

# **Software Design Specification Document**

## **Car Park Management System**

Version: 1.0



School of Computing, Communication & Media Studies

Department of Computing

## Revision Page

### a. Overview

Car Park Management System is a system designed for car parking company to manage their car park effectively. It allows admin to track the usage of car park, manage car park slot and edit the parking rate. It allows end user to track the location of their car, total parking hour(s) and total parking fees. The app provides a user-friendly interface with simple and clear icon.

### b. Target Audience

The system is suitable for valet parking, car park company, mall, or any location with car park.

### c. Version Control History

Version	Primary Author(s)	Description of Version	Date Completed
1.0	Enoch Leong Qi Cong	CPMS 1.0	15/3/2024

#### **Note:**

This template is an annotated outline for a software design document adapted from the IEEE Recommended Practice for Software Design Descriptions. The IEEE Recommended Practice for Software Design Descriptions have been reduced in order to simplify this assignment while still retaining the main

components and providing a general idea of a project definition report. Please refer to IEEE Std 1016-1998 1 for the full IEEE Recommended Practice for Software Design Descriptions. Examples of models are from Satzinger (2011). Compiled by Shahliza Abdul Halim, PhD and checked by Shahida Sulaiman, PhD on 2 May 2016.

# Table of Contents

1.0 Introduction .....	7
1.1 Purpose .....	7
1.2 Scope .....	7
1.3 Definitions, Acronyms and Abbreviations .....	8
1.4 Reference Materials.....	8
1.5 Overview .....	8
2.0 Overall Description .....	10
2.1 Product Perspective .....	10
2.1.1 User Interfaces .....	10
2.1.2 Hardware Interfaces .....	10
2.1.3 Communication Interfaces.....	10
2.2 Product Feature .....	11
3.0 System Architectural Design and Strategies .....	12
3.1 Architectural pattern and Rationale .....	12
3.2 System Architecture.....	13
3.3 Use Case Diagram .....	14
3.4 Subsystem Architecture components.....	14
3.4.1 Subsystem Architecture Components .....	16
3.5 System States .....	16
3.5.1 Login .....	17
3.5.2 Logut .....	18
3.5.3 Add Car to Slot .....	19
3.5.4 Edit Car Info.....	20
3.5.5 Edit Parking Rate .....	21
3.5.6 Search Vehicle.....	22
3.5.7 View Total Parked Hour .....	23
3.5.8 View parking fees .....	24
3.5.9 Make Payment .....	25
4.0 Detailed Description of Modules .....	25
4.1 Complete Package Diagram .....	26

4.2 Modules Detailed Descriptions .....	26
4.2.1 Module <Login> .....	26
4.2.2 Module <Logout> .....	28
4.2.3 Module <Add Car to Slot> .....	30
4.2.4 Module <Edit Car Info> .....	33
4.2.5 Module <Edit Parking Rate> .....	36
4.2.6 Module <Search Vehicle> .....	38
4.2.7 Module <View Total Parked Hour> .....	40
4.2.8 Module <View Parking Fees> .....	42
4.2.9 Module <Make Payment> .....	45
5.0 Design Pattern .....	47
5.1 Design Pattern and Rationale .....	47
5.2 Design Pattern Implementation .....	49
5.2.1 Singleton Design Pattern .....	49
5.2.2 Facade Design Pattern .....	50
5.2.3 Command Design Pattern .....	51
6.0 Data Design .....	51
6.1 Data Description .....	51
6.2 Data Dictionary .....	53
7.0 User Interface Design .....	53
7.1 Overview of User Interface .....	54
7.2 Screen Images .....	55

## **1.0 Introduction**

### **1.1 Purpose**

The objective of this document is to delineate the precise requirements elicited by the Car Park Management System (CPMS) project from stakeholders within the organization. It outlines the functionality, performance expectations, interface specifications, quality attributes, and compliance requirements of the CPMS application. This document is intended for system developer, project manager, configuration manager and client.

### **1.2 Scope**

The Car Park Management System (CPMS) shall primarily support the following operations:

- i. User Authentication and Registration:
  - Users shall be able to log in securely to their accounts.
  - New users shall be able to register for an account.
- ii. Parking Slot Management:
  - Users shall be able to view available parking slots.
  - Admins shall be able to add, remove, and edit parking slots.
- iii. Car Parking and Retrieval:
  - Users shall be able to park their cars in available slots.
  - Users shall be able to retrieve their parked cars.
- iv. Fee Calculation and Payment:
  - The system shall calculate parking fees based on duration and rate.
  - Users shall be able to make payments through various methods.
- v. Reporting and Analytics:
  - The system shall generate reports on parking occupancy, revenue, etc.
  - Admins shall be able to view analytics and make data-driven decisions.
- vi. Administrative Functions:
  - Admins shall have access to features for managing users, rates, and system settings.

- Admins shall be able to monitor and resolve system issues.

### **1.3 Definitions, Acronyms and Abbreviations**

#### **Definition**

Car Park Management System – A system for managing and overseeing car park operations.

#### **Acronyms**

CPMS - Car Park Management System

SDS – Software Design Specification Document

### **1.4 Reference Materials**

This document is prepared in reference to the following documents:

- i. IEEE std 830-1998, Recommended of Practice for Software Requirement Specification

### **1.5 Overview**

This document consists of 3 sections:

Section 1 contains the purpose, scope, definitions, acronyms and references made to their documents.

Section 2 contains the overall description of the system, including product perspective and product feature.

Section 3 contains system architectural design of CPMS to be implemented.

Section 4 contains detailed description of modules.

Section 5 contains the design pattern of CPMS to be implemented.

Section 6 contains the data design, including data description and data dictionary.

Section 7 contains the user interface design of CPMS.



## 2.0 Overall Description

### 2.1 Product Perspective

#### 2.1.1 User Interfaces

The following table (Table 1.0) summarizes the list of graphical user interface requirements specified for CPMS:

REQUIRMENT ID	Description	Priority	Author
REQ_U1001	CPMS GUI shall have provide login screen for admin to access the system securely.	Medium	Enoch Leong Qi Cong
REQ_U1002	CPMS GUI shall provide a user-friendly dashboard for end users to search the car plate number and view key information at a glance, such as parked location, total parking hour(s) and total parking fees	High	Enoch Leong Qi Cong
REQ_U1003	CPMS GUI shall provide admin access to screens for display car park slot and managing parking slots, including adding, editing, and removing slots.	High	Enoch Leong Qi Cong

Table 1.0: CPMS Graphical User Interface Requirements

#### 2.1.2 Hardware Interfaces

Not applicable

#### 2.1.3 Communication Interfaces

CPMS shall connect to clients using Local Area Network with minimum speed of 1Mbps.

## 2.2 Product Feature

The following table (Table 1.1) contains the list of features to be implemented in HBS.

Feature ID	Feature	Description	Accessible Role
F001	Search Vehicle	To allow the end user to search their vehicle	End User
F002	View total parked hour	To allow the end user to view the total hour of their vehicle parked	End User
F003	View parking fees	To allow the end user to view the total calculated parking fees for their vehicle	End User
F004	Make payment	To allow end user to make payment before they go out the parking	End User
F005	Login	To allow admin login to the system	Admin
F006	Logout	To allow admin logout from the system	Admin
F007	Add Car to Slot	To allow admin add car plate number to the available car park slot	Admin
F008	Edit Car Info	To allow admin to edit the car info for error handling	Admin

Table 1.1: Product Features

## **3.0 System Architectural Design and Strategies**

### **3.1 Architectural pattern and Rationale**

The Car Park Management System (CPMS) employs the Model-View-Controller (MVC) architecture pattern, which organizes the application into three key components: Model, View, and Controller.

**Model:** The Model component represents the data and defines the storage of all application's data objects. It encapsulates the business logic and interacts with the database to retrieve and manipulate data related to parking slots, transactions, and system settings.

**View:** The View component is associated with user interfaces, providing the presentation layer of the application. It includes different views such as login screens, dashboard, parking slot management interfaces, and administrative panels. Each view presents data to the user in a visually appealing and intuitive manner.

**Controller:** The Controller component handles the request handling and serves as an intermediary between the Model and View components. It receives user input from the View, processes the requests, interacts with the Model to retrieve or update data, and then renders the appropriate response back to the user through the View.

MVC promotes modularity by separating the concerns of data, presentation, and user interaction. This separation allows for easier maintenance and scalability of the application. MVC supports multiple views for the same data, enabling the development of different user interfaces without duplicating business logic. This flexibility enhances the user experience and accommodates diverse user needs. The MVC paradigm facilitates the initial planning phase of the application by providing a clear structure for organizing code and defining responsibilities. It also helps in limiting code duplication and simplifies maintenance tasks, making it easier to update and enhance the application over time.

### 3.2 Layered System Architecture

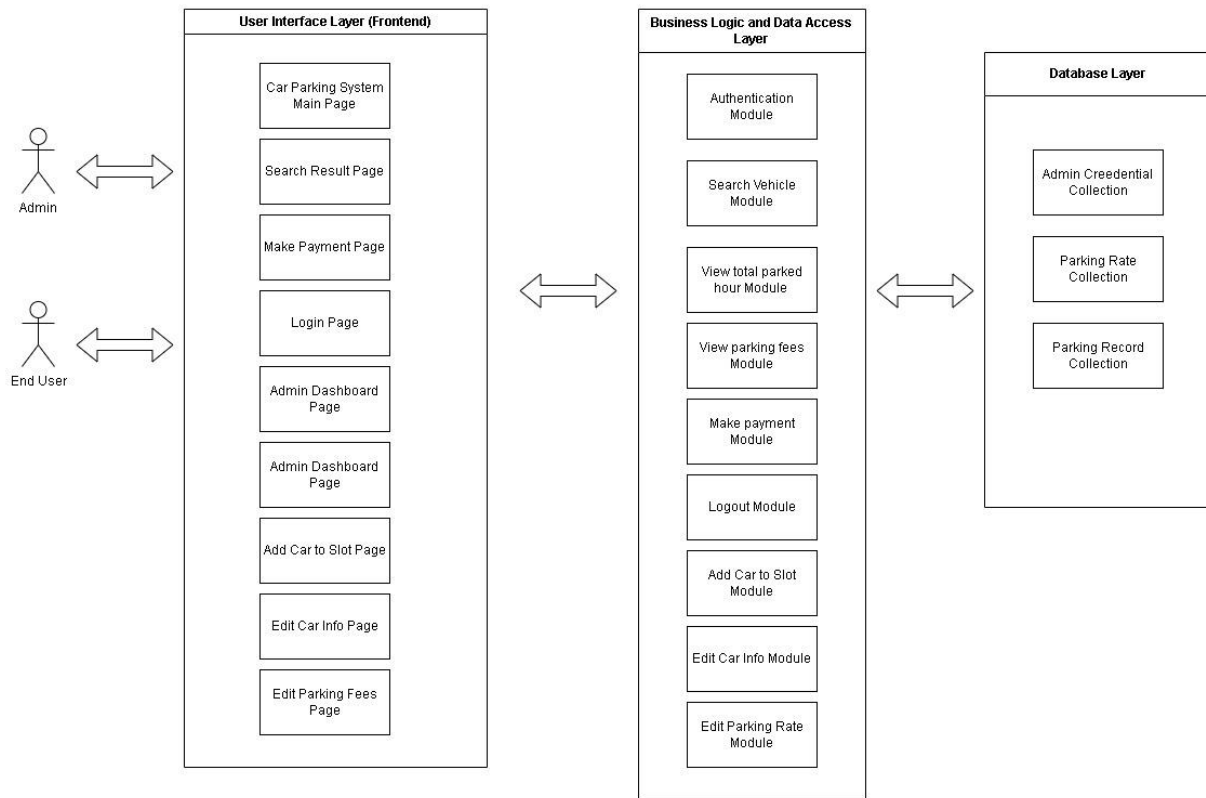


Figure 2.0: Layered System Architecture for Car Parking Management System (CPMS)

### 3.3 Use Case Diagram

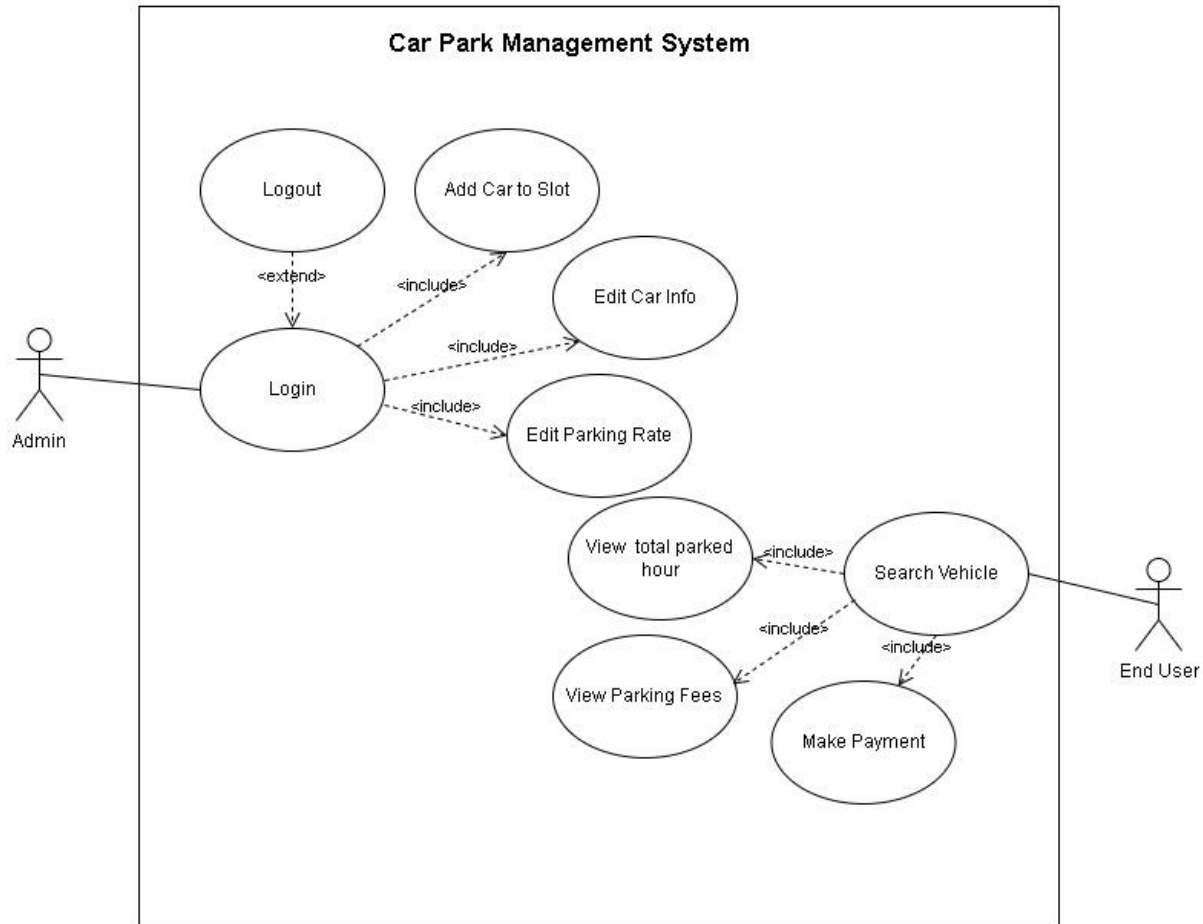


Figure 2.1: Use Case of Car Park Management System

### 3.4 Subsystem Architecture components

This section illustrates the components within each subsystem of Car Park Management System (CPMS) using component diagram.

The subsystems of Car Park Management System (CPMS) are listed in the table below (Table 2.0) along with their tasks.

Component	Tasks
-----------	-------

Search Vehicle	- To allow the end user to search their vehicle
View total parked hour	- To allow the end user to view the total hour of their vehicle parked
View parking fees	- To allow the end user to view the total calculated parking fees for their vehicle
Make payment	- To allow end user to make payment before they go out the parking
Login	- To allow admin login to the system
Logout	- To allow admin logout from the system
Add Car to Slot	- To allow admin add car plate number to the available car park slot
Edit Car Info	- To allow admin to edit the car info for error handling
Edit Parking Rate	- To allow admin to edit the parking rate

Table 2.0: Car Park Management System (CPMS) Subsystems

### 3.4.1 Subsystem Architecture Components

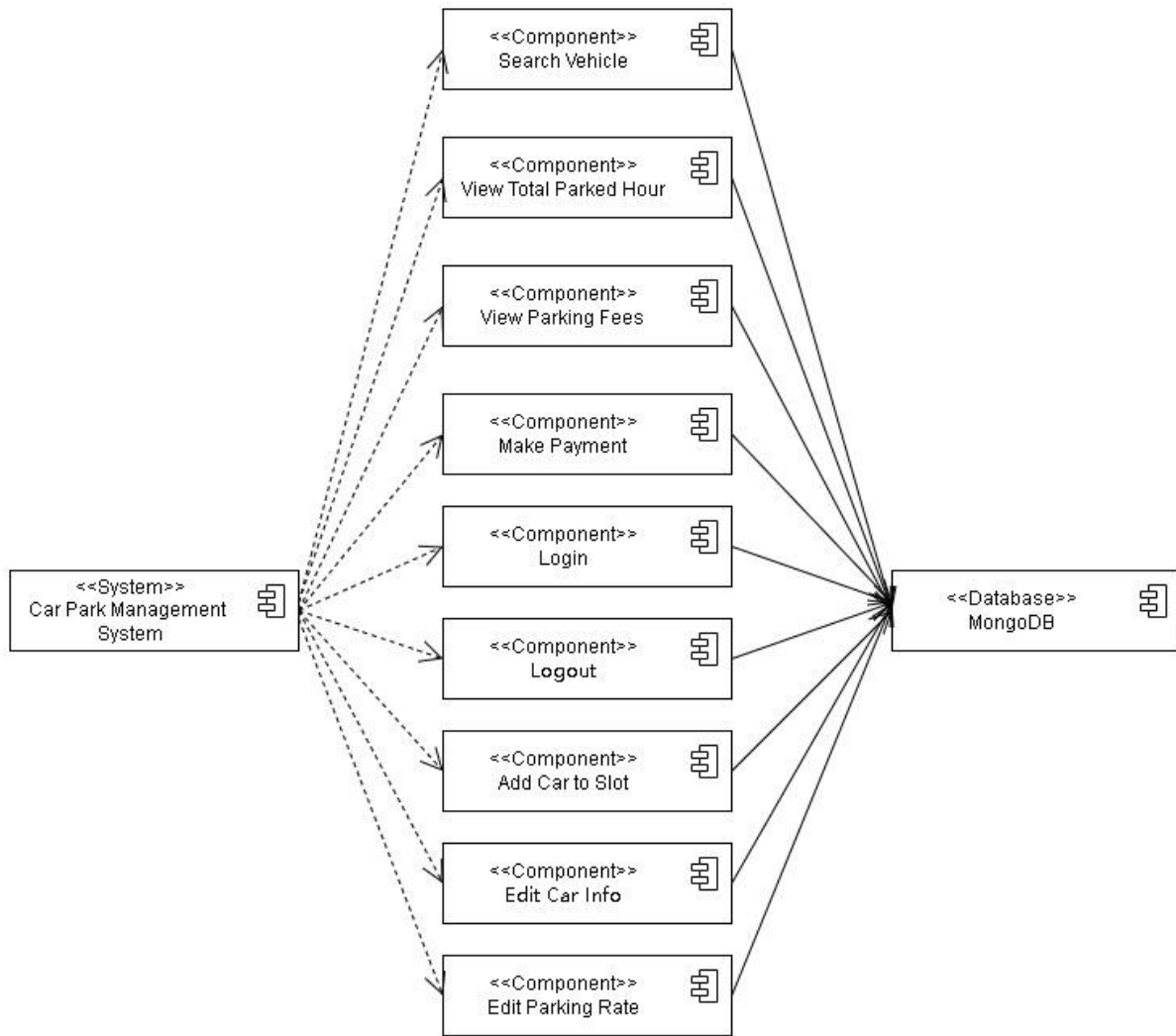


Figure 2.2: Component Diagram

### 3.5 System States

This section explains the states of each picture.

### 3.5.1 Login

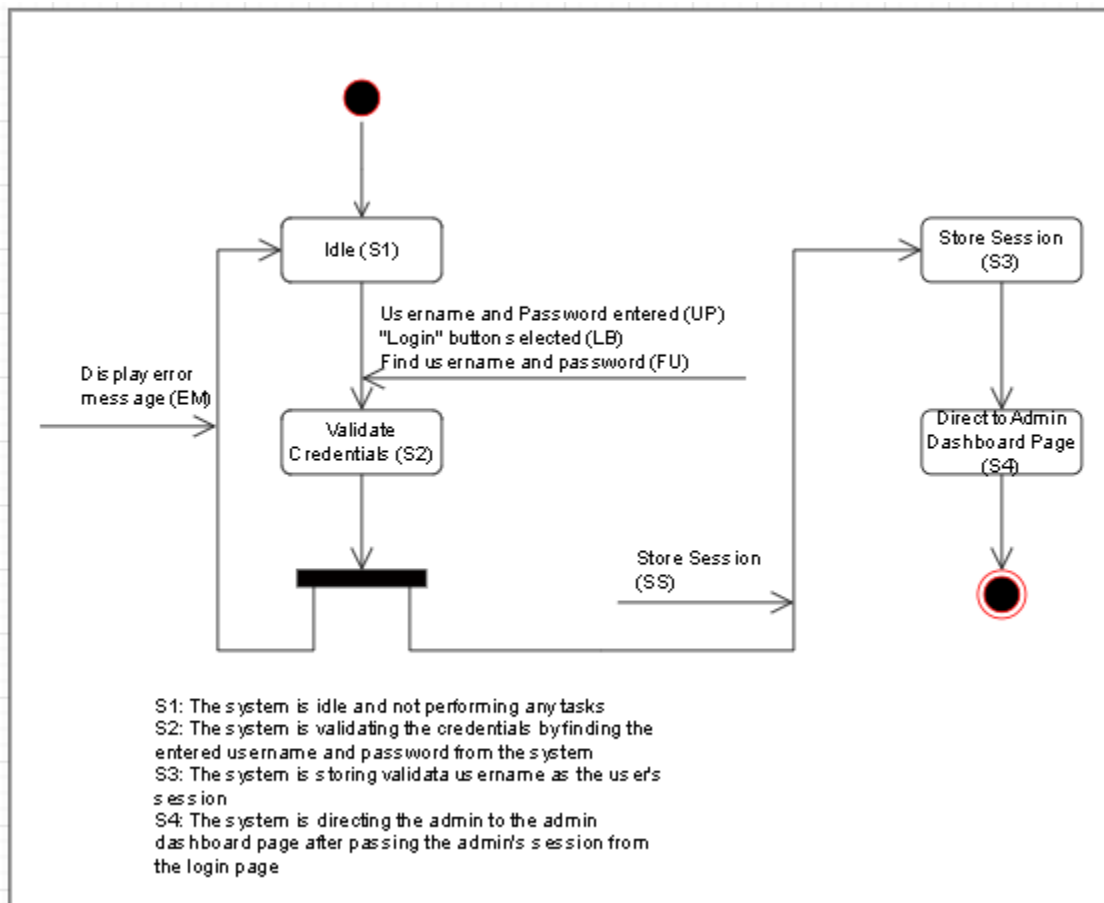


Figure 2.3: State Diagram - Login

This system state is Login. When the user enters the login credentials. The system will validate whether the login credential is correct or not.



### 3.5.2 Logut

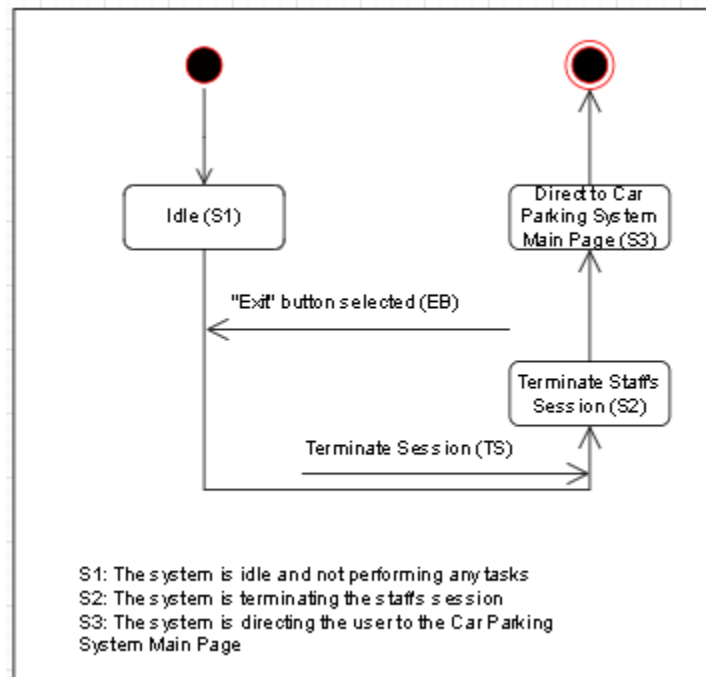


Figure 2.4: State Diagram - Logout

This system state is Logout. When the admin clicks "Exit" button. The system will terminate the current login session.

### 3.5.3 Add Car to Slot

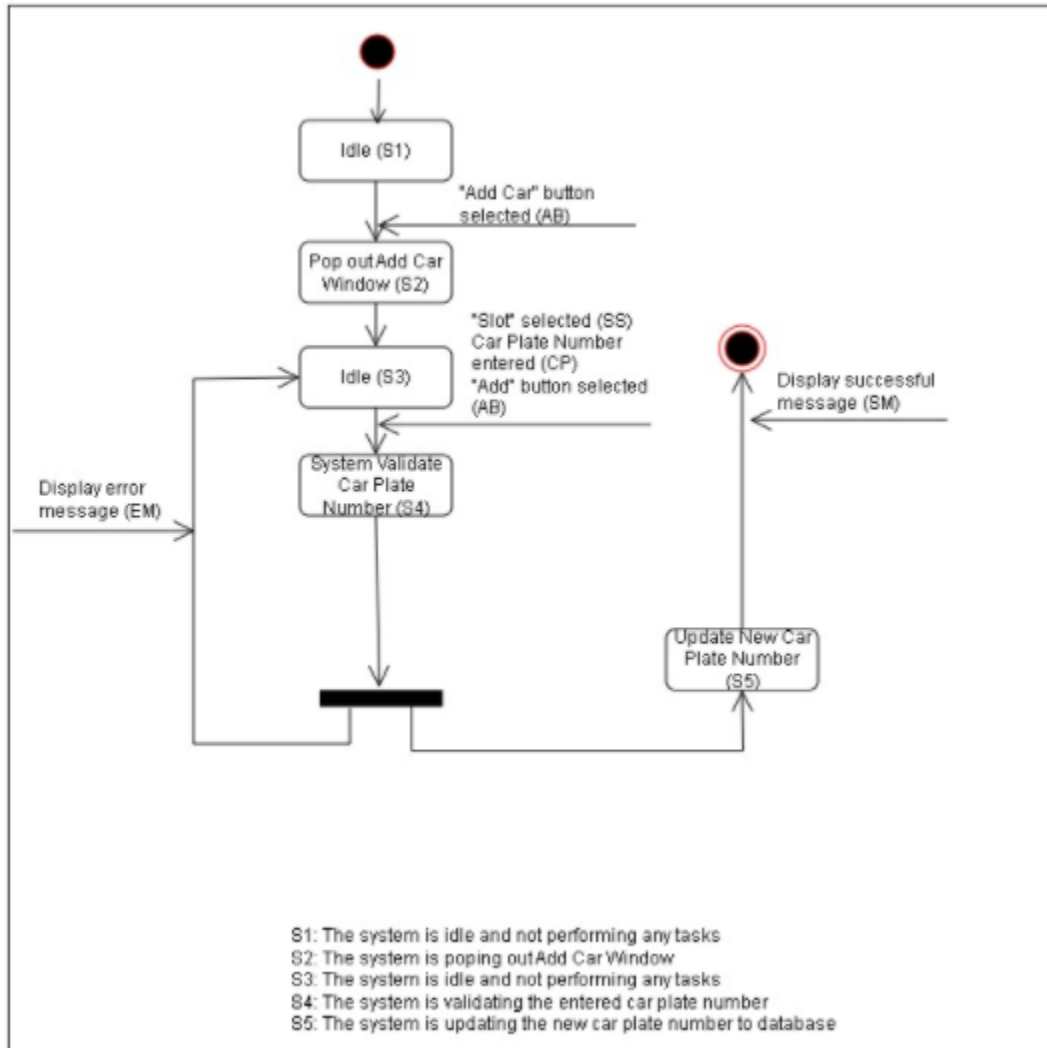


Figure 2.5: State Diagram – Add Car to Slot

This system state is Add Car to Slot. Admin can add the car plate number to the slot that the car parked. The information will be validated after the operation is performed.

### 3.5.4 Edit Car Info

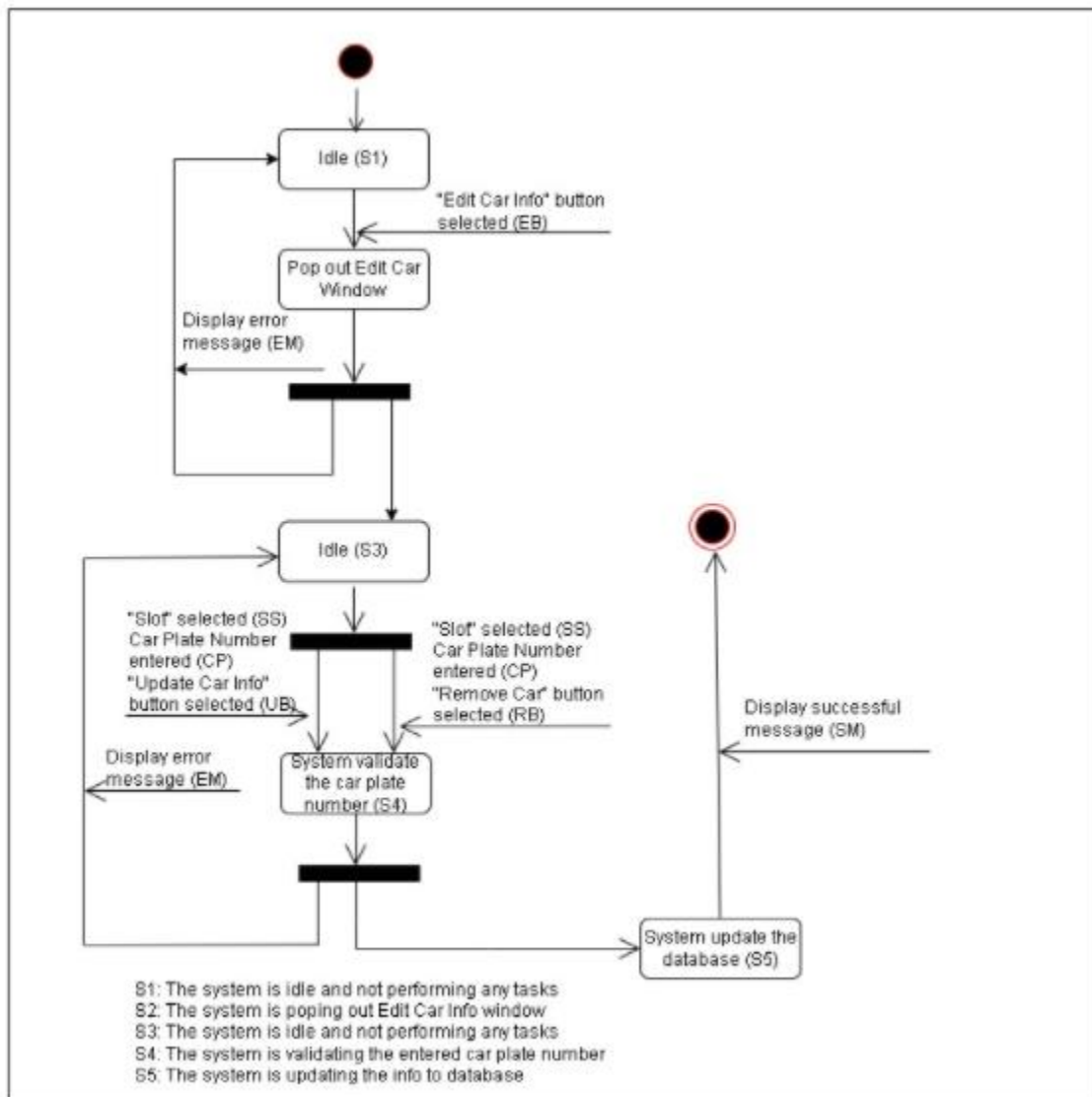


Figure 2.6: State Diagram – Edit Car Info

This system state is Edit Car Info. Admin can manage the car plate number on selected slot and remove the car for error handling. The information will be validated after the operations are performed.

### 3.5.5 Edit Parking Rate

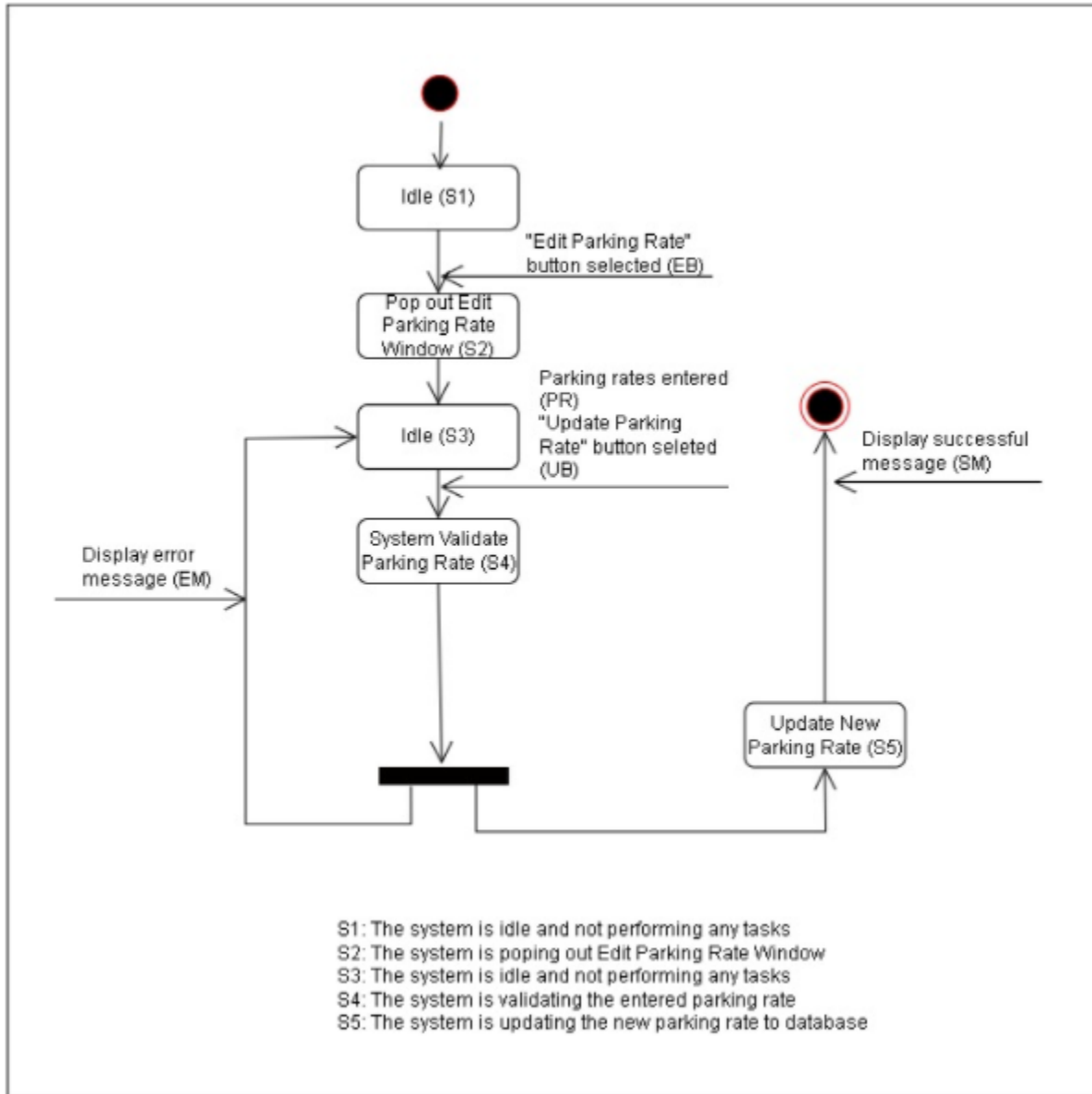


Figure 2.7: State Diagram – Edit Parking Rate

This system state is Edit Parking Rate. Admin can edit the parking rate for every hour in this state.

### 3.5.6 Search Vehicle

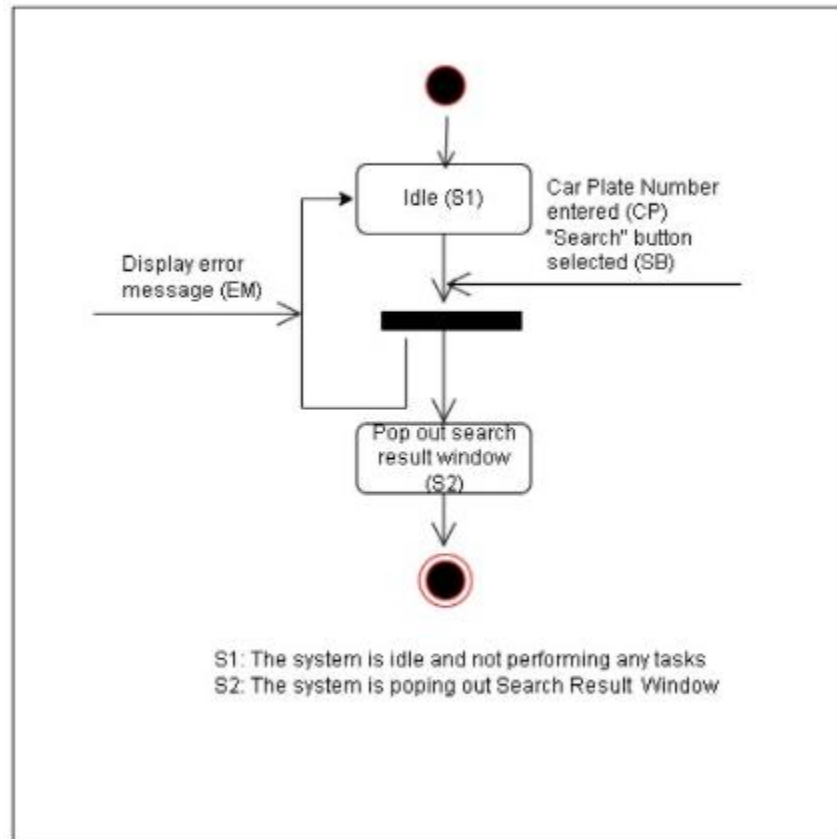


Figure 2.8: State Diagram – Search Vehicle

This system state is Search Vehicle. End user can keep track their car info. The system will validate the car plate number entered by end user. The car info will display such as parking location, total parked hour, and total parking fees after the validation.

### 3.5.7 View Total Parked Hour

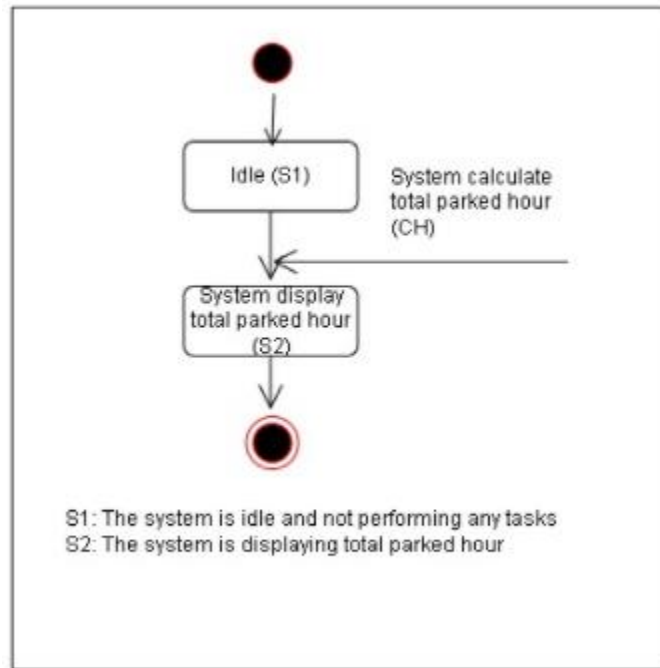


Figure 2.9: State Diagram - Total Parked Hour

This system state is Total Parked Hour. System will calculate and display the info to end user in this system state.

### 3.5.8 View parking fees

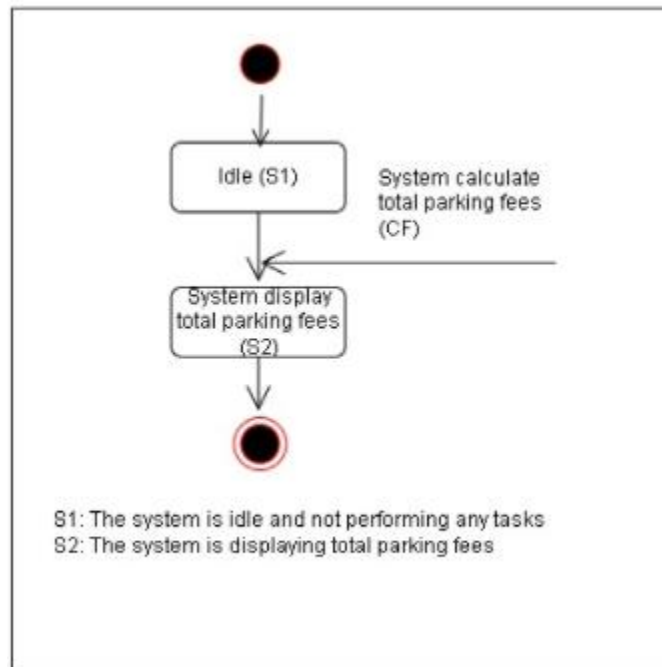


Figure 2.9: State Diagram - Total Parking Fees

This system state is View Parking Fees. System will calculate and display the info to end user in this system state.

### 3.5.9 Make Payment

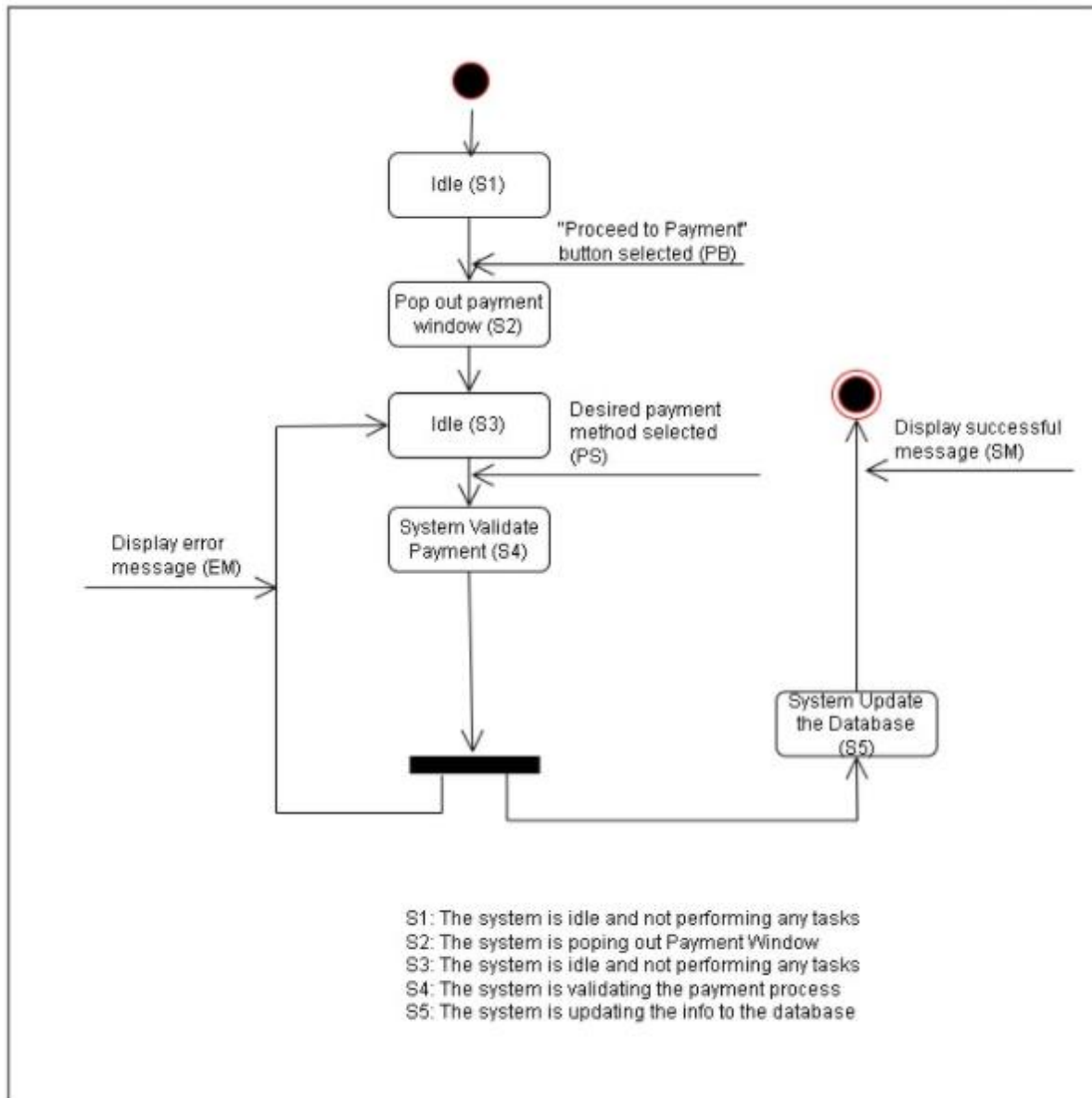


Figure 2.10: State Diagram – Make Payment

This system state is Make Payment. System will validate the payment make by end user. Upon successful, system will update and remove the car.

## 4.0 Detailed Description of Modules



## 4.1 Complete Package Diagram

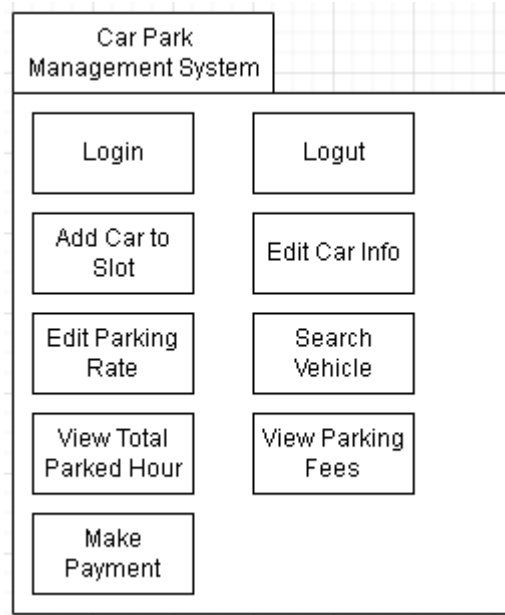


Figure 3.0: Subsystem of Car Park Management System

## 4.2 Modules Detailed Descriptions

### 4.2.1 Module <Login>

#### 4.2.1.1 P001: Package Diagram <Login>

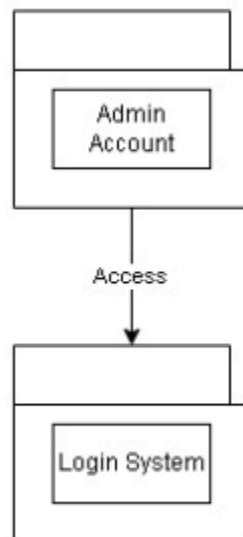


Figure 4.2.1.1.1: P001: Package Diagram <Login>

#### 4.2.1.2 Class Diagram

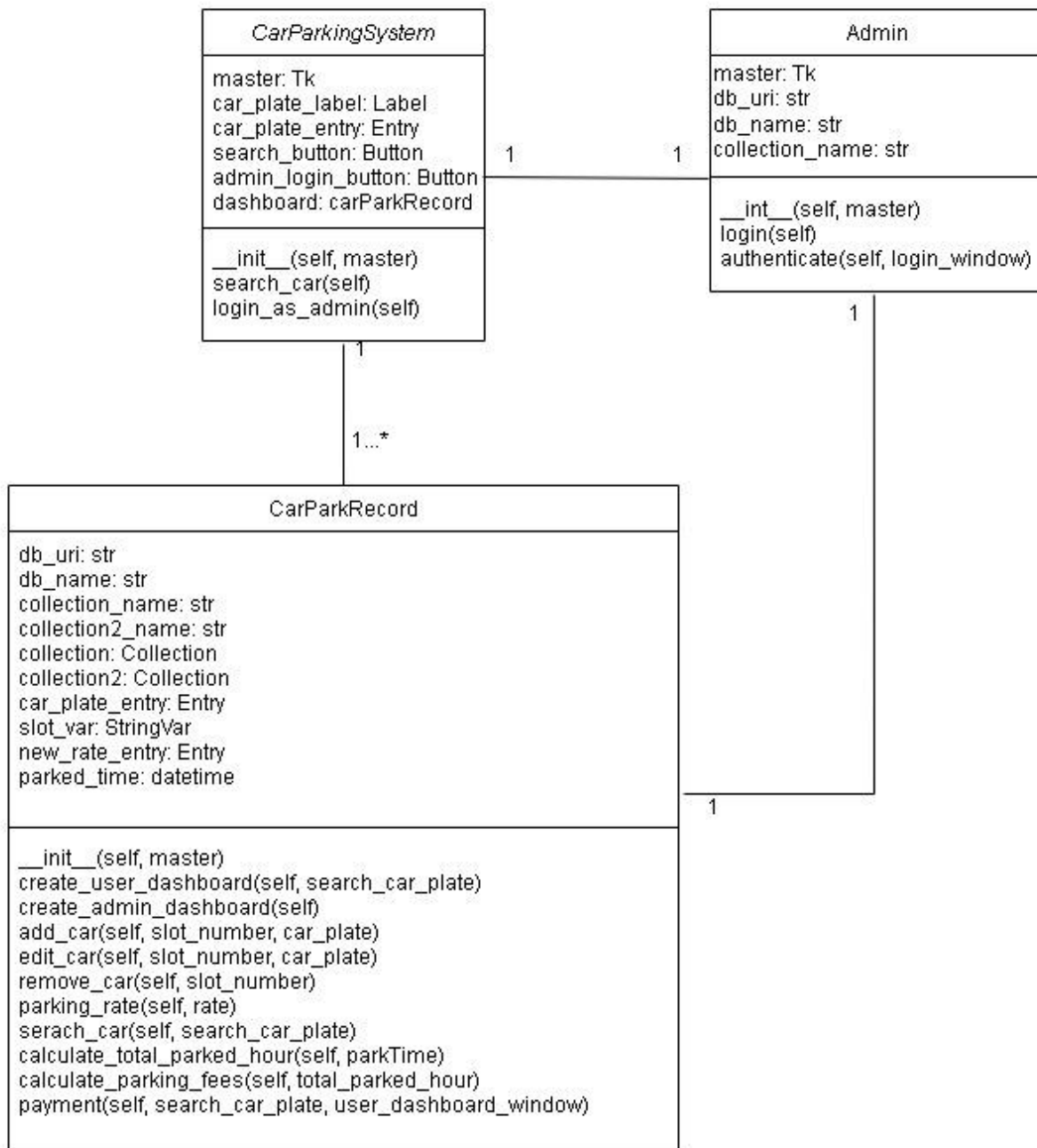


Figure 4.2.1.2.1: Class Diagram <Login>

Implementation:

```
def authenticate(self, login_window):  
    # Connect to MongoDB and retrieve admin credentials  
    client = pymongo.MongoClient(self.db_uri)  
    db = client[self.db_name]  
    collection = db[self.collection_name]  
    admin = collection.find_one({"username": self.username_entry.get(), "password": self.password_entry.get()})  
  
    if admin:  
        self.dashboard = carParkRecord(self.master)  
        login_window.destroy()  
        self.dashboard.create_admin_dashboard()  
    else:  
        messagebox.showerror("Authentication Result", "Invalid username or password.")
```

Figure 4.2.1.2.2: Implementation of <Login>

## 4.2.2 Module <Logout>

### 4.2.2.1 P002: Package Diagram <Logout>

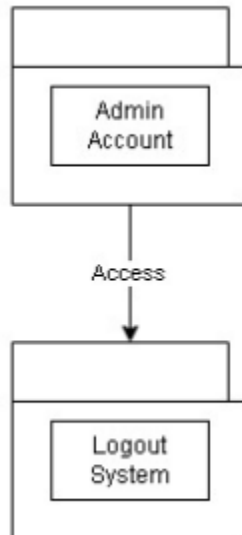


Figure 4.2.2.1.1: P002: Package Diagram <Logout>

#### 4.2.2.2 Class Diagram

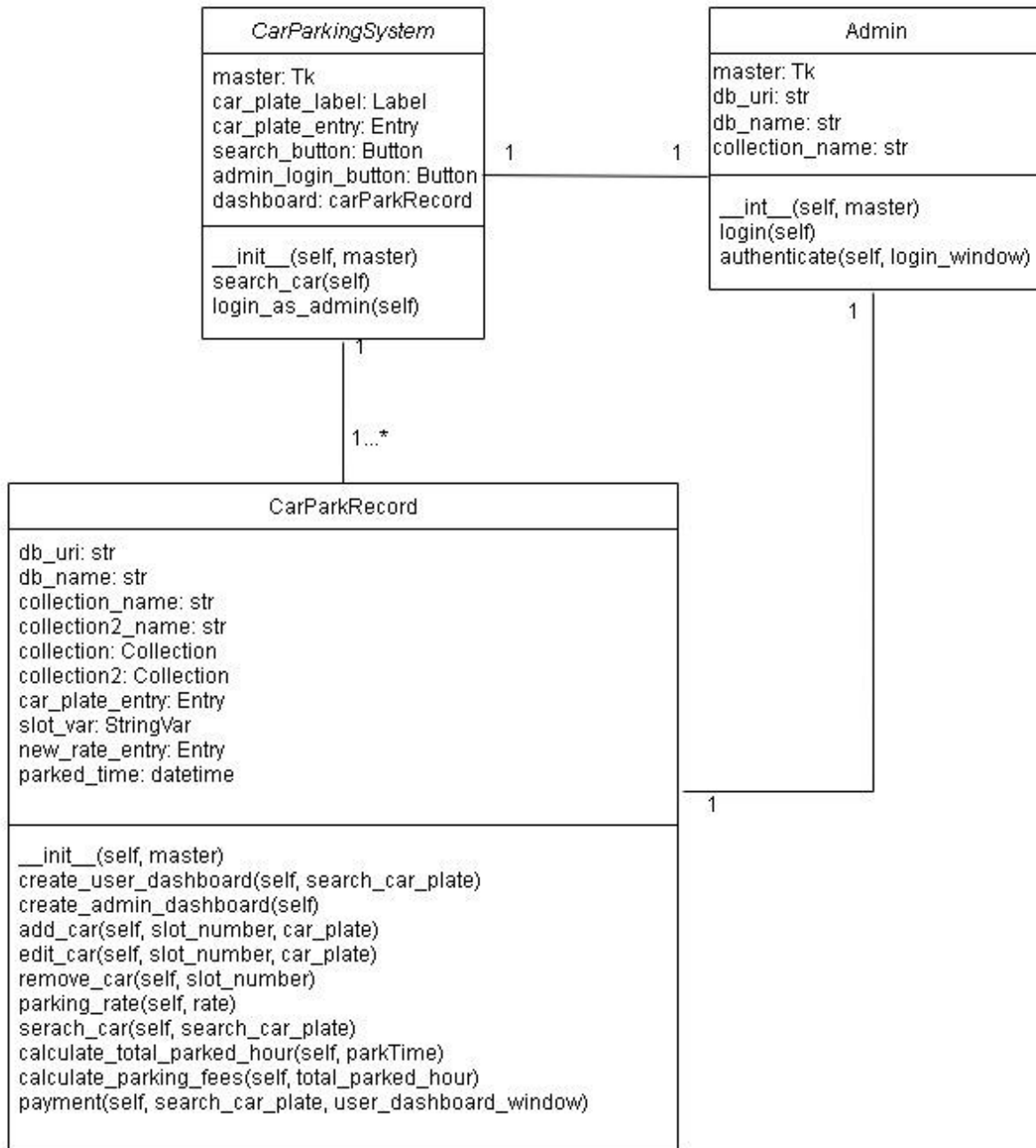


Figure 4.2.2.2.1: Class Diagram <Account Registration>

Implementation:

```
exit_button = tk.Button(self.admin_dashboard_window, text="Exit", command=self.admin_dashboard_window.destroy)
exit_button.pack()
```

Figure 4.2.2.2: Implementation of <Logout>

#### 4.2.3 Module <Add Car to Slot>

##### 4.2.3.1 P003: Package Diagram <Add Car to Slot>

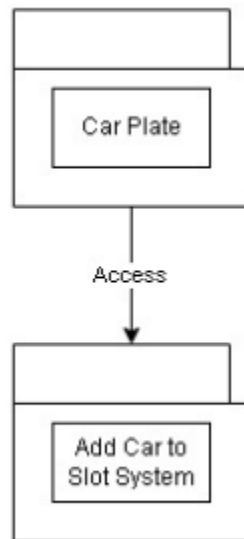


Figure 4.2.3.1: P003: Package Diagram <Add Car to Slot>

#### 4.2.3.2 Class Diagram

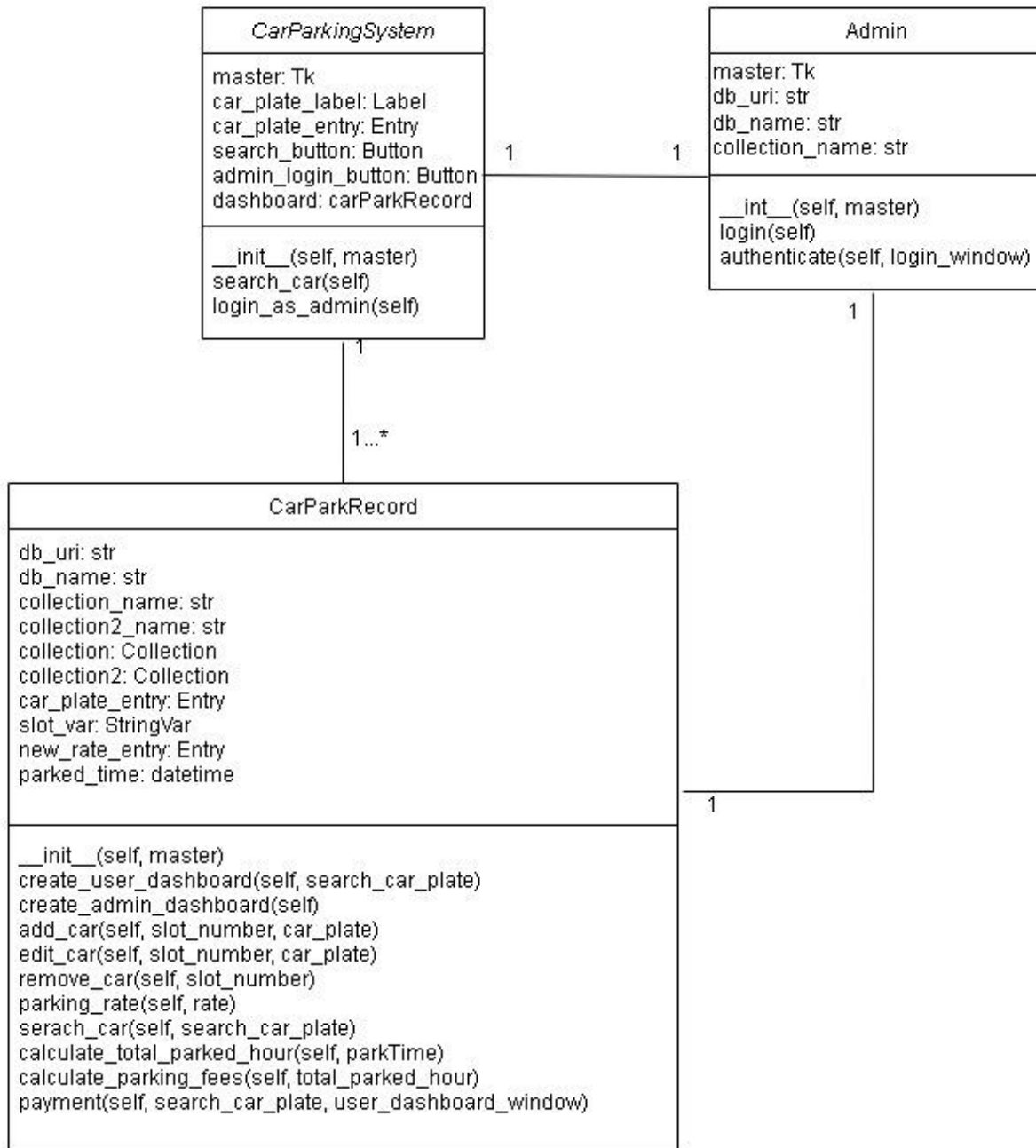


Figure 4.2.3.2.1: Class Diagram <Add Car to Slot>

Implementation:

```

def add_car_to_slot(add_car_window):
    slot_number = self.slot_var.get()
    car_plate = self.car_plate_entry.get().upper().replace(" ", "")

    if not re.match("^[A-Za-z0-9]+$", car_plate):
        messagebox.showwarning("Warning", "Car plate can only contain alphabets and numbers.")
        return

    existing_car = self.collection.find_one({"carPlate": car_plate})
    if existing_car:
        messagebox.showwarning("Warning", "This car plate is already parked.")
        return

    self.car_plate_entry.delete(0, tk.END) # Clear the current text
    self.car_plate_entry.insert(0, car_plate) # Insert the modified plate number
    if not slot_number:
        messagebox.showwarning("Warning", "Please select a slot number.")
        return
    if not car_plate:
        messagebox.showwarning("Warning", "Please enter a car plate number.")
        return
    self.add_car(slot_number, car_plate)
    refresh_dashboard()
    add_car_window.destroy()

```

Figure 4.2.3.4.2: Implementation of Add Car to Slot

#### 4.2.4 Module <Edit Car Info>

##### 4.2.4.1 P004: Package Diagram <Edit Car Info>

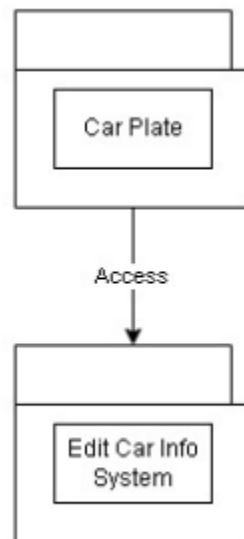


Figure 4.2.4.1.1: P004: Package Diagram <Edit Car Info>



#### 4.2.4.2 Class Diagram

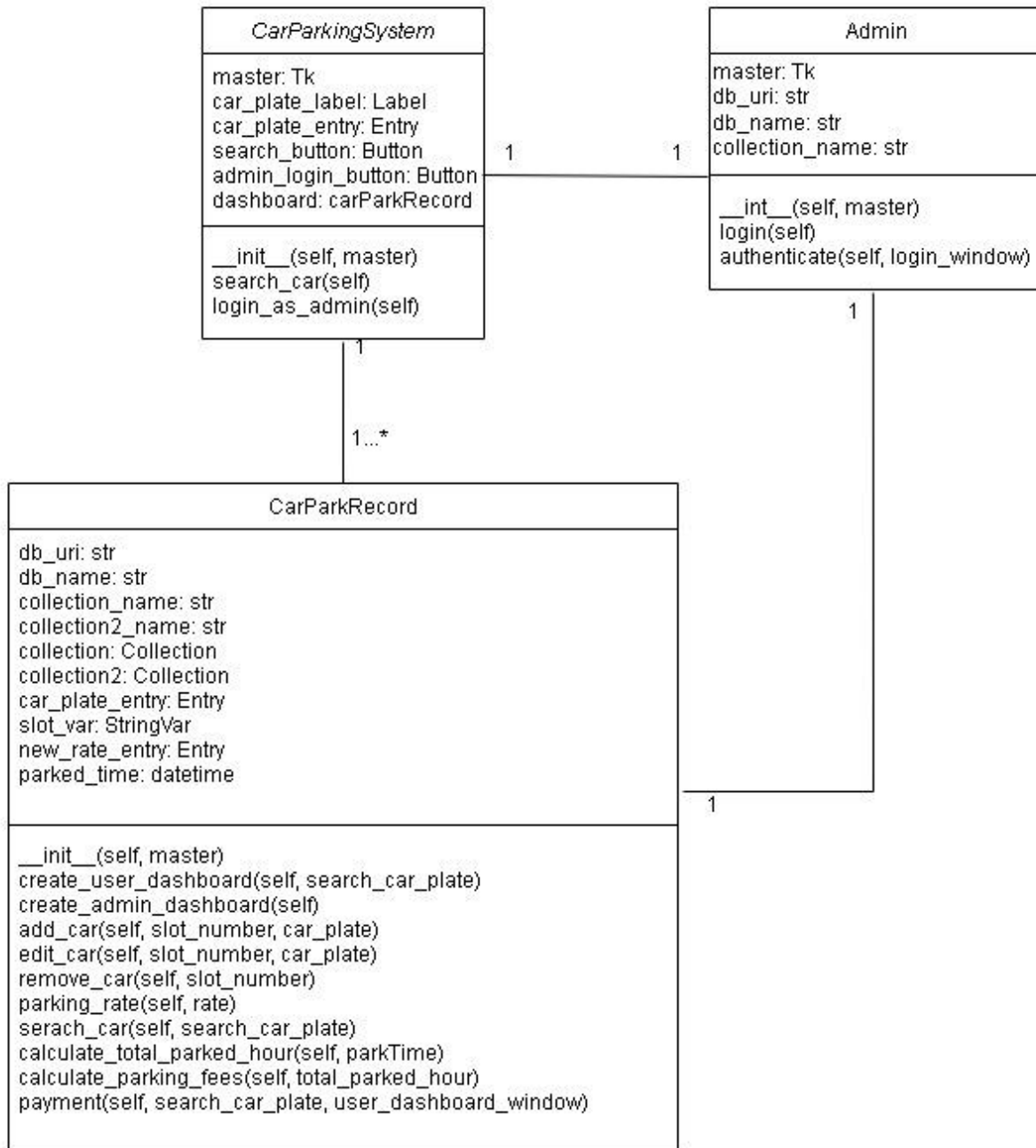


Figure 4.2.4.2.1: Class Diagram <Edit Car Info>

Implementation:

```

# Function to edit car information
def edit_car_info(edit_car_window):
    slot_number = self.slot_var.get()
    car_plate = self.car_plate_entry.get().upper().replace(" ", "")

    if not re.match("^[A-Za-z0-9]+$", car_plate):
        messagebox.showwarning("Warning", "Car plate can only contain alphabets and numbers.")
        return

    existing_car = self.collection.find_one({"carPlate": car_plate})
    if existing_car:
        messagebox.showwarning("Warning", "This car plate is already parked.")
        return

    self.car_plate_entry.delete(0, tk.END) # Clear the current text
    self.car_plate_entry.insert(0, car_plate) # Insert the modified plate number
    if not slot_number:
        messagebox.showwarning("Warning", "Please select a slot number.")
        return
    if not car_plate:
        messagebox.showwarning("Warning", "Please enter a car plate number.")
        return
    self.edit_car(slot_number, car_plate)
    refresh_dashboard()
    edit_car_window.destroy()

```

Figure 4.2.4.2.2: Implementation of <Edit Car Info>

## 4.2.5 Module <Edit Parking Rate>

### 4.2.5.1 P005: Package Diagram <Edit Parking Rate>

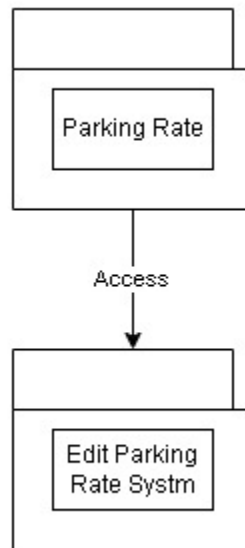


Figure 4.2.5.1.1: P005: Package Diagram <Edit Parking Rate>

#### 4.2.5.2 Class Diagram

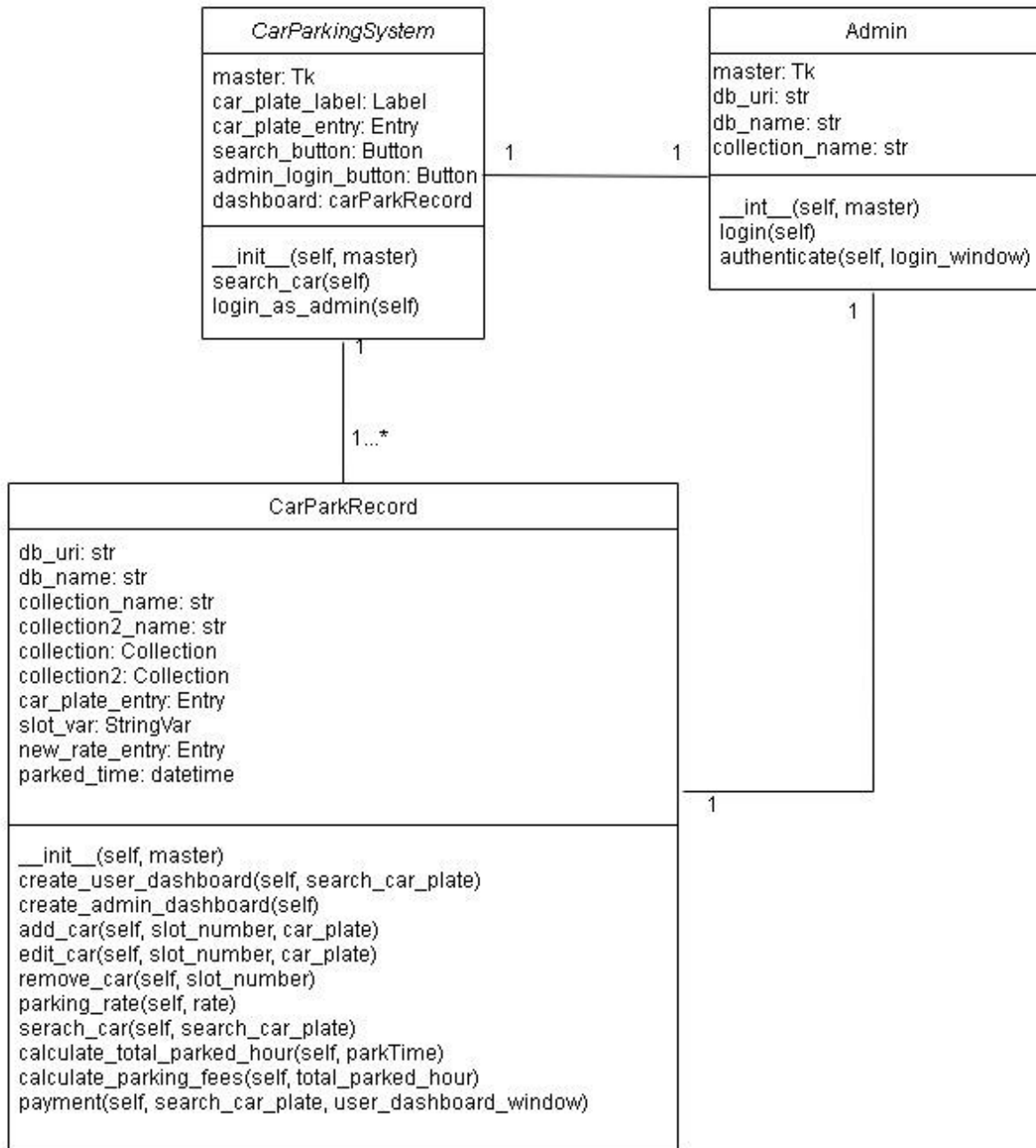


Figure 4.2.5.2.1: Class Diagram <Edit Parking Rate>

Implementation:

```

def edit_parking_rate(edit_parking_rate_window):
    parking_rate = self.new_rate_entry.get()
    try:
        parking_rate = float(parking_rate) # Convert the input to a float
        if parking_rate <= 0:
            messagebox.showerror("Error", "Parking rate must be greater than 0.")
        else:
            parking_rate = '{:.2f}'.format(parking_rate) # Format to two decimal places
            self.parking_rate(parking_rate)
            edit_parking_rate_window.destroy() # Close the window after successful update
    except ValueError:
        messagebox.showerror("Error", "Please enter a valid numeric value for the rate.")

```

Figure 4.2.5.2.2: Implementation of <Edit Parking Rate>

## 4.2.6 Module <Search Vehicle>

### 4.2.6.1 P006: Package Diagram <Search Vehicle>

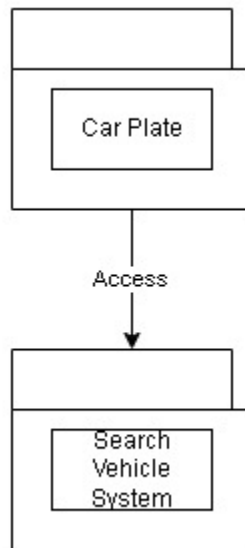


Figure 4.2.6.1.1: P006: Package Diagram <Search Vehicle>

#### 4.2.6.2 Class Diagram

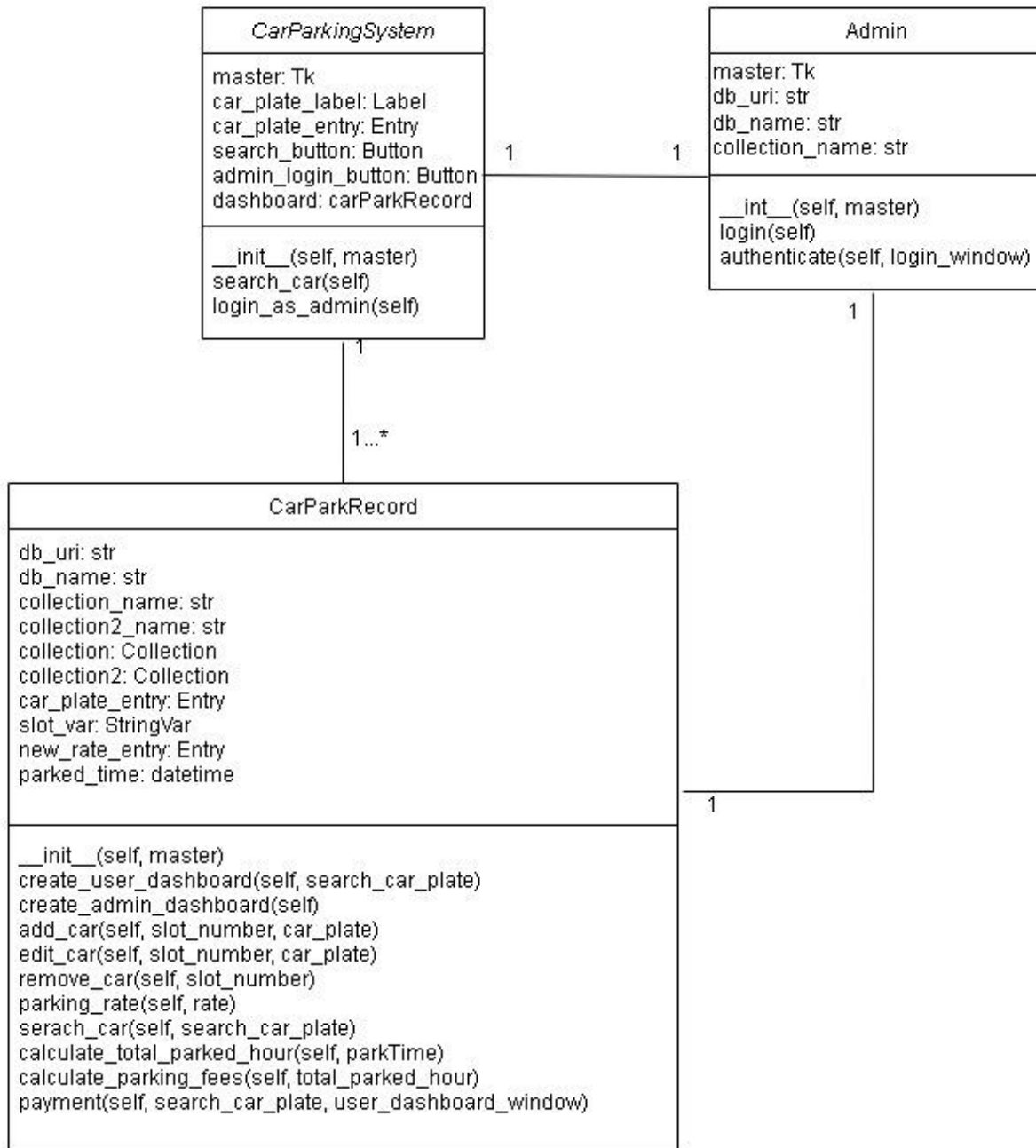


Figure 4.2.6.2.1: Class Diagram <Search Vehicle>

Implementation:

```

def serach_car(self, search_car_plate):
    car = self.collection.find_one({"carPlate": search_car_plate})
    if car:
        #messagebox.showinfo("Search Result", f"Car with plate number '{search_car_plate}' found!")
        self.create_user_dashboard(search_car_plate)
    else:
        messagebox.showwarning("Search Result", f"'{search_car_plate}' not found.")

```

Figure 4.2.6.2.2: Implementation of <Search Vehicle>

## 4.2.7 Module <View Total Parked Hour>

### 4.2.7.1 P007: Package Diagram <View Total Parked Hour>

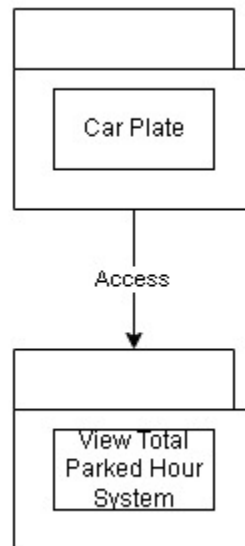


Figure 4.2.7.1.1: P007: Package Diagram <View Total Parked Hour>

#### 4.2.7.2 Class Diagram

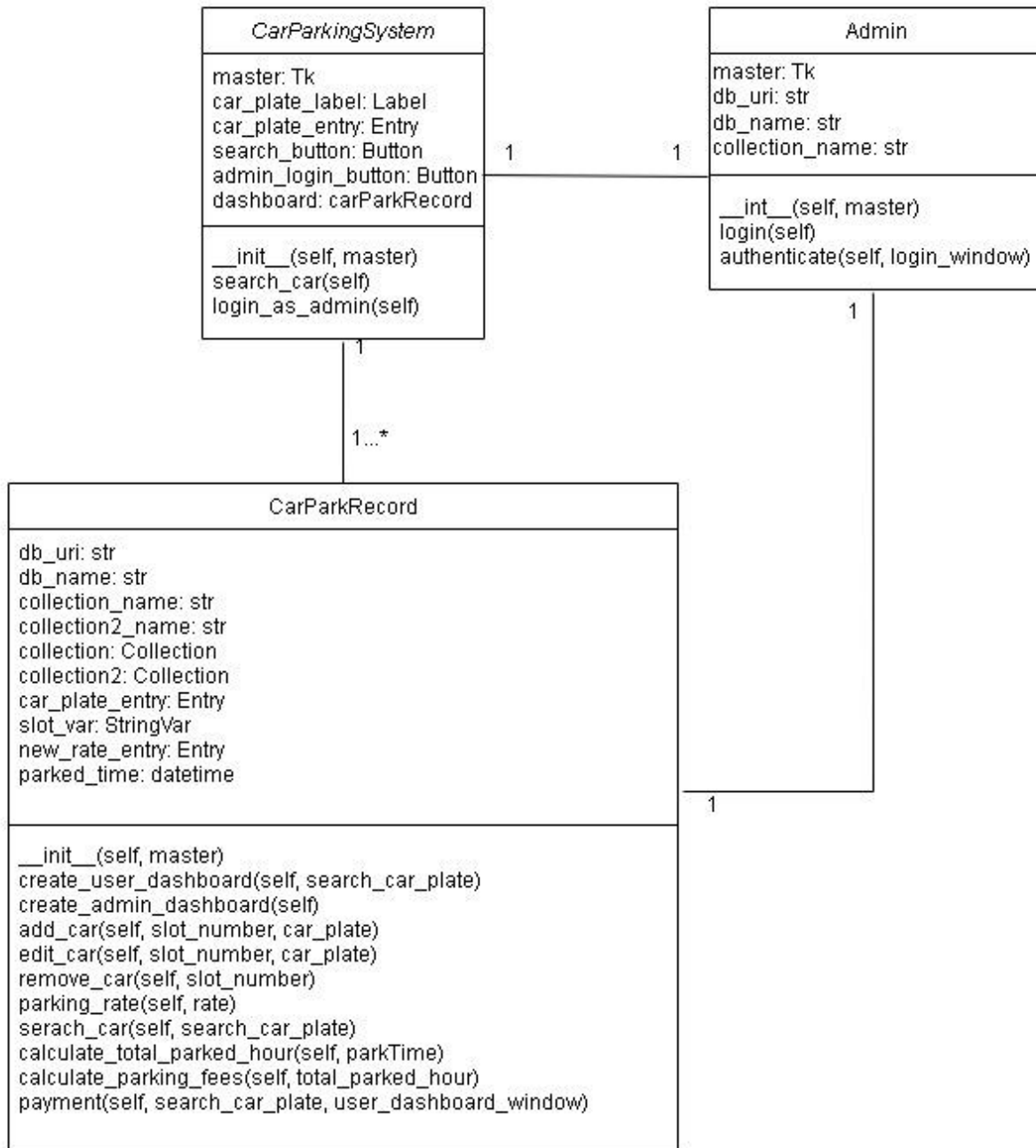


Figure 4.2.7.2.1: Class Diagram <View Total Parked Hour>

Implementation:



```

def calculate_total_parked_hour(self, parkTime):
    current_time = datetime.now()
    parked_time = current_time - parkTime
    hours_parked = parked_time.total_seconds() / 3600 # Convert seconds to hours
    if hours_parked < 1:
        return 1 # If parked for less than an hour, count as 1 hour
    else:
        return int(hours_parked) + 1 # Round up to the next hour

```

Figure 4.2.7.2.2: Implementation of <View Total Parked Hour>

## 4.2.8 Module <View Parking Fees>

### 4.2.8.1 P008: Package Diagram <View Parking Fees>

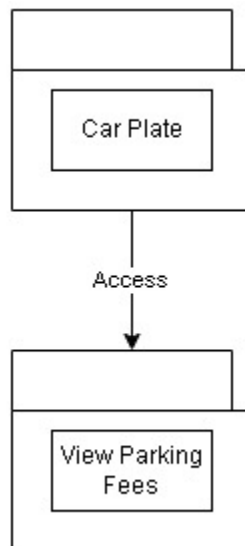


Figure4.2.8.1.1: P008: Package Diagram <View Parking Fees>

#### 4.2.8.2 Class Diagram

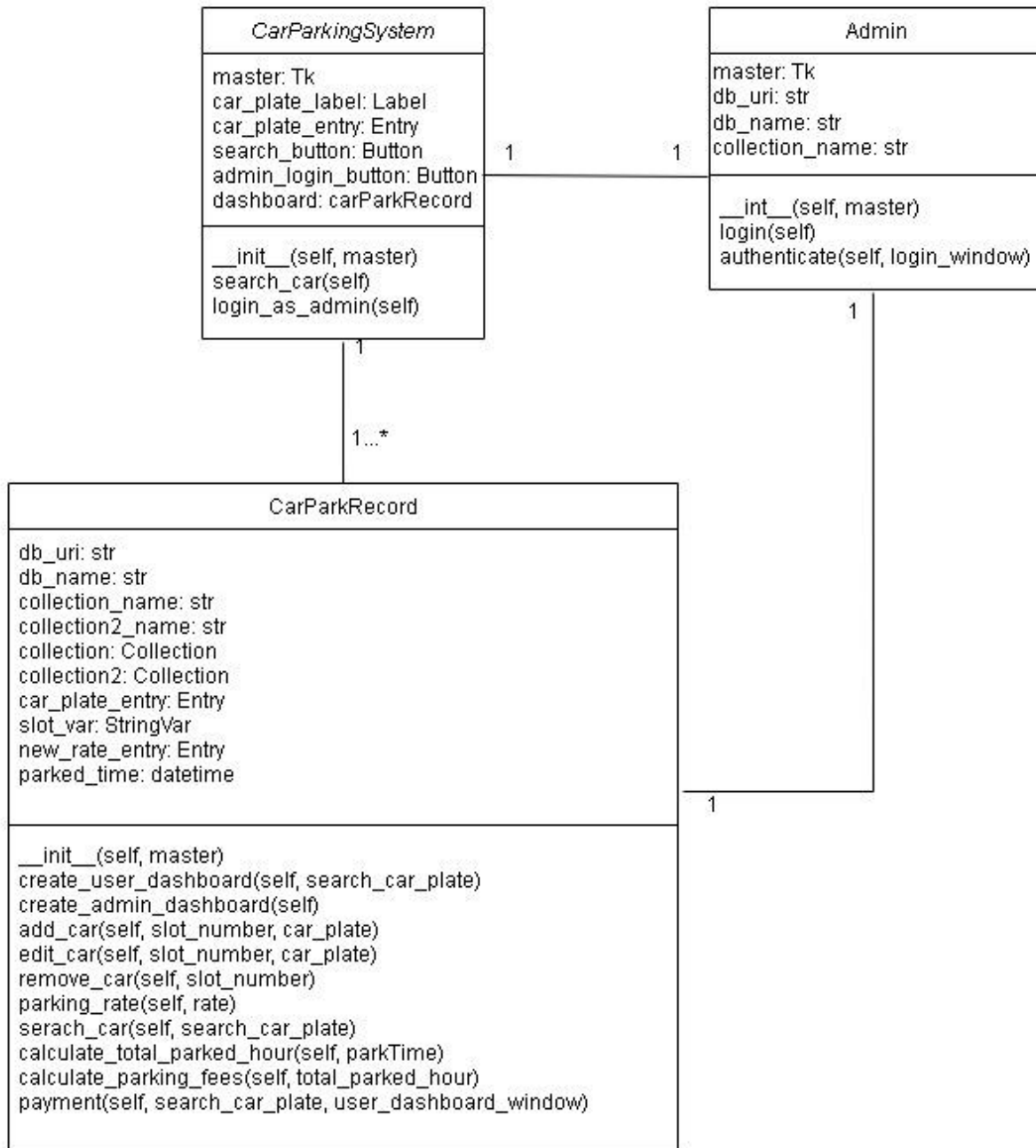


Figure 4.2.8.2.1: Class Diagram <View Parking Fees>

Implementation:

```

def calculate_parking_fees(self, total_parked_hour):
    current_rate_document = self.collection2.find_one()
    if current_rate_document:
        current_rate = current_rate_document.get("rate", 0) # Get the current parking rate
        try:
            current_rate = float(current_rate) # Convert the rate to float
        except ValueError:
            return "Invalid rate"

        try:
            total_parked_hour = float(total_parked_hour) # Convert the total parked hour to float
        except ValueError:
            return "Invalid total parked hour"

        parking_fees = current_rate * total_parked_hour
        parking_fees = '{:.2f}'.format(parking_fees) # Format to two decimal places
        return parking_fees
    else:
        return "Rate document not found"

```

Figure 4.2.8.2.2: Implementation of <View Parking Fees>

## 4.2.9 Module <Make Payment>

### 4.2.9.1 P008: Package Diagram <Make Payment>

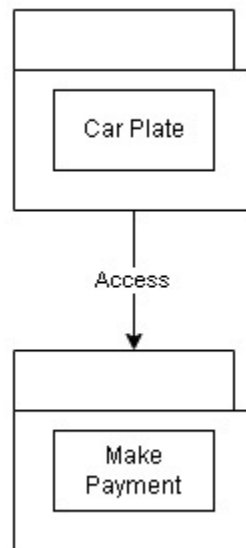


Figure4.2.8.1.1: P008: Package Diagram <Make Payment>

#### 4.2.9.2 Class Diagram

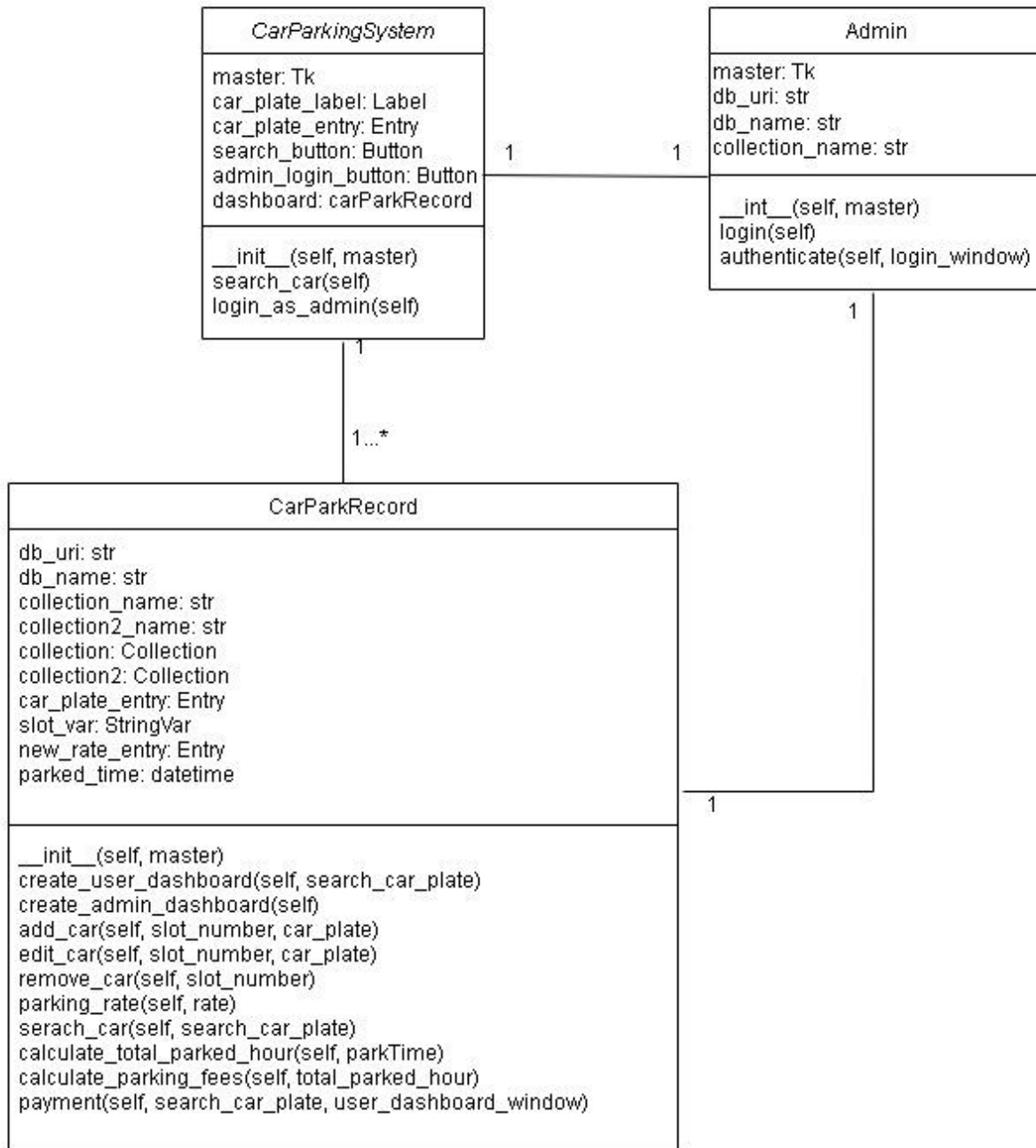


Figure 4.2.8.2.1: Class Diagram <View Parking Fees>

Implementation:

```
def payment(self, search_car_plate, user_dashboard_window):
    messagebox.showinfo("payment successful", f"Payment for '{search_car_plate}' successful")
    self.collection.update_one({"carPlate": search_car_plate},
                               {"$set": {"status": "0", "carPlate": "", "parkTime": ""}})
    user_dashboard_window.destroy()
```

Figure 4.2.8.2.2: Implementation of <Make Payment>

## 5.0 Design Pattern

### 5.1 Design Pattern and Rationale

In the Car Park Management System (CPMS), various functionalities such as user authentication, parking slot management, and transaction handling entail intricate interactions with the MongoDB database. Managing these interactions directly within the application's codebase can lead to increased complexity and reduced maintainability.

To simplify the communication between the CPMS application, we can implement several design patterns:

1. Singleton Design Pattern: The Singleton pattern ensures that a class has only one instance and provides a global point of access to it. In our CPMS, the "CarParkingSystemGUI" class can be encapsulated as a Singleton instance. This ensures that there's only one instance of the "CarParkingSystemGUI" class throughout the application's lifecycle. By enforcing this constraint, we guarantee that there's only a single point of access to the main functionality and state of the car parking system. This Singleton instance of "CarParkingSystemGUI" serves as the central point for managing the entire parking system, including interactions with administrators, parking records, and other subsystems. It optimizes resource usage and maintains consistency across the application by preventing multiple instances from being created. With the Singleton pattern applied to the "CarParkingSystemGUI" class, we ensure that the core

functionality of the parking system remains unified and easily accessible, facilitating efficient management of parking operations and resources.

2. **Facade Design Pattern:** The Facade design pattern in the “CarParkingSystemGUI” class serves as a unified entry point and controller for the entire car parking management system. Acting as a simplified interface, the Facade abstracts the complexities of the underlying subsystems, including the admin module and car parking record module, into a single cohesive unit. At its core, the “CarParkingSystemGUI” Facade encapsulates the intricate details of interactions with different subsystems, shielding clients from the complexities of managing these subsystems individually. This encapsulation not only simplifies the usage of the system but also promotes modularity and extensibility by decoupling clients from the implementation details of subsystems. Through the Facade, clients interact seamlessly with the car parking system, issuing high-level commands and requests without needing to understand the inner workings of the subsystems. This abstraction fosters a more intuitive user experience and facilitates ease of maintenance and future enhancements. Furthermore, the “CarParkingSystemGUI” Facade centralizes control and coordination across the entire system, ensuring consistent behavior and reliable operation. It serves as a centralized hub for managing user interactions, processing requests, and orchestrating communication between subsystems.
3. **Command Design Pattern:** The Command pattern encapsulates a request as an object, thereby allowing parameterization of clients with queues, requests, and operations. In our CPMS, Command objects can be utilized to represent actions such as adding a car to a parking slot, editing car information, and removing a car from the slot. These Command objects encapsulate the necessary data and operations required to execute the corresponding actions. The Command pattern enables decoupling of the requester of an action from the object that performs the action, providing flexibility and extensibility to the system.

By employing the Singleton, Facade, and Command design patterns in the CPMS application, we can effectively manage the complexity of entire system interactions. The Singleton ensures efficient resource utilization, the Facade simplifies system interactions, and the Command pattern provides a flexible and decoupled approach to implementing application functionalities. Overall, these design patterns enhance the maintainability, scalability, and usability of the CPMS application.

## 5.2 Design Pattern Implementation

### 5.2.1 Singleton Design Pattern

```
self.dashboard = carParkRecord(self.master)
self.dashboard.serach_car(car_plate)
```

Figure 5.2.1.1: Implementation of Singleton Design Pattern

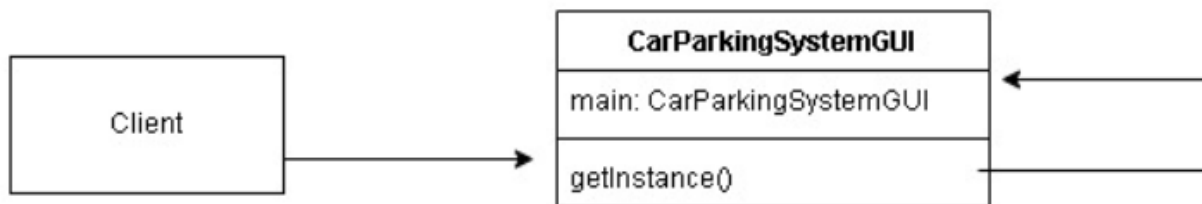


Figure 5.2.1.2: Structure of Singleton Design Pattern



## 5.2.2 Facade Design Pattern

```
class CarParkingSystemGUI:
    def __init__(self, master):
        self.master = master
        self.master.title("Car Parking System")
        self.admin = Admin(master)
        self.master.geometry("400x200") # Set the size of the window

        # Create and pack GUI elements
        self.car_plate_label = tk.Label(self.master, text="Enter Car Plate Number:")
        self.car_plate_label.pack()

        self.car_plate_entry = tk.Entry(self.master)
        self.car_plate_entry.pack()

        self.search_button = tk.Button(self.master, text="Search", command=self.search_car)
        self.search_button.pack()

        self.admin_login_button = tk.Button(self.master, text="Login as Admin", command=self.login_as_admin)
        self.admin_login_button.pack(side=tk.TOP, anchor=tk.NE)

    def search_car(self):
        car_plate = self.car_plate_entry.get().upper().replace(" ", "") # Convert to uppercase and remove spaces
        self.car_plate_entry.delete(0, tk.END) # Clear the current text
        self.car_plate_entry.insert(0, car_plate) # Insert the modified plate number
        self.dashboard = carParkRecord(self.master)
        self.dashboard.serach_car(car_plate)

    def login_as_admin(self):
        self.admin.login()
```

Figure 5.2.2.1 Implementation of Facade Design Pattern

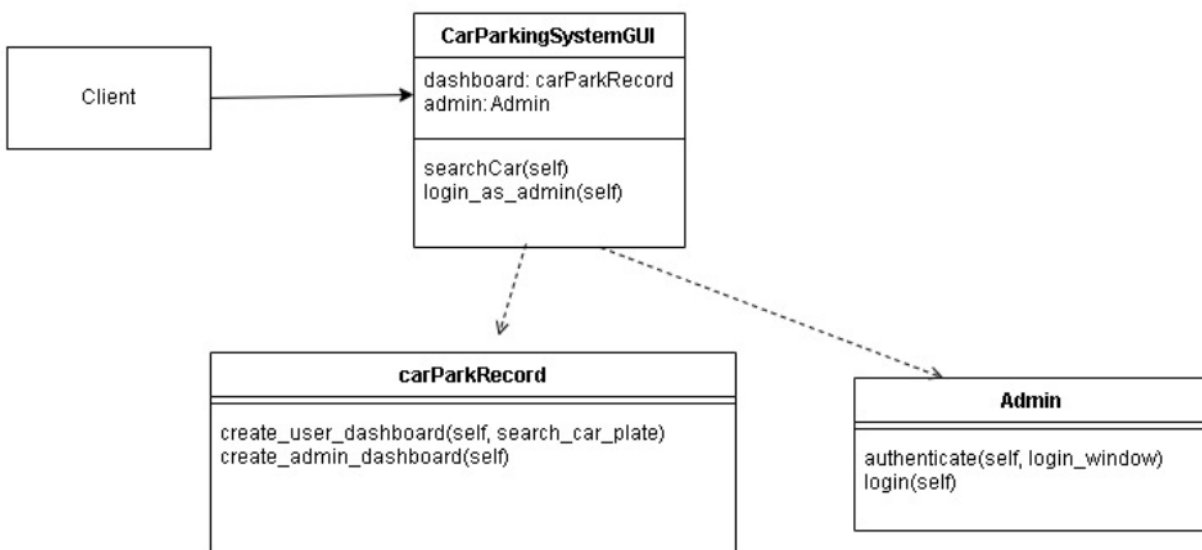


Figure 5.2.2.1 Structure of Facade Design Pattern

### 5.2.3 Command Design Pattern

```
self.search_button = tk.Button(self.master, text="Search", command=self.search_car)
self.search_button.pack()

self.admin_login_button = tk.Button(self.master, text="Login as Admin", command=self.login_as_admin)
self.admin_login_button.pack(side=tk.TOP, anchor=tk.NE)
```

Figure 5.2.3.1 Implementation of Command Design Pattern

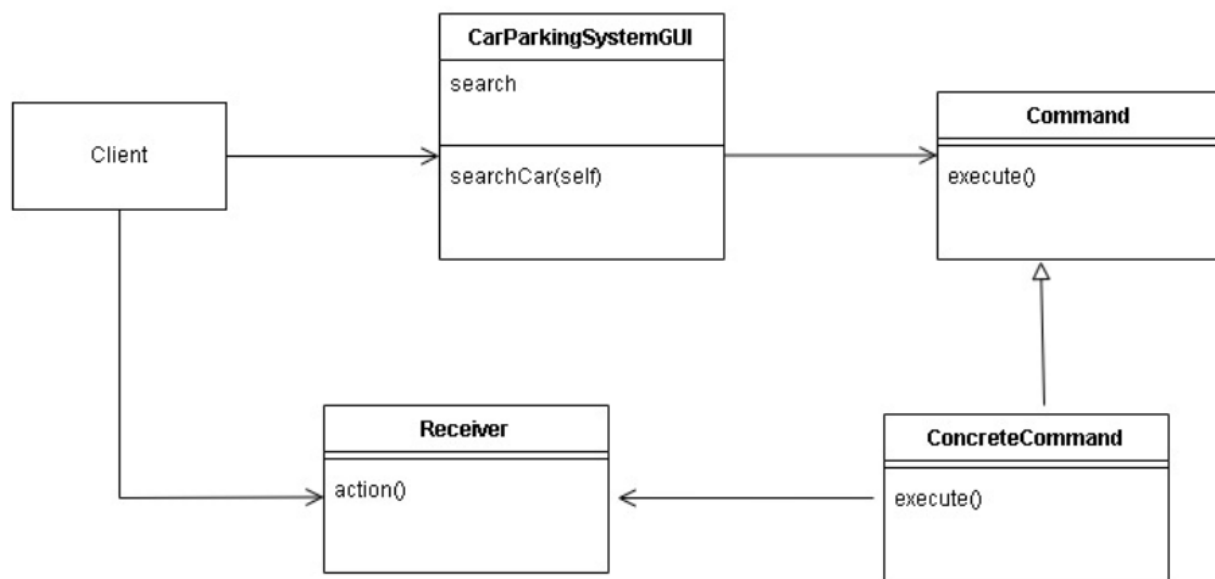


Figure 5.2.3.2 Structure of Command Design Pattern

## 6.0 Data Design

### 6.1 Data Description

To support the functionality of the Car Park Management System (CPMS), a well-structured data model is essential to ensure effective operation and maintainability. It's crucial to apply systematic analysis principles to both functional aspects and data representation. This involves defining data flow and content, identifying relevant data entities, and considering alternative data structures that may impact system design.

In our implementation, MongoDB serves as the primary data store for the CPMS. MongoDB stores data in collections, which contain documents organized in a flexible, schema-less format. This NoSQL database offers advantages such as flexibility, scalability, and ease of integration with our Python-based system.

Key features of MongoDB for CPMS include:

1. **Document-Oriented Storage:** MongoDB stores data in JSON-like documents, allowing for flexible schema designs and seamless handling of complex data structures.
2. **Scalability:** MongoDB's distributed architecture enables horizontal scalability, allowing the CPMS to handle increasing data volumes and user traffic efficiently.
3. **Real-Time Data Sync:** MongoDB supports real-time data synchronization, ensuring that changes made to the database are immediately reflected across all connected instances. This feature is valuable for real-time monitoring and reporting in the CPMS.
4. **Integration with Python:** MongoDB offers robust drivers and libraries for Python, facilitating seamless integration with our Python-based CPMS implementation.
5. **Web-Based Administration:** MongoDB provides a user-friendly web-based interface, MongoDB Compass, for managing the database. This tool allows administrators to visualize data, run queries, and perform administrative tasks efficiently.

By leveraging MongoDB as the underlying data store for CPMS, we ensure efficient data storage, retrieval, and management. The flexibility and scalability of MongoDB empower CPMS to adapt to changing requirements and handle large volumes of parking-related data effectively. Additionally, MongoDB's real-time data synchronization capabilities enable real-time monitoring and reporting, enhancing the overall functionality and usability of the CPMS.

Below are the list of the database(s) or data storage items.

username	password
----------	----------

Table 3.1: Collection - adminCredential

rate
------

Table 3.2: Collection - parkingRate

slot	carPlate	parkTime	status
------	----------	----------	--------

Table 3.3: Collection - parkingRecord

## 6.2 Data Dictionary

The table (Table 3.4) below list the system entities or major data along with their types and descriptions.

Table Name	Field Item	Data Type	Data Format	Field Size
adminCredential	username	string		254
adminCredential	password	string		254
parkingRate	rate	string		254
parkingRecord	slot	string		254
parkingRecord	carPlate	string		254
parkingRecord	parkTime	date		7
parkingRecord	status	string		254

Table 3.4: Data Dictionary Table

## 7.0 User Interface Design

## **7.1 Overview of User Interface**

The CPMS interface encompasses several key components designed to streamline user interactions and administrative tasks. The CPMS greets users with a login interface upon accessing the application. This interface serves as the gateway for both customers and administrators to access their respective accounts and functionalities within the system. By providing a secure and user-friendly login process, the CPMS ensures authorized access to the system's features.

Upon successful login, administrators are equipped with an admin dashboard equipped with advanced functionalities for overseeing parking operations. This dashboard grants administrators' full control over parking slots, car records, and parking rates. With features such as adding, editing, and removing cars from parking slots, as well as updating parking rates, the admin dashboard facilitates efficient management of parking facilities and ensures optimal utilization of resources.

Upon found searched car plate, end users are presented with a user dashboard tailored to their needs. This dashboard serves as a centralized hub for displaying crucial information such as parked car details, total parked hour, and parking fees. By offering a clear and intuitive interface, the user dashboard empowers customers to monitor their parking status and proceed with payment seamlessly.

The CPMS provides a dedicated payment window for users to complete their transactions securely. This window displays the total payment amount and offers flexible payment methods such as credit/debit card or e-wallet. By simplifying the payment process and offering multiple payment options, the CPMS enhances user convenience and ensures a seamless checkout experience.

Throughout the interface, users encounter functionality buttons such as "Proceed to Payment" and "Exit" to navigate through the application. These buttons enable users to perform various actions such as initiating payment or closing the application window, enhancing overall usability and user engagement within the CPMS.

Various input fields and dropdown menus are strategically placed within the interface for user interaction. These elements allow users to input essential information such as car plate numbers, select parking slots, and specify new parking rates, facilitating smooth data entry and customization of preferences.

The CPMS interface incorporates error messages and notifications to provide real-time feedback to users. For instance, warnings may be displayed for invalid input or notifications for successful operations such as payment completion. By offering clear and concise feedback, the CPMS enhances user confidence and minimizes user errors during interactions with the system.

## 7.2 Screen Images

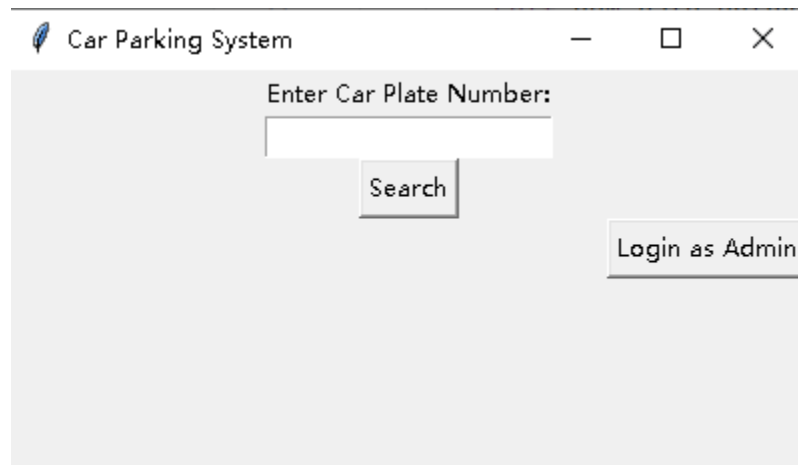
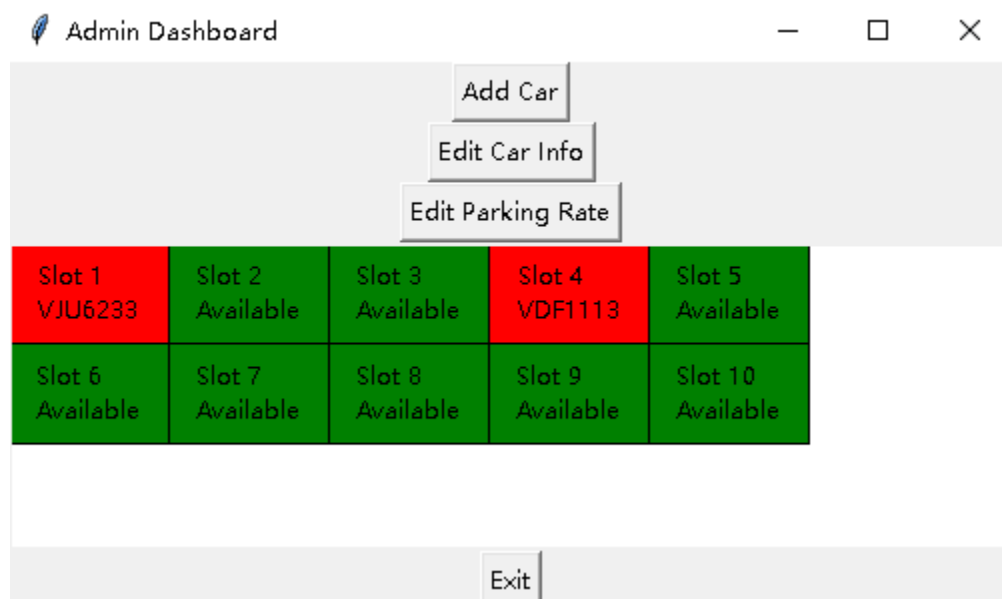


Figure 4.1: Interface of Car Parking System Main Page



A screenshot of a Java Swing window titled "Admin Login". The window has a light gray background and standard window controls (minimize, maximize, close) in the title bar. It contains two text input fields: the first is labeled "Username:" and the second is labeled "Password:". Below the password field is a "Login" button.

Figure 4.2: Interface of Login Page



A screenshot of a Java Swing window titled "Admin Dashboard". The window has a light gray background and standard window controls. It features three buttons stacked vertically: "Add Car", "Edit Car Info", and "Edit Parking Rate". Below these buttons is a table with 10 slots, arranged in two rows of five. The first row has a red background for Slot 1 and Slot 4, and a green background for Slot 2, Slot 3, and Slot 5. The second row has a green background for all slots. Slot 1 contains the text "Slot 1" and "VJU6233". Slot 4 contains the text "Slot 4" and "VDF1113". All other slots contain the text "Slot X" and "Available". Below the table is an "Exit" button.

Slot 1 VJU6233	Slot 2 Available	Slot 3 Available	Slot 4 VDF1113	Slot 5 Available
Slot 6 Available	Slot 7 Available	Slot 8 Available	Slot 9 Available	Slot 10 Available

Figure 4.3.: Interface of Admin Dashboard

Select Slot:

2

Enter Car Plate Number:

Add

Figure 4.4: Interface of Add Car to Slot Page

Select Slot:

1

Enter Car Plate Number:

Update Car Info

Remove Car

Figure 4.5: Interface of Edit Car Info Page

Current Rate (RM): 1.00

Enter New Rate (RM):

Update Parking Rate

Figure 4.6: Interface of Edit Parking Fees Page



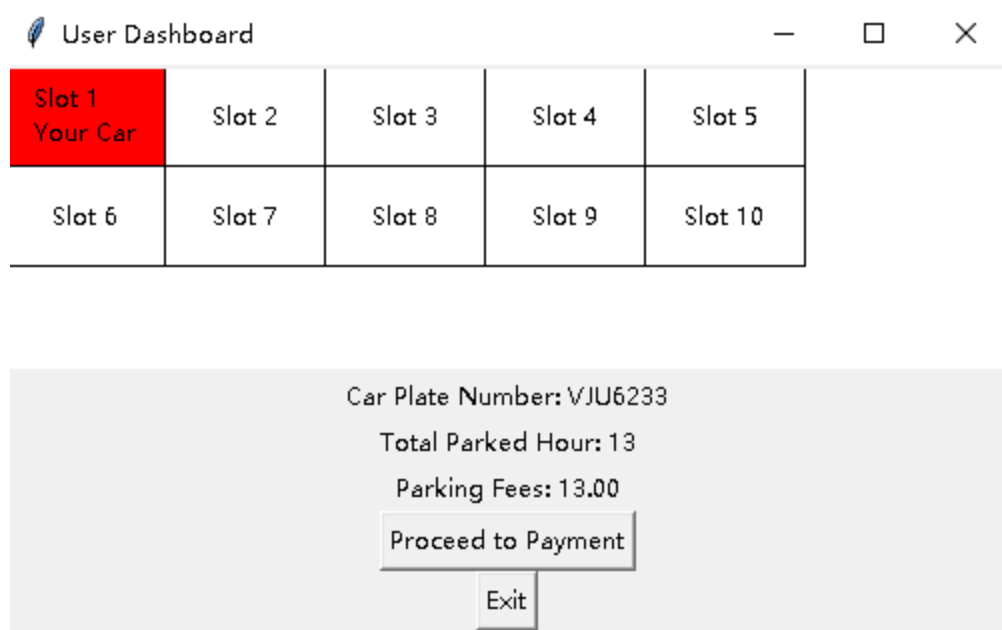


Figure 4.7: Interface of Search Result Page

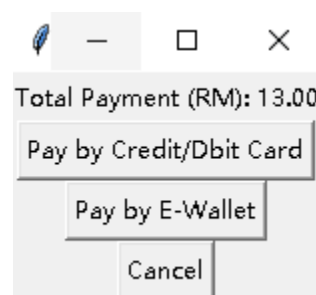


Figure 4.8: Interface of Make Payment Page