

Smart City Parking

Management System

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

Table of Contents:

1. Introduction.
 2. Database Design.
 3. Performance Optimization.
 4. Simulation.
 5. Performance Testing.
 6. Reporting (Jasper).
 7. Screenshots and Functionality.
 8. Conclusion.
-

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.

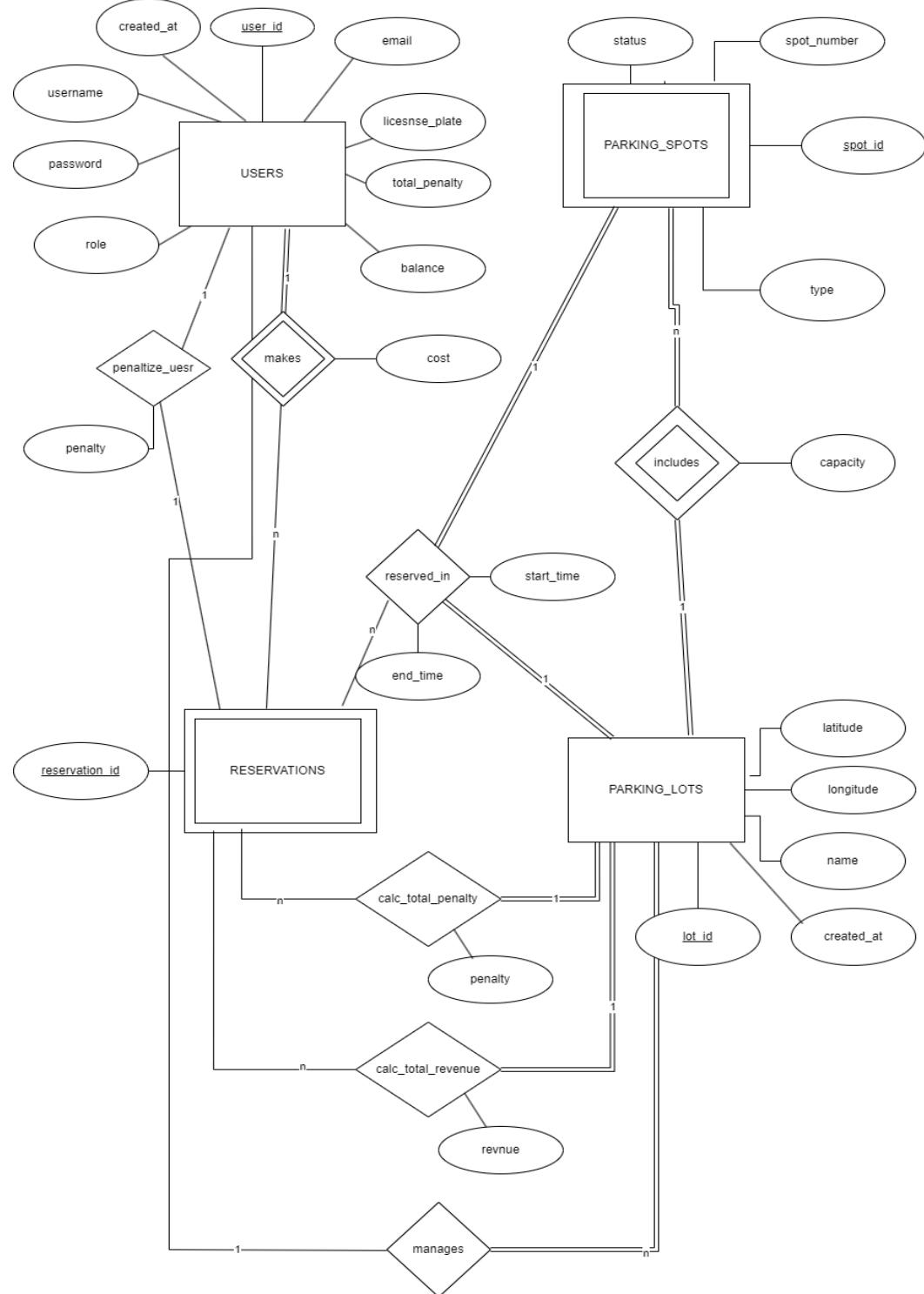
7. Scalability and Efficiency: Ensure the system is capable of handling high traffic volumes and multiple simultaneous transactions without compromising performance.

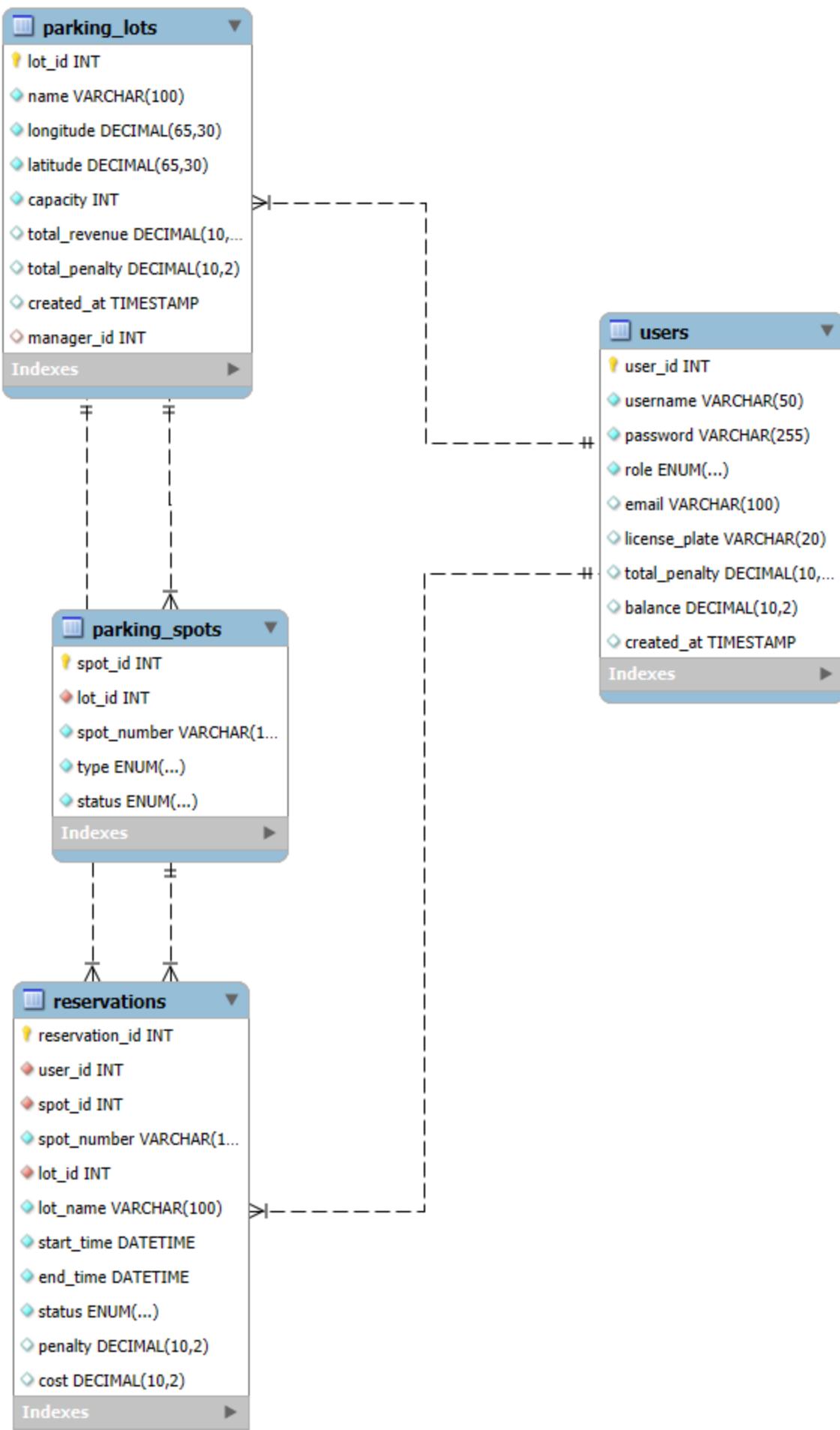
Technology Stack:

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

2. Database Design:

ERD Diagram:





Database Schema:

```
1 • CREATE DATABASE smart_city_parking;
2 SHOW DATABASES;
3 USE smart_city_parking;
4

5
6   CREATE TABLE users (
7     user_id INT AUTO_INCREMENT PRIMARY KEY,
8     username VARCHAR(50) NOT NULL UNIQUE,
9     password VARCHAR(255) NOT NULL,
10    role ENUM('DRIVER', 'MANAGER', 'ADMIN') NOT NULL,
11    email VARCHAR(100) UNIQUE,
12    license_plate VARCHAR(20), -- Only for drivers
13    total_penalty DECIMAL(10, 2) DEFAULT 0.00, -- For no-show or overstay
14    balance DECIMAL(10, 2) DEFAULT 0.00,
15    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
16  );
17
18
19   CREATE TABLE parking_lots (
20     lot_id INT AUTO_INCREMENT PRIMARY KEY,
21     name VARCHAR(100) NOT NULL,
22     longitude DECIMAL(65, 30) NOT NULL,
23     latitude DECIMAL(65, 30) NOT NULL,
24     capacity INT NOT NULL, -- Total number of spots
25     total_revenue DECIMAL(10, 2) DEFAULT 0.00,
26     total_penalty DECIMAL(10, 2) DEFAULT 0.00,
27     created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ,
28     manager_id INT ,
29     FOREIGN KEY (manager_id) REFERENCES users(user_id) ON DELETE SET NULL
30  );
31
32   CREATE TABLE parking_spots (
33     spot_id INT AUTO_INCREMENT PRIMARY KEY,
34     lot_id INT NOT NULL,
35     spot_number VARCHAR(10) NOT NULL, -- Unique per lot
36     type ENUM('REGULAR', 'DISABLED', 'EV') NOT NULL,
37     status ENUM('AVAILABLE', 'OCCUPIED', 'RESERVED') NOT NULL DEFAULT 'AVAILABLE',
38     FOREIGN KEY (lot_id) REFERENCES parking_lots(lot_id) ON DELETE CASCADE
39  );
40
```

```

42  ↗ CREATE TABLE reservations (
43      reservation_id INT AUTO_INCREMENT PRIMARY KEY,
44      user_id INT NOT NULL,
45      spot_id INT NOT NULL,
46      spot_number VARCHAR(10) NOT NULL,
47      lot_id INT NOT NULL,
48      lot_name VARCHAR(100) NOT NULL,
49      start_time DATETIME NOT NULL,
50      end_time DATETIME NOT NULL,
51      status ENUM('ACTIVE', 'COMPLETED', 'CANCELLED', 'NO_SHOW') NOT NULL DEFAULT 'ACTIVE',
52      penalty DECIMAL(10, 2) DEFAULT 0.00, -- For no-show or overstay
53      cost DECIMAL(10, 2) DEFAULT 0.00,
54      FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,
55      FOREIGN KEY (spot_id) REFERENCES parking_spots(spot_id) ON DELETE CASCADE,
56      FOREIGN KEY (lot_id) REFERENCES parking_lots(lot_id) ON DELETE CASCADE
57  );

```

Database Generation Script (SQL Files):

```

1 import pymysql
2
3 # Database connection details
4 db_config = {
5     "host": "localhost",
6     "user": "root",
7     "password": "smart_city_parking",
8     "database": "smart_city_parking"
9 }
10
11 # Coordinates and other details
12 coordinates = [
13     (31.258352933146817, 29.981002714990263),
14     (31.20836298155645, 29.923393916509106),
15     (31.245725767240582, 29.965868537277913),
16     (31.236611050301057, 29.951834406735134),
17     (31.228463715093138, 29.942403546882044),
18     (31.218009329994388, 29.92733759714959),
19     (31.25602426146235, 29.98355999465894),
20     (31.249734785675155, 29.991109772853328)
21 ]
22
23
24 # Insert data into parking_lots table
25 def insert_parking_lots(cursor):
26     for i, (latitude, longitude) in enumerate(coordinates):
27         name = f'Parking Lot {i + 1}'
28         capacity = 100 # Example capacity
29         manager_id = i + 1 # Assign manager IDs sequentially from 1 to 8
30         cursor.execute('''
31             INSERT INTO parking_lots (name, longitude, latitude, capacity, total_revenue, total_penalty, manager_id)
32             VALUES (%s, %s, %s, %s, 0, 0, %s)
33             ''', (name, longitude, latitude, capacity, manager_id))
34

```

```
36 # Insert data into users table
37 def insert_users(cursor):
38     for i in range(1, 9):
39         username = f'manager{i}'
40         password = '$2a$10$6lHJU7vPaaknD4LRShR8IeYvSGxZy63SUsKLXmxYxRMo4P/mj3g6W'
41         role = 'MANAGER'
42         email = f'manager{i}@example.com'
43         cursor.execute('''
44             INSERT INTO users (username, password, role, email)
45             VALUES (%s, %s, %s, %s)
46             ''', (username, password, role, email))
47
48
49 # Main function to execute the insertion
50 def insert_data():
51     try:
52         conn = pymysql.connect(**db_config)
53         cursor = conn.cursor()
54
55         print("Inserting data into users table...")
56         insert_users(cursor)
57
58         print("Inserting data into parking_lots table...")
59         insert_parking_lots(cursor)
60
61         conn.commit()
62
63         print("Entries added successfully!")
64
65     except pymysql.MySQLError as err:
66         print(f"Error: {err}")
67
68     finally:
69         # Close the cursor and connection
70         if 'cursor' in locals():
71             cursor.close()
72         if 'conn' in locals():
73             conn.close()
```

```

1 import pymysql
2 import random
3
4 # Database connection details
5 db_config = {
6     "host": "localhost",
7     "user": "root",
8     "password": "smart_city_parking",
9     "database": "smart_city_parking"
10 }
11
12 # Spot types and statuses
13 spot_types = ['REGULAR', 'DISABLED', 'EV']
14 spot_statuses = ['AVAILABLE', 'OCCUPIED', 'RESERVED']
15
16 # Insert parking spots into the parking_spots table
17 def insert_parking_spots(cursor, lot_id, num_spots):
18     for spot_num in range(1, num_spots + 1):
19         spot_type = random.choice(spot_types)
20         spot_status = random.choice(spot_statuses) if random.random() < 0.3 else 'AVAILABLE' # 30% chance for non-available
21         spot_number = f"S{spot_num:03d}" # Format as S001, S002, etc.
22         cursor.execute('''
23             INSERT INTO parking_spots (lot_id, spot_number, type, status)
24             VALUES (%s, %s, %s, %s)
25             ''', (lot_id, spot_number, spot_type, spot_status))
26
27 # Main function to execute the insertion
28 def insert_data():
29     try:
30         conn = pymysql.connect(**db_config)
31         cursor = conn.cursor()
32
33         # Retrieve lot IDs from the parking_lots table
34         cursor.execute("SELECT lot_id FROM parking_lots")
35         lot_ids = cursor.fetchall()
36
37         print("Inserting parking spots for each lot...")
38         for lot_id_tuple in lot_ids:
39             lot_id = lot_id_tuple[0]
40             num_spots = 50 # Example: 50 spots per lot
41             insert_parking_spots(cursor, lot_id, num_spots)
42
43         conn.commit()
44         print("Parking spots added successfully!")
45
46     except pymysql.MySQLError as err:
47         print(f"Error: {err}")
48
49     finally:
50         # Close the cursor and connection
51         if 'cursor' in locals():
52             cursor.close()
53         if 'conn' in locals():
54             conn.close()
55
56 # Run the function
57 if __name__ == "__main__":
58     insert_data()
59

```

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 = ?)  
""";
```

```
1 import pymysql
2
3 # Database connection details
4 db_config = {
5     "host": "localhost",
6     "user": "root",
7     "password": "smart_city_parking",
8     "database": "smart_city_parking"
9 }
10
11 # Coordinates and other details
12 coordinates = [
13     (31.258352933146817, 29.981002714990263),
14     (31.20836298155645, 29.923393916509106),
15     (31.245725767240582, 29.965868537277913),
16     (31.236611050301057, 29.951834406735134),
17     (31.228463715093138, 29.942403546882044),
18     (31.218009329994388, 29.92733759714959),
19     (31.25602426146235, 29.98355999465894),
20     (31.249734785675155, 29.991109772853328)
21 ]
22
23
24 # Insert data into parking_lots table
25 def insert_parking_lots(cursor):
26     for i, (latitude, longitude) in enumerate(coordinates):
27         name = f'Parking Lot {i + 1}'
28         capacity = 100 # Example capacity
29         manager_id = i + 1 # Assign manager IDs sequentially from 1 to 8
30         cursor.execute('''
31             INSERT INTO parking_lots (name, longitude, latitude, capacity, total_revenue, total_penalty, manager_id)
32             VALUES (%s, %s, %s, %s, 0, 0, %s)
33             ''', (name, longitude, latitude, capacity, manager_id))
```

```
36 # Insert data into users table
37 def insert_users(cursor):
38     for i in range(1, 9):
39         username = f'manager{i}'
40         password = '$2a$10$6lHJU7vPaaknD4LRShR8IeYvSGxZy63SUsKLXmxYxRMo4P/mj3g6W'
41         role = 'MANAGER'
42         email = f'manager{i}@example.com'
43         cursor.execute('''
44             INSERT INTO users (username, password, role, email)
45             VALUES (%s, %s, %s, %s)
46             ''', (username, password, role, email))
47
48
49 # Main function to execute the insertion
50 def insert_data():
51     try:
52         conn = pymysql.connect(**db_config)
53         cursor = conn.cursor()
54
55         print("Inserting data into users table...")
56         insert_users(cursor)
57
58         print("Inserting data into parking_lots table...")
59         insert_parking_lots(cursor)
60
61         conn.commit()
62
63         print("Entries added successfully!")
64
65     except pymysql.MySQLError as err:
66         print(f"Error: {err}")
67
68     finally:
69         # Close the cursor and connection
70         if 'cursor' in locals():
71             cursor.close()
72         if 'conn' in locals():
73             conn.close()
```

```

1 import pymysql
2 import random
3
4 # Database connection details
5 db_config = {
6     "host": "localhost",
7     "user": "root",
8     "password": "smart_city_parking",
9     "database": "smart_city_parking"
10 }
11
12 # Spot types and statuses
13 spot_types = ['REGULAR', 'DISABLED', 'EV']
14 spot_statuses = ['AVAILABLE', 'OCCUPIED', 'RESERVED']
15
16 # Insert parking spots into the parking_spots table
17 def insert_parking_spots(cursor, lot_id, num_spots):
18     for spot_num in range(1, num_spots + 1):
19         spot_type = random.choice(spot_types)
20         spot_status = random.choice(spot_statuses) if random.random() < 0.3 else 'AVAILABLE' # 30% chance for non-available
21         spot_number = f"S{spot_num:03d}" # Format as S001, S002, etc.
22         cursor.execute('''
23             INSERT INTO parking_spots (lot_id, spot_number, type, status)
24             VALUES (%s, %s, %s, %s)
25             ''', (lot_id, spot_number, spot_type, spot_status))
26
27 # Main function to execute the insertion
28 def insert_data():
29     try:
30         conn = pymysql.connect(**db_config)
31         cursor = conn.cursor()
32
33         # Retrieve lot IDs from the parking_lots table
34         cursor.execute("SELECT lot_id FROM parking_lots")
35         lot_ids = cursor.fetchall()
36
37         print("Inserting parking spots for each lot...")
38         for lot_id_tuple in lot_ids:
39             lot_id = lot_id_tuple[0]
40             num_spots = 50 # Example: 50 spots per lot
41             insert_parking_spots(cursor, lot_id, num_spots)
42
43         conn.commit()
44         print("Parking spots added successfully!")
45
46     except pymysql.MySQLError as err:
47         print(f"Error: {err}")
48
49     finally:
50         # Close the cursor and connection
51         if 'cursor' in locals():
52             cursor.close()
53         if 'conn' in locals():
54             conn.close()
55
56 # Run the function
57 if __name__ == "__main__":
58     insert_data()
59

```

```

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;
}

4 usages  ▲ Bahaa Khaled
@Override
public Optional<User> getUserId(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> getUsername(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> getEmail(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;
}
```

```
+ 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;
    }
}
```

```
5     @Override
6     @Transactional
7     public boolean deleteReservation(int reservationId) {
8         // Retrieve the reservation to get the spot ID
9         Optional<Reservation> reservationOpt = getReservationById(reservationId);
10        if (reservationOpt.isEmpty()) {
11            throw new IllegalStateException("Reservation not found.");
12        }
13
14        // Free up the parking spot
15        String updateSpotSql = "UPDATE parking_spots SET status = 'AVAILABLE' WHERE spot_id = ?";
16        jdbcTemplate.update(updateSpotSql, reservationOpt.get().getSpotId());
17
18        // Delete the reservation
19        String deleteReservationSql = "DELETE FROM reservations WHERE reservation_id = ?";
20        return jdbcTemplate.update(deleteReservationSql, reservationId) > 0;
21    }
22
23
24    1 usage  ↳ Bahaa Khaled
25    @Override
26    @Transactional
27    public boolean isSpotAvailableForDuration(int spotId, LocalDateTime startTime, LocalDateTime endTime) {
28        String sql = "SELECT COUNT(*) FROM reservations " +
29                    "WHERE spot_id = ? AND " +
30                    "(start_time < ? AND end_time > ?) AND " +
31                    "status = 'ACTIVE'";
32        Integer count = jdbcTemplate.queryForObject(sql, Integer.class, spotId, endTime, startTime, startTime);
33        return count == 0;
34    }
35
36
37    1 usage  ↳ Bahaa Khaled
38    @Override
39    public List<Reservation> getExpiredReservations(LocalDateTime currentTime) {
40        String sql = """
41            SELECT * FROM reservations
42            WHERE end_time < ? AND status = 'ACTIVE'
43        """;
44        return jdbcTemplate.query(sql, reservationRowMapper, currentTime);
45    }
46
```

```
14     @Override
15     public List<Reservation> getReservationsByStatus(String status) {
16         String sql = """
17             SELECT * FROM reservations
18             WHERE status = ?
19             """;
20         return jdbcTemplate.query(sql, reservationRowMapper, status);
21     }
22
23     1 usage  ↳ Bahaa Khaled
24     @Override
25     public List<Reservation> getReservationsBySpotStatus(String status) {
26         String sql = """
27             SELECT * FROM reservations
28             WHERE spot_id IN (SELECT spot_id FROM parking_spots WHERE status = ?)
29             """;
30         return jdbcTemplate.query(sql, reservationRowMapper, status);
31     }
32
33     1 usage  ↳ Bahaa Khaled
34     @Override
35     public List<Reservation> getStartedReservations(LocalDateTime currentTime) {
36         String sql = """
37             SELECT * FROM reservations
38             WHERE start_time < ? AND status = 'ACTIVE'
39             """;
40         return jdbcTemplate.query(sql, reservationRowMapper, currentTime);
41     }
42
43     no usages  ↳ Bahaa Khaled
44     @Override
45     public List<Reservation> getReservationsByLotId(int lotId) {
46         String sql = """
47             SELECT * FROM reservations
48             WHERE lot_id = ?
49             """;
50         return jdbcTemplate.query(sql, reservationRowMapper, lotId);
51     }
52 }
```

```
    @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 ▲ Abdullah911+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);
    }
}
```

```
    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);
    }

```

```

44      no usages ▾ Bahaa Khaled
45  ⏷ @Override
46  public int createParkingLot(ParkingLot parkingLot) {
47      String sql = "INSERT INTO parking_lots (name, longitude, latitude, capacity, total_revenue, total_penalty) VALUES (?, ?, ?, ?, ?, ?)";
48      KeyHolder keyHolder = new GeneratedKeyHolder();
49
50      jdbcTemplate.update(connection -> {
51          PreparedStatement ps = connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
52          ps.setString( parameterIndex: 1, parkingLot.getName());
53          ps.setBigDecimal( parameterIndex: 2, parkingLot.getLongitude());
54          ps.setBigDecimal( parameterIndex: 3, parkingLot.getLatitude());
55          ps.setInt( parameterIndex: 4, parkingLot.getCapacity());
56          ps.setBigDecimal( parameterIndex: 5, parkingLot.getTotalRevenue());
57          ps.setBigDecimal( parameterIndex: 6, parkingLot.getTotalPenalty());
58          return ps;
59      }, keyHolder);
60
61      return keyHolder.getKey() != null ? keyHolder.getKey().intValue() : -1;
62  }
63
64      no usages ▾ Bahaa Khaled
65  ⏷ @Override
66  public Optional<ParkingLot> getParkingLotById(int lotId) {
67      String sql = "SELECT * FROM parking_lots WHERE lot_id = ?";
68      return jdbcTemplate.query(sql, parkingLotRowMapper, lotId).stream().findFirst();
69
70      1 usage ▾ Bahaa Khaled
71  ⏷ @Override
72  public List<ParkingLot> getAllParkingLots() {
73      String sql = "SELECT * FROM parking_lots";
74      return jdbcTemplate.query(sql, parkingLotRowMapper);
75  }
76
77      * All JDBC API
78
79  ⏷ @Override
80  public boolean updateParkingLot(ParkingLot parkingLot) {
81      String sql = "UPDATE parking_lots SET name = ?, longitude = ?, latitude = ?, capacity = ?, total_revenue = ?, total_penalty = ? WHERE lot_id = ?";
82      return jdbcTemplate.update(sql, parkingLot.getName(), parkingLot.getLongitude(), parkingLot.getLatitude(), parkingLot.getCapacity(), parkingLot.getTotalRevenue(), parkingLot.getTotalPenalty());
83
84      no usages ▾ Bahaa Khaled
85  ⏷ @Override
86  public boolean deleteParkingLot(int lotId) {
87      String sql = "DELETE FROM parking_lots WHERE lot_id = ?";
88      return jdbcTemplate.update(sql, lotId) > 0;
89  }
90
91      2 usages ▾ Abdullah911
92  ⏷ @Override
93  ⏷ @Transactional
94  public void updateTotalRevenue(int lotId, BigDecimal cost) {
95      String sql = "UPDATE parking_lots SET total_revenue = total_revenue + ? WHERE lot_id = ?";
96      jdbcTemplate.update(sql, cost, lotId);
97  }
98
99      3 usages ▾ Abdullah911
100  ⏷ @Override
101  ⏷ @Transactional
102  public void updateTotalPenalty(int lotId, BigDecimal penalty) {
103      String sql = "UPDATE parking_lots SET total_penalty = total_penalty + ? WHERE lot_id = ?";
104      jdbcTemplate.update(sql, penalty, lotId);
105  }

```

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
  }, 1000);
};
```

```
// 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
};

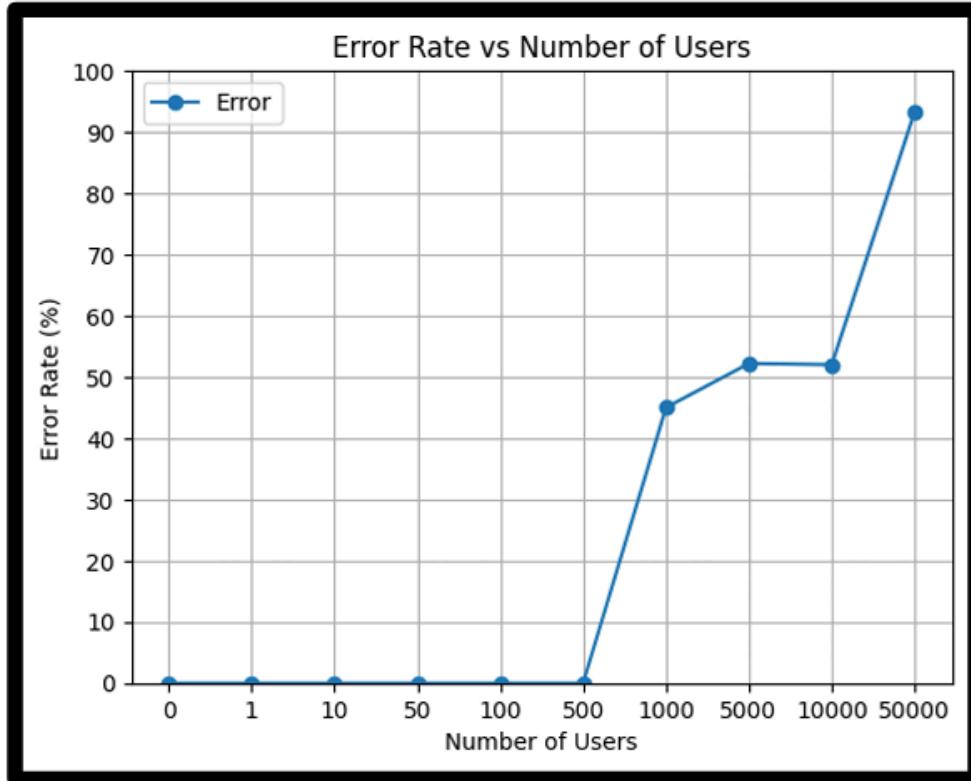
main();
```



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.

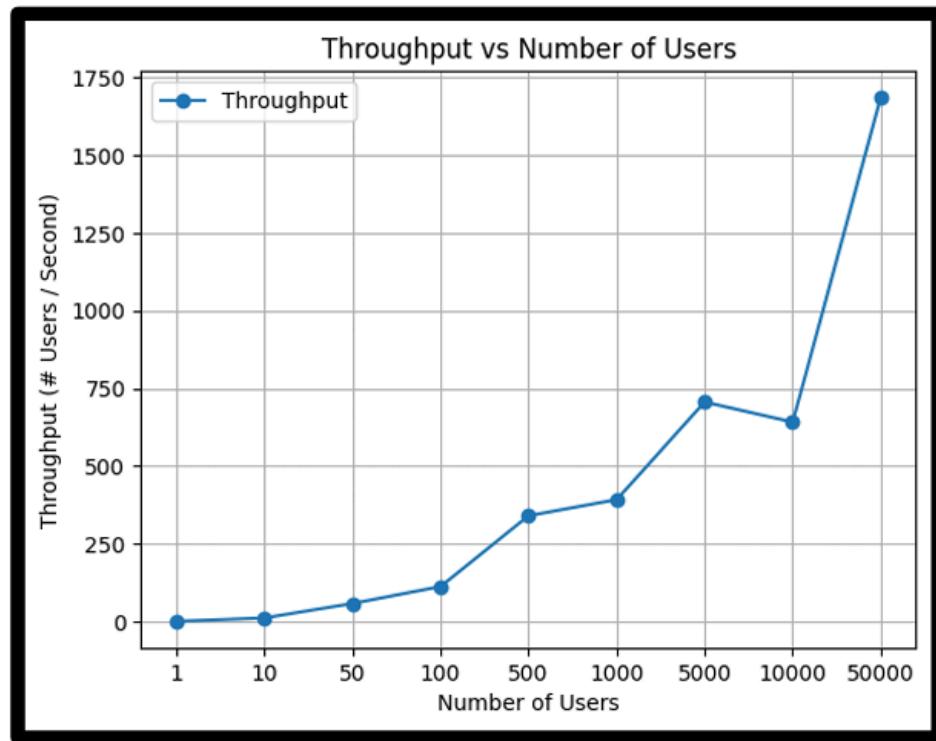
Error Rate (%) with different N:



	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).

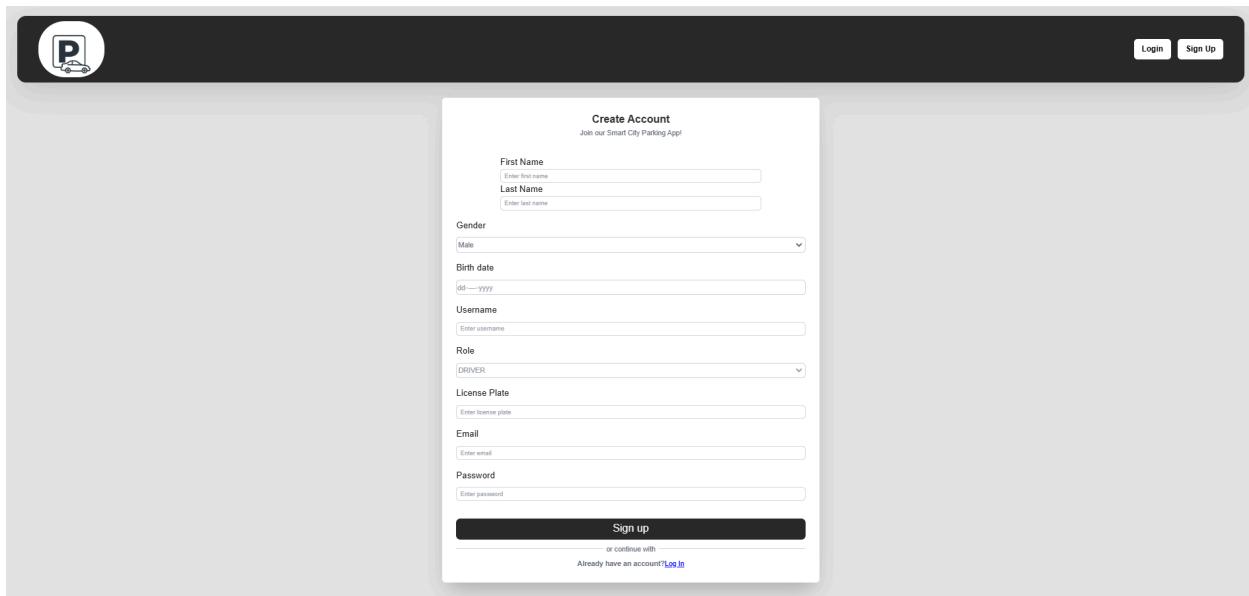
Manager reports :

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:



The screenshot shows the 'Create Account' page of the Smart City Parking App. At the top, there's a logo of a car with a 'P' inside a circle, followed by the text 'Join our Smart City Parking App!'. On the right side, there are 'Login' and 'Sign Up' buttons. The main form is titled 'Create Account' and contains fields for First Name, Last Name, Gender (Male), Birth date, Username, Role (set to 'DRIVER'), License Plate, Email, and Password. Below the form is a 'Sign up' button and a link to 'or continue with' social media accounts (Facebook, Google, and others). There's also a note about already having an account.

Create Account
Join our Smart City Parking App!

First Name
Last Name
Gender
Male
Birth date
dd - yyyy
Username
Role
DRIVER
License Plate
Email
Password

Sign up
or continue with
Already have an account? [Log In](#)

SignUp Form for DRIVER:

Create Account

Join our Smart City Parking App!

First Name

Last Name

Gender

Birth date

Username

Role

License Plate

Email

Password

Sign up

or continue with

Already have an account? [Log In](#)

SignUp Form for MANAGER:

Create Account

Join our Smart City Parking App!

First Name

Last Name

Gender

Birth date

Username

Role

Email

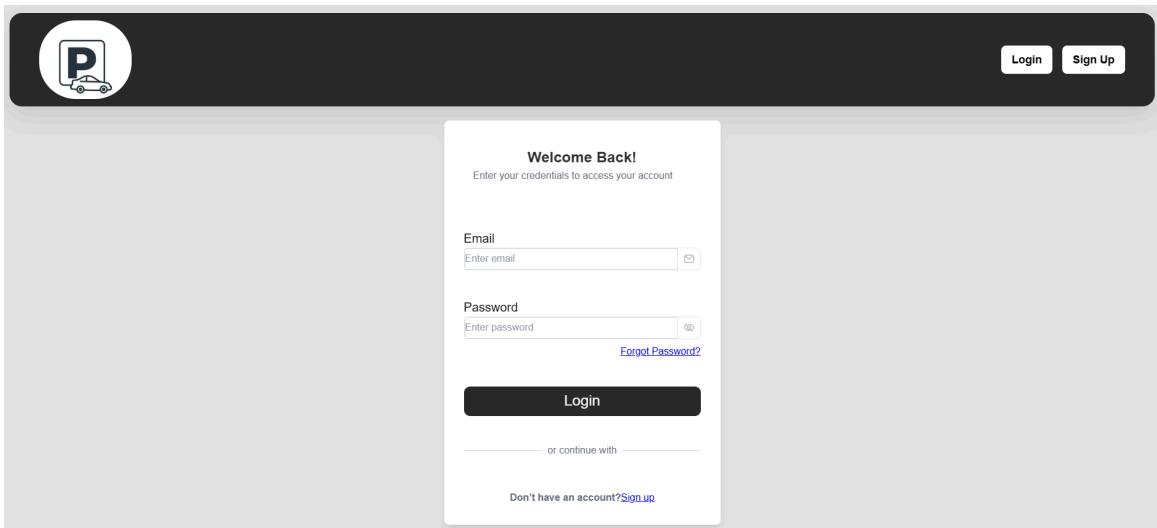
Password

Sign up

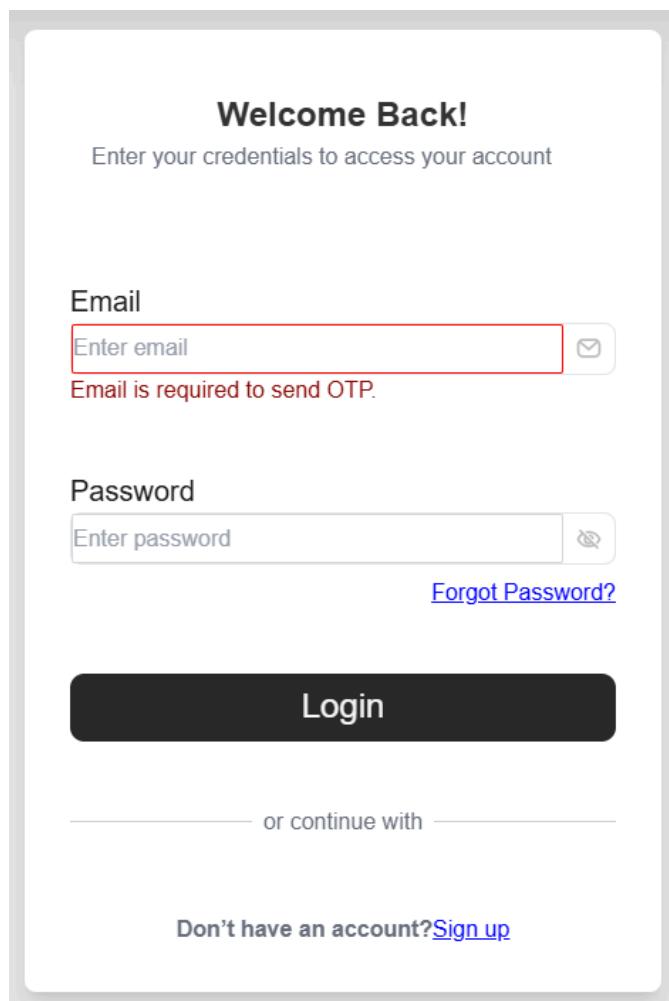
or continue with

Already have an account? [Log In](#)

LoginForm:



ForgetPassword:



Welcome Back!

Enter your credentials to access your account

Email

ranimeshehata@gmail.com



Password

Enter password



[Forgot Password?](#)

Login

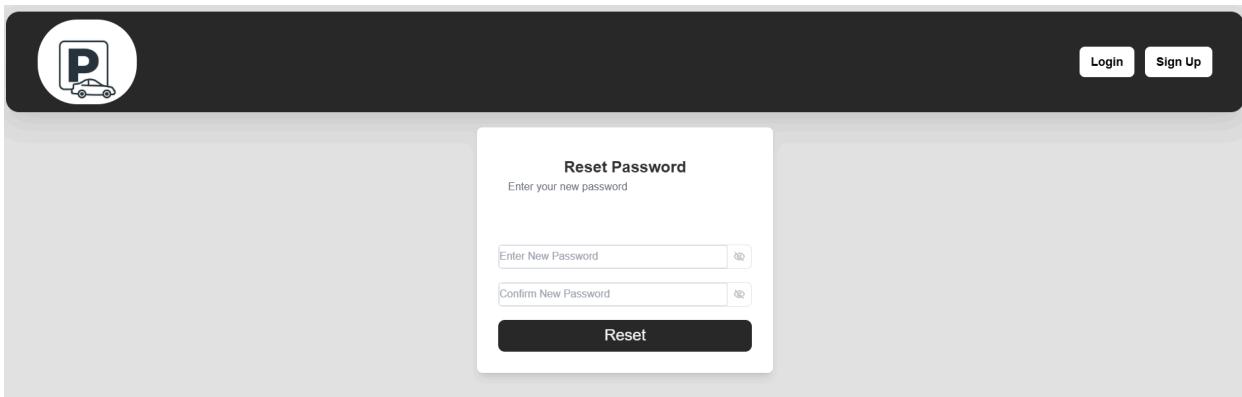
Enter OTP



or continue with

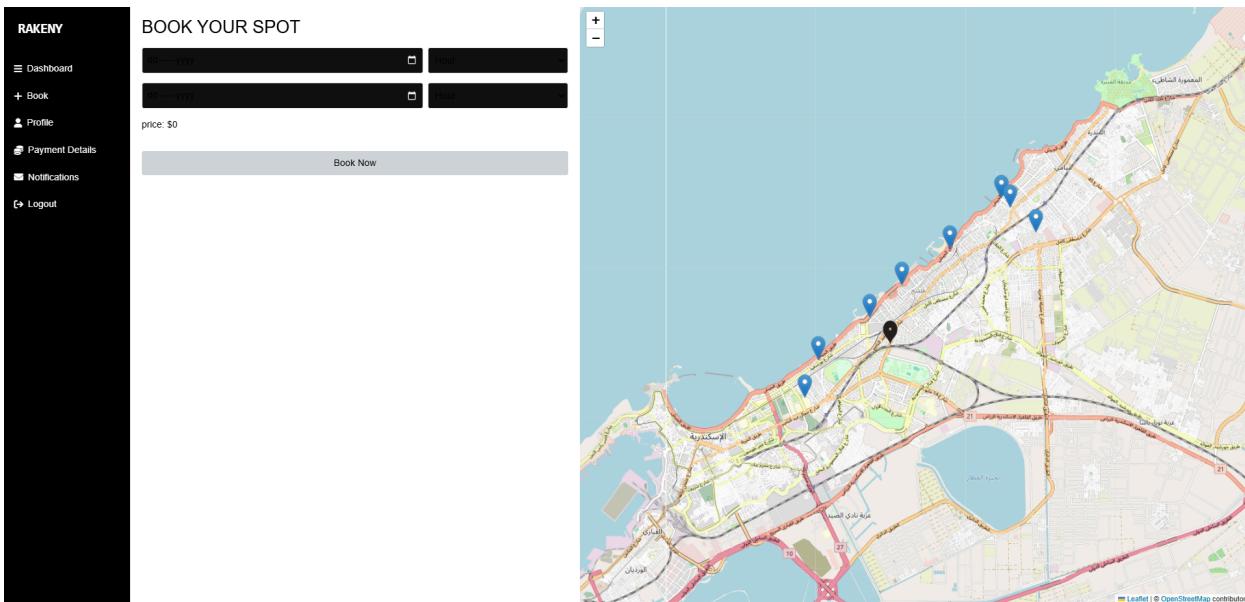
Don't have an account? [Sign up](#)

Reset Password Form:



The image shows a dark-themed web page with a white header bar. On the left is a circular logo containing a white 'P' with a small car icon inside. On the right are two buttons: 'Login' and 'Sign Up'. Below the header is a white modal window titled 'Reset Password' with the sub-instruction 'Enter your new password'. It contains two input fields: 'Enter New Password' and 'Confirm New Password', each with a small eye icon to the right. At the bottom of the modal is a black 'Reset' button.

Main Page:



The image displays the main application interface. On the left is a vertical black sidebar menu with white text and icons. It includes 'RAKENY' at the top, followed by 'Dashboard', 'Book', 'Profile', 'Payment Details', 'Notifications', and 'Logout'. The main content area has a light gray background. At the top left is a section titled 'BOOK YOUR SPOT' with two dropdown menus for selecting dates ('dd--mm--yyyy') and times ('hour'). Below this is a text field showing 'price: \$0' and a large 'Book Now' button. To the right is a map of a coastal city area, showing roads, landmarks, and several blue location markers. A zoom control with '+' and '-' buttons is located in the top-left corner of the map area. The map also includes Arabic place names like 'المنطقة الصناعية' and 'حي المنشآت'.

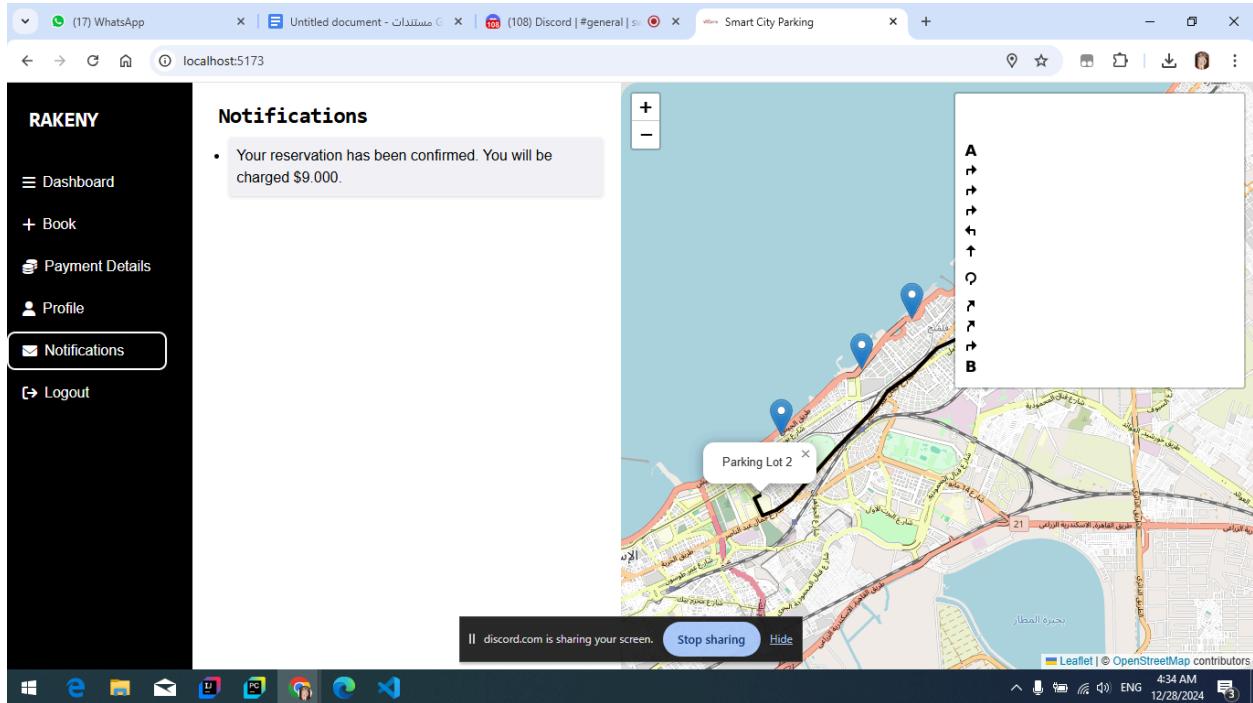
Reserving :

The screenshot shows the 'RAKENY' parking booking interface. On the left, a sidebar menu includes 'Dashboard', '+ Book', 'Payment Details', 'Profile', 'Notifications', and 'Logout'. The main area is titled 'BOOK YOUR SPOT' and displays two date/time input fields. Below this, a message says 'price: \$0'. A grid of 15 parking spot icons (S001-S035) is shown, all labeled 'AVAILABLE'. To the right is a map of a coastal city area with a parking lot highlighted. A callout box on the map points to 'Parking Lot 2'. The bottom status bar shows system information: 'Leaflet | © OpenStreetMap contributors', '4:33 AM', 'ENG', and the date '12/28/2024'.

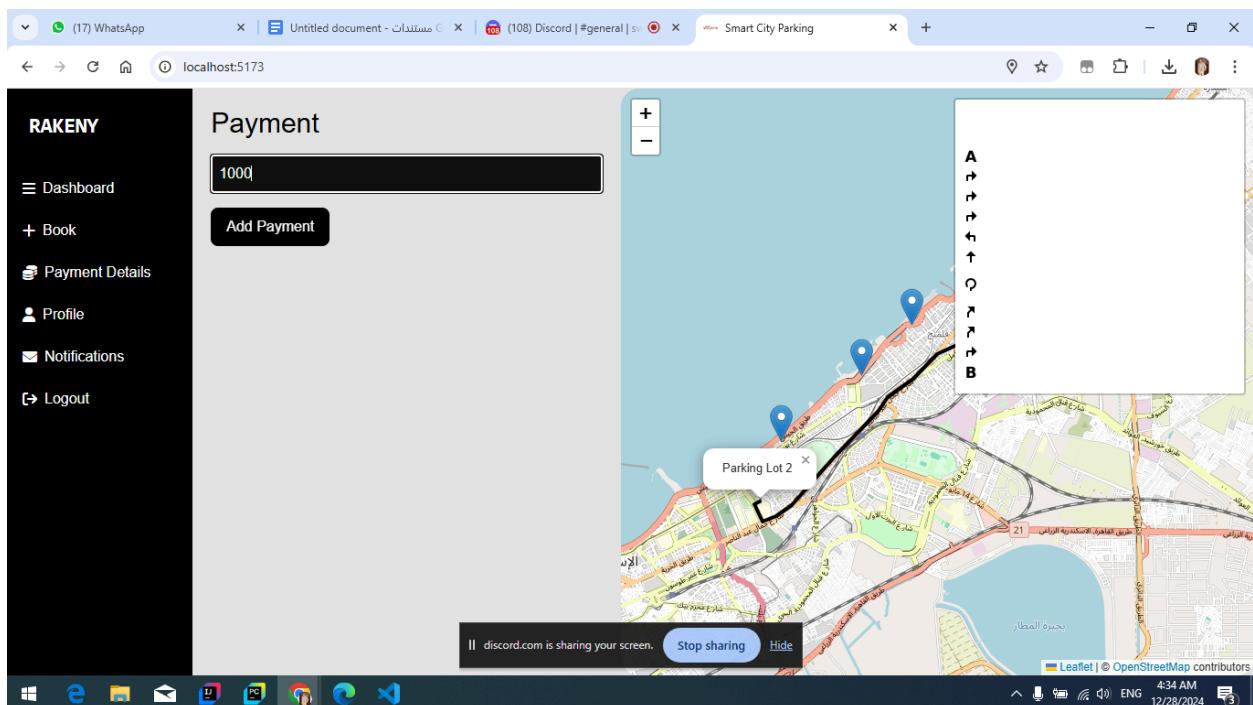
Reservations :

The screenshot shows the 'RAKENY' parking booking confirmation interface. The sidebar menu is identical to the previous screen. The main area now shows a large empty box with three buttons at the bottom: 'Edit', 'Cancel', and 'I am there'. To the right is the same map as before, with a callout box pointing to 'Parking Lot 2'. A notification at the bottom of the screen states 'discord.com is sharing your screen.' with 'Stop sharing' and 'Hide' buttons. The bottom status bar shows 'Leaflet | © OpenStreetMap contributors', '4:34 AM', 'ENG', and the date '12/28/2024'.

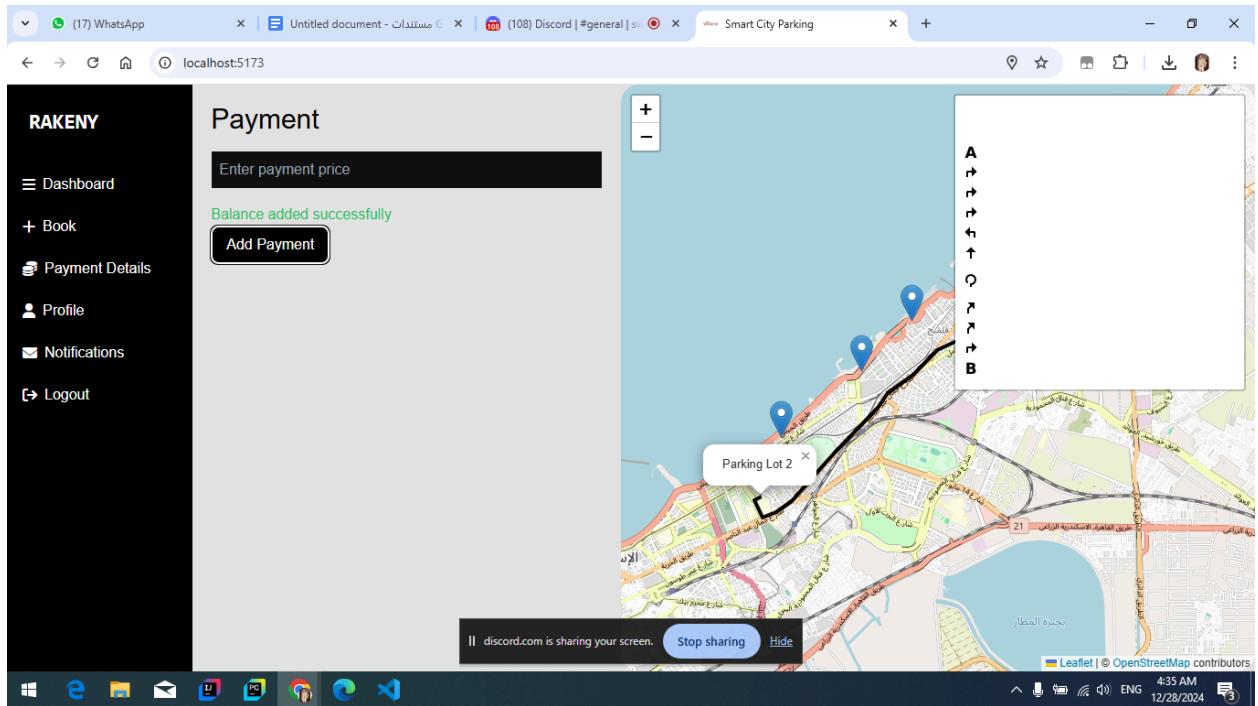
Notifications :



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.