# Design a Smart Parking Lot System

## 1. Data Model

### Database Schema

- **ParkingSpots**: Contains information about each parking spot, including spot ID, size (e.g., motorcycle, car, bus), floor level, and availability status.
- **Vehicles**: Stores details of vehicles, like vehicle ID, size, and owner information.
- **ParkingTransactions**: Records each parking transaction with details such as transaction ID, vehicle ID, spot ID, entry time, exit time, and parking fee.

1. **Parking Spot**:
   - `spotID` (Primary Key)
   - `floorNumber`
   - `spotSize` (e.g., small, medium, large)
   - `isOccupied` (Boolean)
2. **Vehicle**:
   - `vehicleID` (Primary Key)
   - `vehicleType` (e.g., motorcycle, car, bus)
   - `ownerID`
3. **Parking Session** (Optional, depending on the need for tracking active sessions):
   - `sessionID` (Primary Key)
   - `vehicleID` (Foreign Key, references Vehicle)
   - `spotID` (Foreign Key, references Parking Spot)
   - `entryTime`
4. **ParkingTransactions**:

- `transactionID` (Primary Key)
- `vehicleID` (Foreign Key, references Vehicle)
- `spotID` (Foreign Key, references Parking Spot)
- `entryTime`
- `exitTime`
- `parkingFee`

5. **Fee Structure**:
   - `vehicleType`
   - `duration`
   - `rate`

## Relationships

- Each **ParkingTransaction** is associated with one **Vehicle** and one **ParkingSpot**.
- **ParkingSpots** are updated in real-time to reflect availability.
- **During Check-In**:
  - Create a new record in the `ParkingTransactions` table when a vehicle enters, recording the `VehicleID`, `SpotID`, and `EntryTime`.
- **During Check-Out**:
  - Update the relevant `ParkingTransactions` record with `ExitTime` and `ParkingFee` when the vehicle exits.

## 2. Algorithm for Spot Allocation

## Steps

1. **Vehicle Entry**: Detect vehicle size upon entry.
2. **Spot Search**: Search for an available spot that matches the vehicle size.
3. **Spot Assignment**: Assign the nearest available spot to the vehicle.
4. **Update Database**: Mark the spot as occupied in the database.

## Optimization

- Use a priority queue for quick retrieval of nearest available spots.
- Cache frequently accessed data for faster performance.

## 3. Fee Calculation Logic

## Components

- **Base Rate**: Defined for each vehicle type.
- **Time-Based Rate**: Additional charges based on the duration of stay.
- **Special Rates**: Discounts or surcharges for peak hours or special events.

## Calculation

- Calculate the total time of stay from entry and exit timestamps.
- Apply the base rate and time-based rate according to vehicle type and parking duration.

## 4. Concurrency Handling

## Strategies

- **Locking Mechanisms**: Implement locks to prevent simultaneous access to the same parking spot record.
- **Transaction Management**: Use database transactions to ensure data integrity during simultaneous check-ins and check-outs.
- **Real-Time Updates**: Employ a publish-subscribe model or websockets for real-time updates of parking spot availability.

## Scalability

- Design the system to be horizontally scalable to handle increased load.
- Use load balancers to distribute requests evenly across servers.

## 5. Reporting and Analytics

- The `ParkingTransactions` table serves as a rich data source for generating reports and analytics, such as:
  - Daily or monthly revenue from parking fees.
  - Average parking duration.
  - Usage patterns of parking spots.

# Component Design with Design Patterns

## Entry/Exit Management Module

- **Design Pattern**: Observer Pattern for sensor event handling.
- **Implementation**: Listeners for vehicle entry and exit events.
- **Data Structure**: Queue for managing entry and exit requests to handle peak time traffic.

## Spot Allocation Engine

- **Design Pattern**: Strategy Pattern for different allocation strategies based on vehicle type.
- **Implementation**: Algorithms for nearest available spot allocation.
- **Data Structure**: Min-heap or balanced tree for efficient spot lookup.

## Fee Calculation Module

- **Design Pattern**: Factory Method for different fee calculation strategies.
- **Implementation**: Classes for each vehicle type fee calculation.
- **Data Structure**: Hash table for storing base rates and time-based rates.

## Database

- **Design Pattern**: DAO (Data Access Object) for data abstraction and encapsulation.
- **Implementation**: Separate DAOs for ParkingSpot, Vehicle, and ParkingTransaction.

- **Data Structure**: Relational tables with appropriate indexing for performance.

## User Interface

- **Design Pattern**: MVC (Model-View-Controller) for separation of concerns.
- **Implementation**: Views for parking spot availability, transaction details; Controllers for handling user requests.

## Security Module

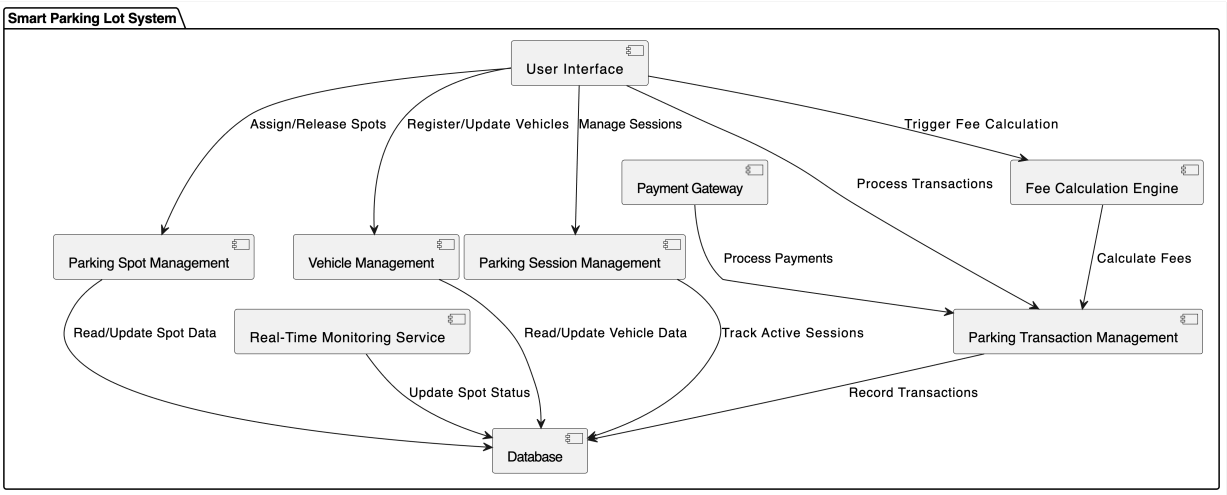
- **Design Pattern**: Singleton for global security configurations.
- **Implementation**: Encryption and decryption services, authentication, and authorization checks.

## API Gateway

- **Design Pattern**: API Gateway Pattern for request routing, composition, and protocol translation.
- **Implementation**: Routing requests to appropriate components, load balancing.

## Connections:

- Sensors at entry/exit points connect to the Entry/Exit Management Module.
- Spot Allocation Engine communicates with the Database to update spot status.
- Fee Calculation Module interacts with the Database for transaction data and with payment systems for processing fees.
- User Interface retrieves and displays data from the Database and interacts with other modules for user actions.
- All components connect to the Security Module for secure operations.
- API Gateway acts as an intermediary for all inter-component communications.


Smart Parking Lot System
User Interface
Assign/Release Spots
Register/Update Vehicles
Manage Sessions
Trigger Fee Calculation
Payment Gateway
Process Transactions
Fee Calculation Engine
Parking Spot Management
Vehicle Management
Parking Session Management
Process Payments
Calculate Fees
Read/Update Spot Data
Real-Time Monitoring Service
Read/Update Vehicle Data
Track Active Sessions
Parking Transaction Management
Update Spot Status
Record Transactions
Database