Slika 80 prikazuje klasu *AdminPitanje.java* koja se koristi pri dodavanju i brisanju pitanja od strane administratora. Ako korisnik ulazi u aktivnost od navedene klase tako što je odabrao izmjenu pitanja, upitnik vidljiv administratoru se popunjava označenim pitanjem.

U slučaju da administrator koristi opciju dodavanja pitanja, administratoru se otvara aktivnost s praznim upitnikom. Slika unutar upitnika sadrži *onClickListener* koji poziva funkciju *oznaciSliku()* gdje korisnik bira sliku iz galerije na mobilnom uređaju, te se pomoću nadjačane funkcije *onActivityResult()* slika učitava na *layout* trenutne aktivnosti. Kada administrator odluči dodati ili izmijeniti pitanje u odabranom setu i popunio je ispravno upitnik, pritišče tipku za potvrdu. Tipka sadrži *onClickListener* koji poziva funkciju *izmjenaPitanja()* ako administrator izmjenjuje pitanja ili poziva funkciju *ucitavanjeBroja()* u slučaju da je administrator odabrao dodavanje pitanja kao opciju uređivanja.

```
if(filePath!=null){

    if(!Objects.equals(getIntent().getStringExtra( name: "SLIKA"), b: "")){
        StorageReference photoRef = fStorage.getReferenceFromUrl(Objects.requireNonNull(getIntent().getStringExtra( name: "SLIKA")));
        photoRef.delete();
    }

    String Kategorija1 = getIntent().getStringExtra( name: "IDKATEGORIJE");
    String IDseta1 = getIntent().getStringExtra( name: "IDSETA");

    final StorageReference reference = storageReference.child("Pitanja/"+ Kategorija1 +"/"+ IDseta1 + "/" + UUID.randomUUID().toString());
    reference.putFile(filePath).addOnCompleteListener((task) → {
                    reference.getDownloadUrl().addOnSuccessListener((OnSuccessListener) (uri) → {
                            urlNoveSlike = uri.toString();

                            fAuth.getCurrentUser().getIdToken( b: true).addOnCompleteListener((task) → {
                                    if (task.isSuccessful()) {...}
                            });
                    });
            });
```

Slika 81. Dio programskog koda funkcije *izmjenaPitanja()*

Ako administrator pritisne tipku za potvrdu ažuriranja pitanja pokreće se funkcija *izmjenaPitanja()* čiji je dio programskog rješenja vidljiv na slici 81. Ako administrator promjeni sliku pitanja u upitniku, slika se sprema na *Cloud Storage* te se uzima njezin link za skidanje. Zatim se radi zahtjev na temelju upita, tako da se upitnik stavlja u *JSON* datoteku i pozivom se sprema preko starog pitanja označenog od strane administratora. Kada se pitanje spremi, administratoru se ispisuje obavijest da je pitanje izmijenjeno te se vraća na prethodnu aktivnost.

U slučaju da administrator dodaje novo pitanje u odabrani set, aplikacija kao i prije spomenuto poziva funkciju *ucitavanjeBroja()*. Funkcija radi na identičan način kao i istoimena funkcija iz prethodno objašnjene klase *EditiranjePitanja.java.* Dakle ona nabavlja broj pitanja koji se nalazi u odabranom setu te poziva funkciju *updateanjeBroja(),* koja kao i funkcija istog imena iz prethodno objašnjene klase mijenja broj pitanja iz

odabranog seta. Razlika je u tome što navedena funkcija iz klase *AdminPitanje.java* povećava broj pitanja za 1 pošto korisnik dodaje pitanje, a ne briše ga, te poziva funkciju *dodavanjePitanja().* Unutar navedene funkcije se prvo provjeri dali je administrator dodao sliku, te se slika sprema na *Cloud Storage* u slučaju da je dodana, te se također uzima link za skidanje slike. Kreira se *JSON* datoteka korištena u zahtjevu s ostalim potrebnim varijablama te se radi poziv. Kada se pitanje doda na *Cloud Firestore*, funkcija ispisuje obavijest da je pitanje dodano te se otvara aktivnost gdje administrator odmah može izmjenjivati pitanja iz odabranog seta.

7. Zaključak

Iz ovog diplomskog rada može se zaključiti da još uvijek nema velikog izbora kvalitetnih aplikacija za svrhu polaganja ispita u autoškolama. Postoji dosta manjih aplikacija koje ne nude dovoljno opcija. Također se može zaključiti kako većina kandidata za vozača spada u mlađi dio populacije stanovništva i te su upoznati s mobilnim uređajima i njihovim korištenjem. Kandidati za vozače u današnje vrijeme uglavnom uče gradivo iz knjige. Knjige s gradivom se plaćaju te se moraju zamijeniti svaki put kada dođe do promjena sadržaja gradiva, što znači da je potrebno tiskati nove knjige.

Ovaj diplomski rad se sastoji od teorijskog i praktičnog dijela razvoja aplikacije za učenje propisa u prometu te su navedene činjenice i zaključci najviše utjecali na kreiranje mobilne aplikacije. Samim korištenjem aplikacije, učenje gradiva postaje pristupačnije i zanimljivije korisnicima. Uz navedeno, također je ekološki prihvatljivije od učenja gradiva iz knjiga za koje je potrebno koristiti papir. Aplikacija je za razliku od knjige besplatna, te kada se dogodi promjena u sadržaju gradiva, aplikaciji se ažurira baza podataka dok knjige nemaju tu mogućnost.

Za razvoj aplikacije je odabran *android studio* koji koristi *java* programski jezik i *xml* za implementaciju aplikacije. Aplikacija sadrži mnoštvo izbornika za korisnike i za administratora. Korisnici moraju imati korisnički račun za pristup sadržaju aplikacije, a mogu ga kreirati na samoj aplikaciji, te ga po želji uređivati tijekom korištenja same aplikacije. Unutar aplikacije je moguće rješavati vježbe i ispite i također se mogu pregledati upute za rješavanje navedenog. Uz to, korisnik može učiti gradivo koje je raspoređeno na kategorije propisa i prve pomoći, od kojih svaka ima svoje setove. Setovi učitavaju gradivo s baze podataka. Korisnik ima mogućnost pregleda rezultata riješenih vježbi i ispita, te dodatno može detaljno pregledati svaki riješeni ispit. Administrator ima dodatan izbornik čija je svrha uređivanje pitanja određenih kategorija i setova na bazi podataka. Za potrebe baze podataka se koristi *Firebase API* uz pomoć više ovisnosti, te uz korištenje *REST API* web servisa od samoga *Firebase*-a.

Uzimajući u obzir sve probleme kod nabavljanja i učenja gradiva kod kandidata za vozača, te probleme s mijenjanjem sadržaja gradiva, razvijena aplikacija pruža

kandidatima za vozače mogućnost pristupa gradivu s dlana ruke koristeći vlastite mobilne uređaje.

Literatura

Knjige:

1) Rogić Ž. (2012), *Prva pomoć,* Izdavač: Hrvatski crveni križ
2) Gargenta M., Nakamura M. (2014*), Learning Android: Develop Mobile Apps Using Java and Eclipse 2nd edition,* Izdavač: O'Reilly Media

Članci:

1) Ministarstvo unutarnjih poslova (2011), *Pravilnik o načinu obavljanja i organiziranja vozačkih ispita te načinu izdavanja i oduzimanja dopuštenja ovlaštenom ispitivaču* (NN 141/2011), Dostupno na: https://narodne-novine.nn.hr/clanci/sluzbeni/2011_12_141_2829.html, [Pristupljeno: 3. 8. 2020.]

Internet izvori:

1) Android Studio, Dostupno na: https://searchmobilecomputing.techtarget.com/definition/Android-Studio [Pristupljeno: 5.8.2020.]
2) Java, Dostupno na: https://www.techopedia.com/definition/3927/java [Pristupljeno: 5.8.2020.]
3) Java Development Kit, Dostupno na: https://www.techopedia.com/definition/5594/java-development-kit-jdk [Pristupljeno: 5.8.2020.]
4) What is Java virtual machine?, Dostupno na: http://net-informations.com/java/intro/jvm.htm [Pristupljeno: 6.8.2020.]
5) Android App, Dostupno na: https://www.techopedia.com/definition/25099/android-app [Pristupljeno: 6.8.2020.]
6) Mobile Operating System, Dostupno na: https://www.techopedia.com/definition/3391/mobile-operating-system-mobile-os [Pristupljeno: 7.8.2020.]
7) Android SDK, Dostupno na: https://www.techopedia.com/definition/4220/android-sdk [Pristupljeno 8.8.2020.]

8) XML in Android, Dostupno na: https://abhiandroid.com/ui/xml [Pristupljeno: 8.8.2020.]
9) What is Firestore?, Dostupno na: https://howtofirebase.com/what-is-firebase-fcb8614ba442 [Pristupljeno: 9.8.2020.]
10) Cloud Firestore, Dostupno na: https://firebase.google.com/docs/firestore [Pristupljeno: 9.8.2020.]
11) Cloud Storage, Dostupno na: https://firebase.google.com/docs/storage [Pristupljeno: 10.8.2020.]
12) Firebase Authentication, Dostupno na: https://firebase.google.com/docs/auth [Pristupljeno: 10.8.2020.]
13) Firebase Security Rules, Dostupno na: https://firebase.google.com/docs/rules [Pristupljeno: 12.8.2020.]
14) RESTful API, Dostupno na: https://searchapparchitecture.techtarget.com/definition/RESTful-API [Pristupljeno: 13.8.2020.]
15) REST API, Dostupno na: https://firebase.google.com/docs/firestore/use-rest-api [Pristupljeno: 13.8.2020.]

Popis slika

Sažetak

Ovaj diplomski rad se temelji na teorijskom i praktičnom dijelu razvoja aplikacije za učenje propisa u prometu. U teorijskom dijelu rada su objašnjeni ispiti u autoškolama, te korištene tehnologije za razvoj aplikacije. U praktičnom dijelu rada se nalaze opisani *UML use case* i *sequence* dijagrami aplikacije, te potom slijedi opis implementacije i funkcionalnosti aplikacije. Aplikacija za učenje propisa u prometu je praktičnije rješenje za učenje gradiva potrebnog za prolaske ispita autoškole. Aplikacija nudi rješavanje vježbi i pitanja te pristup potrebnog gradiva za polaganje ispita s korisnikovog dlana. Uz navedeno nudi i pregled rješenja vježbi i ispita, uz dodatan detaljan pregled svakog riješenog ispita. Osim toga, aplikacija nudi administratoru mogućnost uređivanja gradiva na bazi podataka.

Ključne riječi: autoškola, mobilna aplikacija, Firebase, android

Summery

This thesis is based on the theoretical and practical part of the development of applications for learning traffic regulations. The theoretical part of the paper explains the exams in driving schools, and the technologies used for application development. The practical part of the paper contains the described UML use case and sequence diagrams of the application, followed by a description of the implementation and functionality of the application. The application for learning traffic regulations is a more practical solution for learning the material needed to pass the driving school exam. The application offers solving exercises and exams. It also offers access to the necessary material for taking exams from the user's palm. In addition to the above, it also offers an overview of the results of exercises and exams, with an additional detailed overview of each solved exam. In addition, the application offers the administrator the ability to edit materials on the database.

Keywords: driving school, mobile application, Firebase, android