**Spring Data JPA and Hibernate**

**Exercise 1: Overview and Setup**

Create a Spring Boot project named EmployeeManagementSystem with dependencies: Spring Data JPA, H2 Database, Spring Web, and Lombok.

# 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 Entities**

Define the Employee and Department entities with appropriate relationships.

// Employee.java

@Entity

@Table(name = "employees")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

@ManyToOne

@JoinColumn(name = "department\_id", nullable = false)

private Department department;

// getters and setters

}

// Department.java

@Entity

@Table(name = "departments")

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

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

private List<Employee> employees = new ArrayList<>();

// getters and setters

}

**Exercise 3: Creating Repositories**

Create repositories for Employee and Department entities.

// EmployeeRepository.java

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

List<Employee> findByDepartmentId(Long departmentId);

}

// DepartmentRepository.java

public interface DepartmentRepository extends JpaRepository<Department, Long> {

}

**Exercise 4: Implementing CRUD Operations**

Implement RESTful CRUD operations.

// EmployeeController.java

@RestController

@RequestMapping("/api/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 ResponseEntity<Employee> updateEmployee(@PathVariable Long id, @RequestBody Employee employeeDetails) {

Employee employee = employeeRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Employee not found"));

employee.setName(employeeDetails.getName());

employee.setEmail(employeeDetails.getEmail());

employee.setDepartment(employeeDetails.getDepartment());

Employee updatedEmployee = employeeRepository.save(employee);

return ResponseEntity.ok(updatedEmployee);

}

@DeleteMapping("/{id}")

public Map<String, Boolean> deleteEmployee(@PathVariable Long id) {

Employee employee = employeeRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Employee not found"));

employeeRepository.delete(employee);

Map<String, Boolean> response = new HashMap<>();

response.put("deleted", Boolean.TRUE);

return response;

}

}

// DepartmentController.java

@RestController

@RequestMapping("/api/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 ResponseEntity<Department> updateDepartment(@PathVariable Long id, @RequestBody Department departmentDetails) {

Department department = departmentRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Department not found"));

department.setName(departmentDetails.getName());

Department updatedDepartment = departmentRepository.save(department);

return ResponseEntity.ok(updatedDepartment);

}

@DeleteMapping("/{id}")

public Map<String, Boolean> deleteDepartment(@PathVariable Long id) {

Department department = departmentRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Department not found"));

departmentRepository.delete(department);

Map<String, Boolean> response = new HashMap<>();

response.put("deleted", Boolean.TRUE);

return response;

}

}

**Exercise 5: Defining Query Methods**

Define custom query methods using method names and annotations.

// EmployeeRepository.java

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

List<Employee> findByNameContaining(String name);

@Query("SELECT e FROM Employee e WHERE e.email = ?1")

Employee findByEmailAddress(String email);

}

**Exercise 6: Implementing Pagination and Sorting**

Add pagination and sorting to employee search functionality.

// EmployeeController.java

@GetMapping("/paged")

public Page<Employee> getAllEmployeesPaged(Pageable pageable) {

return employeeRepository.findAll(pageable);

}

@GetMapping("/sorted")

public List<Employee> getAllEmployeesSorted(Sort sort) {

return employeeRepository.findAll(sort);

}

**Exercise 7: Enabling Entity Auditing**

Enable auditing for tracking creation and modification.

// AuditModel.java

@MappedSuperclass

@EntityListeners(AuditingEntityListener.class)

public abstract class AuditModel {

@CreatedDate

@Temporal(TemporalType.TIMESTAMP)

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

private Date createdAt;

@LastModifiedDate

@Temporal(TemporalType.TIMESTAMP)

@Column(name = "updated\_at", nullable = false)

private Date updatedAt;

// getters and setters

}

// Employee.java and Department.java extend AuditModel

**Exercise 8: Creating Projections**

Create projections to fetch specific data subsets.

// EmployeeProjection.java

public interface EmployeeProjection {

String getName();

String getDepartmentName();

}

// EmployeeRepository.java

@Query("SELECT e.name as name, d.name as departmentName FROM Employee e JOIN e.department d")

List<EmployeeProjection> findEmployeeNameAndDepartment();

Here are the code snippets for each exercise in the Employee Management System project using Spring Data JPA and Hibernate:

**Exercise 1: Overview and Setup**

Create a Spring Boot project named EmployeeManagementSystem with dependencies: Spring Data JPA, H2 Database, Spring Web, and Lombok.

properties

Copy code

# 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 Entities**

Define the Employee and Department entities with appropriate relationships.

java

Copy code

// Employee.java

@Entity

@Table(name = "employees")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

@ManyToOne

@JoinColumn(name = "department\_id", nullable = false)

private Department department;

// getters and setters

}

// Department.java

@Entity

@Table(name = "departments")

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

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

private List<Employee> employees = new ArrayList<>();

// getters and setters

}

**Exercise 3: Creating Repositories**

Create repositories for Employee and Department entities.

java

Copy code

// EmployeeRepository.java

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

List<Employee> findByDepartmentId(Long departmentId);

}

// DepartmentRepository.java

public interface DepartmentRepository extends JpaRepository<Department, Long> {

}

**Exercise 4: Implementing CRUD Operations**

Implement RESTful CRUD operations.

java

Copy code

// EmployeeController.java

@RestController

@RequestMapping("/api/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 ResponseEntity<Employee> updateEmployee(@PathVariable Long id, @RequestBody Employee employeeDetails) {

Employee employee = employeeRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Employee not found"));

employee.setName(employeeDetails.getName());

employee.setEmail(employeeDetails.getEmail());

employee.setDepartment(employeeDetails.getDepartment());

Employee updatedEmployee = employeeRepository.save(employee);

return ResponseEntity.ok(updatedEmployee);

}

@DeleteMapping("/{id}")

public Map<String, Boolean> deleteEmployee(@PathVariable Long id) {

Employee employee = employeeRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Employee not found"));

employeeRepository.delete(employee);

Map<String, Boolean> response = new HashMap<>();

response.put("deleted", Boolean.TRUE);

return response;

}

}

// DepartmentController.java

@RestController

@RequestMapping("/api/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 ResponseEntity<Department> updateDepartment(@PathVariable Long id, @RequestBody Department departmentDetails) {

Department department = departmentRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Department not found"));

department.setName(departmentDetails.getName());

Department updatedDepartment = departmentRepository.save(department);

return ResponseEntity.ok(updatedDepartment);

}

@DeleteMapping("/{id}")

public Map<String, Boolean> deleteDepartment(@PathVariable Long id) {

Department department = departmentRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Department not found"));

departmentRepository.delete(department);

Map<String, Boolean> response = new HashMap<>();

response.put("deleted", Boolean.TRUE);

return response;

}

}

**Exercise 5: Defining Query Methods**

Define custom query methods using method names and annotations.

java

Copy code

// EmployeeRepository.java

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

List<Employee> findByNameContaining(String name);

@Query("SELECT e FROM Employee e WHERE e.email = ?1")

Employee findByEmailAddress(String email);

}

**Exercise 6: Implementing Pagination and Sorting**

Add pagination and sorting to employee search functionality.

java

Copy code

// EmployeeController.java

@GetMapping("/paged")

public Page<Employee> getAllEmployeesPaged(Pageable pageable) {

return employeeRepository.findAll(pageable);

}

@GetMapping("/sorted")

public List<Employee> getAllEmployeesSorted(Sort sort) {

return employeeRepository.findAll(sort);

}

**Exercise 7: Enabling Entity Auditing**

Enable auditing for tracking creation and modification.

java

Copy code

// AuditModel.java

@MappedSuperclass

@EntityListeners(AuditingEntityListener.class)

public abstract class AuditModel {

@CreatedDate

@Temporal(TemporalType.TIMESTAMP)

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

private Date createdAt;

@LastModifiedDate

@Temporal(TemporalType.TIMESTAMP)

@Column(name = "updated\_at", nullable = false)

private Date updatedAt;

// getters and setters

}

// Employee.java and Department.java extend AuditModel

**Exercise 8: Creating Projections**

Create projections to fetch specific data subsets.

java

Copy code

// EmployeeProjection.java

public interface EmployeeProjection {

String getName();

String getDepartmentName();

}

// EmployeeRepository.java

@Query("SELECT e.name as name, d.name as departmentName FROM Employee e JOIN e.department d")

List<EmployeeProjection> findEmployeeNameAndDepartment();

**Exercise 9: Customizing Data Source Configuration**

Customize and manage multiple data sources.

# application.properties for multiple data sources

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

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

# Configurations for multiple DataSources

@Configuration

public class DataSourceConfig {

@Bean

@Primary

@ConfigurationProperties(prefix = "spring.datasource.primary")

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**

Leverage Hibernate-specific features.

// Hibernate-specific annotations in entities

@Entity

@Table(name = "employees")

@org.hibernate.annotations.DynamicUpdate

public class Employee {

// Fields and methods

}

// Batch processing configuration in application.properties

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

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