**Exercise 1: Spring Boot Project Setup**

1. **pom.xml** - Add the required dependencies:

<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>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<scope>provided</scope>

</dependency>

</dependencies>

1. **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

**Exercise 2: Creating JPA Entities**

1. **Employee.java**:

import javax.persistence.\*;

import lombok.\*;

@Entity

@Table(name = "employees")

@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;

}

1. **Department.java**:

import javax.persistence.\*;

import lombok.\*;

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;

@OneToMany(mappedBy = "department", cascade = CascadeType.ALL, fetch = FetchType.LAZY)

private List<Employee> employees;

}

**Exercise 3: Creating Repositories**

1. **EmployeeRepository.java**:

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

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

}

1. **DepartmentRepository.java**:

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

public interface DepartmentRepository extends JpaRepository<Department, Long> {

}

**Exercise 4: Implementing CRUD Operations**

1. **EmployeeController.java**:

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();

}

@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).orElseThrow();

employee.setName(employeeDetails.getName());

employee.setEmail(employeeDetails.getEmail());

return employeeRepository.save(employee);

}

@DeleteMapping("/{id}")

public void deleteEmployee(@PathVariable Long id) {

employeeRepository.deleteById(id);

}

}

1. **DepartmentController.java**:

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();

}

@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).orElseThrow();

department.setName(departmentDetails.getName());

return departmentRepository.save(department);

}

@DeleteMapping("/{id}")

public void deleteDepartment(@PathVariable Long id) {

departmentRepository.deleteById(id);

}

}

**Exercise 5: Defining Query Methods**

1. **Custom Query for EmployeeRepository**:

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

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

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

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

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

}

**Exercise 6: Implementing Pagination and Sorting**

1. **Pagination and Sorting in EmployeeController**:

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.GetMapping;

import org.springframework.web.bind.annotation.RequestParam;

@RestController

@RequestMapping("/employees")

public class EmployeeController {

@Autowired

private EmployeeRepository employeeRepository;

@GetMapping("/paginated")

public Page<Employee> getPaginatedEmployees(@RequestParam int page, @RequestParam int size) {

Pageable pageable = PageRequest.of(page, size, Sort.by("name").ascending());

return employeeRepository.findAll(pageable);

}

}

**Exercise 7: Enabling Entity Auditing**

1. **Enable Auditing in Application Class**:

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);

}

}

1. **Audit Fields in Employee Entity**:

import org.springframework.data.annotation.CreatedDate;

import org.springframework.data.annotation.LastModifiedDate;

import org.springframework.data.jpa.domain.support.AuditingEntityListener;

import javax.persistence.\*;

import java.time.LocalDateTime;

@EntityListeners(AuditingEntityListener.class)

@Entity

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

private LocalDateTime createdDate;

@LastModifiedDate

private LocalDateTime lastModifiedDate;

}

1. **Audit Fields in Department Entity**:

import javax.persistence.\*;

import java.time.LocalDateTime;

import org.springframework.data.annotation.CreatedDate;

import org.springframework.data.annotation.LastModifiedDate;

import org.springframework.data.jpa.domain.support.AuditingEntityListener;

@EntityListeners(AuditingEntityListener.class)

@Entity

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@CreatedDate

private LocalDateTime createdDate;

@LastModifiedDate

private LocalDateTime lastModifiedDate;

}

**Exercise 8: Creating Projections**

1. **Define Interface-Based Projection** (Employee Name and Email Projection):

public interface EmployeeNameEmailProjection {

String getName();

String getEmail();

}

1. **Using Projections in Repository**:

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

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

List<EmployeeNameEmailProjection> findAllProjectedBy();

}

**Exercise 9: Customizing Data Source Configuration**

1. **application.properties**

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

spring.datasource.username=employee\_user

spring.datasource.password=employee\_password

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

spring.jpa.hibernate.ddl-auto=update

1. **Multiple Data Source Configuration** :

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

import org.springframework.boot.jdbc.DataSourceBuilder;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import javax.sql.DataSource;

@Configuration

public class DataSourceConfig {

@Bean(name = "employeeDataSource")

public DataSource employeeDataSource() {

return DataSourceBuilder.create()

.url("jdbc:h2:mem:employeeDb")

.username("employee\_user")

.password("employee\_password")

.build();

}

@Bean(name = "departmentDataSource")

public DataSource departmentDataSource() {

return DataSourceBuilder.create()

.url("jdbc:h2:mem:departmentDb")

.username("department\_user")

.password("department\_password")

.build();

}

}

**Exercise 10: Hibernate-Specific Features**

1. **Using Hibernate-Specific**

import org.hibernate.annotations.BatchSize;

@Entity

@BatchSize(size = 30)

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;

}

1. **Configuring Hibernate Properties for Optimal Performance**:

In application.properties:

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

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

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

1. **Batch Processing in Repository**:

import org.springframework.transaction.annotation.Transactional;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void saveAllEmployees(List<Employee> employees) {

employeeRepository.saveAll(employees);

}

}