3 Lab: Getting started with Spring Boot applications

v2025-10-09

Learning objectives

- Understand the typical “anatomy” of a Spring Boot project (based on Web MVC with relational data sources)
- Implement simple backend application in Spring Boot to expose data through a REST API.

Key Points

- The Spring framework is widely used for Java-based backend development. Spring Boot facilitates the use of Spring by assuming reasonable configuration, based on settings and class path dependencies.
- A Spring Boot application includes several life cycle managed “components” that are automatically “discovered”. You should not instantiate the framework managed components; they will be provided for you (with concurrency management, security, etc.).

3.1 What is Spring Boot?

Research online and try to elaborate a personal response (in your own words) to the above question. Try to include/relate the following topics in your answer:

- what is an "enterprise application"
- what are the leading frameworks for “Java enterprise applications”?
- how does Spring Boot relate to the Spring framework and Jakarta?
- name a few examples of automatic configuration (defaults) made by Spring Boot.
- how does Sprint Boot relate to your previous learning path/course subjects?

3.2 Getting started tutorial: REST web services

a) Complete the instructional Guide on RESTful web services development.

The guide offers you an option to start from scratch or take some shortcuts. Be sure to complete each step to get familiar with the basics too.

Note: you can run the application from the IDE or command line, e.g.:

```
$ mvn spring-boot:run
```

b) Closely observe the console output while the application is starting.

Explain the references to “Tomcat” in the log (i.e., what is it and why is used in this context).

Note: errors, i.e., stack traces, are dumped into the console log.

3.3 User management example

For this activity we will use the "Spring tutorial" example, available from GitHub, which is a Spring Boot application. As in the previous example (which you may recognized diluted in the code), this project also implements a RESTful API.

Note: the project contains more elements and configuration that we need for now; just focus on the elements required for the proposed tasks.

a) Get a local copy of the project and open it in your IDE.

b) Explore the project structure.

   What is the rationale for packages "data", "services", "boundary"?

   Note: you may wish to include the concepts of "layers" and "separation of concerns".

c) What is the relationship between the classes "data.User", "data.UserRepository", "services.UserServiceImpl" and "boundary.UserRestController"?

   Which specific class-level annotations can one notice?

   Supplement your answer with a diagram (especially, a class diagram).

d) Based on what you could learn from the code, try to interact with the REST endpoints available. (Note: you need to run the Spring Boot application.)

   For that, you may use the browser, the "curl" utility or any other REST client, such as Postman.

   Install the Postman app (desktop) and use it to create and list users in the application[1].

e) Inspect the project dependencies.

   What are the "spring-boot-starter-*" dependencies (a.k.a. "starters")?

f) Explore the dependencies included in the POM and the (1st level) transitive dependencies included in spring-boot-starter-data and complete the table:

	Module/component
Spring framework API to interact with (relational) databases.	
Implementation of the reference Java Persistence API (JPA).	
Specific database drivers included in this project.	

g) The project is running as expected but you haven't created any database… Have you already found where the data is being saved?

3.4 Meals booking

Consider the activity from Lab 1 (#1.3) for the meals booking application.

Assume that you now want to create the backend for this application. Let us assume that another team will implement a mobile application to book and update meals tickets and you are expected to create a simple Spring Boot application that exposes an API with endpoint to request a new booking (returning a code), check existing bookings, check a specific booking by ticket code, update the

[1] This is a good example of an activity in which you should take some screenshots and include in the "lab study notes" (readme.md) to provide evidence you got the expected results.

booking (cancel or use in the check-in). Keep as simple as possible (e.g.: for now, you don't need to represent users, or food units, etc., just the booking/ticket).

a) Based on what you observed about the "anatomy" of typical Spring Boot applications, explain which classes and Spring components you plan to use (support with an illustration).

   Where will you use the (production) code you have already from Lab 1, with the meals booking logic?

b) Start a new Spring Boot project. You may use the Spring Initializr (or similar support from your IDE). Be sure to include proper *starters* (Figure 1).

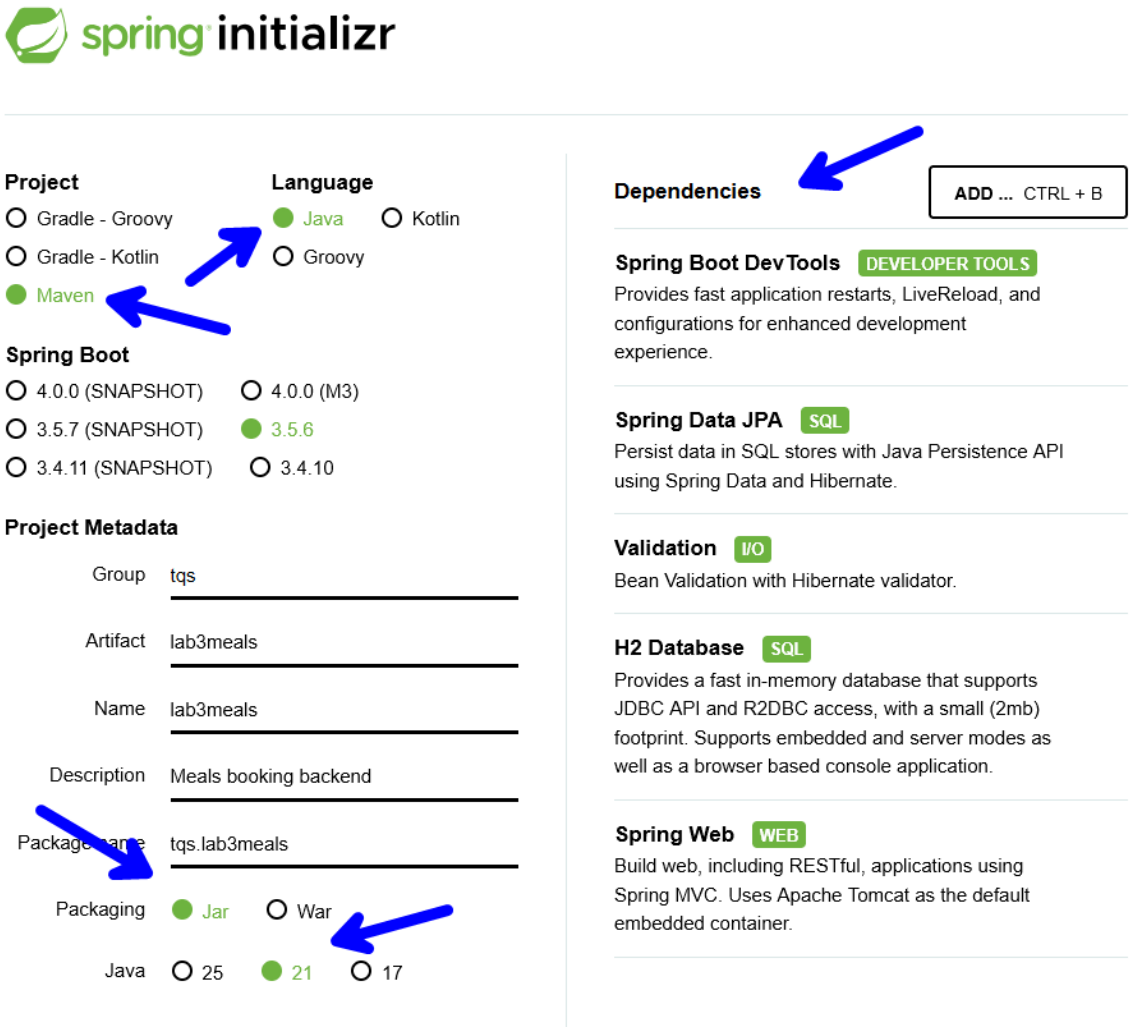

Figure 1: Project settings to include in the build tool.

c) One you have the project, use a similar organization as for the previous example and prepare:

- 🗁 data: create your entity and repository.
- 🗁 service: implement business logic. Components thar are to be found (and managed) by Spring can be marked as @Service.
- 🗁 boundary: create a controller to expose the REST endpoints.

Placement of @SpringBootApplication class
For the automatic discovery of components, the class marked as the entry point, i.e., annotated with @SpringBootApplication must be in the base package. All other components can be placed at the same level or, better, in a sub-package.

You should be able to learn from the previous example and adapt to your own problem. Implement the API and use Postman to confirm that the expected endpoints work as expected.

Mind the semantics of HTTP methods in a RESTful API!

d) Up to now, your database has been reset every time you run and stop the project, which is not the intended behavior of a backend. Let us address this by specifying a persistent data source, using PostgreSQL.

While it is possible to install and configure PostgreSQL in your system, it is much more convenient to use a "managed" approach to "fire up" and "dismiss" services and infrastructure as needed. For that, we will use Docker containers.

- Install Docker in your system (if not yet available). Confirm Docker is running:

```
$ docker ps
```

- Start a container with PostgreSQL engine, e.g.:

```
docker run --name postgresdb -e POSTGRES_USER=admin -e POSTGRES_PASSWORD=secret   -e
POSTGRES_DB=meals_db -p 5432:5432 -d postgres:latest
```

Take note of the configuration elements in color (which can be changed); you will use them to configure the database connection.

You should be able to get a successful connection to the port mapping you have specified, e.g.:

```
$ nc -zv 127.0.0.1 5432
```

- Remove (or comment out) the dependency for the H2 database in the POM and, instead, include the dependency for PostgreSQL. Note: use Spring Initializr to peek the right one.
- Add the database connection info to the configuration file (using the parameters defined in *docker run*) as well as the intended generation guideline to Hibernate (e.g.: *spring.jpa.hibernate.ddl-auto=update*)
- Run your backend. Confirm the services are available using PostgreSQL. Confirm the data has effectively persisted across applications restarts.

Supplementary materials

Usefull hibernate properties

spring.jpa.hibernate.ddl-auto=*none* → *update, drop-create,…*

spring.jpa.show-sql=*false* → *true*