Exercise 6: Implementing Pagination and Sorting

Understanding the Requirements

We'll enhance the employee search functionality by adding pagination and sorting capabilities using Spring Data JPA's `Pageable` and `Sort` interfaces.

Modifying the Repository

Java

```java
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

public interface  EmployeeRepository extends JpaRepository<Employee,
Long> {
    Page<Employee> findAll(Pageable pageable);

    @Query("SELECT e FROM Employee  e WHERE e.department.name =
:departmentName")
    Page<Employee> getEmployeesByDepartmentName(@Param("departmentName")
String departmentName, Pageable pageable);
}
```

Modifying the Controller

Java

```java
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Sort;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
```

```
@RestController
@RequestMapping("/employees")
public class EmployeeController {

    @Autowired
    private EmployeeRepository employeeRepository;

    @GetMapping
    public Page<Employee>   getEmployees(
            @RequestParam(value = "page", defaultValue = "0") Integer
page,
            @RequestParam(value = "size", defaultValue = "10") Integer
size,
            @RequestParam(value = "sort", required = false)   String sort)
{

        Sort sortBy = null;
        if (sort != null) {
            sortBy = Sort.by(sort);
        }

        Pageable pageable = PageRequest.of(page, size, sortBy);
        return employeeRepository.findAll(pageable);
    }
}
```

Explanation

- **Pageable Interface:** Represents pagination information, including page number, page size, and sorting.
- **Page Interface:** Represents a page of data.
- **Sort Interface:** Represents sorting information, including sort properties and directions.
- **PageRequest:** A concrete implementation of Pageable.
- **@RequestParam:** Used to bind query parameters to method parameters.

Additional Considerations

- **Custom Sorting:** You can create custom sorting logic using `Sort.by` with multiple fields and directions.
- **Pagination and Sorting in Custom Queries:** The `getEmployeesByDepartmentName` method demonstrates how to apply pagination and sorting to custom queries.
- **Error Handling:** Handle potential errors, such as invalid page numbers or sort properties.
- **Performance Optimization:** Consider using appropriate data structures and indexes for efficient pagination and sorting.