Exercise 1: Employee Management System - Overview and Setup

1. Creating a Spring Boot Project

Use Spring Initializr:

- Open [Spring Initializr](https://start.spring.io/) in your browser.
- Fill in the project details:
  - Project: Maven Project
  - Language: Java
  - Spring Boot: Choose the latest stable version
  - Project Metadata:
    - Group: `com.example`
    - Artifact: `EmployeeManagementSystem`
    - Name: `EmployeeManagementSystem`
    - Package Name: `com.example.employeemanagementsystem`
    - Packaging: Jar
  - Add Dependencies:
    - Spring Web
    - Spring Data JPA
    - H2 Database

2. Configuring Application Properties

application.properties –

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

# Enable H2 Console (optional, useful for debugging)
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
# JPA Hibernate settings
spring.jpa.hibernate.ddl-auto=update
```

Exercise 2: Employee Management System - Creating Entities

1. Creating JPA Entities

Create the `Employee` and `Department` entities in the `com.example.employeemanagementsystem.model` package.

1. Employee Entity:

Java -

```
package com.example.employeemanagementsystem.model;
import jakarta.persistence.*;
import lombok.Data;

@Data
@Entity
@Table(name = "employees")
public class Employee {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;
   private String name;
   private String email;
   @ManyToOne
   @JoinColumn(name = "department_id")
   private Department department;
}
```

2. Department Entity:

Java-

```
package com.example.employeemanagementsystem.model;
import jakarta.persistence.*;
import lombok.Data;
import java.util.List;
```

```
@Data
@Entity
@Table(name = "departments")
public class Department {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    @OneToMany(mappedBy = "department", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private List<Employee> employees;
}
```

Create JPA repositories for the entities to perform CRUD operations.

1. Employee Repository:

Java-

```
package com.example.employeemanagementsystem.repository;
import com.example.employeemanagementsystem.model.Employee;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
}
```

2. Department Repository:

Java-

```
package com.example.employeemanagementsystem.repository;
import com.example.employeemanagementsystem.model.Department;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
@Repository
public interface DepartmentRepository extends JpaRepository<Department, Long> {
}
```

Implementing Services

Create services to handle business logic for the entities.

1. Employee Service:

Java-

```
package com.example.employeemanagementsystem.service;
import com.example.employeemanagementsystem.model.Employee;
import com.example.employeemanagementsystem.repository.EmployeeRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
import java.util.Optional;

@Service
public class EmployeeService {

    @Autowired
    private EmployeeRepository employeeRepository;

    public List<Employee> getAllEmployees() {
        return employeeRepository.findAll();
    }

    public Optional<Employee> getEmployeeById(Long id) {
        return employeeRepository.findById(id);
    }

    public Employee saveEmployee(Employee employee) {
        return employeeRepository.save(employee);
    }

    public void deleteEmployee(Long id) {
        employeeRepository.deleteById(id);
    }
}
```

2. Department Service:

Java-

```
package com.example.employeemanagementsystem.service;
import com.example.employeemanagementsystem.model.Department;
import com.example.employeemanagementsystem.repository.DepartmentRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class DepartmentService {
    @Autowired
    private DepartmentRepository departmentRepository;

    public List<Department> getAllDepartments() {
        return departmentRepository.findAll();
    }

    public Optional<Department> getDepartmentById(Long id) {
        return departmentRepository.findById(id);
    }

    public Department saveDepartment(Department department) {
        return departmentRepository.save(department);
    }

    public void deleteDepartment(Long id) {
        departmentRepository.deleteById(id);
    }
}
```

Implementing Controllers

Create REST controllers to expose endpoints for managing employees and departments.

1. Employee Controller”

Java-

```
package com.example.employeemanagementsystem.controller;
import com.example.employeemanagementsystem.model.Employee;
import com.example.employeemanagementsystem.service.EmployeeService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/employees")
public class EmployeeController {
    @Autowired
    private EmployeeService employeeService;

    @GetMapping
    public List<Employee> getAllEmployees() {
        return employeeService.getAllEmployees();
    }

    @GetMapping("/{id}")
    public ResponseEntity<Employee> getEmployeeById(@PathVariable Long id) {
        Optional<Employee> employee = employeeService.getEmployeeById(id);
        return employee.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());
    }

    @PostMapping
    public Employee createEmployee(@RequestBody Employee employee) {
```

```
        return employeeService.saveEmployee(employee);
    }

    @PutMapping("/{id}")
    public ResponseEntity<Employee> updateEmployee(@PathVariable Long id, @RequestBody
Employee employeeDetails) {
        Optional<Employee> employee = employeeService.getEmployeeById(id);
        if (employee.isPresent()) {
            Employee updatedEmployee = employee.get();
            updatedEmployee.setName(employeeDetails.getName());
            updatedEmployee.setEmail(employeeDetails.getEmail());
            updatedEmployee.setDepartment(employeeDetails.getDepartment());
            return ResponseEntity.ok(employeeService.saveEmployee(updatedEmployee));
        } else {
            return ResponseEntity.notFound().build();
        }
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteEmployee(@PathVariable Long id) {
        employeeService.deleteEmployee(id);
        return ResponseEntity.noContent().build();
    }
}
```

2. Department Controller:

Java-

```
package com.example.employeemanagementsystem.controller;
import com.example.employeemanagementsystem.model.Department;
import com.example.employeemanagementsystem.service.DepartmentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
```

```
import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/departments")
public class DepartmentController {

    @Autowired
    private DepartmentService departmentService;

    @GetMapping
    public List<Department> getAllDepartments() {
        return departmentService.getAllDepartments();
    }

    @GetMapping("/{id}")
    public ResponseEntity<Department> getDepartmentById(@PathVariable Long id) {
        Optional<Department> department = departmentService.getDepartmentById(id);
        return department.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());
    }

    @PostMapping
    public Department createDepartment(@RequestBody Department department) {
        return departmentService.saveDepartment(department);
    }

    @PutMapping("/{id}")
    public ResponseEntity<Department> updateDepartment(@PathVariable Long id, @RequestBody
Department departmentDetails) {
        Optional<Department> department = departmentService.getDepartmentById(id);
        if (department.isPresent()) {
            Department updatedDepartment = department.get();
            updatedDepartment.setName(departmentDetails.getName());
            return ResponseEntity.ok(departmentService.saveDepartment(updatedDepartment));
```

```
            } else {
                return ResponseEntity.notFound().build();
            }
        }

        @DeleteMapping("/{id}")
        public ResponseEntity<Void> deleteDepartment(@PathVariable Long id) {
            departmentService.deleteDepartment(id);
            return ResponseEntity.noContent().build();
        }
    }
```

Testing the Application

1. Run the Application:
   - Execute the `main` method in `EmployeeManagementSystemApplication` class.
2. Access the H2 Console:
   - Go to `http://localhost:8080/h2-console`
     - JDBC URL: `jdbc:h2:mem:testdb`
     -Username: `sa`
     - Password: `password`
3. Test Endpoints:

`/employees`: Retrieve all employees.

- GET `/employees/{id}`: Retrieve an employee by ID.
- POST `/employees`: Create a new employee.
- PUT `/employees/{id}`: Update an employee.
- DELETE `/employees/{id}`: Delete an employee.
- GET `/departments`: Retrieve all departments.
- GET `/departments/{id}`: Retrieve a department by ID.
- POST `/departments`: Create a new department.
- PUT `/departments/{id}`: Update a department.
- DELETE `/departments/{id}`: Delete a department.

This setup should give you a working Employee Management System with basic CRUD operations for employees and departments.

Exercise 3: Employee Management System - Creating Repositories

1. Overview of Spring Data Repositories

Benefits of using Spring Data Repositories:

- Simplicity : Spring Data repositories reduce boilerplate code by providing a set of default methods for performing CRUD operations on entities.
- Consistency : By using repository interfaces, you ensure consistent data access patterns across your application.
- Derived Query Methods : Spring Data provides the ability to define custom queries by simply declaring method signatures in repository interfaces.
- Support for Pagination and Sorting : Repositories come with built-in support for pagination and sorting of results.

2. Creating Repositories

Create interfaces for `EmployeeRepository` and `DepartmentRepository` extending `JpaRepository`.

1. Employee Repository:

Java-

```
package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.model.Employee;

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

import org.springframework.stereotype.Repository;

import java.util.List;


@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

   // Derived query method to find employees by department name

   List<Employee> findByDepartmentName(String departmentName);


   // Derived query method to find employees by name

   List<Employee> findByNameContainingIgnoreCase(String name);

}
```

2. Department Repository:

Java-

```
package com.example.employeemanagementsystem.repository;
import com.example.employeemanagementsystem.model.Department;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface DepartmentRepository extends JpaRepository<Department, Long> {
    Department findByName(String name);
}
```

Exercise 4: Employee Management System - Implementing CRUD Operations

1. Basic CRUD Operations

Employee Service :

Java-

```
package com.example.employeemanagementsystem.service;
import com.example.employeemanagementsystem.model.Employee;
import com.example.employeemanagementsystem.repository.EmployeeRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
import java.util.Optional;
@Service
public class EmployeeService {

    @Autowired
    private EmployeeRepository employeeRepository;

    public List<Employee> getAllEmployees() {
        return employeeRepository.findAll();
    }

    public Optional<Employee> getEmployeeById(Long id) {
        return employeeRepository.findById(id);
    }

    public Employee saveEmployee(Employee employee) {
        return employeeRepository.save(employee);
    }

    public Employee updateEmployee(Long id, Employee employeeDetails) {
        return employeeRepository.findById(id).map(employee -> {
```

```
            employee.setName(employeeDetails.getName());
            employee.setEmail(employeeDetails.getEmail());
            employee.setDepartment(employeeDetails.getDepartment());
            return employeeRepository.save(employee);
        }).orElseThrow(() -> new RuntimeException("Employee not found with id " + id));
    }

    public void deleteEmployee(Long id) {
        employeeRepository.deleteById(id);
    }

    public List<Employee> getEmployeesByDepartmentName(String departmentName) {
        return employeeRepository.findByDepartmentName(departmentName);
    }

    public List<Employee> searchEmployeesByName(String name) {
        return employeeRepository.findByNameContainingIgnoreCase(name);
    }
}
```

Department Service :

Java-

```
package com.example.employeemanagementsystem.service;
import com.example.employeemanagementsystem.model.Department;
import com.example.employeemanagementsystem.repository.DepartmentRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
import java.util.Optional;

@Service
public class DepartmentService {

    @Autowired
```

```
    private DepartmentRepository departmentRepository;

    public List<Department> getAllDepartments() {
        return departmentRepository.findAll();
    }

    public Optional<Department> getDepartmentById(Long id) {
        return departmentRepository.findById(id);
    }

    public Department saveDepartment(Department department) {
        return departmentRepository.save(department);
    }

    public Department updateDepartment(Long id, Department departmentDetails) {
        return departmentRepository.findById(id).map(department -> {
            department.setName(departmentDetails.getName());
            return departmentRepository.save(department);
        }).orElseThrow(() -> new RuntimeException("Department not found with id " + id));
    }

    public void deleteDepartment(Long id) {
        departmentRepository.deleteById(id);
    }

    public Department getDepartmentByName(String name) {
        return departmentRepository.findByName(name);
    }
}
```

2. Implement RESTful Endpoints

Employee Controller:

Java-

```
package com.example.employeemanagementsystem.controller;
import com.example.employeemanagementsystem.model.Employee;
import com.example.employeemanagementsystem.service.EmployeeService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.List;
import java.util.Optional;

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

    @Autowired
    private EmployeeService employeeService;

    @GetMapping
    public List<Employee> getAllEmployees() {
        return employeeService.getAllEmployees();
    }

    @GetMapping("/{id}")
    public ResponseEntity<Employee> getEmployeeById(@PathVariable Long id) {
        Optional<Employee> employee = employeeService.getEmployeeById(id);
        return employee.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());
    }

    @PostMapping
    public Employee createEmployee(@RequestBody Employee employee) {
        return employeeService.saveEmployee(employee);
```

```
    }
    @PutMapping("/{id}")
    public ResponseEntity<Employee> updateEmployee(@PathVariable Long id, @RequestBody Employee
employeeDetails) {
        try {
            Employee updatedEmployee = employeeService.updateEmployee(id, employeeDetails);
            return ResponseEntity.ok(updatedEmployee);
        } catch (RuntimeException e) {
            return ResponseEntity.notFound().build();
        }
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteEmployee(@PathVariable Long id) {
        employeeService.deleteEmployee(id);
        return ResponseEntity.noContent().build();
    }

    @GetMapping("/search")
    public List<Employee> searchEmployeesByName(@RequestParam String name) {
        return employeeService.searchEmployeesByName(name);
    }

    @GetMapping("/department/{departmentName}")
    public List<Employee> getEmployeesByDepartment(@PathVariable String departmentName) {
        return employeeService.getEmployeesByDepartmentName(departmentName);
    }
}
```

Department Controller:

Java-

```
package com.example.employeemanagementsystem.controller;
import com.example.employeemanagementsystem.model.Department;
import com.example.employeemanagementsystem.service.DepartmentService;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.List;
import java.util.Optional;
@RestController
@RequestMapping("/departments")
public class DepartmentController {

    @Autowired
    private DepartmentService departmentService;

    @GetMapping
    public List<Department> getAllDepartments() {
        return departmentService.getAllDepartments();
    }

    @GetMapping("/{id}")
    public ResponseEntity<Department> getDepartmentById(@PathVariable Long id) {
        Optional<Department> department = departmentService.getDepartmentById(id);
        return department.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());
    }

    @PostMapping
    public Department createDepartment(@RequestBody Department department) {
        return departmentService.saveDepartment(department);
    }

    @PutMapping("/{id}")
    public ResponseEntity<Department> updateDepartment(@PathVariable Long id, @RequestBody
Department departmentDetails) {
        try {
            Department updatedDepartment = departmentService.updateDepartment(id, departmentDetails);
            return ResponseEntity.ok(updatedDepartment);
```

```
            } catch (RuntimeException e) {
                return ResponseEntity.notFound().build();
            }
        }

        @DeleteMapping("/{id}")
        public ResponseEntity<Void> deleteDepartment(@PathVariable Long id) {
            departmentService.deleteDepartment(id);
            return ResponseEntity.noContent().build();
        }

        @GetMapping("/name/{name}")
        public ResponseEntity<Department> getDepartmentByName(@PathVariable String name) {
            Department department = departmentService.getDepartmentByName(name);
            if (department != null) {
                return ResponseEntity.ok(department);
            } else {
                return ResponseEntity.notFound().build();
            }
        }
    }
```

Testing the Application :

1. Use Postman or cURL:
   - Test the RESTful endpoints for employees and departments:
     - `GET /employees` - Retrieve all employees.
     - `GET /employees/{id}` - Retrieve an employee by ID.
     - `POST /employees` - Create a new employee.
     - `PUT /employees/{id}` - Update an existing employee.
     - `DELETE /employees/{id}` - Delete an employee.
     - `GET /employees/search?name={name}` - Search employees by name.
     - `GET /employees/department/{departmentName}` - Get employees by department name.
     - `GET /departments` - Retrieve all departments.

- `GET /departments/{id}` - Retrieve a department by ID.
- `POST /departments` -

Create a new department.

- `PUT /departments/{id}` - Update an existing department.
- `DELETE /departments/{id}` - Delete a department.
- `GET /departments/name/{name}` - Retrieve a department by name.

Exercise 5: Employee Management System - Defining Query Methods

1. Defining Query Methods :

Custom Query Methods Using Keywords:

EmployeeRepository :

Java-

```
package com.example.employeemanagementsystem.repository;
import com.example.employeemanagementsystem.model.Employee;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
import java.util.List;

@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
   List<Employee> findByDepartmentName(String departmentName);
   List<Employee> findByNameContainingIgnoreCase(String name);
   @Query("SELECT e FROM Employee e WHERE e.email = :email")
   Employee findEmployeeByEmail(@Param("email") String email);
   @Query("SELECT e FROM Employee e WHERE e.department.id = :departmentId")
   List<Employee> findByDepartmentId(@Param("departmentId") Long departmentId);
}
```

2. Named Queries

Named queries are defined at the entity level and allow us to reuse queries across the application. Here's how you can define and use them:

Define Named Queries:

Java-

```
package com.example.employeemanagementsystem.model;
import jakarta.persistence.*;
@Entity
@Table(name = "employees")
@NamedQueries({
   @NamedQuery(name = "Employee.findByDepartmentNameNamedQuery",
           query = "SELECT e FROM Employee e WHERE e.department.name = :departmentName"),
   @NamedQuery(name = "Employee.findByEmailNamedQuery",
           query = "SELECT e FROM Employee e WHERE e.email = :email")
})
public class Employee {

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

   private String name;
   private String email;

   @ManyToOne
   @JoinColumn(name = "department_id")
   private Department department;

   // Getters and setters...
}
```

Use Named Queries:

EntityManager:

Java-

```
package com.example.employeemanagementsystem.service;
import com.example.employeemanagementsystem.model.Employee;
import com.example.employeemanagementsystem.repository.EmployeeRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import jakarta.persistence.EntityManager;
import jakarta.persistence.PersistenceContext;
import jakarta.persistence.TypedQuery;
import java.util.List;
import java.util.Optional;
@Service
public class EmployeeService {
    @Autowired
    private EmployeeRepository employeeRepository;
    @PersistenceContext
    private EntityManager entityManager;

    public List<Employee> getEmployeesByDepartmentNameNamedQuery(String departmentName) {
        TypedQuery<Employee> query =
entityManager.createNamedQuery("Employee.findByDepartmentNameNamedQuery", Employee.class);
        query.setParameter("departmentName", departmentName);
        return query.getResultList();
    }

    public Employee findEmployeeByEmailNamedQuery(String email) {
        TypedQuery<Employee> query =
entityManager.createNamedQuery("Employee.findByEmailNamedQuery", Employee.class);
        query.setParameter("email", email);
        return query.getSingleResult();
    }
}
```

Exercise 6: Employee Management System - Implementing Pagination and Sorting

Repository Update:

Java-

```
package com.example.employeemanagementsystem.repository;
import com.example.employeemanagementsystem.model.Employee;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
    // Other query methods...

    // Pagination method
    Page<Employee> findAll(Pageable pageable);
}
```

Service Method for Pagination:

Java-

```
package com.example.employeemanagementsystem.service;
import com.example.employeemanagementsystem.model.Employee;
import com.example.employeemanagementsystem.repository.EmployeeRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Service;

@Service
public class EmployeeService {
```

```
    @Autowired
      private EmployeeRepository employeeRepository;

      public Page<Employee> getEmployeesWithPagination(Pageable pageable) {
         return employeeRepository.findAll(pageable);
      }
   }
```

Controller Endpoint for Pagination:

Java-

```
package com.example.employeemanagementsystem.controller;
import com.example.employeemanagementsystem.model.Employee;
import com.example.employeemanagementsystem.service.EmployeeService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.web.bind.annotation.*;

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

   @Autowired
   private EmployeeService employeeService;

   @GetMapping("/page")
   public Page<Employee> getAllEmployeesWithPagination(Pageable pageable) {
      return employeeService.getEmployeesWithPagination(pageable);
   }
}
```

2. Sorting

Repository Update:

Java-

```
package com.example.employeemanagementsystem.repository;
import com.example.employeemanagementsystem.model.Employee;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
   // Other query methods...

   // Sorting and Pagination method
   Page<Employee> findAll(Pageable pageable);
   List<Employee> findAll(Sort sort);
}
```

Service Method for Sorting:

Java-

```
package com.example.employeemanagementsystem.service;
import com.example.employeemanagementsystem.model.Employee;
import com.example.employeemanagementsystem.repository.EmployeeRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;
import java.util.List;
```

```
@Service
public class EmployeeService {

    @Autowired
    private EmployeeRepository employeeRepository;

    public Page<Employee> getEmployeesWithPaginationAndSorting(Pageable pageable) {
        return employeeRepository.findAll(pageable);
    }

    public List<Employee> getEmployeesWithSorting(Sort sort) {
        return employeeRepository.findAll(sort);
    }
}
```

Controller Endpoint for Pagination and Sorting:

Java-

```
package com.example.employeemanagementsystem.controller;
import com.example.employeemanagementsystem.model.Employee;
import com.example.employeemanagementsystem.service.EmployeeService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.web.bind.annotation.*;
import java.util.List;
@RestController
@RequestMapping("/employees")
public class EmployeeController {

    @Autowired
    private EmployeeService employeeService;
    @GetMapping("/page")
    public Page<Employee> getAllEmployeesWithPaginationAndSorting(Pageable pageable) {
```

```
        return employeeService.getEmployeesWithPaginationAndSorting(pageable);
    }

    @GetMapping("/sorted")
    public List<Employee> getAllEmployeesWithSorting(Sort sort) {
        return employeeService.getEmployeesWithSorting(sort);
    }
}
```

Testing Pagination and Sorting :

1. Pagination:

- Use the endpoint `GET /employees/page` with query parameters like `?page=0&size=5` to fetch paginated results.

2. Sorting:

- Use the endpoint `GET /employees/sorted` with a `Sort` parameter like `?sort=name,asc` or `?sort=name,desc` to fetch sorted results.

3. Combined Pagination and Sorting:

- Combine both pagination and sorting using the endpoint `GET /employees/page` with parameters like `?page=0&size=5&sort=name,asc`.

Exercise 7: Employee Management System - Enabling Entity Auditing

1. Enable Auditing:

Step 1: Enable Auditing in Configuration

First, enable JPA auditing by adding the `@EnableJpaAuditing` annotation to your main application class.

Java-

```
package com.example.employeemanagementsystem;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.data.jpa.repository.config.EnableJpaAuditing;

@SpringBootApplication
@EnableJpaAuditing
public class EmployeeManagementSystemApplication {

   public static void main(String[] args) {
      SpringApplication.run(EmployeeManagementSystemApplication.class, args);
   }
}
```

Step 2: Configure AuditorAware

AuditorAware :

Java-

```
package com.example.employeemanagementsystem.config;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.domain.AuditorAware;
import java.util.Optional;

@Configuration
public class AuditorAwareImpl implements AuditorAware<String> {
   @Override
   public Optional<String> getCurrentAuditor() {
```

```
            return Optional.of("admin");
        }
    }
```

Step 3: Add Auditing Annotations to Entities

Annotate the `Employee` and `Department` entities with auditing annotations.

Java-

```
    package com.example.employeemanagementsystem.model;
    import jakarta.persistence.*;
    import org.springframework.data.annotation.CreatedBy;
    import org.springframework.data.annotation.CreatedDate;
    import org.springframework.data.annotation.LastModifiedBy;
    import org.springframework.data.annotation.LastModifiedDate;
    import org.springframework.data.jpa.domain.support.AuditingEntityListener;
    import java.time.LocalDateTime;
    @Entity
    @Table(name = "employees")
    @EntityListeners(AuditingEntityListener.class)
    public class Employee {

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

        private String name;
        private String email;

        @ManyToOne
        @JoinColumn(name = "department_id")
        private Department department;

        @CreatedBy
        private String createdBy;
```

```
    @CreatedDate
    private LocalDateTime createdDate;

    @LastModifiedBy
    private String lastModifiedBy;

    @LastModifiedDate
    private LocalDateTime lastModifiedDate;
}

@Entity
@Table(name = "departments")
@EntityListeners(AuditingEntityListener.class)
public class Department {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    @CreatedBy
    private String createdBy;

    @CreatedDate
    private LocalDateTime createdDate;

    @LastModifiedBy
    private String lastModifiedBy;

    @LastModifiedDate
    private LocalDateTime lastModifiedDate;
}
```

Exercise 8: Employee Management System - Creating Projections

1. Define Projections :
   Interface-Based Projection:

Create interfaces to define projections for the `Employee` and `Department` entities.

Java-

```
package com.example.employeemanagementsystem.projection;
public interface EmployeeProjection {
   Long getId();
   String getName();
   String getEmail();
   String getDepartmentName();
}

public interface DepartmentProjection {
   Long getId();
   String getName();
}
```

Class-Based Projection:

Java-

```
package com.example.employeemanagementsystem.dto;
public class EmployeeDTO {
   private Long id;
   private String name;
   private String email;
   private String departmentName;
   public EmployeeDTO(Long id, String name, String email, String departmentName) {
      this.id = id;
      this.name = name;
      this.email = email;
      this.departmentName = departmentName;
   }
}
```

```
public class DepartmentDTO {
    private Long id;
    private String name;

    public DepartmentDTO(Long id, String name) {
        this.id = id;
        this.name = name;
    }
}
```

2. Use Projections in Repository Methods

Using Interface-Based Projection:

Java-

```
package com.example.employeemanagementsystem.repository;
import com.example.employeemanagementsystem.model.Employee;
import com.example.employeemanagementsystem.projection.EmployeeProjection;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;
import java.util.List;

@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
    @Query("SELECT e.id as id, e.name as name, e.email as email, e.department.name as
departmentName FROM Employee e")
    List<EmployeeProjection> findAllEmployeeProjections();
}
```

Using Class-Based Projection:

Java-

```
package com.example.employeemanagementsystem.repository;
import com.example.employeemanagementsystem.model.Employee;
import com.example.employeemanagementsystem.dto.EmployeeDTO;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;
import java.util.List;

@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
   @Query("SELECT new com.example.employeemanagementsystem.dto.EmployeeDTO(e.id, e.name, e.email, e.department.name) FROM Employee e")
   List<EmployeeDTO> findAllEmployeeDTOs();
}
```

3. Fetching Projections in the Service Layer

Java-

```
package com.example.employeemanagementsystem.service;
import com.example.employeemanagementsystem.dto.EmployeeDTO;
import com.example.employeemanagementsystem.projection.EmployeeProjection;
import com.example.employeemanagementsystem.repository.EmployeeRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
@Service
public class EmployeeService {

   @Autowired
   private EmployeeRepository employeeRepository;
   public List<EmployeeProjection> getAllEmployeeProjections() {
      return employeeRepository.findAllEmployeeProjections();
   }
```

```
    public List<EmployeeDTO> getAllEmployeeDTOs() {
        return employeeRepository.findAllEmployeeDTOs();
    }
}
```

4. Fetching Projections in the Controller Layer

Java-

```
package com.example.employeemanagementsystem.controller;
import com.example.employeemanagementsystem.dto.EmployeeDTO;
import com.example.employeemanagementsystem.projection.EmployeeProjection;
import com.example.employeemanagementsystem.service.EmployeeService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import java.util.List;

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

    @Autowired
    private EmployeeService employeeService;

    @GetMapping("/projections")
    public List<EmployeeProjection> getEmployeeProjections() {
        return employeeService.getAllEmployeeProjections();
    }

    @GetMapping("/dto")
    public List<EmployeeDTO> getEmployeeDTOs() {
        return employeeService.getAllEmployeeDTOs();
    }
}
```

Testing Entity Auditing and Projections

1. Entity Auditing:

- Verify that the `createdBy`, `createdDate`, `lastModifiedBy`, and `lastModifiedDate` fields are populated and updated appropriately in the database.

2. Projections:

- Use the endpoints `GET /employees/projections` and `GET /employees/dto` to fetch data with projections.

- Ensure that the projection results only contain the specified fields.

Exercise 9: Employee Management System - Customizing Data Source Configuration

1. Spring Boot Auto-Configuration

Default Data Source Configuration:

application.properties :

```
# Default Data Source Configuration
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=password
spring.datasource.driver-class-name=org.h2.Driver
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update
```

2. Externalizing Configuration

**Externalize Configuration in `application.properties`:

application.properties :

```
# Default H2 Data Source Configuration
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=password
spring.datasource.driver-class-name=org.h2.Driver
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update

# MySQL Data Source Configuration
app.datasource.mysql.url=jdbc:mysql://localhost:3306/employee_db
app.datasource.mysql.username=root
app.datasource.mysql.password=yourpassword
app.datasource.mysql.driver-class-name=com.mysql.cj.jdbc.Driver
```

Manage Multiple Data Sources:

Java -

```
package com.example.employeemanagementsystem.config;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.boot.jdbc.DataSourceBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Primary;
import org.springframework.core.env.Environment;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import org.springframework.jdbc.datasource.DataSourceTransactionManager;
import org.springframework.orm.jpa.JpaTransactionManager;
import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;
import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;
import javax.sql.DataSource;
import java.util.HashMap;

@Configuration
@EnableJpaRepositories(
    basePackages = "com.example.employeemanagementsystem.repository",
    entityManagerFactoryRef = "entityManagerFactory",
    transactionManagerRef = "transactionManager"
)
public class DataSourceConfig {

    @Autowired
    private Environment env;

    @Primary
    @Bean(name = "dataSource")
    @ConfigurationProperties(prefix = "spring.datasource")
    public DataSource dataSource() {
```

```
    return DataSourceBuilder.create().build();
}

@Bean(name = "mysqlDataSource")
@ConfigurationProperties(prefix = "app.datasource.mysql")
public DataSource mysqlDataSource() {
    return DataSourceBuilder.create().build();
}

@Primary
@Bean(name = "entityManagerFactory")
public LocalContainerEntityManagerFactoryBean entityManagerFactory() {
    LocalContainerEntityManagerFactoryBean em = new LocalContainerEntityManagerFactoryBean();
    em.setDataSource(dataSource());
    em.setPackagesToScan("com.example.employeemanagementsystem.model");

    HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
    em.setJpaVendorAdapter(vendorAdapter);
    em.setJpaPropertyMap(hibernateProperties());

    return em;
}

@Bean(name = "mysqlEntityManagerFactory")
public LocalContainerEntityManagerFactoryBean mysqlEntityManagerFactory() {
    LocalContainerEntityManagerFactoryBean em = new LocalContainerEntityManagerFactoryBean();
    em.setDataSource(mysqlDataSource());
    em.setPackagesToScan("com.example.employeemanagementsystem.model");

    HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
    em.setJpaVendorAdapter(vendorAdapter);
    em.setJpaPropertyMap(hibernateProperties());
    return em;
}
```

```
    @Primary
    @Bean(name = "transactionManager")
    public JpaTransactionManager transactionManager() {
        JpaTransactionManager transactionManager = new JpaTransactionManager();
        transactionManager.setEntityManagerFactory(entityManagerFactory().getObject());
        return transactionManager;
    }

    @Bean(name = "mysqlTransactionManager")
    public DataSourceTransactionManager mysqlTransactionManager() {
        DataSourceTransactionManager transactionManager = new DataSourceTransactionManager();
        transactionManager.setDataSource(mysqlDataSource());
        return transactionManager;
    }

    private HashMap<String, Object> hibernateProperties() {
        HashMap<String, Object> properties = new HashMap<>();
        properties.put("hibernate.hbm2ddl.auto", env.getProperty("spring.jpa.hibernate.ddl-auto"));
        properties.put("hibernate.dialect", env.getProperty("spring.jpa.database-platform"));
        return properties;
    }
}
```

Switching Between Data Sources:

You can switch between the data sources by specifying the data source bean to use for different repositories or services.

Exercise 10: Employee Management System - Hibernate-Specific Features

1. Hibernate-Specific Annotations

Example of Hibernate-Specific Annotations:

Java-

```
package com.example.employeemanagementsystem.model;
import jakarta.persistence.*;
import org.hibernate.annotations.Cache;
import org.hibernate.annotations.CacheConcurrencyStrategy;
import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.UpdateTimestamp;
import java.time.LocalDateTime;
@Entity
@Table(name = "employees")
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "department_id")
    private Department department;

    @CreationTimestamp
    private LocalDateTime createdDate;

    @UpdateTimestamp
    private LocalDateTime lastModifiedDate;
}
```

2. Configuring Hibernate Dialect and Properties

application.properties :

```
# Hibernate Configuration
spring.jpa.hibernate.ddl-auto=update
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.use_sql_comments=true
spring.jpa.properties.hibernate.show_sql=true
```

3. Batch Processing

Enable Batch Processing:

application.properties :

```
# Hibernate Batch Processing
spring.jpa.properties.hibernate.jdbc.batch_size=20
spring.jpa.properties.hibernate.order_inserts=true
spring.jpa.properties.hibernate.order_updates=true
```

Implementing Batch Processing:

Java -

```
package com.example.employeemanagementsystem.service;
import com.example.employeemanagementsystem.model.Employee;
import com.example.employeemanagementsystem.repository.EmployeeRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import jakarta.transaction.Transactional;
import java.util.List;

@Service
public class EmployeeService {

    @Autowired
    private EmployeeRepository employeeRepository;
```

```
    @Transactional
    public void saveAllEmployees(List<Employee> employees) {
        employeeRepository.saveAll(employees);
    }
}
```

Testing Data Source Configuration and Hibernate Features

1. Data Source Configuration:
   - Verify that the application can connect to and use multiple data sources.
   - Test CRUD operations on both data sources.

2. Hibernate Features:
   - Check that the entity timestamps (`createdDate` and `lastModifiedDate`) are being automatically managed.
   - Verify that caching is working by observing reduced database queries.
   - Use batch processing to save or update multiple records and observe the performance improvement.