**SDD**

Parking Garage Project

Software Design Document

# Revision History

| Date | Revision | Description | Author |
| --- | --- | --- | --- |
| 3/22/2025 | 1.0 | Started adding content to the SDD from SRS. | Kurt, Vishal, Raymond |
| 4/3/2025 | 2.0 | Expanded on content in SDD and added class candidates. | Kurt, Vishal, Raymond |
| 4/4/2025 | 3.0 | Expanded initial class diagram and created more sequence and use case diagrams to cover more in depth. | Kurt, Vishal, Raymond |
| 4/6/2025 | 4.0 | Worked on some class candidates and use cases. | Kurt, Vishal, Raymond |
| 4/9/2025 | 5.0 | Added some missing class candidates and updated class diagrams. | Kurt, Vishal, Raymond |

# Table of Contents

[Revision History 2](#_ffcjvve8rkh1)

[Table of Contents 3](#_cl1tmj2disvg)

[● Introduction 5](#_5el1inidzkoa)

[● System Overview 6](#_7bkvluik8pns)

[Entry Module: 6](#_tid45t9733vg)

[● Automated Fee Calculation: 7](#_hzdri0dnyys5)

[● Issuing Penalties: 7](#_myxq891qx1ts)

[● Types of Logged Data: 7](#_dxu6yhao7lh)

[● Data Accessibility: 8](#_yzxwy26gulbp)

[● User Management: 8](#_iruk1zl78nso)

[● System Configuration: 8](#_gn0nqcdwsurc)

[● Maintenance Alerts: 8](#_pkt9se0aatu)

[● Multi-Garage Management: 8](#_2zhh9l20051c)

[● Use Cases 9](#_yf1v1k3wn70w)

[1. Use Case: Login and Logout 9](#_yoxt4wq97c99)

[2. Use Case: Ticket Handling 9](#_z891yjif5shp)

[3. Use Case: Handling Parking Data 10](#_xx4u9rbq1r0m)

[4. Use Case: Payment Processing 10](#_lpslftu354v7)

[5. Use Case: Automatic vs. Manual Payment Handling 11](#_xppx7ykbgzqe)

[6. Use Case: Add Parking Levels 11](#_migiv033f58l)

[UML Use Case Diagrams Document 13](#_di1b5bmitpy4)

[1. Use Case: Login and Logout 13](#_9ipg5hxlouoe)

[2. Use Case: Ticket Handling 13](#_3qzgzd8m47zm)

[3. Use Case: Handling Parking Data 14](#_1j1wt3fyo6e)

[4. Use Case: Payment Processing 14](#_gynrsha139tv)

[5. Use Case: Automatic vs. Manual Payment Handling 15](#_si2md9gimctk)

[6. Use Case: Add Parking Levels 15](#_qkn8ww60v9f6)

[Class Diagrams 16](#_z43vm5817sbq)

[Class Diagram: 16](#_jh0667uqctlr)

[Sequence Diagrams 17](#_xbho7s906lrv)

[Sequence Diagram - Employee Side 18](#_f3p3yce0r79p)

[Sequence Diagram for Use Case 1: Login and Logout 19](#_yl9s42336mcj)

[Sequence Diagram for Use Case 2: Ticket Handling 20](#_ndm7fnsj8su5)

[Sequence Diagram for Use Case 3: Handling Parking Data 21](#_6jj6ytnmj7r7)

[Sequence Diagram for Use Case 4: Payment Processing 22](#_g8vqhob4fxw4)

[Sequence Diagram for Use Case 5: Automatic vs. Manual Payment Handling 23](#_57i8ulgzpvei)

[Sequence Diagram for Use Case 6: Add Parking Levels 24](#_8xxrccay5tw)

[● Classes and Objects 25](#_w0dchs3e9l74)

[ParkingGarage 25](#_ws05y23n9xup)

[ParkingLevel 25](#_mtxjf3p7thur)

[ParkingSpace 26](#_uzgvotsu8gkl)

[EntranceDisplayBoard 26](#_scs6cpm9ippv)

[EntryKiosk 26](#_ucq5ewld7g6h)

[ExitKiosk 26](#_vgxfx62k0tx)

[Ticket 27](#_xutuc7ed8tjn)

[Payment 27](#_k4zy2ovy924n)

[Employee 28](#_k08pizpl8oau)

[ParkingAttendant extends Employee 28](#_g6a49pj0um7u)

[Admin extends Employee 28](#_8xu4di64kya7)

[Customer 29](#_7dc2bs4t29e0)

[SystemLog 29](#_arqatyf5rcnx)

[Hardware 30](#_c3notclzi5eb)

[Gate 30](#_farhiouiet76)

[Client 30](#_6gkmxdl688wj)

[Server 31](#_ijoajlzb4s79)

[ClientHandler 31](#_913hvj8jtyg4)

[GUI 32](#_2u991y7yqrq5)

[ParkingGarageSystemFacade 32](#_gbib0a0mkoi)

# Introduction

1.1. Overview

The parking garage software delivers administrative logging reports and a graphical user interface (GUI) for users. The software will be integrated to hardware components such as automated payment kiosks, tracking monitors/sensors, hardware components for entering/exiting barriers and as well as parking garage’s work tablets.

1.2. Purpose

The parking garage software is intended to systematically manage and streamline an overhead for parking garage operations by tracking all vehicles entering and exiting, calculating cost, monitoring available parking spaces, and logging facility-related reports.

1.3 Scope

The system allows customers to enter and exit the parking garage, track the available spaces, process payments, and generate usage reports. The system will also include features such as automated ticketing, real-time space monitoring, and integration through our payment system.

1.4 Target Audience

1.4.1. Investors seeking to bring the parking garage system

1.4.2. Potential business associates such as hardware solution manufacturers and parking resources.

1.4.3. Neighboring businesses

1.4.4. Entertainment venues, sports stadiums, and large event spaces

# System Overview

2.1. Software System’s functionality

The Vehicle Entry & Exit Management module is responsible for handling vehicle movement within the parking facility. It ensures smooth operations at entry and exit points using automated ticketing and fee calculation mechanisms.

#### Entry Module:

* Upon arrival, the system detects an incoming vehicle using RFID scanners, or a ticket dispenser.
* If the garage has available space, the system generates a unique ticket with the following details:
  + Ticket ID (Unique Identifier)
  + Entry Time
  + Exit Time
* The barrier gate automatically opens for the vehicle to enter after a ticket has been issued.

Exit Module:

When a vehicle approaches the exit, the dispenser takes back the ticket to scan internally.

* The system calculates the total parking duration and fee based on a predefined rate.
* Payment is requested via automated kiosks or manual employee processing.
* Upon successful payment, the system:
  + Updates the garage database to free up the occupied parking space.
  + Generates a digital or printed receipt.
  + Opens the exit barrier for the vehicle to leave.
* If a vehicle exits without a valid ticket, an employee override option is available to manually process the exit.

2[.1.1. Vehicle Entry & Exit Management](#)

* Entry module: Issuing/Generating a ticket and prints a ticket containing the start time, parking location, and a unique id.
* Exit module: Calculates fees and processes payments.

2[.1.2. Parking Space Management](#)

* Real-time Availability Tracking: Monitoring guest entering and exiting the parking facility, we are able to provide the status of available parking.
* The system updates availability status dynamically and displays real-time capacity on an electronic board placed near the entrance.
* Parking zones can belong in different categories (general parking, reserved spots, electric vehicle charging stations, handicapped spots) for better customer satisfaction.
* If a discrepancy is detected (e.g., a vehicle leaves but the space is still marked as occupied), an alert is sent for manual verification.

2[.1.3. Payment Processing](#)

#### Automated Fee Calculation:

* + The parking fee is determined based on:
    - Duration of stay (time-in vs. time-out).
    - Parking zone (regular, premium, reserved).
  + The system automatically computes and displays the payable amount upon ticket scanning.
* Enables Multiple Payment Methods:
  + Automated Payments:
    - Credit/Debit Cards (via self-service kiosks or online portal).
    - Mobile Payments (Google Pay, Apple Pay, etc.).
  + Manual Payments:
    - Cash payments at the employee checkout counter.
    - Employee-assisted card transactions.

#### Issuing Penalties:

* + Lost Ticket Fee: If a customer loses their ticket, an employee must manually retrieve entry details and charge a default fee.
  + Overstay Charges: Vehicles parked beyond permitted hours will be charged extra, as per garage policy.
  + Unauthorized Parking Violations: If a vehicle is parked in a restricted zone, penalties are recorded and linked to the ticket.

2[.1.4. Logging and Garage Report](#)s

The system continuously logs transactions, occupancy data, and security events to provide detailed insights for management.

#### Types of Logged Data:

* + Occupancy Reports:
    - Daily, weekly, and monthly reports showing parking utilization trends.
  + Revenue Reports:
    - Total revenue generated per time period, categorized by payment method.
    - Breakdown of penalties and manual adjustments.
  + Employee Activity Reports:
    - Login/logout timestamps for employees handling transactions.
    - Employee overrides (e.g., manually processed exits).
  + Security Logs:
    - Unauthorized access attempts.
    - Sensor malfunctions and system errors.

#### Data Accessibility:

* + Admins and managers can generate reports through a secured web interface.
  + Reports can be exported in multiple formats (PDF, CSV, Excel).
  + System logs are archived periodically for auditing and compliance.

2[.1.5. Administration and Maintenance](#)

This module provides administrative control over system settings, configurations, and maintenance operations.

#### User Management:

* + Employee access is role-based, granting permissions based on job responsibilities where managers can override employees and gain more access for logs and etc.
  + Credentials are required for login.

#### System Configuration:

* + Parking rates, penalties, and policies can be configured through the admin dashboard.
  + Hardware components (sensors, gates, kiosks) can be monitored remotely for malfunctions.

#### Maintenance Alerts:

* + The system detects and logs any software malfunctions and bugs by handling errors.
  + Admins receive automatic notifications when maintenance is required.
  + A maintenance tracking system logs completed repairs and scheduled servicing.

#### Multi-Garage Management:

* + If multiple parking locations are managed under the same system, administrators can monitor all garages from a central dashboard.
  + Reports can be generated per garage or aggregated across multiple locations.

# Use Cases

## **1. Use Case: Login and Logout**

Use Case ID: UC-01  
Relevant Requirements: Employee Login, Graphical User Interface  
Primary Actor: Employee (Attendant, Manager)  
Pre-conditions: The system must be powered on; the user must have valid login credentials.  
Post-conditions: The user is successfully logged in or out of the system.

Basic Flow:

1. The user navigates to the login page.
2. The user enters their username and password.
3. The system verifies the credentials.
4. If valid, the system grants access based on the user's role.
5. The user is redirected to the dashboard.
6. When finished, the user selects the logout option, and the session ends.

Extensions/Alternate Flows:

* If credentials are incorrect, the system displays an error message.
* If three failed attempts occur, the account is locked for security.

Exceptions:

* System error preventing login.
* Unexpected session timeout during login.

Related Use Cases: Ticket Handling, Payment Processing

## **2. Use Case: Ticket Handling**

Use Case ID: UC-02  
Relevant Requirements: Ticketing System, Parking Space Tracking  
Primary Actor: System, Customer  
Pre-conditions: The parking garage has available space; the system is active.  
Post-conditions: The customer receives a ticket, and a parking space is then occupied.

Basic Flow:

1. The customer arrives at the parking garage entry.
2. The system checks for available parking spaces.
3. If a spot is available, the system generates a ticket.
4. The system assigns a parking space and updates the total capacity.
5. The ticket is printed and given to the customer.

Extensions/Alternate Flows:

* If the garage is full, the system displays a "Garage Full" message and does not issue a ticket.
* If the ticket printer fails, an employee must manually generate a ticket.

Exceptions:

* System error preventing ticket issuance.

Related Use Cases: Handling Parking Data, Payment Processing

## **3. Use Case: Handling Parking Data**

Use Case ID: UC-03  
Relevant Requirements: Parking Space Tracking, Multi-Garage Management  
Primary Actor: System, Employee  
Pre-conditions: The system is running and actively tracking vehicles.  
Post-conditions: The parking availability data is updated accurately.

Basic Flow:

1. A vehicle enters or exits the parking garage.
2. The system updates the number of occupied and available spaces.
3. The system assigns or frees up a parking space based on the transaction.
4. The system logs the event for tracking purposes.

Extensions/Alternate Flows:

* If a sensor fails to detect a vehicle, the system logs an error.
* Employees can manually adjust parking counts in case of a discrepancy.

Exceptions:

* System communication failure with parking sensors.

Related Use Cases: Ticket Handling, Payment Processing

## **4. Use Case: Payment Processing**

Use Case ID: UC-04  
Relevant Requirements: Payment Method, Parking Duration  
Primary Actor: Customer, Employee, System  
Pre-conditions: The customer has a valid ticket and is ready to leave.  
Post-conditions: The payment is processed, and the exit gate opens.

Basic Flow:

1. The customer drives to the exit gate.
2. The system scans the ticket and calculates the fee based on duration.
3. The system prompts for payment.
4. The customer completes the payment using an accepted method (cash, card).
5. Upon successful payment, the system opens the exit gate.

Extensions/Alternate Flows:

* If the customer lost their ticket, an employee manually retrieves parking data and processes the payment.
* If the payment terminal fails, the customer must use another terminal or pay manually.

Exceptions:

* System error preventing payment processing.

Related Use Cases: Automatic vs. Manual Payment

## **5. Use Case: Automatic vs. Manual Payment Handling**

Use Case ID: UC-05  
Relevant Requirements: Employee Charging, Payment Processing  
Primary Actor: Customer, Employee, System  
Pre-conditions: The system has ticket data stored for each customer.  
Post-conditions: The customer is charged correctly based on the method used.

Basic Flow (Automatic Payment):

1. A customer with an auto-payment account is detected at the exit.
2. The system verifies the linked payment method and calculates the fee.
3. The system processes the payment automatically.
4. If successful, the exit gate opens.

Basic Flow (Manual Payment):

1. A customer without auto-payment drives to an employee checkout booth.
2. The employee scans the ticket and retrieves the parking fee.
3. The customer makes the payment.
4. The employee confirms payment and allows exit.

Extensions/Alternate Flows:

* If automatic payment fails, the system prompts for manual payment.
* If the employee is unable to process a payment, the customer must use another payment method.

Exceptions:

* System failure preventing payment authorization.

Related Use Cases: Payment Processing, Ticket Handling

## **6. Use Case: Add Parking Levels**

Use Case ID: UC-06  
Relevant Requirements:  
Primary Actor: Admin  
Pre-conditions: Admin is authenticated and logged into the system. The system is operational and connected to the central database or employee files.  
Post-conditions: The new parking level or spaces are successfully added and visible in the system. The garage layout is updated. Reported logs are updated to reflect the changes made to the system.

Basic Flow (Automatic Payment):

1. Admin selects "Add Parking Level" from the system dashboard.
2. System prompts for:
   * Level ID or Name
   * Number of parking spaces
   * Level-specific attributes (e.g., access restrictions, vehicle type support)
3. Admin enters the required information.
4. Admin clicks "Submit".
5. System validates input.
6. System adds the new level and initializes a specified number of empty parking spaces.
7. Confirmation is displayed to the admin.

Extensions/Alternate Flows:

* If automatic payment fails, the system prompts for manual payment.
* If the employee is unable to process a payment, the customer must use another payment method.

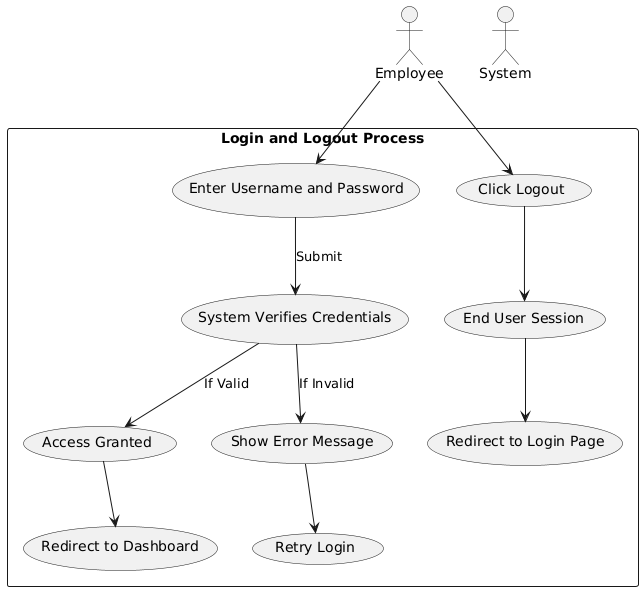
1. Admin selects "Add Parking Spaces" from the system dashboard.
2. System prompts for:
   * Existing Level ID
   * Number of spaces to add
   * Space-specific details (e.g., space size, type – compact, large, electric)
3. Admin enters the information.
4. Admin clicks "Submit".
5. System validates that the level exists and input is correct.
6. System updates the level with the new spaces.
7. Confirmation is shown.

Exceptions:

* System failure: Preventing payment authorization.
* Invalid Input: If required fields are missing or invalid, the system displays an error and requests correction.
* Duplicate Level ID: If a Level ID already exists, the system prompts admin to enter a unique identifier.
* Nonexistent Level (when adding space): System shows an error if the specified level does not exist.

# UML Use Case Diagrams Document

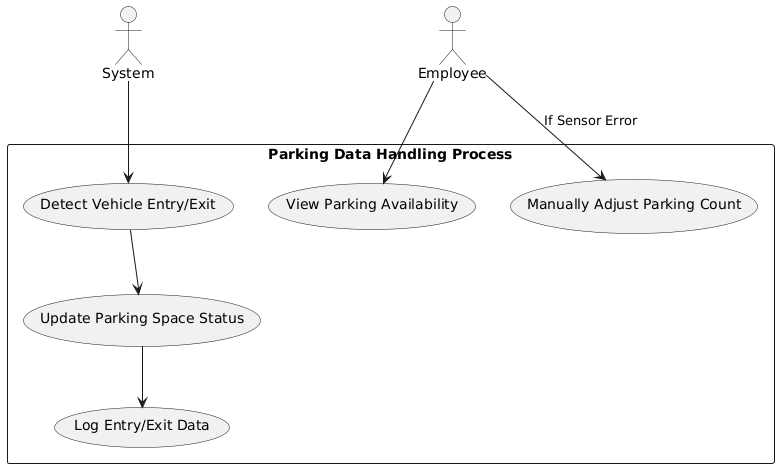
## **1. Use Case: Login and Logout**



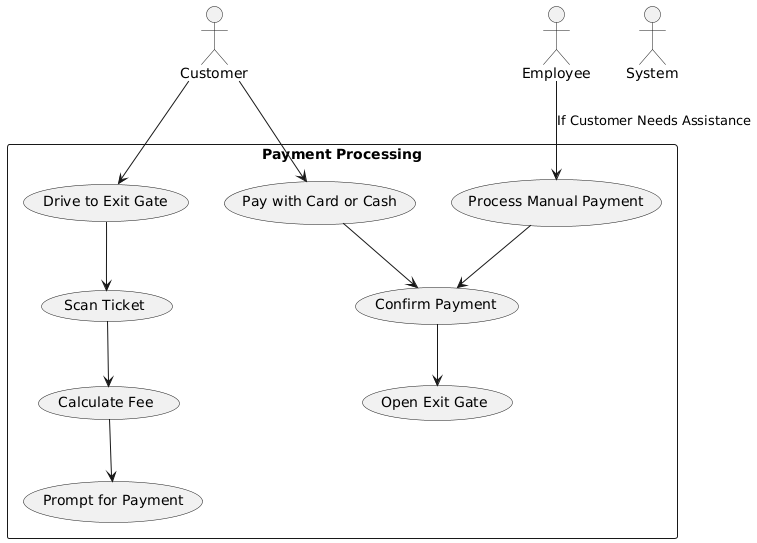
## **2. Use Case: Ticket Handling**

## 

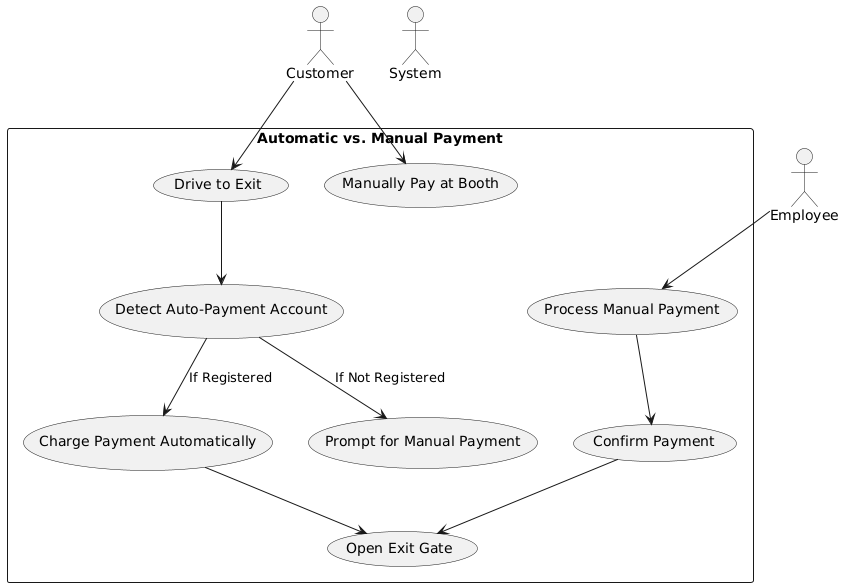
## **3. Use Case: Handling Parking Data**



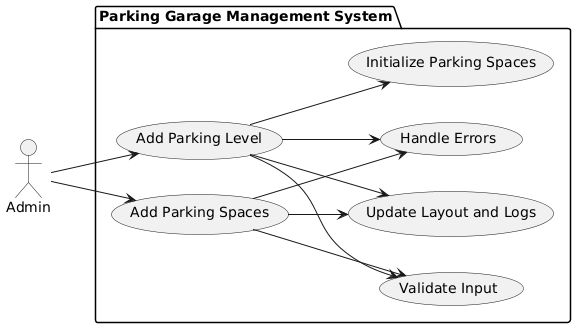
## **4. Use Case: Payment Processing**



## **5. Use Case: Automatic vs. Manual Payment Handling**

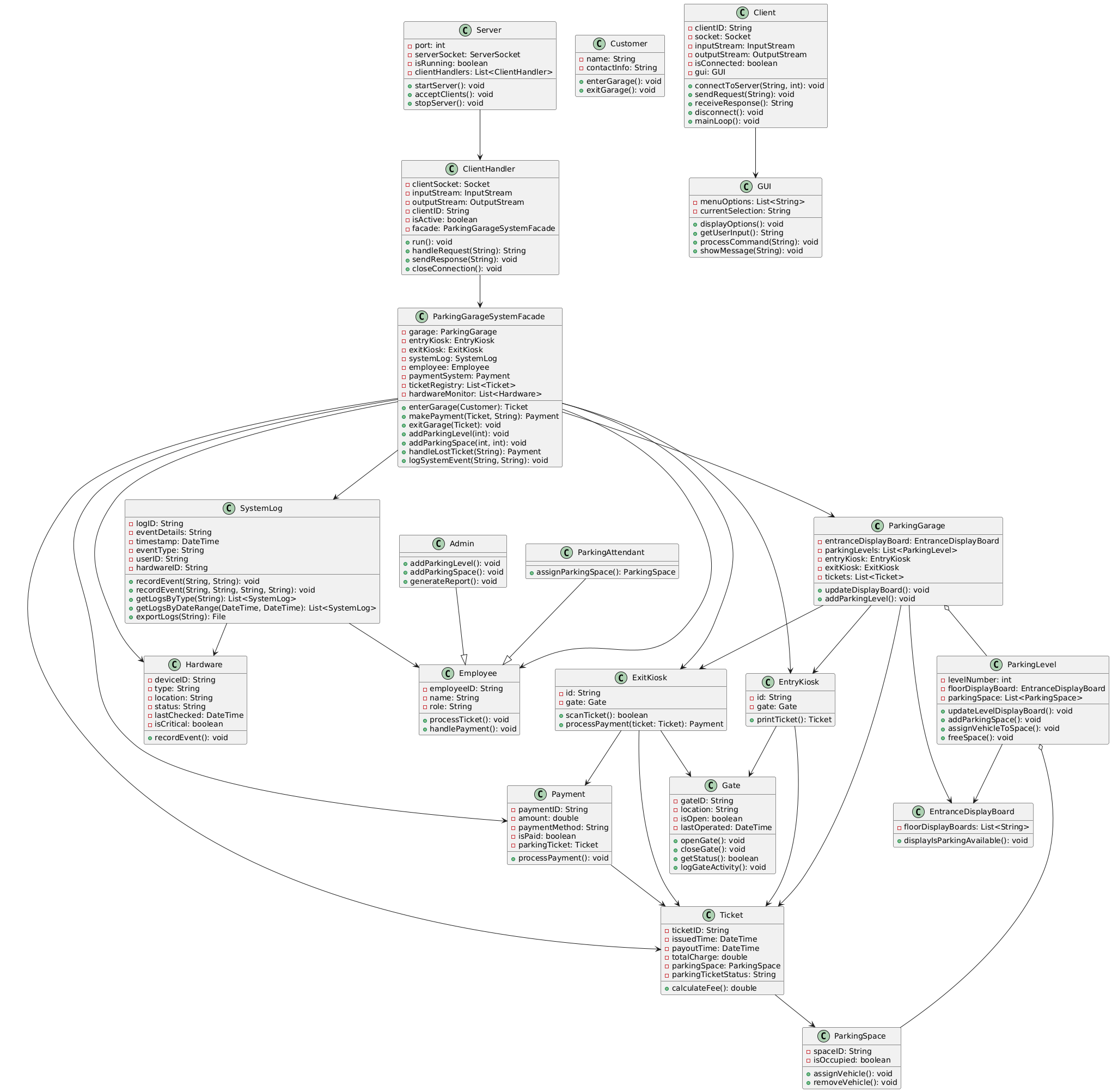


## **6. Use Case: Add Parking Levels**

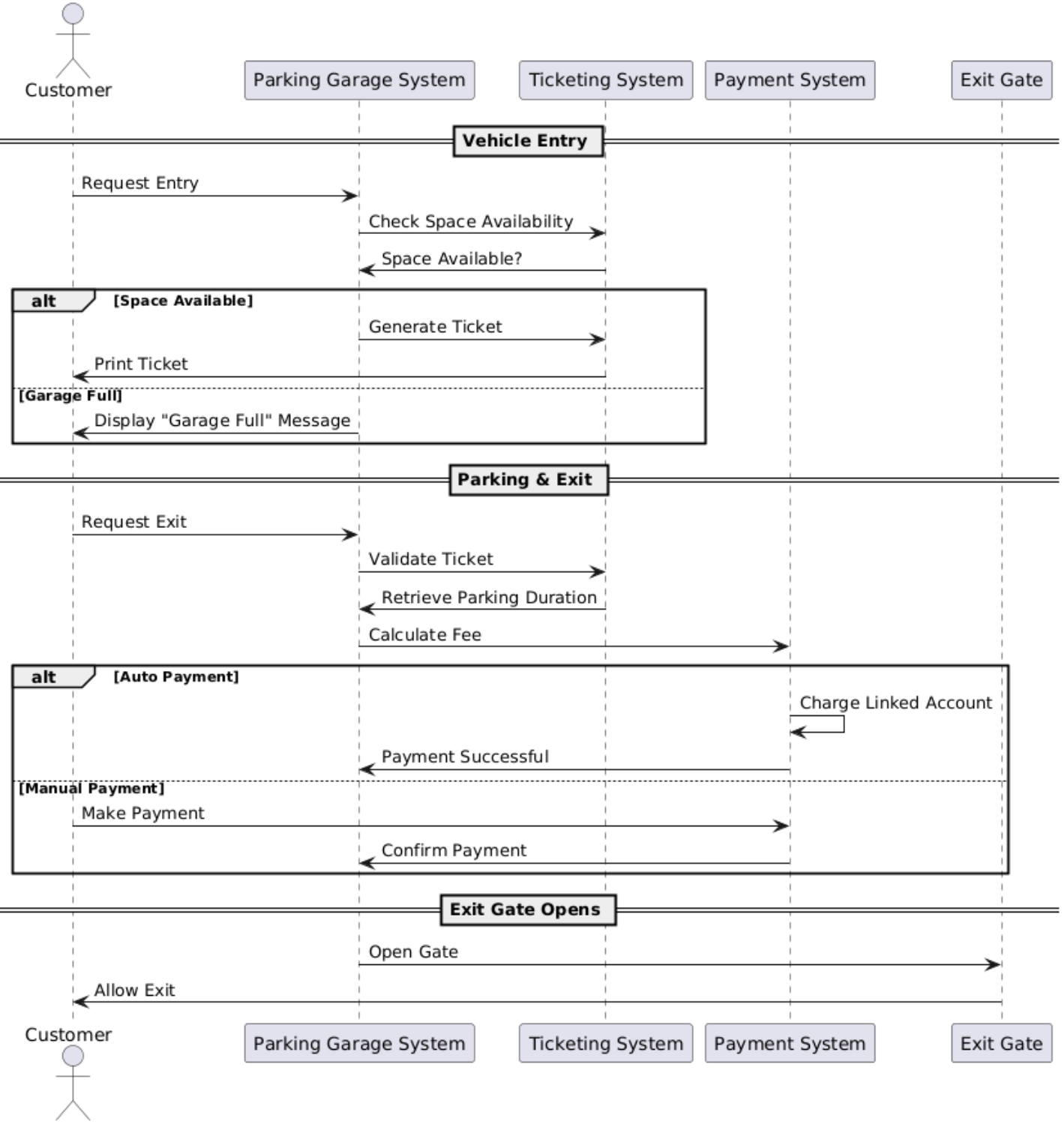


# Class Diagrams

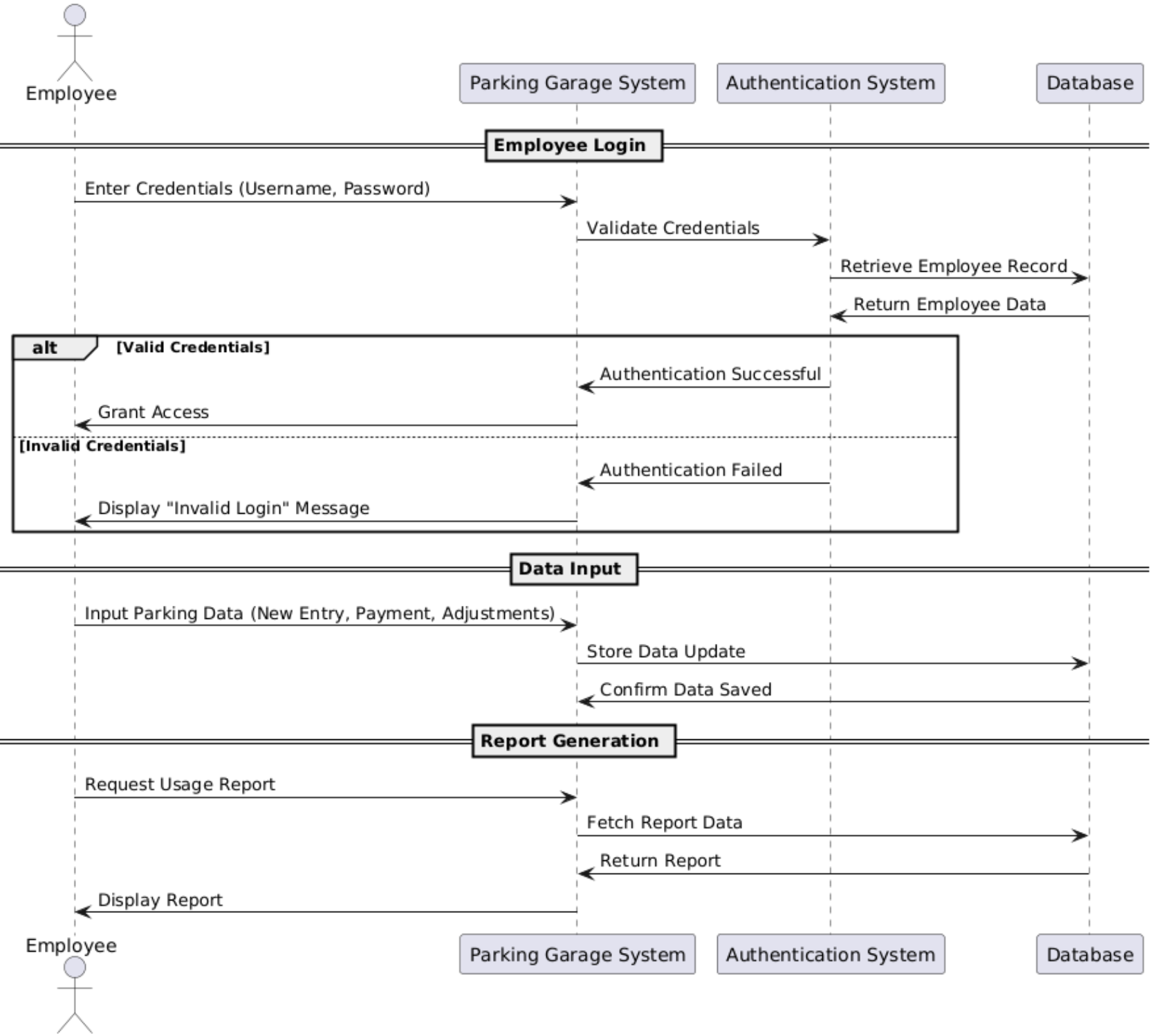
### **Class Diagram:**



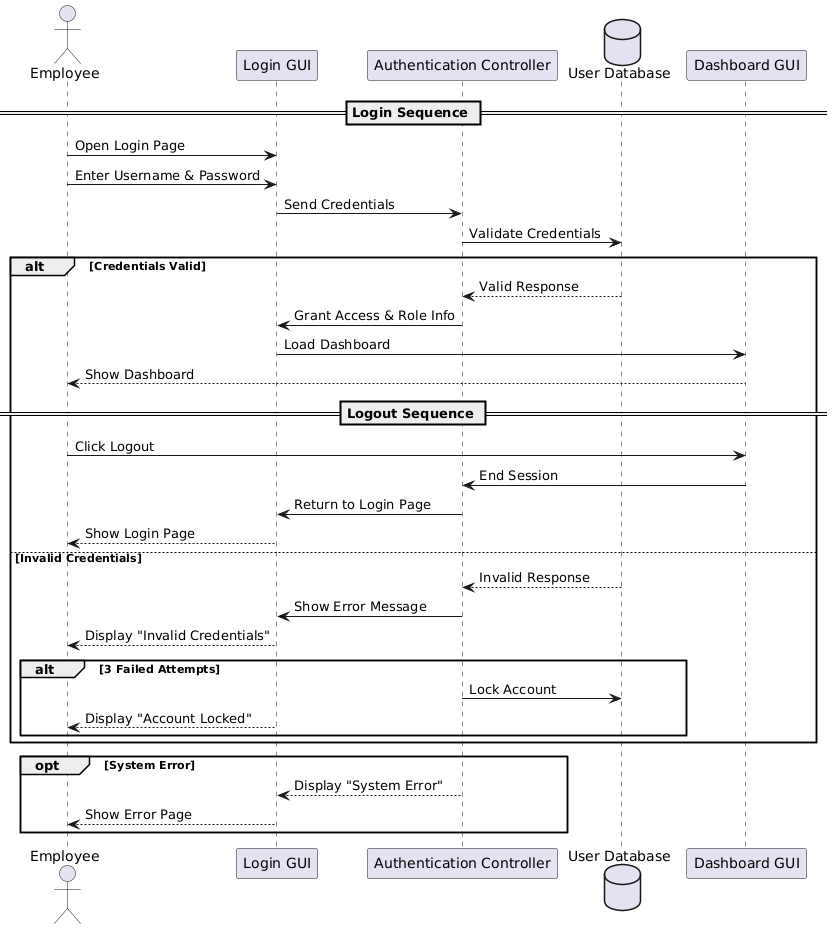
# Sequence Diagrams



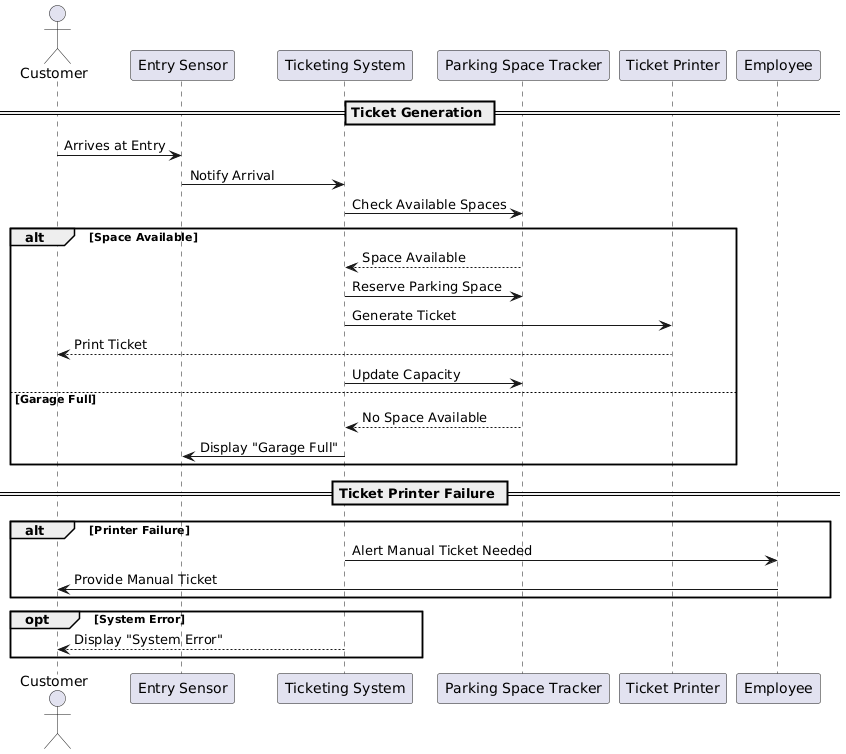
## **Sequence Diagram - Employee Side**



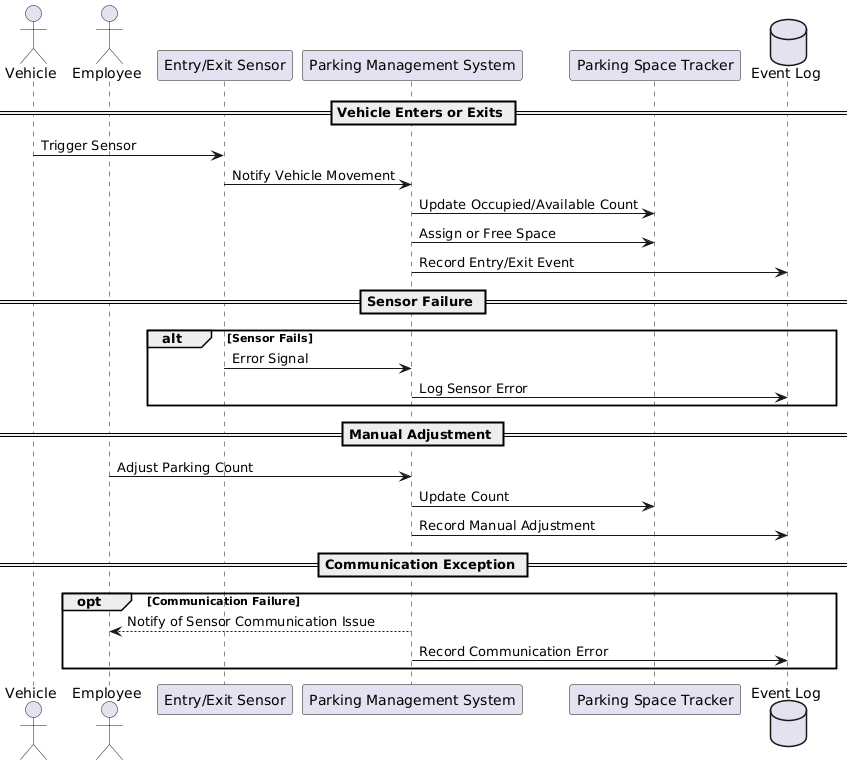
### **Sequence Diagram for Use Case 1: Login and Logout**



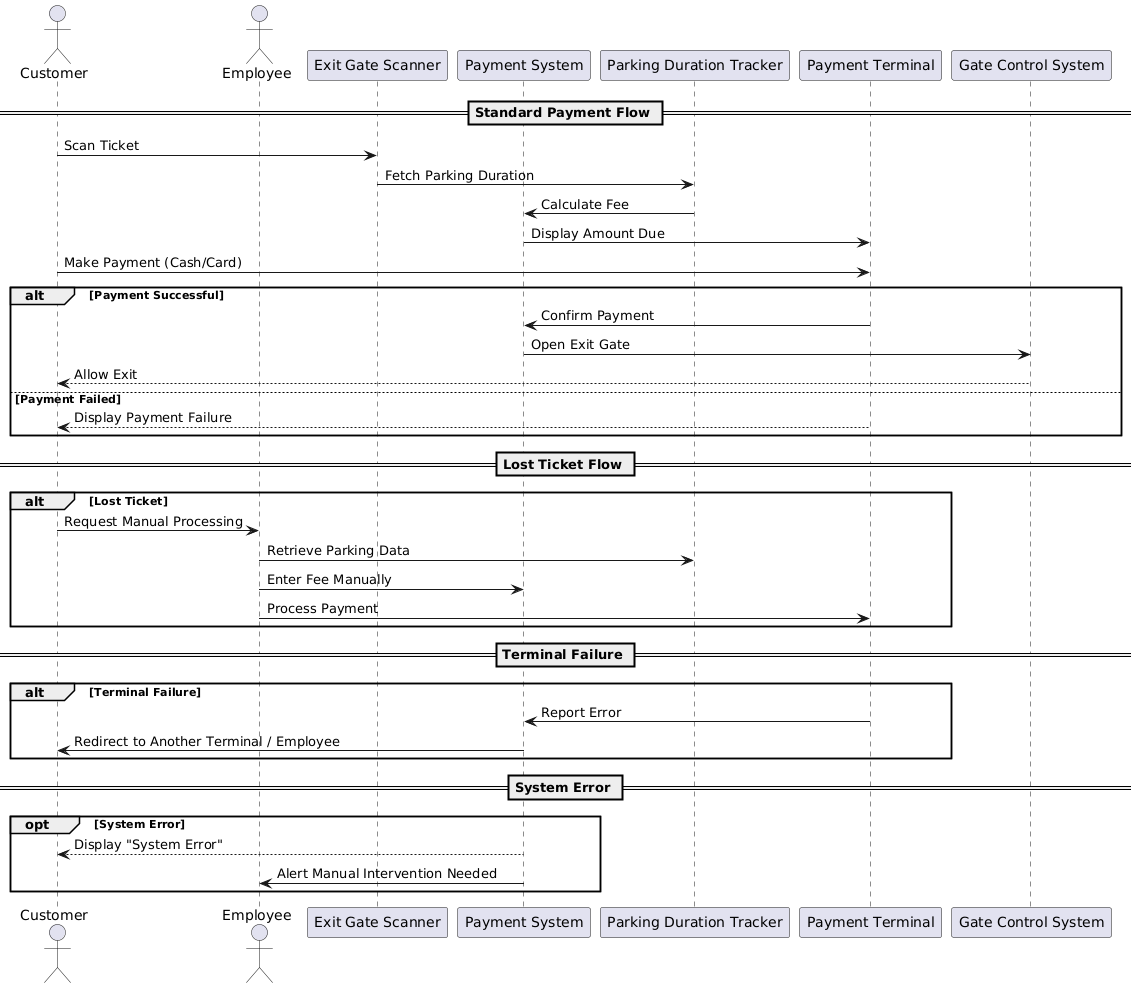
### **Sequence Diagram for Use Case 2: Ticket Handling**



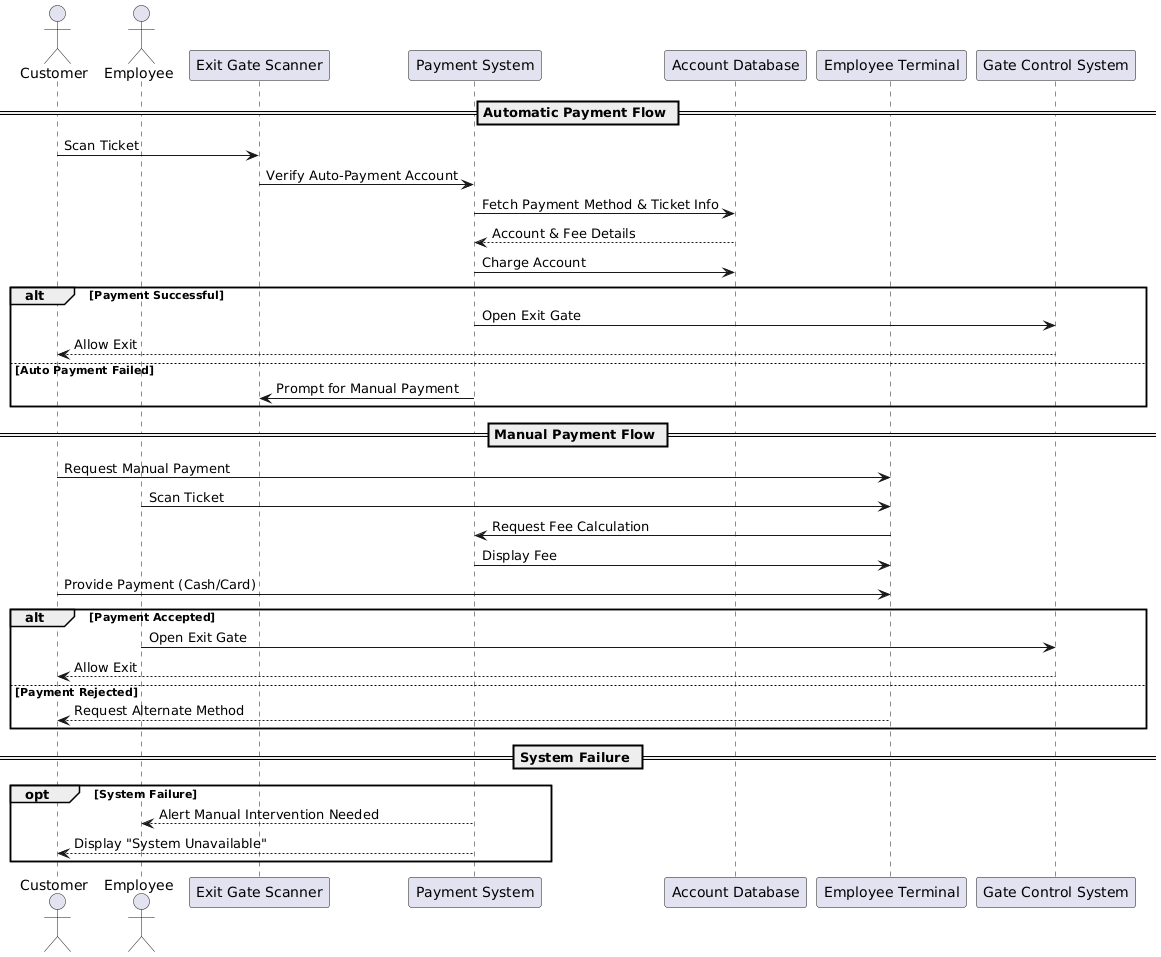
### **Sequence Diagram for Use Case 3: Handling Parking Data**



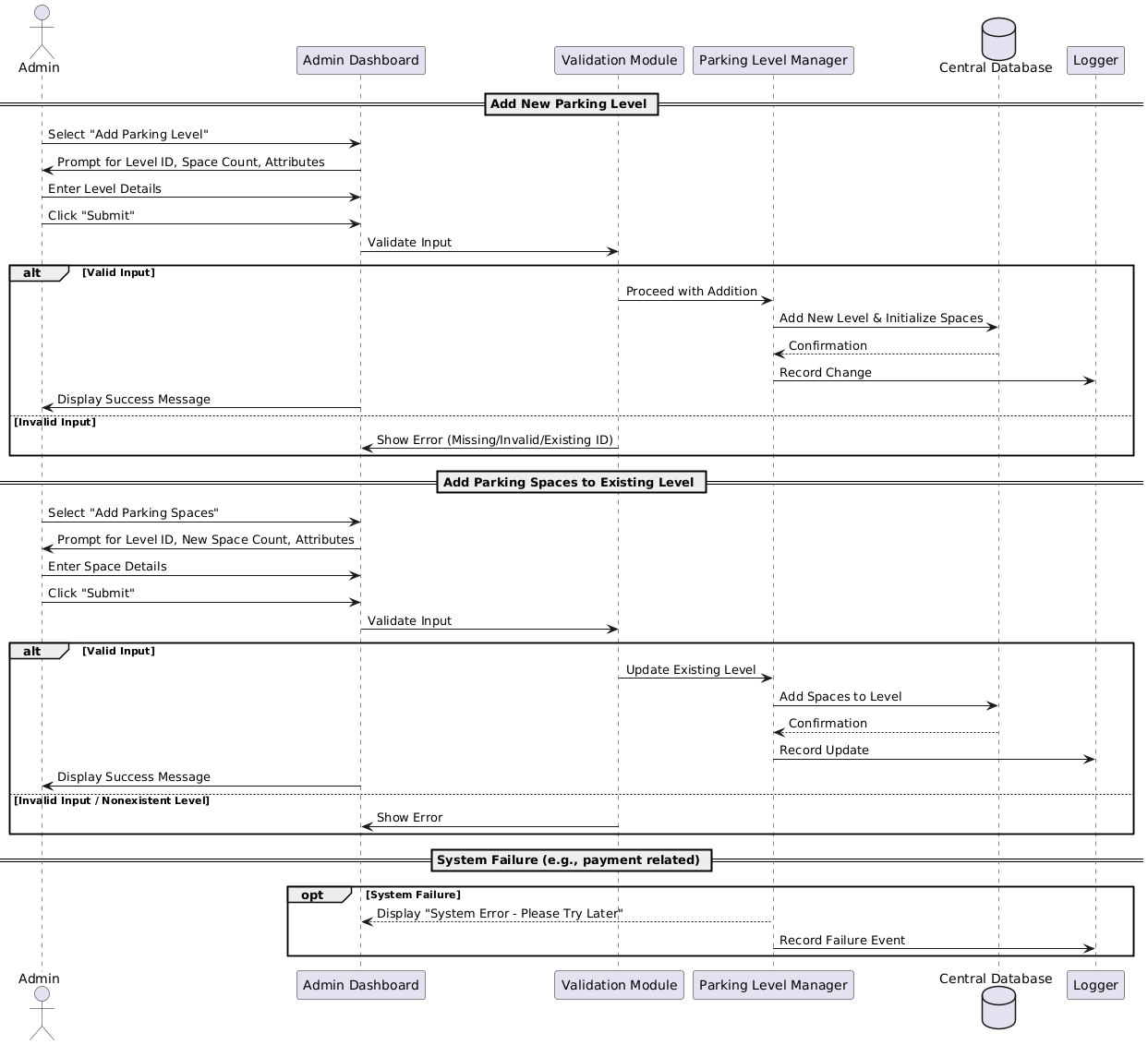
### **Sequence Diagram for Use Case 4: Payment Processing**



### **Sequence Diagram for Use Case 5: Automatic vs. Manual Payment Handling**



### **Sequence Diagram for Use Case 6: Add Parking Levels**



# Classes and Objects

#### 

#### ParkingGarage

Class Variables

* entranceDisplayBoard: EntranceDisplayBoard – Displays parking availability to oncoming vehicles.
* parkingLevels: List<ParkingLevel> – A list of all parking levels in the garage.
* entryKiosk: EntryKiosk – Kiosk where vehicles enter and tickets are printed.
* exitKiosk: ExitKiosk – Kiosk for scanning tickets and handling payments.
* tickets: List<Ticket> – A record of all issued parking tickets.

Methods:

* updateDisplayBoard(): void – Updates the entrance display based on parking availability.
* addParkingLevel(): void – Adds a new parking level to the garage.

#### ParkingLevel

Class Variables

* levelNumber: int – The level number or identifier.
* floorDisplayBoard: EntranceDisplayBoard – A display board showing availability on this level.
* parkingSpace: List<ParkingSpace> – List of parking spaces on the level.

Methods

* updateLevelDisplayBoard(): void – Updates the floor-level display.
* addParkingSpace(): void – Adds new spaces to this level.
* assignVehicleToSpace(): void – Assigns a vehicle to an available space.
* freeSpace(): void – Frees a parking space when a vehicle exits.

#### ParkingSpace

Class Variables

* spaceID: String – Unique identifier for the parking space.
* isOccupied: boolean – Indicates whether the space is currently occupied.

Methods

* assignVehicle(Vehicle v): void – Marks the space as occupied by the vehicle.
* removeVehicle(): void – Frees the space when a vehicle exits.

#### EntranceDisplayBoard

Class Variables

* floorDisplayBoards: List<String> – Text/status displays for each level.

Methods

* displayIsParkingAvailable(): void – Displays whether parking is currently available.

#### EntryKiosk

Class Variables

* id: String – Identifier for the kiosk.
* gate: Gate – Entry gate controlled by the kiosk.

Methods

* printTicket(Vehicle vehicle): Ticket – Prints a new ticket for the entering vehicle.

#### ExitKiosk

Class Variables

* id: String – Identifier for the kiosk.
* gate: Gate – Exit gate controlled by the kiosk.

Methods

* scanTicket(Ticket ticket): boolean – Scans the customer's ticket to validate.

#### Ticket

Class Variables

* ticketID: String – Unique ticket identifier.
* issuedTime: LocalTime – Time the ticket was issued.
* payoutTime: LocalTime – Time the customer exits.
* totalCharge: double – Total amount to be paid.
* parkingTicketStatus: String – Status ("Active", "Paid", "Lost"). - (ENUM)

Methods

* calculateFee(): double – Calculates total parking fee based on duration.

#### 

#### Payment

Class Variables

* paymentID: String – Unique ID for the payment transaction.
* amount: double – Amount to be paid.
* paymentMethod: String – Payment type (e.g., "Card", "Cash"). (ENUM)
* isPaid: boolean – Status of the payment.
* parkingTicket: Ticket – Associated ticket being paid for.

Methods

* processPayment(): void – Processes payment before exit.

#### 

#### Employee (Interface)

Class Variables

* parkingAttendantID: String – Unique identifier for the parkingAttendant.
* name: String – Full name of the parking attendant.

Methods:

* processTicket(): void – Handles the manual creation and issuing of tickets.
* processTicket(Ticket ticket): void – Handles the manual verification, scanning, and setting status of existing tickets.
* handlePayment(Ticket ticket): void – Manages the processing of customer payments, including manual or automated methods.

#### 

#### ParkingAttendant

Class Variables

* parkingAttendantID: String – Unique identifier for the parkingAttendant.
* name: String – Full name of the parking attendant.

Methods:

* processTicket(): void – Handles the manual creation and issuing of tickets.
* handlePayment(Ticket ticket): void – Manages the processing of customer payments, including manual or automated methods.

#### Admin

Class Variables

* AdminID: String – Unique identifier for the admin.
* name: String – Full name of the admin.

Methods

* addParkingLevel(): void – Adds a new parking level to the system.
* addParkingSpace(): void – Adds new parking spaces to an existing level.
* generateReport(): Report – Generates reports on garage activity, revenue, or performance.

#### Customer

Class Variables:

* name: String – Full name of the customer.
* contactInfo: String – Contact details (e.g., phone number or email).

Methods:

* enterGarage(): void – Allows the customer to enter the parking garage.
* exitGarage(): void – Allows the customer to exit after completing payment.

#### SystemLog

Class Variables:

* logID: String – Unique identifier for each log entry.
* eventDetails: String – Description of the logged event (e.g., "Payment successful", "Vehicle entered", "Sensor failure").
* timestamp: DateTime – Date and time the event occurred.
* eventType: String – Type of event ("INFO", "ERROR", "WARNING", "SECURITY").
* userID: String – (Optional) ID of the user or employee associated with the event.
* hardwareID: String – (Optional) ID of hardware involved in the event.

Methods:

* recordEvent(String eventDetails, String eventType): void – Logs a new system event with the current timestamp.
* recordEvent(String eventDetails, String eventType, String userID, String hardwareID): void – Overloaded version to capture more context.
* getLogsByType(String eventType): List<SystemLog> – Retrieves logs filtered by event type.
* getLogsByDateRange(DateTime from, DateTime to): List<SystemLog> – Retrieves logs within a specific time range.
* exportLogs(String format): File – Exports logs for auditing in CSV, PDF, etc.

#### 

#### 

#### Hardware

Class Variables

* deviceID: String – Unique identifier for the hardware component.
* type: String – Type of device ("scanner", "gate", "camera", "sensor" etc.).
* location: String – Physical location within the garage.
* status: String – Operational status ("active", "offline", "maintenance").
* lastChecked: DateTime – Last time the device was tested or pinged.
* isCritical: boolean – Indicates if it's critical to garage operation (e.g., gates, ticket printers).

Methods

* recordEvent(): Logs system activities such as vehicle entries, payments, and errors.

#### Gate

Class Variables

* gateID: String – Unique identifier for the gate.
* location: String – Position of the gate (e.g., "Entry", "Exit").
* isOpen: boolean – Indicates whether the gate is currently open.
* lastOperated: DateTime – Timestamp of the last gate operation.

Methods

* openGate(): void – Opens the gate to allow a vehicle to pass.
* closeGate(): void – Closes the gate after vehicle passage.
* getStatus(): boolean – Returns whether the gate is open or closed.
* logGateActivity(): void – Records gate operations in the system log.

#### Client

Class Variables

* clientID: String – Unique ID for the client instance
* socket: Socket – Network socket used for communication
* inputStream: InputStream – Stream to receive data from the server
* outputStream: OutputStream – Stream to send data to the server
* isConnected: boolean – Tracks the connection status with the server
* GUI – Reference to the GUI component to interact with user input/output

Methods

* connectToServer(String serverIP, int port): void – Establishes a connection to the server using the given IP and port
* sendRequest(String request): void – Sends a formatted request string to the server
* receiveResponse(): String – Reads and returns a response string from the server
* disconnect(): void – Gracefully closes the socket and communication streams
* mainLoop(): void – Runs the client’s interactive session using the GUI to handle input and display output

#### Server

Class Variables

* port: int – The port number the server listens on
* serverSocket: ServerSocket – The main socket for listening
* isRunning: boolean – Indicates whether the server is actively running
* clientHandlers: List<ClientHandler> – Tracks connected client threads

Methods

* startServer(): void – Starts the server and initializes the socket
* acceptClients(): void – Accepts incoming client connections in a loop and creates a ClientHandler for each
* stopServer(): void – Closes all client connections and shuts down the server socket

#### ClientHandler

Class Variables

* clientSocket: Socket – The socket connected to a specific client
* inputStream: InputStream – Input stream from the client
* outputStream: OutputStream – Output stream to the client
* clientID: String – Unique ID for this client session
* isActive: boolean – Indicates if the handler is running
* facade: ParkingGarageSystemFacade – Reference to the system logic layer

Methods

* run(): void – Main execution loop that reads commands, processes them, and sends back results
* handleRequest(String request): String – Parses and processes the client request, invokes appropriate methods from the facade, and returns the response
* sendResponse(String response): void – Sends a formatted response back to the client
* closeConnection(): void – Closes all streams and the socket when the client disconnects

#### GUI

Class Variables

* menuOptions: List<String> – List of available actions the user can perform
* currentSelection: String – Tracks the currently selected menu action

Methods

* displayOptions(): void – Displays the list of available commands or options to the user
* getUserInput(): String – Captures and returns input from the user
* processCommand(String command): void – Interprets and triggers the selected command logic in the client
* showMessage(String message): void – Displays a message or result from the server back to the user

#### ParkingGarageSystemFacade

Class Variables

* garage: ParkingGarage – The central garage object managing levels, spaces, and tickets
* entryKiosk: EntryKiosk – Used for ticket issuing and entry gate control
* exitKiosk: ExitKiosk – Used for payment processing and exit gate control
* systemLog: SystemLog – Logs all system-level events and activities
* employee: Employee – Reference to the logged-in employee, if applicable
* paymentSystem: Payment – Used to process and track payments
* ticketRegistry: List<Ticket> – Stores all tickets issued in the system
* hardwareMonitor: List<Hardware> – Tracks the status of all hardware components

Methods

* enterGarage(Customer customer): Ticket – Issues a ticket, assigns an available space, logs the entry, and opens the entry gate
* makePayment(Ticket ticket, String method): Payment – Calculates the fee, processes payment using the given method, and marks the ticket as paid
* exitGarage(Ticket ticket): void – Verifies payment, opens the exit gate, logs the exit, and frees the associated parking space
* addParkingLevel(int levelNum): void – Adds a new parking level to the system with the specified identifier
* addParkingSpace(int levelNum, int spaceCount): void – Adds the specified number of spaces to an existing parking level
* handleLostTicket(String customerName): Payment – Applies a default lost ticket fee, logs the event, and processes a manual payment
* logSystemEvent(String message, String type): void – Records a system event in the logs (e.g., INFO, ERROR, WARNING, SECURITY)