A Real-Time Chat And Communication App

Team Members	NM Id
Rajaputhiran.M	837199F79B01E5DA5384C2ADE68B0151
Navivarma.N	4885287B8AC7041A58147F86EAC66179
Senthamilselvi.S	7FB726798188E74AA1B4B3540CF804C2
Mahalakshmi .P	E75B9BBEBD82E54892C4649A52DF3F53

Description:

ChatConnect is a sample project built using the Android Compose UI toolkit. It demonstrates how to create a simple chat app using the Compose libraries. The app allows users to send and receive text
messages. The project showcases the use of Compose's declarative UI and state management capabilities. It also includes examples of how to handle input and navigation using composable functions and how to use data from a firebase to populate the UI

Required Initial Steps:

Step 1.:Download Android Studio: - Visit the official Android Studio download page:

https://developer.android.com/studio.

1. **Review and accept the terms and conditions if prompted.**
2. **The download will begin automatically.**

Step 2.:Install Android Studio:

For Windows:

1. **Run the installer: Double-click the downloaded `.exe` file.**
2. **Follow the installation wizard:**

○ Choose installation options (use default settings unless you have specific preferences). ○ Android Studio requires a minimum of 4 GB of RAM (recommended 8 GB), and a screen resolution of at least 1280x800.

3. **Install Android SDK: The installer will automatically install the Android SDK, Android Emulator, and other necessary tools.**
4. **Complete installation: Click `Finish` once the process is complete.**
5. **3. Follow the setup wizard: Android Studio will guide you through the rest of the installation.**

Step3: First Launch and Setup:

6. **Choose UI Theme**: Select between `Light` or `Dark` theme.**
7. **Install SDK Components**: Android Studio will download necessary SDK components (such as the latest Android platform and tools). This might take some time, depending on your internet speed.**

Step 4: Configure Android Studio:

8. **-Install additional SDK packages: You might need to install certain packages depending on your Android development needs (e.g., specific Android API versions or system images for the emulator).**
9. **Set up the Android Emulator: If you plan to use the Android Emulator for testing, you can set it up through the AVD (Android Virtual Device) Manager.**

Step5: Verify Installation:

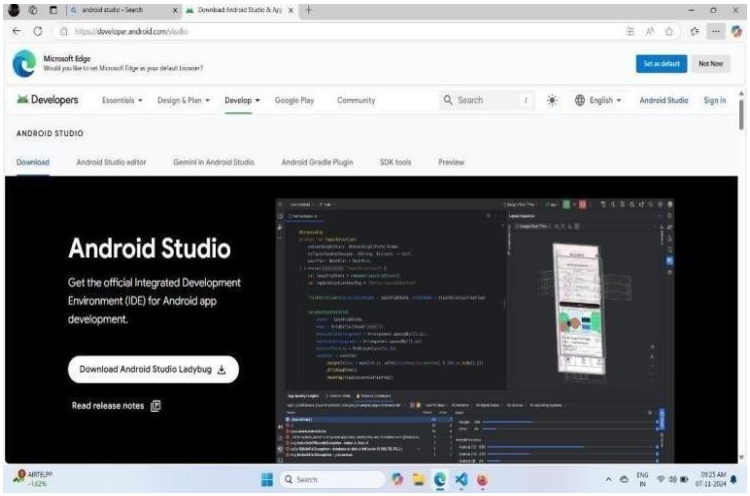
10. **Go to `File` → `New` → `New Project`.**
11. **Choose a template (e.g., `Empty Activity`).**
12. **Configure the project (e.g., name, package name).**
13. **Click `Finish`.**

Step6: Install Android Device/SDK Drivers (If needed):

14. **You have enabled USB debugging on the device (`Settings` → `About phone` →**

Tap `Build number` 7 times to unlock developer options → `Developer options`→ Enable `USB debugging`).

Screen Shots:



Creating New Project:

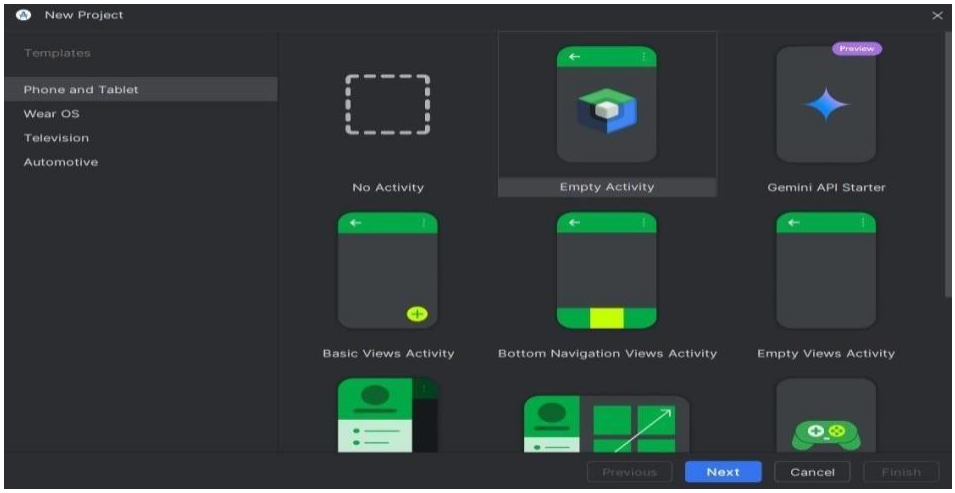
Step1: Open Android Studio:

Launch Android Studio on your computer.

Step2:Start a New Project:

On the welcome screen, click "Start a new Android Studio project".

Screen Shots:



Step3:Configure Your Project:

3. **Name:** Enter your app's name.
4. **Package Name:** Enter a unique identifier (e.g., `com.example.myapp`).
5. **Language:** Choose between Java or Kotlin.
6. **Minimum SDK:** Choose the lowest Android version your app will support.

Step4:Choose a Template:

Select a project template (e.g., "Empty Activity"for a simple starting point), then click "Next".

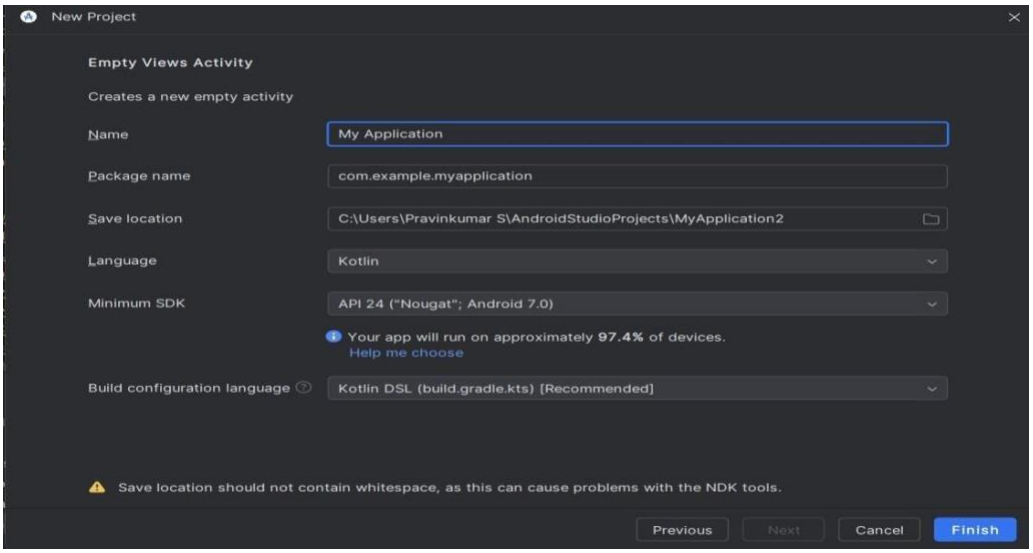
Step5: Configure Activity:

1. **Activity Name:** Default is `MainActivity` (you can change it).
2. **Layout Name:** Default is `activity_main.xml`.
3. **-Click "Finish".**

Step6:Run the App:

Once the project is set up, click the "Run"button to launch the app on an emulator or a physical device.

Screen Shots:



Integrating Firebase:

Step 1: Set up Firebase Project:

1. Go to the [Firebase Console](https://console.firebase.google.com/).
2. Click “Add project” and follow the steps to create a new Firebase project.
3. Once created, navigate to the “Project Overview” page in the Firebase Console.

Step 2: Add Firebase to Your Android Project:

4. In the Firebase Console, click on the “Android icon” to add Firebase to your Android app.
5. Register your app by entering your “Android package name”(found in `AndroidManifest.xml`) and optionally, the app’s “nickname”.

Step 3: Configure Firebase SDK in Android Studio:

6. In “Android Studio”, open your project and ensure you have the Firebase SDK dependencies in place.
7. Modify your `project-level build.gradle’ file to include the Google services plugin:

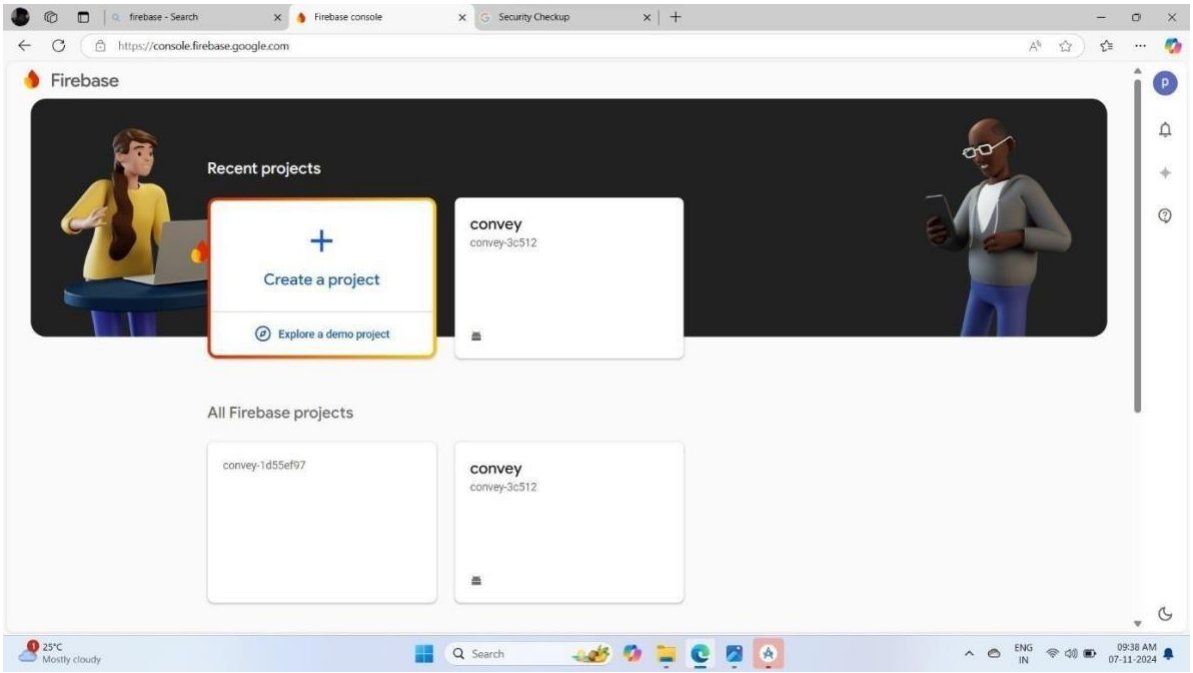
Step 4: Sync the Project:

Click “Sync Now” in the bar that appears at the top of the file to sync your project with Firebase.

Step 5: Use Firebase Features:

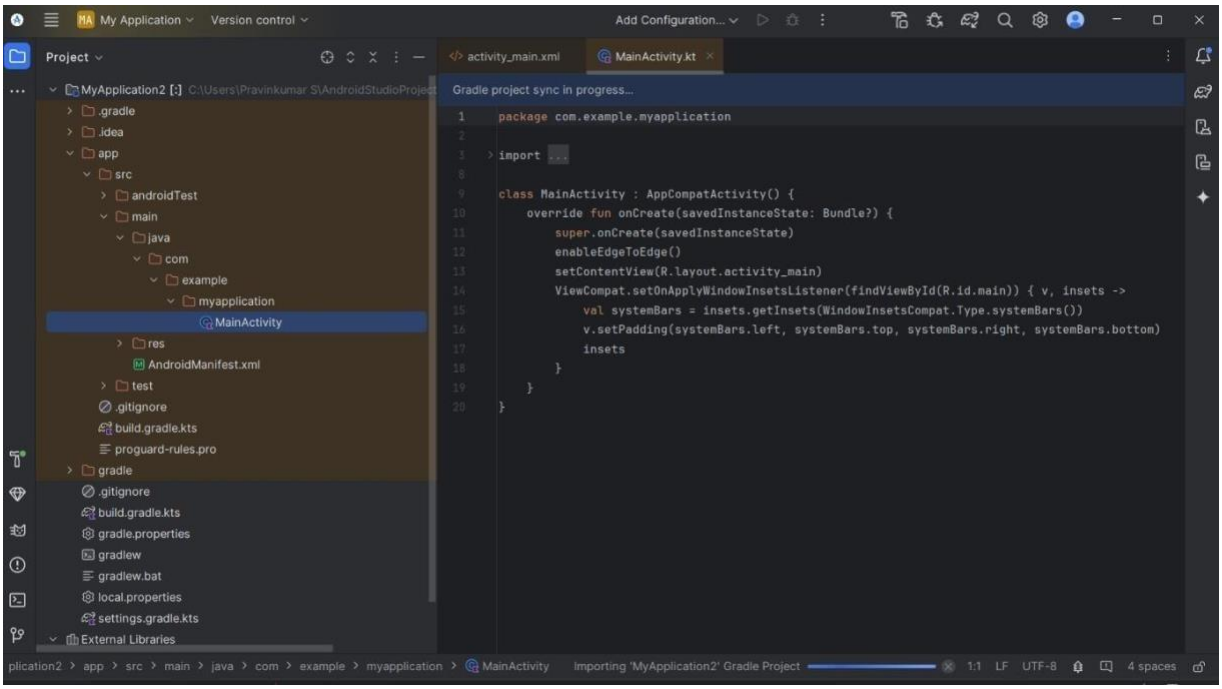
Now, you can use other Firebase features like Firestore, Firebase Authentication, Realtime Database, etc., by adding the respective Firebase dependencies and following their integration steps.

Screen Shots:



Main Activity.kt File:

Step 1: Define the MainActivity in `MainActivity.kt` Screen Shots:



8. “Navigate to MainActivity.kt” under `src > main > java > com.example.myapp` (your app package).

Step 3: Create the Navigation Package:

9. Right-click on the `com.example.myapp` packagein the `Project` view.
10. Select New > Package and name it `nav`.
11. This will create a new package `nav` inside your `java > com.example.myapp` directory.

Step 4: Add Navigation Components:

12. In the `nav` package, create a new Kotlin class for a second screen, such as `SecondActivity.kt`:

13. This is a new screen you'll navigate to when the button in `MainActivity` is clicked.

Step 5: Update `activity_main.xml` for Navigation:

1. Open `res > layout > activity_main.xml`.
2. Modify it to include a Button that will trigger the navigation.

Step 6: Add Navigation Code to `MainActivity.kt`:

1. Open `MainActivity.kt` and import the `Intent` class for navigating between activities.
2. Update the `OnClickListener` for the button to navigate to `NextActivity`.

MileStone05:

Running The Application:

Step 1: Open Android Studio and Run the Chat Application

3. **Launch Android Studio and open your existing chat app project or create a new one.**

4. **Connect a Device (either an emulator or a physical Android device).**
    - If using an emulator, open the AVD Manager and select an Android Virtual Device.
    - If using a physical device, enable Developer Mode on the device and connect it via USB.

5. **Click the Green Run Button (Shift + F10) in Android Studio to compile and run the app.** ○ Wait for the APK to be built and installed on the emulator or device.

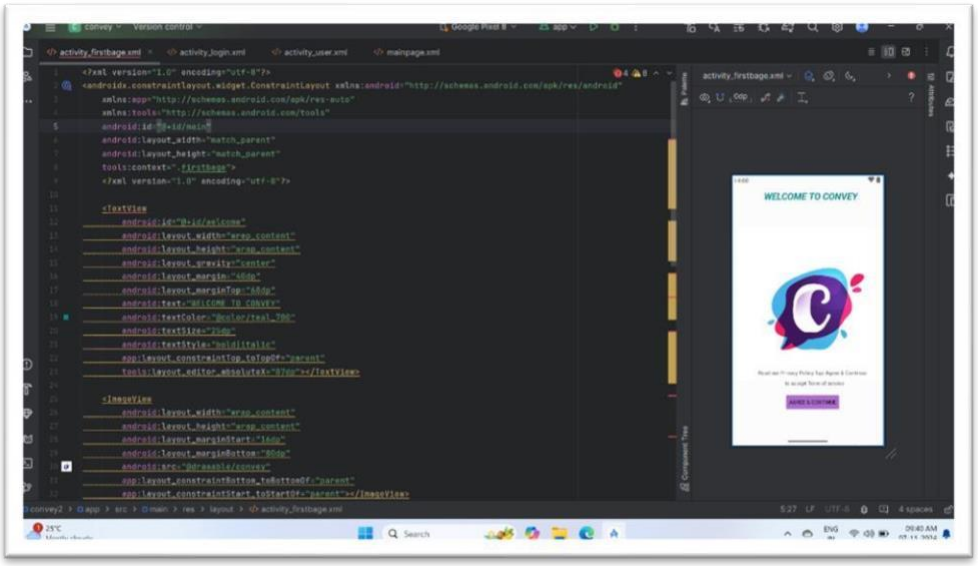
    - The chat application should open up automatically after it is successfully deployed.

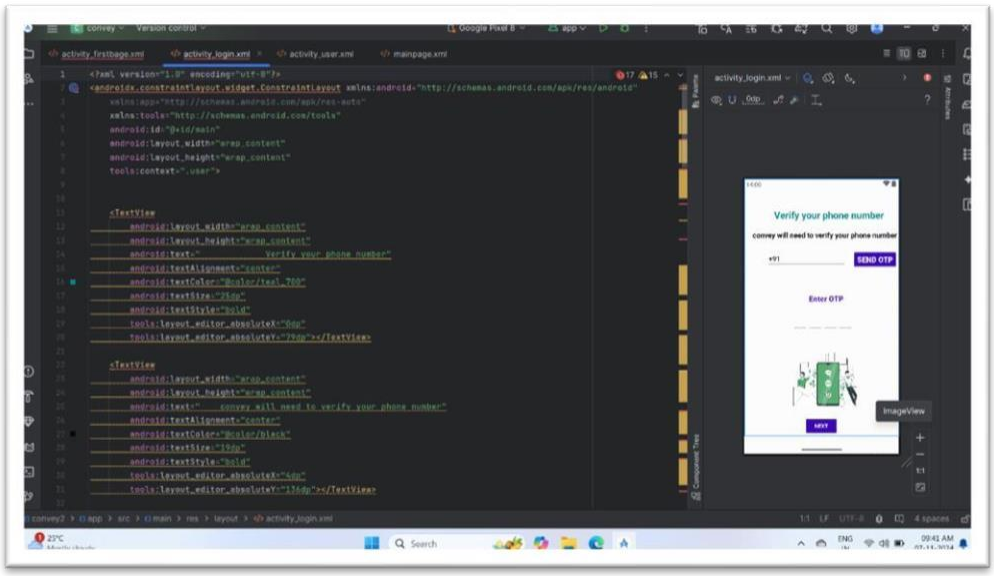
Step1: User Authentication:

1. Authentication Systems:
2. Commonly used methods for authentication include **email/password** logins, social media logins (e.g., Google, Facebook), or even **phone number authentication.
3. Many chat apps use services like Firebase Authentication for easy integration of login functionality.

2. Frontend (User Interface):

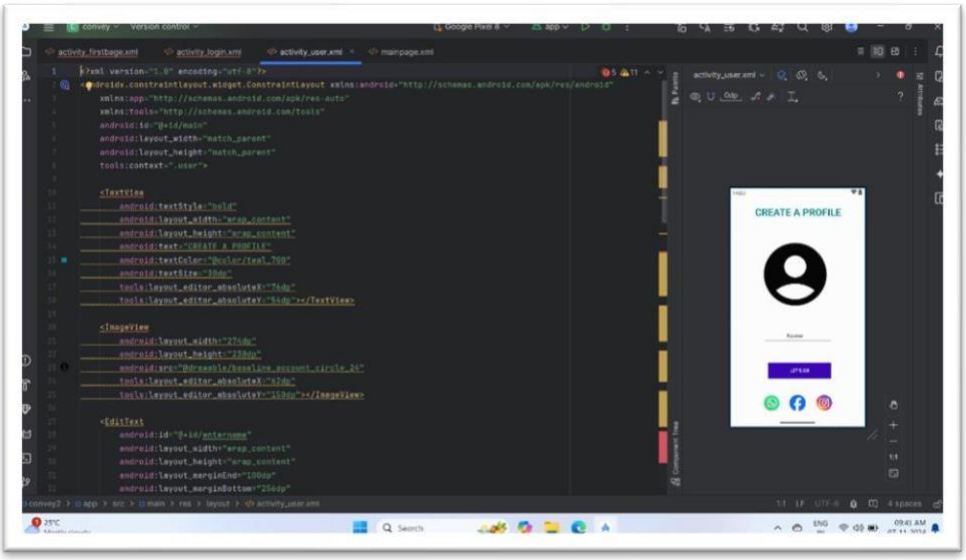
1. Message input field**: Where users can type their messages.
2. Send button: To send the typed message.
3. Message display area: A list or chat bubble view (usually a
4. RecyclerView or ListView) that displays sent and received messages.
5. User avatars and timestamps: To personalize the messages and show when they were sent.
6. Push notifications: To alert the user when a new message is received (optional)
7. In Android, common UI components for this include:
8. `EditText` for message input.
9. `Button` for sending messages.
10. `RecyclerView` for displaying messages.





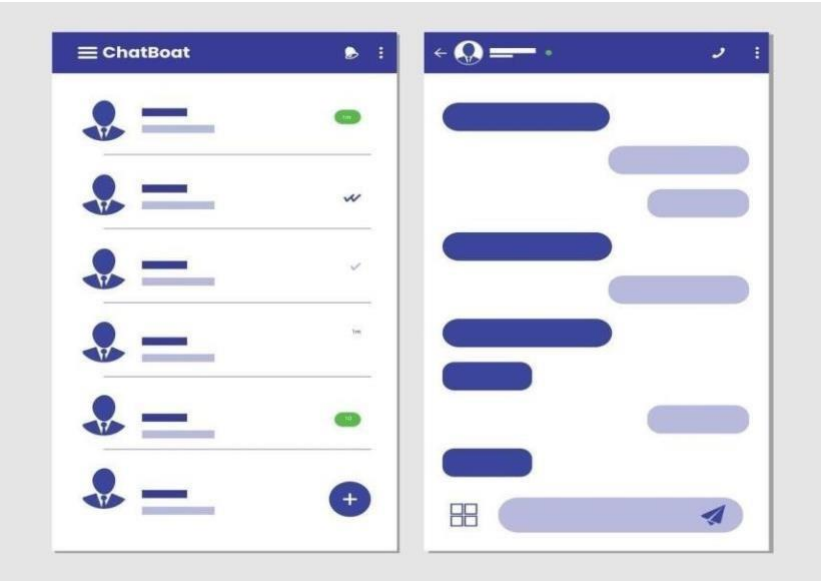
Step3: Backend (Server/Cloud Database):

1. Cloud Database:
2. Firebase provides both **Realtime Database** and **Firestore**, which are cloud-hosted NoSQL databases that automatically sync data across devices in real-time.
3. Messages are stored as records in the database (e.g., a collection of messages with user IDs, message content, and timestamps).
4. Database Structure Example (using Firebase Realtime Database):
5. ```plaintext



Step4: Sending and Receiving Messages:

1. User Action: A user types a message in the input field and clicks the “send button”.
2. Backend Action: The app then sends this message to the backend (e.g., Firebase). This involves writing the message to a database with relevant information (e.g., sender, message text, timestamp).
3. Data Sync: The backend ensures that the message is stored correctly and that it syncs with all other connected clients. This is handled via “real-time listeners” that monitor changes in the message database.



Step6:Security and Data Privacy:

- “Authentication and Authorization”: Make sure that users are authenticated correctly and that they can only access their own messages.
- “End-to-End Encryption” (optional but common): Some chat apps, like WhatsApp, offer end-to-end encryption, meaning messages are encrypted on the sender’s device and decrypted only on the receiver’s device, making it impossible for even the server to read the content.
- “Database Security Rules”: For Firebase, you can set “security rules”to ensure that only authorized users can read and write to certain parts of the database. ○ Example rule to allow users to access their own messages:

```
json
{
  "rules": {
    "messages": {
      "$message_id": {
        ".read": "auth != null && data.child('sender').val() === auth.uid",
".write": "auth != null"
      }
    }
  }
```

Code:

Main Activity Code:

```
package com.example.chatapp

import android.os.Bundle
import android.text.Editable
import android.text.TextWatcher
import android.widget.EditText
import android.widget.ImageView
import android.widget.LinearLayout
import androidx.appcompat.app.AppCompatActivity
```

```
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.google.android.material.floatingactionbutton.FloatingActionButton

class MainActivity : AppCompatActivity() {

    private lateinit var recyclerView: RecyclerView
    private lateinit var messageInput: EditText
    private lateinit var sendButton: FloatingActionButton
    private lateinit var chatAdapter: ChatAdapter
    private val chatMessages = mutableListOf<ChatMessage>()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Initialize views
        recyclerView = findViewById(R.id.recyclerView)
        messageInput = findViewById(R.id.messageInput)
        sendButton = findViewById(R.id.sendButton)

        // Set up RecyclerView
        recyclerView.layoutManager = LinearLayoutManager(this)
        chatAdapter = ChatAdapter(chatMessages)
        recyclerView.adapter = chatAdapter

        // Add text change listener to message input
        messageInput.addTextChangedListener(object : TextWatcher {
            override fun afterTextChanged(s: Editable?) {}
```

```
            override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {}

            override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {
                sendButton.isEnabled = s?.isNotEmpty() == true
            }
        })

        // Send message on button click
        sendButton.setOnClickListener {
            val messageText = messageInput.text.toString()
            if (messageText.isNotBlank()) {
                sendMessage(messageText)
            }
        }
    }

    // Function to send a message
    private fun sendMessage(messageText: String) {
        // Create a new message object
        val message = ChatMessage(messageText, System.currentTimeMillis())
        chatMessages.add(message)
        chatAdapter.notifyItemInserted(chatMessages.size
        recyclerView.scrollToPosition(chatMessages.size - 1)

        // Clear input field
        messageInput.text.clear()
    }
}
```

Main Activity.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
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"
    tools:context=".MainActivity">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toTopOf="@+id/messageInput"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <EditText
        android:id="@+id/messageInput"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="Type a message"
        android:layout_margin="16dp"
        app:layout_constraintBottom_toTopOf="@+id/sendButton"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/recyclerView"/>

    <com.google.android.material.floatingactionbutton.FloatingActionButton
```

```
    android:id="@+id/sendButton"
    android:layout_width="56dp"
    android:layout_height="56dp"
    android:src="@drawable/ic_send"
    android:tint="@android:color/white"
    android:layout_margin="16dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent" />

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

Message.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:padding="10dp">

    <TextView
        android:id="@+id/messageText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="16sp"
        android:textColor="@android:color/black"/>

    <TextView
        android:id="@+id/timestampText"
        android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:textSize="12sp"
    android:textColor="@android:color/darker_gray"/>
</LinearLayout>
```

Conclusion:

Real-Time Chat Communication App for Android

In conclusion, the development of a real-time chat communication app for Android provides a seamless and efficient platform for users to connect and communicate in real time. By leveraging technologies such as WebSockets, Firebase, or XMPP, the app can ensure fast, reliable message delivery, while features like user authentication, push notifications, and media sharing enhance the overall user experience.