Defining Query Methods - Assignment

Assignment: Implement AppController to Fetch Data from API

Objective

In this project, you will implement the key components of a simple e-commerce system using Spring Boot, focusing on the use of **Spring Data JPA** to define query methods for database operations. The system will involve creating and retrieving products from a database using custom query methods. You will need to implement the controller, entity, and repository layers of the project.

You will be given a template project with the controller fine, entity file, and repository file blanked out. Your task is to implement these components while following the instructions. The primary goal of this project is to get familiar with how Spring Boot integrates with JPA to define and execute custom queries for data access.

Project Setup

Your project structure is:
- **Main Application Class**: The main entry point of the Spring Boot application.
- **Controller Class**: A REST controller that will handle HTTP requests related to products.
- **Entity Class**: A JPA entity class that represents the `Product` entity in the database.
- **Repository Interface**: A Spring Data JPA repository interface to manage database operations for `Product`.

Key Components to Implement

1. Controller Class

In the `ProductController` class, you need to implement the following functionality:

- Class must have proper annotation to make it a rest controller.
- **Autowired Dependency**: The `ProductRepository` should be injected into the `ProductController` to interact with the database.

- **Method 1: Create Product (POST)**
  - **Method Name**: `createProduct`
  - **Return Type**: `Product`
  - **Parameters**: A `Product` object passed in the body of the HTTP POST request.

- **HTTP Method and Endpoint**: The method should handle HTTP POST requests to the `/api/products` endpoint.
- **Functionality**: Saves the product using the save method of the repository and returns the saved product.

- **Method 2: Find Products by Name (GET)**
- **Method Name**: `findByName`
- **Return Type**: `List<Product>`
- **Parameters**: A `name` query parameter to search for products by name.
- **HTTP Method and Endpoint**: Handles GET requests to `/api/products/search/findByName`.
- **Functionality**: Returns a list of products with the given name using findByName method of repository.

- **Method 3: Find Products by Price Range (GET)**
- **Method Name**: `findByPriceBetween`
- **Return Type**: `List<Product>`
- **Parameters**: `minPrice` and `maxPrice` query parameters.
- **HTTP Method and Endpoint**: Handles GET requests to `/api/products/search/findByPriceBetween`.
- **Functionality**: Returns a list of products within the specified price range using findByPriceBetween of repository.

2. Entity Class

In the `Product` entity class, you need to implement the following:
1. **Entity Annotation**: Add the appropriate annotation to the class to mark it as a JPA entity, which will map to a database table.
2. **Primary Key**: Add an annotation to the `id` field to mark it as the primary key, and set it to auto-generate its value using Identity technique.

3. Repository Interface

In the `ProductRepository` interface, you need to implement the following:
1. **Repository Annotation**: Ensure the interface is marked as a Spring Data JPA repository.
3. **Custom Query Methods**:
    - **Find by Name**: Implement a method to retrieve products by their name. The method should be named `findByName` and accept a `String name` parameter.
    - **Find by Price Range**: Implement a method to retrieve products within a specified price range. The method should be named `findByPriceBetween` and accept `Double minPrice` and `Double maxPrice` parameters.

Execution Steps to Follow

1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**
2. **To open the command terminal the test takers need to go to the Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal.**
3. **cd into your backend project folder.**
4. **To build your project use command:**

   mvn clean package -Dmaven.test.skip
5. **To launch your application, move into the target folder (cd target). Run the following command to run the application:**

   java -jar <your application jar file name>
6. **To test any Restful application, the last option on the left panel of IDE, you can find ThunderClient, which is the lightweight equivalent of POSTMAN. Please use 127.0.0.1 instead of localhost to test rest endpoints.**
7. **Mandatory: Before final submission run the following command:**

   mvn test
8. **You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.**