Exercise 1: Employee Management System - Overview and Setup

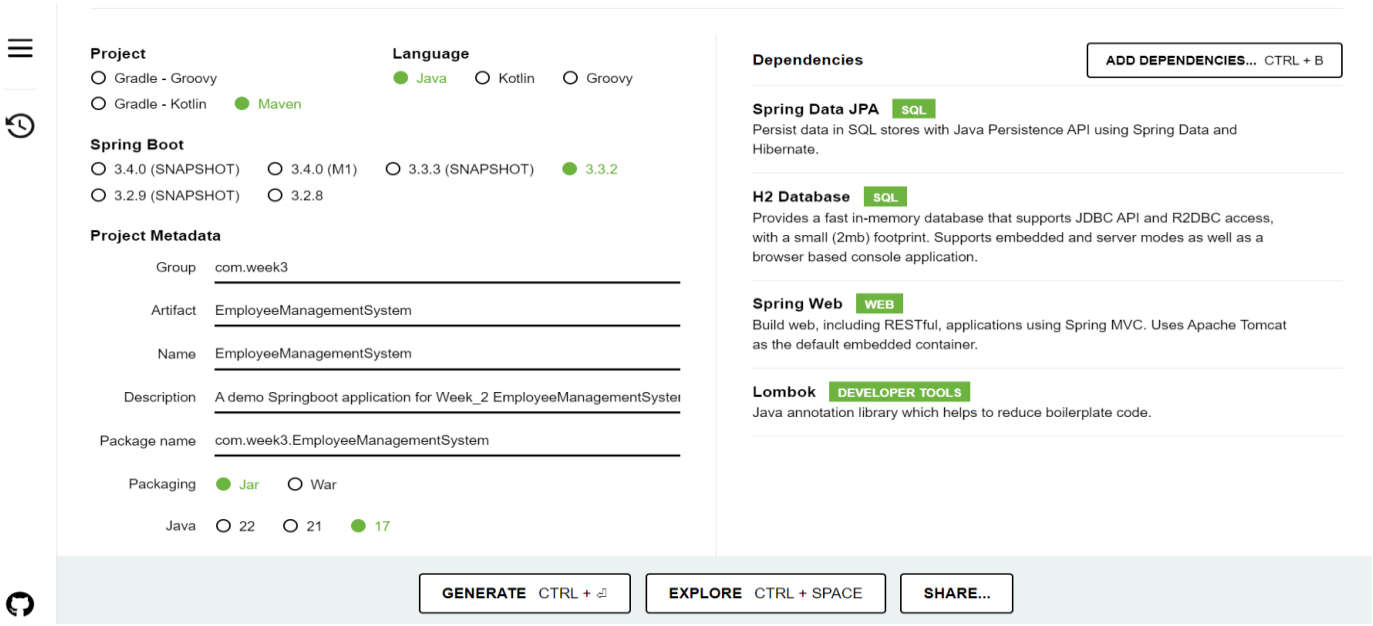
Business Scenario:

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**.

[All the dependencies and project information of EmployeeManagementSystem are added in the pom.xml file.]



2. **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

[The H2 database connection is configured in the application.properties file inside src/main/resources of the project EmployeeManagementSystem.]

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**.
    - Define **Department** entity with fields: **id, name**.

[These two entities are defined in the package com.week3.EmployeeManagementSystem.model of src/main/java with comments for better understanding of the instructions]

2. **Mapping Entities to Database Tables:**
    - Use annotations like **@Entity, @Table, @Id, @GeneratedValue**, etc.
    - Define one-to-many relationship between **Department** and **Employee**.

[All the annotations and one-to-many relationship between Department and Employee is defined in the package com.week3.EmployeeManagementSystem.model of src/main/java with comments for better understanding of the instructions]

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.

[Some benefits of using Spring Data repositories are –

- **Simplicity:** Provides a simple and consistent way to access data.
- **Less Boilerplate Code:** Automatically implements common CRUD operations.
- **Integration with Spring:** Easily integrates with other Spring components, providing overall data access solution.]

2. **Creating Repositories:**
    - Create **EmployeeRepository** and **DepartmentRepository** interfaces extending **JpaRepository**.
    - Define derived query methods in these repositories.

[EmployeeRepository and DepartmentRepository are defined in the package com.week3.EmployeeManagementSystem.repository of src/main/java of the EmployeeManagementSystem project with comments for better understanding of the instructions.]

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**.

[EmployeeController and DepartmentController are defined in the package com.week3.EmployeeManagementSystem.controller of the src/main/java in the project EmployeeManagementSystem. These controllers will utilize the EmployeeRepository and DepartmentRepository to perform the required operations. Comments are also used for better understanding of the instructions.]

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.

[@Query annotation with queries are defined in the EmployeeRepository and DepartmentRepository of package com.week3.EmployeeManagementSystem.repository of src/main/java of the EmployeeManagementSystem project with comments for better understanding of the functionalities.]

2. **Named Queries:**
    - Define and execute named queries with **@NamedQuery** and **@NamedQueries**.

[@NamedQuery and @NamedQueries are added in the Employee class of package com.week3.EmployeeManagementSystem.model in the src/main/java folder of the project EmployeeManagementSystem. Comments are added for better understanding.]

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**.

[Page and Pageable are handled in the EmployeeRepository class of package com.week3.EmployeeManagementSystem.repository in the folder src/main/java of the project EmployeeManagementSystem with comments for better understanding.]

2. **Sorting:**
    - Add sorting functionality to your queries.
    - Combine pagination and sorting in your search endpoint.

[The EmployeeController class of com.week3.EmployeeManagementSystem.controller package in the src/main/java is also updated to include the pagination and sorting functionalities in the project EmployeeManagementSystem. Comments are also updated for better understanding.]

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.

[According to the first step @EnableJpaAuditing annotation is attached with the EmployeeManagementSystemApplication class of com.week3.EmployeeManagementSystem package. Comments are provided for better understanding of instructions.]

    - Use annotations like **@CreatedBy, @LastModifiedBy, @CreatedDate**, and **@LastModifiedDate**.

[A auditing configuration class is designed to set up the auditor aware bean in the folder src/main/java of the package com.week3.EmployeeManagementSystem.config.Also the

Employee and Department entities are annotated with auditing annotations. Comments are provided for better understanding of the instructions.]

Exercise 8: Employee Management System - Creating Projections

Business Scenario:

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

[Interface-based projections allow to define an interface with getter methods corresponding to the fields you want to fetch.]

Instructions:

1. **Projections:**
    - Define interface-based and class-based projections.

[EmployeeProjection and DepartmentProjection are two interface-based projection defined in the package com.week3.EmployeeManagementSystem.projection of the folder src/main/java in the project EmployeeManagementSystem. The projections are also used in the repositories. Again EmployeeDTO and DepartmentDTO are two class based projections defined in the package com.week3.EmployeeManagementSystem.dto of the folder src/main/java in the project EmployeeManagementSystem. Class based projections are also used in repositories. The controller in the src/main/java are also updated to include the new projection methods. Detailed comments are specified for better understanding of the instructions.]

    - Use **@Value** and constructor expressions to control the fetched data.

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.

[Spring Boot's auto-configuration automatically sets up a single DataSource based on properties in application.properties. Primary and Secondary Datasource Configurations are added to leverage data sources in the application.properties in the src/main/resources folder. Comments are provided for better understanding.]

- Manage multiple data sources within your application.

[Multiple datasources are managed using PrimaryDataSourceConfig and SecondaryDataSourceConfig which are defined in the package com.week3.EmployeeManagementSystem.config of the folder src/main/java in the project EmployeeManagementSystem. Comments are provided for better understanding.]

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.

[Hibernate specific annotations are added to the Employee as well as Department entity of the package com.week3.EmployeeManagementSystem.model. Comments are provided for better understanding.]

2. **Configuring Hibernate Dialect and Properties:**
    - Configure Hibernate dialect and properties for optimal performance.

[Hibernate dialect and additional properties are defined in the application.properties file in the src/main/resources folder. Comments are provided for better understanding.]

3. **Batch Processing:**
    - Implement batch processing with Hibernate for bulk operations.

[Hibernate Batch Processing is defined in the application.properties file and the batch processing is implemented in a EmployeeService method in the package com.week3.EmployeeManagementSystem.service.]
