

Module 4: Backend Server

Lecture 2: Spring Boot Project Setup and Configuration

URL + port basics

Maven + dependencies

Project structure tour

Configuration + Tomcat

Orientation and prerequisites

Outcomes and plan

By the end of today, you can:

- Generate a Spring Boot project
- Understand a basic Spring Boot project structure
- Run the app using Maven wrapper
- Make and verify a configuration change
- Explain what “embedded server” means and why Tomcat exists

Today’s flow:

- URL and port basics
- Maven and dependencies basics
- Generate a project
- Tour the structure
- Run the app, then change config
- Plug in the Student mini-project

Uniform Resource Locator (URL) and port

A URL is how a client reaches your app:

- Host: which machine the app is on
- Port: which running program on that machine
- Path: which feature or resource inside the app

Ports matter because:

- One machine can run many programs
- Each program listens on a different port
- If a port is already in use, your server cannot start on that port

`https://api.schoolapp.com:8443/students`

Question: Break this into host, port, and path: `https://api.schoolapp.com:8443/students` Answer in chat.

Running a server app

Normal Java program:

- Runs a flow and exits

Server style Java app:

- Starts and keeps running
- Waits for requests
- You stop it manually

Why this matters today:

- When the Spring Boot app starts successfully, your terminal does not exit immediately
- It stays running because the server is alive and listening on a port

Maven and dependencies

Why Maven exists:

- Real projects need external libraries for common work
- Without Maven, each developer ends up downloading files manually and managing versions.
 - Example: database drivers, JSON conversion, testing support

What Maven gives you:

- A consistent way to build and run a Java project
- A consistent way to download dependencies automatically

Key file you will see today:

- pom.xml is the project's build file
 - It lists dependencies and important project metadata

Question: Why is it risky for every developer to manually download libraries and add them to a project? Answer in chat.

Creating a Spring Boot project

Generate the project using start.spring.io

Why we use start.spring.io:

- It generates a working Spring Boot project with the correct structure
- It prevents setup mistakes and saves time

What we pick today:

- Build tool: Maven
- Language: Java
- Dependencies: Spring Web, only

Why we keep dependencies minimal today:

- Fewer moving parts means less confusion
- Today is about setup, structure, and configuration, not building features

Generator choices you must understand

Packaging: JAR (Java ARchive)

- JAR is a packaged Java application you run like a normal program
 - Purpose: makes it easy to distribute Java applications or libraries as one file instead of many

Group and artifact

- Group is the namespace of your organization or project family
 - It becomes the base package name in Java
 - Example: com.guvi
- Artifact is the application name
 - It becomes the project name and appears inside the build file
 - Example: student-app

Question: If your group is com.guvi and your artifact is student-app, what would a sensible Java package name look like for your code? Answer in chat.

What the download contains and what it proves

After download, you should recognize these:

- src/main/java for application code
- src/main/resources for configuration files
- src/test/java for test code
- pom.xml for dependencies and build configuration
- mvnw and mvnw.cmd so everyone can run the same Maven commands

What we do next:

- Open the project in IntelliJ
- Confirm project metadata inside pom.xml
- Run the app using Maven wrapper
- Make one config change and re-run to verify the effect

Question: Open pom.xml and find these two values: what is the artifactId, and what Java version is the project set to use?
Answer in chat.

Project structure tour

Spring Boot project layout

- Application code in `src/main/java`
- Configuration and non-code files in `src/main/resources`
- Tests in `src/test/java`

Why this structure matters:

- Build tools and IDEs assume this layout by default
- It keeps production code, config, and tests clearly separated

Question: You want to change a runtime setting without touching Java code. Which folder do you check first? Answer in chat.

src/main/java and the main class

What lives in src/main/java:

- Your application entry point class
- Feature code in packages you create

How to spot the entry point quickly:

- It is the class with public static void main(String[] args)
- In Spring Boot, it is usually also the class that contains the application startup call

Question: Find the entry point class name in your project and paste only the class name. Answer in chat.

src/main/resources and configuration

What lives in src/main/resources:

- application.properties or application.yml
- Any non-code files your app needs at runtime

Why config lives here:

- You can change settings without editing Java code
- You avoid rebuilding the entire project for simple environment changes

Question: Give one example of a value that should live in configuration, not hardcoded in Java. Answer in chat.

pom.xml: The project contract

What pom.xml controls:

- Which libraries the project uses
- Which Java version the project targets
- Build and run behavior through Maven

What beginners must be able to do:

- Identify the project name used by Maven
- Identify which dependencies were added

Question: Find the Spring Web dependency in pom.xml. What is its artifactId? Answer in chat.

Changing the project structure

Yes, you can change the default layout:

- Maven can change source and resources folders in build configuration
- IntelliJ can mark folders as Sources or Resources

Why teams usually avoid changing it:

- Everyone expects the default structure when reading a repo
- Builds, IDE settings, and CI pipelines become harder to standardize
- New team members spend time relearning your custom layout instead of shipping code

Running the Spring Boot app

Run the app using Maven wrapper

Why we use Maven wrapper in this course:

- Everyone runs the same commands regardless of local setup
- Fewer “works on my machine” issues in a batch

Run commands:

- Mac and Linux: ./mvnw spring-boot:run
- Windows: mvnw.cmd spring-boot:run

What success looks like:

- Logs mention the port the server is listening on
- Logs indicate the application started
- The terminal stays running because the server is alive and waiting

Question: Run the command and paste only the port number you see in the logs. Answer in chat.

Stop, restart, and fix the most common failure

How to stop the running app:

- Terminal: Ctrl+C
- IntelliJ: Stop button

Most common startup failure:

- Port already in use

Two simple fixes:

- Stop the other program using that port and rerun
- Change the port in application.properties and rerun

Question: If you hit “port already in use”, what is the fastest fix you would try first, and why? Answer in chat.

Configuration basics

Configuration and why it exists

Configuration means values that should change without changing Java code.

Key idea:

- Code contains business rules and behavior
- Configuration contains environment-specific values you may change often
 - Example: server port, database URL, log level

Spring Boot reads config from application.properties or application.yml in src/main/resources.

Question: Give one example of a value that should be configuration, not Java code, and explain why. Answer in chat.

application.properties

- We will use application.properties for our Spring Boot application
- What is it? Simple key-value pairs, allowing us to simply change our project configuration

Example setting:

```
server.port=9090  
logging.level.root=INFO
```

What you should observe after restart:

- Startup logs show the new port
- The server now listens on that port

Question: If the port becomes 9090, what'll be the URL to the server? Answer in chat.

Embedded server and Tomcat

Embedded server

In Spring Boot, the server runs inside your Java application.

- When you run the app, it starts a server automatically
- You do not install a separate server to run it locally
- The Java process stays alive because the server is listening

Question: What is one advantage of the server being embedded inside the app? Answer in chat.

What is Tomcat & why does it exist?

Tomcat is the server component that:

- Listens on a port
- Receives HTTP requests from clients
- Hands the request to your application so it can respond

What this helps you understand:

- Your app is a long-running program that can respond to client requests

Question: In your own words, what is Tomcat doing while your Spring Boot app is running? Answer in chat.

Plug in the Student mini-project

Bringing the Student mini-project into Spring Boot

We already built student features in plain Java.

Now we reuse the same logic inside a Spring Boot project:

- Business logic stays the same
- Project structure and startup style changes
- Configuration moves to application.properties

Question: What stays the same when we move the student project into Boot: the business logic or the startup and wiring? Why?
Answer in chat.

Activity

Activity goal:

- Create a Spring Boot project
- Explain the role of major folders and key files
- Run the app using Maven wrapper
- Change the port in configuration and verify it
- Plug in the student mini-project and run the student flow at startup

Submission expectation:

- You can explain what each major folder and file is responsible for
- You can explain what changed when moving from plain Java to Spring Boot