Software Engineering

Software Design Specification

Parking Management System

4 April 2024

Group Members:
IFI2022023
IFI2022008
IIT2022257
IIT2022265

Contents

1 Introduction

1.1 Purpose of this document

The purpose of this document is to provide a comprehensive software design specification for the development of a web-based parking management system. This system aims to facilitate users in locating available parking spaces near their current location, as well as enabling them to reserve a parking space in advance. Additionally, the system will provide administrative functionalities for parking lot owners or managers to manage bookings, monitor parking lot occupancy, and facilitate vehicle check-ins and check-outs.

1.2 Scope of the development project

The scope of this development project encompasses the design and implementation of a user-friendly web application that allows users to search for nearby parking lots based on their location, view parking availability in real-time, and make reservations for parking spaces. The system will also include an administrative interface to manage parking lot data, monitor bookings, and facilitate vehicle entry and exit processes.

1.3 Definitions, acronyms, and abbreviations

- **CMS** - Content Management System
- **GPS** - Global Positioning System
- **HTML** - Hypertext Markup Language
- **IEEE** - Institute of Electrical and Electronics Engineers
- **API** - Application Programming Interface
- **UI** - User Interface

1.4 References

- IEEE Std 1016-2009 - IEEE Standard for Information Technology - Systems Design - Software Design Descriptions

- IEEE Std 830-1998 - IEEE Recommended Practice for Software Requirements Specifications

1.5 Overview of document

This document provides a detailed software design specification following the IEEE standard format. It outlines the conceptual architecture, logical architecture, execution architecture, design decisions, and pseudocode for components of the parking management system. Additionally, it includes appendices containing relevant diagrams, tables, and additional information to support the software design and development process.

2 Conceptual Architecture/Architecture Diagram

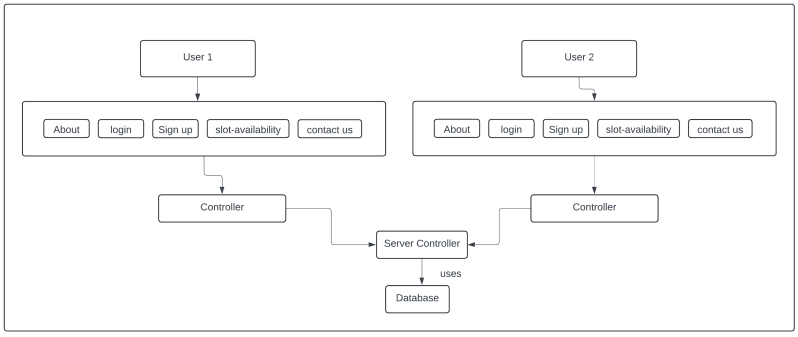


Figure 1: Architecture Diagram 1

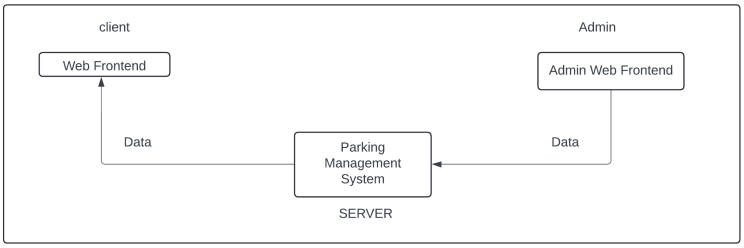


Figure 2: Architecture Diagram 2

2.1 Structure and relationships

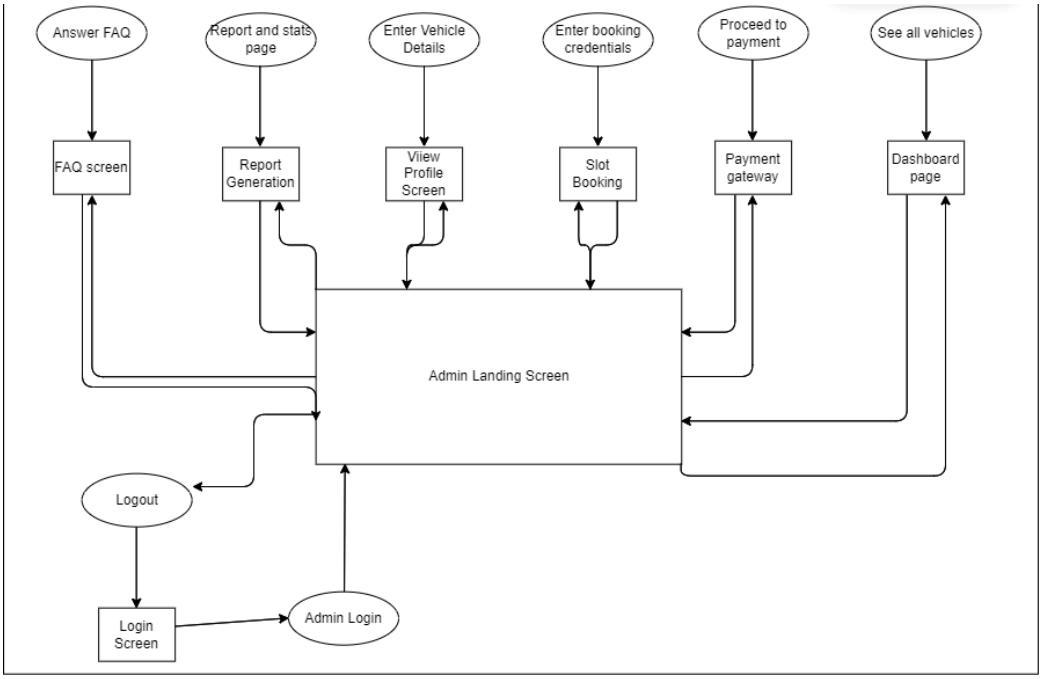


Figure 3: Admin's Side

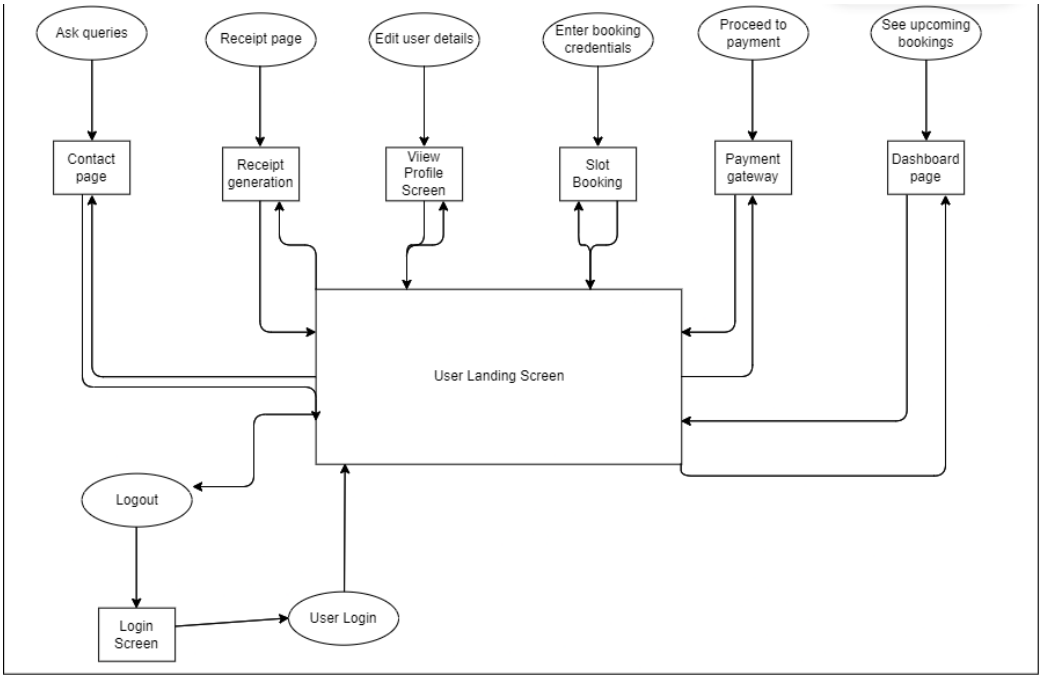


Figure 4: User's Side

2.2 User interface issues

User A

- The user interface should be intuitive and easy to navigate, following common conventions used in mobile applications.
- Links between screens should use descriptive and easily understandable labels such as "Search Parking", "Book Space", and "Admin Dashboard".
- Consistency in UI elements placement should be maintained, with important links appearing prominently on the screen for easy access.
- Since User A is proficient with technology, the UI can incorporate advanced features like real-time updates on parking availability and interactive maps for searching parking locations.

User B

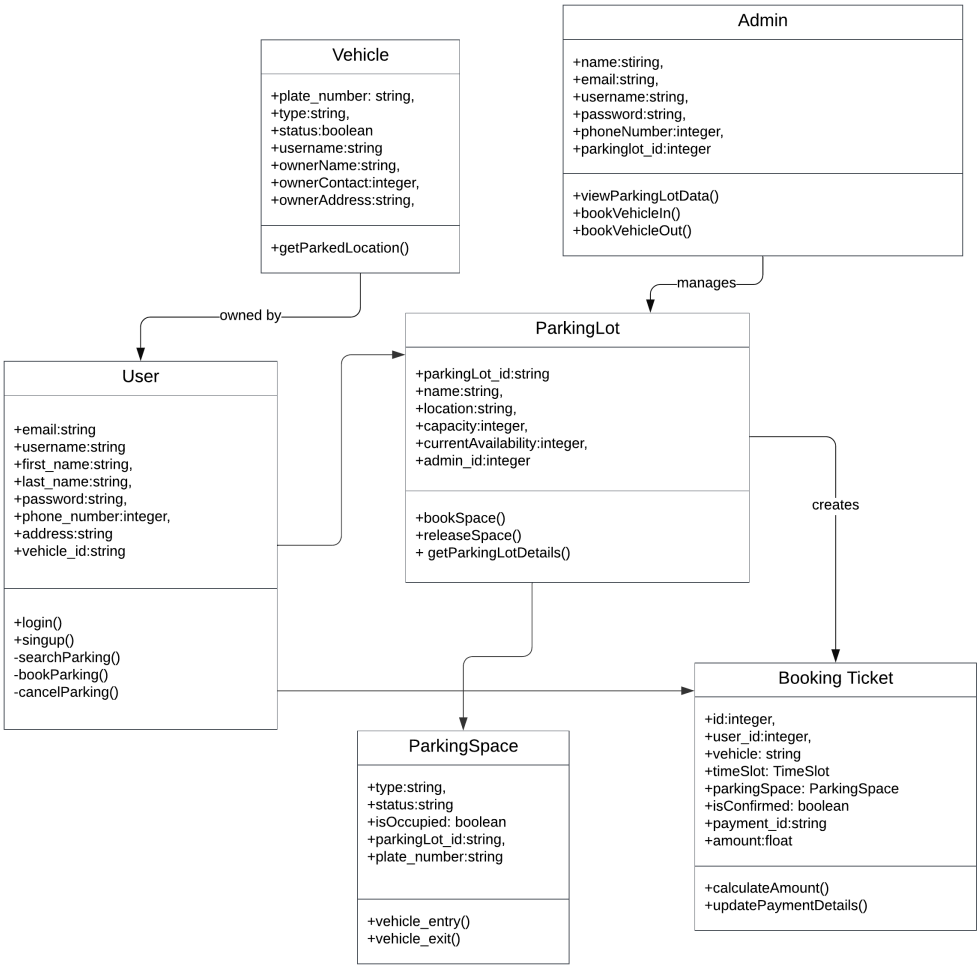
- Clear on-screen directions should be provided to guide the user through the process of searching and booking parking spaces.
- The UI should be designed to grab the attention of the user and make it clear that the application is easy to use.
- On-click task implementation should be emphasized, allowing the user to perform actions with minimal effort.
- The color combination should be chosen to ensure readability in various lighting conditions, including direct sunlight.
- Text size should be reasonably large to improve readability, especially for users who may have difficulty with smaller text.

User C

- The UI should be designed to be easily operable by faculty members who are well-versed in using mobile applications.
- Although the primary users are students and administrators, faculty members should still be able to access relevant information such as parking availability and usage statistics.
- The UI should facilitate easy access to information such as the status of parking lots, recent collaborations, and network expansion efforts of the university.
- Information retrieval should require minimal effort, with important updates and notifications accessible with just a few clicks.

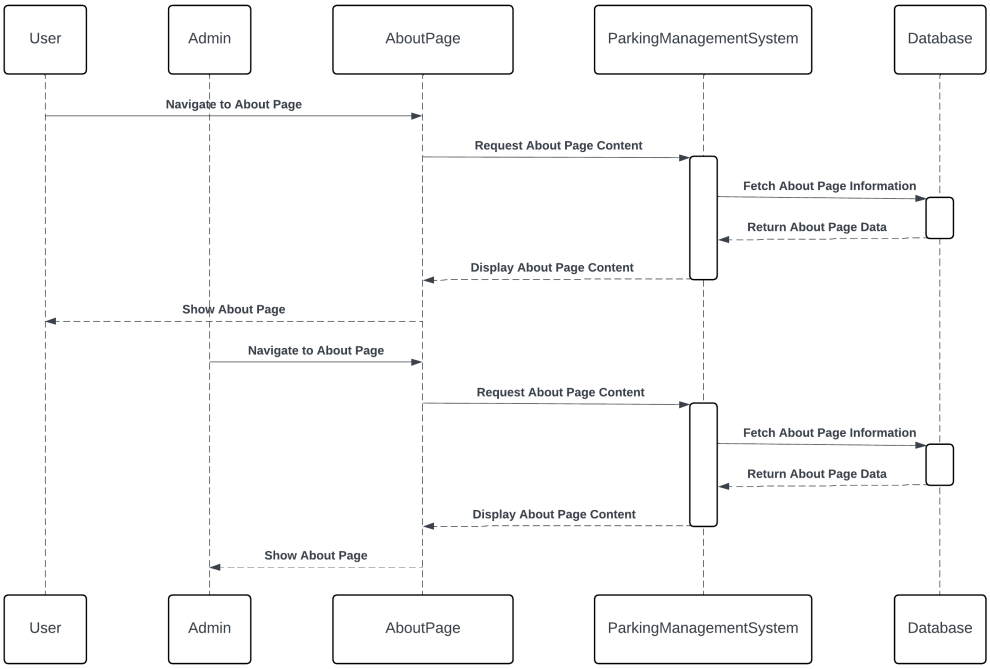
3 Logical Architecture (Class Diagram, Sequence Diagram, State Diagram)

Class Diagram

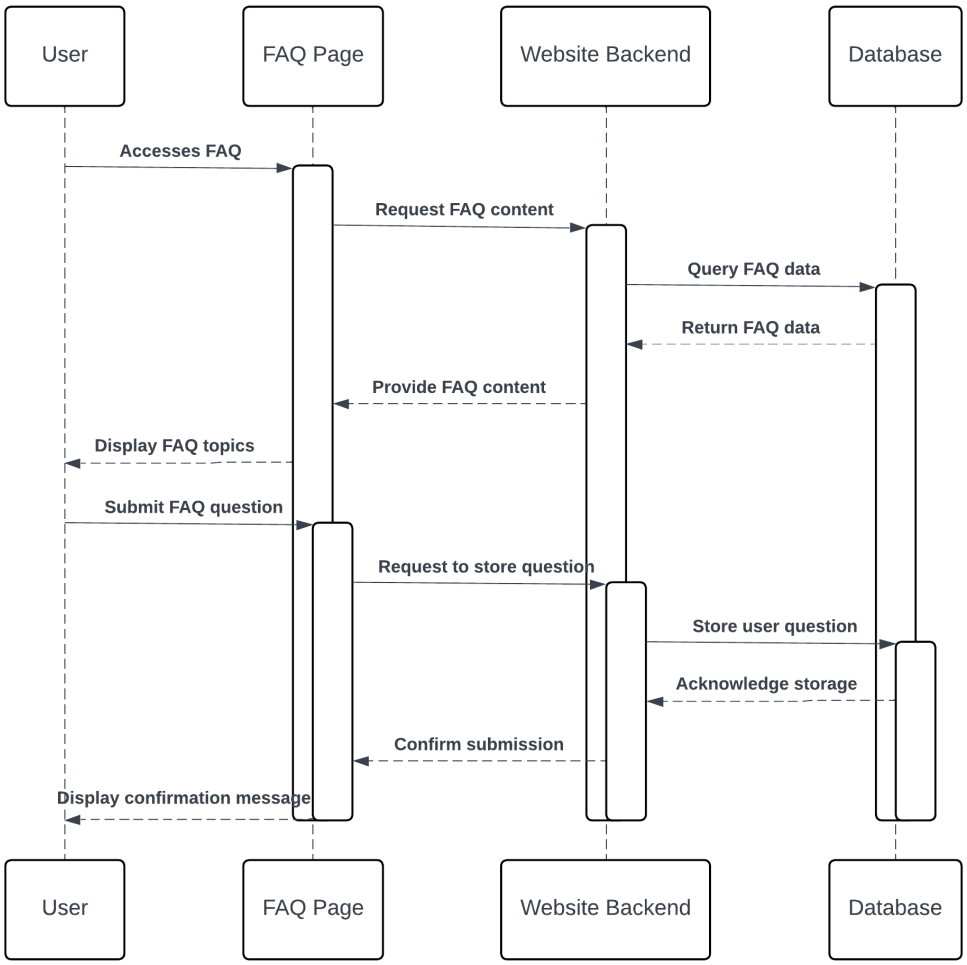


Sequence Diagrams

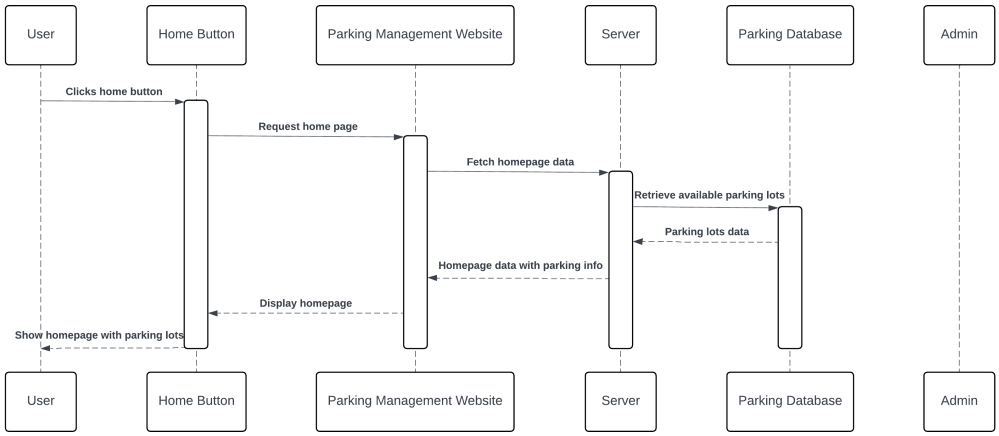
Sequence Diagram: About Page



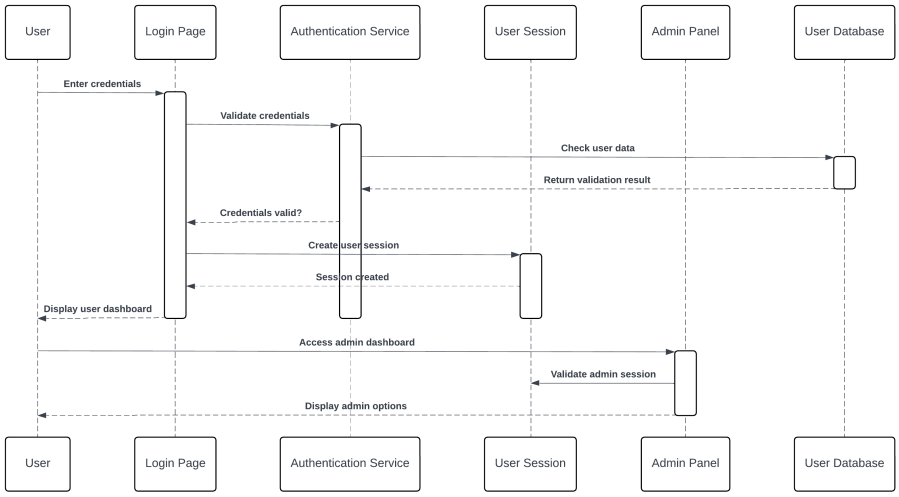
Sequence Diagram: FAQ



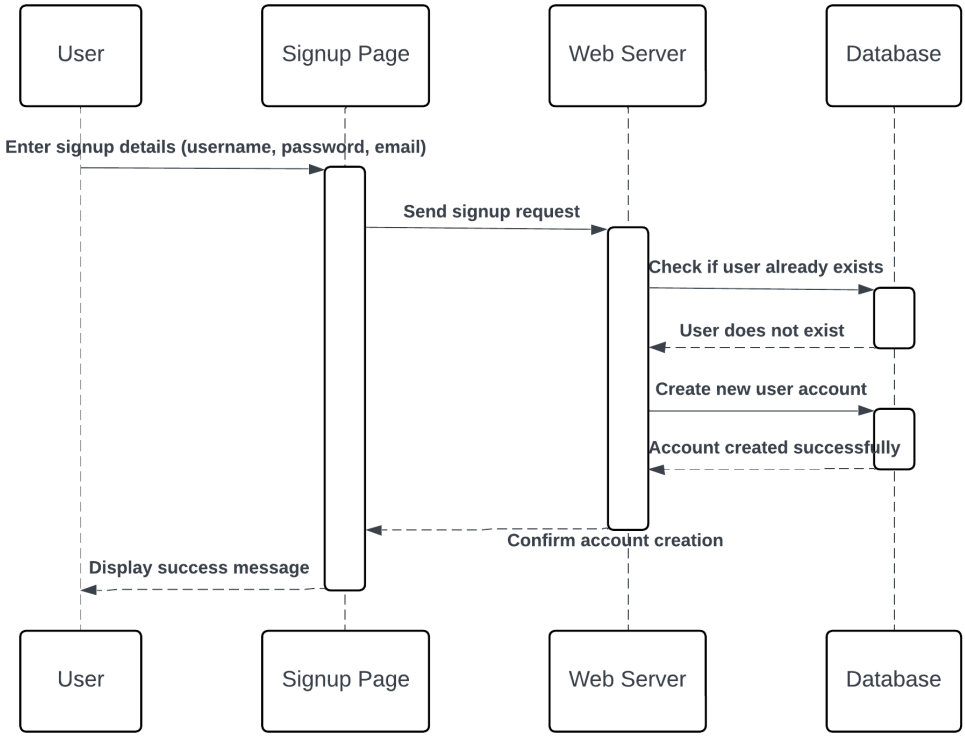
Sequence Diagram: Home Page



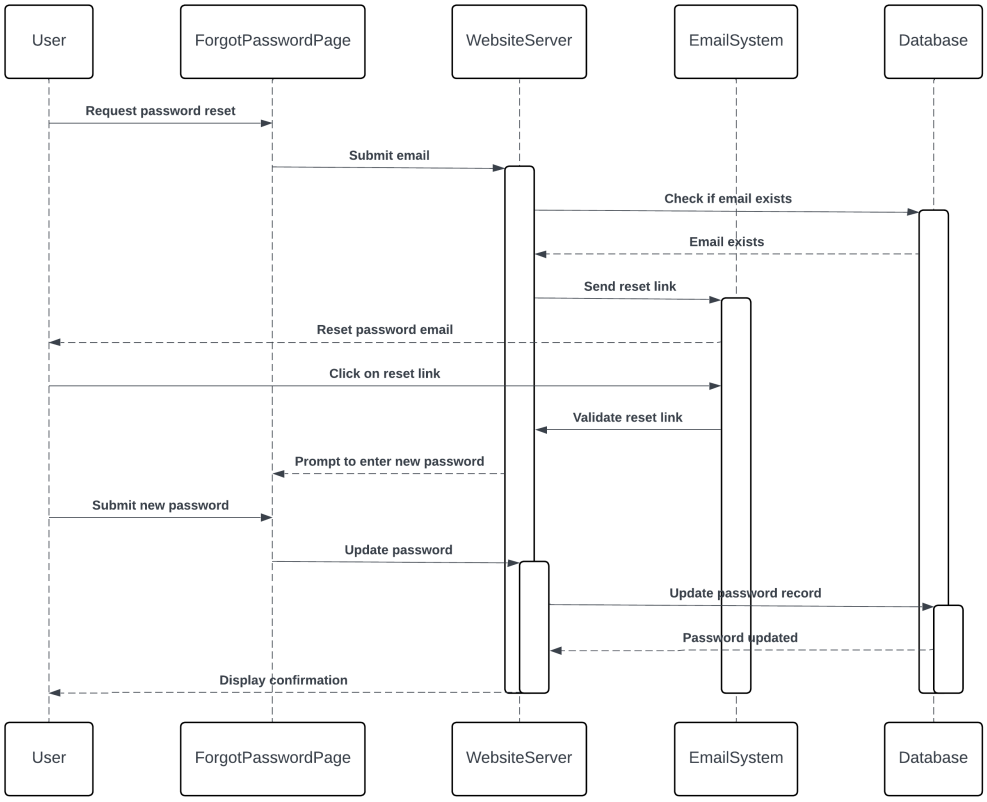
Sequence Diagram: LogIn



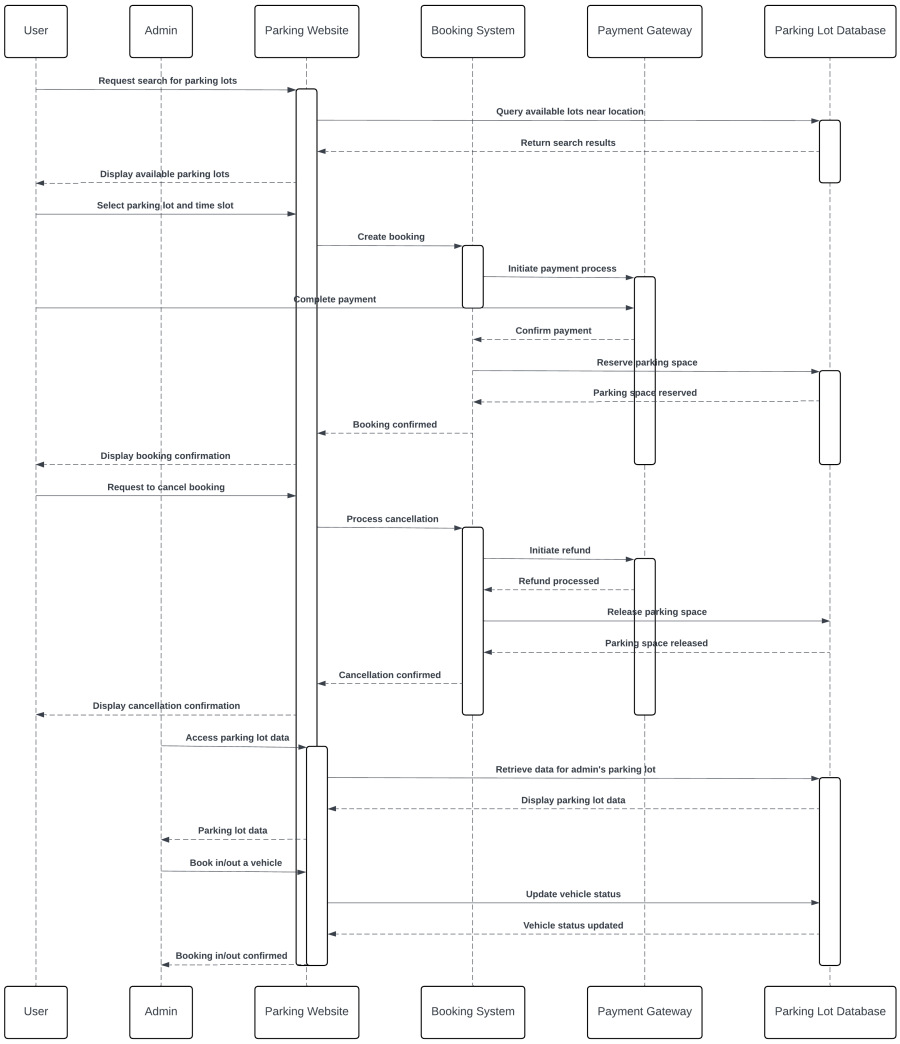
Sequence Diagram: SignUp



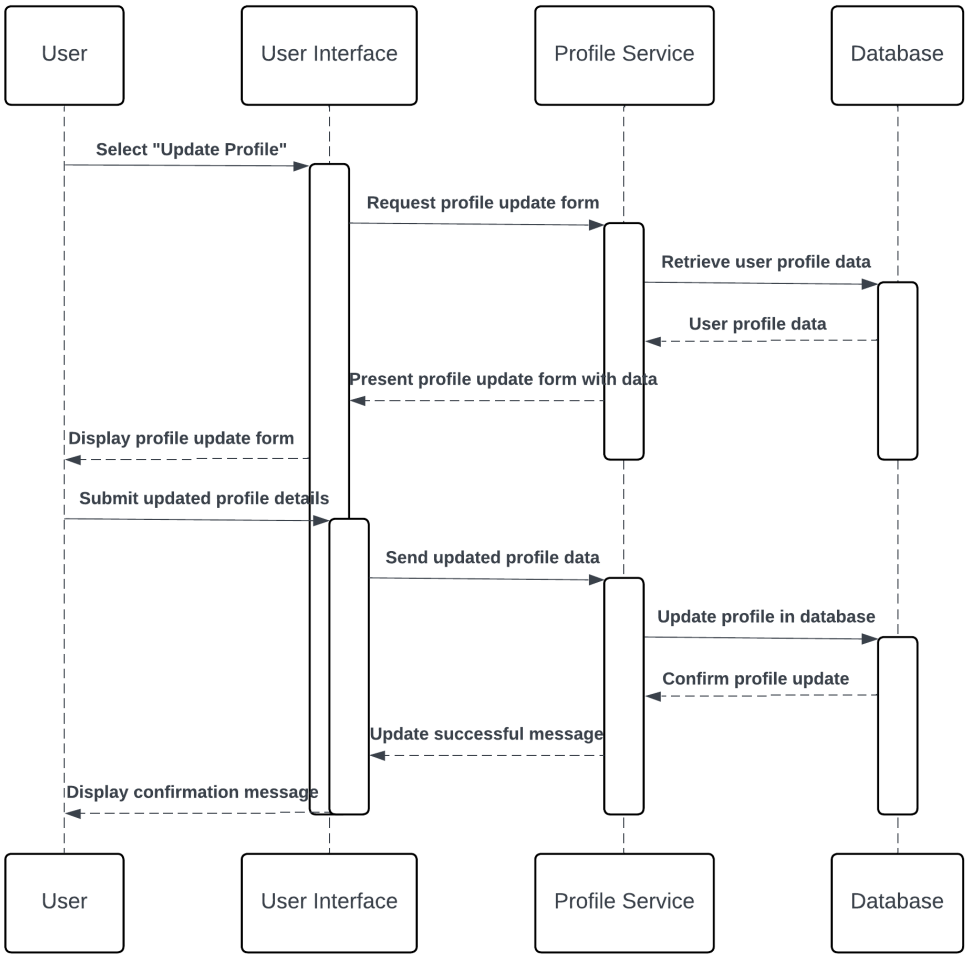
Sequence Diagram: Forgot Password



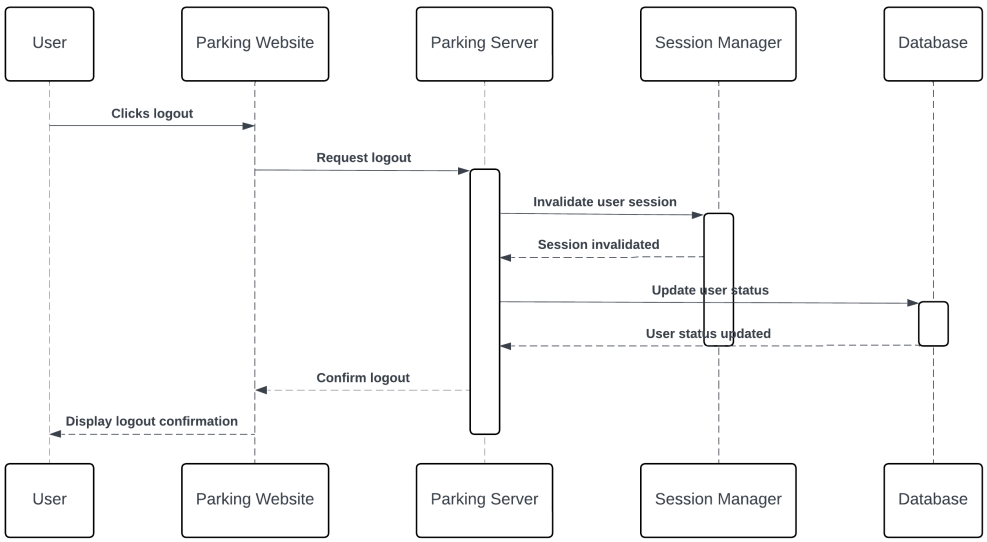
Sequence Diagram: Booking



Sequence Diagram: Update Profile

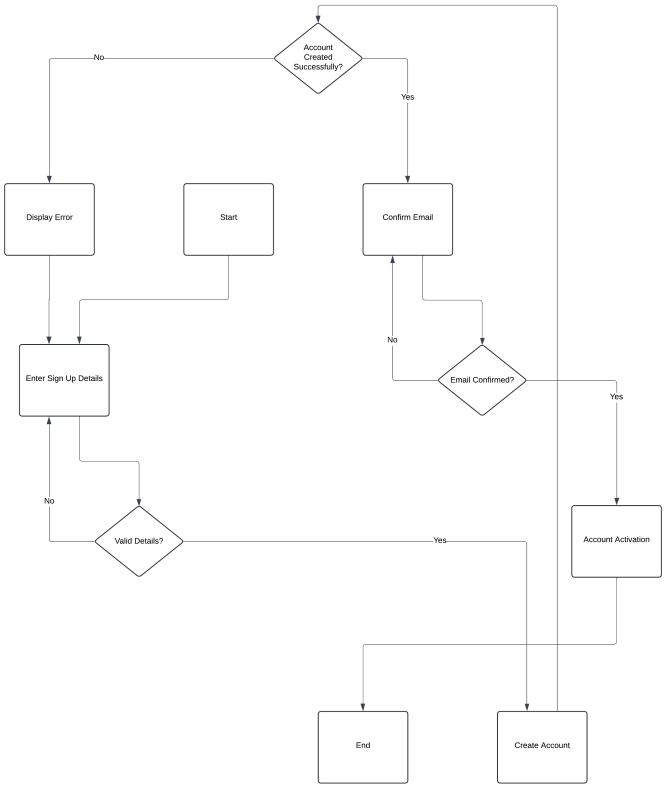


Sequence Diagram: Log Out

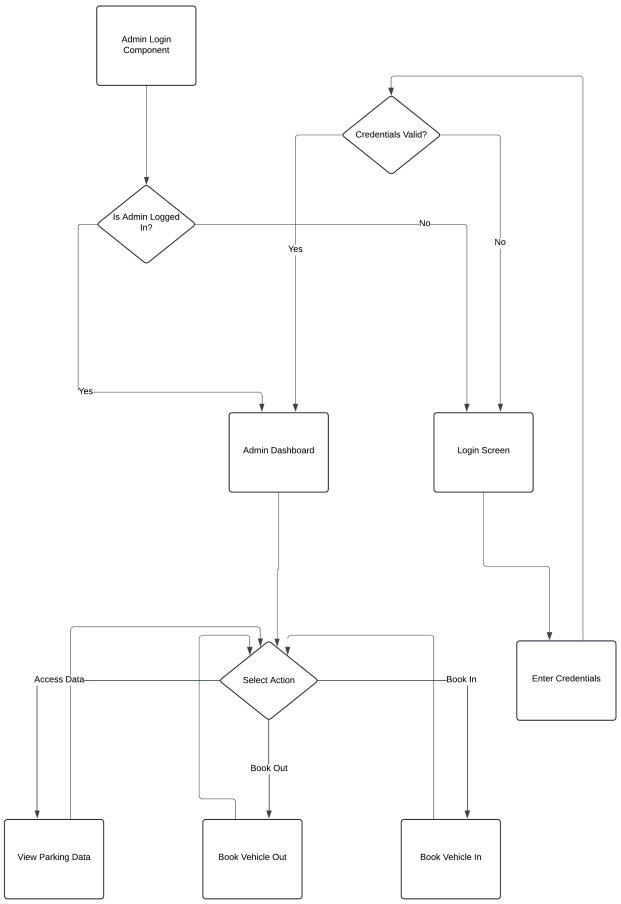


State Diagrams

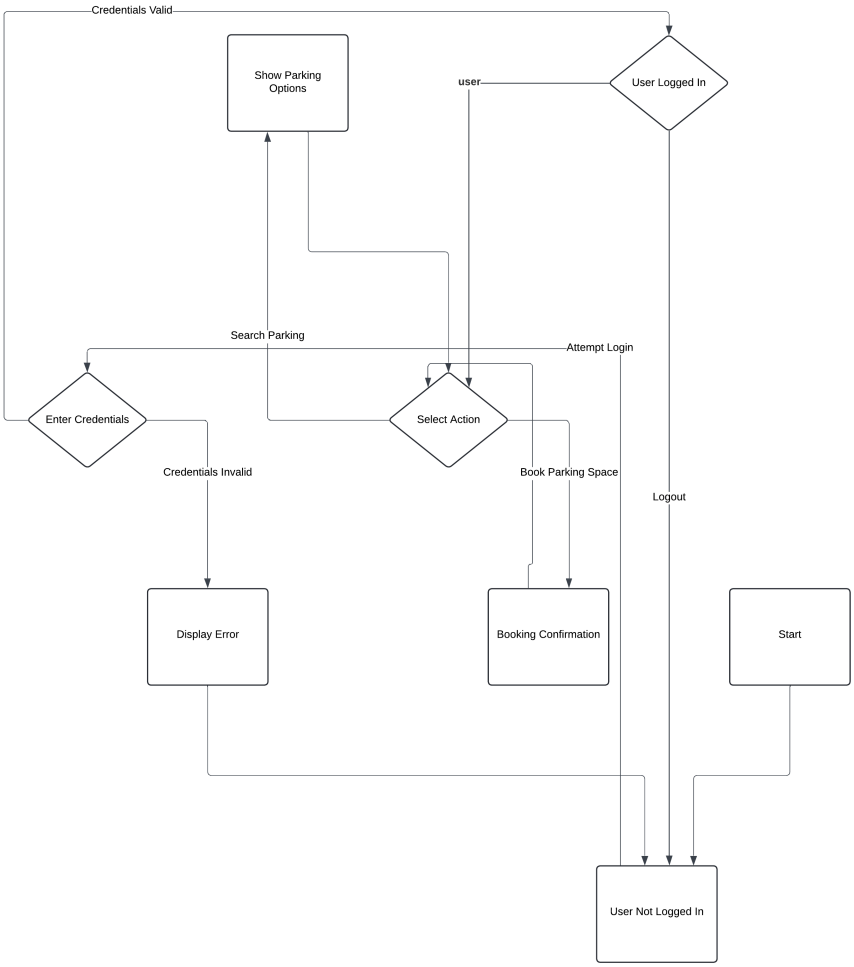
State Diagram: Sign Up



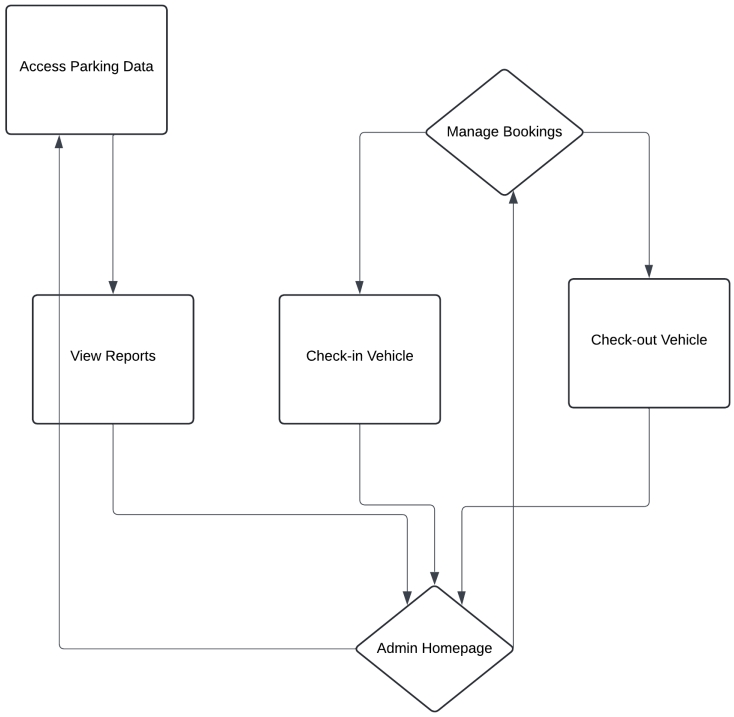
State Diagram: Admin Login



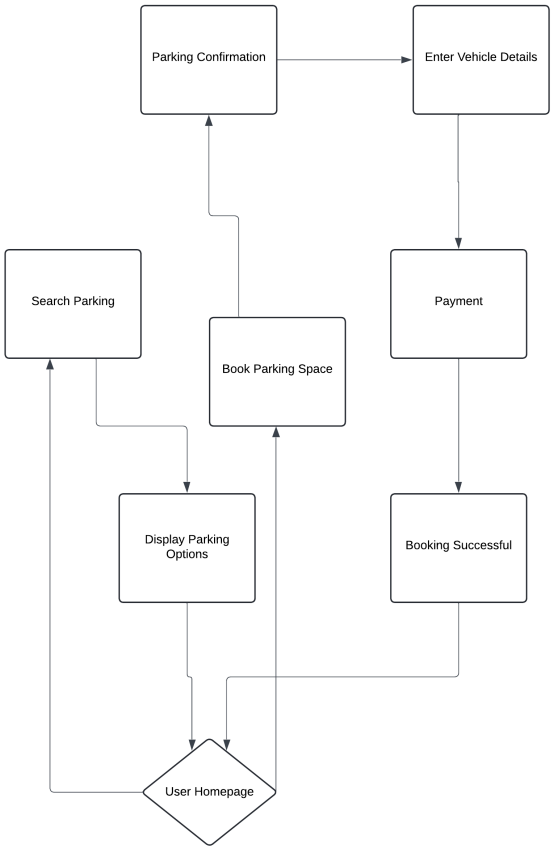
State Diagram: User Login



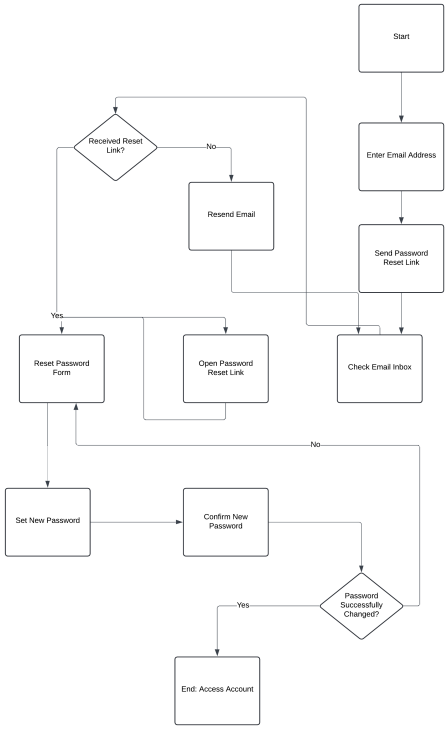
State Diagram: Admin Dashboard



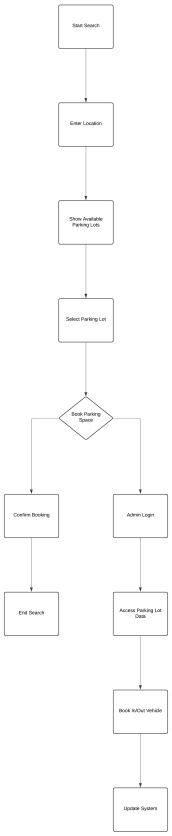
State Diagram: User Dashboard



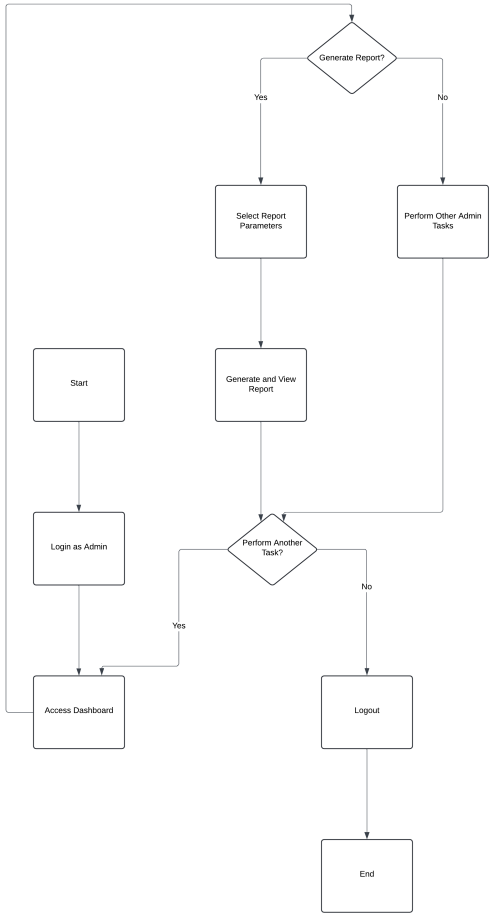
State Diagram: Forgot Password



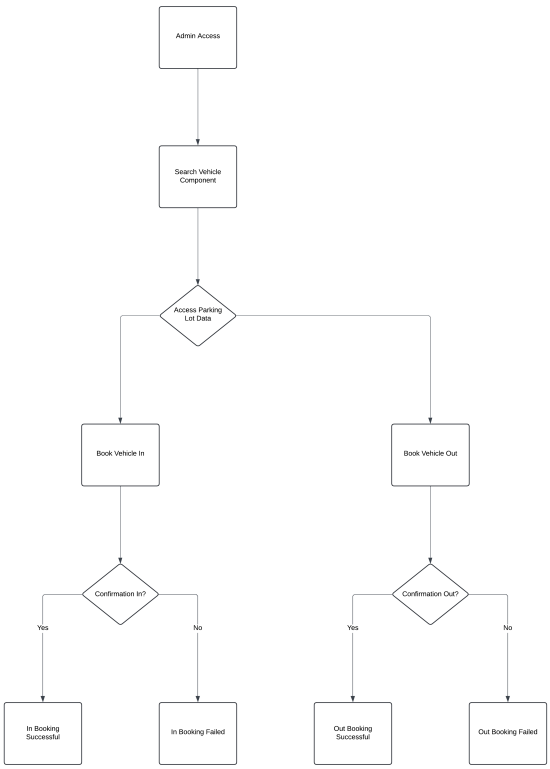
State Diagram: Search Parking Lot



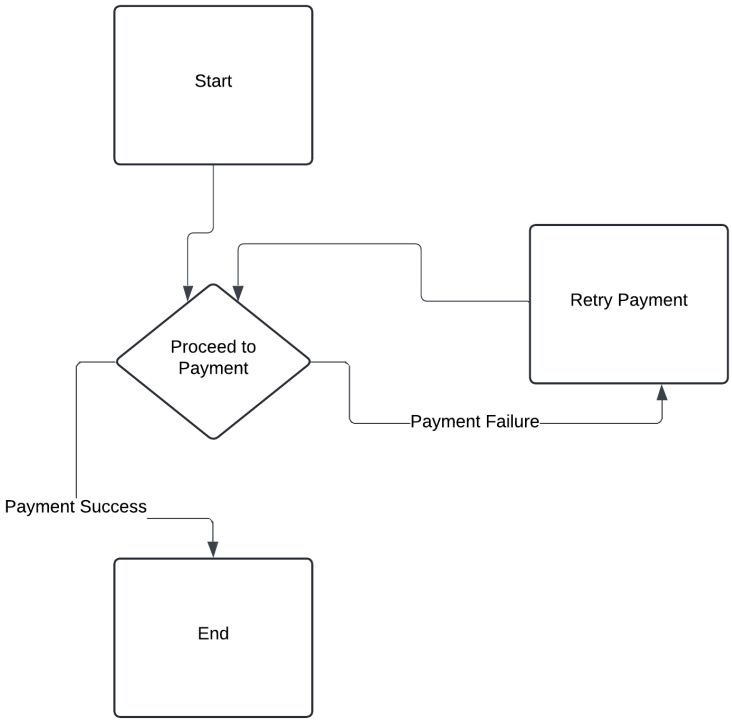
State Diagram: Generate Report



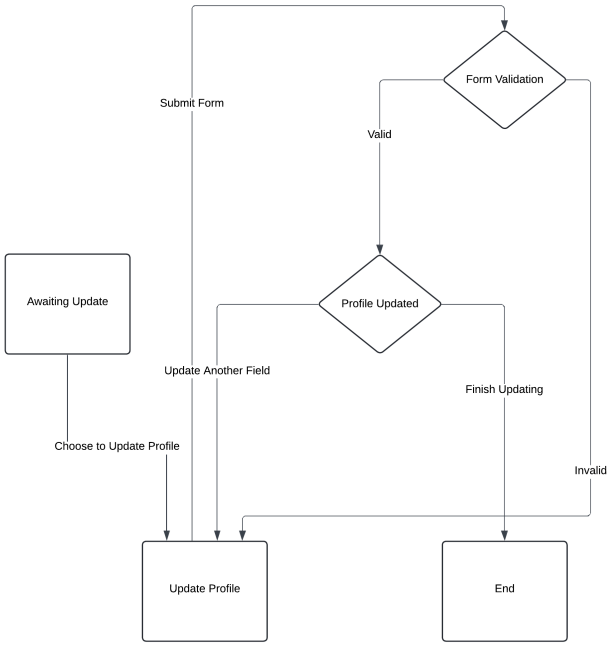
State Diagram: Vehicle In/Out



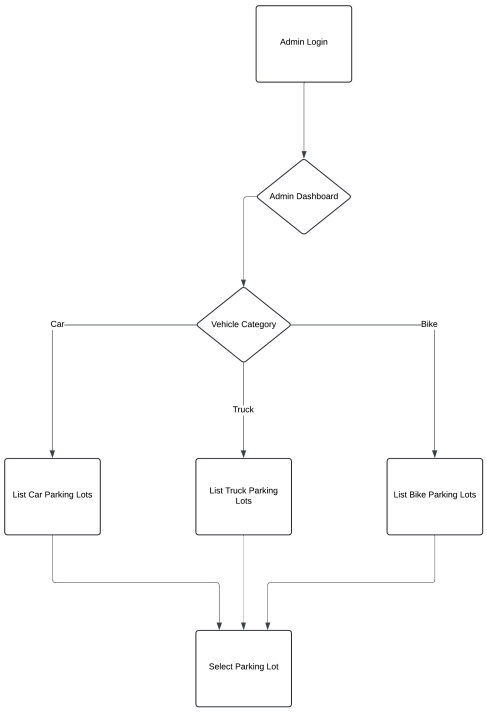
State Diagram: Payment



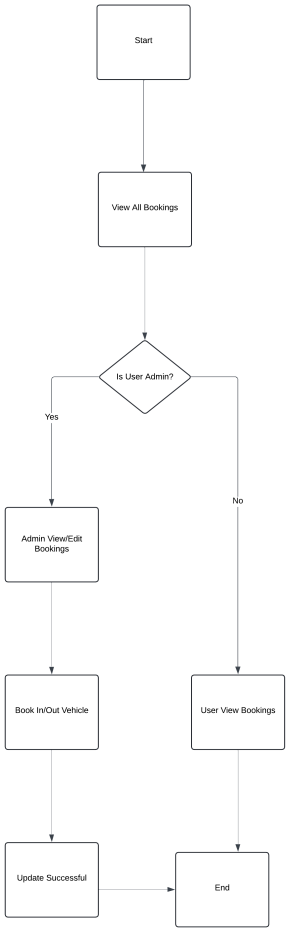
State Diagram: Update Profile



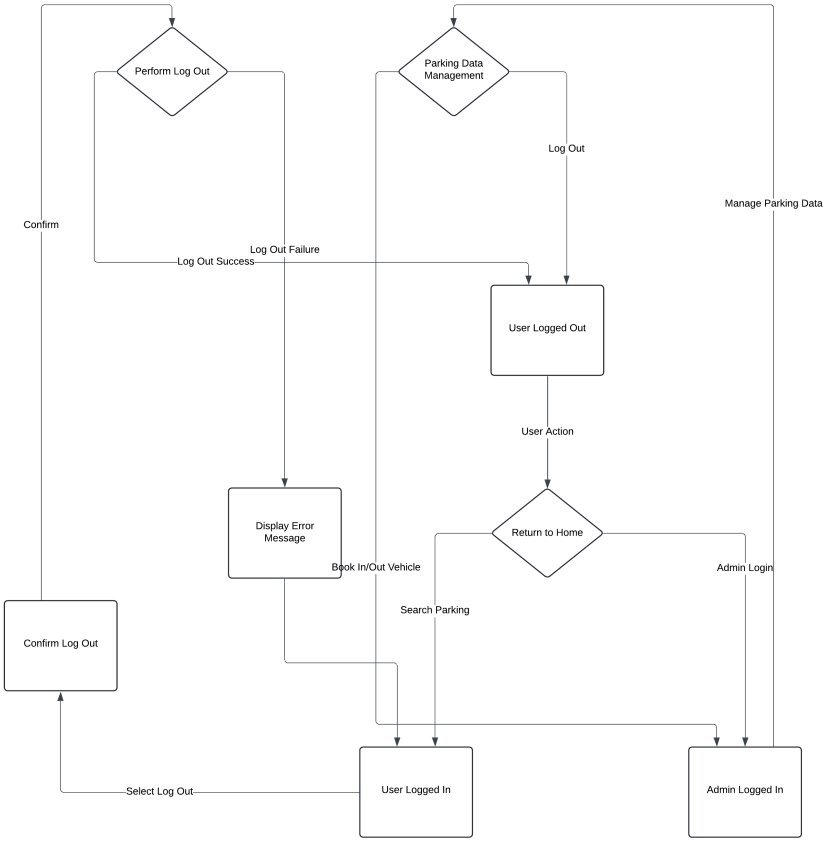
State Diagram: Vehicle Category



State Diagram: View All Bookings



State Diagram: Log Out



3.1 Logical Architecture Description

The logical architecture of the parking management system outlines the high-level structure and organization of its components and interactions. It focuses on the conceptual design of the system, abstracting away implementation details to provide a clear understanding of how different parts of the system collaborate to achieve its objectives. This description defines the logical relationships between modules, subsystems, and external interfaces, providing a blueprint for system development and maintenance.

3.1.1 Class Diagram Explanation:

Here is the explanation of the class diagram:

1. The `User` class has attributes such as email, username, first name, last name, password, phone number, address, and vehicle id. It has methods like `login` and `signup`.

2. The `Vehicle` class is associated with the `User` class and has attributes like plate number, type, status, username, owner name, owner contact, and owner address. It has methods like `searchParking`, `bookParking`, and `cancelParking`.

3. The `ParkingLot` class has attributes like parkingLot_id, name, location, capacity, current availability, and admin_id. It has methods like `bookSpace`, `releaseSpace`, and `getParkingLotDetails`.

4. The `ParkingSpace` class is associated with the `ParkingLot` class and has attributes like type, status, isOccupied, parkingLot_id, and plate_number.

5. The `Admin` class has attributes like name, email, username, password, phone number, and parkinglot_id. It has methods like `viewParkingLotData`, `bookVehicleIn`, and `bookVehicleOut`. The `Admin` class manages the `ParkingLot` and creates `ParkingSpace`.

6. The `BookingTicket` class has attributes like id, user_id, vehicle, timeSlot, parkingSpace, isConfirmed, payment id, and amount. It has methods like `calculateAmount` and `updatePaymentDetails`.

The relationships between the classes are represented by lines. The "owned by" line between `User` and `Vehicle` signifies that a user owns a vehicle. The "manages" and "creates" lines from `Admin` to `ParkingLot` and `ParkingSpace` respectively signify that an admin manages a parking lot and creates parking spaces.

3.1.2 Sequence Diagram:

A sequence diagram illustrates the flow of interactions between various components or actors within the parking management system over time. It shows the sequence of messages exchanged between objects or actors, depicting the order in which actions occur and how they are interrelated. Sequence diagrams help visualize the dynamic behavior of the system, including user interactions, system responses, and data exchanges. They are valuable for understanding the system's functionality, identifying potential bottlenecks, and refining the system's design.

1. **FAQ Page:**

   This sequence diagram illustrates the interaction between a user and the FAQ page of the parking management website. The user initiates the interaction by accessing the FAQ page through the website. The website server retrieves the FAQ data from the database and sends it to the user's browser. The user can then view the FAQ content directly on the website.

2. **Home Page:**

   This sequence diagram depicts the interaction between a user and the home page of the parking management website. The user accesses the home page, and the website server retrieves the relevant data from the database to populate the page. The user can navigate through the home page and interact with various elements, such as buttons or links, to access parking management features or other sections of the website.

3. **Accessing Login Page:**

   The user initiates the interaction by accessing the login page of the parking management website.

4. **Providing Credentials:**

   The user provides their username and password on the login page.

5. **Verifying Credentials:**

   The authentication service receives the user's credentials and verifies them against the data stored in the database. If the credentials are valid, the authentication service proceeds to authenticate the user.

6. **Establishing User Session:**

   Upon successful authentication, the authentication service establishes a user session, granting access to the admin panel or user dashboard based on the user's role.

7. **Signing Up for an Account:**

   The user has the option to sign up for a new account if they don't have one already. The signup process involves providing necessary details such as username, email, and password.

8. **Resetting Password:**

   If the user forgets their password, they can initiate the password reset process. The user provides their email address, and a reset password link is sent to their email.

9. **Making a Booking:**

   The user can make a booking for a parking space by selecting the desired parking lot, date, and time slot. The booking request is sent to the website server, which processes it and updates the database accordingly.

10. **Updating Profile Information:**

    The user has the ability to update their profile information such as username, email, or password. The updated information is sent to the website server, which in turn updates the user's data in the database.

11. **Log Out:**

Finally, the user can choose to log out of the system, terminating their session .The logout request is sent to the website server, which clears the user's session data, effectively logging them out of the system.

3.1.3 State Diagram:

A state diagram, also known as a state machine diagram, represents the different states that an object or system can transition through during its lifetime and the events that trigger those transitions. In the context of the parking management system, a state diagram could depict the various states that a parking space can be in (e.g., available, occupied, reserved) and the events that cause transitions between these states (e.g., vehicle arrival, reservation expiration). State diagrams provide a visual representation of system behavior, helping to model complex state-dependent logic and ensure system correctness and robustness.

1. **Sign Up:**

   This state diagram illustrates the different states and transitions involved in the sign-up process for new users on the parking management system. It includes states such as providing user details, verifying information, and completing the registration.

2. **Admin Login:**

   This state diagram represents the states and transitions involved in the login process for administrators accessing the parking management system. It includes states such as providing admin credentials, authentication, and accessing the admin dashboard.

3. **User Login:**

   This state diagram depicts the states and transitions for user login on the parking management system. It includes states such as providing user credentials, authentication, and accessing the user dashboard.

4. **Admin Dashboard:**

   This state diagram outlines the various states and transitions within the admin dashboard of the parking management system. It includes states such as viewing reports, managing parking lots, and accessing user data.

5. **User Dashboard:**

   This state diagram illustrates the states and transitions within the user dashboard of the parking management system. It includes states such as searching for parking lots, making bookings, and viewing booking history.

6. **Forgot Password:**

   This state diagram represents the states and transitions involved in the password reset process for users who have forgotten their passwords. It includes states such as providing email for password reset, generating reset link, and updating password.

7. **Search Parking Lot:**

   This state diagram depicts the states and transitions involved in searching for available parking lots on the parking management system. It includes states such as entering search criteria, retrieving search results, and selecting a parking lot.

8. **Generate Report:**

   This state diagram illustrates the states and transitions involved in generating reports on the parking management system. It includes states such as selecting report parameters, generating report data, and viewing generated reports.

9. **Vehicle In/Out:**

   This state diagram represents the states and transitions involved in the process of vehicles entering and exiting parking lots. It includes states such as vehicle arrival, parking space allocation, vehicle departure, and parking space release.

10. **Payment:**

    This state diagram outlines the states and transitions involved in the payment process for parking bookings on the parking management system. It includes states such as selecting payment method, processing payment, and receiving payment confirmation.

11. **Update Profile:**

    This state diagram depicts the states and transitions involved in updating user profiles on the parking management system. It includes states such as providing updated profile information, validating changes, and saving updated profile data.

12. **Vehicle Category:**

    This state diagram illustrates the states and transitions related to selecting vehicle categories during the booking process on the parking management system. It includes states such as choosing vehicle type, specifying vehicle details, and confirming selection.

13. **View All Bookings:**

    This state diagram represents the states and transitions involved in viewing all bookings on the parking management system. It includes states such as accessing booking records, filtering bookings, and viewing booking details.

14. **Log Out:**

    This state diagram outlines the states and transitions involved in the log out process for users on the parking management system. It includes states such as initiating log out, confirming log out, and terminating user session.

3.2 User Class

Attributes:

- email: string
- username: string
- first name: string
- last name: string
- password: string
- phone_number: integer

- address: string
- vehicle id: string

Description:

The User class represents individuals who interact with the parking management system. Users can create accounts, log in, and perform actions such as booking parking spaces. Each user has personal information stored, including their email, username, name, password, phone number, address, and the ID of their associated vehicle. The login method verifies user credentials and initiates a session, while the signup method creates a new user account in the system.

Methods:

- `login()`
  - Input: Email, Password
  - Output: User is logged in and directed to the appropriate landing page.
  - Description: Authenticates the user credentials and initiates a session for the user.
- `signup()`
  - Input: Email, Username, First Name, Last Name, Password, Phone Number, Address, Vehicle ID
  - Output: User account is created.
  - Description: Registers a new user with the provided details and creates a new user profile in the system.

3.3 Vehicle Class

Attributes:

- plate_number: string
- type: string

- status: boolean
- username: string
- ownerName: string
- ownerContact: integer
- ownerAddress: string

Description:

The Vehicle class encapsulates information about vehicles registered in the parking system. It stores details such as the plate number, type, status, and ownership information. Vehicles can search for available parking spaces, book parking spots, and cancel existing bookings. This class facilitates interactions between vehicles and the parking infrastructure.

Methods:

- `searchParking()`
  - Input: None
  - Output: List of available parking spaces.
  - Description: Searches for available parking spaces for the vehicle.
- `bookParking()`
  - Input: Parking Space details
  - Output: Booking confirmation.
  - Description: Books a parking space for the vehicle.
- `cancelParking()`
  - Input: Booking details
  - Output: Cancellation confirmation.
  - Description: Cancels a previously booked parking space.

3.4 ParkingLot Class

Attributes:

- parkingLot_id: string
- name: string
- location: string
- capacity: integer
- currentAvailability: integer
- admin_id: integer

Description:

The ParkingLot class represents physical parking lots managed by the system. It contains attributes like the ID, name, location, capacity, current availability, and the ID of the admin overseeing operations. This class provides methods for booking and releasing parking spaces within the lot, as well as retrieving details about the parking lot's capacity and availability.

Methods:

- `bookSpace()`
  - Input: Vehicle details, User details
  - Output: Parking space booking details.
  - Description: Allocates a parking space to a vehicle and updates the parking lot's availability.
- `releaseSpace()`
  - Input: Parking space details
  - Output: Updated availability status.
  - Description: Releases a previously occupied parking space and updates the parking lot's availability.
- `getParkingLotDetails()`

- Input: None
- Output: Parking lot information.
- Description: Provides details about the parking lot including capacity and current availability.

3.5 ParkingSpace Class

Attributes:

- type: string
- status: string
- isOccupied: boolean
- parkingLot_id: string
- plate_number: string

Description:

The ParkingSpace class defines individual parking spaces within a parking lot. It holds information such as the type, status, occupancy status, and the ID of the parking lot it belongs to. Instances of this class are managed within ParkingLot objects and reflect the real-world parking spaces available for vehicles to use.

3.6 Admin Class

Attributes:

- name: string
- email: string
- username: string
- password: string
- phoneNumber: integer

- parkinglot_id: integer

Description:

The Admin class represents administrative users responsible for managing parking lots. Admins have attributes like name, email, username, password, phone number, and the ID of the parking lot they oversee. They can view parking lot data, manually assign parking spaces to vehicles upon entry, and release parking spaces when vehicles exit. This class empowers administrators to efficiently manage parking operations.

Methods:

- `viewParkingLotData()`
  - Input: None
  - Output: Parking lot data.
  - Description: Allows the admin to view detailed information about the parking lot they manage.
- `bookVehicleIn()`
  - Input: Vehicle details
  - Output: Parking space assignment.
  - Description: Manually assigns a parking space to a vehicle upon entry.
- `bookVehicleOut()`
  - Input: Vehicle details
  - Output: Parking space release.
  - Description: Manually releases a parking space when a vehicle exits.

3.7 BookingTicket Class

Attributes:

- id: integer

- user_id: integer
- vehicle: string
- timeSlot: TimeSlot
- parkingSpace: Parking Space
- isConfirmed: boolean
- payment id: string
- amount: float

Description:

The BookingTicket class tracks bookings made by users for parking spaces. It stores details such as the ticket ID, user ID, vehicle information, time slot, booked parking space, confirmation status, payment ID, and amount charged. Methods within this class calculate the parking fee based on the duration and other factors, and update the ticket with payment details once the payment is processed. This class ensures accurate record-keeping and facilitates seamless parking transactions.

Methods:

- `calculateAmount()`
  - Input: TimeSlot, Parking Space details
  - Output: Calculated amount for the parking.
  - Description: Calculates the amount to be charged for the booked parking space based on the time slot and other factors.
- `updatePaymentDetails()`
  - Input: Payment details
  - Output: Updated booking ticket with payment information.
  - Description: Updates the booking ticket with payment details once the payment is processed.

4 Execution Architecture

The execution architecture delineates the organizational framework of the parking management system. It encompasses various components such as the user interface for parking lot search and booking, backend servers for processing reservations and managing parking lot data, as well as integration with hardware devices like sensors or ticket dispensers for monitoring occupancy and facilitating vehicle entry/exit. This architecture may span across client-side applications, server-side logic, and database systems, incorporating communication protocols and concurrency mechanisms to handle simultaneous user interactions.

4.1 Reuse and relationships to other products

In the context of the parking management system, there exist opportunities for leveraging existing components and integrating with external products to augment functionality and efficiency. For instance, integration with mapping APIs can enhance location-based services, facilitating users in finding nearby parking lots. Furthermore, collaboration with payment gateways or mobile payment apps can streamline payment processing for parking reservations. Relationships with other products could also involve integration with navigation applications for real-time directions to available parking spaces or partnerships with parking lot operators to access and manage reservations.

5 Design decisions and tradeoffs

Design decisions within the parking management system encompass critical choices regarding the architecture of the reservation system and the selection of database technologies for storing parking lot data. Tradeoffs may arise between scalability and complexity, where centralized booking systems may offer simplicity but face scalability challenges, contrasting distributed systems which offer better performance but entail increased complexity. Similarly, the selection of hardware devices for vehicle detection and access control involves tradeoffs between cost, reliability, and functionality, influencing the overall effectiveness of the system.

6 Pseudocode for components

6.1 Class Name: Register

```
Class Name: User

Method: register()
Pseudocode:
    Input: username, email, password
    Output: Success message or error

    1. Validate input fields.
    2. Check if the username and email are unique.
    3. Hash the password for security.
    4. Store user data in the database.
    5. Return success message if registration is
       successful, otherwise return an error.
```

6.2 Class Name: Authentication

```
Method: login()
Pseudocode:
    Input: username, password
    Output: Success message or error

    1. Validate input fields.
    2. Query the database to retrieve user data based on
        the username.
    3. Verify the hashed password with the input
       password.
    4. If authentication is successful, create a session
        for the user.
    5. Return success message if login is successful,
       otherwise return an error.
```

6.3 Class Name: UserProfile

```
Method: viewProfile()
Pseudocode:
    Input: user_id
    Output: User profile data

    1. Query the database to retrieve user data based on
        the user_id.
    2. Display user profile data.

Method: updateProfile()
Pseudocode:
    Input: user_id, updated_profile_data
    Output: Success message or error

    1. Validate input fields.
    2. Update user profile data in the database.
    3. Return success message if profile update is
       successful, otherwise return an error.
```

6.4 Class Name: ParkingLot

```
Method: searchNearby()
Pseudocode:
    Input: user_location
    Output: List of nearby parking lots

    1. Query the database to retrieve parking lots near
       the user_location.
    2. Return the list of nearby parking lots.

Method: bookSpace()
Pseudocode:
    Input: user_id, parking_lot_id, time_slot
    Output: Success message or error
```

```
    1. Check if the parking space is available for the
       specified time_slot.
    2. Reserve the parking space for the user.
    3. Update the database with the booking information.
    4. Return success message if booking is successful,
       otherwise return an error.
```

6.5 Class Name: Admin

```
Method: login()
Pseudocode:
    Input: admin_username, admin_password
    Output: Success message or error

    1. Validate input fields.
    2. Verify admin credentials against stored data.
    3. If authentication is successful, create a session
        for the admin.
    4. Return success message if login is successful,
       otherwise return an error.
```

6.6 Class Name: ParkingLotManagement

```
Method: viewParkingLotData()
Pseudocode:
    Input: admin_id
    Output: Parking lot data

    1. Query the database to retrieve parking lot data
       managed by the admin.
    2. Display parking lot data.
```

6.7 Class Name: VehicleManagement

```
Method: bookInOutVehicle()
Pseudocode:
    Input: admin_id, vehicle_info, action (book in or
       book out)
    Output: Success message or error

    1. Validate input fields.
    2. Perform the specified action (book in or book out
       ) for the vehicle.
    3. Update the database with the vehicle's status.
    4. Return success message if the action is
       successful, otherwise return an error.
```

6.8 Class Name:FAQ

```
Method: viewFAQ()
Pseudocode:
    Input: None
    Output: List of frequently asked questions

    1. Query the database to retrieve frequently asked
       questions.
    2. Display the list of FAQs.

Method: postQuestion()
Pseudocode:
    Input: user_id, question
    Output: Success message or error

    1. Validate input fields.
    2. Store the user's question in the database.
    3. Return success message if the question is posted
       successfully, otherwise return an error.

Method: answerQuestion()
Pseudocode:
    Input: admin_id, question_id, answer
```

```
    Output: Success message or error

    1. Validate input fields.
    2. Check if the admin has permission to answer the
       question.
    3. Update the database with the admin's answer to
       the question.
    4. Return success message if the answer is posted
       successfully, otherwise return an error.
```

6.9 Class Name: Payment

```
Method: processPayment()
Pseudocode:
    Input: user_id, amount, payment_method
    Output: Success message or error

    1. Validate input fields.
    2. Process the payment using the specified payment
       method (e.g., credit card, PayPal, etc.).
    3. Update the user's payment record in the database.
    4. Return success message if the payment is
       processed successfully, otherwise return an error
       .

Method: refundPayment()
Pseudocode:
    Input: payment_id, refund_amount
    Output: Success message or error

    1. Validate input fields.
    2. Check if the payment with the specified
       payment_id exists.
    3. Process the refund for the specified
       refund_amount.
    4. Update the payment record in the database to
       reflect the refund.
    5. Return success message if the refund is processed
        successfully, otherwise return an error.
```

```
Method: viewPaymentHistory()
Pseudocode:
    Input: user_id
    Output: List of payment transactions

    1. Query the database to retrieve payment
       transactions for the specified user_id.
    2. Display the list of payment transactions.
```

6.10 Class Name: Report Generation

```
Method: generateReport()
Pseudocode:
    Input: start_date, end_date
    Output: Report file or error message

    1. Validate input fields.
    2. Query the database to retrieve relevant data for
       the specified date range.
    3. Generate a report based on the retrieved data.
    4. Save the report file to a specified location.
    5. Return the path to the report file if generation
       is successful, otherwise return an error.

Method: sendReportByEmail()
Pseudocode:
    Input: email_address, report_file_path
    Output: Success message or error

    1. Validate input fields.
    2. Check if the report file exists at the specified
       path.
    3. Send the report file as an email attachment to
       the specified email address.
    4. Return success message if the email is sent
       successfully, otherwise return an error.
```

6.11 Class Name: Notification

```
Method: sendNotification()
Pseudocode:
    Input: user_id, message
    Output: Success message or error

    1. Validate input fields.
    2. Send a notification message to the specified
       user_id.
    3. Return success message if the notification is
       sent successfully, otherwise return an error.
```

6.12 Class Name: Reservation

```
Method: makeReservation()
Pseudocode:
    Input: user_id, reservation_details
    Output: Success message or error

    1. Validate input fields.
    2. Check if the reservation can be made based on
       availability and other constraints.
    3. Reserve the specified item (e.g., hotel room,
       restaurant table) for the user.
    4. Update the database with the reservation details.
    5. Return success message if the reservation is
       successful, otherwise return an error.
```