ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδιασμός και Ανάπτυξη Μικρο-Εφαρμογών Κοινωνικής Δικτύωσης/

Design and Development of Mini-Social Networks

Διπλωματική Εργασία

Καταράκης Δημήτριος

ΕΠΙΒΛΕΠΩΝ/ΟΝΤΕΣ:

Τσομπανοπούλου Παναγιώτα

Βασιλακόπουλος Μιχαήλ

Βόλος 2018

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Design and Development of Mini-Social Networks

Diploma XXXXXXXX

Katarakis Dimitrios

Supervisor/s:

Panagiota Tsompanopoulou

Michael Vassilakopoulos

Volos 2018

ΕΥΧΑΡΙΣΤΙΕΣ ή ΣΧΟΛΙΑ

Θα ήθελα να ευχαριστήσω θερμά την κ Αναπληρώτρια καθηγήτρια του Πανεπιστημίου Θεσσαλίας κα Τσομπανοπούλου Παναγιώτα για την καθοδήγησή της και την κατανόησή της στις δύσκολίες που αντιμετώπισα. Χωρίς την υπομονή της και την ευγενή αντιμετώπισή της η ολοκλήρωση της εργασίας αυτής θα ήταν αδύνατη. Επίσης, θα ήθελα να ευχαριστήσω τον φίλο και συνάδελφο κ. Κωνσταντίνο Μάριο Ρουμπέα για την πολύτιμη συνδρομή του. Τέλος θα ήθελα να εκφράσω από τα βάθη της καρδιάς μου την αγάπη και ευγνωμοσύνη μου στην οικογένειά μου για τη συμπαράστασή τους και τη στήριξή τους και ιδιαίτερα τη μητέρα μου στην οποία αφιερώνω την εργασία αυτή.

Βόλος, Ιούλιος 2018

Καταράκης Δημήτριος

ΠΕΡΙΛΗΨΗ

Η αλματώδης τεχνολογική εξέλιξη, η καθιέρωση φτηνού και γρήγορου διαδικτύου και η μείωση του κόστους υλοποίησης εφαρμογών έχουν αλλάξει ριζικά την καθημερινότητα του καταναλωτή. Η ενημέρωση, δικτύωση, επικοινωνία και διεκπεραίωση καθημερινών αναγκών μέσω έξυπνων συσκευών, κινητών, tablet κτλ. έχουν δημιουργήσει νέες καταναλωτικές ανάγκες και αποτελούν πρόσφορο έδαφος για την βιομηχανία εφαρμογών. Η ανάγκη για συνεχή παρουσία στο διαδίκτυο και εξατομίκευση των εφαρμογών ώστε να ανταποκρίνονται στις ανάγκες του κάθε ατόμου αποτελούν βάση ανάπτυξης για την εργασία αυτή. Σκοπός της εργασίας είναι η δημιουργία ενός template, δηλαδή μιας λειτουργικής, βασικής μορφής εφαρμογής, η οποία θα παρουσιάζει τις βασικές λειτουργίες τις οποίες κάθε κοινωνικό δίκτυο θα πρέπει να περιλαμβάνει και θα αποτελεί τη βάση για την υλοποίηση πιο εξεζητημένων εφαρμογών που θα ανταποκρίνονται σε συγκεκριμένες ομάδες και θα υλοποιούν στοχευμένες λύσεις.

ABSTRACT

The rapid technological advancement, the introduction of cheap and fast internet connections and the reduction of application costs have fundamentally changed the consumer's daily life. Performing everyday tasks, communicating and meeting daily quotas through mobile devices, tablets, etc. have introduced new consumer needs and present new opportunities for the application industry. The need for continuous web presence and the personalization of applications to meet the needs of each individual is a field of interest for this thesis. The purpose of the thesis is to create a template, a functional, basic application form, which will explore the functions that every social network should include and will be the grounds for implementing more sophisticated applications that respond to specific groups and will implement targeted solutions.

TABLE OF CONTENTS

CHAPTER 1:

INTRODUCTION

In 2018, mobile phone use dictates a major part of our daily life and mobile applications are essential, providing tools and assistance to day to day tasks. We use applications to connect and converse with friends, family and colleagues; pay taxes or just a coffee; order pizza or even buy a car; take photos of pretty much everything; and a plethora of other things. According to statistics we are increasingly spending more and more time in our mobile devices as they integrate in our daily lives. The average mobile app user spends 2.3 hours daily in the US and these numbers are expected to grow year by year.

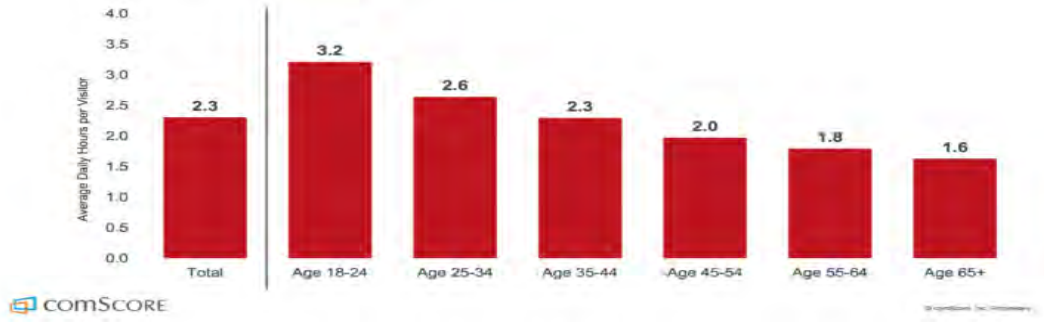

Figure 1: Average daily mobile usage in the US[1]

Even more stunning is the fact that mobile phones gain more and more ground versus the leading platforms a few years back such as Desktops. Younger generations highly prefer using mobile apps instead of desktop apps. This fact hints to the huge technological leaps mobile apps have made the past decade, thus being able to compete regarding their functionality and features with full scale desktop apps.

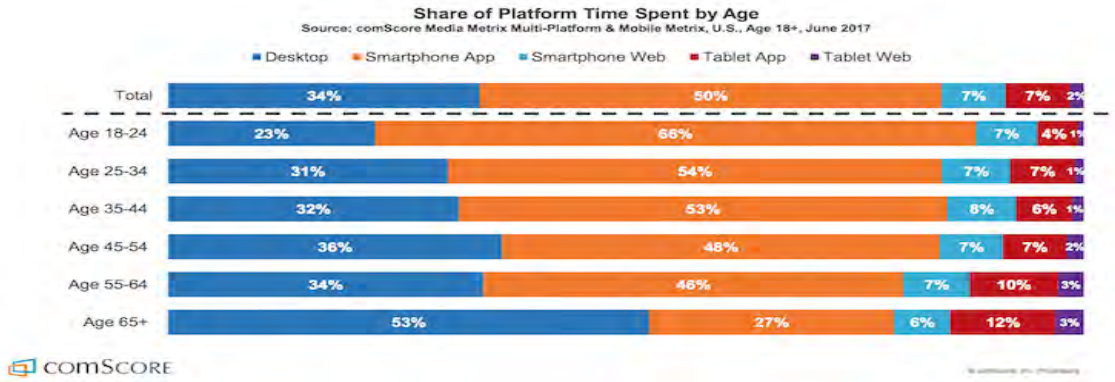

Figure 2: Mobile Usage Share compared to other platforms[2]

---

[1] Image Source: https://www.statista.com/
[2] Image Source: https://www.statista.com/

Currently the Android App Store features 3.3 million apps. In 2015, global mobile app revenues amounted to 69.7 billion U.S. dollars. In 2020, mobile apps are projected to generate 188.9 billion U.S. dollars in revenues via app stores and in-app advertising.

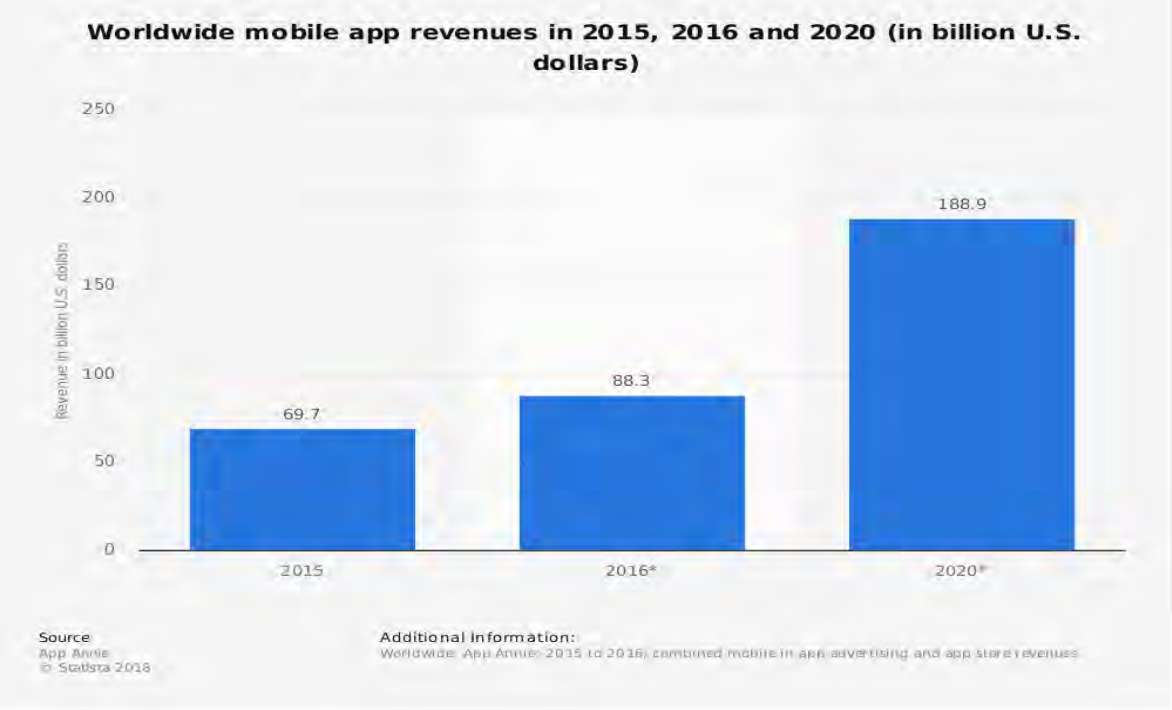

Figure 3: Worldwide app revenues in 2015, 2016 and 2020[3]

1.1 Purpose and objectives

It is evident from both the statistical analysis and the observation of current trends that the mobile app is very promising and will expand in the next years. There are multiple areas of interests when it comes to mobile app development and the future market demands. One of those areas is the personalization of app and the rising need to create apps that are simpler yet more suited for certain needs. Given that need in this thesis we will develop and present a basic structure of a mini network app. The app will feature basic capabilities that can be reproduced and then be built upon to meet the various needs of future clients and individuals. As of right now Facebook obtains the highest market share when it comes to social network apps. However, there are many apps which specialize in different fields and use the same "social network" concept such as

[3] Image Source: https://www.statista.com/

Endomondo Sports Tracker used to measure and share fitness goals or LinkedIn which aim to monetize the social network for professionals. Our app won't use a specific concept or field, but it will explore basic functions needed by the majority of social networking apps such as: a news section; follow function; instant messaging; and unique profiles.

CHAPTER 2: ANDROID

INTRODUCTION

2.1 Android History and Versions

The Android Incorporation was founded in Palo Alto, California in October, 2003 by Andy Rubin, On 17th August 2005 Android was acquired by Google and became a subsidiary of Google Incorporation. On 5th November 2007 Google formed the Open Handset Alliance (OHA), a group of 84 technology and mobile companies aiming to accelerate innovation in mobile, offering consumers a richer, more affordable and improved mobile experience. The first milestone was reached on October 22, 2008 when the HTC Dream was released in the USA market, being the first commercially available smartphone running on Android. HTC Dream was utilizing the initial version of Android (1.0). Interestingly Android versions are named after sweets! Latest version is Android 8.1 with Oreo API level 27.

Android Updates And New Releases So Far
Since 2008 - 2018

	Code Name	Version Number
01	Didn't use code name	1.0
02	Didn't use code name	1.1
03	Cupcake	1.5
04	Donut	1.6
05	Eclair	2.0 – 2.1
06	Froyo	2.2 – 2.2.3
07	Gingerbread	2.3 – 2.3.7
08	Honeycomb	3.0 – 3.2.6
09	Ice Cream Sandwich	4.0 – 4.0.4
10	Jelly Bean	4.1 – 4.3.1
11	KitKat	4.4 – 4.4.4
12	Lollipop	5.0 - 5.1.1
13	Marshmallow	6.0 – 6.0.1
14	Nougat	7.0 – 7.1.2
15	Oreo	8.0 – 8.1

Figure 4: Android Version History[4]

[4] Image Source: http://www.techsmartglobe.com/android-os/

2.2 Android Features

Today Android boasts an 85.9% global market share and it is highly preferred by developers since it provides a number of advantages over other platforms:

1. Android is Open Source and royalty-free eliminating licensing costs.
2. Larger developer and community reach means better support for developers.
3. Increased marketing comes along with multiple sales channels.
4. Android is the best mobile platform between the application and processes architecture thus enabling integration and tweaking of mobile apps according to business needs.
5. All framework provided by the Android community is free reducing the cost of development.
6. The huge market share guarantees a higher success rate.
7. Rich Development Environment enhanced by the multiple sets of libraries.

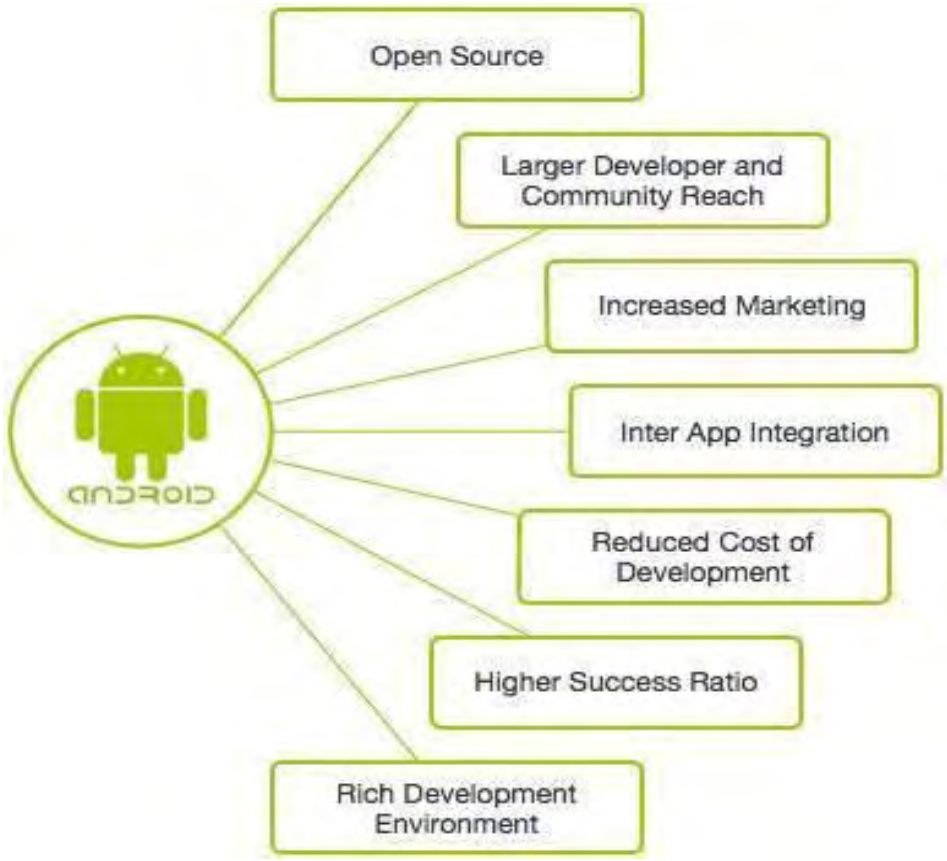

Figure 5: Why Android?[5]

[5] Image Source: https://www.tutorialspoint.com/android/android_overview.htm

2.3 The Android Software

Android is an open source, Linux-based software stack for mobile devices. It includes an operating system, middleware and key applications. The tools and APIs necessary for developing applications on the Android platform are all provided by the Android SKD. It utilizes the Java programming language or more recently the Kotlin programming language. The major components of the Android platform are categorized in the following diagram.

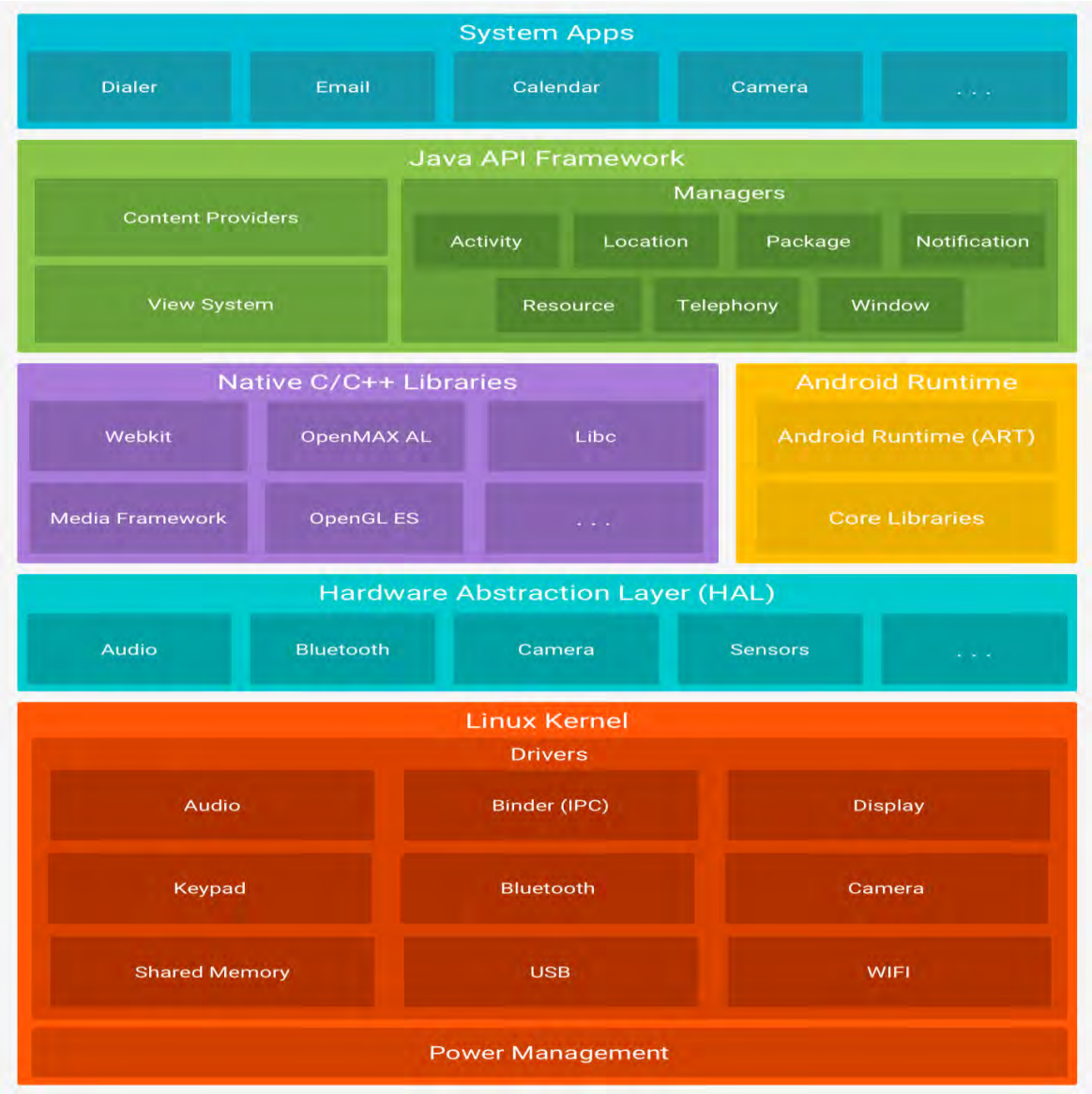

Figure 6: The Android software stack[6]

[6] Image Source: https://developer.android.com/guide/platform/

2.3.1 Operating System

The foundation of the Android platform is the Linux Kernel enabling it to take advantage of its' unique functions such as:

- Low level Memory Management: allocating memory to a new file, freeing memory when specific file is deleted etc.
- Power Management: providing power to devices such as the Bluetooth, phone camera etc.
- Resource Management: it allocates resources to each process enabling extensive multitasking.
- Driver Management: it automatically handles the installation of various drivers.

2.3.2 MiddleWare

The Android software stack contains various middleware divided into three sub-layers:

1. *Native Applications:*

   The use of Java programming language alone does not provide interaction with native applications, such as programs written in C, Assembly etc, which is why the support of native libraries for interacting with low level media components is essential.

2. *Application Framework:*

   The Android Framework is the set of APIs that allow developers to effortlessly write applications for android phones. It provides "shortcuts" to implementations. For example accessing the wifi of a device doesn't require vast amounts of code since a wifiManager class handles all the tasks related to wifi. Putting it simply the Application Framework layer provides higher level services to applications in the form of Java classes which can be used freely by developers.

3. *Android Runtime(ART):*

   ART is an application runtime environment used by the Android operating system. Art replaced the outdated Dalvik virtual machine used till Android version 5.0. It compiles the intermediate language, Dalvik bytercode, into a system dependent binary. The code of the application is pre-compiles during installation removing the lag accompanying the launch of an application on a device. Furthermore, ART

replaces the JIT compilation and since it runs machine code directly (native execution) results in less CPU usage and battery drain.

2.3.3 Applications:

It is the top layer of the Android Architecture. All applications using the Android Framework utilize android runtime and libraries. On the other hand android runtime and native libraries use the Linux Kernel.

2.4 Android Application Components

Application components act as the essential building blocks of an Android application. All the information regarding these components is provided by the application manifest file, the AndroidManifest.xml, that contains a description of each component of an application and their interaction. There are four major application components:

1. *Activities:*

   An Activity represents the user interface of a single screen. An application can have multiple Activities, each one of them operating independently, however Activities can be linked together. Activities are always defined in AndroidManifest.xml.

2. *Services:*

   Background processes which do not require user interaction are described as services (eg. Music playing in the background).

3. *Broadcast Receivers:*

   Broadcast Receivers handle the communication between the Android Operation System and the applications. They do not implement a user interface but they can create a notification to inform the user when an event happens.

4. *Content Providers:*

   They are used for data sharing between applications since android doesn't natively support direct data sharing between applications.

S.No.	Application Components	Description
1	Activities	* Dictates user interface * Handles user interaction with the screen
2	Services	* Handles the running processes in the background of an application
3	Broadcast Receivers	* Handles interaction between the operating system (Android) and various applications
4	Content Providers	* Handles all the issues related to the data and its management (database management)

Table 1: Basic Application Components

2.5 Activities and Fragments

2.5.1 Activity Lifecycle

An Activity represents a single screen with which the user can interact with. A Java class s considered an Activity it extends the Activity class. The system manages all Activities as an activity stack. Upon Activity start it is placed on the top of the stack and becomes the running Activity. Any previous Activity is placed below it in the stack and won't come to the foreground until the new Activity exits.

An Activity has four states:

1. *Foreground State:*

   An Activity is running or is active only if it is on the foreground of the screen, essentially at the top of the stack.

2. *Paused State:*

   When an Activity is visible but has lost focus it is in paused state. A paused Activity is active, meaning it maintains all state and information and remains attached to

the window manager. However it can be killed by the system in case of low memory.

3. *Background State:*

   When an Activity is moved aside by another activity and it is invisible it stops. The Activity still retains all state and information but its window is hidden and will often be killed by the system in low memory situations.

4. *Destroyed State:*

   In cases of low system memory if an Activity is either on Paused State or Background State the system can request it to finish or just kill its process. If the Activity has to be redisplayed to the user then it must be restarted and restored to its previous state.

The movement through an activity's lifecycle uses the following methods:

- *onCreate():*

  It is called when the Activity is initially created. All the normal static set up (views creation, data binding to lists etc.) should be done in this method. It also provides a Bundle containing the Activity's previously frozen state and it is always followed by onStart().

- *onRestart():*

  It is called after the Activity has been stopped, prior to it being started again. Also always followed by onStart().

- *onStart():*

  It is called when he Activity is becoming visible to the user. It is followed by onResume() in case the Activity comes to foreground or by onStop() if it becomes hidden.

- *onResume():*

  It is called when the Activity starts interacting with the user. In this case the Activity is on top of the activity stack and the user input is going to it. It is followed by onPause().

- *onPause():*

  It is called when the system is about to start resuming a previous Activity. The paused Activity does not receive user input and is unable to execute any code. It is

followed by onResume() if the Activity returns back to the foreground or by onStop() if it becomes invisible to the user.

- *onDestroy():*

  It is called in the end before the Activity is destroyed. The Activity can be destroyed if it is finishing or if the system temporarily destroys this instance of the activity to save space.

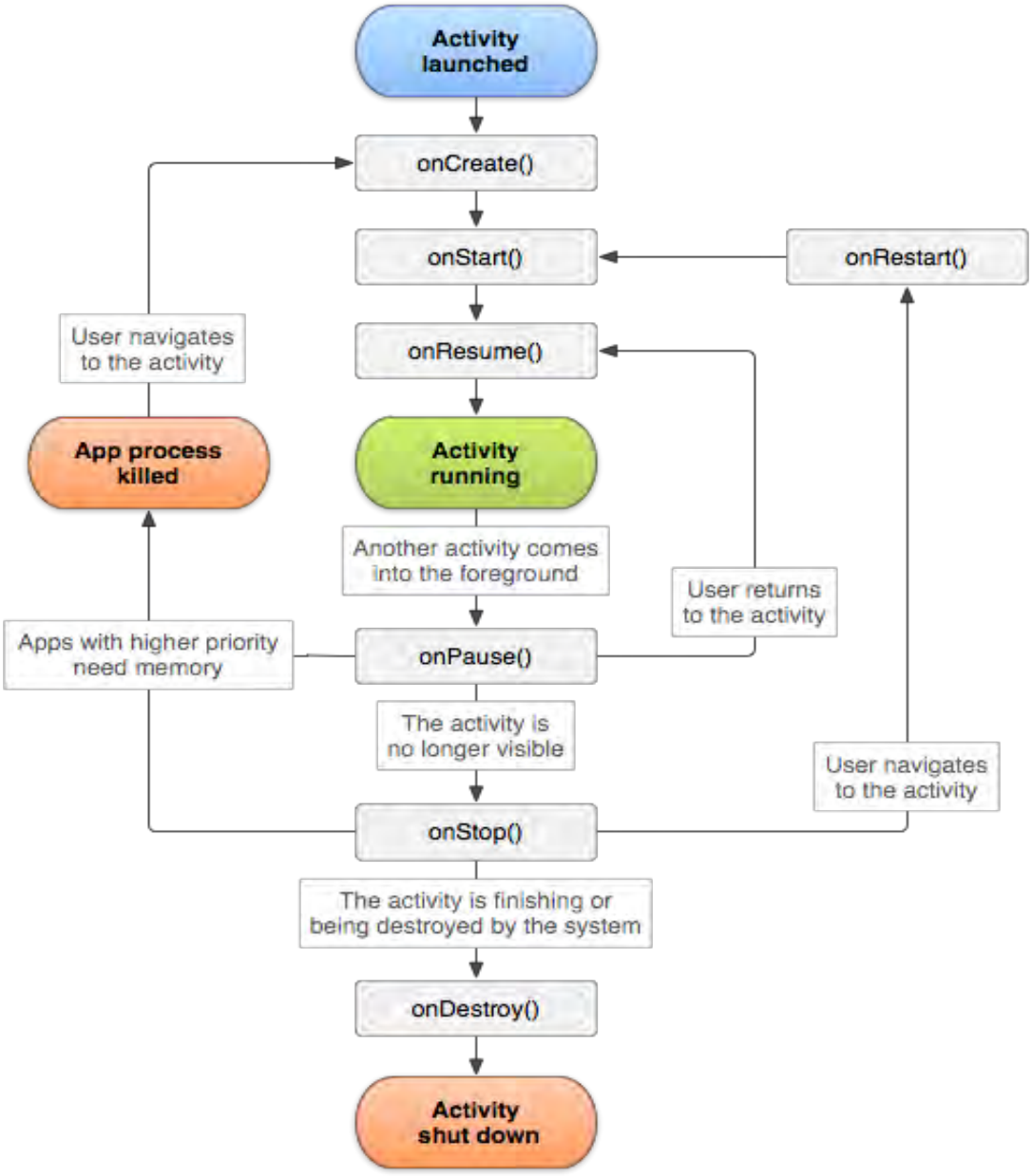

Figure 7: Activity Lifecycle[7]

[7] Image Source: https://developer.android.com/guide/

2.5.2 Fragments

A fragment is a modular section of an activity. It has its own lifecycle and input events, which you can add or remove when the activity is running. Essentially fragments are reusable UI components. The use of fragments presents many advantages. Notably fragments provide:

- Flexible user interface across different screen sizes.
- Fixed, scrolling or swipe tab displays.
- Dialog boxes.
- Actionbar customization with the list and tab modes.

It is recommended to create fragments in the Java code instead of the XML given that this approach enables us to change the UI in runtime. The fragments and the transactions between fragments (add, replace or remove fragments) are handled by the FragmentManager. Every Activity has its own FragmentManager that can be accessed through the getFragmentManager() method.

A Fragment has its own lifecycle and processes its own events. The Fragment lifecycle contains callback methods similar to an activity, such as onCreate(), onStart(), onPause() and onStop().

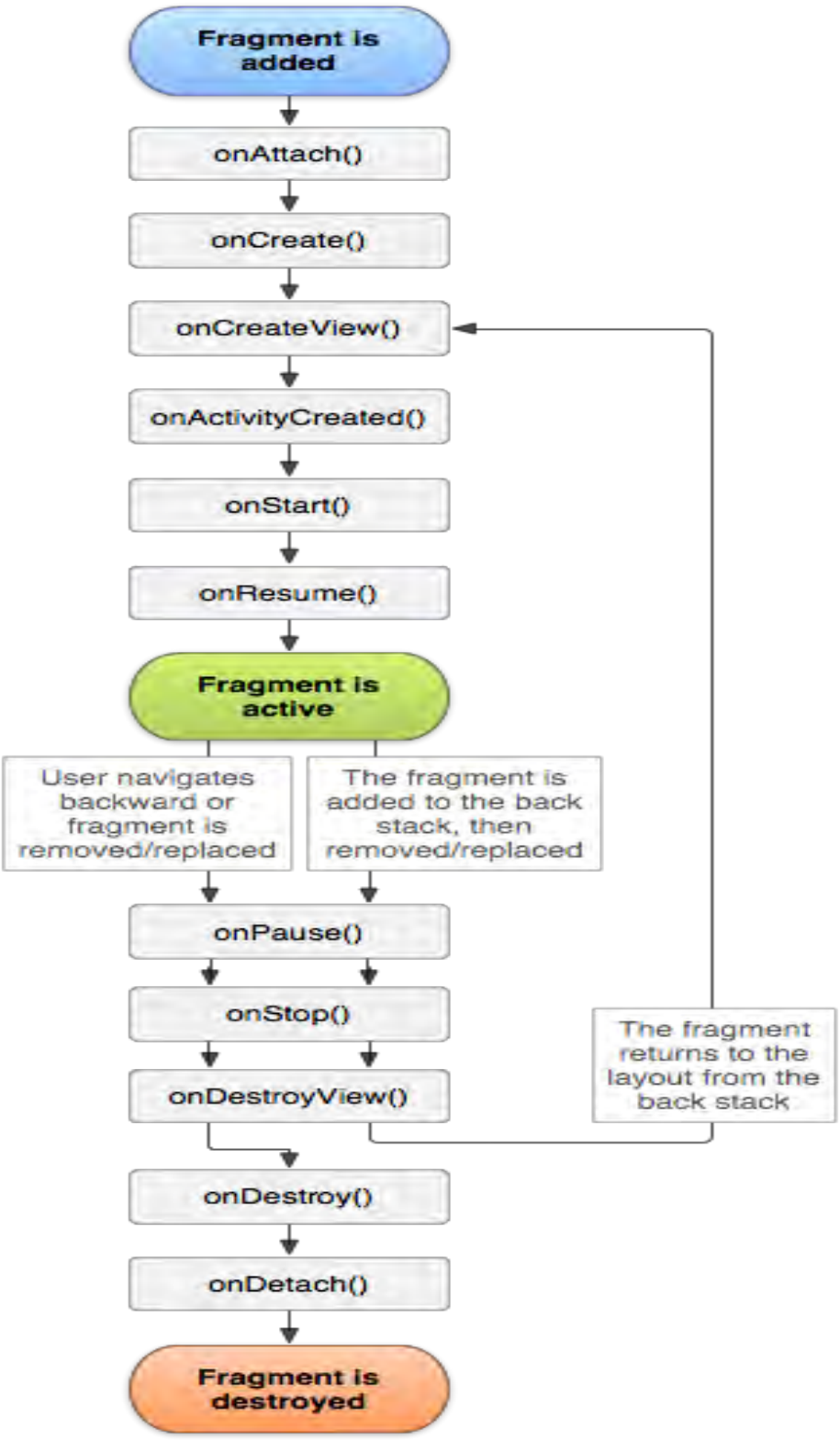

Figure 8: Fragment Lifecycle[8]

[8] Image Source: https://developer.android.com/guide/

2.6 Intents

An Intent is a messaging object used to request an action from another app component. This way an android component can request actions from and by different components. Intents are used mainly to start an Activity, start a Service, deliver a broadcast etc. There are 2 types of Intents:

- *Explicit Intent:*

  If the target component is specified upon the creation of the Intent then it is an explicit Intent.

- *Implicit Intent:*

  Implicit Intent doesn't specify the component thus appropriate information must be included to the system in order to determine which of the available components should be used.

2.7 Services

An Android Service is a long running task or process without any user interaction. It can take two forms:

1. *Started:*

   An application component can start a Service by calling startService(). After it is started, it can run in the background indefinitely even if the component that started it is destroyed.

2. *Bound:*

   A Service is bound when an application component binds to it by calling bindService(). It allows components to interact with the service and it runs for as long as the application component is bound to it. It can also be bound to multiple components at once and it is destroyed only when all the components unbind it.

The Service Lifecycle depends on the way it is created (started or bound). The typical callback methods of a Service are:

- *onStartCommand():*

  This method is called by the system when a component request the start of the service. After the execution of the method the service can run in the background indefinitely and it can be stopped only by calling stopSelf() or stopService().

- *onBind():*

  On the other hand this method is called when a component requests to bind with a Service. Upon the implementation of this method an interface must be provided so the clients can communicate with the Service.

- *onCreate():*

  This method is called only when the service is first created and it performs setup procedures.

- *onDestroy():*

  When the Service is no longer used and there is need to free up resources such as threads, listeners, receivers etc. onDestroy() is called.

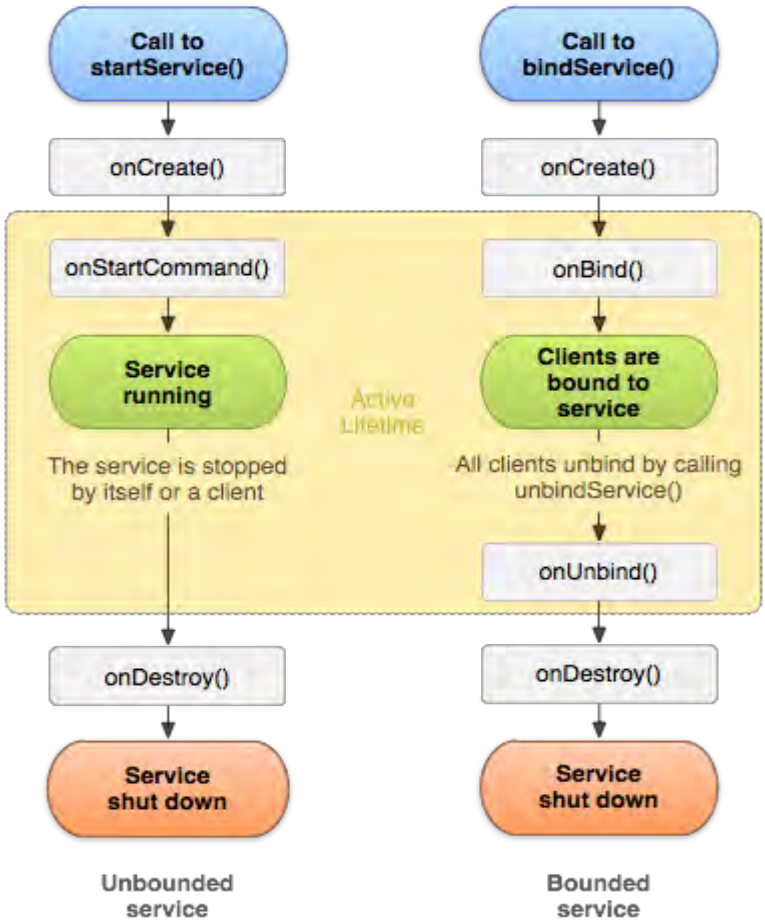

Figure 9: Service Lifecycle[9]

[9] Image Source: https://developer.android.com/guide/

CHAPTER 3: FIREBASE

INTRODUCTION

3.1 Web Services

Web services provide an object-oriented web-based interface to a database server. They are open standard (XML, HTTP, etc.) based applications that can be used by web servers or by a mobile app in order to exchange data and construct a user interface to the end user. Usually web services technologies, such as Amazon Web Services (AWS), allow their subscribers access to virtual clusters of computers that emulate most aspects of a real computer; multiple operating systems; networking; pre-loaded application software such as web servers, databases, CRM etc. Utilizing the web services programmers use server side scripting techniques to provide a customized interface for the user. These scripts may assemble client characteristics used in customizing the response based on those characteristics, the user's requirements, access rights, etc.

3.2 Firebase Platform

Firebase is a web services platform ideal for mobile application development. Essentially it is a mobile and web app development platform that provides developers with innovative and unique tools and services that enable the effortless development of high-quality applications and allow the developer to grow the user base and earn more profit using automated technologies.

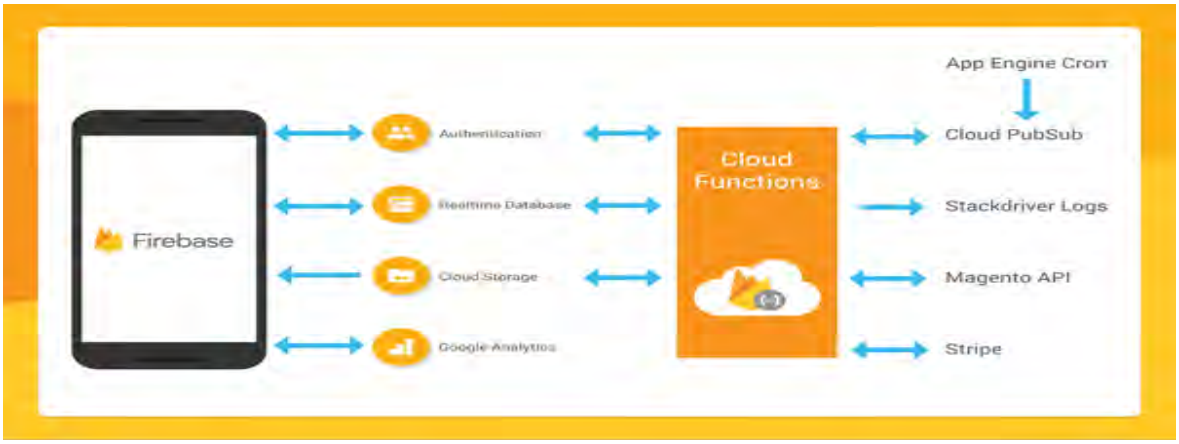

Figure 10: Firebase Basic Features[10]

[10] Image Source: https://hackernoon.com/introduction-to-firebase-218a23186cd7

3.3 Firebase Services

There is a variety of Firebase services which are presented in Figure 8. Our application won't take advantage of each single service. However, the development process uses a Realtime Database, Authentication Services, Cloud Storage and Cloud Messaging.

Figure 11: Firebase Services[11]

3.3.1 Realtime Database

The Firebase Realtime Database enables developers to store and sync data with a NoSQL cloud database. Data is stored as JSON and synchronized in realtime to every connected client. All the app clients share one Realtime Database instance and automatically receive updates with the newest data. The Data remain available even when the app goes offline.

3.3.2 Authentication

Firebase Authentication provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to an app. The authentication allows the app to securely save user data in the cloud and provide identical personalized experience across all of the user's devices. It supports password authentication, phone number authentication and authentication through popular federated identity providers like Google, Facebook, Twitter, GitHub etc.

[11] Image Source: https://www.quora.com/Is-Firebase-a-good-choice-to-build-a-CMS

3.3.3 Cloud Storage

Cloud Storage is a simple yet powerful and cost-effective object storage service. The Firebase SDKs used by Cloud Storage provide top edge Google security to file uploads and downloads for all Firebase apps, regardless of network quality. Cloud Storage can store images, audio, video, or other user-generated content.

3.3.4 Cloud Messaging

In order to notify a client app that new email or other data is available to sync we can use Firebase Cloud Messaging. Through Cloud Messaging notification messages can be send in order to drive user re-engagement and retention. For use cases such as instant messaging, a message can transfer a payload of up to 4KB to a client app.

CHAPTER 4:

ENVIRONMENT SETUP

4.1 Setting up Android Studio project

Before starting the implementation of our project we need to create a project in Android Studio and define the parameters of app. Since every app should aim for the highest possible compatibility with all the majority of the devices on the market we need to select an appropriate SDK. The project uses API 15: Android 4.0.3 (IceCreamSandwich) since it is the only one that can run on the majority of the devices according to Android Studio project manager.

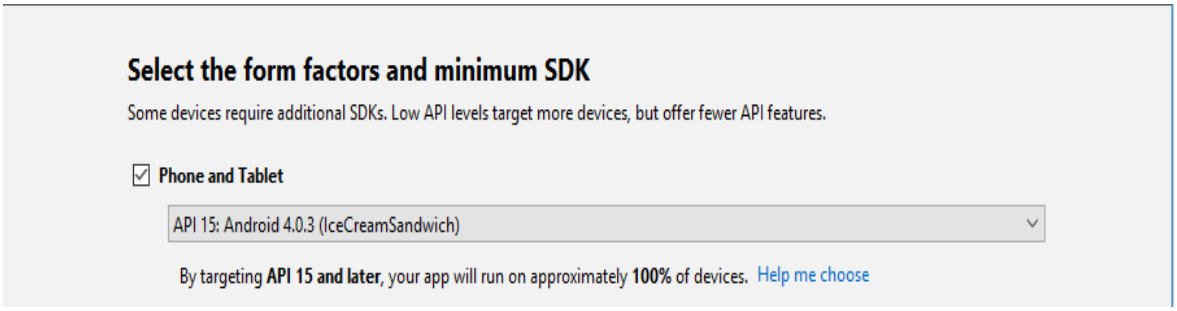

Figure 12: Target API and compatibility

Regarding the project structure we comply by the standard practices. All resources are located in the res folder and divided accordingly. Styles, colors and strings xml files are located in the values folder. Images are distributed quality wise in the drawables folder and the layouts are defined in the layout folder. All our java code is located in the java folder. The dependancies and packages are defined inside the build.grandle (Module:app) file. A rough breakdown can be seen on the instance of the project file manager below. Finally the AndroidManifest.xml file contains the package name, the permissions required for the proper use of the app, the app name and icon and the definition of the Activities of the app.

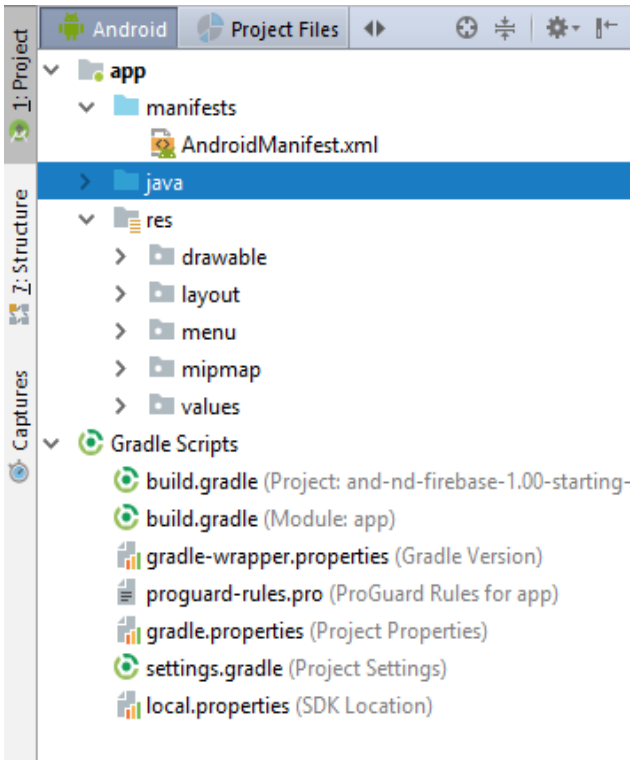

Figure 13: Android Project Structure

4.2 Setting up Firebase Services

To use the Firebase services we need to register an account. Following the account creation we create a new Firebase Project that we will link to our app. The firebase project we contain the realtime database, the Authentication services and the storage used for images and files. Upon project creation we can customize each of the above services using the firebase control panel as shown in Figure 14. The firebase console also provides free analytics for our app. For our purposes we will use the free version of firebase since we do not expect huge traffic. For commercial apps firebase offers different upgrade plans.

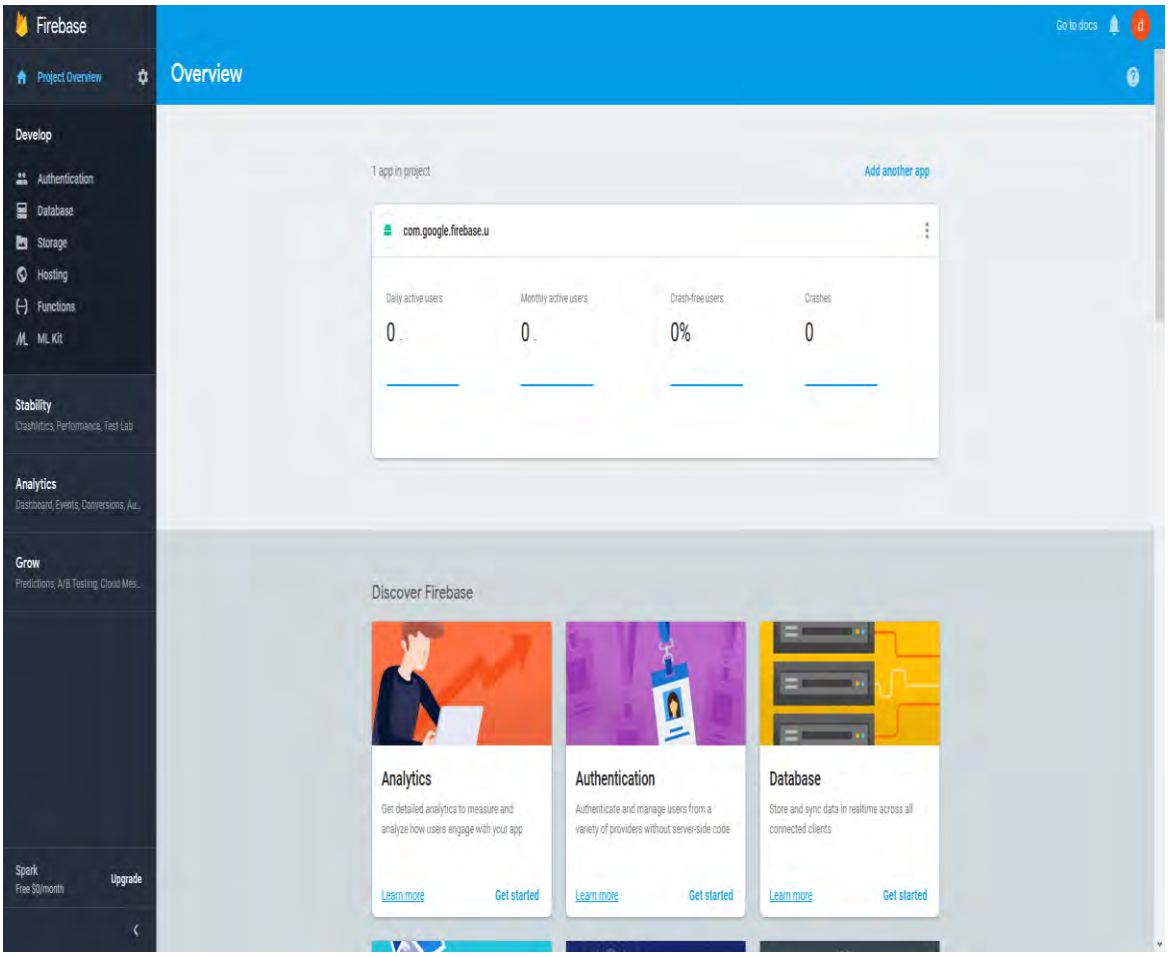

Figure 14: Firebase Project Console

4.2.1 Creating and configuring realtime Database

The first service our app uses is the realtime Database. When configuring the realtime Database it is extremely important to properly define the rules. The rules are those that will provide the necessary protection to our server and distribute access to the users. For example when it comes to messaging our rule set (Figure 15) dictate that users need to be authenticated to read and write data and also define the structure of a message (if it contains text or images only).

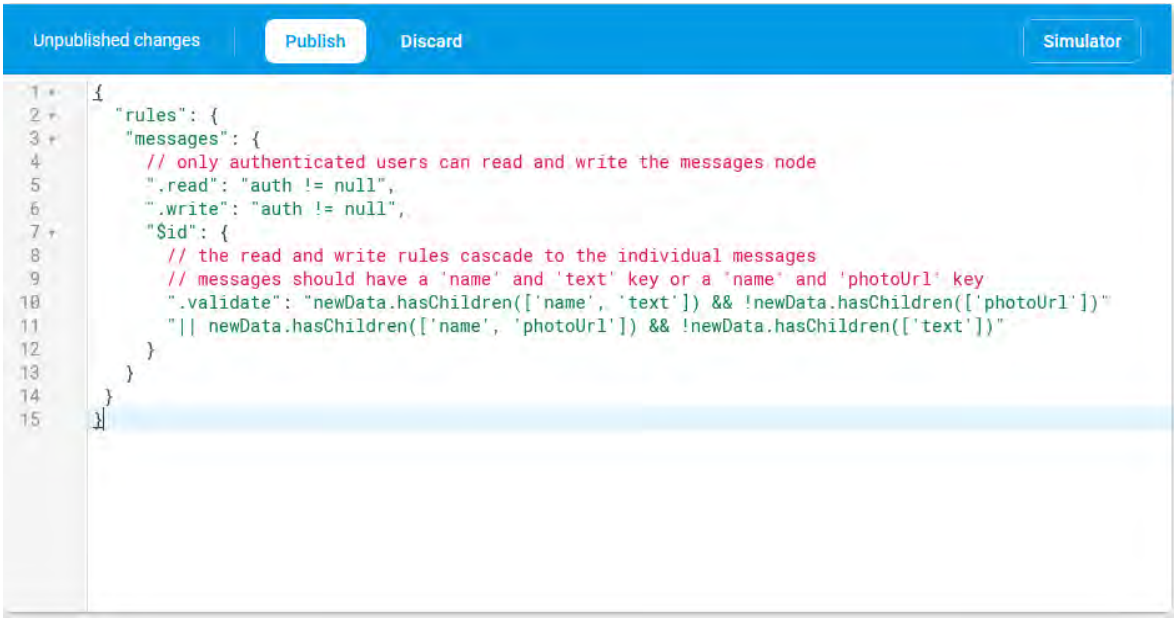

Figure 15: Realtime Database Rule set

In general Firebase allows three main rule types: .read, .write. and .validate. Each of these can be set to “true” or “false” and can apply to the whole database or a particular location in the database depending on how they are configured.

4.2.2 Authentication

One of the main advantages of Firebase is the build in Authentication system. Adding an authentication method to our project services is quite simple. All we need to do is just enable it from the console and then we can add it to our java code in the android app. The Authentication panel also provides a complete list of our userbase along with their info. Our project uses Google and email Authentication (Figure 16).

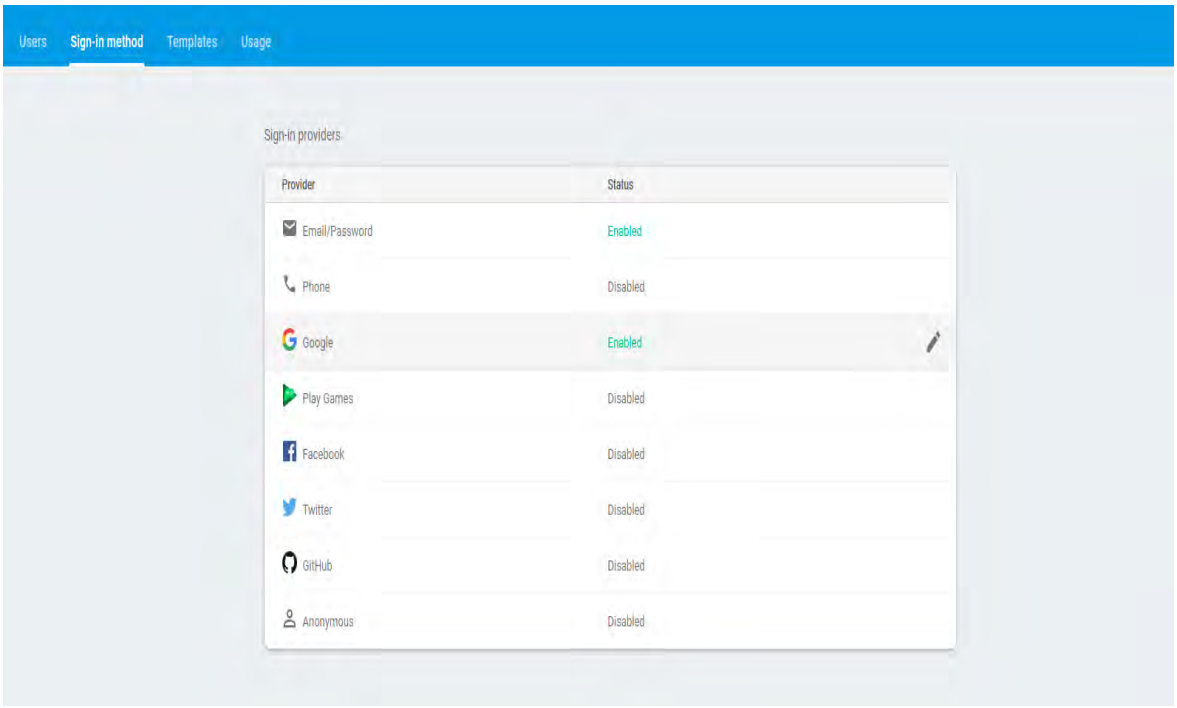

Figure 16: Authentication Methods

4.2.3 Storage Service

Finally our Storage service where images and media files are stored has the same properties as the realtime database. We also provide a similar set of rules to restrict unwanted access. The Storage also enables the administrator to arrange the files into folders and categorize them.

CHAPTER 5:

FEATURES AND CODE

ANALYSIS

5.1 LogIn – SignUp Screen

When the user opens the app the first screen he is prompted to is the Login – Signup screen. In order to use the Firebase Auth services we need to add the component in our app. This is done by updating the dependencies in the build.grandle file. Since we are going to use multiple services our build.grandle file should contain all of them (code snippet 1).

```
dependencies {
    implementation fileTree(include: ['*.jar'], dir: 'libs')
    testImplementation 'junit:junit:4.12'
    implementation 'com.android.support:design:24.2.0'
    implementation 'com.android.support:appcompat-v7:24.2.1'

    // Displaying images
    implementation 'com.github.bumptech.glide:glide:3.6.1'
    //firebase realtime database
    implementation 'com.google.firebase:firebase-database:15.0.0'
    implementation 'com.google.firebase:firebase-auth:15.0.0'
    // FirebaseUI for Firebase Auth
    implementation 'com.firebaseui:firebase-ui-auth:3.3.1'
    //Firebase Storage (Images)
    implementation 'com.google.firebase:firebase-storage:16.0.1'
}
```

Code Snippet 1: Firebase Dependencies

Firebase has a predefined screen to handle Login – Signup. However, upon community request the screen design can be changed through explicit styles defined in styles.xml. The default Login Screen is presented in Figure 17.

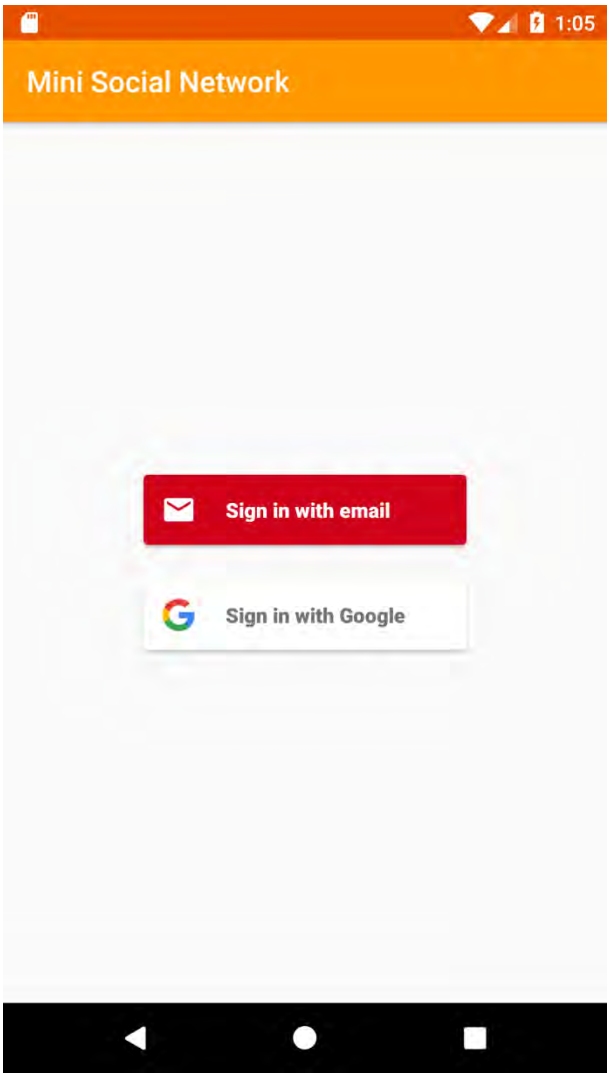

Figure 17: Firebase Default Login Screen

To change the design and make the app more aesthetically refined we defined different colors and styles. We also need to change the AuthenticationListener (code snippet 2) in our java code. This way when the Authentication page is being initialized, by entering the line .setTheme(R.style.LoginTheme), Android Studio loads the style.xml we created instead of the Firebase default one.

```
mAuthStateListener = new FirebaseAuth.AuthStateListener() {
        @Override
        public void onAuthStateChanged(@NonNull FirebaseAuth firebaseAuth) {
            FirebaseUser user = firebaseAuth.getCurrentUser();
            if (user != null) {
                // User is signed in
                onSignedInInitialize(user.getDisplayName());
            } else {
                // User is signed out
                onSignedOutCleanup();
                startActivityForResult(
                        AuthUI.getInstance()
                                .createSignInIntentBuilder()
                                .setIsSmartLockEnabled(false)
                                .setAvailableProviders(Arrays.asList(
                                        new AuthUI.IdpConfig.EmailBuilder().build(),
                                        new AuthUI.IdpConfig.GoogleBuilder().build()))
                                .setTheme(R.style.LoginTheme)
                                .build(),
                        RC_SIGN_IN);
            }
        }
    };
}
```

Code Snippet 2: Custom Login Theme

The app design changes according to Figure 18.

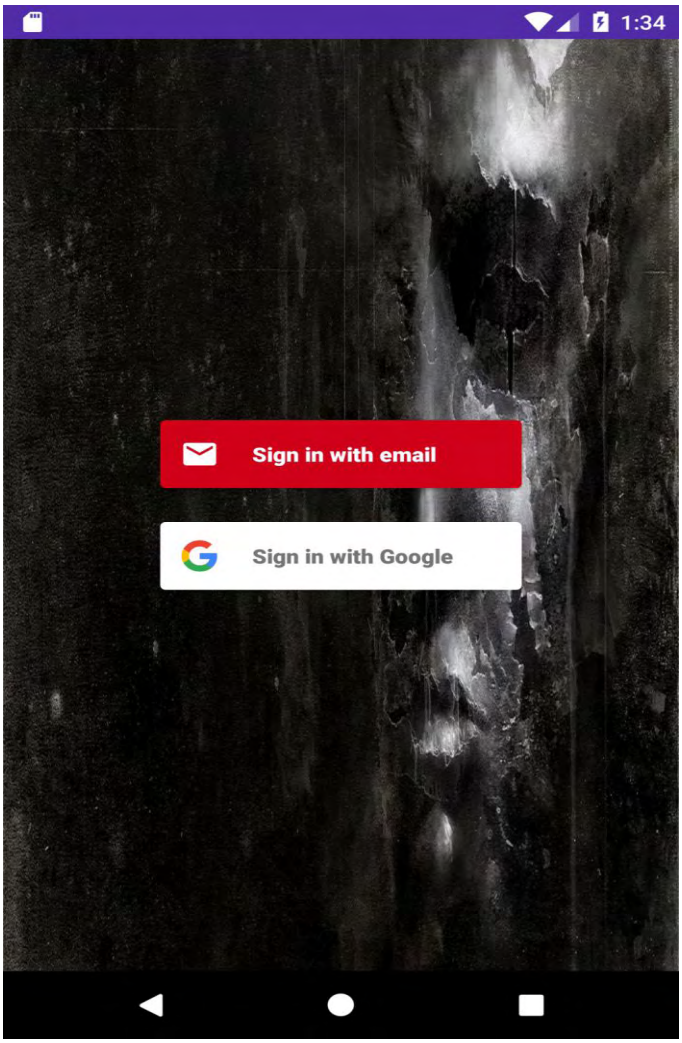

Figure 18: Updated Login Design

5.2 News Section

Upon Login the user is redirected to the News screen. The app contains a screen where everyday news can be posted. Each news entry can contain an image, date, author, section, brief description and title. All the data are fetched from the database. The news are organized in a list manner. New news at the moment can only be added by the administration using the Firebase Console. For testing purposes we used articles provided by Guardian News and implemented them in our app. The list view is depicted in figure 19.

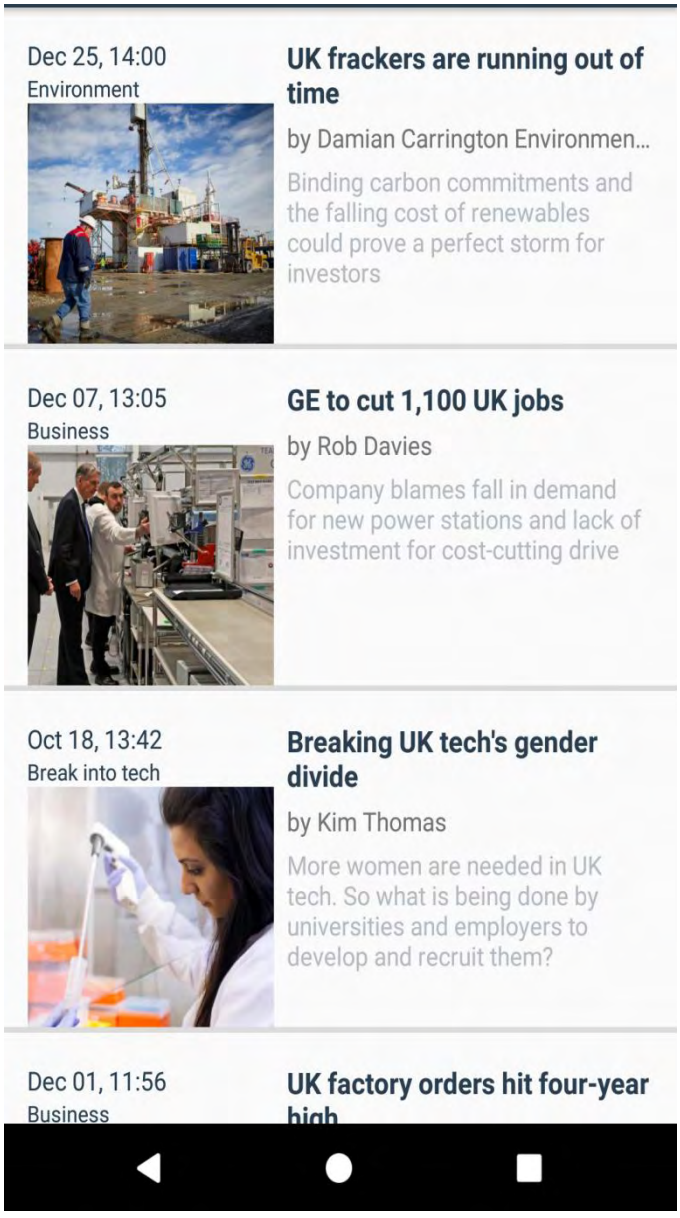

Figure 19: News Section - Posts list

Each news post is created using a Post Loader and a Post Adapter. The Post adapter method performs actions related to requesting and receiving data from our database. It is also tasked with handling possible exceptions such as problems contacting the database or identifying missing components of the Post. Code snippet 3 demonstrates such an exception. In case the photo from a post is missing then the Imageview visibility containing the image is set to GONE, practically removing the "empty" image cell and shrinking the post to contain only text.

```
boolean isPhoto = message.getPhotoUrl() != null;
if (isPhoto) {
    messageTextView.setVisibility(View.GONE);
    photoImageView.setVisibility(View.VISIBLE);
    Glide.with(photoImageView.getContext())
            .load(message.getPhotoUrl())
            .into(photoImageView);
} else {
    messageTextView.setVisibility(View.VISIBLE);
    photoImageView.setVisibility(View.GONE);
    messageTextView.setText(message.getText());
}
authorTextView.setText(message.getName());
```

Code Snippet 3: Handling Missing Image

We should also note that in case our images are outside of scope (Huge dimensions or too small) to avoid presenting a distorted content to the user we can use the Picasso library. The Picasso library enables us to resize the image or the Imageview to avoid such graphic complications. Given that our posts are handled by an administrator and not the average user we did not implement such a function, however it should be noted for future implementations that allow user posts. An example of the Picasso library is shown in code snippet 4.

```
if (!currentStory.getImageLink().matches("")) {
    Picasso.with(getContext())
            .load(currentStory.getImageLink())
            .resize((int)
getContext().getResources().getDimension(R.dimen.width_of_article_image), (int)
getContext().getResources().getDimension(R.dimen.height_of_article_image))
            .placeholder(R.drawable.image_placeholder)
            .error(R.drawable.no_image_to_download)
            .centerCrop()
            .into(imageView);
} else {
    Picasso.with(getContext())
            .load(R.drawable.no_image_to_download)
            .resize((int)
getContext().getResources().getDimension(R.dimen.width_of_article_image), (int)
getContext().getResources().getDimension(R.dimen.height_of_article_image))
            .centerCrop()
            .into(imageView);
}
```

Code Snippet 4: Picasso Library Example

5.3 Instant Messaging Function and Image Sharing

Possibly the most important feature a social network should implement is messaging. Messaging allows users to connect with each other real time and converse without costs. Our app implements a group chat indented only for the registered users. All messages are delivered real time and stored to the RealTime Database. Furthemore the use of the Realtime Database means that there is no need to refresh the page/ screen in order to read the messages. Users will receive notifications upon new messages and will be able to read them at once. Upon exit from the app the messages aren't stored in the local memory. The message history is automatically synced when the user logs in again from any device. Usually messages are kept simple containing only text. However upon further implementation we can also add images, photos and emoji sharing. Our app supports text and image sharing from the phones local storage. The personalized messaging experience is derived from the rules implemented in the realtime Database (Figure 15).

We should note nonetheless that image sharing does not implement the realtime Database rules. Sharing images requires the Storage service, which is why a new set of rules must be written for images. The key points remain the same (code snippet 5), meaning we still require user authentication for read and write. Images however present more challenges. The app handles rare cases where the image upload exceeds a reasonable size. The rules in code snippet 5 (3rd match block) are written in the storage console and prevent users from sending images that exceed 5MB size.

```
service firebase.storage {
  match /b/{bucket}/o {
    match /{allPaths=**} {
      allow read, write: if request.auth != null;
    }

    match /images/{imageId} {
      // Only allow uploads of any image file that's less than 5MB
      allow write: if request.resource.size < 5 * 1024 * 1024
                   && request.resource.contentType.matches('image/.*');
    }
  }
}
```

Code Snippet 5: Image Sharing Rule Set

An instance of Messaging can be seen in Figure 20.

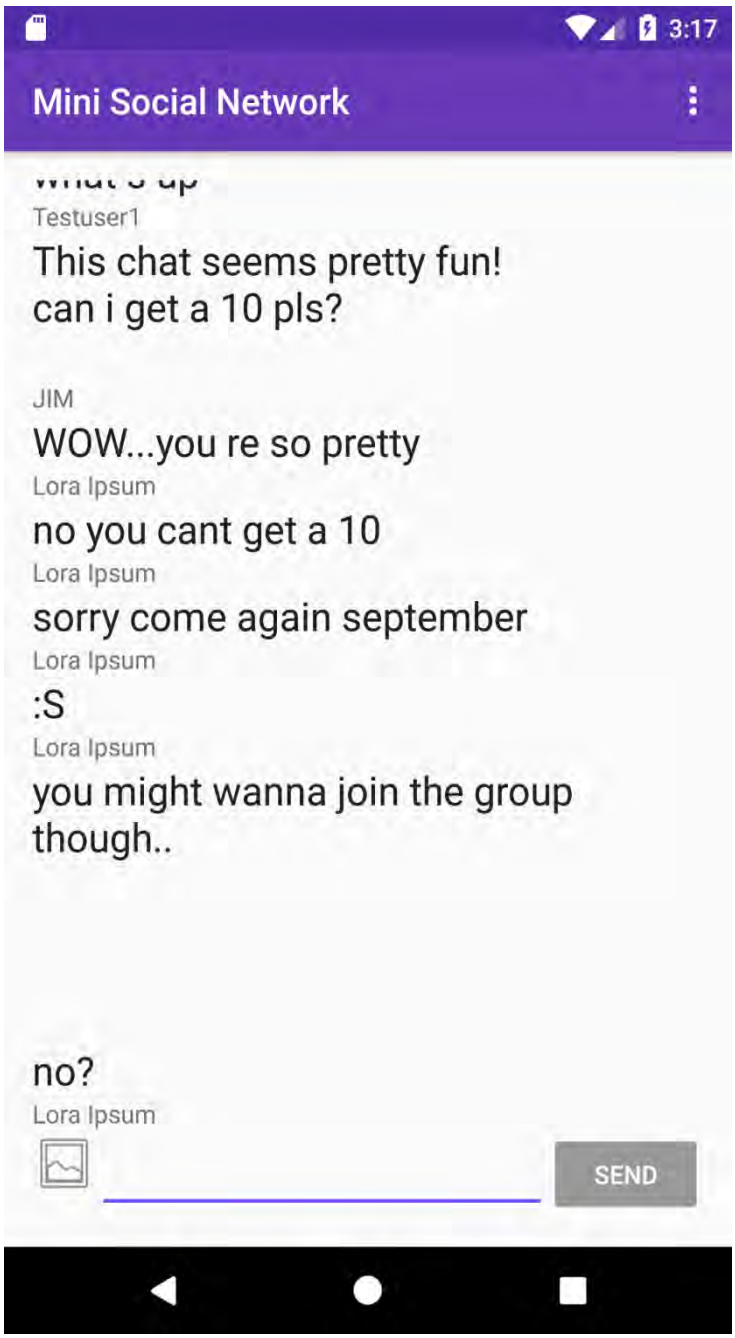

Figure 20: Chat instance

CHAPTER 6:

CONCLUSION

6.1 Results

Upon starting the implementation of this project we set certain goals. The aim was to indentify and implement the basic functions of a Social network. The development process involved the use of the latest technology stacks and tools. The Firebase Services were implemented providing databases and servers built for messaging and social apps. We can safely conclude that the base template for building a social network was achieved. All the capabilities of the app are deemed necessary for any social app. The development process was also done in a way that allows further development and additions.

6.2 Future Research Possibilities

The development of the Mini social Network app was executed in mind to future development. The goal was not to create and present a complete social network equivalent to facebook or other major networks. The aim was to create a template for future applications to build upon. As of such, future research should aim to implement and develop further those basic applications. Improvement of the messaging feature such as sending and receiving emojis or files is a field of interest. Providing administration accounts and capabilities through the app and not the firebase console could also prove and interesting topic. In general, there are many possibilities for improvement and extension of the template app.

REFERENCES

[1] Antoni Zolciak. [Online]. Available: https://insanelab.com/blog/mobile-development/mobile-app-development-trends-2018/

[2] Artyom Dogtiev. [Online]. Available: http://www.businessofapps.com/data/app-statistics/

[3] Brief History of Mobile Apps. [Online]. Available: https://expertise.jetruby.com/brief-history-of-mobile-apps-286fbbf766a9

[4] Android - Statistics & Facts. [Online]. Available: https://www.statista.com/topics/876/android/

[5] Developers. [Online]. Available: https://developer.android.com/guide/platform/

[6] tutorialspoint. [Online]. Available: https://www.tutorialspoint.com/android/android_application_components.htm

[7] W3schools. [Online]. Available: http://www.w3school.in/w3schools/android-tutorial/android-application-components

[8] Developers. [Online]. Available: https://developer.android.com/reference/android/app/Activity

[9] Developers. [Online]. Available: https://developer.android.com/reference/android/content/Intent

[10] Developers. [Online]. Available: https://developer.android.com/guide/components/services

[11] tutorialspoint. [Online]. Available: https://www.tutorialspoint.com/webservices/index.htm

[12] Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Amazon_Web_Services

[13] Hackermoon. [Online]. Available: https://hackernoon.com/introduction-to-firebase-218a23186cd7

[14] Firebase. [Online]. Available: https://firebase.google.com/docs

APPENDICES

APPENDIX A:

GLOSSARY OF ACRONYMS

API - Application Programming Interface

SDK - Software Development Kit

APK - Android Application Package

APP - Application

GUI - Graphical User Interface

JSON - JavaScript Object Notation

XML - eXtensible Markup Language

ART – Android Runtime

PNG – Portable Network Graphics

APPENDIX B:

TABLE OF FIGURES

APPENDIX C:

TABLE OF CODE SNIPPETS