### 1. Implementing CRUD Operations

#### 1.1 Basic CRUD Operations with JpaRepository

Spring Data JPA’s JpaRepository provides built-in methods for basic CRUD operations. These methods include:

* **Create**: save()
* **Read**: findById(), findAll()
* **Update**: save()
* **Delete**: deleteById(), delete()

#### 1.2 Implementing RESTful Endpoints

Create REST controllers for Employee and Department to handle CRUD operations through HTTP requests.

### ****EmployeeController****

Here’s how to implement CRUD operations for the Employee entity:

**EmployeeController.java**

package com.example.employeemanagementsystem.controller;

import com.example.employeemanagementsystem.model.Employee;

import com.example.employeemanagementsystem.repository.EmployeeRepository;

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

import org.springframework.http.HttpStatus;

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 EmployeeRepository employeeRepository;

@GetMapping

public List<Employee> getAllEmployees() {

return employeeRepository.findAll();

}

@GetMapping("/{id}")

public ResponseEntity<Employee> getEmployeeById(@PathVariable("id") Long id) {

Optional<Employee> employee = employeeRepository.findById(id);

return employee.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());

}

@PostMapping

public ResponseEntity<Employee> createEmployee(@RequestBody Employee employee) {

Employee savedEmployee = employeeRepository.save(employee);

return ResponseEntity.status(HttpStatus.CREATED).body(savedEmployee);

}

@PutMapping("/{id}")

public ResponseEntity<Employee> updateEmployee(@PathVariable("id") Long id, @RequestBody Employee employee) {

if (!employeeRepository.existsById(id)) {

return ResponseEntity.notFound().build();

}

employee.setId(id);

Employee updatedEmployee = employeeRepository.save(employee);

return ResponseEntity.ok(updatedEmployee);

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteEmployee(@PathVariable("id") Long id) {

if (!employeeRepository.existsById(id)) {

return ResponseEntity.notFound().build();

}

employeeRepository.deleteById(id);

return ResponseEntity.noContent().build();

}

}

### ****DepartmentController****

Similarly, implement CRUD operations for the Department entity:

**DepartmentController.java**

package com.example.employeemanagementsystem.controller;

import com.example.employeemanagementsystem.model.Department;

import com.example.employeemanagementsystem.repository.DepartmentRepository;

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

import org.springframework.http.HttpStatus;

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 DepartmentRepository departmentRepository;

@GetMapping

public List<Department> getAllDepartments() {

return departmentRepository.findAll();

}

@GetMapping("/{id}")

public ResponseEntity<Department> getDepartmentById(@PathVariable("id") Long id) {

Optional<Department> department = departmentRepository.findById(id);

return department.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());

}

@PostMapping

public ResponseEntity<Department> createDepartment(@RequestBody Department department) {

Department savedDepartment = departmentRepository.save(department);

return ResponseEntity.status(HttpStatus.CREATED).body(savedDepartment);

}

@PutMapping("/{id}")

public ResponseEntity<Department> updateDepartment(@PathVariable("id") Long id, @RequestBody Department department) {

if (!departmentRepository.existsById(id)) {

return ResponseEntity.notFound().build();

}

department.setId(id);

Department updatedDepartment = departmentRepository.save(department);

return ResponseEntity.ok(updatedDepartment);

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteDepartment(@PathVariable("id") Long id) {

if (!departmentRepository.existsById(id)) {

return ResponseEntity.notFound().build();

}

departmentRepository.deleteById(id);

return ResponseEntity.noContent().build();

}

}

### 2. Testing CRUD Operations

#### 2.1 Manual Testing

You can test the endpoints using tools like Postman or curl:

**Create Employee**:

* + **POST** /employees
  + **Body**: {"name": "John Doe", "email": "john.doe@example.com", "department": {"id": 1}}

**Get All Employees**:

* + **GET** /employees

**Get Employee by ID**:

* + **GET** /employees/{id}

**Update Employee**:

* + **PUT** /employees/{id}
  + **Body**: {"name": "John Smith", "email": "john.smith@example.com", "department": {"id": 1}}

**Delete Employee**:

* + **DELETE** /employees/{id}

**Create Department**:

* + **POST** /departments
  + **Body**: {"name": "Engineering"}

**Get All Departments**:

* + **GET** /departments

**Get Department by ID**:

* + **GET** /departments/{id}

**Update Department**:

* + **PUT** /departments/{id}
  + **Body**: {"name": "Sales"}

**Delete Department**:

* + **DELETE** /departments/{id}

#### 2.2 Automated Testing

You can also write integration tests using Spring Boot’s testing support. Here’s a basic example for EmployeeController:

**EmployeeControllerTest.java**

package com.example.employeemanagementsystem;

import com.example.employeemanagementsystem.model.Employee;

import com.example.employeemanagementsystem.model.Department;

import com.example.employeemanagementsystem.repository.EmployeeRepository;

import com.example.employeemanagementsystem.repository.DepartmentRepository;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

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

import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;

import org.springframework.http.MediaType;

import org.springframework.test.web.servlet.MockMvc;

import org.springframework.test.web.servlet.setup.MockMvcBuilders;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.\*;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@WebMvcTest

public class EmployeeControllerTest {

@Autowired

private MockMvc mockMvc;

@Autowired

private EmployeeRepository employeeRepository;

@Autowired

private DepartmentRepository departmentRepository;

@BeforeEach

public void setup() {

mockMvc = MockMvcBuilders.standaloneSetup(new EmployeeController(employeeRepository)).build();

departmentRepository.save(new Department(null, "Engineering"));

}

@Test

public void testCreateEmployee() throws Exception {

String employeeJson = "{\"name\":\"John Doe\",\"email\":\"john.doe@example.com\",\"department\":{\"id\":1}}";

mockMvc.perform(post("/employees")

.contentType(MediaType.APPLICATION\_JSON)

.content(employeeJson))

.andExpect(status().isCreated())

.andExpect(jsonPath("$.name").value("John Doe"));

}

@Test

public void testGetEmployee() throws Exception {

Employee employee = new Employee();

employee.setName("Jane Doe");

employee.setEmail("jane.doe@example.com");

employee.setDepartment(departmentRepository.findByName("Engineering").orElse(null));

employeeRepository.save(employee);

mockMvc.perform(get("/employees/{id}", employee.getId()))

.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("Jane Doe"));

}

// Additional tests for update and delete

}