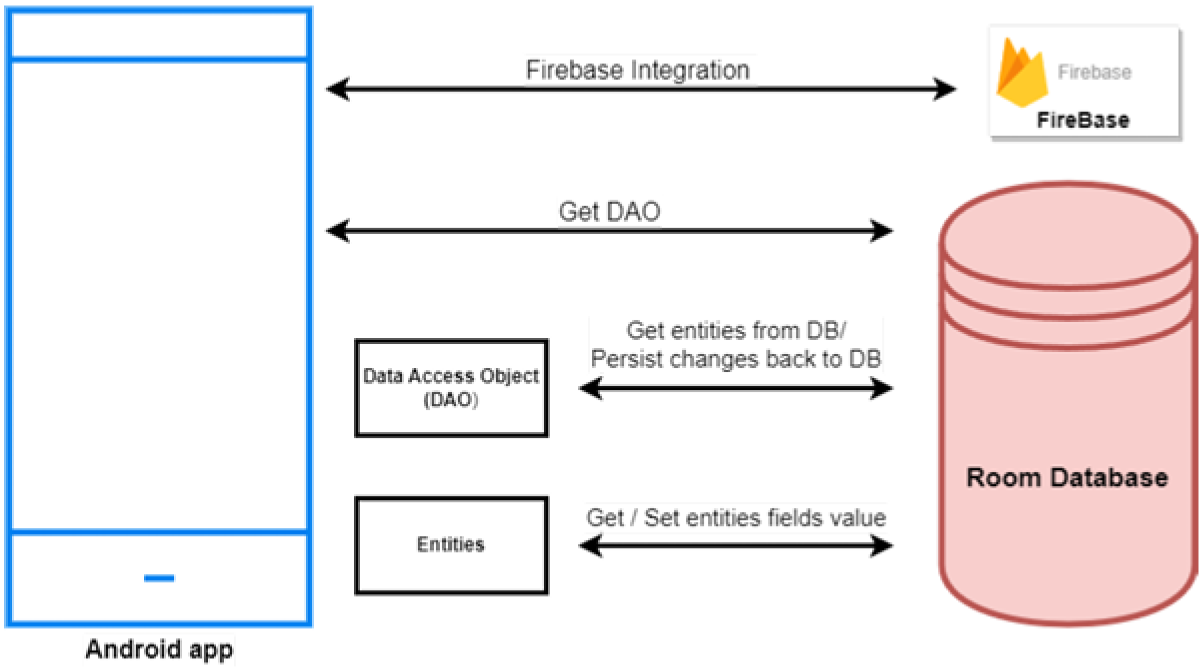
Chat Connect - A Real-Time Chatting and Communications App

Introduction:

BaatCheet is a project built using the Android Compose UI toolkit. It gives an idea about how to create a chat app using the Compose libraries. The app allows users to send and receive text messages through direct messages or through groups. The project showcases the use of Compose's declarative UI and state management capabilities. It also includes ways in which users can learn how to handle input and navigation using composable functions and how to use data from a firebase to populate the UI.

Technical Architecture:



Prerequisites:

To complete this project, one must require the following software's, concepts, and packages

Android Studio is the official integrated development environment (IDE) for Android app development. It provides a user-friendly environment for creating, testing, and debugging Android applications. Android Studio offers tools for designing user interfaces, writing code in Kotlin or Java, and managing project resources. It also includes an emulator for testing apps on various Android devices. Concepts such as MVP Architecture, Push Notifications, and Real-time communications are also important for the success of this project. Implementation of Firebase is also a crucial part for this project.

To install Android Studio, follow the link given below

Link: Click Here to install Android Studio.

Project Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques of MVP Architecture, Firebase integration and creating projects using android studio.
- Understand the basic concepts regarding foreground and background threads and services.
- Implementation of push notifications and real-time chatting features.

Project Flow:

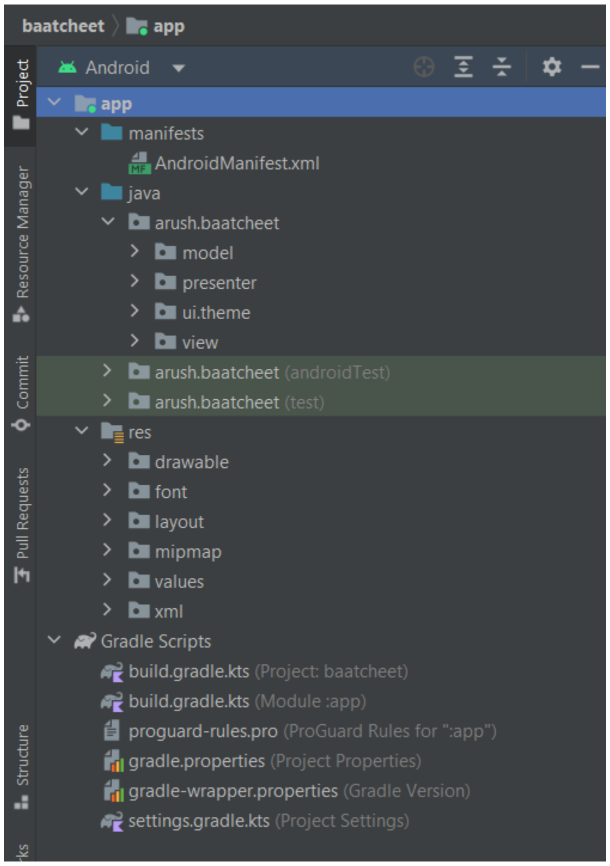
- The user registers into the application using their mobile numbers. Verification is done through a One Time Password(OTP) which allows the user to log into the app.
- Main page consists of a list of chats and methods to create new instances of chats. It also has a way to navigate to the user profile.
- The user profile has user details with a means to edit them, if need be.

To accomplish this, we have to complete all the activities and tasks listed below

- Login Activity
    1. Importing necessary libraries
    2. Authorization
- Home Page Activity
    1. Importing important libraries and packages.
    2. Creating logic and connecting it with UI using Presenter file.
    3. Building the UI
    4. Accessing profile
    5. Adding Contacts
- Profile Activity
    1. Importing libraries and building the UI
    2. Saved Messages Logic
    3. Edit Page redirection.
- Chat Activity
    1. Building Chat Screen UI
- Model Building
    1. Encrypting and Decrypting Data
    2. User Details Model
    3. Background Services
    4. Login Model
    5. Database Handling Model
    6. File Handling Model

Project Structure:

Create a Project folder which contains files as shown below



```
app
  manifests
    AndroidManifest.xml
  java
    arush.baatcheet
      model
      presenter
      ui.theme
      view
    arush.baatcheet (androidTest)
    arush.baatcheet (test)
  res
    drawable
    font
    layout
    mipmap
    values
    xml
Gradle Scripts
  build.gradle.kts (Project: baatcheet)
  build.gradle.kts (Module :app)
  proguard-rules.pro (ProGuard Rules for ":app")
  gradle.properties (Project Properties)
  gradle-wrapper.properties (Gradle Version)
  settings.gradle.kts (Project Settings)
```

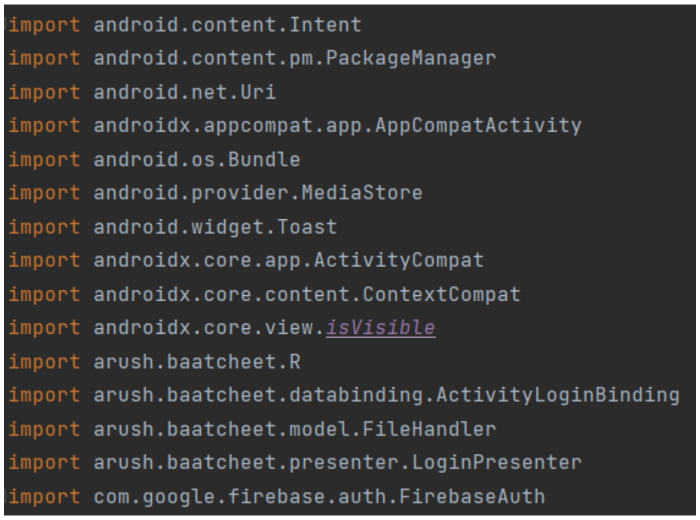
The **model folder** consists of the "business" model and the methods and rules by which data is handled. The **presenter folder** acts as an intermediary between the model and the view codes. The **ui.theme folder** consists of the theme of the project. The **view folder** contains the UI for the project.

The res section and gradle section consists of resources and implementations required for the project.

Milestone 1: Login Activity

Activity 1: Importing necessary libraries

The initial step is to import all the necessary libraries.



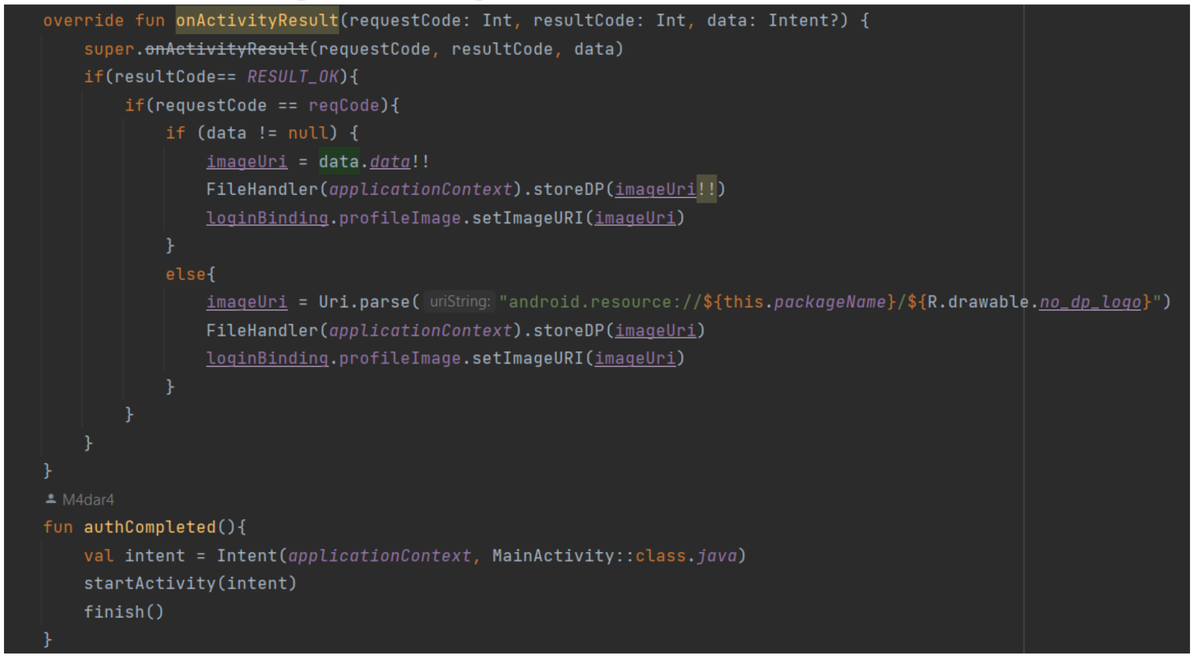
```kotlin
import android.content.Intent
import android.content.pm.PackageManager
import android.net.Uri
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.provider.MediaStore
import android.widget.Toast
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import androidx.core.view.isVisible
import arush.baatcheet.R
import arush.baatcheet.databinding.ActivityLoginBinding
import arush.baatcheet.model.FileHandler
import arush.baatcheet.presenter.LoginPresenter
import com.google.firebase.auth.FirebaseAuth
```

Activity 2: Authorization

To authorize the user trying to access the app, it first fetches the data to check if the user is already part of the application. If the user isn't, the user is requested to register into the application using their mobile number.

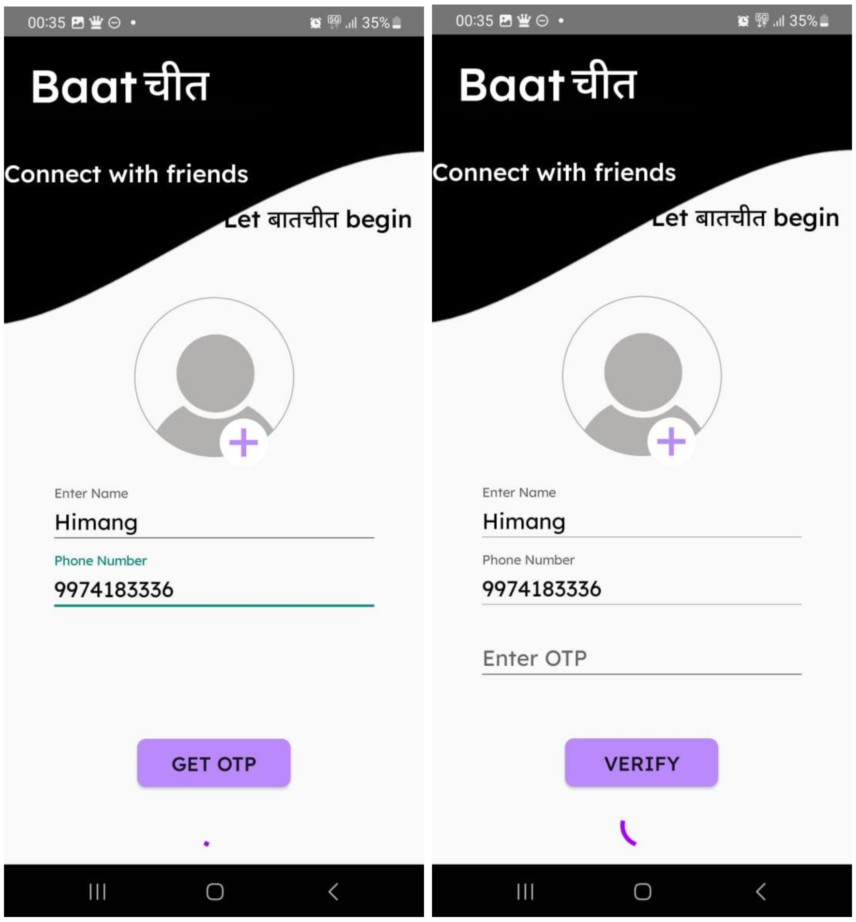
Once the user fills all the required details, a One Time Password(OTP) is sent on their mobile phone which they have to enter.

Then the authorization process takes place and enrolls the user.



```kotlin
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.~~onActivityResult~~(requestCode, resultCode, data)
    if(resultCode== RESULT_OK){
        if(requestCode == reqCode){
            if (data != null) {
                imageUri = data.data!!
                FileHandler(applicationContext).storeDP(imageUri!!)
                loginBinding.profileImage.setImageURI(imageUri)
            }
            else{
                imageUri = Uri.parse( uriString: "android.resource://${this.packageName}/${R.drawable.no_dp_logo}")
                FileHandler(applicationContext).storeDP(imageUri)
                loginBinding.profileImage.setImageURI(imageUri)
            }
        }
    }
}
M4dar4
fun authCompleted(){
    val intent = Intent(applicationContext, MainActivity::class.java)
    startActivity(intent)
    finish()
}
```

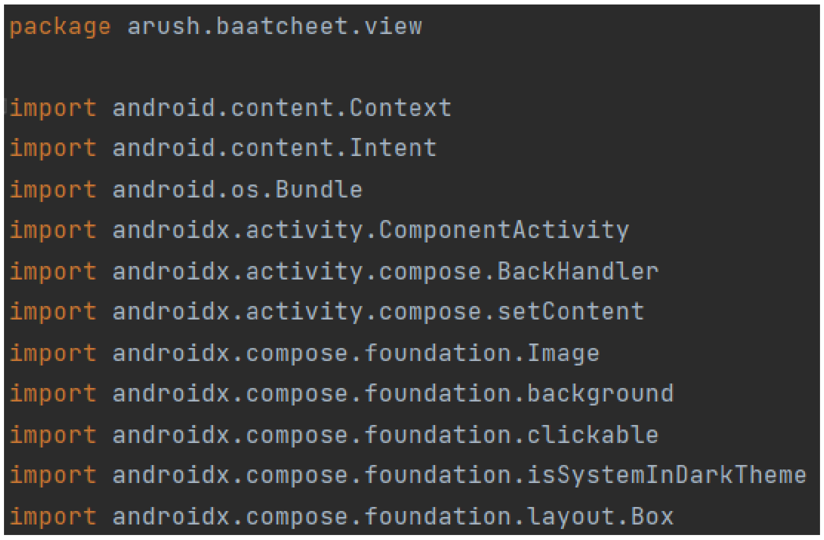
Output:



Milestone 2: Main Activity

Activity 1: Importing important libraries and packages.

Importing necessary packages and libraries.



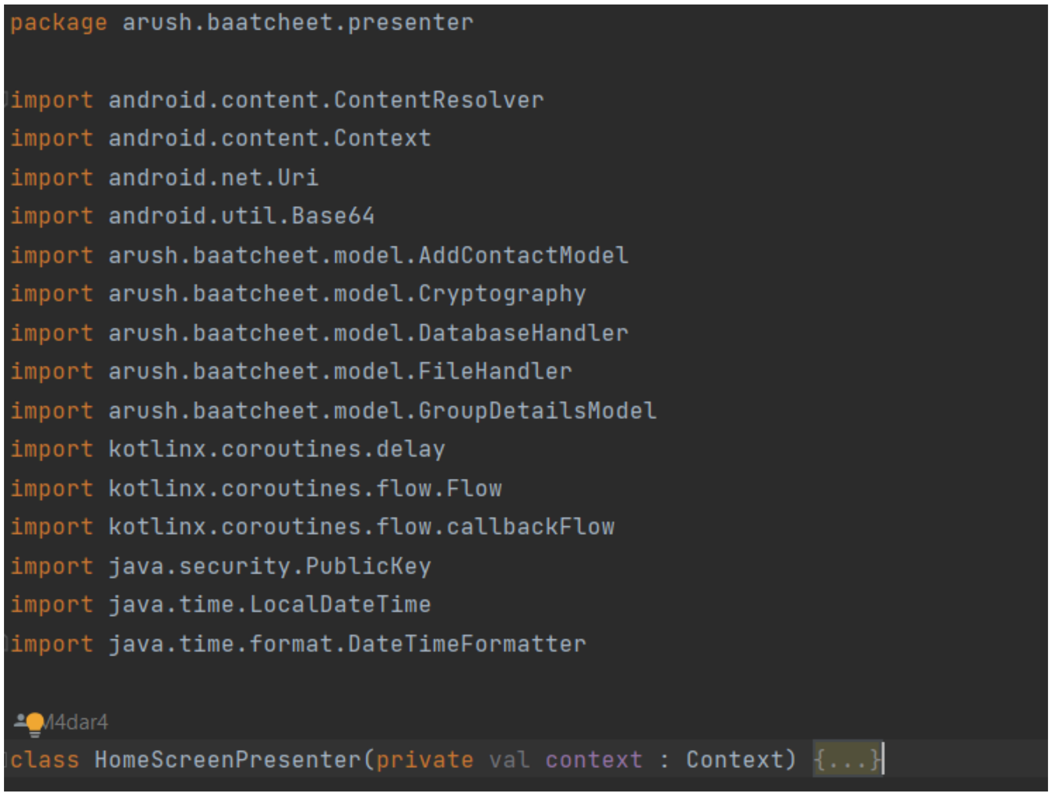
```
package arush.baatcheet.view

import android.content.Context
import android.content.Intent
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.BackHandler
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.isSystemInDarkTheme
import androidx.compose.foundation.layout.Box
```

Plus, 50 more imports.

Here, the UI of the app is created to show the Home Screen of the application. It consists of a basic layout with links to creating new chats and groups, and profile viewing. Also, it consists of a simple search bar which allows users to search other users on the application or invite them.

Activity 2: Creating logic and connecting it with UI using Presenter file.

The logic of Home Screen is connected to the view model using HomeScreenPresenter as shown below.

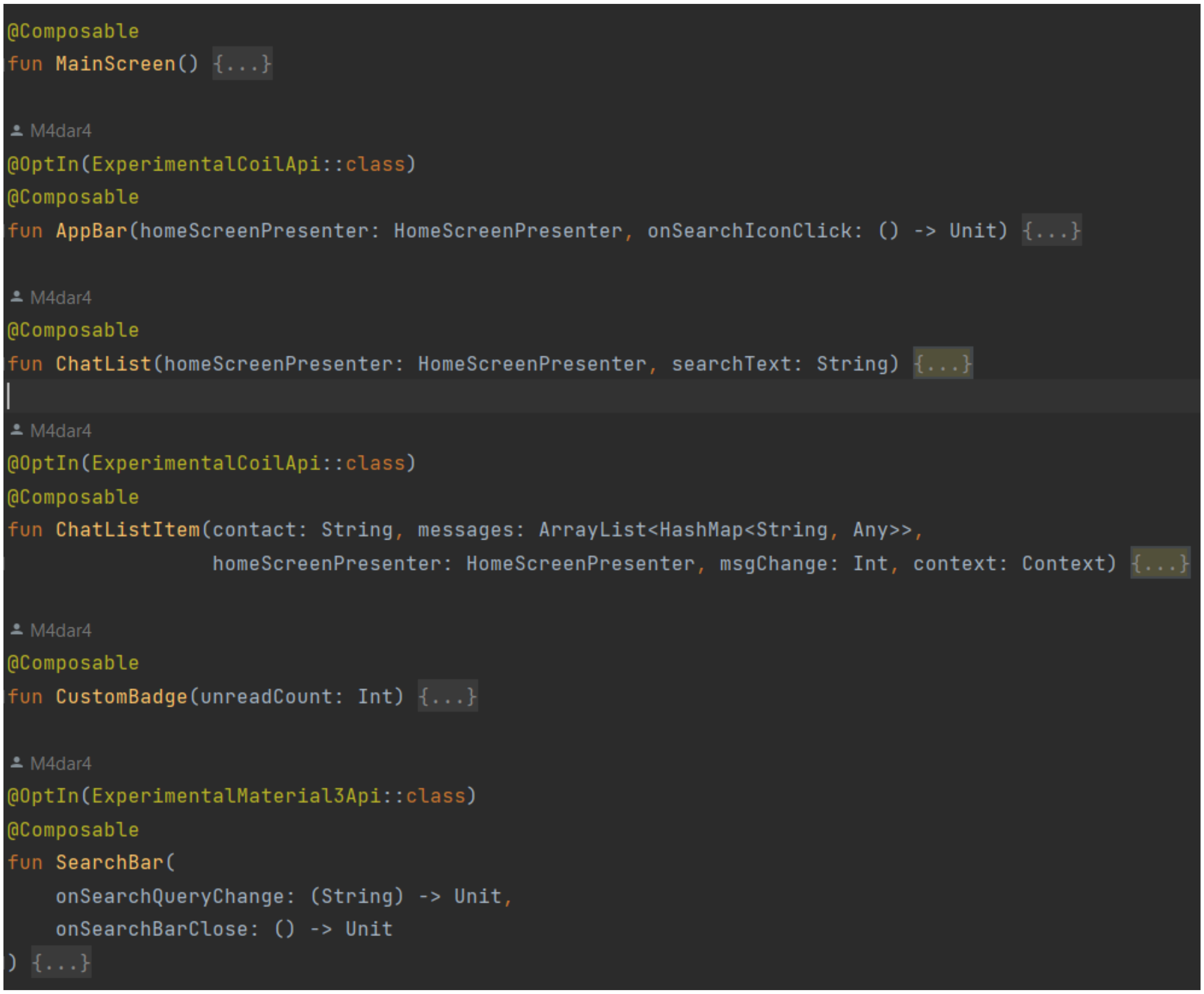


```
package arush.baatcheet.presenter

import android.content.ContentResolver
import android.content.Context
import android.net.Uri
import android.util.Base64
import arush.baatcheet.model.AddContactModel
import arush.baatcheet.model.Cryptography
import arush.baatcheet.model.DatabaseHandler
import arush.baatcheet.model.FileHandler
import arush.baatcheet.model.GroupDetailsModel
import kotlinx.coroutines.delay
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.callbackFlow
import java.security.PublicKey
import java.time.LocalDateTime
import java.time.format.DateTimeFormatter

M4dar4
class HomeScreenPresenter(private val context : Context) {...}
```

This HomeScreenPersenter communicates with different models present in the models folder. It also consists of logic for receiving chats and sending chats in the Chat Screen.

Activity 3: Building the UI
The UI consists of the following:

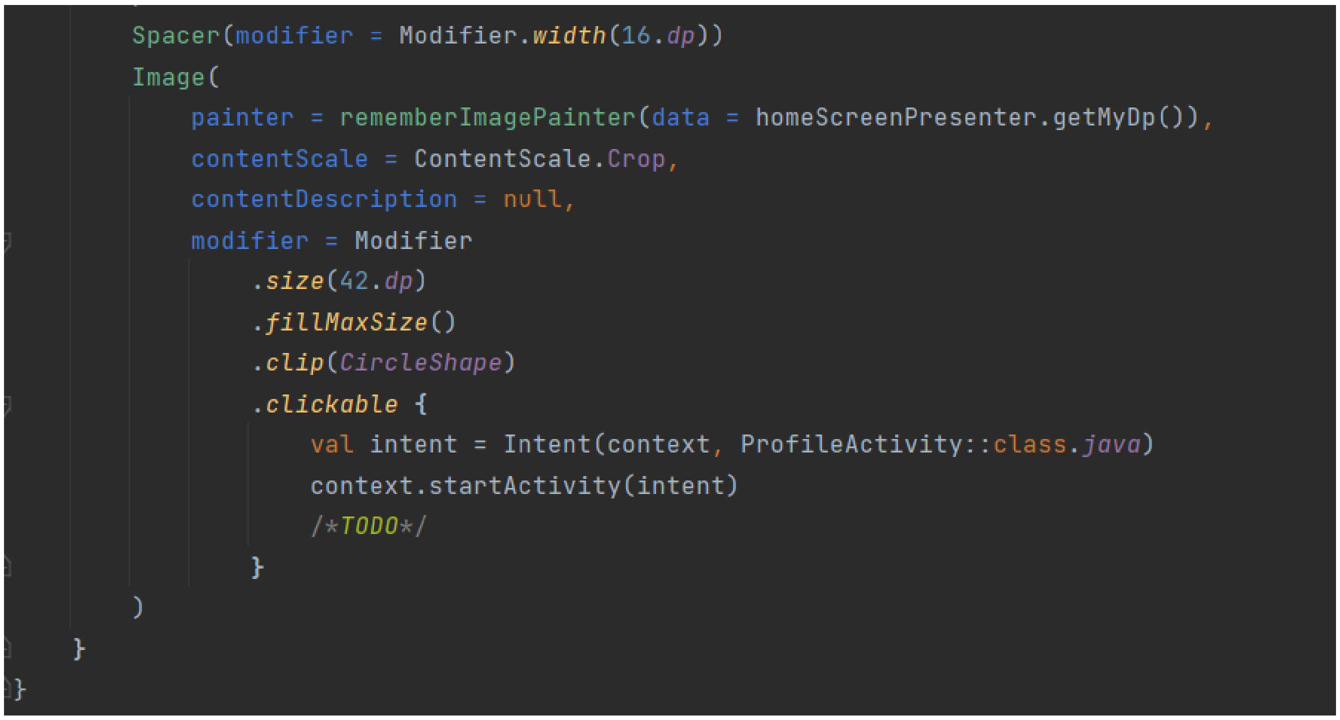
- App Bar
- Chat List showing all the previous conversations
- A button to add new contacts and groups
- Search bar



```
@Composable
fun MainScreen() {...}

M4dar4
@OptIn(ExperimentalCoilApi::class)
@Composable
fun AppBar(homeScreenPresenter: HomeScreenPresenter, onSearchIconClick: () -> Unit) {...}

M4dar4
@Composable
fun ChatList(homeScreenPresenter: HomeScreenPresenter, searchText: String) {...}

M4dar4
@OptIn(ExperimentalCoilApi::class)
@Composable
fun ChatListItem(contact: String, messages: ArrayList<HashMap<String, Any>>,
                 homeScreenPresenter: HomeScreenPresenter, msgChange: Int, context: Context) {...}

M4dar4
@Composable
fun CustomBadge(unreadCount: Int) {...}

M4dar4
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun SearchBar(
    onSearchQueryChange: (String) -> Unit,
    onSearchBarClose: () -> Unit
) {...}
```

Activity 4: Accessing Profile

To access the current user's profile, the user can access it using the App Bar, which shows the user's profile picture.

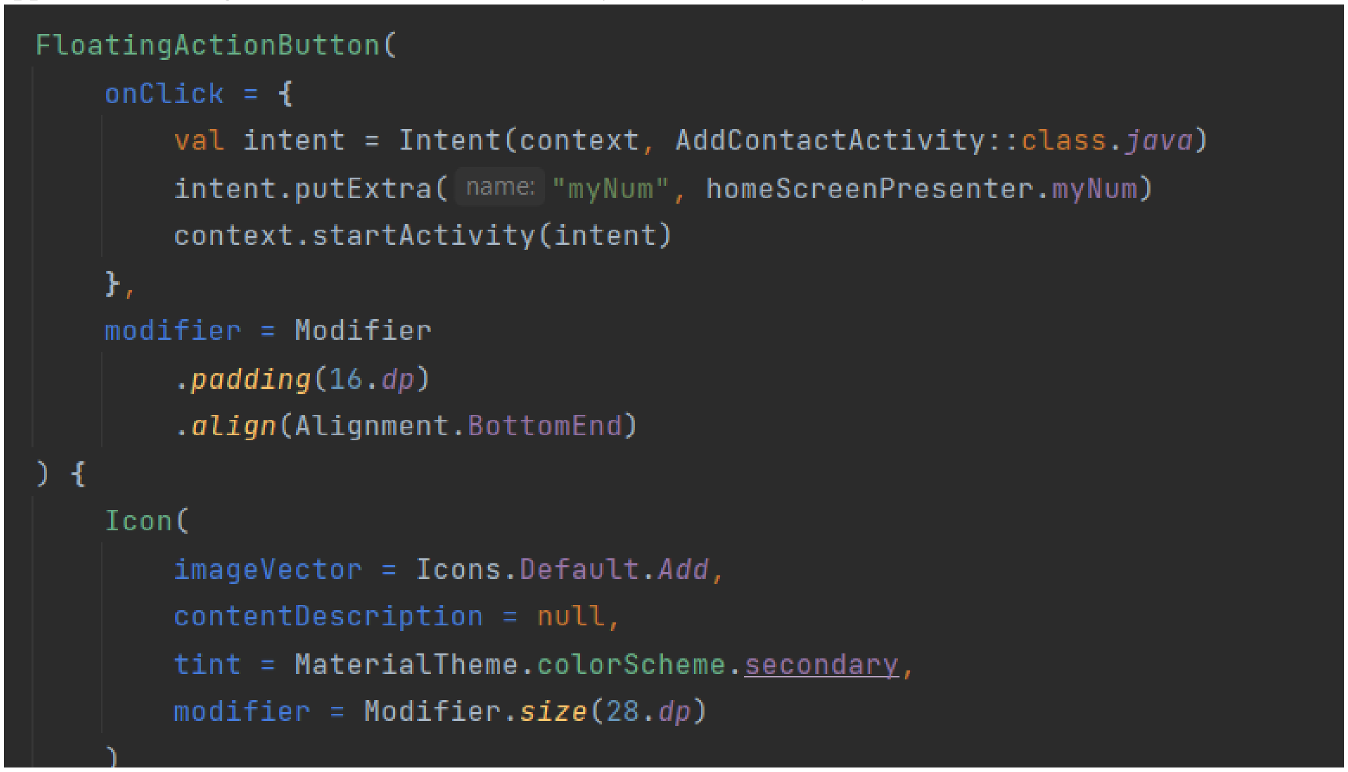


```
        Spacer(modifier = Modifier.width(16.dp))
        Image(
            painter = rememberImagePainter(data = homeScreenPresenter.getMyDp()),
            contentScale = ContentScale.Crop,
            contentDescription = null,
            modifier = Modifier
                .size(42.dp)
                .fillMaxSize()
                .clip(CircleShape)
                .clickable {
                    val intent = Intent(context, ProfileActivity::class.java)
                    context.startActivity(intent)
                    /*TODO*/
                }
        )
    }
}
```

Here, getMyDp function extracts the user's DP(Display Picture) from the database and displays it in this field. The intent variable is used to navigate to the Profile Activity.

Activity 5: Adding Contacts

The Main Activity also includes a feature which allows the user to add contacts to this application. Navigation to Add Contact Activity from Main Activity is as shown below.

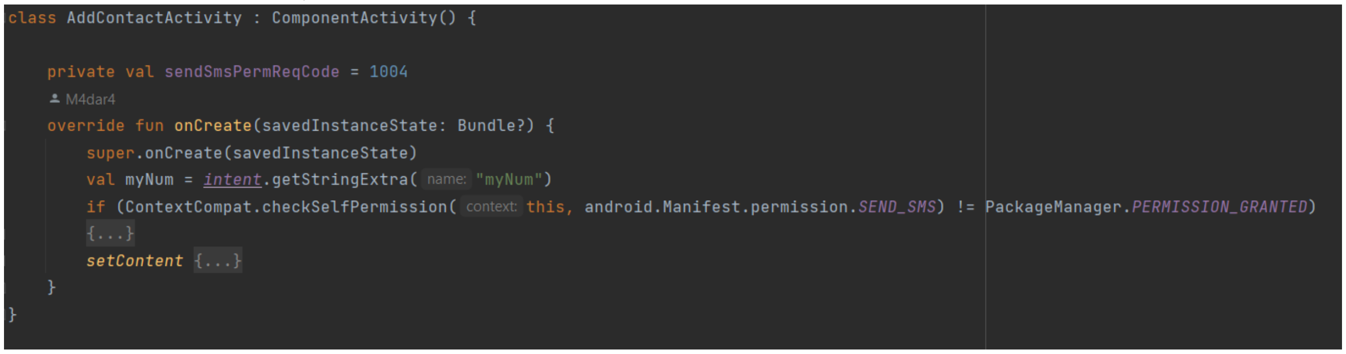


```
FloatingActionButton(
    onClick = {
        val intent = Intent(context, AddContactActivity::class.java)
        intent.putExtra( name: "myNum", homeScreenPresenter.myNum)
        context.startActivity(intent)
    },
    modifier = Modifier
        .padding(16.dp)
        .align(Alignment.BottomEnd)
) {
    Icon(
        imageVector = Icons.Default.Add,
        contentDescription = null,
        tint = MaterialTheme.colorScheme.secondary,
        modifier = Modifier.size(28.dp)
    )
```

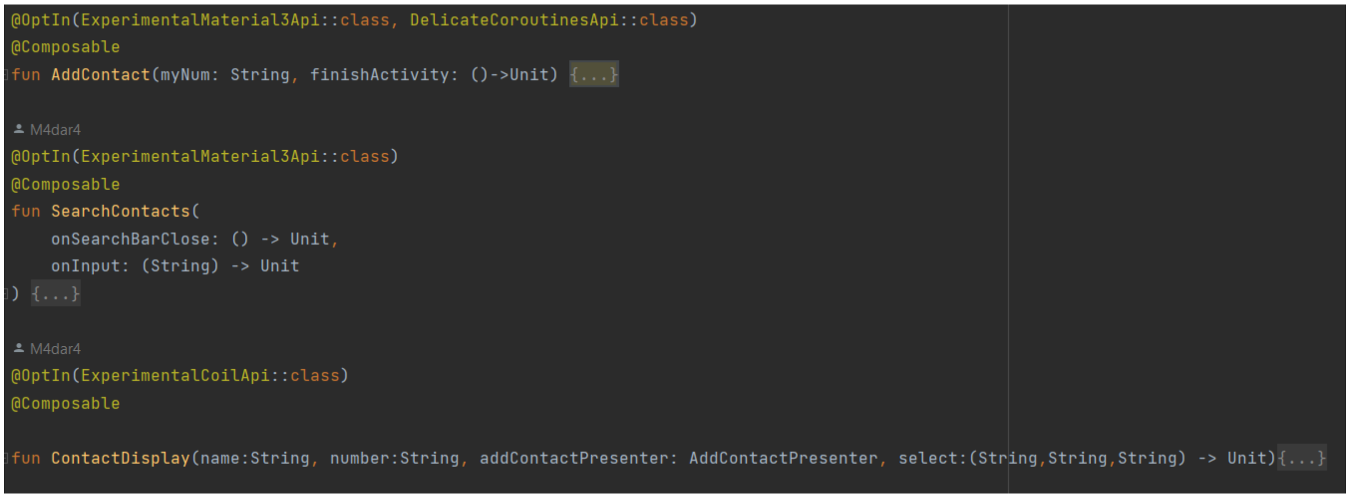
The FloatingActionButton is part of the Top App function.

This navigates the user to AddContactActivity which includes the UI code and implementation of logic through a Presenter called AddContactPresenter.

The AddContactActivity code can be summarized as shown below.

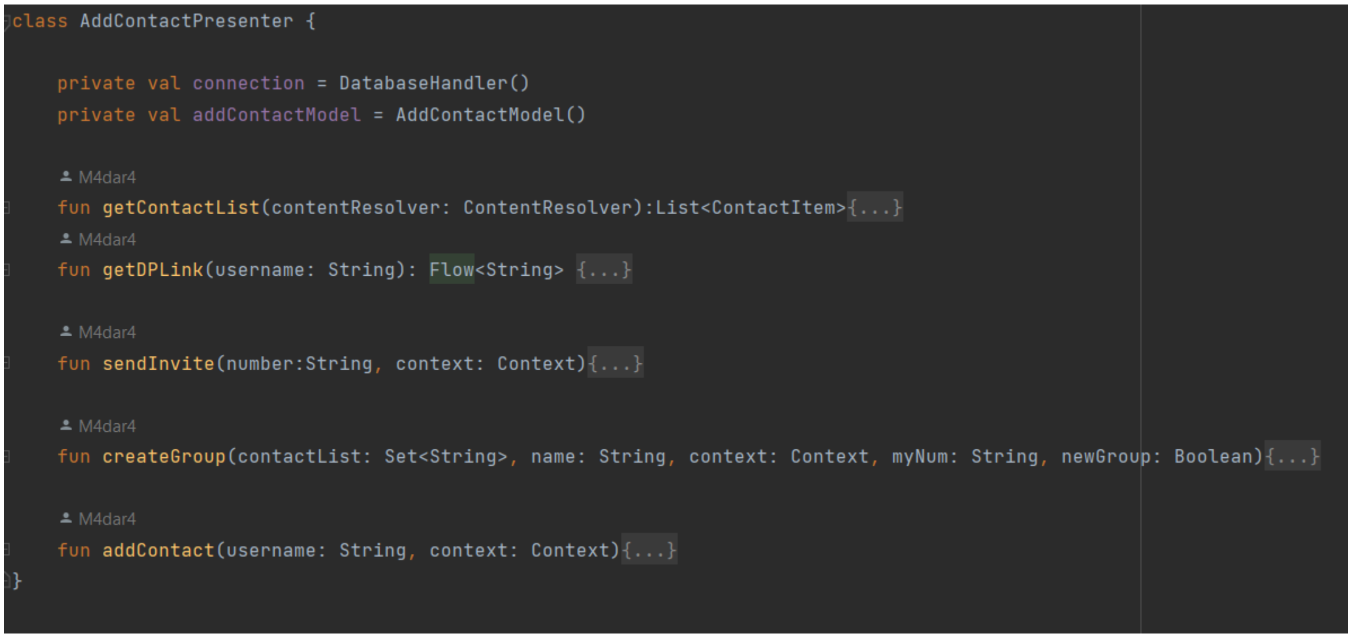


```
class AddContactActivity : ComponentActivity() {

    private val sendSmsPermReqCode = 1004
    M4dar4
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val myNum = intent.getStringExtra( name: "myNum")
        if (ContextCompat.checkSelfPermission( context: this, android.Manifest.permission.SEND_SMS) != PackageManager.PERMISSION_GRANTED)
        {...}
        setContent {...}
    }
}
```



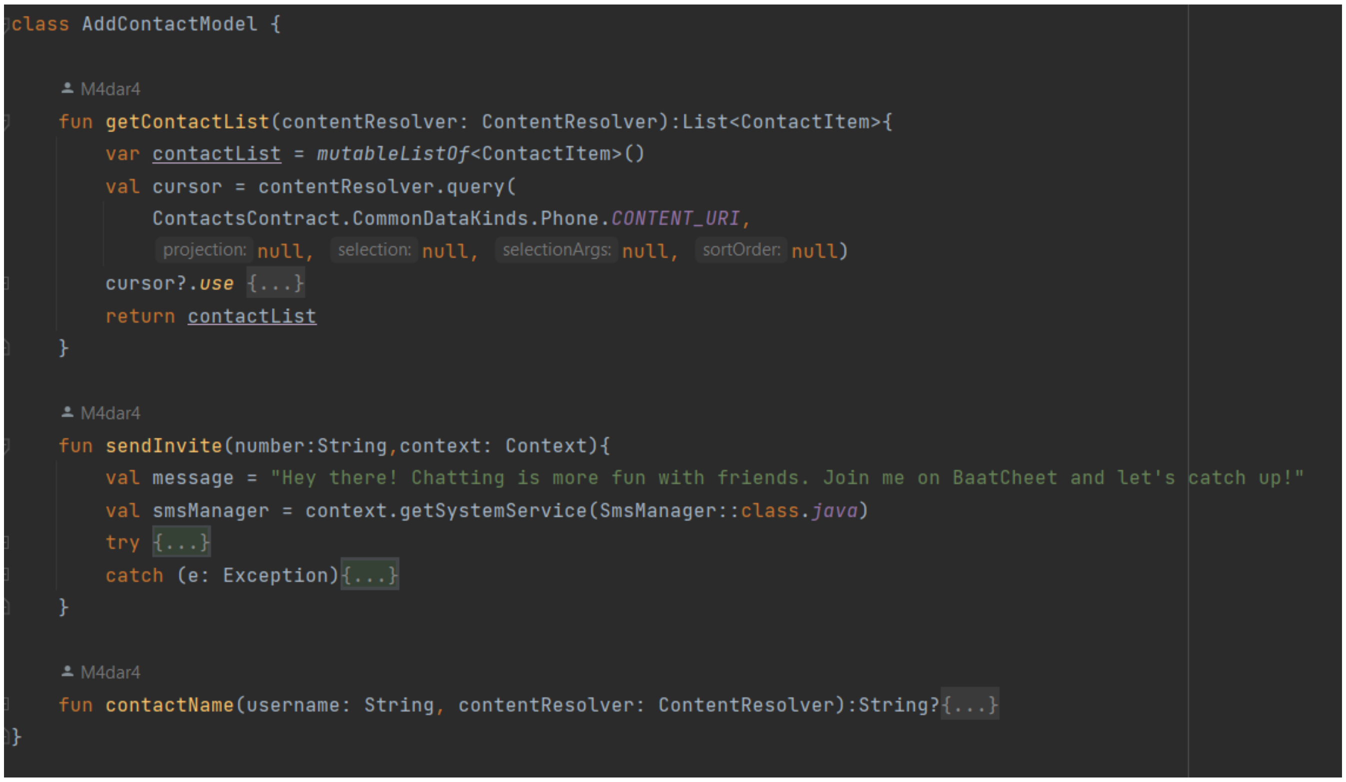
```
@OptIn(ExperimentalMaterial3Api::class, DelicateCoroutinesApi::class)
@Composable
fun AddContact(myNum: String, finishActivity: ()->Unit) {...}

M4dar4
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun SearchContacts(
    onSearchBarClose: () -> Unit,
    onInput: (String) -> Unit
) {...}

M4dar4
@OptIn(ExperimentalCoilApi::class)
@Composable

fun ContactDisplay(name:String, number:String, addContactPresenter: AddContactPresenter, select:(String,String,String) -> Unit){...}
```

The AddContactPresenter code can be summarized as shown below.



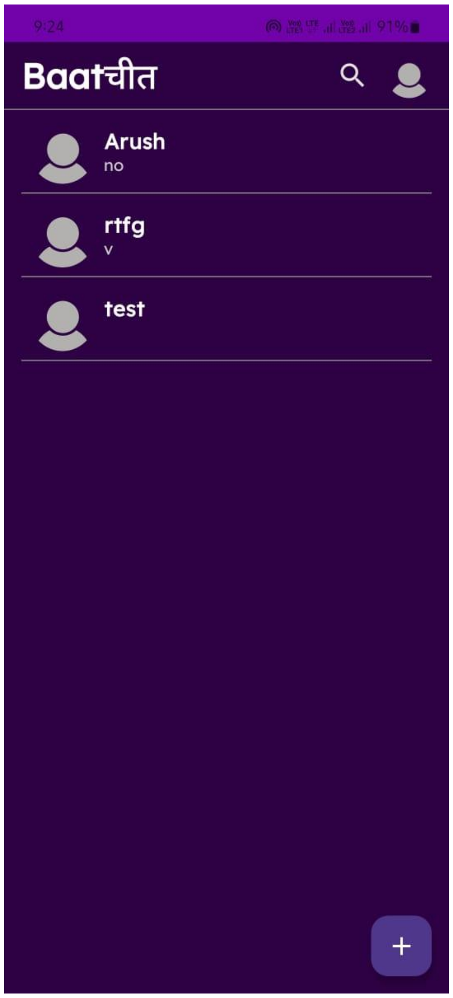
```
class AddContactPresenter {

    private val connection = DatabaseHandler()
    private val addContactModel = AddContactModel()

    M4dar4
    fun getContactList(contentResolver: ContentResolver):List<ContactItem>{...}
    M4dar4
    fun getDPLink(username: String): Flow<String> {...}

    M4dar4
    fun sendInvite(number:String, context: Context){...}

    M4dar4
    fun createGroup(contactList: Set<String>, name: String, context: Context, myNum: String, newGroup: Boolean){...}

    M4dar4
    fun addContact(username: String, context: Context){...}
}
```

And the AddContactModel code can be summarized as shown below.

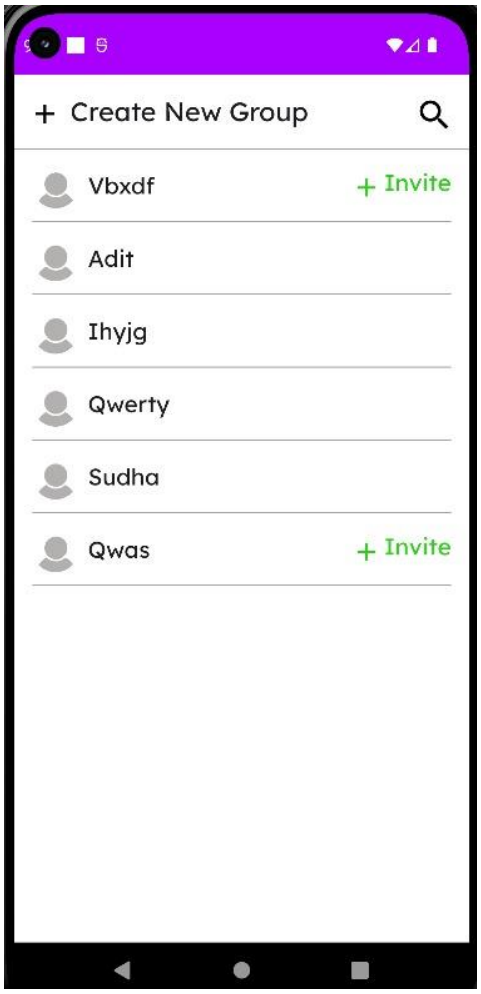


```
class AddContactModel {

    M4dar4
    fun getContactList(contentResolver: ContentResolver):List<ContactItem>{
        var contactList = mutableListOf<ContactItem>()
        val cursor = contentResolver.query(
            ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
             projection: null,  selection: null,  selectionArgs: null,  sortOrder: null)
        cursor?.use {...}
        return contactList
    }

    M4dar4
    fun sendInvite(number:String,context: Context){
        val message = "Hey there! Chatting is more fun with friends. Join me on BaatCheet and let's catch up!"
        val smsManager = context.getSystemService(SmsManager::class.java)
        try {...}
        catch (e: Exception){...}
    }

    M4dar4
    fun contactName(username: String, contentResolver: ContentResolver):String?{...}
}
```

Output:



Arush
no
rtfg
v
test



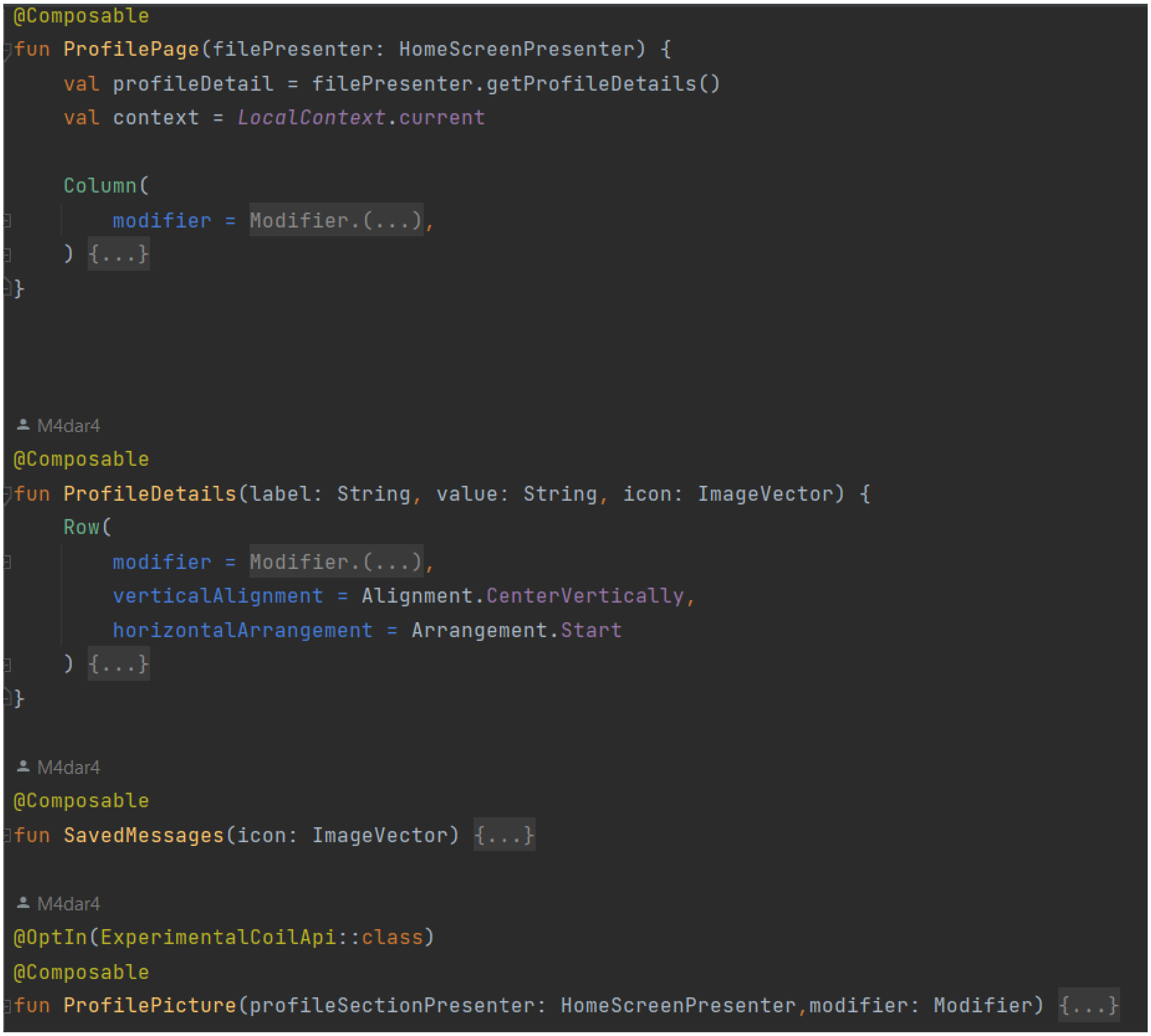
Vbxdf + Invite
Adit
Ihyjg
Qwerty
Sudha
Qwas + Invite

Milestone 3: Profile Activity

Activity 1: Importing libraries and building the UI

All the required libraries are imported into the file for seamless working of the code. There are a total of 51 libraries imported essential for seamless working.

The code includes some important functions which are essential for a proper UI representation. The codes are shown below in the image.



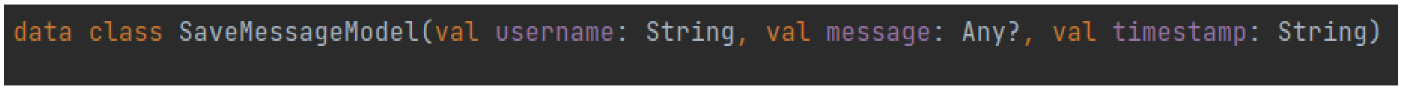
```
@Composable
fun ProfilePage(filePresenter: HomeScreenPresenter) {
    val profileDetail = filePresenter.getProfileDetails()
    val context = LocalContext.current

    Column(
        modifier = Modifier.(...),
    ) {...}
}



👤 M4dar4
@Composable
fun ProfileDetails(label: String, value: String, icon: ImageVector) {
    Row(
        modifier = Modifier.(...),
        verticalAlignment = Alignment.CenterVertically,
        horizontalArrangement = Arrangement.Start
    ) {...}
}

👤 M4dar4
@Composable
fun SavedMessages(icon: ImageVector) {...}

👤 M4dar4
@OptIn(ExperimentalCoilApi::class)
@Composable
fun ProfilePicture(profileSectionPresenter: HomeScreenPresenter,modifier: Modifier) {...}
```

Activity 2: Saved Messages Logic

The saved messages logic can be divided into 2 different classes. The first class is a data class which specifies the structure of the message while the second class integrates it with the view code to display the saved messages.

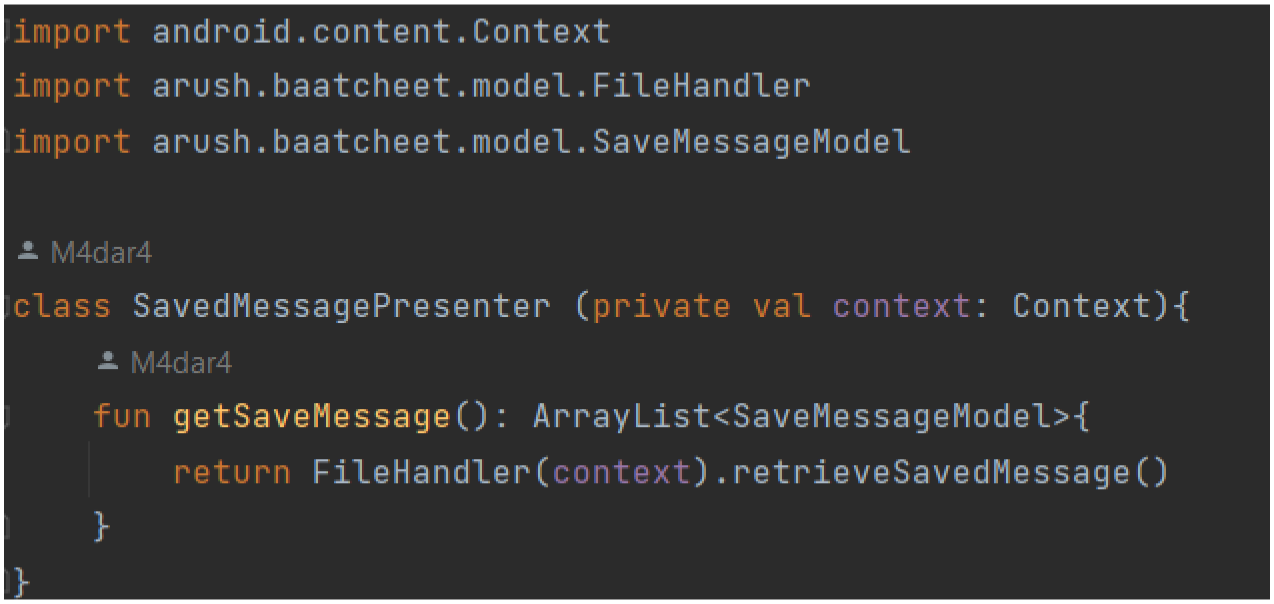
These are shown as below.

The Data Class



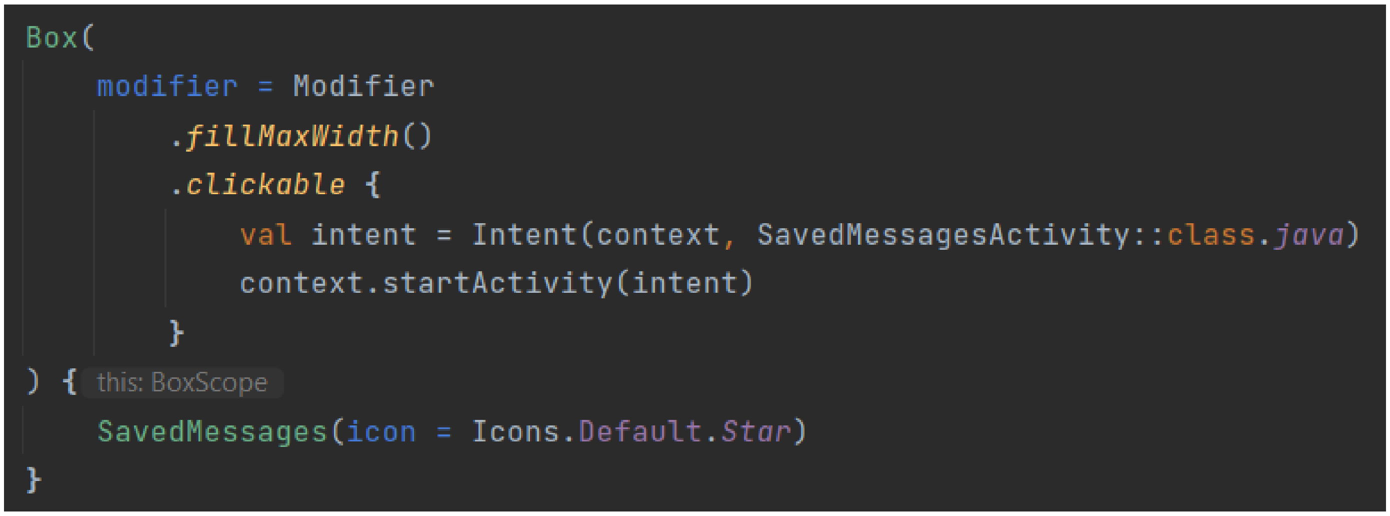
```kotlin
data class SaveMessageModel(val username: String, val message: Any?, val timestamp: String)
```

The Presenter class

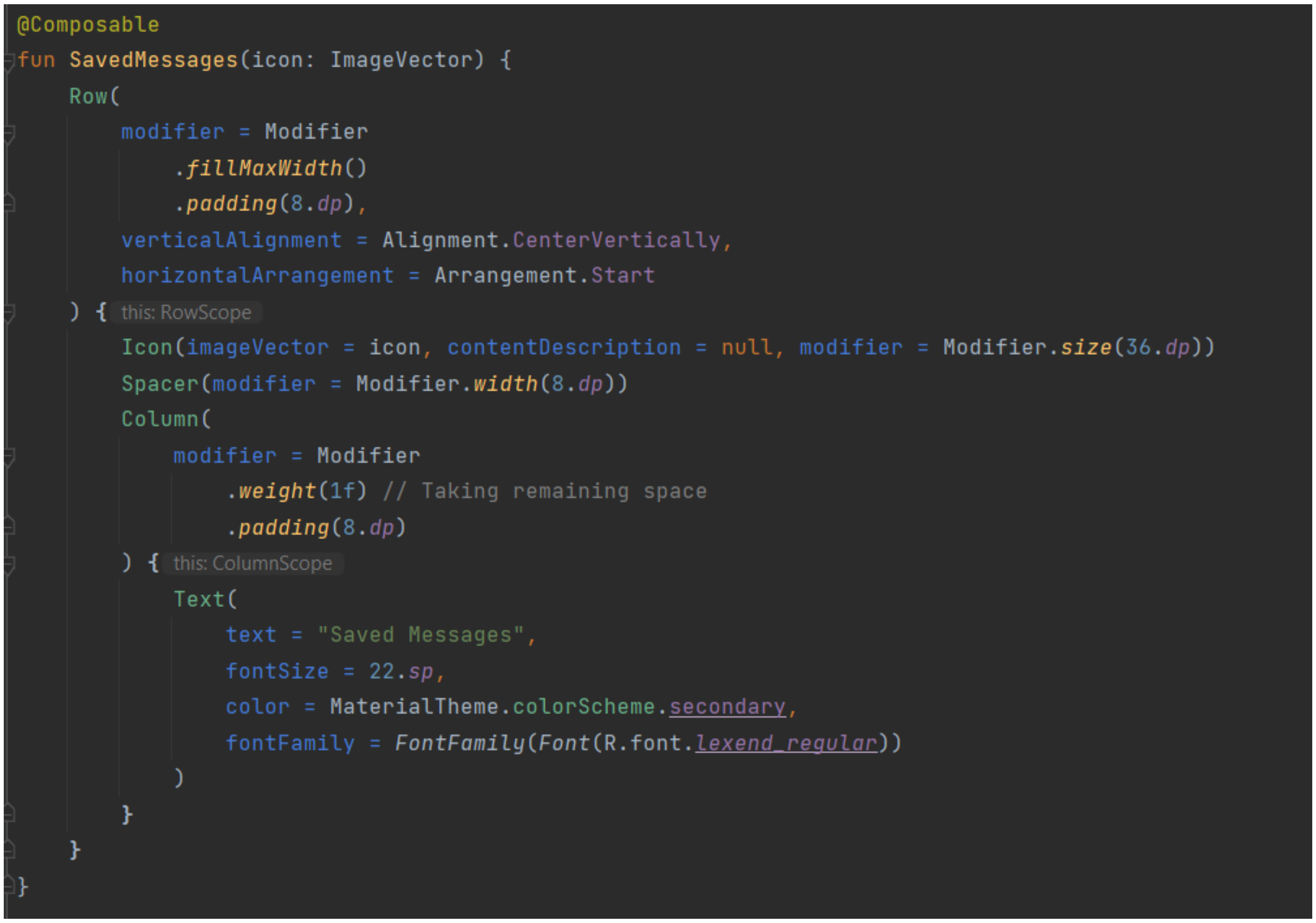


```kotlin
import android.content.Context
import arush.baatcheet.model.FileHandler
import arush.baatcheet.model.SaveMessageModel

 M4dar4
class SavedMessagePresenter (private val context: Context){
     M4dar4
    fun getSaveMessage(): ArrayList<SaveMessageModel>{
        return FileHandler(context).retrieveSavedMessage()
    }
}
```

The presenter is called in the view model as shown below.



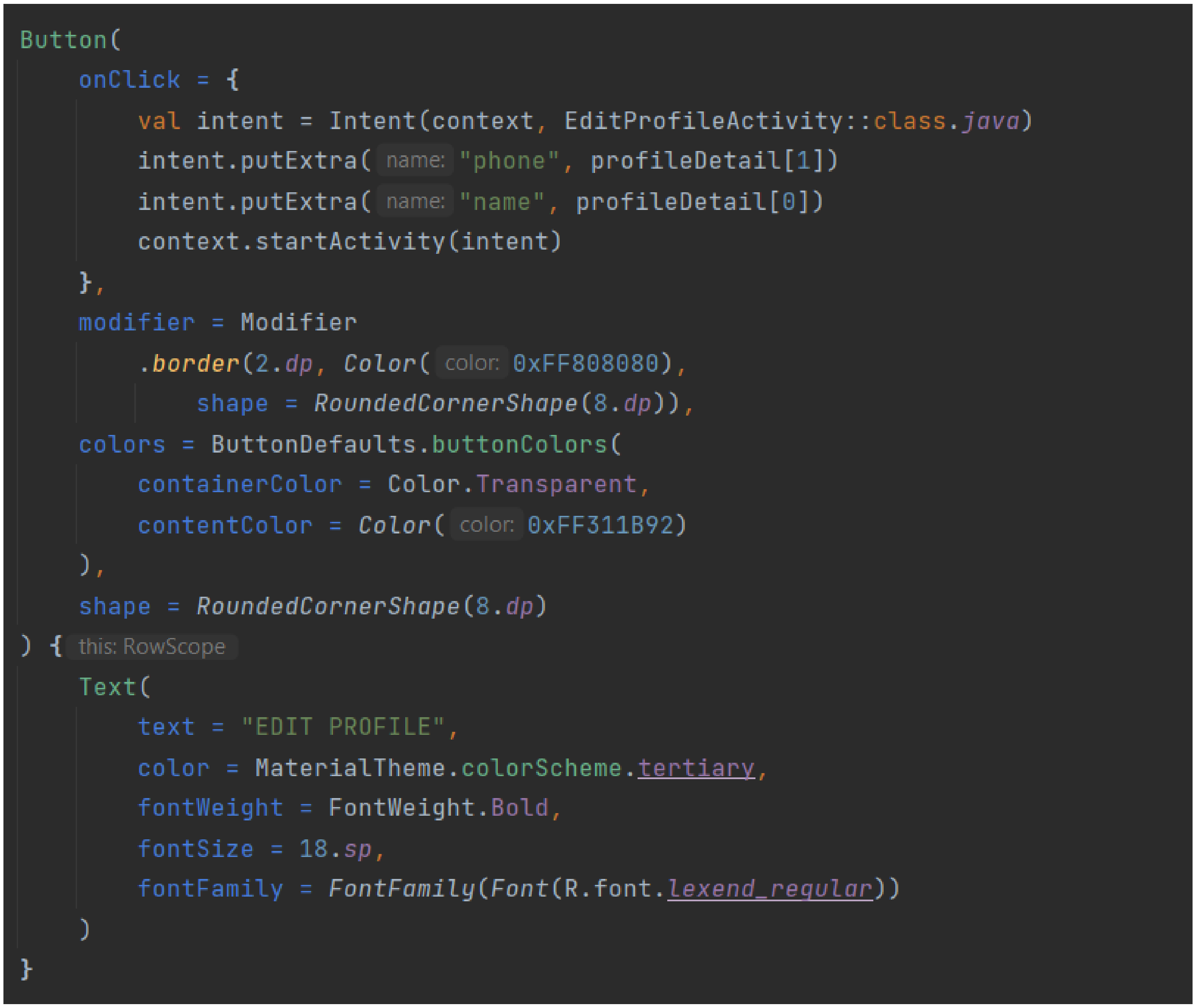
```kotlin
Box(
    modifier = Modifier
        .fillMaxWidth()
        .clickable {
            val intent = Intent(context, SavedMessagesActivity::class.java)
            context.startActivity(intent)
        }
) { this: BoxScope
    SavedMessages(icon = Icons.Default.Star)
}
```



```kotlin
@Composable
fun SavedMessages(icon: ImageVector) {
    Row(
        modifier = Modifier
            .fillMaxWidth()
            .padding(8.dp),
        verticalAlignment = Alignment.CenterVertically,
        horizontalArrangement = Arrangement.Start
    ) { this: RowScope
        Icon(imageVector = icon, contentDescription = null, modifier = Modifier.size(36.dp))
        Spacer(modifier = Modifier.width(8.dp))
        Column(
            modifier = Modifier
                .weight(1f) // Taking remaining space
                .padding(8.dp)
        ) { this: ColumnScope
            Text(
                text = "Saved Messages",
                fontSize = 22.sp,
                color = MaterialTheme.colorScheme.secondary,
                fontFamily = FontFamily(Font(R.font.lexend_regular))
            )
        }
    }
}
```

Activity 3: Edit Page redirection

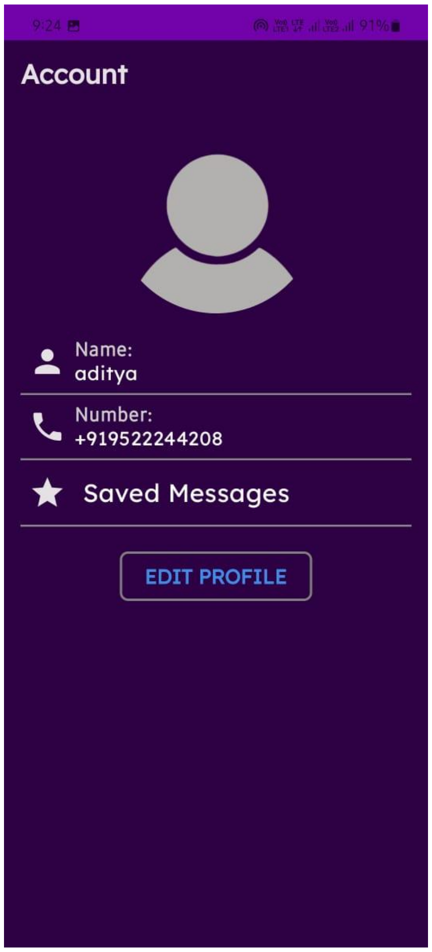
The user also has the ability to edit their profile. To implement this, the Edit Profile Activity is called in the Profile Activity as shown below.

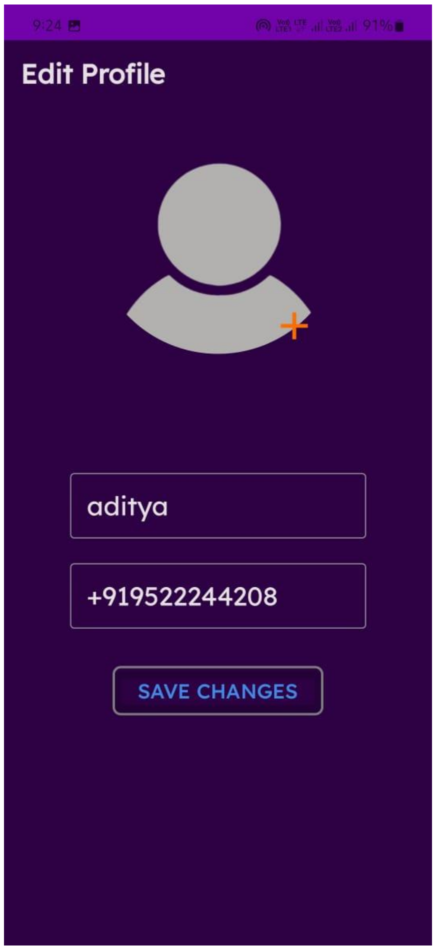


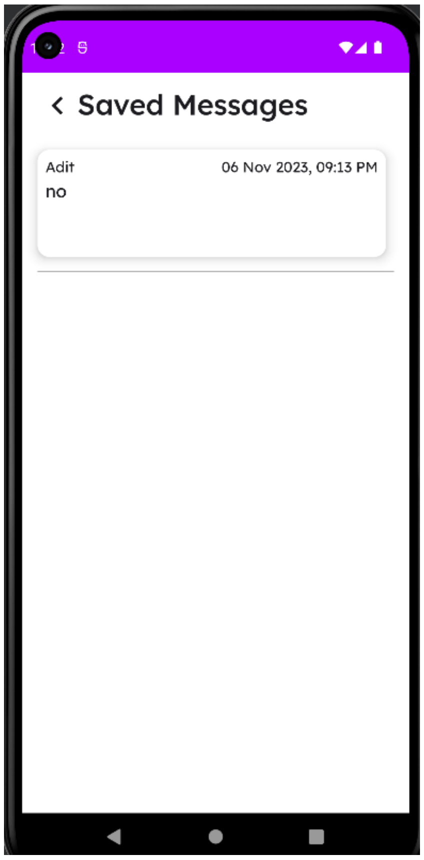
```kotlin
Button(
    onClick = {
        val intent = Intent(context, EditProfileActivity::class.java)
        intent.putExtra( name: "phone", profileDetail[1])
        intent.putExtra( name: "name", profileDetail[0])
        context.startActivity(intent)
    },
    modifier = Modifier
        .border(2.dp, Color( color: 0xFF808080),
            shape = RoundedCornerShape(8.dp)),
    colors = ButtonDefaults.buttonColors(
        containerColor = Color.Transparent,
        contentColor = Color( color: 0xFF311B92)
    ),
    shape = RoundedCornerShape(8.dp)
) { this: RowScope
    Text(
        text = "EDIT PROFILE",
        color = MaterialTheme.colorScheme.tertiary,
        fontWeight = FontWeight.Bold,
        fontSize = 18.sp,
        fontFamily = FontFamily(Font(R.font.lexend_regular))
    )
}
```

This redirects the user to the edit profile page where they can edit their profile.

Output:





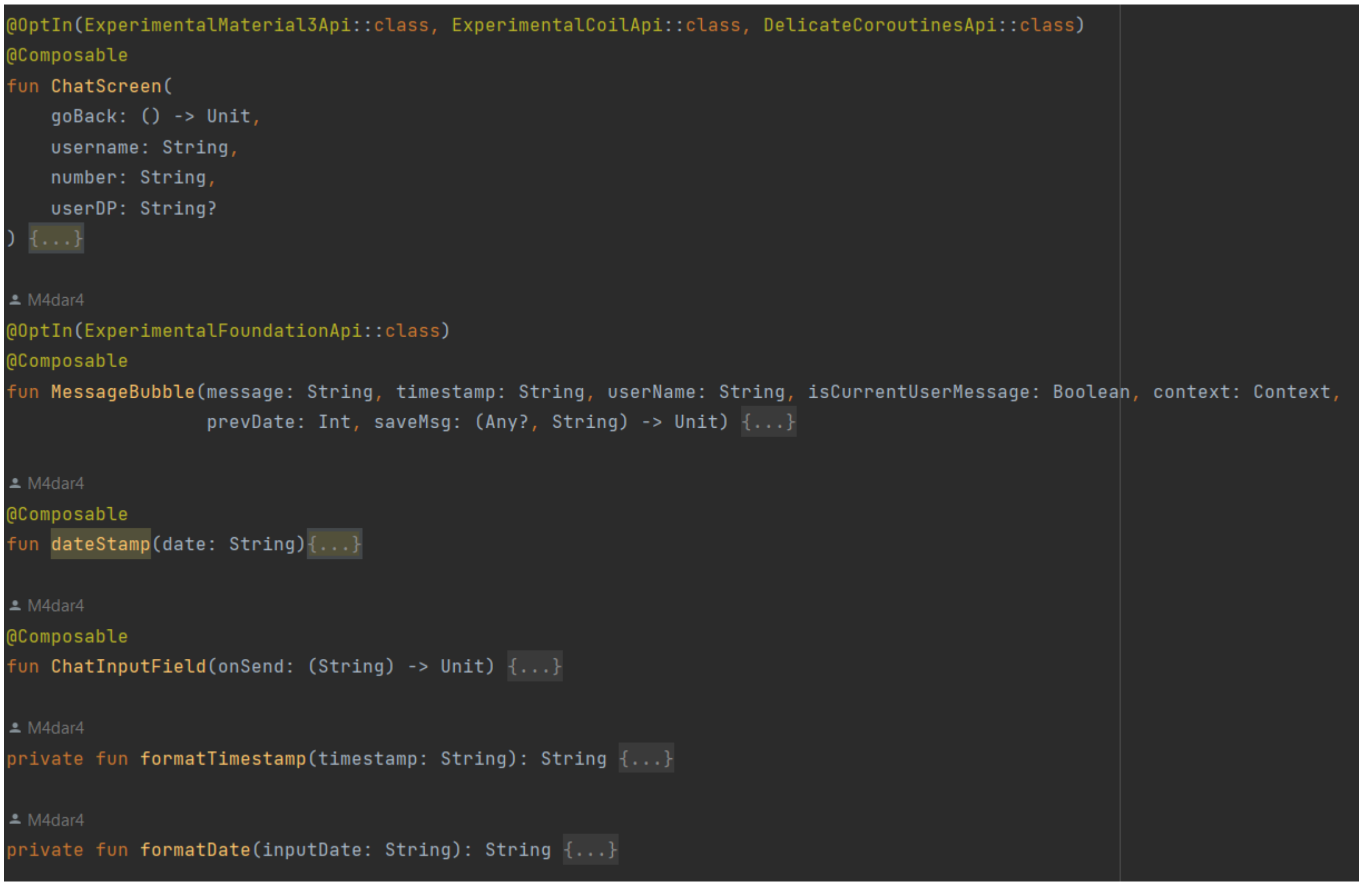


Adit 06 Nov 2023, 09:13 PM
no

Milestone 4: Chat Activity

Activity 1: Building Chat Screen UI

The most important feature was to build a user friendly chat screen, which allows users to interact with each other seamlessly. This was made possible as shown below.



```
@OptIn(ExperimentalMaterial3Api::class, ExperimentalCoilApi::class, DelicateCoroutinesApi::class)
@Composable
fun ChatScreen(
    goBack: () -> Unit,
    username: String,
    number: String,
    userDP: String?
) {...}

M4dar4
@OptIn(ExperimentalFoundationApi::class)
@Composable
fun MessageBubble(message: String, timestamp: String, userName: String, isCurrentUserMessage: Boolean, context: Context,
                  prevDate: Int, saveMsg: (Any?, String) -> Unit) {...}

M4dar4
@Composable
fun dateStamp(date: String){...}

M4dar4
@Composable
fun ChatInputField(onSend: (String) -> Unit) {...}

M4dar4
private fun formatTimestamp(timestamp: String): String {...}

M4dar4
private fun formatDate(inputDate: String): String {...}
```

These functions allow seamless interaction and implementation of chatting functionality.

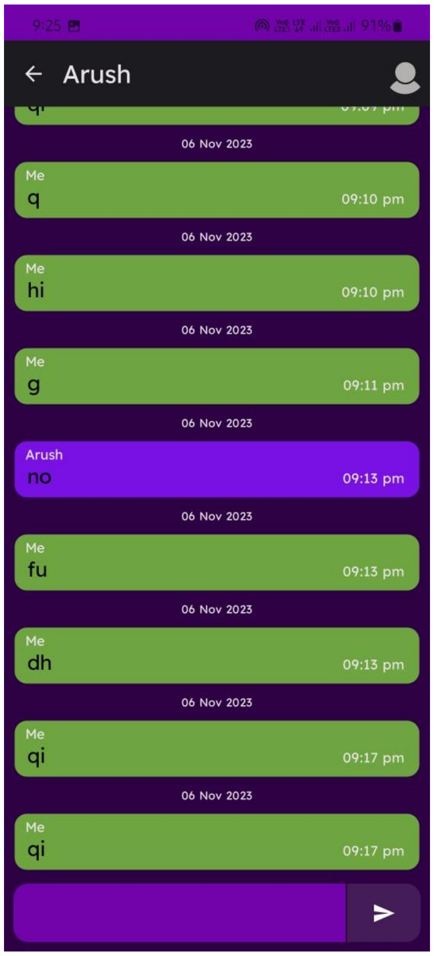
The MessageBubble function provides shape and structure to the chat context to the user interface.

dateStamp function signifies the change of day while communicating with each other.

ChatInputField function gives users the ability to input their messages or thoughts which they wish to communicate to others.

formatTimestamp and formatDate functions format those features which soothes the human eye.

Output:


Arush

06 Nov 2023

Me
q 09:10 pm

06 Nov 2023

Me
hi 09:10 pm

06 Nov 2023

Me
g 09:11 pm

06 Nov 2023

Arush
no 09:13 pm

06 Nov 2023

Me
fu 09:13 pm

06 Nov 2023

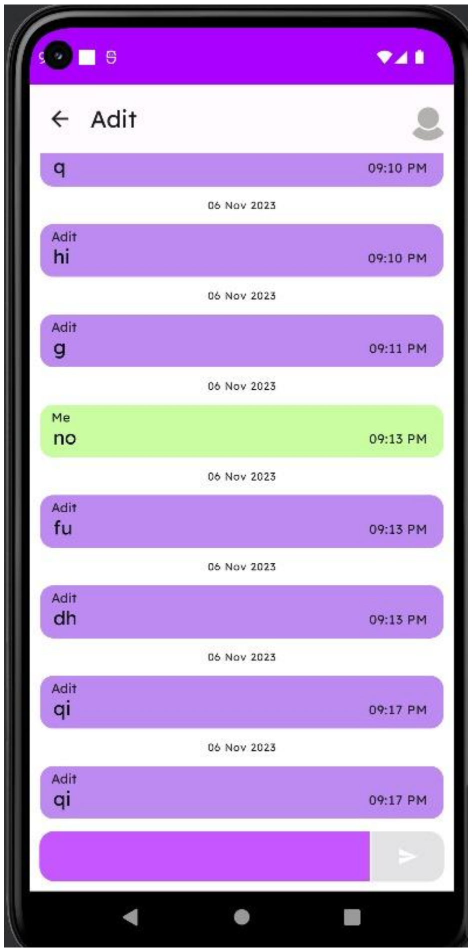
Me
dh 09:13 pm

06 Nov 2023

Me
qi 09:17 pm

06 Nov 2023

Me
qi 09:17 pm


Adit

q 09:10 PM

06 Nov 2023

Adit
hi 09:10 PM

06 Nov 2023

Adit
g 09:11 PM

06 Nov 2023

Me
no 09:13 PM

06 Nov 2023

Adit
fu 09:13 PM

06 Nov 2023

Adit
dh 09:13 PM

06 Nov 2023

Adit
qi 09:17 PM

06 Nov 2023

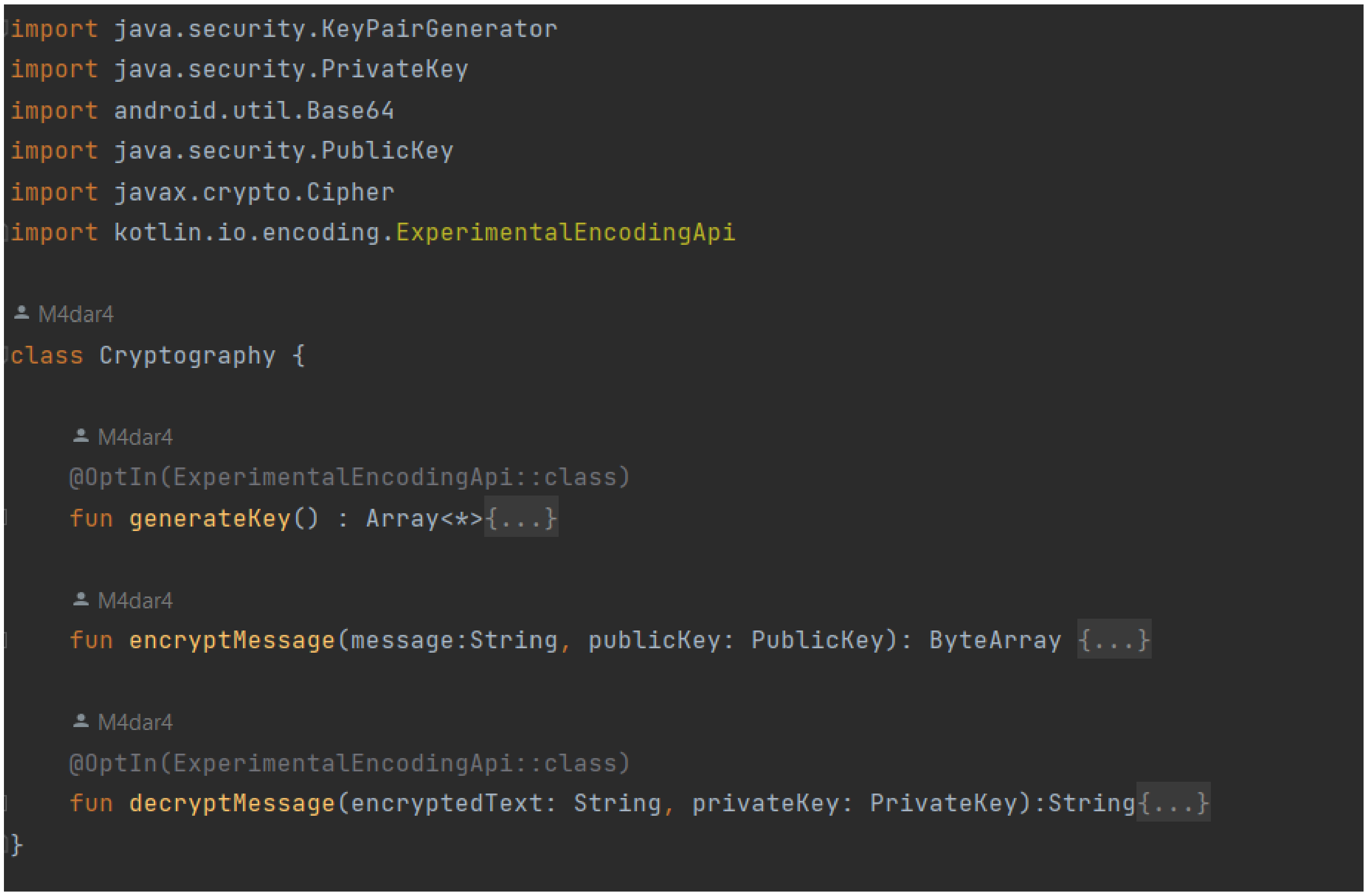
Adit
qi 09:17 PM

Milestone 5: Model Building

This milestone includes major backend functionalities which allow this app to function properly.

Activity 1: Encryption and Decryption

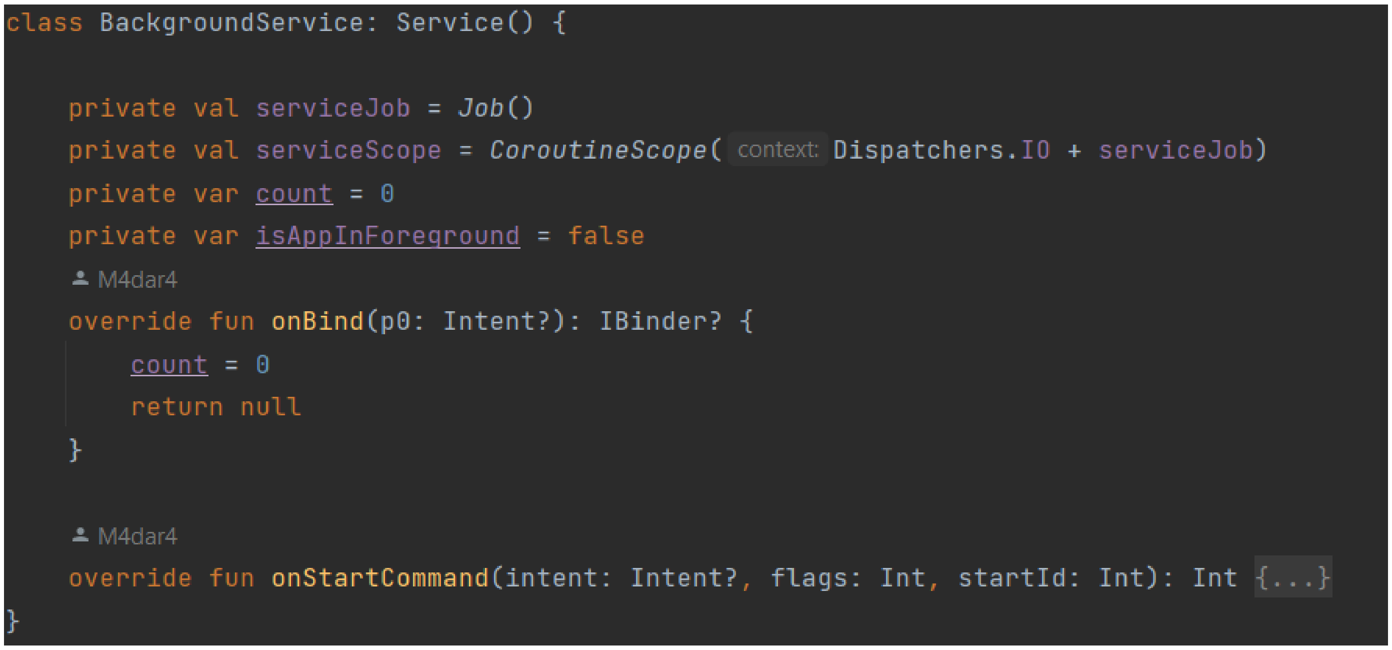
Encrypting and Decrypting data is important in today’s world and this model allows this as shown below.



```
import java.security.KeyPairGenerator
import java.security.PrivateKey
import android.util.Base64
import java.security.PublicKey
import javax.crypto.Cipher
import kotlin.io.encoding.ExperimentalEncodingApi

M4dar4
class Cryptography {

    M4dar4
    @OptIn(ExperimentalEncodingApi::class)
    fun generateKey() : Array<*>{...}

    M4dar4
    fun encryptMessage(message:String, publicKey: PublicKey): ByteArray {...}

    M4dar4
    @OptIn(ExperimentalEncodingApi::class)
    fun decryptMessage(encryptedText: String, privateKey: PrivateKey):String{...}
}
```

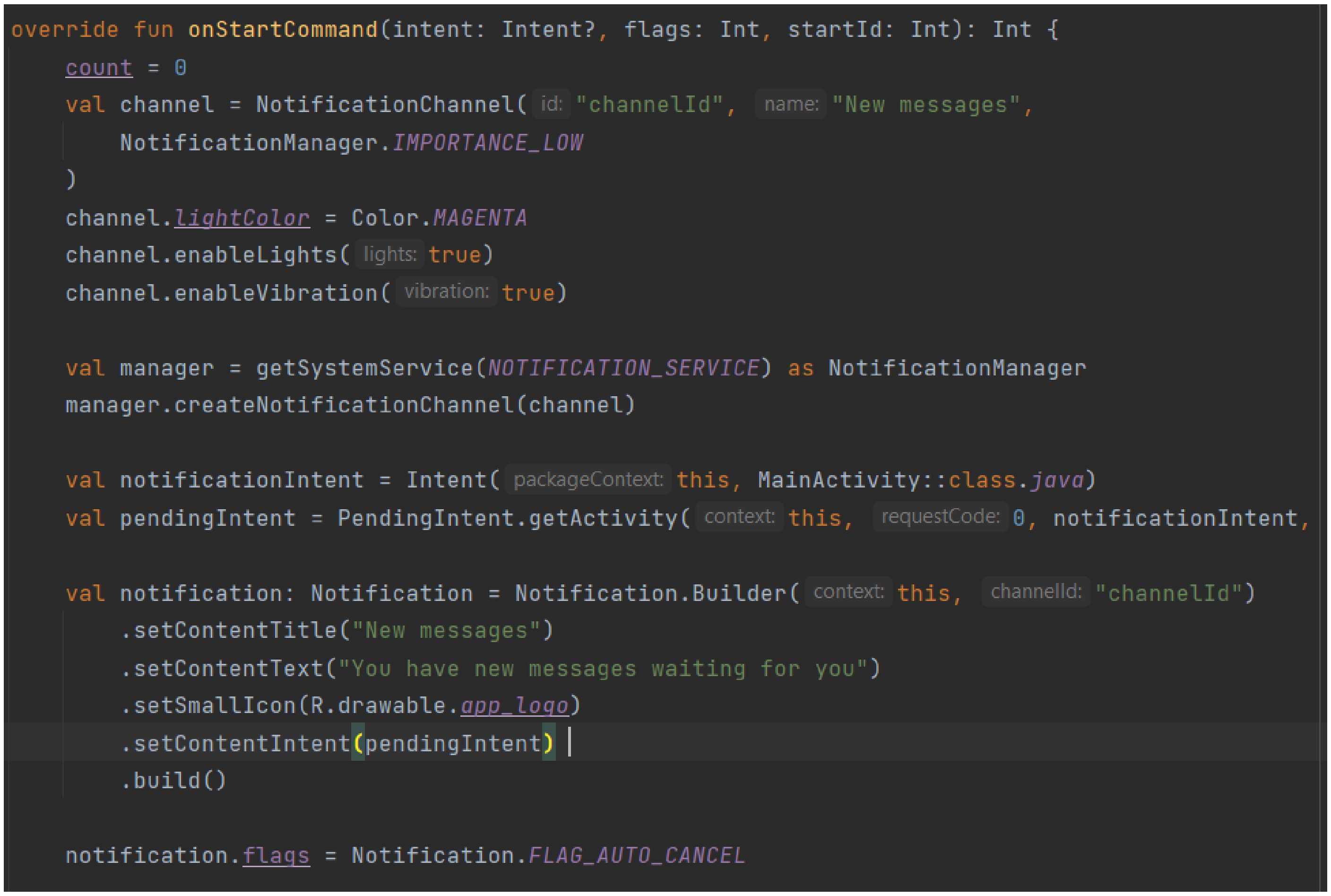
Activity 2: Background Services

Background services allow the app to run seamlessly without any lag and freezes. This code focuses mainly on keeping the important functions running in the background without the user needing to wait for the app.



```
class BackgroundService: Service() {

    private val serviceJob = Job()
    private val serviceScope = CoroutineScope( context: Dispatchers.IO + serviceJob)
    private var count = 0
    private var isAppInForeground = false
    M4dar4
    override fun onBind(p0: Intent?): IBinder? {
        count = 0
        return null
    }

    M4dar4
    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {...}
}
```

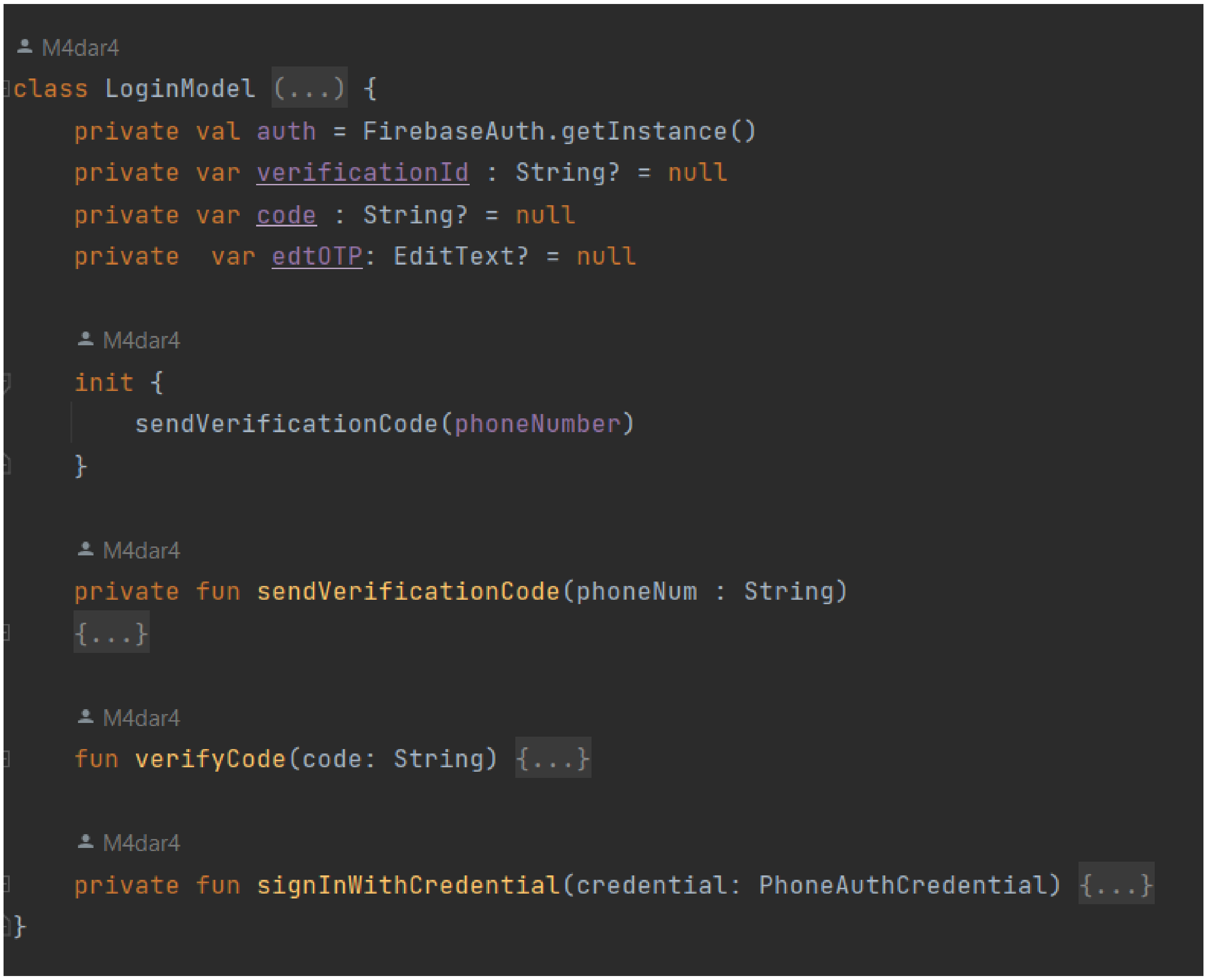
It also includes push notifications features as shown below.



```
override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
    count = 0
    val channel = NotificationChannel( id: "channelId",  name: "New messages",
        NotificationManager.IMPORTANCE_LOW
    )
    channel.lightColor = Color.MAGENTA
    channel.enableLights( lights: true)
    channel.enableVibration( vibration: true)

    val manager = getSystemService(NOTIFICATION_SERVICE) as NotificationManager
    manager.createNotificationChannel(channel)

    val notificationIntent = Intent( packageContext: this, MainActivity::class.java)
    val pendingIntent = PendingIntent.getActivity( context: this,  requestCode: 0, notificationIntent,

    val notification: Notification = Notification.Builder( context: this,  channelId: "channelId")
        .setContentTitle("New messages")
        .setContentText("You have new messages waiting for you")
        .setSmallIcon(R.drawable.app_logo)
        .setContentIntent(pendingIntent)
        .build()

    notification.flags = Notification.FLAG_AUTO_CANCEL
```

Activity 4: Login Model

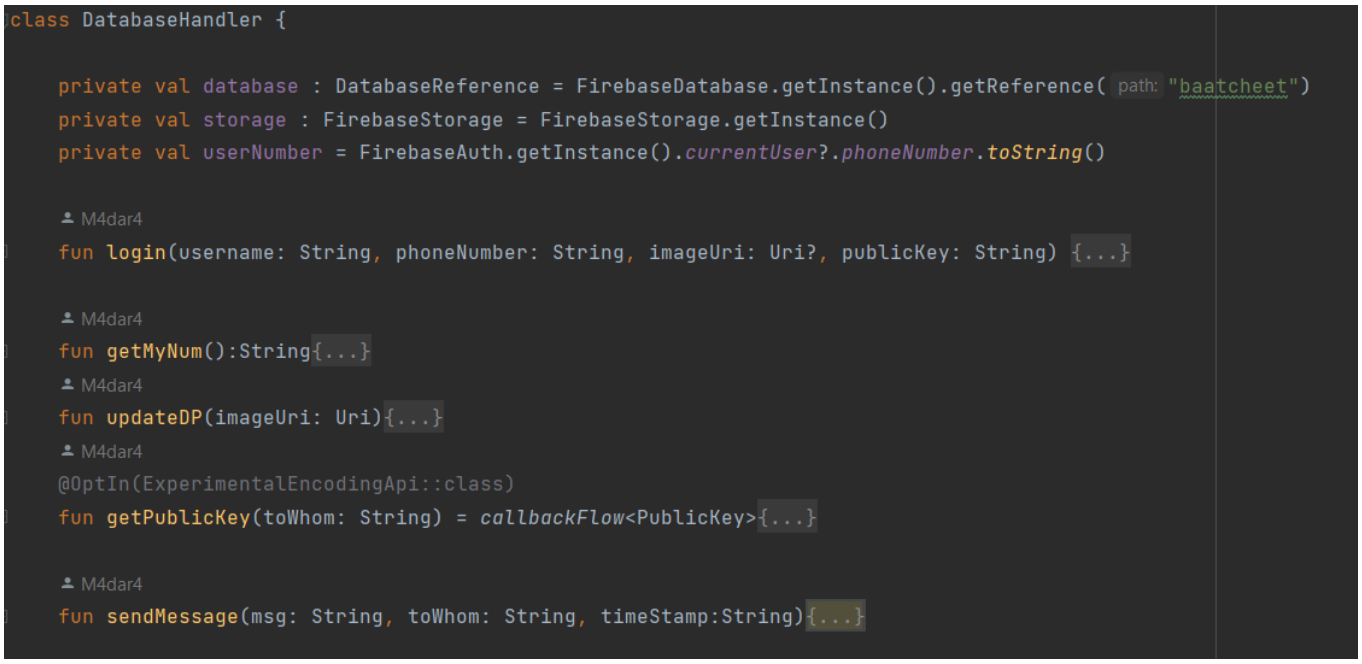
This model includes all logic behind authentication, verification and logging into the application as shown below.



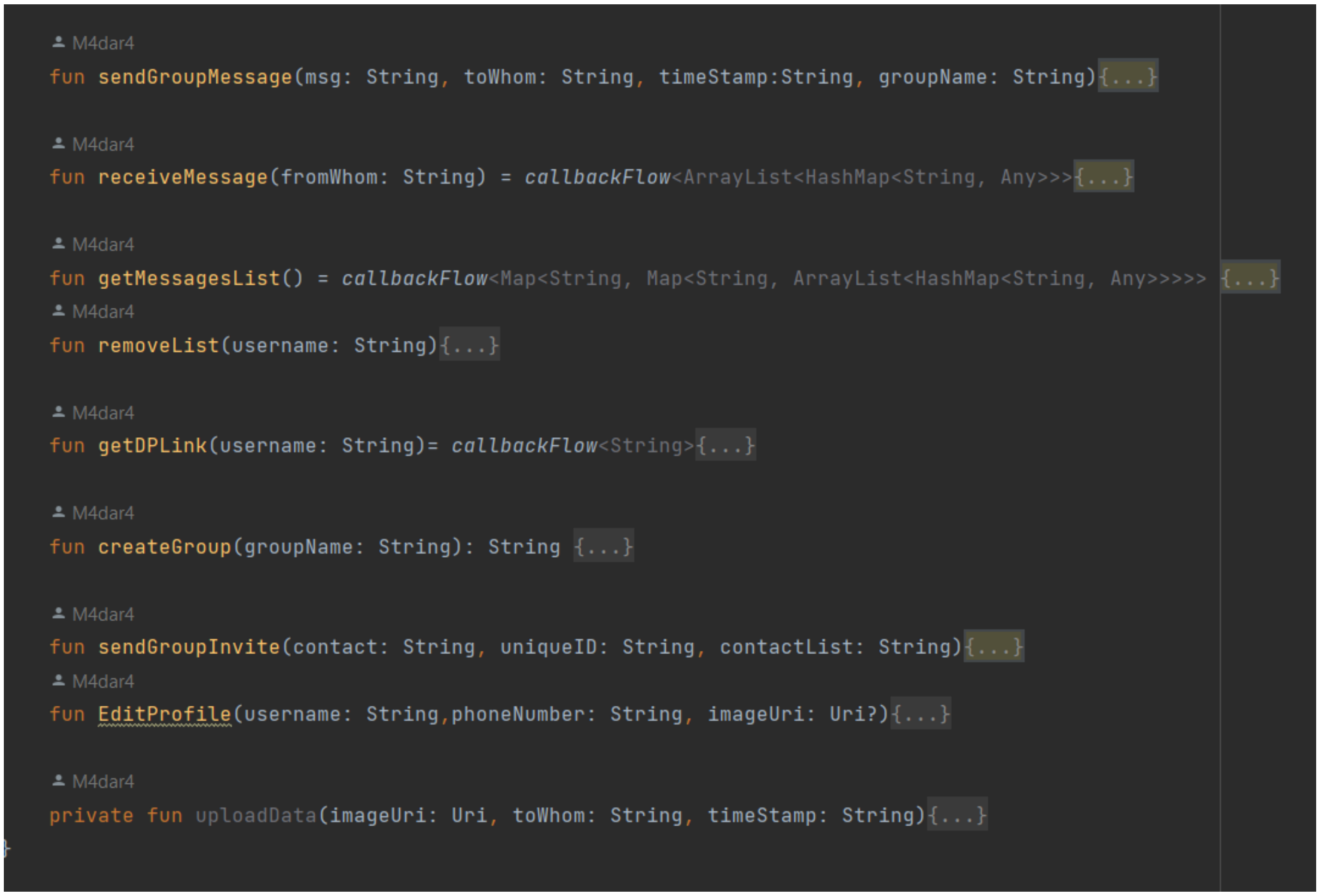
```
M4dar4
class LoginModel (...) {
    private val auth = FirebaseAuth.getInstance()
    private var verificationId : String? = null
    private var code : String? = null
    private  var edtOTP: EditText? = null

    M4dar4
    init {
        sendVerificationCode(phoneNumber)
    }

    M4dar4
    private fun sendVerificationCode(phoneNum : String)
    {...}

    M4dar4
    fun verifyCode(code: String) {...}

    M4dar4
    private fun signInWithCredential(credential: PhoneAuthCredential) {...}
}
```

Activity 5: Database Handling Model

This is an important part of the project as it communicates with the database to send and receive data from it. The class DatabaseHandler includes all the shown functions below.



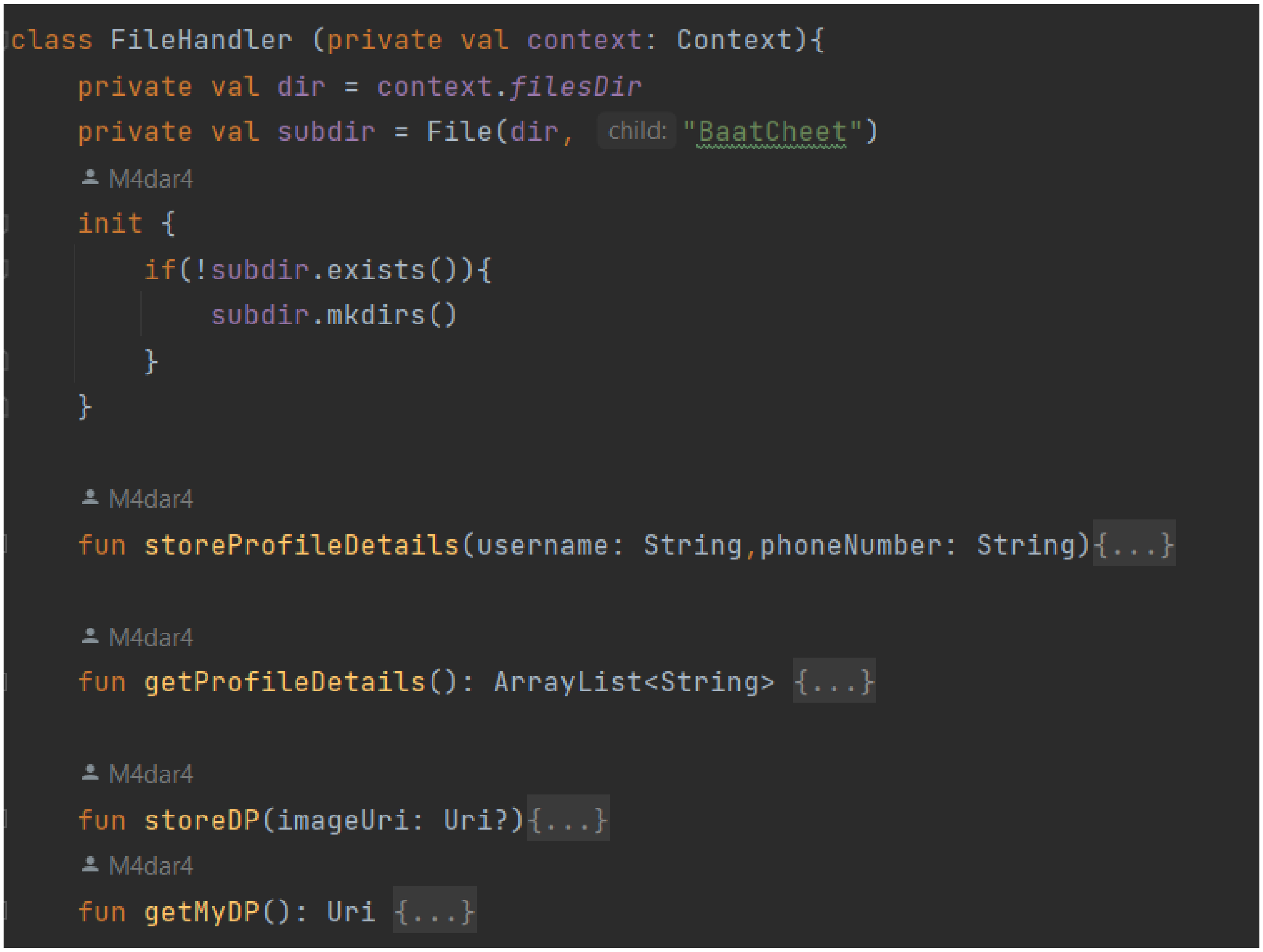
```
class DatabaseHandler {

    private val database : DatabaseReference = FirebaseDatabase.getInstance().getReference( path: "baatcheet")
    private val storage : FirebaseStorage = FirebaseStorage.getInstance()
    private val userNumber = FirebaseAuth.getInstance().currentUser?.phoneNumber.toString()

    M4dar4
    fun login(username: String, phoneNumber: String, imageUri: Uri?, publicKey: String) {...}

    M4dar4
    fun getMyNum():String{...}
    M4dar4
    fun updateDP(imageUri: Uri){...}
    M4dar4
    @OptIn(ExperimentalEncodingApi::class)
    fun getPublicKey(toWhom: String) = callbackFlow<PublicKey>{...}

    M4dar4
    fun sendMessage(msg: String, toWhom: String, timeStamp:String){...}
```



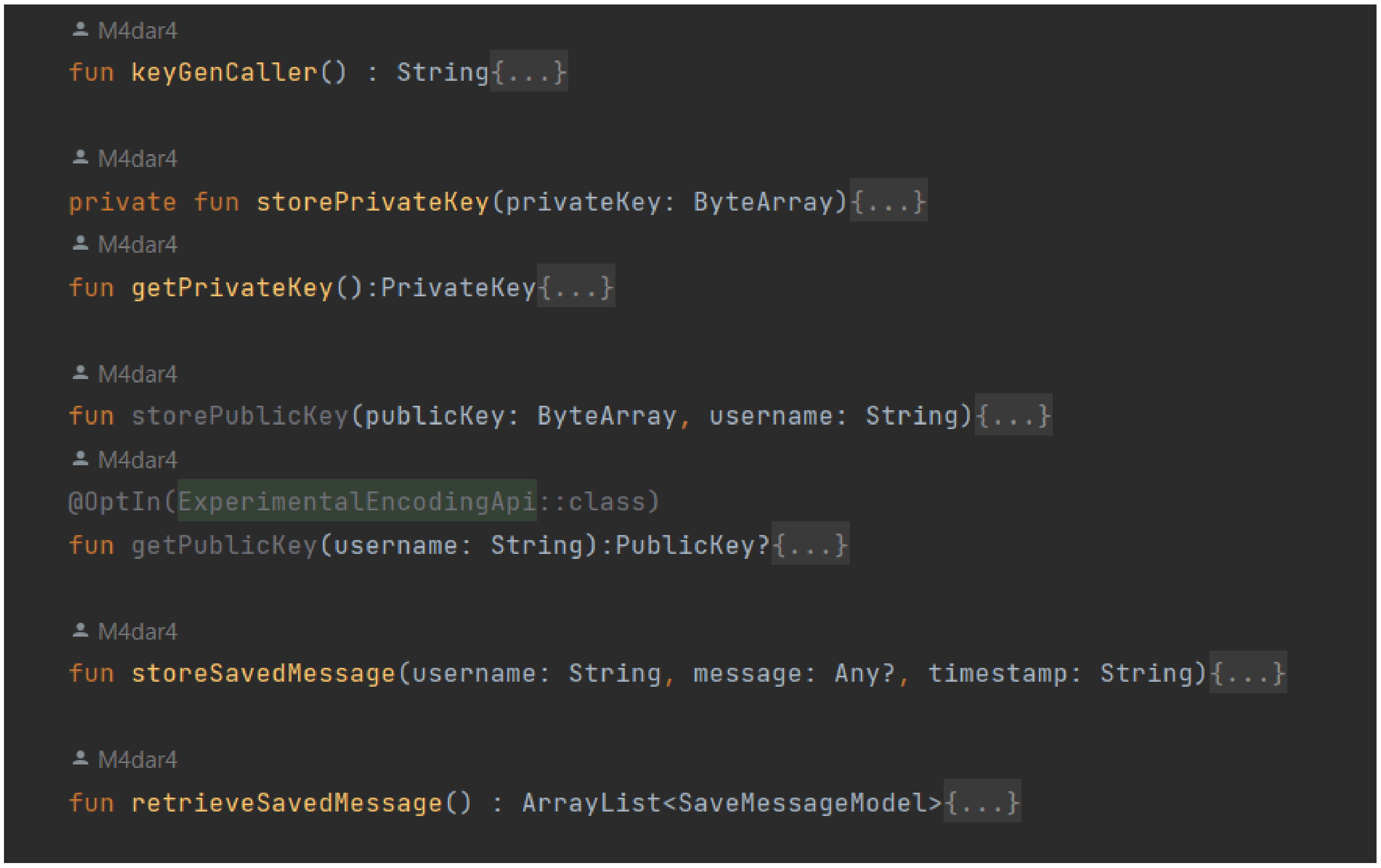
```
    M4dar4
    fun sendGroupMessage(msg: String, toWhom: String, timeStamp:String, groupName: String){...}

    M4dar4
    fun receiveMessage(fromWhom: String) = callbackFlow<ArrayList<HashMap<String, Any>>>{...}

    M4dar4
    fun getMessagesList() = callbackFlow<Map<String, Map<String, ArrayList<HashMap<String, Any>>>>> {...}
    M4dar4
    fun removeList(username: String){...}

    M4dar4
    fun getDPLink(username: String)= callbackFlow<String>{...}

    M4dar4
    fun createGroup(groupName: String): String {...}

    M4dar4
    fun sendGroupInvite(contact: String, uniqueID: String, contactList: String){...}
    M4dar4
    fun EditProfile(username: String,phoneNumber: String, imageUri: Uri?){...}

    M4dar4
    private fun uploadData(imageUri: Uri, toWhom: String, timeStamp: String){...}
}
```

Activity 6: File Handling Model

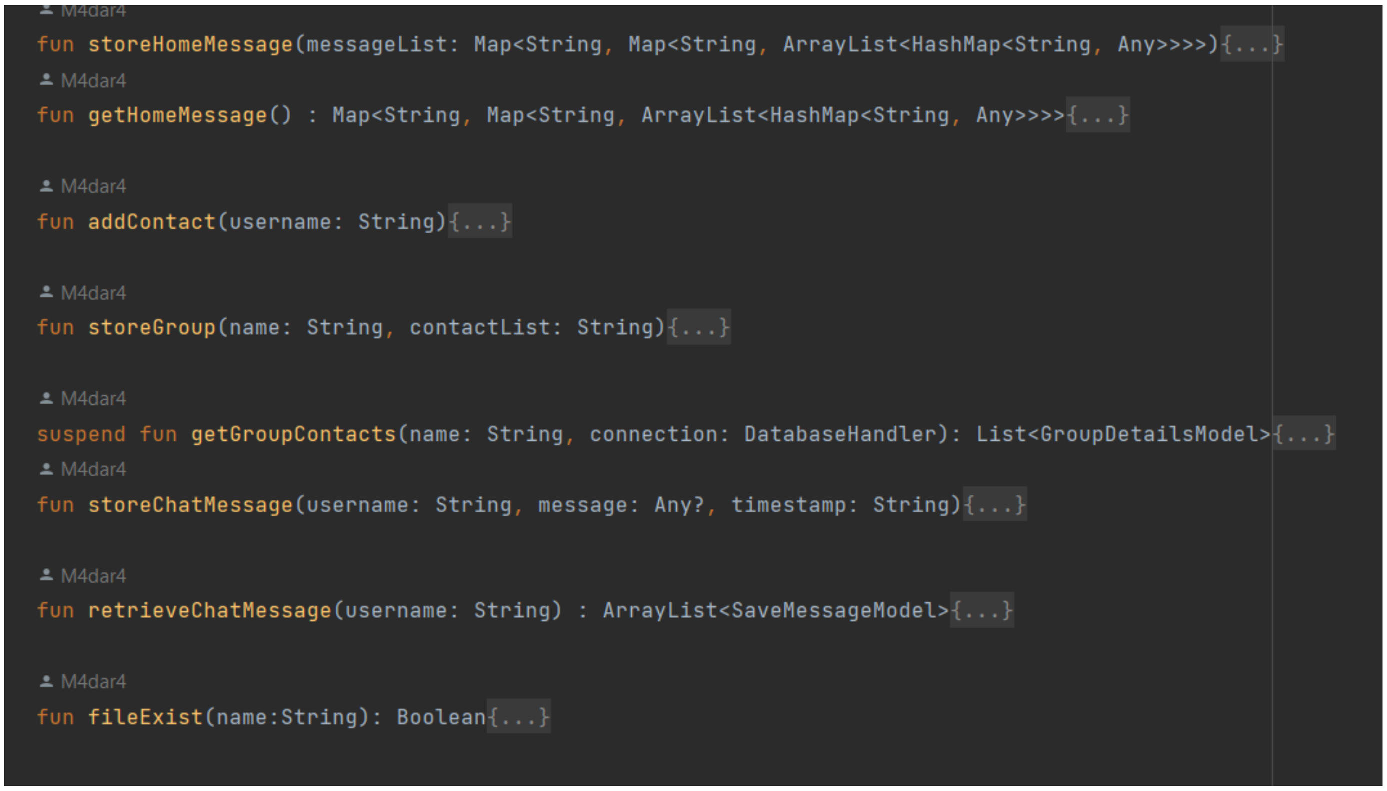
File Handler is also an important part of this project. Some of the functions it provides are shown below.



```
class FileHandler (private val context: Context){
    private val dir = context.filesDir
    private val subdir = File(dir,  child: "BaatCheet")
    M4dar4
    init {
        if(!subdir.exists()){
            subdir.mkdirs()
        }
    }

    M4dar4
    fun storeProfileDetails(username: String,phoneNumber: String){...}

    M4dar4
    fun getProfileDetails(): ArrayList<String> {...}

    M4dar4
    fun storeDP(imageUri: Uri?){...}
    M4dar4
    fun getMyDP(): Uri {...}
```



```
M4dar4
fun keyGenCaller() : String{...}

M4dar4
private fun storePrivateKey(privateKey: ByteArray){...}
M4dar4
fun getPrivateKey():PrivateKey{...}

M4dar4
fun storePublicKey(publicKey: ByteArray, username: String){...}
M4dar4
@OptIn(ExperimentalEncodingApi::class)
fun getPublicKey(username: String):PublicKey?{...}

M4dar4
fun storeSavedMessage(username: String, message: Any?, timestamp: String){...}

M4dar4
fun retrieveSavedMessage() : ArrayList<SaveMessageModel>{...}
```



```
M4dar4
fun storeHomeMessage(messageList: Map<String, Map<String, ArrayList<HashMap<String, Any>>>>){...}
M4dar4
fun getHomeMessage() : Map<String, Map<String, ArrayList<HashMap<String, Any>>>>{...}

M4dar4
fun addContact(username: String){...}

M4dar4
fun storeGroup(name: String, contactList: String){...}

M4dar4
suspend fun getGroupContacts(name: String, connection: DatabaseHandler): List<GroupDetailsModel>{...}
M4dar4
fun storeChatMessage(username: String, message: Any?, timestamp: String){...}

M4dar4
fun retrieveChatMessage(username: String) : ArrayList<SaveMessageModel>{...}

M4dar4
fun fileExist(name:String): Boolean{...}
```