A FIREBASE FILESYSTEM

A Project

Presented to the faculty of the Department of Computer Science

California State University, Sacramento

Submitted in partial satisfaction of
the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

by

Neetha Janardhana

SPRING
2017

A FIREBASE FILESYSTEM

A Project

by

Neetha Janardhana

Approved by:

__________________________________, Committee Chair
Dr. Jinsong Ouyang

__________________________________, Second Reader
Dr. Scott Gordon

____
Date

Student: Neetha Janardhana

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit is to be awarded for the project.

____________________________, Graduate Coordinator   _____________________
Dr. Jinsong Ouyang   Date

Department of Computer Science

Abstract

of

A FIREBASE FILESYSTEM

by

Neetha Janardhana

Nowadays, in this data driven world, there is always a necessity for storing and sharing the data securely. As per the big data statistics, it is said that about 2.5 quintillion data is created every day in the form of text files, images, PDFs and many more. Some of these files are important that they are required to be stored safely and sometimes, not only just storing but instead, they are required to be shared in a secure way. Thus, an android application is developed to render storing of the files over the cloud in a traditional manner just like in Windows and Linux. The application enables the users to sign in using their Google account to create and share the root level directories via cloud by granting access permissions to the group members and other signed in users. Firebase cloud service is used in this application for storing the data which provides – secure Authentication using Google account, Firebase Storage which can store petabytes of data.

_______________________, Committee Chair
Dr. Jinsong Ouyang

____
Date

ACKNOWLEDGEMENTS

I would like express my sincere thanks to my advisor, Dr. Jinsong Ouyang, for his time, support and sharing his expertise throughout this project. His suggestions have made this project more challenging and interesting.

In addition, I am also thankful to Dr. Scott Gordon, second reader for this project for taking the time to review my work.

My sincere gratitude also goes to the entire Department of Computer Science at California State University for providing me with the opportunity to pursue MS program.

Last but not the least; I thank my dearest husband and my parents for supporting me spiritually throughout this tenure.

I take this opportunity to thank everyone who directly or indirectly supported me to finish my work.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Firebase File System is an Android Application that allows the android users to store the files in the cloud tool called Firebase. It also lets users to share the files with each other. The app permits users to share the files in a secure way. In order to securely share the files with the other users Google Authentication is integrated to the app for eliminating unauthorized access. Some of the other features that are implemented in this app are, creating groups, granting access permissions for the directories and files by the group owner or by the Owner of the file, uploading and downloading of the files by the users who are granted with write access (W), viewing and downloading of the files by the users who have read with download access (D) and inspecting the files/directories, if the users have read only access (R).

1.1 File Storage System

File System is a systematic way of storing the files into folders and organized by naming conventions. They are presented in the hierarchy of directories, subdirectories and files. Files are stored in the file system with the limited amount of metadata, such as file name, creation date, file type and so on. Figure 1 shown below represents the traditional way of storing the files.

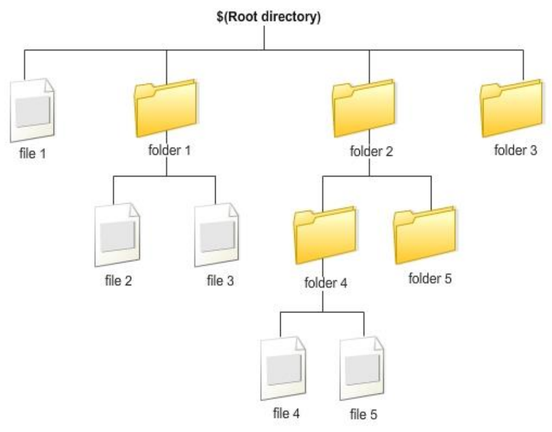


Figure 1: File System [1]

1.2 Firebase

Firebase is a cloud based mobile backend as a service that lets the developer build a mobile app on IOS, Android and the web without backend management. One of the advantages of using firebase is that, the transmission of data is fast and it provides an API that allows developers to store and sync data across multiple clients. In chapter 4, we will discuss more features of Firebase in detail.

CHAPTER 2

PROJECT WORKFLOW

The application enables the users to login to the app via Google accounts. Users are given an option to create groups under the user's own dashboard after logging in. Each group can have a folder structure. Group owner who creates the folder structure has all the rights to read, download and write for the file structure.

2.1 Project Workflow

The figure 3 is the project workflow. When a user is logging in for the first time into the app, Google accounts which are synched to the app user's device will be displayed on the login screen. Apart from the account displayed, if user wants to login from different account, then the user can click on "Add Account" button. By selecting one of the Google accounts, user is able to login to the app.

2.2 User Responsibilities

A user is given an option of creating groups in his dashboard. By creating the group, user becomes the owner of the group. A folder structure can be added to every group immediately after the group creation or at the later point of time by the owner. As discussed previously in this chapter, the owner of the group has all the rights in controlling the access of the file structure that is, he has the rights to grant or revoke the access permissions of the group users for the file structure. And based on the access permissions, group users can view the contents the file, download, upload and delete the files. In order to provide users with the access permissions, owner needs to add users to his group. By default group users are provided with read access permission by the group owner.

An user can also be a part of another group i,e folder structure that the user is a part of. These file structure is displayed along with the group owner’s name. Based on the access permission set, user can manipulate the file system. Figure 2 shows roles of user in the app.

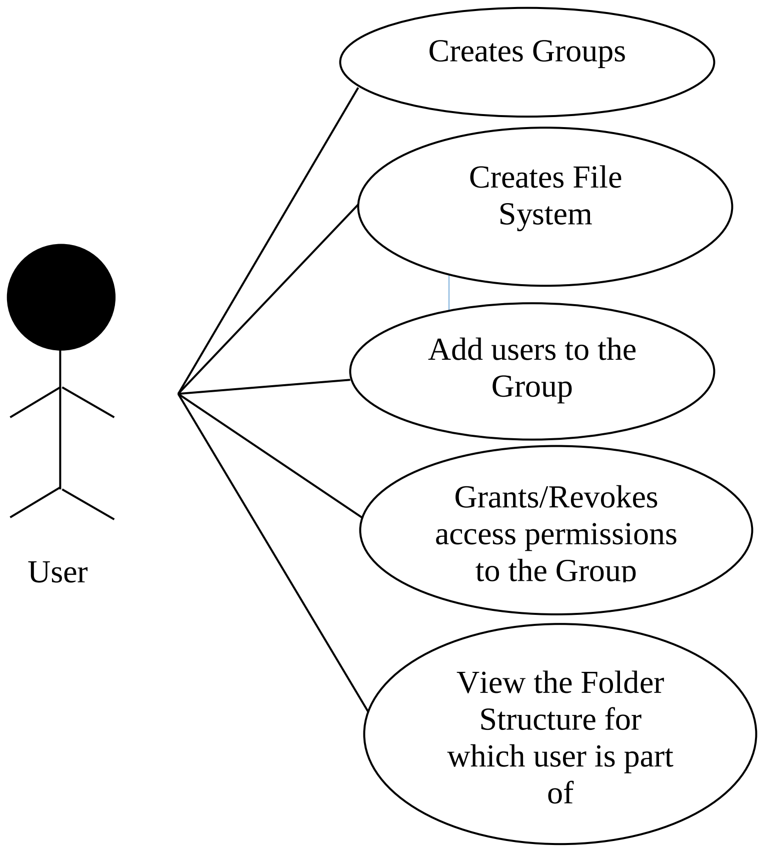

Creates Groups
Creates File System
Add users to the Group
Grants/Revokes access permissions to the Group
View the Folder Structure for which user is part of
User

Figure 2: User Use Case Diagram

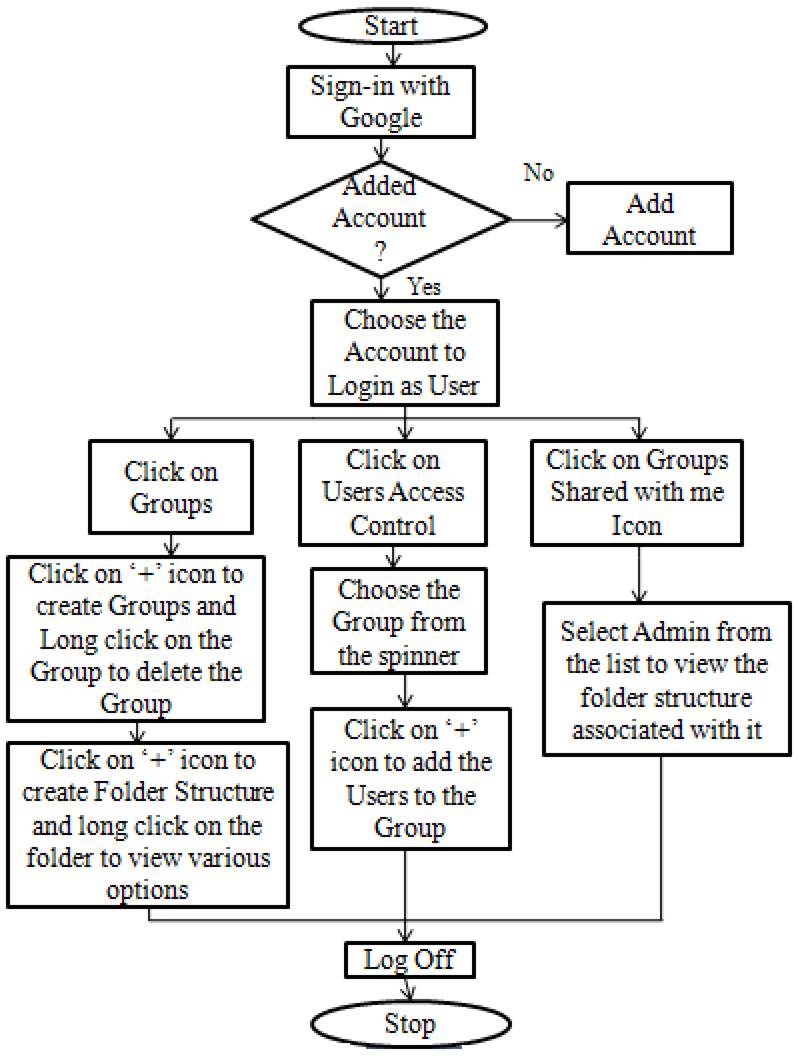

Start
Sign-in with Google
Added Account ?
No
Add Account
Yes
Choose the Account to Login as User
Click on Groups
Click on Users Access Control
Click on Groups Shared with me Icon
Click on '+' icon to create Groups and Long click on the Group to delete the Group
Choose the Group from the spinner
Select Admin from the list to view the folder structure associated with it
Click on '+' icon to create Folder Structure and long click on the folder to view various options
Click on '+' icon to add the Users to the Group
Log Off
Stop

Figure 3: Project Flow Diagram

CHAPTER 3

ANDROID FUNDAMENTALS

3.1 Android Application Fundamentals

Android apps are coded in Java Programming language. The code is compiled by Android SDK tools combined with any data and resource files into an Android package called as APK with .apk as suffix. This .apk file consists of all the necessary data required to install an app and is used during the installation process. [2]

Following are the security features that each Android app lives with:

- Each android app has its own user hence; Android operating system is called as multi user Linux system.
- By default, a Linux user ID is assigned to each app and access permissions for the files are set based on the user ID.
- Every app has its own Virtual Machine so that it can run individually.

3.2 Application Components

App components are the basic building blocks of Android apps. A system or a user can enter into an app with the entry point provided for each component. [2]

There are four different kinds of app components:

- Activities.
- Services.
- Content providers.
- Broadcast receiver.

3.2.1 Activities

Activities are used to interact with the users. Each screen in the User Interface (UI) represents an activity. An activity helps in keeping track of the data that users care about and ensures that the process is executed by the system. It also stores the previously used data by the previous process when the app goes to stopped activity and helps in restoring the previous activity. Each activity has a layout XML and java file.

When an app is launched, the main activity gets started. The series of lifecycle of the activity is shown in figure 4.

- onCreate() – this is called when the activity is created.
- onStart() – this is called when the activity becomes visible to the users.
- onResume() – this is called when the user resumes to the activity which was paused.
- onPause() – this is called when the current running activity is paused. The paused activity do not execute any code
- onStop() – this is called when the activity is not visible.
- OnDestroy() – this is called before the activity is destroyed.

- OnResart() – this is called when the activity restarts after stopping.

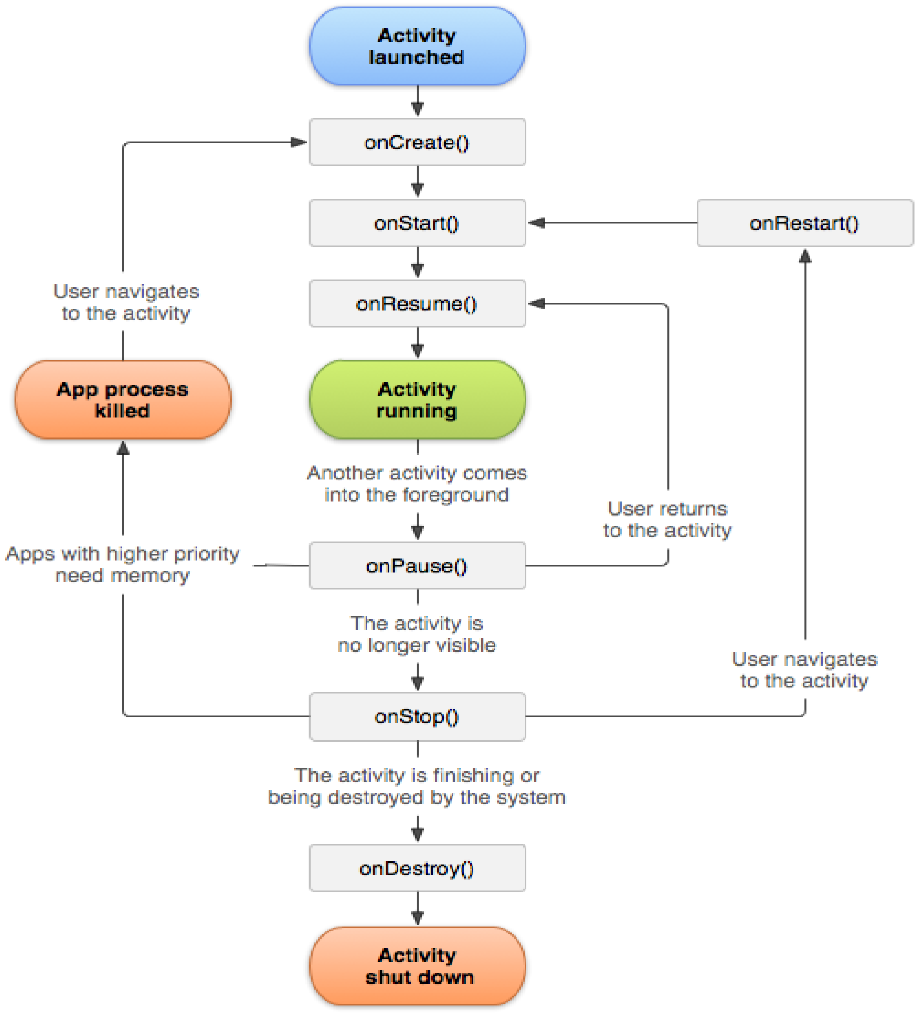

Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Figure 4: Activity Life Cycle [3]

3.2.2 Services

Services are the components that make long running process to run in the background. They do not provide any UI and if the user switches to another application the current service can still run in the background [2].

A service is started by calling startActivity() and is bound when the activity binds by calling bindService().

3.2.3 Broadcast Receivers

Broadcast Receivers are the components that simply respond to the broadcast messages that the app receives from other applications. For example, some applications initiates broadcast messages to let another application to know some data is getting downloaded and after the data download is complete, it will let user know that the data is ready to use [2].

3.2.4 Content Providers

Content Providers component manages access to structured set of data. It is an interface that links the data and running code of two individual processes. Content providers provide security for the data by encapsulation [2].

3.3 Fragments

Fragments are the subcomponent of the Activity. They are placed in an activity and interacted via fragment manager. The lifecycle of the fragment depends on the lifecycle of the Activity as shown in the figure 5.

- OnAttach() – this is called the fragment is associated with the activity.
- onCreateView() – this is called when the fragment draws its UI for the first time.

- onActivityCreated() – this is called when the host activity is created and after onCreateView() is called.
- onDestroyView() – this is called when the hierarchy associated with the fragments is destroyed.
- onDetach() - this is called when the fragment disassociated from the activity.

3.4 Setting up Android Environment

Android Application can be developed in many operating systems such as Microsoft Windows XP or later version, Mac OS X 10.5.8 or later Intel chip and Linux including GNU C library 2.7 or later. Android development tools are freely available in web. Before starting Android application programming, install either of the following software. In this project, Android Studios is used to develop the application [5].

- Java JDK5 or later
- Android Studios

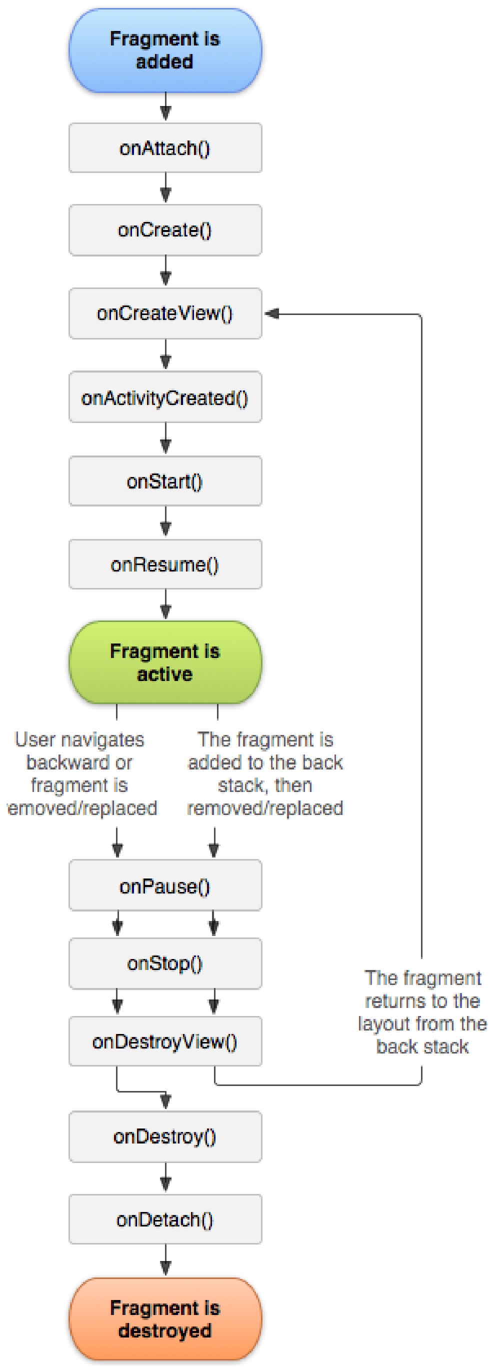


Figure 5: Fragment Life Cycle [4]

3.5 Creating Android Project

After installing Android Studios, a new Android project could be created by clicking on "Start a new Android Studio project" in the Android Studio Setup Wizard window, as shown in figure 6. This will pop up "Create New Project" window shown in figure 7 and 8. Hit on next button after entering Application name and Minimum SDK. Next, choose the activity for the app as shown in figure 9 and in the final stage the development tool to write the application code is ready and is as in the figure 10.

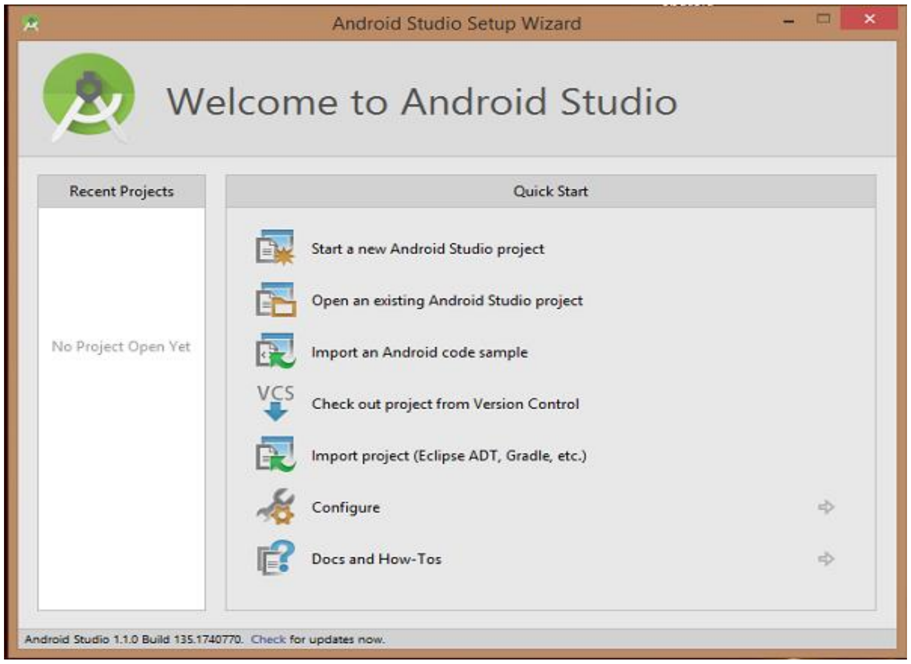
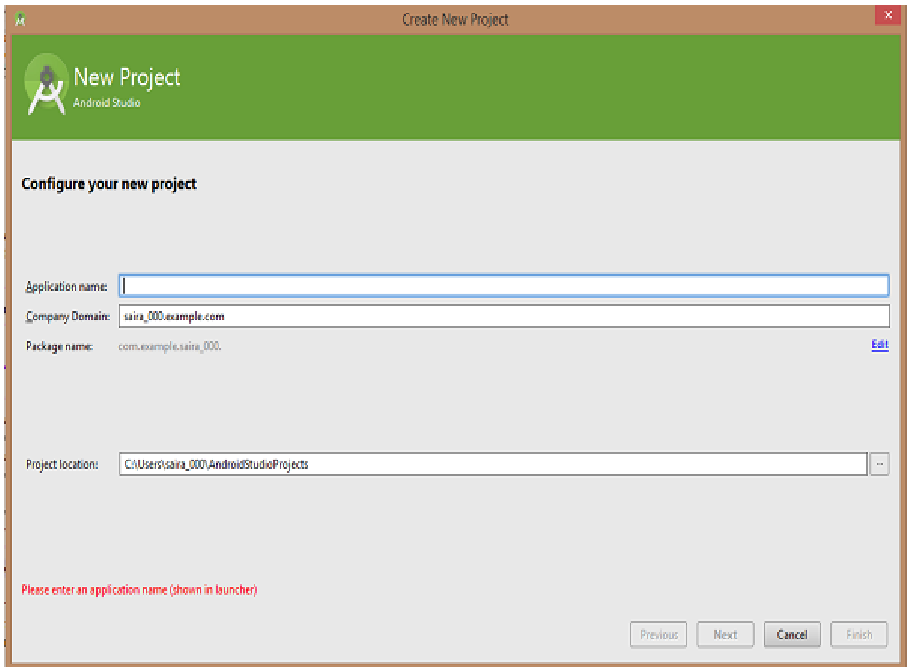

Android Studio Setup Wizard
Welcome to Android Studio
Recent Projects
No Project Open Yet
Quick Start
Start a new Android Studio project
Open an existing Android Studio project
Import an Android code sample
Check out project from Version Control
Import project (Eclipse ADT, Gradle, etc.)
Configure
Docs and How-Tos
Android Studio 1.1.0 Build 135.1740770. Check for updates now.

Figure 6: Android Setup Wizard [6]



Configure your new project

Application name:

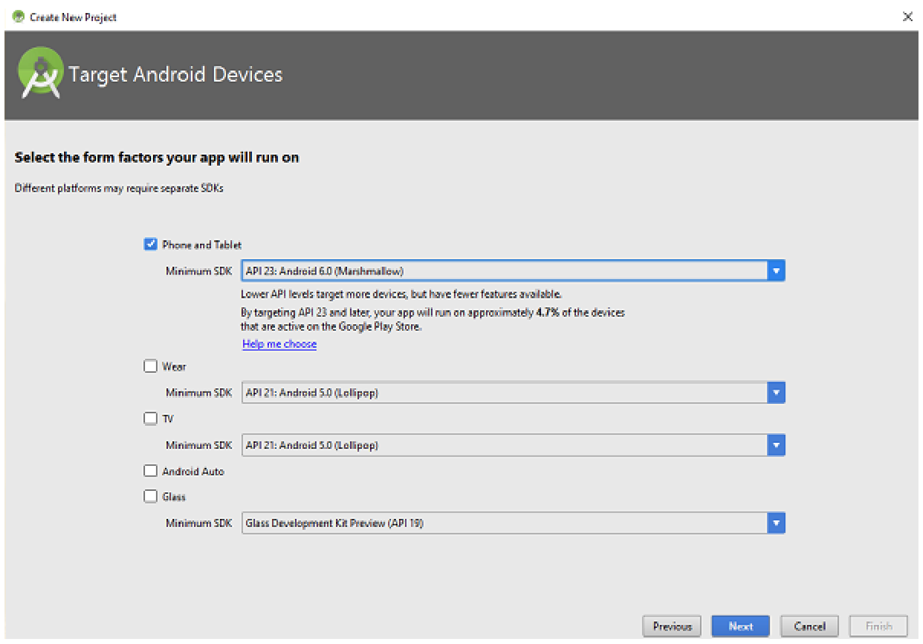
Company Domain: saira_000.example.com

Package name: com.example.saira_000. Edit

Project location: C:\Users\saira_000\AndroidStudioProjects

Please enter an application name (shown in launcher)

Figure 7: New Project Creation Window 1 [6]



Select the form factors your app will run on

Different platforms may require separate SDKs

[✓] Phone and Tablet

Minimum SDK API 23: Android 6.0 (Marshmallow)

Lower API levels target more devices, but have fewer features available.
By targeting API 23 and later, your app will run on approximately 4.7% of the devices that are active on the Google Play Store.
Help me choose

[ ] Wear

Minimum SDK API 21: Android 5.0 (Lollipop)

[ ] TV

Minimum SDK API 21: Android 5.0 (Lollipop)

[ ] Android Auto

[ ] Glass

Minimum SDK Glass Development Kit Preview (API 19)

Figure 8: New Project Creation Window 2 [6]

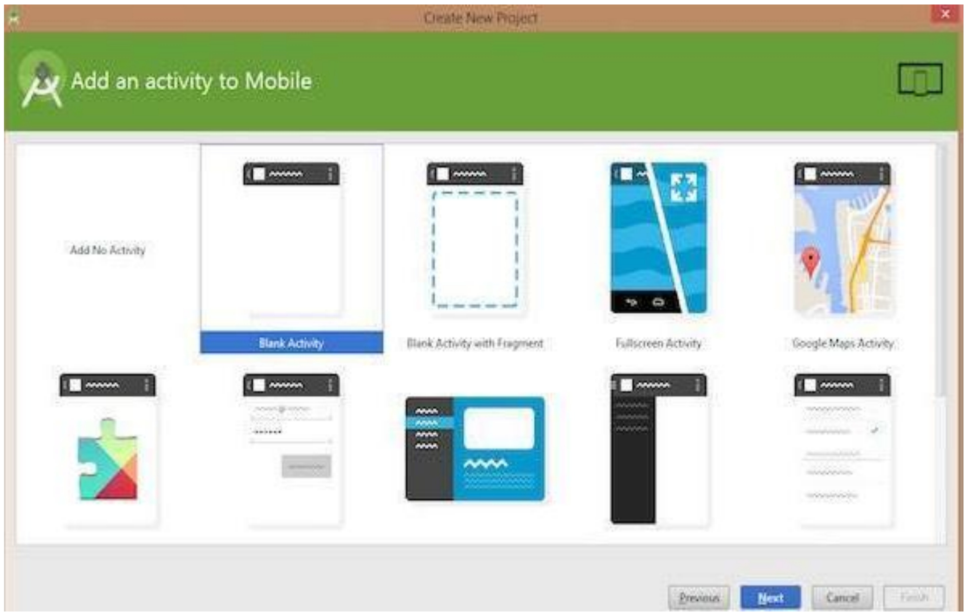


Figure 9: Add Activity Window [6]

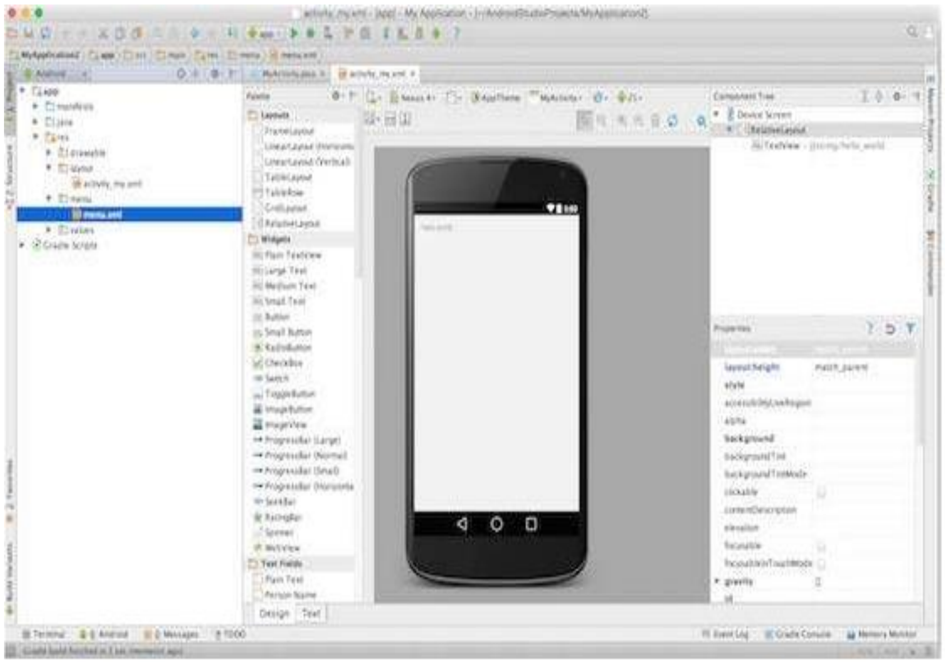


Figure 10: Hello World Android Project [6]

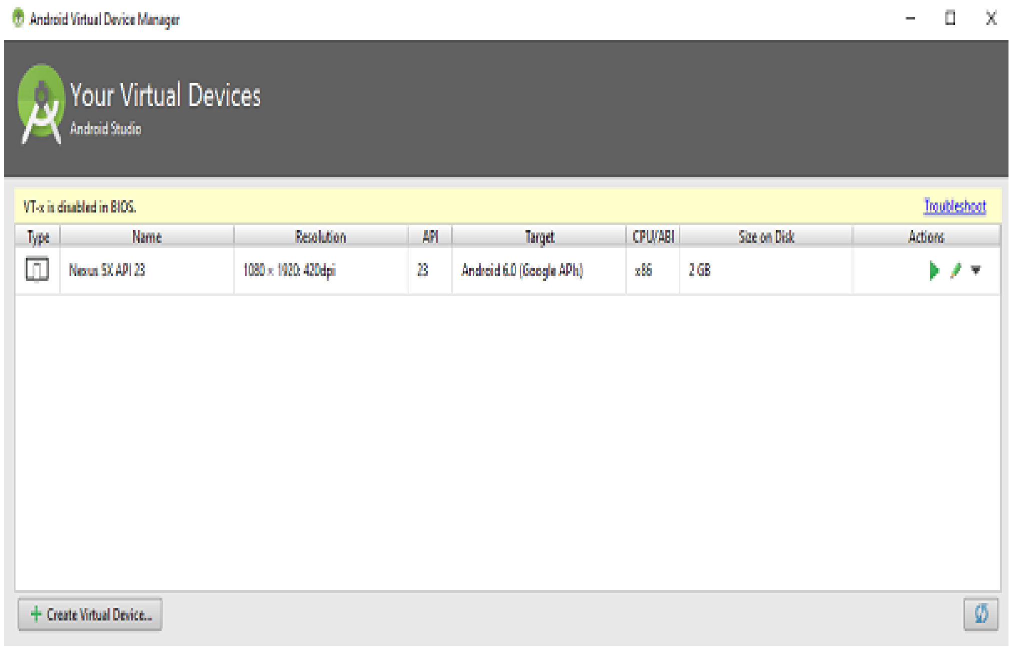
3.6 Creating Android Virtual Device

To test Android applications, users need Android Virtual Device (AVD). In order to set up a virtual device which is called Emulator, click on “AVD Manager” icon and then hit on “Create Virtual Device” button as shown in the figure 11. Choose the device definition to be tested on and hit next button. After selecting system image and verifying configuration, click on finish to complete setting up of the emulator.

3.7 Launching App on Android Device

An android device could also be used for testing app. Some of the apps for example, apps that works on wifi, cannot be tested on emulator since emulator does not support wifi. So, it becomes necessary to use android devices in order to test the applications.

To test the app, first it is required to enable developer settings. To enable it, go to “Settings” and then “About device”. Then click on “Build number” for 5 to 7 times which will enable developer options on phone. Now click on “Developer options” to turn on USB Debugging from the menu. Finally, connect the device to PC via USB cable and click on AVD manager to launch the app on android device.


Android Virtual Device Manager

Your Virtual Devices
Android Studio

VT-x is disabled in BIOS. Troubleshoot

Type	Name	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	Nexus 5X API 23	1080 × 1920: 420dpi	23	Android 6.0 (Google APIs)	x86	2 GB	

Figure 11: Android Virtual Device Manager

CHAPTER 4

FIREBASE

As discussed above, Firebase is a cloud based mobile backend as a service that lets the developer build a mobile app on IOS, Android and the Web without any backend management. Firebase helps the developer to develop high quality and bug free apps. Some of the most popular and important features of Firebase that makes the developer to achieve these qualities are Firebase Analytics, which is the heart of the Firebase, Firebase Authentication, which lets the user to sign in with various third party providers, Firebase Real-time database to store the real time data and Firebase Storage, which lets the users to store the data in the cloud platform.

In this chapter, we are going to discuss few of the prerequisites needed, in order to start the development using Firebase. Later, we will look into the features in detail.

4.1 Prerequisites

1. Install Firebase SDK

   In order to add Firebase to Android project, there are certain prerequisites that are to be taken care. The running device must be Android 4.0 or above with Google play service 10.2.1 or higher.

2. Adding the app to Firebase console

   a. Create a Firebase project in the firebase console by clicking on Create New Project. Otherwise, click on Import Google Project.
   b. Click on Add Firebase to Android app and follow the steps to add the app to the Firebase.
   c. Provide the app’s package name.
3. Android Studios 1.5 and above [7].

4.2 Firebase Analytics

It is extremely important to build a successful app. For build a successful app, there are plenty of different kinds of analytics tools for the developers to use. There are in-app behavioral analytics, which measures the app users, what they are doing and so on. There is also attribute analytics, which measure effectiveness of advertizing and growth campaigns. Quite often these work is being done using completely different libraries that is, reports are existing in various tools across the web. And making them interact which other is quite challenging. Firebase Analytics makes it easy to provide all the data that mobile developers need in one single place. It gives the developers free unlimited logging and reporting. It is not just important to track and report who the users are and what they are doing. It is as essential as to discover how different group of users behaves by setting custom user properties. For example, if a developer has created a music app and want to find out whether classical music fans are browsing more albums than jazz fusion fans. Then, the developer can easily get the data by setting up custom user properties.

By installing Firebase SDK, Analytics automatically starts providing insight into the app and starts receiving the demographic information about the app users, how often they visit the app and so on. Firebase Analytics not only measures what is happening inside the app, but it also lets developers to combine behavioral reporting and attribution reporting [8].

4.3 Firebase Authentication for Android

Many of the applications need to know the identity of the user so they can provide a customized experience and keep their data secure. Firebase supports different ways for users to authenticate. If users want to authenticate with their Facebook account, it could be easily built. Firebase Auth has built in functionality for third party providers such as Facebook, Twitter, Github and Google. It can also integrate with the existing account system. Developers are given the choice about how to present login to user.

Once the user is authenticated, information about the user is returned to the device using callbacks. Later, this information contains a unique ID (UID) which is guaranteed to be distinct across all the providers, never changing for specific authenticated users. This ID will be used to identify the user and what part of the backend system they are authorized to access. Firebase will also manage the user session so that the user will remain logged in after the browser or application restarts [9].

4.4 Firebase Real-time Database

Firebase real-time database makes setting up and maintaining database easy. It also makes data synchronized in real time and provides offline support. It lets developers to store and sync data between users in real time. This makes it easy for users to access their data from any device, web or mobile, and it helps user collaborate with one another. Whenever users update data in the real-time database, it stores the data in the cloud and simultaneously notifies all interested devices in milliseconds.

The real-time database could also be optimized for offline use. Whenever users lose their connection, the database STK uses a local cache on the device to serve and store changes. This means that when users come back online, their local data is automatically synchronized. So then how to keep ones data secure? The answer is using the security rules. Security rules can be used to specify who has access to what data and how the database should be structured. By default the access to read and write the data is restricted to only the authenticated users. But, if the rules are made public, then the data is open to public and the users who have not using the app could also access the data. The security rules are securely stored with the real-time database on the servers [10].

To set up the Firebase Real-time database, first add the Firebase Real-time database dependency to build.gradle file and write it to the database by using getInstance() method.

4.5 Firebase Storage

We all know from experience that people love to share things about themselves, such as photos, video, GIFs and sometimes docs, PDFs and many more. Storing files and sharing them across is little complicated. So how does one make that happen? Here is the Firebase Storage that helps to make it happen. Firebase Storage API lets users to upload files to cloud so they could be shared with anyone else. And if there are any specific rules for sharing files with certain users, it could be protected for users logged in with Firebase Authentication. Security, of course, is the primary concern. All transfers are performed over a secure connection. Also all transfers with Firebase Storage API are robust and will automatically resume in case the connection is lost. This is essential for transferring large files over slow or unreliable mobile connections. Finally, Firebase Storage backend by Google Cloud Storage scales to petabytes in order to store as much as data that users want with high availability and global redundancy [11].

CHAPTER 5

IMPLEMENTATION

5.1 Adding Android project to Firebase

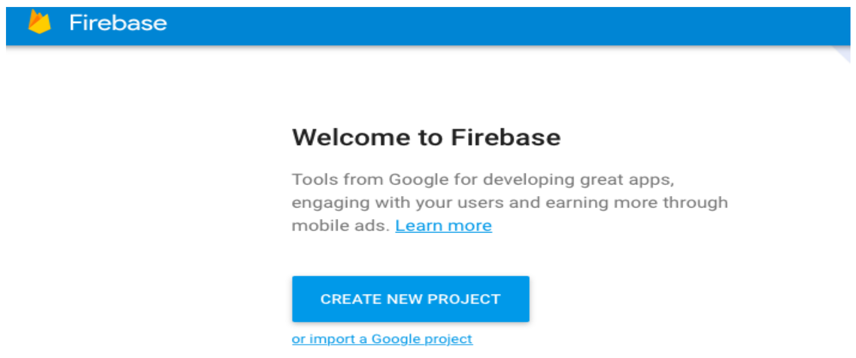
Adding the project to Firebase could be done by

- Clicking on tools in Android Studios and then selecting Firebase Assistant window.
- Expand one of the feature and click on Connect to Firebase button.

Android project can also be added manually by logging in to the Firebase console. But, using Firebase Assistant to add the project would save time. Also, Firebase Assistant automatically syncs up the gradle dependencies so that developers need not have to spend much time on adding the dependencies.

To add project manually, below given steps are followed

1. Login to Firebase console for Android and click on create new project as shown in the figure 12 below.



Firebase

Welcome to Firebase

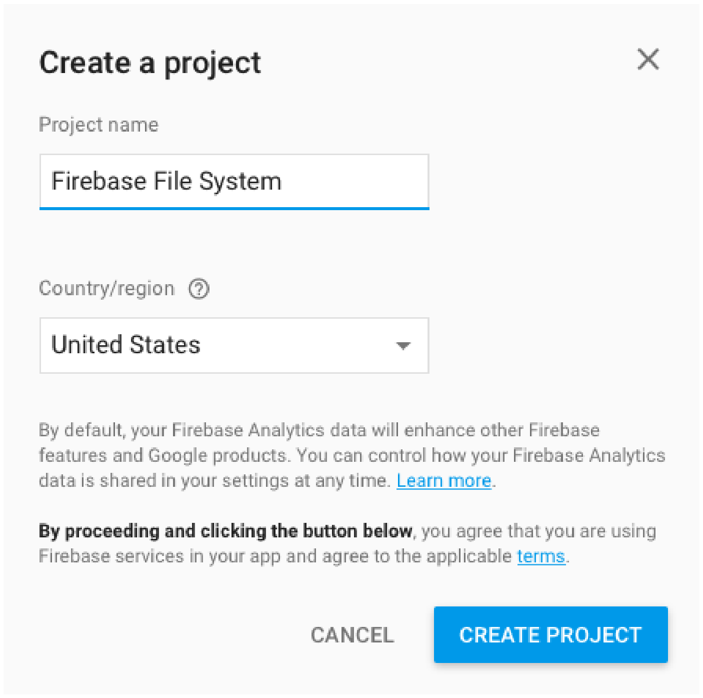
Tools from Google for developing great apps, engaging with your users and earning more through mobile ads. Learn more

CREATE NEW PROJECT

or import a Google project

Figure 12: Firebase Welcome Window

2. A pop up appears on the screen asking for Project name and Country. Enter the information and hit create project button as in figure 13 below.



Create a project

Project name: Firebase File System

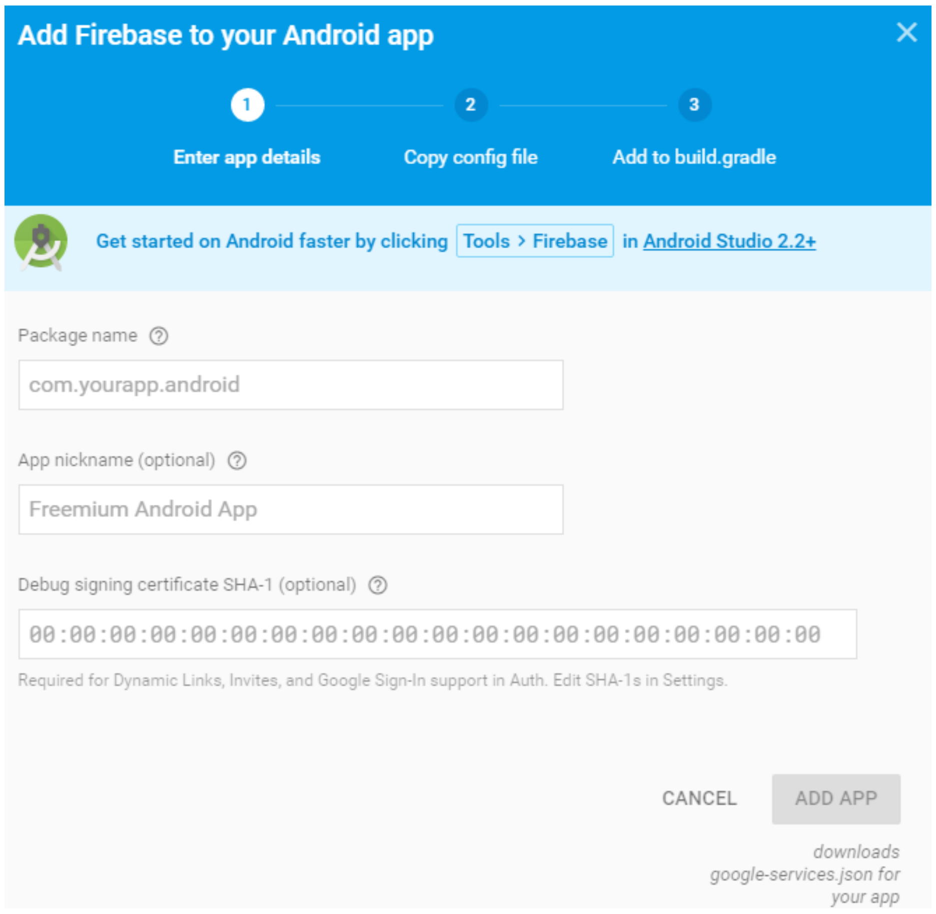
Country/region: United States

By default, your Firebase Analytics data will enhance other Firebase features and Google products. You can control how your Firebase Analytics data is shared in your settings at any time. Learn more.

By proceeding and clicking the button below, you agree that you are using Firebase services in your app and agree to the applicable terms.

Figure 13: Adding Project to Firebase

3. Click on “Add Firebase to your Android app” button to add the project to Firebase which pops up a window asking for information about the app such as package name and SHA-1 certificate code, which is required for Dynamic links, Invites, and Google Sign-In support in Auth as shown in the figure 14 below.



Add Firebase to your Android app

1 Enter app details
2 Copy config file
3 Add to build.gradle

Get started on Android faster by clicking Tools > Firebase in Android Studio 2.2+

Package name: com.yourapp.android

App nickname (optional): Freemium Android App

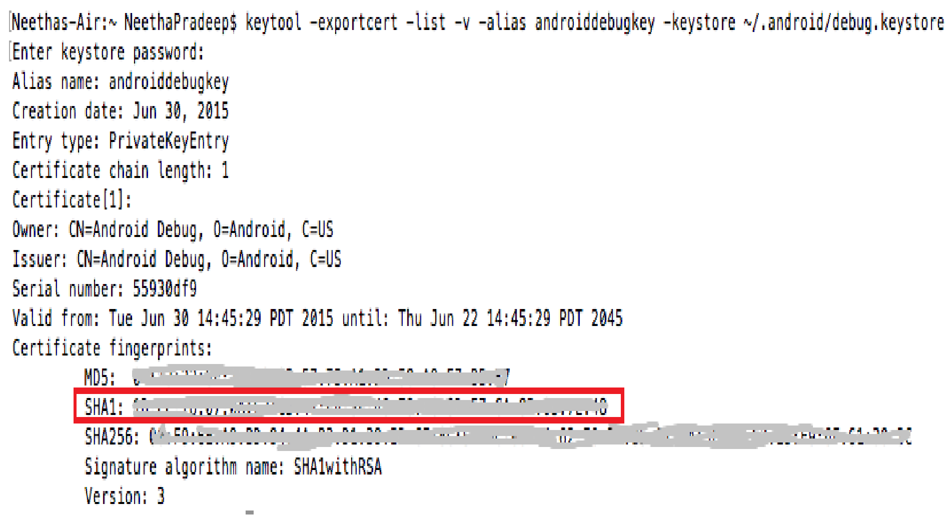
Debug signing certificate SHA-1 (optional): 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00

Required for Dynamic Links, Invites, and Google Sign-In support in Auth. Edit SHA-1s in Settings.

Figure 14: SHA-1 Certificate Window

4. To obtain SHA-1 certificate code, go to the terminal and type the below code line in the command line and enter ‘android’ as the password when prompted. This would give a SHA-1 certificate number, which is shown in figure 15. Copy paste the certificate number in the box provided and clicks on “Add App” button.

```
keytool -exportcert -list -v -alias androiddebugkey -keystore
~/.android/debug.keystore
```



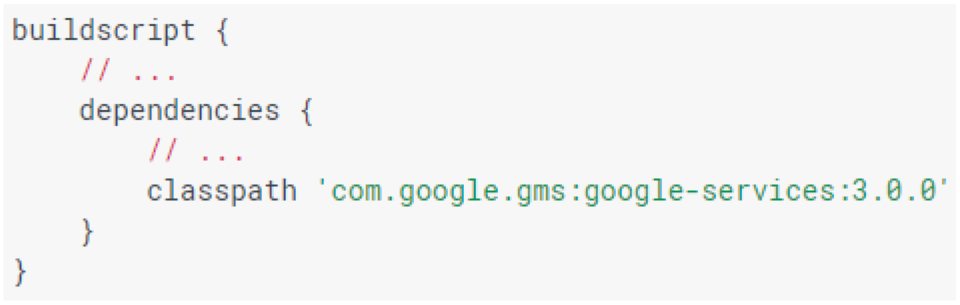
```
Neethas-Air:~ NeethaPradeep$ keytool -exportcert -list -v -alias androiddebugkey -keystore ~/.android/debug.keystore
Enter keystore password:
Alias name: androiddebugkey
Creation date: Jun 30, 2015
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Android Debug, O=Android, C=US
Issuer: CN=Android Debug, O=Android, C=US
Serial number: 55930df9
Valid from: Tue Jun 30 14:45:29 PDT 2015 until: Thu Jun 22 14:45:29 PDT 2045
Certificate fingerprints:
         MD5:  [redacted]
         SHA1: [redacted]
         SHA256: [redacted]
         Signature algorithm name: SHA1withRSA
         Version: 3
```

Figure 15: SHA-1 Certificate Code

5. Clicking on the “Add App” button would download a google-services.json file. Copy the json file into the project’s module folder.

6. Lastly, integrate app to Firebase library by adding the below lines in the build.gradle file

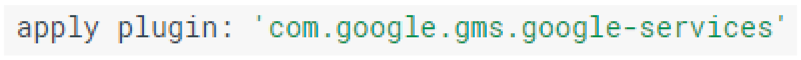
   In the root-level build.gradle file, add as shown in the below figure 16 and 17.



```
buildscript {
    // ...
    dependencies {
        // ...
        classpath 'com.google.gms:google-services:3.0.0'
    }
}
```

Figure 16: Firebase Library Dependency 1 [12]

And for the module gradle file, add the below line at the end of the file.



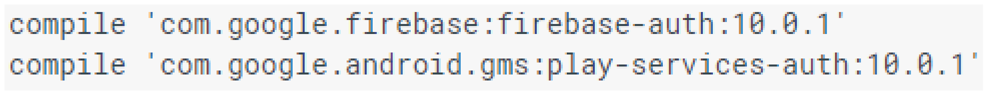
```
apply plugin: 'com.google.gms.google-services'
```

Figure 17: Firebase Library Dependency 2 [12]

5.2 Firebase Google Authentication

After registering the android project to Firebase, it is now time to add Firebase Authentication to the app.

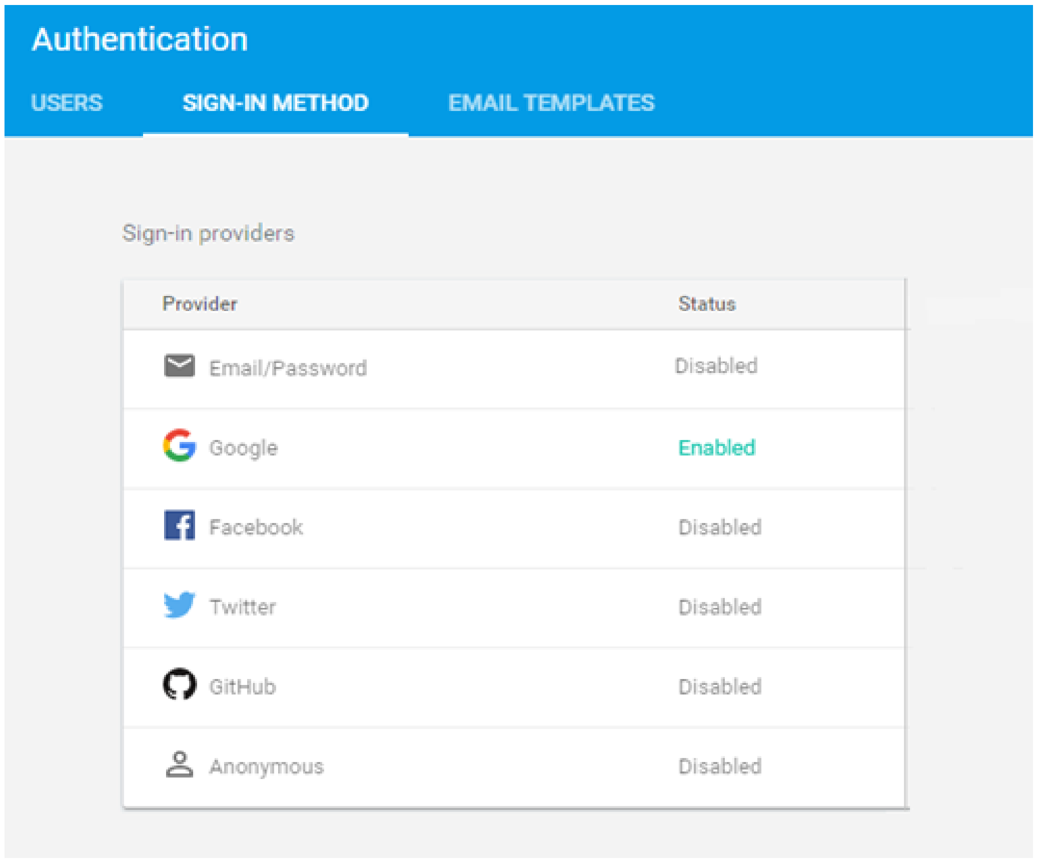
- To add the gradle dependency for Authentication, add the below code in the build.gradle file as shown in Figure 18 below.



```
compile 'com.google.firebase:firebase-auth:10.0.1'
compile 'com.google.android.gms:play-services-auth:10.0.1'
```

Figure 18: Firebase Authentication Gradle dependency [13]

- As mentioned earlier, there are multiple third party providers such as Google, Twitter, Facebook, Github etc, which could be used for Authentication. Figure 19 shows the variety of sign-in options that are built for Firebase Authentication. Enabling any one of the options will help the app in successful Authentication.



Sign-in providers

Provider	Status
Email/Password	Disabled
Google	Enabled
Facebook	Disabled
Twitter	Disabled
GitHub	Disabled
Anonymous	Disabled

Figure 19: Sign-In Methods

5.2.1 Integrating with Google Account

In the onCreate event, the default configuration for Google sign-in is implemented using GoogleSignInOptions.builder object and a token id for authenticated users is requested. This object is passed to another object GoogleApiClient, which provides a common entry point to all the Google play services and manages the network connection between the user's device and each Google sign-in service.

Figure 20 shows the login account screen as a result of the code implementation shown below.

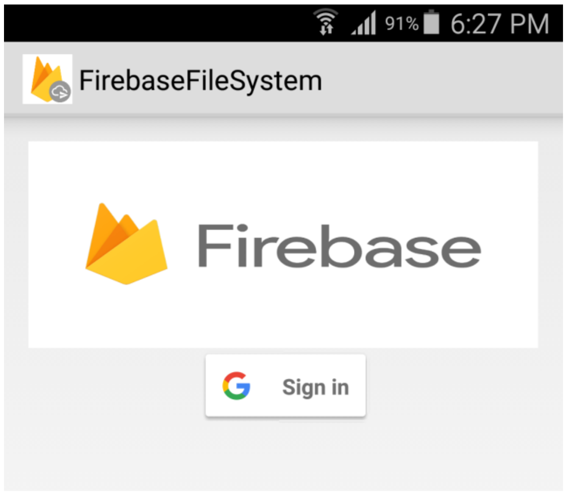


Figure 20: Google Sign-in Screen

```
GoogleSignInOptions gso = new
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
            .requestIdToken(getString(R.string.default_web_client_id))
            .requestEmail()
            .build();
 mGoogleApiClient = new GoogleApiClient.Builder(this)
            .enableAutoManage(this, this)
            .addApi(Auth.GOOGLE_SIGN_IN_API, gso)
            .build();

 mAuth = FirebaseAuth.getInstance();
```

5.2.2 Choosing Google Account from the Intent

User clicks on sign-in button to launch the sign-in intent by passing the signinIntent and request code for startActivityForResult. The user will be prompted to select a

Google account to log in as shown in figure 21. Once the user selects the account intent data, request code and result code are returned back to the activity. Get the GoogleSignInResult data from intent data received and validate if Google sign-in was successful. If yes, get GoogleSignInAccount details from the GoogleSignInResult object and pass it for Firebase Authentication. Get the token id from the GoogleSignInAccount object and use it to extract the credentials for Firebase login using GoogleAuthProvider.getCredential(). Use these credentials for signing in to Firebase with the help of signInWithCredential method as shown in the implementation code below.

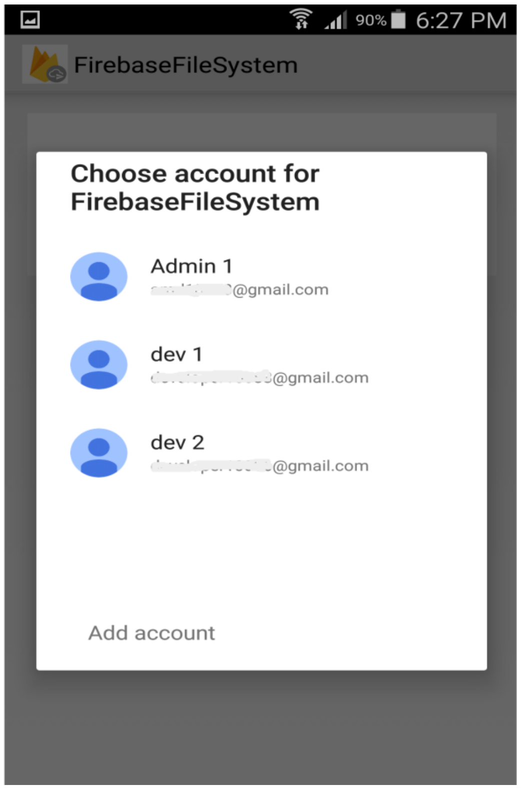
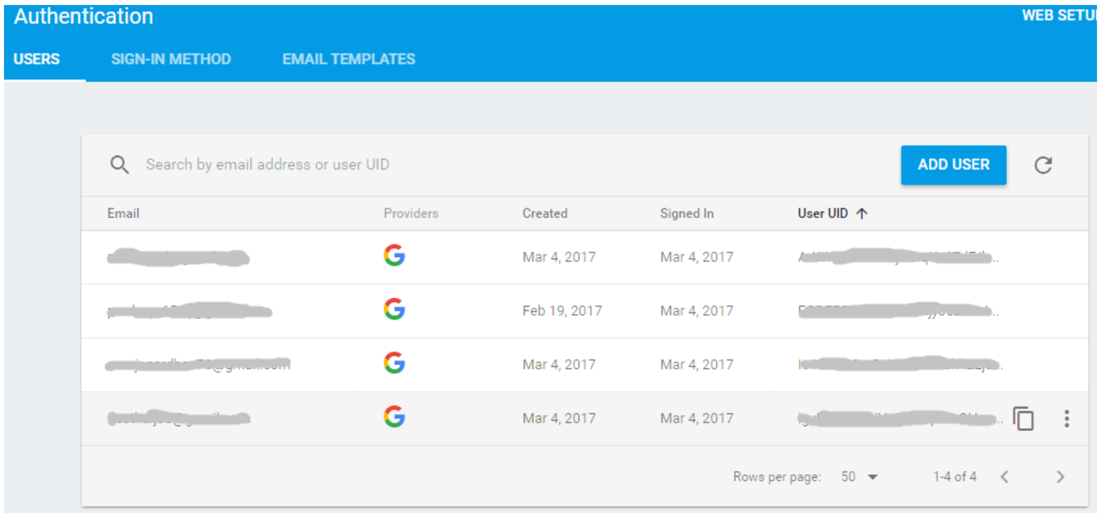


Figure 21: Choose Account Screen

```
   private void signIn() {
      Auth.GoogleSignInApi.signOut(mGoogleApiClient);
      Intent signInIntent = Auth.GoogleSignInApi.getSignInIntent(mGoogleApiClient);
      startActivityForResult(signInIntent, RC_SIGN_IN);
   }

   @Override
   public void onActivityResult(int requestCode, int resultCode, Intent data) {
      super.onActivityResult(requestCode, resultCode, data);
      if (requestCode == RC_SIGN_IN) {
         GoogleSignInResult result = Auth.GoogleSignInApi.
         getSignInResultFromIntent(data);
         if (result.isSuccess()) {
            // Google Sign In was successful, authenticate with Firebase
            GoogleSignInAccount account = result.getSignInAccount();
            firebaseAuthWithGoogle(account);
         }
      }
   }

   private void firebaseAuthWithGoogle(GoogleSignInAccount acct) {
      AuthCredential credential = GoogleAuthProvider.getCredential(acct.getIdToken(),
null);
      mAuth.signInWithCredential(credential)
            .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
               @Override
               public void onComplete(@NonNull Task<AuthResult> task) {
                  Log.d(TAG, "signInWithCredential:onComplete:" + task.isSuccessful());
                  if (!task.isSuccessful()) {
                        Log.w(TAG, "signInWithCredential", task.getException());
                        Toast.makeText(SignInActivity.this, "Authentication failed.",
                        Toast.LENGTH_SHORT).show();
           }else{
               if(FirebaseAuth.getInstance().getCurrentUser().
               getDisplayName().split(";").length>1) {
                   if(FirebaseAuth.getInstance().getCurrentUser().
                   getDisplayName().split(";")[1].equals("ADMIN")) {
                        Intent MainIntent = new Intent(SignInActivity.this,
                        AdminMainActivity.class);
                        startActivity(MainIntent);
            } else if (FirebaseAuth.getInstance().
                         getCurrentUser().getDisplayName().split(";")[1].
                         equals("USERS")) {
                               Intent MainIntent = new
Intent(SignInActivity.this,UserMainActivity.class);
startActivity(MainIntent);
          }
       }
      else{
               spnUsrType.setVisibility(View.VISIBLE);
               findViewById(R.id.sign_in_button).setVisibility(View.GONE);
               btnGo.setVisibility(View.VISIBLE);
            }
          }
       }
   });
  }
```

Once the user has finished creating the account, the information of the user gets saved in the Firebase console as a part of Authentication. In the users tab, one can view the email address and UID which gets created after the creation of the account. Figure 22 shows the Firebase Auth screen with the data saved in the backend.


Authentication

USERS | SIGN-IN METHOD | EMAIL TEMPLATES | WEB SETU

Search by email address or user UID | ADD USER

Email	Providers	Created	Signed In	User UID ↑
[redacted]	G	Mar 4, 2017	Mar 4, 2017	[redacted]
[redacted]	G	Feb 19, 2017	Mar 4, 2017	[redacted]
[redacted]	G	Mar 4, 2017	Mar 4, 2017	[redacted]
[redacted]	G	Mar 4, 2017	Mar 4, 2017	[redacted]

Rows per page: 50   1-4 of 4

Figure 22: Firebase Auth Users

5.3 Firebase Real-Time Database

In order to add database into the app, first add the gradle dependency in the build.gradle file, as shown in figure 23.

```
compile 'com.google.firebase:firebase-database:10.0.1'
```

Figure 23: Firebase Real-Time Gradle dependency [8]

5.3.1 Reading from the Database

Firebase data is retrieved and read from the firebase database by attaching asynchronous listener to the reference. Listener addListenerForSingleValueEvent() method reads the data from the path and looks for the changes to add a ValueEventListner() to the database. It uses an event callback onDataChange() method to read a static snapshot of the contents at the path specified. Below is the code that shows the implementation details, which include reading the data from the database.

```
    fbdb = FirebaseDatabase.getInstance();
     fbdbRef = fbdb.getReference();
     fbdbRef.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
           lstGroup.clear();
           String
              strGroups="ADMIN/"+FirebaseAuth.getInstance().getCurrentUser().getUid()
              +"/GROUPS/";
           DataSnapshot childData = dataSnapshot.child(strGroups);
           for (DataSnapshot Groups : childData.getChildren()){
               lstGroup.add(Groups.getKey());
            }
           ArrayAdapter<String> GroupAapter = new
              ArrayAdapter<String>(GroupActivity.this,
              android.R.layout.simple_list_item_1, lstGroup);
           lstGroupDisp = (ListView) findViewById(R.id.lstgroups);
           lstGroupDisp.setAdapter(GroupAapter);
        }
  @Override
        public void onCancelled(DatabaseError error) {
           Log.w(TAG, "Failed to read value.", error.toException());
        }
     });
```

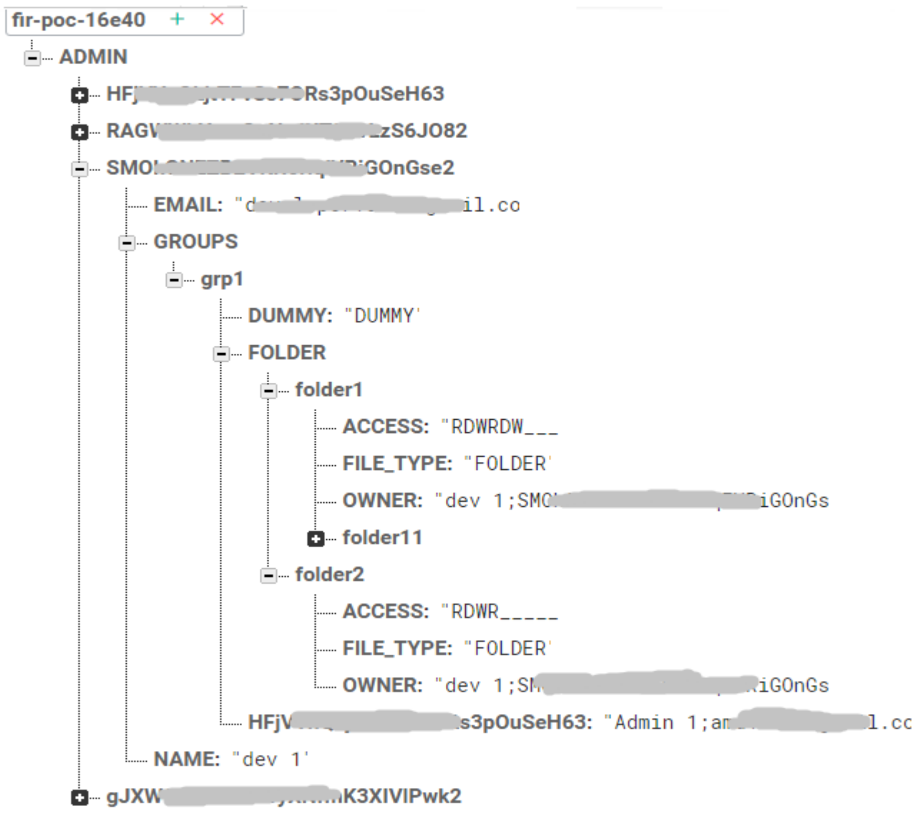
5.3.2 Writing to the Database

JSON type hash map is used to store the firebase data into the Firebase Database. The method setValue() is used to perform write operation to firebase database stored in the form of key value pair. Below is the implementation code for writing the data to the firebase database.

```
database = FirebaseDatabase.getInstance();
 DBUserListRef = database.getReference();
 Map<String, Object> childUpdates = new HashMap<>();
 childUpdates.put(strRole + "/" +
FirebaseAuth.getInstance().getCurrentUser().getUid() + "/" + "EMAIL",
mEmailField.getText().toString());
 childUpdates.put(strRole + "/" +
FirebaseAuth.getInstance().getCurrentUser().getUid() + "/" + "NAME",
FirebaseAuth.getInstance().getCurrentUser().getDisplayName().split(";")[0]);
DBUserListRef.updateChildren(childUpdates);
```

5.3.3 Implementation of Creating Groups and Access Control

When users log in, information such as email addresses, groups, folders, usernames are stored in Firebase Real-Time Database. This information is stored in the form tree structure. Group information is stored beneath each user’s UID under the “Groups” subtitle by the respective group names. And the folders pertaining to each group are stored under the “Folder” subtitle. Access control, file type and the owner of the file/directory are some of the other information that is also stored under every folder created. Figure 24 shows the tree structure of the Firebase Database storage.



```
fir-poc-16e40  +  ×
ADMIN
    HFj[redacted]Rs3pOuSeH63
    RAGV[redacted]zS6JO82
    SMO[redacted]GOnGse2
        EMAIL: "d[redacted]il.co
        GROUPS
            grp1
                DUMMY: "DUMMY'
                FOLDER
                    folder1
                        ACCESS: "RDWRDW___
                        FILE_TYPE: "FOLDER'
                        OWNER: "dev 1;SMO[redacted]iGOnGs
                        folder11
                    folder2
                        ACCESS: "RDWR_____
                        FILE_TYPE: "FOLDER'
                        OWNER: "dev 1;S[redacted]iGOnGs
                HFj[redacted]s3pOuSeH63: "Admin 1;am[redacted]l.cc
        NAME: "dev 1'
    gJXW[redacted]K3XIVIPwk2
```

Figure 24: Firebase Database Tree Structure

The home page looks like the figure 25 shown below, which consists of “Groups”, “Users Access Control” and “Group Shared with me” icons. By clicking on “Groups” icon users are navigated to GroupActivity screen, where users can create groups and add folders to the groups. The access permissions for a particular folder are displayed at the lower right side corner of every folder name and its owner is shown in the opposite side. These access permissions consists of series of 9 letters, where the first three letters are the permission set for the owner, next three for the group users and the last three are for other users outside the group. By default, the access permissions is set to read, download and write for the owners and read only for the group users as shown in figure 26.

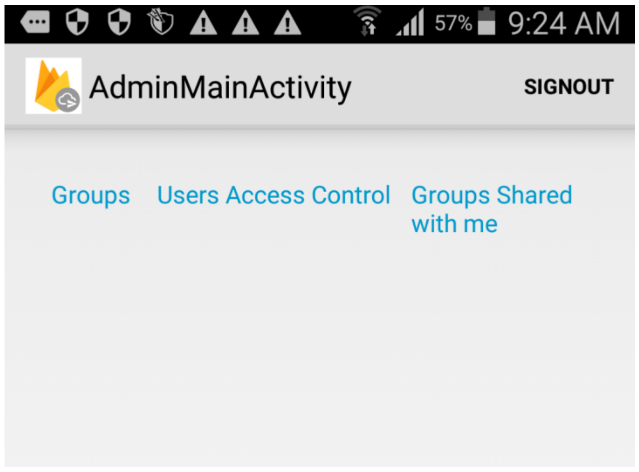


Figure 25: User Home Page

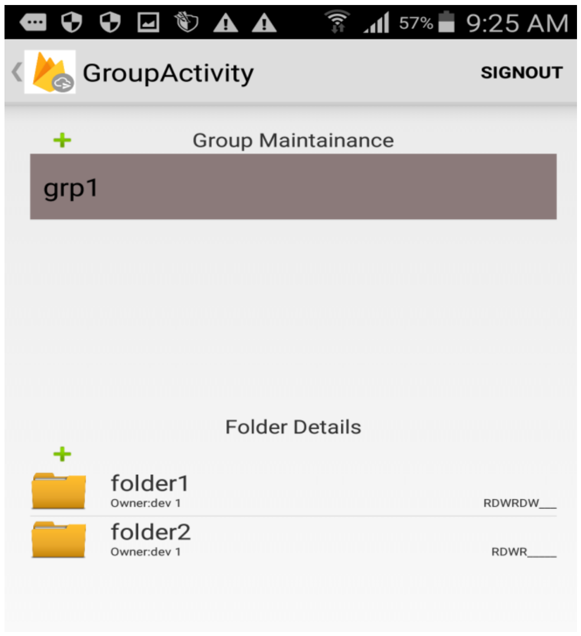


Figure 26: Group Creation Screen

During onCreate event in GroupActivity, list of existing groups under the user dashboard are retrieved from the Firebase Database using GetGroupList method and displayed in the form of a list. A create and remove buttons are instantiated and an OnClick listener is attached to these buttons in the list. These 2 buttons will be used to add or remove Groups. To retrieve data from Firebase Real-time Database an instance is created to the database using getInstance() and the a reference object is being added. After that, reference object is attached to addListenerForSingleValueEvent. This would return entire database object in the form of datasnapshot. By doing this, user can retrieve the required data based on the reference path to the required child element in the datasnapshot object. Next, in the AccessControl activity, GetGroupUserNames method will be used to fetch the list of users assigned to a particular group created under the current user and GetUserNames method will fetch the list of users, who are not under any group as shown in the below implemented code.

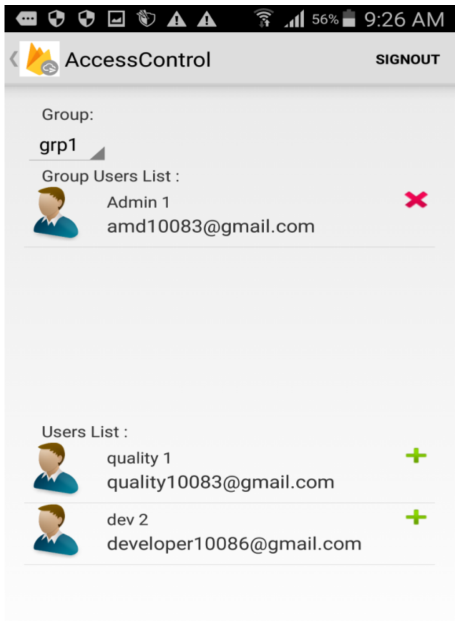
```
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_group);
   getActionBar().setDisplayHomeAsUpEnabled(true);
   GetGroupList();
   btnCreate = (Button) findViewById(R.id.btncreate);
   btnRemove=(Button)findViewById(R.id.btnremove);
   txtGropName=(EditText)findViewById(R.id.txtgroupnm);
   btnCreate.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view) {
         //Below will create an entry in firebase database to
         fbdb = FirebaseDatabase.getInstance();
         fbdbRef = fbdb.getReference();
```

```
            Map<String, Object> childUpdates = new HashMap<>();
            childUpdates.put("ADMIN/"+FirebaseAuth.
                                                            getInstance().
                                                    getCurrentUser().
                                                    getUid()+"/GROUPS/"+
                                                    txtGropName.getText().
                                                    toString()+"/DUMMY","DUMMY");
            fbdbRef.updateChildren(childUpdates);
            GetGroupList();
        }
    });

    btnRemove.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            fbdb = FirebaseDatabase.getInstance();
            if(!strSelGroup.equals("")) {
                fbdbRef = fbdb.getReference("ADMIN/" + FirebaseAuth.getInstance().
                        getCurrentUser().getUid() +
                        "/GROUPS/" + strSelGroup);
                fbdbRef.removeValue();
                GetGroupList();
            }
        }
    });

  private void GetGroupUserNames(String Item){
    lstGrpUser.clear();
    lstGrpAllUser.clear();
    final String strgrpNm =Item;
    FbDB = FirebaseDatabase.getInstance();
    FbDBRef = FbDB.getReference();
    FbDBRef.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            DataSnapshot childData = dataSnapshot.child("ADMIN/"+
              FirebaseAuth.getInstance().
            getCurrentUser().getUid()+"/GROUPS/"+strgrpNm);

           for (DataSnapshot groups : childData.getChildren()){
              if(!groups.getKey().equals("DUMMY")){
                 String strUID = groups.getKey();
                 String strName = groups.getValue().toString().split(";")[0]+";"+strgrpNm;
                 String strEmail = groups.getValue().toString().split(";")[1];
                 lstGrpUser.add(new UserListClass(strName,strEmail,strUID,
                                    R.drawable.user));
              }
           }
           DataSnapshot AllGrpUsers = dataSnapshot.child("ADMIN/"+ FirebaseAuth.
             getInstance().getCurrentUser().getUid()+"/GROUPS");
           for (DataSnapshot Grp : AllGrpUsers.getChildren()){
              for (DataSnapshot GrpUsr : Grp.getChildren()) {
                 if (!GrpUsr.getKey().equals("DUMMY")) {
                    String strUID = GrpUsr.getKey();
                    String strName = GrpUsr.getValue().toString().
                     split(";")[0] + ";" +        strgrpNm;
                    String strEmail = GrpUsr.getValue().toString().split(";")[1];
                    lstGrpAllUser.add(new UserListClass(strName,
                     strEmail, strUID, R.drawable.user));
                 }
              }
           }
           ArrayAdapter<UserListClass> adapter = new AccessControl.MyListAdapter();
           ListView list = (ListView) findViewById(R.id.lstgrpusers);
           list.setAdapter(adapter);
        }

        @Override
        public void onCancelled(DatabaseError error) {
           Log.w(TAG, "Failed to read value.", error.toException());
        }
     });
  }
```

```
private void GetUserNames(){
    lstUser.clear();
    final String strgrpNm =selectedItem;
    FbDB = FirebaseDatabase.getInstance();
    FbDBRef = FbDB.getReference();
    FbDBRef.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            DataSnapshot childData = dataSnapshot.child("USERS");
            for (DataSnapshot users : childData.getChildren()){
              DataSnapshot Admin = users.child("ADMINGRP/" +
                      FirebaseAuth.getInstance().getCurrentUser().getUid()+"/GROUP");
               if(!Admin.exists()||!Admin.getValue().toString().equals(selectedItem)) {
                    DataSnapshot UserName = users.child("NAME");
                    DataSnapshot UserEmail = users.child("EMAIL");
                    boolean blnadd = true;
                    for (UserListClass user : lstGrpAllUser) {
                        if (user.getStrUID().equals(users.getKey())) {
                            blnadd = false;
                            break;
                        } else {
                            blnadd = true;
                        }
                    }
                    if (blnadd) {
                        lstUser.add(new UserListClass(UserName.getValue().toString(),
                         UserEmail.getValue().toString(), users.getKey(), R.drawable.user));
                    }
               }
            }
            lstUser.removeAll(lstGrpAllUser);
                   ArrayAdapter<UserListClass> adapter = new
                   AccessControl.MyUserListAdapter();
            ListView list = (ListView) findViewById(R.id.lstusers);
            list.setAdapter(adapter);
        }
        @Override
        public void onCancelled(DatabaseError error) {
            Log.w(TAG, "Failed to read value.", error.toException());
        }
    });
}
```

The UI figure 27 shows the access control activity screen. Select the appropriate group from the dropdown to add and remove the user from the group.

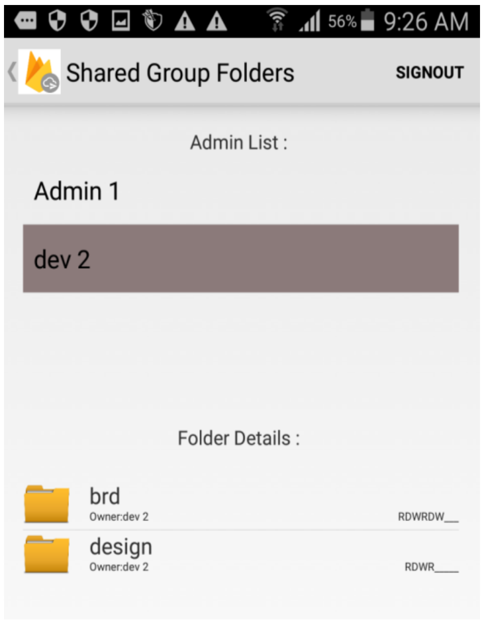
Users who are granted with the group access can view the folders by clicking on “Folder Shared with me” as shown in figure 28. Based on the access permission granted by the group owners, users can access the folders. Figure 29 shows the various options that users gain when all the 3 access permissions are set. Long click on one of the folders/file in order to view the different options implemented. These options are delete, change access permission and downloading the files.



Group:
grp1
Group Users List :
Admin 1
amd10083@gmail.com

Users List :
quality 1
quality10083@gmail.com
dev 2
developer10086@gmail.com

Figure 27: Access Control Screen



Admin List :
Admin 1
dev 2

Folder Details :
brd
Owner:dev 2
RDWRDW__
design
Owner:dev 2
RDWR____

Figure 28: Shared Folder View Screen

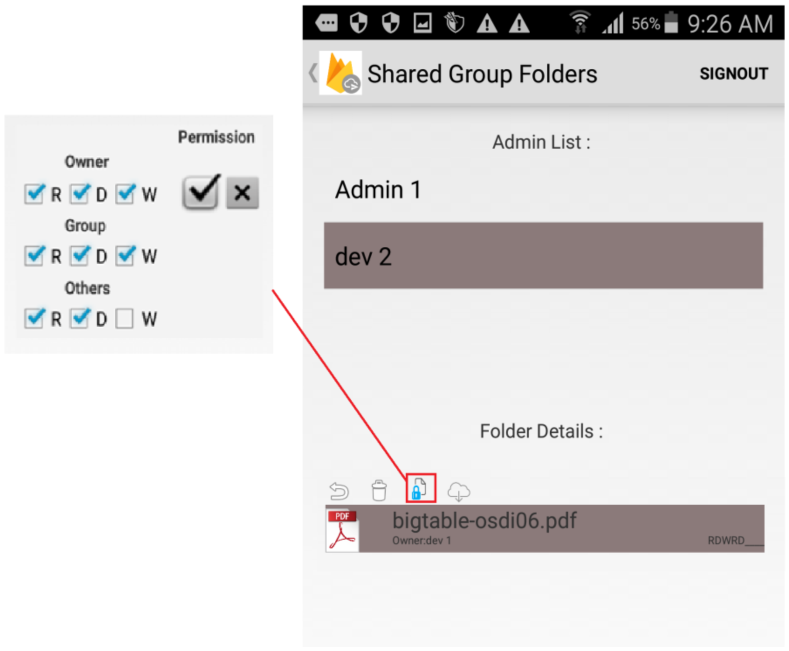
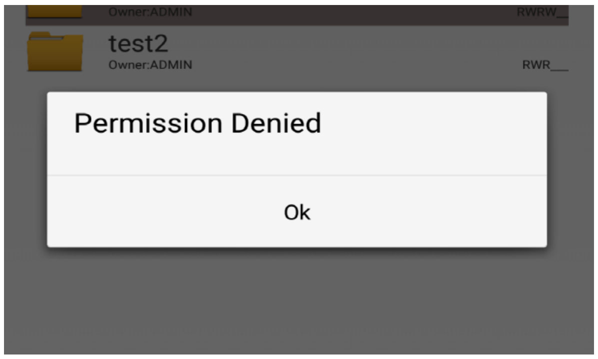


Figure 29: Users Options

If users are not granted with any of the access for the folders or a file, then users would get a “Permission Denied” message as shown in figure 30.



Permission Denied

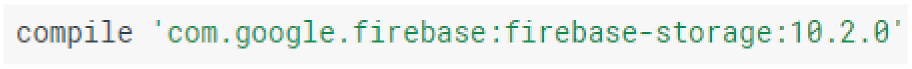
Ok

Figure 30: Denial Message Screen

5.4 Firebase Storage Implementation

Files that are uploaded by users are stored in Firebase Storage in a secure way. Based on the access provided by the owner, group users and others can access the files.

To start with Firebase Storage, add the below code to build.gradle file, as shown in figure 31.



```
compile 'com.google.firebase:firebase-storage:10.2.0'
```

Figure 31: Firebase Storage Gradle dependency [11]

To access the Firebase Storage, user must create Firebase Storage instance and get storage reference for the instance created. After that, get the URI of the file that is to be stored in firebase. Create an UploadTask for the reference object and pass the file URI to upload the file in firebase storage and add addOnSuccessListener, addOnFailureListener listener to the upload task. If upload fails, onFailure method would fire an exception message or if upload is successful, onSuccess method will fire and UploadTask.TaskSnapshot object will be made available to this method. Extract DownloadURL from TaskSnapshot object and update the same to Firebase Database that can be used for downloading of this file from Firebase Storage as shown in the below given code.

```
private void FireBaseStorage(String fpath) {
        Uploadprogress=new ProgressDialog(UserFolderView.this);
        Uploadprogress.setMessage("Uploading File...");
        Uploadprogress.setProgressStyle(ProgressDialog.STYLE_SPINNER);
        Uploadprogress.setIndeterminate(true);
        Uploadprogress.show();
        FirebaseStorage storage = FirebaseStorage.getInstance();
        final StorageReference storageRef = storage.getReferenceFromUrl("gs://fir-poc-
         16e40.appspot.com");
        Uri file = Uri.fromFile(new File(fpath));
        StorageReference riversRef =
        storageRef.child(strPath+"/"+file.getLastPathSegment());
        UploadTask uploadTask = riversRef.putFile(file);
        uploadTask.addOnFailureListener(new OnFailureListener() {
           @Override
           public void onFailure(@NonNull Exception exception) {
          }
        }).addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {
           @Override
           public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
           final Uri downloadUrl = taskSnapshot.getDownloadUrl();

                UserdatabaseRef.child(FirebaseAuth.getInstance().getCurrentUser().getUid()).
                addListenerForSingleValueEvent(
                     new ValueEventListener() {
                        @Override
                        public void onDataChange(DataSnapshot dataSnapshot) {
                           if (FirebaseAuth.getInstance().getCurrentUser().getUid()== null) {
                              Log.e(TAG, "User " +
                                FirebaseAuth.getInstance().getCurrentUser().getUid() + " is
                                unexpectedly null");
                              Toast.makeText(UserFolderView.this,
                                   "Error: could not fetch user.",
                                   Toast.LENGTH_SHORT).show();
                           } else {
                       writeNewPost(FirebaseAuth.getInstance().getCurrentUser().getEmail(),
                                downloadUrl);
                              Uploadprogress.cancel();
                              GetFolderList();
                           }
                        }
                        @Override
```

```
                        public void onCancelled(DatabaseError databaseError) {
                            Log.w(TAG, "getUser:onCancelled", databaseError.toException());
                        }
                    });
            }
        });
    }
```

Users click on "Upload" icon to upload the files in the firebase storage. When the file is successfully uploaded, users can view the file by clicking on the filename as shown in figure 32. Users can upload files of type text (.txt), word (.doc), PDF (.pdf), image (.png, .jpg), spreadsheet (.xls), audio (.ogg, .mp3) and video (.mp4) as this app supports these files.

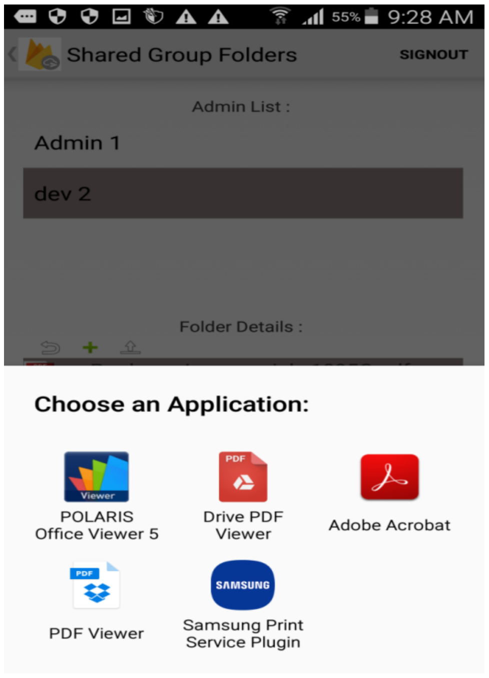


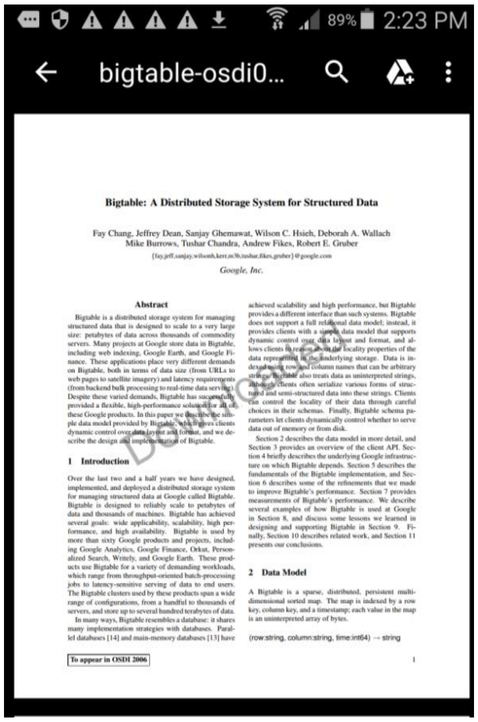
Figure 32: File View Screen

5.5 Download Properties

When uploaded files are downloaded by the app users, the files get locally saved in the device. It is stored under the user's username folder in the FirebaseFileSystem directory, which is created automatically in the device file manager when the app is installed. In order to differentiate the pdf files present in the cloud with the one downloaded, a watermark "Downloaded" is being stamped on all the pages of the file as shown in figure 33. The following code shows the implementation of stamping the watermark. For the below code to work, the library file itextpdf.jar file must be imported into the project libs section.

```
Public void manipulatePdf(String src, String dest) throws IOException,
DocumentException {
        PdfReader reader = new PdfReader(src);
        int n = reader.getNumberOfPages();
        PdfStamper stamper = new PdfStamper(reader, new FileOutputStream(dest));
        Font f = new Font(Font.FontFamily.HELVETICA, 60);
        Phrase p = new Phrase("Downloaded", f);
        Image img = Image.getInstance(1, 1, 3, 8, new byte[] { (byte)255, (byte)0, (byte)0
});
        float w = img.getScaledWidth();
        float h = img.getScaledHeight();
        PdfGState gs1 = new PdfGState();
        gs1.setFillOpacity(0.3f);
        PdfContentByte over;
        Rectangle pagesize;
        float x, y;
        for (int i = 1; i <= n; i++) {
            pagesize = reader.getPageSizeWithRotation(i);
            x = (pagesize.getLeft() + pagesize.getRight()) / 2;
            y = (pagesize.getTop() + pagesize.getBottom()) / 2;
            over = stamper.getOverContent(i);
            over.saveState();
            over.setGState(gs1);
            ColumnText.showTextAligned(over, Element.ALIGN_CENTER, p, x, y, 30);
            over.addImage(img, w, 0, 0, h, x - (w / 2), y - (h / 2));
            over.restoreState();
        }
        stamper.close();
        reader.close();
    }
}
```

Once a file is downloaded, a “Download” property is set for the file to notify users from downloading the same file again as shown in figure 34.



Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com
Google, Inc.

Abstract

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Bigtable, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving). Despite these varied demands, Bigtable has successfully provided a flexible, high-performance solution for all of these Google products. In this paper we describe the simple data model provided by Bigtable, which gives clients dynamic control over data layout and format, and we describe the design and implementation of Bigtable.

1 Introduction

Over the last two and a half years we have designed, implemented, and deployed a distributed storage system for managing structured data at Google called Bigtable. Bigtable is designed to reliably scale to petabytes of data and thousands of machines. Bigtable has achieved several goals: wide applicability, scalability, high performance, and high availability. Bigtable is used by more than sixty Google products and projects, including Google Analytics, Google Finance, Orkut, Personalized Search, Writely, and Google Earth. These products use Bigtable for a variety of demanding workloads, which range from throughput-oriented batch-processing jobs to latency-sensitive serving of data to end users. The Bigtable clusters used by these products span a wide range of configurations, from a handful to thousands of servers, and store up to several hundred terabytes of data.

In many ways, Bigtable resembles a database: it shares many implementation strategies with databases. Parallel databases [14] and main-memory databases [13] have achieved scalability and high performance, but Bigtable provides a different interface than such systems. Bigtable does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of the data represented in the underlying storage. Data is indexed using row and column names that can be arbitrary strings. Bigtable also treats data as uninterpreted strings, although clients often serialize various forms of structured and semi-structured data into these strings. Clients can control the locality of their data through careful choices in their schemas. Finally, Bigtable schema parameters let clients dynamically control whether to serve data out of memory or from disk.

Section 2 describes the data model in more detail, and Section 3 provides an overview of the client API. Section 4 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 5 describes the fundamentals of the Bigtable implementation, and Section 6 describes some of the refinements that we made to improve Bigtable's performance. Section 7 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 8, and discuss some lessons we learned in designing and supporting Bigtable in Section 9. Finally, Section 10 describes related work, and Section 11 presents our conclusions.

2 Data Model

A Bigtable is a sparse, distributed, persistent multi-dimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes.

(row:string, column:string, time:int64) → string

To appear in OSDI 2006

1

Figure 33: Downloaded PDF File

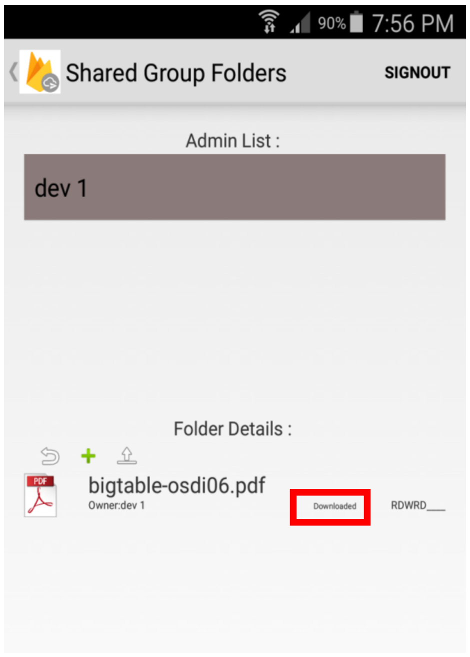


Figure 34: File Download Screen

CHAPTER 6

CONCLUSION

The File System is a useful app for the app users to store the file in a traditional hierarchical way. The app is being developed to work like the file system works in Linux. The app provides security features such as, group owners permitting users to access the folders, revoking permissions that were provided to the group members, file owners granting or revoking the access permissions to the group members and others, secure authentication into the app using Google account. All the features mentioned above are implemented in Java programming language using Android Studios.

Finally, developing an app using Firebase, which is a cloud tool introduced by Google was a good experience and I enjoyed learning it. Firebase has a variety of features for the developers to try out apart from Authentication, Storage and Real-time Database which is being used in this app. Test Lab, Notifications, Remote Config are few of them, which are well documented and could be easily implemented.

6.1 Future Work

The application is currently designed to integrate with the Google account. There are many other third party authentication services available which could be implemented. Adding to that, the app enables a user to be part of single group. This could be enhanced by making users to be part of multiple groups created by a single user.

REFERENCES

[1] Hierarchical File System, Image. [Online]. Available: https://www.google.com/imghp?hl=en&ei=xY7cWLjLIoPEmwHfvra4Dg&ved=0EKouCAIoAQ [Accessed: July 2016]

[2] Android Component Guide, Android Fundamentals. [Online]. Available: https://developer.android.com/guide/components/fundamentals.html [Accessed: September 2016]

[3] Android Developer Guide, Activity Life Cycle. [Online]. Available: https://developer.android.com/reference/android/app/Activity.html [Accessed: November 2016]

[4] Android Developer Guide, Life Cycle of Fragment. [Online]. Available: https://developer.android.com/guide/components/fragments.html [Accessed: November 2016]

[5] Android Studio Setup, Android versions. [Online]. Available: https://www.tutorialspoint.com/android/android_environment_setup.htm [Accessed : December 2016]

[6] Android Studio Setup, Project kickoff. [Online]. Available: https://www.tutorialspoint.com/android/android_studio.htm [Accessed: December 2016]

[7] Firebases Setup, Prerequisites. [Online]. Available: https://firebase.google.com/docs/android/setup [Accessed: February 2017]

[8] Firebase Analytics. Firebase Analytics Tutorial. [Online]. Available: https://firebase.google.com/docs/analytics/android/start/ [Accessed: January 2017]

[9] Firebase Authentication. Firebase Authentication Tutorial. [Online]. Available: https://firebase.google.com/docs/auth/android/google-signin [Accessed: January 2017]

[10] Firebase Real-time Database. Firebase Database Tutorial. [Online]. Available: https://firebase.google.com/docs/database [Accessed: January 2017]

[11] Firebase Storage. Firebase Storage Tutorial. [Online]. Available: https://firebase.google.com/docs/storage/android/start [Accessed: January 2017]

[12] Firebase Integration. Firebase Project Setup. [Online]. Available: https://firebase.google.com/docs/android/setup [Accessed: February 2017]

[13] Firebase Auth. Firebase Google Sign-In. [Online]. Available: https://firebase.google.com/docs/auth/android/google-signin [Accessed: February 2017]