1. MAIN REPORT

1.1. Objective & Scope of the Project

The primary objective of the **FoodieBookin** project is to create a seamless and user-friendly shop-to-table experience for food enthusiasts. This platform aims to provide a comprehensive solution for users to discover, select, and enjoy a variety of dishes from the comfort of their homes. The project encompasses a robust authentication system to ensure secure access and personalized experiences for users. The home section presents a curated list of enticing dishes, facilitating easy exploration and selection. Users can effortlessly add their chosen items to the cart, streamlining the ordering process.

The scope extends to a well-integrated cart system, allowing users to review and modify their selections before proceeding to payment. The payment module ensures a secure and efficient transaction process, enhancing user trust. By focusing on these key features, FoodieBookin aspires to redefine the online food ordering experience. This project addresses the evolving needs of modern consumers who seek convenience without compromising on the quality of their culinary experiences. Through meticulous attention to authentication, menu presentation, cart management, and secure payments, FoodieBookin aims to emerge as a leading platform, offering a delightful fusion of technology and gastronomy.

1.2. Theoretical Background Definition of Problem

The concept of **FoodieBookin** envisions a seamless Shop-to-Table experience by integrating key functionalities such as authentication, a home list of dishes, adding items to the cart, and completing the payment process. Authentication is fundamental to ensure secure and personalized user interactions within the platform, safeguarding sensitive information and building trust. It establishes a secure gateway, allowing users to access personalized features, track orders, and maintain a consistent profile.

The Home List of Dishes serves as the digital menu, presenting a diverse array of culinary offerings. This feature aims to enhance user engagement by providing a visually appealing and user-friendly interface, enabling customers to explore and discover various dishes effortlessly. The integration of an intuitive add-to-cart mechanism streamlines the user experience, allowing customers to compile their desired items seamlessly. This process mirrors the physical act of selecting items in a traditional restaurant, fostering a sense of familiarity in the digital realm.

The addition of a secure payment system completes the **Shop-to-Table journey,** ensuring a frictionless transaction process. By incorporating industry-standard security measures, FoodieBookin prioritizes user data protection and financial security. The theoretical underpinning of this platform lies in its commitment to combining technology and gastronomy, delivering a harmonious blend of convenience and culinary exploration to users in the ever-evolving digital landscape.

1.3. System Analysis & Design vis-a-vis User Requirements

System Analysis & Design for FoodieBookin involves a meticulous examination of user requirements to ensure a seamless shop-to-table experience.

a. Authentication:

The system prioritizes user security by implementing robust authentication mechanisms. This involves user registration and login functionalities to safeguard personal information and transaction details. By employing secure authentication protocols, FoodieBookin ensures that only authorized users can access the platform, establishing trust in the user community.

b. Home List of Dishes

The heart of FoodieBookin lies in its user-friendly interface, presenting a comprehensive home list of dishes. Through effective categorization and search features, users can effortlessly navigate the diverse culinary offerings. This design aspect focuses on enhancing user experience, making it intuitive and enjoyable for customers to explore and discover new dishes based on their preferences.

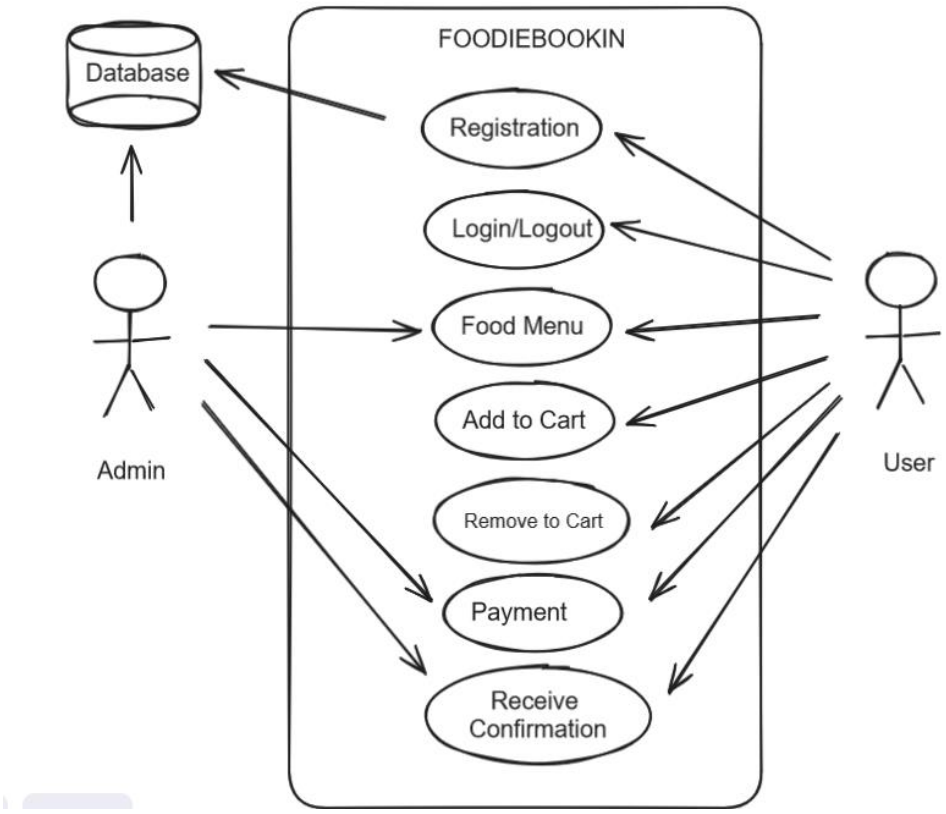
c. Add/remove to Cart:

Streamlining the ordering process, the Add to Cart functionality facilitates a convenient selection of dishes. Users can easily compile their desired items before proceeding to checkout. This design choice simplifies the user journey, providing a smooth and efficient way for customers to customize their orders without any unnecessary complexities.

d. Payment:

The payment process is designed for efficiency and security. Multiple payment options are integrated to cater to diverse user preferences. The system ensures that the payment gateway is seamless, quick, and reliable, thereby enhancing the overall user satisfaction and encouraging repeat transactions.

User Requirements : Use -Case Diagram



Use Case: FoodBookin

1. Introduction:

The FoodBookin is a software application designed to facilitate the ordering of food items by users. It provides a user-friendly interface for browsing a menu, adding items to a cart, making payments, and receiving order confirmations. The system involves various modules, including Registration, Login/Logout, Food Menu, Add to Cart, Remove from Cart, Payment, and Order Confirmation.

2. Actors:

- User: A person who interacts with the FoodBookin to browse the menu, place orders, and manage their account.

- Admin: An administrator user responsible for managing the food menu, user accounts, and overall system settings.

- Database: The backend storage system that stores user information, food menu details, and transaction records.

3. Pre-condition:

- The FoodBookin is installed and accessible.
- Users and admin accounts have been created in the system.
- The food menu is populated with items and prices.
- The database is operational and contains necessary information.

4. Post-condition:

- User orders are successfully processed, and confirmations are sent.
- Admin can manage the food menu, user accounts, and system settings.
- Database records are updated with new orders and user information.

5. Flow of Events:

5.1. Basic Flow:

- User Registration:
  1. User interacts with the system and selects the registration option.
  2. User provides required information, such as name, contact details, and delivery address.
  3. System validates the information and creates a new user account.

- User Login:
  1. User enters their credentials (username and password) to log into the system.
  2. System verifies the credentials and grants access to the user.

- Food Menu:
  1. User browses the available food items and selects items to add to the cart.
  2. System displays details of each food item, including price and customization options.

- Add to Cart:
  1. User adds selected items to the shopping cart.
  2. System updates the cart with the selected items and calculates the total cost.

- Remove from Cart:
  1. User removes items from the cart if needed.
  2. System updates the cart and recalculates the total cost.

- Payment:

1. User proceeds to the payment screen, enters payment details, and confirms the order.
2. System processes the payment through secure channels.

- Order Confirmation:

1. System generates an order confirmation with details of the order and estimated delivery time.
2. User receives the confirmation, and the order details are stored in the database.

5.2. Alternate Flow:

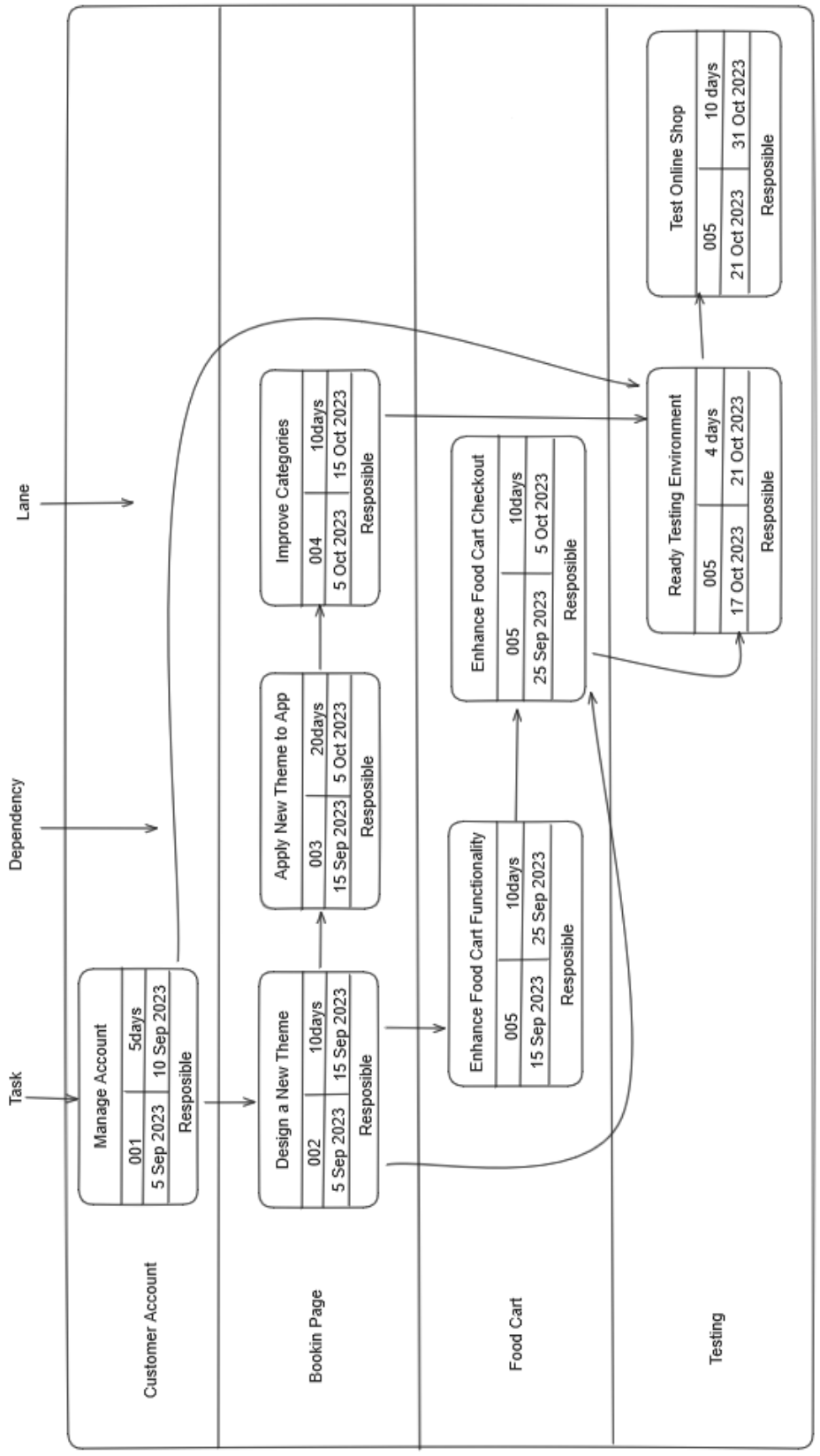
- User Login Failure:

- If the user enters incorrect credentials during login, the system displays an error message and prompts the user to re-enter the correct credentials.

6. Special Requirements:

- The FoodBookin should integrate secure payment gateways.
- The system should provide real-time updates on order status.
- Admin should have tools to manage the food menu and user accounts efficiently.

1.4. System Planning (PERT Chart)



The development and enhancement plan for FoodBookin involves a series of interconnected tasks, each contributing to the overall improvement of the application. The Project Evaluation and Review Technique (PERT) chart illustrates the sequential and dependent nature of these tasks.

The first task, Manage Account, involves implementing user account management features. This step is crucial for establishing a secure and personalized user experience. Once account management is in place, the next task, Design a New Theme, aims to enhance the visual aesthetics of the application. This includes conceptualizing and creating a new theme that aligns with the brand and user preferences.

Following the theme design, the task Apply New Theme to App involves integrating the newly designed theme into the existing application. This process ensures a cohesive and visually appealing user interface. Subsequently, Improve Categories focuses on refining the categorization of food items, optimizing the menu browsing experience.

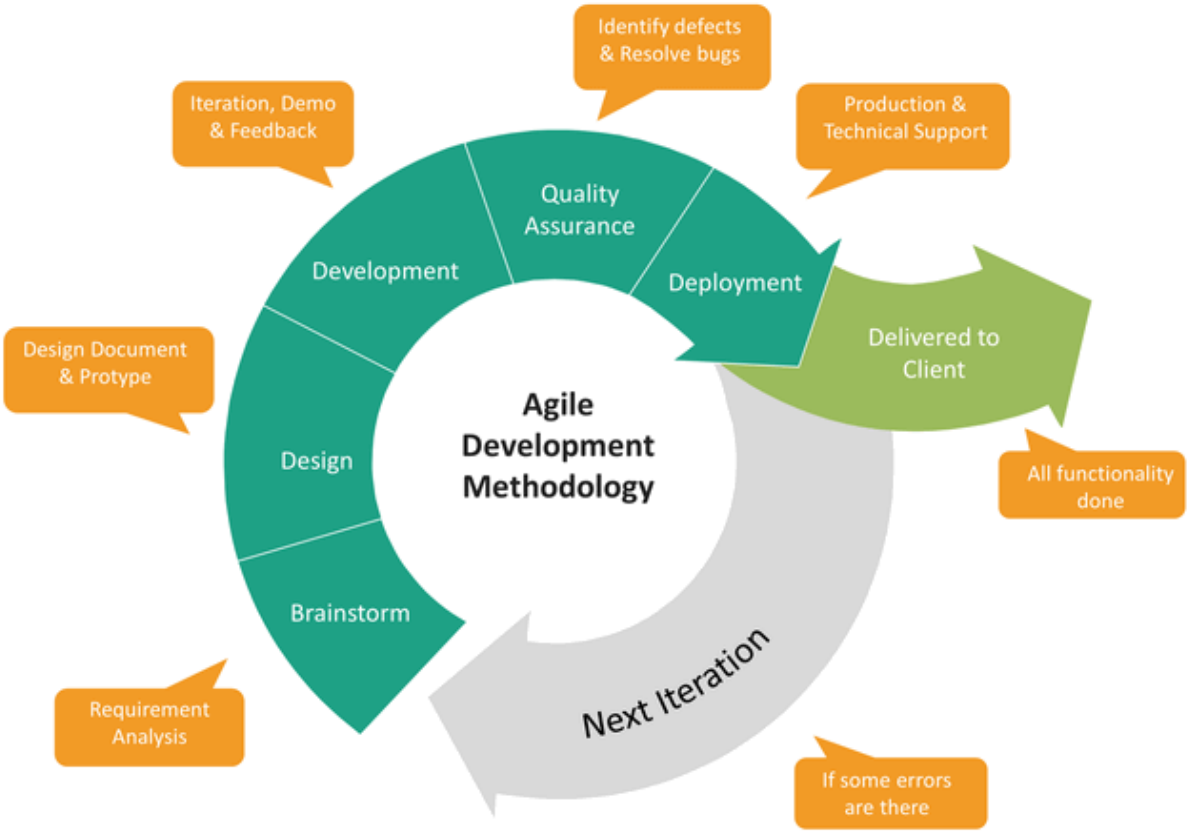
The tasks Enhance Food Cart Functionality and Enhance Food Cart Checkout work in tandem to improve the functionality and user experience of the shopping cart feature. This involves streamlining the process of adding, modifying, and removing items from the cart, as well as optimizing the checkout process for a smoother transaction.

The task Ready Training Environment is designed to prepare the necessary training resources and environment for both users and support staff. This includes documentation, tutorials, and simulated scenarios to ensure a comprehensive and effective training process. Simultaneously, Test Online Shop involves rigorous testing of the entire online shopping functionality, including user interactions, cart management, and payment processing, to identify and rectify any issues before deployment.

Each of these tasks is interconnected, with dependencies dictating the flow of the project. For example, Design a New Theme must precede Apply New Theme to App, and Manage Account should be completed before Improve Categories to ensure a logical and efficient progression of development. The PERT chart serves as a visual representation of these dependencies and aids in project planning and execution.

1.5. Methodology Adopted:

The development of the FoodBookin system followed an agile methodology, allowing for iterative and flexible development. This approach facilitated frequent collaboration with stakeholders to gather feedback and incorporate changes throughout the project lifecycle. The development process encompassed requirement analysis, design, implementation, testing, and deployment**.**



System Implementation:

FoodBookin was implemented using the Flutter framework, providing a cross-platform solution for both Android and iOS. The user interface was designed and developed in Android Studio and Visual Studio Code, utilizing their rich set of tools and features.

Details of Hardware & Software Used:-

Hardware:

- The system is designed to operate on standard smartphones with Android or iOS operating systems.
- Minimum hardware requirements include a device with at least 2 GB RAM and a dual-core processor.

Software:

Flutter Framework:

Flutter was chosen for its ability to create natively compiled applications for mobile from a single codebase.

Firebase:

Firebase services were integrated to manage user authentication, real-time database for menu updates, and cloud functions for payment processing.

Android Studio:

Android Studio served as the primary integrated development environment (IDE) for Android app development.

Visual Studio Code:

Visual Studio Code was used for its versatility and ease of use in cross-platform development.

System Maintenance:

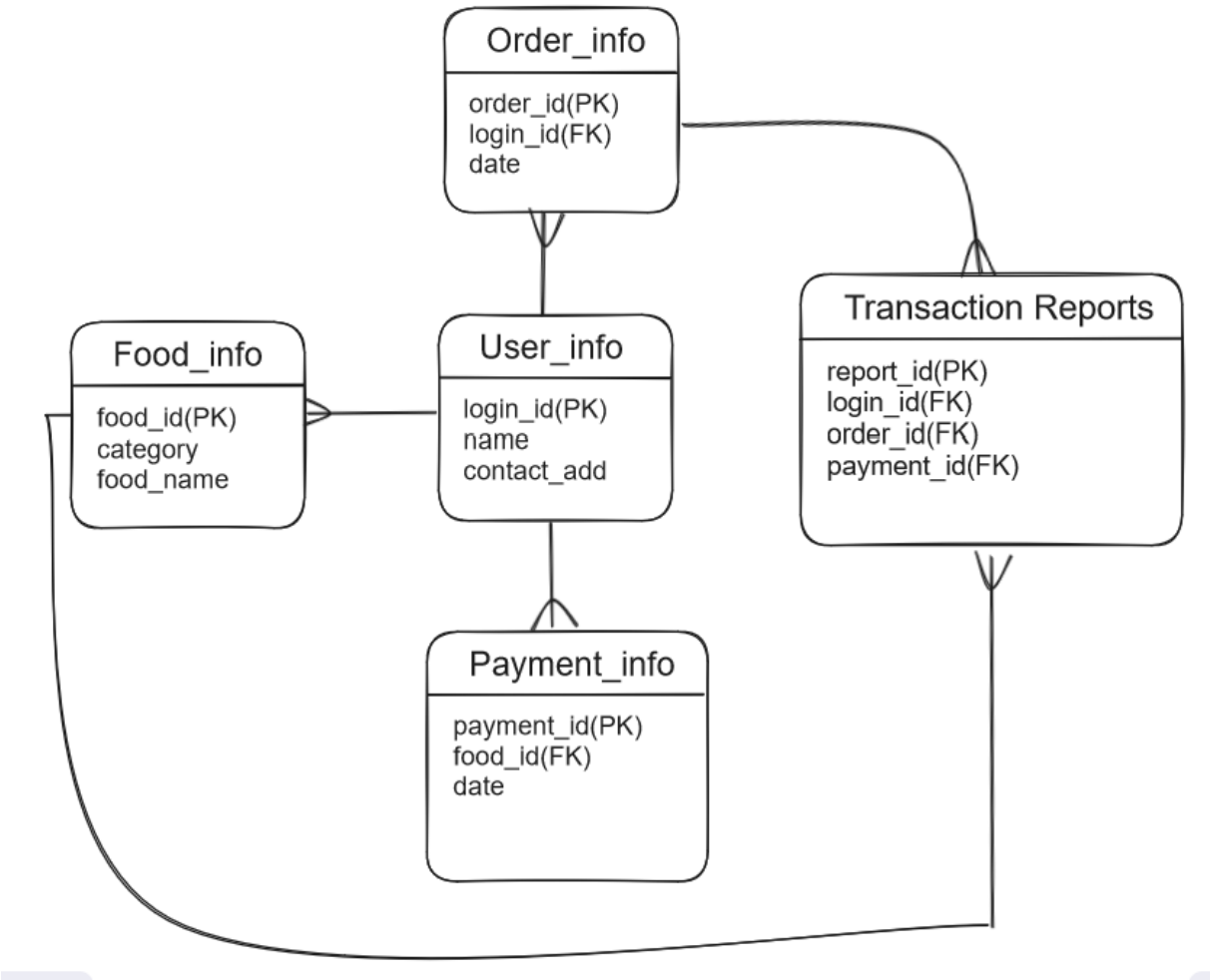
Regular updates and maintenance are planned to ensure the system's smooth operation. Maintenance tasks include bug fixes, feature enhancements, and updates to accommodate changes in platform requirements. The Firebase real-time database allows for seamless updates to the food menu and ensures that users receive the latest information.

Evaluation:

The system's performance and user experience will be continually evaluated through user feedback, monitoring system analytics, and addressing reported issues. This ongoing evaluation process aims to identify areas for improvement and enhance the overall efficiency and user satisfaction of FoodBookin.

1.6. Detailed Life Cycle of the Project

1.6.1 ERD



The Entity-Relationship (ER) diagram for FoodBookin encapsulates the intricate relationships between various entities, illustrating the system's data model. At its core are five major entities: User_info, Order_info, Food_info, Payment_info, and TransactionReports.

User_info encompasses essential details about users, such as usernames and passwords, crucial for authentication. This entity establishes a connection with other entities, particularly with Order_info, where each user may have multiple orders associated with their account.

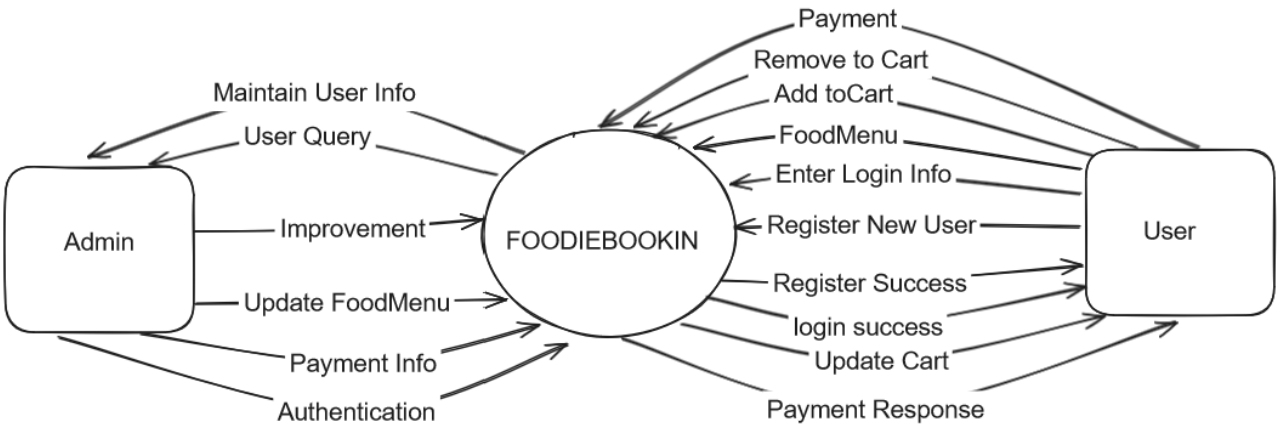
The Order_info entity serves as a central hub for order-related information. It captures data like order numbers, timestamps, and details about the items ordered. This entity is intricately linked to both User_info and Food_info. Users place orders, and each order comprises multiple food items, creating a many-to-many relationship between Order_info and Food_info.

Food_info contains details about the available food items in the system, including names, descriptions, and prices. This entity is connected to Order_info through the order placement process, enabling the association of specific food items with individual orders.

The Payment_info entity manages information related to user payments. It records transaction details, linking back to the Order_info entity, establishing a relationship that connects the financial aspect of the system with specific orders.

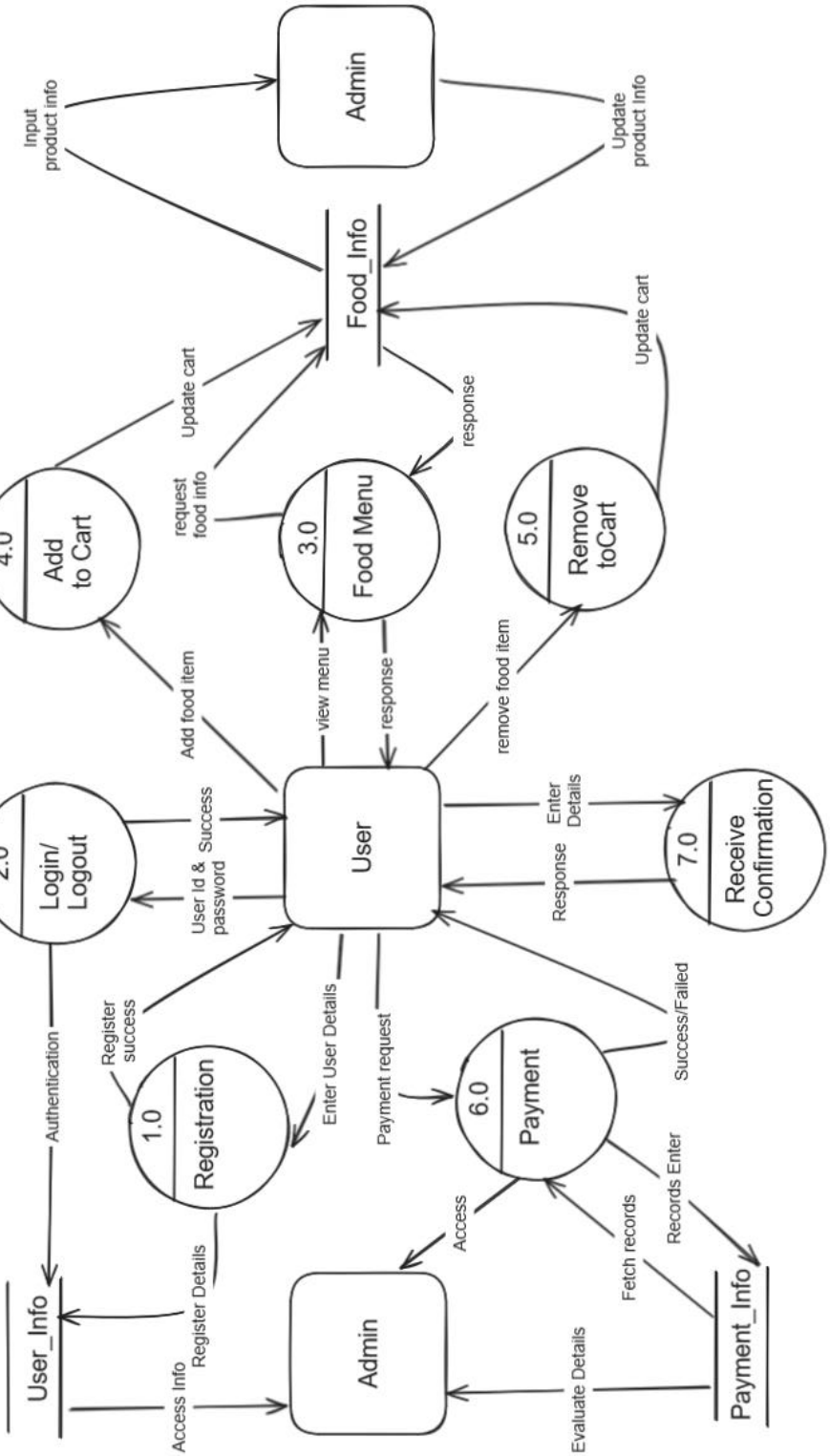
Lastly, the TransactionReports entity consolidates data related to transaction summaries. It draws information from Payment_info, providing a comprehensive overview of financial transactions within the system.

Level 0 Data Flow Diagram (DFD)



The context-level diagram for FoodieBookin provides a high-level overview of the system's interaction with external entities. At the center of the diagram is the FoodieBookin system, represented as a singular entity. The two primary external entities are Admin and User. The Admin interacts with the system to manage various functionalities, including user accounts, menu updates, and transaction monitoring. On the other hand, the User engages with the system for activities such as registering, logging in and out, browsing the food menu, managing the shopping cart, making payments, and receiving order confirmations. This context-level diagram offers a concise visual representation of the key entities and their interactions within the FoodieBookin system.

Level 1 Data Flow Diagram (DFD)



The Level 1 Data Flow Diagram (DFD) for FoodieBookin provides a high-level overview of the system's functional modules and their interactions. At the center of the diagram is the main process, representing the FoodieBookin system. The primary external entities are Admin and User, each interacting with the system through specific use-case modules.

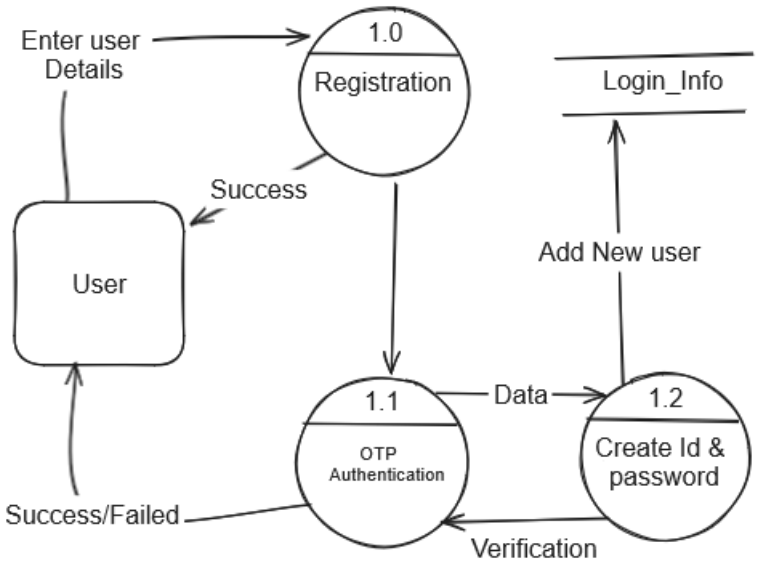
The Register module allows new users to create accounts, capturing and validating their information. The Login/Logout module facilitates secure user authentication, ensuring access control. The FoodMenu module presents the available food items, allowing users to browse and make selections. The Add to Cart and Remove from Cart modules manage the user's shopping cart, enabling dynamic updates.

The Payment module handles secure transaction processing, connecting with external payment services. The Receive Confirmation module ensures users and admins receive timely order confirmations, creating a feedback loop. Admin interacts with the system to manage food menus and oversee transactions.

This Level 1 DFD provides a concise representation of the system's major modules, external entities, and their interconnections, offering a foundational understanding of the data flow within FoodieBookin.

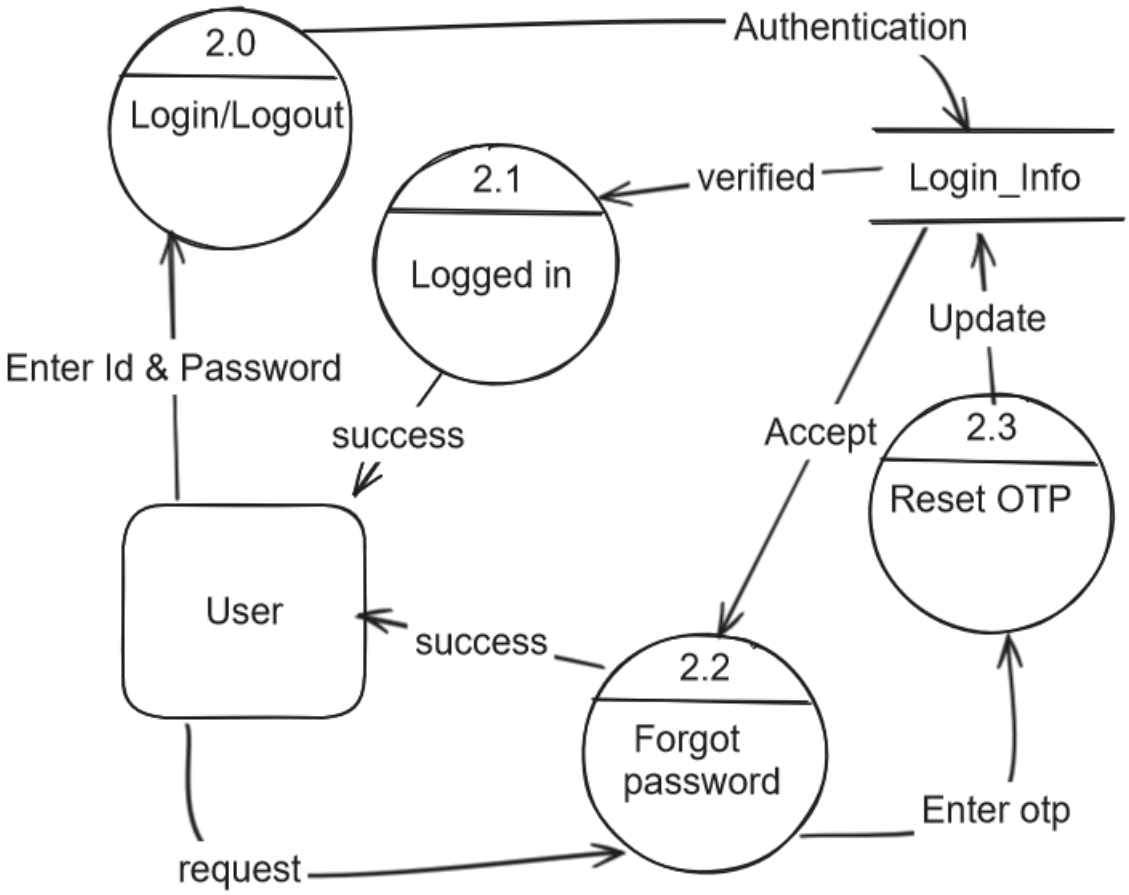
Level 2 Data Flow Diagram (DFD)

Module 1: Registration



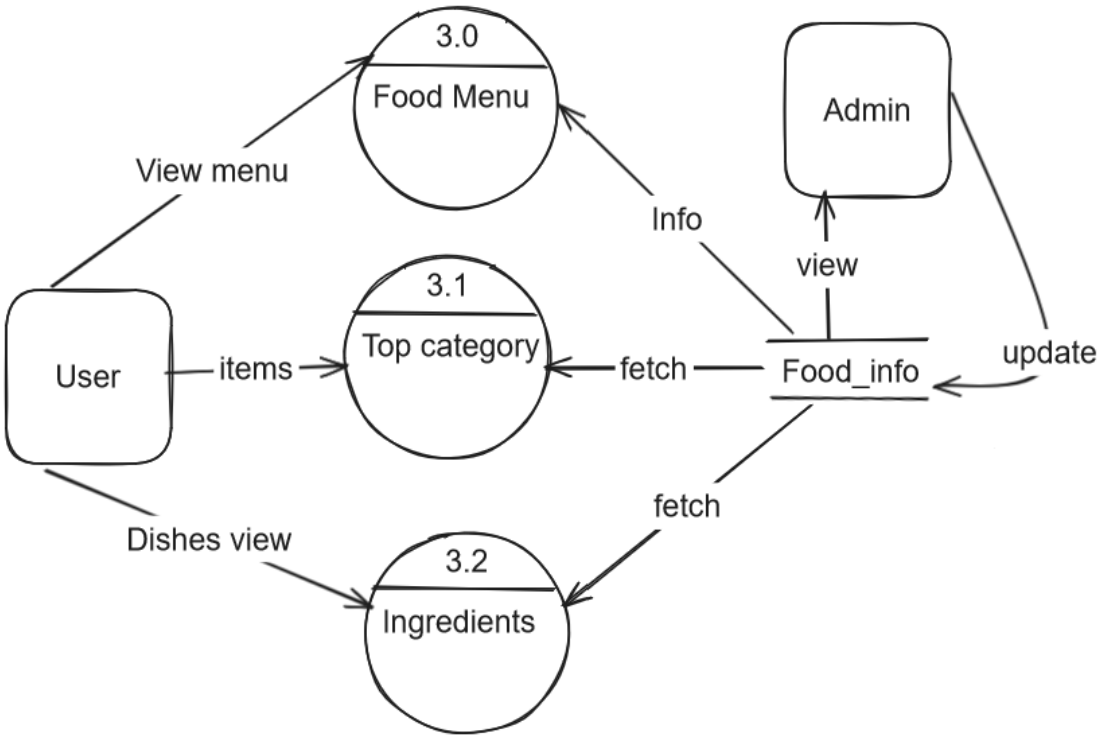
The Level 2 Data Flow Diagram (DFD) for the registration module in FoodieBookin illustrates the detailed flow of data within this specific function. At the center of the diagram is the Register process, representing the registration module. The process takes inputs such as user details and outputs the registered user information. It interacts with the Database entity, depicting the storage and retrieval of user data. The User Interface is shown as the external entity providing input, emphasizing the user's interaction in the registration process. This Level 2 DFD succinctly captures the data flow and interactions involved in user registration within the broader FoodieBookin system.

Module 2: Login/Logout



In the level 2 Data Flow Diagram (DFD) for FoodieBookin , the focus is on the detailed processes within the login/logout module. The diagram illustrates the flow of data and activities associated with user authentication. The login/logout module involves interactions between the Admin and User entities. When a user attempts to log in, the system verifies the credentials, granting access upon successful authentication. In the case of logout, the user's session is terminated. This DFD level 2 provides a concise representation of the specific actions and data flows within the login/logout module, enhancing the understanding of the authentication processes in FoodieBookin .

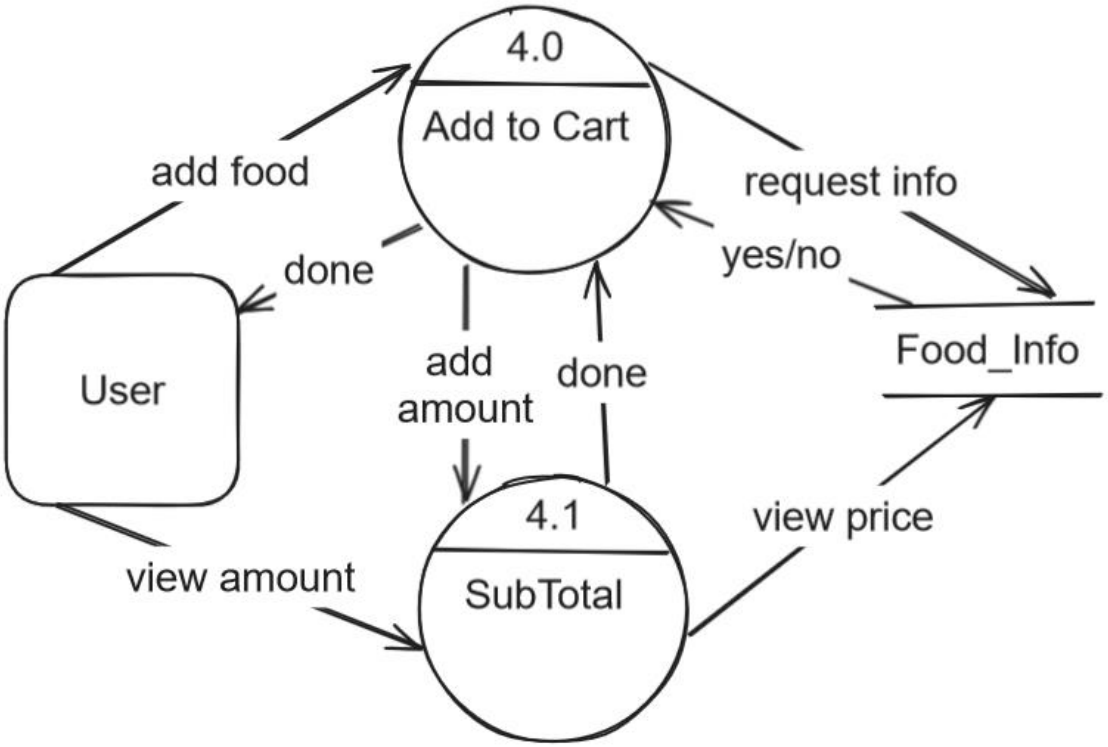
Module 3: FoodMenu



The Level 2 Data Flow Diagram (DFD) for the FoodieBookin system specifically focuses on the foodmenu module. In this diagram, the main entities include Admin and User, representing the system's stakeholders. The process of foodmenu is depicted, illustrating the flow of data within this module.

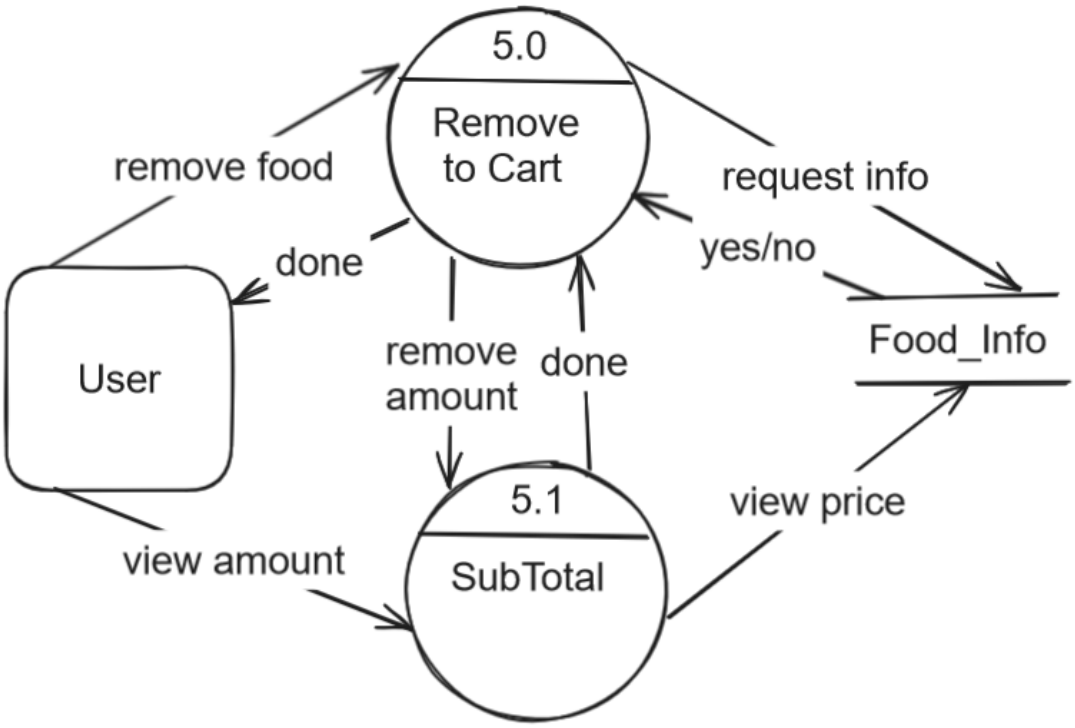
The foodmenu module allows users to interact with the system by browsing and selecting food items. The DFD showcases how information flows from the foodmenu process, enabling users to view and choose items. It also implies interactions with the Admin for menu updates. The diagram simplifies the understanding of data movement within the foodmenu functionality, emphasizing its importance in the overall FoodieBookin system.

Module 4: Add to Cart



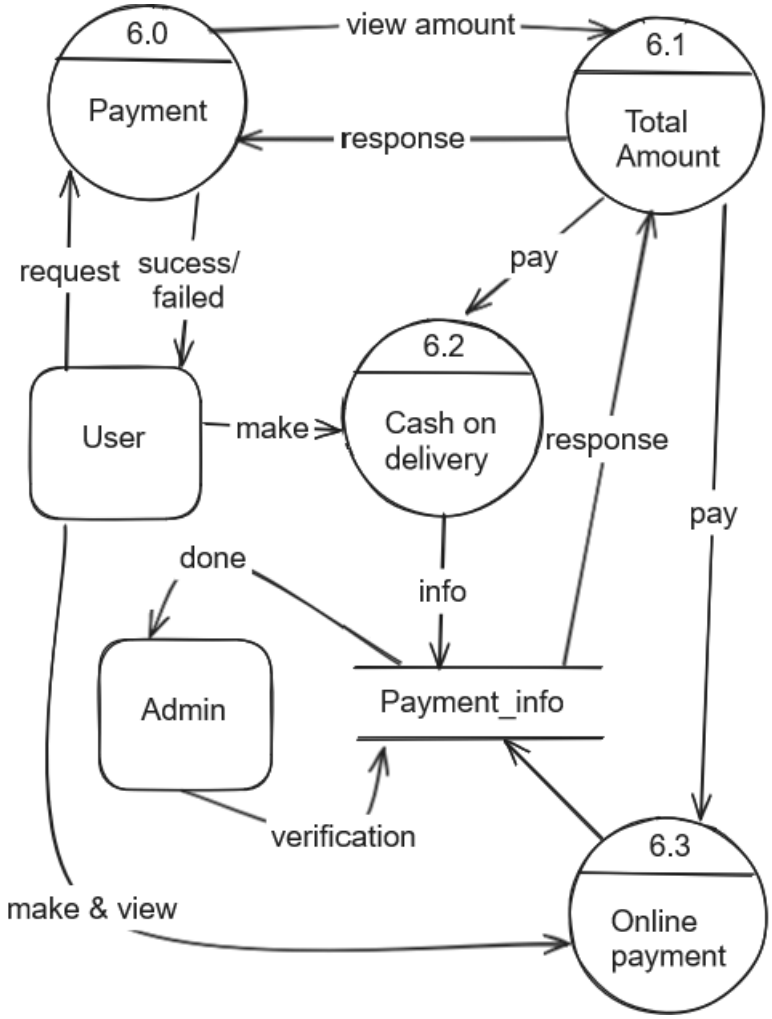
The Level 2 Data Flow Diagram (DFD) for FoodieBookin specifically focuses on the Add to Cart use case within the broader system. At this level, the diagram provides a detailed view of the processes, data stores, and data flows involved in the Add to Cart functionality. The primary entities include the Admin and User, while modules encompass Register, Login/Logout, Food Menu, Add to Cart, Remove from Cart, Payment, and Receive Confirmation. The Add to Cart module specifically delineates the steps and interactions involved in selecting and adding items to the user's cart, illustrating the seamless flow of data within this specific function.

Module 4: Remove to Cart



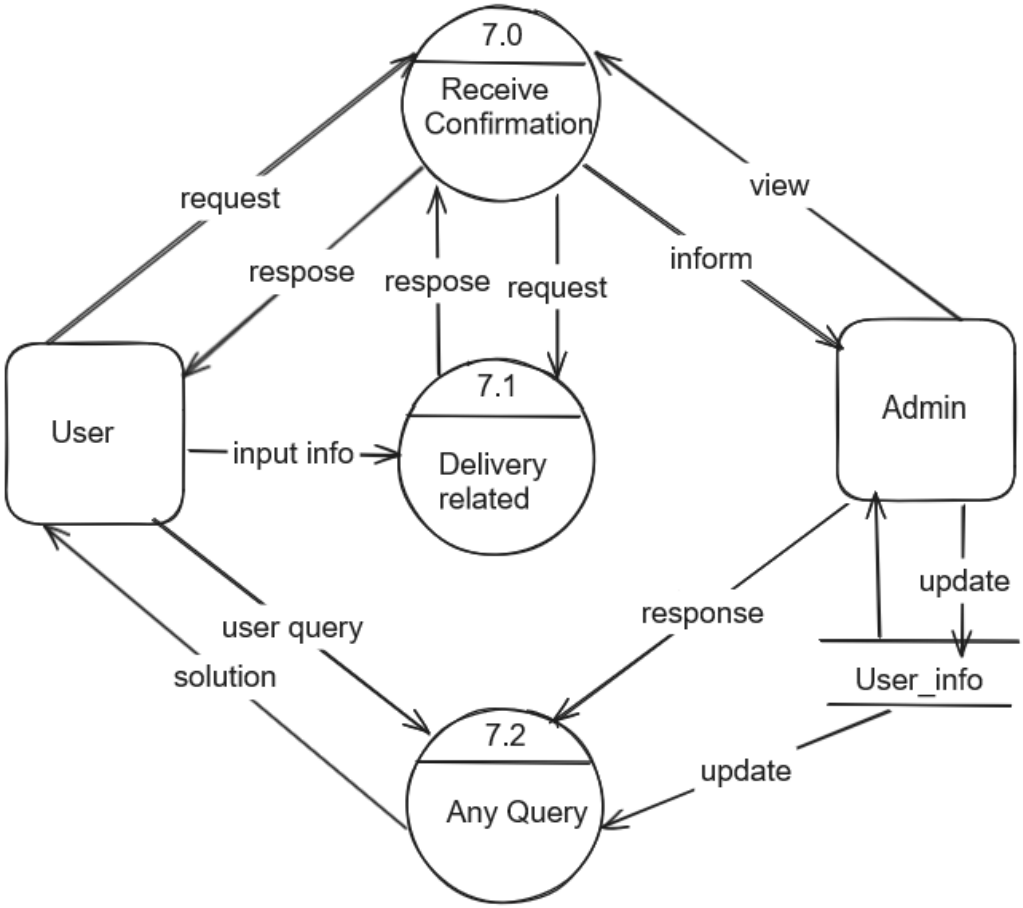
In the level 2 Data Flow Diagram (DFD) for FoodieBookin , the Remove from Cart module is detailed. This module focuses on the process of users removing items from their shopping cart. The diagram illustrates the flow of data and processes involved in this specific action. It begins with the user's interaction with the Remove from Cart function in the user interface, triggering a data flow that updates the Cart data store. The system processes this request, adjusts the cart contents accordingly, and provides feedback to the user. This level of detail in the DFD highlights the specific components and interactions associated with the Remove from Cart functionality in a concise and comprehensive manner.

Module 4: Payment



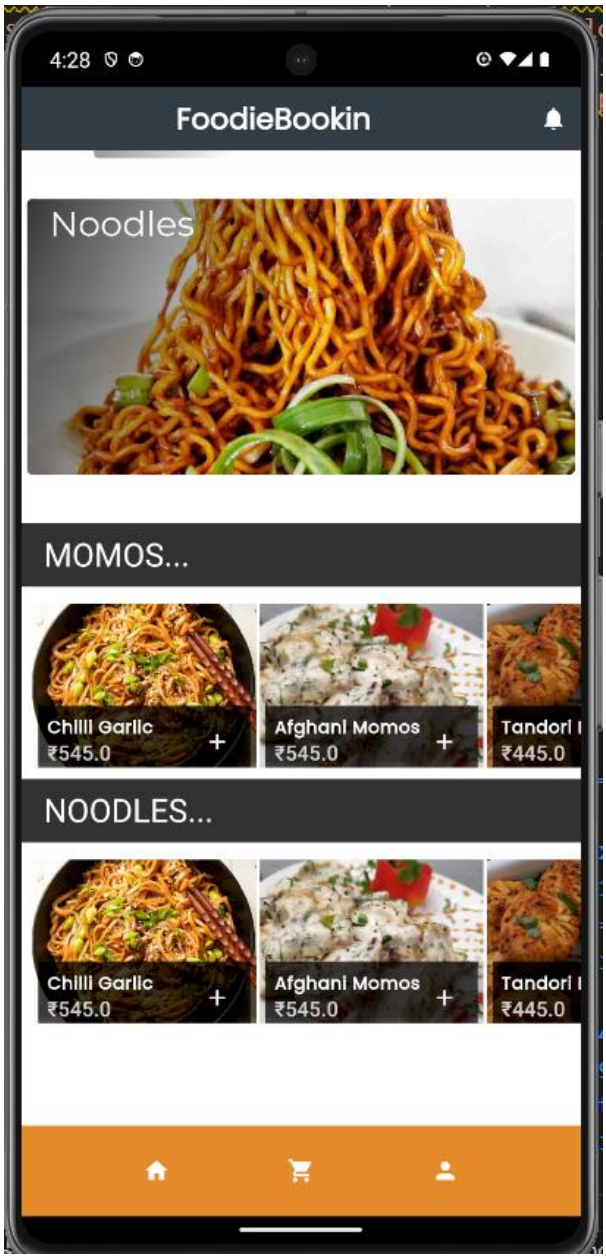
In the Level 2 Data Flow Diagram (DFD) for the FoodieBookin system, the focus is on the payment module. The diagram illustrates the flow of data within this specific module. Entities such as User and Admin interact with the Payment module through various processes, including order confirmation and transaction processing. Data flows between these entities and processes, showcasing the intricate steps involved in payment transactions. This detailed representation allows for a clear understanding of the specific functionalities and interactions within the payment module of FoodieBookin .

Module 4: Receive Confirmation



The Level 2 Data Flow Diagram (DFD) for FoodieBookin primarily focuses on the Receive Confirmation module. In this context, the diagram illustrates the specific processes and data interactions related to the confirmation receipt within the system. It encapsulates the flow of information from various sources, demonstrating how user actions, order processing, and payment confirmation contribute to the final step of receiving confirmation. This detailed level of the DFD provides a succinct representation of the intricacies involved in confirming and completing transactions within the FoodieBookin application.

1.6.2 Input and Output Screen Design



1.6.3 Process involved

The FoodBookin system encompasses a series of fundamental processes that collectively contribute to its seamless and efficient functionality. The first crucial process is user authentication, which occurs when users input their credentials on the login screen. This information is then verified against the Firebase authentication service, ensuring that only authorized users gain access to the system. This layer of security enhances the overall integrity of the platform.

The second integral process involves menu browsing and cart management. Users navigate through the food menu using an intuitive interface, allowing them to peruse available items and make selections. The user interface facilitates the dynamic addition or removal of items from the virtual cart, providing a responsive and interactive experience. Additionally, users have the flexibility to customize their selections based on individual preferences, adding a personalized touch to their orders.

The final key process in the FoodBookin system is payment processing. Once users finalize their orders, the payment screen is activated to facilitate the secure collection and processing of payment details. This critical step ensures the confidentiality and integrity of financial transactions. The use of Firebase cloud functions enhances the system's security during payment processing, providing a robust and reliable mechanism for completing transactions.

Collectively, these processes form a coherent and user-centric workflow within the FoodBookin system. From the initial authentication to the dynamic management of food selections and the secure processing of payments, each step is meticulously designed to enhance the user experience while maintaining the integrity and security of the entire transactional process. The result is a comprehensive and user-friendly system that seamlessly integrates authentication, menu exploration, cart management, and payment processing into a cohesive and efficient platform.

2. CODING AND SCREENSHOTS OF THE PROJECT

2.1 MAIN CODE

```
import 'package:demo/firebase_options.dart';
import 'package:demo/src/features/auth/screens/splash/splash.dart';
import 'package:demo/src/features/cart/controlers/blocs/cart_bloc.dart';
import 'package:demo/src/features/home/screen/home/Bottom.dart';
import 'package:demo/src/repository/auth_repo/authentication_repo.dart';
import 'package:demo/src/utils/theme/theme.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:get/get.dart';
import 'package:get/route_manager.dart';

import 'src/features/cart/controlers/blocs/cart_event.dart';


final ColorScheme lColorScheme = ColorScheme.fromSwatch(
    primarySwatch: Colors.blue, brightness: Brightness.light);
final ColorScheme dColorScheme = ColorScheme.fromSwatch(
    primarySwatch: Colors.amber, brightness: Brightness.dark);
void main() {
  WidgetsFlutterBinding.ensureInitialized();

  Firebase.initializeApp(options:
DefaultFirebaseOptions.currentPlatform).then(
    (value) => Get.put(
      AuthenticationRepo(),
    ),
  );

  runApp(const FoodApp());
}

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

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
```

```
    return BlocProvider(
      create: (context) => CartBloc()..add(CartStarted()),
      child: GetMaterialApp(
        title: Row Chinese,
        debugShowCheckedModeBanner: false,
        theme: Apptheme.lighttheme,
        darkTheme: Apptheme.darktheme,
        themeMode: ThemeMode.system,
        transitionDuration: const Duration(milliseconds: 500),
        home:
            // BottomScreen(),
            CircularProgressIndicator(),
      ),
    );
  }
}
```

2.2 SPLASH SCREEN

CODE:

```
import
package:demo/src/commonWiget/customAnimation/UpwardAnimationWidget.dart;
import
'package:demo/src/commonWiget/customAnimation/UpwardAnimationcontroller.dart';
import package:demo/src/commonWiget/customAnimation/UpwardAnimationmodel.dart;
import package:demo/src/constants/colors.dart;
import package:demo/src/constants/image_str.dart;
import package:demo/src/constants/sizes.dart;
import package:flutter/material.dart;
import package:get/get.dart;

import ../../../../constants/text_str.dart;

class SplashSc extends StatelessWidget {
  SplashSc({super.key});
  @override
  Widget build(BuildContext context) {
    var isDark = MediaQuery.of(context).platformBrightness == Brightness.dark;

    final size = MediaQuery.of(context).size;
    final controller = Get.put(UpwardAnimationController());
    controller.startAnimate();
    return SafeArea(
      child: Scaffold(
        backgroundColor: isDark ? mDarkColor : mWhiteColor,
        body: Stack(
          children: [
            UpwardAnimation(
              durationInMls: 1700,
              animPosition: CAnimatedposition(
                topAfter: -20,
                topBefore: -40,
                leftAfter: -55,
                leftBefore: -80,
              ),
              child: Image(
                image: AssetImage(
                  mSplashTopIcon,
                ),
                height: size.height  0.2,
                width: size.height  0.4,
              ),
            ),
            UpwardAnimation(
              durationInMls: 1700,
```

```
            animPosition: CAnimatedposition(
                topAfter: 200,
                topBefore: 200,
                leftAfter: mDefaultSize,
                leftBefore: -80,
                rightAfter: 0,
                rightBefore: 0),
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Text(
                  mAppName,
                  style: Theme.of(context).textTheme.headlineSmall,
                ),
                Text(
                  mAppTagline,
                  textAlign: TextAlign.justify,
                  style: Theme.of(context).textTheme.headlineMedium,
                ),
              ],
            ),
          ),
          UpwardAnimation(
            durationInMls: 1700,
            animPosition: CAnimatedposition(
              rightAfter: 0,
              bottomBefore: 50,
              bottomAfter: 200,
            ),
            child: Image(
              image: AssetImage(mWelcomeimg),
              width: size.width  0.9,
            ),
          ),
          UpwardAnimation(
            durationInMls: 1700,
            animPosition: CAnimatedposition(
                bottomBefore: 0,
                bottomAfter: 70,
                rightAfter: mDefaultSize,
                rightBefore: mDefaultSize),
            child: Container(
              width: mSplashContainerSize,
              height: mSplashContainerSize,
              decoration: BoxDecoration(
                  borderRadius: BorderRadius.circular(150),
                  color: mPrimaryColor),
            ),
```

```
            ),
          ],
        ),
      ),
    );
  }
}
```

OUTPUT:

2.3 WELCOME SCREEN

CODE:

```
import
'package:demo/src/commonWiget/customAnimation/UpwardAnimationWidget.dart';
import
'package:demo/src/commonWiget/customAnimation/UpwardAnimationmodel.dart';
import 'package:demo/src/constants/colors.dart';
import 'package:demo/src/constants/image_str.dart';
import 'package:demo/src/constants/sizes.dart';
import 'package:demo/src/constants/text_str.dart';
import 'package:demo/src/features/auth/screens/login/login_screen.dart';
import 'package:demo/src/features/auth/screens/signin/signin_screen.dart';
import 'package:flutter/material.dart';
import 'package:get/get.dart';

import
'../../../../commonWiget/customAnimation/UpwardAnimationcontroller.dart';

class AuthScreen extends StatefulWidget {
  AuthScreen({super.key});

  @override
  State<AuthScreen> createState() => _AuthScreenState();
}

class _AuthScreenState extends State<AuthScreen> {
  @override
  Widget build(BuildContext context) {
    final controller = Get.put(UpwardAnimationController());
    controller.startAuthAnimate();

    final isDark = MediaQuery.of(context).platformBrightness ==
Brightness.dark;
    var height = MediaQuery.of(context).size.height;
    return Scaffold(
      backgroundColor: isDark ? mDarkColor : Colors.white,
      body: Stack(
        children: [
          UpwardAnimation(
            durationInMls: 1300,
            animPosition: CAnimatedposition(
                bottomAfter: 0,
                bottomBefore: -600,
                leftAfter: 0,
                leftBefore: 0,
                rightAfter: 0,
```

```
            rightBefore: 0,
            topAfter: 0,
            topBefore: 0),
        child: Container(
          padding: const EdgeInsets.all(mDefaultSize),
          child: Column(
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            children: [
              Image(
                height: height * 0.48,
                width: 700,
                image: AssetImage(mWelcomeimg2),
                fit: BoxFit.cover,
              ),
              Column(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  Text(
                    mWelcomeTitle,
                    style: Theme.of(context).textTheme.headlineMedium,
                  ),
                  Text(
                    mWelcomeSubTitle,
                    style: Theme.of(context).textTheme.titleSmall,
                    textAlign: TextAlign.center,
                  ),
                ],
              ),
              Row(
                children: [
                  Expanded(
                    child: OutlinedButton(
                      onPressed: () => Get.to(() => loginScreen()),
                      child: Text(
                        "login".toUpperCase(),
                        style: isDark
                            ? Theme.of(context).textTheme.labelLarge
                            : TextStyle(color: Colors.black),
                      ),
                      style: Theme.of(context).outlinedButtonTheme.style,
                    ),
                  ),
                  SizedBox(
                    width: 10,
                  ),
                  Expanded(
                    child: ElevatedButton(
                      // onPressed: () {},
```

```
                        onPressed: () => Get.to(() => SigninScreen()),
                        child: Text(
                          "Signup".toUpperCase(),
                          style: Theme.of(context).textTheme.labelLarge,
                        ),
                        style: Theme.of(context).elevatedButtonTheme.style,
                      ),
                    ),
                  ],
                )
              ],
            ),
          ),
        ),
      ],
    ),
  );
  }
}
```

OUTPUT:

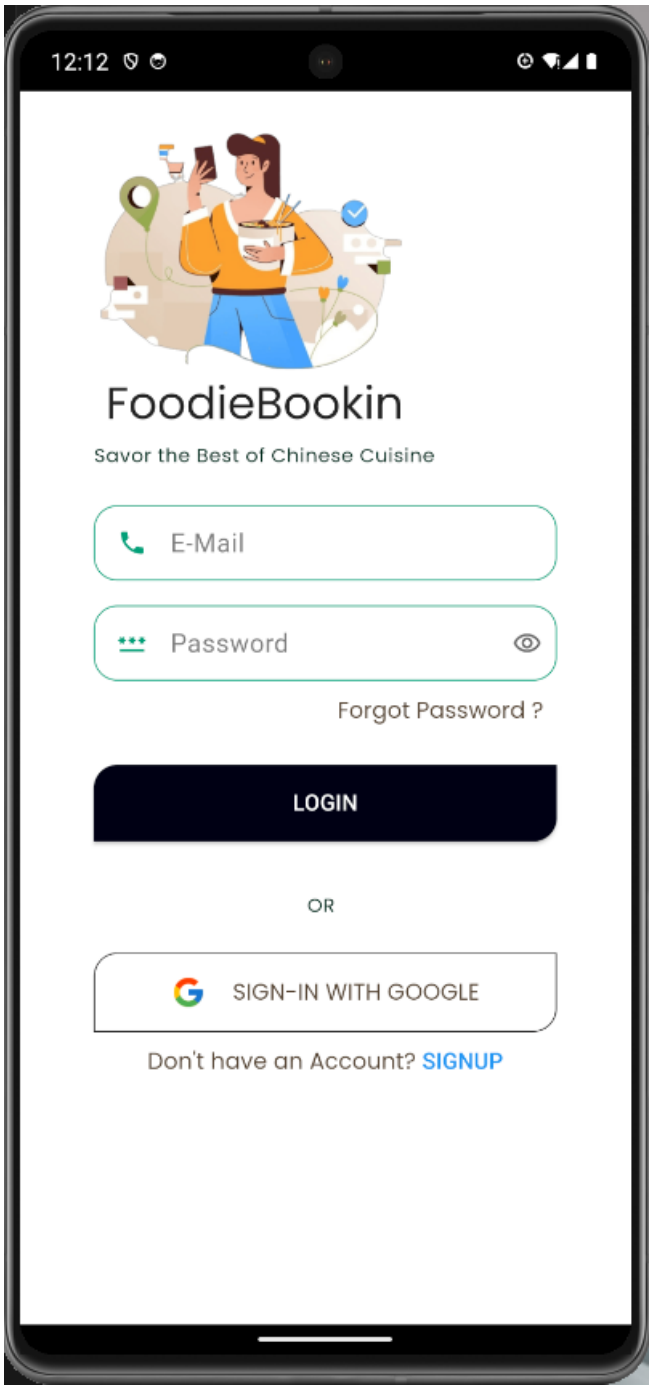

12:05
Delisious FoodieBookin
Let's Savor Authentic Chinese Flavors with
Every Bite: Your Ultimate Food Adventure!
LOGIN
SIGNUP

2.4 LOGIN SCREEN

CODE:

```
import 'package:demo/src/constants/image_str.dart';
import 'package:demo/src/constants/text_str.dart';
import 'package:demo/src/features/auth/screens/login/widget/loginfooter.dart';
import 'package:demo/src/features/auth/screens/login/widget/loginform.dart';
import 'package:flutter/material.dart';

import '../../../../commonWiget/form/formHeaderWidget.dart';

class loginScreen extends StatelessWidget {
  loginScreen({super.key});

  @override
  Widget build(BuildContext context) {
    final size = MediaQuery.of(context).size;
    return SafeArea(
      child: Scaffold(
        body: SingleChildScrollView(
            child: Container(
          padding: EdgeInsets.symmetric(vertical: 30, horizontal: 60),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.stretch,
            children: [
              HeaderForm(
                size: size,
                mheight: 0.19,
                title: mLoginPagetitle,
                subtitle: mLoginSubPagetitle,
                img: mWelcomeimg,
              ),
              const LoginForm(),
              LoginFooter(sizeh: size)
            ],
          ),
        )),
      ),
    );
  }
}
```

OUTPUT:

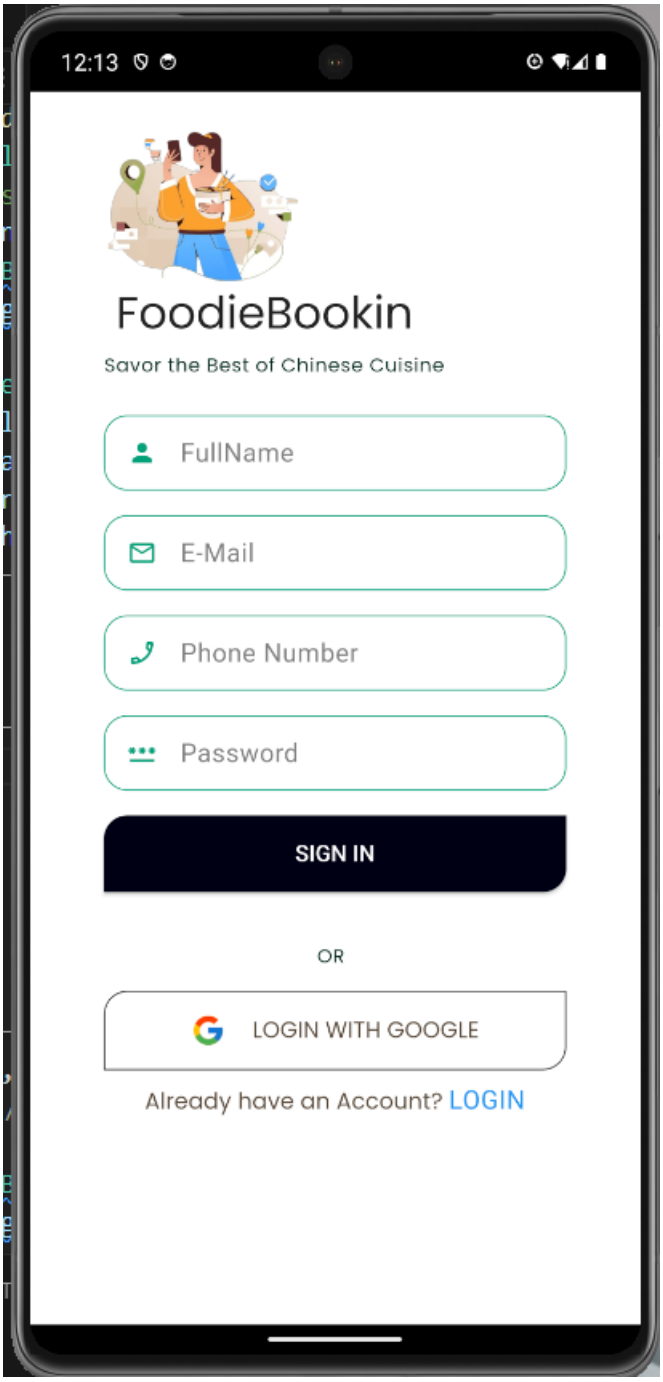

12:12
FoodieBookin
Savor the Best of Chinese Cuisine
E-Mail
Password
Forgot Password ?
LOGIN
OR
SIGN-IN WITH GOOGLE
Don't have an Account? SIGNUP

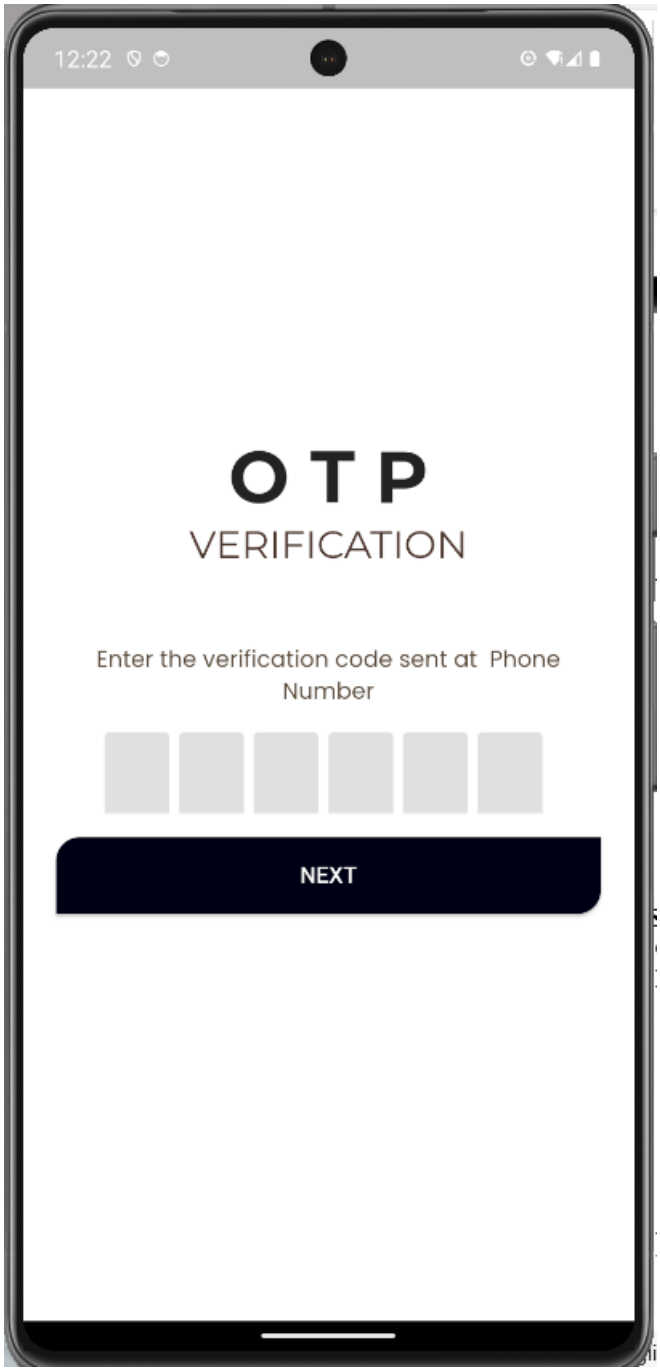
2.5 SIGNIN SCREEN

CODE :

```
import 'package:demo/src/constants/image_str.dart';
import 'package:demo/src/constants/text_str.dart';
import 'package:demo/src/features/auth/screens/login/widget/loginfooter.dart';
import 'package:demo/src/features/auth/screens/login/widget/loginform.dart';
import
'package:demo/src/features/auth/screens/signin/widget/login_footer.dart';
import 'package:demo/src/features/auth/screens/signin/widget/signform.dart';
import 'package:flutter/material.dart';

import '../../../../commonWiget/form/formHeaderWidget.dart';

class SigninScreen extends StatelessWidget {
  SigninScreen({super.key});

  @override
  Widget build(BuildContext context) {
    final size = MediaQuery.of(context).size;
    return SafeArea(
      child: Scaffold(
        body: SingleChildScrollView(
            child: Container(
          padding: EdgeInsets.symmetric(vertical: 30, horizontal: 60),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.stretch,
            children: [
              HeaderForm(
                  mtextAlign: TextAlign.left,
                  size: size,
                  img: mWelcomeimg,
                  mheight: 0.12,
                  title: mSignUpPagetitle,
                  subtitle: mSignUPSubPagetitle),
              const SignForm(),
              SigninFooter(sizeh: size)
            ],
          ),
        )),
      ),
    );
  }
}
```

OUTPUT:


12:13
FoodieBookin
Savor the Best of Chinese Cuisine
FullName
E-Mail
Phone Number
Password
SIGN IN
OR
LOGIN WITH GOOGLE
Already have an Account? LOGIN

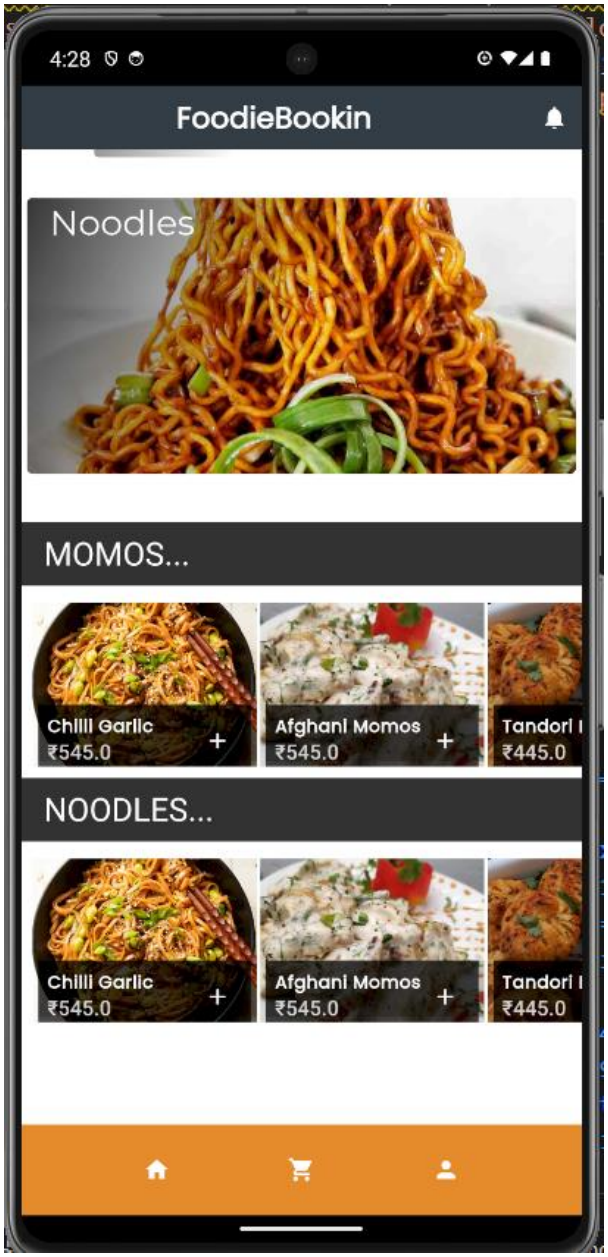

12:22
OTP
VERIFICATION
Enter the verification code sent at Phone Number
NEXT

2.6 HOME SCREEN

CODE:

```
import 'package:carousel_slider/carousel_slider.dart';
import
'package:demo/src/features/home/screen/Product/model/productmodel.dart';
import 'package:demo/src/features/home/screen/category/model/hcategory.dart';
import 'package:demo/src/features/home/screen/home/widget/nameOfproduct.dart';
import
'package:demo/src/features/home/screen/home/widget/productCarousel.dart';
import 'package:flutter/material.dart';

import 'widget/categ.dart';

class HomeScreen extends StatelessWidget {
  HomeScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return SingleChildScrollView(
      child: Column(
        children: [
          Container(
              child: CarouselSlider(
            options: CarouselOptions(
              aspectRatio: 1.5,
              enlargeCenterPage: true,
              scrollDirection: Axis.vertical,
              // autoPlay: true,
            ),
            items: Category.categories
                .map((category) => ImagCategory(category: category))
                .toList(),
          )),
          NamingOfProduct(name: "Momos..."),
          // ProductOne(product: Product.products[0]),
          ProductCarousel(products: Product.products),

          NamingOfProduct(name: "Noodles..."),
          // ProductOne(product: Product.products[0]),

          ProductCarousel(products: Product.products),
        ],
      ),
    );
```

```
    }
}
```


4:28
FoodieBookin
Noodles
MOMOS...
Chilli Garlic
₹545.0
Afghani Momos
₹545.0
Tandori I
₹445.0
NOODLES...
Chilli Garlic
₹545.0
Afghani Momos
₹545.0
Tandori I
₹445.0

2.7 CART SCREEN

CODE:

```
import 'package:demo/src/features/cart/controlers/blocs/cart_bloc.dart';
import 'package:demo/src/features/cart/model/cart_m.dart';
import 'package:demo/src/features/cart/widget/cart-list.dart';
import 'package:demo/src/features/payment/screen/pyscreen.dart';

import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:get/get.dart';

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

  @override
  Widget build(BuildContext context) {
    return BlocBuilder<CartBloc, CartState>(
      builder: (context, state) {
        if (state is CartLoading) {
          return Center(
            child: CircularProgressIndicator(),
          );
        }
        if (state is CartLoaded) {
          return Padding(
            padding: const EdgeInsets.symmetric(horizontal: 20, vertical: 10),
            child: Column(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: [
                SingleChildScrollView(
                  child: Column(
                    children: [
                      SizedBox(
                        height: 410,
                        child: ListView.builder(
                          itemCount: state.cart
                              .productQuantity(state.cart.products)
                              .keys
                              .length,
                          itemBuilder: (context, index) {
                            return CartProductCart(
                              product: state.cart
                                  .productQuantity(state.cart.products)
                                  .keys
                                  .elementAt(index),
                              quantity: state.cart
```

```
                            .productQuantity(state.cart.products)
                            .values
                            .elementAt(index),
                      );
                    },
                  ),
                ),
              ],
            ),
          ),
          Column(
            children: [
              Divider(
                color: Theme.of(context).dividerColor,
                thickness: 3,
              ),
              Padding(
                padding: const EdgeInsets.symmetric(
                    vertical: 10.0, horizontal: 30.0),
                child: Column(
                  children: [
                    Row(
                      mainAxisAlignment: MainAxisAlignment.spaceBetween,
                      children: [
                        Text(
                          'subtotal'.toUpperCase(),
                          style: Theme.of(context)
                              .textTheme
                              .labelSmall!
                              .copyWith(
                                  fontSize: 12,
                                  fontWeight: FontWeight.w700),
                        ),
                        Text(
                          '\u{20B9} ${state.cart.subtotalString}',
                          style: Theme.of(context)
                              .textTheme
                              .labelSmall!
                              .copyWith(
                                  fontSize: 12,
                                  fontWeight: FontWeight.w700),
                        )
                      ],
                    ),
                    Row(
                      mainAxisAlignment: MainAxisAlignment.spaceBetween,
                      children: [
                        Text(
```

```
                        'delivery fee'.toUpperCase(),
                        style: Theme.of(context)
                            .textTheme
                            .labelSmall!
                            .copyWith(
                                fontSize: 12,
                                fontWeight: FontWeight.w700),
                      ),
                      Text(
                        '\u{20B9} ${state.cart.deliveryfeeString}',
                        style: Theme.of(context)
                            .textTheme
                            .labelSmall!
                            .copyWith(
                                fontSize: 12,
                                fontWeight: FontWeight.w700),
                      )
                    ],
                  ),
                ],
              ),
            ),
            InkWell(
              onTap: () {
                Get.to(() => Pyscreen(
                      amount: 4870,
                    ));
              },
              child: Stack(
                children: [
                  Container(
                    width: MediaQuery.of(context).size.width,
                    height: 60,
                    decoration: BoxDecoration(
                        color:
                            const Color.fromARGB(255, 173, 195, 231)),
                  ),
                  Container(
                    width: MediaQuery.of(context).size.width - 10,
                    height: 55,
                    margin: EdgeInsets.all(1),
                    decoration: BoxDecoration(
                        color: Colors.black.withOpacity(0.7)),
                    child: Padding(
                      padding: const EdgeInsets.symmetric(
                          vertical: 10.0, horizontal: 20.0),
                      child: Row(
                        mainAxisAlignment:
```

```
                                MainAxisAlignment.spaceBetween,
                            children: [
                              Text(
                                'total'.toUpperCase(),
                                style: Theme.of(context)
                                    .textTheme
                                    .labelLarge!
                                    .copyWith(
                                        fontSize: 14,
                                        fontWeight: FontWeight.w700),
                              ),
                              Text(
                                '\u{20B9} ${state.cart.totalString()}',
                                style: Theme.of(context)
                                    .textTheme
                                    .labelLarge!
                                    .copyWith(
                                        fontSize: 14,
                                        fontWeight: FontWeight.w700),
                              )
                            ],
                          ),
                        ),
                      )
                    ],
                  ),
                ),
                SizedBox(
                  height: 10,
                ),
                Row(
                  mainAxisAlignment: MainAxisAlignment.spaceBetween,
                  children: [
                    OutlinedButton(
                        onPressed: () {}, child: Text("Cash On
Delivery")),
                    OutlinedButton(
                        onPressed: () {
                          Get.to(() => Pyscreen(
                                amount: state.cart.totalString().toInt(),
                              ));
                        },
                        child: Text(
                          "Payment Now ",
                        )),
                  ],
                )
              ],
```

```
                ),
              ],
            ),
          );
        } else {
          return Text("Something went wrong ");
        }
      },
    );
  }
}
```

OUTPUT:

3. CONCLUSION AND FUTURE SCOPE

In conclusion, FoodieBookin presents a comprehensive and user-centric food ordering system with key modules including user registration, login/logout, food menu exploration, cart management (add to/remove from cart), secure payment processing, and the receipt of order confirmations. The system, designed for both users and administrators, ensures a seamless and efficient experience in the realm of online food booking. The integration of these modules creates a robust platform that caters to the diverse needs of users while providing administrators with the tools to manage and oversee the operations effectively.

Looking towards the future, there are exciting prospects for enhancing FoodieBookin. Potential avenues for improvement may include the integration of advanced recommendation systems for personalized menus, incorporation of feedback mechanisms for users to share their experiences, and exploration of emerging technologies like AI to optimize system performance. Additionally, scalability and adaptability to accommodate a growing user base and evolving technological landscapes should be considered for sustained success. Continuous refinement and innovation will play a pivotal role in ensuring FoodieBookin remains at the forefront of the online food ordering domain, providing a delightful experience for users and administrators alike.

4. REFERENCES

I referred to various online resources to enhance my understanding and implementation of key functionalities. The list of references includes tutorials, documentation, and guides related to UI design using Flutter. Notable websites that provided valuable insights and guidance in implementing user registration, login/logout, food menu display, cart management (add to cart and remove from cart), payment processing, and confirmation receipt modules are as follows:

1. Flutter Documentation:

The official Flutter documentation served as a comprehensive reference for UI design, widget implementation, and best practices in Flutter development.

2. Udemy - Flutter Course:

A Udemy course on Flutter provided practical examples and in-depth explanations, helping me grasp the nuances of building a responsive and aesthetically pleasing user interface.

3. Stack Overflow:

The Stack Overflow community was instrumental in troubleshooting specific issues and offering solutions related to Flutter development and UI design.

4. Medium - Flutter Community:

Articles and tutorials from the Flutter community on Medium offered valuable insights into advanced UI design concepts and Flutter development techniques.

5. GitHub Repositories:

Open-source projects on GitHub, especially those related to Flutter UI design and app development, provided real-world examples and code snippets that I could adapt for my project.

These references collectively played a crucial role in shaping the project's user interface and functionality, ensuring a well-rounded and efficient implementation of features in the FoodieBookin application.