Spring Data Workshop
Lab 3
Exercises

1 Lab setup

Make sure you have the following items installed

- MySQL 8.x
- Latest LTS JDK version (at this point: JDK 21)
- Spring Tool Suite (STS) or IntelliJ IDEA
- Latest version of Maven (at this point: Maven 3.9.9)
- A free account at Postman and installed the Postman app
- A suitable text editor (Notepad ++)
- A utility to extract zip files (7-zip)

In each of the main lab folders, there are two subfolders: `changes` and `final`. The `changes` subfolder just holds the source code files for the lab, while the `final` subfolder holds the complete Eclipse project starting from its project root folder. We will use the code from the `changes` subfolder to build up our applications from scratch and you can always fall back on the complete Eclipse project if you encounter any errors while building up the application.

2 Designing REST API

In this exercise, we will repeat the implementation of a REST API service for the lab where we use a MySQL database table as the underlying persistence storage. We will create the underlying classes that model the business domain and then provide implementation for the REST API endpoints that the service will expose in order to manipulate a collection of objects from these classes.

The REST API that we create will maintain a list of Employees with the following schema:

Employee
`id: Integer`
`name: String`
`age: Integer`
`gender : Character`
`hire_date : LocalDate`
`salary : Float`

This schema corresponds to the MySQL database table `employees` that we created in Lab 1.

In MySQL command line client, check whether this table still exists:

```
SHOW TABLES;
```

If not, you can always restore the original table content which is already in the `sqltables` folder of your `labcode` folder.

Open a command prompt in this folder (or navigate to it), then type:

```
mysql -u root -p workshopdb < emptable.sql
```

You will again be prompted for the admin (root) password.

Back in the MySQL command line client, verify that the table has been created with:

```
SHOW TABLES;
```

Check the contents of the table with:

```
SELECT * FROM employees;
```

The REST API service that you develop will expose the following API endpoints for consumption by a service:

Method	**Endpoint**	**Description**
GET	`/api/employees`	Get the list of employees
GET	`/api/employees/{id}`	Retrieve the employee with the specified id
POST	`/api/employees`	Add a new employee to the list of employees
PUT	`/api/employees/{id}`	Make a modification to an employee with the specified id
DELETE	`/api/employees/{id}`	Delete an employee with the specified id

3 Creating the application package structure

Start up STS. Switch to the Java EE perspective.

Go to File -> New -> Other -> Spring Boot -> Spring Starter Project. Complete it with the following details:

Name: `JPARestExercise`
Group: `com.exercise.jpa`
Artifact: `JPARestExercise`
Version: `0.0.1-SNAPSHOT`
Description: `REST API with Spring Data JPA using MySQL app for developers`
Package: `com.exercise.jpa`

Add the following dependencies:

```
Web -> Spring Web
SQL -> Spring Data JPA
SQL -> MySQL Driver
Developer Tools -> Project Lombok
Developer Tools -> Spring Boot DevTools
```

The Spring Boot DevTool dependency will automatically restart the Spring Boot application whenever it detects changes in your compiled application code – this will facilitate the development of REST API service as you will not need to start / restart the service every time you change your source code and save.

In `src/main/java`, create the following packages which are going to have the following purposes:

Package name	**Purpose**
`com.exercise.jpa.model`	Holds all the classes for the business domain model. This will be stored in the database table via Spring Data JPA, and will thus be @Entity classes.
`com.exercise.jpa.dto`	Holds all the DTO classes that encapsulate the data to be exchanged between the client and service. This includes custom error messages.
`com.exercise.jpa.repository`	Holds the user-defined interfaces that extend on the standard Spring Data JPA Repositories (CrudRepository, PagingAndSortingRepository, etc)
`com.exercise.jpa.service`	Holds the classes that extract data from the repository interfaces and perform any required business logic on them
`com.exercise.jpa.controller`	Holds all the @RestController classes that implement the various @XXXMapping methods for the REST API utilizing the various Service classes.
`com.exercise.jpa.exception`	Holds all the user defined exceptions and the primary @ControllerAdvice exception handling class containing the individual @ExceptionHandler methods

The main @SpringBootApplication class (JpaRestExerciseApplication) will reside in the top level package: `com.exercise.jpa` so that Spring will scan all its subpackages to locate @Component / @Service / @Repository classes for DI.

4 Implementing basic GET, POST, PUT and DELETE

Create the following files in the following folders:

In `src/main/resources`, create:

`application.properties`

Configure this REST service to start on port 8081 so that you can test it in conjunction with `JPARestApp` in the corresponding lab for this exercise (that runs on port 8080)

In `com.exercise.jpa.model`, place:

`Employee`

In `com.exercise.jpa.controller`, create:

`EmployeeController`

In `com.exercise.jpa.service`, create:

`EmployeeService`

In `com.exercise.jpa.repository`, create:

`EmployeeRepository`

In `com.exercise.jpa.dto`, create:

```
EmployeeDTO
CustomErrorMessage
```

In `com.exercise.jpa.exception`, create:

```
EmployeeControllerExceptionHandler
IncorrectJSONFormatException
IncorrectURLFormatException
```

4.1 Test GET implementation

Make a GET request to:

`localhost:8081/api/employees`

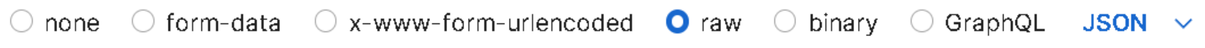
and verify that the complete list of employees in `employees` table is returned correctly in JSON.

4.2 Test POST implementation

Test creating a new Employee record with a POST request to:

```
localhost:8081/api/employees
```

with the following raw JSON content in the body:



○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

```
{
    "name": "Peter Parker",
    "age": 22,
    "gender": "M",
    "hire_date": "2025-12-22",
    "salary": 6600.80
}
```

Check that a status 201 Created is returned with the following URL in the Location header:

http://localhost:8081/api/developers/xxx
where xxx is the id of the newly created record in the Employees table.

You can also verify by making another GET request to retrieve all the employees:

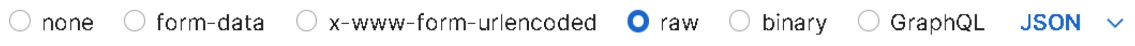
```
localhost:8081/api/employees
```

4.3 Test PUT implementation

Make a PUT request to:

```
localhost:8081/api/employees/2
```

with the following raw JSON content in the body:



○ none ○ form-data ○ x-www-form-urlencoded ● raw ○ binary ○ GraphQL JSON

```
{
    "name": "Jane Segal",
    "age": 22,
    "gender": "F",
    "hire_date": "2018-08-15",
    "salary": 6000.66
}
```

Verify that this returns with status 200 OK.

To verify that the developer with id 2 has had the details changed accordingly, make a GET request to:

```
localhost:8081/api/employees
```

4.4 Test DELETE implementation

Make a DELETE request to:

```
localhost:8080/api/employees/2
```

Verify that this returns with status 200 OK.

To verify that the employee with id 2 has been deleted, make a GET request to:

```
localhost:8080/api/employees
```

Alternatively, verify from the MySQL command line client.

You can repeat this operation to delete a few other random records and verify accordingly.

We can check for similar errors as in the case of the PUT operation.

Make a DELETE request to:

```
localhost:8080/api/employees/999
```

Verify that the error message returned points out that no developer with such id exists.

Make a DELETE request to:

```
localhost:8080/api/employees/3sw
```

Verify that the error message returned points out that the developer id needs to be specified correctly as a number.
NOTE: If you have deleted too many rows from the table, you can always restore the original table content which is already in the `sqltables` folder of your `labcode` folder.

Open a command prompt in this folder (or navigate to it), then type:

```
mysql -u root -p workshopdb < emptable.sql
```

You will again be prompted for the admin (root) password.
Back in the MySQL command line client, verify that the table has been created with:

```
SHOW TABLES;
```

Check the contents of the table with:

```
SELECT * FROM employees;
```