International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Development and Implementation of an Efficient Chatting Application for Mobile Devices

Bhavana A[1], *Gaurav Chauhan*[2]

[1]Computer Science and Engineering Department Presidency University Bangalore, India bhavana@presidencyuniversity.in
[2]Computer Engineering and Data Science Presidency University Bangalore, India 201910100532@presidencyuniversity.in

DOI- https://doi.org/10.55248/gengpi.4.523.43343

ABSTRACT—

This paper presents the development and implementation of a chat application called "The Inner Circle." The application provides a user-friendly interface that displays texts sent by two individuals in different locations, with each person's messages positioned on the right or left side of the chat container, respectively. The messages are arranged in chronological order, enabling users to follow the conversation seamlessly. Additionally, the app allows users to log in, add contacts, and view the online status of other users to facilitate effective communication.

Keywords—Mobile application platform, User friendly Interface, Inner circle.

I. Introduction

The use of chat applications for written communication has become increasingly popular, allowing people to easily and quickly exchange messages regardless of their location. Lot of people shy away from using social media and any other platform used for communication due to a number of reasons:

Privacy: Many people are apprehensive about sharing personal information online and worry about the potential misuse or unauthorized access to their data. High-profile incidents of data breaches and privacy violations have contributed to these concerns.

Cyberbullying and harassment: Social media platforms can become breeding grounds for cyberbullying and harassment. Some individuals may have experienced or witnessed such behavior, leading them to avoid these platforms to protect their mental well-being.

Fear of addiction and time-wasting: With the launch of so many features on the existing chatting apps it is hard for people to not get addicted to them. Some people may choose to avoid these platforms to prevent themselves from becoming overly dependent on them or to focus on more productive activities.

Comparisons and self-esteem issues: Constant exposure to carefully curated highlight reels or stories on social media can lead to feelings of inadequacy or low self-esteem. People may avoid these platforms to protect their mental health and maintain a positive self-image.

Thus, a lot of factors contribute to the social cost of asking questions [1]. On the other side of the spectrum, there are some problems associated with allowing any user to submit answers to questions:

Information overload and distraction: The constant stream of information, updates, and notifications on social media platforms can be overwhelming and distracting. Some individuals may prefer to maintain a calmer and more focused environment by staying away from these platforms.

Security concerns: Allowing anyone to be able to connect can compromise the safety and privacy of users.

While solutions to these problems can be difficult to implement for a large-scale The Inner-circle application, a small scale platform meant to be used at primary levels in a group or organization with the following simple features can provide satisfactory results:

One time sign-in for users: Allowing anybody in a group to connect with the people they care and communicate effectively. Here the key focus is going to be their chat, the communication they want to have without getting influenced by the extra features which divide their attention. Since the platform is meant to be deployed and made available to small departments instead of the whole world, developing an application with only the necessary features is important.

Only let verified users connect: Allowing only verified users improves the quality of the user experience and conversations prevents issues like distraction and sharing inappropriate content by unverified users.

II. Technologies used

For building the Inner- Circle chatting application as described in the introduction we essentially need to do mobile application development.

A. Mobile Application Development.

It involves creating software applications for mobile devices, utilizing network connections and remote computing resources. Modern smartphones offer various features such as Bluetooth, NFC, GPS, sensors, and cameras, which developers can leverage to create innovative apps. The success of mobile applications lies in their ability to utilize these features effectively.:

Mobile App: Mobile apps differ from integrated software systems found on PCs, providing limited and isolated functionality. Each app serves a specific purpose, such as gaming, calculations, or web browsing. This specificity allows users to customize their devices according to their preferences and needs.

Mobile OS: A mobile operating system (OS) enables the execution of application software on mobile devices. It functions similarly to computer operating systems but is lightweight and simplified. Developers use different software development kits (SDKs) and tools for development.

Android Studio : Android Studio is an integrated development environment (IDE) used to build Android applications. It provides tools for building, testing, and debugging apps across various devices. Features of Android Studio include a fast emulator, support for Gradle, code templates, GitHub integration, Google Cloud Platform support, and extensive tools and frameworks..

Android application components : Android application components are building blocks for creating Android apps. These components include activities, services, broadcast receivers, and content providers. Additional components like fragments, views, layouts, intents, resources, and the manifest file contribute to the overall functionality and user interface of the app.

III. System Design

A. Application Components of the Project

Application components are core building blocks of an android application. It is an entry point for system or users from which they can enter in app. The project consists of the following components:

Activities : An activity represents a single screen with a user interface, in-short Activity performs actions on the screen. The project consists of 7 activities with “SignInActivity” being the first screen visible to user.

The other activities can be visited according to following flowchart:

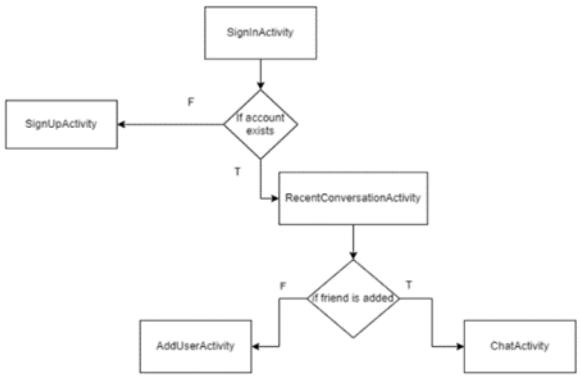


Fig 1 : Flow Between Activities.

Views View is the basic building block of UI (User Interface) in android. View refers to the android.view.View class, which is the super class for all the GUI components. The app contains Text View, Image View, Button, Edit Text, List View, Progress Bar, Card View which helps in achieving the flow of Activities and design the layout responsively.

External Views are installed like RoundIcon for round icon and SSP for scalable size units according to device characteristics.

- Bindings: Bindings is a feature which allows to more easily write code that interacts with views. Once view binding is enabled in a module, it generates a binding class for each XML layout file present in that module. An instance of a binding class contains direct references to all views that have an ID in the corresponding layout. This allows us to avoid findViewById as all the views are directly accessible.

- Layouts: Android Layout is used to define the user interface that holds the UI controls or widgets that will appear on the screen of an android application or activity screen. The app is designed using Constraint Layout, Linear Layout, ScrollView layout. ScrollView Layout allows to have more customizable options on scrolling options.
- Intents: The app communicates or transfers the control and information from one Activity to other Activity with the help of intents.

IV. User Interface Design

The user interface (UI) for an Android app is built as a hierarchy of layouts and widgets.

The layouts are View Group objects, containers that control how their child views are positioned on the screen. Widgets are View objects, UI components such as buttons and text boxes.

The Application is designed using the components of android such as, Layouts, Buttons, Text Views, Edit Texts, Card Layouts etc.

Following flowchart gives brief description and design of application and its flow:

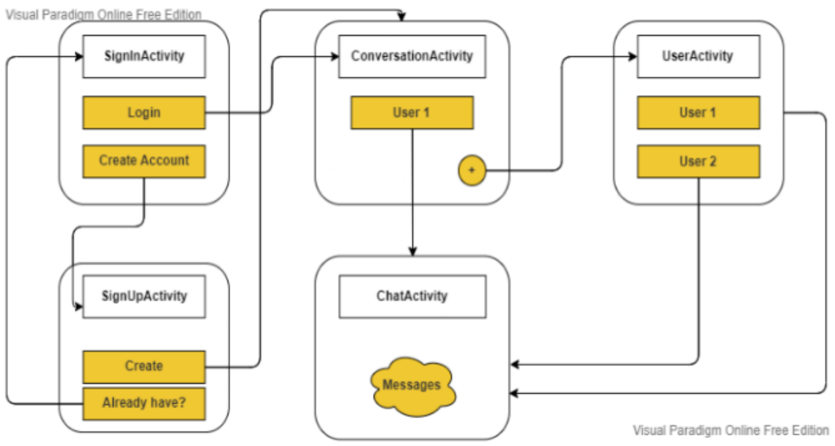


Figure 1: The design of Application

V. Application walkthrough

On click on Sign Up button with User name, valid Email, Valid Password and Valid reconfirm password, account Successfully gets created and redirected to home page.

On click Already Registered? Login Here, Redirects to Sign In activity.

Tapping on floating + (Add) Button on Main Activity, Redirects to User Activity.

Tapping on Users on User Activity, Starts a new conversation with them.

Typing in the Send Message field and tapping Send in Chat Activity, Sends a new chat message displayed on right side.

Receiving a new chat from user in Chat Activity, Receives chat is displayed on the left side.

Tapping on a user from recent conversations in Main Activity, Opens an existing conversation with them.

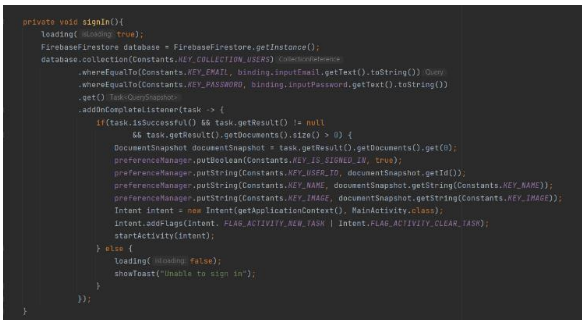
Opening a conversation when user is online, Shows the online indicator.

VI. Implementation details

This section will give a high-level view of how we built the application the inner-circle using the architecture described in the previous section. This section will contain explanation of the modules with Java Code and XML code.

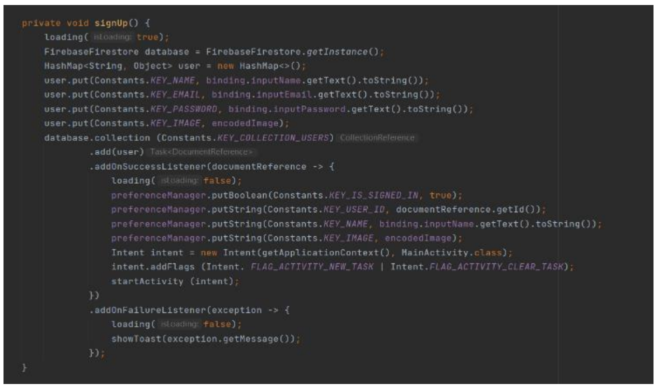
A. Java Code

1) SignInActivity.java This file contains the code to for the home page. It allows users to sign in if they have account otherwise it also contains link to other activity such as SignUpActivity. This file makes use of Intent to connect to other activity on click of the respective button.



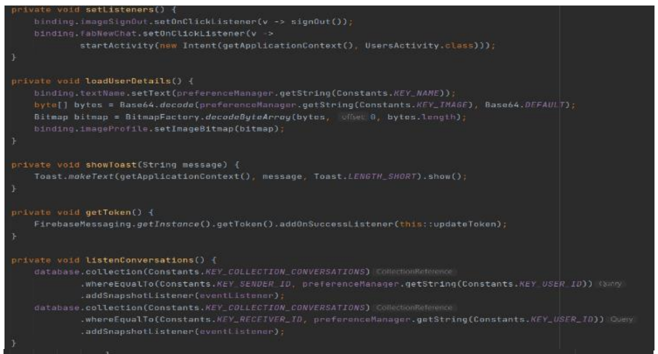
```
private void signIn(){
    loading( isLoading: true);
    FirebaseFirestore database = FirebaseFirestore.getInstance();
    database.collection(Constants.KEY_COLLECTION_USERS) CollectionReference
            .whereEqualTo(Constants.KEY_EMAIL, binding.inputEmail.getText().toString()) Query
            .whereEqualTo(Constants.KEY_PASSWORD, binding.inputPassword.getText().toString())
            .get() Task<QuerySnapshot>
            .addOnCompleteListener(task -> {
                if(task.isSuccessful() && task.getResult() != null
                        && task.getResult().getDocuments().size() > 0) {
                    DocumentSnapshot documentSnapshot = task.getResult().getDocuments().get(0);
                    preferenceManager.putBoolean(Constants.KEY_IS_SIGNED_IN, true);
                    preferenceManager.putString(Constants.KEY_USER_ID, documentSnapshot.getId());
                    preferenceManager.putString(Constants.KEY_NAME, documentSnapshot.getString(Constants.KEY_NAME));
                    preferenceManager.putString(Constants.KEY_IMAGE, documentSnapshot.getString(Constants.KEY_IMAGE));
                    Intent intent = new Intent(getApplicationContext(), MainActivity.class);
                    intent.addFlags(Intent. FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
                    startActivity(intent);
                } else {
                    loading( isLoading: false);
                    showToast("Unable to sign in");
                }
            });
}
```

2) SignUpActivity.java This file takes in data from the user which includes, name, profile picture, email and password to sign up. The email and password are sent to the Firebase for storing purpose.

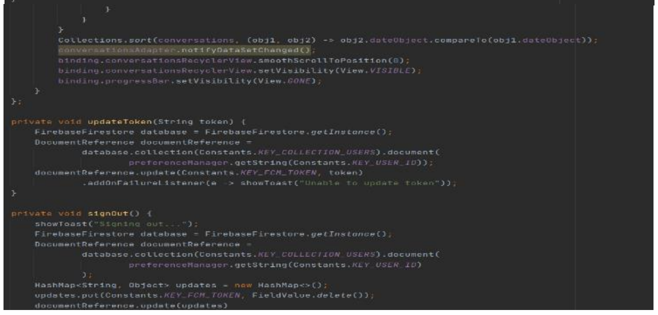


```
private void signUp() {
    loading( isLoading: true);
    FirebaseFirestore database = FirebaseFirestore.getInstance();
    HashMap<String, Object> user = new HashMap<>();
    user.put(Constants.KEY_NAME, binding.inputName.getText().toString());
    user.put(Constants.KEY_EMAIL, binding.inputEmail.getText().toString());
    user.put(Constants.KEY_PASSWORD, binding.inputPassword.getText().toString());
    user.put(Constants.KEY_IMAGE, encodedImage);
    database.collection (Constants.KEY_COLLECTION_USERS) CollectionReference
            .add(user) Task<DocumentReference>
            .addOnSuccessListener(documentReference -> {
                loading( isLoading: false);
                preferenceManager.putBoolean(Constants.KEY_IS_SIGNED_IN, true);
                preferenceManager.putString(Constants.KEY_USER_ID, documentReference.getId());
                preferenceManager.putString(Constants.KEY_NAME, binding.inputName.getText().toString());
                preferenceManager.putString(Constants.KEY_IMAGE, encodedImage);
                Intent intent = new Intent(getApplicationContext(), MainActivity.class);
                intent.addFlags (Intent. FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
                startActivity (intent);
            })
            .addOnFailureListener(exception -> {
                loading( isLoading: false);
                showToast(exception.getMessage());
            });
}
```

3) MainActivity.java This file loads up on signup. It contains all the recent chat with people and option to add new users along with text existing users.

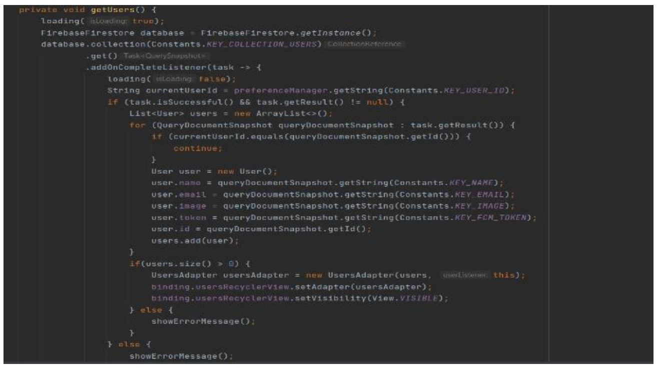


```
private void setListeners() {
    binding.imageSignOut.setOnClickListener(v -> signOut());
    binding.fabNewChat.setOnClickListener(v ->
            startActivity(new Intent(getApplicationContext(), UsersActivity.class)));
}

private void loadUserDetails() {
    binding.textName.setText(preferenceManager.getString(Constants.KEY_NAME));
    byte[] bytes = Base64.decode(preferenceManager.getString(Constants.KEY_IMAGE), Base64.DEFAULT);
    Bitmap bitmap = BitmapFactory.decodeByteArray(bytes,  offset: 0, bytes.length);
    binding.imageProfile.setImageBitmap(bitmap);
}

private void showToast(String message) {
    Toast.makeText(getApplicationContext(), message, Toast.LENGTH_SHORT).show();
}

private void getToken() {
    FirebaseMessaging.getInstance().getToken().addOnSuccessListener(this::updateToken);
}

private void listenConversations() {
    database.collection(Constants.KEY_COLLECTION_CONVERSATIONS) CollectionReference
            .whereEqualTo(Constants.KEY_SENDER_ID, preferenceManager.getString(Constants.KEY_USER_ID)) Query
            .addSnapshotListener(eventListener);
    database.collection(Constants.KEY_COLLECTION_CONVERSATIONS) CollectionReference
            .whereEqualTo(Constants.KEY_RECEIVER_ID, preferenceManager.getString(Constants.KEY_USER_ID)) Query
            .addSnapshotListener(eventListener);
}
```



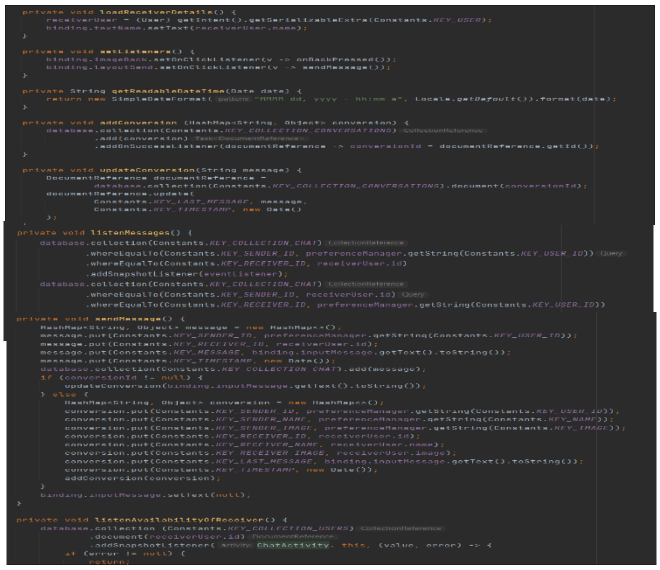
```
                }
            }
        }
        Collections.sort(conversations, (obj1, obj2) -> obj2.dateObject.compareTo(obj1.dateObject));
        conversationsAdapter.notifyDataSetChanged();
        binding.conversationsRecyclerView.smoothScrollToPosition(0);
        binding.conversationsRecyclerView.setVisibility(View.VISIBLE);
        binding.progressBar.setVisibility(View.GONE);
    }
};

private void updateToken(String token) {
    FirebaseFirestore database = FirebaseFirestore.getInstance();
    DocumentReference documentReference =
            database.collection(Constants.KEY_COLLECTION_USERS).document(
                    preferenceManager.getString(Constants.KEY_USER_ID));
    documentReference.update(Constants.KEY_FCM_TOKEN, token)
            .addOnFailureListener(e -> showToast("Unable to update token"));
}

private void signOut() {
    showToast("Signing out...");
    FirebaseFirestore database = FirebaseFirestore.getInstance();
    DocumentReference documentReference =
            database.collection(Constants.KEY_COLLECTION_USERS).document(
                    preferenceManager.getString(Constants.KEY_USER_ID)
            );
    HashMap<String, Object> updates = new HashMap<>();
    updates.put(Constants.KEY_FCM_TOKEN, FieldValue.delete());
    documentReference.update(updates)
```

4) UsersActivity.java This file lists all the users by fetching the user’s data from firebase firestore. The user can simply click on another user to start texting with them.



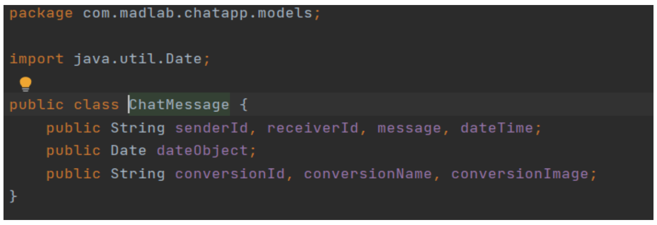
```java
private void getUsers() {
    loading( true);
    FirebaseFirestore database = FirebaseFirestore.getInstance();
    database.collection(Constants.KEY_COLLECTION_USERS)
            .get()
            .addOnCompleteListener(task -> {
                loading( false);
                String currentUserId = preferenceManager.getString(Constants.KEY_USER_ID);
                if (task.isSuccessful() && task.getResult() != null) {
                    List<User> users = new ArrayList<>();
                    for (QueryDocumentSnapshot queryDocumentSnapshot : task.getResult()) {
                        if (currentUserId.equals(queryDocumentSnapshot.getId())) {
                            continue;
                        }
                        User user = new User();
                        user.name = queryDocumentSnapshot.getString(Constants.KEY_NAME);
                        user.email = queryDocumentSnapshot.getString(Constants.KEY_EMAIL);
                        user.image = queryDocumentSnapshot.getString(Constants.KEY_IMAGE);
                        user.token = queryDocumentSnapshot.getString(Constants.KEY_FCM_TOKEN);
                        user.id = queryDocumentSnapshot.getId();
                        users.add(user);
                    }
                    if(users.size() > 0) {
                        UsersAdapter usersAdapter = new UsersAdapter(users,  this);
                        binding.usersRecyclerView.setAdapter(usersAdapter);
                        binding.usersRecyclerView.setVisibility(View.VISIBLE);
                    } else {
                        showErrorMessage();
                    }
                } else {
                    showErrorMessage();
```

5) ChatActivity.java This file fetches all the conversation with a user and display on the screen in sorted order according to the time. Also, user can send a message from this file which will be stored in the database.



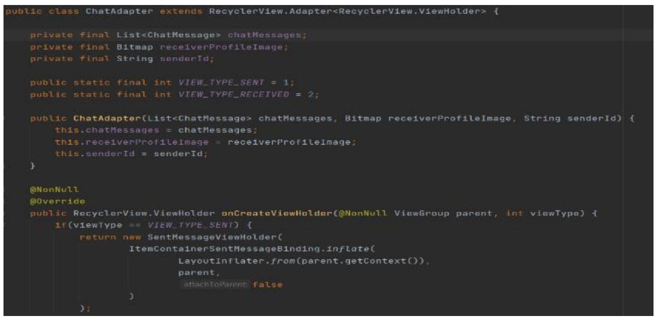
```java
private void loadReceiverDetails() {
    receiverUser = (User) getIntent().getSerializableExtra(Constants.KEY_USER);
    binding.textName.setText(receiverUser.name);
}

private void setListeners() {
    binding.imageBack.setOnClickListener(v -> onBackPressed());
    binding.layoutSend.setOnClickListener(v -> sendMessage());
}

private String getReadableDateTime(Date date) {
    return new SimpleDateFormat( "MMMM dd, yyyy - hh:mm a", Locale.getDefault()).format(date);
}

private void addConversion (HashMap<String, Object> conversion) {
    database.collection(Constants.KEY_COLLECTION_CONVERSATIONS)
            .add(conversion)
            .addOnSuccessListener(documentReference -> conversionId = documentReference.getId());
}

private void updateConversion(String message) {
    DocumentReference documentReference =
            database.collection(Constants.KEY_COLLECTION_CONVERSATIONS).document(conversionId);
    documentReference.update(
            Constants.KEY_LAST_MESSAGE, message,
            Constants.KEY_TIMESTAMP, new Date()
    );
private void listenMessages() {
    database.collection(Constants.KEY_COLLECTION_CHAT)
            .whereEqualTo(Constants.KEY_SENDER_ID, preferenceManager.getString(Constants.KEY_USER_ID))
            .whereEqualTo(Constants.KEY_RECEIVER_ID, receiverUser.id)
            .addSnapshotListener(eventListener);
    database.collection(Constants.KEY_COLLECTION_CHAT)
            .whereEqualTo(Constants.KEY_SENDER_ID, receiverUser.id)
            .whereEqualTo(Constants.KEY_RECEIVER_ID, preferenceManager.getString(Constants.KEY_USER_ID))
private void sendMessage() {
    HashMap<String, Object> message = new HashMap<>();
    message.put(Constants.KEY_SENDER_ID, preferenceManager.getString(Constants.KEY_USER_ID));
    message.put(Constants.KEY_RECEIVER_ID, receiverUser.id);
    message.put(Constants.KEY_MESSAGE, binding.inputMessage.getText().toString());
    message.put(Constants.KEY_TIMESTAMP, new Date());
    database.collection(Constants.KEY_COLLECTION_CHAT).add(message);
    if (conversionId != null) {
        updateConversion(binding.inputMessage.getText().toString());
    } else {
        HashMap<String, Object> conversion = new HashMap<>();
        conversion.put(Constants.KEY_SENDER_ID, preferenceManager.getString(Constants.KEY_USER_ID));
        conversion.put(Constants.KEY_SENDER_NAME, preferenceManager.getString(Constants.KEY_NAME));
        conversion.put(Constants.KEY_SENDER_IMAGE, preferenceManager.getString(Constants.KEY_IMAGE));
        conversion.put(Constants.KEY_RECEIVER_ID, receiverUser.id);
        conversion.put(Constants.KEY_RECEIVER_NAME, receiverUser.name);
        conversion.put(Constants.KEY_RECEIVER_IMAGE, receiverUser.image);
        conversion.put(Constants.KEY_LAST_MESSAGE, binding.inputMessage.getText().toString());
        conversion.put(Constants.KEY_TIMESTAMP, new Date());
        addConversion(conversion);
    }
    binding.inputMessage.setText(null);
}

private void listenAvailabilityOfReceiver() {
    database.collection (Constants.KEY_COLLECTION_USERS)
            .document(receiverUser.id)
            .addSnapshotListener( ChatActivity.this, (value, error) -> {
        if (error != null) {
            return;
```

6) ChatMessage.java It is a model describing the nature of a chat message and its properties. For example, senderId, recieverId, timestamp, senderImage etc. are some properties to list the few which describes the chat message.



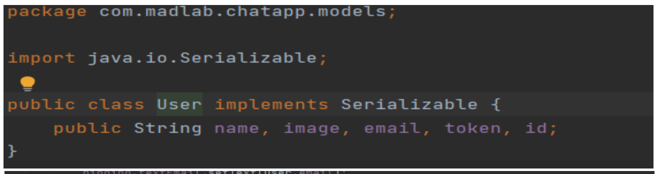
```java
package com.madlab.chatapp.models;

import java.util.Date;

public class ChatMessage {
    public String senderId, receiverId, message, dateTime;
    public Date dateObject;
    public String conversionId, conversionName, conversionImage;
}
```

7) ChatAdapter.java This file's class basically extends RecyclerView. It provides an easy way to efficiently display large sets of data. You supply the data and define how each item looks, and the RecyclerView library dynamically creates the elements when they're needed as it's needed by dynamic number of chats between two users.

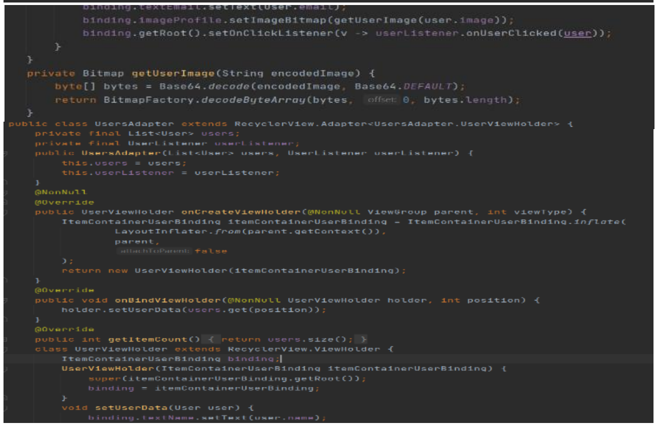


```java
public class ChatAdapter extends RecyclerView.Adapter<RecyclerView.ViewHolder> {

    private final List<ChatMessage> chatMessages;
    private final Bitmap receiverProfileImage;
    private final String senderId;

    public static final int VIEW_TYPE_SENT = 1;
    public static final int VIEW_TYPE_RECEIVED = 2;

    public ChatAdapter(List<ChatMessage> chatMessages, Bitmap receiverProfileImage, String senderId) {
        this.chatMessages = chatMessages;
        this.receiverProfileImage = receiverProfileImage;
        this.senderId = senderId;
    }

    @NonNull
    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        if(viewType == VIEW_TYPE_SENT) {
            return new SentMessageViewHolder(
                    ItemContainerSentMessageBinding.inflate(
                            LayoutInflater.from(parent.getContext()),
                            parent,
                            false
                    )
            );
```

8) User.java It is model describing the attributes of the user. Basic attributes like Name, Email and profile picture are attached with each of user.

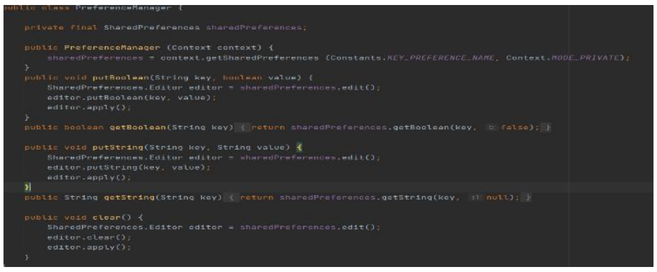


```java
package com.madlab.chatapp.models;

import java.io.Serializable;

public class User implements Serializable {
    public String name, image, email, token, id;
}
```



```java
                binding.textEmail.setText(user.email);
                binding.imageProfile.setImageBitmap(getUserImage(user.image));
                binding.getRoot().setOnClickListener(v -> userListener.onUserClicked(user));
            }
        }
        private Bitmap getUserImage(String encodedImage) {
            byte[] bytes = Base64.decode(encodedImage, Base64.DEFAULT);
            return BitmapFactory.decodeByteArray(bytes, offset: 0, bytes.length);
        }
public class UsersAdapter extends RecyclerView.Adapter<UsersAdapter.UserViewHolder> {
    private final List<User> users;
    private final UserListener userListener;
    public UsersAdapter(List<User> users, UserListener userListener) {
        this.users = users;
        this.userListener = userListener;
    }
    @NonNull
    @Override
    public UserViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        ItemContainerUserBinding itemContainerUserBinding = ItemContainerUserBinding.inflate(
                LayoutInflater.from(parent.getContext()),
                parent,
                attachToParent: false
        );
        return new UserViewHolder(itemContainerUserBinding);
    }
    @Override
    public void onBindViewHolder(@NonNull UserViewHolder holder, int position) {
        holder.setUserData(users.get(position));
    }
    @Override
    public int getItemCount() { return users.size(); }
    class UserViewHolder extends RecyclerView.ViewHolder {
        ItemContainerUserBinding binding;
        UserViewHolder(ItemContainerUserBinding itemContainerUserBinding) {
            super(itemContainerUserBinding.getRoot());
            binding = itemContainerUserBinding;
        }
        void setUserData(User user) {
            binding.textName.setText(user.name);
```

9) UserAdapter.java This file also implements RecyclerView for user due to dynamic nature of user creating accounts. Each of those users are needed to be displayed on add page which is easily done by recycler view.

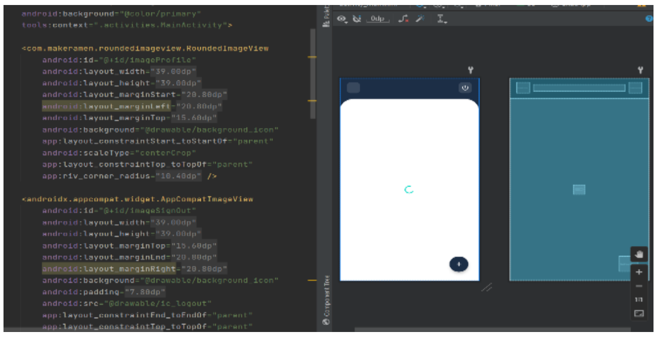


```java
public class PreferenceManager {

    private final SharedPreferences sharedPreferences;

    public PreferenceManager (Context context) {
        sharedPreferences = context.getSharedPreferences (Constants.KEY_PREFERENCE_NAME, Context.MODE_PRIVATE);
    }
    public void putBoolean(String key, boolean value) {
        SharedPreferences.Editor editor = sharedPreferences.edit();
        editor.putBoolean(key, value);
        editor.apply();
    }
    public boolean getBoolean(String key) { return sharedPreferences.getBoolean(key, b: false); }

    public void putString(String key, String value) {
        SharedPreferences.Editor editor = sharedPreferences.edit();
        editor.putString(key, value);
        editor.apply();
    }
    public String getString(String key) { return sharedPreferences.getString(key, s1: null); }

    public void clear() {
        SharedPreferences.Editor editor = sharedPreferences.edit();
        editor.clear();
        editor.apply();
    }
```

10) PreferenceManager.java This file contains the common methods to get the shared preferences between various activities and screens.

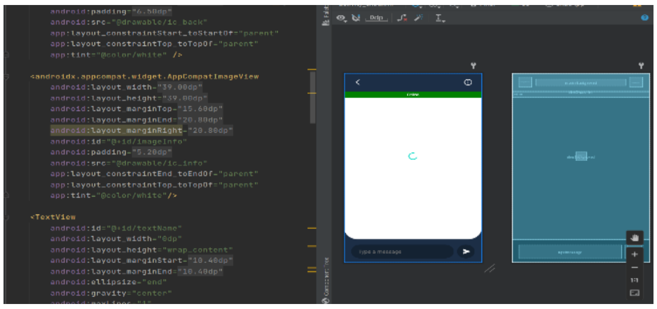
B. XML Code

1) activity_main.xml This page contains the XML code for the base of application. It acts as the container to hold other components like recent messages, users, chat, message etc.



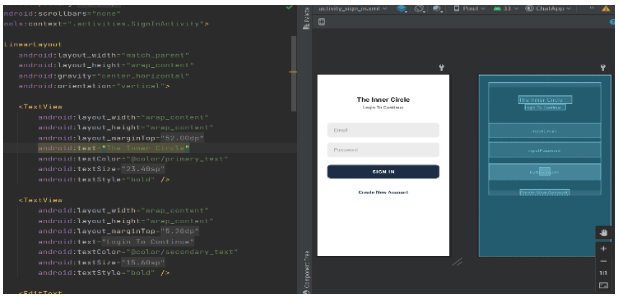
```xml
    android:background="@color/primary"
    tools:context=".activities.MainActivity">

    <com.makeramen.roundedimageview.RoundedImageView
        android:id="@+id/imageProfile"
        android:layout_width="39.00dp"
        android:layout_height="39.00dp"
        android:layout_marginStart="20.80dp"
        android:layout_marginLeft="20.80dp"
        android:layout_marginTop="15.60dp"
        android:background="@drawable/background_icon"
        app:layout_constraintStart_toStartOf="parent"
        android:scaleType="centerCrop"
        app:layout_constraintTop_toTopOf="parent"
        app:riv_corner_radius="10.40dp" />

    <androidx.appcompat.widget.AppCompatImageView
        android:id="@+id/imageSignOut"
        android:layout_width="39.00dp"
        android:layout_height="39.00dp"
        android:layout_marginTop="15.60dp"
        android:layout_marginEnd="20.80dp"
        android:layout_marginRight="20.80dp"
        android:background="@drawable/background_icon"
        android:padding="7.80dp"
        android:src="@drawable/ic_logout"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"
```

2) activity_chat.xml This file contains the XML code for the interface which is encountered by users when tap on other users to chat with.



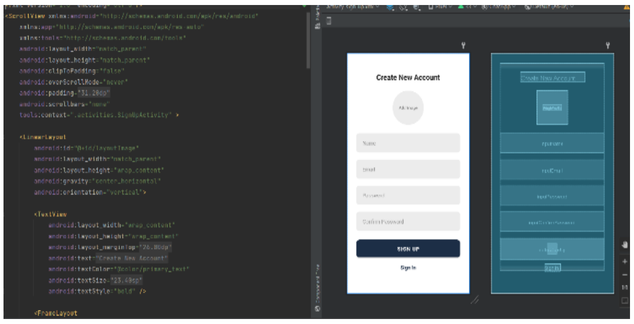
```
        android:padding="6.50dp"
        android:src="@drawable/ic_back"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:tint="@color/white" />

    <androidx.appcompat.widget.AppCompatImageView
        android:layout_width="39.00dp"
        android:layout_height="39.00dp"
        android:layout_marginTop="15.60dp"
        android:layout_marginEnd="20.80dp"
        android:layout_marginRight="20.80dp"
        android:id="@+id/imageInfo"
        android:padding="5.20dp"
        android:src="@drawable/ic_info"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:tint="@color/white"/>

    <TextView
        android:id="@+id/textName"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="10.40dp"
        android:layout_marginEnd="10.40dp"
        android:ellipsize="end"
        android:gravity="center"
```

3) activity_signin.xml This file contains the XML code for the user sign in activity. The design basically consists of EditText and Button.



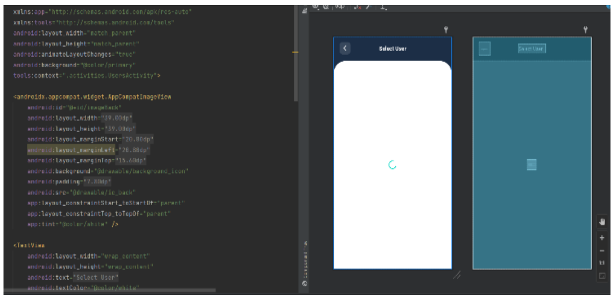
```
ndroid:scrollbars="none"
ools:context=".activities.SignInActivity">

LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_horizontal"
    android:orientation="vertical">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="52.00dp"
        android:text="The Inner Circle"
        android:textColor="@color/primary_text"
        android:textSize="23.40sp"
        android:textStyle="bold" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="5.20dp"
        android:text="Login To Continue"
        android:textColor="@color/secondary_text"
        android:textSize="15.60sp"
        android:textStyle="bold" />
```

4) activity_signup.xml This file contains the XML code for the signup activity. It is designed using TextView, EditText, ImageViews and Buttons.



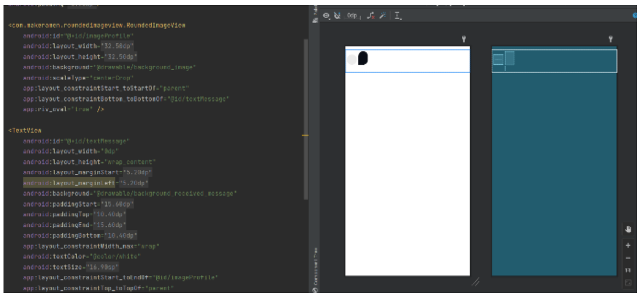
```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:clipToPadding="false"
    android:overScrollMode="never"
    android:padding="31.20dp"
    android:scrollbars="none"
    tools:context=".activities.SignUpActivity" >

    <LinearLayout
        android:id="@+id/layoutImage"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:orientation="vertical">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="26.00dp"
            android:text="Create New Account"
            android:textColor="@color/primary_text"
            android:textSize="23.40sp"
            android:textStyle="bold" />

        <FrameLayout
```

5) activity_users.xml This file contains the XML code for adding other users with whom one wants to start the conversation.



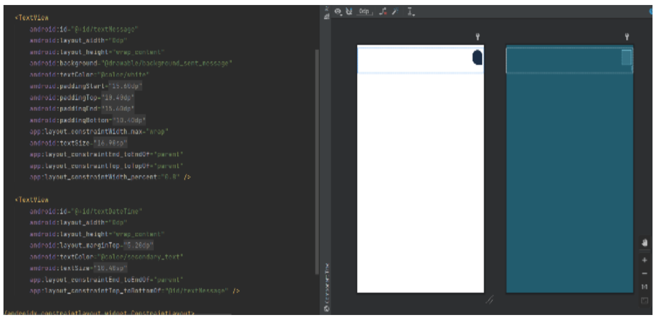
```
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:animateLayoutChanges="true"
    android:background="@color/primary"
    tools:context=".activities.UsersActivity">

    <androidx.appcompat.widget.AppCompatImageView
        android:id="@+id/imageBack"
        android:layout_width="39.00dp"
        android:layout_height="39.00dp"
        android:layout_marginStart="20.80dp"
        android:layout_marginLeft="20.80dp"
        android:layout_marginTop="15.60dp"
        android:background="@drawable/background_icon"
        android:padding="7.80dp"
        android:src="@drawable/ic_back"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:tint="@color/white" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Select User"
        android:textColor="@color/white"
```

6) item_container_received_message.xml This file contains the XML code that consists of TextView and RoundImage which describes how received message bubble should look like. It also displays of time of message received.



```
    <com.makeramen.roundedimageview.RoundedImageView
        android:id="@+id/imageProfile"
        android:layout_width="32.50dp"
        android:layout_height="32.50dp"
        android:background="@drawable/background_image"
        android:scaleType="centerCrop"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintBottom_toBottomOf="@id/textMessage"
        app:riv_oval="true" />

    <TextView
        android:id="@+id/textMessage"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="5.20dp"
        android:layout_marginLeft="5.20dp"
        android:background="@drawable/background_received_message"
        android:paddingStart="15.60dp"
        android:paddingTop="10.40dp"
        android:paddingEnd="15.60dp"
        android:paddingBottom="10.40dp"
        app:layout_constraintWidth_max="wrap"
        android:textColor="@color/white"
        android:textSize="16.90sp"
        app:layout_constraintStart_toEndOf="@id/imageProfile"
        app:layout_constraintTop_toTopOf="parent"
```

7) item_container_sent_message.xml This file contains the XML code that consists of TextView and RoundImage which describes how sent message bubble should look like. Likewise, time of sending message is also shown.



```
<TextView
    android:id="@+id/textMessage"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:background="@drawable/background_sent_message"
    android:textColor="@color/white"
    android:paddingStart="15.60dp"
    android:paddingTop="10.40dp"
    android:paddingEnd="15.60dp"
    android:paddingBottom="10.40dp"
    app:layout_constraintWidth_max="wrap"
    android:textSize="16.90sp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintWidth_percent="0.8" />

<TextView
    android:id="@+id/textDateTime"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="5.20dp"
    android:textColor="@color/secondary_text"
    android:textSize="10.40sp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toBottomOf="@id/textMessage" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

VII. Performance and scalability

The project focuses on providing a simple and user-friendly chat application that emphasizes clean and self-explanatory user interfaces. It aims to facilitate communication between individuals who are physically apart, offering a more convenient and efficient mode of communication compared to traditional methods like emails or faxes.

The performance and scalability of the project are crucial considerations. By utilizing efficient coding practices, optimizing resource usage, and implementing proper data management techniques, the app can deliver fast and responsive performance. Additionally, the app can be designed to handle increasing user demands and accommodate a growing user base through scalable architecture and efficient data handling mechanisms. These measures ensure that the app can handle large volumes of chat messages and support a growing number of users without compromising performance or user experience.

VIII. Conclusion

The app offers reliability, time savings and easy control. Two users can efficiently and effectively communicate with each other . This application will show the texts sent by person1 and text sent by person 2 from two different location. From the perspective of person1, he will see the texts he sends on the right side of the container and the texts sent by person2 will be seen by person1 on the left hence differentiating the texts of both person and text will be seen from top to bottom by in order the time when they were sent.

By analyzing the existing system, we have come to a conclusion that the propose system will not only aid the communication but also helps keep a record of conversation efficiently.

References

(i). Google Nanodegree-Android Basics Course-Udacity.

(ii). Basic Chatting App Using Firebase - International Research Journal of Engineering and Technology (IRJET).

(iii). “Developing an Android Application for Chatting”, International Journal of Future Innovative Science and Engineering Research, Volume-2, Issue-2, JUNE – 2016.

(iv). Androio Studio : https://developer.android.com/build.