Backend Test Scenarios for Parking Spots

Test Scenario 1: Successful Parking Spot Registration

**Objective**: Validate that a new parking spot can be registered successfully when all required fields are provided correctly.

**Preconditions**:

* No existing parking spot with the same vehicle plate.

**Steps**:

1. Send a **POST** request to the /register\_parking\_spot endpoint with the following JSON payload:
2. {
3. "user\_id": 1,
4. "vehicle\_plate": "ABC1234",
5. "spot\_number": "101",
6. "apartment": "A1",
7. "block": "B1"
8. }

Copiar

**Expected Results**:

* The response status should be 201 (Created).
* The JSON response should include a success message: {"message": "Veículo cadastrado com sucesso!"}.

Test Scenario 2: Duplicate Vehicle Plate Registration Attempt

**Objective**: Ensure the registration fails if the vehicle plate already exists in the database.

**Preconditions**:

* A parking spot already exists with the vehicle plate "ABC1234".

**Steps**:

1. Send a **POST** request to the /register\_parking\_spot endpoint with a JSON payload having the same vehicle plate:
2. {
3. "user\_id": 2,
4. "vehicle\_plate": "ABC1234",
5. "spot\_number": "102",
6. "apartment": "A2",
7. "block": "B2"
8. }

Copiar

**Expected Results**:

* The response status should be 400 (Bad Request).
* The JSON response should contain an error message: {"message": "Veículo já cadastrado!"}.

Test Scenario 3: Retrieve All Parking Spots

**Objective**: Validate that all parking spots can be retrieved successfully.

**Preconditions**:

* At least one parking spot must exist in the database.

**Steps**:

1. Send a **GET** request to the /parking\_spots endpoint.

**Expected Results**:

* The response status should be 200 (OK).
* The response should provide a JSON array containing all parking spot details (id, vehicle\_plate, spot\_number, apartment, block, user\_id).

Test Scenario 4: Update Parking Spot Details

**Objective**: Validate that an existing parking spot can be updated successfully.

**Preconditions**:

* A parking spot exists with id = 1.

**Steps**:

1. Send a **PUT** request to the /parking\_spot/1 endpoint with the following JSON payload to update details:
2. {
3. "vehicle\_plate": "XYZ9876",
4. "spot\_number": "103",
5. "apartment": "A3",
6. "block": "B3"
7. }

Copiar

**Expected Results**:

* The response status should be 200 (OK).
* The JSON response should include a success message: {"message": "Vaga atualizada com sucesso!"}.

Test Scenario 5: Delete a Parking Spot

**Objective**: Validate that a parking spot can be deleted successfully.

**Preconditions**:

* A parking spot exists with id = 1.

**Steps**:

1. Send a **DELETE** request to the /delete\_parking\_spot/1 endpoint.

**Expected Results**:

* The response status should be 200 (OK).
* The JSON response should include a success message: {"message": "Vaga deletada com sucesso!"}.

Step-by-Step Execution Using Postman

For each test scenario, the execution steps in Postman are:

1. **Open Postman**: Ensure you have Postman installed and open it.
2. **Start a New Request**: Click on "New" and select "Request" from the dropdown.
3. **Choose Method**: Select the appropriate HTTP method (POST, GET, PUT, DELETE) based on the scenario.
4. **Enter URL**: Input the local URL of your endpoint (http://localhost:<porta>/<endpoint>). Replace <porta> with your actual port number (default is often 5000).
5. **Configure Headers**: For POST and PUT requests, set Content-Type to application/json in Headers.
6. **Body Content**: For POST and PUT requests, choose Raw and JSON in the Body tab, and input the scenario-specific JSON payload.
7. **Send Request**: Click "Send" and observe the result.
8. **Validate Response**: Check the Status code and Body response against expected outcomes for each scenario.