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

**1. Creating a Spring Boot Project:**

* **Folder:** Root Project Directory
* **Setup:**
  + Initialize a Spring Boot project named EmployeeManagementSystem.
  + Add the following dependencies in pom.xml:

<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>

<scope>provided</scope>

</dependency>

</dependencies>

**2. Configuring Application Properties:**

* **Folder:** src/main/resources
* **File:** application.properties

**Code:**

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

spring.h2.console.enabled=true

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

**1. Creating JPA Entities:**

Folder: src/main/java/com/example/EmployeeManagementSystem/entity

**Employee.java:**

package com.example.EmployeeManagementSystem.entity;

import jakarta.persistence.\*;

import lombok.Data;

@Entity

@Table(name = "employees")

@Data

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.java:**

package com.example.EmployeeManagementSystem.entity;

import jakarta.persistence.\*;

import lombok.Data;

import java.util.List;

@Entity

@Table(name = "departments")

@Data

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**

**1. Creating Repositories:**

Folder: src/main/java/com/example/EmployeeManagementSystem/repository

**EmployeeRepository.java:**

package com.example.EmployeeManagementSystem.repository;

import com.example.EmployeeManagementSystem.entity.Employee;

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

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

}

**DepartmentRepository.java:**

package com.example.EmployeeManagementSystem.repository;

import com.example.EmployeeManagementSystem.entity.Department;

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

public interface DepartmentRepository extends JpaRepository<Department, Long> {

}

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

**1. Implementing RESTful CRUD Operations:**

* **Folder:** src/main/java/com/example/EmployeeManagementSystem/controller
* **EmployeeController Code (EmployeeController.java):**

java

Copy code

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.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/employees")

public class EmployeeController {

@Autowired

private EmployeeRepository employeeRepository;

@GetMapping

public List<Employee> getAllEmployees() {

return employeeRepository.findAll();

}

@GetMapping("/{id}")

public Employee getEmployeeById(@PathVariable Long id) {

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

}

@PostMapping

public Employee createEmployee(@RequestBody Employee employee) {

return employeeRepository.save(employee);

}

@PutMapping("/{id}")

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

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

if (employee != null) {

employee.setName(employeeDetails.getName());

employee.setEmail(employeeDetails.getEmail());

employee.setDepartment(employeeDetails.getDepartment());

return employeeRepository.save(employee);

}

return null;

}

@DeleteMapping("/{id}")

public void deleteEmployee(@PathVariable Long id) {

employeeRepository.deleteById(id);

}

}

* **DepartmentController Code (DepartmentController.java):**

java

Copy code

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.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/departments")

public class DepartmentController {

@Autowired

private DepartmentRepository departmentRepository;

@GetMapping

public List<Department> getAllDepartments() {

return departmentRepository.findAll();

}

@GetMapping("/{id}")

public Department getDepartmentById(@PathVariable Long id) {

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

}

@PostMapping

public Department createDepartment(@RequestBody Department department) {

return departmentRepository.save(department);

}

@PutMapping("/{id}")

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

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

if (department != null) {

department.setName(departmentDetails.getName());

return departmentRepository.save(department);

}

return null;

}

@DeleteMapping("/{id}")

public void deleteDepartment(@PathVariable Long id) {

departmentRepository.deleteById(id);

}

}

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

**1. Defining Query Methods:**

* **Folder:** src/main/java/com/example/EmployeeManagementSystem/repository
* **Custom Query Methods in EmployeeRepository.java:**

java

Copy code

package com.example.EmployeeManagementSystem.repository;

import com.example.EmployeeManagementSystem.entity.Employee;

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;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

List<Employee> findByDepartmentName(String departmentName);

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

List<Employee> findEmployeesByName(@Param("name") String name);

}

**2. Named Queries:**

* **Folder:** src/main/java/com/example/EmployeeManagementSystem/entity
* **Named Query in Employee.java:**

java

Copy code

@Entity

@Table(name = "employees")

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

@Data

public class Employee {

// ...

}

**Exercise 6: Employee Management System - Implementing Pagination and Sorting**

**1. Pagination and Sorting:**

* **Folder:** src/main/java/com/example/EmployeeManagementSystem/controller
* **Pagination and Sorting in EmployeeController.java:**

package com.example.EmployeeManagementSystem.controller;

import com.example.EmployeeManagementSystem.entity.Employee;

import com.example.EmployeeManagementSystem.repository.EmployeeRepository;

import org.springframe

work.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.\*;

@RestController

@RequestMapping("/employees")

public class EmployeeController {

@Autowired

private EmployeeRepository employeeRepository;

@GetMapping("/page")

public Page<Employee> getAllEmployeesPaginated(

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

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

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

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

return employeeRepository.findAll(pageable);

}

}

**Exercise 7: Employee Management System - Enabling Entity Auditing**

**1. Entity Auditing:**

* **Folder:** src/main/java/com/example/EmployeeManagementSystem/entity
* **Auditing Fields in Employee.java:**

import org.springframework.data.annotation.CreatedDate;

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)

@Data

public class Employee {

// ...

@CreatedDate

private LocalDateTime createdDate;

@LastModifiedDate

private LocalDateTime lastModifiedDate;

}

* **Enable Auditing in Application Class (EmployeeManagementSystemApplication.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);

}

}

**Exercise 8: Employee Management System - Creating Projections**

**1. Creating Projections:**

* **Folder:** src/main/java/com/example/EmployeeManagementSystem/projection
* **Interface-Based Projection (EmployeeNameAndDepartment.java):**

package com.example.EmployeeManagementSystem.projection;

public interface EmployeeNameAndDepartment {

String getName();

String getDepartmentName();

}

* **Using Projections in EmployeeRepository.java:**

package com.example.EmployeeManagementSystem.repository;

import com.example.EmployeeManagementSystem.entity.Employee;

importcom.example.EmployeeManagementSystem.projection.EmployeeNameAndDepartment;

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

import java.util.List;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

List<EmployeeNameAndDepartment> findBy();

}

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

**1. Customizing Data Source Configuration:**

* **Folder:** src/main/resources
* **File:** application.properties
* **Externalize Configuration:**

properties

Copy code

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.username=sa

spring.datasource.password=password

# Secondary DataSource (example)

secondary.datasource.url=jdbc:h2:mem:seconddb

secondary.datasource.username=sa

secondary.datasource.password=password

* **Folder:** src/main/java/com/example/EmployeeManagementSystem/config
* **Configuration Class (DataSourceConfig.java):**

package com.example.EmployeeManagementSystem.config;

import org.springframework.boot.context.properties.ConfigurationProperties;

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 DataSourceConfig {

@Bean

@ConfigurationProperties(prefix = "spring.datasource")

public DataSource dataSource() {

return new DriverManagerDataSource();

}

@Bean(name = "secondaryDataSource")

@ConfigurationProperties(prefix = "secondary.datasource")

public DataSource secondaryDataSource() {

return new DriverManagerDataSource();

}

}

**Exercise 10: Employee Management System - Hibernate-Specific Features**

**1. Hibernate-Specific Features:**

* **Folder:** src/main/java/com/example/EmployeeManagementSystem/entity
* **Hibernate Annotations in Employee.java:**

import org.hibernate.annotations.BatchSize;

import org.hibernate.annotations.Fetch;

import org.hibernate.annotations.FetchMode;

@Entity

@Table(name = "employees")

@BatchSize(size = 10)

@Fetch(FetchMode.SELECT)

@Data

public class Employee {

// ...

}

* **Folder:** src/main/resources
* **File:** application.properties
* **Hibernate Properties:**

properties

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect

spring.jpa.properties.hibernate.jdbc.batch\_size=50

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

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

* **Batch Processing in EmployeeRepository.java:**

java

Copy code

import org.springframework.transaction.annotation.Transactional;

@Transactional

public void batchUpdate(List<Employee> employees) {

int batchSize = 50;

for (int i = 0; i < employees.size(); i++) {

employeeRepository.save(employees.get(i));

if (i % batchSize == 0 && i > 0) {

employeeRepository.flush();

employeeRepository.clear();

}

}

}