Park Zone

Software Requirements Specifications

Group 7

Revision History

Date	Revision	Description	Author
09/19/2025	1.0	Initial Version, utilized a combination of thoughts from Sophia and Mario. Revision by Jose.	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	Jose Ignacio Zaragoza
09/26/2025	1.4	Formated the relevant document sections, added constraints, and fixed the table of contents. Used Mario's document for majority of the changes. Relevant Documents sections still needs 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 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

Table of contents

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:

- **Server Application**: Backend systems managing databases, payments, reservations, and reporting
- **Client Application**: User-facing mobile/web 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 that 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 (compact, large, motorcycle, or EV), 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)
- Only compatible vehicle types can reserve certain spot categories (EV, handicapped, motorcycle)
- System shall assume a constant internet connection
- 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).
- Cloud hosting services (AWS, Azure, or Google Cloud).
- GPS and mapping services.
- Hardware sensor integration capabilities.

3. Functional Requirements

3.1. Server-Side Core Systems

3.1.1. Database Management & Real-Time Tracking

- **FR-001**: System shall maintain a real-time inventory of all parking spaces
- **FR-002**: System shall update space availability within 5 seconds of a status change
- **FR-003**: System shall store user profiles, transaction history, and garage data
- **FR-004**: System shall support multi-level garage configurations
- **FR-004**: System shall provide different parking spots for different vehicle types (compact, large, EV, motorcycle).

3.1.2. Payment Processing Backend

- **FR-005**: System shall process card payments
- **FR-006**: System shall calculate fees based on duration, vehicle type, and pricing rules
- **FR-007**: System shall handle refunds for canceled reservations
- **FR-008**: System shall generate payment receipts and transaction logs
- **FR-009**: System shall have a per-hour pricing model, $5 for the first hour, $3 for the second and third, and $2 for the remaining hours.

3.1.3. Reservation Logic & Validation

- **FR-010**: System shall allow users to reserve spaces up to 24 hours in advance
- **FR-011**: System shall release reservations after a 5-minute grace period
- **FR-012**: System shall prevent double-booking of spaces
- **FR-013**: System shall send confirmation notifications for reservations
- **FR-014**: System shall support compact, large, EV, and motorcycles.

3.1.4. Report Generation Engine

- **FR-014**: System shall generate daily/weekly/monthly usage reports
- **FR-015**: System shall track revenue, occupancy rates, and user behavior
- **FR-016**: System shall identify overstay incidents and violations
- **FR-017**: System shall export reports in PDF and CSV formats

3.2. Client Application Interface

3.2.1 Parking Space Selector GUI

- **FR-018:** Interface shall display available spaces in real-time
- **FR-019:** Interface shall allow filtering by accessibility and EV charging
- **FR-020:** Interface shall show pricing information before selection
- **FR-021:** Interface shall provide a garage layout visualization

3.2.2 Payment Interface & Forms

- **FR-022:** Interface shall support multiple payment methods
- **FR-023:** Interface shall calculate total costs before payment
- **FR-024:** Interface shall store preferred payment methods securely
- **FR-025:** Interface shall handle payment failures gracefully

3.2.3 Reservation Booking Interface

- **FR-026**: Interface shall allow time-based reservations
- **FR-027**: Interface shall show reservation confirmations
- **FR-028**: Interface shall allow reservation modifications/cancellations
- **FR-029**: Interface shall display countdown timers for reserved spaces

3.2.4 User Dashboard & Profile Management

- **FR-030**: System shall maintain user profiles with vehicle information
- **FR-031**: System shall display parking history and receipts
- **FR-032**: System shall show rewards points and available discounts
- **FR-033**: System shall allow users to report issues and violations

3.3. System Integration Requirements

3.3.1 Server-Client Communication Protocols

- **FR-034**: System shall use HTTPS for all client-server communication
- **FR-035**: System shall implement WebSocket connections for real-time updates
- **FR-036**: System shall provide RESTful APIs for all operations
- **FR-037**: System shall handle network interruptions gracefully
- **FR-038**: System shall sync data across all client instances within 10 seconds
- **FR-039**: System shall maintain data consistency during high-traffic periods
- **FR-040**: System shall queue updates during temporary network outages
- **FR-041**: System shall resolve data conflicts using timestamp precedence

3.3.2 Real-Time Data Synchronization

- **FR-042**: System shall integrate with payment gateway APIs
- **FR-043**: System shall connect with the garage sensor hardware
- **FR-044**: System shall interface with access control systems
- **FR-045**: System shall support third-party mapping services

3.3.3 External System Interfaces

- **FR-046**: System shall send push notifications for reservations
- **FR-047**: System shall alert users of overstay situations
- **FR-048**: System shall notify operators of system issues
- **FR-049**: System shall send promotional offers to frequent users
- **FR-050**: System shall provide charging and pay panels for EVs

3.4 User Experience Features

3.4.1 Rewards & Gamification Elements

- **FR-051**: System shall award points based on usage duration
- **FR-052**: System shall allow points redemption for free parking
- **FR-053**: System shall provide loyalty tier benefits
- **FR-054**: System shall track and display usage statistics

3.4.2 Ticket Generation and Payment

- **FR-055**: System shall allow the use of credit/debit cards for payment at the time of reservation
- **FR-056**: System shall use the license plate entered during the time of reservation as part of the ticket ID
- **FR-057**: System shall provide the user with a receipt containing their vehicle information, payment method, total, and ticket number

4. Non-Functional Requirements

4.1 Security and Privacy Requirements

4.1.1 Authentication and Authorization

- **NFR-001**: System shall require strong password policies (8+ characters, mixed case, numbers)
- **NFR-002**: System shall implement multi-factor authentication for operators
- **NFR-003**: System shall use role-based access control (customer, operator, admin)
- **NFR-004**: System shall automatically log out inactive sessions after 30 minutes

4.1.2 System Security

- **NFR-005**: System shall log all security-relevant events with timestamps
- **NFR-006**: System shall implement rate limiting and throttling to prevent abuse such as DoS.
- **NFR-007**: System shall use HTTPS/TLS 1.3 for all communications
- **NFR-008**: System shall regularly update and patch security vulnerabilities

4.2 Environmental Requirements

4.2.1 Hardware Environment

- **NFR-009**: Server shall run on cloud infrastructure (AWS/Azure/GCP)
- **NFR-010**: System shall support both portrait and landscape orientations

4.2.2 Software Environment

- **NFR-011**: Server shall be compatible with Linux-based operating systems
- **NFR-012**: Database shall use PostgreSQL or equivalent relational database
- **NFR-013**: System shall be deployment-ready using containerization (Docker)

4.2.3 Integration Environment

- **NFR-014**: System shall integrate with existing garage management systems
- **NFR-015**: System shall support common sensor protocols (REST APIs, MQTT)
- **NFR-016**: System shall work with standard payment terminals
- **NFR-017**: System shall accommodate various garage layouts and configurations

4.2.4 Parking Garage Layout

- **NFR-018**: Multiple floors should be available to customers.
- **NFR-019**: Multiple entry and exit points to avoid traffic jams in the parking garage.

4.3 Performance Requirements

4.3.1 Response Time

- **NFR-020**: System shall respond to user actions within 2 seconds under normal load
- **NFR-021**: Real-time updates shall propagate within 5 seconds
- **NFR-022**: Payment processing shall be completed within 10 seconds
- **NFR-023**: Report generation shall complete within 30 seconds for standard reports

4.3.2 Throughput and Scalability

- **NFR-024**: System shall support minimum 1,000 concurrent users
- **NFR-025**: System shall handle 10,000+ parking transactions per day
- **NFR-026**: Database shall support horizontal scaling for growth
- **NFR-027**: System shall maintain performance during peak usage (weekends, events)

4.3.3 Availability and Reliability

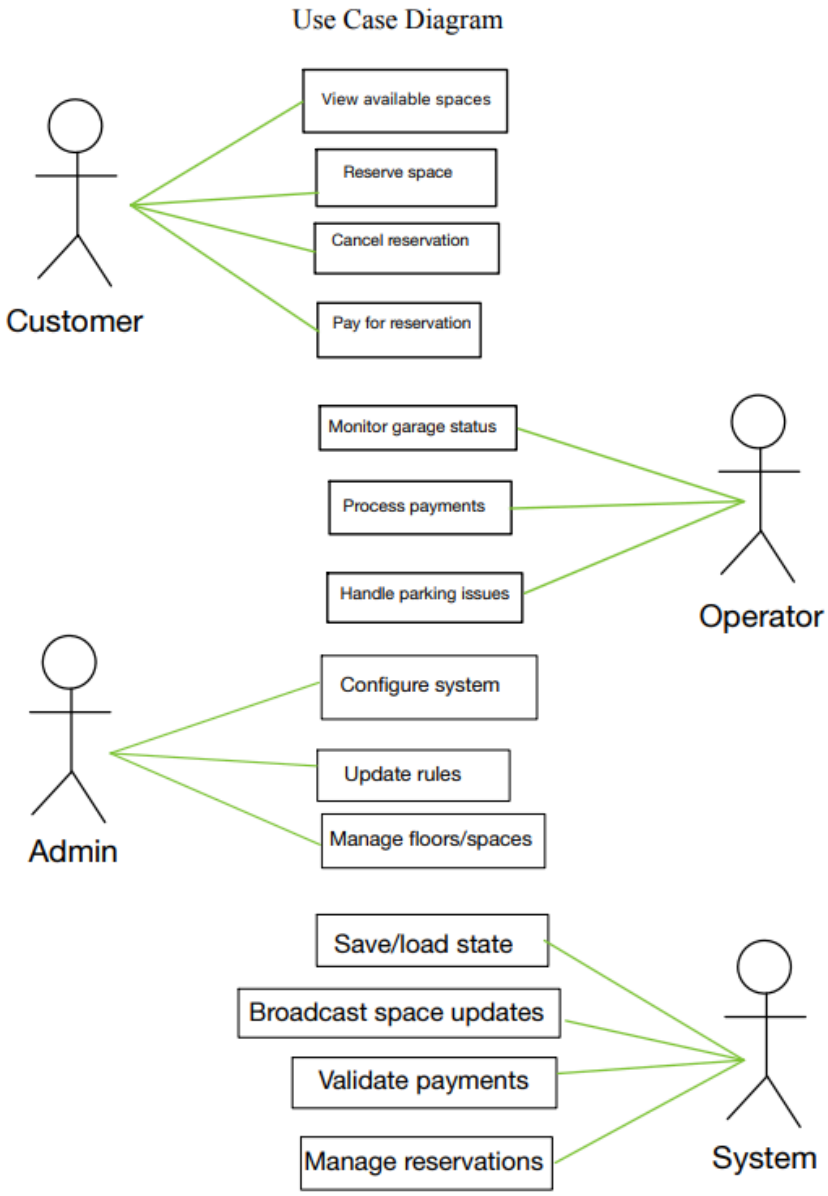
- **NFR-028**: System shall maintain 99.5% uptime during business hours
- **NFR-029**: System shall implement automatic failover for critical components
- **NFR-030**: System shall perform automated daily backups
- **NFR-031**: System shall recover from failures within 15 minutes

4.3.4 Resource Usage

- **NFR-032**: Server shall efficiently handle database queries without performance degradation
- **NFR-033**: System shall optimize bandwidth usage for real-time updates
- **NFR-034**: 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 (motorcycle, EV, handicapped).
- **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.

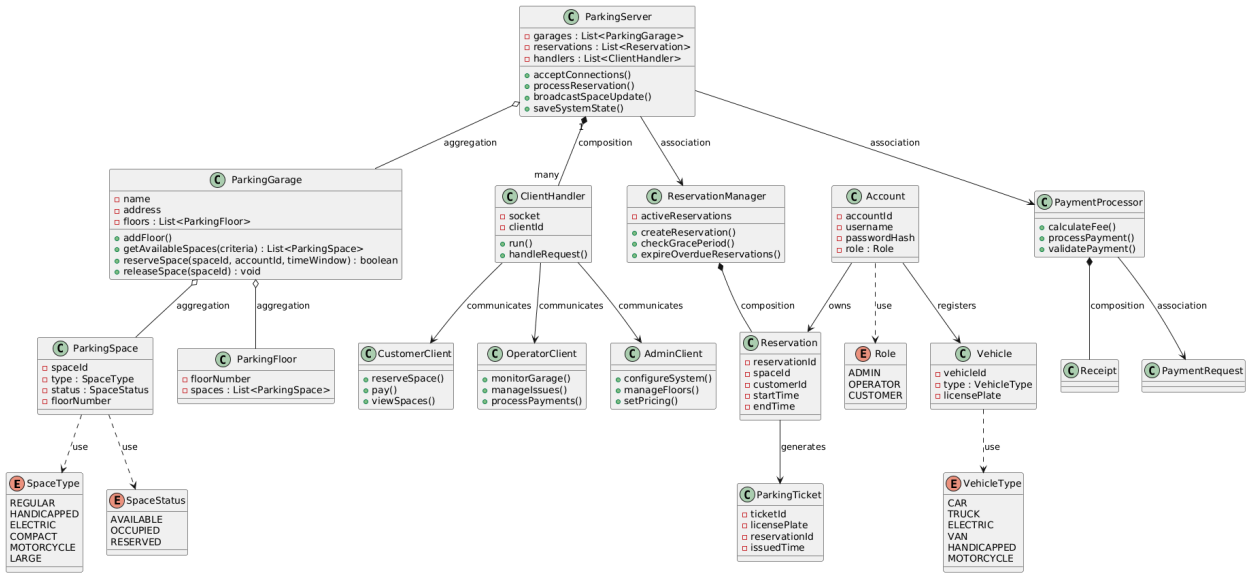
5.4 Class Diagram (UML)

Main Classes for the Parking Lot System:

- **ParkingServer:** The central part of the organization for which this software has been designed. Can hold multiple garages with their own floors and parking spots..
- **ParkingGarage:** Defines each floor on the parking lot, which has multiple parking spots.
- **ParkingSpace:** Represents a single parking spot. Our system will support different parking spots
  1. Handicapped
  2. Compact
  3. Large
  4. Motorcycle
  5. Electric
- **Account:** Defines the user's role in the system: Admin, Operator, and Customer
- **Receipt:** 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

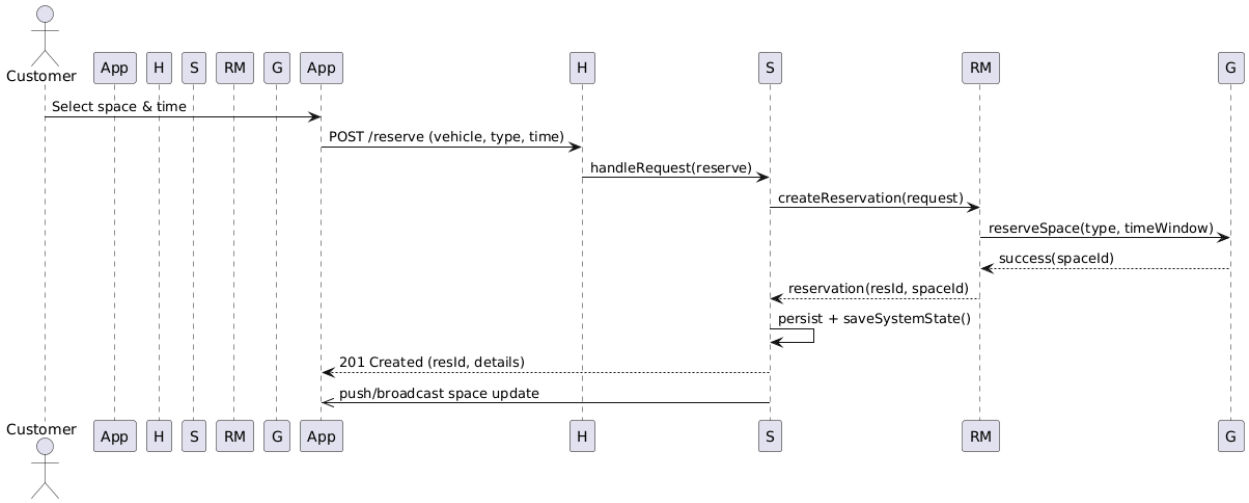
1. Car
2. Truck
3. Electric (EV)
4. Van
5. Handicapped
6. Motorcycle.

- **PaymentProcessor:** This class will be responsible for making payments. The system will support credit/debit card transactions.
- **OperatorClient:** 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.
- **CustomerClient:** Enables the customer (End User) to pay for tickets electronically, view spaces, and reserve a space.
- **AdminClient**: This class will encapsulate the operations that an Admin can perform, like configure the system, manage the parking floor, and set pricing.
- **Reservation**: This class will hold the customer's reservation information, as well as their ID
- **ReservationManager**: This class manages the creation of reservations, as well as manages overdue reservations and their respective grace periods.

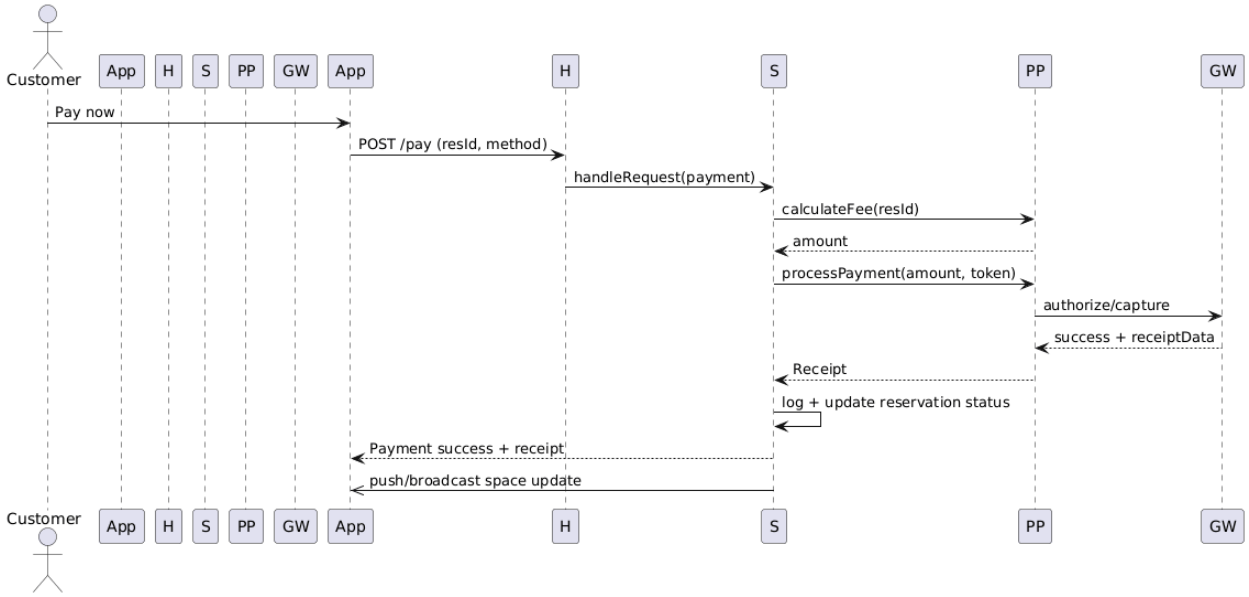


5.5 Sequence Diagrams

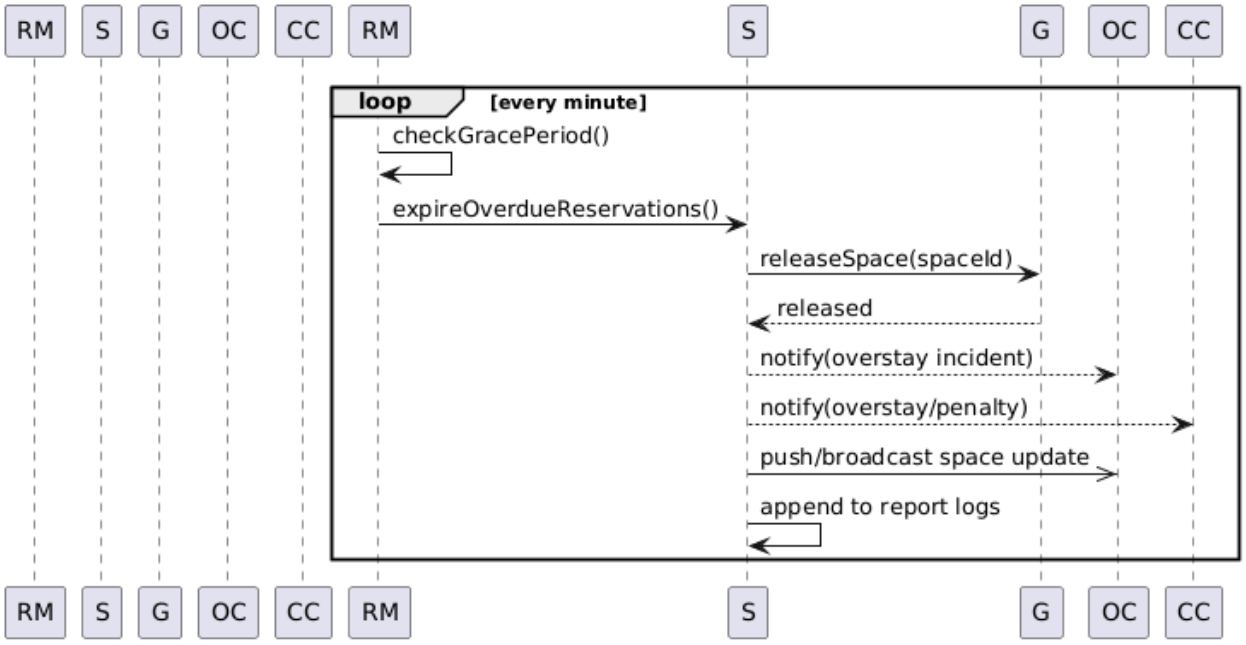
- Customer reserving a parking space



- Customer paying for a reservation



- Overstay Expiration


RM
S
G
OC
CC
RM
S
G
OC
CC
loop
[every minute]
checkGracePeriod()
expireOverdueReservations()
releaseSpace(spaceId)
released
notify(overstay incident)
notify(overstay/penalty)
push/broadcast space update
append to report logs
RM
S
G
OC
CC
RM
S
G
OC
CC