Mobile Programming Final Project

Marwan Abdelhamid 2206174

Peter Hany 2306135

Project Title: Mobile Computing Chat Application

1.Introduction

We developed a real-time chat application called Class Chat. The goal was to build a working mobile app that uses cloud services to handle data instead of a traditional server. I built the app using **Flutter** and connected it to **Firebase** to handle the backend requirements.

How it Works The application allows users to sign up for an account using their email and password. Once logged in, they enter a shared chat room where they can send messages to other users. The main focus of the project was to ensure that communication happens instantly when one person sends a text, it appears on everyone else's screen immediately.

1.1 Key Features

- **Login & Registration:** Users can securely create accounts and sign in.
- **Instant Chat:** We used the Realtime Database so messages sync instantly across devices.
- **User Profiles:** When a user registers, their details (like join date) are saved to Cloud Firestore and displayed in a side menu.
- **Interactive Features:** We added extra functionality where users can double-tap to like a message or long-press to delete their own messages.

2. Implementation Details

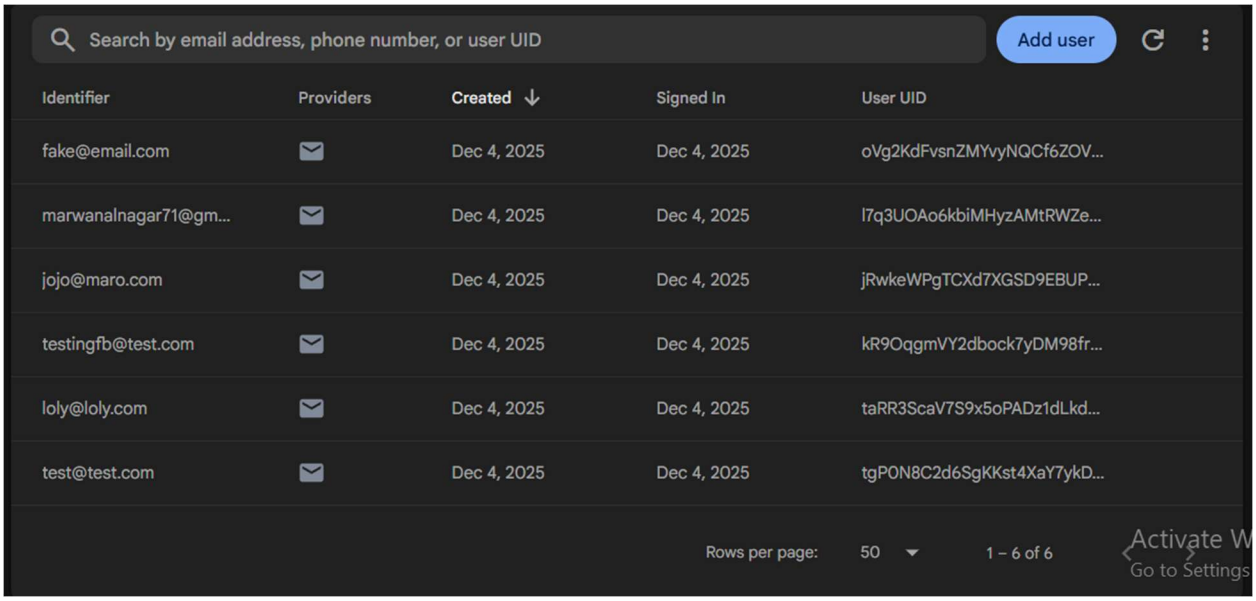
2.1 Firebase Authentication

To handle user security, We integrated the **Firebase Authentication** service. The goal was to ensure that only registered users could access the chat room.

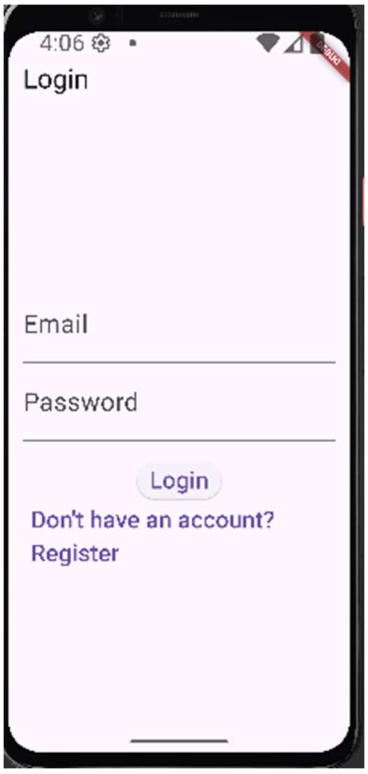
Integration Steps:

1. **Console Setup:** First, I went to the Firebase Console and enabled the Email/Password sign-in provider. This allows Firebase to manage credentials for me.
2. **Code Implementation:** We installed the firebase_auth package in Flutter. We created a dedicated service class (AuthService) to handle the logic.
    - **Registration:** When a new user signs up, the app calls (createUserWithEmailAndPassword)
    - **Login:** When a user logs in, it calls (signInWithEmailAndPassword)
3. **User Flow:** We built a LoginScreen that toggles between Login and Register modes. Once the authentication is successful, the app checks

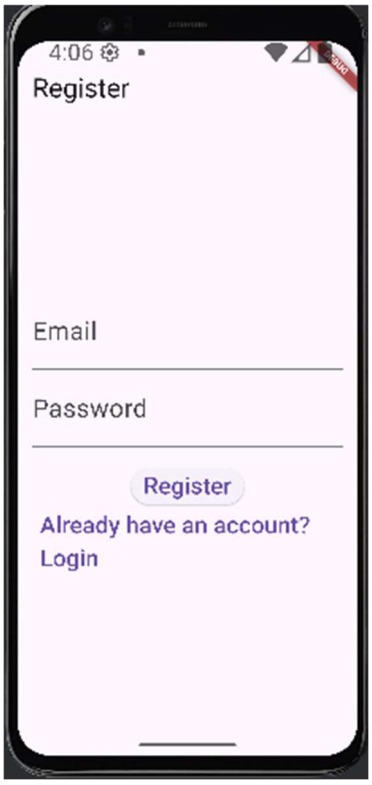
if the user object is not null and immediately navigates them to the Chat Screen.



List of all users that firebase authenticated



Login Screen



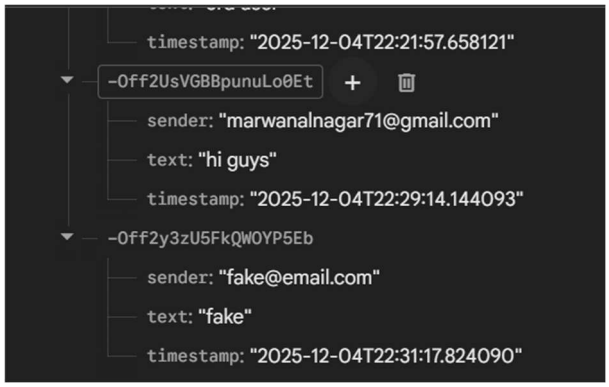
Register Screen

2.2 Realtime Database

For the core messaging feature, We applied the **Firebase Realtime Database**. This component is responsible for storing and syncing the chat messages between all users.

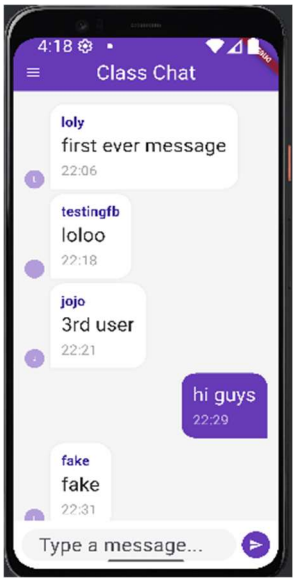
Integration Steps:

1. **Database Setup:** We created a Realtime Database in the console and set the security rules to **test mode** to allow easy reading and writing during development.
2. **Sending Messages:** In the code, We used the **firebase_database** package. When a user presses Send, the app creates a reference to a messages node in the database. We use the **.push().set()** method to add the message text, the sender's email, and a timestamp.
3. **Receiving Messages:** To make the chat real time, We used a **StreamBuilder**. This widget listens directly to the database. Whenever a new message is added to the database by anyone, the stream detects the change and automatically updates the list on the screen without needing to refresh.



Screenshot from the Realtime Database showing that whenever a message is sent its added instantly into the database with the sender email, message content and the timestamp of the message.

The group chat messages showing which user sent which message



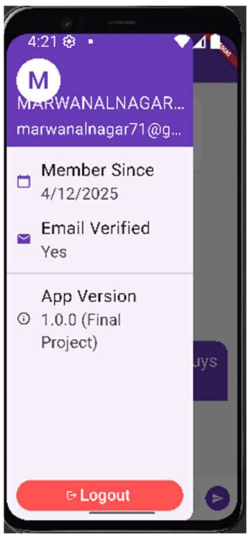
2.3 Firebase Firestore Database

We implemented **Cloud Firestore** to handle user profile data. While the Realtime Database is great for chat, Firestore is better for structured data like user details, so I used it to store account information.

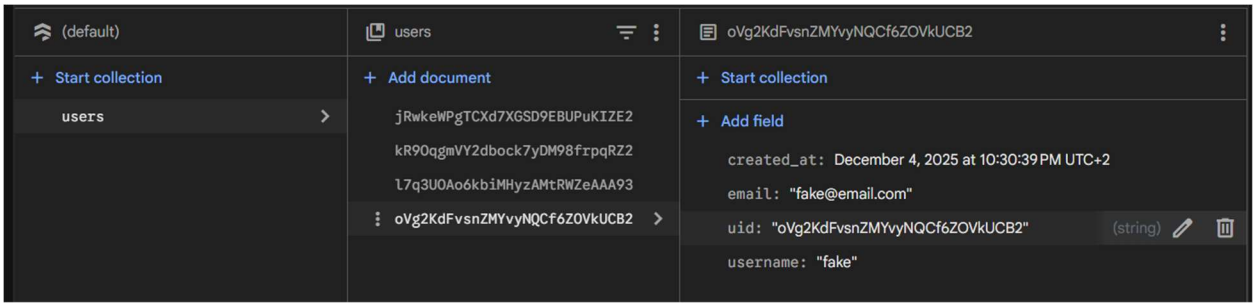
Integration Steps:

1. **Writing Data (Registration):** We modified the registration logic in our code. Now, immediately after a user successfully creates an account, the app takes their unique ID **(uid)** and creates a document in the users collection in **Firestore.** This document saves their email and the exact timestamp of when they joined.
2. **Reading Data (Profile):** To prove the data is being stored and retrieved correctly, We built a **Side Menu (Drawer)** in the app. When opened, it uses a **FutureBuilder** to fetch the user's specific document from **Firestore** and displays their "Member Since" date. This demonstrates that the app can both write to and read from the **Firestore database**.

Screenshot showing user profile data which is retrieved from Cloud Firestore



Firestore Database console



3.Justification

3.1 Firebase Authentication

This service handles the entire login and sign-up process for the app. It securely manages user sessions and passwords, allowing us to implement a safe login system without writing complex security code. It ensures that every message sent is cryptographically linked to a verified user identity, preventing users from impersonating others or accessing private conversations anonymously.

3.2 Cloud Firestore

We used Firestore to store **User Profiles** and account details. Its document-based structure is ideal for storing static user information (like email and username) separate from the high-frequency chat logs in the Realtime Database.

3.3 Firebase Realtime Database

This service handles the Core Messaging Feature. It was selected because it allows us to utilize a StreamBuilder to listen for database changes directly. This ensures that whenever a message is pushed to the database (using .push().set()), the chat list updates instantly on the screen without the user needing to refresh.