PRANAV RAIKAR

D15A 47

Experiment 6

AIM: To Connect Flutter UI with fireBase database

Theory:

Connecting a Flutter app to Firebase involves several steps, including setting up a

Firebase project, configuring the Flutter app to use Firebase, and integrating Firebase

SDKs into your Flutter code.

1. Firebase Setup: First, you need to create a Firebase project in the Firebase

console (https://console.firebase.google.com/). This project will host your app's

data and services.

2. Add Firebase to Flutter App: You'll need to add the Firebase SDK to your Flutter

app. This involves adding Firebase configuration files to your project and

updating your Flutter app's dependencies to include Firebase SDKs.

3. Initialize Firebase: In your Flutter app, you'll initialize Firebase by calling

Firebase.initializeApp(). This should typically be done at the beginning of your

app's lifecycle, such as in the main() function or in the initState() method of your

main widget.

4. Use Firebase Services: Once Firebase is initialized, you can use various Firebase

services in your Flutter app, such as Authentication, Firestore, Realtime Database,

Cloud Storage, Cloud Messaging, and more.

**Main.dart:**

import 'package:flutter/material.dart';

import 'package:flutter\_stripe/flutter\_stripe.dart';

import 'package:food\_panda/admin/admin\_login.dart';

import 'package:food\_panda/pages/bottomnav.dart';

import 'package:food\_panda/pages/onboard.dart';

import 'package:firebase\_core/firebase\_core.dart';

import 'package:food\_panda/widget/app\_constant.dart';

import 'firebase\_options.dart';

void main() async {

WidgetsFlutterBinding.ensureInitialized();

Stripe.publishableKey = publishableKey;

await Firebase.initializeApp(

options: FirebaseOptions(

apiKey: 'AIzaSyBftcoBEue-M8hQfcB-M-7ZAbhm\_5PVmck',

appId: '1:726847533270:android:f8faab0dba68351c9b1977',

messagingSenderId: '726847533270',

projectId: 'fooddeliveryapppranav',

storageBucket: 'fooddeliveryapppranav.appspot.com',

),

);

runApp(const MyApp());

}

class MyApp extends StatelessWidget {

const MyApp({super.key});

// This widget is the root of your application.

@override

Widget build(BuildContext context) {

return MaterialApp(

debugShowCheckedModeBanner: false,

title: 'Flutter Demo',

theme: ThemeData(

// This is the theme of your application.

//

// TRY THIS: Try running your application with "flutter run". You'll see

// the application has a purple toolbar. Then, without quitting the app,

// try changing the seedColor in the colorScheme below to Colors.green

// and then invoke "hot reload" (save your changes or press the "hot

// reload" button in a Flutter-supported IDE, or press "r" if you used

// the command line to start the app).

//

// Notice that the counter didn't reset back to zero; the application

// state is not lost during the reload. To reset the state, use hot

// restart instead.

//

// This works for code too, not just values: Most code changes can be

// tested with just a hot reload.

colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),

useMaterial3: true,

),

home: Onboard(),

// home: AdminLogin(),

// home: BottomNav(),

);

}

}

class MyHomePage extends StatefulWidget {

const MyHomePage({super.key, required this.title});

// This widget is the home page of your application. It is stateful, meaning

// that it has a State object (defined below) that contains fields that affect

// how it looks.

// This class is the configuration for the state. It holds the values (in this

// case the title) provided by the parent (in this case the App widget) and

// used by the build method of the State. Fields in a Widget subclass are

// always marked "final".

final String title;

@override

State<MyHomePage> createState() => \_MyHomePageState();

}

class \_MyHomePageState extends State<MyHomePage> {

int \_counter = 0;

void \_incrementCounter() {

setState(() {

// This call to setState tells the Flutter framework that something has

// changed in this State, which causes it to rerun the build method below

// so that the display can reflect the updated values. If we changed

// \_counter without calling setState(), then the build method would not be

// called again, and so nothing would appear to happen.

\_counter++;

});

}

@override

Widget build(BuildContext context) {

// This method is rerun every time setState is called, for instance as done

// by the \_incrementCounter method above.

//

// The Flutter framework has been optimized to make rerunning build methods

// fast, so that you can just rebuild anything that needs updating rather

// than having to individually change instances of widgets.

return Scaffold(

appBar: AppBar(

// TRY THIS: Try changing the color here to a specific color (to

// Colors.amber, perhaps?) and trigger a hot reload to see the AppBar

// change color while the other colors stay the same.

backgroundColor: Theme.of(context).colorScheme.inversePrimary,

// Here we take the value from the MyHomePage object that was created by

// the App.build method, and use it to set our appbar title.

title: Text(widget.title),

),

body: Center(

// Center is a layout widget. It takes a single child and positions it

// in the middle of the parent.

child: Column(

// Column is also a layout widget. It takes a list of children and

// arranges them vertically. By default, it sizes itself to fit its

// children horizontally, and tries to be as tall as its parent.

//

// Column has various properties to control how it sizes itself and

// how it positions its children. Here we use mainAxisAlignment to

// center the children vertically; the main axis here is the vertical

// axis because Columns are vertical (the cross axis would be

// horizontal).

//

// TRY THIS: Invoke "debug painting" (choose the "Toggle Debug Paint"

// action in the IDE, or press "p" in the console), to see the

// wireframe for each widget.

mainAxisAlignment: MainAxisAlignment.center,

children: <Widget>[

const Text(

'You have pushed the button this many times:',

),

Text(

'$\_counter',

style: Theme.of(context).textTheme.headlineMedium,

),

],

),

),

floatingActionButton: FloatingActionButton(

onPressed: \_incrementCounter,

tooltip: 'Increment',

child: const Icon(Icons.add),

), // This trailing comma makes auto-formatting nicer for build methods.

);

}

}

**Database.dart:**

import 'package:cloud\_firestore/cloud\_firestore.dart';

class DatabaseMethods {

Future addUserDetail(Map<String, dynamic> userInfoMap, String id) async {

return await FirebaseFirestore.instance

.collection('users')

.doc(id)

.set(userInfoMap);

}

UpdateUserwallet(String id, String amount) async {

return await FirebaseFirestore.instance

.collection("users")

.doc(id)

.update({"Wallet": amount});

}

Future addFoodItem(Map<String, dynamic> userInfoMap, String name) async {

return await FirebaseFirestore.instance.collection(name).add(userInfoMap);

}

Future<Stream<QuerySnapshot>> getFoodItem(String name) async {

return await FirebaseFirestore.instance.collection(name).snapshots();

}

}

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Conclusion :Thus I learnt to Connect Flutter UI with fireBase database