**DineDeal - Mobile Application**



**By:**

**Syed Kamran Shah**

**35453**

**Safiullah Shahid**

**35857**

**Muhammad Ismail**

**35897**

**Supervised by:**

**Dr. Musharaf Ahmed**

**Faculty of Computing**

**Riphah International** **University, Islamabad**

**Year 2024**

**A Dissertation Submitted To**

**Faculty of Computing,**

**Riphah International University, Islamabad**

**As a Partial Fulfillment of the Requirement for the Award of**

**the Degree of**

**Bachelor of Science in Software Engineering**

# Faculty of Computing

# Riphah International University, Islamabad

Date: ??????

**Final Approval**

# Committee:

# 1

Dr. Musharaf Ahmad

(Supervisor) (Head of Department/chairman)

**Declaration**

We hereby declare that this document, titled “DineDeal,” neither as a whole nor in part has been copied from any other source. It is further declared that this project was carried out by the undersigned entirely based on our personal efforts under the proficient guidance of our supervisor Dr. Musharaf Ahmed. If any part of the system is proven to be copied or reproduced from any source, we shall face the consequences.

**Syed Kamran Qadeer Shah**

**35453**

**Safiullah Shahid**

**35857**

**Muhammad Ismail**

**35897**

**Dedication**

We dedicate this project to **Allah Almighty**, the source of wisdom, inspiration, and knowledge. We also dedicate this project to our families, friends, and teachers, whose support and encouragement have been fundamental to the success of this project. A special thanks goes to our supervisor, Dr. Musharaf Ahmed, for his consistent guidance and support.

**Acknowledgement**

We are immensely grateful to **Allah Almighty** for giving us the strength and perseverance to complete this project. Our deepest gratitude goes to **Dr. Musharaf Ahmed**, whose insights, patience, and support were essential to the success of this work. We would also like to extend our appreciation to all faculty members of Riphah International University for their constant encouragement.

**Abstract**

DineDeal is a mobile-first restaurant reservation platform designed to optimize table occupancy during off-peak hours by offering dynamic discounts. It bridges the gap between users seeking discounted dining experiences and restaurants aiming to fill empty tables during non-peak periods.

The platform incorporates **real-time booking management**, **dynamic discounting**, and **pre-payment options** to reduce no-show rates.

The project aims to tackle shortcomings in existing platforms like OpenTable and FirstTable.

### Table of Contents

#### 1. ****Introduction****

* 1.1 Opportunity and Stakeholder
  + Opportunity
  + Stakeholders
    - Guests/Users
    - Registered Users
    - Restaurants
    - Administrators
* 1.2 Problem Statement
  + Lack of Dynamic Discounting
  + No Pre-Payment Options
  + Limited Restaurant Control
* 1.3 Project Scope
  + Guests
  + Registered Users
  + Restaurants
  + Admin Dashboard
* 1.4 Proposed Solution
  + Real-Time Dynamic Discounts
  + Pre-Payment Options
  + Dashboard for Restaurants
* 1.5 Objectives

#### 2. ****Literature/Market Survey****

* 2.1 Introduction
* 2.2 Literature Review/Market Survey
  + Existing Systems
    - OpenTable
    - FirstTable
    - Key Gaps Identified
  + User Survey Insights
    - Age Group
    - Dining Frequency
    - Awareness of Table Booking Apps
    - Use of Mobile Apps to Browse Restaurants
    - Important Features in a Restaurant App
    - Usefulness of Reading User Reviews
    - Likelihood of Recommending an App Offering Discounts
    - Willingness to Pay a Reservation Fee
* 2.3 Brainstorming
* 2.4 Summary

#### 3. ****Requirement Analysis****

* 3.1 Introduction
* 3.2 Problem Scenarios
  + Problem Statement 1: Optimizing Off-Peak Restaurant Hours
  + Problem Statement 2: High No-Show Rates
  + Problem Statement 3: Limited Restaurant Control
* 3.3 Functional Requirements
  + User Features
    - Browse Restaurants
    - Register
    - Explore Deals
    - Reservation Management
    - Table Availability
    - Bookmark Restaurants
    - Add to Favorites
    - Leave Reviews
  + Restaurant Owner Features
    - Update Information
    - Manage Reservations
    - Offer Discounts
    - Set Pre-Payment Options
  + Admin Features
    - Monitor and Manage Restaurants
    - Generate Reports
* 3.4 Non-Functional Requirements
  + Performance
  + Scalability
  + Usability
  + Reliability
  + Security
  + Maintainability
* 3.5 Summary

#### 4. ****System Design****

* 4.1 Introduction
* 4.2 Architectural Design
* 4.3 Detailed Design
  + 4.3.1 Use Case Diagram
  + 4.3.2 Descriptive Use Cases
    - Browse Restaurants
    - Register
    - Explore Deals
    - Accept Reservation
    - Mark User as Arrived
    - Remove Restaurant
    - Login
    - Add Favorite Deals
    - Browse Table Availability
    - Update Location
    - Delete Review
    - Reserve Table
    - Leave Review
    - Update Deals
    - Set Pre-Payment Option
    - Edit Discounts
    - Delete Deals
    - View Reservation History
    - Post Deals
    - Decline Reservation
    - Mark User as Not Arrived
    - Bookmark Restaurant
    - Make Payment
    - Receive Reservation Confirmation
    - Cancel Reservation
    - Edit Review
    - Generate Revenue Report
    - Monitor Reservations
    - Monitor Reviews
    - Refund
    - Edit Profile
    - Monitor Users
    - Monitor Restaurants
    - Set Table Availability
    - Select Menu
    - Reset Password
* 4.4 Activity Diagram
  + Register
  + Login
  + Explore Deals
  + Accept Reservation
  + Browse Restaurants
  + Reserve Table
  + Mark User as Arrived
  + Mark User as Not Arrived
  + Remove Restaurant
  + Cancel Reservation
  + View Reservation History
  + Receive Reservation Confirmation
  + Leave Reviews
  + Browse Table Availability
  + Set Table Availability
  + Set Discounts
  + Monitor Reviews
  + Update Location
  + Delete Review
  + Edit Review
  + Generate Revenue Reports
  + Post Deals
  + Update Deals
  + Delete Deals
  + Make Payment
  + Decline Reservation
  + Bookmark Restaurant
  + Monitor Reservations
  + Edit Profile
  + Refund
  + Monitor Users
  + Monitor Restaurants
  + Add Favorite Deals
  + Set Pre-Payment Option
  + Select Menu

#### 5. ****Implementation****

#### 6. ****Testing and Evaluation****

* 5.1 Introduction
* 5.2 List of Test Cases
  + Register User
  + Login User
  + Edit Profile
  + Make Reservations
  + Cancel Reservation
  + Set Discounts
  + Set Pre-Payment Amount
  + Update Location

**Chapter 1:**

**Introduction**

**Chapter 1:**

### Introduction

DineDeal is a mobile application focused on improving restaurant table occupancy during off-peak hours by offering flexible and dynamic discounts to users. The idea is to help restaurants maximize their revenue during quieter times while providing users with a platform to book tables at discounted rates. By leveraging technology, DineDeal provides a user-friendly interface for seamless reservation management.

This project addresses the limitations in existing platforms, such as lack of real-time discount flexibility, high no-show rates.

### 1.1 Opportunity and Stakeholder

DineDeal capitalizes on the growing demand for online restaurant bookings, particularly during off-peak hours when restaurants struggle to fill tables. By offering dynamic discounts and easy-to-use mobile interfaces, the app creates a win-win situation for both users and restaurants.

* **Opportunity**:
  1. The global restaurant reservation market is growing at a rate of **15% per year**.
  2. Online booking platforms have seen a **20-25% increase** in usage, with higher engagement on mobile devices.
  3. Restaurants face challenges filling tables during off-peak hours but can increase bookings by **15-30%** with discounts.
* **Stakeholders**:
  1. **Guests/Users**: Individuals exploring restaurant options on the app but cannot do reservations until they create an account on the platform.
  2. **Registered Users**: Users with accounts who benefit from booking history, and discount offers.
  3. **Restaurants**: Partners managing their bookings, discounts, and customer engagements.
  4. **Administrators**: System admins managing user accounts and generating reports.

### 1.2 Problem Statement

The current restaurant reservation platforms, such as OpenTable and FirstTable, have significant gaps in their offerings, including:

1. **Lack of Dynamic Discounting**: Discounts are either non-existent (OpenTable) or fixed (FirstTable), limiting restaurant flexibility.
2. **No Pre-Payment Options**: High no-show rates are common on existing platforms due to the lack of pre-payment or deposit systems.
3. **Limited Restaurant Control**: Restaurants have limited ability to control availability or customize promotions on existing platforms.

**Impact:**  
Restaurants miss out on potential revenue from unsold tables, and users are limited in their choices for discounted dining experiences. The absence of pre-payment options leads to lost revenue from no-shows.

### 1.3 Project Scope

**1**. **Guests**

**Role Overview**: Guests are users who can explore restaurants but need to create an account to make reservations.

**Key Responsibilities**:

1. **Browse Restaurants**: Search and view restaurant information such as menus, hours, and reviews.
2. **View Restaurant Details**: Access restaurant profiles including location and available services.
3. **Create an Account**: Register to make reservations, manage bookings, and receive personalized services.

2. **Registered Users**

**Role Overview**: Registered users are guests who have created an account, allowing for a more personalized experience and advanced booking management.

**Key Responsibilities**:

1. **Manage Account**: Update personal details.
2. **Browse Restaurants**: Explore multiple restaurants, view menus, and find deals.
3. **Table Reservation**: Reserve tables at selected restaurants based on availability.
4. **Receive Notifications**: Get notified about the confirmation or decline of reservations.
5. **Save Favorites**: Bookmark preferred restaurants for easy access.
6. **Leave Reviews**: Write reviews for restaurants after dining.

3. **Restaurants**

**Role Overview**: Restaurants are businesses listed in the app, responsible for managing their reservation availability and interacting with users.

**Key Responsibilities**:

1. **Update Information**: Maintain accurate details such as menus, hours, and deals.
2. **Offer Discounts**: Set and update real-time discounts based on bookings.
3. **Pre**-**payment Option:** Restaurant can enable and set the pre-payment amount on menu or deals.
4. **Manage Availability**: Control table availability and adjust booking slots in real time.
5. **Handle Reservations**: Confirm or cancel incoming reservations.
6. **Receive Notifications**: Get alerts for new reservations and cancellations.
7. **Check User Reviews**: View user reviews about the restaurant.

4. **Admin Dashboard**

**Role Overview**: The admin dashboard allows administrators to manage the system, oversee activities, and generate reports.

**Key Responsibilities**:

1. **Login**: Secure access to the admin interface.
2. **Generate Reports**: Create reports on revenue.
3. **Manage Restaurants**: Remove restaurants based on their bad reviews or requested to be removed.
4. **Browse Restaurant Details**: View restaurant profiles including menus, operating hours, and services.
5. **Check User Reviews**: View user reviews and ratings about restaurants and their services.

### 1.4 Proposed Solution

DineDeal offers a comprehensive solution to these challenges by:

1. **Providing Real-Time Dynamic Discounts**: Restaurants can create and adjust discounts based on real-time demand and table availability.
2. **Offering Pre-Payment Options**: To reduce no-show rates, DineDeal incorporates flexible pre-payment and deposit options.
3. **Dashboard for Restaurants**: Restaurants are given powerful tools to manage bookings, adjust availability, and customize discounts and pre-payments dynamically.

### 1.5 Objectives

The primary objectives of the DineDeal project are:

1. To provide users with a **convenient, mobile-friendly platform** for booking discounted restaurant tables.
2. To offer **real-time discount flexibility** to restaurants, helping them maximize occupancy during off-peak hours.
3. To reduce the issue of **no-shows** through the implementation of **pre-payment Option**.
4. To deliver a **dashboard** for restaurants to manage bookings, pre-payments, discounts, and availability seamlessly.

**Chapter 2:**

**Literature/Market Survey**

**Chapter 2: Literature/Market Survey**

### 2.1 Introduction

This chapter provides an overview of the restaurant reservation market, focusing on existing systems, their features, and gaps in their offerings. It also examines how **DineDeal** addresses these gaps. A market survey was conducted to gather insights about user behavior and preferences, which informed the development of DineDeal’s key differentiating features.

### 2.2 Literature Review/Market Survey

The market for restaurant reservations has several well-established systems, such as **OpenTable** and **FirstTable**, which offer a variety of features but also suffer from certain limitations. By understanding these platforms and their gaps, we were able to identify opportunities for innovation with DineDeal.

**Existing Systems**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **OpenTable** | **First Table** | **DineDeal** |
| Table Reservations | ✔ | ✔ | ✔ |
| Dynamic Discounts(Real-Time) | ❌ | ❌ | ✔ |
| Fixed Discounts (Early Dining) | ❌ | ✔ | ✔ |
| Pre-Payment/Deposit Option | ❌ | ❌ | ✔ |
| No-Show Prevention (Pre-Payment) | ❌ | ❌ | ✔ |
| Restaurant Control Over Discounts | ❌ | ❌ | ✔ |
| User Reviews & Ratings | ✔ | ❌ | ✔ |
| Admin Dashboard for Restaurants | ✔ | ❌ | ✔ |
| Refund Management | ❌ | ❌ | ✔ |
| Booking History for Users | ✔ | ❌ | ✔ |
| Favorite Restaurants | ✔ | ❌ | ✔ |

**1. OpenTable**

**Limitations:**

* No dynamic discounting.
* High no-show rates.
* No pre-payment options to secure reservations.

**2. FirstTable**

**Limitations:**

* Fixed discount model (usually 50%), limiting flexibility.
* Limited restaurant control over table management.
* No tools for restaurants to manage real-time availability.

**Key Gaps Identified:**

* **Lack of Dynamic Discounting**: Both platforms lack dynamic discount options, which limits a restaurant's ability to adjust pricing based on demand.
* **No Pre-Payment Options**: High no-show rates are a recurring issue as neither platform offers pre-payment or deposit features to reduce cancellations.
* **Limited Restaurant Control**: Restaurants have restricted control over real-time booking adjustments and customizing promotions, resulting in lower occupancy during off-peak hours.

To better understand the target users and validate the need for the features DineDeal offers, a survey was conducted with **98 participants**. The following insights were gathered:

**1. Age Group**

Majority are **18-25 years old (88.7%), followed by 28-35 (9.2%).** Targeting younger demographics is essential for app adoption.

**2. Dining Frequency**  
**58.2%** dine out **2-3 times a week**. Regular diners prioritize convenience, making them the core user base.

**3. Awareness of Table Booking Apps**  
**84.7%** were **unaware** of any table booking apps, indicating a market gap and growth potential for DineDeal.

**4. Use of Mobile Apps to Browse Restaurants**  
**44.9%** use apps **occasionally**, while **36.7%** use them **rarely**. The potential exists to increase usage with more engaging features.

**5. Important Features in a Restaurant App**  
**Menu variety** (**58.2%**), **Pricing and deals** (**58.2%**), and **Table availability** (**50%**) are top user priorities. Focus on offering these features.

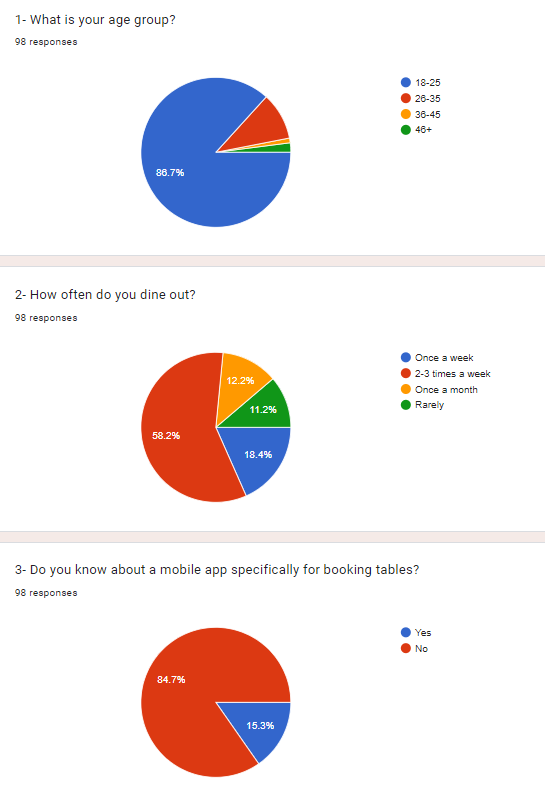
**6. Usefulness of Reading User Reviews**  
**71.4%** find reviews **very helpful**, suggesting that user feedback should be emphasized on the platform.

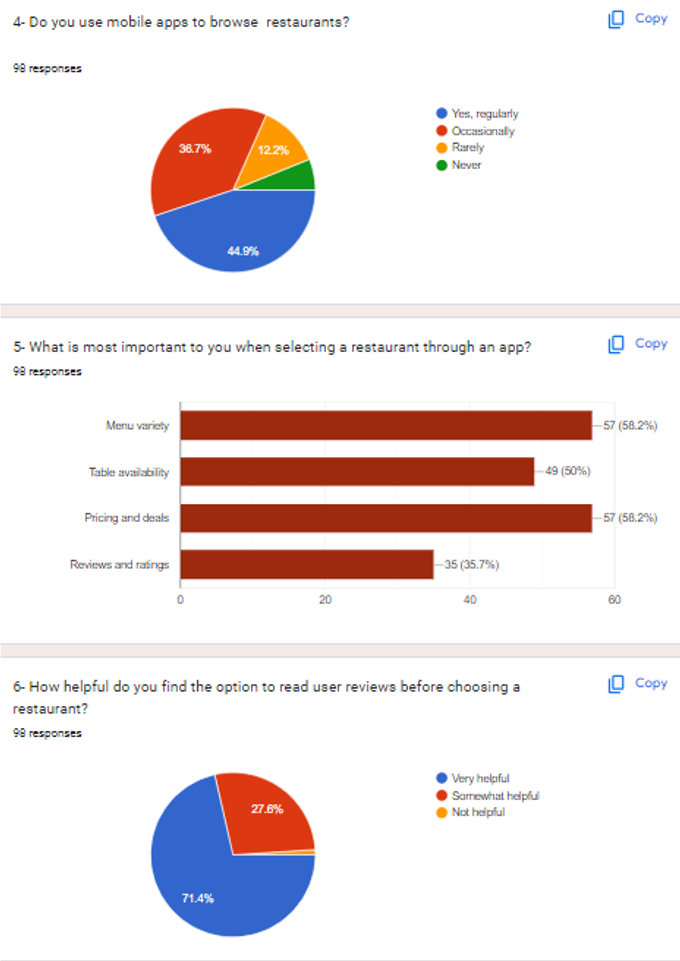
**7. Likelihood of Recommending an App Offering Discounts**  
**60.2%** would **very likely** recommend a restaurant app offering discounts, supporting the need for real-time, dynamic discounting.

**8. Willingness to Pay a Reservation Fee**  
**65.3%** are willing to pay a small fee to guarantee table reservations, reinforcing the value of a pre-payment feature to reduce no-shows.

### 2.3 Brainstorming

Several brainstorming sessions were conducted to identify critical features that could differentiate **DineDeal** from existing platforms. A major insight from these sessions was the need for dynamic, real-time discounts that adjust based on demand. By addressing the limitations of existing systems and focusing on customer pain points, DineDeal can provide a more versatile and efficient solution for both users and restaurants.







### 2.4 Summary

The literature and market survey highlight significant gaps in existing restaurant reservation platforms that DineDeal aims to address. Through features such as real-time discount flexibility, pre-payment options, and better control for restaurants over table management and availability. By targeting the identified preferences and behaviors of younger, frequent diners and offering the features they value most, DineDeal has the potential to capture a significant share of the restaurant reservation market.

**Chapter 3:**

**Requirement Analysis**

**Chapter 3: Requirement Analysis**

### 3.1 Introduction

In this chapter, we will discuss the Functional & Non-Functional requirements of our project “DineDeal”. Prior to that, we will discuss all the problem statements identified. The functional requirements were gathered through interviews and brainstorming. The non-functional requirements are derived based on system performance, security, and reliability needs.

### 3.2 Problem Scenarios

|  |  |
| --- | --- |
| **Problem Statement** | **Details** |
| **Problem Statement 1** | The problem of optimizing off-peak restaurant hours. |
| **Affects** | Restaurants |
| **Result** | Missed revenue opportunities and underutilization of capacity during off-peak hours. |
| **Benefits of a Solution** | Real-time dynamic discounting based on demand allows restaurants to attract more customers during slow periods. |

|  |  |
| --- | --- |
| **Problem Statement** | **Details** |
| **Problem Statement 2** | High no-show rates due to lack of pre-payment options. |
| **Affects** | Restaurants |
| **Result** | Loss of revenue and operational inefficiencies. |
| **Benefits of a Solution** | Introducing pre-payment or deposit systems can significantly reduce no-show rates, ensuring revenue even if cancellations occur. |

|  |  |
| --- | --- |
| **Problem Statement** | **Details** |
| **Problem Statement 3** | Limited control for restaurants over real-time booking adjustments and promotions. |
| **Affects** | Restaurants |
| **Result** | Restaurants are unable to efficiently manage table availability or offer custom promotions in real-time, impacting customer satisfaction and revenue. |
| **Benefits of a Solution** | Providing a user-friendly admin dashboard gives restaurants full control over reservations, dynamic discounts, and promotions. |

**3.3 Functional Requirements**

1. **Browse Restaurants**: Allow users to view restaurant listings, view details.
2. **Register**: Enable users to register by entering details and receiving a verification email.
3. **Explore Deals**: Provide a display of available deals for users.
4. **Reservation Management**:
   * **Accept Reservation**: Allow restaurant owners to accept reservations and notify users.
   * **Decline Reservation**: Enable owners to decline reservations, updating the user.
   * **Mark as Arrived/Not Arrived**: Allow owners to update the reservation status based on user attendance.
   * **Cancel Reservation**: Permit users to cancel reservations within the allowed time, updating availability.
5. **Table Availability**: Show users the table availability and enable owners to set table schedules.
6. **Bookmark**: Bookmark restaurants for quick use.
7. **Add to favorites**: Enable users to add deals to favorites.
8. **User Account Management**:
   * **Login**: Validate and allow access for registered users.
   * **Edit Profile**: Let users update profile details.
   * **View Reservation History**: Display past reservation details to users.
9. **Payments and Refunds**:
   * **Make Payment**: Process payments based on selected payment methods.
   * **Refund**: Issue partial refunds for eligible canceled reservations.
10. **Restaurant Owner Features**:
    * **Update Deals**: Allow owners to add, edit, or delete deals.
    * **Set Discounts**: Enable owners to apply discounts to menus or deals.
    * **Manage Reservations**: Allow owners to view, accept, or decline reservation requests.
11. **Admin Functions**:
    * **Monitor and Manage**: Allow admins to monitor restaurants, users, reservations, and reviews.
    * **Remove Restaurant**: Enable removal of restaurants with bad reviews or upon request.
    * **Generate Reports**: Admins can create revenue reports.

### 3.4 Non-Functional Requirements

1. **Performance**:
   * The system should display restaurant and deal information within 3 seconds of user requests.
   * Reservation processing and status updates should complete within 2 second.
2. **Scalability**:
   * The system must support up to 5000 concurrent users.
3. **Usability**:
   * Interface should be intuitive for both registered and guest users, with clear navigation for restaurant browsing and reservations.
   * Notifications should be immediate upon changes in reservation or payment status.
4. **Reliability**:
   * System uptime should be at least 99.9%, with fallback options for reservation and payment errors.
5. **Security**:
   * User data, especially payment details, must be encrypted.
   * Verification emails and secure authentication for all users and admins.
6. **Maintainability**:
   * Codebase should be modular to allow updates to specific functionalities (like reservation management) without system downtime.

### 3.5 Summary

In this chapter, we outlined the functional and non-functional requirements that are essential for the successful implementation of the DineDeal platform. These requirements address the main challenges identified, including off-peak table optimization, reduced no-show rates, and enhanced control for restaurants through an admin dashboard.

**Chapter 4:**

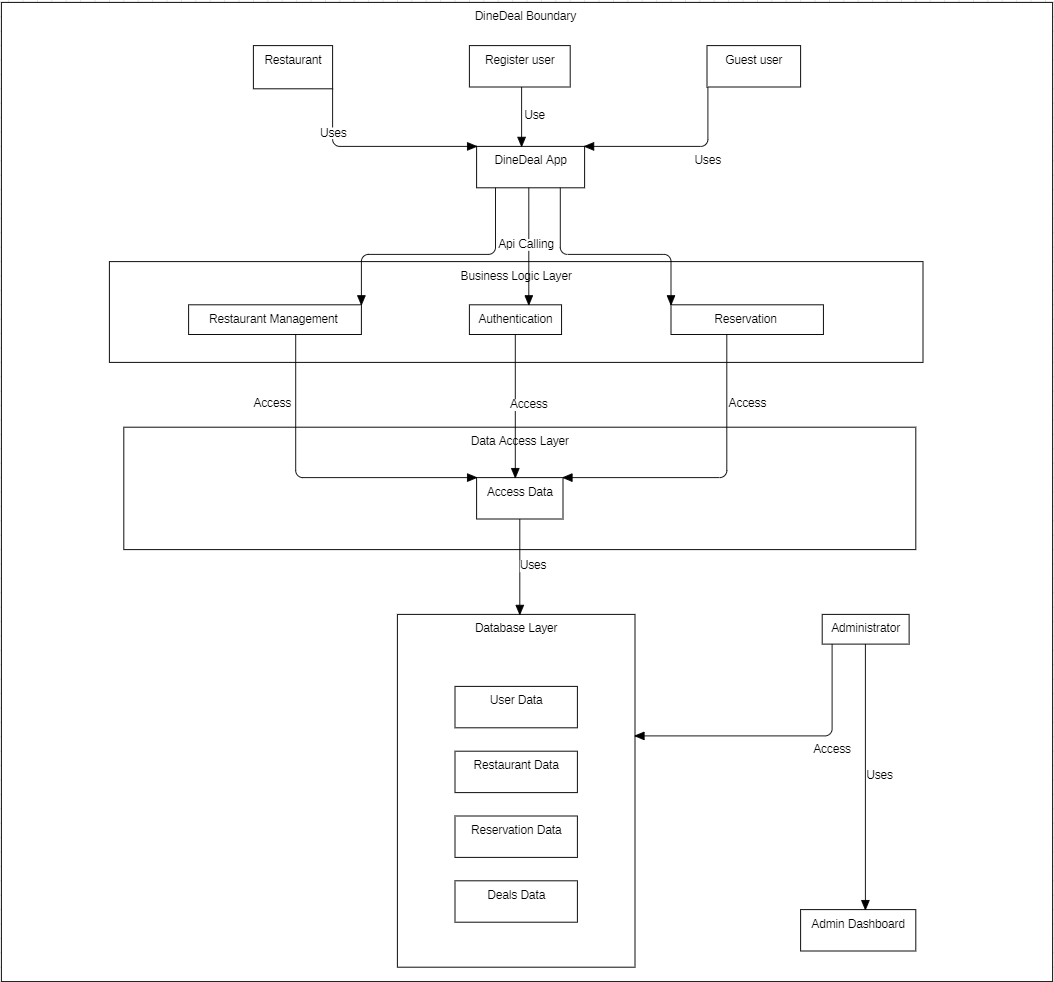
**System Design**

**Chapter 4: System Design**

**4.1 Introduction**

This chapter describes the design aspects of the DineDeal application, detailing the architectural framework, system components, database schema, and interaction diagrams that facilitate the implementation of the restaurant reservation system. The design is structured to ensure scalability, usability, and efficient management of reservations, user accounts, and restaurant profiles.

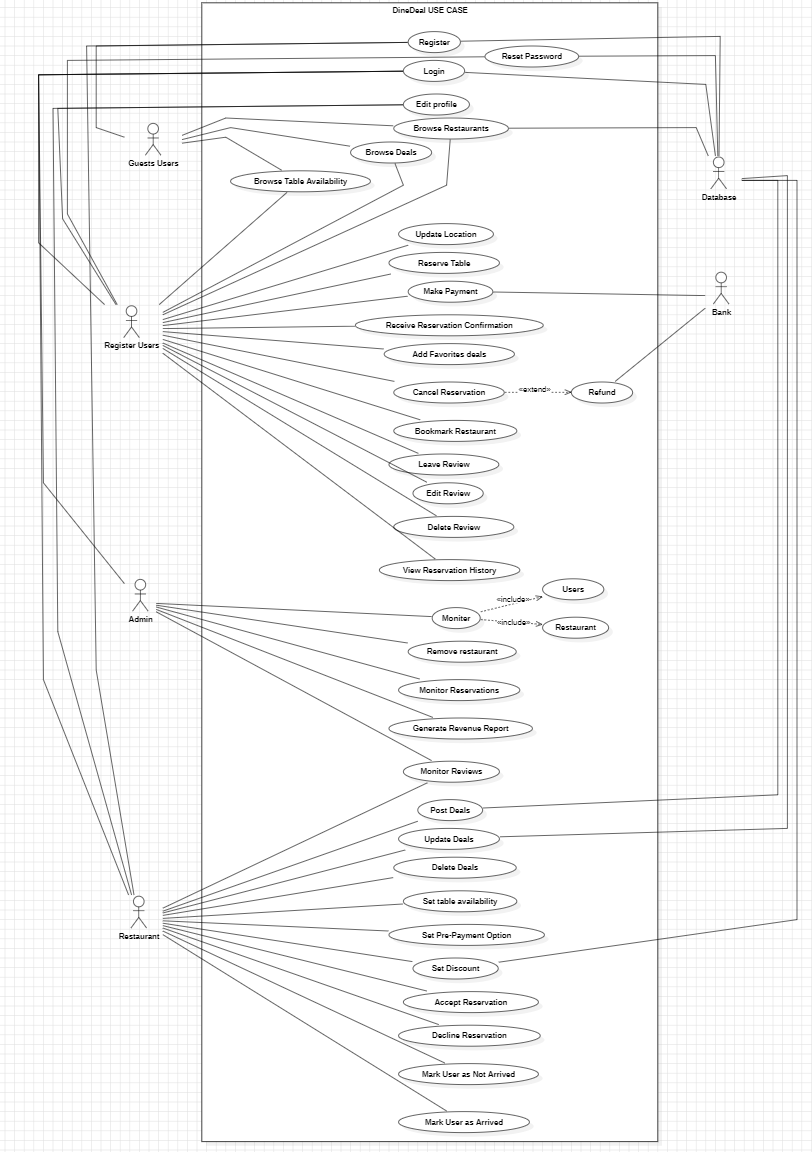
### 4.2 Architectural Design

****

#### 4.3 Detailed Design

This section provides detailed designs for various components, including use case diagrams and descriptive use cases.

### 4.3.1 Use Case Diagram

****

### 4.3.2 Descriptive Use Cases

### Browse Restaurants

|  |  |
| --- | --- |
| **Use Case ID** | **UC01** |
| **Use Case** | Browse Restaurants |
| **Actors** | Registered User, Guest User |
| **Preconditions** | User must access the app. |
| **Basic Flow** | 1. User navigates to the restaurant section. 2. User can search restaurants. 3. System displays the restaurant list. 4. User selects a restaurant to view details (menu, location, pricing, and reviews). |
| **Actor Actions** | Navigate, select, chose a restaurant. |
| **System Response** | List of restaurants is displayed. |
| **Alternative Flow** | 1. If no restaurants, system shows a "No results found" message. 2. User can clear and search again. |
| **Post Condition** | User can view restaurant details. |

### Register

|  |  |
| --- | --- |
| **Use Case ID** | **UC02** |
| **Use Case** | Register |
| **Actors** | Guest User |
| **Preconditions** | User must have valid information (email, password). |
| **Basic Flow** | 1. User selects the "Register" option. 2. User enters information (name, email and password). 3. System validates and creates the user account. |
| **Actor Actions** | Fill out form, submit details. |
| **System Response** | User account is created. |
| **Alternative Flow** | 1. If email, username or password is not valid, system displays an error message. |
| **Post Condition** | Account is created and updated in database. |

### Explore Deals

|  |  |
| --- | --- |
| **Use Case ID** | **UC03** |
| **Use Case** | Explore Deals |
| **Actors** | Registered User, Guest User |
| **Preconditions** | User must access the deals section. |
| **Basic Flow** | 1. User navigates to "Deals". 2. User applies filters to view deals. 3. System displays deals based on the filter. 4. User selects a deal to view details. |
| **Actor Actions** | Filter deals, view details. |
| **System Response** | Display deals and details. |
| **Alternative Flow** | 1. If no deals are available, system shows a "No deals available" message. 2. User can expand filters and retry. |
| **Post Condition** | User can view deal details. |

### Accept Reservation

|  |  |
| --- | --- |
| **Use Case ID** | **UC04** |
| **Use Case** | Accept Reservation |
| **Actors** | Restaurant Owner |
| **Preconditions** | Restaurant Owner has received a reservation request. |
| **Basic Flow** | 1. Restaurant Owner views pending reservation requests. 2. Owner selects a reservation to approve. 3. System confirms reservation acceptance. 4. User receives notification of accepted reservation. |
| **Actor Actions** | Select and confirm acceptance of reservation. |
| **System Response** | Reservation status is updated and user notified. |
| **Alternative Flow** | If approval fails, system displays an error, allowing the owner to retry. |
| **Post Condition** | Reservation is confirmed and user is notified. |

### Mark User as Arrived

|  |  |
| --- | --- |
| **Use Case ID** | **UC05** |
| **Use Case** | Mark User as Arrived |
| **Actors** | Restaurant Owner |
| **Preconditions** | User has a confirmed reservation and arrives at the restaurant. |
| **Basic Flow** | 1. Restaurant Owner verifies user's arrival. 2. Owner selects "Mark as Arrived". 3. System updates reservation status to "Arrived". |
| **Actor Actions** | Select and confirm user's arrival. |
| **System Response** | Status is updated to "Arrived". |
| **Alternative Flow** | If system update fails, an error message is shown, and the owner can retry. |
| **Post Condition** | User is marked as arrived, allowing review access. |

### Remove Restaurant

|  |  |
| --- | --- |
| **Use Case ID** | **UC06** |
| **Use Case** | Remove Restaurant |
| **Actors** | Admin |
| **Preconditions** | Restaurant have bad reviews or requested for removal. |
| **Basic Flow** | 1. Admin reviews restaurant. 2. Admin selects "Remove Restaurant". 3. System removes the restaurant from listings. |
| **Actor Actions** | Select and confirm removal. |
| **System Response** | Restaurant is removed from listings. |
| **Alternative Flow** | If removal fails, system displays an error and allows retry. |
| **Post Condition** | Restaurant is removed from user access. |

### Login

|  |  |
| --- | --- |
| **Use Case ID** | **UC07** |
| **Use Case** | Login |
| **Actors** | Registered User |
| **Preconditions** | User must have a registered account. |
| **Basic Flow** | 1. User selects "Login". 2. User enters email and password. 3. System verifies and grants access to the account. |
| **Actor Actions** | Enter login credentials. |
| **System Response** | User is logged in successfully. |
| **Alternative Flow** | If login fails, system displays an error and try again. |
| **Post Condition** | User is logged in. |

### Add Favorite Deals to Cart

|  |  |
| --- | --- |
| **Use Case ID** | **UC08** |
| **Use Case** | Add Favorite Deals |
| **Actors** | Registered User |
| **Preconditions** | User must be logged in and viewing deals. |
| **Basic Flow** | 1. User selects a deal to add to favorite. 2. User clicks add to favorites. 3. System confirms deal is added. |
| **Actor Actions** | Select deal and add to favorite. |
| **System Response** | Deal is added. |
| **Alternative Flow** | If deal is unavailable, system shows error. |
| **Post Condition** | Deal is in the user’s favorites. |

### Browse Table Availability

|  |  |
| --- | --- |
| **Use Case ID** | **UC09** |
| **Use Case** | Browse Table Availability |
| **Actors** | Registered User |
| **Preconditions** | User must be logged in. |
| **Basic Flow** | 1. User selects a restaurant. 2. Go on reserve and select the available table. 3. System checks and reserve the table. |
| **Actor Actions** | Choose restaurant and go to reserve. |
| **System Response** | Display available tables. |
| **Alternative Flow** | If no tables are available, select other times. |
| **Post Condition** | User can view availability. |

### 

### Update Location

|  |  |
| --- | --- |
| **Use Case ID** | **UC10** |
| **Use Case** | Update Location |
| **Actors** | Registered User |
| **Preconditions** | User must be logged in. |
| **Basic Flow** | 1. User navigates to profile settings. 2. User selects "Update Location". 3. User enters new location. 4. System updates and saves the location. |
| **Actor Actions** | Enter new location and submit. |
| **System Response** | Location is updated successfully. |
| **Alternative Flow** | If location update fails, system displays an error, allowing the user to retry. |
| **Post Condition** | User’s location is updated in the system. |

### Delete Review

|  |  |
| --- | --- |
| **Use Case ID** | **UC11** |
| **Use Case** | Delete Review |
| **Actors** | Registered User |
| **Preconditions** | User must have an existing review. |
| **Basic Flow** | 1. User navigates to their reviews. 2. User selects a review to delete. 3. User confirms deletion. 4. System deletes the review. |
| **Actor Actions** | Select review and confirm deletion. |
| **System Response** | Review is successfully deleted. |
| **Alternative Flow** | If deletion fails, system displays an error and allows the user to retry. |
| **Post Condition** | Review is removed from the system. |

### Reserve Table

|  |  |
| --- | --- |
| **Use Case ID** | **UC12** |
| **Use Case** | Reserve Table |
| **Actors** | Registered User |
| **Preconditions** | User must be logged in and table availability must be confirmed. |
| **Basic Flow** | 1. User selects date, time, and table. 2. User confirms the reservation details. 3. System processes and confirms the reservation. |
| **Actor Actions** | Enter reservation details and confirm. |
| **System Response** | Reservation is successfully created. |
| **Alternative Flow** | If reservation fails, system displays an error message, and user can retry. |
| **Post Condition** | Table reservation is confirmed. |

### Leave Review

|  |  |
| --- | --- |
| **Use Case ID** | **UC13** |
| **Use Case** | Leave Review |
| **Actors** | Registered User |
| **Preconditions** | User must have checked in to the restaurant. |
| **Basic Flow** | 1. User navigates to the review section. 2. User writes a review and submits it. 3. System posts the review for the restaurant. |
| **Actor Actions** | Write and submit review. |
| **System Response** | Review is successfully posted. |
| **Alternative Flow** | If submission fails, system displays an error, and user can retry. |
| **Post Condition** | Review is visible on the restaurant's profile. |

### Update Deals

|  |  |
| --- | --- |
| **Use Case ID** | **UC14** |
| **Use Case** | Update Deals |
| **Actors** | Restaurant Owner |
| **Preconditions** | Owner must have existing deals to update. |
| **Basic Flow** | 1. Owner selects a deal to update. 2. Owner edits deal details. 3. System updates the deal. |
| **Actor Actions** | Edit deal details and submit. |
| **System Response** | Deal information is successfully updated. |
| **Alternative Flow** | If update fails, system displays an error, allowing the owner to retry. |
| **Post Condition** | Deal is updated successfully. |

### Set Pre-Payment Option

|  |  |
| --- | --- |
| **Use Case ID** | **UC15** |
| **Use Case** | Set Pre-Payment Option |
| **Actors** | Restaurant Owner |
| **Preconditions** | Owner must be logged in. |
| **Basic Flow** | 1. Owner navigates to payment settings. 2. Owner enables the pre-payment option and sets the amount. 3. System saves the updated settings. |
| **Actor Actions** | Set pre-payment amount and confirm. |
| **System Response** | Pre-payment option is set successfully. |
| **Alternative Flow** | If setting fails, system displays an error and allows owner to retry. |
| **Post Condition** | Pre-payment option is enabled and configured. |

### Edit Discounts

|  |  |
| --- | --- |
| **Use Case ID** | **UC16** |
| **Use Case** | Set Discount |
| **Actors** | Restaurant Owner |
| **Preconditions** | Owner must have a menu or deal to apply discount on it. |
| **Basic Flow** | 1. Owner selects a menu or deal. 2. Owner enables and set the discount amount on it. 3. System updates the discount. |
| **Actor Actions** | Set discount details and submit. |
| **System Response** | Discount information is updated successfully. |
| **Alternative Flow** | If update fails, system displays an error and allows retry. |
| **Post Condition** | Discount is updated on the platform. |

### Delete Deals

|  |  |
| --- | --- |
| **Use Case ID** | **UC17** |
| **Use Case** | Delete Deals |
| **Actors** | Restaurant Owner |
| **Preconditions** | Owner must have existing deals to delete. |
| **Basic Flow** | 1. Owner selects a deal to delete. 2. Owner confirms deletion. 3. System removes the deal from the listing. |
| **Actor Actions** | Select deal and confirm deletion. |
| **System Response** | Deal is successfully deleted. |
| **Alternative Flow** | If deletion fails, system displays an error, allowing retry. |
| **Post Condition** | Deal is removed from user access. |

### View Reservation History

|  |  |
| --- | --- |
| **Use Case ID** | **UC18** |
| **Use Case** | View Reservation History |
| **Actors** | Registered User |
| **Preconditions** | User must have prior reservations. |
| **Basic Flow** | 1. User navigates to reservation history. 2. System displays a list of past reservations. 3. User selects a reservation to view details. |
| **Actor Actions** | View and select past reservation. |
| **System Response** | Displays past reservation details. |
| **Alternative Flow** | If no reservations exist, system displays a "No history available" message. |
| **Post Condition** | User can view details of past reservations. |

### Post Deals

|  |  |
| --- | --- |
| **Use Case ID** | **UC19** |
| **Use Case** | Post Deals |
| **Actors** | Restaurant Owner |
| **Preconditions** | Owner must be logged in and ready to post deals. |
| **Basic Flow** | 1. Owner selects "Post New Deal". 2. Owner enters details about the deal. 3. System publishes the deal for users to view. |
| **Actor Actions** | Enter and submit deal details. |
| **System Response** | Deal is successfully posted. |
| **Alternative Flow** | If posting fails, system displays an error, allowing retry. |
| **Post Condition** | New deal is visible to users. |

### Decline Reservation

|  |  |
| --- | --- |
| **Use Case ID** | **UC20** |
| **Use Case** | Decline Reservation |
| **Actors** | Restaurant Owner |
| **Preconditions** | Restaurant Owner has received a reservation request. |
| **Basic Flow** | 1. Owner views pending reservation requests. 2. Owner selects a reservation to decline. 3. System updates status to "Declined" and notifies the user. |
| **Actor Actions** | Select reservation and confirm decline. |
| **System Response** | Reservation is declined and user notified. |
| **Alternative Flow** | If decline fails, system displays an error and allows retry. |
| **Post Condition** | Reservation is declined, and user is informed. |

### Mark User as Not Arrived

|  |  |
| --- | --- |
| **Use Case ID** | **UC21** |
| **Use Case** | Mark User as Not Arrived |
| **Actors** | Restaurant Owner |
| **Preconditions** | User has a confirmed reservation but has not arrived within the allotted time. |
| **Basic Flow** | 1. Owner verifies user's absence. 2. Owner selects "Mark as Not Arrived". 3. System updates reservation status and frees up the table. |
| **Actor Actions** | Confirm user’s absence and mark as "Not Arrived". |
| **System Response** | Status is updated, and table is available for other reservations. |
| **Alternative Flow** | If marking fails, system displays an error and allows retry. |
| **Post Condition** | User is marked as not arrived. |

### Bookmark Restaurant

|  |  |
| --- | --- |
| **Use Case ID** | **UC22** |
| **Use Case** | Bookmark Restaurant |
| **Actors** | Registered User |
| **Preconditions** | User must be logged in. |
| **Basic Flow** | 1. User views restaurant details. 2. User selects "Bookmark". 3. System adds restaurant to the user’s bookmarks. |
| **Actor Actions** | Select restaurant and add to bookmarks. |
| **System Response** | Restaurant is successfully bookmarked. |
| **Alternative Flow** | If bookmarking fails, system displays an error and allows retry. |
| **Post Condition** | Restaurant is bookmarked for quick access. |

### Make Payment

|  |  |
| --- | --- |
| **Use Case ID** | **UC23** |
| **Use Case** | Make Payment |
| **Actors** | Registered User |
| **Preconditions** | User must have a confirmed reservation with a payment requirement. |
| **Basic Flow** | 1. User navigates to payment section. 2. User selects payment method. 3. User enters payment details and confirms. 4. System processes payment. |
| **Actor Actions** | Enter payment details and confirm. |
| **System Response** | Payment is processed successfully. |
| **Alternative Flow** | If payment fails, system displays an error and allows retry. |
| **Post Condition** | Payment is successfully completed. |

### Receive Reservation Confirmation

|  |  |
| --- | --- |
| **Use Case ID** | **UC24** |
| **Use Case** | Receive Reservation Confirmation |
| **Actors** | Registered User |
| **Preconditions** | User has successfully reserved a table. |
| **Basic Flow** | 1. System generates reservation confirmation. 2. Confirmation is sent to the user's email and in-app notification. |
| **Actor Actions** | Review confirmation details. |
| **System Response** | Confirmation is sent successfully. |
| **Alternative Flow** | If confirmation fails to send, user can contact support. |
| **Post Condition** | User has confirmation details for their reservation. |

### Cancel Reservation

|  |  |
| --- | --- |
| **Use Case ID** | **UC25** |
| **Use Case** | Cancel Reservation |
| **Actors** | Registered User |
| **Preconditions** | User must have a confirmed reservation. |
| **Basic Flow** | 1. User selects the reservation to cancel (within time restrictions). 2. User confirms the cancellation. 3. System cancels reservation and updates the status. |
| **Actor Actions** | Select reservation and confirm cancellation. |
| **System Response** | Reservation is canceled if the user cancels within time limit. |
| **Alternative Flow** | If cancellation fails, system displays an error and allows retry. |
| **Post Condition** | Reservation is canceled and table is available. |

### Edit Review

|  |  |
| --- | --- |
| **Use Case ID** | **UC26** |
| **Use Case** | Edit Review |
| **Actors** | Registered User |
| **Preconditions** | User must have an existing review. |
| **Basic Flow** | 1. User navigates to their reviews. 2. User selects a review to edit. 3. User updates review content. 4. System saves the updated review. |
| **Actor Actions** | Edit review and submit changes. |
| **System Response** | Review is updated successfully. |
| **Alternative Flow** | If editing fails, system displays an error, allowing retry. |
| **Post Condition** | Review is successfully updated. |

### Generate Revenue Report

|  |  |
| --- | --- |
| **Use Case ID** | **UC27** |
| **Use Case** | Generate Revenue Report |
| **Actors** | Admin |
| **Preconditions** | Admin must have access to revenue data. |
| **Basic Flow** | 1. Admin navigates to reports section. 2. Admin selects "Generate Revenue Report". 3. System compiles data and generates the report. |
| **Actor Actions** | Choose report options and confirm generation. |
| **System Response** | Revenue report is generated successfully. |
| **Alternative Flow** | If report generation fails, system displays an error and allows retry. |
| **Post Condition** | Revenue report is available for review. |

### Monitor Reservations

|  |  |
| --- | --- |
| **Use Case ID** | **UC28** |
| **Use Case** | Monitor Reservations |
| **Actors** | Admin |
| **Preconditions** | Admin must have access to reservation data. |
| **Basic Flow** | 1. Admin navigates to reservations section. 2. Admin views current and past reservations. 3. Admin applies filters to view reservation details. |
| **Actor Actions** | Apply filters and monitor reservations. |
| **System Response** | Reservations are displayed based on selected criteria. |
| **Alternative Flow** | If no reservations found, system shows a "No reservations available" message. |
| **Post Condition** | Admin can monitor reservations effectively. |

### Monitor Reviews

|  |  |
| --- | --- |
| **Use Case ID** | **UC29** |
| **Use Case** | Monitor Reviews |
| **Actors** | Admin |
| **Preconditions** | Admin must have access to reviews data. |
| **Basic Flow** | 1. Admin navigates to reviews section. 2. Admin views all reviews for different restaurants. |
| **Actor Actions** | Monitor reviews. |
| **System Response** | Reviews are displayed. |
| **Alternative Flow** | If no reviews are found, system displays a "No reviews available" message. |
| **Post Condition** | Admin can monitor reviews. |

### Refund

|  |  |
| --- | --- |
| **Use Case ID** | **UC30** |
| **Use Case** | Refund |
| **Actors** | Registered User, Admin |
| **Preconditions** | User must have canceled a reservation eligible for a refund. |
| **Basic Flow** | 1. User requests a refund for a canceled reservation. 2. System calculates refund amount with a small deduction. 3. System processes and issues refund to the user. |
| **Actor Actions** | Submit refund request. |
| **System Response** | Refund is processed and issued successfully. |
| **Alternative Flow** | If refund fails, system displays an error and allows retry. |
| **Post Condition** | User receives a partial refund. |

### Edit Profile

|  |  |
| --- | --- |
| **Use Case ID** | **UC31** |
| **Use Case** | Edit Profile |
| **Actors** | Registered User |
| **Preconditions** | User must be logged in. |
| **Basic Flow** | 1. User navigates to profile settings. 2. User edits profile details (name, contact information, etc.). 3. System saves changes. |
| **Actor Actions** | Make changes and save profile. |
| **System Response** | Profile details are successfully updated. |
| **Alternative Flow** | If update fails, system displays an error and allows retry. |
| **Post Condition** | User’s profile is updated. |

### Monitor Users

|  |  |
| --- | --- |
| **Use Case ID** | **UC32** |
| **Use Case** | Monitor Users |
| **Actors** | Admin |
| **Preconditions** | Admin must have access to user data. |
| **Basic Flow** | 1. Admin navigates to user management. 2. Admin views and filters user information. 3. Admin selects a user to view detailed information. |
| **Actor Actions** | Monitor and manage user accounts. |
| **System Response** | User data is displayed based on selected criteria. |
| **Alternative Flow** | If no users are found, system shows a "No users available" message. |
| **Post Condition** | Admin can monitor user activity. |

### Monitor Restaurant

|  |  |
| --- | --- |
| **Use Case ID** | **UC33** |
| **Use Case** | Monitor Restaurant |
| **Actors** | Admin |
| **Preconditions** | Admin must have access to restaurant data. |
| **Basic Flow** | 1. Admin navigates to restaurant management. 2. Admin views and filters restaurant details. 3. Admin selects a restaurant to view further information. |
| **Actor Actions** | Monitor and manage restaurant profiles. |
| **System Response** | Restaurant data is displayed based on selected criteria. |
| **Alternative Flow** | If no restaurants are found, system shows a "No restaurants available" message. |
| **Post Condition** | Admin can monitor and manage restaurant profiles. |

### Set Table Availability

|  |  |
| --- | --- |
| **Use Case ID** | **UC34** |
| **Use Case** | Set Table Availability |
| **Actors** | Restaurant Owner |
| **Preconditions** | Owner must have access to table settings. |
| **Basic Flow** | 1. Owner navigates to table management. 2. Owner sets availability for tables (date, time, capacity). 3. System updates table status accordingly. |
| **Actor Actions** | Set and save table availability. |
| **System Response** | Table availability is updated successfully. |
| **Alternative Flow** | If update fails, system displays an error and allows retry. |
| **Post Condition** | Table availability is updated for reservations. |

### 

### Select Menu

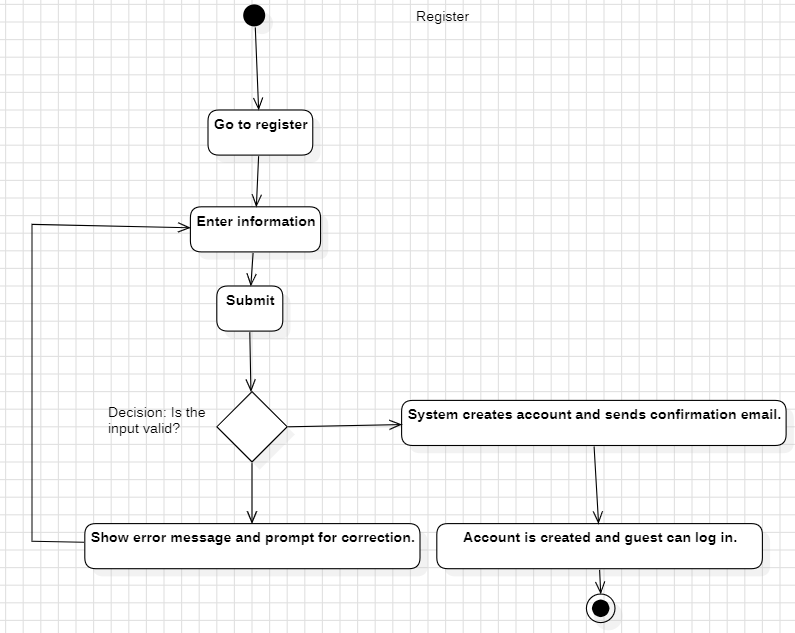
|  |  |
| --- | --- |
| **Use Case ID** | **UC35** |
| **Use Case** | Select Menu |
| **Actors** | Registered User, Guest User |
| **Preconditions** | User must access a restaurant’s page with available menu items. |
| **Basic Flow** | 1. User navigates to a restaurant’s page.  2. User selects the "Menu" section.  3. System displays the list of available menu items, categorized by type (appetizers, main courses, desserts, etc.).  4. User views menu details (item name, description, price, and any applicable dietary labels). |
| **Actor Actions** | Navigate to menu section, view menu items. |
| **System Response** | Display the restaurant’s full menu, including item details. |
| **Alternative Flow** | 1. If the menu is unavailable, the system displays a "Menu not available" message.  2. User can browse other sections of the restaurant’s profile. |
| **Post Condition** | User can view all menu items and make informed choices. |

**Reset Password**

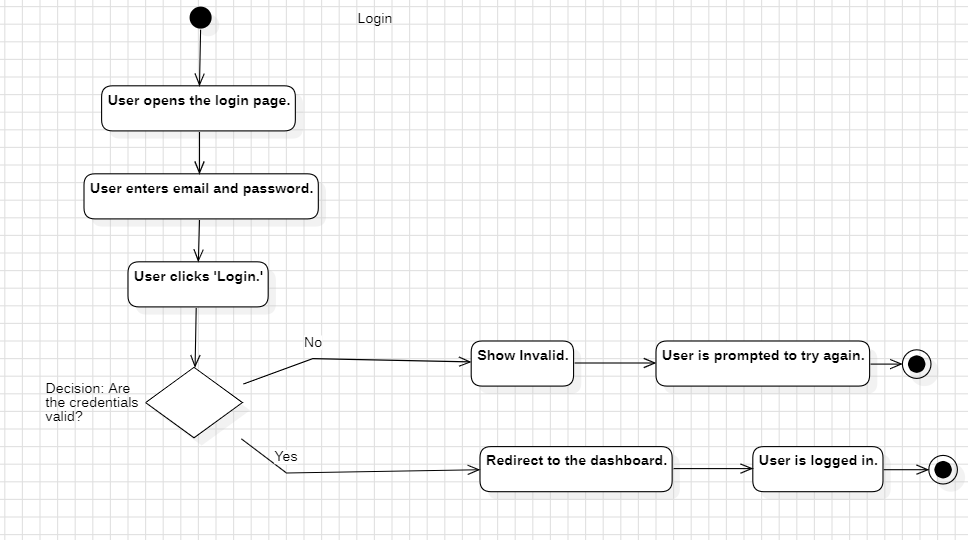
|  |  |
| --- | --- |
| **Use Case ID** | **UC36** |
| **Use Case** | Reset Password |
| **Actors** | Registered User |
| **Preconditions** | User must have a registered account with a valid email address. |
| **Basic Flow** | 1. User selects the "Forgot Password" option on the login screen. 2. User enters their registered email address. 3. System sends otp on that email. 4. User use that otp to reset the password. 5. User creates and confirms a new password. 6. 8. System updates the password and confirms the reset. |
| **Actor Actions** | Request password reset, access email, reset password. |
| **System Response** | Validates email, sends otp, and updates the password. |
| **Alternative Flow** | 1. If the email is not registered, the system displays an error message ("Email not found"). 2. If the otp code expires, the system prompts the user to request a new code. |

#### 4.4 Activity Diagram

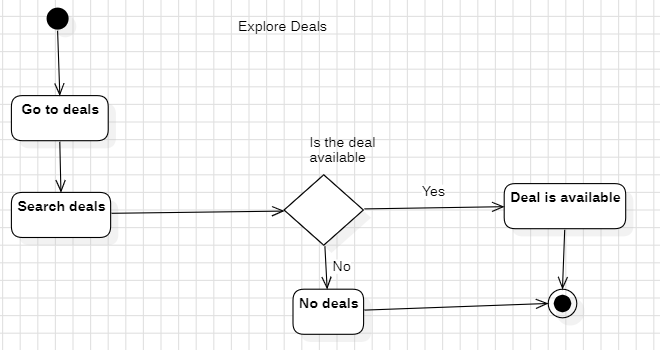
1. **Register**

****

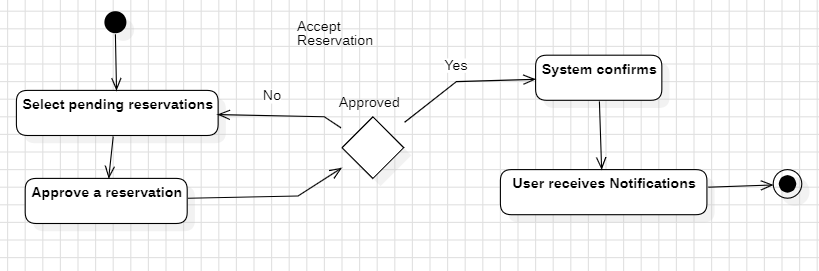
1. **Login**

****

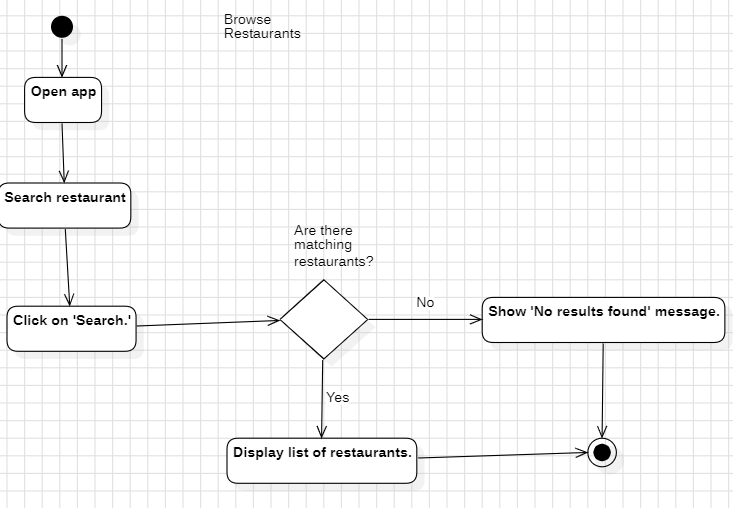
1. **Explore Deals**

****

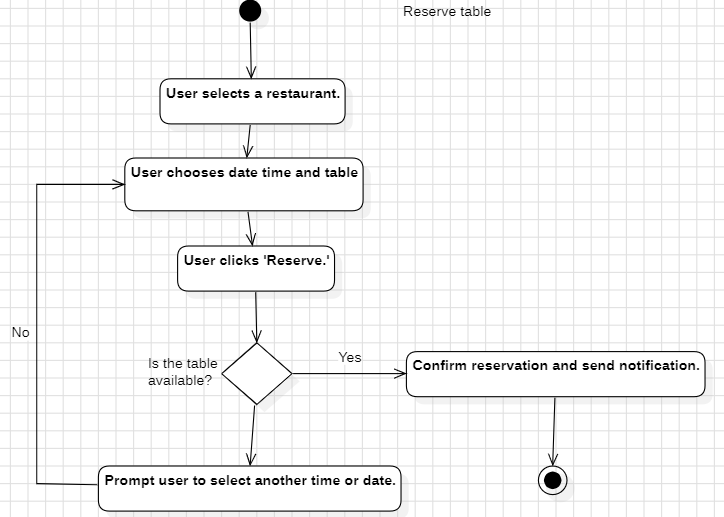
1. **Accept Reservation**

****

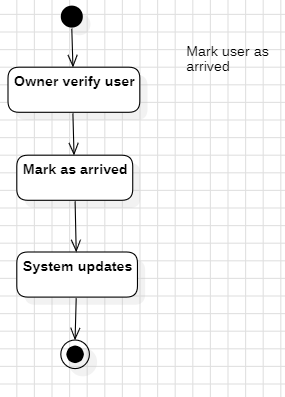
1. **Browse Restaurants**

****

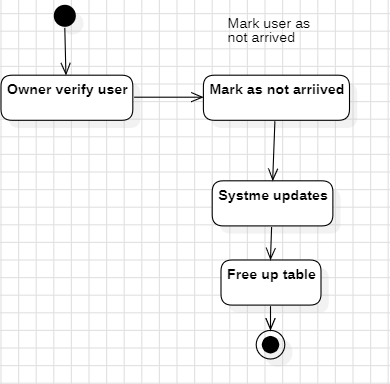
1. **Reserve table**

****

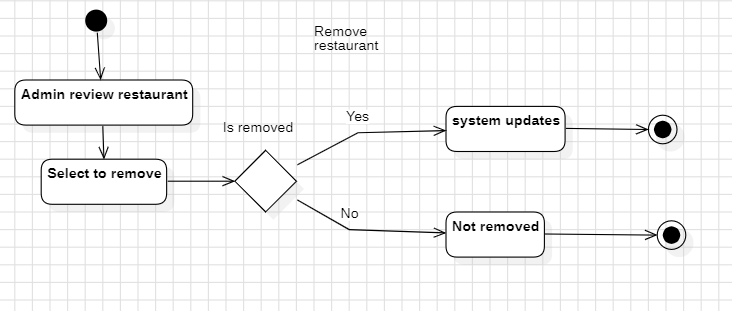
1. **Mark user as arrived**

****

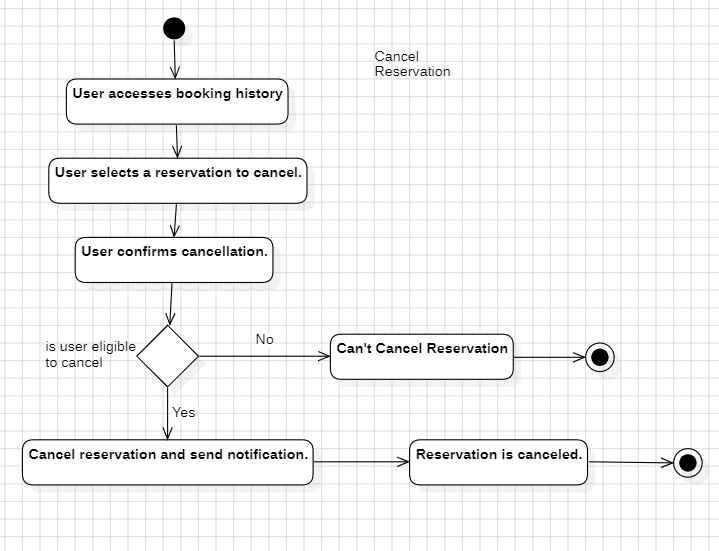
1. **Mark user as not arrived**

****

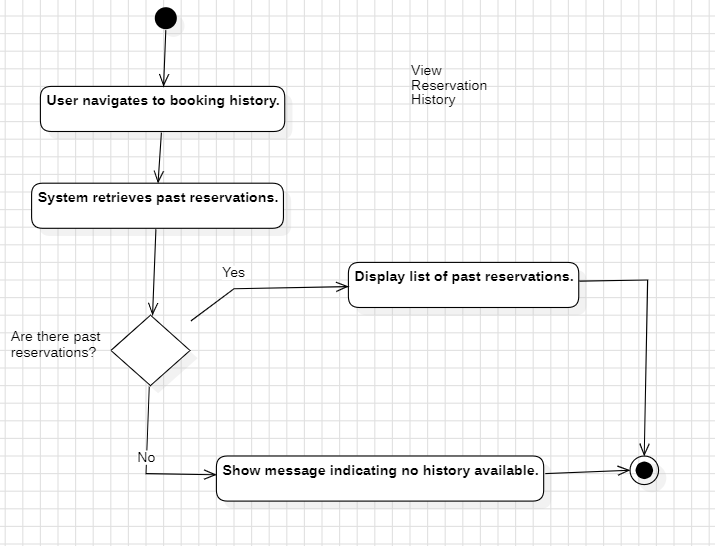
1. **Remove restaurant**

****

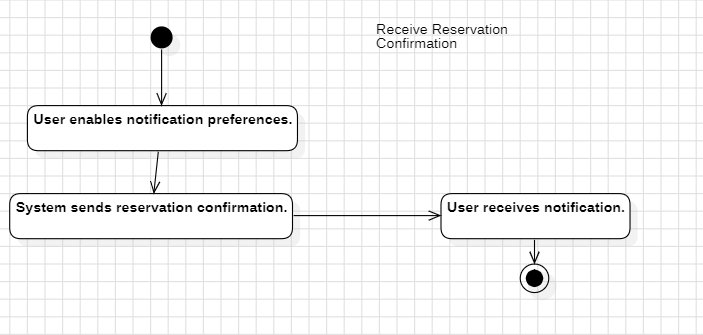
1. **Cancel Reservation**

****

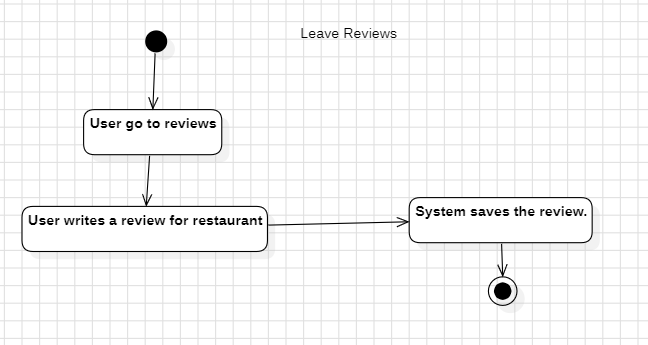
1. **View Reservation History**

****

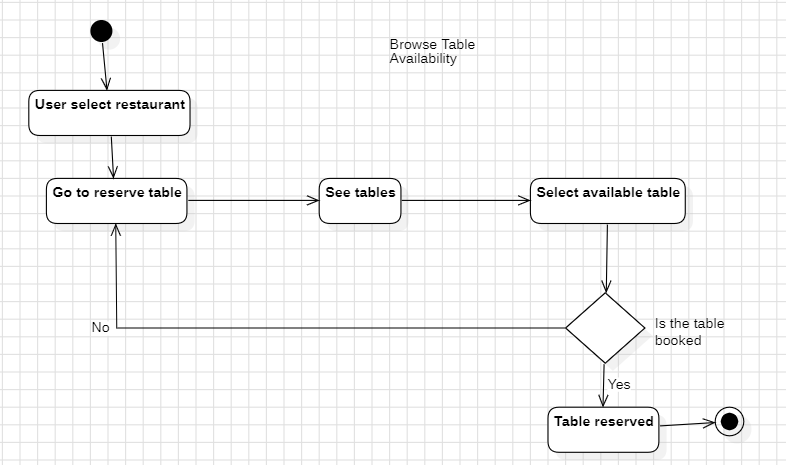
1. **Receive Reservation Confirmation**

****

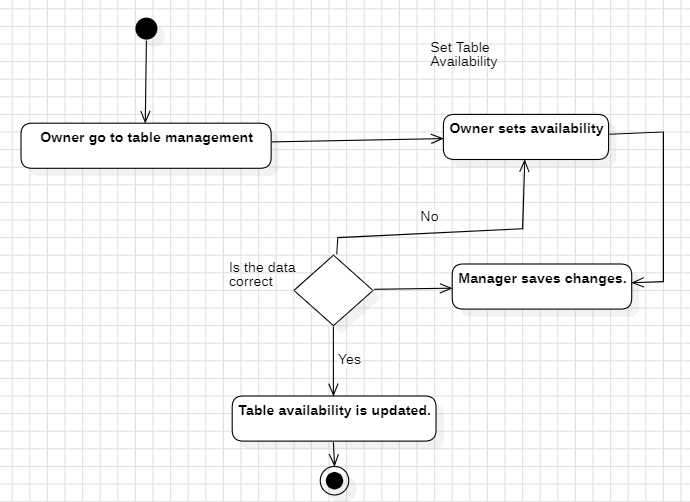
1. **Leave Reviews**

****

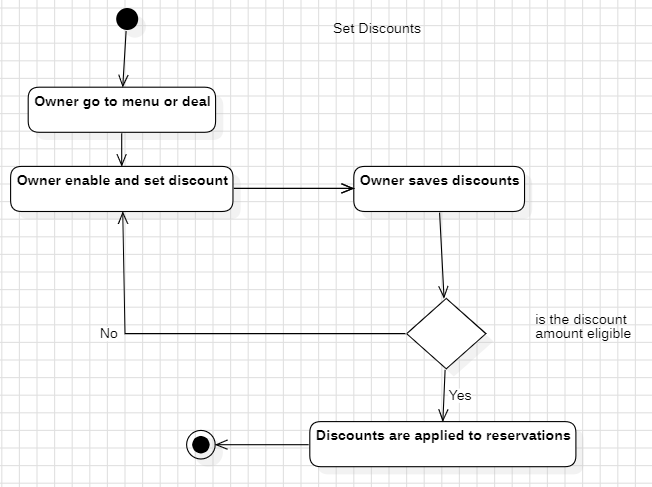
1. **Browse Table Availability**

****

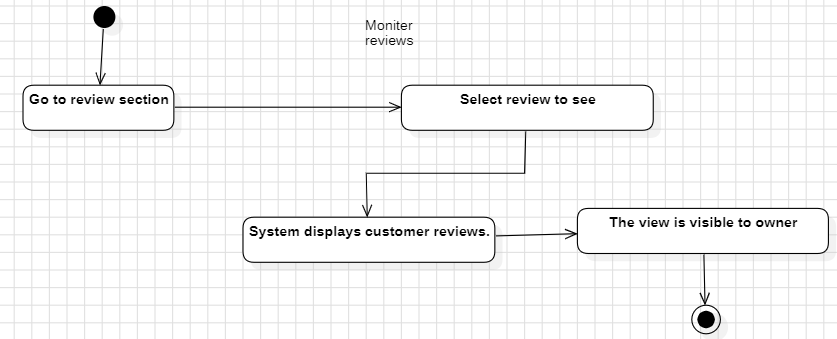
1. **Set Table Availability**

****

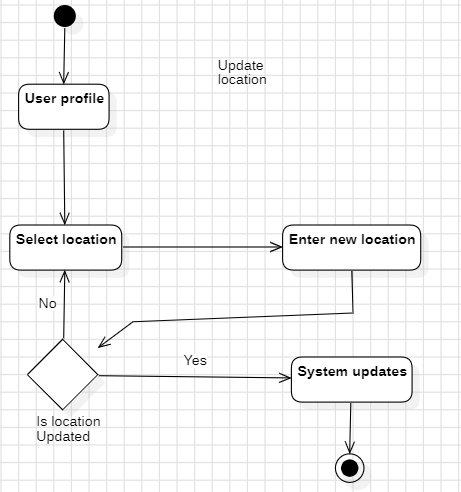
1. **Set Discounts**

****

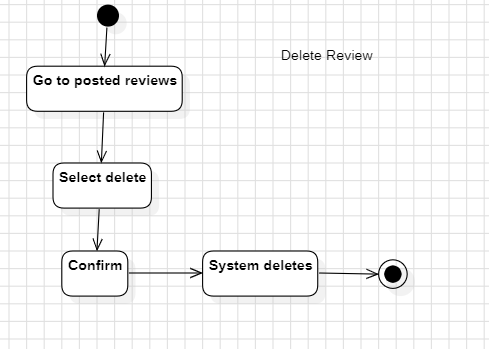
1. **Moniter reviews**

****

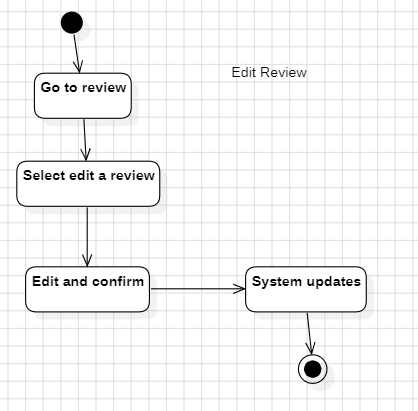
1. **Update location**

****

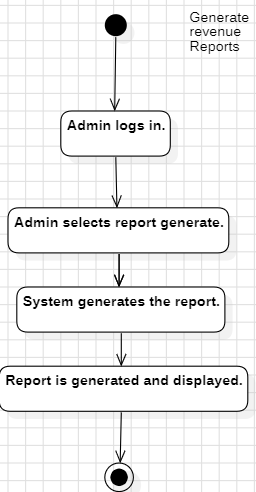
1. **Delete Review**

****

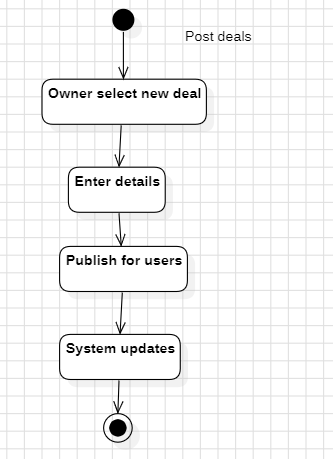
1. **Edit Review**

****

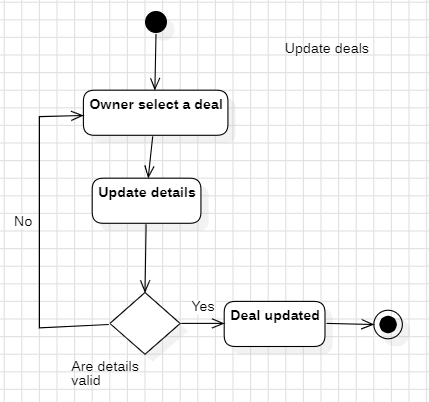
1. **Generate revenue Reports**

****

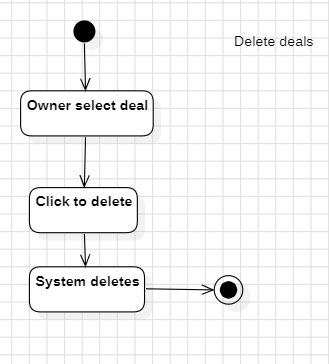
1. **Post deals**

****

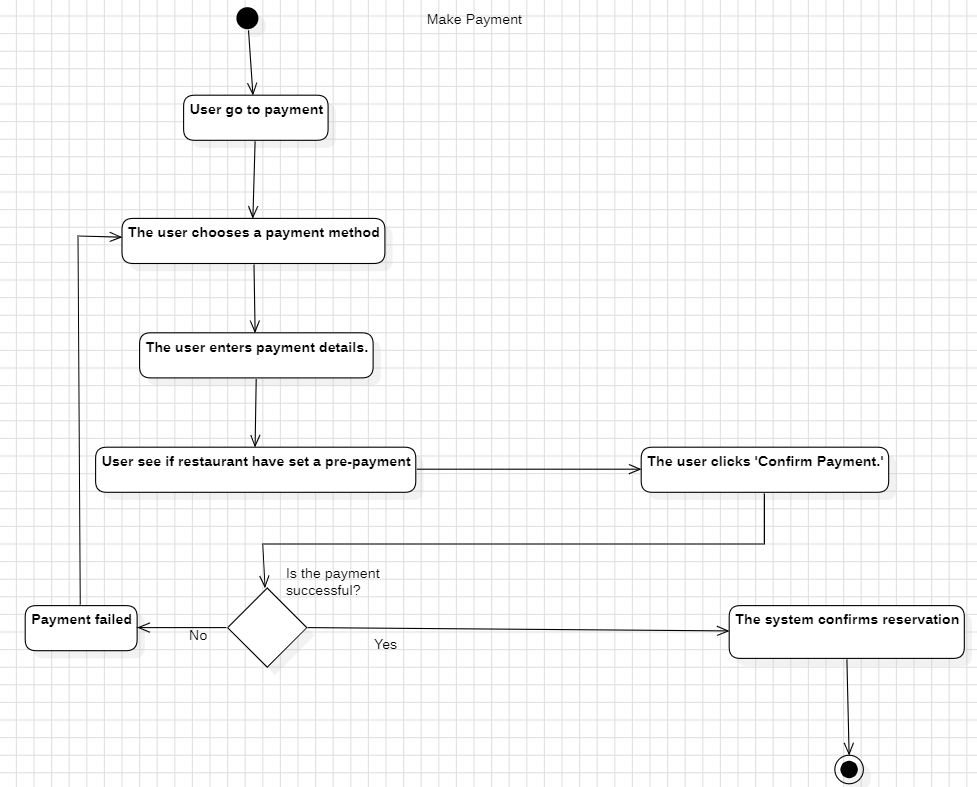
1. **Update deals**

****

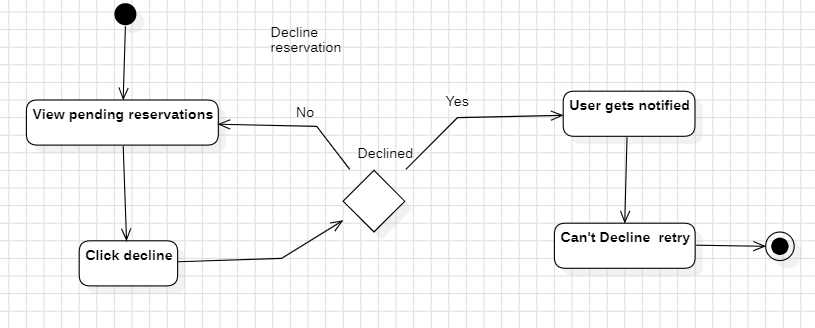
1. **Delete deals**

****

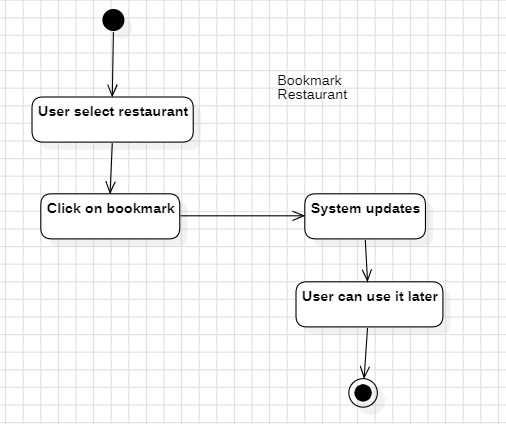
1. **Make Payment**

****

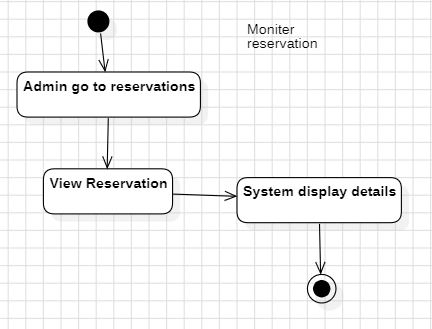
1. **Decline reservation**

****

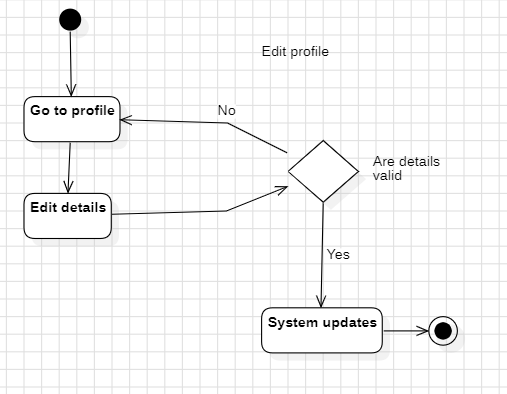
1. **Bookmark Restaurant**

****

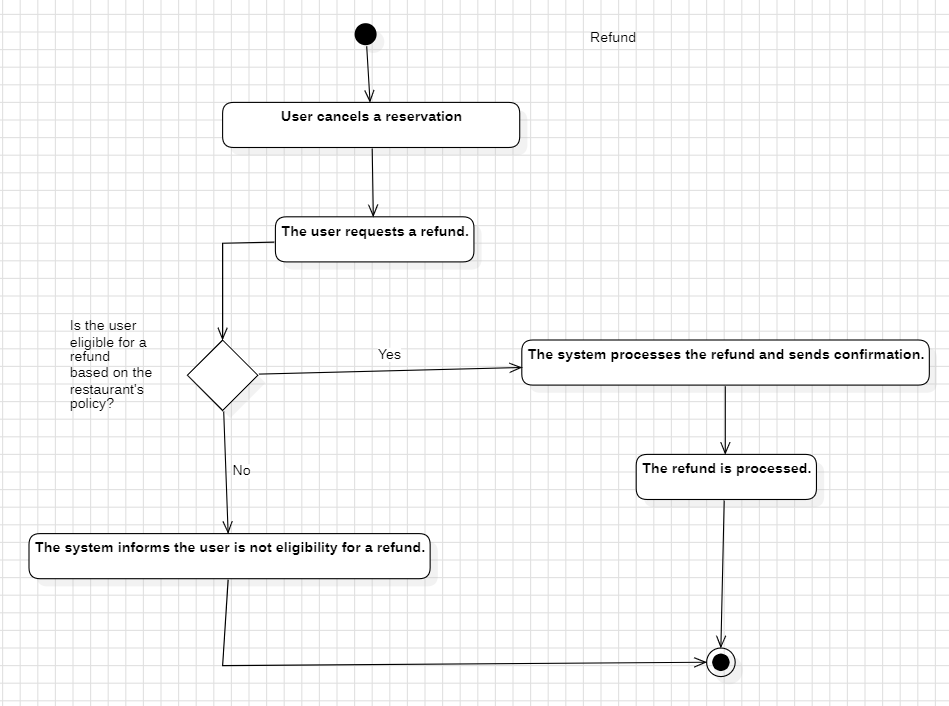
1. **Moniter reservation**

****

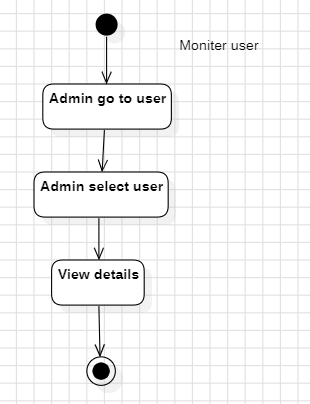
1. **Edit profile**

****

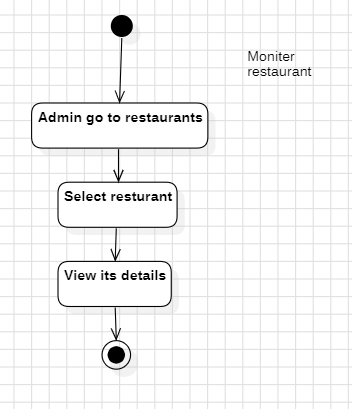
1. **Refund**

****

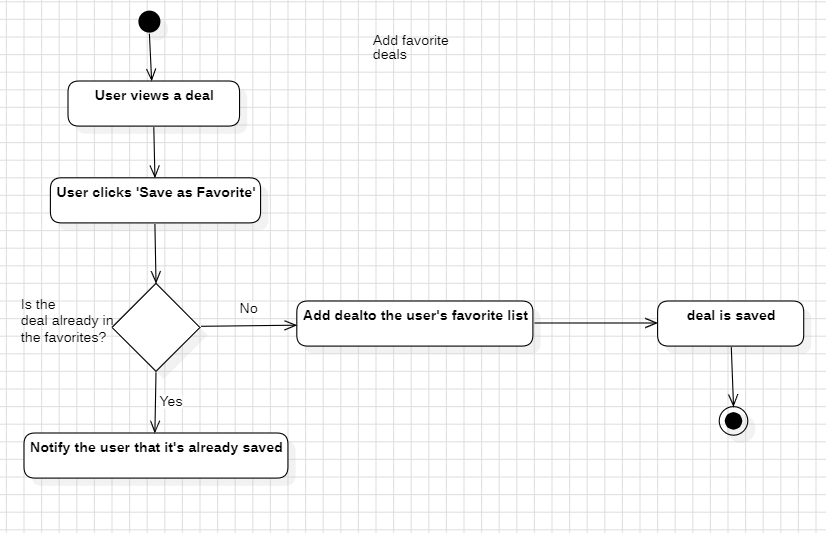
1. **Moniter user**

****

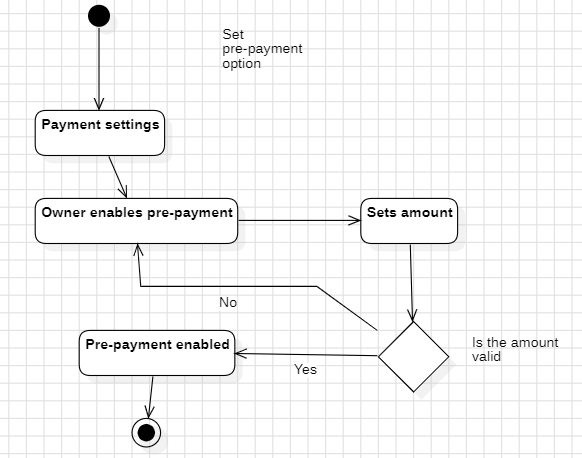
1. **Moniter restaurant**

****

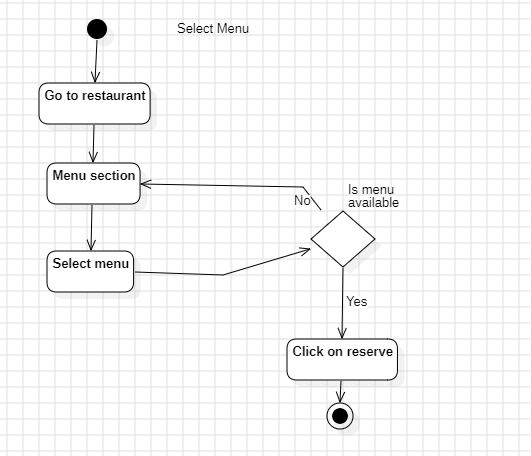
1. **Add favorite deals**

****

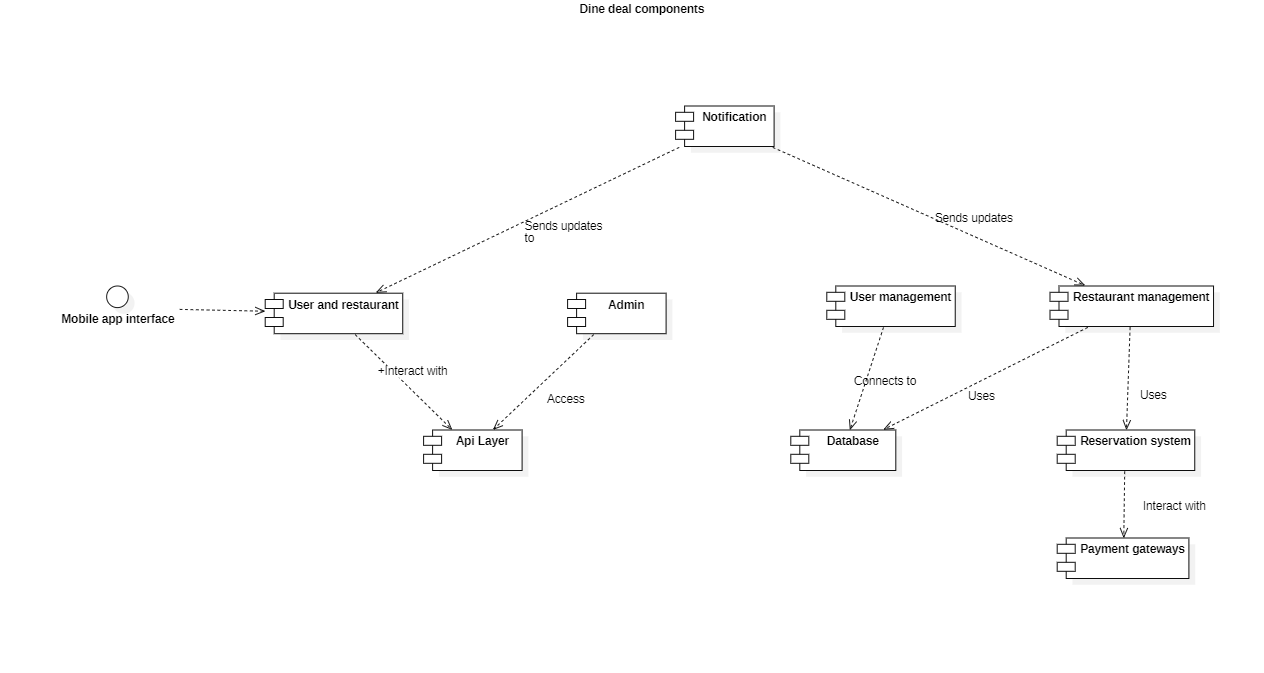
1. **Set pre-payment option**

****

1. **Select Menu**

****

**4.5 Component Diagram:**

****

**Chapter 5:**

**Implementation**

**Chapter 5:**

**Implementation**

#### 5.1 Endeavour:

* **5.1.1 Team**
  + Syed Kamran Shah
  + Safiullah Shahid
  + Muhammad Ismail
* **5.1.2 Work Breakdown Structure**
  + **Project Management**
    - 1.1 Work Breakdown Structure (WBS)
    - 1.2 Roles & Responsibility Matrix
  + **Reports / Documentation**
    - 2.1 Team Members and Project Proposal
    - 2.2 Project Proposal Document
      * 2.2.1 Opportunity and Stakeholders
      * 2.2.2 Existing Systems
      * 2.2.3 Problem Statement
      * 2.2.4 Proposed Solution
      * 2.2.5 Project Scope
        + 2.2.5.2 Guests
        + 2.2.5.3 Registered Users
        + 2.2.5.4 Restaurants
        + 2.2.5.5 Administrators
      * 2.2.6 Technologies
    - 2.4 Process Document
      * 2.4.1 Documentation
    - 2.5 Planning Document
      * 2.5.1 Problem the Software Will Solve
      * 2.5.2 Development Approach
      * 2.5.3 Primary Functions
      * 2.5.4 Order of Development
      * 2.5.5 Leadership Roles
      * 2.5.6 Team Member Responsibilities
    - 2.8 Final Documentation Introduction
    - 2.9 Literature / Market Survey
      * 2.9.1 Surveys
      * 2.9.2 Brainstorming
    - 2.10 Requirements Analysis
      * 2.10.1 Functional Requirements
      * 2.10.2 Non-Functional Requirements
      * 2.10.3 Stakeholder Requirements
    - 2.11 System Design
      * 2.11.1 Interface Design
        + 2.11.1.1 Figma app desgin
      * 2.11.2 Architectural Design
      * 2.11.3 Use Cases
      * 2.11.4 Component Diagram
      * 2.11.5 Deployment Diagram
      * 2.11.6 Activity Diagrams
    - 2.12 Implementation
    - 2.13 Testing & Performance Evaluation
      * 2.13.1 Unit Testing
      * 2.13.2 Integration Testing
  + **System**
    - 3.1 Development Environment
      * 3.1.1 IDE (Visual Studio Code)
      * 3.1.2 Version Control (Git)
      * 3.1.3 Server
      * 3.1.4 Database
        + 3.1.4.1 User Information
    - 3.2 Graphics
      * 3.2.1 App UI Design
        + 3.2.1.1 Features and Layouts
    - 3.3 APIs and Backend
      * 3.3.1 Restaurant Management APIs
      * 3.3.2 User Authentication APIs
      * 3.3.3 Reservation APIs
    - 3.4 Mobile App
      * 3.4.1 Frontend
        + 3.4.1.1 Main Page
        + 3.4.1.2 Browse Restaurants
        + 3.4.1.3 Reservation Management
        + 3.4.1.4 Deals and Offers
        + 3.4.1.5 Profile
        + 3.4.1.6 Notifications
      * 3.4.2 Backend
        + 3.4.2.1 API Integrations
        + 3.4.2.3 Report Generation
        + 3.4.2.4 Security and Authentication

**5.1.1.1 Roles and responsibilities:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **WBS #** | **WBS Deliverable** | **Activity #** | **Activity Description** | **Duration (Days)** | **Responsible Team Member(s) & Role(s)** |
| 1 | Project Management | 1.1 | Develop Work Breakdown Structure (WBS) | 2 | Syed Kamran Shah (Project Manager and UI/UX designer) |
| 1 | Project Management | 1.2 | Assign Roles and Responsibilities | 1 | Syed Kamran Shah (Project Manager) |
| 1 | Project Management | 1.3 | Set up Change Control System | 2 | Syed Kamran Shah (Project Manager) |
| 2 | Reports / Documentation | 2.1 | Create Team Members and Project Proposal | 3 | All members |
| 2 | Reports / Documentation | 2.2 | Proposal Document | 4 | Syed Kamran Shah (Project Manager) |
| 2 | Reports / Documentation | 2.3 | Stakeholder Interviews and Surveys | 3 | All members |
| 2 | Reports / Documentation | 2.4 | Finalize Literature and Market Survey | 5 | Syed Kamran and Ismail |
| 3 | System Design | 3.1 | Define Architectural Design | 4 | Syed Kamran Shah |
| 3 | System Design | 3.2 | Create Database Schema | 3 | Safiullah Shahid |
| 3 | System Design | 3.3 | Design Use Case Diagrams | 3 | Mohammad Ismail |
| 4 | Frontend Development | 4.1 | Build Restaurant and User Interface Components | 10 | Safiullah Shahid |
| 4 | Frontend Development | 4.2 | Develop Reservation and Deals Modules | 12 | Mohammad Ismail |
| 4 | Backend Development | 4.3 | Develop API Endpoints for Reservations and Users | 15 | Syed Kamran Shah |
| 4 | Backend Development | 4.4 | payment | Will do | All members |
| 5 | Testing and Evaluation | 5.1 | Perform Unit Testing | 5 | Safiullah Shahid |
| 5 | Testing and Evaluation | 5.2 | Conduct Integration Testing | 4 | Safiullah Shahid |
| 5 | Testing and Evaluation | 5.3 | Conduct User Acceptance Testing | 3 | All Team Members |
| 6 | Maintenance | 6.1 | Monitor System Performance | Ongoing | All Team Members |
| 6 | Maintenance | 6.2 | Provide Bug Fixes and Updates | Ongoing | All Team Members |

**5.2 Components, Libraries, and APIs**

Key technologies and libraries used in the development include:

Frontend Framework: Flutter for cross-platform mobile application development.

Backend Framework: Node.js with Express.js for API development.

Database: MongoDB for managing user, restaurant, and reservation data.

Payment: API for secure transactions.

**5.3 IDE, Tools, and Techniques**

The following tools and techniques facilitated the development process:

IDE: Visual Studio Code for coding and debugging.

Version Control: Git for collaboration and source code management.

Testing Frameworks: Postman for API testing.

**5.4 Best Practices and Coding Standards**

The development adhered to industry best practices to ensure scalability, maintainability, and performance:

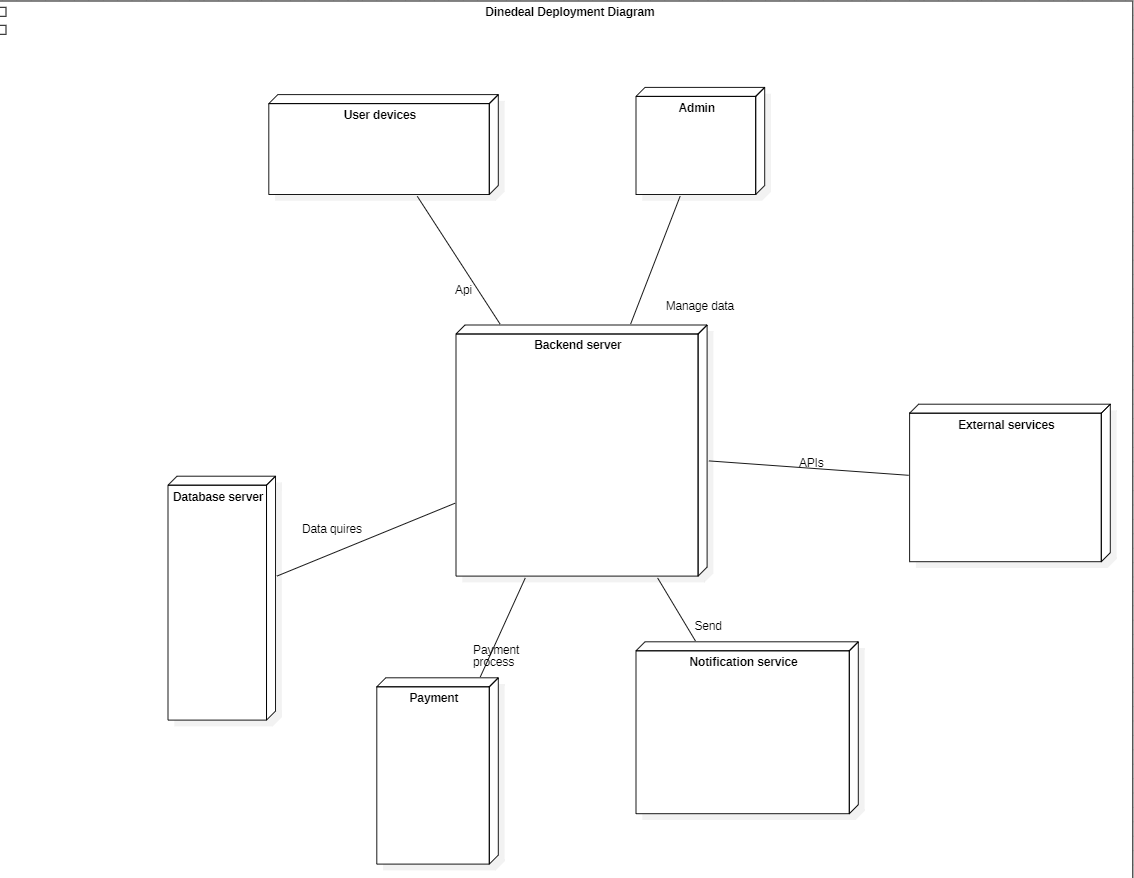
Modular Code Structure: To allow easy updates and new feature integration.

Responsive Design: Optimized for mobile devices with varying screen sizes.

Secure Development: Encryption for sensitive data.

Documentation: Comprehensive inline documentation.

**5.5 Deployment Diagram**



**5.6 Summary**

The implementation phase established a solid foundation for the DineDeal application, ensuring all core functionalities were operational. The use of modern tools and methodologies streamlined the development process and delivered a robust, scalable, and user-friendly platform.

**Chapter 6:**

**Testing and Evaluation**

**Chapter 6:**

**Testing and Evaluation**

**6.1 Introduction:**

For assuring the quality of the system, testing is the most essential step for assuring the quality of the system. The purpose of testing is to find out the system errors and bugs in the system. So, in this chapter we will discuss testing of our Dinedeal Mobile app. We will create the test cases to check that the system is working correctly while completing the necessary requirements.

**6.2 List of Test Cases:**

**6.2.1: Register user**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test**  **Case**  **ID** | **Scenario** |  | **ECP** | | **Expected Output** |
| **Username** | **Email** | **Password** |  |
| 1 | User enter data. | Mohammad Ismail | Ismail123@gmail.com | {A, B, C, ...}  {a, b, c, …}  {1, 2, 3, …} | User registered successfully. |
| 2 | User entered invalid data. | $$234##77 | Yqwu87234@gmail.com | {!, @, #, …} | User not registered. |

**6.2.2: Login user**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test**  **Case**  **ID** | **Scenario** | **ECP** | | **Expected Output** |
| **Email** | **Password** |  |
| 1 | User enter valid credentials. | Ismail123@gmail.com | {A, B, C, ...}  {a, b, c, …} | User login successfully. |
| 2 | User entered invalid credentials. | Yqwu87234@gmail.com | {1, 2, 3, …}  {!, @, #, …} | User not logged in. |

**6.2.3: Edit Profile**

**6.2.4: Make Reservations**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test**  **Case**  **ID** | **Scenario** |  | **ECP** | | **Expected Output** |
| **Date** | **Time** | **Table** |  |
| 1 | User select available date, time and table. | 12/12/2024 | 4:30pm | Table 9 | Table reserved. |
| 2 | User selected unavailable slots. | 19/12/2024 | 4:50pm | Table 6 | Table not reserved. |

**6.2.5: Cancel Reservation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test**  **Case**  **ID** | **Scenario** | **ECP** | **Expected Output** |
| **Reservation** |  |
| 1 | User cancel table within 10 min after booking. | Reserved table 9 | Reservation canceled. |
| 2 | User cancel table after 10 mins of reservation. | Reserved table 3 | Can’t cancel reservation. |

**6.2.6: Set Discounts**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test**  **Case**  **ID** | **Scenario** | **ECP** | **Expected Output** |
| **Amount** |  |
| 1 | Restaurant enter valid discount amount. | 200 Rupees | Discount added. |
| 2 | Restaurant enter invalid amount. | 10000 Rupees | Discount not added. |

**6.2.7: Set Pre-Payment Amount**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test**  **Case**  **ID** | **Scenario** | **ECP** | **Expected Output** |
| **Pre-payment** |  |
| 1 | Restaurant enter valid amount. | 200 Rupees | Pre-payment is available. |
| 2 | Restaurant enter invalid amount. | 10000 Rupees | Pre-payment not added. |

**6.2.8: Update Location**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test**  **Case**  **ID** | **Scenario** | **ECP** | **Expected Output** |
| **Location** |  |
| 1 | User select available location. | Islamabad | Location is updated. |
| 2 | User select invalid location. | Australia | Location is unavailable. |