Ardent Computech Pvt. Ltd.
Drives you to the Industry
Module 132, SDF Building, Sector V, Salt Lake, Pin - 700091
www.ardentcollaborations.com

Hybrid Mobile Application Development using Flutter

A Project Report for Industrial Training and Internship in

ARDENT GROUP OF COMPANIES
INTERNSHIP | DEVELOPMENT | PROJECT
An ISO 9001 : 2015 Certified Company

submitted by

Humaira Sadia

In the partial fulfillment of the award of the degree of

Computer Science Engineering

in the

Bachelor of Technology (BTECH).

Of

Aliah University Newtown, West Bengal,

Plot No.- IIA/27, AA II, Newtown, Kolkata, Kadampukur, West Bengal 700156

\

CERTIFICATE FROM SUPERVISOR

This is to certify that **Humaira Sadia**" have completed the project titled "Hybrid Mobile Application Development using Flutter" under my supervision during the period from "06.07.2024" to "08.08.24" which is in partial fulfilment of requirements for the award of the **CSE** degree and submitted to the Department of "**Bachelor of Technology (BTECH)**" of **"Aliah University"**.

_____

Signature of the Supervisor

Date: 10/08/2024

Name of the Project Supervisor: Anindya Mukherjee

BONAFIDE CERTIFICATE

Certified that this project work was carried out under my supervision

"Hybrid Mobile Application Development using Flutter" is the bonafide work of

Name of the student: Humaira Sadia

SIGNATURE : Humaira Sadia.

PROJECT MENTOR : Anindya Mukherjee

SIGNATURE : _____

EXAMINERS

Ardent Original Seal

Acknowledgment

I would like to express my deepest gratitude to everyone who supported me throughout my internship project on Hybrid Mobile Application Development using Flutter at Ardent Computech Pvt. Ltd.

First and foremost, I would like to thank Mr. Anindya Mukherjee, my project supervisor, for his invaluable guidance, insightful feedback, and continuous encouragement. His expertise and support were instrumental in the successful completion of this project.

I am also profoundly grateful to my parents and family, whose unwavering support and understanding provided me with the strength and motivation to persevere throughout this journey. Their encouragement has been a constant source of inspiration.

This project has been a significant learning experience, and I am sincerely thankful to everyone who contributed to it.

Index

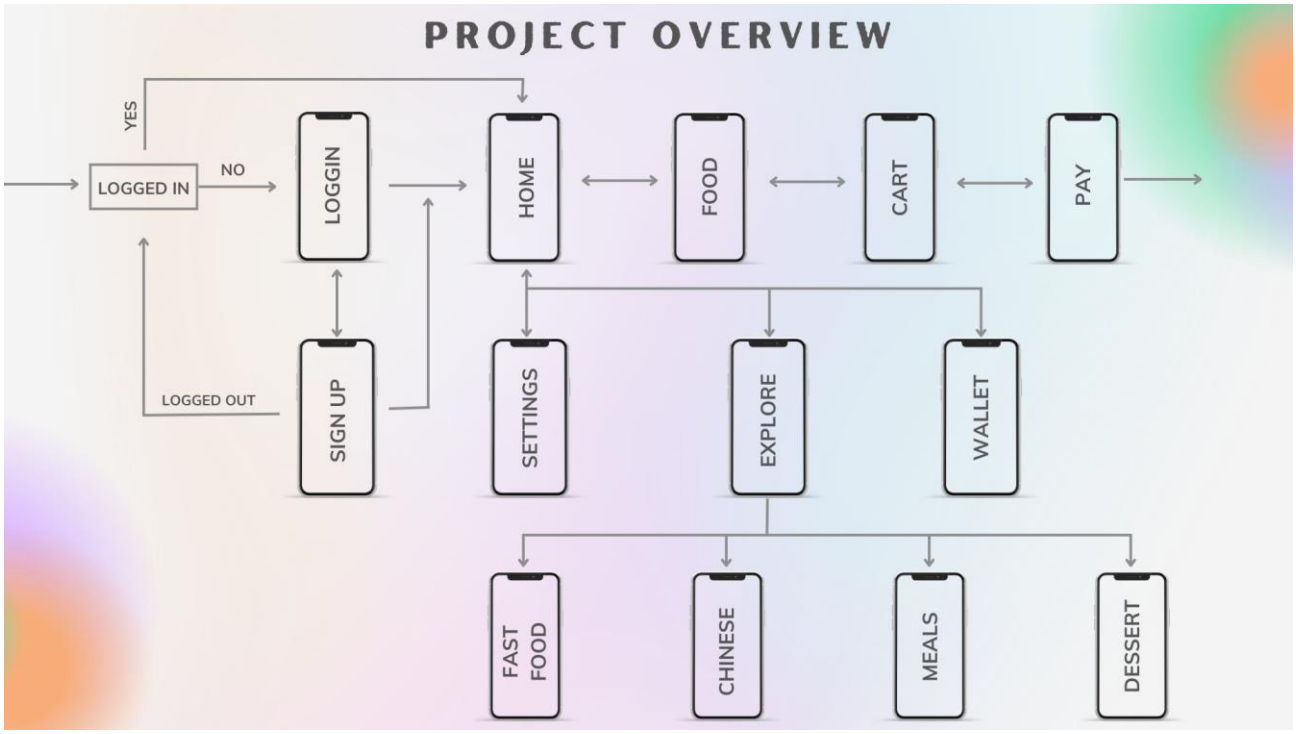
Name: **Humaira Sadia**, Roll no: **CSE214059**, Semester: **7th**, Year: **4th**,

College: **Aliah University**, Department: **CSE (BTECH)**

ONLINE FOOD DELIVERY APP USING FLUTTER

1.1 PROJECT OVERVIEW:

A food delivery app was developed using Flutter and Firebase, designed for a seamless user experience. The app allows users to browse menus, place orders, and track deliveries in real-time. Key features include a user-friendly interface, secure authentication, and integrated payment options. The project focused on creating a reliable and efficient platform for food ordering and delivery.



1.2 PROJECT DESCRIPTION: Food Delivery App Using Flutter and Firebase

As part of an internship at Ardent Computech Pvt. Ltd., a comprehensive food delivery application was developed using Flutter and Firebase. The primary goal of the project was to create a user-friendly and efficient platform that allows users to browse, order, and track their favorite meals from local restaurants, all within a seamless mobile experience.

- **User-Friendly Interface:** Design an intuitive and visually appealing user interface that allows users to easily navigate through the app, browse menus, and place orders with minimal effort.
- **Real-Time Data Handling:** Utilize Firebase for real-time database management, ensuring that users receive up-to-date information on restaurant offerings, order status, and delivery tracking.
- **Authentication and User Management:** Implement secure user authentication, enabling users to sign up, log in, and manage their profiles. This includes integrating social login options and handling user sessions effectively.
- **Order Management System:** Develop a robust order management system that allows users to place orders, customize their meals, and view order history. Restaurants can also manage incoming orders and update their status in real-time.
- **Payment Integration:** Integrate secure payment gateways to facilitate smooth and reliable transactions within the app, ensuring a secure and convenient checkout process for users.

Project Features:

- **Restaurant Listings:** Display a list of available restaurants with detailed information, including menus, ratings, and reviews, allowing users to make informed choices.
- **Search and Filter Options:** Provide users with the ability to search for specific dishes or restaurants and filter results based on cuisine, price range, and other preferences.
- **Cart and Checkout:** A dynamic cart system that allows users to add, remove, or modify items before proceeding to checkout. The checkout process includes order confirmation and payment options.
- **Order Tracking:** Enable users to track their orders in real-time, from preparation to delivery, ensuring transparency and reducing wait-time uncertainty.
- **Firebase Backend Integration:** Leverage Firebase's real-time database, authentication, and cloud functions to manage user data, orders, and notifications efficiently.

Outcomes: The project culminated in a fully functional food delivery app that effectively meets user needs for convenience, speed, and reliability. The use of Flutter provided a smooth and responsive user interface, while Firebase ensured real-time data synchronization and robust backend support.

Scope : Future enhancements for FeastFi could involve expanding the app's functionality with features like personalized recommendations, AI-driven order suggestions, and integration with multiple payment gateways. Additionally, the app can be scaled to accommodate more restaurants, incorporate advanced delivery tracking, and support multi-language options to cater to a broader audience.

SOFTWARE REQUIREMENT SPECIFICATIONS

Software Requirements Specifications provides an overview of the entire project. It is a description of a software system to be developed, laying out functional and non-functional requirements. The software requirements specifications document enlists enough and necessary requirements that are required for the project development. To derive the requirements we need to have clear and through understanding of the project to be developed. This is prepared after the detailed communication with the project team and the customer.

The developer is responsible for:-

- ✓ Developing the system, which meets the SRS and solving all the requirements of the system.
- ✓ Demonstrating the system and installing the system at client's location after acceptance testing is successful.
- ✓ Submitting the required user manual describing the system interfaces to work on it and also the documents of the system.
- ✓ Conducting any user training that might be needed for usingthe system.
- ✓ Maintain the system for a period of one year after installation.

HARDWARE REQUIREMENT

1. **Processor**:
   - Minimum: Intel Core i3 or equivalent
   - Recommended: Intel Core i5 or AMD Ryzen 5 or higher
2. **RAM**:
   - Minimum: 4 GB
   - Recommended: 8 GB or more
3. **Storage**:
   - Minimum: 10 GB free space (more may be required for larger projects)
   - Recommended: SSD with 100 GB or more of free space
4. **Graphics Card**:
   - Minimum: Integrated graphics are sufficient
   - Recommended: Dedicated graphics card for better performance, especially if using emulators
5. **Screen Resolution**:
   - Minimum: 1366x768 pixels
   - Recommended: 1920x1080 pixels or higher
6. **Internet Connection**:
   - Stable internet access for downloading dependencies, cloud services, and accessing Firebase.

SOFTWARE REQUIREMENT

1. **Operating System**:
   - **Windows**: Windows 10 or later
   - **macOS**: macOS Catalina or later (necessary for iOS development)
   - **Linux**: Ubuntu 20.04 or later or any other modern Linux distribution
2. **Development Tools**:
   - **Flutter SDK**: Install the Flutter SDK from the official Flutter website.
   - **Dart SDK**: Comes bundled with Flutter, but ensure it's up to date.
3. **IDE (Integrated Development Environment)**:
   - **Visual Studio Code**: Popular with Flutter developers; install the Flutter and Dart plugins.
   - **Android Studio**: Recommended for its rich Flutter plugin support; includes Android SDK for Android development.
4. **Firebase Tools**:
   - **Firebase CLI**: Install the Firebase CLI via npm (npm install -g firebase-tools) to manage Firebase services.
   - **Firebase Console**: Use the Firebase Console for project management and configuration.
5. **Emulators/Simulators**:
   - **Android Emulator**: Included with Android Studio; configure via AVD Manager.
6. **Version Control**:
   - **Git**: Install Git for version control and collaboration.
7. **Additional Dependencies**:
   - **Node.js**: Required for some Firebase tools and for managing Firebase functions.
   - **JDK (Java Development Kit)**: Required for Android development (usually bundled with Android Studio).

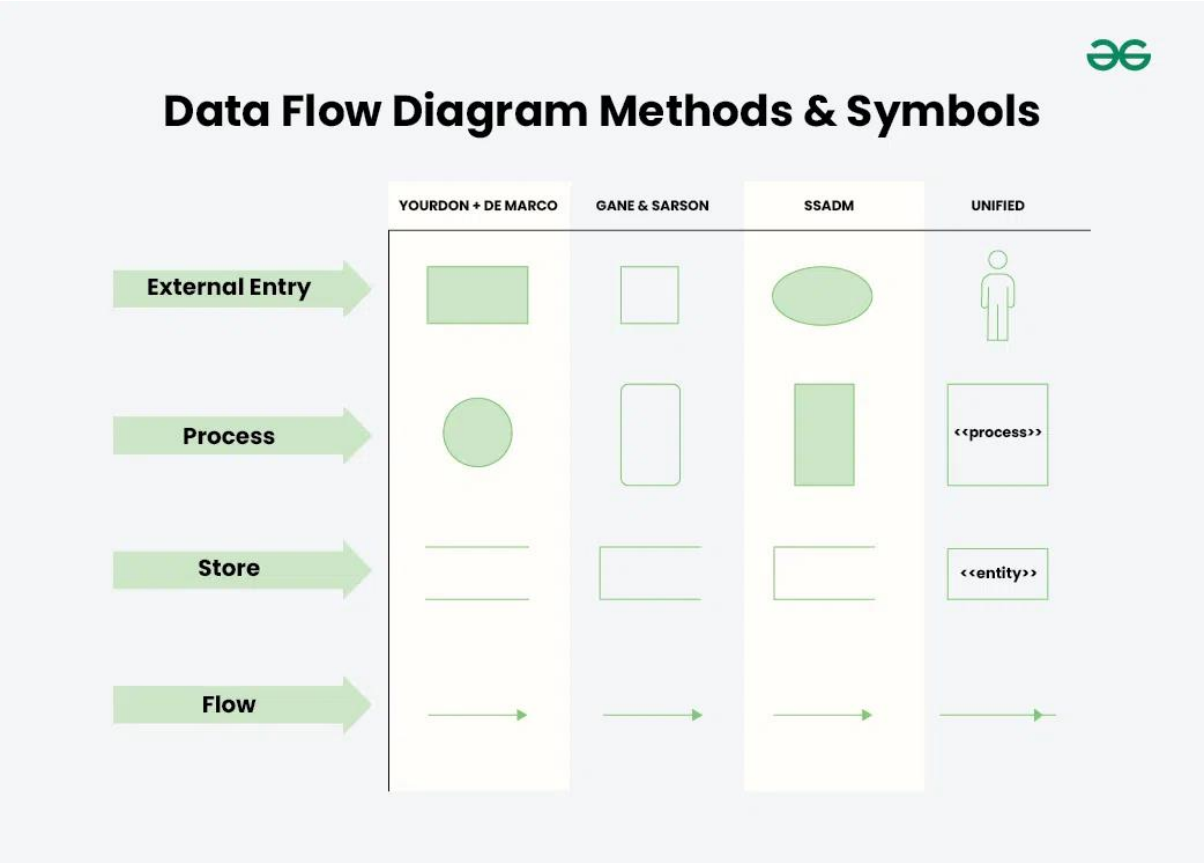
DATA-FLOW DIAGRAM

A **Data Flow Diagram (DFD)** is a graphical representation used to illustrate the flow of data within a system. It helps visualize how data moves through various processes and how it interacts with different components of the system. Here's a description of its key elements:

1. **Processes:** Represented by circles or rectangles with rounded corners, processes describe the operations or functions that transform input data into output data. They show what happens to the data within the system.
2. **Data Flows:** Shown as arrows, data flows indicate the movement of data between processes, data stores, and external entities. They illustrate the direction and flow of information.
3. **Data Stores:** Depicted as open-ended rectangles or parallel lines, data stores represent repositories where data is held for later use. These can be databases, files, or other storage systems.
4. **External Entities:** Represented by rectangles, external entities are sources or destinations of data that interact with the system but are outside its boundaries. They could be users, other systems, or external databases.

Description of a Data Flow Diagram:

A Data Flow Diagram provides a clear and organized way to understand how data is processed and managed within a system. It visualizes how different components of the system interact, how data is transformed and stored. DFDs are used in various stages of system analysis and design to ensure that data handling is efficient, accurate, and aligned with system requirements. They are valuable tools for developers, analysts, and stakeholders to communicate and validate the system's data management processes.



Steps to Construct Data Flow Diagram:-

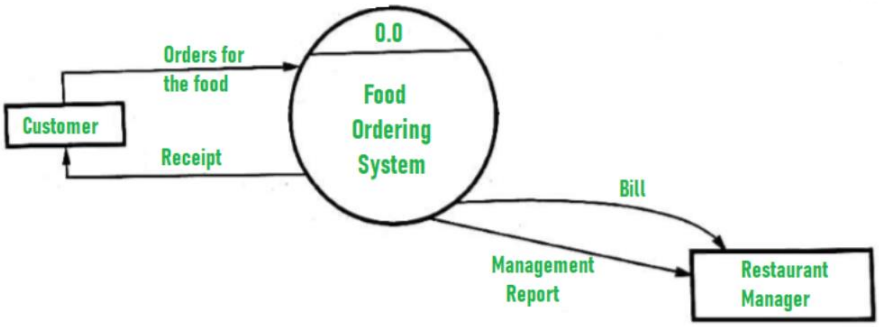
Four Steps are generally used to construct a DFD.

- Process should be named and referred for easy reference. Eachname should be representative of the reference.
- The destination of flow is from top to bottom and from left to right.
- When a process is distributed into lower level details they are numbered.
- The names of data stores, sources and destinations are writtenin capital letters.

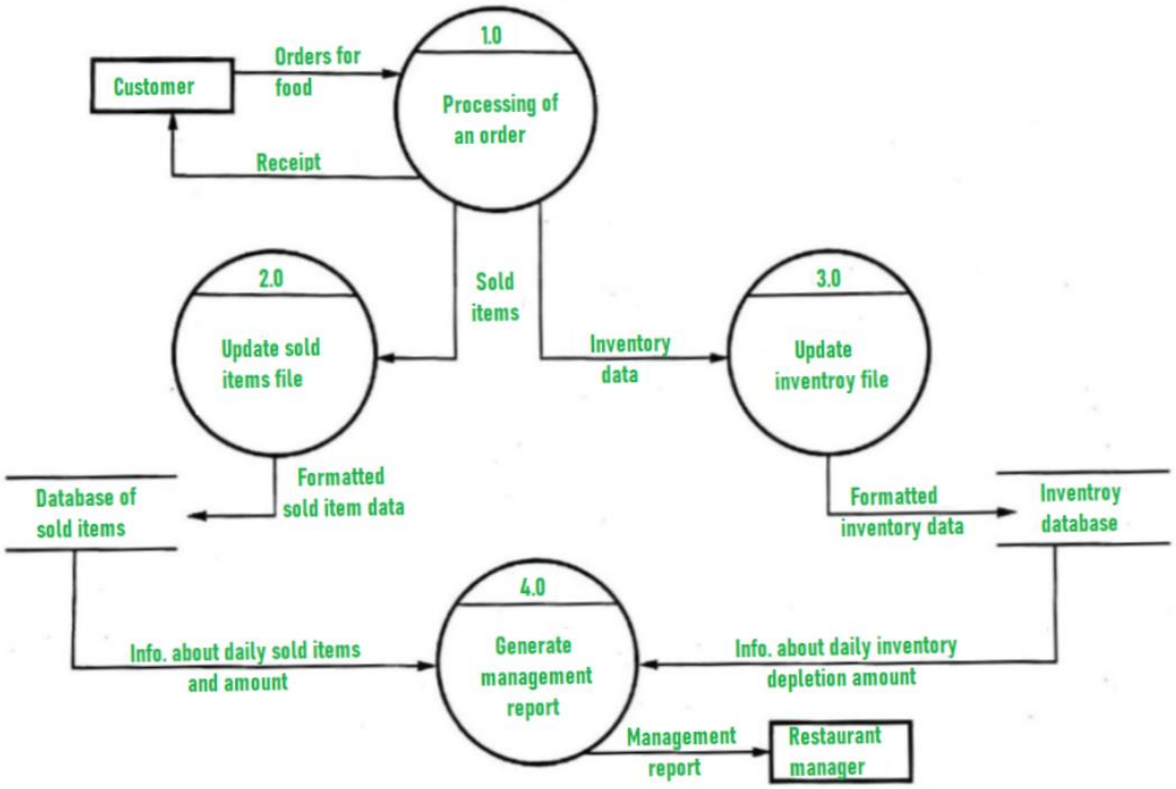
Rules for constructing a Data Flow Diagram:-

- Arrows should not cross each other.
- Squares, Circles, Files must bear a name.
- Decomposed data flow squares and circles can have samenames.
- Draw all data flow around the outside of the diagram.

Data-Flow Diagram Level-0



Data-Flow Diagram Level-1



Coding Standard Followed & Screenshots

main.dart

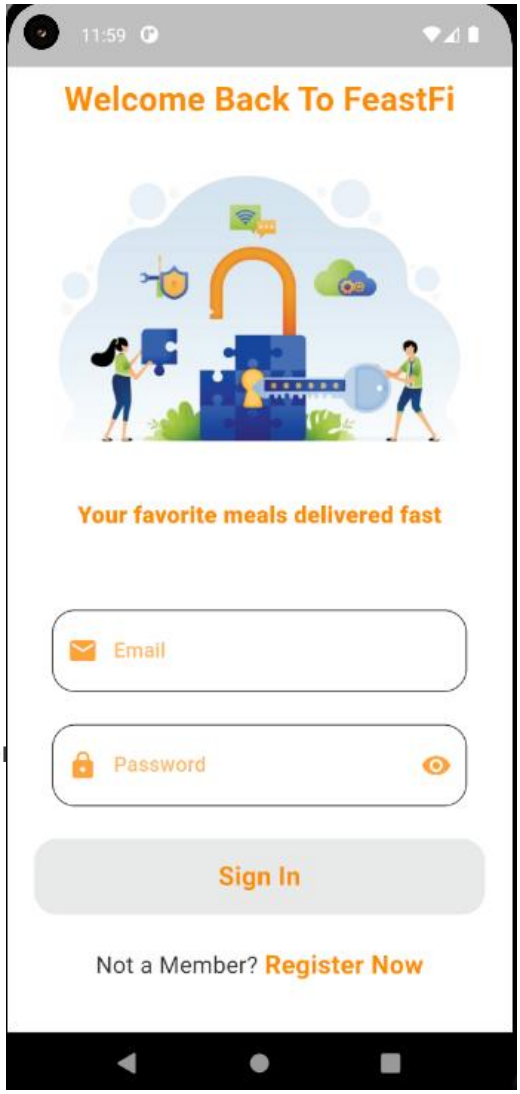
```
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
import 'package:food_odering/components/cartModel.dart';
import 'package:food_odering/firebase_options.dart';
import 'package:food_odering/models/userData.dart';
import 'package:food_odering/pages/adminPannel.dart';
import 'package:food_odering/pages/cart.dart';
import 'package:food_odering/pages/home.dart';
import 'package:food_odering/pages/login.dart';
import 'package:food_odering/pages/register.dart';
import 'package:food_odering/themes/theme_provider.dart';
import 'package:provider/provider.dart';

Future<void> main() async {
 WidgetsFlutterBinding.ensureInitialized();

 await Firebase.initializeApp(
  options: DefaultFirebaseOptions.currentPlatform,
 );

 runApp(
  MultiProvider(
   providers: [
    ChangeNotifierProvider(create: (context) => ThemeProvider()),
    ChangeNotifierProvider(create: (context) => CartModel()),
```

```
      ChangeNotifierProvider(create: (context) => UserData()),
      ChangeNotifierProvider(create: (context) => UserData()),
    ],
    child: const MyApp(),
   ),
  );
}

class MyApp extends StatelessWidget {
 const MyApp({super.key});

 @override
 Widget build(BuildContext context) {
  return MaterialApp(
   debugShowCheckedModeBanner: false,
   theme: Provider.of<ThemeProvider>(context).themeData,
   home: LoginPage(
    onTap: () {},
   ),
   routes: {
    'register': (context) => Register(onTap: () {}),
    'login': (context) => LoginPage(onTap: () {}),
    'home': (context) => HomePage(),
    'adminPanel': (context) => Adminpannel(),
    'cart': (context) => MyCart(),
   },
  );
 }
}
```

Login.dart

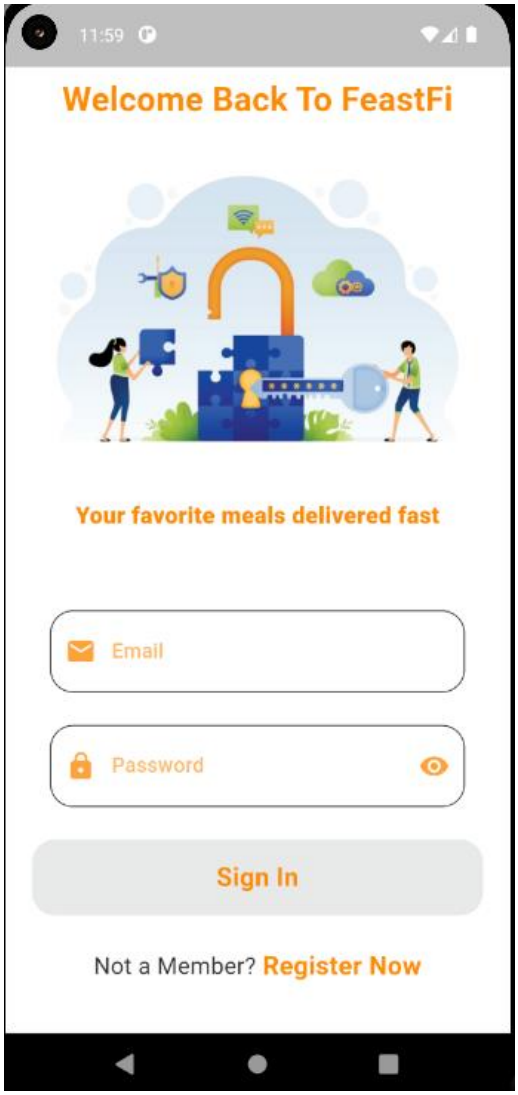
```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:food_odering/components/my_button.dart';
import 'package:food_odering/components/my_textField.dart';
import 'package:food_odering/models/userUtil.dart';
import 'package:provider/provider.dart';
import 'package:food_odering/models/userData.dart'; // Import the UserData provider

class LoginPage extends StatefulWidget {
 final void Function()? onTap;

 const LoginPage({super.key, required this.onTap});

 @override
 State<LoginPage> createState() => _LoginPageState();
}

class _LoginPageState extends State<LoginPage> {
 final TextEditingController emailController = TextEditingController();
 final TextEditingController passwordController = TextEditingController();

 Future<void> login() async {
  try {
   // Authenticate user
   if (emailController.text == 'admin22@gmail.com' &&
      passwordController.text == 'admin@22') {
    Navigator.pushNamed(context, 'adminPanel');
   } else {
```

```
  UserCredential userCredential =
      await FirebaseAuth.instance.signInWithEmailAndPassword(
    email: emailController.text,
    password: passwordController.text,
  );

  // Fetch and update user data
  await fetchAndUpdateUserData(context);

  // Navigate to home page on successful login
  Navigator.pushReplacementNamed(context, 'home');
 }
} on FirebaseAuthException catch (e) {
 String errorMessage;

 switch (e.code) {
  case 'user-not-found':
   errorMessage = 'No user found for that email.';
   break;
  case 'wrong-password':
   errorMessage = 'Wrong password provided for that user
   break;
  default:
   errorMessage = 'An undefined Error happened.';
 }

 ScaffoldMessenger.of(context).showSnackBar(
  SnackBar(
   content: Text(errorMessage),
  ),
 );
} catch (e) {
 print("Failed to log in: $e");
```



```
   ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(
     content: Text("Failed to log in: $e"),
    ),
   );
  }
}

void register() {
 // Navigate to register screen
 Navigator.pushNamed(context, 'register');
}

@override
Widget build(BuildContext context) {
 return Scaffold(
  backgroundColor: Theme.of(context).colorScheme.background,
  body: Column(
   children: [
    Container(
     padding:
       const EdgeInsets.symmetric(vertical: 40.0, horizontal: 20.0),
     child: Column(
      children: [
       SizedBox(height: 20),
       Text(
        "Welcome Back To FeastFi",
        style: TextStyle(
         fontSize: 26,
         fontWeight: FontWeight.bold,
         color: Theme.of(context).colorScheme.primary,
        ),
       ),
```

```
    const SizedBox(height: 20),
    // Image
    Image.asset('assets/login_img.png'),
    const SizedBox(height: 20),
    // App slogan
    Text(
     "Your favorite meals delivered fast",
     style: TextStyle(
      fontSize: 18,
      fontWeight: FontWeight.w900,
      color: Theme.of(context).colorScheme.primary,
     ),
    ),
   ],
  ),
),
// Scrollable content
Expanded(
 child: SingleChildScrollView(
  padding: const EdgeInsets.all(20.0),
  child: Column(
   children: [
    // Email textfield
    MyTextField(
     controller: emailController,
     hintText: "Email",
     obscuredText: false,
    ),
    const SizedBox(height: 25),

    // Password textfield
    MyTextField(
     controller: passwordController,
```



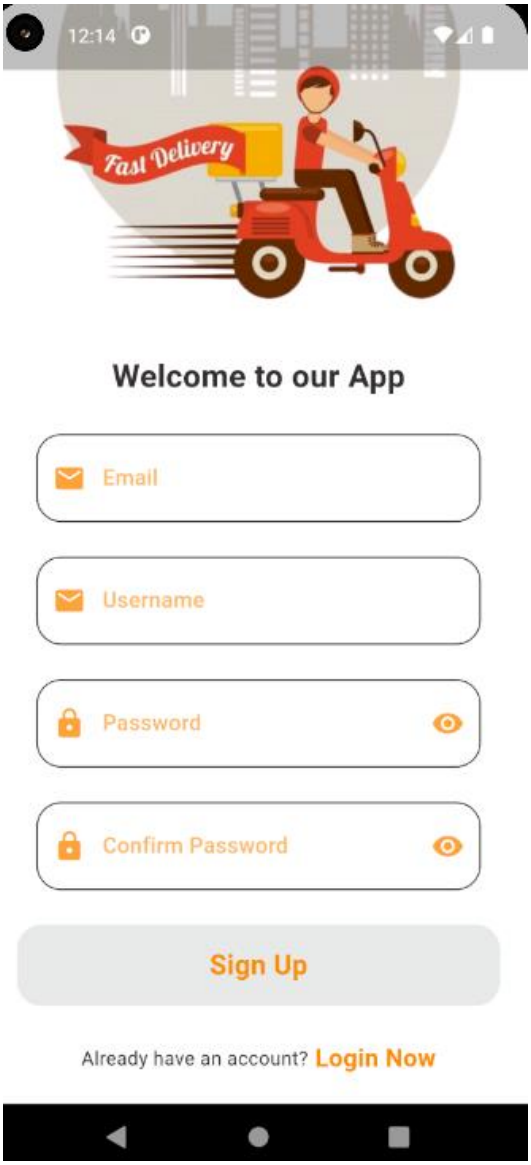
```
 hintText: "Password",
 obscuredText: true,
),
const SizedBox(height: 25),

// Sign in button
MyButton(
 text: "Sign In",
 onTap: login,
),
const SizedBox(height: 25),

// Not a member? Register now
Row(
 mainAxisAlignment: MainAxisAlignment.center,
 children: [
  Text(
   "Not a Member?",
   style: TextStyle(
    fontSize: 18,
    color: Theme.of(context).colorScheme.inversePrimary,
   ),
  ),
  const SizedBox(width: 5),
  GestureDetector(
   onTap: register,
   child: Text(
    "Register Now",
    style: TextStyle(
     fontSize: 20,
     color: Theme.of(context).colorScheme.primary,
     fontWeight: FontWeight.bold,
    ),..}
```

Register.dart

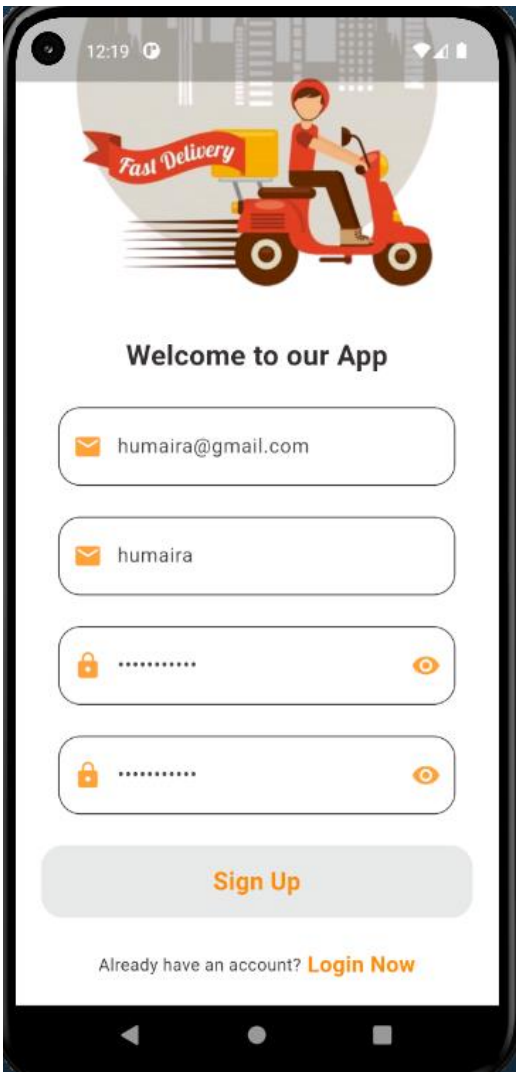
```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:food_odering/components/my_button.dart';
import 'package:food_odering/components/my_textField.dart';
import 'package:food_odering/models/userUtil.dart';

class Register extends StatefulWidget {
 final TextEditingController emailController = TextEditingController();
 final TextEditingController userController = TextEditingController();
 final TextEditingController passwordController = TextEditingController();
 final TextEditingController confpasswordController = TextEditingController();

 final void Function()? onTap;
 Register({super.key, required this.onTap});

 @override
 State<Register> createState() => _RegisterState();
}

class _RegisterState extends State<Register> {
 @override
 Widget build(BuildContext context) {
  return Scaffold(
   backgroundColor: Theme.of(context).colorScheme.background,
   body: Center(
    child: SingleChildScrollView(
     padding: const EdgeInsets.all(20.0),
```

```
child: Column(
 mainAxisAlignment: MainAxisAlignment.center,
 children: [
  // Logo
  Image.asset('assets/register.png'),

  // App slogan
  Text(
   "Welcome to our App",
   style: TextStyle(
    fontSize: 23,
    fontWeight: FontWeight.w700,
    color: Theme.of(context).colorScheme.inversePrimary,
   ),
  ),
  const SizedBox(height: 25),

  // Email textfield
  MyTextField(
   controller: widget.emailController,
   hintText: "Email",
   obscuredText: false,
  ),
  const SizedBox(height: 25),

  // Username textfield
  MyTextField(
   controller: widget.userController,
   hintText: "Username",
   obscuredText: false,
  ),
  const SizedBox(height: 25),
```

```
// Password textfield
MyTextField(
 controller: widget.passwordController,
 hintText: "Password",
 obscuredText: true,
),
const SizedBox(height: 25),

// Confirm Password textfield
MyTextField(
 controller: widget.confpasswordController,
 hintText: "Confirm Password",
 obscuredText: true,
),
const SizedBox(height: 25),

// Sign up button
MyButton(
 text: "Sign Up",
 onTap: () async {
  // Get text values from controllers
  String email = widget.emailController.text;
  String password = widget.passwordController.text;
  String confirmPassword = widget.confpasswordController.text;
  String username = widget.userController.text;

  if (password != confirmPassword) {
   ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(
     content: Text("Passwords do not match."),
    ),
   );
   return;
```



```
}

try {
 // Create user with email and password
 UserCredential userCredential = await FirebaseAuth.instance
    .createUserWithEmailAndPassword(
  email: email,
  password: password,
 );

 // Get the user ID
 String userId = userCredential.user!.uid;

 // Store additional user information in Firestore
 await FirebaseFirestore.instance
    .collection('users')
    .doc(userId)
    .set({
  'name': username,
  'email': email,
  'createdAt': FieldValue.serverTimestamp(),
 });

 // Fetch and update user data in the provider
 await fetchAndUpdateUserData(context);

 print("Signed up Successfully $userId");
 Navigator.pushNamed(context,
    'home'); // Navigate to home page after successful signup
} catch (e) {
 print("Failed to sign up: $e");
 ScaffoldMessenger.of(context).showSnackBar(
  SnackBar(
```



```
        content: Text("Failed to sign up: $e"),
      ),
     );
    }
  },
),
const SizedBox(height: 25),

// Already have an account? Login now
Row(
 mainAxisAlignment: MainAxisAlignment.center,
 children: [
  Text(
   "Already have an account?",
   style: TextStyle(
    color: Theme.of(context).colorScheme.inversePrimary,
   ),
  ),
  const SizedBox(width: 5),
  GestureDetector(
   onTap: () {
    Navigator.pushNamed(context, 'login');
   },
   child: Text(
    "Login Now",
    style: TextStyle(
     color: Theme.of(context).colorScheme.primary,
     fontWeight: FontWeight.bold,
     fontSize: 18,
    ),
   ),
  ),
 ],
```

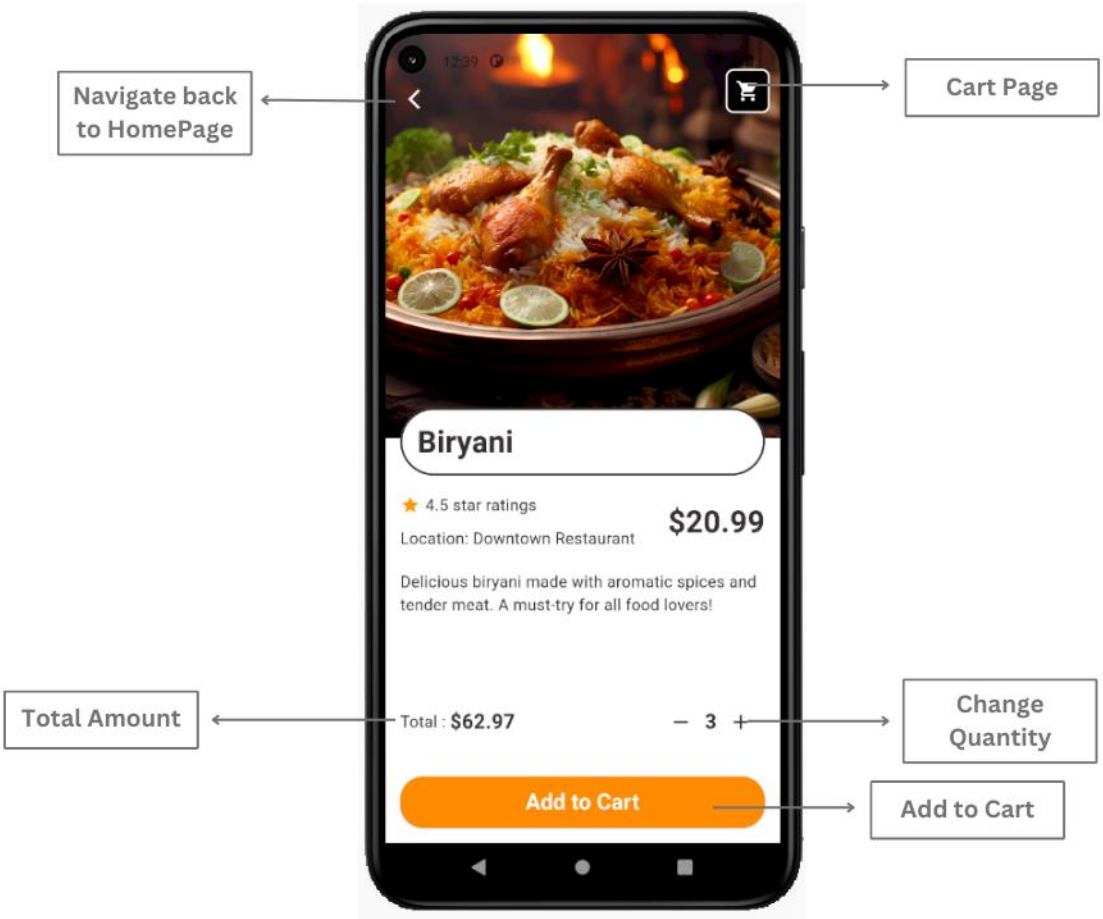
```
class _HomePageState extends State<HomePage> {
 @override
 Widget build(BuildContext context) {
  // Access UserData from the Provider
  final userData = Provider.of<UserData>(context);

  return Scaffold(
   body: CustomScrollView(
    slivers: [
     SliverAppBar(
      elevation: 2,
      expandedHeight: 160.0,
      stretch: true,
      automaticallyImplyLeading: false,
      flexibleSpace: MyLocation(
       username: userData.username, // Fetch username from UserData
       location: userData.location, // Fetch location from UserData
      ),
     ),
     SliverToBoxAdapter(
      child: Container(
       height: 200,
       child: MySlidder(),
      ),
     ),
     SliverList(
      delegate: SliverChildBuilderDelegate(
       (BuildContext context, int index) {
        return Container(
```

```
            color: Theme.of(context).colorScheme.secondary,
            child: MyVerticalFood(),
          );
        },
        childCount: 5,
      ),
    ),
  ],
 ),
 bottomNavigationBar: const BottomNavigation(),
);
}
```

ItemList.dart

```
Container(
    padding: EdgeInsets.all(15),
    child: Material(
     elevation: 5.0,
     borderRadius: BorderRadius.circular(15),
     child: GestureDetector(
      onTap: () {
       Navigator.push(
          context, MaterialPageRoute(builder: (context) => MyFoodPage()));
      },
      child: Container(
       // color: Colors.red,
       padding: EdgeInsets.all(5),
       child: Row(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
         // Image on the left
         Image.asset(
          "assets/food/biryani.png",
          height: 130,
         ),
         SizedBox(width: 16),
         Expanded(
          child: Column(
           crossAxisAlignment: CrossAxisAlignment.start,
           children: [
            Text(
             'Arsalan',
             style: TextStyle(
```

```
    fontWeight: FontWeight.bold,
    fontSize: 25,
    color: Theme.of(context).colorScheme.inversePrimary,
   ),
  ),
  SizedBox(height: 8),
  Text(
   '• Biryani',
   style: TextStyle(
    fontSize: 17,
    fontWeight: FontWeight.w600,
    color: Theme.of(context).colorScheme.inversePrimary,
   ),
  ),
  SizedBox(height: 4),
  Text(
   '• North Indian',
   style: TextStyle(
    fontSize: 17,
    fontWeight: FontWeight.w600,
    color: Theme.of(context).colorScheme.inversePrimary,
   ),
  ),
  SizedBox(height: 4),
  Text(
   '• Rs3.6 for one',
   style: TextStyle(
    fontSize: 17,
    fontWeight: FontWeight.w600,
    color: Theme.of(context).colorScheme.inversePrimary,
```

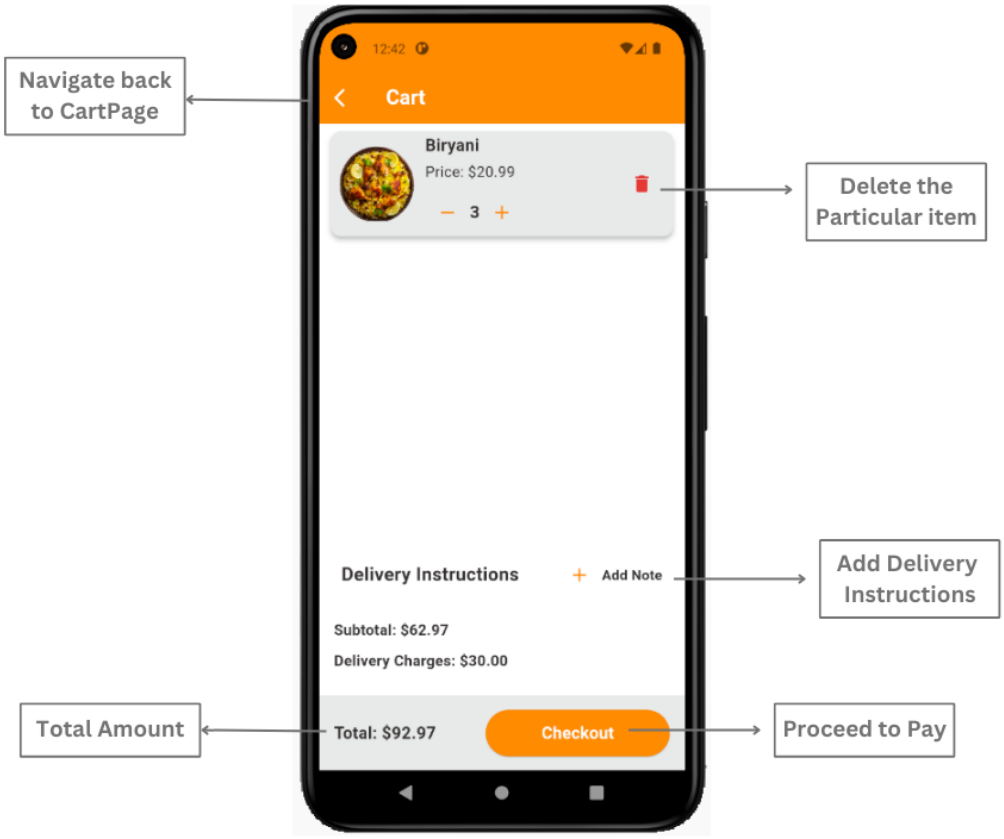


cartModel.dart

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

class CartItem {
 final String name;
 final double price;
 final int quantity;
 final String image;

 CartItem({
  required this.name,
  required this.price,
  required this.quantity,
  required this.image,
 });
```

```
}

class CartModel extends ChangeNotifier {
 final List<CartItem> _items = [];

 List<CartItem> get items => _items;

 double get totalPrice =>
    _items.fold(0, (sum, item) => sum + (item.price * item.quantity));

 void addItem(String name, double price, int quantity, String image) {
  final existingItemIndex = _items.indexWhere((item) => item.name == name);
  if (existingItemIndex >= 0) {
    _items[existingItemIndex] = CartItem(
      name: name,
      price: price,
      quantity: _items[existingItemIndex].quantity + quantity,
      image: image,
    );
  } else {
    _items.add(
       CartItem(name: name, price: price, quantity: quantity, image: image));
  }
  notifyListeners();
 }

 void removeItem(String name) {
  _items.removeWhere((item) => item.name == name);
  notifyListeners();
 }

 void clearCart() {
  _items.clear();
```

```
 notifyListeners();
}

void updateQuantity(String name, int quantity) {
 final existingItemIndex = _items.indexWhere((item) => item.name == name);
 if (existingItemIndex >= 0 && quantity > 0) {
  _items[existingItemIndex] = CartItem(
   name: name,
   price: _items[existingItemIndex].price,
   quantity: quantity,
   image: _items[existingItemIndex].image,
  );
  notifyListeners();
 } else if (quantity <= 0) {
  removeItem(name);
 }
```



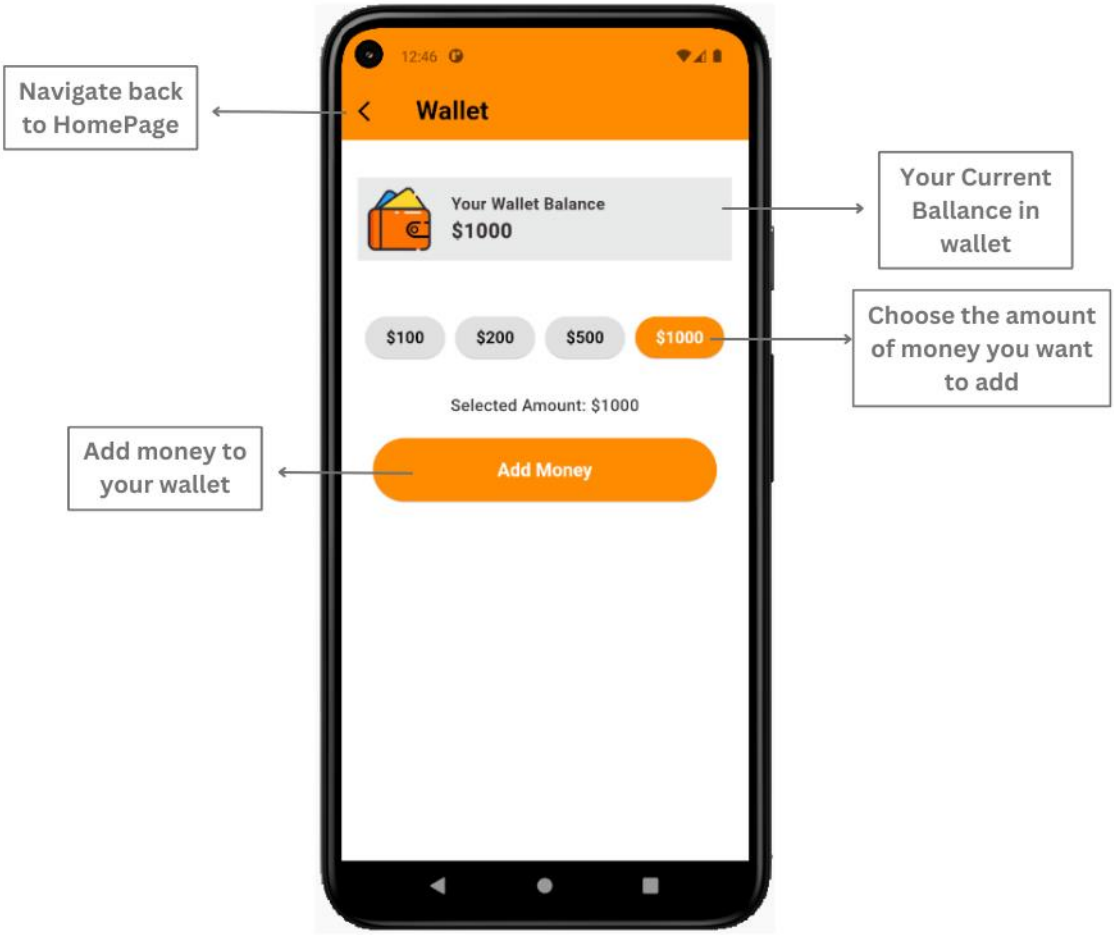
wallet.dart

```
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      automaticallyImplyLeading: false,
      leading: IconButton(
        icon: Icon(Icons.arrow_back_ios),
        onPressed: () {
          Navigator.pushReplacement(
            context,
            MaterialPageRoute(builder: (context) => const HomePage()),
          );
        },
      ),
      title: Text(
        'Wallet',
        style: TextStyle(fontWeight: FontWeight.w700, fontSize: 25),
      ),
      backgroundColor: Theme.of(context).colorScheme.primary,
    ),
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.center,
        children: [
          // Wallet balance header with top margin
          Container(
            margin: EdgeInsets.only(top: 20),
            color: Theme.of(context).colorScheme.secondary,
            child: Row(
```

```
    mainAxisAlignment: MainAxisAlignment.start,
    children: [
     Image.asset(
      "assets/icons/wallet.png",
      height: 80,
     ),
     SizedBox(width: 10),
     Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
       Text(
        "Your Wallet Balance",
        style: TextStyle(
         fontSize: 16,
         fontWeight: FontWeight.w600,
         color: Theme.of(context).colorScheme.inversePrimary,
        ),
       ),
       Text(
        "\$${_selectedAmount}",
        style: TextStyle(
          color: Theme.of(context).colorScheme.inversePrimary,
          fontWeight: FontWeight.w800,
          fontSize: 20),
       )
      ],
     )
    ],
   ),
  ),
  SizedBox(height: 50),

  // Money options
```

```
Row(
 mainAxisAlignment: MainAxisAlignment.center,
 children: [
  _buildAmountButton(100),
  SizedBox(width: 10),
  _buildAmountButton(200),
  SizedBox(width: 10),
  _buildAmountButton(500),
  SizedBox(width: 10),
  _buildAmountButton(1000),
 ],
),
SizedBox(height: 30),
// Display selected amount
Text(
 'Selected Amount: \$$_selectedAmount',
 style: TextStyle(
  color: Theme.of(context).colorScheme.inversePrimary,
  fontSize: 16,
  fontWeight: FontWeight.w500,
 ),
),
SizedBox(height: 20),

// Add Money Button
ElevatedButton(
 onPressed: _addMoney,
 child: Text('Add Money'),
 style: ElevatedButton.styleFrom(
  foregroundColor: Theme.of(context).colorScheme.background,
  backgroundColor: Theme.of(context).colorScheme.primary,
  padding: EdgeInsets.symmetric(horizontal: 120, vertical: 20),
  textStyle: TextStyle(
```

```
            fontSize: 18,
            fontWeight: FontWeight.bold,
          ),
        ),
      ),
    ],
  ),
 ),
);
}
```

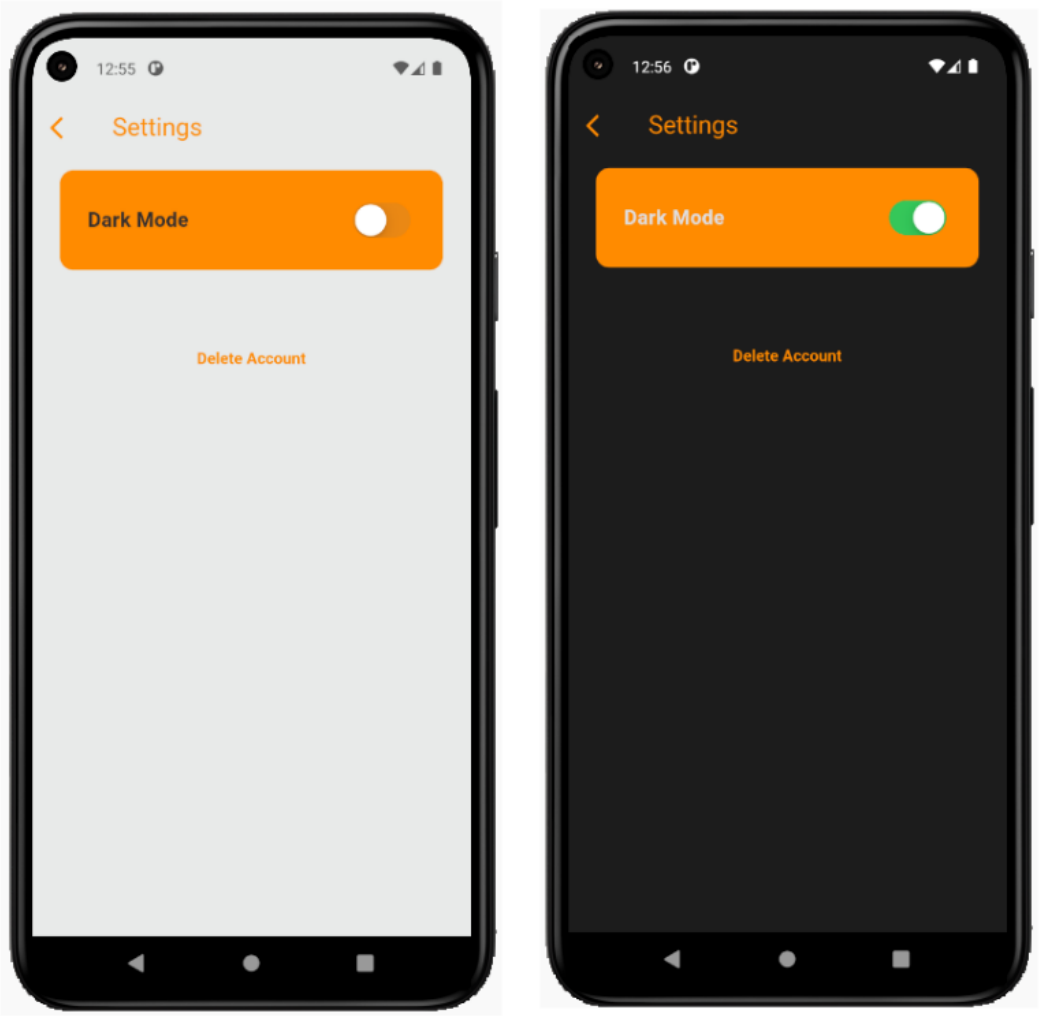


Setting.dart

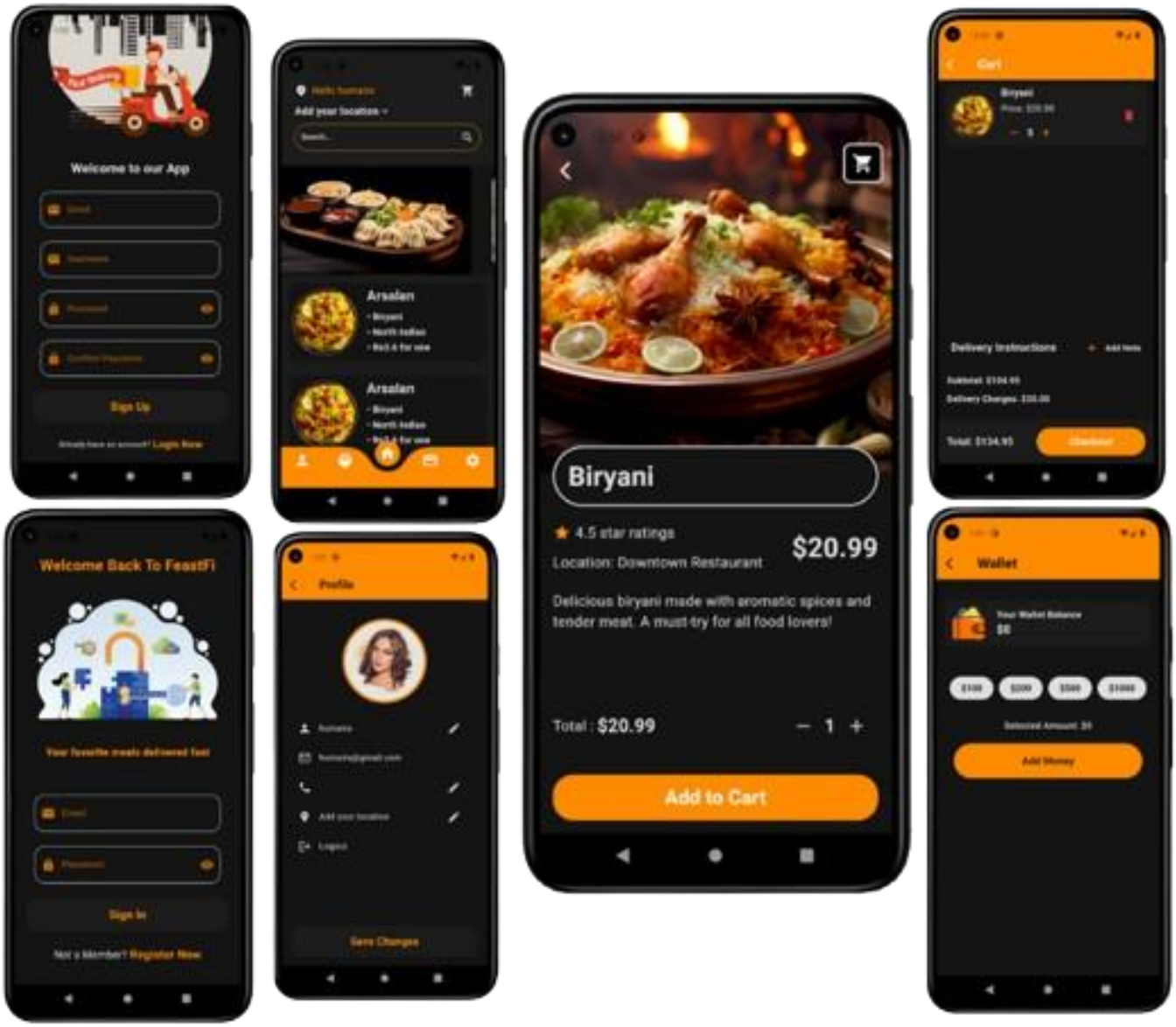
```
Widget build(BuildContext context) {
  return Scaffold(
   backgroundColor: Theme.of(context).colorScheme.secondary,
   appBar: AppBar(
    automaticallyImplyLeading: false,
    leading: IconButton(
     icon: Icon(
      Icons.arrow_back_ios,
      color: Theme.of(context).colorScheme.primary,
     ),
     onPressed: () {
      Navigator.pushReplacement(
       context,
       MaterialPageRoute(builder: (context) => const HomePage()),
      );
     },
    ),
    backgroundColor: Theme.of(context).colorScheme.secondary,
    title: Text(
     "Settings",
     style: TextStyle(color: Theme.of(context).colorScheme.primary),
    ),
   ),
   body: Column(
    children: [
     Container(
      decoration: BoxDecoration(
       color: Theme.of(context).colorScheme.primary,
       borderRadius: BorderRadius.circular(12),
      ),
```

```
  margin: EdgeInsets.only(left: 25, top: 10, right: 25),
  padding: EdgeInsets.all(25),
  child: Row(
   mainAxisAlignment: MainAxisAlignment.spaceBetween,
   children: [
    Text(
     "Dark Mode",
     style: TextStyle(
      fontSize: 18,
      fontWeight: FontWeight.bold,
      color: Theme.of(context).colorScheme.inversePrimary,
     ),
    ),
    CupertinoSwitch(
     value: Provider.of<ThemeProvider>(context, listen: false)
        .isDarkMode,
     onChanged: (value) =>
        Provider.of<ThemeProvider>(context, listen: false)
          .toggleTheme(),
    ),
   ],
  ),
 ),
 SizedBox(height: 20),
 Container(
  decoration: BoxDecoration(
   // color: Colors.redAccent,
   borderRadius: BorderRadius.circular(12),
  ),
  margin: EdgeInsets.only(left: 25, top: 10, right: 25),
  padding: EdgeInsets.all(25),
  child: Row(
   mainAxisAlignment: MainAxisAlignment.center,
```

```
      children: [
       TextButton(
        onPressed: () => _deleteAccount(context),
        child: Text(
         "Delete Account",
         style: TextStyle(
          fontWeight: FontWeight.bold,
         ),
        ),
       ),
      ],
     ),
    ),
  ],
```



APPLICATION IN DARK MODE



FIREBASE COMMANDS AND DEPENDENCIES

FlutterFire :- https://firebase.flutter.dev/docs/overview

Commands –

- **flutter pub add firebase_core**

  # Install the CLI if not already done so
- **dart pub global activate flutterfire_cli**

  # Run the `configure` command, select a Firebase project and platforms
- **flutterfire configure**

  # lib/main.dart
- **import 'package:firebase_core/firebase_core.dart';**
- **import 'firebase_options.dart';**

- **void main() async {**
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
  options: DefaultFirebaseOptions.currentPlatform,
  );
  runApp(MyApp());
  }

Dependencies –

_firebase_core: ^3.3.0
firebase_auth: ^5.1.4
firebase_database: ^11.0.4
cloud_firestore: ^5.2.1

APPLICATION IN DARK MODE

Introduction

Testing ensures the FeastFi app functions correctly by identifying and fixing errors. It improves the app's integrity and confirms it meets user requirements. Thorough testing is crucial as incomplete testing can lead to high costs and potential system failures.

Objectives

- Find and Report Bugs: Identify as many issues as possible to enhance app stability.
- Ensure Usability: Validate that the app performs well across various platforms.
- Confirm User Interface: Test the user interface for ease of use and functionality.

Process Overview

1. Identify Requirements: Use the app specifications to determine what needs testing.
2. Select Test Cases: Determine which tests apply to each feature.
3. Review: Ensure test data and cases are thorough and adequate.
4. Define Expected Results: Document what outcomes are expected from each test.
5. Document Configuration: Record the test cases, data, and configurations.
6. Perform Tests: Execute the tests according to the plan.
7. Record Results: Document the results and any issues found.
8. Report Issues: Use a Bug Report Form for any problems encountered.

9. **Submit Documentation: Ensure all test documents and reports are updated and reviewed.**

Testing Steps

- **Unit Testing: Test individual components (e.g., login, search) to ensure they function as expected during development.**
- **Integration Testing: Check interactions between components (e.g., ensuring the search results correctly update the home page).**
- **Validation Testing: Verify the app functions as expected by the user and meets all specifications.**

Test Cases Each test case should include:

- **Identifier: Unique ID for the test case.**
- **Test Condition: Specific scenario being tested.**
- **Input Data: Data used for testing.**
- **Expected Result: Anticipated outcome of the test.**

Testing Plan for FeastFi App

White Box Testing

- Overview: Focuses on the internal code and structure. Inputs and outputs are tested directly at the code level, bypassing the UI.
- Approach: Apply Branch Testing to cover all conditions and ensure each is executed at least once. Each function of the app will be tested independently by deriving test cases from the code.

Black Box Testing

- Overview: Tests the app's functionality from an end-user perspective without examining the internal code.
- Approach: Use Equivalence Partitioning and Boundary Value Analysis to verify that the app handles various inputs correctly and produces the expected outputs.

System Testing

- Overview: Focuses on detecting faults that arise only in the integrated system. Key areas include performance and function validation.
- Approach: Perform black-box testing to assess the overall functionality and performance of the app.

Output Testing

- Overview: Ensures the app produces the required outputs in the specified format.
- Approach: Verify that the output formats on the screen and in hard copies meet user requirements and are correctly implemented.

User Acceptance Testing

- Overview: Validates the system from the user's perspective to ensure it meets their needs and expectations.
- Focus Areas:
  - Input Screen Design
  - Output Screen Design
  - Report and Output Formats
- Approach: Engage with users throughout the development process to gather feedback and make necessary adjustments.

Goal of Testing

- Objective: Ensure the system meets expectations and is free from critical bugs. If discrepancies are found, they should be fixed to ensure the system operates correctly.

CONCLUSION

The FeastFi Flutter app project has achieved its objectives through thorough development and testing phases. The combination of white box and black box testing has ensured that the app is both technically sound and user-friendly.

Key Outcomes:

- **Code Integrity:** White box testing validated comprehensive code coverage, resulting in a stable application.
- **Functionality:** Black box testing confirmed the app's features work as intended, providing a seamless user experience.
- **Performance:** System testing demonstrated effective performance and integration with external components.
- **Output Accuracy:** Output testing verified correct data formatting for on-screen and printed results.
- **User Acceptance:** Continuous user feedback ensured alignment with design and functionality expectations.

In summary, the FeastFi app is a well-executed food delivery solution, reflecting a strong commitment to quality and user satisfaction. The insights gained will support ongoing improvements and future projects.

BIBLOGRAPHY

- https://firebase.flutter.dev/docs/overview
- https://www.w3schools.com
- https://www.slideshare.com
- https://www.scribd.com
- https://www.tutorialspoint.com
- https://www.geeksforgeeks.org.com
- https://www.youtube.com
- https://www.javatpoint.com