

Lab 3 Multi-layer web applications with Spring Boot

Updated: 2020-11-02.

Introduction to the lab

Learning outcomes

- Create a web application project with Spring Boot and Spring Initializr, combining the appropriate “starter” dependencies.
- Create and persist entities into a relational database using Spring Data.
- Expose a RESTful API, using Spring Annotations.

References and suggested readings

- [“7 Things to Know for Getting Started With Spring Boot”](#)

Submission

Students should submit in Moodle evidence of their own work. A (single) zip file with one folder per section (ies3.1, ies3.2,...) should be prepared; in the root, there should be a file "[readme.md](#)" with the author ID.

You should use the **Readme.md as a notepad**; use this file both to take notes that will help you study later, and to provide a log of the work done.

The items that required an explicit response (in the Readme) are marked with 📄.

3.1 Layered applications in Spring

The Spring Boot site offers an extensive list of [getting started guides](#). You will use the “[Building REST services with Spring](#)” for this exercise.

Note: to build Spring Boot applications, consider using a IntelliJ IDEA (Ultimate edition) or VS Code [with Spring Tools](#), or [Eclipse with Spring Tools](#).

- a) Be sure to get a general understanding about this [tutorial example](#). This time, we will use the available code and study it (instead of developing).

Download and run the [code project](#). Look for the “**rest**” **sub-project** inside the main, root project and run the `payroll.PayrollApplication`.

The application does not have web pages; it exposes (REST) endpoints for client programs to connect (as defined in `EmployeeController`).

- b) Access the REST endpoints using [Postman utility](#) (or [curl](#), from command line).

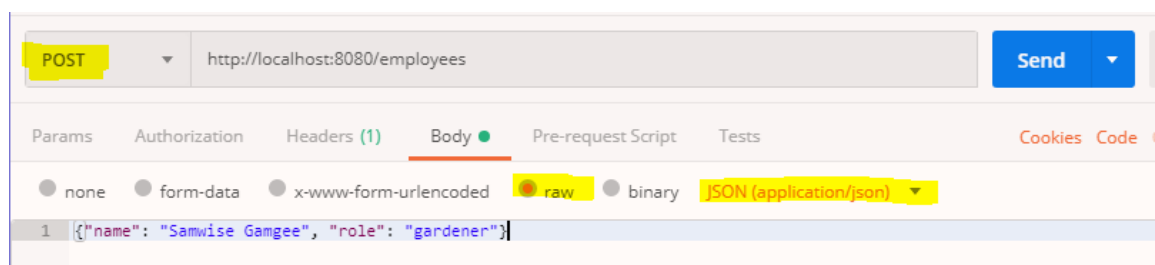


Figure 1- POST method, with JSON payload in the body of the request, to insert a new Employee.

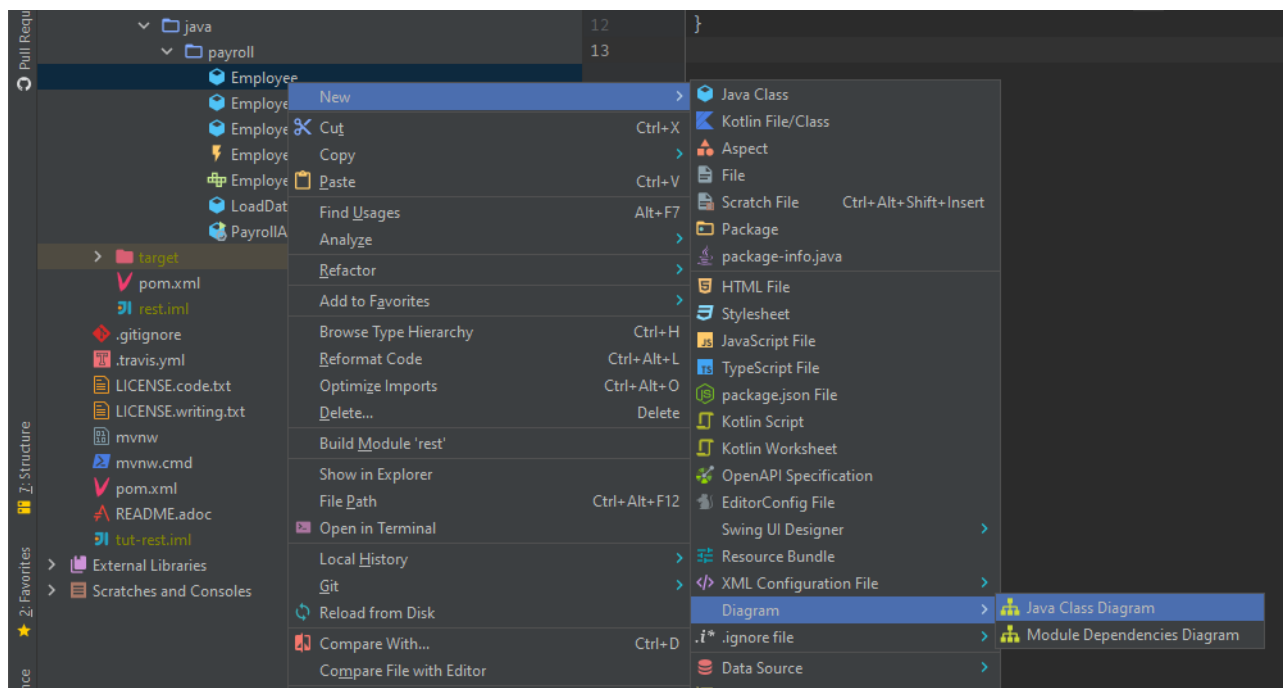
Be sure to list all, filter one, insert and update Employees, i.e., try the several web methods. Take note of the URL used and report in the README.

Note the standard way in which the “resources” are exposed in the REST API. (For a more detailed discussion on the designing REST API, you may check this e-book “[Web API Design: The Missing Link](#)”.)

- c) What happens to your data when the application is stopped and restarted? How could you change that behavior?

You don't need to implement a solution; just explain it (with details).

- What would be the proper HTTP Status code to get when searching an API for non-existent <http://localhost:8080/employees/987987> ?
- Be sure to provide evidence (e.g.: screenshots, JSON results view) that you have successfully used the API to insert new entries.
- Create a layered architecture view (UML diagram), displaying the key abstractions in the solution, in particular: entities, repositories and REST controllers.
- Describe the role of the elements modeled in the previous point.



Create UML diagrams from code in IntelliJ.

3.2 Accessing JPA Data with REST interface

The [Java Persistence API](#) (JPA) defines a standard interface to manage data over relational databases. Most of the work is provided by the adaption layer that implements the JPA specification and converts objects into relational data and vice-versa. A common implementation is the Hibernate framework. Spring Data uses and enhances the JPA.

When you use JPA, your Java code is independent from the specific database implementation.

- d) In this exercise you will need an instance of MySQL server (stick with **version 5.7**). If you don't have one already, consider using a [Docker container](#), e.g.:

```
$ docker run --name mysql5 -e MYSQL_ROOT_PASSWORD=password -e MYSQL_DATABASE=demo
-e MYSQL_USER=demo -e MYSQL_PASSWORD=password -p 3306:3306 -d mysql/mysql-
server:5.7
```

- e) [This guide](#) will walk you through the details of preparing the entities, the database and expose RESTful endpoint for remote access, for a basic Employee management application.

This application is functionally comparable to the previous one, but, this time, be sure to start the project from the scratch¹ (instead of downloading the solution).

In the Spring Initializr, add the “starters” dependencies for Spring **Web**, Spring Data **JPA**, **MySQL driver**, **DevTools** and **Validation**.

Take the tutorial and be sure to complete the following tasks:

- f) Create the Employee **entity**.
- g) Create a “**Repository**” of Employees.
- h) Create a “(REST) **controller**” to expose the Employee entity.
- i) Be sure to define the database [connection properties](#) in the Application.properties resource file.

E.g:

```
# MySQL
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/demo
spring.datasource.username=demo
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect

# Strategy to auto update the schemas (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update
```




Test your application using *curl* or the Postman tool. Be sure to insert and list content.

- j) Enhance you REST API with a method to search an employee by email (search by email).

Be sure to extend the Repository with a [corresponding query method](#)².

Add a filter option to the Employees listing method by [using an URL parameter](#), e.g.:

```
http://localhost:8080/api/v1/employees?email=big@here.com
```

-  Explain the annotations @Table, @Column, @Id found in the Employee entity.
-  Explain the use of the annotation @Autowired (in the Rest Controller class).
-  Submit the complete project (after a “clean”).

3.3 From data to presentation (Thymeleaf)

In [this guide](#), you will use the same strategy for data management, with Repository and Controller components, to develop an Issues Reporting solution.

¹ You are expected to create the project, classes and methods declaration “by hand”. Feel free to copy and paste the methods’ body, as they tend to be verbose.

² There is a lot of information in the hyperlinked page. Be selective and use just what you need in this context.

This example shows how to connect with the web pages, using a templating system (Thymeleaf).

Note: the authors use the Gradle build system. While you may use it to, we suggest keeping Maven as the build tool.

Using the guide for support, be sure to:



- k) Create a project using the dependencies: Spring Web, Spring Data JPA, H2 database, Thymeleaf, DevTools.
- l) Create the controller for the web pages flow.
- m) Create the Issue entity.
- n) Create the Thymeleaf templates and set the required data-binding.
- o) Configure the repositories and save data to the H2 database.

Note: the previous tasks are explained along the tutorial guide. You should **stop at task #8** in the guide. At this point, you should be able to see the web pages, insert data into the database, and invoke the REST API.

- p) Setup a Docker container to host a persistent database (e.g.: Postgres, MySQL). You may already have it from previous exercises.

Adapt the project to work with this database, instead of the H2 database. Be sure to review the dependencies and set the application.properties file.

- q) Deploy the “Issues Reporting” application to a Docker container, too; i.e., [dockerize your Spring Boot application](#).

-  Project with the Issue Tracking application (after a “clean”).
-  Docker compose configuration file.