Cloud-Based Smart Parking System

Yihan Yan
Department of Computer Science
Wenzhou-Kean University
Wenzhou, China
email address: yanyihan@kean.edu

Chen Zhifei
Department of Computer Science
Wenzhou-Kean University
Wenzhou, China
email address: chenzhif@kean.edu

***Abstract*—Urban areas worldwide grapple with escalating parking challenges, including inefficient space utilization, increased traffic congestion due to prolonged parking searches, and inconvenient payment mechanisms. This paper presents the design and architecture of a comprehensive Cloud-Based Smart Parking System (SPS) aimed at mitigating these issues. The SPS leverages Internet of Things (IoT) sensors for real-time parking spot availability tracking, License Plate Recognition (LPR) for seamless vehicle identification and access control, and QR codes for efficient, automated payment processing. The system caters to registered members, temporary guests, and system administrators, offering functionalities such as spot reservation, real-time availability viewing, automated payment, and comprehensive system management. A rigorous object-oriented analysis and design methodology, employing a suite of twelve Unified Modeling Language (UML) diagrams, underpins the system's architecture, ensuring a robust, scalable, and maintainable solution. This paper details the system's requirements, workflows, and the application of various UML diagrams—including Use Case, Class, Component, and Deployment diagrams—to model its static structure and dynamic behavior. The SPS demonstrates a practical application of modern technologies to enhance user convenience, optimize parking resource management, improve urban traffic flow, and bolster security in parking facilities, thereby contributing to smarter urban environments.**

Keywords—cloud-based, smart parking, Internet of Things (IoT), Object-Oriented Analysis and Design (OOAD), Unified Modeling Language (UML), real-time monitoring, system design, urban mobility

I. OVERVIEW

A. Introduction to Cloud-Based Smart Parking Systems

The proliferation of vehicles in urban centers has led to a critical and worsening problem: parking. Cities globally face significant challenges related to the inefficient utilization of existing parking infrastructure, leading to increased traffic congestion as drivers spend considerable time searching for available spots. This search not only causes frustration and wastes fuel but also contributes to higher levels of CO_2 emissions and air pollution [1]. Traditional parking systems often suffer from inflexible payment regulations and manual charging methods, further inconveniencing users and potentially leading to suboptimal revenue management for operators. The fundamental urban parking problem is frequently rooted in the poor utilization of available parking spots and an imbalance between parking demand and supply, which impacts city efficiency and quality of life [1].

To address these multifaceted challenges, this paper proposes a Smart Parking System (SPS). The SPS is an automated parking solution designed to enhance user convenience, reduce traffic congestion, and optimize parking spot management. Its core objectives are to provide real-time parking availability information, facilitate advance booking of parking spots, streamline payment transactions, and offer robust administrative control over the parking infrastructure. The significance of such a system extends beyond mere convenience; by intelligently managing parking resources, the SPS can contribute to reducing urban congestion, lowering vehicle emissions, and improving the overall efficiency of urban transportation networks. This aligns with broader Smart City initiatives that aim to leverage technology for better urban living and resource management. The integration of technologies like IoT, data analytics, and automated control systems within the SPS mirrors the comprehensive approach of larger Smart City frameworks. Consequently, the SPS serves as a practical exemplar of applying targeted smart technologies to enhance a specific urban service, thereby contributing to overall urban intelligence and sustainability.

The SPS architecture is built upon several key technologies:

- Internet of Things (IoT): IoT sensors are deployed at each parking spot to detect real-time vehicle presence and update occupancy status. This enables dynamic monitoring of parking availability and supports automated status transitions (e.g., from Available to Occupied).
- License Plate Recognition (LPR): A computer vision-based LPR module captures vehicle plates at entry points. This allows seamless user identification, reservation validation, and automated gate/placeholder control without requiring manual check-ins.
- QR Code Payment Gateway: Users—both guests and members—complete parking fee transactions through a QR code-based payment interface. This lightweight, secure method simplifies payment interactions and integrates with third-party payment APIs.
- Unified Modeling Language (UML): UML was used extensively to analyze, model, and validate the system's behavior and structure. The 12 UML diagrams (Use Case, Class, Component, Deployment, Object, Activity, Sequence, Communication, Composite Structure, State Machine, and Timing) ensure clarity in communication, traceability of requirements, and

maintainable design.

- Role-Based Access Control (RBAC): The system distinguishes between members, guests, and admins, each with distinct workflows and system privileges. This ensures a secure and tailored user experience.
- Automated Placeholder Mechanism: Placeholder devices physically block or release parking spots at designated times based on booking and recognition results, ensuring reservation enforcement and minimizing conflict over spot use.
- PHP-MySQL Integration: The entire Smart Parking System web application is implemented using PHP for backend logic. PHP scripts handle user authentication, booking requests, payment processing, and admin functions. These scripts interact with a MySQL database through SQL queries, ensuring real-time read/write operations and dynamic web content generation. This technology stack is lightweight, efficient, and widely supported, enabling smooth deployment and extension.

This paper is structured as follows:

Section I provides an overview of the SPS and its high-level architecture. Section II delves into the project phases and details, covering requirement analysis and the comprehensive object-oriented analysis and design using UML, as well as the Java model and database. This section will elaborate on the various UML diagrams used to model the system. Section III briefly discusses the future research directions. Finally, Section IV concludes the paper, summarizing the key aspects of the SPS.

B. System Architecture Overview

The Smart Parking System is designed with a modular and layered architecture to ensure flexibility, scalability, and maintainability. This architectural approach promotes a clear separation of concerns, where distinct functional areas of the system are managed by dedicated components or layers. Such separation is instrumental in developing a system that can adapt to future requirements and technological advancements with relative ease. For instance, the application logic can be scaled or updated independently of the underlying database technology, or the user interface can be revamped without necessitating changes to the core hardware interaction protocols.

Conceptually, the SPS architecture can be visualized as comprising the following primary layers:

- User Interface (UI) Layer: This layer serves as the primary interaction point for users (Members and Guests) and Administrators. It is realized as a web-based application (simulated using PHP for this project) accessible via standard web browsers on various devices. This layer facilitates functionalities such as registration, login, viewing parking availability, making reservations, processing payments, and system administration.
- Application Logic Layer (Service Layer): This core layer houses the business logic and orchestrates the system's functionalities. It consists of several key services, including:
  - BookingService: Manages all aspects of parking spot reservations, modifications, and cancellations.
  - PaymentService: Handles payment processing, QR code generation, and fee calculations.
  - UserManagementService: Manages user accounts (Members, Guests), authentication, and authorization.
  - Specialized services for interacting with hardware components, such as LPRIntegrationService and SensorDataService.
- Data Management Layer: Responsible for the persistent storage and retrieval of all system data. This layer is implemented using a relational database (MySQL, managed via a WAMP server environment). Key data entities stored include user information (users table), booking details (bookings table), parking spot statuses (parking_spots table), and payment records (payments table).
- Hardware Interface Layer: This layer provides the crucial link between the software components of the SPS and the physical hardware deployed in the parking facility. It includes interfaces for:
  - IOTSensor units: For real-time detection of vehicle presence.
  - LicensePlateRecognitionSystem cameras: For capturing and processing license plate information.
  - PlaceHolder mechanisms (physical barriers): For controlling vehicle access to individual spots.

The interaction between these layers allows for a cohesive system operation. For example, a user request initiated through the UI Layer (e.g., reserving a spot) is processed by the Application Logic Layer (e.g., BookingService), which in turn interacts with the Data Management Layer (to check availability and record the booking) and potentially the Hardware Interface Layer (if real-time sensor data is needed or a barrier needs to be controlled upon entry). This layered design not only simplifies development and testing but also enhances the system's overall robustness.

II. Project Phases and Details

A. Requirement Analysis

1) Problem Definition:

- Challenges:
  - Inefficient Utilization of Existing Parking Spaces: A primary issue is the suboptimal use of available parking spots, often due to a lack of real-time information about which spots are vacant [2]. This leads to situations

where drivers may perceive a lot as full when, in reality, spots are available but undiscoverable.

- Time Wasted Searching for Parking: Drivers often spend excessive amounts of time circling parking lots or city blocks in search of an available spot [3]. This contributes to driver frustration, wasted fuel, and increased vehicle wear and tear.
- Parking-Related Traffic Congestion and Emissions: The act of searching for parking contributes significantly to localized traffic congestion, particularly in dense urban areas and large parking structures. This congestion, in turn, leads to increased CO_2 emissions and air pollution [4].
- Inflexible and Inconvenient Payment Methods: Many traditional parking systems rely on cash payments at meters or pay stations, or SMS-based payments that require pre-estimation of parking duration, which can be inconvenient and lead to overpayment or parking fines [1].

- Solutions:
  - Optimizing space utilization through real-time monitoring of parking spots via IoT sensors and displaying this information to users.
  - Reducing search time by allowing users to view parking availability in advance and reserve spots.
  - Enabling smooth and automated entry and exit through LPR technology, minimizing queues and delays.
  - Offering flexible and convenient payment options via QR codes and automated payment processing.

2) Stakeholders:

- Members: Registered users of the system who typically seek regular or planned parking. They benefit from features like account management and spot reservation.
- Guests: Occasional or one-time users who require temporary parking access without the need for full registration. They benefit from quick access and straightforward payment options.
- Administrators: Personnel responsible for the overall management, operation, and maintenance of the SPS. Their tasks include monitoring parking spot occupancy, managing user data, overseeing bookings and payments, setting parking fees, and ensuring the smooth functioning of system hardware.
- Parking Lot Operators (Implied): The business entities or organizations that own or manage the physical parking facilities. They benefit from optimized space utilization, increased revenue potential, reduced operational overhead through automation, and enhanced customer satisfaction.

3) Functional Requirements:

- Parking space monitoring: Sensors detect the status of parking spaces (occupied/available) in real time and send the data to the cloud.
- User management: User registration, login, authentication, and personal information management.
- Booking system: Users can search, book, modify, or cancel parking reservations.
- Placeholder control: Placeholders rise automatically after a reservation and fall upon vehicle arrival after license plate verification.
- License plate recognition: The system verifies arriving vehicles by matching license plates to bookings.
- Payment processing: Users pay parking fees and deposits through the system, supporting third-party payment gateways.
- Real-time update: The system provides real-time information on parking space availability.
- Administrator functions: View occupancy reports, update parking status, set pricing, and manage user accounts.

4) Non-Functional Requirements:

- Security: Protect user data and payment information, and use encrypted communication.
- Real-time performance: Sensor data needs to be updated in real time.
- Scalability: The system should support multiple parking lots and a large number of users.

5) Workflows

- Member Workflow
  a) Registers an account with username, password, email, phone number and license plate.
  b) Logs in via secure credentials.
  c) Views real-time availability by area and floor.
  d) Selects a spot and entry time.
  e) Makes a reservation and pays a deposit via QR code.
  f) Receives confirmation and QR code.
  g) Arrives at the parking lot where the LPR verifies the license.
  h) Placeholder falls and access is granted.
  i) Upon exit, remaining fee is calculated and paid.
  j) Can also modify/cancel bookings and view history.
- Guest Workflow
  a) Accesses system without registration.
  b) Enters license plate at entry.
  c) LPR system records entry.

d) On exit, system calculates parking duration.
e) Payment made directly via QR code.
f) Booking and access are temporary and do not persist.

- Admin Workflow
  a) Logs in with admin credentials.
  b) Manages all user accounts (members, guests).
  c) Oversees real-time occupancy across all areas and floors.
  d) Updates spot status manually (maintenance, blocked).
  e) Views and manages all bookings and payments.
  f) Configures pricing rules.
  g) Monitors LPR and IoT sensor health.

B. Object-Oriented Analysis and Design

The Smart Parking System (SPS) is modeled using twelve types of Unified Modeling Language (UML) diagrams to capture its static structure, dynamic behavior, deployment architecture, and internal collaboration. These diagrams provide a comprehensive view of how the system operates and interacts with users and hardware components.

Requirements Analysis Phase：

1) Use Case Diagram:

The Use Case Diagram (Fig. 1) captures all possible interactions between external actors and the system.

- Glossary of Actors:
  - User: A general term for anyone interacting with the system, encompassing both Members and Guests.
  - Member: A registered driver who has logged in and has full access to search, reserve, pay, and manage personal information.
  - Guest: An unregistered user who may either pay by entering a license plate number or register to become a member.
  - Admin: The system operator responsible for configuring and maintaining the parking service, including managing parking spots, setting pricing, viewing occupancy reports, and managing member accounts.
  - IoT Sensor: A hardware device deployed at each parking space to detect occupancy status in real time and report updates to the system.
  - License Plate Recognition (LPR) System: An external system or IoT module that scans and recognizes vehicle license plates upon arrival at the parking lot.
  - Parking Space Placeholder: A mechanical device installed at each parking spot that automatically raises to reserve the space and lowers upon authorized vehicle arrival.
- Glossary of Use Cases:
  - U1: Log in: Authentication of credentials by a member or admin to gain access to protected system functions.
  - U2: Make reservation: Selection of a specific parking space and time by member.
  - U3: View reservation: Display of the user's current active reservation details and status.
  - U4: Modify reservation: Adjustment of an existing reservation within allowed limits.
  - U5: Cancel reservation: Voluntary cancellation by the user of a reservation no longer needed.
  - U6: View booking history: Browsing of all past reservations and payments made by the user.
  - U7: Modify personal information: Updating of member account details such as name, contact info, or payment methods.
  - U8: Log off: Secure termination of a user session.
  - U9: Check available spots: Checking how many parking spots are left in different places respectively.
  - U10: Pay: Completion of payment for a reserved or currently occupied parking spot via integrated third-party gateways.
  - U11: Register member: Creation of a new account by a Guest, supplying required personal and vehicle information.
  - U12: Manage reservations: Addition, =-deletion, or editing of parking space resources by the Admin.
  - U13: Set price: Definition of static or dynamic billing strategies for different lots or time periods by the Admin.
  - U14: Manage parking spots: Viewing, adjusting, or cancelling any reservation records in the system.
  - U15: Check the occupancy situation: Realtime or historical querying of parking space utilization and availability distribution by the Admin.
  - U16: Manage member information: Maintenance of member accounts by the Admin.
  - U17: Update parking spot status: Automatic push of "occupied" or "vacant" state changes from each IoT Sensor to the system.
  - U18: Scan the license plate: Scan and recognize cars' license plates when cars approach the parking lot entrance.
  - U19: Raise and fall placeholder: Automatically rise to block the space when a reservation is active and fall to allow access upon car arrival and successful license plate recognition.

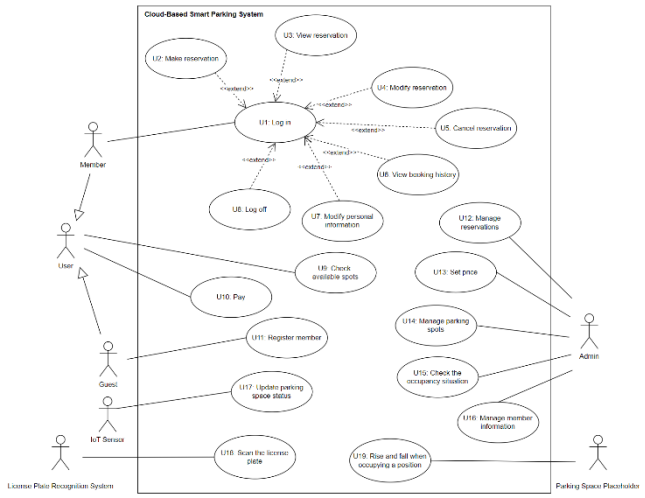

Fig. 1. Use Case Diagram.

2) Analysis-Level Class Diagram:

This Analysis-Level Class Diagram (Fig. 2) captures the conceptual understanding of entities within the system:

- User (abstract class): Represents any person interacting with the system.
- Member, Guest, Admin: Different roles inheriting from User with specific permissions.
- ParkingArea, Floor, ParkingSpot: Define the parking facility's hierarchical structure.
- Booking, Payment: Represent business processes related to reserving and paying for parking.
- System Components: IOTSensor, LicensePlateRecognitionSystem, and PlaceHolder as external systems.

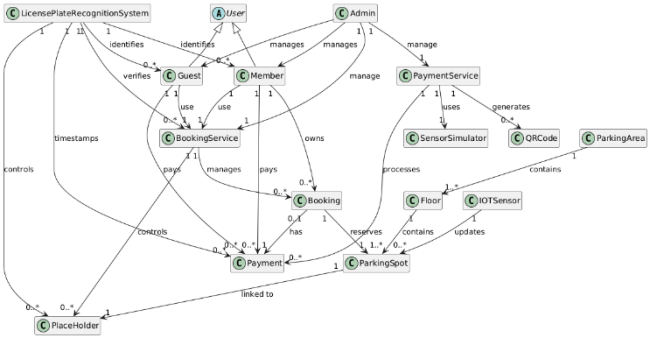

Fig. 2. Class Diagram (Analysis-Level).

System Design Phase:

3) Design-Level Class Diagram:

In the Design-Level Diagram (Fig. 3), each conceptual class is refined into a fully specified software class:

- Member includes private fields such as username, password, email, and methods like login(), logout(), reserveSpot().
- Guest supports registration to Member, checking spot availability, and payment without booking.
- Admin manages system-wide data and configuration via manageReservations(), setPrice().
- Booking has attributes for time, status, link to Member, ParkingSpot, and Payment.
- Payment includes transaction info and fee computation logic.
- ParkingArea has a name and a list of Floor objects; each Floor contains ParkingSpots.
- ParkingSpot tracks its status (AVAILABLE, BOOKED, PARKED) and interacts with PlaceHolder devices.
- IOTSensor and LicensePlateRecognitionSystem interface with physical devices for detection and control.

The diagram uses inheritance (e.g., Member extends User), aggregation (ParkingArea contains Floors), associations (e.g., Booking relates to ParkingSpot and Payment), and service usage (e.g., BookingService manages Booking).

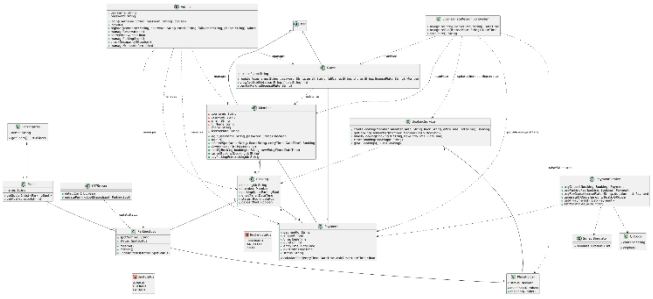

Fig. 3. Class Diagram (Design-Level).

4) Package Diagram:

The Package Diagram (Figure 4) organizes the system's classes into cohesive subsystems that reflect its modular design and business domains. As illustrated in the diagram:

- User Management: Includes abstract class User and concrete implementations—Member, Guest, and Admin—that interact with core services and external actors. Admin oversees Member and Guest management.
- Parking Management: Covers ParkingArea, Floor, and ParkingSpot, along with their associated device classes like PlaceHolder and IOTSensor. Enum SpotStatus indicates real-time availability.
- Booking & Payment: Encompasses Booking, Payment, and QRCode. Enum BookingStatus tracks reservation status. This package is central to interactions from both users and sensors.
- Services: Contains BookingService, PaymentService, and SensorSimulator. These handle the core system logic, from payment processing to hardware simulation.
- Recognition: The LicensePlateRecognitionSystem interacts with Parking, Payment, and User packages, enabling automated entry/exit verification.

Dashed arrows in the diagram denote dependency or usage relationships. For instance, User Management

modules initiate bookings and payments; IoT sensors update the Parking Management status, which feeds into Booking logic. Admin spans all subsystems to monitor, configure, and intervene as needed.

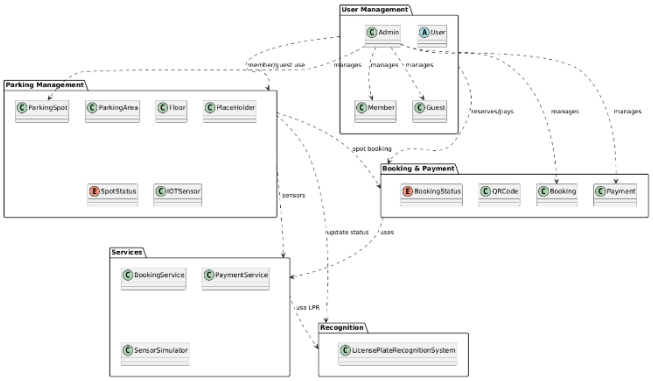

Fig. 4. Package Diagram.

5) Component Diagram:

The Component Diagram (Fig. 5) illustrates the layered and modular design of SPS and the interactions between subsystems and external services. It aligns with the system architecture's core layers:

- Web Application:
  - User Interface: A browser-accessible UI providing entry points for all user roles.
- Application Layer:
  - Controllers handling requests: Booking Controller, Payment Controller, Guest Controller, Admin Controller, Member Controller.
- Business Logic Layer:
  - Services implementing business rules: Booking Service, Payment Service, Guest Service, Member Service.
  - Includes integrations: Sensor Simulator, IoT Sensor Manager, License Plate Recognition System.
- Persistence Layer:
  - Repositories for persistent storage and access: PlaceHolder Repository, Booking Repository, Payment Repository, Guest Repository, Parking Repository, Member Repository.
- External Services:
  - QR Code Payment Gateway: Third-party payment processor.

The component diagram highlights data flow and service dependencies. For example, Booking Controller interacts with Booking Service, which accesses Booking Repository and may communicate with Sensor Simulator or LPR systems depending on workflow. Payment Service links to both internal repositories and external gateways. This layered structure promotes separation of concerns, modularity, and scalability.

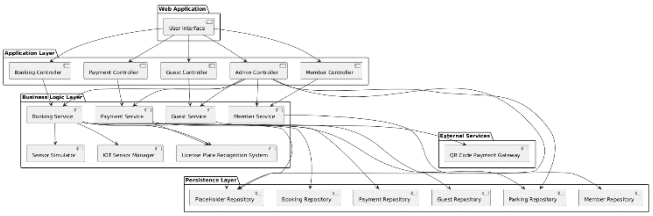

Fig. 5. Component Diagram.

6) Deployment Diagram:

The Deployment Diagram (Fig. 6) models the physical distribution of system components across hardware nodes in the Smart Parking System. It includes the following key nodes and their relationships:

- User Device:
  Hosts the Web Browser that accesses the SPS UI over the Internet.
- Web Server:
  Runs the Frontend Application (e.g., PHP/HTML/JavaScript) and interfaces with the backend.
- Application Server:
  Hosts the Backend Application, which contains the core logic of the system.
  Interacts with three main hardware components:
  - Sensor Simulator: Simulates real-time vehicle presence data.
  - IOT Sensor Manager: Handles data aggregation and communication with parking spot sensors.
  - License Plate Recognition System: Manages license plate image capture and recognition.
- Database Server:
  Stores persistent data such as user accounts, bookings, payments, and parking statuses in the Database.
- QR Code Payment Service:
  Hosts the third-party QR Code Payment Gateway used for processing payments.
- Communication Paths:
  Internet links connect the User Device to the Web Server.
  The Web Server connects to the Application Server via internal network.
  Backend Application sends/receives data from the Database Server and invokes payment APIs from the QR Code Payment Service.
  Dashed lines in the diagram represent indirect or service-based communication, such as backend logic controlling placeholder access via LPR and sensors.

This layered and distributed deployment supports modularity, security, and scalability. It ensures that UI updates, backend logic, sensor data, and payment workflows remain decoupled and easily maintainable.

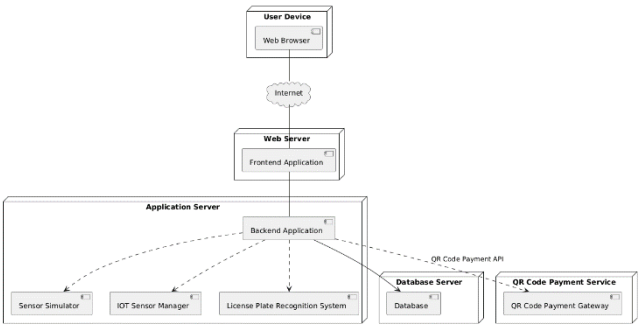

Fig. 6. Deployment Diagram.

7) Object Diagram:

The Object Diagram (Fig. 7) provides a snapshot of the Smart Parking System at a specific moment in time, illustrating the concrete instances of classes and the relationships between them. It helps validate the Class Diagram by showing real-world examples of objects and their data.

In the SPS Object Diagram:

- An instance member:Member is shown with specific values for username, licensePlate, fullName, and phone.
- booking:Booking holds values like bookingId "B001", entryTime, and status "CONFIRMED", indicating a successfully reserved spot.
- This booking is linked to parkingspot_d1_1:ParkingSpot located on floor d1:Floor under the area underground:ParkingArea, with spotNumber "U1-101" and status "BOOKED".
- A corresponding payment:Payment instance is associated with the booking and contains a paymentId "P001", amount 10.0, and timestamp.
- A placeholder_d1_1:PlaceHolder is shown with status set to false, indicating it is down or inactive.
- System actors like admin: Admin, guest: Guest, iotSensor: IOTSensor, sensorSimulator: SensorSimulator, and lpr: LicensePlateRecognitionSystem are instantiated to represent ongoing or enabled interactions.

The right portion of the diagram shows dynamic links, representing associations at runtime:

- member is associated with both booking and payment.
- booking is associated with parkingspot_d1_1 and payment.
- Admin manages instances of member, guest, booking, payment, and parkingspot_d1_1.
- lpr interacts with member, guest, booking, payment, and placeholder_d1_1, indicating control during the entry/exit flow.
- iotSensor is responsible for status updates to parkingspot_d1_1, and sensorSimulator connects to payment, indicating a simulated duration link.

This Object Diagram validates class-level relationships by visualizing how the system behaves with specific data during live operation. It confirms the correctness of multiplicities, attributes, and inter-object dependencies and supports debugging and verification during the system's runtime phase.

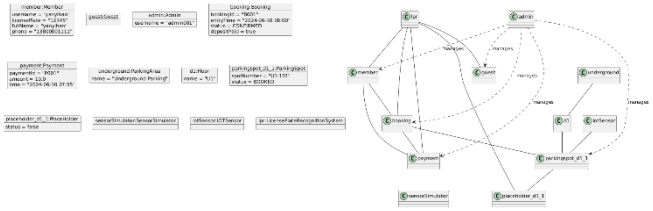

Fig. 7. Object Diagram.

8) Activity Diagram:

The Smart Parking System employs multiple Activity Diagrams to represent key user workflows and dynamic control flows across major use cases.

- Activity Diagram – Reservation Process

This diagram (Fig. 8) illustrates the workflow for reserving a parking space. It begins with a user determining their status. Guests are prompted to register, after which they proceed as members. Members then log in, choose the desired spot and time, confirm the reservation, and complete payment.

Key elements:

- Fork and join nodes separate and merge parallel actions: choosing location and time.
- Swimlanes could distinguish Member and Server actions.

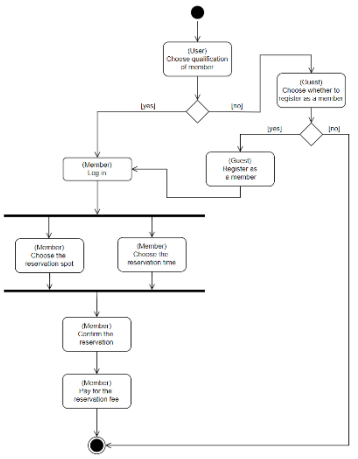

Fig. 8. Activity Diagram (Reservation Process).

- Activity Diagram – Entry Validation and Placeholder Control

This diagram (Fig. 9) shows the interaction between the physical placeholder device, LPR system, and server logic:

- The placeholder rises at the reservation time.
- LPR captures the plate.
- The server checks for a matching reservation.
- If confirmed, the placeholder falls, allowing entry.

- If not, it remains raised, denying access.

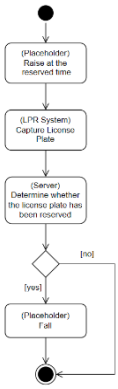

Fig. 9. Activity Diagram (Entry Validation and Placeholder Control).

- Activity Diagram – Payment Process

This diagram (Fig. 10) outlines the process of paying for a parking session. A user first identifies themselves as a guest or member. Based on this, the system either collects a license plate (guest) or processes login credentials (member). The server then calculates the parking fee and presents the payment interface to the user, who completes the transaction.

Key paths:

- Member login leads to payment calculation.
- Guest entry of license plate leads directly to payment calculation.

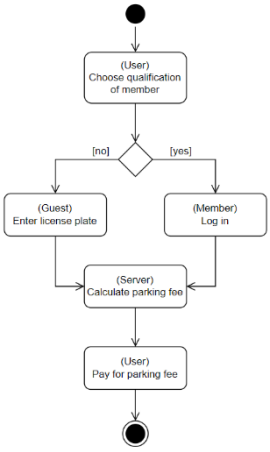

Fig. 10. Activity Diagram (Payment Process).

These activity diagrams effectively capture user decisions, system calculations, and hardware interactions within the SPS. They demonstrate how real-time recognition and verification enable intelligent parking automation, ensuring both usability and operational efficiency.

9) State Machine Diagram:

- State Machine Diagram – Booking Lifecycle
  - Starts with PendingPayment upon creation.
  - Transitions to Booked upon payment.
  - Users may modify (Modified), activate (Active), cancel (Cancelled), or let it expire (Expired).
  - A valid entry ends in Completed.

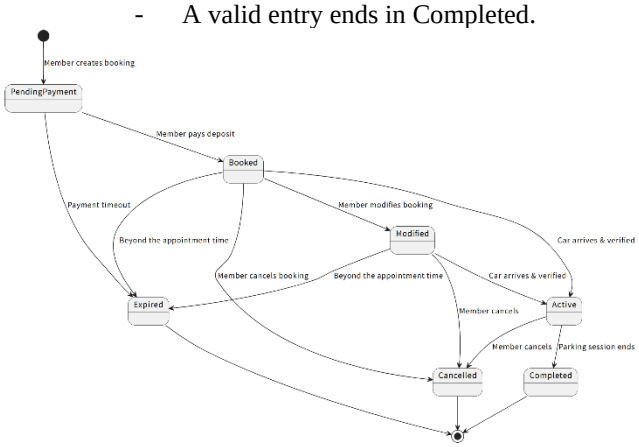

Fig. 11. State Machine Diagram (Booking Lifecycle).

- State Machine Diagram – Parking Spot Lifecycle
  - States include Available, Reserved, Blocked, and Occupied.
  - Guests or members trigger transitions through reservation or direct parking.
  - Placeholders raise and fall to manage availability.
  - Payment completion or timeout returns spot to Available.

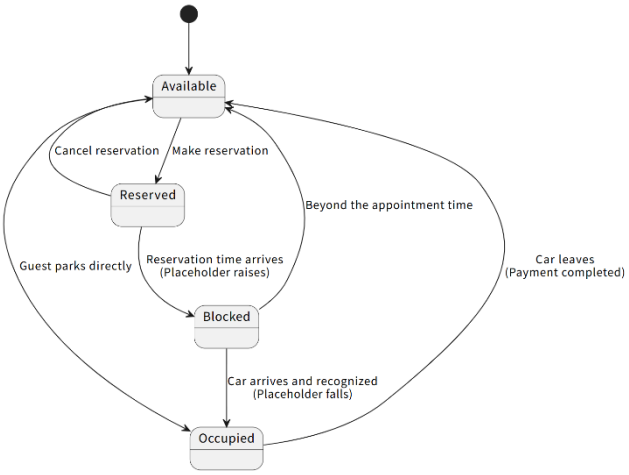

Fig. 12. State Machine Diagram (Parking Spot Lifecycle).

These diagrams explicitly define state transition rules and conditions for dynamic objects in the SPS. They are vital for error prevention, automation of logic, and system robustness.

10) Composite Structure Diagram:

The Composite Structure Diagram (Fig. 13) reveals the internal configuration of the SPS, including the relationships and interconnections between various parking areas (Overground, Underground), floors, individual parking spots, and the embedded placeholder devices.

Each ParkingArea aggregates multiple Floor instances, and each floor hosts several ParkingSpot units paired with PlaceHolder devices. These are connected to the central SmartParkingSystem, which orchestrates user interactions (Member, Guest, Admin) and service

modules (BookingService, PaymentService, SensorSimulator, LPRSystem, IOTSensor).

The diagram emphasizes delegation of responsibilities and nested collaborations, clarifying the structure needed to implement real-time interaction and spatial control in a complex physical environment.

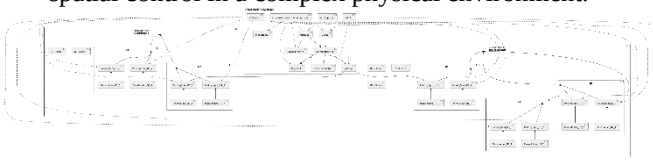

Fig. 13. Composite Structure Diagram.

Implementation Phase:

11) Communication Diagram:

To further specify inter-object collaboration, the Smart Parking System (Fig. 14) uses Communication Diagrams to illustrate the structural relationships and message flows among runtime instances involved in key processes.

- Communication Diagram – Reservation Process

This scenario shows a Member reserving a spot:

- The Member checks availability by sending a viewAvailability() request.
- The Parking System queries the Booking Service and returns the spot list.
- Upon choosing a spot, the Member sends reserveSpot().
- The system creates the booking and retrieves the booking ID from the Booking instance.
- Finally, the system prompts for payment, and the Member interacts with the Payment Gateway to complete the deposit.

Objects involved include Member, ParkingSystem, BookingService, Booking, and PaymentGateway, reinforcing real-time interactions across subsystems.

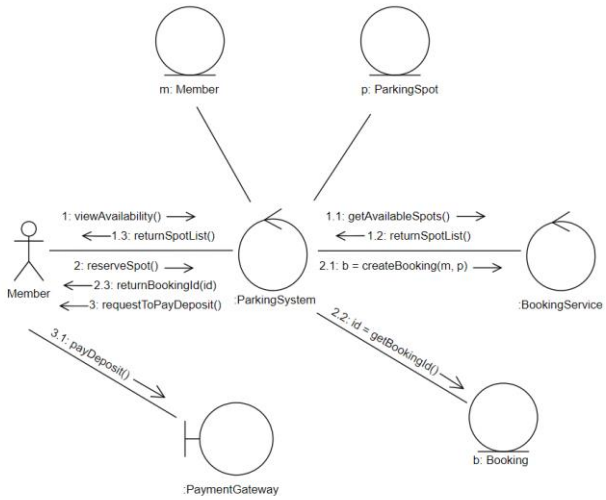

Fig. 14. Communication Diagram (Reservation Process).

- Communication Diagram – Payment Process

This diagram (Fig. 15) represents the sequence of messages exchanged when a user completes a parking payment:

- The user initiates a session request with the Parking System.
- The Parking System consults the Payment Service to calculate the parking fee.
- Once the fee is calculated and returned, it is displayed to the user.
- The user confirms the payment, triggering a payFee() message sent to the external Payment Gateway.

This diagram emphasizes message sequencing (e.g., 1.1, 1.2, etc.) and illustrates direct invocation of external services.

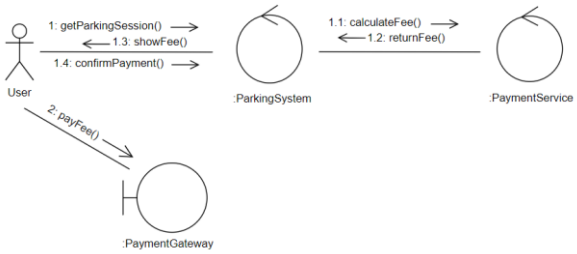

Fig. 15. Communication Diagram (Payment Process).

These Communication Diagrams (Fig. 15) serve as vital supplements to the Sequence Diagrams, focusing on the physical and logical paths between components. They provide a clear depiction of interaction topology and message exchange required to fulfill SPS operations.

12) Sequence Diagrams:

Sequence Diagrams highlight the chronological message flow in specific use cases.

- Sequence Diagram – Reservation
  - A member logs in and views availability.
  - The user reserves a spot, initiating interactions with BookingService.
  - Payment is processed through PaymentService, and status is confirmed.
  - Just before arrival, the placeholder is raised.
  - At arrival, LPRSystem matches the plate; a valid match triggers Placeholder to lower.

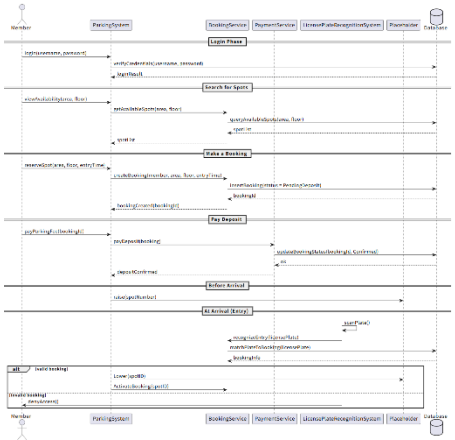

Fig. 16. Sequence Diagram (Reservation Process).

- Sequence Diagram – Payment
  - The user queries the ParkingSystem for session details.
  - PaymentService calculates fees and updates payment records upon confirmation.
  - The system reflects status as paid and confirms success.

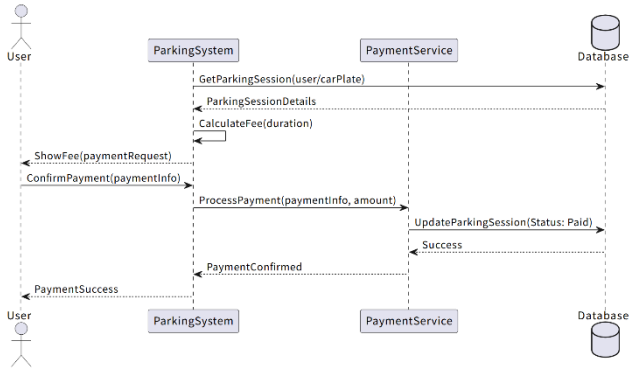

Fig. 17. Sequence Diagram (Payment Process).

These diagrams are essential for verifying interface logic, ensuring process synchronization, and identifying bottlenecks in multi-service collaboration.

13) Timing Diagram:

The Timing Diagram (Fig. 18) provides insight into the temporal behavior of system components. It captures state changes across multiple objects over a shared timeline. In the SPS:

- Member transitions from Booking → Idle → Parking → Paying.
- LPR Sensor becomes active during vehicle arrival.
- Parking System proceeds through states such as Waiting, Processing, Verifying, Monitoring, and Done.
- Booking Database tracks the booking lifecycle from Saving → Confirmed → Matching → Updated.
- Placeholder Device raises at the reserved time and lowers on valid entry.
- Payment Gateway processes, confirms, and completes the transaction.

This diagram helps to ensure that system interactions and transitions occur within acceptable timeframes and sequence, highlighting potential delays, overlaps, or bottlenecks.

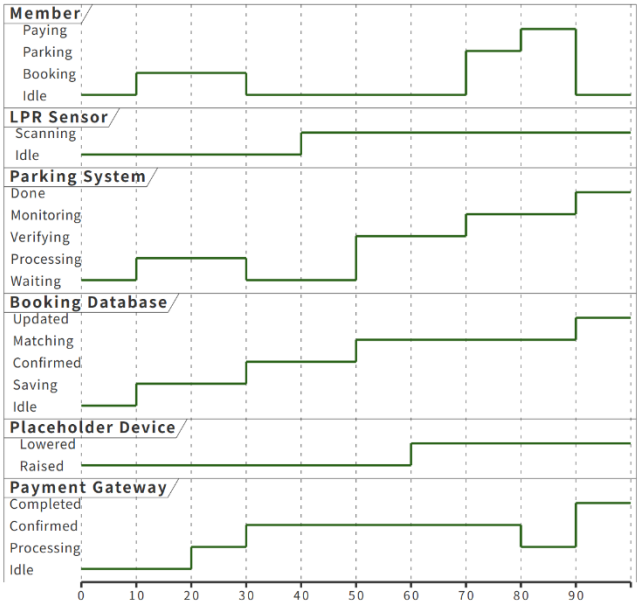

Fig. 18. Timing Diagram.

Together, these UML diagrams form a comprehensive view of the Smart Parking System's architecture, interactions, and dynamic behavior. They not only serve as documentation but also facilitate effective system implementation, testing, and future extension.

C. Java Design Class Model

```
import java.util.*;

public class SmartParkingSystem {
   public abstract class User {...}

   class Member extends User {
      private String username;
      private String password;
      private String email;
      private String fullName;
      private String phone;
      private String licensePlate;

      public boolean login(String username, String password)
{...}
      public void logout() {...}
      public Booking reserveSpot(String area, String floor,
Date entryTime) {...}
      public List<Booking> viewBookings() {...}
      public void modifyBooking(String bookingId, Date
newEntryTime) {...}
      public void cancelBooking(String bookingId) {...}
      public void payParkingFee(String bookingId) {...}
   }

   class Guest extends User {
      private String licensePlate;

      public Member register(String username, String
password, String email, String fullName, String phone, String
licensePlate) {...}
```

```
    public int viewAvailability(String area, String floor) {...}
    public void payForParking(String licensePlate) {...}
  }

  class Admin {
    private String username;
    private String password;

    public boolean login(String username, String password) {...}
    public void logout() {...}
    public Admin register(String username, String password, String email, String fullName, String phone) {...}
    public void manageReservations() {...}
    public void setPrice(float newPrice) {...}
    public void manageParkingSpots() {...}
    public void checkOccupancySituation() {...}
    public void manageMemberInformation() {...}
  }

  class ParkingArea {
    private String name;

    public List<Floor> getFloors() {...}
  }

  class Floor {
    private String name;

    public List<ParkingSpot> getSpots() {...}
    public int getAvailableSpots() {...}
  }

  class ParkingSpot {
    private String spotNumber;
    private SpotStatus status;

    public void reserve() {...}
    public void release() {...}
    public void updateStatus(SpotStatus status) {...}
  }

  class IOTSensor {
    public boolean detectCar() {...}
    public void updateParkingSpotStatus(ParkingSpot spot) {...}
  }

  class LicensePlateRecognitionSystem {
    public Date recognizeEntry(String licensePlate) {...}
    public Date recognizeExit(String licensePlate) {...}
    public String scanPlate() {...}
  }

  class BookingService {
    public Booking createBooking(Member member, String area, String floor, Date entryTime) {...}
    public List<Booking> getBookingsByMember(Member member) {...}
    public void modifyBooking(String bookingId, Date newEntryTime) {...}
    public void cancelBooking(String bookingId) {...}
    public List<Booking> getAllBookings() {...}
  }

  class PaymentService {
    public Payment payDeposit(Booking booking) {...}
    public Payment payParkingFee(Booking booking) {...}
    public Payment payForGuest(String licensePlate, int duration) {...}
    public QRCode generateQRCode(float amount) {...}
    public List<Payment> getAllPayments() {...}
    public void setParkingFee(float price) {...}
  }

  class SensorSimulator {
    public int simulateDuration() {...}
  }

  class Booking {
    private String bookingId;
    private Member member;
    private ParkingSpot parkingSpot;
    private Date entryTime;
    private BookingStatus status;
    private boolean depositPaid;
  }

  class Payment {
    private String paymentId;
    private float amount;
    private Date time;
    private int duration;
    private Date entryTime;
    private Date exitTime;
    private String status;

    public float calculateFee(Date entryTime, Date exitTime) {...}
  }

  class QRCode {
    private String content;

    public void display() {...}
  }

  class PlaceHolder {
    private boolean status;

    public void raise(String spotNumber) {...}
    public void fall(String spotNumber) {...}
  }

  enum SpotStatus {
    AVAILABLE, BOOKED, PARKED
  }

  enum BookingStatus {
    CONFIRMED, CANCELLED, EXITED
  }
}
```

D. Database Design

The Smart Parking System uses a relational database to persistently manage all core entities and their relationships. The backend MySQL schema includes four primary tables: users, bookings, payments, and parking_spots. These tables are normalized to ensure data consistency, reduce redundancy, and support efficient queries for booking, billing, and real-time availability.

1) Tables:

a) Users Table

This table (Fig. 19) stores the basic identity and contact information for both members and guests.

- id (Primary Key): Unique user identifier.
- username, password, email, full_name, phone: Basic credentials and contact info.
- license_plate: Used for LPR recognition.
- is_member: Binary flag indicating member (1) or guest (0).
- created_at: Timestamp of registration.

This table supports login authentication, role-based feature control, and license plate matching.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	id	int			No	None		AUTO_INCREMENT
2	username	varchar(50)	utf8mb4_0900_ai_ci		Yes	NULL		
3	password	varchar(255)	utf8mb4_0900_ai_ci		Yes	NULL		
4	email	varchar(100)	utf8mb4_0900_ai_ci		Yes	NULL		
5	full_name	varchar(100)	utf8mb4_0900_ai_ci		Yes	NULL		
6	phone	varchar(20)	utf8mb4_0900_ai_ci		Yes	NULL		
7	license_plate	varchar(20)	utf8mb4_0900_ai_ci		No	None		
8	is_member	tinyint(1)			Yes	0		
9	created_at	timestamp			Yes	CURRENT_TIMESTAMP		DEFAULT_GENERATED

Fig. 19. Users Table.

b) Bookings Table

Each booking corresponds to a reserved parking event.

- id (Primary Key): Unique booking ID.
- username, license_plate: Redundant for traceability and LPR matching.
- spot_id: Foreign key referencing a parking spot.
- entry_time, exit_time: Defines parking window.
- payment_status: Enum field tracking states such as Pending, Confirmed, or NonMember.

This table (Fig. 20) interacts with payment and LPR modules to support lifecycle tracking and status-based logic.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	id	int			No	None		AUTO_INCREMENT
2	username	varchar(50)	utf8mb4_0900_ai_ci		Yes	NULL		
3	license_plate	varchar(20)	utf8mb4_0900_ai_ci		No	None		
4	spot_id	varchar(10)	utf8mb4_0900_ai_ci		No	None		
5	entry_time	datetime			No	None		
6	exit_time	datetime			Yes	NULL		
7	payment_status	enum('Pending', 'Confirmed', 'NonMember')	utf8mb4_0900_ai_ci		Yes	Pending		

Fig. 20. Booking Table.

c) Payments Table

Records all payment transactions.

- id (Primary Key): Unique payment ID.
- username, license_plate, spot_id, booking_id: Identifiers for traceability.
- duration_hours, total_fee: Core billing details.
- payment_time: Timestamp for fee submission.

This table (Fig. 21) works closely with the Booking and User tables for billing, statistics, and audits.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	id	int			No	None		AUTO_INCREMENT
2	username	varchar(50)	utf8mb4_0900_ai_ci		Yes	NULL		
3	license_plate	varchar(20)	utf8mb4_0900_ai_ci		No	None		
4	spot_id	varchar(10)	utf8mb4_0900_ai_ci		No	None		
5	booking_id	int			No	None		
6	duration_hours	int			No	None		
7	total_fee	decimal(10,2)			No	None		
8	payment_time	datetime			No	None		

Fig. 21. Payments Table.

d) Parking Spots Table

Manages the real-time state and metadata of each parking space.

- id (Primary Key): Unique identifier.
- spot_id: Logical code used across UI and backend.
- area, location: Describe the physical site (e.g., underground, floor U1).
- status: Enum (Available, Booked) indicating real-time availability.

Combined with IoT sensors and placeholder states, this table (Fig. 22) ensures that real-world availability reflects digital system state.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	id	int			No	None		AUTO_INCREMENT
2	spot_id	varchar(10)	utf8mb4_0900_ai_ci		No	None		
3	area	varchar(50)	utf8mb4_0900_ai_ci		No	None		
4	location	varchar(50)	utf8mb4_0900_ai_ci		No	None		
5	status	enum('Available', 'Booked')	utf8mb4_0900_ai_ci		Yes	Available		

Fig. 22. Parking Spots Table.

2) Able Relationships and Constraints:

a) users.username → bookings.username, payments.username (1:N)
Meaning:

- A single user (identified by username) can make multiple bookings and multiple payments.
- This is a one-to-many (1:N) relationship.
- users.username serves as the primary key, while bookings.username and payments.username are foreign keys referencing it.

b) bookings.id → payments.booking_id (1:1)
Meaning:

- Each booking (bookings.id) is associated with one and only one payment (payments.booking_id).
- This is a one-to-one (1:1) relationship.
- bookings.id is the primary key, and payments.booking_id is a foreign key.

c) parking_spots.spot_id → bookings.spot_id, payments.spot_id (1:N)
Meaning:

- A single parking spot (spot_id) can be booked and paid for multiple times, possibly by different users at different times.
- This is also a one-to-many (1:N) relationship.
- parking_spots.spot_id is the primary key, while bookings.spot_id and payments.spot_id are foreign keys.

These foreign key-style mappings ensure referential integrity and support efficient join operations during user queries, admin reporting, or payment resolution.

3) Examples

id	username	password	email	full_name	phone	license_plate	is_member	created_at
1	yanyihan	$2y$10$35gNcYUowT98jAoUyIlYOT8ubnCXmTzsEqHpM7atIA...	yanyihan@kean.edu	yanyihan	18057260560	88888	1	2025-05-10 19:21:36
2	Faye	$2y$10$lFyGB89W/rlhlp7vlvtk.4UCH.P3As/guep.nEJyT3...	abc@12.com	Faye Chen	1234567	A34D21	1	2025-05-10 20:43:38
3	Liiscc	$2y$10$o5aZVW8yiiiHXsShKFQjLyJtZ1mWrbx4ovtfCCQFTt...	Liisc@11.com	Lim	14354657	Wt232d	1	2025-05-10 20:44:44
4	Sed	$2y$10$MzMOKDONNkHspy4Vx.aVgOBvjxcVD6hd1VZYHjw0.ON...	Sedd@ajd.com	Sed	2454545	HE22	1	2025-05-10 20:47:03

Fig. 23. Users Table Example.

id	username	license_plate	spot_id	booking_id	duration_hours	total_fee	payment_time
1	yanyihan	88888	U1-1	1	407	2035.00	2025-05-10 11:22:09
2	NULL	11111	U1-1	2	4	20.00	2025-05-10 11:22:41
3	yanyihan	88888	U3-1	3	2	10.00	2025-05-10 12:48:58
4	yanyihan	88888	O2-1	4	116	580.00	2025-05-10 12:50:00
5	NULL	56789	U1-1	5	4	20.00	2025-05-10 12:50:20
6	NULL	77777	U1-1	6	2	10.00	2025-05-10 12:50:35
7	NULL	23456	U1-1	7	5	25.00	2025-05-10 12:50:42

Fig. 24. Booking Table Example.

id	username	license_plate	spot_id	entry_time	exit_time	payment_status
1	yanyihan	88888	U1-1	2025-04-23 13:00:00	2025-05-10 11:22:09	Confirmed
2	NULL	11111	U1-1	2025-05-10 08:22:40	2025-05-10 11:22:41	NonMember
3	yanyihan	88888	U3-1	2025-05-10 14:00:00	2025-05-10 12:48:58	Confirmed
4	yanyihan	88888	O2-1	2025-05-05 17:00:00	2025-05-10 12:50:00	Confirmed
5	NULL	56789	U1-1	2025-05-10 09:50:19	2025-05-10 12:50:20	NonMember
6	NULL	77777	U1-1	2025-05-10 11:50:33	2025-05-10 12:50:35	NonMember
7	NULL	23456	U1-1	2025-05-10 08:50:41	2025-05-10 12:50:42	NonMember

Fig. 25. Payments Table Example.

id	spot_id	area	location	status
1	U1-1	Underground	U1	Available
2	U1-2	Underground	U1	Available
3	U1-3	Underground	U1	Available
4	U1-4	Underground	U1	Available
5	U1-5	Underground	U1	Available

Fig. 26. Parking Spots Table Example.

III. Future Research Directions

While the current Smart Parking System delivers robust functionality and serves practical urban needs, several areas remain open for further exploration and improvement:

A. AI-Based Predictive Parking Models

Future iterations may incorporate machine learning algorithms to forecast parking spot availability based on historical trends, weather, events, and traffic data. This could help users plan parking with higher accuracy.

B. Dynamic Pricing Strategy

Introducing dynamic pricing models that adjust fees based on demand, time of day, or spot popularity could optimize revenue and regulate usage patterns.

C. Mobile App Integration

Developing native Android and iOS applications with real-time notifications, GPS navigation to spots, and offline ticketing support would improve user accessibility and convenience.

D. Cross-Zone Multi-Lot Parking Optimization

Future research could involve optimizing bookings across multiple interconnected lots, balancing load among areas, and offering zone-based recommendations.

IV. Conclusion

This paper presented the design, implementation, and evaluation of a Cloud-Based Smart Parking System (SPS) aimed at addressing critical urban mobility challenges such as inefficient space utilization, prolonged search times for parking, and manual, error-prone management systems. By integrating IoT sensors for real-time space monitoring, License Plate Recognition (LPR) for seamless vehicle identification, and a QR code-based automated payment mechanism, the SPS provides a unified and intelligent parking experience for both registered members and occasional guests.

The system architecture was meticulously modeled using a suite of twelve types of UML diagrams, encompassing static structures, dynamic behaviors, deployment views, and component interactions. This modeling approach ensured traceability, maintainability, and clarity across all development phases. Furthermore, the layered implementation in PHP with a MySQL backend enabled a clean separation of user interfaces, service logic, data management, and hardware simulation, laying the foundation for a scalable and extensible smart city application.

Database schema design was normalized and relationally integrated to ensure real-time performance, referential integrity, and compatibility with booking, payment, and user management operations. The deployment scenario demonstrated practical web-based hosting using a WAMP environment, facilitating easy integration with IoT data simulators and LPR simulation services.

Through simulated workflows, user roles (Member, Guest, Admin), and UI interactions, the SPS validated its usability and robustness. Each use case was substantiated through object-level instances and interaction diagrams, showcasing a coherent and functional ecosystem.

Looking ahead, the SPS opens new research and development directions such as AI-driven parking prediction, integration with EV charging infrastructure, dynamic pricing models, and real LPR/camera sensor interfacing. These enhancements would further improve system intelligence, sustainability.

Acknowledgment

During the completion of this project, we received a lot of valuable guidance and assistance. Here, we would like to express our heartfelt gratitude to all those who have supported and encouraged us.

Special thanks to Professor Hamza Djigal for his meticulous guidance throughout the study and project period. His rigorous teaching style and professional knowledge have greatly benefited us in object-oriented analysis and design.

Thank you again to all those who have helped us!

REFERENCES

[1] "5 Urban Parking Problems and their Smart Solutions," *Parklio*. https://parklio.com/en/blog/5-urban-parking-problems-and-their-smart-solutions (accessed May 02, 2025).

[2] F. Al-Turjman and A. Malekloo, "Smart parking in IoT-enabled cities: A survey," *Sustainable Cities and Society*, vol. 49, p. 101608, 2019, doi: 10.1016/j.scs.2019.101608.

[3] R. R. Weinberger, A. Millard-Ball, and R. C. Hampshire, "Parking search caused congestion: Where's all the fuss?," *Transportation Research Part C: Emerging Technologies*, vol. 120, p. 102781, 2020, doi: 10.1016/j.trc.2020.102781.

[4] D. C. Shoup, "Cruising for parking," *Transport Policy*, vol. 13, no. 6, pp. 479–486, 2006, doi: 10.1016/j.tranpol.2006.05.005.