**Exercise 6: Employee Management System – Implementing Pagination and Sorting**

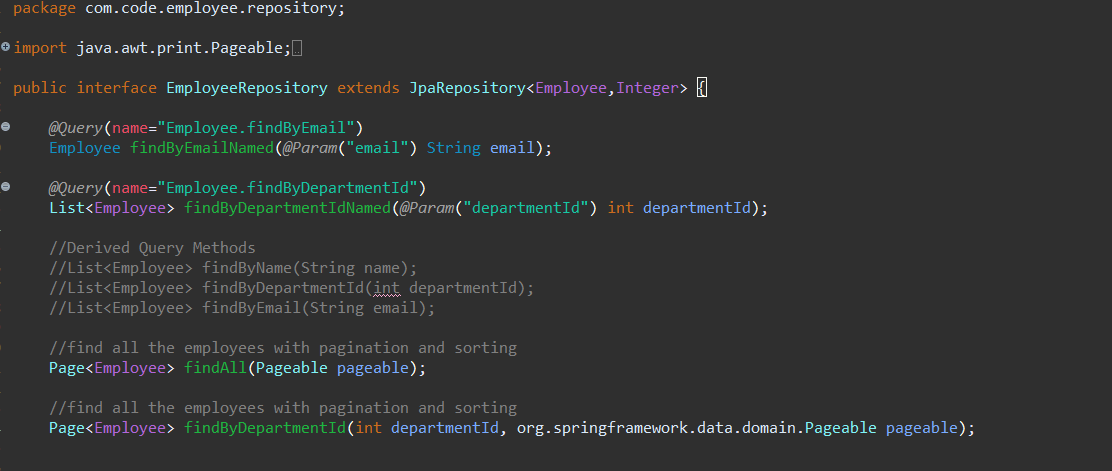
Business Scenario:

Add pagination and sorting capabilities to your employee search functionality.

**1. Pagination:**

* Implement pagination for the employee list using **Page** and **Pageable**

In this exercise I modified the existing code. Pagination allows you to divide the results into pages instead of loading all records at once. Spring Data JPA provides an easy way to implement this using Page and Pageable.



**2. Sorting:**

I added sorting functionality to my queries

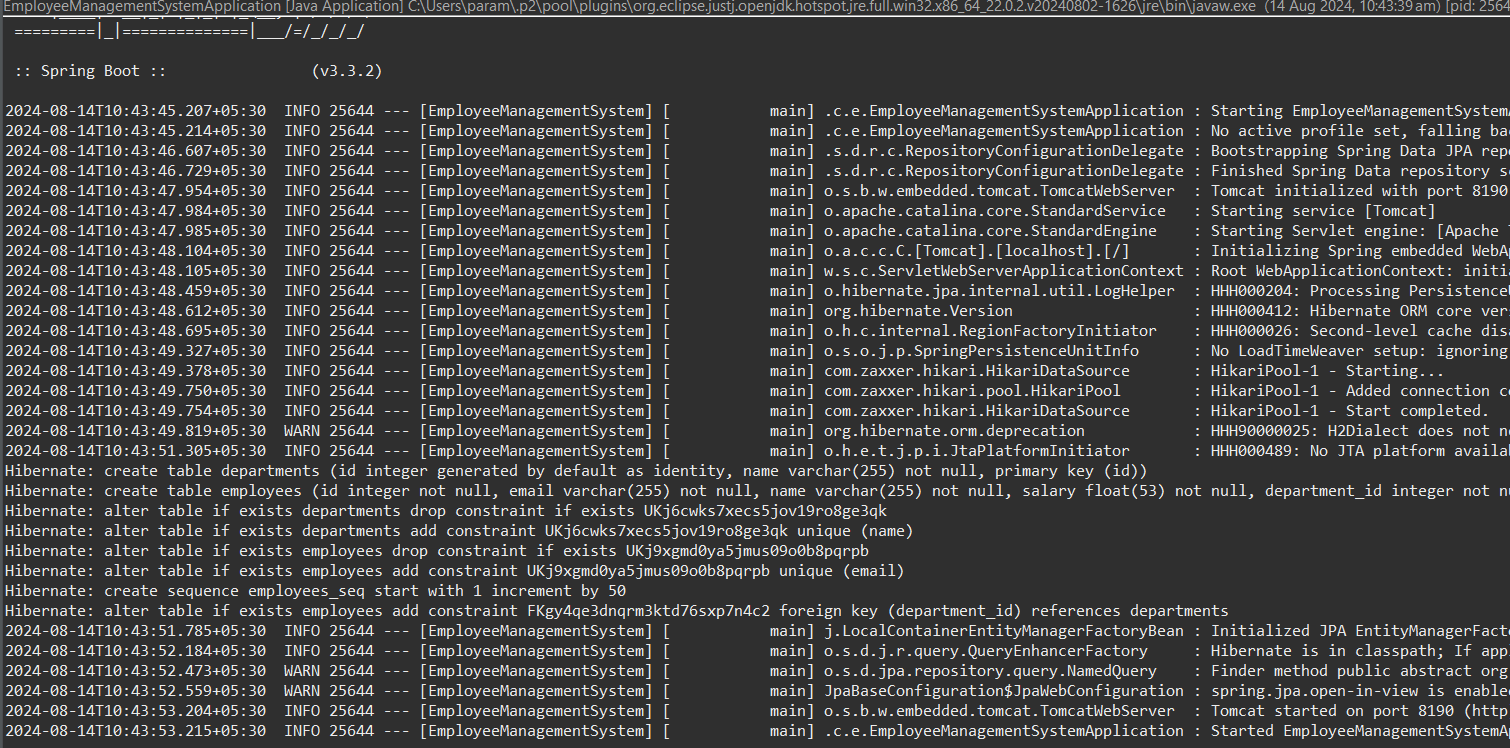
**sortBy**: The field to sort by (e.g., name, salary, department)

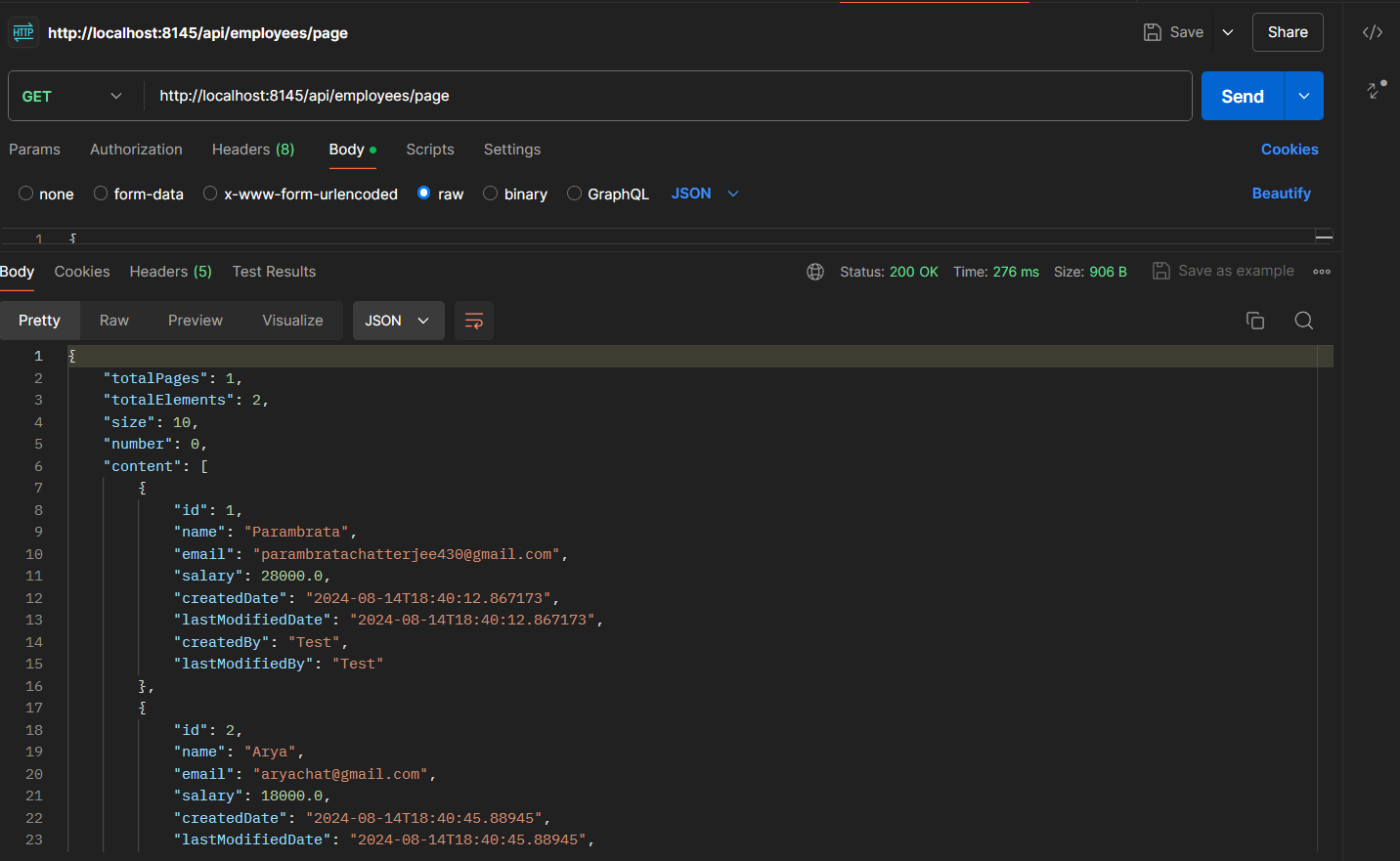
**sortDirection**: The sorting direction (asc for ascending, desc for descending).

Finally, I combined pagination and sorting in the search endpoints.

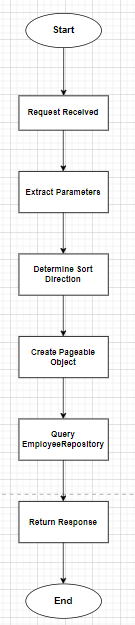


**Output:**

****

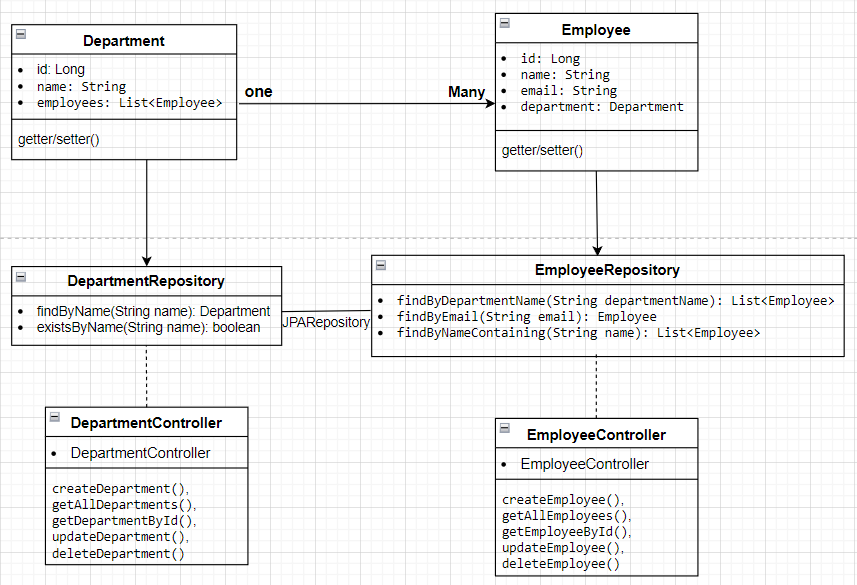


**Flowchart:**

****

1. **Start** - Begin the flowchart.
2. **Request Received** - Represent the API request received at the EmployeeController.
3. **Extract Parameters** - Extract query parameters (page, size, sortBy, sortDirection).
4. **Determine Sort Direction** - Decide if the sorting direction is ascending or descending.
5. **Create Pageable Object** - Create a Pageable object using PageRequest.of() with page, size, and sort parameters.
6. **Query EmployeeRepository** - Query the EmployeeRepository with the Pageable object to get a paginated and sorted list of employees.
7. **Return Response** - Send the paginated and sorted list of employees back as the response.
8. **End** - The process ends.

**Class Diagram:**

****

** Department Class:**

* **Attributes: Represents the id, name, and a list of Employee objects.**
* **Relationship: Has a one-to-many relationship with Employee.**

** Employee Class:**

* **Attributes: Represents the id, name, email, and a reference to the Department object.**
* **Relationship: Each Employee belongs to a single Department.**

** EmployeeRepository Interface:**

* **Methods:**
  + **findByDepartmentName(String departmentName): List<Employee>**
  + **findByEmail(String email): Employee**
  + **findByNameContaining(String name): List<Employee>**

** DepartmentRepository Interface:**

* **Methods:**
  + **findByName(String name): Department**
  + **existsByName(String name): Boolean**

** EmployeeController Class:**

* **Methods:**
* **createEmployee()**
* **getAllEmployees()**
* **getEmployeeById()**
* **updateEmployee()**
* **deleteEmployee().**

** DepartmentController Class:**

* **Methods**
* **createDepartment()**
* **getAllDepartments()**
* **getDepartmentById()**
* **updateDepartment()**
* **deleteDepartment()**

**Analysis:**

**1. Pagination with Page and Pageable**

Pagination allows you to divide the results into pages instead of loading all records at once. Spring Data JPA provides an easy way to implement this using Page and Pageable.

**Modify EmployeeRepository**

In the EmployeeRepository interface, I added a method to fetch a paginated list of employees.

**Update EmployeeController**

Now, update the EmployeeController to handle pagination.

 **page**: The page number (0-indexed).

 **size**: The number of records per page.

**2. Sorting**

We can also sort the employee list based on specific fields.

**1: Modify EmployeeRepository**

The findAll(Pageable pageable) method already supports sorting through the Pageable object.

**Update EmployeeController to include sorting**

Update the getAllEmployees method to support sorting.

 **sortBy**: The field to sort by (e.g., name, salary, department).

 **sortDirection**: The sorting direction (asc for ascending, desc for descending).

**Combining Pagination and Sorting in the Search Endpoint**

The search endpoint now handles both pagination and sorting.

**Relationship:**

* A Department can have many Employees (OneToMany relationship).
* An Employee belongs to one Department (ManyToOne relationship).
* Department to DepartmentRepository
* Employee to EmployeeRepository