**WEEK-3**

**Shubhangi Priya 5008027**

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

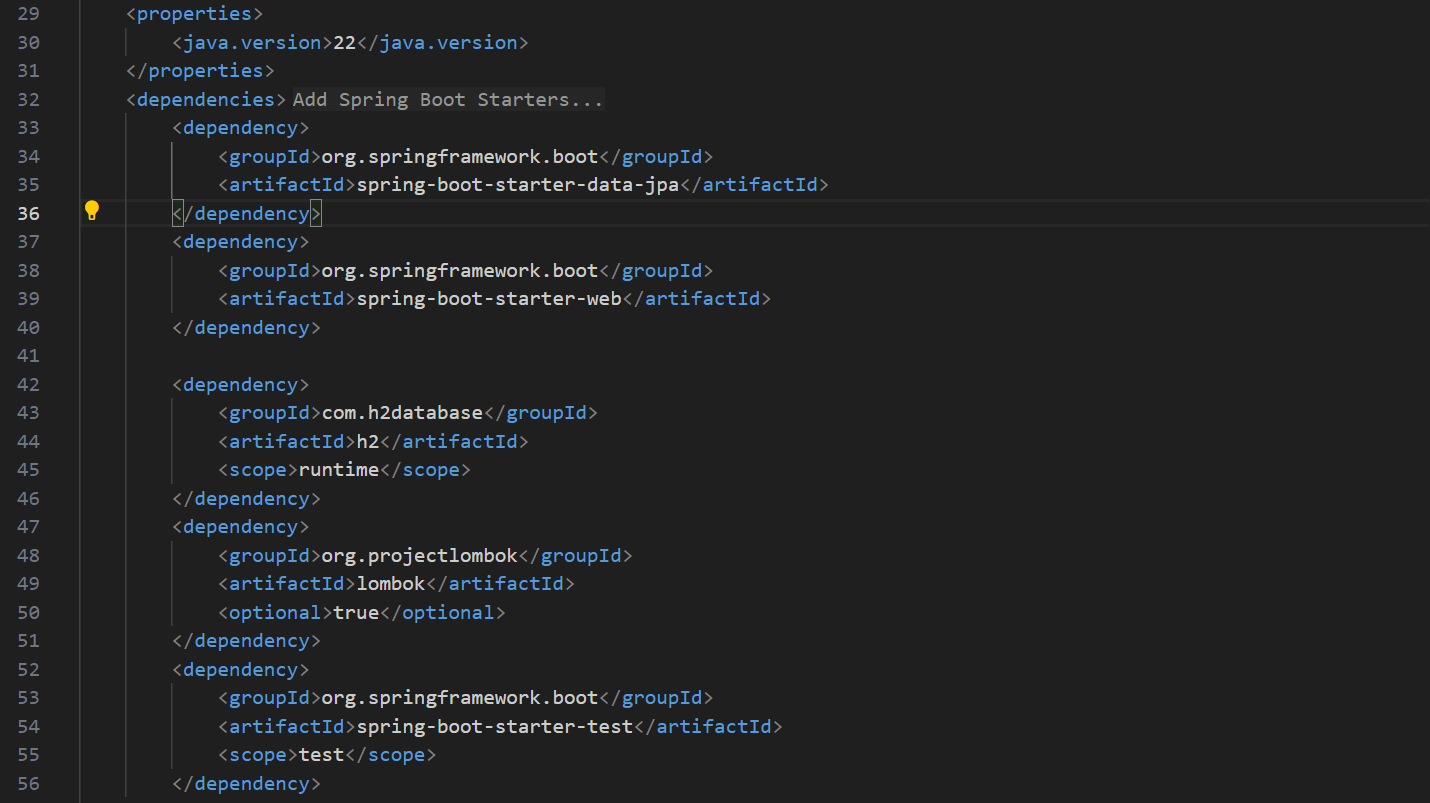
**Business Scenario:**

You are developing an employee management system that will manage employee data, departments, and their relationships.

**Instructions:**

1. **Creating a Spring Boot Project:**
   * Initialize a new Spring Boot project named **EmployeeManagementSystem**.
   * Add dependencies: **Spring Data JPA, H2 Database, Spring Web, Lombok**.

pom.xml

****

1. **Configuring Application Properties:**
   * Configure **application.properties** for H2 database connection.

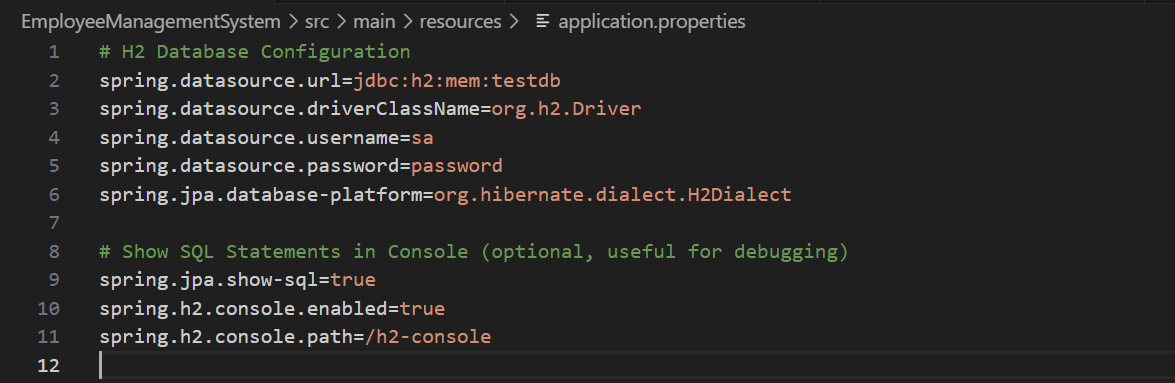
*spring.datasource.url=jdbc:h2:mem:testdb*

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

*spring.datasource.username=sa*

*spring.datasource.password=password*

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



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

**Business Scenario:**

Define JPA entities for Employee and Department with appropriate relationships.

**Instructions:**

1. **Creating JPA Entities:**
   * Define **Employee** entity with fields: **id, name, email, department**.

Employee.java

1. package com.example.EmployeeManagementSystem.model;
2. import lombok.Data;
3. import javax.persistence.\*;
4. @Entity
5. @Table(name = "employees")
6. @Data
7. public class Employee {
8. @Id
9. @GeneratedValue(strategy = GenerationType.IDENTITY)
10. private Long id;
11. private String name;
12. private String email;
13. @ManyToOne
14. @JoinColumn(name = "department\_id")
15. private Department department;
16. }
    * Define **Department** entity with fields: **id, name**.

Department.java

1. package com.example.EmployeeManagementSystem.model;
2. import lombok.Data;
3. import javax.persistence.\*;
4. import java.util.Set;
5. @Entity
6. @Table(name = "departments")
7. @Data
8. public class Department {
9. @Id
10. @GeneratedValue(strategy = GenerationType.IDENTITY)
11. private Long id;
12. private String name;
13. @OneToMany(mappedBy = "department")
14. private Set<Employee> employees;
15. }
16. **Mapping Entities to Database Tables:**
    * Use annotations like **@Entity, @Table, @Id, @GeneratedValue**, etc.
    * Define one-to-many relationship between **Department** and **Employee**.

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

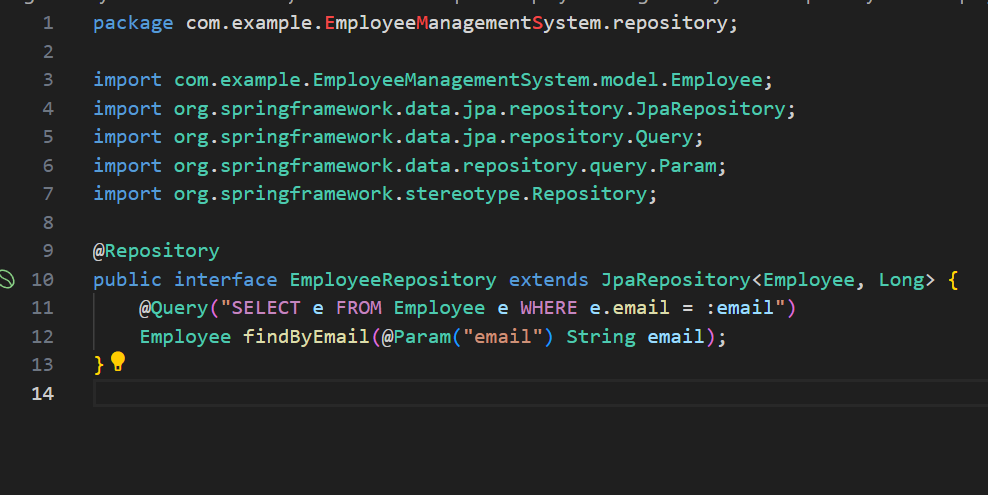
**Business Scenario:**

Create repositories for Employee and Department entities to perform CRUD operations.

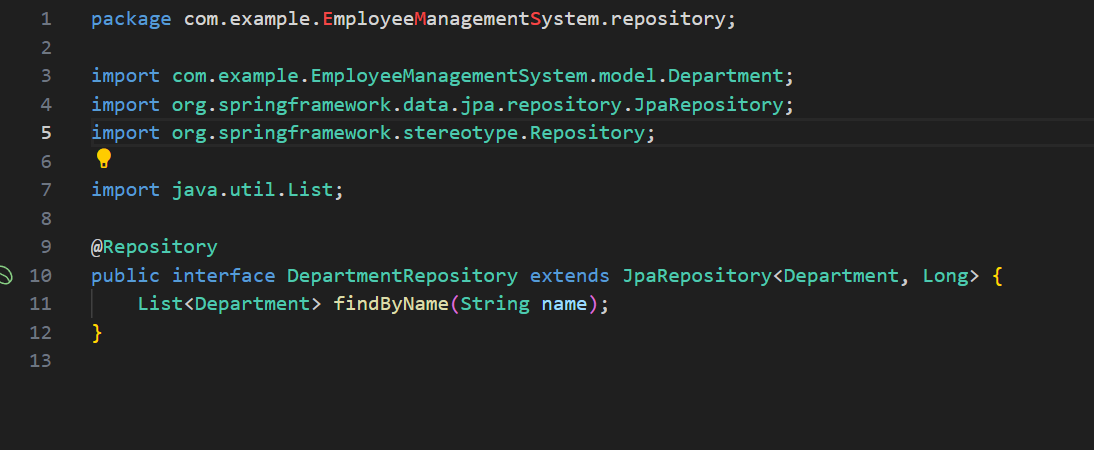
**Instructions:**

1. **Overview of Spring Data Repositories:**
   * Learn the benefits of using Spring Data repositories.
2. **Creating Repositories:**
   * Create **EmployeeRepository** and **DepartmentRepository** interfaces extending **JpaRepository**.
   * Define derived query methods in these repositories.

EmployeeRespository.java



DepartmentRepository.java



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

**Business Scenario:**

Implement CRUD operations for managing employees and departments.

**Instructions:**

1. **Basic CRUD Operations:**
   * Use **JpaRepository** methods to create, read, update, and delete employees and departments.
   * Implement RESTful endpoints for these operations using **EmployeeController** and **DepartmentController**.

DepartmentController.java

1. package com.example.EmployeeManagementSystem.controller;
2. import com.example.EmployeeManagementSystem.model.Department;
3. import com.example.EmployeeManagementSystem.service.DepartmentService;
4. import org.springframework.beans.factory.annotation.Autowired;
5. import org.springframework.http.HttpStatus;
6. import org.springframework.http.ResponseEntity;
7. import org.springframework.web.bind.annotation.\*;
8. import java.util.List;
9. @RestController
10. @RequestMapping("/departments")
11. public class DepartmentController {
12. @Autowired
13. private DepartmentService departmentService;
14. @GetMapping
15. public List<Department> getAllDepartments() {
16. return departmentService.findAll();
17. }
18. @GetMapping("/{id}")
19. public ResponseEntity<Department> getDepartmentById(@PathVariable Long id) {
20. return departmentService.findById(id)
21. .map(department -> new ResponseEntity<>(department, HttpStatus.OK))
22. .orElse(new ResponseEntity<>(HttpStatus.NOT\_FOUND));
23. }
24. @PostMapping
25. public ResponseEntity<Department> createDepartment(@RequestBody Department department) {
26. Department createdDepartment = departmentService.save(department);
27. return new ResponseEntity<>(createdDepartment, HttpStatus.CREATED);
28. }
29. @PutMapping("/{id}")
30. public ResponseEntity<Department> updateDepartment(@PathVariable Long id, @RequestBody Department department) {
31. return departmentService.update(id, department)
32. .map(updatedDepartment -> new ResponseEntity<>(updatedDepartment, HttpStatus.OK))
33. .orElse(new ResponseEntity<>(HttpStatus.NOT\_FOUND));
34. }
35. @DeleteMapping("/{id}")
36. public ResponseEntity<Void> deleteDepartment(@PathVariable Long id) {
37. if (departmentService.delete(id)) {
38. return new ResponseEntity<>(HttpStatus.NO\_CONTENT);
39. }
40. return new ResponseEntity<>(HttpStatus.NOT\_FOUND);
41. }
42. }

EmployeeController.java

1. package com.example.EmployeeManagementSystem.controller;
2. import com.example.EmployeeManagementSystem.model.Employee;
3. import com.example.EmployeeManagementSystem.service.EmployeeService;
4. import org.springframework.beans.factory.annotation.Autowired;
5. import org.springframework.http.HttpStatus;
6. import org.springframework.http.ResponseEntity;
7. import org.springframework.web.bind.annotation.\*;
8. import java.util.List;
9. @RestController
10. @RequestMapping("/employees")
11. public class EmployeeController {
12. @Autowired
13. private EmployeeService employeeService;
14. @GetMapping
15. public List<Employee> getAllEmployees() {
16. return employeeService.findAll();
17. }
18. @GetMapping("/{id}")
19. public ResponseEntity<Employee> getEmployeeById(@PathVariable Long id) {
20. return employeeService.findById(id)
21. .map(employee -> new ResponseEntity<>(employee, HttpStatus.OK))
22. .orElse(new ResponseEntity<>(HttpStatus.NOT\_FOUND));
23. }
24. @PostMapping
25. public ResponseEntity<Employee> createEmployee(@RequestBody Employee employee) {
26. Employee createdEmployee = employeeService.save(employee);
27. return new ResponseEntity<>(createdEmployee, HttpStatus.CREATED);
28. }
29. @PutMapping("/{id}")
30. public ResponseEntity<Employee> updateEmployee(@PathVariable Long id, @RequestBody Employee employee) {
31. return employeeService.update(id, employee)
32. .map(updatedEmployee -> new ResponseEntity<>(updatedEmployee, HttpStatus.OK))
33. .orElse(new ResponseEntity<>(HttpStatus.NOT\_FOUND));
34. }
35. @DeleteMapping("/{id}")
36. public ResponseEntity<Void> deleteEmployee(@PathVariable Long id) {
37. if (employeeService.delete(id)) {
38. return new ResponseEntity<>(HttpStatus.NO\_CONTENT);
39. }
40. return new ResponseEntity<>(HttpStatus.NOT\_FOUND);
41. }
42. }

EmployeeService.java

1. package com.example.EmployeeManagementSystem.service;
2. import com.example.EmployeeManagementSystem.model.Employee;
3. import com.example.EmployeeManagementSystem.repository.EmployeeRepository;
4. import org.springframework.beans.factory.annotation.Autowired;
5. import org.springframework.stereotype.Service;
6. import java.util.List;
7. import java.util.Optional;
8. @Service
9. public class EmployeeService {
10. @Autowired
11. private EmployeeRepository employeeRepository;
12. public List<Employee> findAll() {
13. return employeeRepository.findAll();
14. }
15. public Optional<Employee> findById(Long id) {
16. return employeeRepository.findById(id);
17. }
18. public Employee save(Employee employee) {
19. return employeeRepository.save(employee);
20. }
21. public Optional<Employee> update(Long id, Employee employee) {
22. if (employeeRepository.existsById(id)) {
23. employee.setId(id);
24. return Optional.of(employeeRepository.save(employee));
25. }
26. return Optional.empty();
27. }
28. public boolean delete(Long id) {
29. if (employeeRepository.existsById(id)) {
30. employeeRepository.deleteById(id);
31. return true;
32. }
33. return false;
34. }
35. }

DepartmentService.java

1. package com.example.EmployeeManagementSystem.service;
2. import com.example.EmployeeManagementSystem.model.Department;
3. import com.example.EmployeeManagementSystem.repository.DepartmentRepository;
4. import org.springframework.beans.factory.annotation.Autowired;
5. import org.springframework.stereotype.Service;
6. import java.util.List;
7. import java.util.Optional;
8. @Service
9. public class DepartmentService {
10. @Autowired
11. private DepartmentRepository departmentRepository;
12. public List<Department> findAll() {
13. return departmentRepository.findAll();
14. }
15. public Optional<Department> findById(Long id) {
16. return departmentRepository.findById(id);
17. }
18. public Department save(Department department) {
19. return departmentRepository.save(department);
20. }
21. public Optional<Department> update(Long id, Department department) {
22. if (departmentRepository.existsById(id)) {
23. department.setId(id);
24. return Optional.of(departmentRepository.save(department));
25. }
26. return Optional.empty();
27. }
28. public boolean delete(Long id) {
29. if (departmentRepository.existsById(id)) {
30. departmentRepository.deleteById(id);
31. return true;
32. }
33. return false;
34. }
35. }

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

**Business Scenario:**

Enhance your repository to support custom queries.

**Instructions:**

1. **Defining Query Methods:**
   * Use keywords in method names to create custom query methods.
   * Implement custom query methods using the **@Query** annotation.
2. **Named Queries:**
   * Define and execute named queries with **@NamedQuery** and **@NamedQueries**.

EmployeeRepository.java

1. package com.example.EmployeeManagementSystem.repository;
2. import com.example.EmployeeManagementSystem.model.Employee;
3. import org.springframework.data.jpa.repository.JpaRepository;
4. import org.springframework.data.jpa.repository.Query;
5. import org.springframework.data.repository.query.Param;
6. import org.springframework.stereotype.Repository;
7. @Repository
8. public interface EmployeeRepository extends JpaRepository<Employee, Long> {
9. // Custom query method to find an employee by email
10. @Query("SELECT e FROM Employee e WHERE e.email = :email")
11. Employee findByEmail(@Param("email") String email);
12. // Custom query method to find employees by department name
13. @Query("SELECT e FROM Employee e WHERE e.department.name = :departmentName")
14. List<Employee> findByDepartmentName(@Param("departmentName") String departmentName);
15. }

Employee.java

1. package com.example.EmployeeManagementSystem.model;
2. import lombok.Data;
3. import javax.persistence.\*;
4. import java.util.Set;
5. @Entity
6. @Table(name = "employees")
7. @NamedQueries({
8. @NamedQuery(name = "Employee.findByEmail", query = "SELECT e FROM Employee e WHERE e.email = :email"),
9. @NamedQuery(name = "Employee.findByDepartmentName", query = "SELECT e FROM Employee e WHERE e.department.name = :departmentName")
10. })
11. @Data
12. public class Employee {
13. @Id
14. @GeneratedValue(strategy = GenerationType.IDENTITY)
15. private Long id;
16. private String name;
17. private String email;
18. @ManyToOne
19. @JoinColumn(name = "department\_id")
20. private Department department;
21. }

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

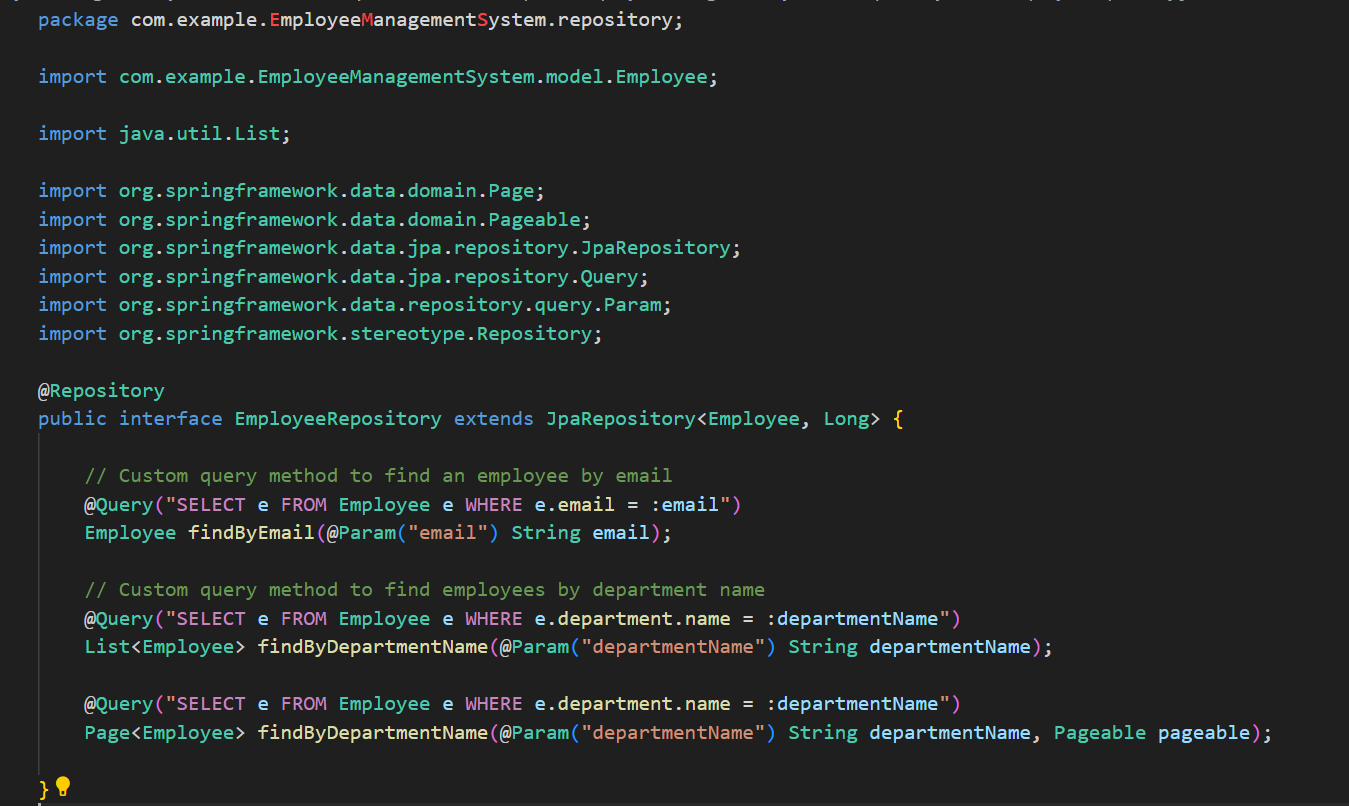
**Business Scenario:**

Add pagination and sorting capabilities to your employee search functionality.

**Instructions:**

1. **Pagination:**
   * Implement pagination for the employee list using **Page** and **Pageable**.
2. **Sorting:**
   * Add sorting functionality to your queries.
   * Combine pagination and sorting in your search endpoint.

EmployeeRepository.java



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

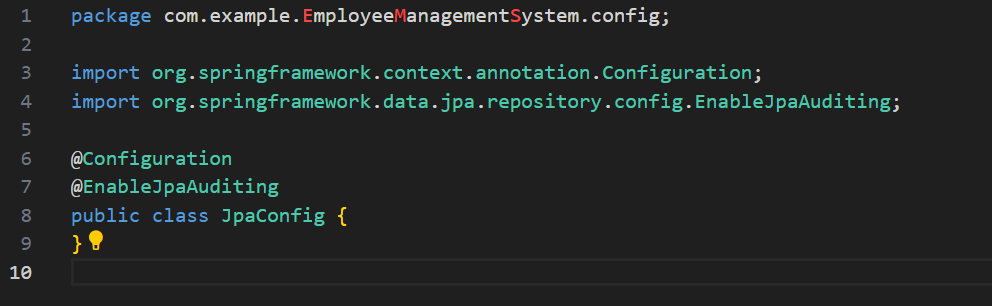
**Business Scenario:**

Implement auditing to track the creation and modification of employees and departments.

**Instructions:**

1. **Entity Auditing:**
   * Enable auditing in your application by configuring auditing properties.
   * Use annotations like **@CreatedBy, @LastModifiedBy, @CreatedDate**, and **@LastModifiedDate**.

JpaConfig.java



Employee.java

1. package com.example.EmployeeManagementSystem.model;
2. import lombok.Data;
3. import org.springframework.data.annotation.CreatedDate;
4. import org.springframework.data.annotation.LastModifiedDate;
5. import org.springframework.data.jpa.domain.support.AuditingEntityListener;
6. import javax.persistence.\*;
7. import java.time.LocalDateTime;
8. import java.util.Set;
9. @Entity
10. @Table(name = "employees")
11. @EntityListeners(AuditingEntityListener.class)
12. @Data
13. public class Employee {
14. @Id
15. @GeneratedValue(strategy = GenerationType.IDENTITY)
16. private Long id;
17. private String name;
18. private String email;
19. @ManyToOne
20. @JoinColumn(name = "department\_id")
21. private Department department;
22. @CreatedDate
23. @Column(name = "created\_date", updatable = false)
24. private LocalDateTime createdDate;
25. @LastModifiedDate
26. @Column(name = "last\_modified\_date")
27. private LocalDateTime lastModifiedDate;
28. }

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

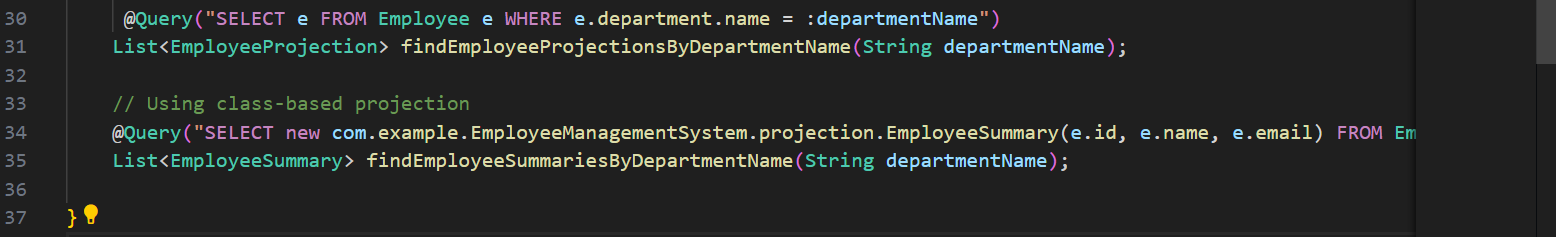
**Business Scenario:**

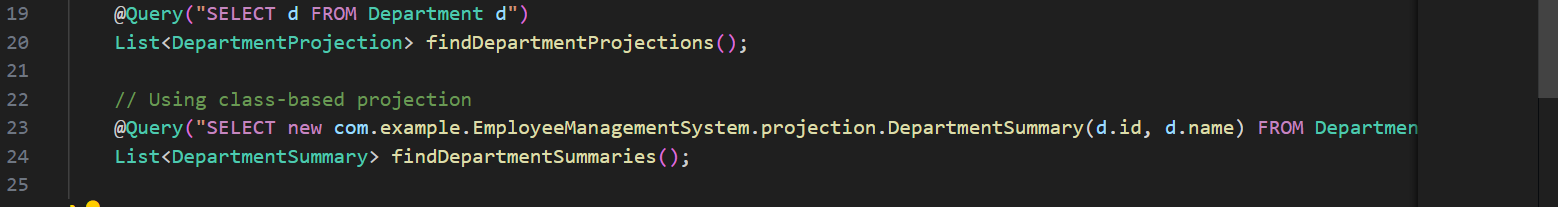
Create projections to fetch specific data subsets from the employee and department entities.

**Instructions:**

1. **Projections:**
   * Define interface-based and class-based projections.
   * Use **@Value** and constructor expressions to control the fetched data.

EmployeeRepository.java

****DepartmentRepository.java



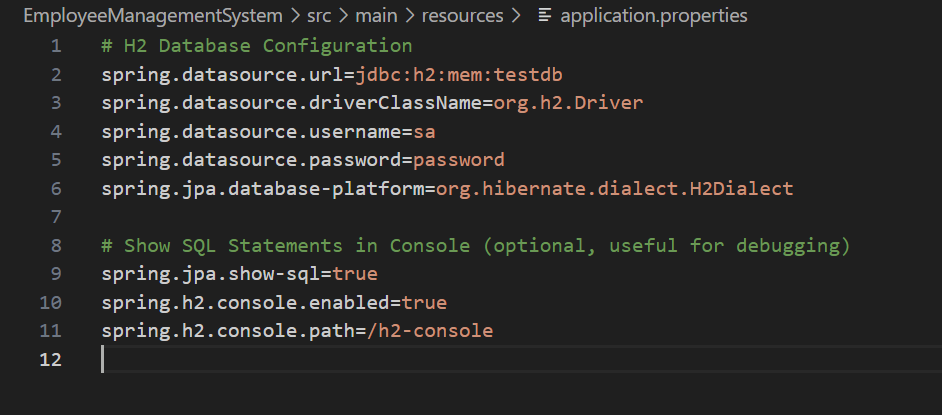
**Exercise 9: Employee Management System - Customizing Data Source Configuration**

**Business Scenario:**

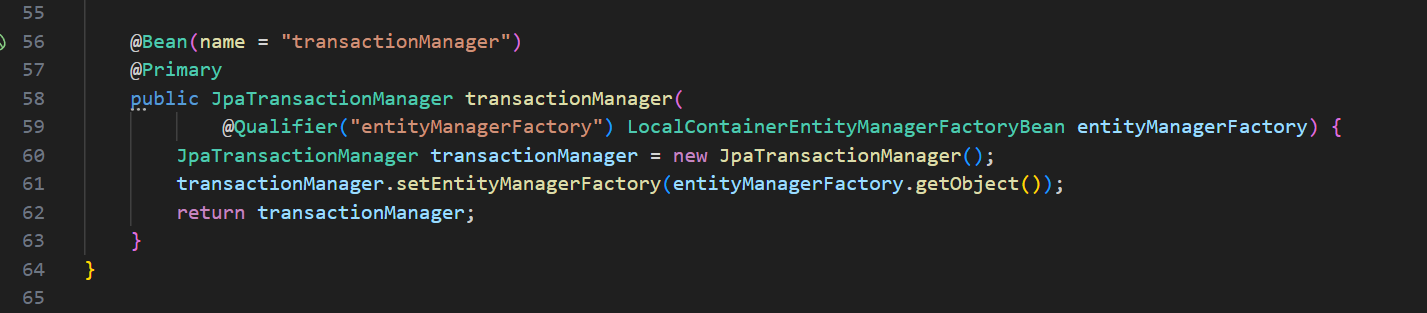
Customize your data source configuration and manage multiple data sources.

**Instructions:**

1. **Spring Boot Auto-Configuration:**
   * Leverage Spring Boot auto-configuration for data sources.
2. **Externalizing Configuration:**
   * Externalize configuration with application.properties.
   * Manage multiple data sources within your application.



DataSourceConfig.java

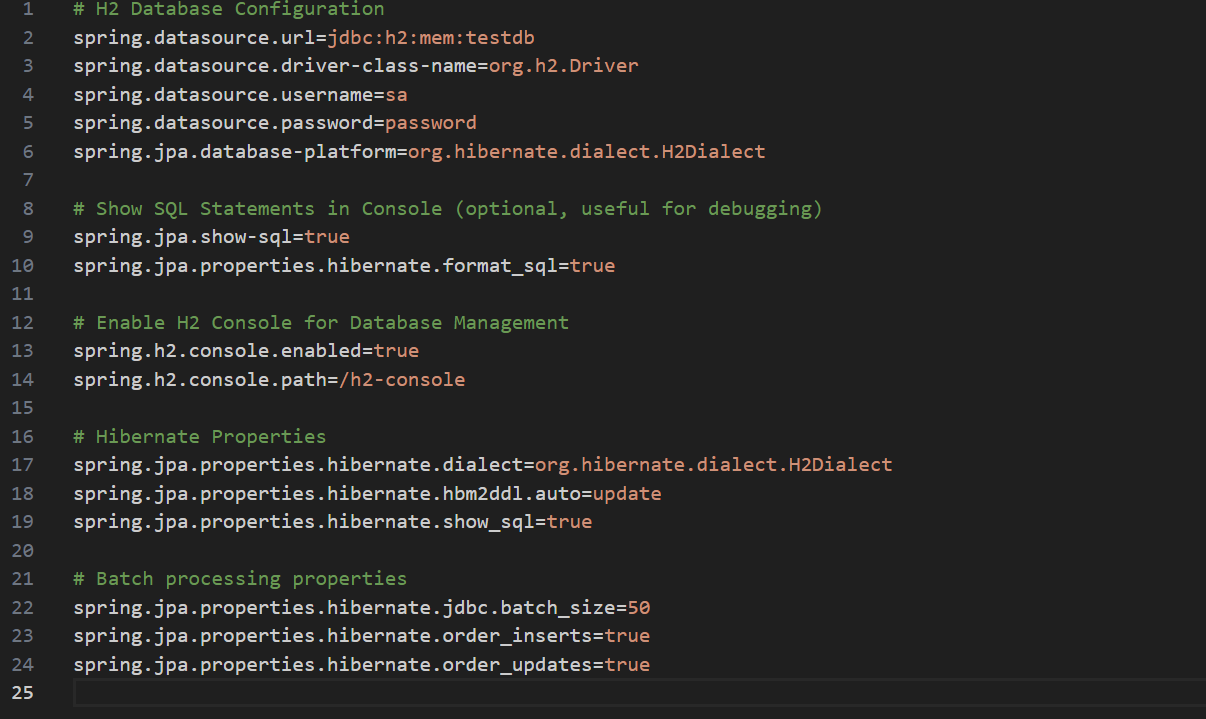
**Exercise 10: Employee Management System - Hibernate-Specific Features**

**Business Scenario:**

Leverage Hibernate-specific features to enhance your application's performance and capabilities.

**Instructions:**

1. **Hibernate-Specific Annotations:**
   * Use Hibernate-specific annotations to customize entity mappings.
2. **Configuring Hibernate Dialect and Properties:**
   * Configure Hibernate dialect and properties for optimal performance.
3. **Batch Processing:**
   * Implement batch processing with Hibernate for bulk operations.



SHUBHANGI PRIYA

[shubhupriya123@gmail.com](mailto:shubhupriya123@gmail.com)

5008027