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

To implement pagination and sorting in an Employee Management System, you typically use Spring Data JPA if you're working with a Java-based backend. Here's a step-by-step guide to help you

implement these features.

1. **Setup and Dependencies**

Ensure that you have the following dependencies in your pom.xml (if using Maven):

<dependencies>

<!-- Other dependencies -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

</dependencies>

1. **Entity Class**

Assume you have an Employee entity:

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Employee { @Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private Long id;

private String name;

private String department; private double salary;

// Getters and Setters

}

1. **Repository Interface**

Your repository interface should extend JpaRepository, which provides methods for pagination and sorting:

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

}

1. **Service Layer**

In the service layer, create a method to fetch paginated and sorted results:

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Pageable;

import org.springframework.data.domain.Sort;

import org.springframework.stereotype.Service;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

public Page<Employee> getAllEmployees(int page, int size, String sortBy, String sortDir) {

Sort sort = sortDir.equalsIgnoreCase(Sort.Direction.ASC.name()) ? Sort.by(sortBy).ascending()

: Sort.by(sortBy).descending();

Pageable pageable = PageRequest.of(page, size, sort);

return employeeRepository.findAll(pageable);

}

}

1. **Controller Layer**

Expose the service in a REST controller:

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class EmployeeController {

@Autowired

private EmployeeService employeeService;

@GetMapping("/employees")

public Page<Employee> getEmployees(

@RequestParam(defaultValue = "0") int page, @RequestParam(defaultValue = "10") int size,

@RequestParam(defaultValue = "id") String sortBy,

@RequestParam(defaultValue = "asc") String sortDir) {

return employeeService.getAllEmployees(page, size, sortBy, sortDir);

}

}

1. **Testing the Endpoint**

You can test the endpoint by hitting the following URL:

http://localhost:8080/employees?page=0&size=5&sortBy=name&sortDir=desc

This will return the first page of employees, with 5 employees per page, sorted by name in descending order.

1. **Handling Edge Cases (Optional)**
   * **Validation:** Validate the page, size, sortBy, and sortDir parameters to ensure they are within acceptable ranges/values.
   * **Error Handling:** Implement error handling to manage scenarios like invalid sorting fields or directions.
2. **Front-End Considerations**

If you're working with a front-end, you'll typically display pagination controls (like "Next", "Previous", and page numbers) and allow the user to choose sorting options.