Low-Level Design for Smart Parking Lot Backend System

Problem Statement

Imagine a parking lot in an urban area with multiple floors and numerous parking spots. Your task is to create a low-level design for a system that efficiently manages the parking process. The system should automatically assign parking spots based on vehicle size and availability, track the time each vehicle spends in the parking lot, and calculate parking fees upon exit.

Functional Requirements

1. Parking Spot Allocation: Automatically assign an available parking spot to a vehicle when it enters, based on the vehicle's size (e.g., motorcycle, car).
2. Check-In and Check-Out: Record the entry and exit times of vehicles.
3. Parking Fee Calculation: Calculate fees based on the duration of stay and vehicle type.
4. Real-Time Availability Update: Update the availability of parking spots in real-time as vehicles enter and leave.

Design Aspects

1. Data Model

Database Schema

1. Users Table
    - user_id (PK)
    - username
    - password
    - vehicle_type (Enum: motorcycle, car)
2. Vehicles Table
    - vehicle_id (PK)
    - user_id (FK)

    - vehicle_type (Enum: motorcycle, car)
    - license_plate
3. ParkingSpots Table
    - spot_id (PK)
    - floor
    - spot_number
    - spot_size (Enum: motorcycle, car)
    - is_available (Boolean)
4. ParkingTransactions Table
    - transaction_id (PK)
    - vehicle_id (FK)
    - spot_id (FK)
    - check_in_time (DateTime)
    - check_out_time (DateTime)
    - fee (Float)

Algorithm for Spot Allocation

Spot Allocation Algorithm

Check-In Service

Check-Out Service

3. Fee Calculation Logic

Fee Calculation Algorithm

4. Concurrency Handling

Optimistic Locking

- Use version numbers in database rows to ensure data integrity.
- Example in SQLAlchemy:
-

5. Service Interactions and Workflow : Check-In Workflow

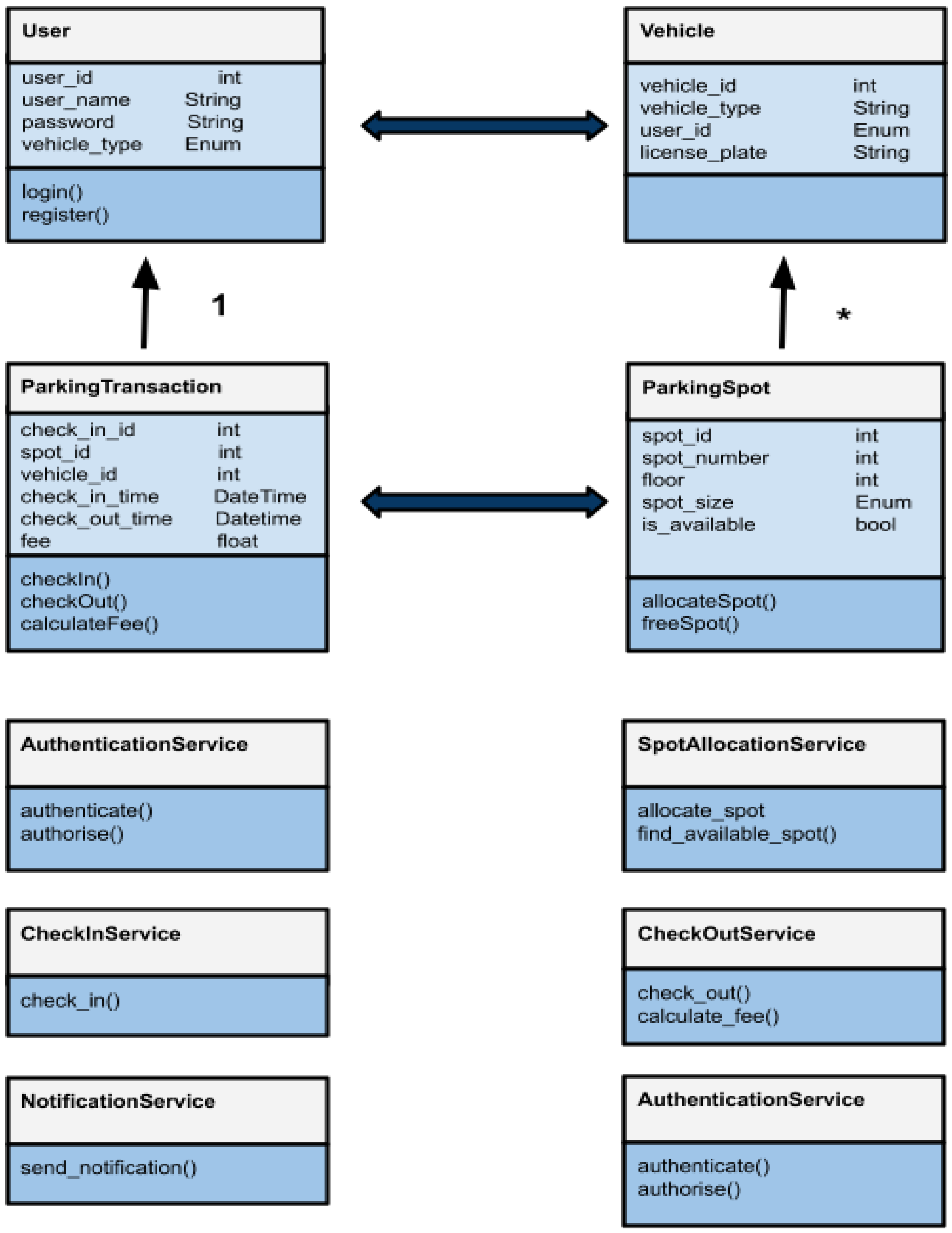
1. User sends a check-in request via mobile app.
2. API Gateway forwards to Check-In Service.
3. Check-In Service calls Spot Allocation Service.
4. Spot Allocation Service updates Parking Spot Database.
5. Check-In Service creates a Parking Transaction and updates ParkingTransactions Database.
6. Notification Service sends check-in confirmation to User.

Check-Out Workflow

1. User sends a check-out request via mobile app.
2. API Gateway forwards to Check-Out Service.
3. Check-Out Service updates ParkingTransactions Database and calculates fee.
4. Check-Out Service updates Parking Spot Database to mark the spot as available.
5. Notification Service sends check-out confirmation and fee details to User.

p.t.o

6. Class Diagram



Class Descriptions

User Class

- Attributes: user_id, username, password, vehicle_type
- Methods: login(), register()

Vehicle Class

- Attributes: vehicle_id, user_id, vehicle_type, license_plate

ParkingSpot Class

- Attributes: spot_id, floor, spot_number, spot_size, is_available
- Methods: allocateSpot(), freeSpot()

ParkingTransaction Class

- Attributes: transaction_id, vehicle_id, spot_id, check_in_time, check_out_time, fee
- Methods: checkIn(), checkOut(), calculateFee()

AuthenticationService Class

- Methods: authenticate(), authorize()

SpotAllocationService Class

- Methods: allocateSpot(), findAvailableSpot()

CheckInService Class

- Methods: checkIn()

CheckOutService Class

- Methods: checkOut(), calculateFee()

NotificationService Class

- Methods: sendNotification()

7. Abstract Classes and Their Subclasses

Abstract Class: ParkingSpot

The ParkingSpot abstract class defines the common attributes and methods for different types of parking spots.

ParkingSpot (Abstract Class)

- **Attributes:**
  - spot_id: Unique identifier for the parking spot.
  - floor: The floor on which the parking spot is located.
  - spot_number: The number of the parking spot.
  - spot_size: The size of the parking spot (e.g., motorcycle, car).
  - is_available: Boolean indicating whether the spot is available.
- **Methods:**
  - allocateSpot(): Marks the spot as allocated.
  - freeSpot(): Marks the spot as free.

MotorcycleSpot (Subclass)

- **Attributes:**
  - Inherits all attributes from ParkingSpot.
- **Methods:**
  - Inherits allocateSpot() and freeSpot() from ParkingSpot.

CarSpot (Subclass)

- **Attributes:**
  - Inherits all attributes from ParkingSpot.
- **Methods:**
  - Inherits allocateSpot() and freeSpot() from ParkingSpot.

Abstract Class: ParkingService

The ParkingService abstract class defines the common methods for services related to parking operations.

ParkingService (Abstract Class)

- **Methods:**
  - performAction(): A placeholder for actions performed by parking services.

CheckInService (Subclass)

- **Methods:**
  - checkIn(vehicle: Vehicle): Handles the check-in process for a vehicle.
  - performAction(): Overrides performAction() to define specific action for check-in.

CheckOutService (Subclass)

- **Methods:**
  - checkOut(vehicle: Vehicle): Handles the check-out process for a vehicle.
  - calculateFee(vehicle: Vehicle): Calculates the parking fee for the vehicle based on duration and type.
  - performAction(): Overrides performAction() to define specific action for check-out.

SpotAllocationService (Subclass)

- **Methods:**
  - allocateSpot(vehicle: Vehicle): Allocates a parking spot to the vehicle based on its type and availability.
  - findAvailableSpot(vehicle: Vehicle): Finds an available spot for the vehicle.
  - performAction(): Overrides performAction() to define specific action for spot allocation.

Abstract Class: Vehicle

The Vehicle abstract class defines the common attributes and methods that are shared across different types of vehicles.

Vehicle (Abstract Class)

- **Attributes:**
  - vehicle_id: Unique identifier for the vehicle.
  - user_id: Identifier for the user who owns the vehicle.
  - vehicle_type: The type of the vehicle (e.g., motorcycle, car).
  - license_plate: The license plate number of the vehicle.
- **Methods:**
  - getSize(): Returns the size of the vehicle.
  - getDetails(): Returns the details of the vehicle.

Motorcycle (Subclass)

- **Attributes:**
  - Inherits all attributes from Vehicle.
- **Methods:**
  - Overrides getSize(): Returns "motorcycle".
  - Inherits getDetails() from Vehicle.

Car (Subclass)

- **Attributes:**
  - Inherits all attributes from Vehicle.
- **Methods:**
  - Overrides getSize(): Returns "car".
  - Inherits getDetails() from Vehicle.

Class Descriptions and Responsibilities

User Class

- **Attributes:**
  - user_id: Unique identifier for the user.
  - username: Username of the user.
  - password: Password for the user account.
  - vehicle_type: Type of vehicle owned by the user.
- **Methods:**
  - login(): Authenticates the user.
  - register(): Registers a new user.

Vehicle Class

- **Attributes:**
  - vehicle_id: Unique identifier for the vehicle.
  - user_id: Identifier for the user who owns the vehicle.
  - vehicle_type: The type of the vehicle.
  - license_plate: The license plate number of the vehicle.

ParkingSpot Class

- **Attributes:**
  - spot_id: Unique identifier for the parking spot.
  - floor: The floor on which the parking spot is located.
  - spot_number: The number of the parking spot.
  - spot_size: The size of the parking spot.
  - is_available: Boolean indicating whether the spot is available.
- **Methods:**
  - allocateSpot(): Marks the spot as allocated.
  - freeSpot(): Marks the spot as free.

ParkingTransaction Class

- **Attributes:**
  - transaction_id: Unique identifier for the parking transaction.

    - vehicle_id: Identifier for the vehicle involved in the transaction.
    - spot_id: Identifier for the parking spot involved in the transaction.
    - check_in_time: The time the vehicle checked in.
    - check_out_time: The time the vehicle checked out.
    - fee: The parking fee calculated for the transaction.
- **Methods:**
    - checkIn(): Records the check-in time for the vehicle.
    - checkOut(): Records the check-out time for the vehicle and calculates the fee.
    - calculateFee(): Calculates the fee based on the duration and vehicle type.

AuthenticationService Class

- **Methods:**
    - authenticate(): Verifies the user's credentials.
    - authorize(): Authorizes the user for specific actions.

SpotAllocationService Class

- **Methods:**
    - allocateSpot(): Allocates a parking spot to the vehicle.
    - findAvailableSpot(): Finds an available spot for the vehicle.

CheckInService Class

- **Methods:**
    - checkIn(): Handles the check-in process for the vehicle.

CheckOutService Class

- **Methods:**
    - checkOut(): Handles the check-out process for the vehicle.
    - calculateFee(): Calculates the parking fee for the vehicle.

NotificationService Class

- **Methods:**

- sendNotification(): Sends notifications to the user (e.g., check-in confirmation, check-out confirmation, fee details).

This design encapsulates the core functionality of the smart parking lot system with clear separation of concerns and responsibilities among the different components.

Abstract Classes in the Design

1. Vehicle
    - An abstract class Vehicle can define common attributes and methods that are shared across different types of vehicles (e.g., Motorcycle, Car).
1. ParkingSpot
    - An abstract class ParkingSpot can define common attributes and methods for different types of parking spots (e.g., MotorcycleSpot, CarSpot).
1. ParkingService
    - An abstract class ParkingService can define common methods for services related to parking operations (e.g., CheckInService, CheckOutService).