SPRING DATA JPA AND HIBERNATE

Superset ID : 6384831

Name : Mohana Priya N

E-mail : mohanapriya.2205056@srec.ac.in

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

Solution:

//pom.xml

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
</dependencies>
```

//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
spring.h2.console.enabled=true
spring.jpa.hibernate.ddl-auto=update
```

2) Exercise 2: Employee Management System - Creating Entities

Solution:

//Department.java

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
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<>();
}
```

//Employee.java

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;
    private String name;
    private String email;
    @ManyToOne
    private Department department;
}
```

3) Exercise 3 : Employee Management System - Creating Repositories

Solution:

//EmployeeRepository.java

```
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
    List<Employee> findByDepartmentName(String name);
}
```

//DepartmentRepository.java

```
public interface DepartmentRepository extends JpaRepository<Department, Long> {
}
```

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

Solution:

//EmployeeController.java

```
@RestController
@RequestMapping("/employees")
@RequiredArgsConstructor
public class EmployeeController {
    private final EmployeeRepository employeeRepository;
    @GetMapping
    public List<Employee> getAll() {
        return employeeRepository.findAll();
    }
```

```
    @PostMapping
    public Employee create(@RequestBody Employee employee) {
        return employeeRepository.save(employee);
    }
    @PutMapping("/{id}")
    public Employee update(@PathVariable Long id, @RequestBody Employee employee) {
        employee.setId(id);
        return employeeRepository.save(employee);
    }
    @DeleteMapping("/{id}")
    public void delete(@PathVariable Long id) {
        employeeRepository.deleteById(id);
    }
}
```

//DepartmentController.java

```
@RestController
@RequestMapping("/departments")
@RequiredArgsConstructor
public class DepartmentController {
    private final DepartmentRepository departmentRepository;
    @GetMapping
    public List<Department> getAll() {
        return departmentRepository.findAll();
    }
    @PostMapping
    public Department create(@RequestBody Department department) {
        return departmentRepository.save(department);
    }
}
```

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

Solution:

//EmployeeRepository.java

```
package com.example.employeemanagement.repository;
import com.example.employeemanagement.entity.Employee;
import com.example.employeemanagement.projection.EmployeeNameView;
import com.example.employeemanagement.dto.EmployeeDTO;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import java.util.List;
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
    List<Employee> findByDepartmentName(String name);
    List<Employee> findByNameContaining(String keyword);
    @Query("SELECT e FROM Employee e WHERE e.email = ?1")
    Employee findByEmail(String email);
    List<EmployeeNameView> findByDepartmentId(Long departmentId);
    @Query("SELECT new com.example.employeemanagement.dto.EmployeeDTO(e.name,
e.email) FROM Employee e")
    List<EmployeeDTO> fetchEmployeeDetails();
}
```

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

Solution:

//EmployeeController.java

```
@RestController
@RequestMapping("/employees")
@RequiredArgsConstructor
public class EmployeeController {
    private final EmployeeRepository employeeRepository;
    @GetMapping("/paged")
public Page<Employee> getPagedEmployees(
```

```
        @RequestParam int page,
        @RequestParam int size,
        @RequestParam(defaultValue = "id") String sortBy
    ) {
        return employeeRepository.findAll(PageRequest.of(page, size, Sort.by(sortBy)));
    }
    @PostMapping
    public Employee create(@RequestBody Employee employee) {
        return employeeRepository.save(employee);
    }
    @PutMapping("/{id}")
    public Employee update(@PathVariable Long id, @RequestBody Employee employee) {
        employee.setId(id);
        return employeeRepository.save(employee);
    }
    @DeleteMapping("/{id}")
    public void delete(@PathVariable Long id) {
        employeeRepository.deleteById(id);
    }
}
```

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

Solution:

//pom.xml

```
<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>
```

//EmployeeManagementSystemApplication.java

```
package com.example.employeemanagement;
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);
    }
}
```

//AuditConfig.java

```
package com.example.employeemanagement.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
public class AuditConfig {
    @Bean
    public AuditorAware<String> auditorProvider() {
        return () -> Optional.of("admin"); // hardcoded user for demo
    }
}
```

//Employee.java

```
package com.example.employeemanagement.entity;
import lombok.*;
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 org.springframework.data.jpa.domain.support.AuditingEntityListener;
import javax.persistence.*;
import java.time.LocalDateTime;
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@EntityListeners(AuditingEntityListener.class)
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;
    @ManyToOne
    private Department department;
```

```
    @CreatedBy
    private String createdBy;
    @LastModifiedBy
    private String modifiedBy;
    @CreatedDate
    private LocalDateTime createdDate;
    @LastModifiedDate
    private LocalDateTime modifiedDate;
}
```

//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
spring.jpa.hibernate.ddl-auto=update
spring.h2.console.enabled=true
```

8) Exercise 8 : Employee Management System - Creating Projections

Solution:

//EmployeeNameView.java

```
package com.example.employeemanagement.projection;
public interface EmployeeNameView {
    String getName();
}
```

//EmployeeDTO.java

```
package com.example.employeemanagement.dto;
public class EmployeeDTO {
    private String name;
    private String email;
```

```
    public EmployeeDTO(String name, String email) {
        this.name = name;
        this.email = email;
    }
    public String getName() {
         return name;
    }
    public String getEmail() {
         return email;
    }
}
```

//EmployeeRepository.java

```
package com.example.employeemanagement.repository;
import com.example.employeemanagement.entity.Employee;
import com.example.employeemanagement.dto.EmployeeDTO;
import com.example.employeemanagement.projection.EmployeeNameView;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import java.util.List;
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
    List<EmployeeNameView> findByDepartmentId(Long departmentId);
    @Query("SELECT new com.example.employeemanagement.dto.EmployeeDTO(e.name,
e.email) FROM Employee e")
    List<EmployeeDTO> fetchEmployeeDetails();
}
```

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

Solution:

//application.properties

```
spring.datasource.url=jdbc:h2:mem:primarydb
spring.datasource.username=sa
spring.datasource.password=password
```

```
spring.datasource.driver-class-name=org.h2.Driver
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update
```

//SecondaryDataSourceConfig.java

```
@Configuration
@EnableTransactionManagement
@EnableJpaRepositories(
        basePackages = "com.example.secondary.repository",
        entityManagerFactoryRef = "secondaryEntityManager",
        transactionManagerRef = "secondaryTransactionManager"
)
public class SecondaryDataSourceConfig {
    @Bean
    @ConfigurationProperties(prefix = "secondary.datasource")
    public DataSource secondaryDataSource() {
        return DataSourceBuilder.create().build();
    }
    @Bean
    public LocalContainerEntityManagerFactoryBean secondaryEntityManager(
            EntityManagerFactoryBuilder builder) {
        return builder
                .dataSource(secondaryDataSource())
                .packages("com.example.secondary.entity")
                .persistenceUnit("secondary")
                .build();
    }
    @Bean
    public PlatformTransactionManager secondaryTransactionManager(
            EntityManagerFactoryBuilder builder) {
        return new JpaTransactionManager(secondaryEntityManager(builder).getObject());
```

```
    }
}
```

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

Solution:

//pom.xml

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</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>
</dependencies>
```

//Employee.java

```
package com.example.employeemanagement.entity;
import lombok.*;
import org.hibernate.annotations.DynamicInsert;
import org.hibernate.annotations.DynamicUpdate;
import javax.persistence.*;
@Entity
@Data
```

```
@NoArgsConstructor
@AllArgsConstructor
@DynamicInsert
@DynamicUpdate
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;
    @ManyToOne
    private Department department;
}
```

//application.properties

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=password
spring.datasource.driver-class-name=org.h2.Driver
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
spring.jpa.properties.hibernate.jdbc.batch_size=20
spring.jpa.properties.hibernate.order_inserts=true
spring.jpa.properties.hibernate.order_updates=true
spring.jpa.hibernate.ddl-auto=update
```

//EmployeeBatchService.java

```
package com.example.employeemanagement.service;
import com.example.employeemanagement.entity.Employee;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import javax.persistence.EntityManager;
```

```
import javax.persistence.PersistenceContext;
import javax.transaction.Transactional;
import java.util.List;
@Service
@RequiredArgsConstructor
public class EmployeeBatchService {
    @PersistenceContext
    private EntityManager entityManager;
    @Transactional
    public void saveBatch(List<Employee> employees) {
        int batchSize = 20;
        for (int i = 0; i < employees.size(); i++) {
            entityManager.persist(employees.get(i));
            if (i % batchSize == 0 && i > 0) {
                entityManager.flush();
                entityManager.clear();
            }
        }
        entityManager.flush();
        entityManager.clear();
    }
}
```

//EmployeeBatchController.java

```
package com.example.employeemanagement.controller;
import com.example.employeemanagement.entity.Employee;
import com.example.employeemanagement.service.EmployeeBatchService;
import lombok.RequiredArgsConstructor;
import org.springframework.web.bind.annotation.*;
import java.util.List;
@RestController
```

```
@RequestMapping("/employees/batch")
@RequiredArgsConstructor
public class EmployeeBatchController {
    private final EmployeeBatchService batchService;
    @PostMapping
    public String saveEmployees(@RequestBody List<Employee> employees) {
        batchService.saveBatch(employees);
        return "Batch insert successful!";
    }
}
```