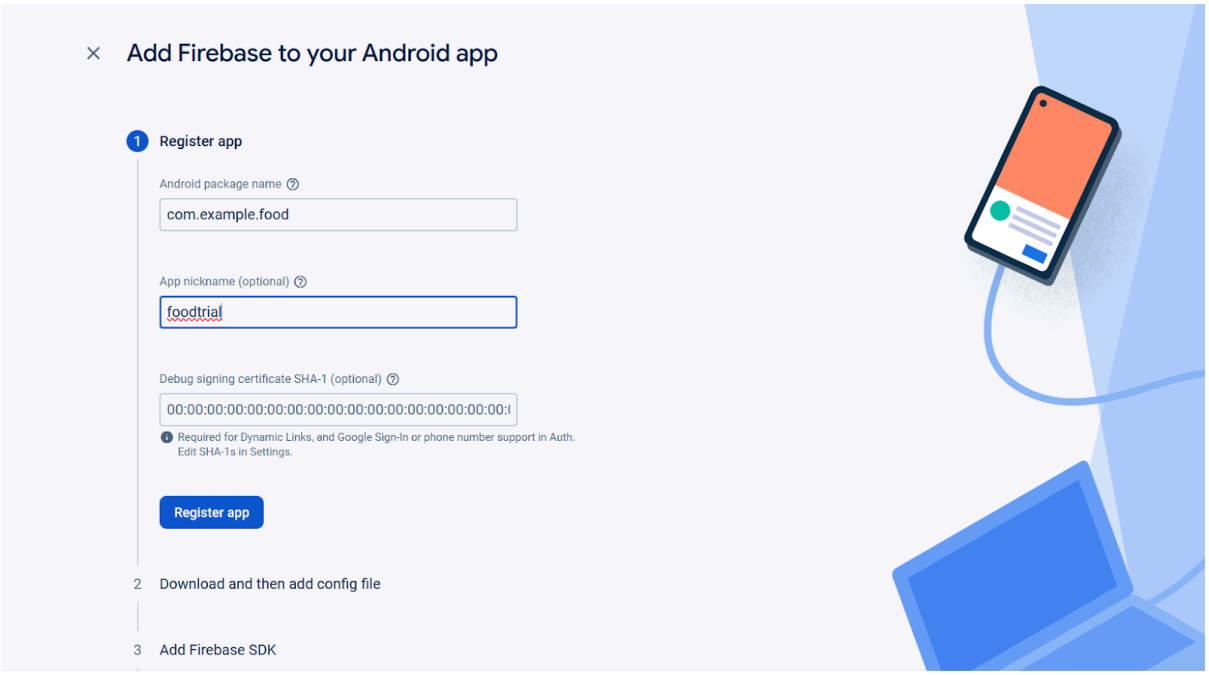
Name - Sejal Maurya **Div - D15A**
Roll No - 35 **Batch - B**

Experiment No -06
Firebase Database

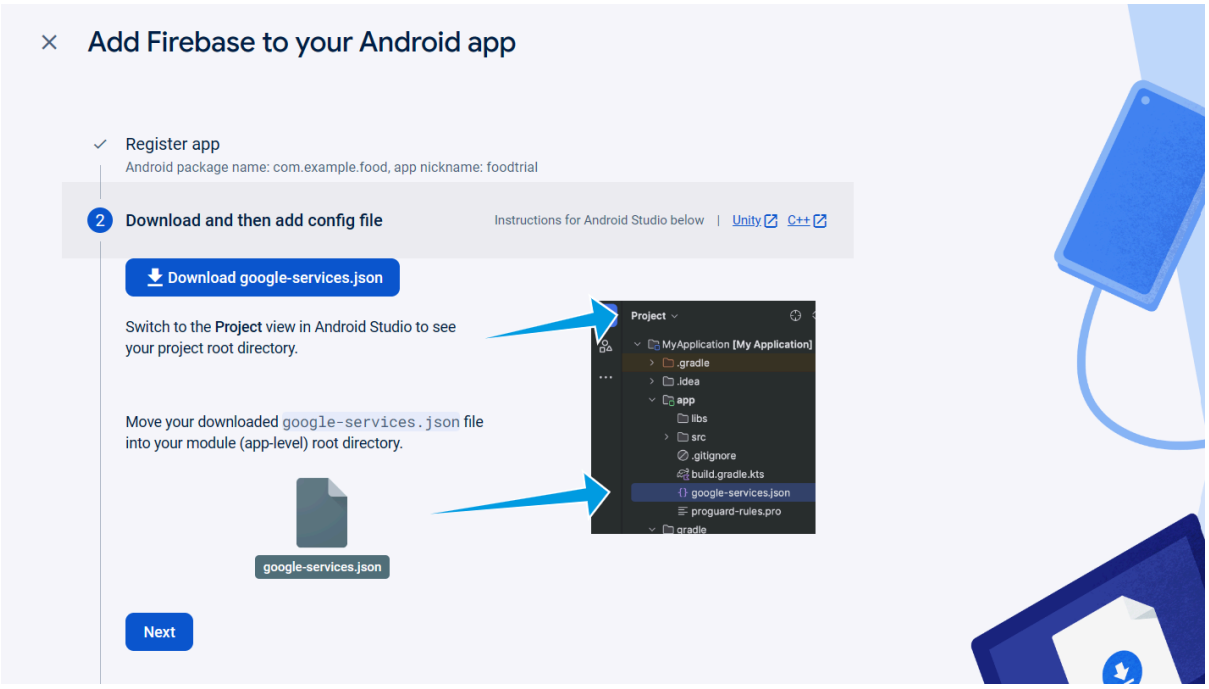
Aim - To connect flutter UI with Firebase database.

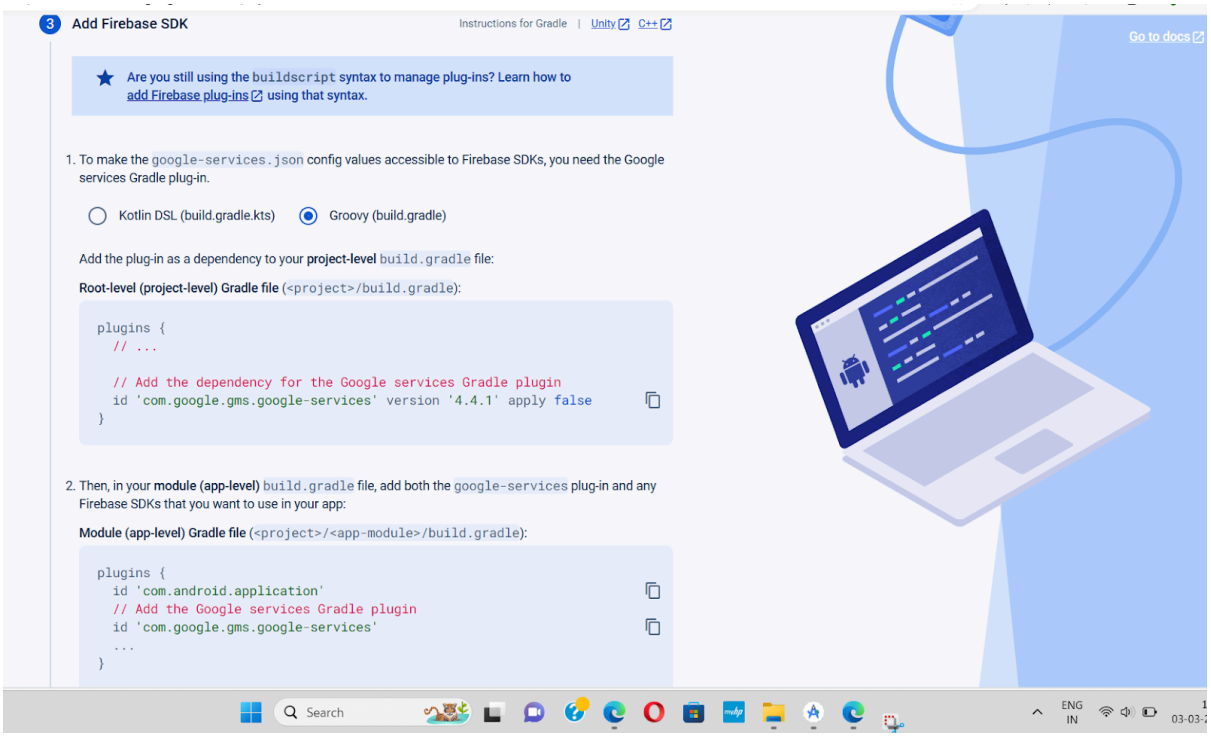
Theory

Firebase Realtime Database is a powerful and easy-to-use solution for developers looking to add real-time data synchronization and offline support to their web and mobile applications. It simplifies the process of building real-time collaborative features and allows developers to focus on creating engaging user experiences.



Download the google-services.json

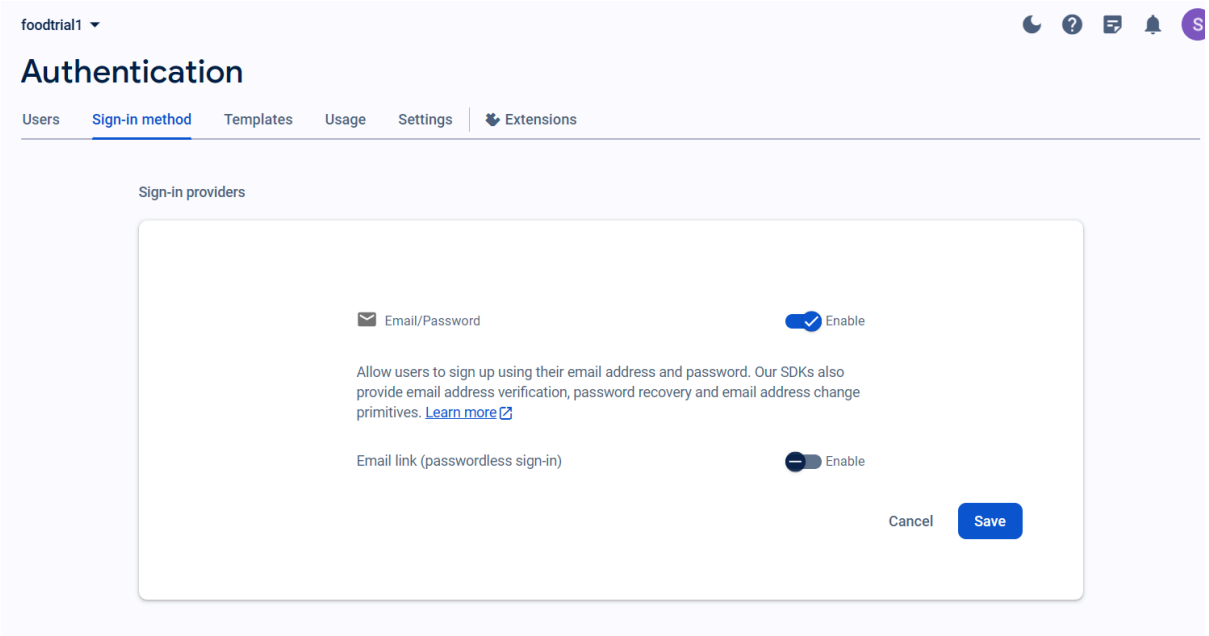


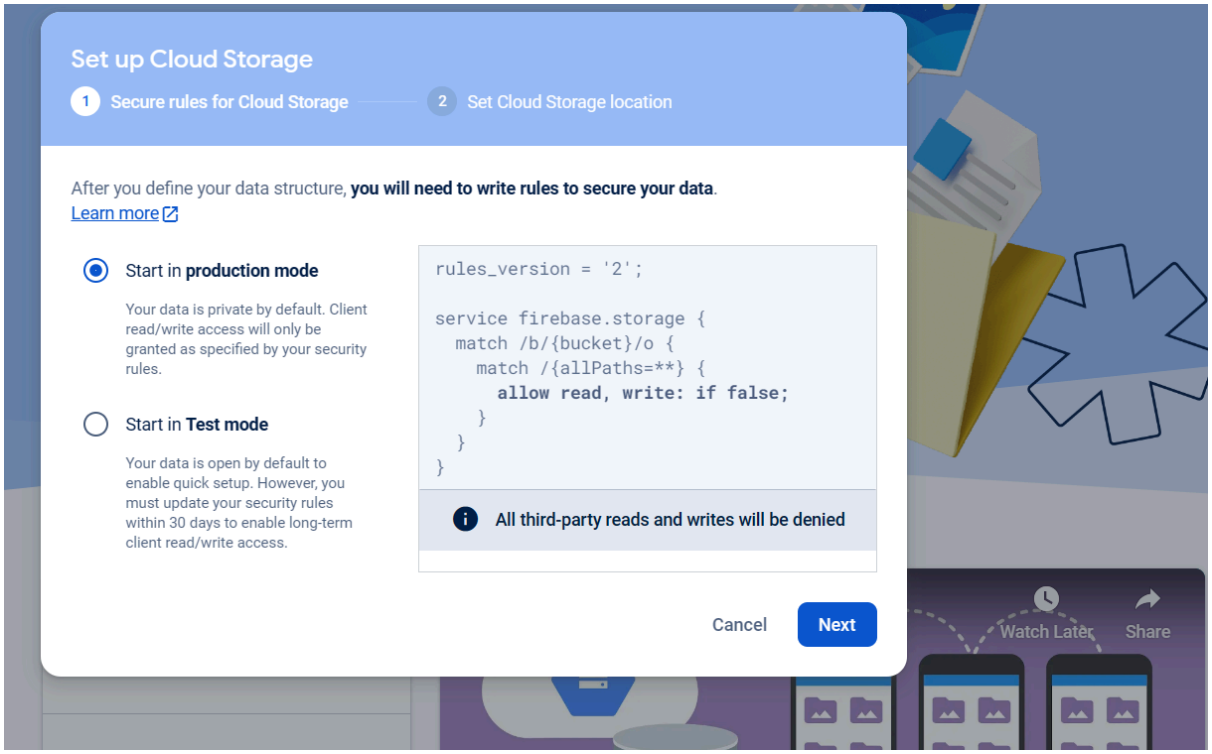


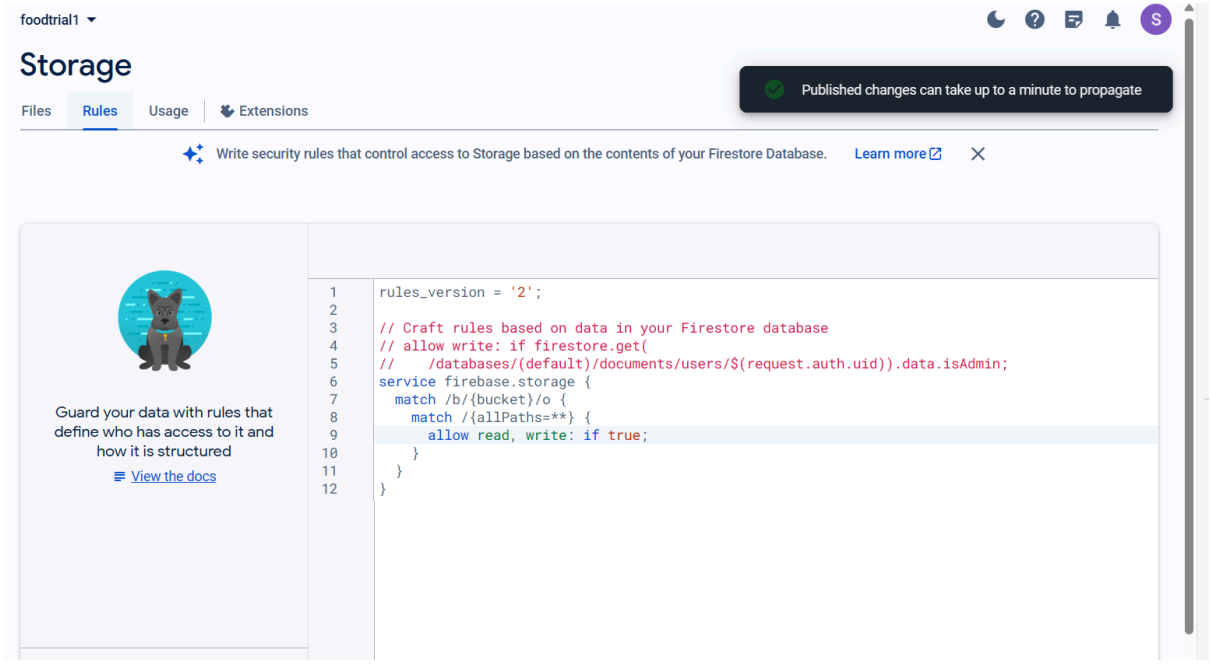
Add dependencies in project-level gradle and app-level gradle.

```
    dependencies {
        classpath 'com.google.gms:google-services:4.4.1'
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
    }
}
```

```
dependencies {
    implementation platform('com.google.firebase:firebase-bom:32.7.3')
    implementation 'com.google.firebase:firebase-analytics'
}
```




Set up Cloud Storage
1 Secure rules for Cloud Storage
2 Set Cloud Storage location
After you define your data structure, you will need to write rules to secure your data.
Learn more
Start in production mode
Your data is private by default. Client read/write access will only be granted as specified by your security rules.
Start in Test mode
Your data is open by default to enable quick setup. However, you must update your security rules within 30 days to enable long-term client read/write access.
rules_version = '2';
service firebase.storage {
match /b/{bucket}/o {
match /{allPaths=**} {
allow read, write: if false;
}
}
}
All third-party reads and writes will be denied
Cancel
Next
Watch Later
Share


foodtrial1
Storage
Files
Rules
Usage
Extensions
Published changes can take up to a minute to propagate
Write security rules that control access to Storage based on the contents of your Firestore Database.
Learn more
Guard your data with rules that define who has access to it and how it is structured
View the docs
rules_version = '2';
// Craft rules based on data in your Firestore database
// allow write: if firestore.get(
// /databases/(default)/documents/users/$(request.auth.uid)).data.isAdmin;
service firebase.storage {
match /b/{bucket}/o {
match /{allPaths=**} {
allow read, write: if true;
}
}
}

```dart
import 'package:food/extra/cartpage.dart';
void main()async{
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
      options: FirebaseOptions(
          apiKey:"AIzaSyASrFb41V8ghwpAFuYA9hkzzpzIl-sjjj4",
          appId: "1:116578659366:android:df8de54c41f6bdee97cae7",
          messagingSenderId: "116578659366",
          projectId: "foodtrial1-e8b96")
  );
  runApp(Myapp());
}
```

Add these dependencies in pubspec.yml

```yaml
  # The following adds the Cupertino Icons font to your appl
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.2
  firebase_core: ^2.25.5
  firebase_auth: ^4.17.6
  image_picker: ^1.0.7
  firebase_storage: ^11.6.7
  cloud_firestore: ^4.15.6
  firebase_messaging: ^14.7.17

dev_dependencies:
  flutter_test:
    sdk: flutter

  # The "flutter_lints" package below contains a set of reco
  # encourage good coding practices. The lint set provided b
```

ument 1/1

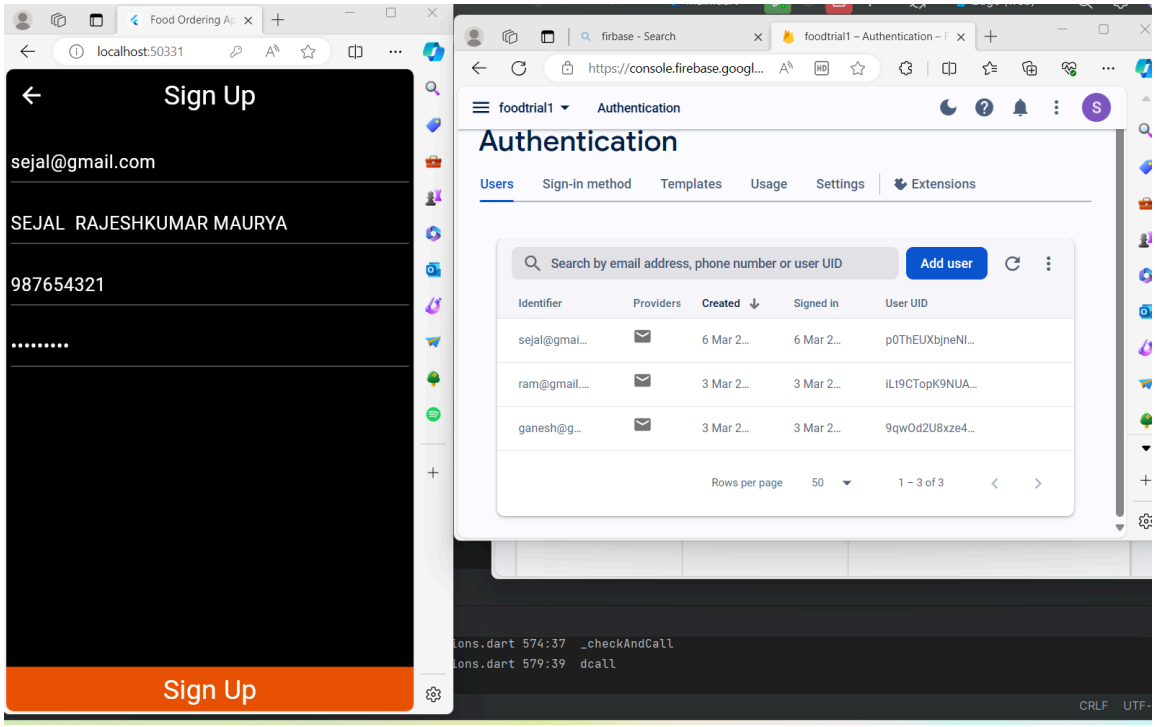
Code -

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


import 'package:firebase_auth/firebase_auth.dart';
import 'package:food/screen/foodinfo.dart';

class LoginPage1 extends StatefulWidget {
  @override
  _LoginPageState createState() => _LoginPageState();
}

class _LoginPageState extends State<LoginPage1> {
  TextEditingController _emailController = TextEditingController();
  TextEditingController _passwordController = TextEditingController();
  final FirebaseAuth _auth = FirebaseAuth.instance;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.black,
      appBar: AppBar(
        backgroundColor: Colors.black,
        leading: Row(
          children: [
            IconButton(
              icon: Icon(Icons.arrow_back, size: 30.0, color: Colors.white,),
              onPressed: () {Navigator.pop(context);},
            ),
          ],
        ),
        title: Text('Welcome To FreshMenu', style: TextStyle(color: Colors.white),),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            TextField(
              style: TextStyle(color: Colors.white),
```

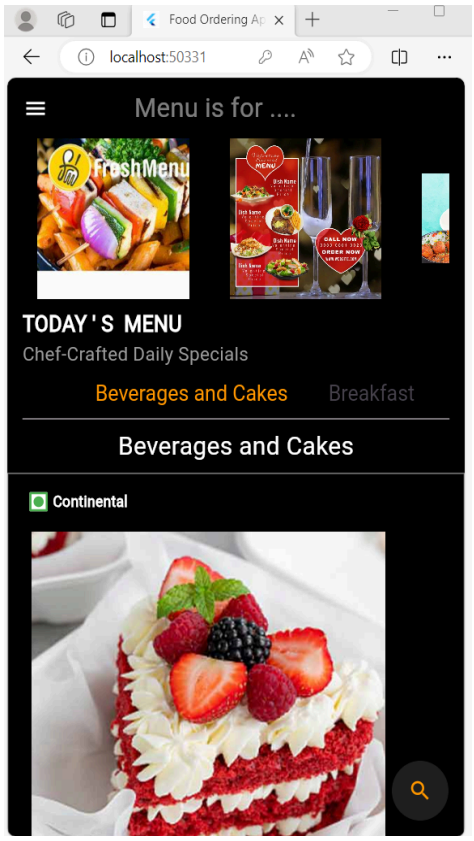
```
          controller: _emailController,
          decoration: InputDecoration(
            labelText: 'Email',
          ),
        ),
        SizedBox(height: 20),
        TextField(
          style: TextStyle(color: Colors.white),
          controller: _passwordController,
          decoration: InputDecoration(
            labelText: 'Password',
          ),
          obscureText: true,
        ),
        SizedBox(height: 20),
        Expanded(
          child: Align(
            alignment: Alignment.bottomCenter,
            child: GestureDetector(
              onTap: _login,
              child: Container(
                height: 50,
                color: Colors.orange[900],
                child: Center(
                  child: Text(
                    'Login',
                    style: TextStyle(color: Colors.white, fontSize: 30),
                  ),
                ),
              ),
            ),
          ),
        ),
      ],
    ),
  ),
);
}

void _login() async {
```

```
    try {
      final String email = _emailController.text.trim();
      final String password = _passwordController.text.trim();

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

      // Check if the user is authenticated
      if (userCredential.user != null) {
        // Navigate to the home page if authentication is successful
        Navigator.pushReplacement(
          context,
          MaterialPageRoute(builder: (context) => FoodInfo()),
        );
      }
    } catch (e) {
      // Handle errors such as invalid credentials, network errors, etc.
      print('Error: $e');
      // You can show a snackbar or dialog to display the error to the user
    }
  }
}
```

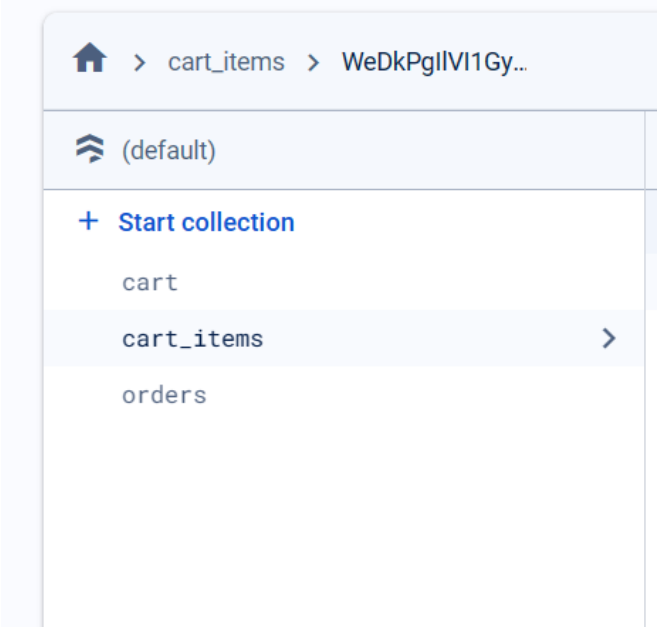
Authentication



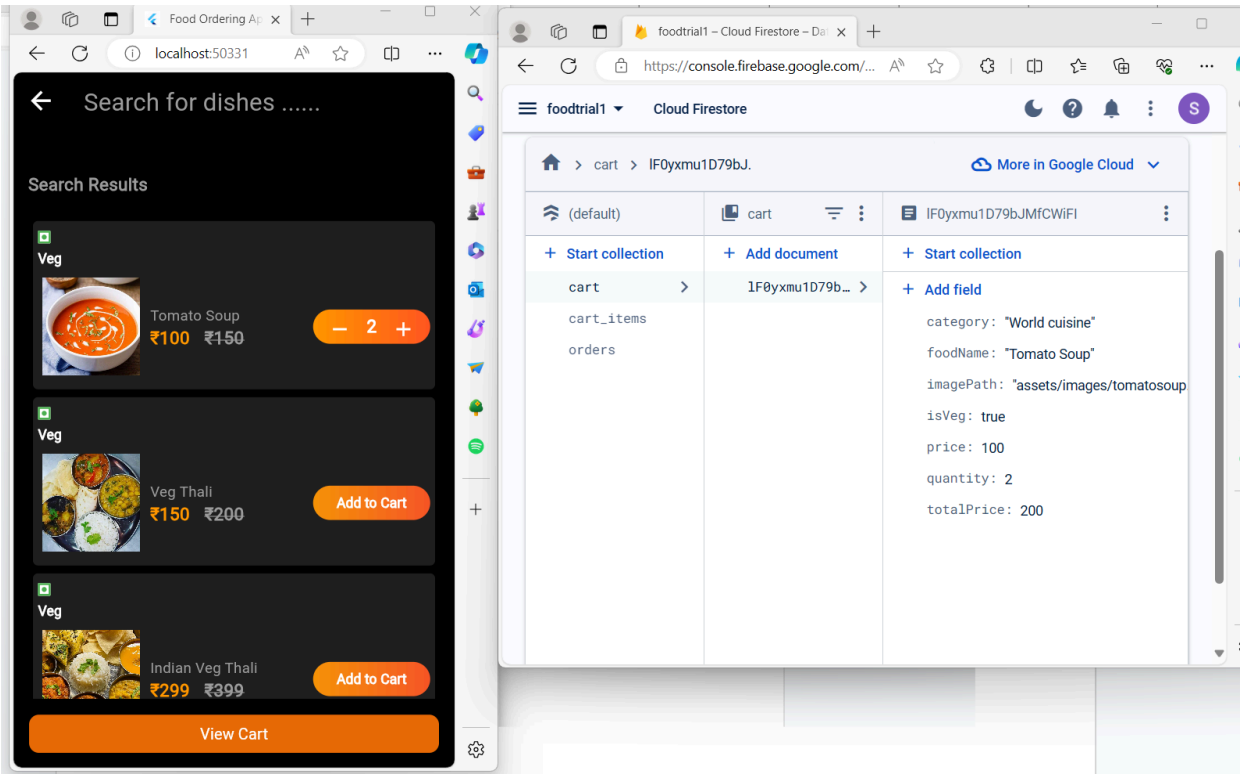
And then when login into the page with right email and password I can moved to the home page



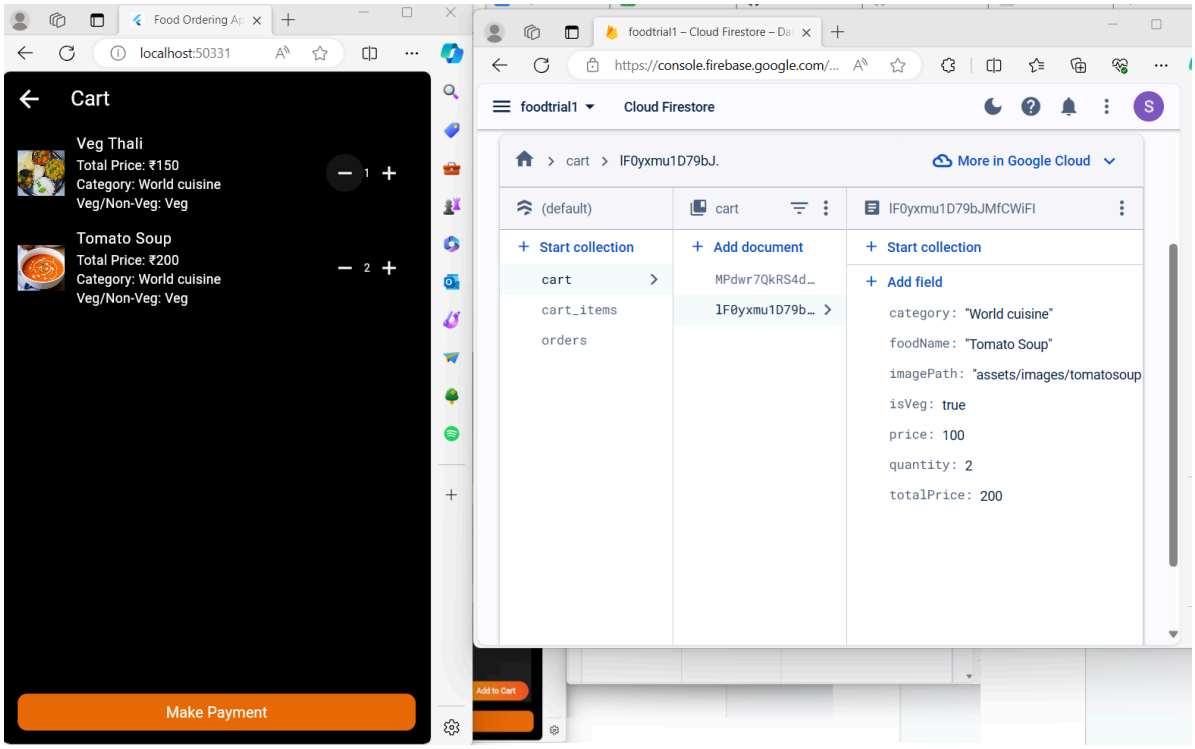
Other database are cart ,cart items, orders



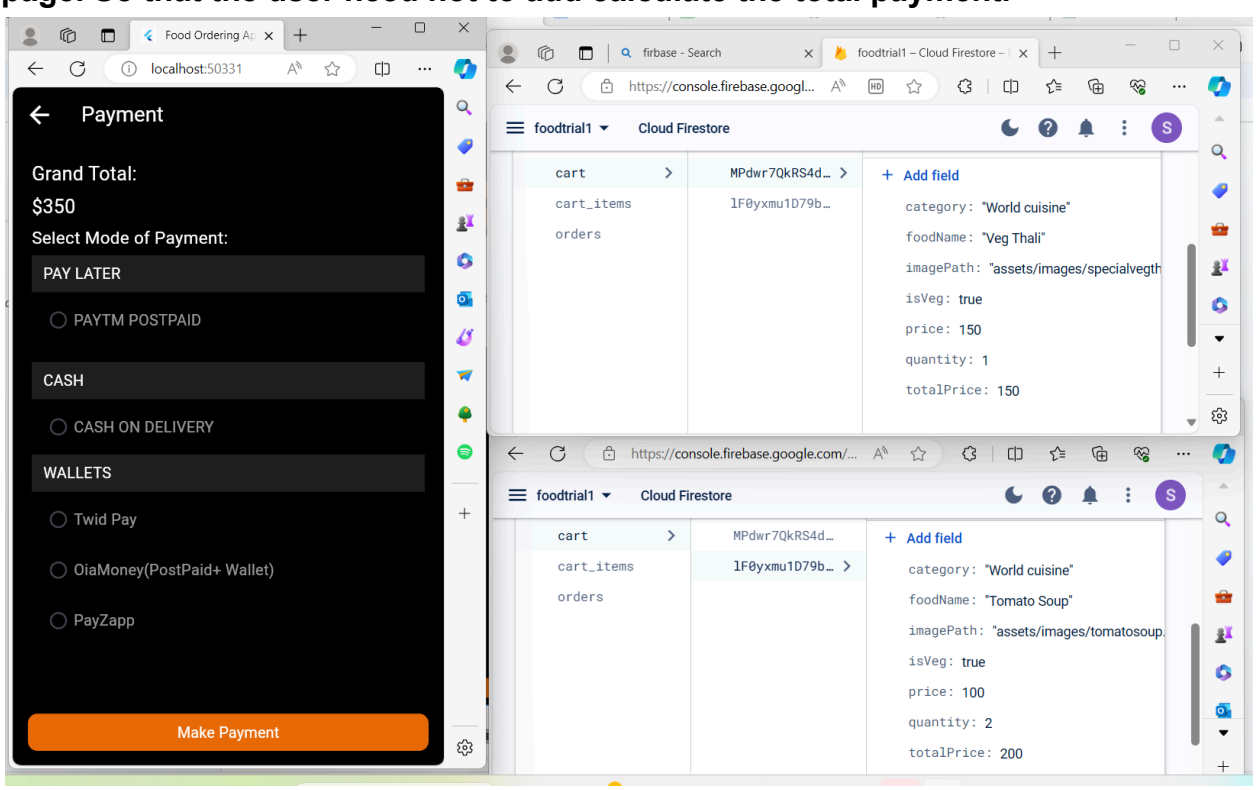
When I added the food items into the cart it stores into the cart database
Here the data is fetch from flutter UI to database



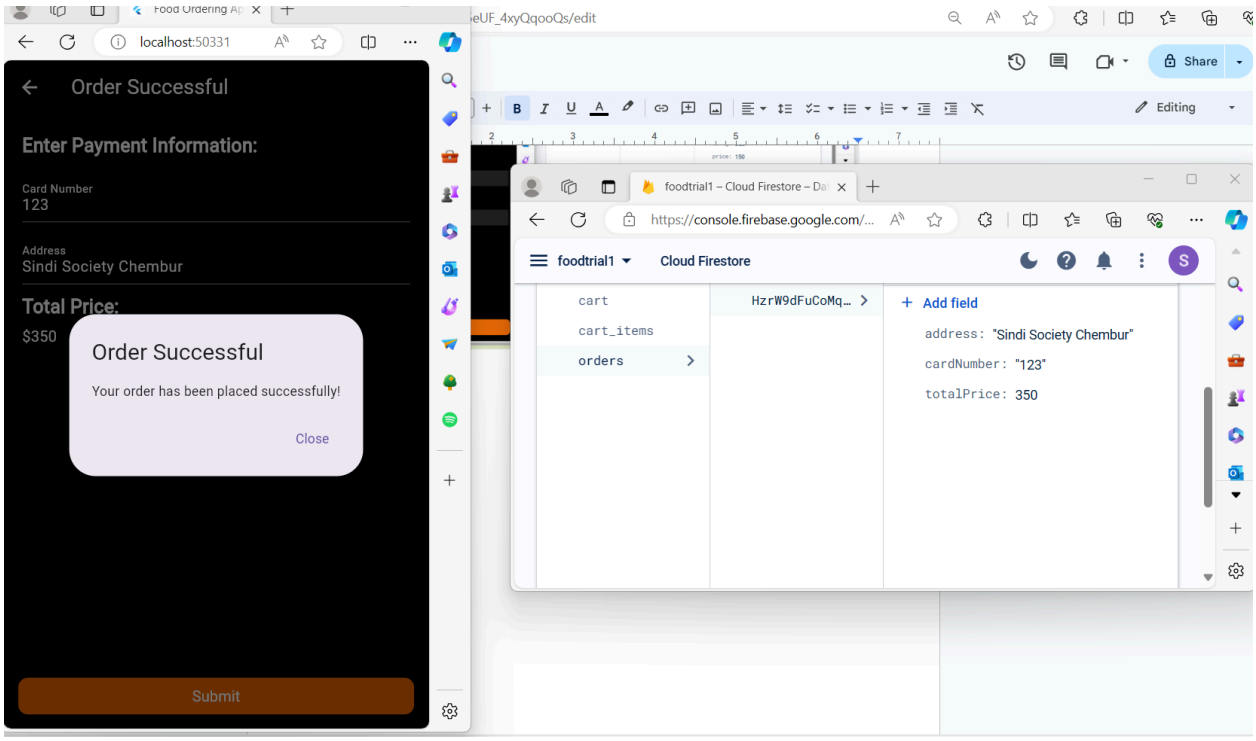
Here the data is fetch from the cloud store to Flutter UI and If I will update the items in flutter UI it also updated in Firebase



Here the data is fetched and after adding the total price it reflects on the payment page. So that the user need not to add calculate the total payment.



Order related information is also stored in the order database.



Conclusion - I learnt about the firebase and successfully implemented it in my flutter project.