Smart City Parking Management System

Name	ID
Bahaa Khaled Mohamed	21010383
Pola Hany Fayez	21010387
Ranime Ahmed Elsayed Shehata	21010531
Abdullah Mohamed Ali	21011644
Nayra Ibrahim Ahmed	21011504

Table of Contents:

1. Introduction:

Overview:

- The **Smart City Parking Management System** is an innovative solution designed to address parking challenges in urban environments. By leveraging cutting-edge technologies such as IoT, real-time data analytics, and dynamic pricing models, the system streamlines parking operations, improves user convenience, and enhances overall traffic management. The system integrates parking lots, meters, and drivers through a centralized database, providing a seamless and efficient way to monitor and manage parking availability, reservations, and pricing.
- This system is tailored for smart city applications, where each parking spot is equipped with simulated IoT capabilities to provide live updates to users. It not only optimizes the use of available parking spaces but also minimizes congestion and improves the urban driving experience.

Goals:

1. **Provide real-time parking spot availability:** provide live updates on parking spot availability across multiple locations, ensuring users have accurate and timely information.
2. **Implement dynamic pricing:** Implement a flexible pricing model that adjusts based on factors such as demand, time of day, and location, encouraging optimal usage of parking resources.
3. **Reservation System**: Allow drivers to search for, reserve, and pay for parking spots in advance.
4. **Role-Based Access**:Support different user roles, including drivers, parking lot administrators, and system administrators, with customized access and permissions.
5. **Web Integration**: Offer a user-friendly web interface where users can search for spots, make reservations, and navigate seamlessly to their selected parking lots.
6. **Enhanced Traffic Flow**: Minimize traffic caused by drivers searching for parking by providing clear, real-time directions to available spots.

7. **Scalability and Efficiency**: Ensure the system is capable of handling high traffic volumes and multiple simultaneous transactions without compromising performance.

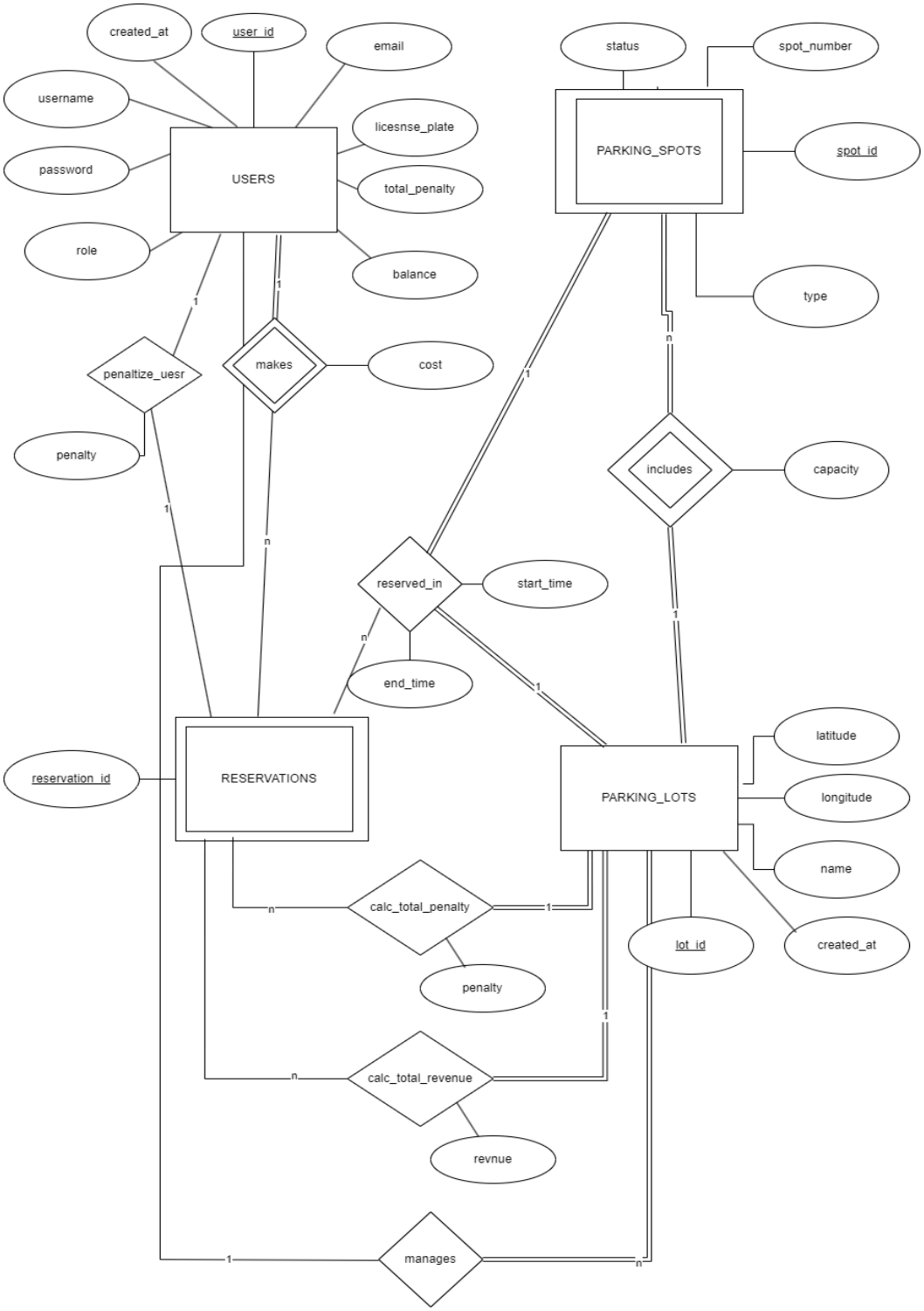
Technology Stack:

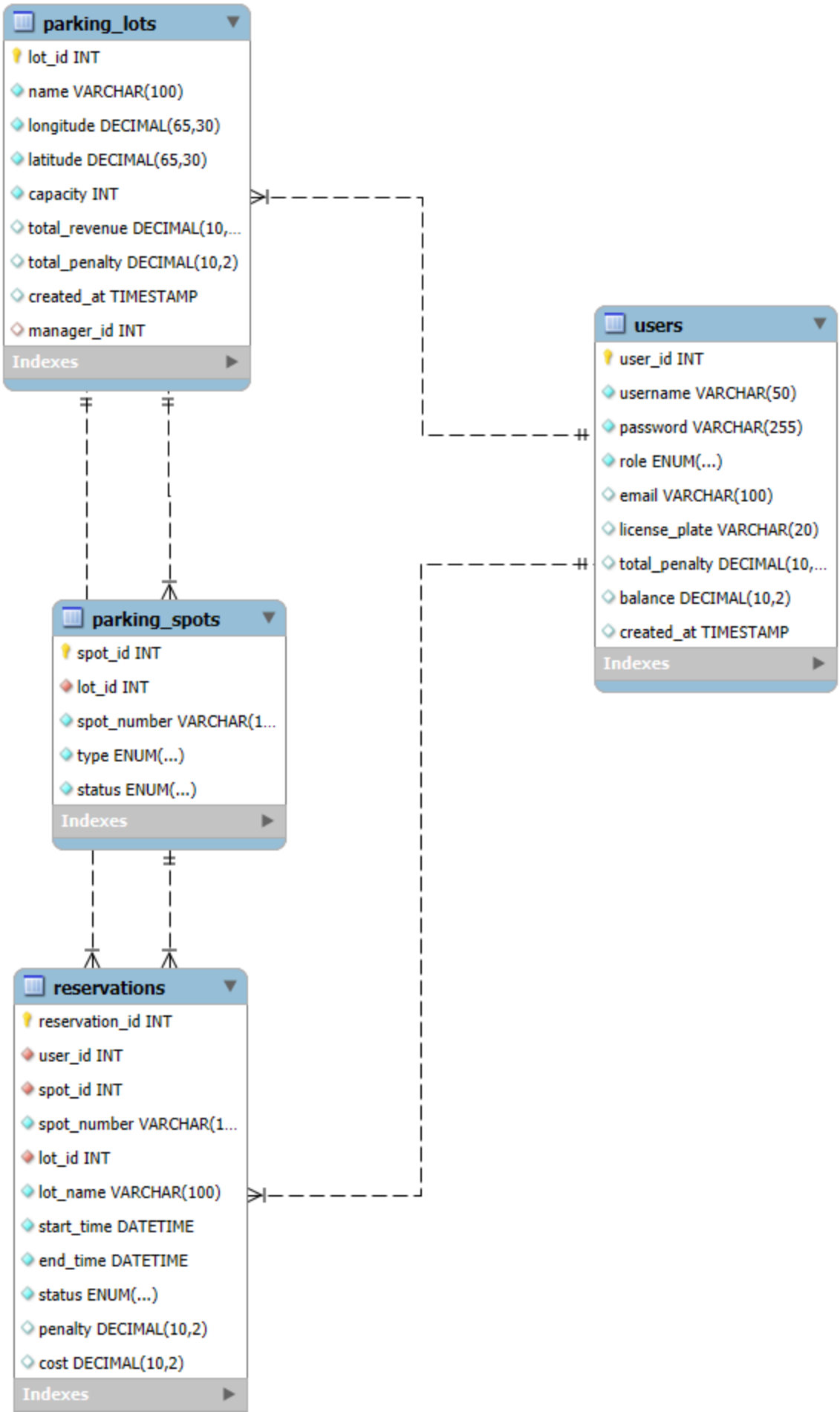
- Backend: Spring Boot.
- Frontend: React.
- Database: MySQL.
- Reporting: JasperReports.
- Simulation: Javascript for IoT Simulation.
- Data Generation: Python.

---

2. Database Design:

ERD Diagram:




parking_lots
lot_id INT
name VARCHAR(100)
longitude DECIMAL(65,30)
latitude DECIMAL(65,30)
capacity INT
total_revenue DECIMAL(10,...
total_penalty DECIMAL(10,2)
created_at TIMESTAMP
manager_id INT
Indexes
users
user_id INT
username VARCHAR(50)
password VARCHAR(255)
role ENUM(...)
email VARCHAR(100)
license_plate VARCHAR(20)
total_penalty DECIMAL(10,...
balance DECIMAL(10,2)
created_at TIMESTAMP
Indexes
parking_spots
spot_id INT
lot_id INT
spot_number VARCHAR(1...
type ENUM(...)
status ENUM(...)
Indexes
reservations
reservation_id INT
user_id INT
spot_id INT
spot_number VARCHAR(1...
lot_id INT
lot_name VARCHAR(100)
start_time DATETIME
end_time DATETIME
status ENUM(...)
penalty DECIMAL(10,2)
cost DECIMAL(10,2)
Indexes

Database Schema:

```sql
CREATE DATABASE smart_city_parking;
SHOW DATABASES;
USE smart_city_parking;

CREATE TABLE users (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    role ENUM('DRIVER', 'MANAGER', 'ADMIN') NOT NULL,
    email VARCHAR(100) UNIQUE,
    license_plate VARCHAR(20), -- Only for drivers
total_penalty DECIMAL(10, 2) DEFAULT 0.00, -- For no-show or overstay
    balance DECIMAL(10, 2) DEFAULT 0.00,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE parking_lots (
    lot_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    longitude DECIMAL(65, 30) NOT NULL,
    latitude DECIMAL(65, 30) NOT NULL,
    capacity INT NOT NULL, -- Total number of spots
total_revenue DECIMAL(10, 2) DEFAULT 0.00,
total_penalty DECIMAL(10, 2) DEFAULT 0.00,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ,
    manager_id INT ,
    FOREIGN KEY (manager_id) REFERENCES users(user_id) ON DELETE SET NULL
);

CREATE TABLE parking_spots (
    spot_id INT AUTO_INCREMENT PRIMARY KEY,
    lot_id INT NOT NULL,
    spot_number VARCHAR(10) NOT NULL, -- Unique per lot
    type ENUM('REGULAR', 'DISABLED', 'EV') NOT NULL,
    status ENUM('AVAILABLE', 'OCCUPIED', 'RESERVED') NOT NULL DEFAULT 'AVAILABLE',
    FOREIGN KEY (lot_id) REFERENCES parking_lots(lot_id) ON DELETE CASCADE
);
```

```sql
CREATE TABLE reservations (
    reservation_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    spot_id INT NOT NULL,
    spot_number VARCHAR(10) NOT NULL,
    lot_id INT NOT NULL,
    lot_name VARCHAR(100) NOT NULL,
    start_time DATETIME NOT NULL,
    end_time DATETIME NOT NULL,
    status ENUM('ACTIVE', 'COMPLETED', 'CANCELLED' , 'NO_SHOW') NOT NULL DEFAULT 'ACTIVE',
    penalty DECIMAL(10, 2) DEFAULT 0.00, -- For no-show or overstay
cost DECIMAL(10, 2) DEFAULT 0.00,
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,
    FOREIGN KEY (spot_id) REFERENCES parking_spots(spot_id) ON DELETE CASCADE,
    FOREIGN KEY (lot_id) REFERENCES parking_lots(lot_id) ON DELETE CASCADE
);
```

Database Generation Script (SQL Files):

```python
import pymysql

# Database connection details
db_config = {
    "host": "localhost",
    "user": "root",
    "password": "smart_city_parking",
    "database": "smart_city_parking"
}

# Coordinates and other details
coordinates = [
    (31.258352933146817, 29.981002714990263),
    (31.20836298155645, 29.923393916509106),
    (31.245725767240582, 29.965868537277913),
    (31.236611050301057, 29.951834406735134),
    (31.228463715093138, 29.942403546882044),
    (31.218009329994388, 29.92733759714959),
    (31.25602426146235, 29.98355999465894),
    (31.249734785675155, 29.991109772853328)
]


# Insert data into parking_lots table
def insert_parking_lots(cursor):
    for i, (latitude, longitude) in enumerate(coordinates):
        name = f'Parking Lot {i + 1}'
        capacity = 100  # Example capacity
        manager_id = i + 1  # Assign manager IDs sequentially from 1 to 8
        cursor.execute('''
            INSERT INTO parking_lots (name, longitude, latitude, capacity, total_revenue, total_penalty, manager_id)
            VALUES (%s, %s, %s, %s, 0, 0, %s)
        ''', (name, longitude, latitude, capacity, manager_id))
```

```python
# Insert data into users table
def insert_users(cursor):
    for i in range(1, 9):
        username = f'manager{i}'
        password = '$2a$10$6lHJU7vPaaknD4LRShR8IeYvSGxZy63SUsKLXmxYxRMo4P/mj3g6W'
        role = 'MANAGER'
        email = f'manager{i}@example.com'
        cursor.execute('''
            INSERT INTO users (username, password, role, email)
            VALUES (%s, %s, %s, %s)
        ''', (username, password, role, email))


# Main function to execute the insertion
def insert_data():
    try:
        conn = pymysql.connect(**db_config)
        cursor = conn.cursor()

        print("Inserting data into users table...")
        insert_users(cursor)

        print("Inserting data into parking_lots table...")
        insert_parking_lots(cursor)

        conn.commit()

        print("Entries added successfully!")

    except pymysql.MySQLError as err:
        print(f"Error: {err}")

    finally:
        # Close the cursor and connection
        if 'cursor' in locals():
            cursor.close()
        if 'conn' in locals():
            conn.close()

```

```python
import pymysql
import random

# Database connection details
db_config = {
    "host": "localhost",
    "user": "root",
    "password": "smart_city_parking",
    "database": "smart_city_parking"
}

# Spot types and statuses
spot_types = ['REGULAR', 'DISABLED', 'EV']
spot_statuses = ['AVAILABLE', 'OCCUPIED', 'RESERVED']

# Insert parking spots into the parking_spots table
def insert_parking_spots(cursor, lot_id, num_spots):
    for spot_num in range(1, num_spots + 1):
        spot_type = random.choice(spot_types)
        spot_status = random.choice(spot_statuses) if random.random() < 0.3 else 'AVAILABLE'  # 30% chance for non-available
        spot_number = f"S{spot_num:03d}"  # Format as S001, S002, etc.
        cursor.execute('''
            INSERT INTO parking_spots (lot_id, spot_number, type, status)
            VALUES (%s, %s, %s, %s)
        ''', (lot_id, spot_number, spot_type, spot_status))

```

```python
# Main function to execute the insertion
def insert_data():
    try:
        conn = pymysql.connect(**db_config)
        cursor = conn.cursor()

        # Retrieve lot IDs from the parking_lots table
        cursor.execute("SELECT lot_id FROM parking_lots")
        lot_ids = cursor.fetchall()

        print("Inserting parking spots for each lot...")
        for lot_id_tuple in lot_ids:
            lot_id = lot_id_tuple[0]
            num_spots = 50  # Example: 50 spots per lot
            insert_parking_spots(cursor, lot_id, num_spots)

        conn.commit()
        print("Parking spots added successfully!")

    except pymysql.MySQLError as err:
        print(f"Error: {err}")

    finally:
        # Close the cursor and connection
        if 'cursor' in locals():
            cursor.close()
        if 'conn' in locals():
            conn.close()

# Run the function
if __name__ == "__main__":
    insert_data()

```

SQL Queries samples:

```
String sql = "SELECT parking_lots.lot_id , parking_lots.longitude ,
parking_lots.latitude , " +
"parking_lots.total_revenue , parking_lots.total_penalty ,
parking_lots.capacity ,t.occupied " +
"from (SELECT lot_id, COUNT(*) as occupied FROM parking_spots where status =
'OCCUPIED' " +
"GROUP BY lot_id) t INNER JOIN parking_lots ON t.lot_id = parking_lots.lot_id
AND parking_lots.manager_id = ? ";
```

```
   String sql = """
   SELECT * FROM reservations
   WHERE start_time < ? AND status = 'ACTIVE'
""";
```

```
   String sql = """
   SELECT * FROM reservations
   WHERE spot_id IN (SELECT spot_id FROM parking_spots WHERE status = ?)
""";
```

```python
import pymysql

# Database connection details
db_config = {
    "host": "localhost",
    "user": "root",
    "password": "smart_city_parking",
    "database": "smart_city_parking"
}

# Coordinates and other details
coordinates = [
    (31.258352933146817, 29.981002714990263),
    (31.20836298155645, 29.923393916509106),
    (31.245725767240582, 29.965868537277913),
    (31.236611050301057, 29.951834406735134),
    (31.228463715093138, 29.942403546882044),
    (31.218009329994388, 29.92733759714959),
    (31.25602426146235, 29.98355999465894),
    (31.249734785675155, 29.991109772853328)
]


# Insert data into parking_lots table
def insert_parking_lots(cursor):
    for i, (latitude, longitude) in enumerate(coordinates):
        name = f'Parking Lot {i + 1}'
        capacity = 100  # Example capacity
        manager_id = i + 1  # Assign manager IDs sequentially from 1 to 8
        cursor.execute('''
            INSERT INTO parking_lots (name, longitude, latitude, capacity, total_revenue, total_penalty, manager_id)
            VALUES (%s, %s, %s, %s, 0, 0, %s)
        ''', (name, longitude, latitude, capacity, manager_id))
```

```python
# Insert data into users table
def insert_users(cursor):
    for i in range(1, 9):
        username = f'manager{i}'
        password = '$2a$10$6lHJU7vPaaknD4LRShR8IeYvSGxZy63SUsKLXmxYxRMo4P/mj3g6W'
        role = 'MANAGER'
        email = f'manager{i}@example.com'
        cursor.execute('''
            INSERT INTO users (username, password, role, email)
            VALUES (%s, %s, %s, %s)
        ''', (username, password, role, email))


# Main function to execute the insertion
def insert_data():
    try:
        conn = pymysql.connect(**db_config)
        cursor = conn.cursor()

        print("Inserting data into users table...")
        insert_users(cursor)

        print("Inserting data into parking_lots table...")
        insert_parking_lots(cursor)

        conn.commit()

        print("Entries added successfully!")

    except pymysql.MySQLError as err:
        print(f"Error: {err}")

    finally:
        # Close the cursor and connection
        if 'cursor' in locals():
            cursor.close()
        if 'conn' in locals():
            conn.close()

```

```python
import pymysql
import random

# Database connection details
db_config = {
    "host": "localhost",
    "user": "root",
    "password": "smart_city_parking",
    "database": "smart_city_parking"
}

# Spot types and statuses
spot_types = ['REGULAR', 'DISABLED', 'EV']
spot_statuses = ['AVAILABLE', 'OCCUPIED', 'RESERVED']

# Insert parking spots into the parking_spots table
def insert_parking_spots(cursor, lot_id, num_spots):
    for spot_num in range(1, num_spots + 1):
        spot_type = random.choice(spot_types)
        spot_status = random.choice(spot_statuses) if random.random() < 0.3 else 'AVAILABLE'  # 30% chance for non-available
        spot_number = f"S{spot_num:03d}"  # Format as S001, S002, etc.
        cursor.execute('''
            INSERT INTO parking_spots (lot_id, spot_number, type, status)
            VALUES (%s, %s, %s, %s)
        ''', (lot_id, spot_number, spot_type, spot_status))

```

```python
# Main function to execute the insertion
def insert_data():
    try:
        conn = pymysql.connect(**db_config)
        cursor = conn.cursor()

        # Retrieve lot IDs from the parking_lots table
        cursor.execute("SELECT lot_id FROM parking_lots")
        lot_ids = cursor.fetchall()

        print("Inserting parking spots for each lot...")
        for lot_id_tuple in lot_ids:
            lot_id = lot_id_tuple[0]
            num_spots = 50  # Example: 50 spots per lot
            insert_parking_spots(cursor, lot_id, num_spots)

        conn.commit()
        print("Parking spots added successfully!")

    except pymysql.MySQLError as err:
        print(f"Error: {err}")

    finally:
        # Close the cursor and connection
        if 'cursor' in locals():
            cursor.close()
        if 'conn' in locals():
            conn.close()

# Run the function
if __name__ == "__main__":
    insert_data()

```

```java
1 usage  Bahaa Khaled
@Override
public int createUser(User user) {
    String sql = "INSERT INTO users (username, password, role, email, license_plate, total_penalty, balance) VALUES (?, ?, ?, ?, ?, ?, ?)";
    KeyHolder keyHolder = new GeneratedKeyHolder();

    jdbcTemplate.update(connection -> {
        PreparedStatement ps = connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
        ps.setString( parameterIndex: 1, user.getUsername());
        ps.setString( parameterIndex: 2, user.getPassword());
        ps.setString( parameterIndex: 3, user.getRole());
        ps.setString( parameterIndex: 4, user.getEmail());
        ps.setString( parameterIndex: 5, user.getLicensePlate());
        ps.setBigDecimal( parameterIndex: 6, user.getTotalPenalty());
        ps.setBigDecimal( parameterIndex: 7, user.getBalance());
        return ps;
    }, keyHolder);

    return keyHolder.getKey() != null ? keyHolder.getKey().intValue() : -1;
}
```

```java
@Override
public Optional<User> getUserById(int userId) {
    String sql = "SELECT * FROM users WHERE user_id = ?";
    return jdbcTemplate.query(sql, userRowMapper, userId).stream().findFirst();
}

1 usage  Bahaa Khaled
@Override
public Optional<User> getUserByUsername(String username) {
    String sql = "SELECT * FROM users WHERE username = ?";
    return jdbcTemplate.query(sql, userRowMapper, username).stream().findFirst();
}

4 usages  Bahaa Khaled
@Override
public Optional<User> getUserByEmail(String email) {
    String sql = "SELECT * FROM users WHERE email = ?";
    return jdbcTemplate.query(sql, userRowMapper, email).stream().findFirst();
}

1 usage  Bahaa Khaled
@Override
public List<User> getAllUsers() {
    String sql = "SELECT * FROM users";
    return jdbcTemplate.query(sql, userRowMapper);
}

5 usages  Bahaa Khaled
@Override
public boolean updateUser(User user) {
    String sql = "UPDATE users SET username = ?, password = ?, role = ?, email = ?, license_plate = ?, total_penalty = ?, balance = ? WHERE user_id = ?";
    return jdbcTemplate.update(sql, user.getUsername(), user.getPassword(), user.getRole(),
            user.getEmail(), user.getLicensePlate(), user.getTotalPenalty(), user.getBalance(), user.getUserId()) > 0;
}
```

```
no usages  Bahaa Khaled
@Override
public boolean deleteUser(int userId) {
    String sql = "DELETE FROM users WHERE user_id = ?";
    return jdbcTemplate.update(sql, userId) > 0;
}

1 usage  Bahaa Khaled
@Override
public boolean updateUserRole(int userId, String newRole) {
    String sql = "UPDATE users SET role = ? WHERE user_id = ?";
    return jdbcTemplate.update(sql, newRole, userId) > 0;
}
```

```
1 usage  Bahaa Khaled +1 *
@Override
@Transactional
public int createReservation(Reservation reservation) {

    String reservationSql = "INSERT INTO reservations (user_id, spot_id,spot_number, lot_id, lot_name, start_time, end_time, status, penalty, cost) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
    KeyHolder keyHolder = new GeneratedKeyHolder();

    jdbcTemplate.update(connection -> {
        PreparedStatement ps = connection.prepareStatement(reservationSql, Statement.RETURN_GENERATED_KEYS);
        ps.setInt( parameterIndex: 1, reservation.getUserId());
        ps.setInt( parameterIndex: 2, reservation.getSpotId());
        ps.setString( parameterIndex: 3, reservation.getSpotNumber());
        ps.setInt( parameterIndex: 4, reservation.getLotId());
        ps.setString( parameterIndex: 5, reservation.getLotName());
        ps.setTimestamp( parameterIndex: 6, java.sql.Timestamp.valueOf(reservation.getStartTime()));
        ps.setTimestamp( parameterIndex: 7, java.sql.Timestamp.valueOf(reservation.getEndTime()));
        ps.setString( parameterIndex: 8, reservation.getStatus());
        ps.setBigDecimal( parameterIndex: 9, reservation.getPenalty());
        ps.setBigDecimal( parameterIndex: 10, reservation.getCost());
        return ps;
    }, keyHolder);

    return keyHolder.getKey() != null ? keyHolder.getKey().intValue() : -1;
}
```

```java
4 usages  Bahaa Khaled
@Override
public Optional<Reservation> getReservationById(int reservationId) {
    String sql = "SELECT * FROM reservations WHERE reservation_id = ?";
    return jdbcTemplate.query(sql, reservationRowMapper, reservationId).stream().findFirst();
}

1 usage  Bahaa Khaled
@Override
public List<Reservation> getReservationsByUserId(int userId) {
    String sql = "SELECT * FROM reservations WHERE user_id = ?";
    return jdbcTemplate.query(sql, reservationRowMapper, userId);
}

no usages  Bahaa Khaled
@Override
public List<Reservation> getReservationsBySpotId(int spotId) {
    String sql = "SELECT * FROM reservations WHERE spot_id = ?";
    return jdbcTemplate.query(sql, reservationRowMapper, spotId);
}

6 usages  Bahaa Khaled +1
@Override
@Transactional
public boolean updateReservation(Reservation reservation) {
    String selectSql = "SELECT reservation_id FROM reservations WHERE reservation_id = ? FOR UPDATE";
    jdbcTemplate.queryForObject(selectSql, Integer.class, reservation.getReservationId());

    String updateSql = "UPDATE reservations SET start_time = ?, end_time = ?, status = ?, penalty = ?, cost = ? WHERE reservation_id = ?";
    return jdbcTemplate.update(updateSql,
            java.sql.Timestamp.valueOf(reservation.getStartTime()),
            java.sql.Timestamp.valueOf(reservation.getEndTime()),
            reservation.getStatus(),
            reservation.getPenalty(),
            reservation.getCost(),
            reservation.getReservationId()) > 0;
}
```

```java
    @Override
    @Transactional
    public boolean deleteReservation(int reservationId) {
        // Retrieve the reservation to get the spot ID
        Optional<Reservation> reservationOpt = getReservationById(reservationId);
        if (reservationOpt.isEmpty()) {
            throw new IllegalStateException("Reservation not found.");
        }

        // Free up the parking spot
        String updateSpotSql = "UPDATE parking_spots SET status = 'AVAILABLE' WHERE spot_id = ?";
        jdbcTemplate.update(updateSpotSql, reservationOpt.get().getSpotId());

        // Delete the reservation
        String deleteReservationSql = "DELETE FROM reservations WHERE reservation_id = ?";
        return jdbcTemplate.update(deleteReservationSql, reservationId) > 0;
    }

    1 usage  Bahaa Khaled
    @Override
    @Transactional
    public boolean isSpotAvailableForDuration(int spotId, LocalDateTime startTime, LocalDateTime endTime) {
        String sql = "SELECT COUNT(*) FROM reservations " +
                "WHERE spot_id = ? AND " +
                "(start_time < ? AND end_time > ?) AND " +
                "status  = 'ACTIVE'";
        Integer count = jdbcTemplate.queryForObject(sql, Integer.class, spotId, endTime, startTime);
        return count == 0;
    }


    1 usage  Bahaa Khaled
    @Override
    public List<Reservation> getExpiredReservations(LocalDateTime currentTime) {
        String sql = """
        SELECT * FROM reservations
        WHERE end_time < ? AND status = 'ACTIVE'
    """;
        return jdbcTemplate.query(sql, reservationRowMapper, currentTime);
    }
```

```java
    @Override
    public List<Reservation> getReservationsByStatus(String status) {
        String sql = """
        SELECT * FROM reservations
        WHERE status = ?
    """;
        return jdbcTemplate.query(sql, reservationRowMapper, status);
    }

    1 usage  Bahaa Khaled
    @Override
    public List<Reservation> getReservationsBySpotStatus(String status) {
        String sql = """
        SELECT * FROM reservations
        WHERE spot_id IN (SELECT spot_id FROM parking_spots WHERE status = ?)
    """;
        return jdbcTemplate.query(sql, reservationRowMapper, status);
    }

    1 usage  Bahaa Khaled
    @Override
    public List<Reservation> getStartedReservations(LocalDateTime currentTime) {
        String sql = """
        SELECT * FROM reservations
        WHERE start_time < ? AND status = 'ACTIVE'
    """;
        return jdbcTemplate.query(sql, reservationRowMapper, currentTime);
    }

    no usages  Bahaa Khaled
    @Override
    public List<Reservation> getReservationsByLotId(int lotId) {
        String sql = """
        SELECT * FROM reservations
        WHERE lot_id = ?
    """;
        return jdbcTemplate.query(sql, reservationRowMapper, lotId);
    }
}
```

```java
@Override
public int createParkingSpot(ParkingSpot parkingSpot) {
    String sql = "INSERT INTO parking_spots (lot_id, spot_number, type, status) VALUES (?, ?, ?, ?)";
    return jdbcTemplate.update(sql, parkingSpot.getLotId(), parkingSpot.getSpotNumber(),
            parkingSpot.getType(), parkingSpot.getStatus());
}

8 usages  Bahaa Khaled
@Override
public Optional<ParkingSpot> getParkingSpotById(int spotId) {
    String sql = "SELECT * FROM parking_spots WHERE spot_id = ?";
    return jdbcTemplate.query(sql, parkingSpotRowMapper, spotId).stream().findFirst();
}

2 usages  Bahaa Khaled
@Override
public List<ParkingSpot> getParkingSpotsByLotId(int lotId) {
    String sql = "SELECT * FROM parking_spots WHERE lot_id = ?";
    return jdbcTemplate.query(sql, parkingSpotRowMapper, lotId);
}


8 usages  Abdullh911 +1
@Override
@Transactional
public boolean updateParkingSpot(ParkingSpot parkingSpot) {
    // First, lock the row for update
    String selectSql = "SELECT spot_id FROM parking_spots WHERE spot_id = ? FOR UPDATE";
    jdbcTemplate.queryForObject(selectSql, Integer.class, parkingSpot.getSpotId());

    // Now, update the parking spot
    String updateSql = "UPDATE parking_spots SET spot_number = ?, type = ?, status = ? WHERE spot_id = ?";
    return jdbcTemplate.update(updateSql, parkingSpot.getSpotNumber(), parkingSpot.getType(),
            parkingSpot.getStatus(), parkingSpot.getSpotId()) > 0;
}
```

```java
no usages  Bahaa Khaled
@Override
@Transactional
public boolean deleteParkingSpot(int spotId) {
    String sql = "DELETE FROM parking_spots WHERE spot_id = ?";
    return jdbcTemplate.update(sql, spotId) > 0;
}


no usages  Bahaa Khaled
@Override
public List<ParkingSpot> findAvailableSpotsByLocation(String location) {
    String sql = """
    SELECT ps.*
    FROM parking_spots ps
    JOIN parking_lots pl ON ps.lot_id = pl.lot_id
    WHERE pl.location = ? AND ps.status = 'AVAILABLE'
""";
    return jdbcTemplate.query(sql, parkingSpotRowMapper, location);
}

no usages  Bahaa Khaled
@Override
public int getLotIdBySpotId(int spotId) {
    String sql = "SELECT lot_id FROM parking_spots WHERE spot_id = ?";
    return jdbcTemplate.queryForObject(sql, Integer.class, spotId);
}
}
```

```java
@Override
public int createParkingLot(ParkingLot parkingLot) {
    String sql = "INSERT INTO parking_lots (name, longitude, latitude, capacity, total_revenue, total_penalty) VALUES (?, ?, ?, ?, ?, ?)";
    KeyHolder keyHolder = new GeneratedKeyHolder();

    jdbcTemplate.update(connection -> {
        PreparedStatement ps = connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
        ps.setString( parameterIndex: 1, parkingLot.getName());
        ps.setBigDecimal( parameterIndex: 2, parkingLot.getLongitude());
        ps.setBigDecimal( parameterIndex: 3, parkingLot.getLatitude());
        ps.setInt( parameterIndex: 4, parkingLot.getCapacity());
        ps.setBigDecimal( parameterIndex: 5, parkingLot.getTotalRevenue());
        ps.setBigDecimal( parameterIndex: 6, parkingLot.getTotalPenalty());
        return ps;
    }, keyHolder);

    return keyHolder.getKey() != null ? keyHolder.getKey().intValue() : -1;
}

no usages  Bahaa Khaled
@Override
public Optional<ParkingLot> getParkingLotById(int lotId) {
    String sql = "SELECT * FROM parking_lots WHERE lot_id = ?";
    return jdbcTemplate.query(sql, parkingLotRowMapper, lotId).stream().findFirst();
}

1 usage  Bahaa Khaled
@Override
public List<ParkingLot> getAllParkingLots() {
    String sql = "SELECT * FROM parking_lots";
    return jdbcTemplate.query(sql, parkingLotRowMapper);
}
```

```java
@Override
public boolean updateParkingLot(ParkingLot parkingLot) {
    String sql = "UPDATE parking_lots SET name = ?, longitude = ?, latitude = ?, capacity = ?, total_revenue = ?, total_penalty = ?  WHERE lot_id = ?";
    return jdbcTemplate.update(sql, parkingLot.getName(), parkingLot.getLongitude(), parkingLot.getLatitude(), parkingLot.getCapacity(), parkingLot.getTotalRevenue() , parkingLot.getTotalPenalt
}

no usages  Bahaa Khaled
@Override
public boolean deleteParkingLot(int lotId) {
    String sql = "DELETE FROM parking_lots WHERE lot_id = ?";
    return jdbcTemplate.update(sql, lotId) > 0;
}

2 usages  Abdullh911
@Override
@Transactional
public void updateTotalRevenue(int lotId, BigDecimal cost) {
    String sql = "UPDATE parking_lots SET total_revenue = total_revenue + ? WHERE lot_id = ?";
    jdbcTemplate.update(sql, cost, lotId);
}


3 usages  Abdullh911
@Override
@Transactional
public void updateTotalPenalty(int lotId, BigDecimal penalty) {
    String sql = "UPDATE parking_lots SET total_penalty = total_penalty + ? WHERE lot_id = ?";
    jdbcTemplate.update(sql, penalty, lotId);
}
```

3. Performance Optimization:

Database indexing:

CREATE INDEX idx_parking_spots_lot_status_type ON parking_spots(lot_id, status, type);
CREATE INDEX idx_reservations_user_id_status ON reservations(user_id, status);
CREATE INDEX idx_reservations_spot_status ON reservations(spot_id, status);
CREATE INDEX idx_users_username_email ON users(username, email);

---

4. Simulation:

import fetch from 'node-fetch';

```
const user = {
  id: 10,
  username: 'sim',
  password: 'Simserver@1234',
  email: 'sim@server.com',
};

const BASE_URL = 'http://localhost:8080';
let activeReservations = [];
let token = 'server';



const constructQueryParams = (params) =>
  Object.entries(params)
    .map(([key, value]) =>
`${encodeURIComponent(key)}=${encodeURIComponent(value)}`)
    .join('&');

const fetchLots = async () => {
```

```
  try {
    const queryParams = constructQueryParams({});
    const url = `${BASE_URL}/lots/getlots?${queryParams}`;

    const response = await fetch(url, {
      method: 'GET',
      headers: {
        'Content-Type': 'application/json',
        Authorization: `Bearer ${token}`,
      },
    });

    if (!response.ok) throw new Error('Failed to fetch lots');
    return await response.json();
  } catch (error) {
    console.error('Error fetching lots:', error);
    return [];
  }
};

const fetchSpots = async (lotId) => {
  try {
    const queryParams = constructQueryParams({ lotId });
    const url = `${BASE_URL}/spots/getspots?${queryParams}`;
    console.log(url);

    const response = await fetch(url, {
      method: 'GET',
      headers: {
        'Content-Type': 'application/json',
        Authorization: `Bearer ${token}`,
      },
    });

    if (!response.ok) throw new Error('Failed to fetch spots');
    const spots = await response.json();

    return spots.filter((spot) => spot.status === 'AVAILABLE');
  } catch (error) {
    console.error(`Error fetching spots for lot ${lotId}:`, error);
```

```
    return [];
  }
};

const reserveSpot = async (spot) => {
  const obj = {
    userId: user.id,
    spotId: spot.spotId,
    startTime : new Date(Date.now() + 60 * 60 * 1000*5).toISOString(),
    endTime: new Date(Date.now() + 60 * 60 * 1000*7).toISOString(),
    spotType: spot.type,
    status: 'ACTIVE',
    cost: 0,
    penalty: 0,
    spotNumber: spot.spotNumber,
    lotId:spot.lotId,
    lotName:"serverRes"
  };
  console.log(obj);

  console.log(`${BASE_URL}/reservations/reserve?reserve=true`);


  try {
    const response = await
fetch(`${BASE_URL}/reservations/reserve?reserve=true`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${token}`,
      },
      body: JSON.stringify(obj),
    });

    if (!response.ok) throw new Error('Failed to reserve spot');
    const reservation = await response.json();

    activeReservations.push(reservation.reservationId);
    console.log(`Reserved spot ${spot.spotId} successfully!`);
  } catch (error) {
```

```
    console.error('Error reserving spot:', error);
  }
};

const cancelReservation = async (reservationId) => {
  try {
    const url =
`${BASE_URL}/reservations/cancel?reservationId=${reservationId}`;
    console.log(url);

    const response = await fetch(url, {
      method: 'DELETE',
      headers: {
        'Authorization': `Bearer ${token}`,
      },
    });

    if (!response.ok) throw new Error('Failed to cancel reservation');
    activeReservations = activeReservations.filter((id) => id !==
reservationId);
    console.log(`Cancelled reservation ${reservationId} successfully!`);
  } catch (error) {
    console.error('Error cancelling reservation:', error);
  }
};

const main = async () => {


  setInterval(async () => {
    const lots = await fetchLots();

    if (lots.length === 0) return;

    const randomLot = lots[Math.floor(Math.random() * lots.length)];
    const spots = await fetchSpots(randomLot.lotId);
    if (spots.length === 0) return;

    const randomSpot = spots[Math.floor(Math.random() * spots.length)];
```

```
    await reserveSpot(randomSpot);
  }, Math.random() * 5000 + 2000); // Random interval between 5-10 seconds

  // Periodically cancel reservations
  setInterval(async () => {
    if (activeReservations.length === 0) return;
    console.log(activeReservations);

    const randomReservation = activeReservations[Math.floor(Math.random()
* activeReservations.length)];
    await cancelReservation(randomReservation);
  }, Math.random() * 7000 + 7000); // Random interval between 7-14 seconds
};

main();

import fetch from 'node-fetch';

const user = {
  id: 10,
  username: 'sim',
  password: 'Simserver@1234',
  email: 'sim@server.com',
};

const BASE_URL = 'http://localhost:8080';
let activeReservations = [];
let token = 'server';



const constructQueryParams = (params) =>
  Object.entries(params)
    .map(([key, value]) =>
`${encodeURIComponent(key)}=${encodeURIComponent(value)}`)
    .join('&');

const fetchLots = async () => {
  try {
    const queryParams = constructQueryParams({});
```

```
    const url = `${BASE_URL}/lots/getlots?${queryParams}`;

    const response = await fetch(url, {
      method: 'GET',
      headers: {
        'Content-Type': 'application/json',
        Authorization: `Bearer ${token}`,
      },
    });

    if (!response.ok) throw new Error('Failed to fetch lots');
    return await response.json();
  } catch (error) {
    console.error('Error fetching lots:', error);
    return [];
  }
};

const fetchSpots = async (lotId) => {
  try {
    const queryParams = constructQueryParams({ lotId });
    const url = `${BASE_URL}/spots/getspots?${queryParams}`;
    console.log(url);

    const response = await fetch(url, {
      method: 'GET',
      headers: {
        'Content-Type': 'application/json',
        Authorization: `Bearer ${token}`,
      },
    });

    if (!response.ok) throw new Error('Failed to fetch spots');
    const spots = await response.json();

    return spots.filter((spot) => spot.status === 'AVAILABLE');
  } catch (error) {
    console.error(`Error fetching spots for lot ${lotId}:`, error);
    return [];
  }
```

```
};

const reserveSpot = async (spot) => {
  const obj = {
    userId: user.id,
    spotId: spot.spotId,
    startTime : new Date(Date.now() + 60 * 60 * 1000*5).toISOString(),
    endTime: new Date(Date.now() + 60 * 60 * 1000*7).toISOString(),
    spotType: spot.type,
    status: 'ACTIVE',
    cost: 0,
    penalty: 0,
    spotNumber: spot.spotNumber,
    lotId:spot.lotId,
    lotName:"serverRes"
  };
  console.log(obj);

  console.log(`${BASE_URL}/reservations/reserve?reserve=true`);


  try {
    const response = await
fetch(`${BASE_URL}/reservations/reserve?reserve=true`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${token}`,
      },
      body: JSON.stringify(obj),
    });

    if (!response.ok) throw new Error('Failed to reserve spot');
    const reservation = await response.json();

    activeReservations.push(reservation.reservationId);
    console.log(`Reserved spot ${spot.spotId} successfully!`);
  } catch (error) {
    console.error('Error reserving spot:', error);
  }
```

```
};

const cancelReservation = async (reservationId) => {
  try {
    const url =
`${BASE_URL}/reservations/cancel?reservationId=${reservationId}`;
    console.log(url);

    const response = await fetch(url, {
      method: 'DELETE',
      headers: {
        'Authorization': `Bearer ${token}`,
      },
    });

    if (!response.ok) throw new Error('Failed to cancel reservation');
    activeReservations = activeReservations.filter((id) => id !==
reservationId);
    console.log(`Cancelled reservation ${reservationId} successfully!`);
  } catch (error) {
    console.error('Error cancelling reservation:', error);
  }
};

const main = async () => {


  setInterval(async () => {
    const lots = await fetchLots();

    if (lots.length === 0) return;

    const randomLot = lots[Math.floor(Math.random() * lots.length)];
    const spots = await fetchSpots(randomLot.lotId);
    if (spots.length === 0) return;

    const randomSpot = spots[Math.floor(Math.random() * spots.length)];

    await reserveSpot(randomSpot);
  }, Math.random() * 5000 + 2000); // Random interval between 5-10 seconds
```

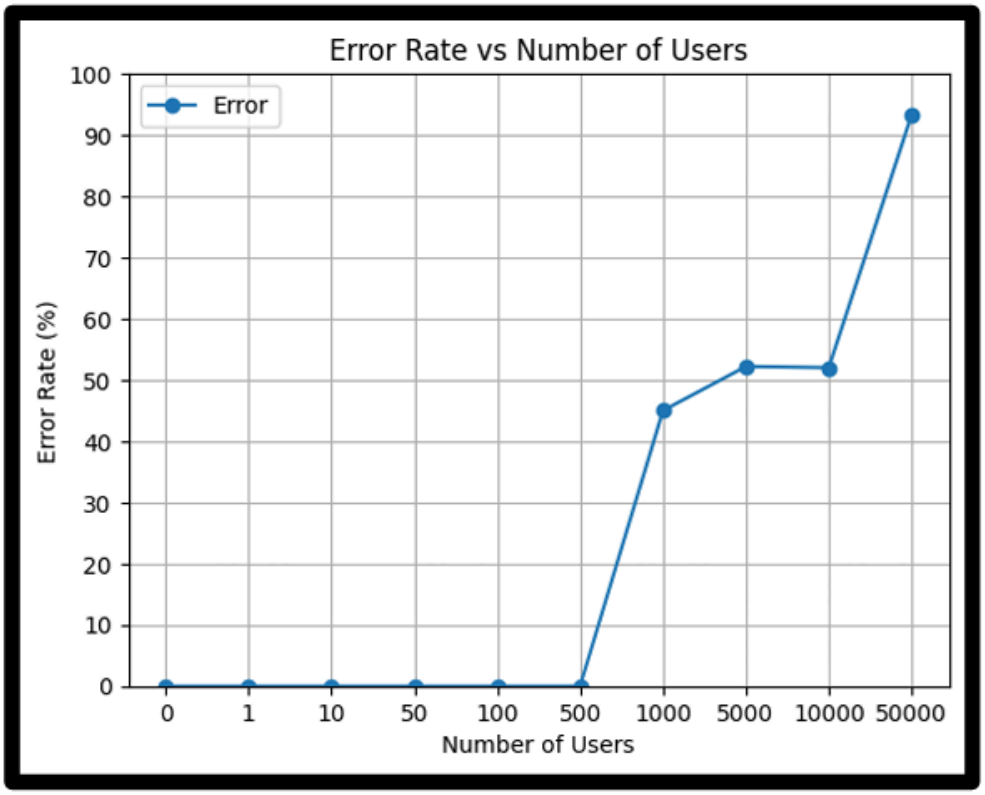
```
  // Periodically cancel reservations
  setInterval(async () => {
    if (activeReservations.length === 0) return;
    console.log(activeReservations);

    const randomReservation = activeReservations[Math.floor(Math.random()
* activeReservations.length)];
    await cancelReservation(randomReservation);
  }, Math.random() * 7000 + 7000); // Random interval between 7-14 seconds
};

main();
```

5. Performance Testing:

• We have runned JMeter with N users and (Ramp-up period = 0.5) & (loop count = 5)

• We generated a summary report for each N.

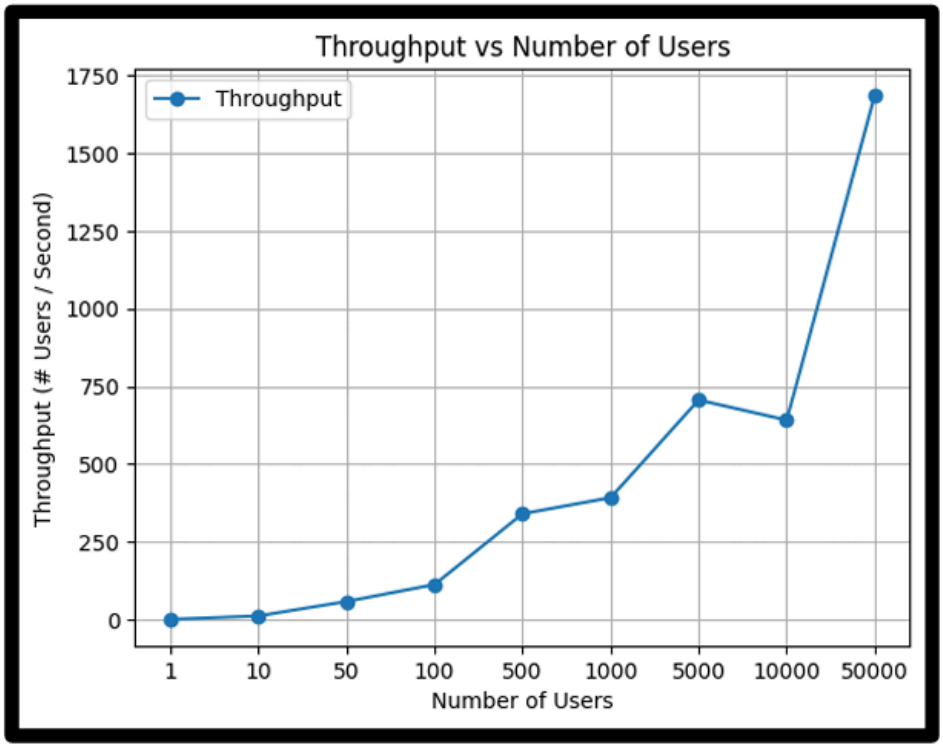
Error Rate (%) with different N:



	0	1	10	50	100	500	1000	5000	10000	50000
Error	0	0	0	0	0	0	45	52.2	52	93.3

	1	10	50	100	500	1000	5000	10000	50000
Throughput	1.2	12.1	58.3	112.8	340.5	392.2	706	641.6	1686

Throughput (#users/sec) with different N:



---

6. Reporting (Jasper):

Admin Reports: admin could get two types of reports

1. Report that has top users who reserved the most spots containing each user information (name , email , license_plate , number of reservations he/she made).
2. Report that has top Lots : top parking lots by revenue. The report contains lots ordered in descending order according to their revenues , containing lots information (lotid , lotname , lotlocation , lot revenue , capacity , and other information).

Manager reports :

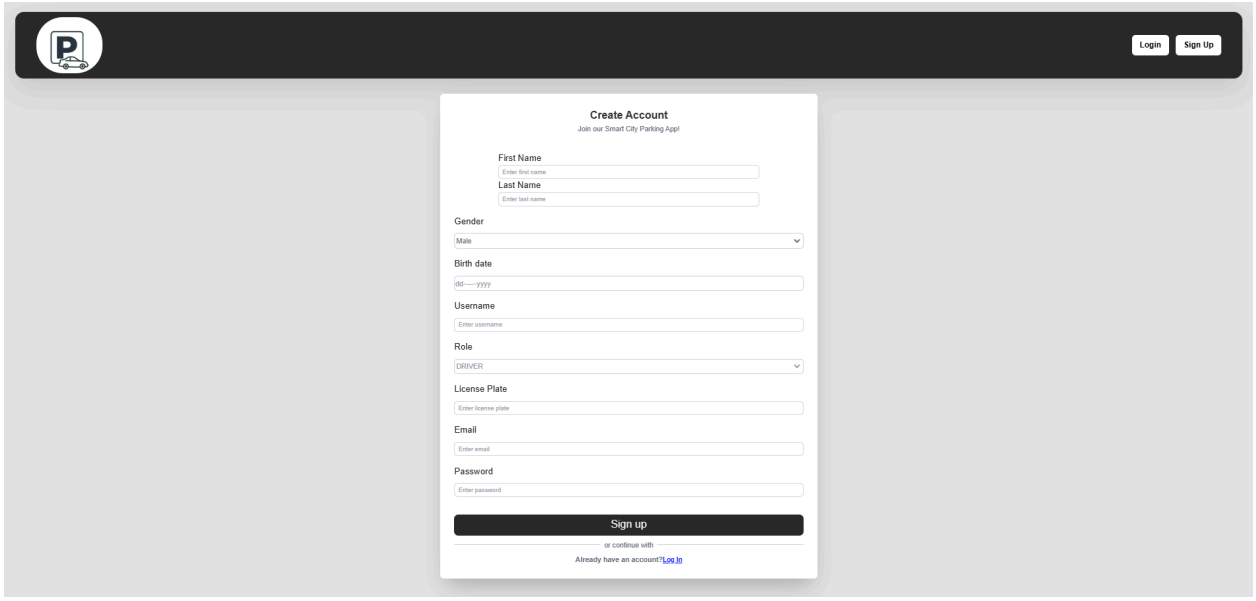
Reports for parking lot managers, each manager could have a report for lots he/she managed , containing lots information (lotid , lotname , lotlocation , lot revenue , capacity , number of occupied spots , occupancy rate).

Each time an admin or manager tries to show the report he / she will get a PDF report of the current real time data.

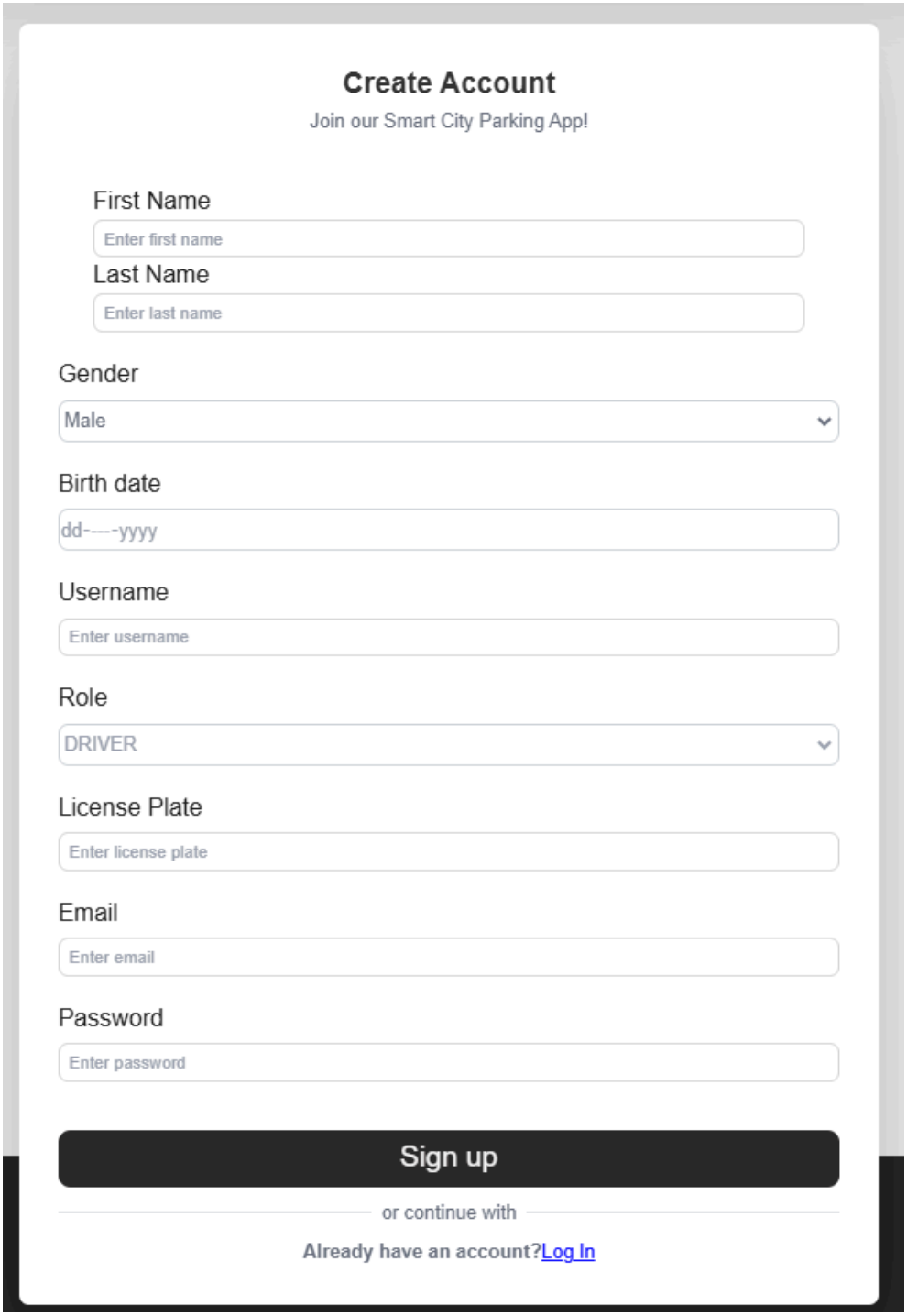
---

7. Screenshots and Functionality:

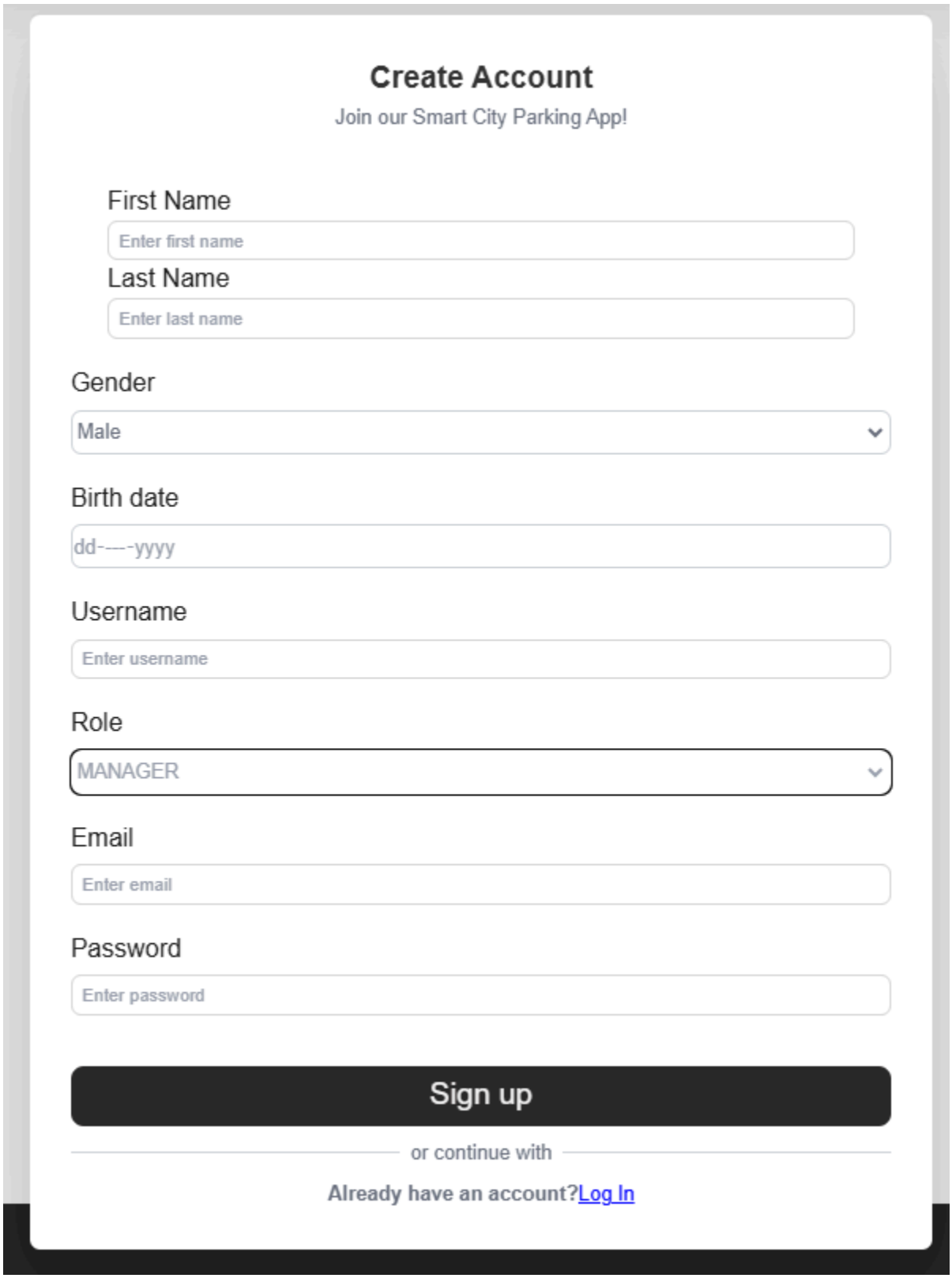
SignUp:



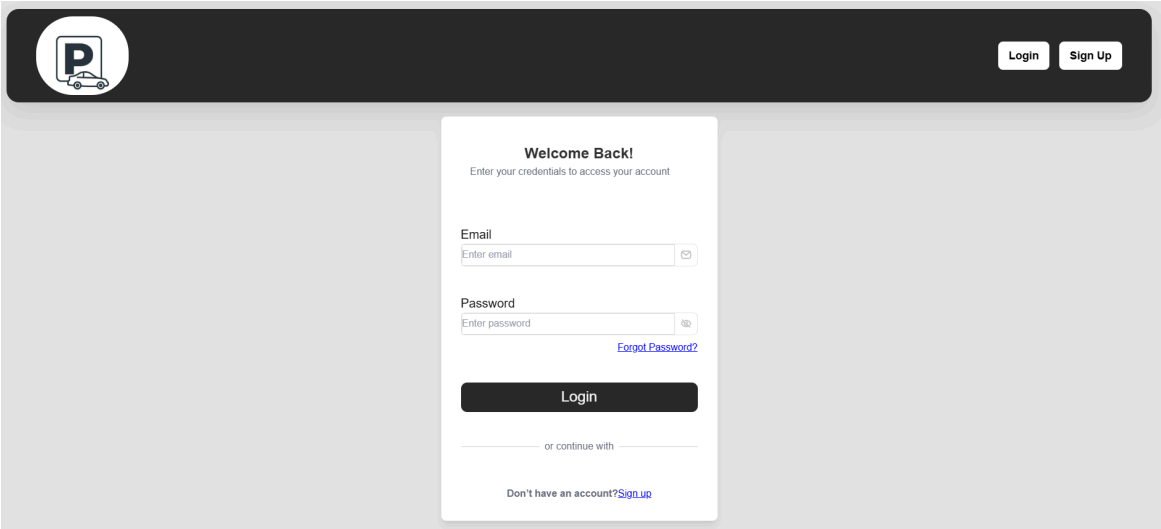
SignUp Form for DRIVER:



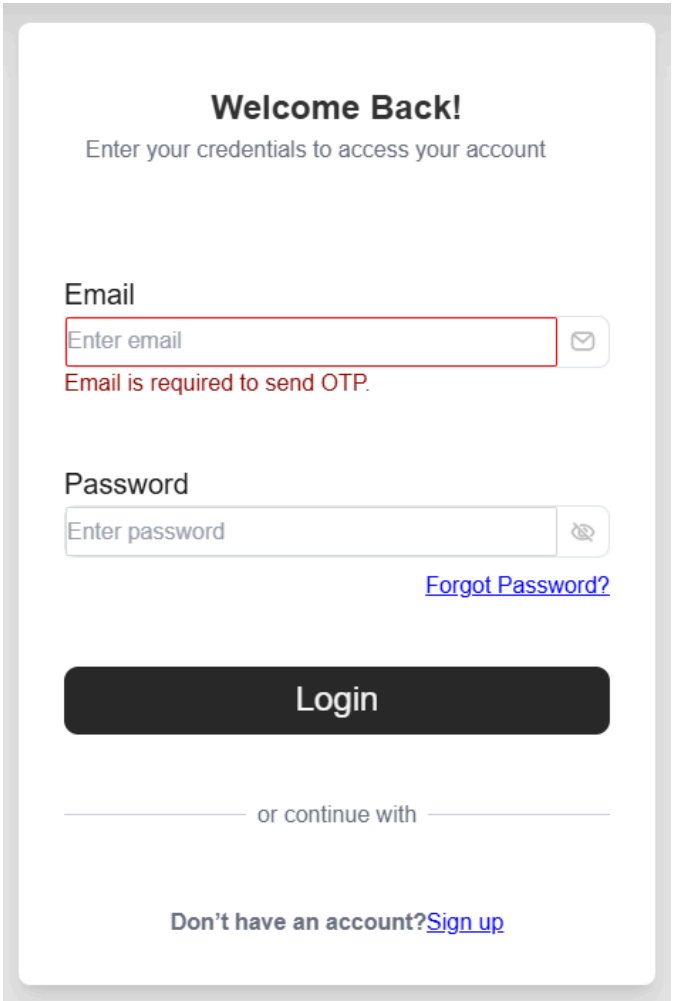
SignUp Form for MANAGER:



LoginForm:



ForgetPassword:



Welcome Back!

Enter your credentials to access your account

Email

ranimeshehata@gmail.com

Password

Enter password

Forgot Password?

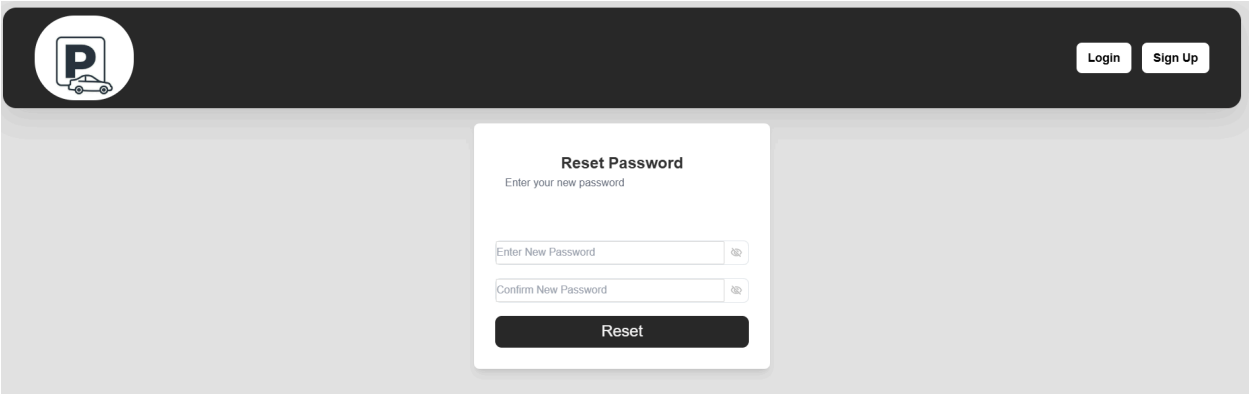
Login

Enter OTP

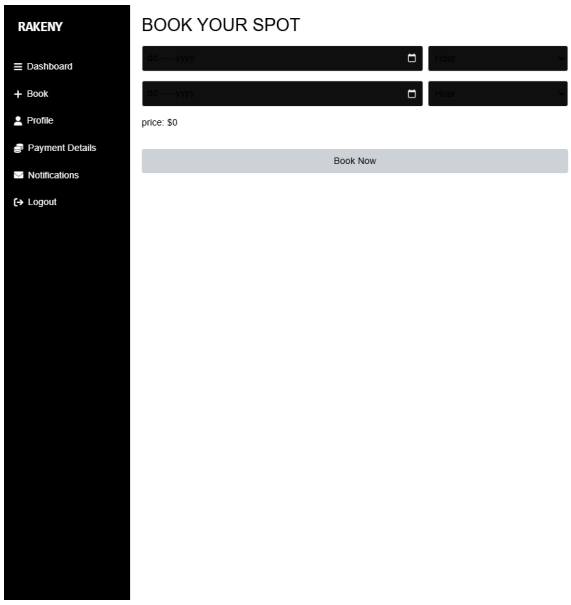
or continue with

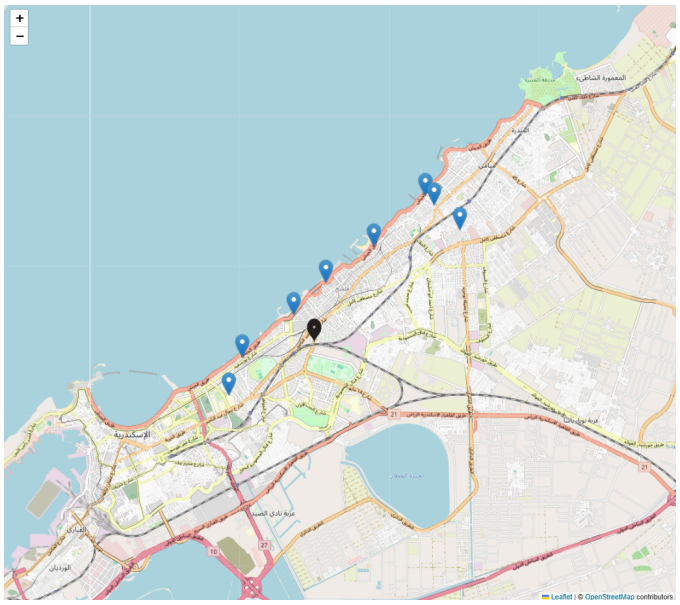
Don’t have an account?Sign up

Reset Password Form:

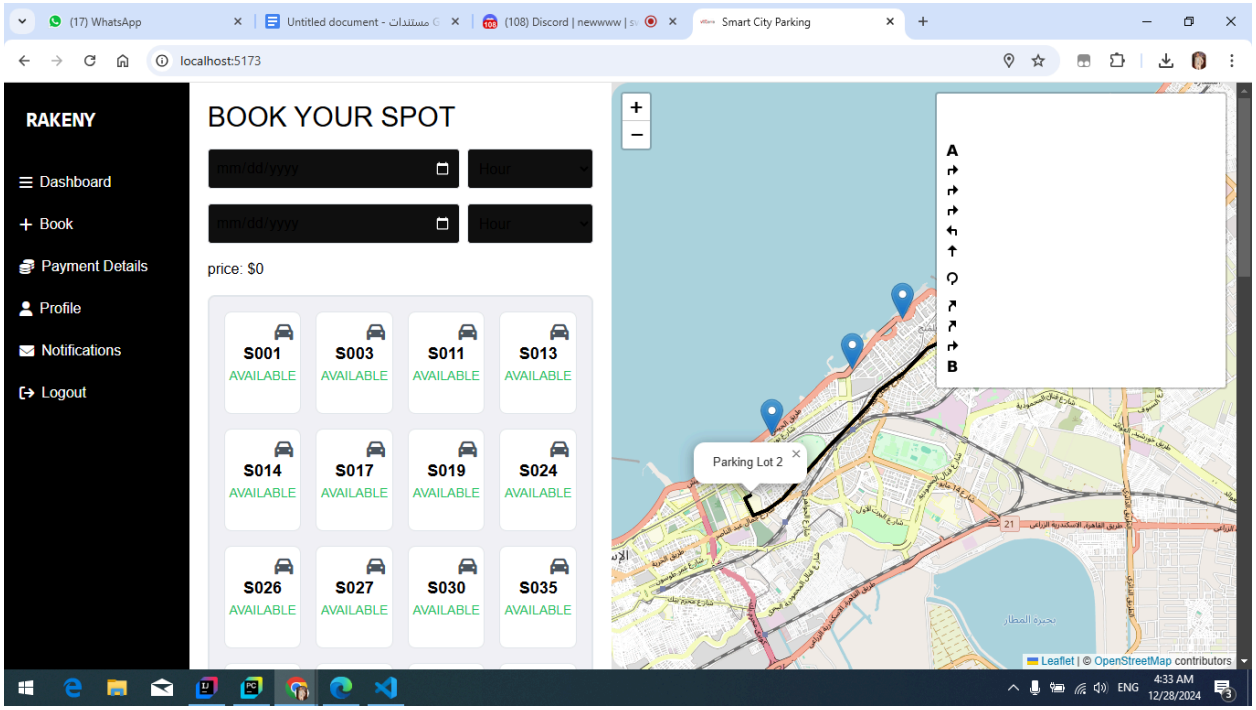


Main Page:

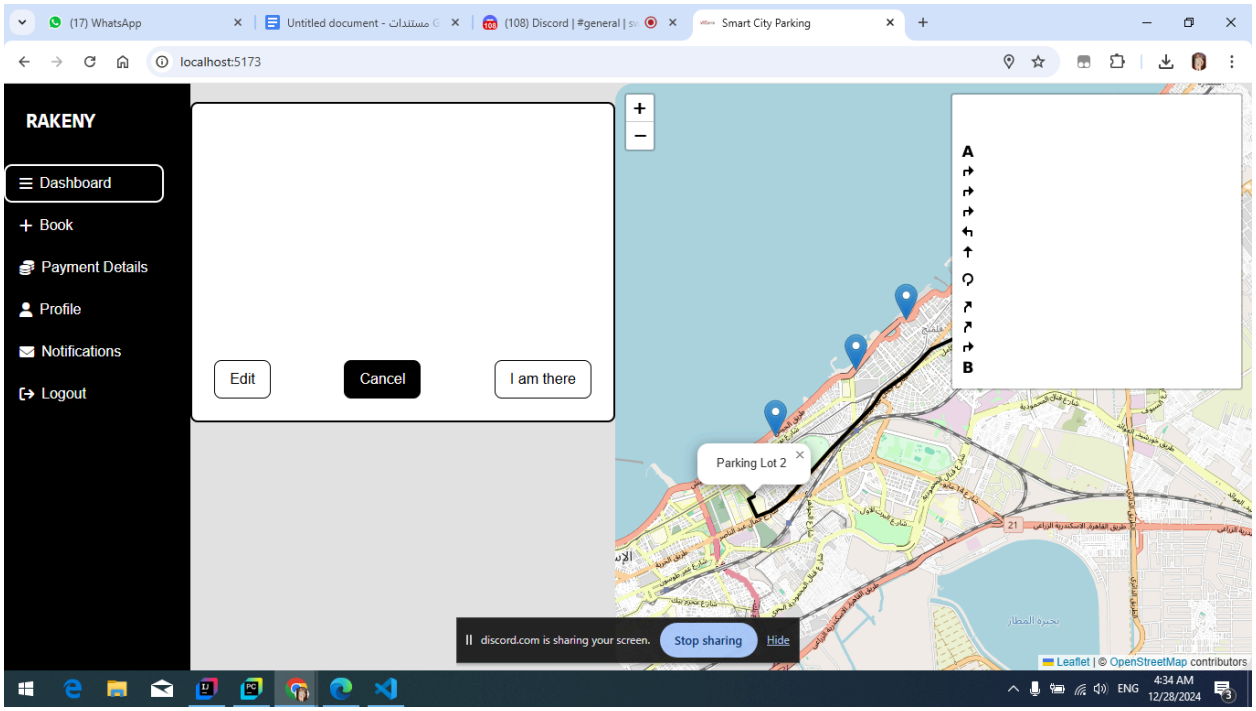




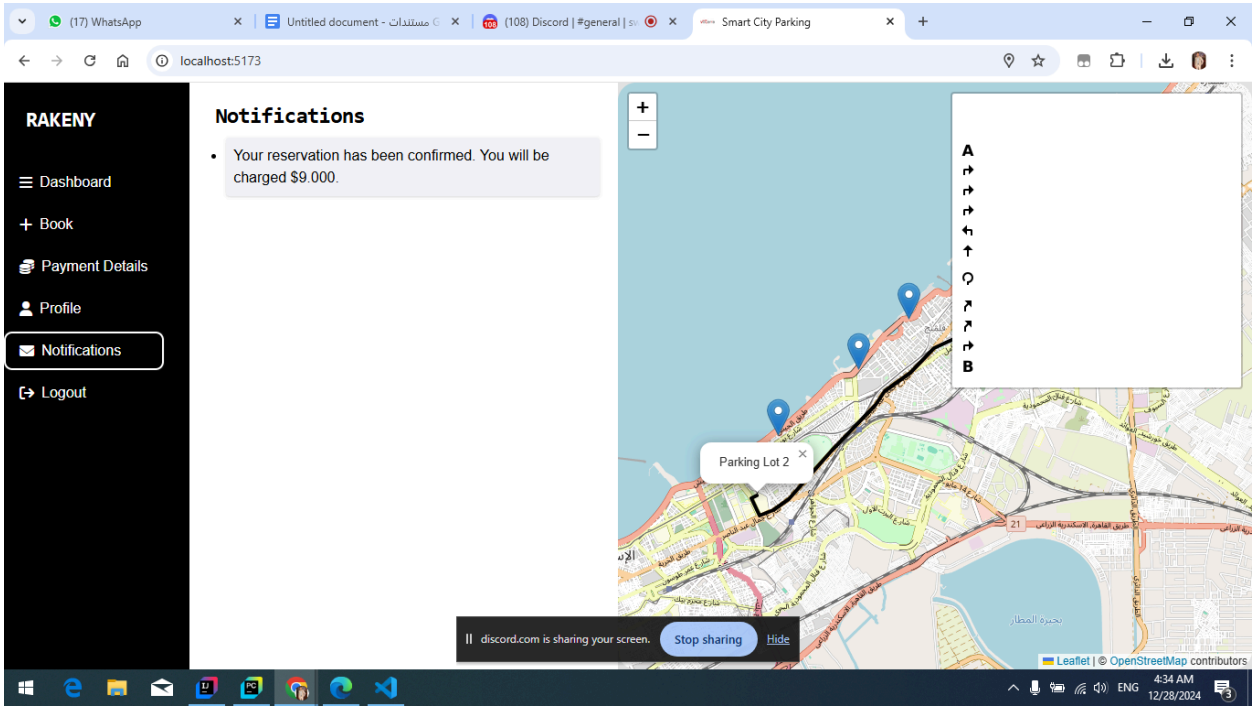
Reserving :



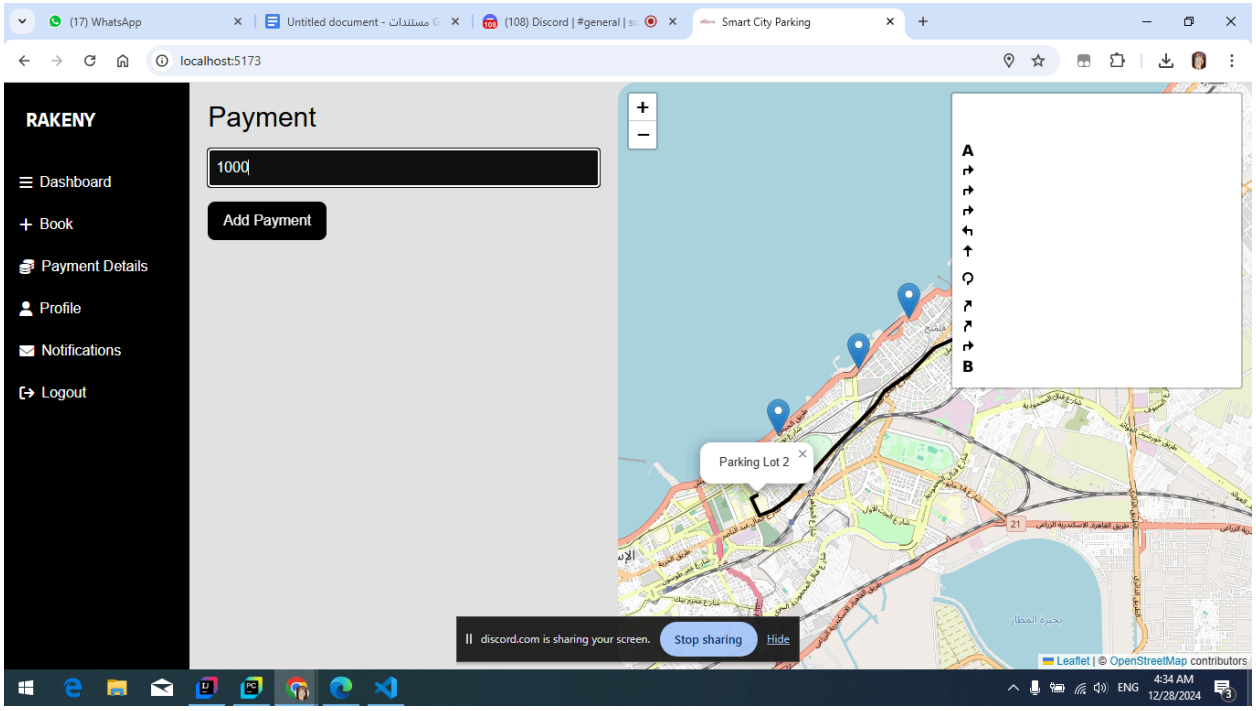
Reservations :



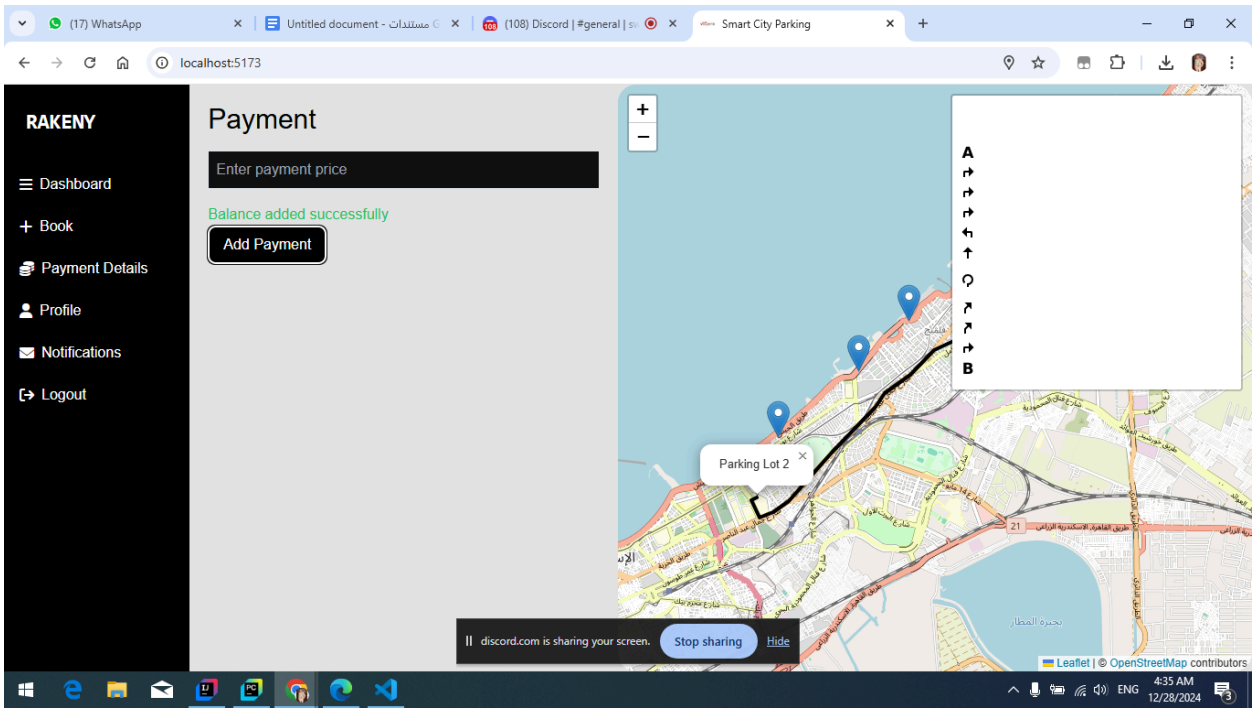
Notifications :



Payment :



Payment success :



---

8. Conclusion:

1. **Simulated IoT Integration**:A simulation of IoT sensors for real-time parking spot monitoring added a realistic smart city functionality without the need for actual hardware.
2. **Event-Driven Architecture**: The use of asynchronous notifications and event-driven updates ensured that changes in parking spot status were reflected instantly across the system, enhancing real-time responsiveness.

3. **Advanced Concurrency Handling**: The system incorporated robust concurrency controls to prevent double bookings during high-demand periods, ensuring consistency and reliability.
4. **Penalty System for No-Shows**: A penalty mechanism for unused reservations incentivized responsible usage and reduced system abuse, improving overall efficiency.
5. **Interactive Reporting Dashboard**: Administrators could visualize performance metrics through dynamic dashboards, offering an innovative way to monitor and manage parking lot operations in real-time.