**Week-3 Exercises**

**Exercise 1: Employee Management System - Overview and Setup**

1. Creating a Spring Boot Project

// a. Initialize a new Spring Boot project named EmployeeManagementSystem using Maven or Gradle

// Example pom.xml for Maven

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/POM/4.0.0">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>employeemanagementsystem</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>EmployeeManagementSystem</name>

<description>Employee Management System using Spring Boot</description>

<properties>

<java.version>17</java.version>

</properties>

<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>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>1.18.24</version>

<scope>provided</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

2. Configuring Application Properties

// a. Configure application.properties for H2 database connection

// src/main/resources/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

// Optional: Enable H2 Console

spring.h2.console.enabled=true

spring.h2.console.path=/h2-console

**Exercise 2: Employee Management System - Creating Entities**

package com.example.employeemanagementsystem.model;

import jakarta.persistence.\*;

import lombok.Data;

import lombok.NoArgsConstructor;

import lombok.AllArgsConstructor;

import java.util.List;

// Employee Entity

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

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;

}

// Department Entity

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@OneToMany(mappedBy = "department")

private List<Employee> employees;

}

**Exercise 3: Employee Management System - Creating Repositories**

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.model.Employee;

import com.example.employeemanagementsystem.model.Department;

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

// Employee Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Custom query methods can be defined here

}

// Department Repository

public interface DepartmentRepository extends JpaRepository<Department, Long> {

// Custom query methods can be defined here

}

**Exercise 4: Employee Management System - Implementing CRUD Operations**

package com.example.employeemanagementsystem.controller;

import com.example.employeemanagementsystem.model.Employee;

import com.example.employeemanagementsystem.model.Department;

import com.example.employeemanagementsystem.service.EmployeeService;

import com.example.employeemanagementsystem.service.DepartmentService;

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

import org.springframework.web.bind.annotation.\*;

import java.util.List;

// Employee Controller

@RestController

@RequestMapping("/employees")

public class EmployeeController {

@Autowired

private EmployeeService employeeService;

@GetMapping

public List<Employee> getAllEmployees() {

return employeeService.getAllEmployees();

}

@GetMapping("/{id}")

public Employee getEmployeeById(@PathVariable Long id) {

return employeeService.getEmployeeById(id);

}

@PostMapping

public Employee createEmployee(@RequestBody Employee employee) {

return employeeService.createEmployee(employee);

}

@PutMapping("/{id}")

public Employee updateEmployee(@PathVariable Long id, @RequestBody Employee employee) {

return employeeService.updateEmployee(id, employee);

}

@DeleteMapping("/{id}")

public void deleteEmployee(@PathVariable Long id) {

employeeService.deleteEmployee(id);

}

}

// Department Controller

@RestController

@RequestMapping("/departments")

public class DepartmentController {

@Autowired

private DepartmentService departmentService;

@GetMapping

public List<Department> getAllDepartments() {

return departmentService.getAllDepartments();

}

@GetMapping("/{id}")

public Department getDepartmentById(@PathVariable Long id) {

return departmentService.getDepartmentById(id);

}

@PostMapping

public Department createDepartment(@RequestBody Department department) {

return departmentService.createDepartment(department);

}

@PutMapping("/{id}")

public Department updateDepartment(@PathVariable Long id, @RequestBody Department department) {

return departmentService.updateDepartment(id, department);

}

@DeleteMapping("/{id}")

public void deleteDepartment(@PathVariable Long id) {

departmentService.deleteDepartment(id);

}

}

Implementing Services

package com.example.employeemanagementsystem.service;

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.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

// Employee Service

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

public List<Employee> getAllEmployees() {

return employeeRepository.findAll();

}

public Employee getEmployeeById(Long id) {

return employeeRepository.findById(id).orElse(null);

}

public Employee createEmployee(Employee employee) {

return employeeRepository.save(employee);

}

public Employee updateEmployee(Long id, Employee updatedEmployee) {

Employee existingEmployee = employeeRepository.findById(id).orElse(null);

if (existingEmployee != null) {

existingEmployee.setName(updatedEmployee.getName());

existingEmployee.setEmail(updatedEmployee.getEmail());

existingEmployee.setDepartment(updatedEmployee.getDepartment());

return employeeRepository.save(existingEmployee);

}

return null;

}

public void deleteEmployee(Long id) {

employeeRepository.deleteById(id);

}

}

// Department Service

@Service

public class DepartmentService {

@Autowired

private DepartmentRepository departmentRepository;

public List<Department> getAllDepartments() {

return departmentRepository.findAll();

}

public Department getDepartmentById(Long id) {

return departmentRepository.findById(id).orElse(null);

}

public Department createDepartment(Department department) {

return departmentRepository.save(department);

}

public Department updateDepartment(Long id, Department updatedDepartment) {

Department existingDepartment = departmentRepository.findById(id).orElse(null);

if (existingDepartment != null) {

existingDepartment.setName(updatedDepartment.getName());

return departmentRepository.save(existingDepartment);

}

return null;

}

public void deleteDepartment(Long id) {

departmentRepository.deleteById(id);

}

}

**Exercise 5: Employee Management System - Defining Query Methods**

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.model.Employee;

import com.example.employeemanagementsystem.model.Department;

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

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

import org.springframework.data.repository.query.Param;

import java.util.List;

// Employee Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Custom query method using keywords

List<Employee> findByName(String name);

// Custom query method using @Query annotation

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

Employee findByEmail(@Param("email") String email);

}

// Department Repository

public interface DepartmentRepository extends JpaRepository<Department, Long> {

// Custom query method using keywords

List<Department> findByName(String name);

}

**Exercise 6: Implementing Pagination and Sorting**

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.PageRequest;

import org.springframework.data.domain.Pageable;

import org.springframework.data.domain.Sort;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

// Employee Controller with Pagination and Sorting

@RestController

@RequestMapping("/employees")

public class EmployeeController {

@Autowired

private EmployeeService employeeService;

@GetMapping

public Page<Employee> getAllEmployees(

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

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

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

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

Pageable pageable = PageRequest.of(page, size, Sort.by(Sort.Direction.fromString(sortDir), sortBy));

return employeeService.getAllEmployees(pageable);

}

}

**Exercise 7: Enabling Entity Auditing**

package com.example.employeemanagementsystem.model;

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 jakarta.persistence.EntityListeners;

import jakarta.persistence.MappedSuperclass;

import java.time.LocalDateTime;

// Base entity class for auditing

@MappedSuperclass

@EntityListeners(AuditingEntityListener.class)

public abstract class Auditable {

@CreatedBy

private String createdBy;

@CreatedDate

private LocalDateTime createdDate;

@LastModifiedBy

private String lastModifiedBy;

@LastModifiedDate

private LocalDateTime lastModifiedDate;

}

// Modify Employee and Department to extend Auditable

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Employee extends Auditable {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

@ManyToOne

@JoinColumn(name = "department\_id")

private Department department;

}

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Department extends Auditable {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@OneToMany(mappedBy = "department")

private List<Employee> employees;

}

**Exercise 8: Creating Projections**

package com.example.employeemanagementsystem.projection;

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

import org.springframework.data.rest.core.config.Projection;

// Interface-based projection

@Projection(name = "employeeProjection", types = Employee.class)

public interface EmployeeProjection {

Long getId();

String getName();

@Value("#{target.department.name}")

String getDepartmentName();

}

// Class-based projection

public class EmployeeSummary {

private Long id;

private String name;

private String departmentName;

public EmployeeSummary(Long id, String name, String departmentName) {

this.id = id;

this.name = name;

this.departmentName = departmentName;

}

// Getters and setters

}

**Exercise 9: Customizing Data Source Configuration**

// Example configuration for multiple data sources in application.properties

# Primary Data Source

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

# Secondary Data Source (Example)

spring.datasource.secondary.url=jdbc:mysql://localhost:3306/secondarydb

spring.datasource.secondary.username=root

spring.datasource.secondary.password=root

// Example of using @Primary and @Qualifier to configure multiple data sources in Spring

@Configuration

public class DataSourceConfig {

@Bean

@Primary

@ConfigurationProperties(prefix = "spring.datasource")

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**

package com.example.employeemanagementsystem.model;

import org.hibernate.annotations.BatchSize;

import org.hibernate.annotations.Fetch;

import org.hibernate.annotations.FetchMode;

import org.hibernate.annotations.SQLDelete;

import org.hibernate.annotations.Where;

import jakarta.persistence.\*;

@Entity

@Table(name = "employee")

@SQLDelete(sql = "UPDATE employee SET deleted = true WHERE id = ?")

@Where(clause = "deleted = false")

@BatchSize(size = 50)

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

@ManyToOne(fetch = FetchType.LAZY)

@Fetch(FetchMode.SELECT)

@JoinColumn(name = "department\_id")

private Department department;

@Column(name = "deleted")

private boolean deleted = false;

}