EXPERIMENT 6

Aim: To connect flutter UI with firebase database

Theory:
Connecting a Flutter application to a Firebase database allows for seamless real-time data storage and retrieval, making apps dynamic and responsive. Firebase offers two main database services: Cloud Firestore (a flexible, scalable NoSQL cloud database) and Realtime Database (a tree-structured JSON database for real-time syncing).

In Flutter, Firebase integration is achieved using the firebase_core and cloud_firestore (or firebase_database) packages. After initializing Firebase in the app, data can be added, read, updated, and deleted directly through Firebase methods. Flutter widgets can be connected to database streams, ensuring that any changes in the database are instantly reflected in the UI.

This integration enhances app functionality by enabling persistent data storage, user-specific content, and real-time updates, making it essential for building modern mobile applications.

Code:

signup.dart
The signup.dart file connects the Flutter UI with Firebase Authentication and Cloud Firestore. When a user signs up, their email and password are authenticated using FirebaseAuth. After successful registration, the user's additional details like name and email are stored in Firestore under the users collection using their unique UID. This demonstrates how user registration data is securely stored in the Firebase database through Flutter.

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:google_sign_in/google_sign_in.dart';

class Login extends StatefulWidget {
  const Login({super.key});

  @override
  State<Login> createState() => _LoginState();
}

class _LoginState extends State<Login> {
  bool isLogin = true;
```

```
final TextEditingController emailController = TextEditingController();
final TextEditingController passwordController = TextEditingController();

@override
void dispose() {
  emailController.dispose();
  passwordController.dispose();
  super.dispose();
}

Future<UserCredential?> signInWithGoogle() async {
  try {
    final GoogleSignInAccount? googleUser = await GoogleSignIn().signIn();

    if (googleUser == null) {
      print("User canceled the Google sign-in.");
      return null;
    }
    final GoogleSignInAuthentication googleAuth =
        await googleUser.authentication;
    final AuthCredential credential = GoogleAuthProvider.credential(
      accessToken: googleAuth.accessToken,
      idToken: googleAuth.idToken,
    );
    return await FirebaseAuth.instance.signInWithCredential(credential);
  } catch (e) {
    print("Google sign-in failed: $e");
    return null;
  }
}

Future<void> createUserWithEmailAndPassword() async {
  try {
    if (isLogin) {
      await FirebaseAuth.instance.signInWithEmailAndPassword(
        email: emailController.text.trim(),
        password: passwordController.text.trim(),
      );
    } else {
```

```
      await FirebaseAuth.instance.createUserWithEmailAndPassword(
        email: emailController.text.trim(),
        password: passwordController.text.trim(),
      );
    }
  } on FirebaseAuthException catch (e) {
    print(e.message);
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.blueGrey[900],
    body: Center(
      child: Padding(
        padding: const EdgeInsets.all(20.0),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Icon(
              Icons.local_shipping,
              size: 80,
              color: Colors.orangeAccent,
            ),
            const SizedBox(height: 20),
            Text(
              isLogin ? 'Truck Map Login' : 'Truck Map Sign Up',
              style: const TextStyle(
                fontSize: 28,
                fontWeight: FontWeight.bold,
                color: Colors.white,
              ),
            ),
            const SizedBox(height: 20),
            _buildTextField(emailController, 'Enter your Email', Icons.email),
            const SizedBox(height: 15),
            _buildTextField(
                passwordController, 'Enter your Password', Icons.lock,
```

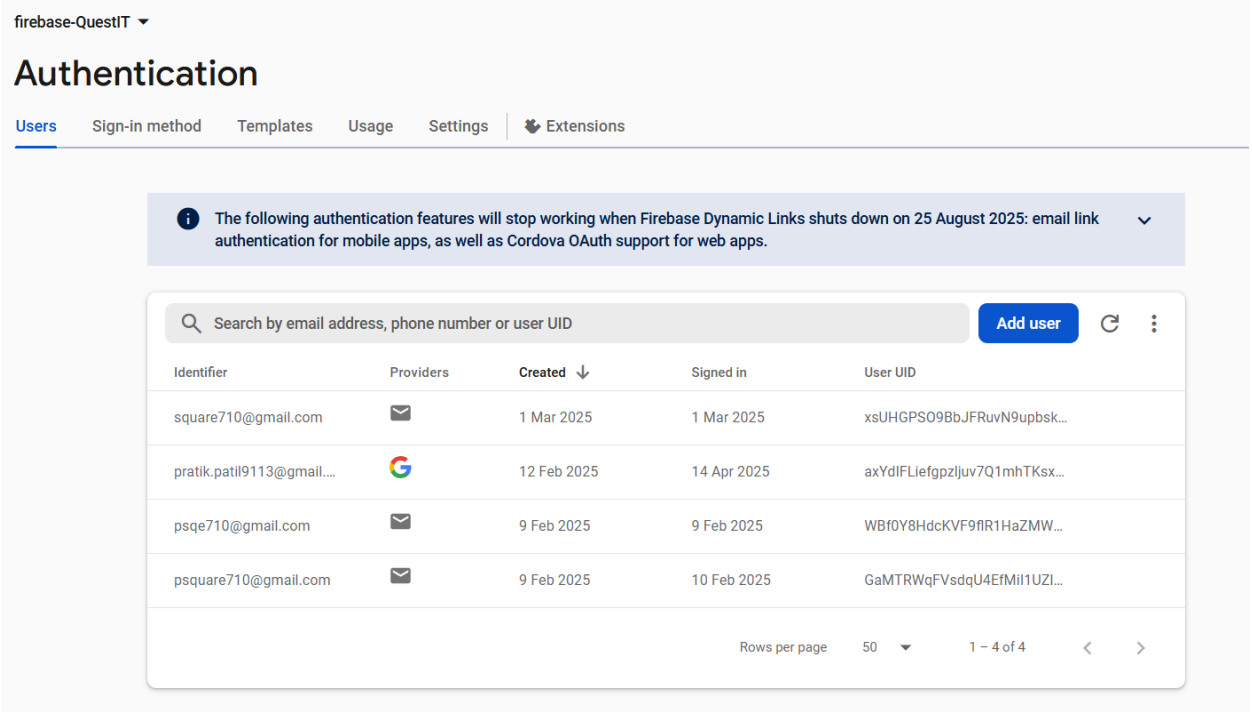
```
            isPassword: true),
        const SizedBox(height: 20),
        _buildButton(isLogin ? 'Login' : 'Sign Up',
            createUserWithEmailAndPassword),
        const SizedBox(height: 10),
        Row(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text(
              isLogin ? 'New here?' : 'Already registered?',
              style: const TextStyle(color: Colors.white),
            ),
            TextButton(
              onPressed: () {
                setState(() {
                  isLogin = !isLogin;
                });
              },
              child: Text(
                isLogin ? 'Sign Up' : 'Login',
                style: const TextStyle(
                    color: Colors.orangeAccent,
                    fontWeight: FontWeight.bold),
              ),
            ),
          ],
        ),
        const SizedBox(height: 10),
        _buildButton('Sign in with Google', signInWithGoogle,
            isGoogle: true),
      ],
    ),
  ),
 ),
);
}

Widget _buildTextField(
  TextEditingController controller, String hint, IconData icon,
```

```
    {bool isPassword = false}) {
  return TextField(
    controller: controller,
    obscureText: isPassword,
    decoration: InputDecoration(
      labelText: hint,
      prefixIcon: Icon(icon, color: Colors.orangeAccent),
      filled: true,
      fillColor: Colors.blueGrey[700],
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(10),
        borderSide: BorderSide.none,
      ),
      labelStyle: const TextStyle(color: Colors.white70),
    ),
    style: const TextStyle(color: Colors.white),
  );
}

Widget _buildButton(String text, Function() onPressed,
    {bool isGoogle = false}) {
  return SizedBox(
    width: double.infinity,
    height: 50,
    child: ElevatedButton(
      onPressed: onPressed,
      style: ElevatedButton.styleFrom(
        backgroundColor: isGoogle ? Colors.white : Colors.orangeAccent,
        foregroundColor: isGoogle ? Colors.black : Colors.white,
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(10),
        ),
      ),
      child: Row(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          if (isGoogle)
            Icon(
              Icons.g_mobiledata,
```

```
    color: Colors.blue,
  ),
if (isGoogle) const SizedBox(width: 10),
Text(
  text,
  style: const TextStyle(fontSize: 18, fontWeight: FontWeight.bold),
```



Conclusion:

This experiment successfully demonstrates how to integrate Firebase services with a Flutter application. By implementing user signup, login, and data submission features, we explored the use of Firebase Authentication for secure user management and Cloud Firestore for real-time database storage. The flow between screens and backend interaction highlights the power and simplicity of using Firebase in Flutter apps, making it an ideal choice for building scalable and responsive mobile applications.