### 

| Park Zone      Software Requirements Specifications  Group 7 |
| --- |

### 

### 

### 

### 

### 

### 

# 

# Revision History

| Date | Revision | Description | Author |
| --- | --- | --- | --- |
| 09/19/2025 | 1.0 | Initial Version, utilized information from each team member | Mario Salinas, Sophia Ronquillo , Jose Ignacio Zaragoza |
| 09/22/2025 | 1.1 | More revision, comments in specific sections left for changes or removal. | Jose Ignacio Zaragoza |
| 09/24/2025 | 1.2 | Small changes to wording, unnecessary/out of scope sections removed | Jose Ignacio Zaragoza |
| 09/25/2025 | 1.3 | More formatting/wording fixes after talking and collaborating with group mate, Mario and Sophia | Jose Ignacio Zaragoza |
| 09/26/2025 | 1.4 | Formatted the relevant document sections, added constraints, and fixed the table of contents. Used Mario’s document for the majority of the changes. Relevant Documents sections still need to be updated. | Jose Ignacio Zaragoza |
| 09/29/2025 | 1.5 | UML documents section complete using Sophia’s updated description, waiting on partners to revise during the next meeting. | Jose Ignacio Zaragoza |
| 09/30/2025 | 1.6 | More changes to the class diagram and its class description. Tiny tweaks needed for my own final changes on the SRS, using some more ideas from Sophia and Mario | Jose Ignacio Zaragoza |
| 10/01/2025 | 1.7 | Double checking and organizing our SRS | Jose, Sophia, Mario |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### 

## 

*Table of contents*

[**Revision History 2**](#_p44hbfylpydr)

[1. Purpose 6](#_s4ful3on6r57)

[1.1. Scope 6](#_ze1doan2xqdq)

[1.2. Definitions, Acronyms, Abbreviations 6](#_xpj1rydnf6e7)

[1.3. References 7](#_xwgs1fw5vdbs)

[1.4 Overview 7](#_ixpdfoncfnqj)

[2. Overall Description 8](#_8mun2mt2cj6d)

[2.1. Product Perspective 8](#_sll2fvqknece)

[2.2. Product Architecture 8](#_c1kp6w2icckp)

[2.3. Product Functionality/Features 8](#_k7ue18vshi0u)

[2.4. Constraints 8](#_dmekcecqz9p)

[2.5. Assumptions and Dependencies 9](#_1ftb5vbjo5wd)

[3. Functional Requirements 10](#_ylldtomihkj4)

[3.1. Common Requirements 10](#_xl9z74ah8qtt)

[3.1.1. All Modules 10](#_o1bizc8u5ubj)

[3.1.2. Server Module Requirements 10](#_lplcq7ffrz6i)

[3.1.3. Space Management Requirements 10](#_lgsc0k8yo6yf)

[3.1.4. Reservation Module Requirements 10](#_2gsbzss176wi)

[3.1.5 Payment Module Requirements 11](#_7vm1l5u7qha4)

[3.1.6 Customer Client Requirements 11](#_ifq10bv7jlc)

[3.1.7 Operator Client Requirements 11](#_wh8af8979anh)

[3.1.8. Administrator Client Requirements 12](#_pywakpnbb0m0)

[3.2. External Interface Requirements 12](#_61qt6hyynfp)

[3.3. Internal Interface Requirements 12](#_rreya37r0isb)

[4. Non-Functional Requirements 13](#_6906a2b391h)

[4.1 Environmental Requirements 13](#_4spfrwrrtjkn)

[4.2.1 Software Environment 13](#_6he3jpdv4u66)

[4.2.2 Environmental Requirements 13](#_o6xg1d8xj5a)

[4.2.3 Performance Requirements 13](#_9z8uh4y395e7)

[4.2 Performance Requirements 13](#_23c0jhrceyy4)

[4.3.1 Response Time 13](#_cw9ilxdzjd)

[4.3.2 Throughput and Scalability 14](#_ihoqqhl4yv60)

[4.3.4 Resource Usage 14](#_nti6ubmmwxxh)

[5. Relevant Documents 15](#_46djb6vf4j67)

[5.1 Use Case Diagram 15](#_vxv1857lcqvu)

[5.2 ParkZone Main Actors 15](#_wiyuqmo6k68q)

[5.3 Use Cases for Parking Lot 16](#_6iifsykcqibt)

[5.4 Class Diagram (UML) 16](#_epmfm7sm1280)

[5.5 Sequence Diagrams 18](#_jj14opu6zkm7)

# 

# 

# 

# 

# 

# 

## 

## 1. Purpose

ParkZone exists to eliminate the frustration of finding parking by connecting drivers with available parking spaces in real-time while maximizing revenue for parking garage operators through intelligent space management and automated billing systems.

### 1.1. Scope

This document defines the requirements for developing ParkZone, a comprehensive parking management system consisting of:

* **Parking Server**: Backend systems managing databases, payments, reservations, and reporting
* **GUI**: User-facing GUI interface for customers and garage operators
* **Hardware Integration**: Sensors, payment terminals, and access control systems

**What's included:**

* Real-time parking availability tracking
* Reservation and payment systems
* User profiles and rewards program
* Operator dashboards and reporting
* Multi-level garage support

**What's excluded:**

* Hardware manufacturing
* Physical installation services
* Integration with municipal parking systems (Phase 2)

### 1.2. Definitions, Acronyms, Abbreviations

* **GUI**: Graphical User Interface
* **API**: Application Programming Interface
* **SRS**: Software Requirements Specification
* **Overstay**: Parking beyond the paid/reserved time limit
* **Grace Period**: 5-minute buffer before penalties apply
* **Garage Operator**: Staff responsible for daily operations such as monitoring occupancy, assisting customers, and resolving parking issues
* **Admin**: Business owner or authorized manager who configures the system, sets pricing rules, manages parking floors/spots, and oversees reporting
* **End User**: Customer seeking parking services
* **EV**: Electric Vehicle

### 1.3. References

* IEEE Std 830-1998 (Software Requirements Specifications)
* ParkZone Project Kickoff Presentation
* Team meeting notes and requirements gathering sessions
* UML Use Case and Class Diagrams

## 1.4 Overview

The ParkZone application is designed for End User (driver/customer) to reserve parking spaces using their license plate at select locations while maximizing revenue for parking Admins (parking garage owners/managers) and ensuring smooth day-to-day operations for the Garage Operators (general staff). Parking availability is tracked using a list of available parking spots and is updated instantly, allowing users to reserve spaces and pay for parking. Admins benefit from the configuration capabilities available to them (pricing rules, layout management) and automated reporting logs, which are generated on a weekly basis, while the Garage Operators deal with real-time monitoring tools and occupancy dashboards.

The Client Application will provide customers with a user-friendly interface to view all available spaces, allow for filtering by vehicle type (Regular, Handicapped, EV, Compact, Motorcycle, and Large), make a reservation, and handle their payments securely. The Server Application will manage those reservations, process their payments, ensure the availability of parking spots is accurate, and generate a report for the parking garage owners. The Hardware Integration Layer will connect these physical payment terminals, access systems, and any sensors to accurately represent the parking garage’s current state with the digital one.

Three key roles are necessary for this application’s operation. The End User can search for any available spots, make reservations, pay, and earn reward points from frequent use. The Garage Operator will have the ability to oversee the parking space usage, handle payment issues, and monitor the parking garage status. The Admin can configure the system, set pricing, manage the parking garage layouts, and review revenue/occupancy reports.

A log will be generated for important events, such as reservations, cancellations, payments, overstays, and any operator actions. Reports that summarize revenue generated, occupancy, and violations are generated on a weekly basis. End Users will be notified for confirmation on their reservations, overstay alerts, or rewards earned, or any promotional offers, while the Garage Operator and Admins will be notified for any system issues.

## 2. Overall Description

### 2.1. Product Perspective

ParkZone is a new, standalone system that bridges the gap between parking supply and demand. The system operates as:

* **Customer-facing application** for finding and reserving spots
* **Operator management console** for garage owners
* **Backend infrastructure** connecting all components
* **Integration layer** for payment processors and hardware sensors

### 2.2. Product Architecture

ParkZone is a new, standalone system that bridges the gap between parking supply and demand. The system operates as:

* **Customer-facing application** for finding and reserving spots
* **Operator management console** for garage owners
* **Backend infrastructure** connecting all components
* **Integration layer** for payment processors and hardware sensors

### 2.3. Product Functionality/Features

**Java Client-Server Architecture:**

**Server Application (Java) - Multi-Threaded Coordinator:**

* **ParkingServer** accepts the client sockets and sends requests to the thread pool of **ClientHandler**’s
* All reserve/release operations will be executed with multiple threads using thread-safe services to prevent corruption when users reserve spaces, as well as the use of version numbers/timestamps to prevent double-booking.
* The **ReservationManager** will run grace period checks and overstay expiration at a time interval to efficiently handle overstays.
* When a **ClientHandler** fails, it will not impact the **ParkingServer**. Any and all failures will be logged, and the thread is recycled.

### 2.4. Constraints

* System shall not allocate more vehicles than the garage’s physical maximum capacity
* Users will only reserve up to a fixed time window (24 hours in advance)
* Vehicle type compatibility enforced (EV, motorcycle, handicapped)
* Overstay detection will always include a grace period (5 minutes)

### 2.5. Assumptions and Dependencies

**Assumptions:**

* Parking garages have basic internet connectivity.
* Users have smartphones with GPS capabilities.
* Garage operators are willing to adopt new technology.
* Sufficient server infrastructure can be secured.

**Dependencies:**

* Third-party payment processor APIs (Stripe, PayPal).
* GPS and mapping services.
* Hardware sensor integration capabilities.

### 

## 

## 

## 

## 

## 3. Functional Requirements

### 3.1. Common Requirements

#### 3.1.1. All Modules

* **FR-001**: System shall maintain a consistent state across all connected clients through centralized server coordination.
* **FR-002**: All timestamps shall use 24-hour format with date in MM/DD/YYYY
* **FR-003**: System shall log all critical operations (reservations, payments, errors) to a text file for audit trail
* **FR-004**: User interfaces shall provide clear error messages when operations fail, including the reason for failure
* **FR-005**: System shall validate all user inputs before processing requests

#### 3.1.2. Server Module Requirements

* **FR-006**: Server shall start on port 8080 and listen for incoming client connections
* **FR-007**: Server shall spawn a dedicated handler thread for each client connection to enable concurrent request processing
* **FR-008**: System shall broadcast space availability updates to all connected clients within a few seconds of a status change using multi-threaded message distribution
* **FR-009**: Server shall save system state to disk every few minutes using a separate persistence thread to avoid blocking client operations
* **FR-010**: System shall use synchronized methods on shared data structures (ParkingGarage, ReservationManager) to ensure thread-safe concurrent access and prevent race conditions

#### 3.1.3. Space Management Requirements

* **FR-011**: System shall maintain 50 parking spaces distributed as follows: Floor 1 (25 spaces), Floor 2 (25 spaces)
* **FR-012**: Space distribution shall include: 10 Large spaces, 25 Regular spaces, 5 Motorcycle spaces, 4 Handicapped spaces, 6 Electric spaces
* **FR-013**: Each parking space shall have a unique identifier combining floor and space number (e.g. “1A-05”)
* **FR-014**: System shall track space status as one of: Available, Reserved, Occupied
* **FR-015**: System shall prevent assignment of the same space to multiple customers

#### 3.1.4. Reservation Module Requirements

* **FR-016**: Customer shall be able to view list of all available spaces with location and type information
* **FR-017**: Customer shall be able to reserve available space for duration between 1 and 4 hours
* **FR-018**: System shall generate unique reservation ID upon successful booking
* **FR-019**: Reservation shall include timestamp, space ID, customer ID, duration and total fee

## 3.1.5 Payment Module Requirements

* **FR-020:** System shall calculate parking fee as $5.00 per hour or fraction there of
* **FR-021:** Payment interface shall support one payment methods: Debit/Credit Card (both simulated)
* **FR-022:** For cash payments, operator shall enter amount received and system shall calculate change due
* **FR-023:** For credit card payments, system shall simulate card processing with 2-second delay
* **FR-024:** System shall generate digital receipt containing: transaction ID, date/time, space ID, duration, amount paid, payment method

## 3.1.6 Customer Client Requirements

* **FR-025:** Customer client shall display the main menu with options: View Spaces, Make Reservation, My Reservations, Payment, Exit
* **FR-026:** Customer shall be able to filter available spaces by type (Regular, Handicapped, EV, Compact, Motorcycle, and Large)
* **FR-027:** Customer shall be able to view the floor map showing space locations and availability
* **FR-028:** Customer client shall display real-time updates when space availability changes

## 3.1.7 Operator Client Requirements

* **FR-029:** Operator client shall display the dashboard showing: total spaces, available spaces, occupied spaces, and current revenue
* **FR-030:** Operator shall be able to view list of all active reservations with customer information
* **FR-031:** Operator shall be able to manually process the customer payment on behalf of customer
* **FR-032:** Operator shall be able to override reservation expiration for special circumstances

## 3.1.8. Administrator Client Requirements

* **FR-033:** Administrator shall be able to modify hourly parking rate
* **FR-034:** Administrator shall be able to mark spaces as out-of-service or return service
* **FR-035:** Administrator shall be able to generate revenue for specified data range
* **FR-036:** Administrator shall be able to view usage statistics: total reservations, average duration, peak hours
* **FR-037:** Administrator shall be able to add or modify space type designations

### 3.2. External Interface Requirements

* **FR-038:** System shall not interface with any external hardware devices; all operations are simulated
* **FR-039:** System shall not integrate with external payment processors; payment processing is simulated within the application
* **FR-040:** System shall export transaction data in comma-separated text file format for external analysis if requested by administrator
* **FR-041:** Export file shall contain fields: transaction\_id, date, time, space\_id, customer\_id, duration, amount, payment\_method

### 3.3. Internal Interface Requirements

## **FR-042:** Server shall communicate with clients using serialized Java objects transmitted over TCP/IP sockets

## **FR-043:** Message protocol shall include message types: REQUEST, RESPONSE, BROADCAST, ERROR

## **FR-0244:** All client requests shall include: message\_type, sender\_id, timestamp, request\_data

## **FR-045:** All server responses shall include: message\_type, recipient\_id, timestamp, response\_data, status\_code

## **FR-046:** Server shall broadcast space updates using SpaceUpdateBroadcast message containing: space\_id, new\_status, timestamp

## **FR-047:** File storage interface shall use Java ObjectOutputStream for serialization and ObjectInputStream for deserialization

## **FR-048:** Data files shall be stored in /data subdirectory relative to the server application location

## 

## 4. Non-Functional Requirements

* **NFR-001**: System shall require strong password policies (8+ characters, mixed case, numbers
* **NFR-002**: System shall use role-based access control (customer, operator, admin)
* **NFR-003**: System shall automatically log out inactive sessions after 30 minutes/remove?
* **NFR-004**: System shall log all security-relevant events with a timestamp

### 4.1 Environmental Requirements

#### 4.2.1 Software Environment

* **NFR-005**: Administrator functions shall require password authentication before accessing configuration options
* **NFR-006**: System shall log all administrative actions with a timestamp and administrative ID

#### 4.2.2 Environmental Requirements

* **NFR-007**: Server application shall run on the same operating system
* **NFR-008**: System shall support common sensor protocols (REST APIs, MQTT)
* **NFR-009**: System shall work with standard payment terminals
* **NFR-010**: System shall accommodate various garage layouts and configurations

#### 4.2.3 Performance Requirements

* **NFR-011:** Server shall respond to client requests (up to 10 concurrent clients).
* **NFR-012:** Space availability broadcasts shall propagate to all clients within 3 seconds of a status change.
* **NFR-013:** Payment processing shall be completed within 5 seconds, including receipt generation

### 4.2 Performance Requirements

#### 4.3.1 Response Time

* **NFR-014:** System shall respond to user actions within 2 seconds under normal load
* **NFR-015:** Real-time updates shall propagate within 5 seconds
* **NFR-016:** Payment processing shall be completed within 10 seconds
* **NFR-017:** Report generation shall complete within 30 seconds for standard reports

#### 4.3.2 Throughput and Scalability

* **NFR-018**: System shall support minimum 60 concurrent users
* **NFR-019**: System shall handle 7000 parking transactions per day
* **NFR-020**: Database shall support horizontal scaling for growth
* **NFR-021**: System shall maintain performance during peak usage (weekends, events)

4.3.3 Availability and Reliability

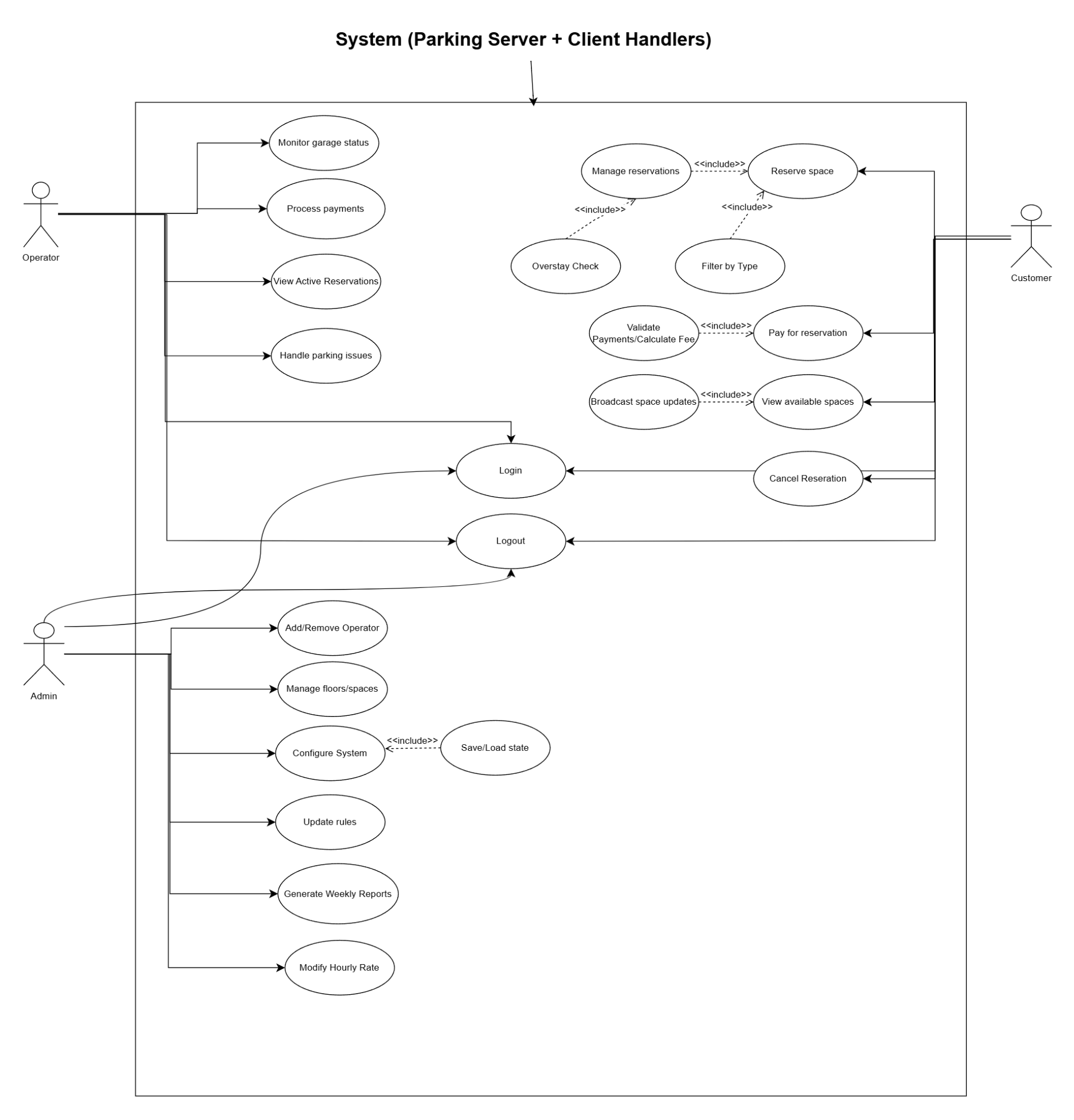
* **NFR-021**: System shall maintain 99.5% uptime during business hours
* **NFR-022**: System shall implement automatic failover for critical components
* **NFR-023**: System shall perform automated daily backups
* **NFR-024**: System shall recover from failures within 15 minutes

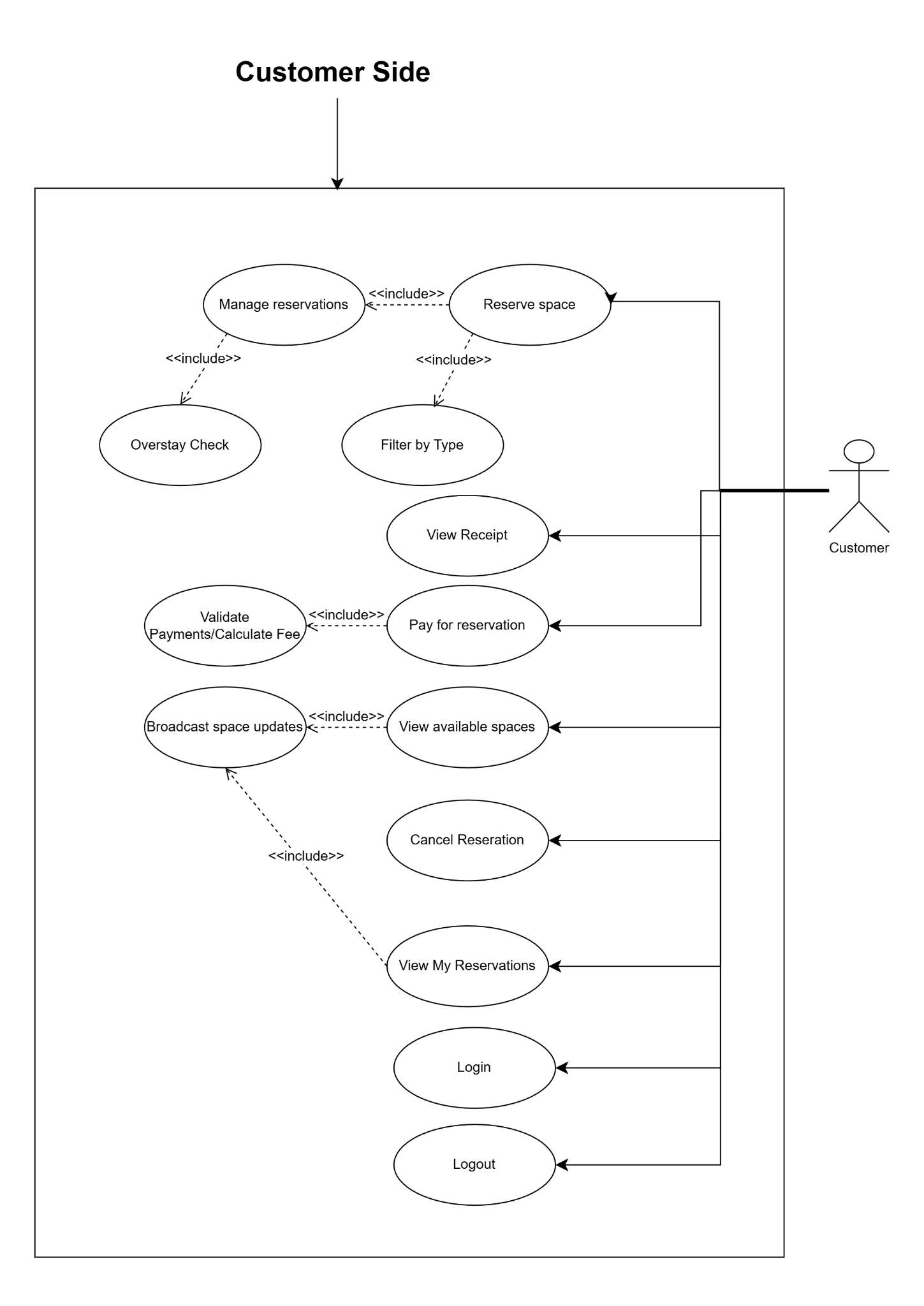
#### 4.3.4 Resource Usage

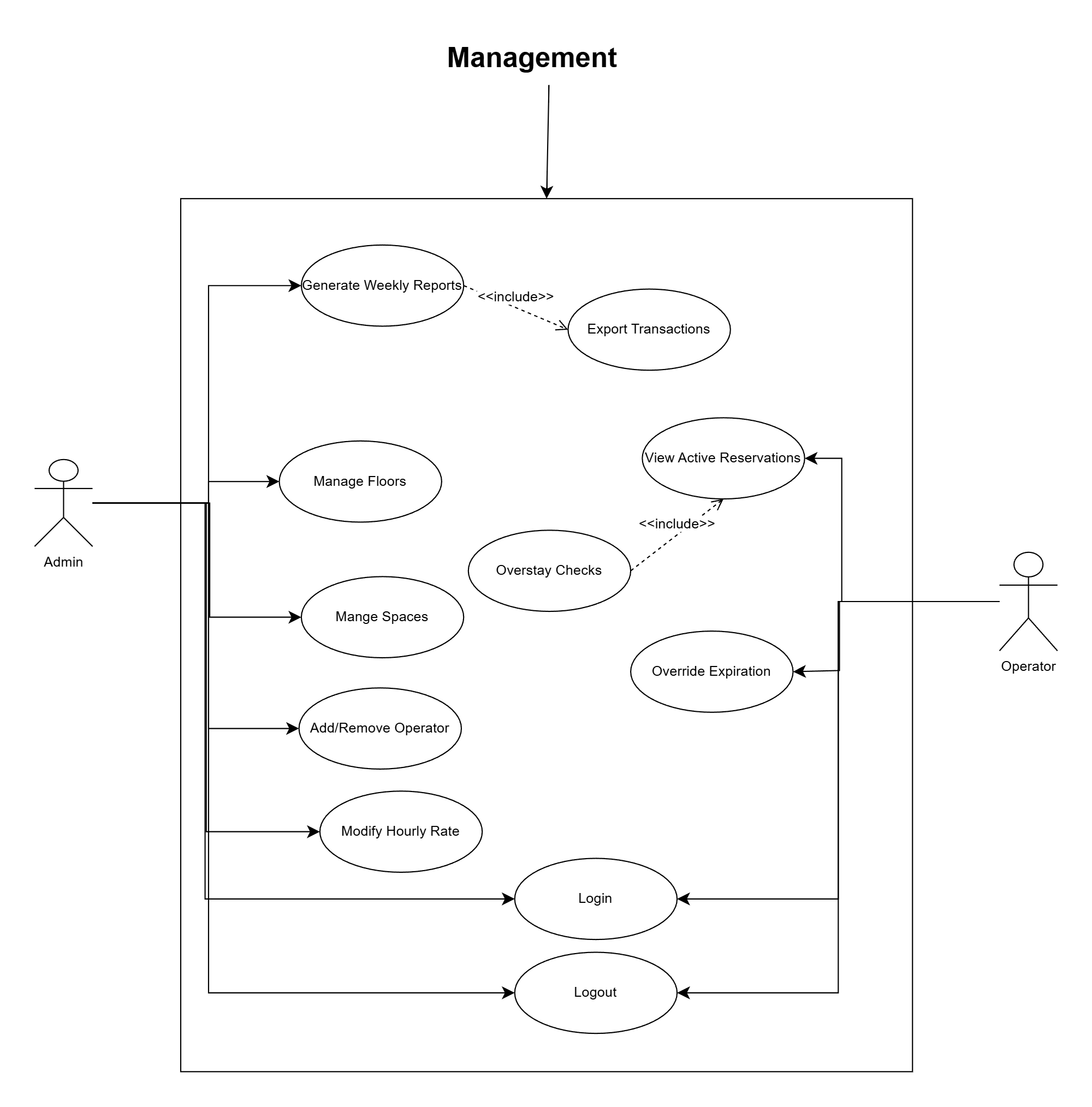
* **NFR-025:** Server shall efficiently handle database queries without performance degradation
* **NFR-026:** System shall optimize bandwidth usage for real-time updates
* **NFR-027:** System shall utilize a queue to handle excessive incoming requests.

## 5. Relevant Documents

## 5.1 Use Case Diagram







## 5.2 ParkZone Main Actors

* **Operator (Garage Operator):** Monitors occupancy, handles issues, and may process payments
* **Customer (End User):** Finds spaces, reserves, pays, and cancels.
* **Admin:** Configures pricing, layout/floors/spaces, rules
* **System (ParkingServer):** Automated system set by Admin that displays messages on different panels, as well as assigns and removes a vehicle from a parking spot when a customer reserves or leaves a space.

## 5.3 Use Cases for Parking Lot

* **Add/Remove/Edit Parking Floor:** Admins can configure their parking facility by adding, removing, or modifying a parking floor from the system. Each floor will have its own display board to show free parking spots.
* **Add/Remove/Edit parking spot:** Admins can add, remove, or modify a parking spot on a parking floor, as well as assign them for specific car types (Regular, Handicapped, EV, Compact, Motorcycle, and Large).
* **Add/Remove a Garage Operator:** Admins can add or remove a Garage Operator from the system.
* **Take ticket:** Customers receive a new parking ticket when reserving a parking spot.
* **Scan ticket:** The System will calculate the total charge when scanning a ticket.
* **Credit card payment:** The Customer can pay the ticket fee with a credit card.
* **Add/Modify parking rate:** Admins can modify the hourly parking rate.

## 5.4 Class Diagram (UML)

Main Classes for the Parking Lot System:

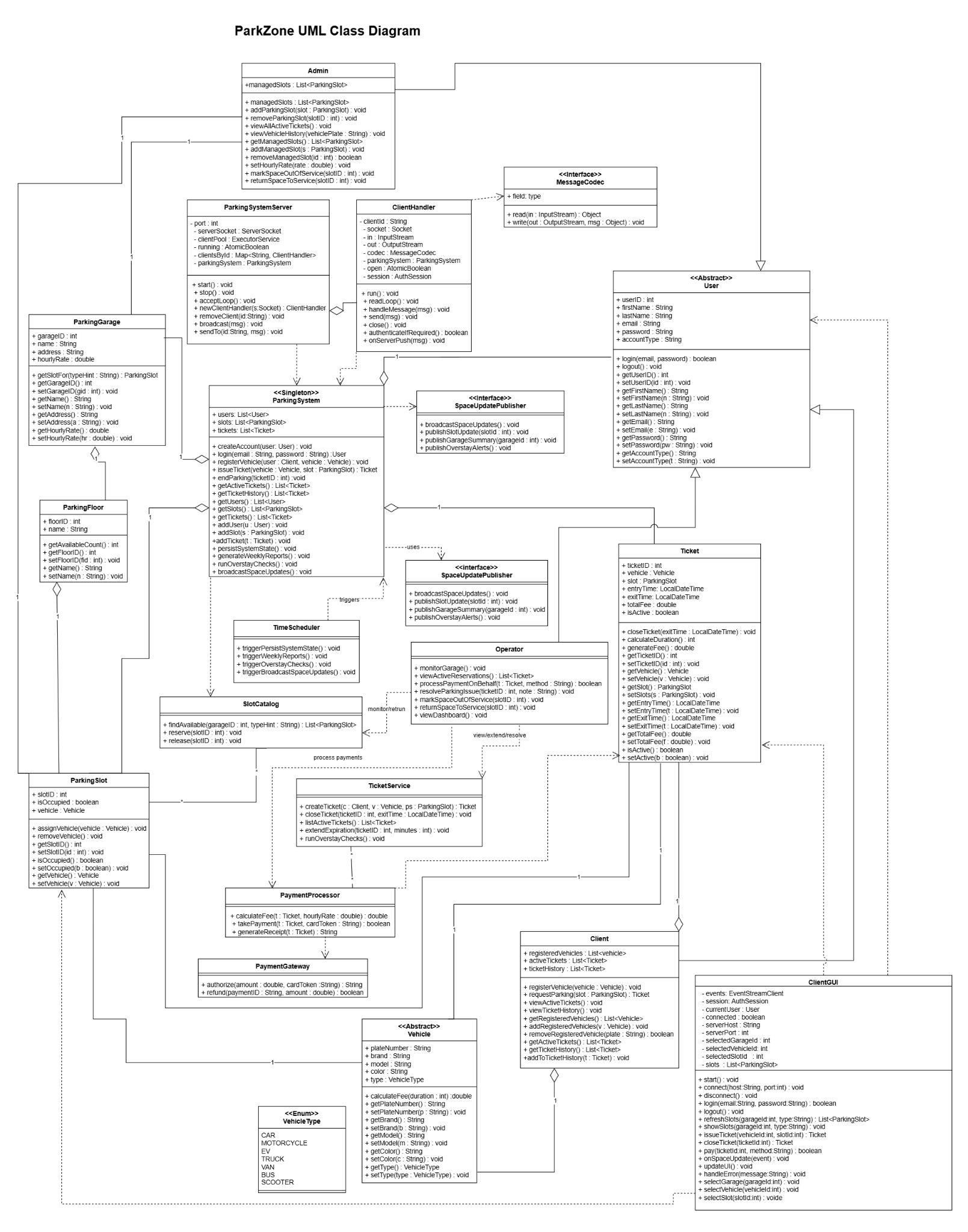
* **ParkingSystem:** The central part of the organization for which this software has been designed. Handles user registration, authentication, and vehicle association. Issues and closes tickets via TicketService, and coordinates persistence, reports, and broadcasts updates.
* **ParkingSystemServer:** The main server-side controller, which opens TCP/IP sockets for clients. Creates ClientHandler threads using an ExecutorService pool, dispatches requests to the ParkingSystem, TicketService, and PaymentProcessor, as well as performs broadcasts for real-time slot updates and system alerts.
* **ParkingGarage:** Defines each floor on the parking lot, which has multiple parking spots.
* **ParkingSlot:** Represents a single parking spot. Our system will support different parking spots

1. Handicapped
2. Compact
3. Large
4. Motorcycle
5. Electric
6. Regular

* **User:** Defines the user’s role in the system: Admin, Operator, and Customer
* **Ticket:** This class will encapsulate a parking ticket. Customers get a digital ticket after reserving a spot. The ticket will include details such as the license plate number and time slot reserved.
* **Vehicle:** Represents the customer’s vehicle. Our system will support different types of vehicles. Here are a couple of examples:

1. Car
2. Truck
3. Electric (EV)
4. Van
5. Handicapped
6. Motorcycle.

* **PaymentProcessor:** This class will be responsible for making payments. The system will support credit/debit card transactions.
* **PaymentGateway:** Helper class for authorizing as well as refunding payments.
* **Operator:** This class will encapsulate all the operations that an Operator can perform, like monitoring the garage status, managing any issues present, and processing payments.
* **ClientHandler:** This class will run and handle all requests from Admins, Operators, and Customers. Authenticates users and parses through serialized messages, invokes server methods for ticket management, payments, and reservations.
* **ClientGUI:** Provides a user interface for end users as well as Operators and Admins. Manges login, slot display, ticket payment, and reservations. Connects to the server over TCP/IP and will display real-time data.
* **Client:** Enables the customer (End User) to pay for tickets electronically, view spaces, and reserve a space.
* **Admin**: This class will encapsulate the operations that an Admin can perform, like configure the system, manage the parking floor, and set pricing.



## 5.5 Sequence Diagrams

## 

