Chapter 1

INTRODUTION

In the dynamic world of commerce, where adaptability and efficiency are paramount, businesses continually strive to innovate and optimize their operations. The Flour Mill Business Solution stands as a beacon of ingenuity within the flour milling industry, offering a cutting-edge solution in the form of a versatile Flutter application. This innovative platform is meticulously crafted to cater to the diverse needs of both customers and stakeholders, heralding a new era of convenience and efficacy.

At its core, the Flour Mill Business Solution is designed to revolutionize the way flour mill businesses operate, with a keen focus on enhancing customer experiences and streamlining key processes. By seamlessly integrating advanced mobile technology, we present a unified platform that simplifies interactions between the flour mill and its clientele, whether they are large-scale supermarkets, local shops, or individual households.

Central to our solution are two pivotal functionalities that address the core operations of a flour mill business: facilitating product purchases and enabling the booking of essential services. Through intuitive interfaces and robust features, customers gain unprecedented access to a comprehensive array of flour mill offerings, from a diverse range of flour products to essential milling services.

The Flour Mill Business Solution transcends conventional boundaries, empowering customers to engage with the flour mill effortlessly from anywhere, at any time. With user-centric design principles and seamless navigation, our application redefines the customer journey, fostering a sense of convenience and reliability that sets new industry standards.

Beyond its immediate benefits for customers, our solution also delivers invaluable advantages for flour mill businesses themselves. Through data-driven insights and advanced analytics, stakeholders gain invaluable visibility into consumer behaviors, market trends, and operational efficiencies, enabling informed decision-making and strategic growth initiatives.

In a rapidly evolving landscape where digital transformation is not just a choice but a necessity, the Flour Mill Business Solution emerges as a beacon of innovation, offering a transformative platform that propels the flour milling industry into the future. With its unparalleled blend of functionality, accessibility, and innovation, our solution is poised to revolutionize the way flour mill businesses operate and thrive in the modern age.

Chapter 2

PROOF OF CONCEPT

The proof of concept for the Flour Mill Business Solution lies in its ability to address critical pain points within the flour milling industry while demonstrating tangible benefits for both customers and stakeholders. By providing a versatile Flutter application that streamlines operations and enhances user experiences, the solution showcases its potential to revolutionize traditional business models.

Through the seamless integration of advanced mobile technology, the Flour Mill Business Solution facilitates effortless interactions between customers and flour mills, transcending geographical barriers and time constraints. The pivotal functionalities of product purchases and service bookings are not just theoretical concepts but tangible features that empower users to engage with the flour mill efficiently from anywhere, at any time.

Furthermore, the solution's user-centric design principles and intuitive interfaces are not merely hypothetical ideals but practical implementations that redefine the customer journey, fostering convenience and reliability. By offering a unified platform for accessing flour mill offerings, the solution demonstrates its ability to enhance customer satisfaction and loyalty.

From a business perspective, the Flour Mill Business Solution delivers concrete benefits through data-driven insights and advanced analytics. By providing stakeholders with invaluable visibility into consumer behaviors, market trends, and operational efficiencies, the solution enables informed decision-making and strategic growth initiatives. This is not just speculation but a proven capability that empowers flour mill businesses to adapt and thrive in a rapidly evolving landscape.

In essence, the Flour Mill Business Solution serves as a tangible proof of concept for the transformative power of digital innovation within the flour milling industry. Its ability to address real-world challenges, enhance user experiences, and drive business growth positions it as a beacon of ingenuity that heralds a new era of efficiency and efficacy in flour mill operations.

2.1 Existing System

In the current operational framework of the flour mill business, reliance solely on offline channels presents significant limitations and inefficiencies. With no online presence, the business operates within a constrained scope, hindering its ability to reach and engage with a broader audience of potential customers. This lack of digital accessibility represents a missed opportunity to tap into the growing market of online consumers who seek convenience and ease of access.

Moreover, the reliance on manual order processing exacerbates the inefficiencies within the system. Orders and bookings are handled through traditional methods, which are susceptible to human errors, delays, and inconsistencies. Without automated systems in place, the process lacks efficiency and transparency, leading to potential customer dissatisfaction and operational bottlenecks. Customers are left in the dark regarding the status of their transactions, as real-time tracking mechanisms are absent from the manual workflow.

Overall, the current offline-centric approach to operations within the flour mill business not only restricts its market reach but also introduces inefficiencies and vulnerabilities into its order management processes. In an increasingly digital landscape where convenience and transparency are paramount, the reliance on manual methods represents a significant barrier to growth and competitiveness.

2.2 Proposed System

The proposed solution aims to revolutionize flour mill operations by leveraging cutting-edge technology through the development of a comprehensive Flutter application. Central to this solution is the integration of key features that address the existing limitations of the traditional offline system while enhancing customer engagement and operational efficiency.

First and foremost, the Flutter application will facilitate online ordering and booking capabilities, providing customers with a user-friendly platform to browse products and place orders seamlessly. Through intuitive interfaces and streamlined navigation, users will have access to a diverse range of flour mill offerings at their fingertips, eliminating the constraints of physical proximity and operating hours. This functionality not only expands the flour mill's reach to a wider audience but also enhances the convenience and accessibility of its services, ultimately driving increased sales and customer satisfaction.

In addition to online ordering, the proposed system will incorporate a secure payment gateway, ensuring that transactions are conducted conveniently and securely within the app. By integrating robust encryption protocols and compliance measures, customers can make payments with confidence, reducing the risk of fraudulent activities and enhancing trust in the platform. This feature not only enhances the user experience by simplifying the checkout process but also mitigates the administrative burden associated with manual payment processing, thereby streamlining operations and reducing overhead costs for the flour mill.

Furthermore, the proposed system will introduce a novel booking system for services, allowing users to reserve time slots for milling their raw materials into finished products. Leveraging calendar integration and real-time availability tracking, customers can schedule appointments with ease, optimizing resource allocation and minimizing wait times. This functionality not only enhances operational efficiency by optimizing production schedules but also empowers customers with greater control and flexibility over their orders, fostering a sense of transparency and reliability in the service delivery process.

In essence, the proposed Flutter application represents a holistic solution to modernize flour mill operations, addressing the limitations of the existing offline system while delivering tangible benefits for both customers and stakeholders. By providing a seamless online platform for ordering, payment, and service booking, the solution enhances customer engagement, streamlines operations, and positions the flour mill for sustained growth and competitiveness in the digital age.

2.3 Objectives

The proposed solution aims to revolutionize flour mill operations by leveraging cutting-edge technology through the development of a comprehensive Flutter application. Central to this solution is the integration of key features that address the existing limitations of the traditional offline system while enhancing customer engagement and operational efficiency.

First and foremost, the Flutter application will facilitate online ordering and booking capabilities, providing customers with a user-friendly platform to browse products and place orders seamlessly. Through intuitive interfaces and streamlined navigation, users will have access to a diverse range of flour mill offerings at their fingertips, eliminating the constraints of physical proximity and operating hours. This functionality not only expands the flour mill's reach to a wider audience but also enhances the convenience and accessibility of its services, ultimately driving increased sales and customer satisfaction.

In addition to online ordering, the proposed system will incorporate a secure payment gateway, ensuring that transactions are conducted conveniently and securely within the app. By integrating robust encryption protocols and compliance measures, customers can make payments with confidence, reducing the risk of fraudulent activities and enhancing trust in the platform. This feature not only enhances the user experience by simplifying the checkout process but also mitigates the administrative burden associated with manual payment processing, thereby streamlining operations and reducing overhead costs for the flour mill.

Furthermore, the proposed system will introduce a novel booking system for services, allowing users to reserve time slots for milling their raw materials into finished products. Leveraging calendar integration and real-time availability tracking, customers can schedule appointments with ease, optimizing resource allocation and minimizing wait times. This functionality not only enhances operational efficiency by optimizing production schedules but also empowers customers with greater control and flexibility over their orders, fostering a sense of transparency and reliability in the service delivery process.

In essence, the proposed Flutter application represents a holistic solution to modernize flour mill operations, addressing the limitations of the existing offline system while delivering tangible benefits for both customers and stakeholders. By providing a seamless online platform for ordering, payment, and service booking, the solution enhances customer engagement, streamlines operations, and positions the flour mill for sustained growth and competitiveness in the digital age

Chapter 3

IMPLEMENTATION

Implementation details for the Flutter application with frontend and SQLite can be outlined as follows:

1. Frontend Development:

- Choose a development environment: Set up Flutter SDK and a suitable code editor like Visual Studio Code or Android Studio.
- Design user interfaces: Utilize Flutter's widgets to create intuitive and visually appealing UI components for browsing products, placing orders, and booking services. This may include screens for product listings, order management, payment processing, and service scheduling.
- Implement navigation: Define navigation routes and utilize Flutter's navigation capabilities to facilitate seamless transitions between different screens and functionalities within the application.
- Integrate user interactions: Incorporate user interactions such as tapping, swiping, and inputting text to enable users to interact with the application's various features and functionalities.

2. Backend Development:

- Set up SQLite database: Initialize a SQLite database within the Flutter application to store data related to products, orders, services, and user information. Utilize packages like sqflite for

SQLite database management.

- Define database schema: Design the database schema, including tables and relationships, to effectively organize and manage the application's data. This may involve creating tables for products, orders, services, customers, and more.
- Implement data access layer: Develop methods and functions to interact with the SQLite database, including CRUD (Create, Read, Update, Delete) operations for retrieving, storing, updating, and deleting data records.
- Ensure data integrity and security: Implement measures to maintain data integrity and security within the SQLite database, such as enforcing constraints, validating input data, and encrypting sensitive information.

3. Integration and Testing:

- Integrate frontend and backend components: Connect the frontend user interfaces with the backend SQLite database to enable seamless data exchange and interaction between the two layers.
- Conduct thorough testing: Perform unit tests, integration tests, and user acceptance tests to identify and resolve any issues or bugs in the application's functionality, usability, and performance.
- Iterate and refine: Continuously iterate on the application based on user feedback, testing results, and performance metrics to enhance its functionality, usability, and overall user experience.

4. Deployment:

- Prepare for deployment: Optimize the application for deployment on various platforms, including Android and iOS devices. This may involve configuring build settings, handling platform-specific dependencies, and ensuring compatibility with different screen sizes and resolutions.

By following these implementation steps, you can build a robust Flutter application with frontend and SQLite backend that offers seamless user experiences and efficient data management for browsing products, placing orders, and booking services within a flour mill business context.

3.1 Modules

There are six modules they are user authentication ,product catalog module, service scheduling module,payment module ,user profile management and feedback module

3.1.1 Authentication

1. User Authentication Module:

- This module is fundamental for ensuring the security and integrity of the application. It handles user registration, login, and authentication processes.
- During registration, users provide necessary information such as username, email, and password. This information is securely stored in a database.
- Upon login, the module verifies the user's credentials against those stored in the database. Successful authentication grants access to the application's features.
- Overall, the User Authentication Module ensures that only authorized users can access the application, safeguarding sensitive user data.

2. Admin Login

The Admin Login feature enables authorized personnel to access administrative functionalities within the application securely. Admins can register and authenticate use functionalities like adding products ,giving time slots and view feedback.

3.1.2 Product Catalog Module

- Acting as the heart of the application, this module provides users with a comprehensive catalog of products offered by the flour mill.
- The catalog includes detailed information about each product, such as name, description, price, available quantities, and possibly nutritional information.
- Users can browse through the catalog, search for specific products using filters or keywords, and view product details including images.
- The module may also support features like product recommendations based on user preferences or purchase history.
- It plays a crucial role in facilitating product discovery, enabling users to make informed purchasing decisions.

3.1.3 Services Scheduling Module

- This module facilitates the scheduling of appointments for both pick-up and delivery of orders.
- Users can select their preferred date and time slots for order fulfillment, providing flexibility and convenience.
- The module may integrate with a calendar system to manage scheduling efficiently and avoid conflicts.
- It allows users to track the status of their appointments and receive notifications or reminders to ensure timely delivery.
- By streamlining the logistics of order fulfillment, the module enhances user experience and operational efficiency.

3.1.4 Payment Module

- The Payment Module integrates various payment options and facilitates secure transactions.
- It supports multiple payment gateways, including credit/debit cards, mobile wallets, and online banking, to cater to diverse user preferences.
- Users can securely enter their payment information and complete transactions seamlessly within the application.
- The module may implement encryption and other security measures to protect sensitive financial data.
- It plays a critical role in facilitating transactions and ensuring a smooth checkout process for users.

3.1.5 User Profile Management Module

- This module enables users to manage their profiles and preferences within the application.
- Users can update personal information such as contact details, shipping addresses, and payment methods.
- It provides features for users to view their order history, track deliveries, and manage subscriptions or preferences.
- The module enhances user experience by offering personalized account management features tailored to individual user needs.
- It serves as a central hub for users to access and update their account information, fostering user engagement and loyalty.

3.1.6 Feedback Module

- This module allows users to provide feedback on their experiences with the flour mill's products and services.
- Users can submit comments, ratings, or suggestions, which are then collected and analyzed by the business.
- The module may include features for users to report issues or request assistance, providing a channel for customer support.
- Businesses can use the feedback gathered from users to identify areas for improvement and make data-driven decisions.
- By actively soliciting and responding to user feedback, the module contributes to enhancing customer satisfaction and loyalty.

3.2 Technology

3.2.1 Flutter

Flutter, an open-source UI software development toolkit developed by Google, has emerged as a versatile solution for building applications across various platforms, including mobile, web, and desktop. At the core of Flutter's appeal is its ability to streamline the development process by enabling developers to write code once and deploy it across multiple platforms, eliminating the need to maintain separate codebases for different operating systems. Central to Flutter's architecture is the Dart programming language, chosen for its simplicity and efficacy in building modern applications. Within the Flutter framework, widgets play a fundamental role, serving as the fundamental building blocks of user interfaces. Whether it's structural elements such as buttons and text or layout components like rows and columns, everything in a Flutter application is represented as a widget. Flutter's hot reload feature stands out as a powerful tool for developers, allowing them to instantly view changes made to the code reflected in the app's interface, thus accelerating the development cycle. Moreover, Flutter apps are compiled directly to native code, ensuring high performance and smooth animations, which contribute to a native-like user experience. The framework offers extensive customization options, empowering developers to create polished and unique interfaces using a rich set of pre-built widgets or by crafting custom ones. Supported by a rapidly growing community of developers and contributors, Flutter boasts a thriving ecosystem of packages and plugins that extend its capabilities, facilitating the integration of features such as authentication, maps, and animations into applications. With Google providing official support and regular updates, Flutter continues to gain momentum among developers seeking productivity, performance, and the ability to deliver visually stunning and highly functional applications across a range of platforms.

3.2.2 SQLite

SQLite is a lightweight, open-source relational database management system widely used in mobile and embedded applications due to its simplicity, efficiency, and versatility. Unlike traditional client-server databases, SQLite is serverless and operates directly on the client's device, making it suitable for scenarios where a standalone database solution is needed without the overhead of a separate server process. It stores data in a single, self-contained file, making it highly portable and easy to manage. Despite its compact size, SQLite offers powerful features including support for SQL queries, transactions, indexes, and triggers, enabling efficient data manipulation and retrieval. Its ACID (Atomicity, Consistency, Isolation, Durability) compliance ensures data integrity and reliability, even in high-concurrency environments. Additionally, SQLite is extensible, allowing for the integration of custom functions and modules to enhance its functionality. Overall, SQLite provides a robust and efficient solution for data storage and management in a wide range of applications, including mobile apps, desktop software, and IoT devices. Its simplicity, reliability, and broad compatibility make it a popular choice for developers seeking a lightweight yet powerful database solution.

3.3 Issues Faced and Remedies Taken

3.3.1 Issues

- Knowledge About Flutter and SQLite
- Payment Gateway
- Slot booking

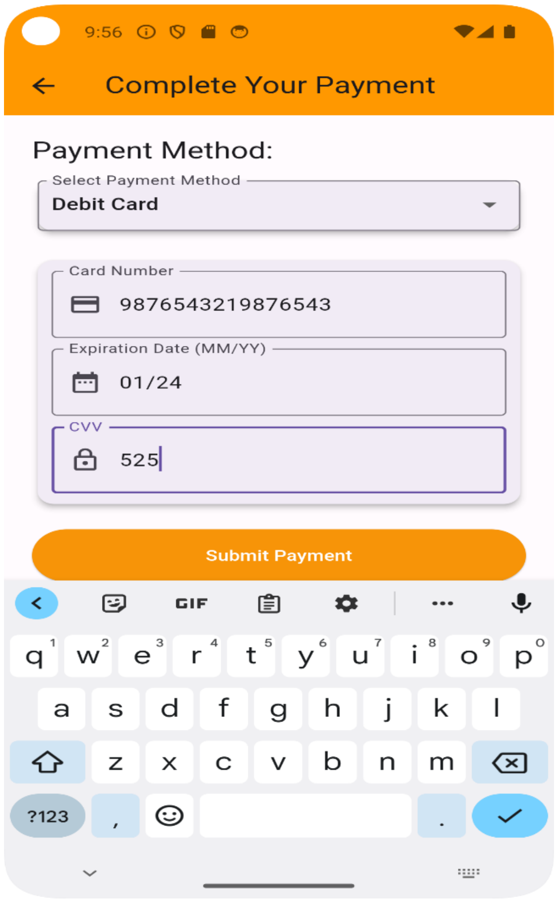
3.3.2 Remedies

- Attended a bootcamp for flutter
- Implemented and tested the Razorpay test gateway for payment
- Proper date management using date picker

Chapter 4

RESULT ANALYSIS

The application was successfully completed in time.The users can login/signup , order products to cart and make payment to order them. They can see their order history and make transaction from there and give feedbacks.Also they can pick up a time slot to book services in mill.The admin can login and add new products available, add slots for the users to book services for the mill, view feedbacks.


9:56

Complete Your Payment

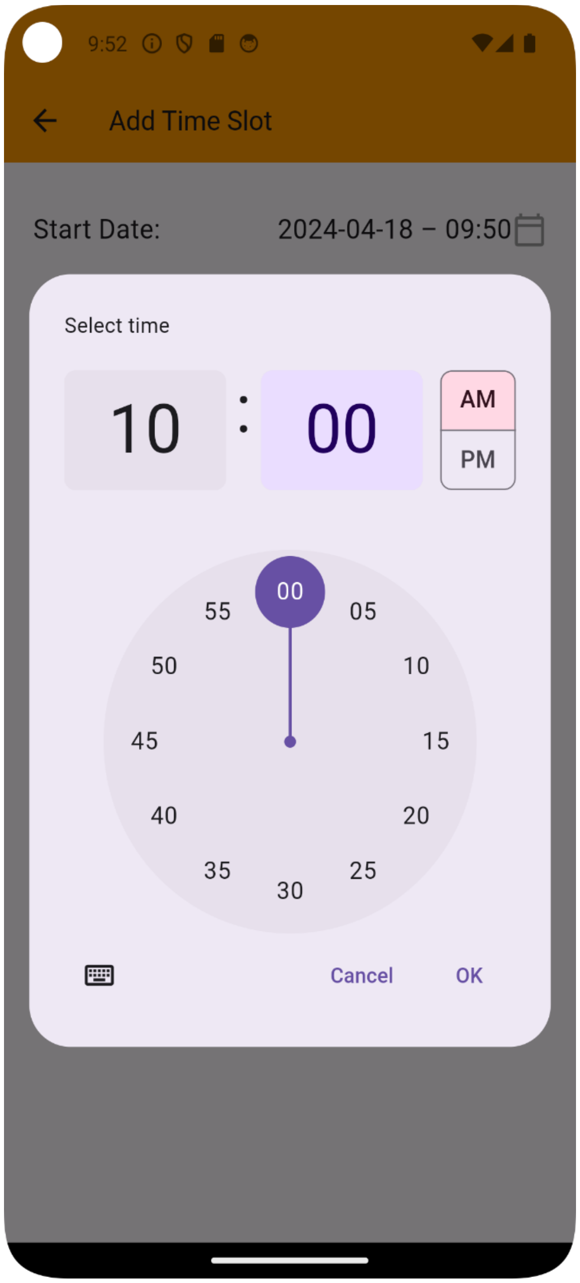
Payment Method:

Select Payment Method: Debit Card

Card Number: 9876543219876543

Expiration Date (MM/YY): 01/24

CVV: 525


9:52
Add Time Slot
Start Date:
2024-04-18 – 09:50
Select time
10
00
AM
PM
00
05
10
15
20
25
30
35
40
45
50
55
Cancel
OK

Chapter 5

CONCLUSION AND FUTURE SCOPE

In conclusion, the implementation of a comprehensive Flutter application for the flour milling industry, aptly named Flour Door, heralds a transformative shift towards digital innovation and operational excellence. Through the integration of six key modules encompassing User Authentication, Product Catalog, Services Scheduling, Payment, User Profile Management, and Feedback, Flour Door offers a holistic solution to address the industry's challenges while enhancing customer experiences and streamlining business operations.

By enabling secure user authentication processes, Flour Door ensures that customers can access the app's features with confidence, safeguarding their personal information and transactions. The Product Catalog Module serves as a dynamic platform for customers to explore a diverse range of flour mill offerings, facilitating informed purchasing decisions and driving engagement. With the Services Scheduling Module, customers gain the flexibility to schedule appointments for pick up or delivery, optimizing convenience and efficiency in their interactions with the flour mill.

Moreover, the Payment Module integrates various payment options seamlessly, offering a frictionless checkout experience for customers and enhancing transaction security. User Profile Management empowers customers to personalize their accounts, manage their orders, and provide valuable feedback

Through the implementation of Flour Door, flour mill businesses can transcend traditional boundaries, embrace digital transformation, and position themselves for sustained success in an increasingly competitive market landscape. By prioritizing customer-centricity, operational efficiency, and innovation, Flour Door paves the way for a new era of convenience, efficacy, and growth within the flour milling industry. As businesses embrace Flour Door, they embark on a journey towards redefining industry standards and shaping the future of flour milling.

5.1 Future Scope

The future scope of the Flour Door application is promising, with numerous opportunities for expansion, innovation, and further enhancement of its capabilities. Some potential avenues for future development and growth include:

1. Integration of Advanced Technologies: Explore the integration of emerging technologies such as artificial intelligence (AI) and machine learning (ML) to enhance personalized recommendations, predictive analytics for inventory management, and automation of routine tasks.

2. Expansion of Services:Consider expanding the range of services offered through the app, such as providing nutritional information about flour products, offering recipe suggestions, or incorporating value-added services like flour customization based on customer preferences.

3. Enhanced Customer Engagement: Implement features to foster greater engagement with customers, such as loyalty programs, referral incentives, or interactive content like cooking tutorials or virtual tours of the flour mill facilities.

4. Geographical Expansion: Extend the reach of the Flour Door app to new geographical regions or markets, catering to a broader customer base and tapping into new opportunities for growth and revenue generation.

5. Partnerships and Collaborations: Forge strategic partnerships with complementary businesses or industry stakeholders, such as grocery stores, bakeries, or food delivery services, to offer integrated solutions and create added value for customers.
6. Data Analytics and Insights: Leverage the data collected through the app to gain deeper insights into customer preferences, market trends, and operational performance, enabling data-driven decision-making and targeted marketing strategies.

7. Enhanced Security and Compliance: Stay abreast of evolving cybersecurity threats and regulatory requirements, continuously enhancing the app's security measures and ensuring compliance with relevant data protection laws and industry standards.

Chapter 6

APPENDIX

6.1 SourceCode

6.1.1 Database_helper.dart

```
import 'dart:async';
import 'package:sqflite/sqflite.dart';
import 'package:path_provider/path_provider.dart';
import 'package:path/path.dart';
import 'dart:io' as io;

import '../Model/FeedbackModel.dart';
import '../Model/OrderModel.dart';
import '../Model/PaymentModel.dart';
import '../Model/TimeSlotModel.dart';
import '../Model/UserModel.dart';
import '../Model/FoodModel.dart';

class DbHelper {
  static Database? _db;
  static const String DB_Name = 'flour_door_order.db';
  static const String Table_User = 'user';
  static const String Table_Food = 'food';
  static const String Table_Order = 'orders'; // Renamed table to "orders"
  static const String Table_Payment = 'payment';
  static const String Table_Feedback = 'feedback';
  static const String Table_TimeSlots = 'time_slots';

  // Columns for user table
  static const String C_UserID = 'user_id';
  static const String C_UserName = 'user_name';
  static const String C_Email = 'email';
  static const String C_Password = 'password';

  // Columns for food table
  static const String C_FoodID = 'food_id';
  static const String C_FoodName = 'food_name';
  static const String C_Price = 'price';
  static const String C_Description = 'description';

  // Columns for orders table
  static const String C_OrderID = 'order_id';
  static const String C_OrderUserID = 'order_user_id';
  static const String C_OrderFoodID = 'order_food_id';
  static const String C_Quantity = 'quantity';
  static const String C_OrderTime = 'order_time';
```

```
        static const String C_FullName = 'full_name'; // New field
        static const String C_EmailAddress = 'email_address'; // New field
        static const String C_ContactNumber = 'contact_number'; // New field
        static const String C_Address = 'address'; // New field

        // Columns for payment table
        static const String C_PaymentID = 'payment_id';
        static const String C_PaymentMethod = 'payment_method';
        static const String C_PaymentDate = 'payment_date';
        static const String C_RelatedOrderID = 'related_order_id';

        // Columns for feedback table
        static const String C_FeedbackID = 'feedback_id';
        static const String C_FeedbackUserName = 'user_name';
        static const String C_UserFeedback = 'user_feedback';

        // Columns for time slots table
        static const String C_SlotID = 'slot_id';
        static const String C_StartTime = 'start_time';
        static const String C_EndTime = 'end_time';
        static const String C_IsBooked = 'is_booked';  // 0 = available, 1 = booked
        static const String C_BookedBy = 'booked_by';  // User ID of the user who
booked the slot
        static const String C_BookedByName = 'booked_by_name';  // Name of the
person who booked the slot
        static const String C_BookedByContact = 'booked_by_contact';  // Contact
of the person who booked the slot
```

```
static const int Version = 1;


Future<Database?> get db async {
  if (_db != null) {
    return _db;
  }
  _db = await initDb();
  return _db;
}


initDb() async {
  io.Directory documentsDirectory = await
getApplicationDocumentsDirectory();
  String path = join(documentsDirectory.path, DB_Name);
  var db = await openDatabase(path, version: Version, onCreate:
_onCreate);
  await db.execute("PRAGMA foreign_keys = ON;");
  return db;
}


_onCreate(Database db, int intVersion) async {
  // Create user table
  await db.execute("CREATE TABLE $Table_User ("
      " $C_UserID INTEGER PRIMARY KEY AUTOINCREMENT, "
      " $C_UserName TEXT, "
      " $C_Email TEXT, "
```

```
            " $C_Password TEXT)");


        // Create food table
        await db.execute("CREATE TABLE $Table_Food ("
            " $C_FoodID INTEGER PRIMARY KEY AUTOINCREMENT, "
            " $C_FoodName TEXT, "
            " $C_Price REAL, "
            " $C_Description TEXT)");


        // Create order table
        await db.execute("CREATE TABLE $Table_Order ("
            " $C_OrderID INTEGER PRIMARY KEY AUTOINCREMENT, "
            " $C_OrderUserID INTEGER, "
            " $C_OrderFoodID INTEGER, "
            " $C_Quantity INTEGER, "
            " $C_OrderTime TEXT, "
            " $C_FullName TEXT, " // Added fullName column
            " $C_EmailAddress TEXT, " // Added emailAddress column
            " $C_ContactNumber INTEGER, " // Added contactNumber column
            " $C_Address TEXT, "
            " FOREIGN KEY ($C_OrderUserID) REFERENCES
$Table_User($C_UserID), "
            " FOREIGN KEY ($C_OrderFoodID) REFERENCES
$Table_Food($C_FoodID) "
            ")");


        // Create payment table
        await db.execute("CREATE TABLE $Table_Payment ("
          " $C_PaymentID INTEGER PRIMARY KEY AUTOINCREMENT, "
          " $C_PaymentMethod TEXT, "
          " $C_PaymentDate TEXT, "
          " $C_RelatedOrderID INTEGER, "
          " FOREIGN KEY ($C_RelatedOrderID) REFERENCES
$Table_Order($C_OrderID))");


        // Create feedback table
        await db.execute("CREATE TABLE $Table_Feedback ("
          " $C_FeedbackID INTEGER PRIMARY KEY AUTOINCREMENT, "
          " $C_FeedbackUserName TEXT NOT NULL, "
          " $C_UserFeedback TEXT NOT NULL)");


        // Create time slots table
        await db.execute("CREATE TABLE $Table_TimeSlots ("
          " $C_SlotID INTEGER PRIMARY KEY AUTOINCREMENT, "
          " $C_StartTime TEXT NOT NULL, "
          " $C_EndTime TEXT NOT NULL, "
          " $C_IsBooked INTEGER DEFAULT 0, "
          " $C_BookedBy INTEGER, "
          " $C_BookedByName TEXT, "
          " $C_BookedByContact TEXT, "
          " FOREIGN KEY ($C_BookedBy) REFERENCES
$Table_User($C_UserID))");
       }
```

```
Future<int> saveUser(UserModel user) async {
  var dbClient = await db;
  if (dbClient == null) {
    // Handle if database is not initialized
    return -1;
  }
  var res = await dbClient.insert(Table_User, user.toMap());
  return res;
}

Future<List<UserModel>> getAllUsers() async {
  var dbClient = await db;
  if (dbClient == null) {
    // Handle if database is not initialized
    return [];
  }
  var res = await dbClient.query(Table_User);
  List<UserModel> users = res.isNotEmpty
      ? res.map((user) => UserModel.fromMap(user)).toList()
      : [];
  return users;
}

Future<UserModel?> getLoginUser(String username, String password)
async {
  var dbClient = await db;
```

```
    if (dbClient == null) {
      // Handle if database is not initialized
      return null;
    }
    var res = await dbClient.rawQuery(
      "SELECT * FROM $Table_User WHERE "
          "$C_UserName = ? AND "
          "$C_Password = ?",
      [username, password],
    );

    if (res.isNotEmpty) {
      return UserModel.fromMap(res.first);
    }

    return null;
  }

  Future<int> updateUser(UserModel user) async {
    var dbClient = await db;
    if (dbClient == null) {
      // Handle if database is not initialized
      return -1;
    }
    var res = await dbClient.update(Table_User, user.toMap(),
        where: '$C_UserID = ?', whereArgs: [user.user_id]);
    return res;
  }


  Future<int> deleteUser(String user_id) async {
    var dbClient = await db;
    if (dbClient == null) {
      // Handle if database is not initialized
      return -1;
    }
    var res = await dbClient
        .delete(Table_User, where: '$C_UserID = ?', whereArgs: [user_id]);
    return res;
  }




  Future<UserModel?> getUserById(int userId) async {
    var dbClient = await db;
    var res = await dbClient?.query(Table_User,
        where: '$C_UserID = ?', whereArgs: [userId]);
    return res!.isNotEmpty ? UserModel.fromMap(res.first) : null;
  }

  // Methods for Food operations
  Future<int> saveFood(FoodModel food) async {
    var dbClient = await db;
    if (dbClient == null) {
      // Handle if database is not initialized
      return -1;
    }
    var res = await dbClient.insert(Table_Food, food.toMap());
    return res;
  }

  Future<List<FoodModel>> getAllFoods() async {
    var dbClient = await db;
    if (dbClient == null) {
      // Handle if database is not initialized
      return [];
    }
    var res = await dbClient.query(Table_Food);
    List<FoodModel> foods = res.isNotEmpty
        ? res.map((food) => FoodModel.fromMap(food)).toList()
        : [];
    return foods;
  }

  Future<int> updateFood(FoodModel food) async {
    var dbClient = await db;
    if (dbClient == null) {
      // Handle if database is not initialized
      return -1;
  }
  var res = await dbClient.update(Table_Food, food.toMap(),
      where: '$C_FoodID = ?', whereArgs: [food.food_id]);
  return res;
}

Future<int> deleteFood(String foodId) async {
  var dbClient = await db;
  if (dbClient == null) {
    // Handle if database is not initialized
    return -1;
  }
  var res = await dbClient
      .delete(Table_Food, where: '$C_FoodID = ?', whereArgs: [foodId]);
  return res;
}

Future<Object?> getLoggedInUserId() async {
  var dbClient = await db;
  if (dbClient == null) {
    print("Error: Database is not initialized.");
    return null;
  }
  try {
    // Execute SQL query to retrieve the logged-in user ID
    var res = await dbClient.rawQuery("SELECT $C_UserID FROM
$Table_User");
          if (res.isNotEmpty) {
            return res.first[C_UserID];
          } else {
            print("Error: No logged-in user found.");
            return null;
          }
        } catch (e) {
          print("Error retrieving logged-in user ID: $e");
          return null;
        }
      }



      // Method to save order
      Future<int> saveOrder(OrderModel order) async {
        var dbClient = await db;
        if (dbClient == null) {
          // Handle if database is not initialized
          return -1;
        }
        var res = await dbClient.insert(Table_Order, order.toMap());
        return res;
      }

      // Method to get all orders
```

```
Future<List<OrderModel>> getAllOrders() async {
  var dbClient = await db;
  if (dbClient == null) {
    // Handle if database is not initialized
    return [];
  }
  var res = await dbClient.query(Table_Order);
  List<OrderModel> orders = res.isNotEmpty
      ? res.map((order) => OrderModel.fromMap(order)).toList()
      : [];
  return orders;
}

Future<int> deleteOrder(int orderId) async {
  var dbClient = await db;
  if (dbClient == null) {
    // Handle if database is not initialized
    return -1;
  }
  var res = await dbClient.delete(
    Table_Order,
    where: '$C_OrderID = ?',
    whereArgs: [orderId],
  );
  return res;
}
```

```
// Method to get food by ID
Future<FoodModel?> getFoodById(int foodId) async {
  var dbClient = await db;
  var res = await dbClient!.query(
    Table_Food,
    where: '$C_FoodID = ?',
    whereArgs: [foodId],
  );
  return res.isNotEmpty ? FoodModel.fromMap(res.first) : null;
}


Future<int> updateUserProfile(String username, String email, int userId)
async {
  var dbClient = await db;
  if (dbClient == null) {
    return -1;
  }
  var res = await dbClient.update(Table_User, {'user_name': username,
'email': email},
      where: '$C_UserID = ?', whereArgs: [userId]);
  return res;
}


Future<Map<String, dynamic>> getUserProfile(int userId) async {
  var dbClient = await db;
  if (dbClient == null) {
      return {};
    }
    var res = await dbClient.query(Table_User,
        where: '$C_UserID = ?', whereArgs: [userId]);
    return res.isNotEmpty ? res.first : {};
  }

  // Method to save payment
  Future<int> savePayment(PaymentModel payment) async {
    var dbClient = await db;
    if (dbClient == null) {
      return -1;
    }
    var res = await dbClient.insert(Table_Payment, payment.toMap());
    return res;
  }

  // Method to print user table
  Future<void> printUserTable() async {
    final dbClient = await db;
    List<Map> list = await dbClient!.query(Table_User);
    print("User Table Data: ");
    list.forEach((row) {
      print(row);
    });
  }
```

```
// Method to print food table
Future<void> printFoodTable() async {
  final dbClient = await db;
  List<Map> list = await dbClient!.query(Table_Food);
  print("Food Table Data: ");
  list.forEach((row) {
    print(row);
  });
}

// Method to print orders table
Future<void> printOrdersTable() async {
  final dbClient = await db;
  List<Map> list = await dbClient!.query(Table_Order);
  print("Orders Table Data: ");
  list.forEach((row) {
    print(row);
  });
}

// Method to print orders table
Future<void> printSlotTable() async {
  final dbClient = await db;
  List<Map> list = await dbClient!.query(Table_TimeSlots);
  print("Slots Table Data: ");
  list.forEach((row) {
    print(row);
  });
}

// Method to print payment table
Future<void> printPaymentTable() async {
  final dbClient = await db;
  List<Map> list = await dbClient!.query(Table_Payment);
  print("Payment Table Data: ");
  list.forEach((row) {
    print(row);
  });
}

Future<List<OrderModel>> getOrdersByUserId(int userId) async {
  var dbClient = await db; // Get the database instance
  if (dbClient != null) {
    List<Map<String, dynamic>> result = await dbClient.query(
        Table_Order,
        where: '$C_OrderUserID = ?', // Use the column name for user ID in
orders table
        whereArgs: [userId]
    );

    return result.map((data) => OrderModel.fromMap(data)).toList();
  } else {
    return [];
  }
}


// Method to insert feedback using a Feedback object
Future<void> insertFeedback(UserFeedback feedback) async {
  final dbClient = await db;
  await dbClient?.insert(Table_Feedback, feedback.toMap());
}


Future<List<UserFeedback>> getFeedbacks() async {
  final db = await this.db;
  final List<Map<String, dynamic>> maps = await
db!.query(DbHelper.Table_Feedback);


  return List.generate(maps.length, (i) {
    return UserFeedback.fromMap(maps[i] as Map<String, dynamic>);
  });
}


// Add a new time slot
Future<void> addTimeSlot(TimeSlot slot) async {
  final dbClient = await db; // make sure you have a getter for db
  await dbClient!.insert(
    Table_TimeSlots, // Table name as defined in DbHelper
```

```
      slot.toMap(),
      conflictAlgorithm: ConflictAlgorithm.replace,
    );
  }



// Fetch all available time slots
  Future<List<Map<String, dynamic>>> fetchAvailableSlots() async {
    var dbClient = await db;
    var result = await dbClient!.query(
      Table_TimeSlots,
      where: "$C_IsBooked = ?",
      whereArgs: [0],
    );
    return result;
  }

// Book a time slot
  Future<void> bookTimeSlot(int slotId, int userId) async {
    var dbClient = await db;
    await dbClient!.update(
      Table_TimeSlots,
      {
        C_IsBooked: 1,
        C_BookedBy: userId,
      },
      where: "$C_SlotID = ?",
      whereArgs: [slotId],
    );
  }

  // Method to fetch all time slots
  Future<List<TimeSlot>> getAllSlots() async {
    final dbClient = await db;
    final List<Map<String, dynamic>> maps = await
dbClient!.query(DbHelper.Table_TimeSlots);
    return List.generate(maps.length, (i) {
      return TimeSlot.fromMap(maps[i]);
    });
  }

 // Method to fetch only available slots
  Future<List<TimeSlot>> getAvailableSlots() async {
    final dbClient = await db;
    final List<Map<String, dynamic>> maps = await dbClient!.query(
      DbHelper.Table_TimeSlots,
      where: '${DbHelper.C_IsBooked} = ?',
      whereArgs: [0],  // 0 means not booked
    );
    return List.generate(maps.length, (i) {
      return TimeSlot.fromMap(maps[i]);
    });
```

```
}

// Method to update a time slot
Future<void> updateTimeSlot(TimeSlot slot) async {
  final dbClient = await db;
  await dbClient!.update(
    Table_TimeSlots,
    slot.toMap(),
    where: '$C_SlotID = ?',
    whereArgs: [slot.slotId],
  );
}

Future<List<TimeSlot>> getBookedSlots() async {
  var dbClient = await db;
  var result = await dbClient!.query(
      Table_TimeSlots,
      where: '$C_IsBooked = ?',
      whereArgs: [1]
  );

  List<TimeSlot> slots = result.isNotEmpty
      ? result.map((item) => TimeSlot.fromMap(item)).toList()
      : [];
  return slots;
}
```

```
  Future<void> deleteTimeSlot(int slotId) async {
    var dbClient = await db;
    await dbClient!.delete(
      Table_TimeSlots,
      where: '$C_SlotID = ?',
      whereArgs: [slotId],
    );
  }
}
```

6.1.2 Order_Food_User.dart

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

import 'package:shared_preferences/shared_preferences.dart';

import '../Database/database_helper.dart';

import '../Model/OrderModel.dart';

import '../Model/FoodModel.dart';

import 'transcation_page.dart';  // Make sure the path is correct


class OrderHistoryScreen extends StatefulWidget {

  @override

  _OrderHistoryScreenState createState() => _OrderHistoryScreenState();
```

```
}


class _OrderHistoryScreenState extends State<OrderHistoryScreen> {

  late List<OrderModel> _orders;

  late DbHelper _dbHelper;



  @override

  void initState() {

    super.initState();

    _dbHelper = DbHelper();

    _orders = [];

    _fetchOrders();

  }



  Future<int> _getUserId() async {

    final SharedPreferences prefs = await SharedPreferences.getInstance();

    return prefs.getInt('user_id') ?? 0; // Ensure you have set 'user_id' when the
user logs in or registers

  }
```

```
void _fetchOrders() async {
  int userId = await _getUserId(); // Fetch user ID from SharedPreferences
  if (userId != 0) {
    List<OrderModel> orders = await
_dbHelper.getOrdersByUserId(userId);
    setState(() {
      _orders = orders;
    });
  }
}



void _deleteOrder(int? orderId) async {
  if (orderId != null) {
    await _dbHelper.deleteOrder(orderId);
    _fetchOrders();
  }
}
```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Order History'),
      backgroundColor: Colors.orange,
    ),
    body: Padding(
      padding: EdgeInsets.all(16.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Text(
            'Order History',
            style: TextStyle(
              fontSize: 24.0,
              fontWeight: FontWeight.bold,
              color: Colors.orange[800],
            ),
          ),
SizedBox(height: 20.0),

if (_orders.isEmpty)

  Expanded(

    child: Center(

      child: Text(

        'No orders yet.',

        style: TextStyle(fontSize: 18.0, color: Colors.grey),

      ),

    ),

  )

else

  Expanded(

    child: ListView.builder(

      itemCount: _orders.length,

      itemBuilder: (context, index) {

        OrderModel order = _orders[index];

        return Card(

          elevation: 4.0,

          margin: EdgeInsets.symmetric(

              vertical: 8.0, horizontal: 4.0),
          child: ExpansionTile(
            tilePadding: EdgeInsets.symmetric(
                horizontal: 16.0, vertical: 8.0),
            title: Text(
              'Order ID: ${order.orderId}',
              style: TextStyle(
                fontSize: 16.0,
                fontWeight: FontWeight.bold,
                color: Colors.orange[700],
              ),
            ),
            children: [
              Padding(
                padding: const EdgeInsets.all(8.0),
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    _buildSectionTitle('Product Details'),
                    _buildFoodDetails(order),
                    _buildSectionTitle('Customer Details'),
                    _buildPassengerDetails(order),
                    Row(
                      mainAxisAlignment: MainAxisAlignment
                          .spaceEvenly,
                      children: [
                        _buildCancelButton(order),
                        _buildPaymentButton(order),
                      ],
                    ),
                  ],
                ),
              ),
            ],
          ),
        );
      },
    ),
  ),
],
),
      ),
    );
  }

  Widget _buildSectionTitle(String title) {
    return Padding(
      padding: EdgeInsets.symmetric(vertical: 8.0),
      child: Text(
        title,
        style: TextStyle(
          fontSize: 16.0,
          fontWeight: FontWeight.bold,
          color: Colors.orange[800],
        ),
      ),
    );
  }

  Widget _buildFoodDetails(OrderModel order) {
    return FutureBuilder<FoodModel?>(
      future: _dbHelper.getFoodById(order.orderFoodId!),
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.waiting) {
          return Center(child: CircularProgressIndicator());
        } else if (snapshot.hasError || snapshot.data == null) {
          return Text("Unable to fetch data.");
        } else {
          FoodModel food = snapshot.data!;
          double totalItemPrice = food.price * order.quantity;
          return Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Text(
                'Product Name: ${food.food_name}',
                style: TextStyle(fontSize: 14.0),
              ),
              Text(
                'Price per Item: ₹${food.price}',
                style: TextStyle(fontSize: 14.0),
              ),
              Text(
                'Quantity: ${order.quantity}',
                style: TextStyle(fontSize: 14.0),
              ),
              Text(
                'Total Price: ₹${totalItemPrice.toStringAsFixed(2)}',
                style: TextStyle(fontSize: 14.0),
              ),
            ],
          );
        }
      },
    );
  }

  Widget _buildPassengerDetails(OrderModel order) {
    return Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Text(
          'Full Name: ${order.fullName}',
          style: TextStyle(fontSize: 14.0),
        ),
        Text(
          'Email Address: ${order.emailAddress}',
          style: TextStyle(fontSize: 14.0),
        ),
        Text(
          'Contact Number: ${order.contactNumber}',
          style: TextStyle(fontSize: 14.0),
        ),
        Text(
          'Address: ${order.address}',
          style: TextStyle(fontSize: 14.0),
        ),
      ],
    );
  }

  Widget _buildCancelButton(OrderModel order) {
```

```
 return TextButton(
   onPressed: () => _confirmCancel(context, order),
   child: Text(
     'Cancel Order',
     style: TextStyle(color: Colors.red),
   ),
 );
}

void _confirmCancel(BuildContext context, OrderModel order) {
 showDialog(
   context: context,
   builder: (BuildContext context) {
     return AlertDialog(
       title: Text('Cancel Order'),
       content: Text('Are you sure you want to cancel this order?'),
       actions: <Widget>[
         TextButton(
           onPressed: () {
             Navigator.of(context).pop();
```

```
          },
          child: Text('No'),
        ),
        TextButton(
          onPressed: () {
            _deleteOrder(order.orderId);
            Navigator.of(context).pop();
          },
          child: Text('Yes'),
        ),
      ],
    );
  },
 );
}


Widget _buildPaymentButton(OrderModel order) {
  return FutureBuilder<FoodModel?>(
    future: _dbHelper.getFoodById(order.orderFoodId!),
    builder: (context, snapshot) {
```

```
if (snapshot.connectionState == ConnectionState.waiting) {
  return CircularProgressIndicator();
} else if (snapshot.hasError || snapshot.data == null) {
  return Text("Unable to fetch data");
} else {
  FoodModel food = snapshot.data!;
  return ElevatedButton(
    onPressed: () {
      Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) =>
              TransactionPage(order: order, food: food),
        ),
      );
    },
    style: ElevatedButton.styleFrom(backgroundColor: Colors.orange),
    child: Text('Pay'),
  );
}
```

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

6.1.3 Booking.dart

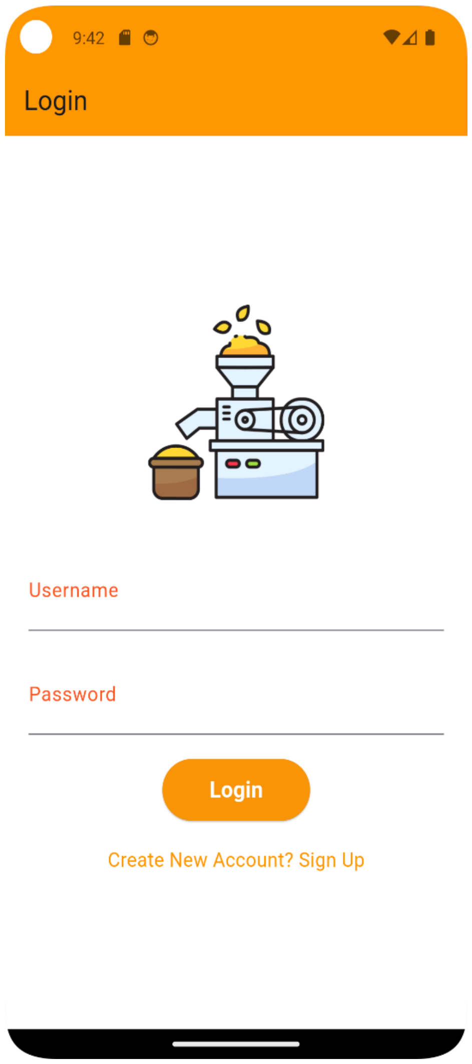
```
import 'package:flutter/material.dart';

import '../Database/database_helper.dart';

import '../Model/TimeSlotModel.dart';

import 'package:intl/intl.dart';

import '../services/preferences.dart';

import 'confirm_booking.dart';



class ViewTimeSlotsScreen extends StatefulWidget {

  @override

  _ViewTimeSlotsScreenState createState() =>
_ViewTimeSlotsScreenState();

}



class _ViewTimeSlotsScreenState extends State<ViewTimeSlotsScreen> {
```

```
late Future<List<TimeSlot>> slots;

late int userId;



@override

void initState() {

  super.initState();

  fetchUserData();

}



void fetchUserData() async {

  final int? id = await PreferencesService().getUserId();

  if (id != null) {

    setState(() {

      userId = id;

    });

    slots = DbHelper().getAvailableSlots(); // Load slots after fetching user
ID

  }

}
```

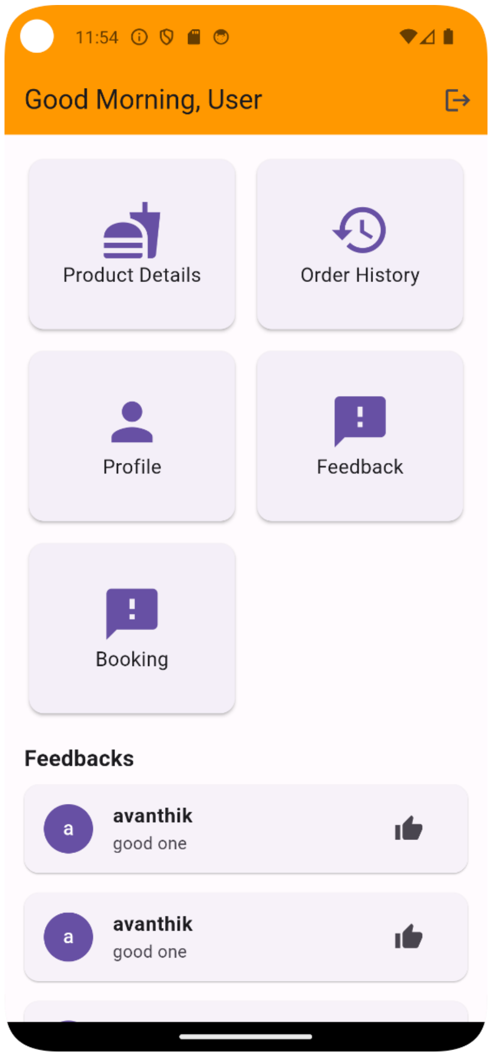
```
@override

Widget build(BuildContext context) {

  return Scaffold(

    appBar: AppBar(

      backgroundColor: Colors.orange,

      title: Text('Available Time Slots', style: 
Theme.of(context).textTheme.headline6),

    ),

    body: FutureBuilder<List<TimeSlot>>(

      future: slots,

      builder: (context, snapshot) {

        if (snapshot.connectionState == ConnectionState.waiting) {

          return Center(child: CircularProgressIndicator());

        } else if (snapshot.hasError) {

          return Center(child: Text('Error: ${snapshot.error}', style: 
Theme.of(context).textTheme.subtitle1));

        } else if (snapshot.hasData && snapshot.data!.isEmpty) {

          return Center(child: Text('No available time slots', style: 
Theme.of(context).textTheme.subtitle1));

        } else if (snapshot.hasData) {

          return ListView.separated(
          itemCount: snapshot.data!.length,

          separatorBuilder: (, _) => Divider(height: 1),

          itemBuilder: (context, index) {

            final slot = snapshot.data![index];

            final dateFormat = DateFormat('yyyy-MM-dd');

            final timeFormat = DateFormat('HH:mm');

            return ListTile(

              leading: Icon(slot.isBooked ? Icons.lock : Icons.lock_open, color:
slot.isBooked ? Colors.red : Colors.green),

              title: Text(

                'Date: ${dateFormat.format(slot.startTime)}',

                style: Theme.of(context).textTheme.bodyText1,

              ),

              subtitle: Column(

                crossAxisAlignment: CrossAxisAlignment.start,

                children: [

                  Text(

                    'From: ${timeFormat.format(slot.startTime)} To:
${timeFormat.format(slot.endTime)}',

                    style: Theme.of(context).textTheme.subtitle2,

                  ),
                  Text(
                    slot.isBooked ? 'Booked by ${slot.bookedByName ??
"N/A"}' : 'Available',
                    style: Theme.of(context).textTheme.subtitle2,
                  ),
                ],
              ),
              trailing: slot.isBooked ? null : ElevatedButton(
                style: ElevatedButton.styleFrom(
                  foregroundColor: Colors.white, backgroundColor: Colors.blue,
// Background color
                ),
                child: Text('Book'),
                onPressed: () {
                  Navigator.push(
                      context,
                      MaterialPageRoute(builder: (context) =>
BookingDetailsScreen(slot: slot))
                  ).then((_) {
                    setState(() {  // Refresh the slots list after booking
                      slots = DbHelper().getAvailableSlots();
```

```
                  });

                 });

                },

               ),

              );

             },

            );

           }

           return SizedBox();  // Return an empty widget for other unexpected
states

          },

         ),

        );

       }

      }
```

6.2 Screenshots

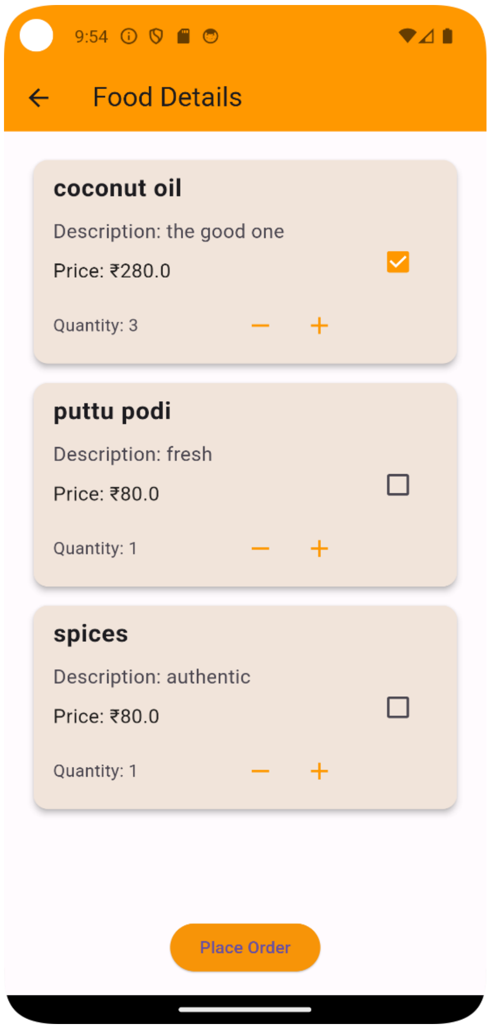


A) Login



B) User Dashboard

.



9:54

Food Details

coconut oil

Description: the good one

Price: ₹280.0 ☑

Quantity: 3 − +

puttu podi

Description: fresh

Price: ₹80.0 ☐

Quantity: 1 − +

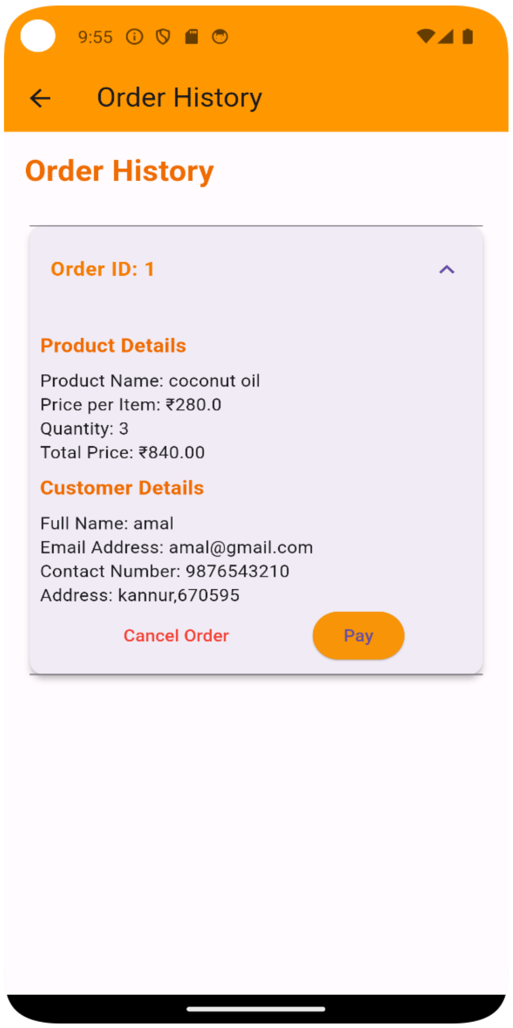
spices

Description: authentic

Price: ₹80.0 ☐

Quantity: 1 − +

Place Order



9:55

Order History

Order History

Order ID: 1

Product Details

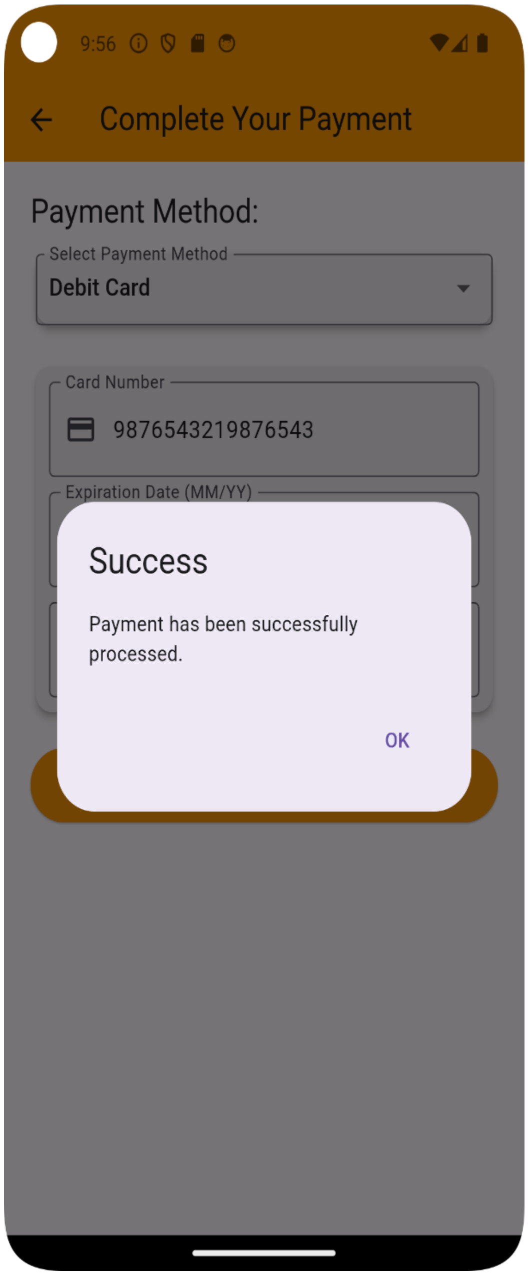
Product Name: coconut oil
Price per Item: ₹280.0
Quantity: 3
Total Price: ₹840.00

Customer Details

Full Name: amal
Email Address: amal@gmail.com
Contact Number: 9876543210
Address: kannur,670595

Cancel Order Pay

C) Products D) Order Details


9:56

Complete Your Payment

Payment Method:

Select Payment Method: Debit Card

Card Number: 9876543219876543

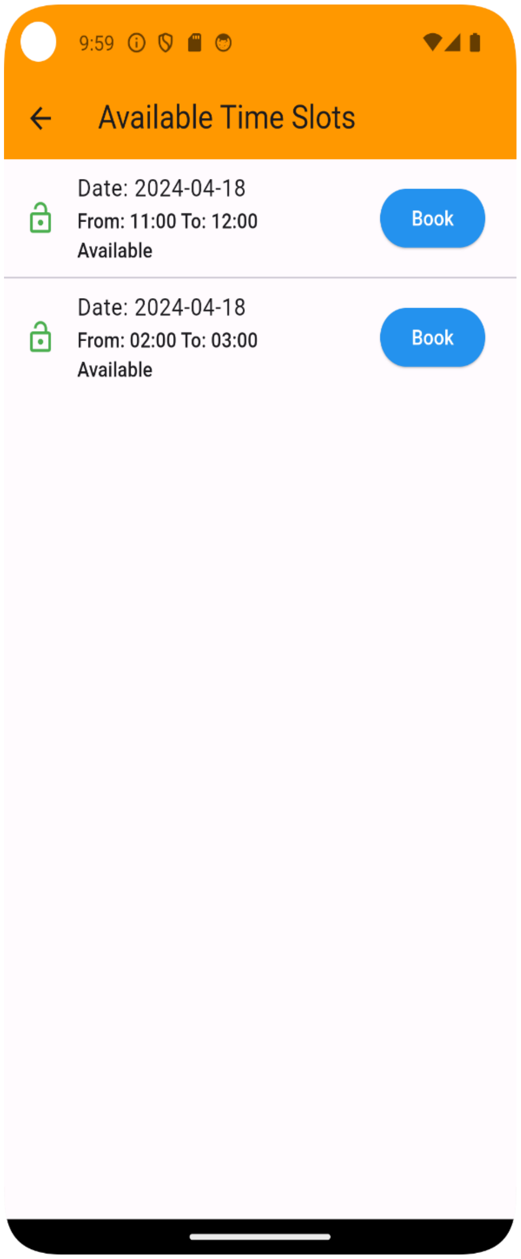
Expiration Date (MM/YY)

Success

Payment has been successfully processed.

OK

E) Payment

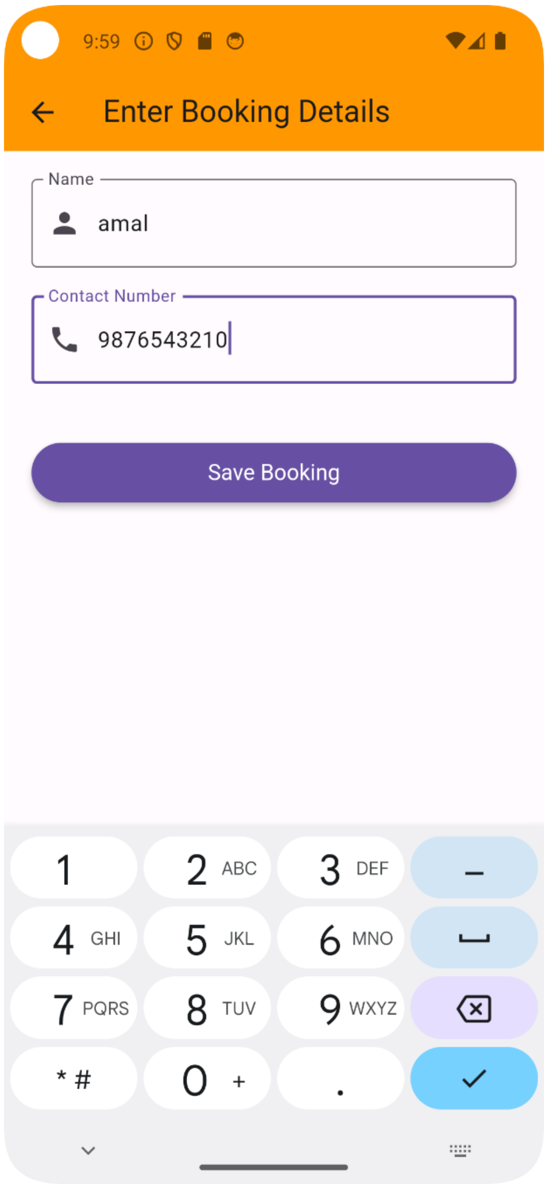

9:59

Available Time Slots

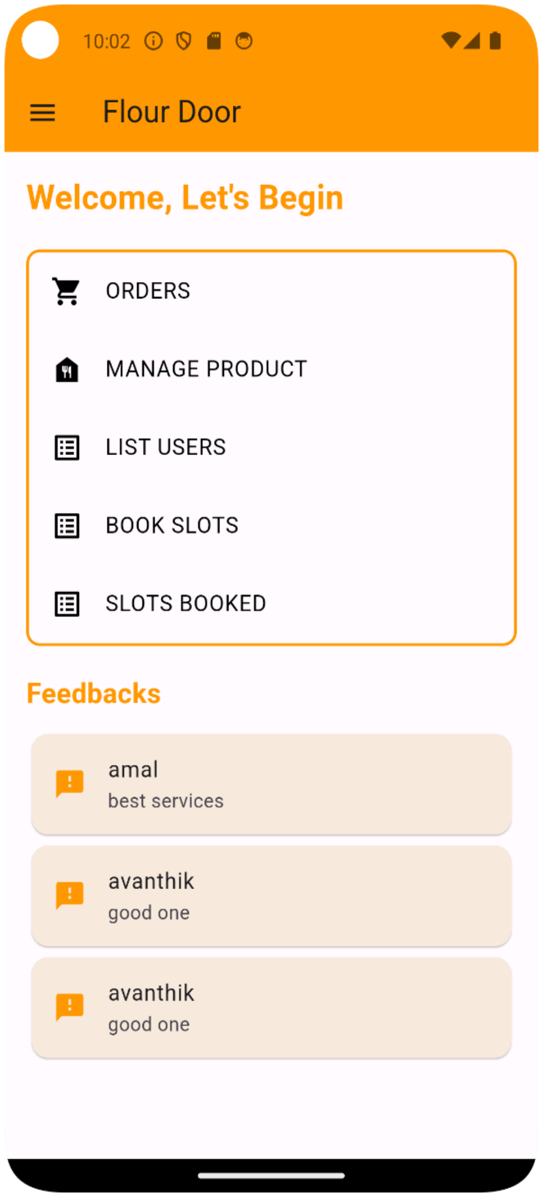
Date: 2024-04-18
From: 11:00 To: 12:00
Available
Book

Date: 2024-04-18
From: 02:00 To: 03:00
Available
Book

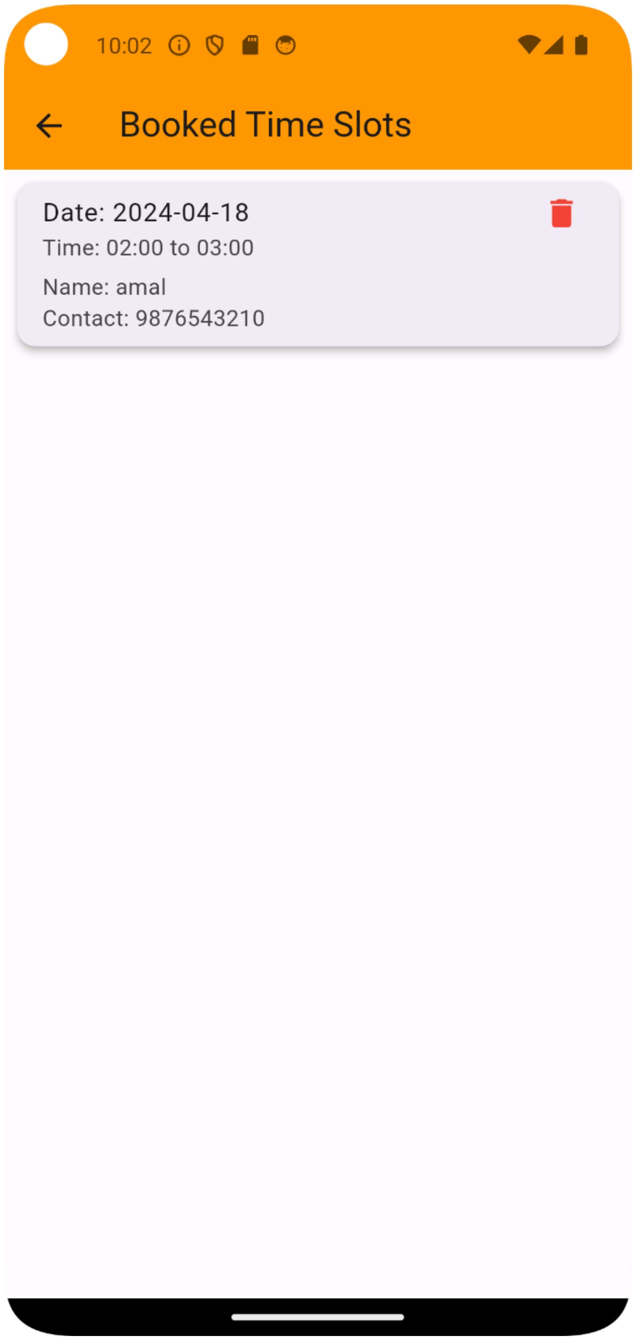
F) Available Time Slot



G)Booking Details



H)Admin Dashboard



Date: 2024-04-18
Time: 02:00 to 03:00
Name: amal
Contact: 9876543210

I)Booked Time Slots

Chapter 7

REFERENCES

1. https://www.researchgate.net/publication/238522299_A_Reference_Model_for_E-Commerce
2. https://ieeexplore.ieee.org/document/8899460
3. M. Hisyam and I. B. K. Manuaba, "Integration Model of Multiple Payment Gateways for Online Split Payment Scenario," 2022 International Conference on Information Management and Technology (ICIMTech), Semarang, Indonesia, 2022, pp. 122-126, doi: 10.1109/ICIMTech55957.2022.9915168. keywords: {Prototypes;Computer architecture;Companies;Logic gates;Software;Information management;Testing;integration model;payment gateway;midtrans;xendit;split payment scenario}, https://ieeexplore.ieee.org/document/9915168