Android mobilna aplikacija za praćenje TV serija i filmova

Deskar, Samanta

Undergraduate thesis / Završni rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:230338

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-28**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

zir.nsk.hr

Image not found or type unknown

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

Stručni studij

ANDROID MOBILNA APLIKACIJA ZA PRAĆENJE TV SERIJA I FILMOVA

Završni rad

Samanta Deskar

Osijek, 2018

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1S: Obrazac za imenovanje Povjerenstva za obranu završnog rada na preddiplomskom stručnom studiju

Osijek, 18.09.2018.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu završnog rada na preddiplomskom stručnom studiju

Ime i prezime studenta:	Samanta Deskar
Studij, smjer:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI4464, 20.09.2017.
OIB studenta:	81761458937
Mentor:	Doc.dr.sc. Josip Balen
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Krešimir Nenadić
Član Povjerenstva:	Krešimir Vdovjak
Naslov završnog rada:	Android mobilna aplikacija za praćenje TV serija i filmova
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rada	U teorijskom dijelu rada potrebno je proučiti i opisati tehnologije za izradu mobilnih aplikacija za Android platformu. U praktičnom dijelu rada potrebno je izraditi Android mobilnu aplikaciju kojom će korisnici moći voditi evidenciju i statistiku odgledanih filmova i epizoda TV serija koje će moći komentirati te pratiti izlazak novih epizoda i filmova.
Prijedlog ocjene pismenog dijela ispita (završnog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	18.09.2018.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 01.10.2018.

Ime i prezime studenta:	Samanta Deskar
Studij:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI4464, 20.09.2017.
Ephorus podudaranje [%]:	5

Ovom izjavom izjavljujem da je rad pod nazivom**: Android mobilna aplikacija za praćenje TV serija i filmova**

izrađen pod vodstvom mentora Doc.dr.sc. Josip Balen

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj

1. UVOD 1
1.1. Zadatak završnog rada 1
2. OPIS KORIŠTENIH TEHNOLOGIJA 2
2.1. Java programski jezik 2
2.2. XML opisni jezik 3
2.3. Firebase 3
3. DEFINIRANJE KORISNIČKIH ZAHTJEVA 5
4. SPECIFIKACIJE APLIKACIJE 12
4.1. Baza podataka 12
4.2. The Movie Database API 14
4.3. Model View Presenter arhitektura 15
5. RAZVOJ APLIKACIJE 16
5.1. Korisničko sučelje 16
5.2. Povezivanje API-a i mobilne aplikacije 17
5.2.1. Primjer dohvaćanja popularnih filmova 19
5.3. Rad s bazom podataka 21
5.3.1. Dodavanje filma na listu gledanja 21
5.3.2. Komentiranje filma 23
6. ZAKLJUČAK 26
LITERATURA 25
SAŽETAK 26
ABSTRACT 26
ŽIVOTOPIS 27

1. UVOD

U današnje vrijeme gledanje TV serija i filmova je popularno. Zbog prekomjernog broja gledanja različitih TV serija i filmova, moguće je zaboraviti što je gledano, a u nekom trenutku i zaboraviti sami sadržaj istoga. Suočeni s tim problemom, osmišljena je mobilna aplikacija koja omogućuje korisnicima označavanje filmova i epizoda TV serija koje su odgledali te također komentiranje, odnosno pisanje kratkog osvrta za svaki film ili epizodu. Korisnici također mogu pretraživati ostale naslove i pronalaziti što bi htjeli gledati, a prijedloge mogu naći i kod ljudi koje prate.

Za izradu mobilne aplikacije korištene su razne tehnologije te jezici. Programski jezik pomoću kojeg je pisan kôd je objektno-orijentirani Java programski jezik, a jezik pomoću kojeg je pisan dizajn je XML. Korištena baza podataka je Googleov Firebase. Neke od biblioteka koje se također su Retrofit za spajanje mobilne aplikacije s APIem, Butterknife za spajanje dizajna s programskim kôdom i Glide za učitavanje slika.

1.1. Zadatak završnog rada

Cilj završnog rada je u teorijskom dijelu proučiti i opisati tehnologije za izradu mobilnih aplikacija za Android platformu, odnosno u praktičnom dijelu potrebno je izraditi Android mobilnu aplikaciju kojom će korisnici moći voditi evidenciju i statistiku odgledanih filmova i TV serija te komentiranje istih i praćenje izlaska novih epizoda željenih TV serija i filmova.

2. OPIS KORIŠTENIH TEHNOLOGIJA

2.1. Java programski jezik

Java je programski jezik korišten pri razvoju mobilne aplikacije i jedan od glavnih programskih jezika koji se koriste pri razvoju Android mobilnih aplikacija. Ovaj programski jezik spada u skupinu viših programskih jezika zajedno s programskim jezicima kao što su C++, Python i Ruby. Problem kod ovakvih programskih jezika, a samim time i kod Java programskog jezika je taj što se kôd prvo mora prebaciti u niži programski jezik, odnosno strojni jezik, a taj proces oduzima vrijeme. Uz navedeni nedostatak, viši programski jezici imaju niz prednosti poput jednostavnijeg programiranja, odnosno potrebno je kraće vrijeme za pisanje, a kôd je kraći i lakše se čita. Također, ovakvi programski jezici se mogu izvoditi na bilo kojem računalu, a vezano za Javu, na računalu je potrebno imati instaliran Java virtualni stroj koji interpretira Java bytekod. Bytekod je sličan strojnom jeziku, a nastaje nakon prevođenja izvornog kôda. Poput C++ programskog jezika, Java je također objektno-orijentirani jezik, a razlika između njih je ta što se sav Java kôd nalazi unutar klasa.

Tipovi podataka koje Java koristi su primitivni tipovi poput *integer, float, double i boolean* te korisnički tipovi podataka koji se nazivaju objekti (Programski kôd 2.1.). Pisanjem Java kôda moguće je služiti se unaprijed napisanim klasama i sučeljima koji se nalaze unutar Java Development Kita (JDK) koji predstavlja API (engl. *Application Programming Interface*). Uz službeni JDK, moguće je preuzeti i dodatne API-e koji također olakšavaju pisanje kôda, a primjeri su Retrofit, Butterknife te Glide [1], [2], [3].

```
public class MovieResponse {

    @SerializedName("results")
    private List<Movie> movieList;

    public List<Movie> getMovieList() {
        return movieList;
    }

    public void setMovieList(List<Movie> movieList) {
        this.movieList = movieList;
    }
}
```

Programski kôd 2.1. Primjer Java korisničkog tipa podataka naziva *MovieResponse*

2.2. XML opisni jezik

XML (engl. *Extensible Markup Language*) je tekstualni format za prikaz strukturiranih informacija poput dokumenata, konfiguracija, raznih podataka i slično. Koristi se za dijeljenje informacija između programa, ljudi, računala ili ljudi i računala. XML dokumenti se mogu vrlo pouzdano obrađivati pomoću računalnih programa, a razlog tomu je što obrada neće započeti dok postoje greške u dokumentu. Umjesto obrade takvog dokumenta, korisnici dobiju poruke o greškama kako bi ih mogli ispraviti. XML kôd se piše unutar oznaka (engl. *tag*), a svaki element mora imat početnu oznaku i oznaku zatvaranja kako bi računalo lakše prepoznalo ako postoji greška. Sadržaj koji se nalazi unutar oznaka opisuje informaciju koja se prenosi. U nastavku je prikaz XML kôda (Programski kôd 2.2.) koji opisuje izgled tekstualnih dijelova na jednom ekranu mobilne aplikacije [4], [5].

```
<TextView
    android:id="@+id/textViewLoginLable"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="@dimen/trailersLabel_marginTop_32"
    android:text="@string/login"
    android:textAlignment="center"
    android:textColor="@color/colorPrimaryDark"
    android:textSize="@dimen/textSize_35"
    android:textStyle="bold"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@id/toolbarLogin" />

<EditText
    android:id="@+id/editTextEmailLogin"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginEnd="@dimen/cardView_margin"
    android:layout_marginLeft="@dimen/cardView_margin"
    android:layout_marginRight="@dimen/cardView_margin"
    android:layout_marginStart="@dimen/cardView_margin"
    android:layout_marginTop="@dimen/marginTop_50"
    android:hint="@string/enter_email"
    android:inputType="textEmailAddress"
    app:layout_constraintTop_toBottomOf="@id/textViewLoginLable" />
```

Programski kôd 2.2. Primjer XML koda

2.3. Firebase

Firebase je Googleova platforma koja pruža usluge autentifikacije, baze podataka u stvarnom vremenu, pohrane podataka poput slika te analitičke usluge. Usluge se mogu koristiti za Android i IOS mobilne aplikacije te internetske stranice. Za korištenje je potrebno omogućiti pristup internetu s obzirom da se svi podaci nalaze na platformi [6].

Firebase Auth je usluga za autentifikaciju korisnika te sadrži ugrađeni autentifikacijski sustav za prijavu pomoću e-pošte i lozinke. Također omogućava OAuth2 za Google, Facebook, Twitter i GitHub korisničke račune. Uz navedene načine, omogućava i korisnički definirane autentifikacije. Svojstvo Firebase Auth usluge je i to što je integrirana s Firebase bazom podataka što olakšava kontrolu pristupa podacima [7].

Firebase Database je baza podataka koja omogućava unos i mijenjanje podataka u stvarnom vremenu. Kao što je navedeno, povezana je s uslugom autentifikacije, a spajanje se odvija pomoću WebSocketa. Sinkronizacija podataka se odvija pomoću jednog WebSocketa najvećom mogućom brzinom koju može pružiti klijentska mreža [8].

Firebase Storage je također dio Firebase platforme, a omogućava pohranu korisnički generiranih sadržaja poput slika ili video zapisa. Podaci se spremaju na Google Cloud Storage, a time je omogućen pristup podacima i putem Firebasea i putem Google Clouda. Firebase Storage je također povezan s uslugom autentifikacije kako bi se lakše kontrolirao pristup podacima [9].

3. DEFINIRANJE KORISNIČKIH ZAHTJEVA

Prije početka razvoja aplikacije, utvrđeno je tko su mogući korisnici te koje su njihove mogućnosti. U tablici 3.1. su navedeni svi zahtjevi, a u tablicama u nastavku korisnički zahtjevi su detaljnije opisani.

Tablica 3.1. Osnovni pregled korisničkih zahtjeva

ID	Naziv
1	Registracija korisnika
2	Prijava korisnika
3	Prikaz filmova
4	Prikaz TV serija
5	Dodavanje na listu gledanja
6	Označavanje kao odgledano
7	Komentiranje
8	Pregled liste gledanja
9	Pregled svih korisnika
10	Pregled profila
11	Praćenje korisnika

Tablica 3.2. opisuje korisnički zahtjev za registraciju. Korisniku je omogućena registracija pomoću korisničkog imena, e-pošte i lozinke, a preduvjeta za registraciju nema. Ukoliko korisnik nema korisnički račun za pristup mobilnoj aplikaciji, tada na početnom zaslonu odabire opciju za registraciju te se nakon toga otvara nova aktivnost. U aktivnosti za registraciju, korisniku se nudi da unose korisničko ime, e-poštu i lozinku te nakon unosa, pritiskom na gumb, potvrđuje uneseno. Nakon korisničke potvrde, pomoću Firebase Auth i Firebase Database usluga podaci se spremaju u bazu podataka.

Tablica 3.3. opisuje korisnički zahtjev za prijavu u mobilnu aplikaciju. Preduvjet za prijavu je da korisnik ima korisnički račun aplikacije ili Google račun. Ako korisnik ima korisnički račun aplikacije, na početnom zaslonu unosi e-poštu i lozinku za prijavu, pritiskom na gumb potvrđuje istu te sustav provjerava nalazi li se korisnik u bazi podataka. Ukoliko se korisnik nalazi u bazi podataka, omogućava mu se pristup. Korisnik također može odabrati i prijavu putem Google računa pritiskom na gumb te tada bira između postojećih računa ili unosi novi račun. Nakon

potvrde računa, sustav ponovno provjerava nalazi li se korisnik u bazi te ukoliko nema podataka, isti se spremaju u bazu podataka, a korisniku se omogućava pristup mobilnoj aplikaciji.

Tablica 3.2. Detaljniji opis korisničkog zahtjeva „Registracija korisnika“

ID korisničkog zahtjeva	1
Naziv korisničkog zahtjeva	Registracija korisnika
Opis korisničkog zahtjeva	Korisnik se registrira pomoću korisničkog imena, e-pošte i lozinke
Preduvjet	Nema
Glavni scenarij	1. Korisnik na početnom zaslonu odabire opciju za registraciju 2. Na ekranu za registraciju korisnik unosi vlastite podatke 3. Korisnik potvrđuje unos 4. Firebase Auth sprema korisnika u bazu
Alternativni scenarij	1. Pritisak na gumb za povratak a) Korisnik se vraća na početni zaslon 2. Korisnik nema omogućenu internetsku vezu a) Dolazi obavijest u obliku *toast* poruke da registracija nije uspjela, moli se korisnika da pokuša ponovno

Tablica 3.4. opisuje dva korisnička zahtjeva, prikaz filmova i prikaz TV serija. Nakon prijave u aplikaciju, korisnik u izborniku može odabrati želi li vidjeti popis filmova ili TV serija. Nakon odabira željenog, otvara se nova aktivnost s popisom odabranog. Pomicanjem prsta po ekranu, korisnik mijenja fragmente te time i određeni poredak filmova ili TV serija.

Tablica 3.3. Detaljniji opis korisničkog zahtjeva „Prijava korisnika“

ID korisničkog zahtjeva	2
Naziv korisničkog zahtjeva	Prijava korisnika
Opis korisničkog zahtjeva	Korisnik se prijavljuje pomoću e-pošte i lozinke ili Google računa
Preduvjet	Korisnički račun spremljen u bazu podataka ili postojanje Googleovog korisničkog računa
Glavni scenarij	1. Korisnik na početnom zaslonu unosi korisničko ime i lozinku 2. Pritiskom na gumb potvrđuje prijavu 3. Sustav provjerava nalazi li se korisnik u bazi podataka te omogućava pristup
Alternativni scenarij	1. Korisnik unosi krive podatke a) Dolazi obavijest u obliku *toast* poruke o neuspjeloj prijavi te se moli korisnika da pokuša ponovno 2. Korisnik se prijavljuje pomoću Google računa a) Korisnik odabire prijavu putem Google računa b) Otvara se Googleov predefinirani prozor u kojem korisnik odabire postojeći račun ili dodaje novi c) Odabirom računa provjerava se nalazi li se korisnik u bazi podataka d) Ako se korisnik ne nalazi u bazi podataka, spremaju se njegovi podaci, a ako korisnik postoji, omogućava se pristup mobilnoj aplikaciji

Tablica 3.5. opisuje korisnički zahtjev za dodavanje filma ili TV serije na listu gledanja. Film ili TV serija se dodaju na listu gledanja pritiskom na gumb pored željenog naslova. Sustav tada dohvaća korisničku identifikacijsku oznaku kao i identifikacijsku oznaku filma ili TV serije te podatke sprema u bazu podataka.

Tablica 3.4. Detaljniji opis korisničkih zahtjeva „Prikaz filmova“ i „Prikaz TV serija“

ID korisničkog zahtjeva	3 i 4
Naziv korisničkog zahtjeva	Prikaz filmova i Prikaz TV serija
Opis korisničkog zahtjeva	Korisnik u izborniku odabire *Movies* kako bi prikazao filmove ili *TV show* kako bi prikazao serije
Preduvjet	Internetska veza
Glavni scenarij	1. Korisnik u izborniku odabire stavku *Movies* ili *TV show* 2. Prikazuju se filmovi ili TV serije 3. Korisnik mijenjanjem fragmenata odabire određeni popis filmova ili TV serija
Alternativni scenarij	1. Korisnik ne odabire nijedno a) Izbornik se zatvara, ostaje ista aktivnost na kojoj je izbornik otvoren

Tablica 3.5. Detaljniji opis korisničkog zahtjeva „Dodavanje na listu gledanja“

ID korisničkog zahtjeva	5
Naziv korisničkog zahtjeva	Dodavanje na listu gledanja
Opis korisničkog zahtjeva	Korisnik pritiskom na gumb pored određenog filma ili TV serije stavlja određeni naslov na vlastitu listu gledanja
Preduvjet	Nema
Glavni scenarij	1. Na glavnom popisu korisnik pritiskom na gumb stavlja film ili TV seriju na listu gledanja 2. Sustav u bazi podataka spaja i sprema identifikacijsku oznaku korisnika i identifikacijsku oznaku filma ili TV serije
Alternativni scenarij	Nema

Tablica 3.6. Detaljniji opis korisničkog zahtjeva „Označavanje kao odgledano“

ID korisničkog zahtjeva	6
Naziv korisničkog zahtjeva	Označavanje kao odgledano
Opis korisničkog zahtjeva	Korisnik svaki film ili epizodu TV serije može označiti kao odgledan
Preduvjet	Nema
Glavni scenarij	1. Korisnik otvara film ili TV seriju 2. Pritiskom na gumb označava film ili epizodu kao odgledanu 3. Sustav spaja i sprema u bazu podataka identifikacijsku oznaku korisnika i identifikacijsku oznaku filma ili epizode TV serije
Alternativni scenarij	Nema

Tablica 3.6. opisuje označavanje filma ili epizode TV serije kao odgledane. Korisnik pritiskom na određeni naslov otvara novu aktivnost u kojoj se nalaze detalji o odabranom. Pritiskom na ikonu oka, korisnik označava odabrano kao odgledano. Ukoliko se radi o filmu, u bazi podataka se mijenja atribut *watched* u *true*, a ukoliko se radi o epizodi TV serije, tada se korisnička identifikacijska oznaka i identifikacijska oznaka epizode spremaju u bazu podataka kao novi unos.

Tablica 3.7. opisuje korisničkih zahtjev za komentiranje. Nakon što korisnik odabere određeni film ili TV seriju, otvara se aktivnost s detaljima odabranog. Unutar aktivnosti se nalazi gumb koji vodi na aktivnost za komentiranje te pritiskom na isti, ona se otvara. Korisniku se odabirom okvira za unos teksta otvara tipkovnica mobilnog telefona te mu se omogućava unos komentara. Nakon unesenog komentara, korisnik pritiskom na ikonu potvrđuje svoj unos, a sustav komentar sprema u bazu podataka. Nakon spremanja u bazu podataka, komentar se prikazuje zajedno s ostalim komentarima.

Tablica 3.8. opisuje korisnički zahtjev za vlastite liste gledanja. Nakon otvaranja izbornika korisnik pritiskom na listu gledanja pokreće novu aktivnost u kojoj se nalaze dva fragmenta, filmovi i TV serije. Sustav iz baze podataka dohvaća korisničku listu gledanja te ju prikazuje, a korisnik pomicanjem prsta po ekranu odabire listu gledanja filmova ili TV serija.

Tablica 3.7. Detaljniji opis korisničkog zahtjeva „Komentiranje“

ID korisničkog zahtjeva	7
Naziv korisničkog zahtjeva	Komentiranje
Opis korisničkog zahtjeva	Korisnik komentira određeni film, TV seriju ili epizodu TV serije
Preduvjet	Nema
Glavni scenarij	1. Korisnik otvara određeni film, TV seriju ili epizodu 2. Korisnik pritiskom na gumb otvara aktivnost za komentiranje 3. Pritiskom na okvir za unos teksta prikazuje se tipkovnica 4. Korisnik unosi komentar 5. Pritiskom na gumb za potvrdu šalje komentar 6. U sustavu se komentar sprema u bazu podataka te prikazuje s ostalim komentarima
Alternativni scenarij	Nema

Tablica 3.8. Detaljniji opis korisničkog zahtjeva „Pregled liste gledanja“

ID korisničkog zahtjeva	8
Naziv korisničkog zahtjeva	Pregled liste gledanja
Opis korisničkog zahtjeva	Korisnik odabiranjem stavke iz izbornika otvara vlastitu listu gledanja
Preduvjet	Dodani filmovi i TV serije na listu
Glavni scenarij	1. Korisnik odabire stavku *Watchlist* u izborniku 2. Sustav dohvaća popis iz baze podataka 3. U aplikaciji se prikazuje popis filmova i TV serija
Alternativni scenarij	Nema

Tablica 3.9. opisuje zahtjev za pregled svih korisnika koji imaju korisničke račune za pristup mobilnoj aplikaciji. Nakon što korisnik otvori izbornik te odabere stavku za prikaz korisnika, otvara se nova aktivnost. Nakon otvaranja aktivnosti, sustav pretražuje bazu podataka te prikazuje sve korisnike.

Tablica 3.9. Detaljniji opis korisničkog zahtjeva „Pregled svih korisnika“

ID korisničkog zahtjeva	9
Naziv korisničkog zahtjeva	Pregled svih korisnika
Opis korisničkog zahtjeva	Korisnik odabiranjem stavke iz izbornika otvara pregled svih korisnika
Preduvjet	Postojanje korisnika u bazi podataka
Glavni scenarij	4. Korisnik odabire stavku *Users* u izborniku 5. Sustav dohvaća popis iz baze podataka 6. U aplikaciji se prikazuje popis svih korisnika
Alternativni scenarij	Nema

Tablica 3.10. opisuje korisnički zahtjev za praćenje drugog korisnika. Kada se korisnik nalazi u aktivnosti gdje se prikazuju svih korisnici, tada ima mogućnost pratiti određenog korisnika. Pritiskom na ikonu za praćenje, sustav dohvaća identifikacijsku oznaku trenutnog korisnika i korisnika kojeg isti želi pratiti. Sustav nakon toga podatke sprema u bazu podataka.

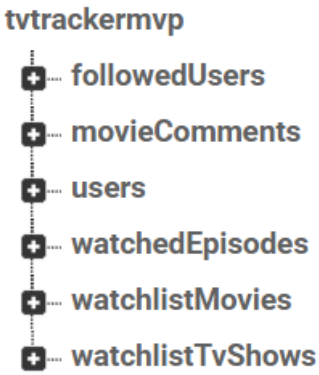
Tablica 3.10. Detaljniji opis korisničkog zahtjeva „Praćenje korisnika“

ID korisničkog zahtjeva	10
Naziv korisničkog zahtjeva	Praćenje korisnika
Opis korisničkog zahtjeva	Korisnik pritiskom na gumb za praćenje korisnika počinje pratiti određenog korisnika
Preduvjet	Postojanje korisnika u bazi podataka
Glavni scenarij	1. Korisnik pritišće gumb za praćenje korisnika 2. Sustav sprema korisnika u bazu podataka praćenih korisnika 3. U aplikaciji se prikazuje poruka da je korisnik praćen
Alternativni scenarij	1. Korisnik je već praćen a) U aplikaciji se prikazuje poruka da se korisnik već prati

4. SPECIFIKACIJE APLIKACIJE

4.1. Baza podataka

Korištena baza podataka unutar aplikacije je Firebase Database usluga Googleove platforme. Podaci unutar baze podataka strukturirani su kao JSON objekti, a može se promatrati kao JSON drvo. Baza podataka nema tablice i zapise poput SQL baza podataka, a kada se doda zapis unutar JSON drveta, stvara se nova grana u postojećoj strukturi s pridruženim ključem [8].



Slika 4.1. Primjer baze podataka korištene u aplikaciji

Prema slici 4.1. vidljivo je da baza podataka ima jednu glavnu granu, *tvtrackermvp* te 6 grana djece:

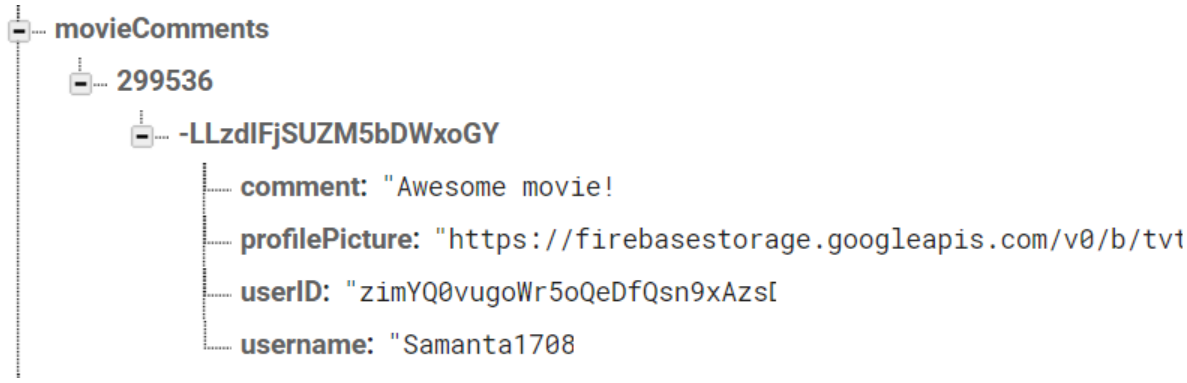
- *followedUsers*
- *movieComments*
- *users*
- *watchedEpisodes*
- *watchlistMovies*
- *watchlistTvShows*

Svaka od grana koje se nazivaju djeca sadrže dodatne grane u kojima se nalaze spremljeni podaci. Grana *followedUsers* se dijeli na grane prema korisničkim identifikacijskim oznakama, a pod svakom identifikacijskom oznakom se nalaze svi korisnici kojeg određeni korisnik prati kao što je prikazano na slici 4.2.



Slika 4.2. Prikaz grananja čvora *followedUsers*

Grane *movieComments* i *tvShowComments* imaju isti princip grananja. Pod identifikacijskom oznakom filma ili TV serije se nalaze svi komentari o određenom filmu ili TV seriji koje su korisnici objavili, a primjer grananja se nalazi na slici 4.3.



Slika 4.3. Prikaz grananja čvora *movieComments*

Također, grane *watchlistMovies* i *watchlistTvShows* imaju isti princip grananja. Svaka od grana sadrži granu dijete s korisničkom identifikacijskom oznakom, a pod tom granom se nalaze dodatne grane s identifikacijskim oznakama filmova ili TV serija. Prikaz grananja se nalazi na slici 4.4.



Slika 4.4. Prikaz grananja čvora *watchlistMovies*.

Identifikacijske oznake korisnika su automatski generirane od strane Firebase Authentication alata, prilikom registracije svakog korisnika, a identifikacijske oznake filmova ili TV serija su dodijeljene korištenjem TMDB API-a koji ih automatski dodjeljuje.

4.2. The Movie Database API

Kako bi aplikacija prikazivala filmove ili TV serije korišten je The Movie Database API (u nastavku TMDB API) koji sadrži bazu podataka svih filmova i TV serija koji su se prikazivali, koji se trenutno prikazuju ili tek trebaju početi s prikazivanjem. Za potrebe izrade mobilne aplikacije, korištene su mogućnosti prikaza popularnih te najbolje ocjenjenih filmova i filmova koji se trenutno prikazuju u kinima i filmova koji će se uskoro prikazivati. Također, za prikaz TV serija korišteni su podaci za popularne TV serije, najbolje ocjenjene te TV serije koje se trenutno prikazuju. Detaljnije objašnjeno dohvaćanje podataka nalazi se u poglavlju 5.2. Povezivanje API-a i mobilne aplikacije.

TMDB API omogućuje tri načina pretrage filmova i TV serija. Prvi način je dodavanjem nastavka */search* na glavnu hipervezu te prosljeđivanjem tekstualnog oblika pretraživanja. Nakon

prosljeđivanja tekstualnog oblika i dodavanjem na hipervezu nakon */search* TMDB API pruža najsličniji rezultat. Drugi način pretrage je dodavanjem */discover* nastavka na glavnu hipervezu, a ovom pretragom TMDB API pruža rezultate filmova i TV serija temeljene na određenim filtrima poput ocjena, certifikata i datuma izlaska. Treći način je dodavanjem */find* nastavka na glavnu hipervezu, a ovim načinom se mogu pretraživati filmovi i TV serije pomoću vanjskih identifikacijskih oznaka generiranih od strane drugih API-a.

Glavne karakteristike TMDB API-a su pružanje najbolje ocjenjenih, popularnih i nadolazećih filmova te filmova koji se trenutno prikazuju u kinima. Na isti način pruža informacije o TV serijama, a uz filmove i TV serije pruža i popularne ljude poput glumaca i redatelja.

Informacije o filmovima koje TMDB API pruža su opis filma, alternativni naslovi, glumci, posada koja je radila na filmu, slike filmova, informacije o izlasku filma, prijevodi, slični filmovi, recenzije te promjene.

Informacije o TV serijama su slične informacijama o filmovima, a uz njih pruža informacije o sezonama i epizodama. Informacije o sezonama koje TMDB API pruža su opis, broj sezone, naziv sezone, glumci, posada, slike i slično, a na isti način pruža i informacije o epizodama [10].

4.3. Model View Presenter arhitektura

Model View Presenter je nastao iz Model View Controller arhitekture, a najčešće se koristi za prikaz podataka na korisničkom sučelju.

Model je sloj koji predstavlja same podatke koji se prikazuju, a oni su definirani u klasama s određenim varijablama koje ih opisuju. Na primjer, klasa *Users* predstavlja model korisnika, a korisnika opisuju svojstva poput njegovog korisničkog imena, e-pošte, korisničke slike profila, lozinke, statusa i korisničke identifikacijske oznake.

Prezentacijski sloj nalazi se između korisničkog sučelja i same baze podataka ili korištenog API-a. Prezentacijski sloj dohvaća zahtjeve s korisničkog sučelja poput pritiska na gumb i taj zahtjev obrađuje na određeni način. Na primjer, pritisak na gumb može značiti da se korisnik želi prijaviti u mobilnu aplikaciju. Prezentacijski sloj tada dohvaća taj zahtjev koji sadrži korisnički unos podataka te poziva bazu podataka koja sadrži model korisnika kako bi provjerila nalazi li se korisnik s određenim svojstvima unutar baze podataka.

Glavni cilj Model View Presenter arhitekture je da se u kôdu odvoji korisnički prikaz, logika obrade podataka i sama baza podataka [11].

5. RAZVOJ APLIKACIJE

5.1. Korisničko sučelje

Da bi aplikacija bila pristupačna korisnicima, potrebno je izraditi korisničko sučelje. Unutar aplikacije nalazi se 8 aktivnosti, a neke od njih sadrže i fragmente pomoću kojih su prikazani podaci bolje organizirani. U tablici 5.1. naveden je popis istih.

Tablica 5.1. Popis aktivnosti i fragmenata unutar aplikacije

Vrsta	Naziv	Opis
Aktivnost	*LoginActivity*	Sučelje za prijavu korisnika, sadrži 2 okvira za unos teksta, gumb za potvrdu prijave te tekst koji vodi na *RegisterActivity*
Aktivnost	*RegisterActivity*	Sučelje za registraciju korisnika, sadrži 3 okvira za unos teksta te gumb za potvrdu registracije
Aktivnost	*MovieActivity*	Sučelje koje sadrži četiri fragmenta za prikaz filmova putem API – a
Aktivnost	*TvShowActivity*	Sučelje koje sadrži 3 fragmenta za prikaz TV serija putem API – a
Aktivnost	*MyProfileActivity*	Sučelje na kojem se nalaze osobni podaci korisnika te gumb za mijenjanje korisničkih podataka te statistike
Aktivnost	*WatchlistActivity*	Sučelje na kojem se nalaze filmovi i TV serije dodane na listu gledanja
Aktivnost	*UsersActivity*	Sučelje u kojem se nalaze 2 fragmenta koji prikazuju sve korisnike u bazi te korisnike koje trenutni korisnik prati
Aktivnost	*UserProfileActivity*	Sučelje u kojem se nalazi prikaz korisničkog profila te gumb koji vodi na prikaz liste gledanja korisnika.
Fragment	*PopularMoviesFragment*	Fragment unutar *MovieActivitya* koji sadrži popularne filmove
Fragment	*TopRatedMoviesFragment*	Fragment unutar *MovieActivitya* koji sadrži najbolje ocjenjene filmove
Fragment	*NowPlayingMoviesFragment*	Fragment unutar *MovieActivitya* koji sadrži filmove koji se prikazuju u kinima.
Fragment	*CommingSoonMoviesFragment*	Fragment unutar *MovieActivitya* koji sadrži filmove koji uskoro izlaze u kina.
Fragment	*PopularTvShowsFragment*	Fragment unutar *TvShowActivitya* koji sadrži popularne TV serije
Fragment	*TopRatedTvShowsFragment*	Fragment unutar *TvShowActivitya* koji sadrži popularne TV serije
Fragment	*LatestTvShowsFragment*	Fragment unutar *TvShowActivitya* koji sadrži TV serije koje su nedavno izašle
Fragment	*OnTheAirTvShowsFragment*	Fragment unutar *TvShowActivitya* koji sadrži TV serije koje se trenutno prikazuju

5.2. Povezivanje API-a i mobilne aplikacije

Korišteni API unutar aplikacije je TMDB API. Povezivanje te zahtjevi rađeni su pomoću Retrofit biblioteke. Za korištenje TMDB API-a potreban je ključ, koji se može dobiti registracijom na TMDB web stranici, a svi zahtjevi šalju se zajedno s ključem. Za preuzimanje podataka koriste se sučelja *ApiService* i *Interactor* te klase *InteractorImpl*, *BackendFactory* i *Response*. U sučelju *ApiService* potrebno je deklarirati metode koje će dohvaćati podatke s TMDB API-a, a u *Interactor* sučelju su navedene metode pomoću kojih se pozivaju metode iz *ApiService* sučelja. *InteractorImpl* definira određena svojstva za svaku metodu iz sučelja *Interactor*, *BackendFactory* konfigurira osnovni URL (engl. *Uniform Resource Locator*) i spaja API s aplikacijom, a *Response* klase definiraju na koji način te koji podaci će se dohvaćati. U nastavku su prikazani primjeri svake klase te zahtjev za prikaz popularnih filmova (Programski kôd 5.1., Programski kôd 5.2., Programski kôd 5.3., Programski kôd 5.4., Programski kôd 5.5.)

```
public interface ApiService {
    @GET("movie/popular/")
    Call<MovieResponse> getPopularMovies(@Query("page") int page,
                                         @Query("api_key") String api_key);
    @GET("movie/now_playing/")
    Call<MovieResponse> getNowPlayingMovies(@Query("page") int page,
                                            @Query("api_key") String api_key);
    @GET("movie/top_rated/")
    Call<MovieResponse> getTopRatedMovies(@Query("page") int page,
                                          @Query("api_key") String api_key);
    @GET("movie/upcoming/")
    Call<MovieResponse> getCommingSoonMovies(@Query("page") int page,
                                             @Query("api_key") String api_key);
    @GET("tv/popular/")
    Call<TvShowResponse> getPopularTvShow(@Query("page") int page,
                                          @Query("api_key") String api_key);
    @GET("tv/top_rated/")
    Call<TvShowResponse> getTopRatedTvShow(@Query("page") int page,
                                           @Query("api_key") String api_key);
    @GET("tv/on_the_air/")
    Call<TvShowResponse> getOnTheAirTvShow(@Query("page") int page,
                                           @Query("api_key") String api_key);
    @GET("search/movie/")
    Call<MovieResponse> getSearchedMovie(@Query("page") int page,
                                         @Query("api_key") String api_key,
                                         @Query("query") String query);
    @GET("search/tv")
    Call<TvShowResponse> getSearchedTvShow(@Query("page") int page,
                                           @Query("api_key") String api_key,
                                           @Query("query") String query);
    @GET("tv/{tv_id}")
    Call<TvShowSeasonResponse> getTvShowSeasons(@Path("tv_id") int tv_id,
                                                @Query("api_key") String api_key);
    @GET("tv/{tv_id}/season/{season_number}")
    Call<TvShowEpisodeResponse> getTvShowEpisodes(@Path("tv_id") int tv_id,
                                                  @Path("season_number")int season_number,
                                                  @Query("api_key") String api_key);
    @GET("movie/{movie_id}")
    Call<Movie> getMovieDetails(@Path("movie_id") int movie_id,
                                @Query("api_key") String api_key);
}
```

Programski kôd 5.1. *ApiService* sučelje

```
public interface MoviesInteractor {

    void getPopularMovies(int page, Callback<MovieResponse> movieResponseCallback);

    void getTopRatedMovies(int page, Callback<MovieResponse> movieResponseCallback);

    void getNowPlayingMovies(int page, Callback<MovieResponse> movieResponseCallback);

    void getComingSoonMoves(int page, Callback<MovieResponse> movieResponseCallback);

    void getSearchedMoves(int page, Callback<MovieResponse> movieResponseCallback, String
query);

}
```

Programski kôd 5.2. *MoviesInteractor* sučelje

```
public class MoviesInteractorImpl implements MoviesInteractor {

    private final ApiService;

    public MoviesInteractorImpl(ApiService apiService) {
        this.apiService = apiService;
    }

    @Override
    public void getPopularMovies(int page, Callback<MovieResponse> movieResponseCallback) {
        apiService.getPopularMovies(page, Constants.API_KEY).enqueue(movieResponseCallback);
    }

    @Override
    public void getTopRatedMovies(int page, Callback<MovieResponse> movieResponseCallback) {
        apiService.getTopRatedMovies(page, Constants.API_KEY).enqueue(movieResponseCallback);
    }

    @Override
    public void getNowPlayingMovies(int page, Callback<MovieResponse> movieResponseCallback)
{
        apiService.getNowPlayingMovies(page,
Constants.API_KEY).enqueue(movieResponseCallback);
    }

    @Override
    public void getComingSoonMoves(int page, Callback<MovieResponse> movieResponseCallback) {

apiService.getCommingSoonMovies(page,Constants.API_KEY).enqueue(movieResponseCallback);
    }

    @Override
    public void getSearchedMoves(int page, Callback<MovieResponse> movieResponseCallback,
                                 String query) {
        apiService.getSearchedMovie(page,Constants.API_KEY,
query).enqueue(movieResponseCallback);
    }
}
```

Programski kôd 5.3. Implementacija *MoviesInteractor* sučelja

```
public class BackendFactory {

    private static Retrofit retrofit = null;

    private static Retrofit getClient(String baseUrl) {

        if (retrofit == null) {
            Interceptor ceptor = new HttpLoggingInterceptor()
                    .setLevel(HttpLoggingInterceptor.Level.BODY);

            OkHttpClient client = new OkHttpClient.Builder()
                    .addInterceptor(ceptor)
                    .build();

            retrofit = new Retrofit.Builder()
                    .client(client)
                    .baseUrl(baseUrl)
                    .addConverterFactory(GsonConverterFactory.create())
                    .build();
        }
        return retrofit;
    }

    private static ApiService getService() {
        return getClient(Constants.BASE_URL).create(ApiService.class);
    }

    public static MoviesInteractor getMoviesInteractor() {
        return new MoviesInteractorImpl(getService());
    }

    public static TvShowsInteractor getTvShowInteractor() {
        return new TvShowsInteractorImpl(getService());
    }

}
```

Programski kôd 5.4. *BackendFactory* klasa

```
public class MovieResponse {

    @SerializedName("results")
    private List<Movie> movieList;

    public List<Movie> getMovieList() {
        return movieList;
    }

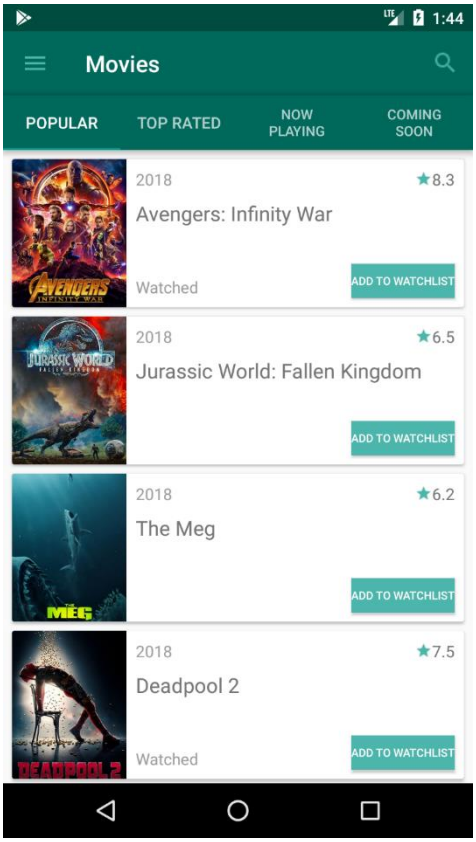
    public void setMovieList(List<Movie> movieList) {
        this.movieList = movieList;
    }
}
```

Programski kôd 5.5. *MovieResponse* klasa

5.2.1. Primjer dohvaćanja popularnih filmova

Kako bi se ostvario prikaz kao na slici 5.1. potrebno je dohvatiti podatke pomoću TMDB API-a. U nastavku je prikazan kôd s objašnjenjima.

Kod pokretanja fragmenta *PopularMoviesFragment* poziva se metoda *getPopularMovies* (Programski kôd 5.6.) koja se nalazi u prezentacijskom sloju fragmenta. Metoda tada poziva *interactor* i njegovu metodu *getPopularMovies* (Programski kôd 5.7.).



Slika 5.1. Prikaz popularnih filmova na korisničkom sučelju aplikacije

```
@Override
public void getPopularMovies(int page) {
    interactor.getPopularMovies(page, getMoviesCallback());
}
```

Programski kôd 5.6. *getPopularMovies* metoda iz prezentacijskog sloja

```
@Override
public void getPopularMovies(int page, Callback<MovieResponse> movieResponseCallback) {
    apiService.getPopularMovies(page, Constants.API_KEY).enqueue(movieResponseCallback);
}
```

Programski kôd 5.7. *getPopularMovies* metoda iz *MoviesInteractorImpl*.

Interactor tada poziva metodu iz sučelja *ApiService*. Nakon što je poslan zahtjev na API, podaci se dohvaćaju pomoću metode *getMoviesCallback* (Programski kôd 5.8.), a prikazuju se na način kako je definirano u klasi *MovieResponse* (Programski kôd 5.5.).

```
private Callback<MovieResponse> getMoviesCallback() {
    return new Callback<MovieResponse>() {
        @Override
        public void onResponse(@NonNull Call<MovieResponse> call,
                               @NonNull Response<MovieResponse> response) {
            if (response.isSuccessful()) {
                getWatchedMovies(response.body().getMovieList(), "getMovies");
            }
        }

        @Override
        public void onFailure(Call<MovieResponse> call, Throwable t) {
            Log.d("TAG", "Fail");
        }
    };
}
```

Programski kôd 5.8. – *getMoviesCallback* metoda za dohvaćanje podataka.

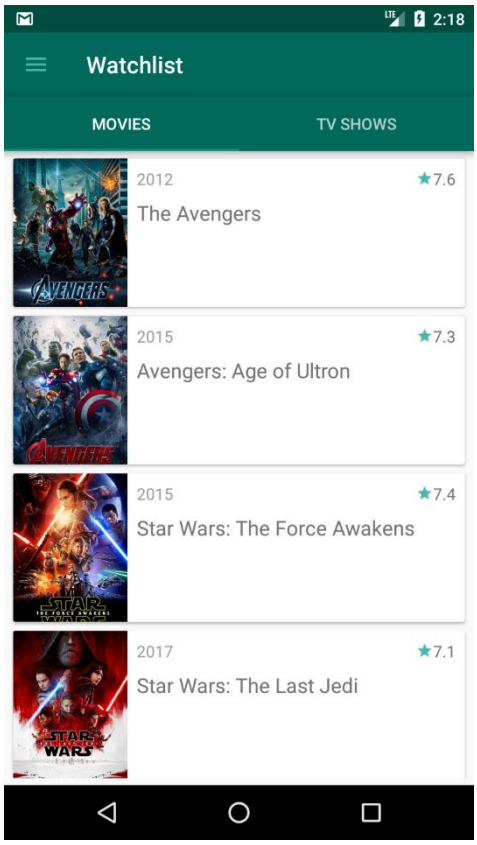
Nakon toga poziva se metoda koja iz baze podataka dohvaća koje je filmove korisnik označio kao odgledane te se nakon toga prikazuju na korisničkom sučelju.

5.3. Rad s bazom podataka

Kao što je već napomenuto, korištena baza podataka je Googleova platforma Firebase s uslugom Firebase Database. Unutar baze podataka se nalaze razni podaci poput korisničkih lista gledanja, korisničkih komentara na filmove i TV serije i slično. U nastavku je opisano dodavanje filma na listu gledanja koja je na korisničkom sučelju prikazana kao na slici 5.2. te komentiranje filma prema slici 5.3.

5.3.1. Dodavanje filma na listu gledanja

Kada korisnik u jednom od prikaza svih filmova pritisne gumb *Add to watchlist* poziva se metoda u prezentacijskom sloju *saveMovieOnWatchlist* (Programski kôd 5.9.) koja tada poziva *AuthenticationHelper* metodu (Programski kôd 5.10.) za dohvaćanje identifikacijske oznake trenutnog korisnika. Nakon dobivene identifikacijske oznake, metoda *saveMovieOnWatchlist* poziva *DatabaseHelper* metodu (Programski kôd 5.11.) za spremanje podataka u bazu podataka.



Slika 5.2. Prikaz korisničke liste gledanja

```
@Override
public void saveMovieOnWatchlist(final Movie movie) {

    final String userID = authenticationHelper.getCurrentUserID();
    databaseHelper.addMovieToWatchlist(new WatchlistMoviesCallback() {
        @Override
        public void onCallbackWatchlistMovies(boolean isOnWatchlist) {
            if (isOnWatchlist) {
                view.toastOnWatchlist(movie.getTitle());
            } else {
                view.toastAddedToWatchlist(movie.getTitle());
            }
        }
    }, movie, userID);
}
```

Programski kôd 5.9. *saveMovieOnWatchlist* metoda u prezentacijskom sloju

```
@Override
public String getCurrentUserID() {

    FirebaseUser user = auth.getCurrentUser();

    if (user!=null){
        return user.getUid();
    }
    else {
        return null;
    }
}
```

Programski kôd 5.10. *getCurrentUserID* metoda u *AuthenticationHelper* klasi

```
@Override
public void addMovieToWatchlist(final WatchlistMoviesCallback moviesCallback,
final Movie movie, final String userID) {

    final DatabaseReference watchlistReference = reference.child("watchlistMovies")
            .child(userID).child(String.valueOf(movie.getId()));
    watchlistReference.addListenerForSingleValueEvent(new ValueEventListener() {
      @Override
      public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
          if (dataSnapshot.exists()) {
              moviesCallback.onCallbackWatchlistMovies(true);
          }
          else {
              watchlistReference.child("title").setValue(movie.getTitle());
              watchlistReference.child("id").setValue(movie.getId());
              watchlistReference.child("poster").setValue(movie.getPoster());
              watchlistReference.child("backdrop").setValue(movie.getBackdrop());
              watchlistReference.child("description").setValue(movie.getDescription());
              watchlistReference.child("releaseDate").setValue(movie.getReleaseDate());
              watchlistReference.child("rating").setValue(movie.getRating());
              watchlistReference.child("watched").setValue("false");
              moviesCallback.onCallbackWatchlistMovies(false);
          }
      }
      @Override
      public void onCancelled(@NonNull DatabaseError databaseError) {
      }
    });
}
```

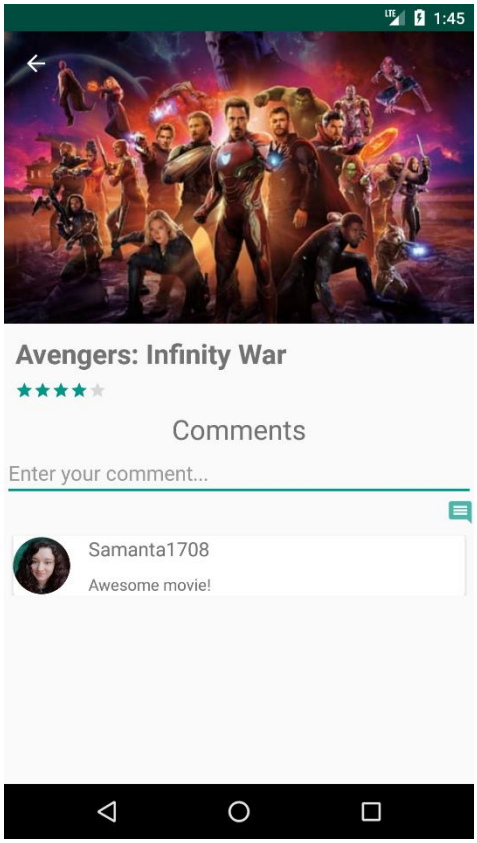
Programski kôd 5.11. *addMovieToWatchlist* metoda u *DatabaseHelper* klasi

Metoda *addMovieToWatchlist* prima *callback* u kojem vraća je li određeni film već dodan na listu gledanja ili ne te također prima model filma i korisničku identifikacijsku oznaku. Pomoću reference na bazu provjerava se za određenu granu, u ovom slučaju granu korisnika u grani *watchlistMovies*, postoji li već unos za određeni film. Ako unos postoji u bazi podataka, u *callback* se vraća *true* kako bi prezentacijski sloj znao da je određeni film već na listi i u skladu s time odredio što će se prikazati na korisničkom sučelju. U slučaju da film ne postoji u bazi podataka, pod granu s korisničkom identifikacijskom oznakom dodaje se novi grana s identifikacijskom oznakom filma u koji se unose podaci iz modela filma, a na kraju se u *callback* vraća *false*. Nakon vraćanja *callbacka*, prezentacijski sloj provjerava što je vraćeno te ako je vraćen *true* tada će se korisničkom sučelju prikazati *toast* poruka kako bi korisnik znao da je film već na njegovoj listi, a u slučaju da je vraćen *false* tada se na korisničkom sučelju prikazuje *toast* poruka kako je film dodan na listu gledanja.

5.3.2. Komentiranje filma

Nakon što korisnik otvori aktivnost s komentarima (Slika 5.3.), može dodati vlastiti komentar na film. Unosom komentara i pritiskom na gumb za slanje, u prezentacijskom sloju pokreće se metoda *saveComment* (Programski kôd 5.12.) koja prvo poziva *AuthenticationHelper* metodu

(Programski kôd 5.10.) kako bi se dobila korisnička identifikacijska oznaka, a zatim se poziva *DatabaseHelper* metoda (Programski kôd 5.13.) za spremanje komentara u bazu podataka.



Slika 5.3. Prikaz aktivnosti za komentiranje

Metoda u *DatabaseHelper* klasi prima i *callback*, u koji postavlja vrijednost *true* ako je komentar uspješno spremljen u bazu podataka te *false* ukoliko je došlo do pogreške. Nakon uspješnog spremanja komentara u bazu, prezentacijski sloj poziva metodu *getMovieComments* kako bi se prikazali svi dosadašnji komentari s novim komentarom.

```
@Override
public void saveComment(final String comment, final int movieID) {

    final String userID = authenticationHelper.getCurrentUserID();

    databaseHelper.getUserInfo(new DatabaseUserCallback() {
        @Override
        public void onCallback(User user) {
            String username = user.getUsername();
            String profilePicture = user.getImage();

            databaseHelper.saveMovieComment(comment, movieID, userID, username,
profilePicture, new MovieSendCommentCallback() {
                @Override
                public void onMovieSendCallback(boolean isSend) {
                    if (isSend) {
                        getComments(movieID);
                        view.setToastCommentSent();
                    }
                }
            });

        }
    },userID);
}
```

Programski kôd 5.12. *saveComment* metoda u prezentacijskom sloju

```
@Override
public void saveMovieComment(String comment, int movieID, String userID,
                             String username, String profilePicture,
                             MovieSendCommentCallback movieSendCommentCallback) {

    DatabaseReference commentsReference =
reference.child("movieComments").child(String.valueOf(movieID));
    DatabaseReference newEntry = commentsReference.push();

    try {
        newEntry.child("comment").setValue(comment);
        newEntry.child("userID").setValue(userID);
        newEntry.child("username").setValue(username);
        newEntry.child("profilePicture").setValue(profilePicture);
        movieSendCommentCallback.onMovieSendCallback(true);
    } catch (DatabaseException e) {
        Log.d("COMMENT_EXEPCITION", e.toString());
        movieSendCommentCallback.onMovieSendCallback(false);
    }

}
```

Programski kôd 5.13. *saveMovieComment* metoda u *DatabaseHelper* klasi

6. ZAKLJUČAK

Izrada završnog rada zahtijevala je istraživanje, proučavanje tehnologija te detaljno planiranje Android mobilne aplikacije. U fazi planiranja određena su korisnička sučelja i zahtjevi korisnika te struktura baze podataka. Nakon planiranja slijedilo je istraživanje. Istraživanjem i proučavanjem tehnologija je određena baza podataka koja će se koristiti, određen je API pomoću kojeg se dohvaćaju filmovi i TV serije te su određene mnoge biblioteke koje su olakšale izradu aplikacije.

Mobilna aplikacija za označavanje odgledanih filmova i TV serija, komentiranje te praćenje ostalih korisnika, olakšava korisnicima praćenje što je odgledano, a što je tek u planu za gledati. Čitanjem komentara o određenom filmu ili TV seriji korisnici mogu dobiti povratnu informaciju drugih gledatelja kako bi znali što očekivati od filma ili TV serije, a praćenjem drugih korisnika mogu pronaći što su oni gledali te na taj način dobiti inspiraciju što bi i sami mogli gledati. Svaki korisnik također može pratiti ukupan broj odgledanih epizoda TV serija ili filmova i na taj način pratiti vlastitu statistiku.

LITERATURA

[1] Java Documentation, About the Java technology, https://docs.oracle.com/javase/8/docs/, rujan 2018

[2] A. D., C. M., Downey, Mayfield, Think Java How to Think Like a Computer Scientist, Green Tea Press, Needham, Massachusetts, 2016

[3] B.B, K.S, Bates, Sierra, Head First Java, O'Reilly Media, Sebastopol, 2005

[4] Wikipedija, XML, https://hr.wikipedia.org/wiki/XML, rujan 2018

[5] Sitepoint, A Really Really Really Good Introduction to XML, https://www.sitepoint.com/really-good-introduction-xml/, rujan 2018

[6] Firebase Documentation, https://firebase.google.com/docs/, rujan 2018

[7] Firebase, Firebase Authentication, https://firebase.google.com/docs/auth/, rujan 2018

[8] Firebase, Firebase Realtime Database, https://firebase.google.com/docs/database/, rujan 2018

[9] Firebase, Cloud Storage, https://firebase.google.com/docs/storage/, rujan 2018

[10] TMDB API Documentation, https://www.themoviedb.org/documentation/api, rujan 2018

[11] Wikipedija, Model View Presenter, https://en.wikipedia.org/wiki/Model–view–presenter, rujan 2018

SAŽETAK

U završnom radu razvijena je Android mobilna aplikacija za praćenje odgledanih filmova i TV serija. Korisnicima je omogućeno dodavanje filmova i TV serija na listu gledanja kao i označavanje istih kao odgledane. Korisnici također mogu komentirati željene naslove, pratiti druge korisnike i njihove liste gledanja te provjeravati vlastitu statistiku odgledanih filmova i epizoda TV serija. U teorijskom dijelu rada opisane se korištene tehnologije, opisan je proces spajanja i dohvaćanja popisa filmova i TV serija putem The Movie Database API-a te spremanje i dohvaćanje podataka iz baze podataka. U praktičnom dijelu utvrđeni su korisnički zahtjevi, izrada baze podataka i dohvaćanje popisa filmova i TV serija. Za razvoj aplikacije korišten je Java programski jezik i razne biblioteke poput Retrofita, Butterknife i Glidea, a baza podataka se nalazi na Googleovoj platformi Firebase.

Ključne riječi: baza podataka, Firebase, komentiranje, lista gledanja, statistika, TMDB API

ABSTRACT

In this bachelor's thesis, an application for tracking movies and TV shows was developed. The application allows users to add movies and TV shows on watchlists and mark movies and TV show episodes as watched. Users can also comment each movie or TV show, follow other users and see their watchlists. Users can also see statistics of watched movies and TV show episodes. The theoretical part describes used technologies and also describes process of connecting and retrieving lists of movies and TV shows from The Movie Database API and saving and retrieving data from database. In the practical part, user requirements were defined, lists of movies and TV shows were retrieved and also the database have been made. For developing application Java programming language was used with libraries such as Retrofit, Butterknife and Glide. The data in the database is saved on Google's platform Firebase.

Keywords: comments, database, Firebase, statistics, TMDB API, watchlist

ŽIVOTOPIS

Samanta Deskar je rođena 17. kolovoza 1996. godine u Virovitici. Od 2003. do 2011. godine pohađa Osnovnu školu Vladimira Nazora u Virovitici. Godine 2011. upisuje Prirodoslovno-Matematičku gimnaziju Petra Preradovića u Virovitici koju završava 2015. godine polaganjem ispita državne mature. Godine 2015. upisuje Elektrotehnički fakultet na Sveučilištu Josipa Jurja Strossmayera u Osijeku, stručni studij Elektrotehnike, smjer Informatika.

Samanta Deskar