**REAL TIME CHAT APPLICATION USING FIREBASE**

**A Project Report submitted in partial fulfillment of the requirements for the award of the degree of**

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**Submitted by**

## D.MANOJ(1210316518)

## D.DAVID(1210316517)

## M.SRUJAN(1210316529)

## K.DATTA(1210316564)

**Under the esteemed guidance of**

**E.NEELIMA**

**ASSISTANT PROFESSOR**



**DEPARTMENT OFCOMPUTER SCIENCE & ENGINEERING**

**GITAM**

**(Deemed to be University)**

**VISAKHAPATNAM**

**APRIL 2020**

# DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**GITAM INSTITUTE OF TECHNOLOGY**

**GITAM**

## (Deemed to be University)



# DECLARATION

IWe,hereby declare that the project report entitled “**REAL TIME CHAT APPLICATION USING FIREBASE**” is an original work done in the Department of Computer Science and Engineering, GITAM Institute of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering. The work has not been submitted to any other college or University for the award of any degree or diploma.

Date: 14/04/2020

## Registration No(s). Name(s) Signature(s)

**1210316518 D.MANOJ**

**1210316517 D.DAVID**

**1210316529 M.SRUJAN**

**1210316564 K.DATTA**

# DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**GITAM INSTITUTE OF TECHNOLOGY**

**GITAM**

## (Deemed to be University)



# CERTIFICATE

This is to certify that the project report entitled “**REAL TIME CHAT APPLICATION USING FIREBASE**” is a bonafide record of work carried out by **D.MANOJ(1210316518),D.DAVID(1210316517),M.SRUJAN(1210316529),K.DATTA(120316564)** students submitted in partial fulfillment of requirement for the award of degree of Bachelors of Technology in Computer Science and Engineering.

|  |  |
| --- | --- |
| **Project Guide** | **Head of the Department** |
|  |  |
| **E.Neelima** | **Dr. K. Thammi Reddy** |
| **Assistant professor** | **Professor** |

**TABLE OF CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| 1. | Abstract | | Page Number |
| 2. | Introduction | | Page Number |
| 3. | Literature Review | | Page Number |
| 4. | Problem Identification & Objectives | | Page Number |
| 5. | System Methodology | | Page Number |
| 6. | Overview of Technologies | | Page Number |
| 7. | Implementation | | Page Number |
|  | 8.1 | Coding | Page Number |
|  | 8.2 | Testing | Page Number |
| 8. | Results & Discussions | | Page Number |
| 9. | Conclusion & Future Scope | | Page Number |
| 10. | References | | Page Number |
|  |  | |  |

**ABSTRACT**

Communication through internet is becoming vital these days. An online communication

allows the users to communicate with other people in a fast and convenient way.

Considering this, the online communication application must be able share the texts or

images or any other files in a faster way with minimum delay or with no delay. Firebase

is one of the platforms which provides a real-time database and cloud services which

allows the developer to make these applications with ease. Instant messaging can be

considered as a platform to maintain communication. Android provides better platform to

develop various applications for instant messaging compared to other platforms such as

iOS. The main objective of this paper is to present a software application for the

launching of a real time communication between operators/users. The system developed

on android will enable the users to communicate with another users through text

messages with the help of internet. The system requires both the device to be connected

via internet. This application is based on Android with the backend provided by google

Firebase.

Keywords: communication; firebase; android; Instant

messaging; real-time databases; group messaging.

**INTRODUCTION**

In the real world the communication plays a very vital role. People have been

communicating with each other through various applications or mediums. In the

beginning people communicated with each other using letters or other sources,

as these mediums could take much time to deliver the content. Cell phones are another

medium of communication but the drawback is for any limited or small message which

need to be passed to another user then phone call is not an ideal way. The developers

then looked to implement a text-based communication which would allow an in

instant communication service. In 1984, the concept of SMS was developed in the

Franco German GSM cooperation by Friedhelm Hillebrand and Bernard Ghillebaert.

The limitation of SMS was the limited size i.e., 128 bytes [1] [2], after the rise of

smartphones from a decade many messaging applications have been developed. Some

are Bluetooth based and some were internet based such as WhatsApp [3],

WeChat [4] and others.

Android is an operating system for mobiles which was developed by google.

This operating system allows the applications to be used on mobiles. As it was

developed by google, android users can develop mobile applications and can be sold

through android application stores such as play store. Firebase is a NoSQL database

which make use of sockets which allows the users to store and retrieve the data from the

database. An Android version should be greater than 2.3, android studio 1.5 or higher

version, and android studio project are the prerequisites to connect the firebase to an

android application. Firebase provides a various kind of services such as:

Firebase Authentication : Firebase Authentication is useful to both developers and

the users. Developing and maintaining sign-in set-up may be a bit difficult and time

taking. Firebase provides an easy API for sign in. It also provides the data backup using

real time databases.

Firebase cloud: For storing the data such as video, text, pictures building the

infrastructure would be difficult and expensive for a new developer so the firebase

provides the platform of cloud storage .

Real time database: It is a cloud hosted NoSQL database. Apart from the authentication,

cloud service and real time databases firebase also provides a service for crash reporting

Crash Reporting: when some unexpected crashes occur in any applications it may be

difficult to conclude why the application crashed. Firebase provides crash reporting

service to deal with these crashes.

This project is concerned of a software application for the establishment of a real

time communication services between users. Chat application many-to-many type of

communication system where the users will able to exchange the messages among

themselves . User can create the chatroom according to the requirement or can also

join to the existing chatroom.

**GOALS**

The basic goal is to create a system which can create chatroom according to the users and store the related data at a single place.

To implement the real time chatting application which can allow the user to make

instant group messages. Implementing google firebase cloud and real time databases to

store the data.

**LITERATURE SURVEY**

Internet communication is getting more and more popular among the public. Apart

from using telephones or automobiles and sending mails, people can now communicate

with each other through the chat technology. The chat, is a kind of Internet technology

that supports human-to-human communication. IC, for instance, is one of the latest chat.

Over the past two years, with the advanced level of technology, there is an increasing

trend of using IC for communication. With IC, users can chat, send messages, files and

URL’s. As per Yahoo Chat is a free online chat room service which was provided

exclusively only for Yahoo! Users and was first launched on January 7, 1997. It was said

to be a feature on the very first release of Yahoo! Pager. The first public version was

released on March 9, 1998. Yahoo Chat allows its users to create public chat rooms, send

text messages and use emoticons. As per Skype is an instant messaging application. It

provides online text message and video chat services. Users may transmit both text and

video messages and allows exchange of digital documents. Skype allows video

conference calls. Chatting applications enables us to stay in connected anytime, anywhere

from any part of the world. There are many applications which provides chat services, the

most popular among them are WhatsApp, Facebook messenger, yahoo IM, emails etc. IC

applications have evolved from being just an another application to send messages. The

latest IC applications have features such as transfer of different types of files, video

chatting, group chat etc. The common disadvantage is that these applications restrict the

size of the files being transferred.

.

The most important features of Internet chat( IC) are

•Authentication and security

• Easy and quick communication

• Image transfer

• Group chat

• emojis

A close up of a logo

Description automatically generated

Figure1: Chatting

**PROJECT SPECIFICATION AND OBJECTIVES**

**PROJECT DESCRIPTION**

In this project, we will learn how to use the Firebase platform to create a chat app on

Android.

### ****TO DO****

* Allow users to sign in.
* Sync data using the Firebase Realtime Database.
* Store binary files in Firebase

**Functional Requirements**:

The primary requirement of IC application is to provide a mechanism for Online Chatting

at a small scale.

**Non-Functional Requirements**:

• Ease of use: The User Interface must be easy to navigate, understand and use.

• Availability: The application must be available to use as and when required.

**Hardware Requirements:**

• Operating System: Windows XP ,android

• Processor: Intel Core 2 Duo 1.8 GHz

• Ram:4 GB

• HDD:1.5GB

**Software Requirements:**

• Language: XML, Java

• Tools: Android studio

• If using a device, a connection cable.

**OBJECTIVES**

* In this application we are using Google firebase as the backend to

store the data of the application such as messages, pictures,

files and more.

* User has to register or sign-in through their

respective mail id and can use the services.

* .Using this application user can communicate with any user all over the world.
* When the user sign-in to the application, user can create or join group

where the communication is need to be done.

* This application is must be able to work on most of the android

devices.

* User can also sign-out from the present device

and can sign-in through another.

* This application must be able to stote messages offline and has the capability to send when it is online.

**IMPORTANT FEATURES**

• Easy and quick communication.

• Unlimited data transfer without the size restriction.

• Group chat

A screenshot of a cell phone

Description automatically generated

Figure2: Our objective is final app should resemble above design

.

**SYSTEM METHODOLOGIES**

**SYSTEM OVERVIEW**

The user is able to interact with UI of the app where he can sign in,sign out,able to

create new groups,join groups,chat text messages,use emojis,send image files.

The UI is simple so anyone using the app for the first time can easily understand how to

use it.The System also has an offline message handling capabilities.The system will be

able to sync messages so fast across devices when connection is back.

**PROJECT DELIVERABLES**

* Android app(APK file)
* Documentation
* Readme file

**HARDWARE INTERFACE**

* Android phone
* 128 mb minimum ram required
* Lan or internet connections
* Processor with speed of 500Mhz

**CONSTRAINTS AND LIMITATIONS**

The app needs internet connection to exchange messages and image files.The user can be

able to install this app only on android devices.The user must need to have an active

google or email account to use this app.The app doesnot have video and audio calling

services .We may add it in later version.

**SYSTEM ARCHITECTURE**

Chat system is peer-to-peer where the users exchange text messages and files. The users

of the system are the client and the server. The architecture is a distributed programming

which consists of two components, the server and the client. The client initiates the

communication by requesting for the server information and display the received chat

messages. The server conducts the chat session and manages all the client. The client and

the server have two type of communication between them. Firstly, control message where

one can join and leave chat session, create a chat room. Secondly, chat message where

one can send and receive messages, transfer image files.

**A close up of a computer

Description automatically generated**

Figure3: Client server Architecture

**METHODOLOGIES**

**Algorithms Used**

There are certain algorithms which are been used to develop the application, which

includes Authentication: Most of the application requires the identity of the user

which will help making the data of the user safer and more secured in a cloud. Firebase

provides backend, SDK and ready to use libraries which help the developer to

provide authentications effortlessly.

The algorithm lets the user to login into the application with a valid email id. The

algorithm first initializes the variable sign\_in to 1. That means true. The user then enters

the email id which is stored in another variable internally in the database. The email

id is then verified and the result is stored in a variable request\_code. If the value of the

request\_code matches with the value of the variable if both the values are same then it is

considered as the email is valid and the user sing in to the application. If the values do not

match then thesignin will not be done and the task ends.

**ALGORITHM1**

Declare and initialize sign\_variable of type private integer to 1

Function onActivityResult

If request code equals to sign\_in then

display signin successful

display chat messages

else

display signin unsuccessful

end task

**Send and receive messages:**

After a successful signin the user can now able to send and receive the messages.

The function onclick is a function defined and the variable of type EditText is declared

and initialized to the id of input text which is retrieved from the layout xml file. Email id

and username of the sender is received from the firebase database instance along with the

text which need to be sent. These two are converted to string and stored in the database

reference of the database root node. When this process is done then the input is set to

null and the user is allowed to send another message. Secondly, for the existing users if

the current user name from the firebase database is not equal to null then the

user will get the previous messages and new messages by calling the message display

function.

**ALGORITHM2**

Create function onclick

Initialize variable of type Edittext to input id

FirebaseDatabase.getInstance().getReference().push().setValue

(new classname(input.getText().toString(),

FirebaseAuth.getInstance().getcurrentuser().getEmail()));

Set the input to null

if (current username from database from database instance is not equal to null)

call function display

end function

A variable of type ListView is initialized to list id. Text and user variables of type

TextView are initialized to text and user id that means the sender text and name are

stored in these is stored into the variables and displayed at the time of the

variables. By using predefined function ‘model.get’ the data is stored into the variables and displayed at the time of the

function call.

**ALGORITHM3**

Create function display

Initialize variable list of type ListView to list id

Declare TextView variables text and user

Initialize text to text id

Initialize user to user id

Set text to model.gettext

Set user to model.getuser

Display list of messages

End function

variables. By using predefined function ‘model.get’ the data is stored into the variables and displayed at the time of the function call.

**Chat room**:

when the user wants to get the information about any topic then the user can search

for the room by giving the keyword. If user finds any relevant chat room then the user

can directly join the room. If there is no chat room with that name existing then user can

create the chat room and refer it to another users. After the users are joined in the chat

room, users can decide whether the messages which they send will be displayed along

with the sender name or not. If the user wants to be visible to other users then the

messages are displayed along with the username.

The following flow chat explains the chat room work flow.

A close up of a logo

Description automatically generated

**OVERVIEW OF TECHNOLOGIES**

**TOOLS AND TECHNOLOGY**

A comphrensive set of tools and technologies required during project development are

detailed.A topic oriented structure is given below.

**QUALITY PLANNING**

Software QFD

**SOFTWARE ANALYSIS,ARCHITECTURE AND DESIGN**

Brainstorming

Mind-Map

UML tools and techniques

Rational Rose ,Together,MS Visio,ArgoUML

**DATABASE MODELLING TOOLS**

NOSQL,Firebase database

**PROJECT MANAGEMENT**

MS Project,ScrumDesk

**EFFORT ESTIMATION**

Top-down,PERT

**RISK IDENTIFICATION,EVALUATION AND MINIMIZATION**

Software-FMEA

**SOFTWARE DEVELOPMENT METHODOLOGIES**

Agile,Scrum

**IMPLEMENTATION LANGUAGES**

Java,XML

**TECHNOLOGIES,LIBRARIES AND FRAMEWORKS**

**JAVA:** JEE/J2EE and other Java libraries for android

**MOBILE:**Android SDK,J2ME

**IDE:** Intellij IDE Android studio

**VERSION CONTROLLING:**Git/Github,MS visual source code

**TESTING**

Installation testing,functional testing,load testing,performance profiling,data integrity

testing,automated testing,etc.

A picture containing drawing, food

Description automatically generatedA picture containing drawing

Description automatically generated

**A picture containing drawing, shirt

Description automatically generatedA close up of a logo

Description automatically generated**

Figure4: Tools and technologies

**IMPLEMENTATION**

The next step is to implement design and features as we discussed using programming

languages.Here, mainly two languages used for development.One is XML for

userinterface ,other is Java for functionality.Firebase needs to be Configured inorder to

meet our requirements(no coding required,so development is faster at backend).

**i)CODING AND TESTING**

**ActivityMain.xml**

*<?***xml version="1.0" encoding="utf-8"***?>*<**LinearLayout  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 tools:context="com.google.firebase.udacity.friendlychat.MainActivity"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
  
 android:orientation="vertical"**>  
 <**LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:orientation="horizontal"**>  
 <**TextView  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="NEW GROUP"  
 android:layout\_weight="1"  
 android:textColor="@color/colorAccent"  
 android:fontFamily="sans-serif"** />  
 <**EditText  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="2"  
 android:hint="enter new group name"  
 android:id="@+id/ett"**/>  
 />  
 <**Button  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="create"  
 android:layout\_weight="1"  
 android:id="@+id/buttt"** />  
 </**LinearLayout**>  
 <**LinearLayout  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"** >  
 <**TextView  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:textAlignment="center"  
 android:layout\_marginLeft="25dp"  
 android:text="or"** />  
 </**LinearLayout**>  
<**LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:orientation="horizontal"**>  
 <**TextView  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="JOIN GROUP"  
 android:layout\_weight="1"  
 android:textColor="@color/colorAccent"  
 android:fontFamily="sans-serif"** />  
  
 <**EditText  
 android:id="@+id/et"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="2"  
 android:hint="enter group name"** />  
  
 <**Button  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Join"  
 android:layout\_weight="1"  
 android:id="@+id/butt"** />  
</**LinearLayout**>  
 <**LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:orientation="horizontal"**>  
 <**TextView  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="CURRENT GROUP"  
 android:layout\_weight="1"  
 android:textColor="@color/colorAccent"  
 android:fontFamily="sans-serif"  
 android:id="@+id/gn"** />  
  
 <**TextView  
 android:id="@+id/gname"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_weight="2"  
 android:visibility="invisible"** />  
  
 </**LinearLayout**>  
  
<**RelativeLayout  
  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:paddingBottom="@dimen/activity\_vertical\_margin"  
 android:paddingLeft="@dimen/activity\_horizontal\_margin"  
 android:paddingRight="@dimen/activity\_horizontal\_margin"  
 android:paddingTop="@dimen/activity\_vertical\_margin"** >  
  
 <**ListView  
 android:id="@+id/messageListView"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:layout\_above="@+id/linearLayout"  
 android:stackFromBottom="true"  
 android:divider="@android:color/transparent"  
 android:transcriptMode="alwaysScroll"  
 tools:listitem="@layout/item\_message"**/>  
  
 <**LinearLayout  
 android:id="@+id/linearLayout"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_alignParentBottom="true"  
 android:layout\_alignParentLeft="true"  
 android:layout\_alignParentStart="true"  
 android:orientation="horizontal"**>  
  
 <**ImageButton  
 android:id="@+id/photoPickerButton"  
 android:layout\_width="36dp"  
 android:layout\_height="36dp"  
 android:background="@android:drawable/ic\_menu\_gallery"** />  
  
 <**EditText  
 android:id="@+id/messageEditText"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_gravity="center\_vertical"  
 android:layout\_weight="1"  
 android:hint="create or join group to chat"**/>  
  
 <**Button  
 android:id="@+id/sendButton"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_gravity="bottom"  
 android:enabled="false"  
 android:text="@string/send\_button\_label"**/>  
  
 </**LinearLayout**>  
  
 <**ProgressBar  
 android:id="@+id/progressBar"  
 style="?android:attr/progressBarStyleLarge"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_centerHorizontal="true"  
 android:layout\_centerVertical="true"**/>  
</**RelativeLayout**>  
</**LinearLayout**>

## Create Firebase console project

## ****Add Firebase to the project****

1. Go to the [Firebase console](https://console.firebase.google.com/).
2. Select **Add project**.
3. Select or enter a Project name.
4. Follow the remaining setup steps in the Firebase console, then click Create project (or Add Firebase, if you're using an existing Google project).

## Add Firebase

1. From the overview screen of your new project, click the Android icon to launch the setup workflow.
2. Enter the codelab's package name: com.google.firebase.codelab.friendlychat
3. Enter the SHA1 of your signing keystore. If you are using the standard debug keystore, use the command below to find the SHA1 hash:

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

**Note**: Your debug keystore, usually called "debug.keystore", is typically located at

$HOME/.android/debug.keystore. If this file does not exist, Android Studio will create it

for you the first time you run any app.

### ****Add google-services.json file to your app****

After adding the package name and SHA1 and selecting Register**, Click Download**

**google-services.json** to obtain your Firebase Android config file then copy the google-

services.json file into the *app* directory in your project. After the file is downloaded we

can **Skip** the next steps shown.

### ****Add google-services plugin to your app****

The google-services plugin uses the google-services.json file to configure your

application to use Firebase. The following line should already be added to the end of the

build.gradle file in the *app* directory of your project (check to confirm):

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

### ****Sync your project with gradle files****

To be sure that all dependencies are available to your app, you should sync your project

with gradle files at this point.

Select **File > Sync Project with Gradle Files** from the Android Studio toolbar.

## Run the starter app

Now that we have imported the project into Android Studio and configured the google-

services plugin with your JSON file, we are ready to run the app for the first time. Connect your Android device, and click **Run** ()in the Android Studio toolbar.

The app should launch on your device. At this point, you should see an empty message

list, and sending and receiving messages will not work. In the next section, we

authenticate users so they can use Friendly Chat.

## Enable Authentication

Let's require a user to sign in before reading or posting any Friendly Chat messages.

### ****Firebase Realtime Database Rules****

Access to your Firebase Database is configured by a set of rules written in a JSON

configuration language.

Go to your project in the Firebase console and select **Database**. Select the **Realtime**

**Database** option (not Cloud Firestore). If prompted for security rules, with choices to

start in either **test mode** or **locked mode**, choose **locked mode**. Once the default rules are

established, select the **Rules** tab and update the rules configuration with the following:

{

"rules": {

".read": "auth != null",

".write": "auth != null"

}

}

Click "Publish" to publish the new rules.

### ****Configure Authentication APIs****

Before your application can access the Firebase Authentication APIs on behalf of our

users, we will have to enable it

1. Navigate to the [Firebase console](http://console.firebase.google.com/) and select your project
2. Select **Authentication**
3. Select the **Sign In Method** tab
4. Toggle the **Google** switch to enabled (blue)
5. Set a support email.
6. Press **Save** on the resulting dialog

### ****Add Firebase Auth dependency****

The firebase-auth SDK allows easy management of authenticated users of your

application. Confirm the existence of this dependency in your app/build.gradle file.

### app/build.gradle

implementation 'com.google.firebase:firebase-auth'

Add the Auth instance variables in the MainActivity class under the // Firebase instance

variables comment:

### MainActivity.java (instance variable)

*// Firebase instance variables*

private FirebaseAuth mFirebaseAuth;

private FirebaseUser mFirebaseUser;

### ****Check for current user****

Now let's modify MainActivity.java to send the user to the sign-in screen whenever they

open the app and are unauthenticated.

Add the following to the onCreate method **after**mUsername has been initialized:

**MainActivity.java**

*// Initialize Firebase Auth*

mFirebaseAuth = FirebaseAuth.getInstance();

mFirebaseUser = mFirebaseAuth.getCurrentUser();

if (mFirebaseUser == null) {

*// Not signed in, launch the Sign In activity*

startActivity(new Intent(this, SignInActivity.class));

finish();

return;

} else {

mUsername = mFirebaseUser.getDisplayName();

if (mFirebaseUser.getPhotoUrl() != null) {

mPhotoUrl = mFirebaseUser.getPhotoUrl().toString();

}

}

Then add a new case to onOptionsItemSelected() to handle the sign out button:

**MainActivity.java**

@Override

public boolean onOptionsItemSelected(MenuItem item) {

switch (item.getItemId()) {

case R.id.sign\_out\_menu:

mFirebaseAuth.signOut();

Auth.GoogleSignInApi.signOut(mGoogleApiClient);

mUsername = ANONYMOUS;

startActivity(new Intent(this, SignInActivity.class));

finish();

return true;

default:

return super.onOptionsItemSelected(item);

}

}

Now we have all of the logic in place to send the user to the sign-in screen when

necessary. Next we need to implement the sign-in screen to properly authenticate users.

### ****Implement the Sign-In screen****

Open the file SignInActivity.java. Here a simple Sign-In button is used to initiate

authentication. In this step you will implement the logic to Sign-In with Google, and then

use that Google account to authenticate with Firebase.

Add an Auth instance variable in the SignInActivity class under the // Firebase instance

variables comment:

**SignInActivity.java**

*// Firebase instance variables*

private FirebaseAuth mFirebaseAuth;

Then, edit the onCreate() method to initialize Firebase in the same way you did

in MainActivity:

**SignInActivity.java**

*// Initialize FirebaseAuth*

mFirebaseAuth = FirebaseAuth.getInstance();

Next, initiate signing in with Google. Update SignInActivity's onClick method to look

like this:

**SignInActivity.java**

@Override

public void onClick(View v) {

switch (v.getId()) {

case R.id.sign\_in\_button:

signIn();

break;

}

}

Add the required signIn method that actually presents the user with the Google Sign-In

UI.

**SignInActivity.java**

private void signIn() {

Intent signInIntent = Auth.GoogleSignInApi.getSignInIntent(mGoogleApiClient);

startActivityForResult(signInIntent, RC\_SIGN\_IN);

}

Next, add the onActivityResult method to SignInActivity to handle the sign in result. If

the result of the Google Sign-In was successful, use the account to authenticate with

Firebase.

**SignInActivity.java**

@Override

public void onActivityResult(int requestCode, int resultCode, Intent data) {

super.onActivityResult(requestCode, resultCode, data);

*// Result returned from launching the Intent from GoogleSignInApi.getSignInIntent(...);*

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);

} else {

*// Google Sign-In failed*

Log.e(TAG, "Google Sign-In failed.");

}

}

}

Add the required firebaseAuthWithGoogle method to authenticate with the signed in

Google account:

**SignInActivity.java**

private void firebaseAuthWithGoogle(GoogleSignInAccount acct) {

Log.d(TAG, "firebaseAuthWithGooogle:" + acct.getId());

AuthCredential credential = GoogleAuthProvider.getCredential(acct.getIdToken(), null);

mFirebaseAuth.signInWithCredential(credential)

.addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {

@Override

public void onComplete(@NonNull Task<AuthResult> task) {

Log.d(TAG, "signInWithCredential:onComplete:" + task.isSuccessful());

*// If sign in fails, display a message to the user. If sign in succeeds*

*// the auth state listener will be notified and logic to handle the*

*// signed in user can be handled in the listener.*

if (!task.isSuccessful()) {

Log.w(TAG, "signInWithCredential", task.getException());

Toast.makeText(SignInActivity.this, "Authentication failed.",

Toast.LENGTH\_SHORT).show();

} else {

startActivity(new Intent(SignInActivity.this, MainActivity.class));

finish();

}

}

});

}

That's it! We have implemented authentication using Google as an Identity Provider in

just a few method calls and without needing to manage any server-side configuration.

### ****Test your work****

Run the app on your device. we should be immediately sent to the sign-in screen. Tap the

Google Sign-In button. we should then be sent to the messaging screen if everything

worked well.

## Read Messages

### ****Import Messages****

1. In our project in Firebase console, select **Database** on the left navigation menu.
2. Select Realtime Database option.

**Note**: If you are presented with database creation option, please select Realtime database

create option and apply the rules as done in previous step.

1. In the overflow menu of the Data tab, select **Import JSON**.
2. Browse to the initial\_messages.json file in the root of the cloned repository, and select it.
3. Click **Import**.

**Note**: This replaces any data currently in your database.

After importing the JSON file, your database should look like this:

root

messages

-K2ib4H77rj0LYewF7dP

text: "hello"

name: "anonymous"

-K2ib5JHRbbL0NrztUfO

text: "how are you"

name: "anonymous"

-K2ib62mjHh34CAUbide

text: "i am fine"

name: "anonymous"

### Add Firebase Realtime Database and Firebase Storage dependencies

In the dependencies block of the app/build.gradle file, the following dependencies should

be included. For this codelab, they are already added for convenience; confirm this by

looking in the app/build.gradle file:

### Dependency in app/build.gradle

implementation 'com.google.firebase:firebase-database'

implementation 'com.google.firebase:firebase-storage'

### Synchronize messages

In this section we add code that synchronizes newly added messages to the app UI by:

* Initializing the Firebase Realtime Database and adding a listener to handle

changes made to the data.

* Updating the RecyclerView adapter so new messages will be shown.
* Adding the Database instance variables with your other Firebase instance

variables in the MainActivity class:

### MainActivity.java

*// Firebase instance variables*

...

private DatabaseReference mFirebaseDatabaseReference;

private FirebaseRecyclerAdapter<FriendlyMessage, MessageViewHolder>

mFirebaseAdapter;

Modify your MainActivity's onCreate method by

replacing *mProgressBar.setVisibility(ProgressBar.INVISIBLE);* with the code defined

below. This code initially adds all existing messages and then listens for new child entries

under the messages path in your Firebase Realtime Database. It adds a new element to the

UI for each message:

### MainActivity.java

*// New child entries*

mFirebaseDatabaseReference = FirebaseDatabase.getInstance().getReference();

SnapshotParser<FriendlyMessage> parser = new SnapshotParser<FriendlyMessage>() {

@Override

public FriendlyMessage parseSnapshot(DataSnapshot dataSnapshot) {

FriendlyMessage friendlyMessage = dataSnapshot.getValue(FriendlyMessage.class);

if (friendlyMessage != null) {

friendlyMessage.setId(dataSnapshot.getKey());

}

return friendlyMessage;

}

};

DatabaseReference messagesRef = mFirebaseDatabaseReference.child(MESSAGES\_CHILD);

FirebaseRecyclerOptions<FriendlyMessage> options =

new FirebaseRecyclerOptions.Builder<FriendlyMessage>()

.setQuery(messagesRef, parser)

.build();

mFirebaseAdapter = new FirebaseRecyclerAdapter<FriendlyMessage, MessageViewHolder>(options) {

@Override

public MessageViewHolder onCreateViewHolder(ViewGroup viewGroup, int i) {

LayoutInflater inflater = LayoutInflater.from(viewGroup.getContext());

return new MessageViewHolder(inflater.inflate(R.layout.item\_message, viewGroup, false));

}

@Override

protected void onBindViewHolder(final MessageViewHolder viewHolder,

int position,

FriendlyMessage friendlyMessage) {

mProgressBar.setVisibility(ProgressBar.INVISIBLE);

if (friendlyMessage.getText() != null) {

viewHolder.messageTextView.setText(friendlyMessage.getText());

viewHolder.messageTextView.setVisibility(TextView.VISIBLE);

viewHolder.messageImageView.setVisibility(ImageView.GONE);

} else if (friendlyMessage.getImageUrl() != null) {

String imageUrl = friendlyMessage.getImageUrl();

if (imageUrl.startsWith("gs://")) {

StorageReference storageReference = FirebaseStorage.getInstance()

.getReferenceFromUrl(imageUrl);

storageReference.getDownloadUrl().addOnCompleteListener(

new OnCompleteListener<Uri>() {

@Override

public void onComplete(@NonNull Task<Uri> task) {

if (task.isSuccessful()) {

String downloadUrl = task.getResult().toString();

Glide.with(viewHolder.messageImageView.getContext())

.load(downloadUrl)

.into(viewHolder.messageImageView);

} else {

Log.w(TAG, "Getting download url was not successful.",

task.getException());

}

}

});

} else {

Glide.with(viewHolder.messageImageView.getContext())

.load(friendlyMessage.getImageUrl())

.into(viewHolder.messageImageView);

}

viewHolder.messageImageView.setVisibility(ImageView.VISIBLE);

viewHolder.messageTextView.setVisibility(TextView.GONE);

}

viewHolder.messengerTextView.setText(friendlyMessage.getName());

if (friendlyMessage.getPhotoUrl() == null) {

viewHolder.messengerImageView.setImageDrawable(ContextCompat.getDrawable(MainActivity.this,

R.drawable.ic\_account\_circle\_black\_36dp));

} else {

Glide.with(MainActivity.this)

.load(friendlyMessage.getPhotoUrl())

.into(viewHolder.messengerImageView);

}

}

};

mFirebaseAdapter.registerAdapterDataObserver(new RecyclerView.AdapterDataObserver() {

@Override

public void onItemRangeInserted(int positionStart, int itemCount) {

super.onItemRangeInserted(positionStart, itemCount);

int friendlyMessageCount = mFirebaseAdapter.getItemCount();

int lastVisiblePosition =

mLinearLayoutManager.findLastCompletelyVisibleItemPosition();

*// If the recycler view is initially being loaded or the*

*// user is at the bottom of the list, scroll to the bottom*

*// of the list to show the newly added message.*

if (lastVisiblePosition == -1 ||

(positionStart >= (friendlyMessageCount - 1) &&

lastVisiblePosition == (positionStart - 1))) {

mMessageRecyclerView.scrollToPosition(positionStart);

}

}

});

mMessageRecyclerView.setAdapter(mFirebaseAdapter);

Appropriately start and stop listening for updates from Firebase Realtime Database.

Update the *onPause* and *onResume* methods in MainActivity as shown below.

### MainActivity.java

@Override

public void onPause() {

mFirebaseAdapter.stopListening();

super.onPause();

}

@Override

public void onResume() {

super.onResume();

mFirebaseAdapter.startListening();

}

### Test message sync

1. Click **Run**().
2. Add new messages directly in the Database section of the Firebase console.

Confirm that they show up in the Friendly-Chat UI.

* Navigate to the Database section of the Firebase console. From the Data tab,

select the '+' sign on the messages element.

* Give the new element a name of -ABCD (note the '-' sign) and leave the value

empty for now.

* Select the '+' sign on the -ABCD element
* Give the new element a name of "name" and value of "Mary"
* Select the '+' sign on the -ABCD element again
* Give the new element a name of "text" and value of "hello"
* Select Add

We just added a realtime database to your app!

## Send Messages

### ****Implement text message sending****

In this section, we will add the ability for app users to send text messages. The code

snippet below listens for click events on the send button, creates a

new FriendlyMessage object with the contents of the message field, and pushes the

message to the database. The push() method adds an automatically generated ID to the

pushed object's path. These IDs are sequential which ensures that the new messages will

be added to the end of the list.

Update the onClick method of mSendButton in the onCreate method in

the MainActivity class. This code is at the bottom of the onCreate method already.

Update the onClick body to match the code below:

### MainActivity.java

mSendButton = (Button) findViewById(R.id.sendButton);

mSendButton.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View view) {

FriendlyMessage friendlyMessage = new

FriendlyMessage(mMessageEditText.getText().toString(),

mUsername,

mPhotoUrl,

null */\* no image \*/*);

mFirebaseDatabaseReference.child(MESSAGES\_CHILD)

.push().setValue(friendlyMessage);

mMessageEditText.setText("");

}

});

### ****Implement image message sending****

In this section, we will add the ability for app users to send image messages. Creating an

image message is done with these steps:

* Select image
* Handle image selection
* Write temporary image message to the RTDB (Realtime Database)
* Begin to upload selected image
* Update image message URL to that of the uploaded image, once upload is complete

### ****SELECT IMAGE****

To add images this codelab uses Cloud Storage for Firebase. Cloud Storage is a good

place to store the binary data of our app.

In the Firebase console select **Storage** in the left navigation panel. Then click **Get**

**Started** to enable Cloud Storage for our project. Continue following the steps in the

prompt, using the suggested defaults.

With the following code snippet you will allow the user to select an image from the

device's local storage. Update the onClick method of mAddMessageImageView in

the onCreate method in the MainActivity class. This code is at the bottom of

the onCreate method already. Update the onClick body to match the code below:

### MainActivity.java

mAddMessageImageView = (ImageView) findViewById(R.id.addMessageImageView);

mAddMessageImageView.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View view) {

Intent intent = new Intent(Intent.ACTION\_OPEN\_DOCUMENT);

intent.addCategory(Intent.CATEGORY\_OPENABLE);

intent.setType("image/\*");

startActivityForResult(intent, REQUEST\_IMAGE);

}

});

#### **Handle image selection and write temp message**

Once the user has selected an image, a call to the MainActivity's onActivityResult will be

fired. This is where we handle the user's image selection. Using the code snippet below,

add the onActivityResult method to MainActivity. In this function we will write a

message with a temporary image url to the database indicating the image is being

uploaded.

### MainActivity.java

@Override

protected void onActivityResult(int requestCode, int resultCode, Intent data) {

super.onActivityResult(requestCode, resultCode, data);

Log.d(TAG, "onActivityResult: requestCode=" + requestCode + ", resultCode=" + resultCode);

if (requestCode == REQUEST\_IMAGE) {

if (resultCode == RESULT\_OK) {

if (data != null) {

final Uri uri = data.getData();

Log.d(TAG, "Uri: " + uri.toString());

FriendlyMessage tempMessage = new FriendlyMessage(null, mUsername, mPhotoUrl,

LOADING\_IMAGE\_URL);

mFirebaseDatabaseReference.child(MESSAGES\_CHILD).push()

.setValue(tempMessage, new DatabaseReference.CompletionListener() {

@Override

public void onComplete(DatabaseError databaseError,

DatabaseReference databaseReference) {

if (databaseError == null) {

String key = databaseReference.getKey();

StorageReference storageReference =

FirebaseStorage.getInstance()

.getReference(mFirebaseUser.getUid())

.child(key)

.child(uri.getLastPathSegment());

putImageInStorage(storageReference, uri, key);

} else {

Log.w(TAG, "Unable to write message to database.",

databaseError.toException());

}

}

});

}

}

}

}

#### **Upload image and update message**

Add the method putImageInStorage to MainActivity. It is called in onActivityResult to

initiate the upload of the selected image. Once the upload is complete you will update the

message to use the appropriate image.

### MainActivity.java

private void putImageInStorage(StorageReference storageReference, Uri uri, final String key) {

storageReference.putFile(uri).addOnCompleteListener(MainActivity.this,

new OnCompleteListener<UploadTask.TaskSnapshot>() {

@Override

public void onComplete(@NonNull Task<UploadTask.TaskSnapshot> task) {

if (task.isSuccessful()) {

task.getResult().getMetadata().getReference().getDownloadUrl()

.addOnCompleteListener(MainActivity.this,

new OnCompleteListener<Uri>() {

@Override

public void onComplete(@NonNull Task<Uri> task) {

if (task.isSuccessful()) {

FriendlyMessage friendlyMessage =

new FriendlyMessage(null, mUsername, mPhotoUrl,

task.getResult().toString());

mFirebaseDatabaseReference.child(MESSAGES\_CHILD).child(key)

.setValue(friendlyMessage);

}

}

});

} else {

Log.w(TAG, "Image upload task was not successful.",

task.getException());

}

}

});

}

### ****Test Sending Messages****

1. Click the **Run** button.
2. Enter a message and hit the send button, the new message should be visible in the

app UI and in the Firebase console.

1. Tap the "+" image to select an image from your device. The new message should

be visible first with a placeholder image, and then with the selected image once

the image upload is complete. The new message should also be visible in the

Firebase console, as an object in the Database and as a blob in Storage.

We have used Firebase to easily build a real-time chat application.

### ****We've covered****

* Firebase Authentication
* Firebase Realtime Database
* Cloud Storage for Firebase

**RESULTS AND DISCUSSIONS**

**RESULT**

The final system will result as a real time communication application which provides the

users to communicate to each other with an ease. The application will have a login page

through which the user can register and login themselves. Home page of the

application contains the previous messages if any. The user can be able to search for the

other user. User can send and receive text messages. The user can create chat rooms and

can search for the content or information. With these chat rooms users

can exchange views and information about various topics.

**APPLICATION EXECUTION RESULTS OF ABOVE CODE**

|  |  |
| --- | --- |
| A screenshot of a cell phone  Description automatically generated | A screenshot of a cell phone  Description automatically generated |
| Figure5: Login screen of app | Figure6: Login screen of app |

|  |  |
| --- | --- |
| A screenshot of a cell phone  Description automatically generated | A screenshot of a cell phone  Description automatically generated |
| Figure7: New Group Creation in app | Figure8: New Group Creation in app |

|  |  |
| --- | --- |
| A screenshot of a cell phone  Description automatically generated |  |
| Figure7: Existing Group join in app | Figure8: Existing Group join in app |
| A screen shot of a computer  Description automatically generated | |
| Figure9: Firebase real time server database(NOSQL) | |
| A screenshot of a computer screen  Description automatically generated  Figure10: Firebase security rules | |
|  | |
| A screen shot of a computer monitor sitting on top of a computer  Description automatically generated | |
| Figure11: Firebase authentication console  **CONCLUSION AND FUTURE SCOPE**  There is always some place for enhancements in any  software application, however good and efficient the  application may be.  Right now, we are dealing with only the instant messaging  between the peers. In future the application may further  developed to include some features such as  1. Voice messaging.  2. Group calling  3. Live streaming  4. Messages auto delete after a given time.  5. Personalized message tunes.  6.Video calling  IC(Internet Chat) can by further implemented by providing security algorithms to encrypt the  data transfer.  IC has features such as text messaging, group chat, data transfer. The main objective of IC was  to develop an application which provides the data transfers which has been implemented.  Firebase provides with unlimited support to add more features easily using remote  config,helps us to track users with analytics,send online notifications to right people at right  time.It helps us to grow more audience and also helps us to earn revenue through admobs.  Firebase helps us to concentrate on our app features rather than worrying about  server configuration and infrastructure.It makes application development faster.We  can earn more functionality with less code.So, firebase gives us opportunity to add  more features in future. | |

**REFERENCES**

1. Anon., 2015. Development of a Health Care Assistant App for the

Seniors. International Journal of Applied Science and

Engineering, pp. 3-5.

2. Jianye Liu; Jiankun Yu, Research on Development of Android

Applications, 4th International Conference on Intelligent

Networks and Intelligent Systems, 15 December 2011

3. Abhinav Kathuria et al, Challenges in Android Application

Development: A Case Study, Vol.4 Issue.5, May- 2015, pg.

294-299

4. Li Ma et al, Research and Development of Mobile Application

for Android Platform, International Journal of Multimedia and

Ubiquitous Engineering 9(4):187-198 • April 2014

5. Nikhil M. Dongre, Nikhil M. Dongre, Journal of Computer

Engineering (IOSR-JCE), Volume 19, Issue 2, Ver. I (Mar.-Apr.

2017), PP 65-77

6. Javed Ahmad Shaheen et al, Android OS with its Architecture and

Android Application with Dalvik Virtual Machine Review,

International Journal of Multimedia and Ubiquitous Engineering

Vol. 12, No. 7 (2017), pp. 19-30

7. Sajid Nabi Khan, Ikhlaq Ul Firdous, Review on Android App

Security, International Journal of Advanced Research in

Computer Science and Software Engineering, Volume 7, Issue 4,

April 2017

8. Lazarela Lazareska, Kire Jakimoski et al, Analysis of the

Advantages and Disadvantages of Android and iOS Systems and

Converting Applications from Android to iOS Platform and Vice

Versa, American Journal of Software Engineering and

Applications 2017; 6(5): 116-120

9. Bin Peng et al, The Android Application Development College

Challenge, 2012 IEEE 14th International Conference on High

Performance Computing and Communication & 2012 IEEE 9th

International Conference on Embedded Software and Systems,

18 October 2012

10. Shao Guo-Hong, Application Development Research Based on

Android Platform,2014 7th International Conference on

Intelligent Computation Technology and Automation, 08 January

2015

[1] A study of internet instant messaging and chat protocols published on 14 August 2006 by R.B.Jennings, E.M Nahum, D.P Olshefski, D Saha, Zon-yin Shae, Christopher J. Waters(http://ieeexplore.ieee.org/document/1668399/) [2] https://en.wikipedia.org/wiki/Instant\_messaging [3] http://www.w3schools.com