**Smart Parking Lot System DESIGN**

Database Design:

1. Parking Spots Table:

spot\_id (Primary Key)

floor\_number

spot\_number

size (e.g., motorcycle, car, bus)

availability (boolean)

2. Vehicles Table:

vehicle\_id (Primary Key)

license\_plate

size (e.g., motorcycle, car, bus)

3. Parking Transactions Table:

transaction\_id (Primary Key)

vehicle\_id (Foreign Key)

entry\_time

exit\_time

fee

Algorithm for Spot Allocation:

When a vehicle enters:

Query the database for available spots based on the vehicle size and floor availability.

Use an algorithm (e.g., round-robin, nearest spot) to assign a spot to the vehicle.

Update the spot availability in the database.

Fee Calculation Logic:

On vehicle exit:

Calculate the duration of stay (exit\_time - entry\_time).

Determine the fee based on the duration and the type of vehicle.

Update the transaction record in the database with the calculated fee.

Concurrency Handling:

Use database transactions to ensure atomicity.

Implement optimistic concurrency control to handle concurrent updates.

Use a distributed database for scalability and availability.

API Design:

1. Vehicle Entry API:

Endpoint: /entry

Method: POST

Request Body: { "license\_plate": "XYZ123", "size": "car" }

Response: { "spot\_id": 123, "entry\_time": "2024-01-04T12:00:00" }

2. Vehicle Exit API:

Endpoint: /exit/{vehicle\_id}

Method: POST

Response: { "exit\_time": "2024-01-04T14:30:00", "fee": 5.00 }

3. Spot Availability API:

Endpoint: /availability

Method: GET

Response: { "available\_spots": { "car": 20, "motorcycle": 15, "bus": 5 } }

4. Real-time Update API:

Use websockets or a publish-subscribe pattern to broadcast updates on spot availability.

High Availability and Scalability:

Use a distributed architecture with load balancing for API servers.

Deploy multiple instances of the system across different availability zones.

Utilize a cloud-based database with auto-scaling capabilities.

Implement caching for frequently accessed data.

Ensure fault tolerance by replicating critical components.

By considering these aspects, the designed smart parking lot system should be capable of efficiently managing parking processes, handling concurrent operations, and providing high availability and scalability.