CORPCHAT MESSAGING APPLICATION

INTRODUCTION

Communication defines not only the quality of human existence and relationships, but also the platform upon which productivity and purpose is being achieved through collaboration and community building. Very few things are as timeless and indispensable as the human need to communicate. The art, culture and need stem from the undated medieval times and reaches not just the 21st century but also the future as man is unlikely to outgrow the need for communication.

I chose to build an instant messaging app to reflect my understanding and appreciation of the history of communication technology from the first instant messaging app, "Talkomatic Program" built on the PLATO System in 1973, to the Unix-based command-line interface "talk" of the early 1990s to Pidgin, Messenger Live and AOL Instant Messaging web apps and more recently, the widely popular the WhatsApp and many other enterprise-variants and implementations of this messaging communications technologies.

While the Technology—both hardware and software—have developed immensely over the last decade, evolving cultures and need-sets have shown that messaging apps are here to stay. Diverse kinds of niches exist that require its own unique implementation of the functionalities of the messaging technology in a way that is tailor-made and specifically engineered with a set of needs in mind.

The goal of this project is to plan, design and implement an android instant messaging application.

So far, the messaging functionality proposed at the analysis and design phase was achieved with extensibility and infrastructure improvement opportunities available going forward. Going into the future, I will have to add more functionality like video call, two-layer authentication feature and upgraded usability UI that facilitates the navigation of this app.

DESIGN

PROBLEM STATEMENT

Corpchat is a consumer chat app that aims to improve and provide alternative communication app to students by providing additional functionality relevant to student experience.

PLANNED FEATURES

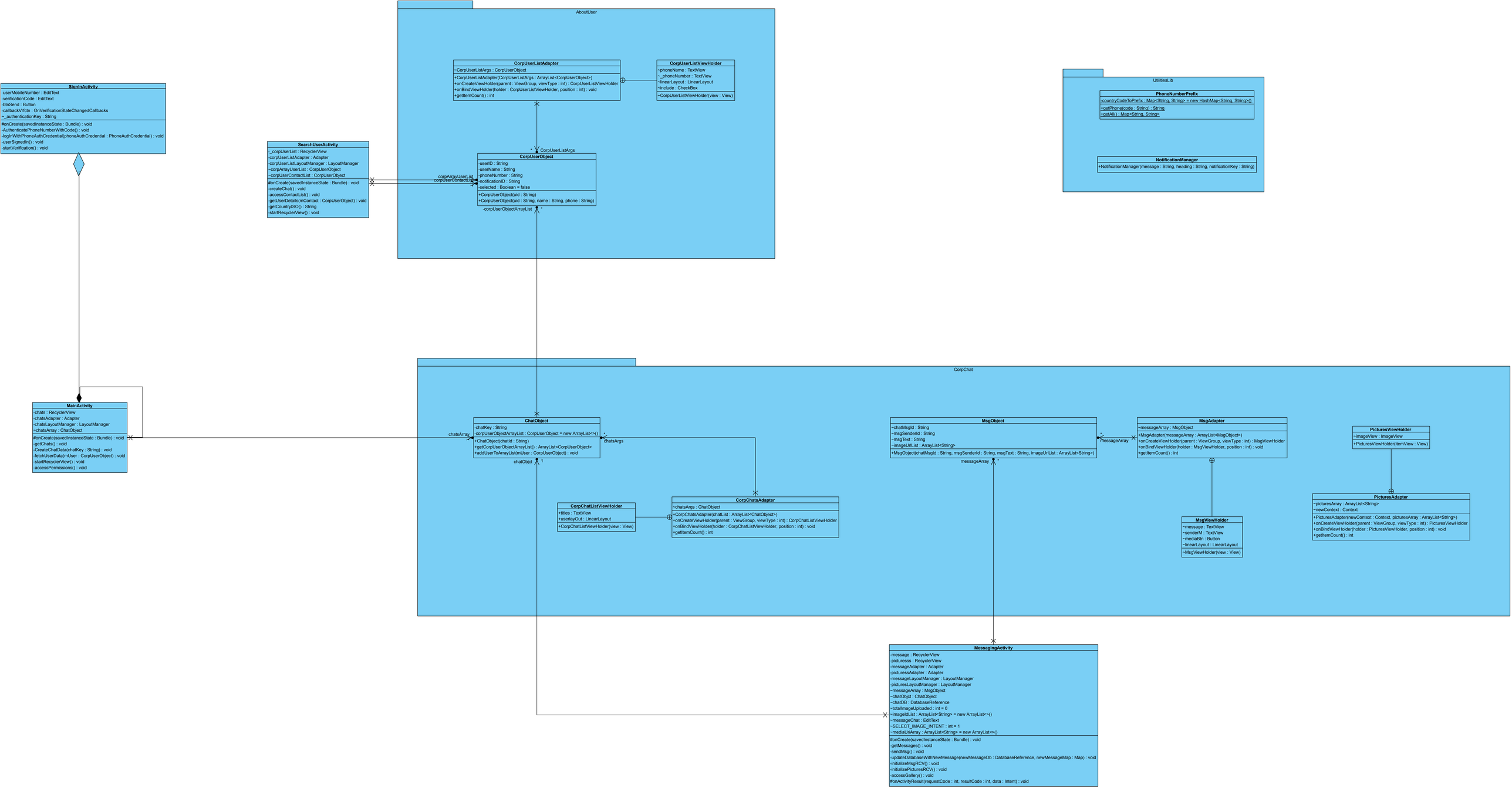
The following features were taken into strategic consideration in view of the problem domain, functional and non-functional requirements:

1. Instant Messaging functionality with a capability to send texts, pictures and create private & group chats real time
2. Display and Search already registered CorpChat contacts on user's phone
3. Implement Firebase Database and Data Storage for Realtime synchronization of chat messages and media
4. Design a suitable Data structure for the storage and retrieval of user data
5. Write Security rules that disallows non-authenticated users from reading and writing data
6. Implement a Push Notification Service by using the One Signal Push Notification Service for Real time dropdown notification using the One Signal Server.
7. Design a robust and aesthetic Login Functionality and Form that matches cutting-edge design pattern
8. Devise a very safe and secure means of handling user authentication for the messaging App.

EXPLAINING MY DESIGN CHOICE

As diligently illustrated in the proceeding Class Diagram, I choose the class diagram over an activity or flow chart diagram owing to the following advantages:

A. Class Diagram Visual Model of System Design and Analysis lends a very strong organization to code construction processes in that it furnishes with the requisite understanding of the Classes, Attributes and methods necessary to back-end implementation. As you would notice from the Class Diagram, I have arranged my project into Packages containing different classes, using the UML notation to assign relationships.
B. I can connect the dots conceptually before implementing the system functionality programmatically.
C. My design using the class diagram also helped me greatly in emphasizing scalability and extensibility design practice into this course work
D. Overall, Class Diagram is a very key element of object-oriented software development, this facilitated my efforts to incorporating key concepts of polymorphism, encapsulation and inheritance hierarchy design—all of which Java Programming Language clearly supports.


AboutUser
CorpUserListAdapter
~CorpUserListArgs : CorpUserObject
+CorpUserListAdapter(CorpUserListArgs : ArrayList<CorpUserObject>)
+onCreateViewHolder(parent : ViewGroup, viewType : int) : CorpUserListViewHolder
+onBindViewHolder(holder : CorpUserListViewHolder, position : int) : void
+getItemCount() : int
CorpUserListViewHolder
~phoneName : TextView
~_phoneNumber : TextView
~linearLayout : LinearLayout
~include : CheckBox
~CorpUserListViewHolder(view : View)
CorpUserListArgs
CorpUserObject
-userID : String
-userName : String
-phoneNumber : String
-notificationID : String
-selected : Boolean = false
+CorpUserObject(uid : String)
+CorpUserObject(uid : String, name : String, phone : String)
-corpUserObjectArrayList
SignInActivity
-userMobileNumber : EditText
-verificationCode : EditText
-btnSend : Button
-callbackVrfctn : OnVerificationStateChangedCallbacks
~_authenticationKey : String
#onCreate(savedInstanceState : Bundle) : void
-AuthenticatePhoneNumberWithCode() : void
-loginWithPhoneAuthCredential(phoneAuthCredential : PhoneAuthCredential) : void
-userSignedIn() : void
-startVerification() : void
SearchUserActivity
-_corpUserList : RecyclerView
-corpUserListAdapter : Adapter
-corpUserListLayoutManager : LayoutManager
~corpArrayUserList : CorpUserObject
~corpUserContactList : CorpUserObject
#onCreate(savedInstanceState : Bundle) : void
-createChat() : void
-accessContactList() : void
-getUserDetails(mContact : CorpUserObject) : void
-getCountryISO() : String
-startRecyclerView() : void
corpArrayUserList
corpUserContactList
UtilitiesLib
PhoneNumberPrefix
-countryCodeToPrefix : Map<String, String> = new HashMap<String, String>()
+getPhone(code : String) : String
+getAll() : Map<String, String>
NotificationManager
+NotificationManager(message : String, heading : String, notificationKey : String)
CorpChat
MainActivity
-chats : RecyclerView
-chatsAdapter : Adapter
-chatsLayoutManager : LayoutManager
~chatsArray : ChatObject
#onCreate(savedInstanceState : Bundle) : void
-getChats() : void
-CreateChatData(chatKey : String) : void
-fetchUserData(mUser : CorpUserObject) : void
-startRecyclerView() : void
-accessPermissions() : void
ChatObject
-chatKey : String
-corpUserObjectArrayList : CorpUserObject = new ArrayList<>()
+ChatObject(chatId : String)
+getCorpUserObjectArrayList() : ArrayList<CorpUserObject>
+addUserToArrayList(mUser : CorpUserObject) : void
chatsArray
chatsArgs
chatObjct
CorpChatListViewHolder
+titles : TextView
+userlayOut : LinearLayout
+CorpChatListViewHolder(view : View)
CorpChatsAdapter
~chatsArgs : ChatObject
+CorpChatsAdapter(chatList : ArrayList<ChatObject>)
+onCreateViewHolder(parent : ViewGroup, viewType : int) : CorpChatListViewHolder
+onBindViewHolder(holder : CorpChatListViewHolder, position : int) : void
+getItemCount() : int
MsgObject
~chatMsgId : String
~msgSenderId : String
~msgText : String
~imageUrlList : ArrayList<String>
+MsgObject(chatMsgId : String, msgSenderId : String, msgText : String, imageUrlList : ArrayList<String>)
messageArray
MsgAdapter
~messageArray : MsgObject
+MsgAdapter(messageArray : ArrayList<MsgObject>)
+onCreateViewHolder(parent : ViewGroup, viewType : int) : MsgViewHolder
+onBindViewHolder(holder : MsgViewHolder, position : int) : void
+getItemCount() : int
thessageArray
MsgViewHolder
~message : TextView
~senderM : TextView
~mediaBtn : Button
~linearLayout : LinearLayout
~MsgViewHolder(view : View)
PicturesViewHolder
~imageView : ImageView
+PicturesViewHolder(itemView : View)
PicturesAdapter
~picturesArray : ArrayList<String>
~newContext : Context
+PicturesAdapter(newContext : Context, picturesArray : ArrayList<String>)
+onCreateViewHolder(parent : ViewGroup, viewType : int) : PicturesViewHolder
+onBindViewHolder(holder : PicturesViewHolder, position : int) : void
+getItemCount() : int
MessagingActivity
-message : RecyclerView
-picturesss : RecyclerView
-messageAdapter : Adapter
-picturesAdapter : Adapter
-messageLayoutManager : LayoutManager
-picturesLayoutManager : LayoutManager
~messageArray : MsgObject
~chatObject : ChatObject
~chatDB : DatabaseReference
~totalImageUploaded : int = 0
~imageUrlList : ArrayList<String> = new ArrayList<>()
~messageChat : EditText
~SELECT_IMAGE_INTENT : int = 1
~mediaUriArray : ArrayList<String> = new ArrayList<>()
#onCreate(savedInstanceState : Bundle) : void
-getMessages() : void
-sendMsg() : void
-updateDatabaseWithNewMessage(newMessageDb : DatabaseReference, newMessageMap : Map) : void
-initializeMsgRCV() : void
-initializePicturesRCV() : void
-accessGallery() : void
#onActivityResult(requestCode : int, resultCode : int, data : Intent) : void

IMPLEMENTATION

INTRODUCTION

While I am very conversant with using C#, Visual Studio and it's supporting Frameworks, this is my very first project using Android Studio, Android Java and some of its dependencies.

After painstakingly designing the chat app, I had to spend weeks understanding the sprawling libraries and dependencies that would support my project.

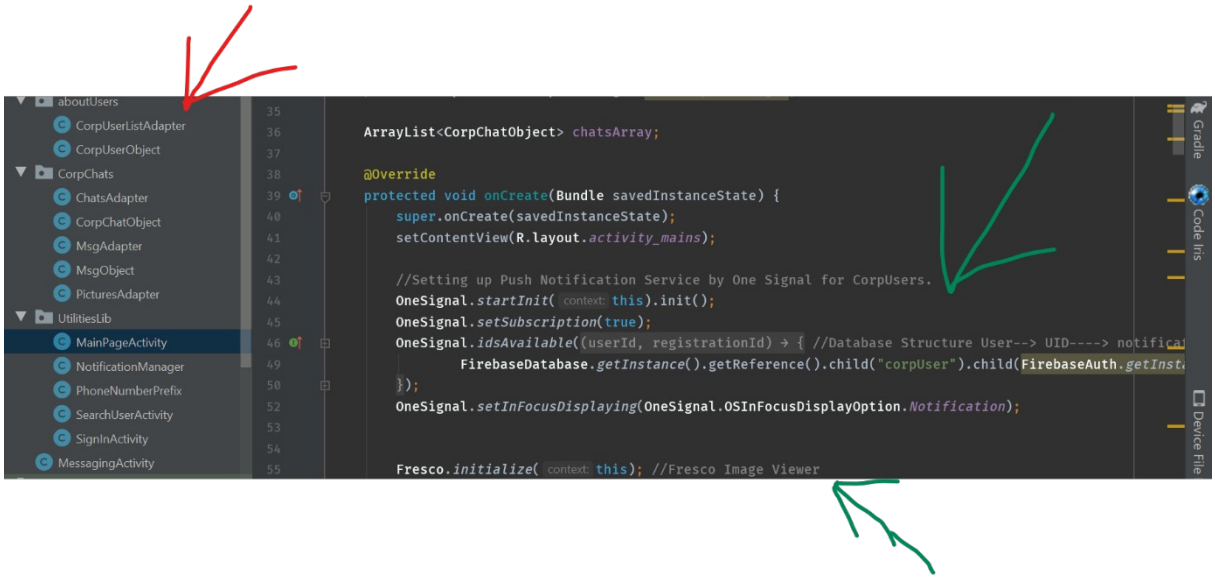
I had to customize the IDE to dark mode, change the font size and personalize it to be milder on the eyes and immersive in looks.

I had downloaded the latest Android Studio, a few challenges which might be known bugs and compatibility issues using the deprecated Support Library and the new AndroidX Libraries occurred. Amongst many things what helped resolved some of the compatibility issues was setting enableJetifier = true at the gradle.properties file, which helps converts the deprecated classes and libraries to AndroidX equivalents; using the setting minifyEnabled true and shrinkResources true helped remove the Dex Count Error and as a result I kept the app from being bloated.

Generally, it was a huge learning experience implementing the design using the Android Studio IDE; Android Studio also have some impressive and efficient auto-completion features: with a single tab, a body of necessary codes created. For example as I type my addValueEventListener, a double tab brings out the body of methods that should be implemented namely, onDataChange that takes the dataSnapshot Parameters, providing the properties to access the state of the database as events are listened to.

Auto-created body of methods providing me an overview of the necessary implementation for a specific piece of code.

CODE STRUCTURE



General Organization

I systematically organized by codes into 3 Packages, comprising of 4 Activities, 4 Adapters for the Viewholders and 5 objects.

I have used 2 external dependencies namely Onesignal and Fresco, whose Libraries I used by calling the API OneSignal.setSubscriptions and Fresco.initialize for Push Notification Services and Image Viewer Capabilities respectively. For more details on this packages you can visit their website at: Fresco and OneSignal.

SIGN IN ACTIVITY

The SignIn Activity is the Launcher Activity: the first activity that is set has the app launches launching the Login Page by - *setContentView(R.layout.activity_mains);*

The following methods or objects were called or instantiated respectively

-FirebaseUser newUser = FirebaseAuth.getInstance().getCurrentUser();

This instance of the user is not pointing to the database but only ushering the user into the MainPage Activity and creating a New Object newUser of the Firebase Class

-startVerification()
PhoneAuthProvider.getInstance().verifyPhoneNumber(userMobileNumber.getText().toString(), 60, TimeUnit.SECONDS, this, callBacks);

The VerifyPhoneNumber is an overload and the parameter list differs based on the use. Here, I need to pass the Mobile Number of User, the time out within which the user must input the Passkey, and lastly, I used a call back that will help me handle a failure or success events of this task.

-final DatabaseReference newUserDB =
newUserDB.addListenerForSingleValueEvent(new ValueEventListener()

Database Reference points to the Data Structure, that I have created at the Realtime Database Firebase Console

-logInWithPhoneCredential(PhoneAuthCredential userPhoneAuthCredential)

This Methods Signs the user in with the credential generated on successful completion of the verification.

-userSignedIn() I implemented this method to validate the presence of a signed in user after authentication is successful to ensure that the sign in proceeded to add a user to the Database and double-checks to prevent null exceptions

-newUserDB.updateChildren(corpUser);

This Updates the children of the newUserDB(with as many objects as there is in the HashMap) in the Data Structure of the Firebase Realtime Database

---

MAIN PAGE ACTIVITY AND SEARCHUSER ACTIVITY

--imageIdList.add(imageId); final StorageReference imageFilePath =
FirebaseStorage.getInstance().getReference().child("corpChats").child(chatObj.getChatKey()).child(msgId).child(imageId);

Implementation for Image File Storage using Firebase Storage and populating the ArrayList with imageID;

Implementing the Image Storage using Firebase Storage to create Virtual Files and using the UploadTask Class to input the the files into the their auto-generated imageUri

```
final StorageReference imageFilePath =
        FirebaseStorage.getInstance().getReference().child("corpChats").child(chatObj.get(

UploadTask uploadImage = imageFilePath.putFile(Uri.parse(imageUri)); //Upload Images to t
```

USE OF ADAPTERS AND RECYCLER VIEW HOLDERS

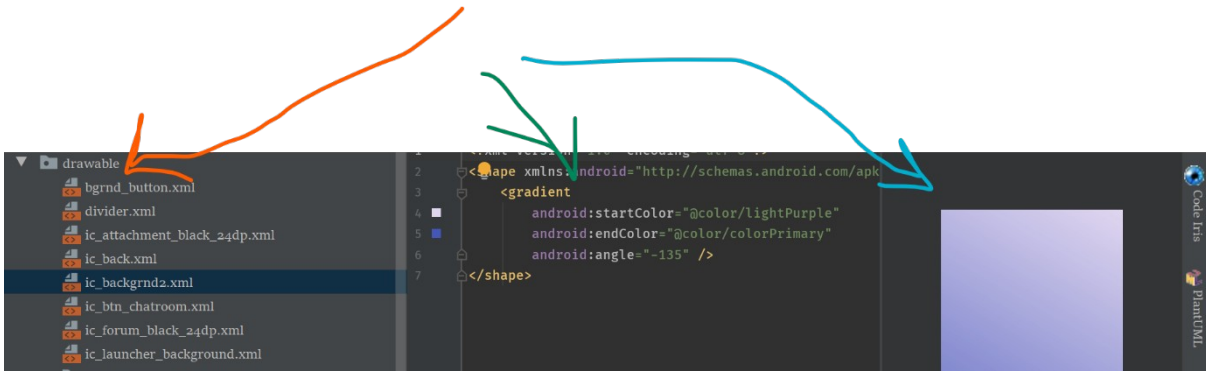
```
@Override
public void onBindViewHolder(@NonNull final CorpUserListAdapter.CorpUserViewHolder holder, final int pos

    holder._userName.setText(corpUserListArgs.get(position).getUserName());
    holder._phoneNumber.setText(corpUserListArgs.get(position).getPhoneNumber());

    holder.add.setOnCheckedChangeListener((compoundButton, isChecked) -> {
            corpUserListArgs.get(holder.getAdapterPosition()).setIsSelected(isChecked);
    });
```

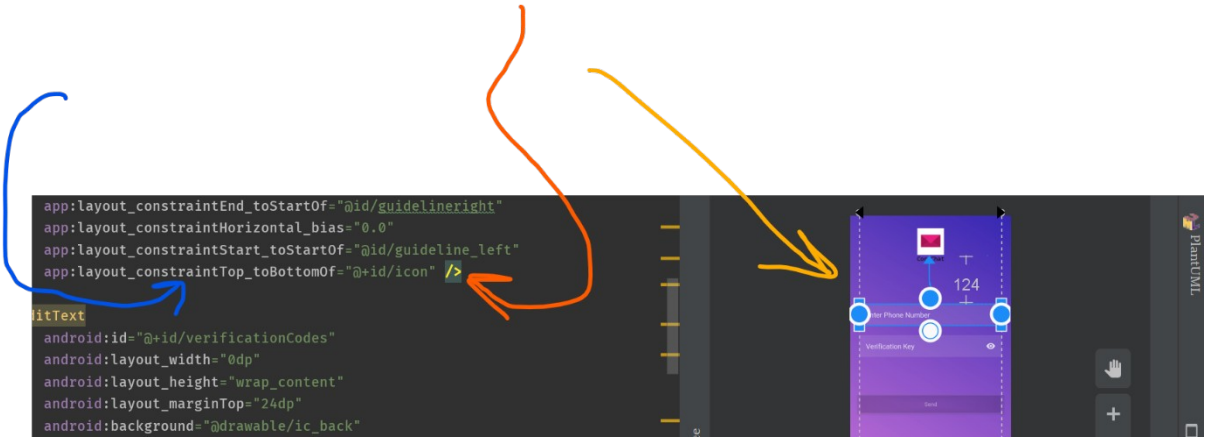
I deployed an extensive use of Recycler View Widgets and its subclass Adapter to manage my views. I have chosen this with the consideration that Recycler Views manages Data Sets whose size is determined at runtime well while optimizing memory usage. The adapter helps bind data sets to the view based on the position of the adapter while enabling improved scrollability than the Listviews approach.

CREATIVE USE OF DRAWABLES



I found a very creative way of using Drawables with the Layout items to create impressive forms and immersive User Interface throughout the app by setting the background: to the id of the drawble already implemented. (e.g *android:background="@drawable/ic_back")* properties of the Layout Objects.

USEFULNESS OF CONSTRAINTLAYOUT



I found the Properties of the Constraint Layout really useful as compared to Linear and Relative Layout respectively because it availed the facility to constrain or guard form elements on the Layout in respect to one another by using properties that allow me to specify the Id of an object on the layout.

DATABASE STRUCTURE

```
@Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) {

    if(dataSnapshot.exists()) {
        //Looping through the chats in the database
```

II used the DataSnapshot Type to access the contents of the Database, while calling the exists() to prevent null values.

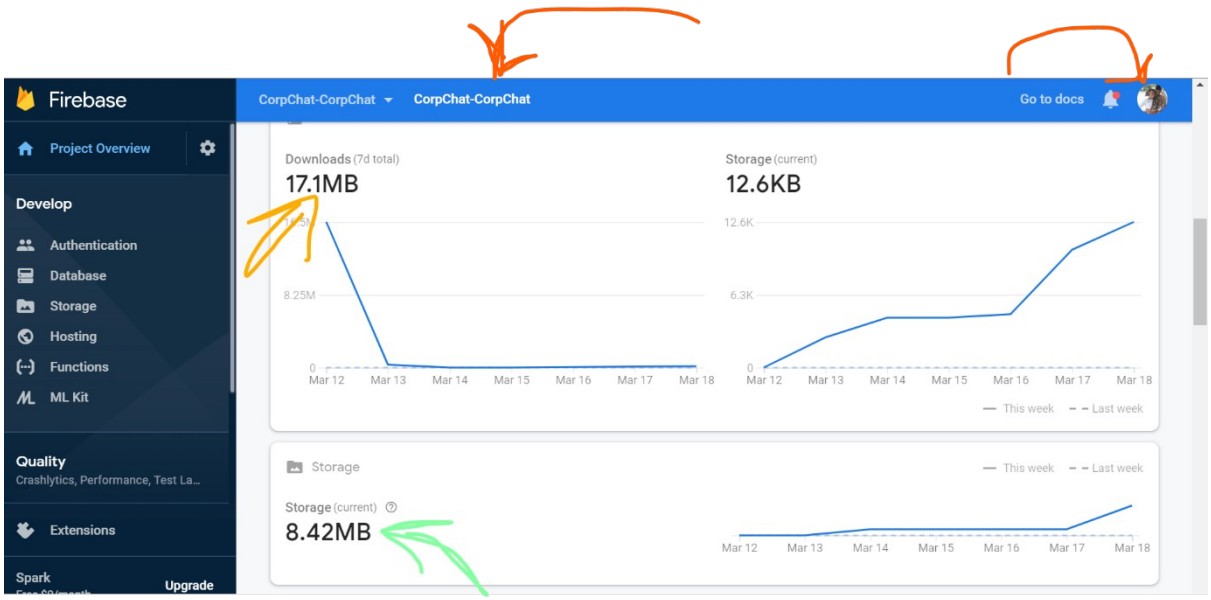
```
rivate void fetchUserData(CorpUserObject user) {
  DatabaseReference userDB = FirebaseDatabase.getInstance().getReference().child("corpUser").child(user.getUserID());
  userDB.addValueEventListener(new ValueEventListener() {
```

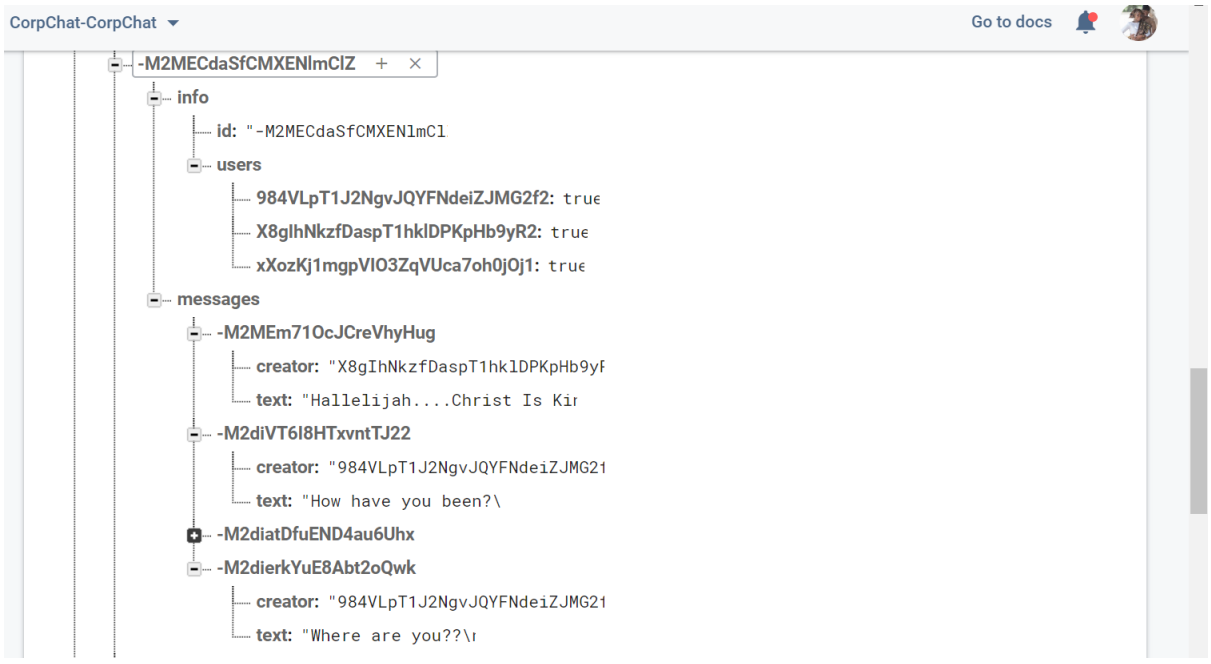
Using the *Firebase.getInstance().getReference()* to dynamically and programmatically populate and structure and input values of Database.

```
OneSignal.setSubscription(false); //This ensures that the notification service is switched off...onClick of the Sign out Button
FirebaseAuth.getInstance().signOut(); //Firebase method to sign out the user from the current session.
Intent signoutIntent = new Intent(getApplicationContext(), SignInActivity.class); //Setting the SignInActivity to an Intent.
signoutIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP); //This ensures that immediately the logout is completed, the user no longer
//has access to the current page being explored
startActivity(signoutIntent); //This basically restarts and returns the SignInActivity Page
finish();
return;
```

signoutIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP); Kindly see in-text commentary above: I have used the addFlags() method to ensure that after Signing out from the Database, the Activity Form or Page becomes inaccessible by returning the user to the Login Page after Signout.

Database Storage and Usage





Database Structure for the Querying, Insertion, Deletion and Update of user data.

FUNCTIONALITY TESTING

OVERVIEW

Bearing in mind the scope and resources available for this project, my aim was to utilize the most time-efficient Testing Framework available. Before settling in for a test method and framework, I considered Local Unit Testing using Junit and Mokito, InstrumentationTesting Using Junit or Mokito and UI Testing using Espresso. Based on time constraint, I went for a UI Testing using the Android Dependency Espresso. I also love the fact that the UI Testing based on espresso allows be to create test cases based on the user interactions. Using the Espresso Test Recorder, I can capture programmatic assertions and user interactions without utilizing the app code base directly, creating a much needed test development speed with impressive test case optimization.

S/N	TEST CASE	INPUT	EXPECTED	ACTUAL	REMARKS
1	Test Case for Valid Phone Number Authentication	Test Phone Number and Database-generated Code	User Logged In	User Logged In	Pass
2	Test Case for Displaying Registered Contacts	Registered Phone Contacts on Contact List	Contacts Detected within App	Contacts Detected within App	Pass
3	Test Case for Creating Groups and Chats	Group Name or Chat List	Chat Room Created	Chat Room Created	Pass
4	Test Case for Sending Messages	Edit text input for Text messages	Message sent and received	Message sent and received	Pass
5	Test Case for Receiving One Signal Push Notification	Sent Instant messages including media and text	Receiver receives Push notification from the One signal Server	Receiver receives Push notification from the One signal Server	Pass
6	Test for Invalid Verification Code	Invalid Verification Code	Authentication denial to read and write from the database	Authentication denial to read and write from the database	Pass
7	Test out for Sign-out from Database	On Click of the Sign-out Button	Sign-out of from the database and a view return to the login page	Sign-out of from the database and a view return to the login page	Pass

CRITIQUE

While I have worked painstakingly and passionately to design and implement this app, and while there are no bugs at the present level of functionality support it produces, there still a lot to be done with the usability and optimization of the app layout.

Obvious constraints includes being the only the developer working this in such a short amount of time; I will like to improve and extend the functionality of this app, by working to improve the navigation to reflect a more logical flow of events; for example, after creating a group chat, the view should just automatically go to the newly created chats. That is definitely more logical and useable.

Furthermore, currently, the chat ID and chat sender ID are being displayed at the chats list and sender's name respectively are only showing the IDs. It is bad from a usability point of view. I should call a corresponding *getName()* method to be passed into the recycler view for every chat object and user object ID queries from the database. I do not have the time to write and debug that write now considering the name.

Additionality, I would like to refurbish the UI to feel more immersive, add more security features like biometric signin. For now, phone number authentication is the only authentication feature created while setting the Database Security Rules for both Read and Write Functions to *auth != null.* Definitely, that is not good enough, as sensitive user data needs a stronger and more complex security to protect sensitive user data. I will need to look more into this going forward.

Lastly, I'd like to add more social media functionalities, like send items from your chat groups to a social media account, video chats, emoji, GIF, etc. but for now, I think am satisfied with the output so far.