MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY

OF CLUJ-NAPOCA, ROMANIA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

TV SHOWS MANAGEMENT AND SOCIAL INTERACTION

LICENSE THESIS

Graduate: **Ban Bogdan Marian**

Supervisor: **Assist. Eng. Cosmina IVAN**

2018

MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY

OF CLUJ-NAPOCA, ROMANIA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA, ROMANIA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

DEAN,
Prof. dr. eng. Liviu MICLEA

HEAD OF DEPARTMENT,
Prof. dr. eng. Rodica POTOLEA

Graduate: **Ban Bogdan Marian**

TV SHOWS MANAGEMENT AND SOCIAL INTERACTION

1. **Project proposal:** *The project proposes an Android application that acts both as a TV show app that provides information and tracking capabilities about them, and a social app allowing users to interact with each other through forums and private chat rooms.*

2. **Project contents:** *Table of contents, Introductions, Project Objectives, Bibliographic Research, Analysis and Theoretical Foundation, Detailed Design and Implementation, Testing and Validation, User's manual, Conclusions, Bibliography and Glossary.*

3. **Place of documentation**: Technical University of Cluj-Napoca, Computer Science Department

4. **Consultants**: Assist. Eng. Cosmina Ivan

5. **Date of issue of the proposal:** November 1, 2017

6. **Date of delivery:** July 09, 2018

Graduate: ______________________________

Supervisor: ______________________________

MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY

OF CLUJ-NAPOCA, ROMANIA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

Declaraţie pe proprie răspundere privind autenticitatea lucrării de licenţă

Subsemnatul(a)____
____,
legitimat(ă) cu _______________ seria _______ nr. ___________________________
CNP ____, autorul lucrării
____
____
____elaborată în vederea susţinerii examenului de finalizare a studiilor de licenţă la Facultatea de Automatică şi Calculatoare, Specializarea ____ din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea _________________ a anului universitar __________, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării, şi în bibliografie.

Declar, că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în faţa unei alte comisii de examen de licenţă.

In cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv, *anularea examenului de licenţă*.

Data	Nume, Prenume
_____________________	_______________________________
	Semnătura

MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY

OF CLUJ-NAPOCA, ROMANIA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

Table of Contents

Chapter 1. Introduction 1

1.1. Project context 1

1.2. Motivation... 1

1.3. Project content 2

Chapter 2. Project Objectives.. 3

2.1. Main Objective 3

2.2. Project Objectives.. 3

2.3. Project properties 4

Chapter 3. Bibliographic Research.. 5

3.1. Cloud computing 5

3.1.1. Essential Characteristics 6

3.1.2. Cloud deployment models 6

3.1.3. Cloud service models.. 7

3.1.4. Conclusions 8

3.2. Push Notifications... 9

3.3. Similar applications 9

3.3.1. Next Episode.. 10

3.3.2. TV Time: Track and Discover Shows 10

3.3.3. Comparative Analysis.. 10

3.3.4. Conclusions 12

Chapter 4. Analysis and Theoretical Foundation.................................. 13

4.1. Conceptual Architecture of the System 13

4.2. Development platform 14

4.3. Functional requirements 17

4.4. Non-functional requirements.. 18

4.5. Use cases of the system 18

4.5.1. Flow of events of the activity diagram 19

4.5.2. Use cases... 20

Chapter 5. Detailed Design and Implementation 35

5.1. System Architecture.. 35

5.2. Technologies used 36

5.2.1. Firebase ... 36
5.2.2. TheMovieDB API ... 41
5.2.3. Glide Framework ... 41
5.3. Main components of the System ... 41
5.3.1. System architecture ... 41
5.3.2. Authentication component ... 42
5.3.3. TV shows component ... 43
5.3.4. Forum component ... 45
5.3.5. Chat component ... 46
5.4. Structure of the RealTime database ... 47
5.4.1. How data is stored ... 47
5.4.2. Structure of the project's database ... 48
Chapter 6. Testing and Validation ... 51
6.1. Manual testing ... 51
6.2. Automated testing ... 52
Chapter 7. User's manual ... 55
7.1. Application requirements ... 55
7.2. Application utilization ... 55
7.2.1. Authentication ... 55
7.2.2. Finding movies or series ... 56
7.2.3. Adding a TV Show to favorites ... 57
7.2.4. Creating a post ... 57
7.2.5. Chatting with another user ... 58
Chapter 8. Conclusions ... 59
8.1. Results ... 59
8.2. Future developments ... 59
Bibliography ... 60
Appendix 1. Glossary ... 62

Chapter 1. Introduction

The purpose of this project is to create an application that enables the end user to find information about and keep track of their favorite TV shows, all the while interacting and sharing with others through the same application, creating a community-like environment.

1.1. Project context

The last two decades have witnessed the biggest spike in technology humankind has ever seen, influencing almost every facet of one's existence. With the rise of technology, which is accelerating our already fast paced lives, the stress we experience on a daily basis seems to be inevitable and it has virtually become a part of our lives. The need for relaxation is thus all the more necessary. The modern world has many ways to achieve this, chief among them being the area of entertainment. Over the course of time, entertainment has served people as an outlet for stress and nowadays it has been proven to be beneficial to the well-being of an individual. There are many forms of entertainment and one of the most popular is watching a movie/ series/ television show.

Not long ago, no one would have predicted that an online network of people talking about their habits and hobbies would become one of the most significant developments of our time. Social networking has created a new definition of how people can connect to each other in the virtual world. Technology evolved from a dial-up internet access with a theoretical transfer speed of 56 kbit/s, which meant over three minutes of download time for a 1 MB photo, to the commonly known expression "there is an app for that". At present, access to the internet is available in almost any place. Indeed, there is an app for almost anything. The number of apps available in the first quarter of 2018 for example, just in Google Play Store, is that of over 3.8 million apps while the Apple store has over 2 million apps.

The project presented is also an Android mobile app which is destined for people who enjoy watching different TV shows and want to have a simple way of browsing through different movies or series and finding new ones, serving as a reminder for when their favorite TV shows air next time.

The applications also aims to have a social aspect, having the capability of creating social interactions by establishing a community and making the end user part of it, making new friends and chatting with them and discussing different topics about TV shows with others. Basically, the application will be creating a platform where users can voice their opinions and find others who share their tastes.

1.2. Motivation

As described from the project context, the application is meant to associate two major roles, one of a TV tracker and overall app designed for searching and finding information about TV shows, while the second is that of a social network providing ways for people to interact with each other.

The motivation for such an app is to combine all these elements into one single app that covers a broader range of functionalities, but at the same time being carefully contained, so they are loosely coupled, for the reason of letting the end user decide which aspects of the application he/she wants to use without being affected by the unused parts of the application.

1.3. Project content

The next chapter of this paper presents the objectives of the project which will detail what the application aims to accomplish in the form of sets of objectives, briefly described and some general aspects about the description of the system will be made.

Chapter 3 will describe some important concepts, like cloud computing and push notifications which are relevant for our application and then a comparative analysis will be made between our application and similar ones from the same domain.

Chapter 4 will present the conceptual architecture, description about the development platform but also the functional and non-functional requirements in a more detailed manner. Then all the use cases that the application has, will be described, analyzing for each use case the actor, the description, preconditions, postconditions, the basic flow and alternative flows.

Chapter 5 presents the fully developed architecture of the system, a relevant description of the technologies used to better understand how the application was developed, the details of implementation, important methods described and finally, the structure of the database.

Chapter 6 describes the details regarding how validating and testing of the application have been made. The two types of testing will be: manual testing and automated testing.

Chapter 7 describes how the application can be installed and presents instructions on how to use the application.

Chapter 8 contains an analysis of what has been made, conclusions regarding the implemented system and future developments of how this system can be improved.

Chapter 2. Project Objectives

2.1. Main Objective

The main objective of this project is to bring a clean and easy-to-use application on Android that clearly separates the two major roles which the project aims to accomplish.

The first role is that of a TV show app that can provide information regarding movies and series, track them, find new ones or get recommendations and other functionalities.

The second role is that of a social network app designed for the purpose of creating a community-like environment by allowing users to express themselves, react to TV shows, create threads on different topics, discover other people's opinions, find people with similar tastes, make new friends and chat with them.

Next up, a set of objectives of the application will be described and finally the most important properties of the system will be specified.

2.2. Project Objectives

Authentication:

- The application will have a way of creating an account and signing in to use all the functionalities of the application but it is not limited to it. If an end user does not want to interact with the community created he/she can still use the application without bothering to create an account.
- The end user will sign in as guest and have all the functionalities regarding the first role of the app, the TV show app available to use.

Find TV shows:

- The application will not only have a simple search that will allow users to search for any movies or series but will also provide different lists of TV shows based on rating, popularity, release date and other aspects which the user will be able to browse.

Adding TV shows to favorites:

- The application will allow users to make lists of their favorite movies or series but some of the information about them will not be saved in the database of the server, but in the local database of the device so the user can access this information even if there is no connection to the Internet.

React to a TV show:

- The application will allow users to express their feelings about any TV show in the form of liking or disliking the content. The users will also be allowed to react to forum implications but the disliking will not be available as opinions of individuals should not be judged.

Forum for any TV show:

- Every TV show that the application provides will contain a forum section which any authenticated user can access. Here the user may create his own thread, or comment to an already created thread.

Feed for each user:

- Since the application aims to create a community, it will have a feed for each authenticated user which will contain his/her information and the activity the user has been doing around the app.

Chat rooms:

- The application allows for searching and finding new people through different methods, users can send friend requests to other users and become friends. When two users are friends they will be able to create a chat room together and make conversations.
- But they are not limited to one chat room, they can create an unlimited number of chat rooms, each with their own title, discussing different topics or different TV shows.

2.3. Project properties

The system will be flexible according to end user’s need, not forcing a creation of an account, but allowing the utilization of the applications with the functionalities that do not require an authenticated user.

The user interface of the system will allow the end user to easily navigate between application’s functionalities and their description will be intuitive, and also error messages will notify the end user when a problem has occurred.

Having such a wide range of possibilities for future implementations, additional functionalities that might be needed in time should be added with the least amount of time required and this can be done by respecting the OOP design principles. On the other hand, too much modularity where it won’t be needed might complicate the project for no good reason.

The system will be secured, passwords of users will be encrypted, data that is to be placed in the database checked before it is saved and permit users to only access the pieces of data that they are allowed to.

Chapter 3. Bibliographic Research

In this chapter we will first look into some concepts related to the domain like cloud computing and push notifications necessary for creating an application of this nature then we will present a comparison between this project and similar applications.

3.1. Cloud computing

First let's define what Cloud is, as George Reese says in his book [1]: The cloud is not simply the latest fashionable term for the Internet. Though the Internet is a necessary foundation for the cloud, the cloud is something more than the Internet. The cloud is where you go to use technology when you need it, for as long as you need it and not a minute more. You do not install anything on your desktop, and you do not pay for the technology when you are not using it. The cloud can be both software and infrastructure. It can be an application you access through the Web or a server that you provision exactly when you need it.

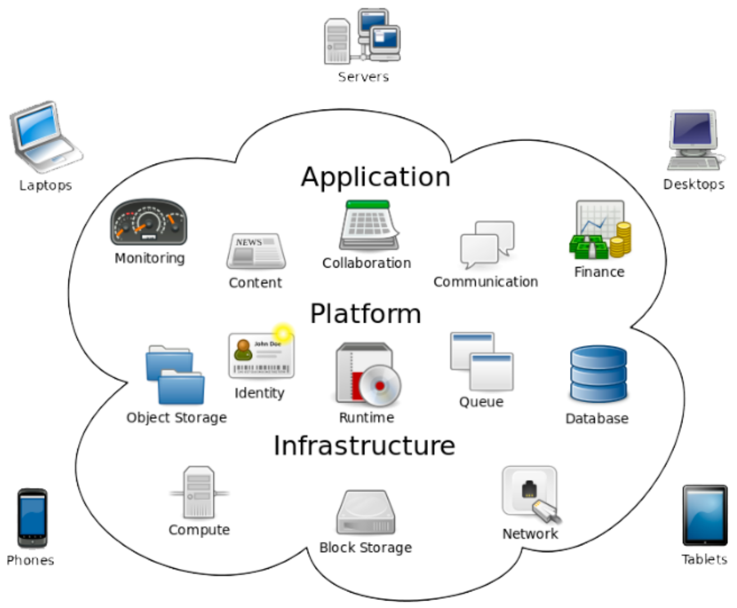


Figure 3.1 – Cloud computing conceptual diagram [2]

As for cloud computing, it is more of a study on how to provide cloud services but a more thorough definition would be from NIST [3] (National Institute of Standards and Technology) which states that cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

3.1.1. Essential Characteristics

These are the characteristics as described by NIST [3]:

- On-demand self-service. A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.
- Broad network access. Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).
- Resource pooling. The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, and network bandwidth.
- Rapid elasticity. Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.
- Measured service. Cloud systems automatically control and optimize resource use by leveraging a metering capability1 at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

3.1.2. Cloud deployment models

Cloud computing is defined with several deployment models, each of which has specific trade-offs for agencies that are migrating services and operations to cloud-based environments.

- Public Cloud – the general public provisions the cloud infrastructure for open use. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider.
- Private Cloud – a private cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises.

- Hybrid Cloud – the cloud infrastructure is a composition of two or more distinct cloud deployment models (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).
- Community Cloud – the Community Cloud is a type of cloud hosting in which the setup is mutually shared between many organizations that belong to a particular community, i.e. banks and trading firms. It is a multi-tenant setup that is shared among several organizations that belong to a specific group which has similar computing apprehensions. [4]

3.1.3. Cloud service models

The three main service models declared by NIST are SaaS, PaaS and IaaS.

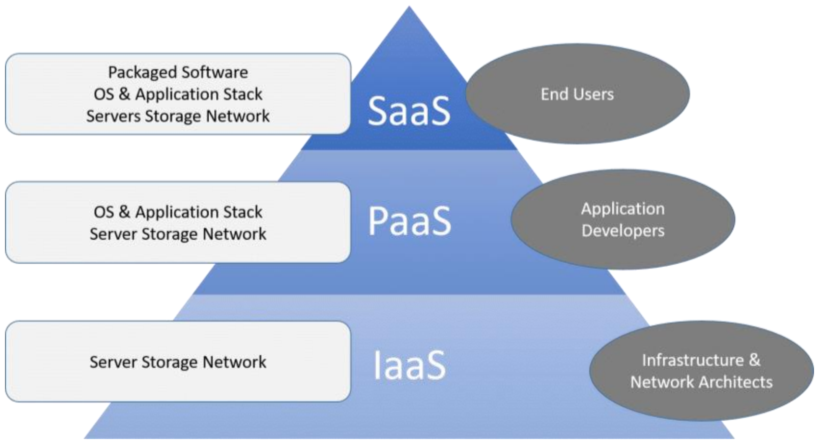


Figure 3.2 – Hierarchical view of Cloud service models [5]

- Software as a Service (SaaS) – the capability provided to the consumer is to use the provider’s applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user specific application configuration settings.
- Platform as a Service (PaaS) – the capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider.3 The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed

applications and possibly configuration settings for the application-hosting environment.

- Infrastructure as a Service (IaaS) – the capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls).[3]

3.1.3.1. Backend as a Service (BaaS)

Backend as a service also known as Mobile backend as a service (MBaaS) is a model for providing web app and mobile app developers with a way to link their applications to backend cloud storage while providing features such as storage, messaging access, push notifications and others.

In other words, MbaaS is delivering a basic stack of storage, messaging, notifications, user management and other essential components for mobile developers. [6]

BaaS is distinct in some ways in relation to the other three service models because it specifically addresses the cloud-computing needs of web and mobile app developers by providing a unified means of connecting their apps to cloud services.

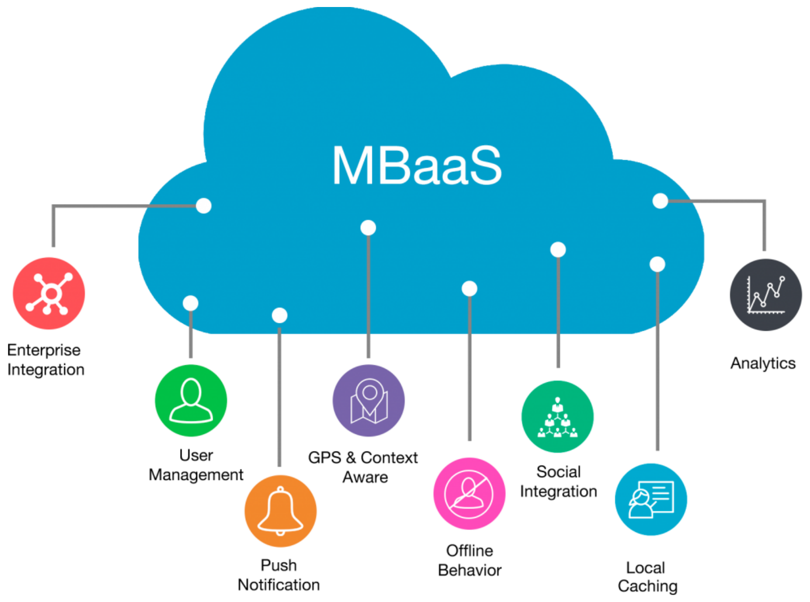


Figure 3.3 – MBaaS facilities [7]

3.1.4. Conclusions

Having considered the many advantages that cloud computing offers as well providing a consistent way of managing backend data, thus saving a lot of time in developing an Android application, the conclusion is that the application's architecture will be based on cloud infrastructure.

3.2. Push Notifications

Push notification, also called server push notification, is the delivery of information from a software application to a computing device without a specific request from the client. Typically, the end user must opt-in to receive alerts; opt-in usually takes place during the install process and end users are provided with a way to manage alerts if they change their minds later on.

An important advantage of push notifications in mobile computing is that the technology doesn't require specific applications on a mobile device to be open in order for a message to be received. This allows a smartphone to receive and display social media or text message alerts even when the device's screen is locked and the social media application that is pushing the notification is closed. [8]

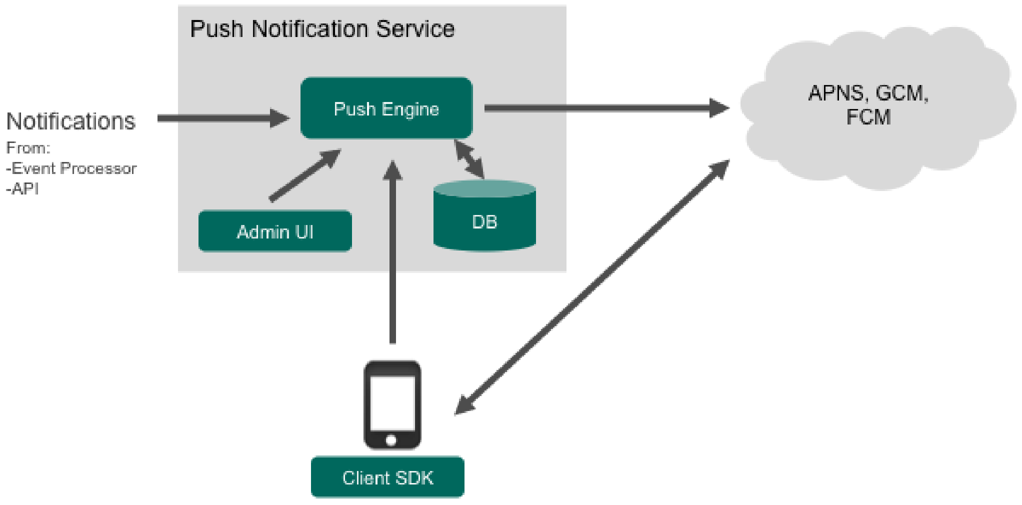


Figure 3.4 – Push notification flow [9]

The concept of push notifications is a significant aspect for this project, to notify a user when the event he subscribed to, occurred. This concept will be further illustrated in the FCM (Firebase Cloud Messaging) section in chapter 5.

3.3. Similar applications

For creating a satisfying and qualitative Android application the two most popular mobile applications related to the objectives of this project were considered which are Next episode and TV Time. Both applications stand strong in some aspects against the other. After describing them, a table of features will be presented to make a comparison between them and our application. The functional requirements identified from these two applications were collected via the direct product and their description.

3.3.1. *Next Episode*

Next Episode is an Android application that easily tracks TV shows and the Movies you watch and discover new - great ones. It has a nice clean interface across both platforms Top charts are visible and you even get personalized recommendations for new shows to follow, based on what you're watching.

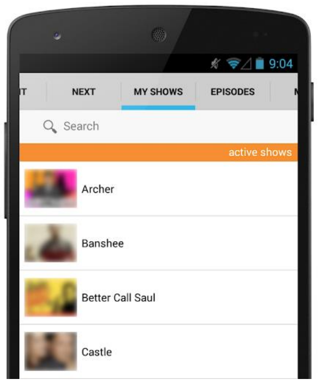


Figure 3.5 – NextEpisode user interface [10]

3.3.2. *TV Time: Track and Discover Shows*

TVShow Time is a simple app on iOS and Android that adds some interesting social elements to tracking TV shows. It has a very light weight interface and allows you to comment on shows with a community of other users of the app.

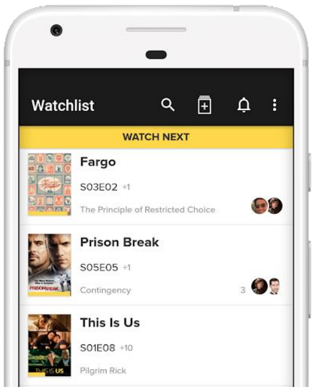


Figure 3.6 – TV Time user interface [11]

3.3.3. *Comparative Analysis*

The functionalities of the two applications described above were identified and the reason for this analysis is to create an idea of how the realization to some extent, of our application’s functionalities will take form.

The features of the applications will be explored in depth in the table of comparisons and below the table it is described what every feature does.

Feature	Miss me?	Next Episode	TV Time
TV Show aspects			
MY SERIES	Yes	Yes	Yes
MY MOVIES	Yes	Yes	No
SEARCH	Yes	Yes	Yes
LATEST SHOWS	Yes	Yes	No
UPCOMING	Yes	Yes	No
AIRING TODAY	Yes	Yes	No
NOW PLAYING	Yes	Yes	No
ON THE AIR	Yes	Yes	No
TOP RATED SHOWS	Yes	Yes	No
POPULAR SHOWS	Yes	Yes	No
RECOMMENDED SHOWS	Yes	Yes	Yes
ADD TO FAVORITES	Yes	Yes	Yes
Community aspects			
CREATE POST	Yes	No	No
COMMENT	Yes	No	Yes
LIKE/DISLIKE	Yes	No	Yes
PROFILE	Yes	No	Yes
FEED	Yes	No	Yes
FRIEND REQUEST	Yes	No	Yes
PENDING	Yes	No	Yes
CHAT ROOMS	Yes	No	No
FRIENDS	Yes	No	No

Table 3.1 – Table of comparison of similar applications

Feature description:

- MY SERIES - check the series that were added to favorites
- MY MOVIES - check the movies that were added to favorites.
- SEARCH - search for any TV show and check the description about it or even search users within the application.
- LATEST SHOWS - check which the newly created TV shows are.
- UPCOMING MOVIES - check the movies that will soon arrive in theaters.
- AIRING TODAY - check series which are airing today.
- NOW PLAYING - check the movies that are now in theaters.
- ON THE AIR - check series that are currently appearing on TV.
- TOP RATED SHOWS - check top rated shows currently on TMDd.
- POPULAR SHOWS- check the most popular shows currently on TMDd.
- ADD TO FAVORITES - user adds favorite shows, and the application will track them.

- RECOMMENDED SHOWS - get personalized recommendations based on the shows or movies the user already watches.
- CREATE POST - create a post on any of the show's forum area for any registered user to see it as well as being able to comment it and/or like it.
- COMMENT - on any post created the user can type a comment in that post.
- LIKE/DISLIKE - like or dislike any show and also, a post can be liked but not disliked.
- PROFILE - each user can customize his/her profile by adding a profile photo and setting the username that will be seen within the application.
- FEED – each user's feed contains the profile of that user as well as his/her activity like what post or comments the user made.
- FRIEND REQUEST - send a friend to any user that was searched via the email, or from the button from the feed of that user.
- PENDING - check pending friend requests from other users.
- CHAT ROOMS - check current active chats and chat with any friend.
- FRIENDS - check friends list and the user may create a chat room with any friend. The user may create an unlimited number of chat rooms with the same friend.

3.3.4. Conclusions

Next Episode is the application that has overall more functionalities than TV Time but there is no interaction between users, you cannot place comments anywhere, you cannot interact with users. Basically, the community part is not available. On the other hand, TV Time does not have a database for movies, it only provides a platform for series. Also, there is no forum like aspect in neither of them.

As it can be seen from the table of comparison, my project aims to take the most important functionalities of the two most popular such applications, and create a new application that integrates all of them.

Chapter 4. Analysis and Theoretical Foundation

This chapter will first introduce the conceptual architecture of the application and then some basic information about the development platform on which the application is built, Android. Then the functional requirements of the application will be analyzed and later presented in the form of use cases and finally the non-functional requirements of the system will be described.

4.1. Conceptual Architecture of the System

In an abstract view, having Firebase in the back-end of the mobile application makes the architecture still a client-server type architecture. The only difference is that the backend APIs that are consumed by the app do not need to be written. Instead, the focus is to build the app logic and let the SDK automatically connect to the various services that Firebase provides, as it is illustrated in figure 4.1.

The main aspect that the figure illustrates is how the push notifications will work through FCM (Firebase Cloud Messaging).

When a downstream message is sent to a client app from an app server, the app server sends the message to an FCM connection server provided by Google; the FCM connection server, in turn, forwards the message to a device that is running the client app. Messages can be sent over HTTP or XMPP (Extensible Messaging and Presence Protocol). Because client apps are not always connected or running, the FCM connection server enqueues and stores messages, sending them to client apps as they reconnect and become available. Also, FCM makes it possible to send messages to client apps directly via the Firebase Notifications Console GUI. [12]

As it can be seen from the architecture in the center of all Firebase components used by the application lies the cloud functions for Firebase. Based on the definition from the official docs regarding cloud functions they: give developers access to Firebase and Google Cloud events, along with scalable computing power to run code in response to those events. While it's expected that Firebase apps will use Cloud Functions in unique ways to meet their unique requirements, typical use cases might fall into these areas:

- Notify users when something interesting happens.
- Perform RealTime Database sanitization and maintenance.
- Execute intensive tasks in the cloud instead of in your app.
- Integrate with third-party services and APIs. [13]

The application also uses other facilities that Firebase offers, besides FCM. The other three that the application uses are: Authentication, Storage and RealTime Database.

More details on how they work with our application will be presented in the Firebase subchapter.

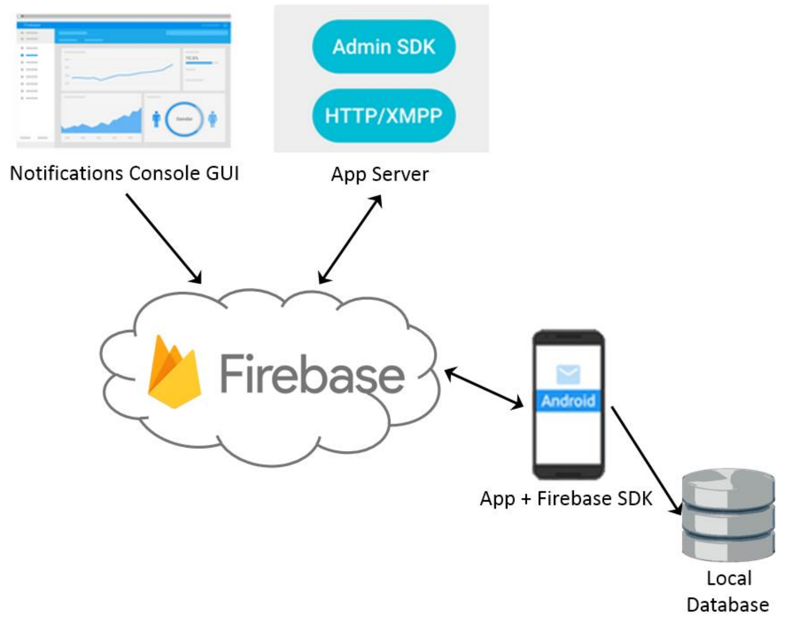


Figure 4.1 – Conceptual Architecture

4.2. Development platform

The development platform of this project is Android [14] which is an open source software stack that includes the operating system, middleware and key mobile applications, along with a set of API libraries for writing applications that can shape the look feel, and function of devices on which they run.

Android applications are written using the Java language [15] which is a platform consisting of virtual machine and an execution environment. The virtual machine is a software-based processor that present an instruction set. The execution environment consists of libraries running programs and interacting with the underlying operating system. Android applications are not run within a Java ME (Mobile Edition) and also Java-compiled classes and executables are not run natively in Android.

Figure 4.2 describes the major components of the Android platform and a short description about each layer will be mentioned.

The Linux Kernel is the foundation of the Android platform. By using it, Android has the capability of providing a development platform on a wide range of hardware, including tablets and television.

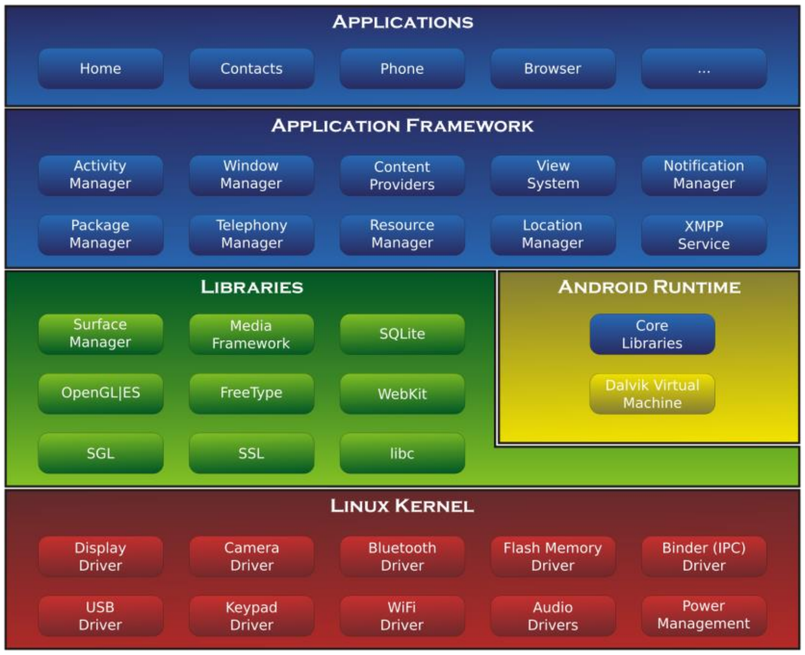


Figure 4.2 – The Android software stack. [16]

Android Runtime is written to run multiple virtual machines on low-memory devices by executing DEX (Dalvik Executable) files, a bytecode format designed especially for Android that's optimized for minimal memory footprint. DEX files can be created by automatically translating compiled applications written in the Java programming language which then are in turn zipped into a single APK file on the device. For devices running Android version 5.0 (API level 21) or higher, each app runs in its own process and with its own instance of the Android Runtime (ART).

In the libraries layer, many core Android system components and services are built from native code that requite native libraries written in C and C++. Java framework APIs can be used to access some of these native libraries to apps.

Application framework or Java framework API provides means of access to the entire Operating System of Android through APIs written in the Java language. These APIs are needed for creating Android apps by reusing the core components and services.

Finally, the Applications layer is the one where the core applications that come with the Android reside, such as SMS, contacts, web browsing and more.

For any Android app, the Activity class is a crucial component. The Android System initiates code in an Activity instance by invoking callback methods that correspond to specific stages of an Activity. The Activity class provides a core set of six callbacks: onCreate(), onStart(), onResume(), onPause(), onStop(), and onDestroy(). Figure 4.3 provides a visual representation of the transition between the stages of an activity lifecycle.

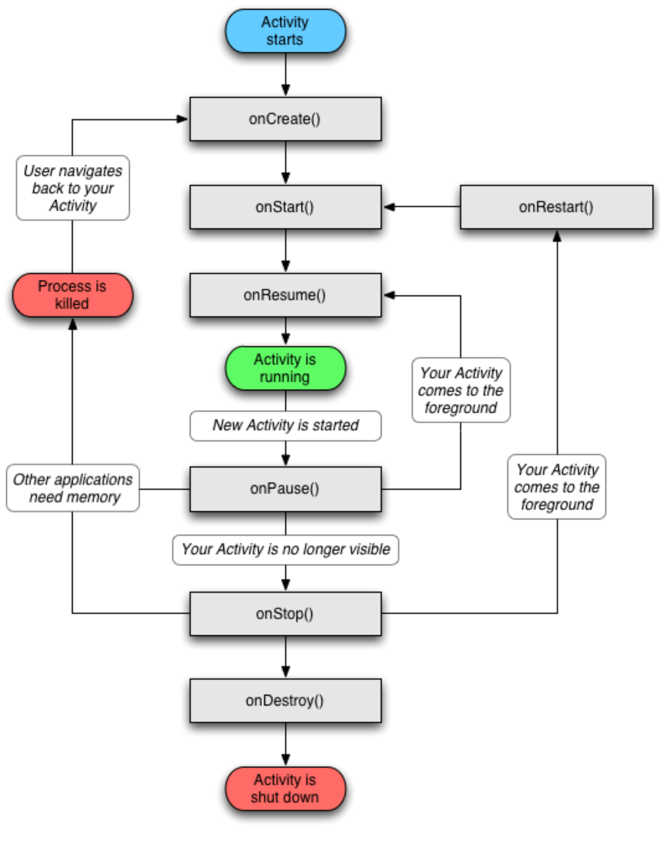
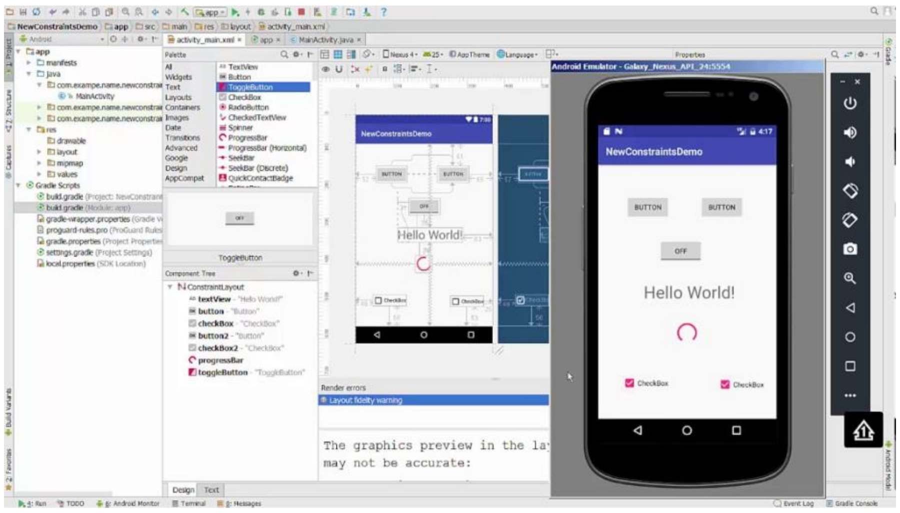

Activity starts
onCreate()
User navigates back to your Activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
Your Activity comes to the foreground
New Activity is started
Other applications need memory
onPause()
Your Activity comes to the foreground
Your Activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Figure 4.3 – Android activity lifecycle [17]

For this project, the implementation of the application was realized in Android Studio, which is the official integrated development environment (IDE) for Google's Android operating system. The version used was 2.3 and below (figure 4.4) is the interface of Android Studio:



```
The graphics preview in the la
may not be accurate:
```

Figure 4.4 – Android Studio 2.3 Graphical User Interface

4.3. Functional requirements

Functional requirements represent the capabilities that should be performed by the system. The following are the applications functionalities:

FR-1	**User authentication/creation**
FR-1.1	The user can create an account
FR-1.2	The user may login to use all the features of the application
FR-1.3	The user may edit its password
FR-1.4	The user may delete its account
FR-2	**Track favorite series or movies**
FR-3	**View upcoming or already aired TV shows in different forms**
FR-3.1	The user may check favorite TV shows
FR-3.2	The user may check latest TV shows
FR-3.3	The user may check upcoming movies or airing today series
FR-3.4	The user may check now playing movies or on the air series
FR-3.5	The user may check top-rated TV shows
FR-3.6	The user may check most popular TV shows
FR-4	**Interact with the community**
FR-4.1	The user may create a post on any TV show’s forum
FR-4.2	The user may comment/like a post
FR-4.3	The user may like/dislike any TV show
FR-4.4	The user may customize its profile
FR-4.5	The user may check other users feed which contains the profile and the activity of that user
FR-4.6	The user may send a friend request to another user
FR-4.7	The user can accept pending friend requests
FR-4.8	The user can create a chat room and communicate with his friend

Table 4.1– Functional Requirements

4.4. Non-functional requirements

Non-functional requirements represent the properties of the system. These properties determine how the system behaves and not what the system does. The following are the applications non-functional requirements:

NFR-1	**Security**
NFR-1.1	The passwords stored in the database are encrypted
NFR-1.2	Routes to specify who has access to what pieces of data
NFR-2	**Usability**
NFR-2.1	The user interface is friendly and easy to use
NFR-2.2	The user may login or use the application as a guest and also may use it online of offline
NFR-3	**Maintainability**
NFR-3.1	The application is easy to keep in safety conditions because of the nature of the data
NFR-3.2	Incorporating new functionalities in the application is easy due to the clean structure of the code which eases the adding of a new service

Table 4.2– Non-Functional Requirements

4.5. Use cases of the system

In this sub-chapter we are going to present the use cases of the application. First, the activity diagram from figure 4.2 involving push notifications which represents the main scenario of utilization will be presented, then for each use case we will detail the primary actor, the description, the basic flow, the alternative flows, the preconditions and the postconditions.

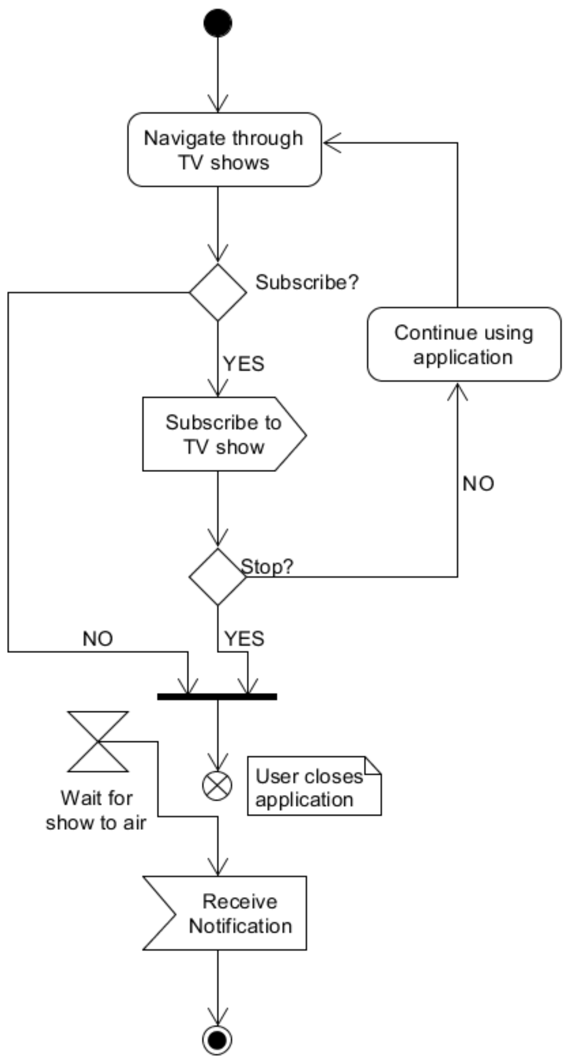


Figure 4.5 – Main activity diagram of the system

4.5.1. Flow of events of the activity diagram

Primary Actor: Any user
Description: One of the main purposes of this application is to notify a user via a push notification that the show he has subscribed to, has aired. This is what this activity diagram captures.
Basic Flow:
Use-Case Start: the start of the use case is that the user starts the application.

1. The user navigates through TV shows from different lists.
2. The user subscribes to a TV show.

3. FCM waits for the time event, which is when the shows airs, to trigger.
4. The show has aired and the push notification is sent.

Use-Case End: the end of the use case is that the notification has been received by the device.

Alternative Flows:

1a. Not subscribing.

1.1. The user chooses to not subscribe to any TV show.
1.2. The user closes the application.

2a. Subscribing cycle.

2.1. The user chooses to subscribe to multiple TV shows.
2.2. The flow returns to step 1.

3a. Error in FCM server.

3.1. The token subscription is not added to the FCM server.
3.2. The user will never receive the notification.

Preconditions:

1. Device must have Android Operating System.
2. Application must be installed on the device.

Postconditions:

1. Closing the application must be successful.
2. Push notification becomes active, is ready to be triggered at the specific date and the message is sent on that specific date.

4.5.2. Use cases

There are two actors that will use the system. The registered user, figure 4.6 and the guest, figure 4.7. Below the use case diagrams for both are presented.

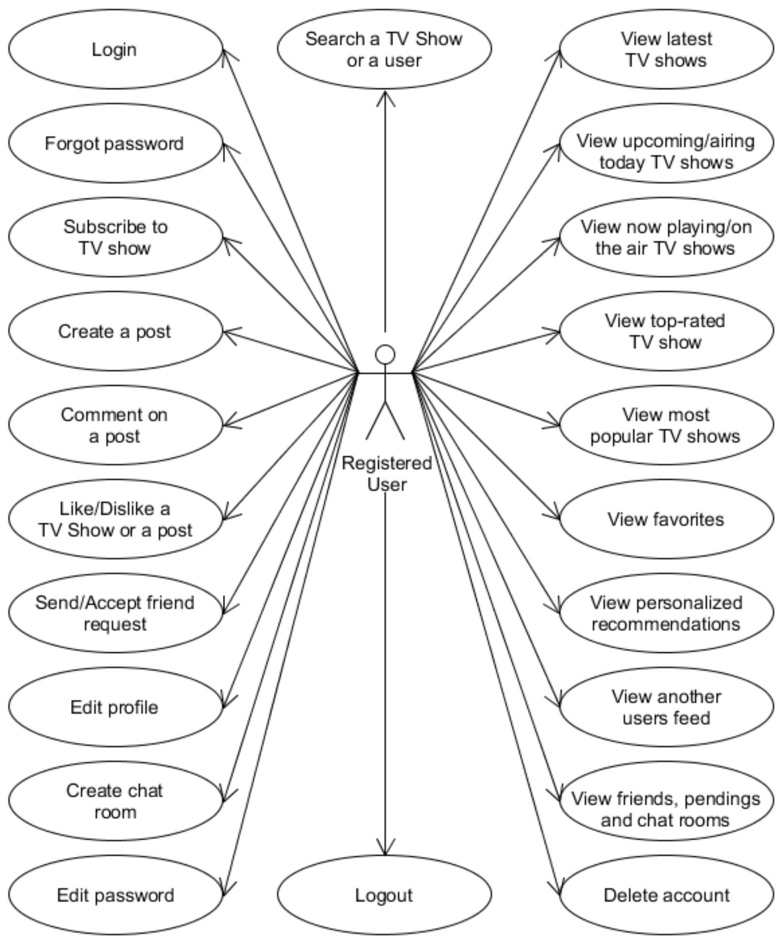

Login
Search a TV Show or a user
View latest TV shows
Forgot password
View upcoming/airing today TV shows
Subscribe to TV show
View now playing/on the air TV shows
Create a post
View top-rated TV show
Comment on a post
View most popular TV shows
Registered User
Like/Dislike a TV Show or a post
View favorites
Send/Accept friend request
View personalized recommendations
Edit profile
View another users feed
Create chat room
View friends, pendings and chat rooms
Edit password
Logout
Delete account

Figure 4.6 – Use case diagram for registered user

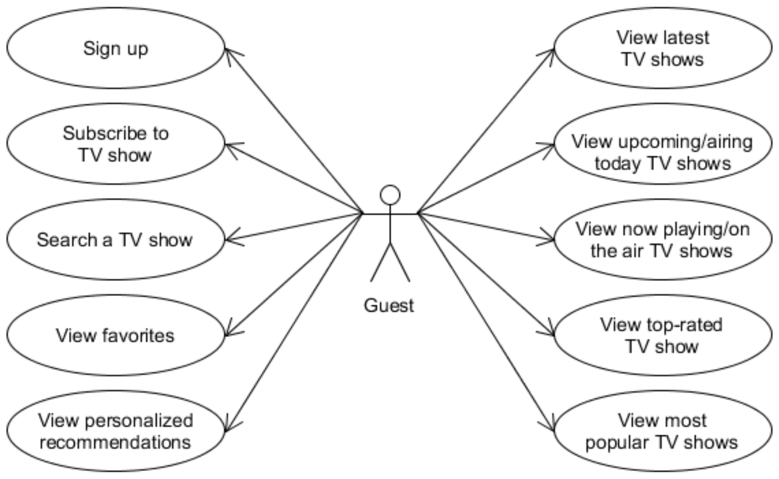

Sign up
View latest TV shows
Subscribe to TV show
View upcoming/airing today TV shows
Search a TV show
View now playing/on the air TV shows
Guest
View favorites
View top-rated TV show
View personalized recommendations
View most popular TV shows

Figure 4.7 – Use case diagram for guest

UC1

Use case name: Login

Primary Actor: Registered user

Description: This use case will describe the registered user logging into the application to use its capabilities to full extent.

Basic Flow:

1. The user types his email and password.
2. The user presses the “Sign in” button.
3. The login is successful and the user is redirected to the application’s main window.

Alternative Flows:

1a. Invalid email or password

1.1 The credentials typed by the user are invalid.
1.2 The user is asked to try again.
1.3 The flow return to step 1.

Preconditions:

1. Device must have Android Operating System.
2. Application must be installed on the device.

Postconditions:

1. The user has successfully signed in into the application.

UC2

Use case name: Forgot password

Primary Actor: Registered user

Description: This use case will describe the registered user recovering his account.

Basic Flow:

1. The user presses the “Forgot password” textview.
2. The user types his email.
3. The user presses the “Forgot password” button.

Alternative Flows:

2a. Invalid email

2.1 The credentials typed by the user is invalid.
2.2 The user is asked to try again.
2.3 The flow return to step 2.

Preconditions:

1. Device must have Android Operating System.
2. Application must be installed on the device.

Postconditions:

1. The user receives an email with a link, where he is instructed to create a new password.

UC3

Use case name: Subscribe to a TV Show

Primary Actor: Any user

Description: This use case will describe the user subscribing to a TV show which means it will be added to the movies or series favorites section.

Basic Flow:

1. The user enters on a particular show from the list.
2. The user presses the “Add to my favorites” button.
3. The user is notified that the TV show has been added to favorites.

Alternative Flows:

-

Preconditions:

1. Device must have Android Operating System.
2. Application must be installed on the device.

Postconditions:

1. The user has successfully added a show to favorites.

UC4

Use case name: Create a post

Primary Actor: Registered user

Description: This use case will describe the registered user creating a post on a TV show’s forum

Basic Flow:

1. The user enters on a particular show from the list.
2. The user presses the “show posts” button
3. The user presses on the “create post” button
4. The user types in the title, a picture and the content of the post.
5. The user send the post by pressing on the send button.

Alternative Flows:

-

Preconditions:

1. Device must have Android Operating System.
2. Application must be installed on the device.
3. User must be logged in.

Postconditions:

1. The user has successfully created a post on a particular TV show.

UC5

Use case name: Comment on a post

Primary Actor: Registered user

Description: This use case will describe the registered user commenting on a post

Basic Flow:

1. The user enters on a particular show from the list.
2. The user presses the “show posts” button
3. The user presses the “comment” button of that post.
4. The user types the comment.

4. The user presses the “Send” button.

Alternative Flows:

-

Preconditions:

1. Device must have Android Operating System.
2. Application must be installed on the device.
3. User must be logged in.

Postconditions:

1. The user has successfully placed comment on a particular post

UC6

Use case name: Like/Dislike a TV show

Primary Actor: Registered user

Description: This use case will describe the registered user reacting to a TV show.

Basic Flow:

1. The user enters on a particular show from the list.
2. The user presses the “thumbs up” or “thumbs down” icon.

Alternative Flows:

-

Preconditions:

1. Device must have Android Operating System.
2. Application must be installed on the device.
3. User must be logged in.

Postconditions:

1. The user has successfully reacted to a particular TV show.

UC7

Use case name: Like a post

Primary Actor: Registered user

Description: This use case will describe the registered user liking a post

Basic Flow:

1. The user enters on a particular show from the list.
2. The user presses the “show posts” button
3. The user presses the “like” button of that post.

Alternative Flows:

-

Preconditions:

1. Device must have Android Operating System.
2. Application must be installed on the device.
3. User must be logged in.

Postconditions:

1. The user has successfully liked a particular post.

UC8

Use case name: Send a friend request

Primary Actor: Registered user

Description: This use case will describe the registered user sending a friend request

Basic Flow:

1. The user presses the “Add friend” button.

2. The user types in the email of the user he wants to be friends with.
3. The user presses the “Ok” button.

Alternative Flows:

1a. Pressing the add friend button

1.1 The user is pressing the add friend button from the user’s feed
1.2 The flow returns to step 3.

3a. Not found

3.1 The user searches for an inexistent user.
3.2 The user is notified that the user doesn’t exist.
3.3 The flow returns to step 1.

3b. Pending request already sent

3.1 The user types an email of a user to which a pending request was already sent.
3.2 The user is notified that the pending was already sent.
3.3 The flow returns to step 1.

3c. Invalid email

3.1 The user searches for an invalid email.
3.2 The user is notified that the email is invalid.
3.3 The flow returns to step 1.

3d. Already friends

3.1 The user searches for a user he is already friends with.
3.2 The user is notified that the user he wanted to send a request is already a friend.
3.3 The flow returns to step 1.

3e. Own email

3.1 The user types in his own email
3.2 The user is notified he cannot befriend himself.
3.3 The flow returns to step 1.

Preconditions:

1. Device must have Android Operating System.
2. Application must be installed on the device.
3. User must be logged in.

Postconditions:

1. The user has successfully followed another user.

UC9

Use case name: Accept a friend request

Primary Actor: Registered user

Description: This use case will describe the registered user accepting a friend request.

Basic Flow:

1. The user goes to the “My conversations” section.
2. The user presses the “Pending button”.
3. The user presses the green check button of the friend request.

Alternative Flows:

3a. Denying a friend request

3.1. The username presses the red x mark button, denying the friend request.

Preconditions:

1. Device must have Android Operating System.
2. Application must be installed on the device.
3. User must be logged in.

Postconditions:

1. The user has successfully edited his desired setting of his account.

UC10

Use case name: Edit profile
Primary Actor: Registered user
Description: This use case will describe the registered user editing his profile and what can he edit. The email verification can only be done once.
Basic Flow:

1. The user goes to the “My Profile” section.
2. The user may edit his username.
3. The user may edit his profile picture
4. The user may verify his account.

Alternative Flows:
2a. Duplicate username

2.1. The username typed is already taken.
2.2. The user is notified that the user is already taken
2.3. The flow return to step 2.

4a. Already verified

4.1 The user has already sent a verification email
4.2 The user is notified that he already sent a verification email.

Preconditions:

1. Device must have Android Operating System.
2. Application must be installed on the device.
3. User must be logged in.

Postconditions:

1. The user has successfully edited his desired setting of his account.

UC11

Use case name: Create chat room
Primary Actor: Registered user
Description: This use case will describe the registered user creating a chat room.
Basic Flow:

1. The user goes to the “My Conversations” section.
2. The user presses the “Friends” button.
3. The user presses on any friend from the friend list.
4. The user types in the title of the new chat room
5. The user presses ok and the new chat is created.
6. The user presses the “Chat rooms” button.
7. The user presses the chat room from the chat room list.
8. The user types a comment and sends a comment.

Preconditions:

1. Device must have Android Operating System.
2. Application must be installed on the device.
3. User must be logged in.
4. User must be friends with the user he wants to create a chat room.

Alternative Flows:

The other user deleted the chat room

7.1. The user is notified by the system that the other user has deleted the chat room

7.2 The flow returns to step 1.

Postconditions:

1. The user has successfully created a chat room and sent a comment in it.

UC12

Use case name: Edit password

Primary Actor: Registered user

Description: This use case will describe the registered user editing his profile and what can he edit. The email verification can only be done once.

Basic Flow:

1. The user goes to the “My Profile” section.
2. The user presses the “Change Pass” button.
3. The user types his new password.
4. The user presses the “Change” button.

Alternative Flows:

4a. Credential too old

4.1. The user hasn’t logged in in this session, so an error occurs.

4.2. The user logs out of his account.

4.3. The user logs in with his old credentials.

4.4. The flow restarts to step 1.

Preconditions:

1. Device must have Android Operating System.
2. Application must be installed on the device.
3. User must be logged in.

Postconditions:

1. The user has successfully edited his password.

UC13

Use case name: Search a TV show.

Primary Actor: Any user

Description: This use case will describe the user searching for a TV show

Basic Flow:

1. The user is to the any section from the series or movies.
2. The user types in the search bar the series or movie he wants to search.
3. The user types the send button from keyboard.

Alternative Flows:

-

Preconditions:

1. Device must have Android Operating System.
2. Application must be installed on the device.

Postconditions:

1. The user receives a list of the TV shows that respect the query typed in.

UC14

Use case name: Search a user
Primary Actor: Registered user
Description: This use case will describe the registered user searching for another user.
Basic Flow:

1. The user goes to the “My conversations” section.
2. The user presses the “Search user” button.
3. The user is types in the email of the user.
4. The user presses the “Ok” button.
5. The user is redirected to the feed of the user he/she searched.

Alternative Flows:
4a. User not found
4.1. The user is notified that there is no user with such an email within the application.
Preconditions:

1. Device must have Android Operating System.
2. Application must be installed on the device.
3. User must be logged in.

Postconditions:

1. The user receives a list of the TV shows that respect the query typed in.

UC15

Use case name: View latest TV shows
Primary Actor: Any user
Description: This use case will describe the user checking the latest TV shows.
This is available for both movies and series.
Basic Flow:

1. The user goes to the “Latest” section.

Alternative Flows:
-
Preconditions:

1. Device must have Android Operating System.
2. Application must be installed on the device.

Postconditions:

1. The user receives a list of the latest TV shows,

UC16

Use case name: View upcoming movies
Primary Actor: Any user

Description: This use case will describe the user checking the upcoming TV shows. This refers only to the movies that will appear in the theaters soon.
Basic Flow:
1. The user goes to the “Upcoming” section.

Alternative Flows:
-

Preconditions:
1. Device must have Android Operating System.
2. Application must be installed on the device.

Postconditions:
1. The user receives a list of the upcoming movies,

UC17
Use case name: View airing today TV series
Primary Actor: Any user
Description: This use case will describe the user checking the airing today TV series. This refers only to the movies that will appear in the theaters soon.
Basic Flow:
1. The user goes to the “Airing today” section.

Alternative Flows:
-

Preconditions:
1. Device must have Android Operating System.
2. Application must be installed on the device.

Postconditions:
1. The user receives a list of the airing today TV series.

UC18
Use case name: View now playing movies
Primary Actor: Any user
Description: This use case will describe the user checking the movies that are now playing in theaters. This only refers to movies.
Basic Flow:
1. The user goes to the “Now playing” section.

Alternative Flows:
-

Preconditions:
1. Device must have Android Operating System.
2. Application must be installed on the device.

Postconditions:
1. The user receives a list of the movies that are now playing in theaters.

UC19
Use case name: View on the air TV series
Primary Actor: Any user

Description: This use case will describe the user checking the TV series that are currently on air.
Basic Flow:
1. The user goes to the “On the Air” section.

Alternative Flows:-
Preconditions:
1. Device must have Android Operating System.
2. Application must be installed on the device.

Postconditions:
1. The user receives a list of the TV series that are currently on air.

UC20
Use case name: View top-rated TV shows
Primary Actor: Any user
Description: This use case will describe the user checking the top-rated TV shows. This is available for both movies and series.
Basic Flow:
1. The user goes to the “Top Rated” section.

Alternative Flows:-
Preconditions:
1. Device must have Android Operating System.
2. Application must be installed on the device.

Postconditions:
1. The user receives a list of the top-rated TV shows.

UC21
Use case name: View most popular TV shows
Primary Actor: Any user
Description: This use case will describe the user checking the most popular TV shows. This is available for both movies and series.
Basic Flow:
1. The user goes to the “Most popular” section.

Alternative Flows:
-
Preconditions:
1. Device must have Android Operating System.
2. Application must be installed on the device.

Postconditions:
1. The user receives a list of the current most popular TV shows,

UC22
Use case name: View favorites
Primary Actor: Any user
Description: This use case will describe the user viewing his list of favorites TV shows.
Basic Flow:
1. The user goes to the “My favorites” section.

Alternative Flows:

-

Preconditions:

1. Device must have Android Operating System.
2. Application must be installed on the device.

Postconditions:

1. The user receives a list of his favorites TV shows.

UC23

Use case name: View recommendations

Primary Actor: Any user

Description: This use case will describe the user viewing his list of recommendations based on TV shows that he added to favorites/subscribed to.

Basic Flow:

1. The user goes to the "My recommendations" section.

Alternative Flows:

1a. Empty list

1.1. The user hasn't subscribed to any TV Show, meaning he hadn't add anything to his favorites, so the list is empty.

Preconditions:

1. Device must have Android Operating System.
2. Application must be installed on the device.

Postconditions:

1. The user receives a list of recommended TV shows.

UC24

Use case name: View another user's feed

Primary Actor: Registered user

Description: This use case will describe the registered user viewing another user's feed.

Basic Flow:

1. The user enters on a particular show from the list.
2. The user presses the "show posts" button
3. The user presses the "comment" button of that post.
4. The user presses on the user's name from comment or post.
5. The user is redirected to that user's feed.

Alternative Flows:

-

Preconditions:

1. Device must have Android Operating System.
2. Application must be installed on the device.
3. User must be logged in.

Postconditions:

1. The user enters to the feed of a user.

UC25

Use case name: View friends list

Primary Actor: Registered user
Description: This use case will describe the registered user viewing his/her friend list
Basic Flow:

1. The user goes to the “My Conversations” section.
2. The user presses the “Friends” button.

Preconditions:

1. Device must have Android Operating System.
2. Application must be installed on the device.
3. User must be logged in.

Alternative Flows:

-

Postconditions:

1. The user receives a list of his/her friends.

UC26
Use case name: View pending friend requests list
Primary Actor: Registered user
Description: This use case will describe the registered user viewing his/her pending friend request list.
Basic Flow:

1. The user goes to the “My Conversations” section.
2. The user presses the “Pendings” button.

Preconditions:

1. Device must have Android Operating System.
2. Application must be installed on the device.
3. User must be logged in.

Alternative Flows:

-

Postconditions:

1. The user receives a list of his/her pending friend request.

UC27
Use case name: View chat rooms list
Primary Actor: Registered user
Description: This use case will describe the registered user viewing his/her chat rooms.
Basic Flow:

1. The user goes to the “My Conversations” section.
2. The user presses the “Chat rooms” button.

Preconditions:

1. Device must have Android Operating System.
2. Application must be installed on the device.
3. User must be logged in.

Alternative Flows:

-

Postconditions:

1. The user receives a list of his/her chat rooms.

UC28

Use case name: Delete account

Primary Actor: Registered user

Description: This use case will describe the registered user viewing another user's profile.

Basic Flow:

1. The user enters the "My Profile" section.
2. The user presses the "Delete account" button.
3. The user confirms the deletion of the account.

Alternative Flows:

-

Preconditions:

1. Device must have Android Operating System.
2. Application must be installed on the device.
3. User must be logged in.

Postconditions:

1. The user successfully deletes his account.

UC29

Use case name: Log out

Primary Actor: Registered user

Description: This use case will describe the registered user logging out of the application.

Basic Flow:

1. The user presses the "Log out" section.

Alternative Flows:

-

Preconditions:

1. Device must have Android Operating System.
2. Application must be installed on the device.
3. User must be logged in.

Postconditions:

1. The user successfully logs out of his account.

UC30

Use case name: Sign up

Primary Actor: Guest

Description: This use case will describe the guest creating an account.

Basic Flow:

1. The guest presses the "Sign up" textview.
2. The user types his email.
3. The user types the password.
4. The user presses the "Sign up" button and is redirected to the application's main window.

Alternative Flows:
2a. Regex error
    2.1. The email provided is an invalid email in syntax.
    2.2. The user is asked to type a valid email
    2.3. The flow returns to step 2.

Preconditions:
1. Device must have Android Operating System.
2. Application must be installed on the device.

Postconditions:
1. The user successfully creates a new account.

Chapter 5. Detailed Design and Implementation

In this chapter we will present the developed system of the application. First the architecture of the system will be shown, then we will be looking into different technologies and frameworks used for creating the software product. After that, the components of the system will be detailed. Finally we will look into the structure of the RealTime database that the application has.

5.1. System Architecture

The architecture of the system is implemented using a client-server based architecture where the client side is represented by the Android application while the server side is hosted by Firebase. The communication between them is possible by using the Firebase API which is provided by the Firebase SDK along with other tools and libraries. The Firebase SDK is integrated within the implemented Android app.

Everything related to user authentication, profile of each user and the database of the app is ensured by Firebase.

The communication between the TMDb (The Movie Database) server and app is made through the TMDb API which fetches all the data needed, regarding the information about TV series in the form of a JSON object which is processed to be shown in a relevant way to the end user. More about this is presented later in the subchapter of technologies about TMDb API.

The notifications console GUI is the Firebase Console which any Firebase project has that allows the administrator to manipulate all the data involving the services of Firebase that are used by the app. For example you can send a push notification or you can change any data from the database, create a new user or delete an existing one and many more.

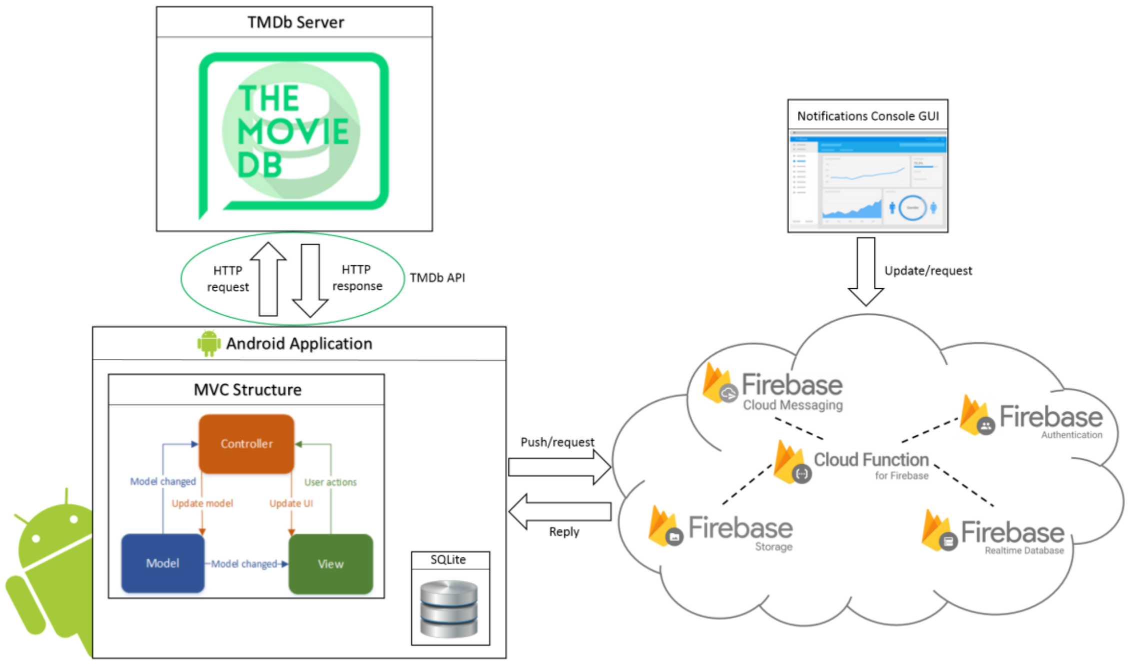


Figure 5.1 – System Architecture

5.2. Technologies used

In this subchapter the main technologies used for this project will be presented. A description about them is necessary to better understand how they were useful for our application. The project is built around a MBaaS called Firebase using four of the features that Firebase has to offer which are Cloud Messaging, Authentication, Storage and RealTime Database. This means there is no need for a server side implementation. The project also uses TheMovieDB API for getting our information regarding the TV shows using the HTTP protocol and Glide framework for mapping images easier within the application.

5.2.1. Firebase

Firebase is a cloud service designed to power real-time, collaborative applications. It is a very powerful mobile and web application development platform that provides many services escaping you all the trouble of server side implementation. Firebase becomes your server, your API and your data store, all the while being flexible for many needs. The goal of Firebase as described by L. Moroney's book is to provide the tools and infrastructure that you need to build great apps, grow a successful business, and earn from your hard work. It's not a replacement for your existing APIs for building Android, iOS, or Web apps. It's an enhancement, giving you common services that you might need – such as a database back end, secure authentication, messaging, and more. This saves you the need to build them yourself, allowing you to focus on what makes your app distinct. Additionally, it has technologies that you can put into your app and site that will help you grow your business through referrals, linking, and more. It has an easy-to-use Advertising API that you can drop into your app to start earning, and importantly, the whole platform is tied together by analytics. [18]

Firebase is a Baas (Backend as a service) which provides a lot of benefits, to name a few:

- Firebase is very easy to implement
- Firebase is cross-platform: Firebase works for web apps, Android apps and iOS apps too.
- Firebase is owned by and built on Google's infrastructure, so there is some assurance on performance and quality of service.
- Because Firebase has a large user base, it very easy to get help
- Firebase offers so many features saving a lot of time and focusing on user experience. Firebase not only offers a real time database. It also helps with authentication, analytics, cloud messaging and so much more.

Next, we will describe in detail the four main services that are used by this project, provided by Firebase.

5.2.1.1. Firebase Authentication

When a distinctive data must be given to a particular user, that user needs to be identified. The most common form is signing in which is provided by Firebase Authentication in a very easy way to integrate in the developing application.

Most apps need to know the identity of a user. Knowing a user's identity allows an app to securely save user data in the cloud and provide the same personalized experience across all of the user's devices. Firebase Authentication provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to your app. It supports authentication using passwords, phone numbers, popular federated identity providers like Google, Facebook and Twitter, and more. [19]

For authenticating in this application the way used is using email and password. Firebase Authentication also handles sending password reset emails which was implemented in the application.

The following diagram demonstrates the basic flow of data and how it travels between frontend and backend using Firebase Auth.

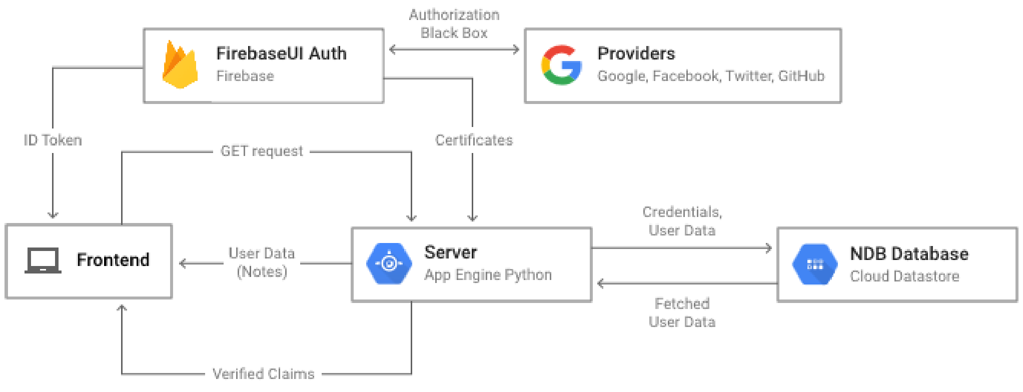


Figure 5.1 – Firebase Auth data flow diagram. [20]

5.2.1.2. Firebase RealTime Database

One of the most recognized facilities that Firebase has in store is its RealTime database that synchronizes in real time the data that is stored as JSON to every connected client as seen in figure 5.2.

This is a cloud-hosted NoSQL-based database. It provides syncing across connected devices and is available when there is no network connectivity through a local cache. It is an event-driven database that works very differently from traditional SQL databases. There's no server-side code and database access tiers; all coding is done in the client. Whenever data changes in the database, events are fired in the client code, and you can then handle and update the state of your user interface in response. It contains an expression-based rules language, called the Firebase RealTime Database Security Rules, which define how data is structured and which users have rights to that data. Its ultimate design is to be responsive, allowing you to build a real time experience that can serve users at high scale. [7]

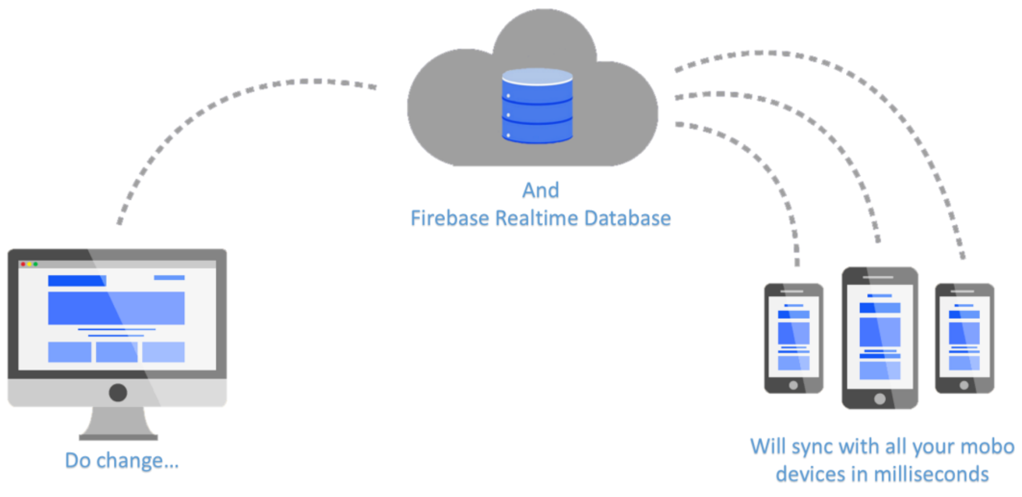


Figure 5.2 – Firebase RealTime database concept

5.2.1.3. *Firebase Cloud Storage*

Besides storing data, storing files such as photos and videos can come in handy and Firebase Cloud Storage is what this is all about. As described from the official documentation Cloud Storage for Firebase is a powerful, simple, and cost-effective object storage service built for Google scale. The Firebase SDKs for Cloud Storage add Google security to file uploads and downloads for your Firebase apps, regardless of network quality. You can use our SDKs to store images, audio, video, or other user-generated content.

The Firebase SDKs for Cloud Storage integrate seamlessly with Firebase Authentication to identify users, and we provide a declarative security language that lets you set access controls on individual files or groups of files, so you can make files as public or private as you want. [21]

For this project, the Firebase Cloud Storage helped storing the profile pictures of the users which created an account and customized their profiles.

5.2.1.4. *Firebase Cloud Messaging*

Firebase Cloud Messaging enables our application to use push notifications. The definition from the official documentation says that Firebase Cloud Messaging (FCM) is a cross-platform service that handles the sending, routing, and queuing of messages between server applications and mobile client apps. FCM is the successor to Google Cloud Messaging (GCM), and it is built on Google Play Services.

Using FCM, app servers can send messages to a single device, to a group of devices, or to a number of devices that are subscribed to a topic. [12]

Before receiving notifications a client app must first register with FCM. The following diagram (figure 5.3) describes the flow of events for the registration of the token.

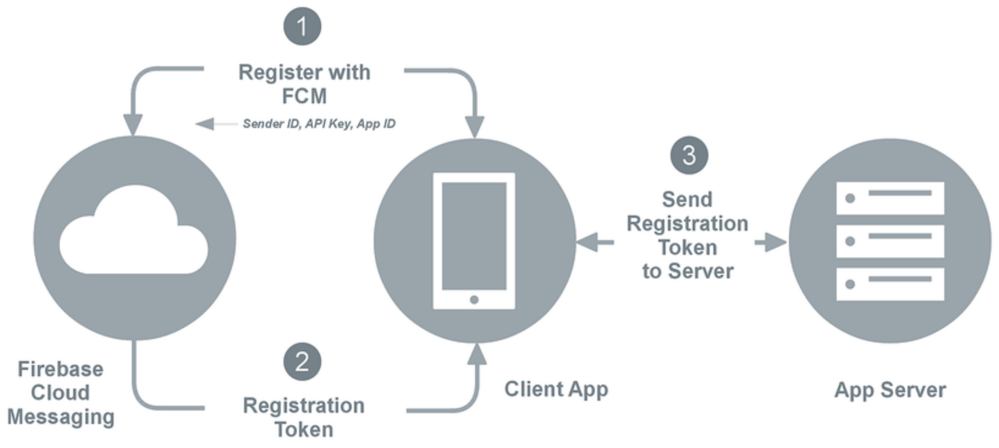


Figure 5.3 – Firebase Cloud Messaging registration diagram [22]

The diagram below (figure 5.8) describes how the downstream messaging goes from the app server to the client app according to the official doc [12]:

1. The app server sends the message to FCM.

2. If the client device is not available, the FCM server stores the message in a queue for later transmission. Messages are retained in FCM storage for a maximum of 4 weeks

3. When the client device is available, FCM forwards the message to the client app on that device.

4. The client app receives the message from FCM, processes it, and displays it to the user. For example, if the message is a remote notification, it is presented to the user in the notification area.

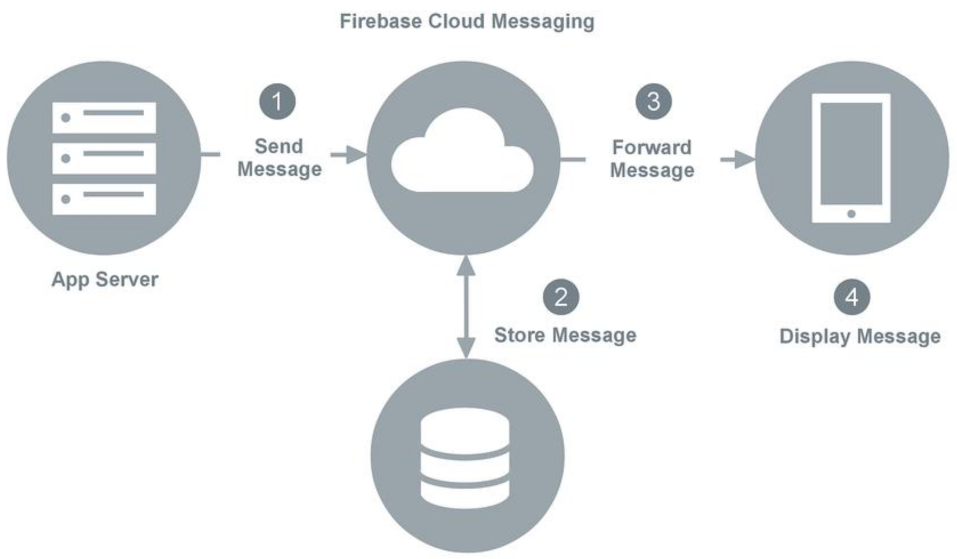


Figure 5.4 – Firebase Cloud Messaging downstream messaging diagram [12]

Our application uses the topic subscription (figure 5.9) type of sending messages as it is best fit for sending notifications to the subscribers because the range is much higher than that of the group of devices type. Topic messages can be sent via the Firebase Console Notifications GUI.

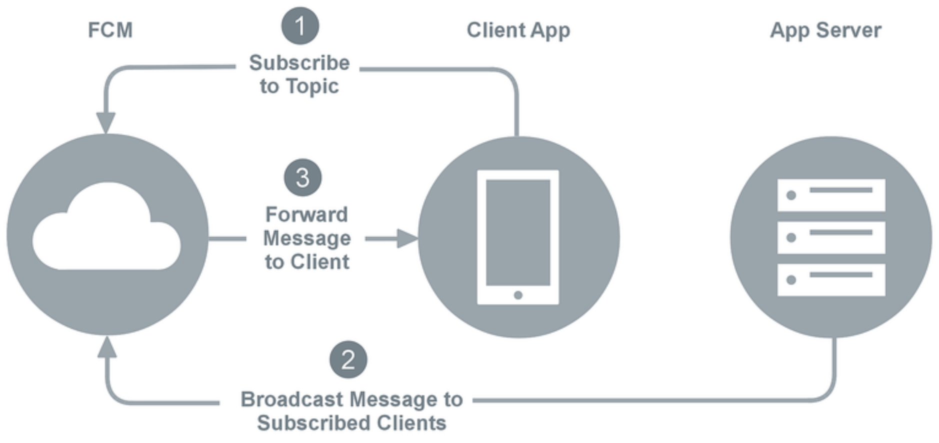

FCM
1
Subscribe to Topic
Client App
App Server
3
Forward Message to Client
2
Broadcast Message to Subscribed Clients

Figure 5.5 – Firebase Cloud Messaging topic subscription diagram [12]

5.2.2. TheMovieDB API

TheMovieDB API is an API (Application Programming Interface) used for working with TV shows like fetching data, running searches, getting recommendations, finding out what's new or what is upcoming and more.

The way the application communicates with TMDb server is through HTTP RESTful API for TMDb. This API is accessed by constructing a RESTful URI for the function of choice, for example getting details of a certain movie or searching with a key word. In order to use the API each developer must first obtain a unique API key by requesting one from TMDb website. Requests that contain an invalid API key, will receive an error, however if the key is valid TMDb returns a JSON object as a result of the request encoded by the URI. From then on, the JSON object is processed to be showed in a relevant way to the end user.

Gathering everything regarding the data of the TV shows for this application, including posters, was made possible by using TMDb API.

5.2.3. Glide Framework

Glide is a framework that was used for loading images into ImageViews in many places of the application. Glide is a fast and efficient open source media management and image loading framework for Android that wraps media decoding, memory and disk caching, and resource pooling into a simple and easy to use interface. [23]

5.3. Main components of the System

5.3.1. System architecture

In this subchapter the main components of the system will be described. Figure 5.10 is a diagram that demonstrates how the components are described and how they interact with each other. Since it is from the point of view of the user interface, it also represents how the navigation of the Views works within the application. This means that all activities or fragments that the application has are specified in this diagram.

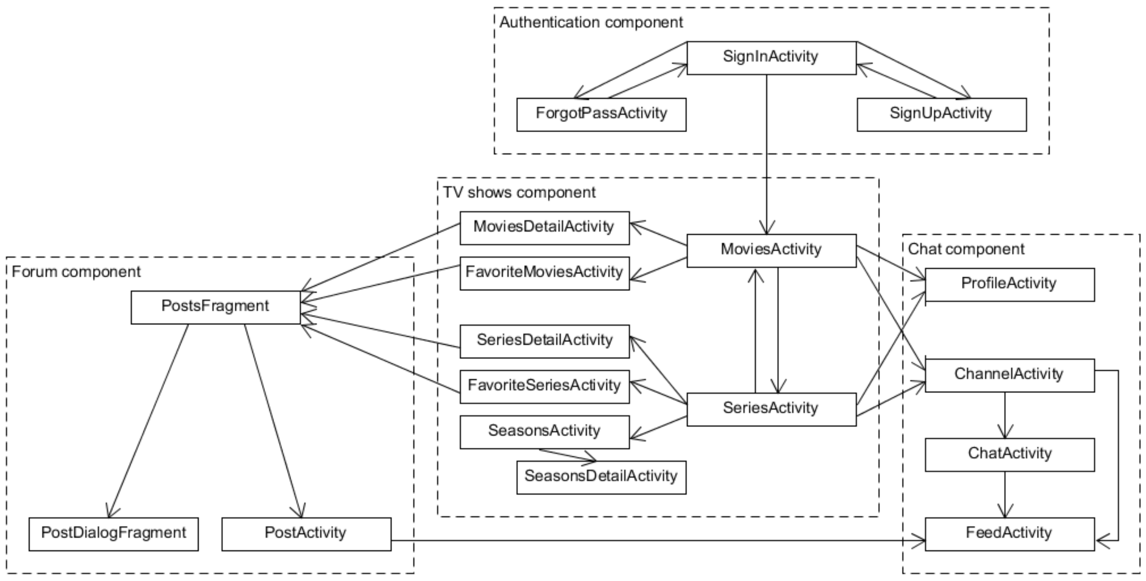


Figure 5.6 – System architecture

The main components of the application are: authentication component, TV shows component, forum component and chat component.

Next we will describe relevant or key classes from these components.

5.3.2. Authentication component

This component realizes all the task that has to do with the signing up and signing in of the application. Without it, no one can enter and use the application's functionalities.

There are two types of users: normal user and guest. A guest does not need credentials to enter the application. Anyone can sign in as guest but it has limited usage of functionality as described in the use cases from the previous chapter.

SignInActivity is the class that deals with the end user wanting to enter the application. The normal user signs in by using his email and the password he created when he signed up.

The userLogin method ensures that the user typed a valid email address and typed the password while showing a progress bar to notify the user that the sign in operation is ongoing. The method signInWithEmailAndPassword from FirebaseAuth class is then called with the checked parameters. On success, a new activity is started which is MoviesActivity and the user is directed to its associated window.

SignUpActivity operates in a similar manner except the method reigsterUser calls the createUserWithEmailAndPassword from FirebaseAuth class which will create a Firebase user within the Firebase project. Upon success, the user is redirected to the SignInActivity to login for the first time.

ForgotPassActivity is the class that has a main method called userForgotPassword which first verifies that the email typed is valid and then calls the sendPasswordResetEmail from FirebaseAuth class which sends an email with a link for the user to reset its password. Upon success the user is again redirected to the SignInActivity to login with the new password.

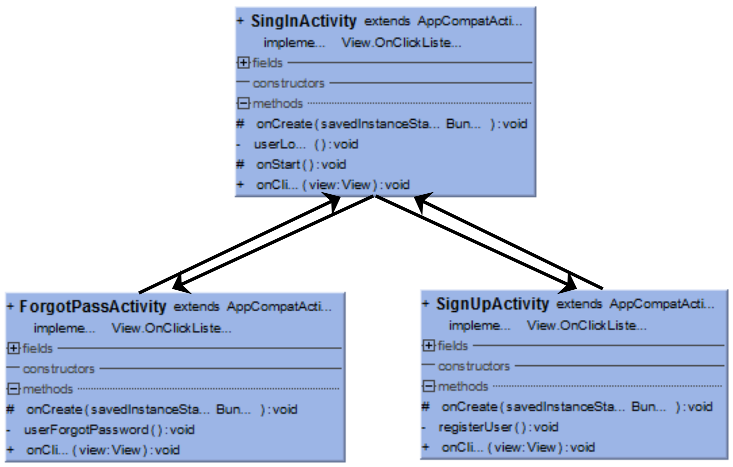

+ SingInActivity extends AppCompatActi...
impleme... View.OnClickListe...
fields
constructors
methods
# onCreate (savedInstanceSta... Bun...) : void
- userLo... ( ) : void
# onStart ( ) : void
+ onCli... (view: View) : void
+ ForgotPassActivity extends AppCompatActi...
impleme... View.OnClickListe...
fields
constructors
methods
# onCreate (savedInstanceSta... Bun...) : void
- userForgotPassword ( ) : void
+ onCli... (view: View) : void
+ SignUpActivity extends AppCompatActi...
impleme... View.OnClickListe...
fields
constructors
methods
# onCreate (savedInstanceSta... Bun...) : void
- registerUser ( ) : void
+ onCli... (view: View) : void

Figure 5.7 – Class diagram “authentication”

5.3.3. TV shows component

This component contains all the classes that are related to fetching the data via the TMDb API which means all the information regarding the movies and TV series that is gathered from the HTTP response is shown to the user in different forms through them.

Some of the information will be saved in the local database of the phone if the user decides to add a particular TV show to favorites.

MoviesActivity is the class that displays different lists of movies such as upcoming or popular with some information about them like release date or rating. The data is gathered by using a private inner class called GetMovies which is an AsynTask that uses an utill class to make a HTTP request to the specific URL that will retrieve a JSON response of that call. Using a JSONObject with the JSON retrieved, each piece of data is placed in a String element and each of them is put in a HashMap<String, String> where the key is the same as the name of that JSONObject from which the value was taken. This is done in the overridden method doInBackground of the GetMovies class.
After that, a new MyAdapter object is created, having the hashmap created previously sent as parameter. MyAdapter is a BaseAdapter that is the link between the data and the AdapterView that displays the data. In our case the AdapterView is a ListView in which we set its adapter the newly create MyAdapter object. This will populate our ListView fields with the correct data.

SeriesActivity and **SeasonsActivity** operate in a similar manner, except that they display different lists of series and the seasons of a particular series, respectively.

MoviesDetailActivity is the class that is accessed when a user presses any of the movies from any list. It contains a private inner class called GetDetails which is an AsynTask that operates in a similar manner with that of GetMovies that was described above. This class displays more information about the movie that was accessed.

A user may like or dislike the particular movie, information which is saved in the RealTime database provided by Firebase.

A user may also add to favorites the particular movie by pressing on add to favorites imageview, which resembles a shape of a heart. The setOnClickListener attached to this imageview saves the movie in the SQlite database and sets the tag of the imageview to “enable” to show to the user that the movie is added to the favorite movies sections. Also here, the method subscribeToTopic from an instance of FirebaseMessaging is called to create a topic with the name of the movie for being able to send a push notification later to this topic. If the user presses on the imageview while the setTag is on “enable” the movie will be removed from favorites, the table of the database updates, the method unsubscribeFromTopic is called to remove the subscription from this topic and the setTag is placed to default, “disable”.

Also, from this class a user may open the forum related to the particular movie, check the posts, comment on any of them or create a new post in that forum.

SeriesDetailActvity and **SeasonsDetailActivity** operate in the same manner expect that they detail the series and the season of the series, respectively. SeriesDetailActivity works the same as MoviesDetailActivity in methods, the information in the SQLite is placed in the series table. But SeasonsDetailActivity does not have a like/dislike capability neither an add to favorites one.

FavoriteMoviesActivity is a class that displays a list of all the movies that were added to favorites by a particular user and saved to the local database. When a user presses on any of the movie from the list it starts a new activity which is MoviesDetailActivity of that particular movie.

FavoriteSeriesActivity is the same as FavoriteMoviesActivity, except that it works with series.

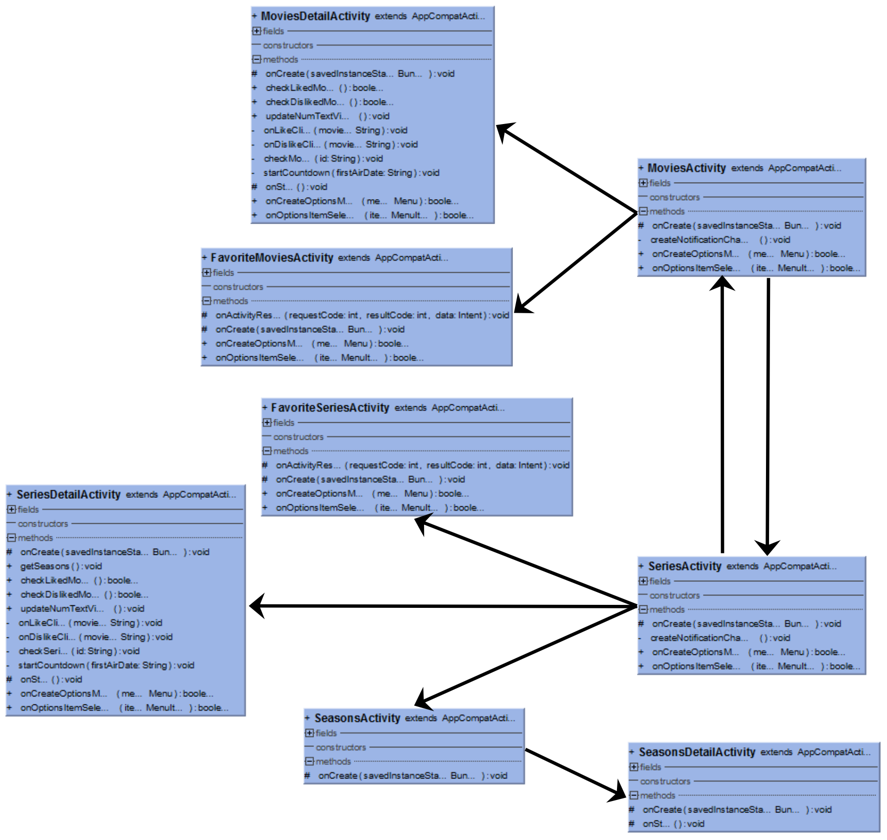

MoviesDetailActivity extends AppCompatActi...
onCreate(savedInstanceSta... Bun...):void
checkLikedMo... ():boole...
checkDislikedMo... ():boole...
updateNumTextVi... ():void
onLikeCli... (movie... String):void
onDislikeCli... (movie... String):void
checkMo... (id: String):void
startCountdown(firstAirDate: String):void
onSt... ():void
onCreateOptionsM... (me... Menu):boole...
onOptionsItemSele... (ite... MenuIt...):boole...
MoviesActivity extends AppCompatActi...
onCreate(savedInstanceSta... Bun...):void
createNotificationCha... ():void
onCreateOptionsM... (me... Menu):boole...
onOptionsItemSele... (ite... MenuIt...):boole...
FavoriteMoviesActivity extends AppCompatActi...
onActivityRes... (requestCode: int, resultCode: int, data: Intent):void
onCreate(savedInstanceSta... Bun...):void
onCreateOptionsM... (me... Menu):boole...
onOptionsItemSele... (ite... MenuIt...):boole...
FavoriteSeriesActivity extends AppCompatActi...
onActivityRes... (requestCode: int, resultCode: int, data: Intent):void
onCreate(savedInstanceSta... Bun...):void
onCreateOptionsM... (me... Menu):boole...
onOptionsItemSele... (ite... MenuIt...):boole...
SeriesDetailActivity extends AppCompatActi...
onCreate(savedInstanceSta... Bun...):void
getSeasons():void
checkLikedMo... ():boole...
checkDislikedMo... ():boole...
updateNumTextVi... ():void
onLikeCli... (movie... String):void
onDislikeCli... (movie... String):void
checkSeri... (id: String):void
startCountdown(firstAirDate: String):void
onSt... ():void
onCreateOptionsM... (me... Menu):boole...
onOptionsItemSele... (ite... MenuIt...):boole...
SeriesActivity extends AppCompatActi...
onCreate(savedInstanceSta... Bun...):void
createNotificationCha... ():void
onCreateOptionsM... (me... Menu):boole...
onOptionsItemSele... (ite... MenuIt...):boole...
SeasonsActivity extends AppCompatActi...
onCreate(savedInstanceSta... Bun...):void
SeasonsDetailActivity extends AppCompatActi...
onCreate(savedInstanceSta... Bun...):void
onSt... ():void
fields
constructors
methods

Figure 5.8 – Class diagram “TV shows”

5.3.4. *Forum component*

This component contains the classes that are related to showing and creating a forum for any of the TV shows.

PostsFragment is the class that populates the RecyclerView with all the posts of that pariculat TV show from the RealTime database. It uses the ViewHolder design pattern which is useful for speeding up rendering the View because it does not call findViewById() frequently during the scrolling of the View, which can slow down performance.

The method setupAdapter creates a FirebaseRecyclerAdapter<Post, PostHolder> where Post is a model which implements Serializable and PostHolder is a static inner class which is a RecyclerView.ViewHolder.

The class also has a like button, and just like the like/dislike buttons from the TV shows, the information about likes is saved in the RealTime database but on a different node, “post_liked”.

PostDialogFragment is the class that displays the detail of creating a new post. The title, a possible image, and the content.

The method sendPost saves all the information that was input by the user in a Post model object, which is then saved as a value in the node “posts” of the RealTime database, the child node being the current user and the subchild being the actual value of the newly created post.

PostActivity is the class which is called when a user enters the comment section of that particular post. The comment section is initialized by using a FirebaseRecyclerAdapter<Comment, CommentHolder> where Comment is a model which implements Serializable and the ViewHolder design pattern.

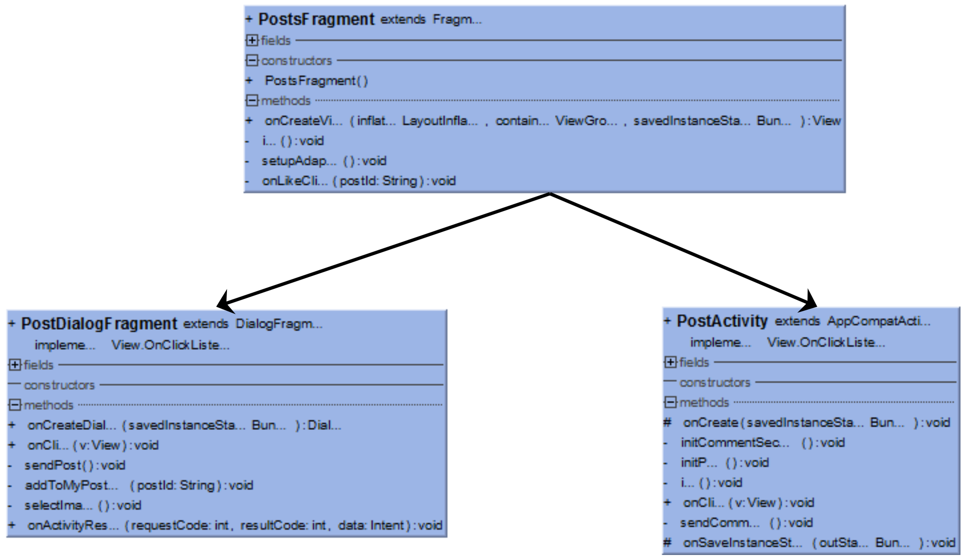

+ PostsFragment extends Fragm...
fields
constructors
+ PostsFragment()
methods
+ onCreateVi... (inflat... LayoutInfla... , contain... ViewGro... , savedInstanceSta... Bun...):View
- i... ():void
- setupAdap... ():void
- onLikeCli... (postId: String):void
+ PostDialogFragment extends DialogFragm...
impleme... View.OnClickListe...
fields
constructors
methods
+ onCreateDial... (savedInstanceSta... Bun...):Dial...
+ onCli... (v: View):void
- sendPost():void
- addToMyPost... (postId: String):void
- selectIma... ():void
+ onActivityRes... (requestCode: int, resultCode: int, data: Intent):void
+ PostActivity extends AppCompatActi...
impleme... View.OnClickListe...
fields
constructors
methods
# onCreate(savedInstanceSta... Bun...):void
- initCommentSec... ():void
- initP... ():void
- i... ():void
+ onCli... (v: View):void
- sendComm... ():void
# onSaveInstanceSt... (outSta... Bun...):void

Figure 5.9 – Class diagram “Forum”

5.3.5. Chat component

This component has to do with how two users can communicate via private messages within the application but also how the feed of the user looks like to other users.

ProfileActivity is the class that contains the settings of the application. Here, the user can edit his/her display name, set a profile picture, change password or delete account. The display name and profile picture is saved into the RealTime database in the node users and the child being the email of the user.

The password is the one from the FirebaseAuth which is not existent in the RealTime database so it secured by the Firebase authentication.

ChannelActivity is the class where the user can find his/her conversations, pending friend requests and friend list. Each three of them use the FirebaseRecyclerAdapter and the ViewHolder design pattern to populate the RecyclerView with the data from the RealTime database.

A new chat room can only be created by pressing on the friend from the friend list. Then, the user will be asked for the title of the new room.

The method searchUser is attached to the setOnClickListener of the searchUser button which asks the user to type the email of the user he wants to search. The email typed is checked to be valid and then checked in the RealTime database if a child with that email exists in the node users. If yes, the user is redirected to the FeedActivity of that user, otherwise the user is notified that the searched user does not exist.

The static method sendPendingRequest from FireBaseUtils is called in the seOnClickListener of the addFriend button which asks the user for the email of the user that he/she wants to send the friend request to. Then the email is validated and the constraints are checked by a static method from FireBaseUtils. The constraints include: valid email, check if email exists in RealTime database, check if email is identical with the current user’s email, check if pending was already sent, check if users are already friends. If every constraint is met, the pending request is sent.

ChatActivity is the class which has to do with the chat room between two users. The View contains a send button and an EditText where the user can type its message to send to the other user. The rest of the View is a RecyclerView which is populated with data from the RealTime database by using the ViewHolder design pattern and a FirebaseRecyclerAdapter<Comment, CommentHolder>. The Comment class is the same model used in the comments for a post.

FeedActivity is the class that displays the profile of another user which includes the display name, the profile picture and the email and also a list of that user’s activity, about what posts he/she created and what comments he/she made. The list is again a RecyclerView which is populated in the same manner as described at ChatActivity, except that the model is an object of the Feed class. If a user presses any of the posts or comments it will be redirected to that posts forum or inside the post the comment was made.

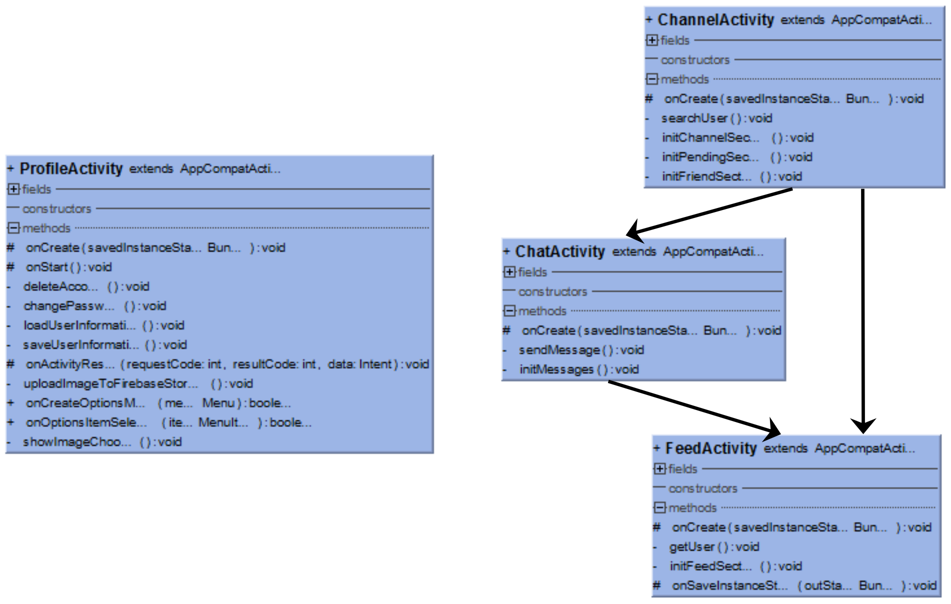

+ ChannelActivity extends AppCompatActi...
fields
constructors
methods
# onCreate (savedInstanceSta... Bun...) : void
- searchUser ( ) : void
- initChannelSec... ( ) : void
- initPendingSec... ( ) : void
- initFriendSect... ( ) : void
+ ProfileActivity extends AppCompatActi...
fields
constructors
methods
# onCreate (savedInstanceSta... Bun...) : void
# onStart ( ) : void
- deleteAcco... ( ) : void
- changePassw... ( ) : void
- loadUserInformati... ( ) : void
- saveUserInformati... ( ) : void
# onActivityRes... (requestCode: int, resultCode: int, data: Intent) : void
- uploadImageToFirebaseStor... ( ) : void
+ onCreateOptionsM... (me... Menu) : boole...
+ onOptionsItemSele... (ite... MenuIt...) : boole...
- showImageChoo... ( ) : void
+ ChatActivity extends AppCompatActi...
fields
constructors
methods
# onCreate (savedInstanceSta... Bun...) : void
- sendMessage ( ) : void
- initMessages ( ) : void
+ FeedActivity extends AppCompatActi...
fields
constructors
methods
# onCreate (savedInstanceSta... Bun...) : void
- getUser ( ) : void
- initFeedSect... ( ) : void
# onSaveInstanceSt... (outSta... Bun...) : void

Figure 5.10 – Class diagram “Chat”

5.4. Structure of the RealTime database

This subchapter describes how the RealTime database data is stored and how the database of our project is structured. RealTime database is a noSQL cloud-hosted database.

5.4.1. How data is stored

All Firebase RealTime database data is stored as JSON objects. You can think of the database as a cloud-hosted JSON tree. Unlike a SQL database, there are no tables or records. When you add data to the JSON tree, it becomes a node in the existing JSON structure with an associated key. You can provide your own keys, such as user IDs or semantic names, or they can be provided for you using push(). [24]

Since sometimes the keys provided are emails, they must respect some constraints which are: they must be UTF-8 encoded, can be a maximum of 768 bytes, and cannot contain ., $, #, [,], /, or ASCII control characters 0-31 or 127.

That is why any key that has an email as a node will not contain any dot, but instead the application will convert the dot into a comma.

One of the best practices of structuring data is avoiding nesting data. The official docs [24] says that because the Firebase RealTime Database allows nesting data up to 32 levels deep, one might be tempted to think that this should be the default structure. However, when one fetches data at a location in the database, he also retrieves all of its child nodes. In addition, when one grants someone read or write access at a node in the database, he also grants them access to all data under that node. Therefore, in practice, it's best to keep the data structure as flat as possible.

5.4.2. Structure of the project's database

Having in mind how the data is stored and the best practices by avoiding nested data there is a larger number of nodes. It is important to organize data such that you best fit your needs and this can be accomplished by analyzing before how you need to retrieve the said data.

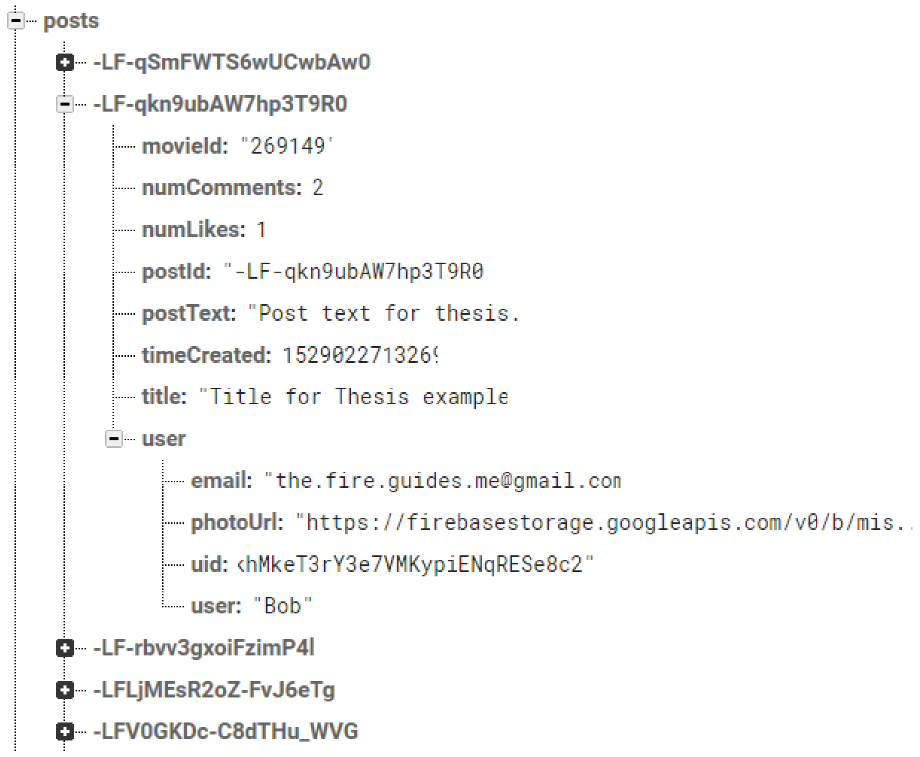
There are 11 main nodes in the database which are: chat_channels, chats, comments, friends, friend_requests, show_contour, show_liked, post_liked, posts, user_record and users.

- chat_channels – is the node in which information about chat rooms is stored, a child will be the email of the user which is part of the chat and the subchild is a Channel model which contains information about the other user and the room itself. This same subchild is created in the child of the other user in the same manner.
- chats – is the node in which the actual messages are stored. The structure is that of a channel id, which was created in the chat_channels, which contains subchilds of messages with the necessary information, like the text, the date and which user typed it.
- comments – this node contains information in the form of post id child, where each post id node has subchilds of the Comment model, with the necessary information.
- friends – this node contains emails as child, and each child has subchilds in the form of User model, which are basically the list of friends that user with the email has.
- friend_requests – the node contains emails as child as well, but the subchilds are the model type of Channel which contains information about the user who sent the request.
- show_contour – each node is the id of a movie or series, and each node contains a contour of number of likes and number of dislikes.
- show_liked and post_liked – both contain an email of a user as node, and a child of a Boolean which lets us know if the user already liked a show or not.
- posts – each node is a post itself, with information regarding itself.
- user_record – each node is an email, and each child is a Feed model, with information about what a user posted or on what post he/she commented on. This main node will help us create the feed section for each user.
- users – contains all the users of the application in the form of emails as nodes, and each node contains information regarding that user.

Let's see a more detailed example of how and why some data is duplicated in the database.

The example is how posts are written into the database. When a post is created, the values of the post itself are stored in a new child post, under the "posts" node, figure 5.14.

Each post contains the necessary details about it and also a nested data about the details of the user. This is needed because we need to know which is the display name chosen by the user and also the profile picture in order to display it in a little CircleImageView.



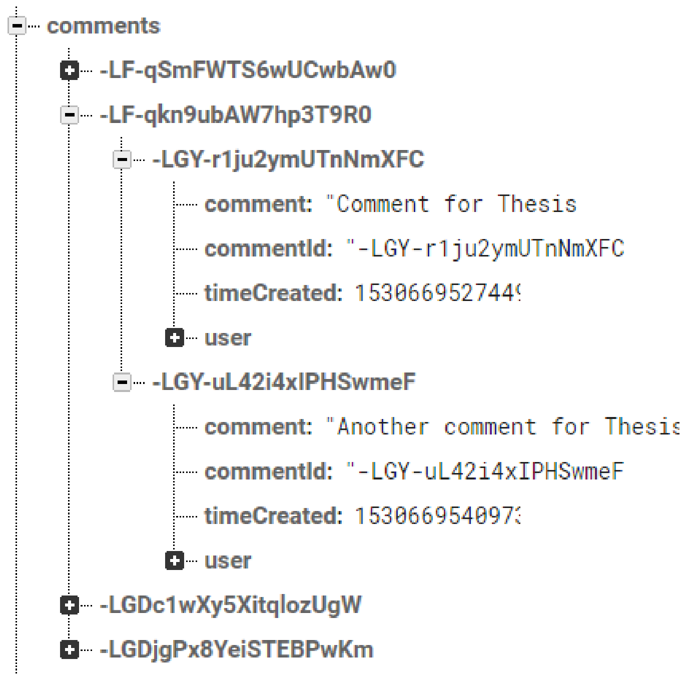
```
posts
  -LF-qSmFWTS6wUCwbAw0
  -LF-qkn9ubAW7hp3T9R0
    movieId: "269149'
    numComments: 2
    numLikes: 1
    postId: "-LF-qkn9ubAW7hp3T9R0
    postText: "Post text for thesis.
    timeCreated: 152902271326
    title: "Title for Thesis example
    user
      email: "the.fire.guides.me@gmail.con
      photoUrl: "https://firebasestorage.googleapis.com/v0/b/mis..
      uid: "hMkeT3rY3e7VMKypiENqRESe8c2"
      user: "Bob"
  -LF-rbvv3gxoiFzimP4l
  -LFLjMEsR2oZ-FvJ6eTg
  -LFV0GKDc-C8dTHu_WVG
```

Figure 5.11 – Posts database structure

The auto-generated name of this child is saved under the name of postId because it will be needed when a new comment is added to this post.

To continue the example, this post contains 2 comments, as seen also from the value of “numComments”. The auto-generated name of the post we discussed appears also in the node of “comments” (figure 5.15) being a child here too with the same value but now every subchild of it is a comment. Each comment contains the text, the id, the time created and the information about the user that commented. This information is again needed, because we need to provide the display name of the user and also its profile picture in a little CircleImageView.

By structuring the data like this, it is easy to fetch all the comments of a single post and populate them in the RecyclerView which is exactly what we needed.



```
comments
    -LF-qSmFWTS6wUCwbAw0
    -LF-qkn9ubAW7hp3T9R0
        -LGY-r1ju2ymUTnNmXFC
            comment: "Comment for Thesis
            commentId: "-LGY-r1ju2ymUTnNmXFC
            timeCreated: 153066952744
            user
        -LGY-uL42i4xIPHSwmeF
            comment: "Another comment for Thesis
            commentId: "-LGY-uL42i4xIPHSwmeF
            timeCreated: 153066954097
            user
    -LGDc1wXy5XitqlozUgW
    -LGDjgPx8YeiSTEBPwKm
```

Figure 5.12 – Comments database structure

After working with RealTime database I have concluded it is better to maintain complexity when writing to the database because you have more control over how things are written. But when reading, you have little control because when you take a node, you have to take everything under the node.

Chapter 6. Testing and Validation

This chapter presents how the testing and validation of the application has been made in order to ensure the correctness of the application's functionalities.

During the development, the testing of the system was established for each new component that was about to be finally added to the project. This was a necessary task in order to find bugs, maintain and check that functionalities are in accordance to the system's requirements and to check if the application failed to act in its desired parameters.

The testing and validation of the system was done from the beginning of the implementation until the end, not only when the final product was finished, but it was present between each stages of iteration.

6.1. Manual testing

Due to the nature of the application, being a mobile app where the most common testing is the manual step by step testing of the functional requirements and being dependent on the data provided by the TMDb API and Firebase API, the application was mostly manually tested during development, following test scenarios that were created based on the system's requirements.

Up next, a relevant test scenario will be described which was verified within the application.

Test scenario:
Title: User has to sign in by using his/her email and password authentication
Scenario 1:

Step 1: The user enters the application and the sign in layout is visible.

Step 2: The user presses the login button.

Expected result: The application notifies the user that email is required and the email field is focused.

Scenario 2:

Step 1: The user enters the application and the sign in layout is visible.

Step 2: The user types in text but not an email format.

Step 3: The user presses the login button

Expected result: The application notifies the user that the email is not valid and the email field is focused.

Scenario 3:

Step 1: The user enters the application and the sign in layout is visible.

Step 2: The user types the email.

Step 3: The user presses the login button

Expected result: The application notifies the user that password is required and the password field is focused.

Scenario 4:

Step 1: The user enters the application and the sign in layout is visible.

Step 2: The user types the email and password but password is shorter than 6 character.

Step 3: The user presses the login button

Expected result: The application notifies the user that the minimum length of password is 6 characters and the password field is focused.

Scenario 5:

Step 1: The user enters the application and the sign in layout is visible.

Step 2: The user types the email and password but password is shorter than 6 character.

Step 3: The user presses the login button

Expected result: The application notifies the user that the minimum length of password is 6 characters and the password field is focused.

Scenario 5:

Step 1: The user enters the application and the sign in layout is visible.

Step 2: The user types the email and password correctly.

Step 3: The user presses the login button

Expected result: The user successfully logins within the application.

6.2. Automated testing

Establishing automated testing for a system requires a testing tool. These tools are meant to control the execution of tests and then compare them with expected results.

Since the project is Firebase embedded, it was a clear choice to user an automated testing provided by Firebase Test Lab.

Firebase Test Lab [25], as the official documentation says is a cloud-based app-testing infrastructure. With one operation, you can test your Android or iOS app across a wide variety of devices and device configurations, and see the results—including logs, videos, and screenshots—in the Firebase console.

There are even several testing tools that Firebase has to offer, instrumentation tests Robo test and game loop tests which is still in beta.

Instrumentation tests [26] are tests that were written specifically to test the app, using the Espresso and UI Automator 2.0 Android test frameworks.

The one used for this application was Robo Test because it provides a rather new way of testing and it fit best to test this kind of application.

Robo Test [26] , as the official documentations says is a test tool that is integrated with Firebase Test Lab. Robo test analyzes the structure of your app's UI and then explores it methodically, automatically simulating user activities. Robo test always simulates the same user activities in the same order when you use it to test an app on a specific device configuration with the same settings. This lets you use Robo test to validate bug fixes and test for regressions.

Not only this, but Robo Test captures log files, saves a series of annotated screenshots, and then creates a video from those screenshots to show the developer how the simulated user operations was performed.

The following graph (figure 6.1), is a small part of the crawl graph created by Robo test when the application's apk was provided to it and it performed its tests on a Samsung Galaxy Nexus 7:

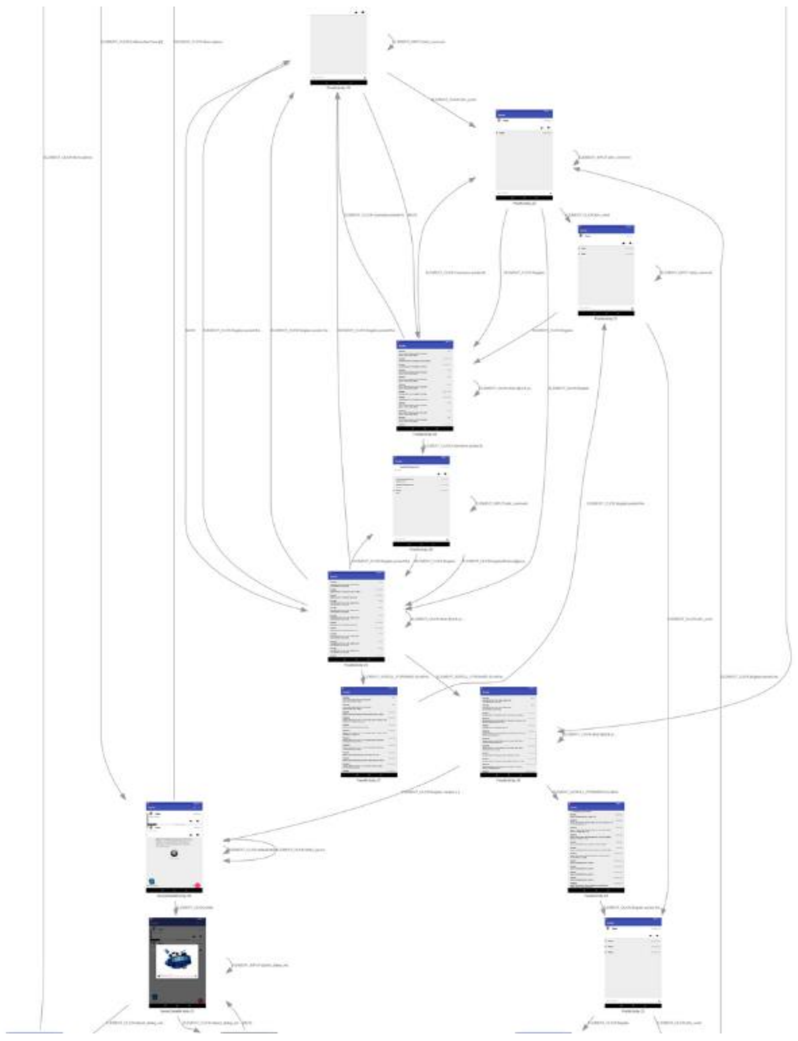


Figure 6.1 – Crawl graph sample

As it can be seen, the testing tool searches for everything and explores user activities, pressing on anything from the UI that a real user might press, in order to find bugs or errors which will then be detailed in an error log for the developer to check.

There is also a video provided simulating the start to end of the testing which helps understand the crawl graph faster or find where the application crashed.

Moreover, if there needs to be more control about the test, or a specific user journey Robo test can accomplish this by using robo scripts. For using robo scripts the official doc [26] says that you need to record yourself walking through a workflow in your app, then upload that recording to the Firebase console to run in Robo tests. When you run a Robo test with a script attached, Robo first steps through your pre-scripted actions then proceeds to explore the app as usual.

Chapter 7. User's manual

This chapter will describe the minimum resources that I needed for the application to successfully run, but also a step by step description of how the application should be installed and used.

7.1. Application requirements

Hardware and software requirements for the successful installation and use of the application:

- A smartphone having an operating system of Android with a minimum version of 4.0.3 or higher
- Minimum of 512 MB of RAM
- Intel Atom® Processor Z2520 1.2 GHz or faster processor
- 200 MB of free space
- Google Play Store available on the device

The application should be available from the Google Play Store where he/she will be able to download the app.

7.2. Application utilization

7.2.1. Authentication

Before being able to use the app, the user must first create an account. The first layout that the user will find is the sign in layout, figure 7.1.

By pressing the sign up button the user will be redirected to the sign up layout which is similar and the user will provide his/her email and a password for his account.

If the user provides an invalid email it will be notified with the corresponding message, same as if the password is too short.

After successfully signing up, the user will be redirected again to the sign in layout. Now he/she can sign in using the credentials that were created from the sign up layout.

If the user forgot his/her password, he can press the "Forgot password?" field where he/she will be asked to type the email where the password reset will be sent. After typing the email and sending it, an email will be received that will further instruct the user how to reset the password.

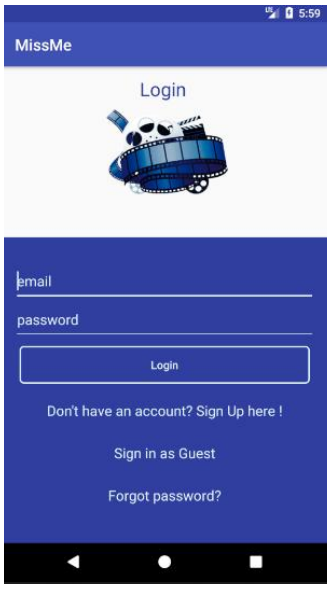
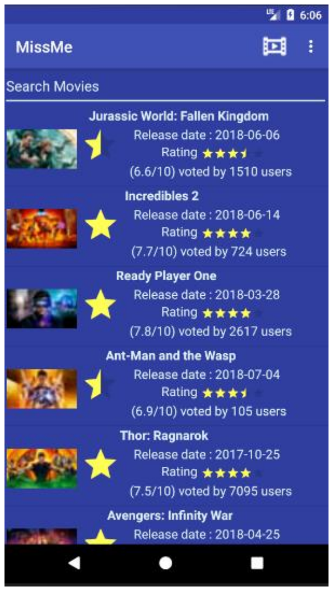


Figure 7.1 – Authentication

7.2.2. Finding movies or series

By using the search bar, depending on what part of the app the user is, movies or series, the user can search for any TV show he/she can think of by typing its title and the app will provide a list of the matching or close to matching text.

Navigating through different lists is very easy, the user only has to open the menu and press on any list he/she desires to see and the list will update with the specified condition, like top rated movies or upcoming movies. For example, figure 7.2 shows the most popular movies.



Jurassic World: Fallen Kingdom
Release date : 2018-06-06
Rating
(6.6/10) voted by 1510 users

Incredibles 2
Release date : 2018-06-14
Rating
(7.7/10) voted by 724 users

Ready Player One
Release date : 2018-03-28
Rating
(7.8/10) voted by 2617 users

Ant-Man and the Wasp
Release date : 2018-07-04
Rating
(6.9/10) voted by 105 users

Thor: Ragnarok
Release date : 2017-10-25
Rating
(7.5/10) voted by 7095 users

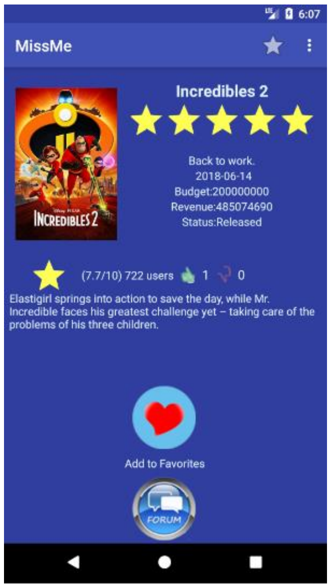
Avengers: Infinity War
Release date : 2018-04-25

Figure 7.2 – Movie list

7.2.3. Adding a TV Show to favorites

When a user browses to a list of any TV show, if any of them seems interesting, the user may press on it and the app will open a more detailed layout about that TV show (figure 7.3). There, the user will find a button that says underneath,” Add to favorites”. If the user desires to add this particular TV show to favorites, he/she just needs to press that button and the heart from inside the button will turn red along with a notification that this TV show was added to favorites.

The list of favorites can be checked in the „My favorite movies” for movies and „My favorite series” for series.



Elastigirl springs into action to save the day, while Mr. Incredible faces his greatest challenge yet – taking care of the problems of his three children.

Figure 7.3 – Adding to Favorites

7.2.4. Creating a post

For creating a post on a particular forum, a user must first go to the detailed layout of the TV show forum he/she wants to create the post. There, by pressing the „forum” button, all posts from that forum will appear along with a circle button for creating a new post which resembles a paper in a blue background. The user will press that button and a window will appear (figure 7.4) that will instruct him/her what the fields that must be completed are. Those are the title, the text and optionally an image. After completing the fields, the user will press on the send button which is find on the right corner and the user will be notified by a progress dialog that the post is being sent. After the post is sent the user should automatically see his/her newly created post in that forum.

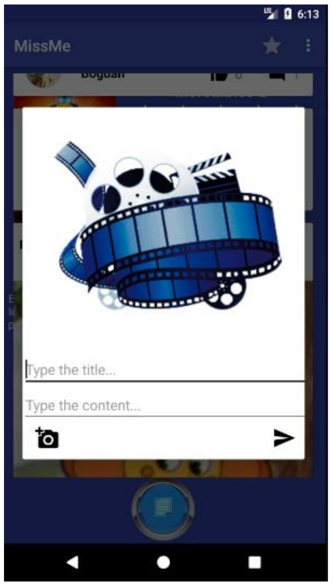
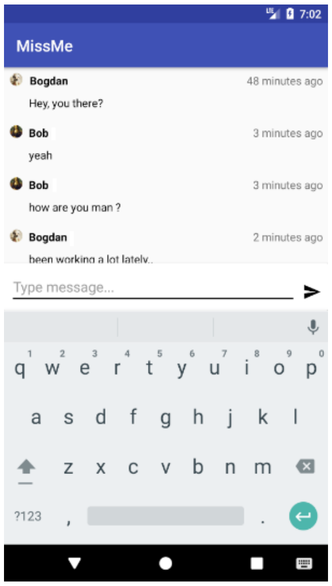


Figure 7.4 – Create a post

7.2.5. *Chatting with another user*

In order to chat with another person, the user must first send a friend request to that user by either pressing the „Add friend” button from the conversations layout or by pressing the two hands shaking icon on that user’s feed. Assuming the other user accepted the friend request, the current can check the list of friends from the conversation layout by pressing the „Friends” button. There, the new friends will be visible and the user will press any item from the list. After pressing, the user will be prompted with a message to provide a title for the new chat room. After pressing „ok”, the new chat room has been created and the user will be able to see it in the chats sections, by pressing the „Chats” button. The user will press on the chat room and from then on he can type any message and send it and the other user will receive his/her messages like in figure 7.5.



Bogdan 48 minutes ago
Hey, you there?

Bob 3 minutes ago
yeah

Bob 3 minutes ago
how are you man ?

Bogdan 2 minutes ago
been working a lot lately..

Figure 7.5 – Chat room

Chapter 8. Conclusions

This chapter will first present the obtained results in contrast with the set of objectives that were defined at the beginning of the project. Secondly, possible improvements will specified that will further improve the application.

8.1. Results

The application created mostly accomplishes the sets of objectives which were the goal of the app as we have created an app which can be a TV Show app, separately, but also a social app allowing users to express their thoughts on TV shows on both forums, which are public and on chat rooms, which are private.

By having a friend request system, users with similar tastes in TV Shows can become friends and chat privately. But also, no user can bother another with private messages, unless they are friends. That is why the chat rooms are restricted only for friends.

Having a backend as service, the application is reliable to function correctly all the time and also reliable in its security. Firebase is backed by Google and it is very secure, so data is very hard to be lost or wrong data would be impossible to arrive at the wrong user.

8.2. Future developments

A first possible improvement would be to have multiple ways of accessing the application, via Google account or Facebook, not forcing the user to create a special account just for this application.

A second possible improvement would be to make spoilers avoidable by allowing users to only see forums in which they marked the TV show as watched. This would not put a user to risk of being spoiled about a scene or a detail about a TV show. To implement this, a marked as watch icon should be provided in the details screen of a TV show.

Another improvement could be regarding the data gathered, which is currently from TMDb server. By retrieving data about TV shows from multiple sources and letting the user decide from which source he/she wants to see the statistics and reviews from critics or just simple users.

Depending where the application is heading and evolving, it can be improved in aspects like being able to post images with comments, having chat rooms with multiple friends joining, expanding the feed of a user by showing what shows he/she liked or what shows were added to favorites, improving the control of the feed of a user by allowing him/her to choose which activity should appear in the feed and many others.

Bibliography

[1] G. Reese, “Cloud Computing Architectures”, O'Reilly Media, 2009
[2] "Cloud computing", 22 April. 2018. [Online]. Available:
https://ro.wikipedia.org/wiki/Cloud_computing
[3] P. Mell, T. Grance, “The NIST Definition of Cloud Computing”, 2011
[4] “Cloud deployment models”, 22 March. 2018. [Online]. Available:
https://dzone.com/articles/cloud-computing-deployment-models
[5] “Clou Service Models”, 28 April 2018 [Online] Available:
https://www.uniprint.net/en/7-types-cloud-computing-structures/
[6] “Mobile Backend as Service” 22 March. 2018. [Online]. Available:
http://apievangelist.com/2012/06/03/rise-of-mobile-backend-as-a-service-mbaas-api-stacks/
[7] “MBaaS facilities”, 28 April 2018 [Online] Available:
https://www.softwebmobility.com/mobile-backend-as-a-service/
[8] “Push notifications” 22 March. 2018. [Online]. Available:
https://searchmobilecomputing.techtarget.com/definition/push-notification
[9] “Push notification flow” 28 April 2018 [Online] Available:
https://www.pivotaltracker.com/blog/how-the-tracker-team-uses-pivotal-push-notification-service/
[10] “Next Episode” 16 March. 2018. [Online]. Available:
https://play.google.com/store/apps/details?id=net.nextepisode.android&hl=en
[11] “TV Show Time” 16 March. 2018. [Online]. Available:
https://play.google.com/store/apps/details?id=com.tozelabs.tvshowtime&hl=en
[12] “Firebase Cloud Messaging” 22 April. 2018. [Online]. Available:
https://docs.microsoft.com/en-us/xamarin/android/data-cloud/google-messaging/firebase-cloud-messaging
[13] “Cloud Functions” 22 April. 2018. [Online]. Available:
https://firebase.google.com/docs/functions/use-cases
[14] R. Meier, “Professional Android 4 Application Development 3rd Edition”, Wrox, 2012
[15] J. Friesen, “Learn Java for Android Development”, Apress, 2010
[16] “Android software stack” 14 May 2018 [Online] Available:
http://atozofandroid.blogspot.com/2010/10/understanding-android-software-stack.html
[17] “Android activity lifecycle” 14 May 2018 [Online] Available:
http://thegeekyway.com/android-activity-lifecycle/
[18] L. Moroney, “The Definitive Guide to Firebase” Seattle, USA: Apress, 2017,
[19] “Firebase Authentication” 22 April. 2018. [Online]. Available:
https://firebase.google.com/docs/auth/
[20] “Firebase Auth” 14 May 2018 [Online] Available:
https://cloud.google.com/appengine/docs/standard/python/authenticating-users-firebase-appengine
[21] “Firebase Cloud Storage” 22 April. 2018. [Online]. Available:
https://firebase.google.com/docs/storage/
[22] “Firebase Cloud Messaging” 14 May 2018 [Online] Available:

https://docs.microsoft.com/en-us/xamarin/android/data-cloud/google-messaging/firebase-cloud-messaging
[23] “Glide Framework” 24 May 2018 [Online] Available:
https://github.com/bumptech/glide
[24] “Database Structure” 26 May 2018 [Online] Available:
https://firebase.google.com/docs/database/android/structure-data
[25] “Firebase lab test” 27 May 2018 [Online] Available:
https://firebase.google.com/docs/test-lab/
[26] “Firebase Robo Test” 27 May 2018 [Online] Available:
https://firebase.google.com/docs/test-lab/android/robo-ux-test

Appendix 1. Glossary

NIST	National Institute of Standards and Technology
MBaaS	Mobile backend as a service
API	Application Programming Interface
SDK	Software Development Kit
HTTP	Hypertext Transfer Protocol
XMPP	Extensible Messaging and Presence Protocol
GUI	Graphical User Interface
FCM	Firebase Cloud Messaging
DEX	Dalvik Executable
TMDb	The Movie Database