Parking Spot

Rizwan Haider

F2023065277

Software Engineering

Submitted to: Dr. Kinza Sardar

Introduction

- **Project Name:** Parking Spot: A Smart Parking Assistance System
- **Overview:** This app is designed to help drivers efficiently locate legal parking spots, reserve place in parking area, locate parked vehicles, and secure vehicles with anti-theft features.
- **Objective:** To reduce the time spent searching for parking spots, improve convenience for drivers, and enhance vehicle security in urban areas.
- **Target Users:** Drivers seeking parking, vehicle owners concerned about theft or fines, and parking administrators.

Requirements (Functional and Non-Functional)

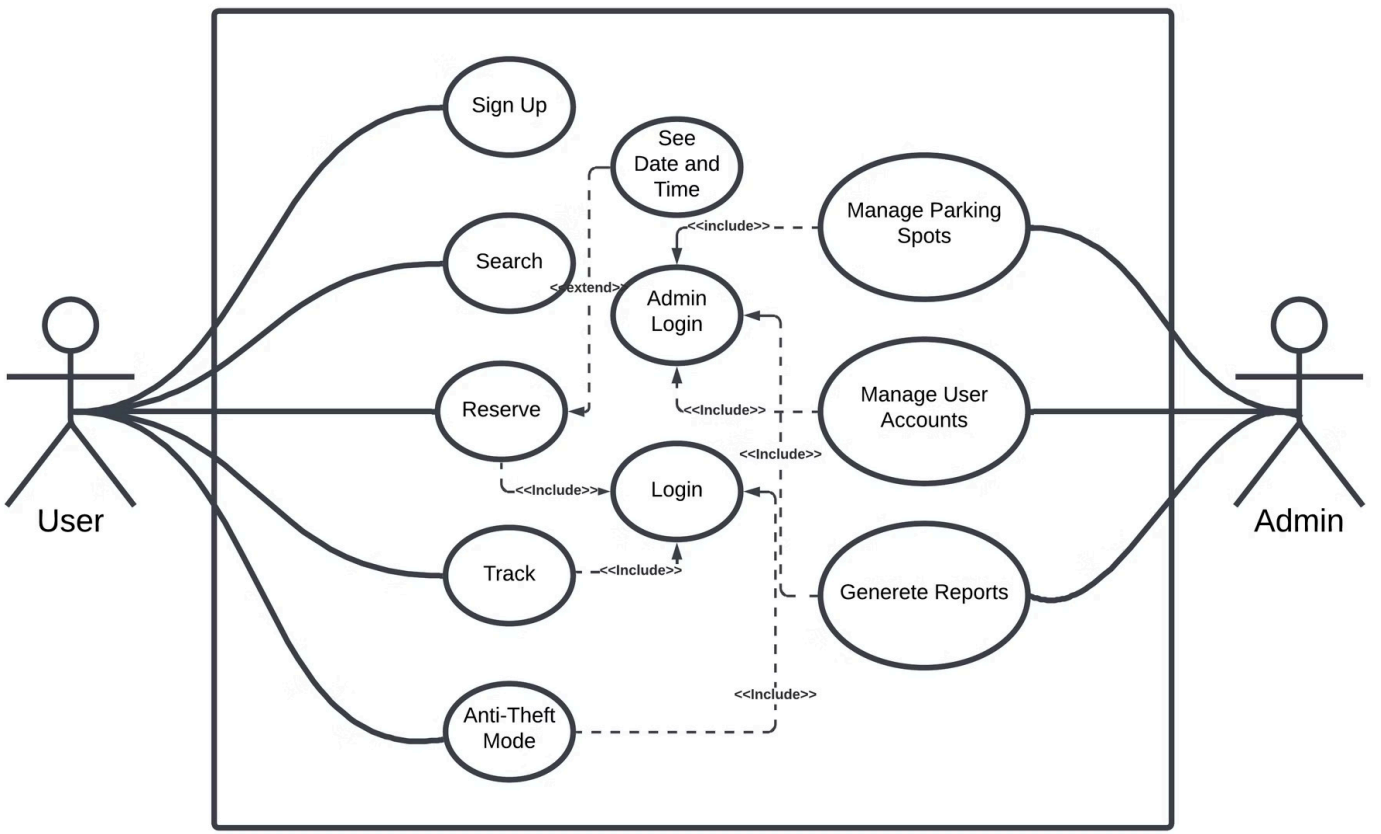
Functional Requirements:

- **User:**
  - Sign Up
  - Login
  - Search for available parking spots
  - Reserve a parking spot in advance
  - Track parked vehicle location
  - Turn on/off anti-theft mode
- **Admin:**
  - Separate admin login
  - View all users and parking spot data
  - Manage Parking Spot
  - Generate reports on parking usage, revenue, etc.

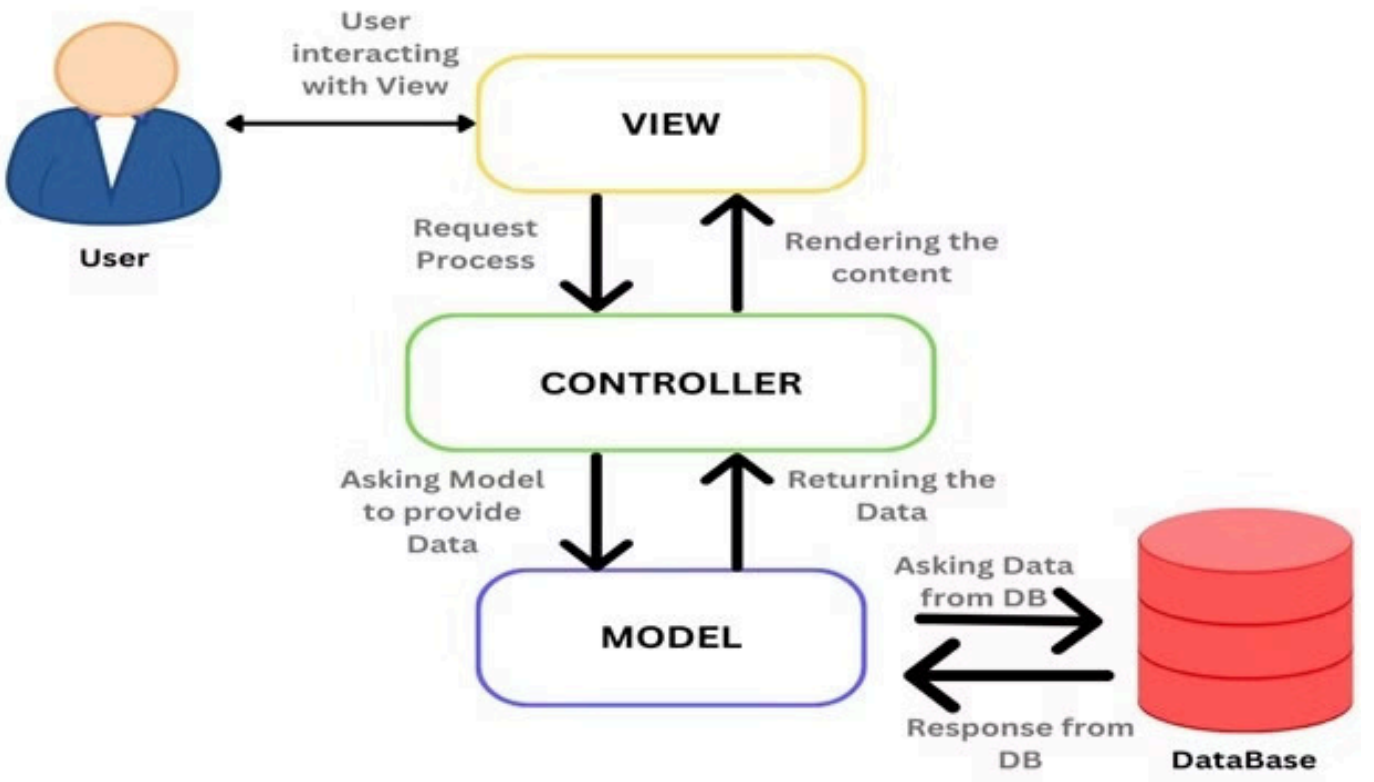
Non-Functional Requirements:

- **Performance:** The app should respond to user queries within 3 seconds.
- **Security:** User data and payment information must be securely encrypted.
- **Scalability:** The system should be able to handle a growing number of users and parking locations.
- **Availability:** The system should be available 24/7, with minimal downtime.

Use Case Diagram



Architecture Diagram MVC



A Smart Parking Assistance System. MVC is a widely used architectural pattern that separates the application into three interconnected components, making it easier to manage, scale, and maintain the system.

How MVC fits into your project:

1. **Model (Data Layer)**
   - Represents the data and business logic of the system.
   - In your case, it would handle data such as user information, parking spot availability, reservations, payments, vehicle tracking, and anti-theft functionality.
   - The **Model** would interact with the **Database** to store and retrieve information about parking spots, users, transactions, etc.
   - Example: **ParkingSpot**, **User**, **Reservation**, and **Payment** classes that manage data and logic.
2. **View (User Interface Layer)**
   - Responsible for rendering the data to the user and providing a way for users to interact with the system.
   - For the **User App**, this could include screens for searching for parking spots, making reservations, tracking the vehicle, and managing the anti-theft system.
   - For the **Admin Dashboard**, it could include interfaces for managing parking spots, viewing user data, generating reports, etc.
   - The **View** only focuses on how data is presented and doesn't contain business logic.
3. **Controller (Application Layer)**
   - Acts as an intermediary between the **Model** and the **View**.
   - It handles the input from the user (via the View), processes it (using the Model), and returns the appropriate output (back to the View).
   - The **Controller** contains the business logic for handling user actions like logging in, searching for parking spots, making reservations, processing payments, and activating anti-theft features.
   - Example: A **ParkingController** that handles requests like reserving a parking spot, a **UserController** that handles user login, or a **ReservationController** that handles creating and updating reservations.

MVC Workflow for Your System:

1. **User Interaction:**
   - The **User** or **Admin** interacts with the **View** (e.g., the mobile app or web dashboard).
2. **Controller Handling:**
   - The **Controller** receives the user input (such as a parking spot search or reservation request).
   - The **Controller** invokes the necessary **Model** (e.g., fetching available parking spots, processing payment, updating reservation data).
3. **Model Processing:**
   - The **Model** handles the business logic, such as checking parking availability, making reservations, processing payments, or updating anti-theft status.
   - It retrieves or stores data in the **Database** as needed.
4. **View Update:**
   - The **Controller** passes the processed data back to the **View**, which updates the user interface to reflect the results (e.g., showing available parking spots, confirming the reservation, displaying payment confirmation).

Example for MVC in Your Parking System:

- **Model:**
  - **ParkingSpot Model:** Defines properties like spot ID, location, availability, and pricing.
  - **User Model:** Contains user-specific information like name, email, and parking history.
  - **Reservation Model:** Manages parking reservations with fields like start time, end time, parking spot ID, and user ID.
  - **Payment Model:** Handles payment data, including amount, payment method, and transaction status.
- **View:**
  - **Mobile App UI (User):** Interface for users to search for available spots, reserve a spot, and view payment details.
  - **Admin Dashboard UI:** Web interface for admins to manage parking spots, users, generate reports, etc.
- **Controller:**
  - **ParkingController:** Handles user requests for searching and reserving parking spots.
  - **UserController:** Manages user authentication (login, signup).
  - **PaymentController:** Handles payment processing (integrating with the payment gateway).

Benefits of Using MVC in Your Project:

- **Separation of Concerns:** It allows you to separate the application logic (Model), user interface (View), and business processing (Controller), making your code easier to maintain and update.
- **Scalability:** As your project grows, it becomes easier to extend and scale the system by adding new features, such as adding new payment methods or integrating more security features.
- **Testability:** Since the components are decoupled, it becomes easier to write unit tests for each component (Model, View, and Controller) independently.

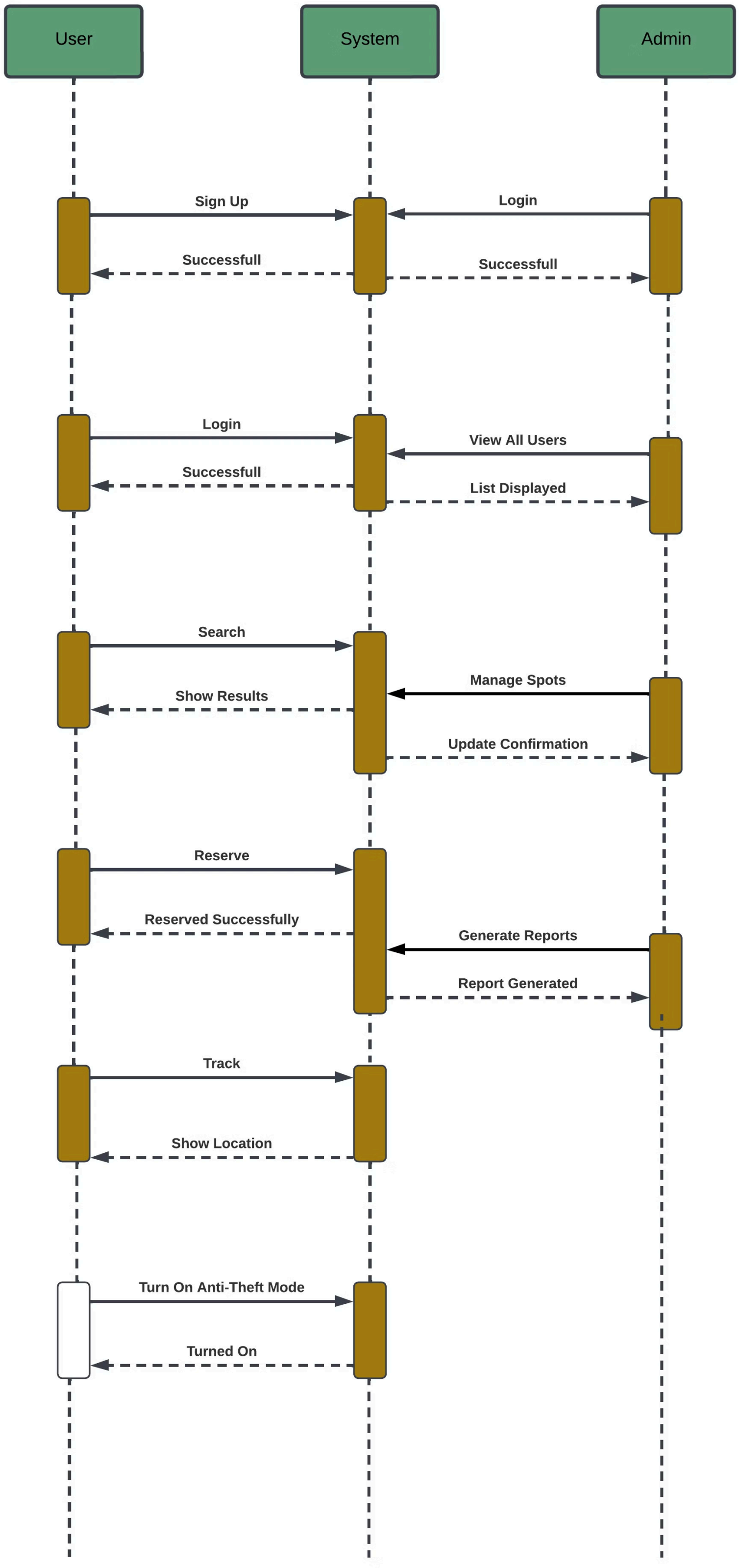
Conclusion:

MVC is a good architectural choice for your **Parking Spot System**. It will allow you to manage different parts of the system independently (e.g., user interface, data management, and business logic), making it more maintainable, scalable, and easier to understand. You can implement it effectively by dividing your system's responsibilities according to the Model, View, and Controller components.

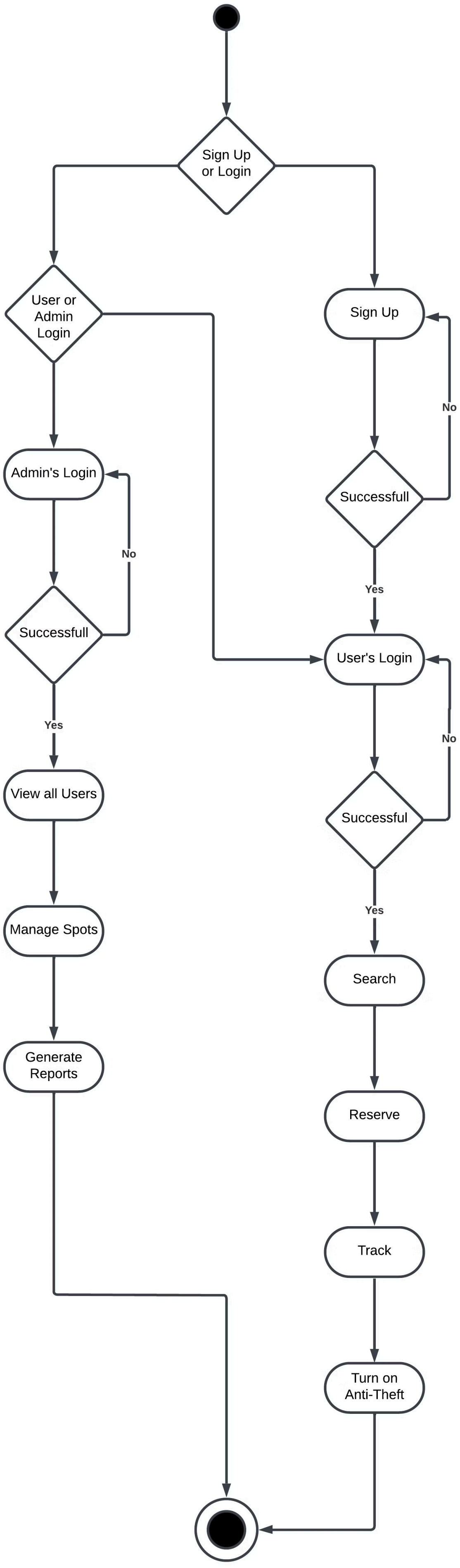
Class Diagrams


Admin
-adminUsername
-adminPassword
+ viewAllUsers()
+ manageParkingSpots()
+ generateReports()

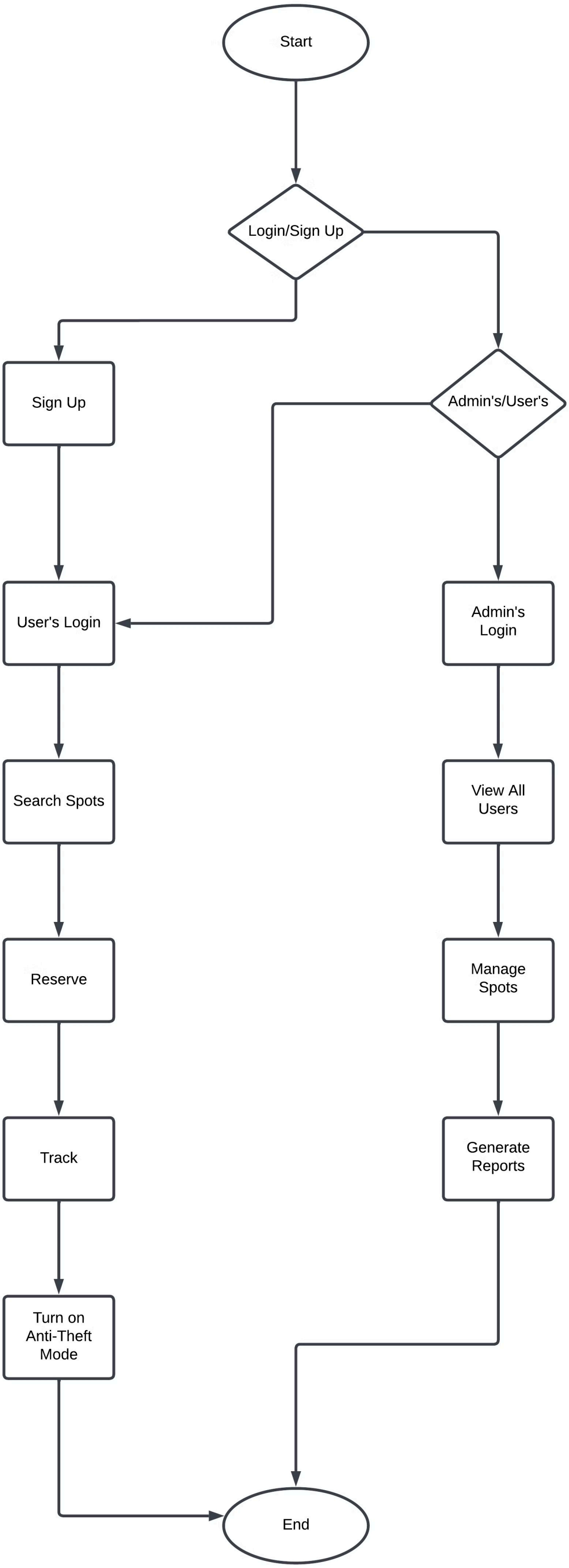
Sequence Diagram



Activity Diagram



Flowchart


Start
Login/Sign Up
Sign Up
Admin's/User's
User's Login
Admin's Login
Search Spots
View All Users
Reserve
Manage Spots
Track
Generate Reports
Turn on Anti-Theft Mode
End

Prototype

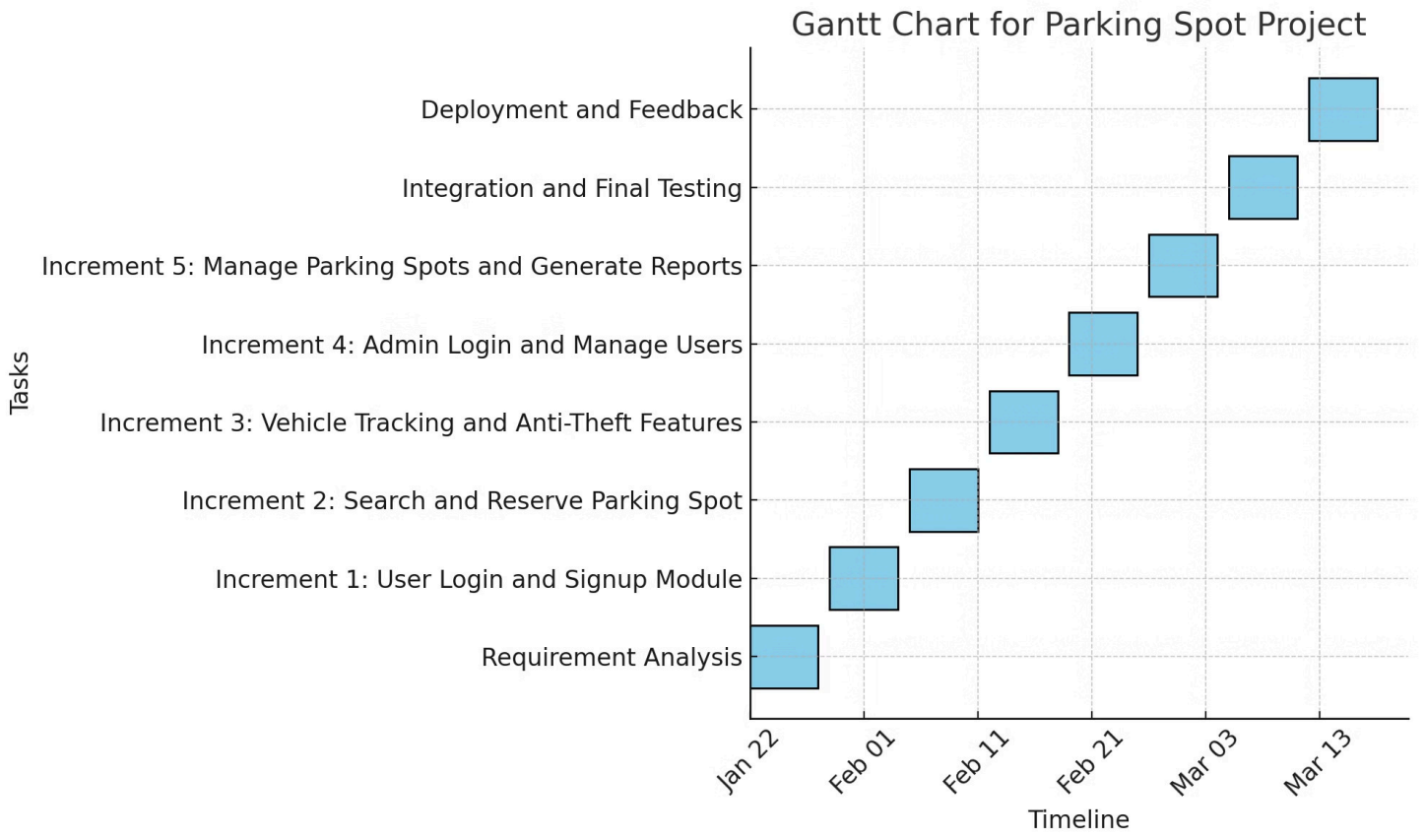
parking-spot.w3spaces.com

parking-spot.w3spaces.com

Create your own website | W3 Spaces

Notice: This Site is User-Created

Gantt Chart


Gantt Chart for Parking Spot Project
Tasks
Deployment and Feedback
Integration and Final Testing
Increment 5: Manage Parking Spots and Generate Reports
Increment 4: Admin Login and Manage Users
Increment 3: Vehicle Tracking and Anti-Theft Features
Increment 2: Search and Reserve Parking Spot
Increment 1: User Login and Signup Module
Requirement Analysis
Jan 22
Feb 01
Feb 11
Feb 21
Mar 03
Mar 13
Timeline

WPS (Work Breakdown Structure)

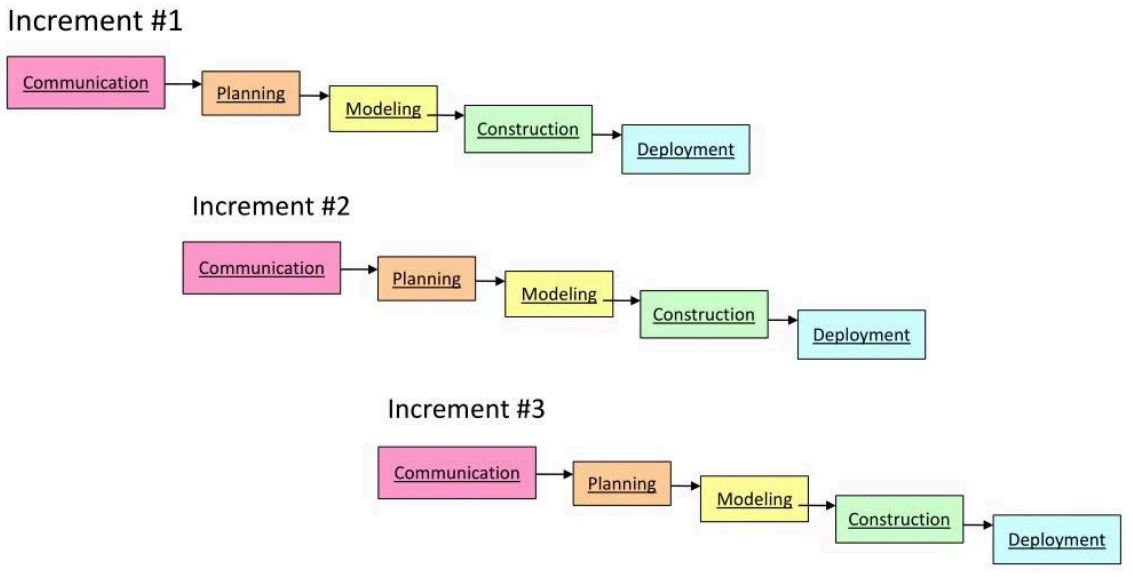
- **Project Initiation:**
  - Define project goals and requirements
  - Assemble project team
- **System Design:**
  - Create use case, sequence, and class diagrams
  - Design system architecture
  - Develop database schema
- **Development:**
  - Develop user and admin functionalities
  - Implement payment integration
  - Build anti-theft features and location tracking
- **Testing:**
  - Perform unit testing for individual components
  - Conduct integration testing for system interactions
- **Deployment:**
  - Deploy the application on cloud servers
  - Set up the production environment
- **Maintenance:**
  - Provide ongoing support and bug fixes
  - Implement periodic system updates and feature enhancements

Testing

- **Unit Testing:** Focus on testing individual components like login, parking spot search, reservation, and anti-theft features.
- **Integration Testing:** Ensure that different modules work together seamlessly, such as the interaction between the reservation system and payment processing.
- **System Testing:** Verify that the entire system functions as expected, ensuring all features (user login, search, reservation, etc.) work in real-world scenarios.
- **Acceptance Testing:** Validate that the system meets the user's needs and performs as expected in a production environment.
- **Performance Testing:** Ensure the system can handle the expected load, with optimal response times even during peak usage.

SDLC

Incremental Model (Diagram)



Conclusion

- The **Parking Spot** app is designed to solve common parking issues by providing real-time information about available spots, ensuring smoother parking experiences for drivers. The system also focuses on enhancing vehicle security through anti-theft features.
- The project aims to reduce the time spent searching for parking, thus improving the overall efficiency of urban mobility.
- With a solid system design, detailed testing, and comprehensive features for both users and admins, the Parking Spot app has the potential to become a reliable and innovative solution in smart parking assistance.