**Exercise 9: Employee Management System - Customizing Data Source**

**Application.properties**

# Spring Boot application name

spring.application.name=EmployeeManagementSystem

# Primary DataSource configuration

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

spring.datasource.primary.driverClassName=org.h2.Driver

spring.datasource.primary.username=sa

spring.datasource.primary.password=password

# Secondary DataSource configuration

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

spring.datasource.secondary.driverClassName=org.h2.Driver

spring.datasource.secondary.username=sa

spring.datasource.secondary.password=password

# JPA Configuration

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

**Department.java**

package com.example.EmployeeManagementSystem.entity;

import jakarta.persistence.\*;

import lombok.Getter;

import lombok.Setter;

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 java.time.LocalDateTime;

import java.util.List;

@Entity

@Table(name = "departments")

@Getter

@Setter

@NamedQueries({

@NamedQuery(name = "Department.findByName", query = "SELECT d FROM Department d WHERE d.name = :name")

})

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@OneToMany(mappedBy = "department")

private List<Employee> employees;

@CreatedDate

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

private LocalDateTime createdDate;

@LastModifiedDate

@Column(name = "last\_modified\_date")

private LocalDateTime lastModifiedDate;

@CreatedBy

@Column(name = "created\_by")

private String createdBy;

@LastModifiedBy

@Column(name = "last\_modified\_by")

private String lastModifiedBy;

// Constructors, if needed, can be added here

}

**Employee.java**

package com.example.EmployeeManagementSystem.entity;

import jakarta.persistence.\*;

import lombok.Getter;

import lombok.Setter;

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 java.time.LocalDateTime;

import java.util.List;

@Entity

@Table(name = "employees")

@Getter

@Setter

@NamedQueries({

@NamedQuery(name = "Employee.findByName", query = "SELECT e FROM Employee e WHERE e.name = :name"),

@NamedQuery(name = "Employee.findByEmail", 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;

@CreatedDate

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

private LocalDateTime createdDate;

@LastModifiedDate

@Column(name = "last\_modified\_date")

private LocalDateTime lastModifiedDate;

@CreatedBy

@Column(name = "created\_by")

private String createdBy;

@LastModifiedBy

@Column(name = "last\_modified\_by")

private String lastModifiedBy;

// Constructors, if needed, can be added here

}

**EmployeeController.java**

package com.example.EmployeeManagementSystem.controller;

import com.example.EmployeeManagementSystem.entity.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.PageRequest;

import org.springframework.data.domain.Pageable;

import org.springframework.data.domain.Sort;

import org.springframework.http.ResponseEntity;

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

import java.util.Optional;

@RestController

@RequestMapping("/employees")

public class EmployeeController {

@Autowired

private EmployeeRepository employeeRepository;

// Create a new Employee

@PostMapping

public Employee createEmployee(@RequestBody Employee employee) {

return employeeRepository.save(employee);

}

// Get all Employees with pagination and sorting

@GetMapping

public Page<Employee> getAllEmployees(

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

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

@RequestParam(defaultValue = "id,asc") String[] sort) {

Sort.Direction direction = Sort.Direction.ASC;

if (sort[1].equalsIgnoreCase("desc")) {

direction = Sort.Direction.DESC;

}

Sort sortOrder = Sort.by(direction, sort[0]);

Pageable pageable = PageRequest.of(page, size, sortOrder);

return employeeRepository.findAll(pageable);

}

// Get Employees by Name with pagination and sorting

@GetMapping("/search")

public Page<Employee> getEmployeesByName(

@RequestParam String name,

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

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

@RequestParam(defaultValue = "id,asc") String[] sort) {

Sort.Direction direction = Sort.Direction.ASC;

if (sort[1].equalsIgnoreCase("desc")) {

direction = Sort.Direction.DESC;

}

Sort sortOrder = Sort.by(direction, sort[0]);

Pageable pageable = PageRequest.of(page, size, sortOrder);

return employeeRepository.findByName(name, pageable);

}

// Get Employees by Email with pagination and sorting

@GetMapping("/searchByEmail")

public Page<Employee> getEmployeesByEmail(

@RequestParam String email,

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

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

@RequestParam(defaultValue = "id,asc") String[] sort) {

Sort.Direction direction = Sort.Direction.ASC;

if (sort[1].equalsIgnoreCase("desc")) {

direction = Sort.Direction.DESC;

}

Sort sortOrder = Sort.by(direction, sort[0]);

Pageable pageable = PageRequest.of(page, size, sortOrder);

return employeeRepository.findByEmail(email, pageable);

}

// Get Employee by ID

@GetMapping("/{id}")

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

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

return employee.map(ResponseEntity::ok)

.orElseGet(() -> ResponseEntity.notFound().build());

}

// Update an Employee

@PutMapping("/{id}")

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

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

if (employeeOptional.isPresent()) {

Employee employee = employeeOptional.get();

employee.setName(employeeDetails.getName());

employee.setEmail(employeeDetails.getEmail());

employee.setDepartment(employeeDetails.getDepartment());

final Employee updatedEmployee = employeeRepository.save(employee);

return ResponseEntity.ok(updatedEmployee);

} else {

return ResponseEntity.notFound().build();

}

}

// Delete an Employee

@DeleteMapping("/{id}")

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

if (employeeRepository.existsById(id)) {

employeeRepository.deleteById(id);

return ResponseEntity.ok().build();

} else {

return ResponseEntity.notFound().build();

}

}

}

**DepartmentController.java**

package com.example.EmployeeManagementSystem.controller;

import com.example.EmployeeManagementSystem.entity.Department;

import com.example.EmployeeManagementSystem.repository.DepartmentRepository;

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 DepartmentRepository departmentRepository;

// Create a new Department

@PostMapping

public Department createDepartment(@RequestBody Department department) {

return departmentRepository.save(department);

}

// Get all Departments

@GetMapping

public List<Department> getAllDepartments() {

return departmentRepository.findAll();

}

// Get Department by ID

@GetMapping("/{id}")

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

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

return department.map(ResponseEntity::ok)

.orElseGet(() -> ResponseEntity.notFound().build());

}

// Update a Department

@PutMapping("/{id}")

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

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

if (departmentOptional.isPresent()) {

Department department = departmentOptional.get();

department.setName(departmentDetails.getName());

final Department updatedDepartment = departmentRepository.save(department);

return ResponseEntity.ok(updatedDepartment);

} else {

return ResponseEntity.notFound().build();

}

}

// Delete a Department

@DeleteMapping("/{id}")

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

if (departmentRepository.existsById(id)) {

departmentRepository.deleteById(id);

return ResponseEntity.ok().build();

} else {

return ResponseEntity.notFound().build();

}

}

}

**DepartmentRepository.java**

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 methods

Department findByName(String name);

}

**EmployeeRepository.java**

package com.example.EmployeeManagementSystem.repository;

import com.example.EmployeeManagementSystem.entity.Employee;

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;

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Derived query methods

List<Employee> findByName(String name);

List<Employee> findByEmail(String email);

// Custom query with @Query annotation

@Query("SELECT e FROM Employee e WHERE e.department.name = :departmentName")

List<Employee> findByDepartmentNameCustom(@Param("departmentName") String departmentName);

// Paginated and sorted queries

Page<Employee> findByName(String name, Pageable pageable);

Page<Employee> findByEmail(String email, Pageable pageable);

// Paginated and sorted query for custom department name

@Query("SELECT e FROM Employee e WHERE e.department.name = :departmentName")

Page<Employee> findByDepartmentNameCustomPaginated(@Param("departmentName") String departmentName, Pageable pageable);

}

**Jpaconfig.java**

package com.example.EmployeeManagementSystem.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.data.domain.AuditorAware;

import org.springframework.data.jpa.repository.config.EnableJpaAuditing;

import java.util.Optional;

@Configuration

@EnableJpaAuditing

public class JpaConfig {

@Bean

public AuditorAware<String> auditorProvider() {

// Replace with actual user context logic

return () -> Optional.of("system"); // Replace "system" with logic to fetch the current user

}

}

**DepartmentDTO.java**

package com.example.EmployeeManagementSystem.projection;

public class DepartmentDTO {

private final String name;

public DepartmentDTO(String name) {

this.name = name;

}

public String getName() {

return name;

}

}

**EmployeeDTO.java**

package com.example.EmployeeManagementSystem.projection;

public class EmployeeDTO {

private final String name;

private final String email;

public EmployeeDTO(String name, String email) {

this.name = name;

this.email = email;

}

public String getName() {

return name;

}

public String getEmail() {

return email;

}

}

**EmployeeProjection.java**

package com.example.EmployeeManagementSystem.projection;

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

public interface EmployeeProjection {

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

String getName();

@Value("#{target.email}")

String getEmail();

}

**DepartmentProjection.java**

package com.example.EmployeeManagementSystem.projection;

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

public interface DepartmentProjection {

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

String getName();

}

**PrimaryDataSourceConfig**

package com.example.EmployeeManagementSystem.config;

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

import org.springframework.boot.autoconfigure.jdbc.DataSourceProperties;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.jdbc.datasource.DriverManagerDataSource;

import javax.sql.DataSource;

@Configuration

public class PrimaryDataSourceConfig {

@Bean

@Qualifier("primaryDataSource")

public DataSource primaryDataSource(DataSourceProperties properties) {

DriverManagerDataSource dataSource = new DriverManagerDataSource();

dataSource.setDriverClassName(properties.getDriverClassName());

dataSource.setUrl(properties.getUrl());

dataSource.setUsername(properties.getUsername());

dataSource.setPassword(properties.getPassword());

return dataSource;

}

}

**SecondaryDataSourceConfig.java**

package com.example.EmployeeManagementSystem.config;

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

import org.springframework.boot.autoconfigure.jdbc.DataSourceProperties;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.jdbc.datasource.DriverManagerDataSource;

import javax.sql.DataSource;

@Configuration

public class SecondaryDataSourceConfig {

@Bean

@Qualifier("secondaryDataSource")

public DataSource secondaryDataSource(DataSourceProperties properties) {

DriverManagerDataSource dataSource = new DriverManagerDataSource();

dataSource.setDriverClassName(properties.getDriverClassName());

dataSource.setUrl(properties.getUrl());

dataSource.setUsername(properties.getUsername());

dataSource.setPassword(properties.getPassword());

return dataSource;

}

}

**PrimaryJpaConfig**

package com.example.EmployeeManagementSystem.config;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

import org.springframework.orm.jpa.JpaTransactionManager;

import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;

import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;

import jakarta.persistence.EntityManagerFactory;

import javax.sql.DataSource;

@Configuration

@EnableJpaRepositories(

basePackages = "com.example.EmployeeManagementSystem.repository.primary",

entityManagerFactoryRef = "primaryEntityManagerFactory",

transactionManagerRef = "primaryTransactionManager"

)

public class PrimaryJpaConfig {

@Bean(name = "primaryEntityManagerFactory")

public LocalContainerEntityManagerFactoryBean primaryEntityManagerFactory(

@Qualifier("primaryDataSource") DataSource dataSource) {

LocalContainerEntityManagerFactoryBean em = new LocalContainerEntityManagerFactoryBean();

em.setDataSource(dataSource);

em.setPackagesToScan("com.example.EmployeeManagementSystem.entity");

em.setJpaVendorAdapter(new HibernateJpaVendorAdapter());

return em;

}

@Bean(name = "primaryTransactionManager")

public JpaTransactionManager primaryTransactionManager(

@Qualifier("primaryEntityManagerFactory") EntityManagerFactory entityManagerFactory) {

JpaTransactionManager transactionManager = new JpaTransactionManager();

transactionManager.setEntityManagerFactory(entityManagerFactory);

return transactionManager;

}

}

**SecondaryJpaConfig**

package com.example.EmployeeManagementSystem.config;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

import org.springframework.orm.jpa.JpaTransactionManager;

import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;

import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;

import jakarta.persistence.EntityManagerFactory;

import javax.sql.DataSource;

@Configuration

@EnableJpaRepositories(

basePackages = "com.example.EmployeeManagementSystem.repository.secondary",

entityManagerFactoryRef = "secondaryEntityManagerFactory",

transactionManagerRef = "secondaryTransactionManager"

)

public class SecondaryJpaConfig {

@Bean(name = "secondaryEntityManagerFactory")

public LocalContainerEntityManagerFactoryBean secondaryEntityManagerFactory(

@Qualifier("secondaryDataSource") DataSource dataSource) {

LocalContainerEntityManagerFactoryBean em = new LocalContainerEntityManagerFactoryBean();

em.setDataSource(dataSource);

em.setPackagesToScan("com.example.EmployeeManagementSystem.entity");

em.setJpaVendorAdapter(new HibernateJpaVendorAdapter());

return em;

}

@Bean(name = "secondaryTransactionManager")

public JpaTransactionManager secondaryTransactionManager(

@Qualifier("secondaryEntityManagerFactory") EntityManagerFactory entityManagerFactory) {

JpaTransactionManager transactionManager = new JpaTransactionManager();

transactionManager.setEntityManagerFactory(entityManagerFactory);

return transactionManager;

}

}

**PrimaryEmployeeRepository**

package com.example.EmployeeManagementSystem.repository.primary;

import com.example.EmployeeManagementSystem.entity.Employee;

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

import org.springframework.stereotype.Repository;

@Repository

public interface PrimaryEmployeeRepository extends JpaRepository<Employee, Long> {

}

**SecondaryEmployeeRespository**

package com.example.EmployeeManagementSystem.repository.secondary;

import com.example.EmployeeManagementSystem.entity.Department;

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

import org.springframework.stereotype.Repository;

@Repository

public interface SecondaryDepartmentRepository extends JpaRepository<Department, Long> {

}

**POM.xml**

<?xml version="1.0" encoding="UTF-8"?>

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

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

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

<version>3.3.2</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.example</groupId>

<artifactId>EmployeeManagementSystem</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>EmployeeManagementSystem</name>

<description>Demo project for Spring Boot</description>

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

<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>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

<configuration>

<excludes>

<exclude>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</exclude>

</excludes>

</configuration>

</plugin>

</plugins>

</build>

</project>