EXPERIMENT 6

AIM: To Set Up Firebase with Flutter for iOS and Android Apps.

Theory

Introduction to Firebase and Flutter Integration

Firebase is a comprehensive platform developed by Google, designed to help developers build high-quality applications for both mobile and web. It provides essential services such as real-time databases, authentication, cloud storage, hosting, and much more. One of the most widely used Firebase services is the Firebase Realtime Database, which is a NoSQL cloud database that allows data to be stored and synced in real-time across all connected devices.

Flutter, on the other hand, is an open-source UI software development kit created by Google, which allows developers to build natively compiled applications for mobile, web, and desktop from a single codebase. Its rich set of pre-designed widgets and powerful tools makes Flutter an attractive option for developing visually appealing and performant applications.

Integrating Firebase with Flutter allows developers to leverage the full potential of Firebase services in their applications. By using Firebase's Realtime Database, Flutter apps can achieve features such as real-time data synchronization, secure authentication, and cloud-based storage. This combination enables developers to create powerful, scalable, and feature-rich mobile and web applications.

Firebase Realtime Database Overview

Firebase Realtime Database is a cloud-hosted NoSQL database that stores data in a JSON-like format. The key characteristic of this database is its real-time synchronization feature, meaning that any changes made to the database are instantly reflected on all clients (i.e., devices) connected to it.

This makes it an ideal solution for applications that require frequent updates and need to maintain synchronized data across multiple users or devices, such as messaging apps, social media platforms, or collaborative tools.

The Firebase Realtime Database is structured as a tree of data, where each node in the tree can contain key-value pairs. This structure allows for easy data retrieval and modification. Firebase's real-time capabilities enable apps to immediately receive updates to the data whenever it changes, without the need to refresh or reload the page.

Additionally, the database supports offline data persistence, meaning that even if the user's device loses its internet connection, the app can still function by using the locally cached data.

Setting Up Firebase in Flutter

To connect a Flutter app with Firebase, the following steps are typically followed:

1. Creating a Firebase Project:

To start using Firebase with Flutter, the first step is to create a Firebase project in the Firebase Console. Once the project is created, developers can associate their Flutter app with the Firebase project by following the platform-specific instructions for Android or iOS. This usually involves configuring API keys, downloading configuration files, and adding them to the Flutter project.

2. Integrating Firebase SDK in Flutter:

After the Firebase project is set up, developers need to integrate Firebase's SDK into the Flutter app. This involves adding the necessary dependencies to the Flutter project's pubspec.yaml file. For Firebase's Realtime Database, the package firebase_database is used. Additionally, Firebase's core SDK (firebase_core) must also be included to initialize Firebase services.

3. Initializing Firebase:

Before any Firebase functionality can be used, it is essential to initialize Firebase in the Flutter app. This is done by calling Firebase.initializeApp() in the main entry point of the app (usually in the main.dart file). Firebase needs to be initialized before interacting with any Firebase services, such as the Realtime Database, Cloud Firestore, or Authentication.

Connecting Firebase to a Flutter app enables developers to create robust, scalable, and real-time applications with ease. Firebase's Realtime Database offers a powerful, cloud-based solution for managing data in real-time, while Firebase Authentication ensures secure access control. By integrating Firebase with Flutter, developers can take advantage of real-time data synchronization, offline support, and a wide range of other Firebase features, allowing them to build feature-rich apps that meet modern user expectations.

Code:

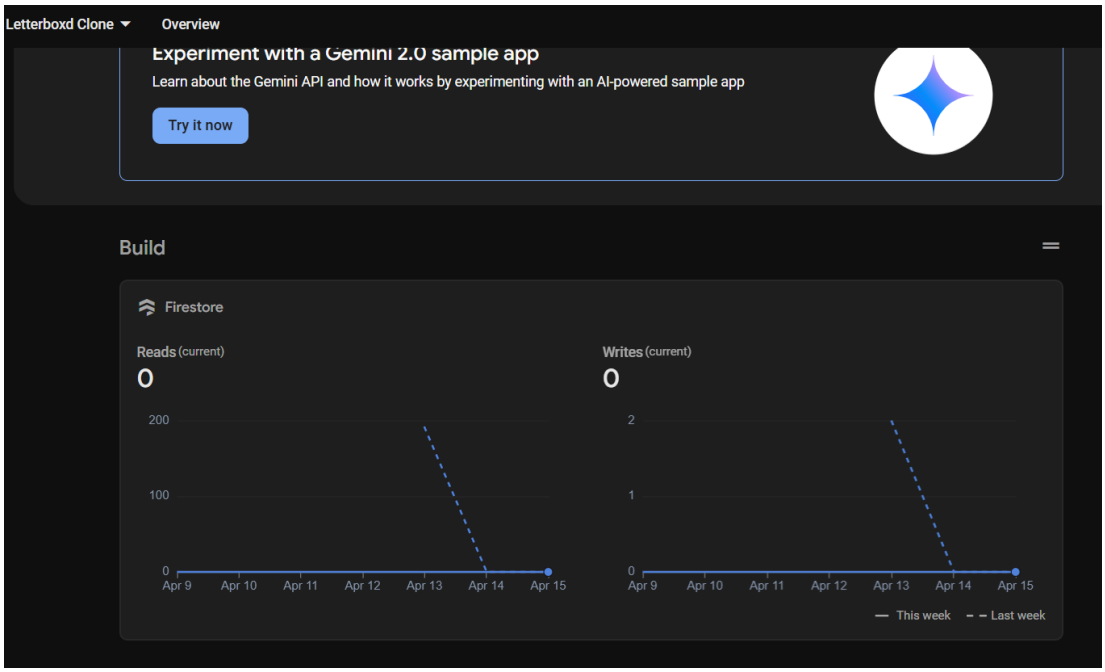
AUTHSERVICE.DART:

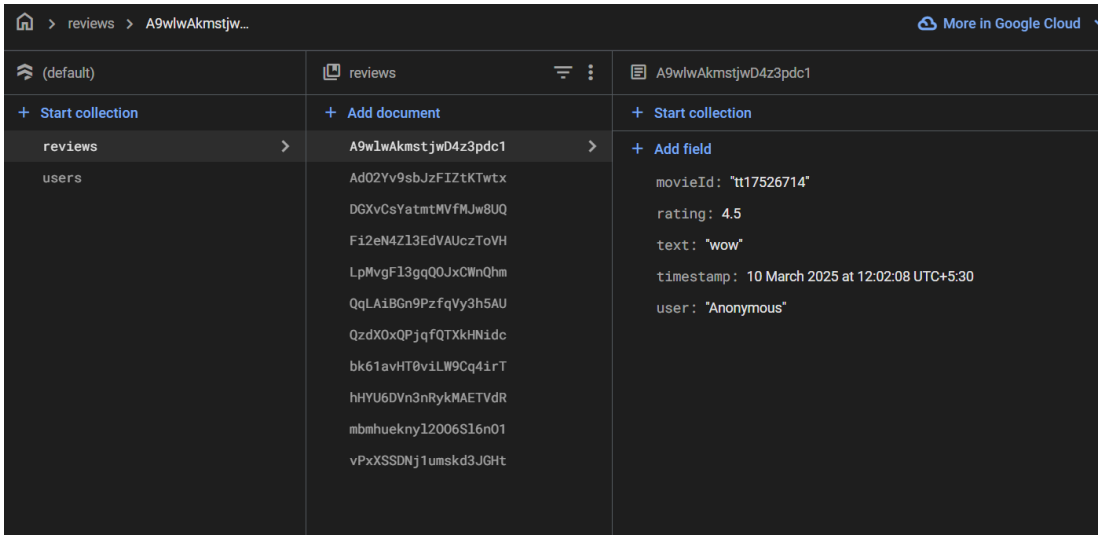
```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
class AuthService {
  final FirebaseAuth _auth = FirebaseAuth.instance;
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;

  // Sign up with email, username, and password
  Future<User?> signUp(String email, String username, String password) async {
    try {
      // Step 1: Create user with email and password
      UserCredential userCredential = await _auth.createUserWithEmailAndPassword(
        email: email,
        password: password,
      );

      // Step 2: Save additional user data (username) to Firestore
      await _firestore.collection('users').doc(userCredential.user!.uid).set({
        'email': email,
        'username': username,
        'createdAt': Timestamp.now(),
      });

      return userCredential.user;
    } catch (e) {
      print("Error during sign-up: $e");
      return null;
    }
  }

  // Sign in with username and password
  Future<User?> signIn(String username, String password) async {
    try {
      // Step 1: Retrieve email associated with the username from Firestore
      QuerySnapshot userQuery = await _firestore
          .collection('users')
          .where('username', isEqualTo: username)
          .limit(1)
          .get();

      if (userQuery.docs.isEmpty) {
        print("No user found with this username.");
        return null;
      }
```

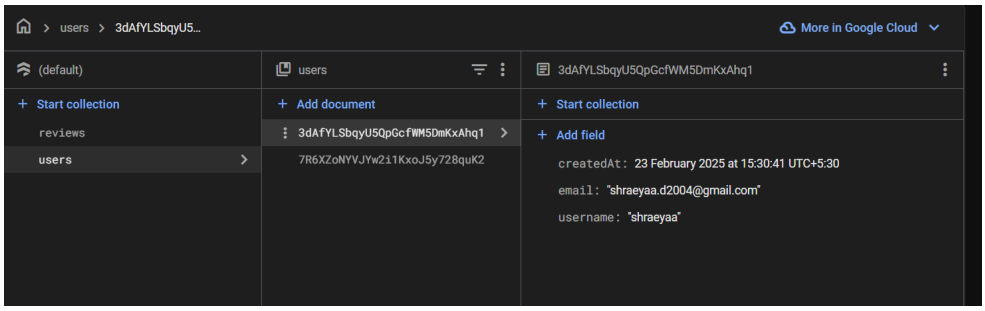
```
      String email = userQuery.docs.first['email'];

      // Step 2: Sign in with email and password
      UserCredential userCredential = await _auth.signInWithEmailAndPassword(
        email: email,
        password: password,
      );

      return userCredential.user;
    } catch (e) {
      print("Error during sign-in: $e");
      return null;
    }
  }

  // Get the current user's username from Firestore
  Future<String?> getUsername(String uid) async {
    try {
      DocumentSnapshot userDoc = await _firestore.collection('users').doc(uid).get();
      return userDoc['username'];
    } catch (e) {
      print("Error fetching username: $e");
      return null;
    }
  }

  // Sign out
  Future<void> signOut() async {
    await _auth.signOut();
  }

  // Get the current user
  User? getCurrentUser() {
    return _auth.currentUser;
  }
}
```

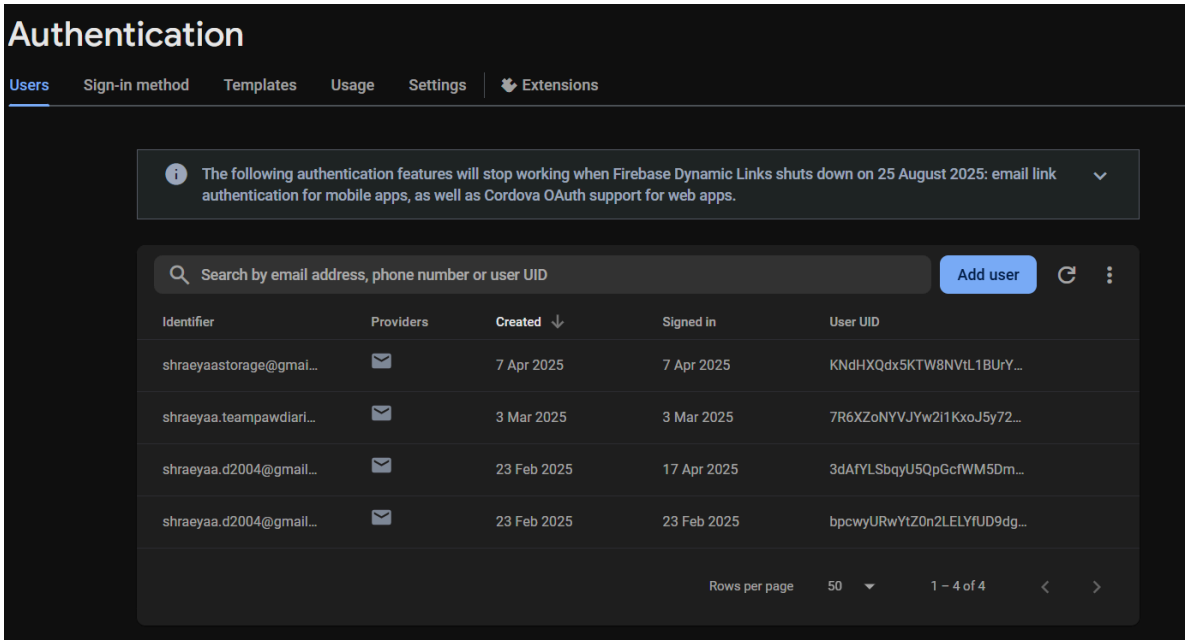
SHRAEYAA DHAIGUDE D15A 15

Screenshots:









Conclusion:
This experiment provided comprehensive, hands-on experience in integrating Firebase services into a Flutter application for both Android and iOS platforms. By incorporating Firebase Authentication, Firestore, and other cloud-based services, it enabled the development of a robust backend system that supports secure user registration, data storage, and real-time data updates. The integration allowed the app to benefit from Firebase's powerful and scalable infrastructure, which is essential for building modern, responsive, and data-driven mobile applications. Additionally, working with Firebase helped in understanding how cloud-based services can enhance the overall functionality, reliability, and user experience of cross-platform mobile apps.