**Spring Data JPA and Hibernate**

**Exercise 1: Overview and Setup**

Create a Spring Boot project named EmployeeManagementSystem with dependencies: Spring Data JPA, H2 Database, Spring Web, and Lombok.

# H2 Database Configuration

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

**Exercise 2: Creating Entities**

Define the Employee and Department entities with appropriate relationships.

Department Entity

package com.example.EmployeeManagementSystem.entity;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.OneToMany;

import jakarta.persistence.Table;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

import java.util.List;

@Entity

@Table(name = "departments")

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

// One department can have many employees

@OneToMany(mappedBy = "department")

private List<Employee> employees;

}

Employee Entity

package com.example.EmployeeManagementSystem.entity;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.JoinColumn;

import jakarta.persistence.ManyToOne;

import jakarta.persistence.Table;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Entity

@Table(name = "employees")

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

// Many employees belong to one department

@ManyToOne

@JoinColumn(name = "department\_id")

private Department department;

}

**Exercise 3: Creating Repositories**

Create repositories for Employee and Department entities.

EmployeeRepository Interface

package com.example.EmployeeManagementSystem.repository;

import com.example.EmployeeManagementSystem.entity.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 ID

List<Employee> findByDepartmentId(Long departmentId);

// Derived query method to find employees by name

List<Employee> findByNameContaining(String name);

}

DepartmentRepository Interface

package com.example.EmployeeManagementSystem.repository;

import com.example.EmployeeManagementSystem.entity.Department;

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

import org.springframework.stereotype.Repository;

@Repository

public interface DepartmentRepository extends JpaRepository<Department, Long> {

// Derived query method to find a department by name

Department findByName(String name);

}

**Exercise 4: Implementing CRUD Operations**

Implement RESTful CRUD operations.

EmployeeService.java

package com.example.EmployeeManagementSystem.service;

import com.example.EmployeeManagementSystem.entity.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);

}

public Employee updateEmployee(Long id, Employee updatedEmployee) {

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

if (existingEmployee.isPresent()) {

Employee employee = existingEmployee.get();

employee.setName(updatedEmployee.getName());

employee.setEmail(updatedEmployee.getEmail());

employee.setDepartment(updatedEmployee.getDepartment());

return employeeRepository.save(employee);

} else {

throw new RuntimeException("Employee not found with id " + id);

}

}

}

DepartmentService.java

package com.example.EmployeeManagementSystem.service;

import com.example.EmployeeManagementSystem.entity.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);

}

public Department updateDepartment(Long id, Department updatedDepartment) {

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

if (existingDepartment.isPresent()) {

Department department = existingDepartment.get();

department.setName(updatedDepartment.getName());

return departmentRepository.save(department);

} else {

throw new RuntimeException("Department not found with id " + id);

}

}

}

EmployeeController.java

package com.example.EmployeeManagementSystem.controller;

import com.example.EmployeeManagementSystem.entity.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("/api/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 updatedEmployee) {

try {

Employee employee = employeeService.updateEmployee(id, updatedEmployee);

return ResponseEntity.ok(employee);

} catch (RuntimeException e) {

return ResponseEntity.notFound().build();

}

}

@DeleteMapping("/{id}")

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

employeeService.deleteEmployee(id);

return ResponseEntity.noContent().build();

}

}

DepartmentController.java

package com.example.EmployeeManagementSystem.controller;

import com.example.EmployeeManagementSystem.entity.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("/api/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 updatedDepartment) {

try {

Department department = departmentService.updateDepartment(id, updatedDepartment);

return ResponseEntity.ok(department);

} catch (RuntimeException e) {

return ResponseEntity.notFound().build();

}

}

@DeleteMapping("/{id}")

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

departmentService.deleteDepartment(id);

return ResponseEntity.noContent().build();

}

}

**Exercise 5: Defining Query Methods**

Define custom query methods using method names and annotations.

// EmployeeRepository.java

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

List<Employee> findByNameContaining(String name);

@Query("SELECT e FROM Employee e WHERE e.email = ?1")

Employee findByEmailAddress(String email);

}

**Exercise 6: Implementing Pagination and Sorting**

Add pagination and sorting to employee search functionality.

// EmployeeController.java

@GetMapping("/paged")

public Page<Employee> getAllEmployeesPaged(Pageable pageable) {

return employeeRepository.findAll(pageable);

}

@GetMapping("/sorted")

public List<Employee> getAllEmployeesSorted(Sort sort) {

return employeeRepository.findAll(sort);

}

**Exercise 7: Enabling Entity Auditing**

Enable auditing for tracking creation and modification.

// AuditModel.java

@MappedSuperclass

@EntityListeners(AuditingEntityListener.class)

public abstract class AuditModel {

@CreatedDate

@Temporal(TemporalType.TIMESTAMP)

@Column(name = "created\_at", nullable = false, updatable = false)

private Date createdAt;

@LastModifiedDate

@Temporal(TemporalType.TIMESTAMP)

@Column(name = "updated\_at", nullable = false)

private Date updatedAt;

// getters and setters

}

// Employee.java and Department.java extend AuditModel

**Exercise 8: Creating Projections**

Create projections to fetch specific data subsets.

// EmployeeProjection.java

public interface EmployeeProjection {

String getName();

String getDepartmentName();

}

// EmployeeRepository.java

@Query("SELECT e.name as name, d.name as departmentName FROM Employee e JOIN e.department d")

List<EmployeeProjection> findEmployeeNameAndDepartment();

Here are the code snippets for each exercise in the Employee Management System project using Spring Data JPA and Hibernate:

**Exercise 9: Customizing Data Source Configuration**

Customize and manage multiple data sources.

# application.properties for multiple data sources

spring.datasource.primary.url=jdbc:h2:mem:primarydb

spring.datasource.secondary.url=jdbc:h2:mem:secondarydb

# Configurations for multiple DataSources

@Configuration

public class DataSourceConfig {

@Bean

@Primary

@ConfigurationProperties(prefix = "spring.datasource.primary")

public DataSource primaryDataSource() {

return DataSourceBuilder.create().build();

}

@Bean

@ConfigurationProperties(prefix = "spring.datasource.secondary")

public DataSource secondaryDataSource() {

return DataSourceBuilder.create().build();

}

}

**Exercise 10: Hibernate-Specific Features**

Leverage Hibernate-specific features.

// Hibernate-specific annotations in entities

@Entity

@Table(name = "employees")

@org.hibernate.annotations.DynamicUpdate

public class Employee {

// Fields and methods

}

// Batch processing configuration in application.properties

spring.jpa.properties.hibernate.jdbc.batch\_size=20

spring.jpa.properties.hibernate.order\_inserts=true