Experiment No. 6

Aim: To Connect Flutter UI with firebase database

Theory:

→ Connecting Flutter UI with Firebase–

Connecting a Flutter UI with a Firebase database involves several steps, including setting up a Firebase project, configuring your Flutter app, and implementing the necessary code to interact with the Firebase database. Here's a step-by-step guide to help you connect Flutter UI with a Firebase database.

➢ Step 1: Set up Firebase Project –

✓ Create a Firebase Project:

- Go to the Firebase Console(https://console.firebase.google.com/).
- Click on "Add Project" and follow the prompts to create a new project.

✓ Add your Flutter app to Firebase:

- After creating the project, click on "Add App" and select the Flutter icon.
- Follow the setup instructions to add the necessary configuration files to your Flutter project.

✓ Enable Firebase services:

- In the Firebase Console, navigate to your project and click on "Develop" > "Database."
- Create a Firestore database or a Realtime Database, depending on your needs.

➢ Step 2: Set up Flutter Project –

✓ Add Dependencies:

- Open your 'pubspec.yaml' file and add the following dependencies.

Dependencies:
Firebase_core: ^latest_version
Cloud_firestore: ^latest_version

- Run `flutter pub get` in the terminal to install the dependencies.

✓ Initialize Firebase in your Flutter App:

- In your main Dart file (usually `main.dart`), initialize Firebase by adding the following code:

```
import 'package:firebase_core/firebase_core.dart';
void main() async {
```

```
 WidgetsFlutterBinding.ensureInitialized();
 await Firebase.initializeApp();
 runApp(MyApp());
}
```

➢ Step 3: Connect Flutter UI with Firebase –

✓ Firestore Database Example:

- If you are using Firestore, you can use the following example to read and write data.

```
import 'package:cloud_firestore/cloud_firestore.dart';
Future<void> addData() async {
 await FirebaseFirestore.instance.collection('your_collection').add({
 'field1': 'value1',
 'field2': 'value2',
 });
}
StreamBuilder(
 stream: FirebaseFirestore.instance.collection('your_collection').snapshots(),
 builder: (context, snapshot) {
 if (!snapshot.hasData) {
 return CircularProgressIndicator(); }
 var documents = snapshot.data.docs; },);
```

✓ Realtime Database Example:

- If you are using the Realtime Database, you can use the following example:

```
import 'package:firebase_database/firebase_database.dart';
DatabaseReference databaseReference =
FirebaseDatabase.instance.reference();
databaseReference.child('your_collection').push().set({
 'field1': 'value1',
 'field2': 'value2',
});
DatabaseReference databaseReference =
FirebaseDatabase.instance.reference();
databaseReference.child('your_collection').once().then((DataSnapshot
snapshot) {
});
```

```
# The following adds the Cupertino Icons font to your application.
# Use with the CupertinoIcons class for iOS style icons.
cupertino_icons: ^1.0.2
firebase_core: ^2.15.1
firebase_auth: ^4.7.3
loading_overlay: ^0.3.0
rflutter_alert: ^2.0.7
```

```
lass SignUpScreen extends StatefulWidget {
 const SignUpScreen({super.key});
 static String id = 'signup_screen';

 @override
 State<SignUpScreen> createState() => _SignUpScreenState();


lass _SignUpScreenState extends State<SignUpScreen> {
 final _auth = FirebaseAuth.instance;
 late String _email;
 late String _password;
 late String _confirmPass;
 bool _saving = false;
```

➢ Step 4: Authentication –

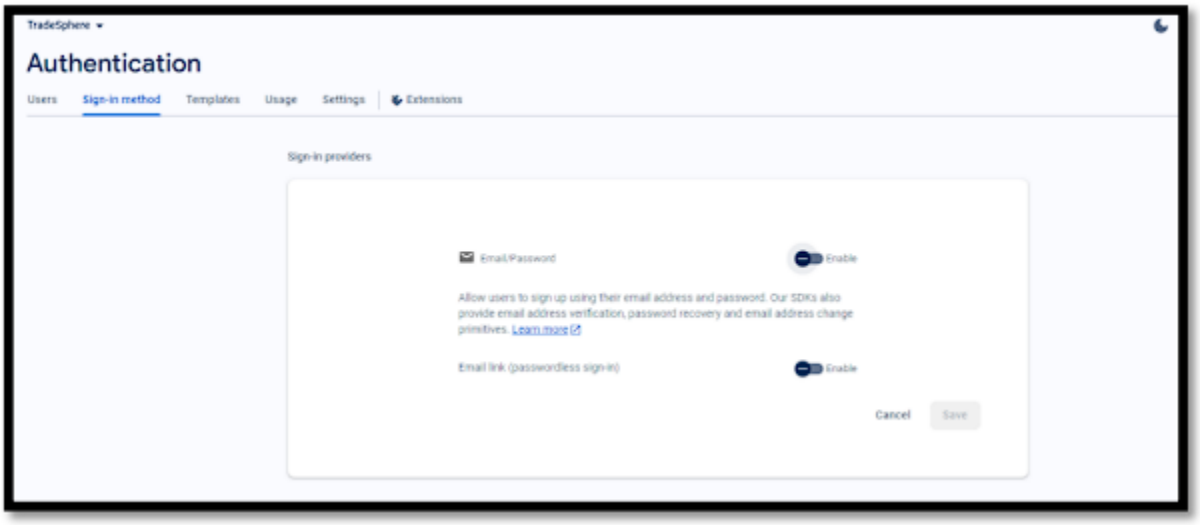
If we need user authentication, Firebase provides authentication services. We can integrate Firebase Authentication to secure your app.

import 'package:firebase_auth/firebase_auth.dart';

Future<void> signIn(String email, String password) async {
 try {
 await FirebaseAuth.instance.signInWithEmailAndPassword(
 email: email,
 password: password,

```
 );
 } catch (e) {
 print(e);
 }
}
```

Remember to replace placeholder values like “your_collection”, “field1”, “value1”, etc., with your actual database collection and field names.



• Conclusion:

So, in summary, we’ve learned how to link our Flutter user interface with a Firebase database. Connecting the visual part of our app with the storage and retrieval of data is a crucial step, and understanding this connection allows us to create dynamic and interactive mobile applications.