

Developing Your First Spring Boot Application

This section describes how to develop a small “Hello World!” web application that highlights some of Spring Boot’s key features. You can choose between Maven or Gradle as the build system.

TIP

The [spring.io](#) website contains many “Getting Started” [guides](#) that use Spring Boot. If you need to solve a specific problem,

Spring Boot / Tutorials / Developing Your First Spring Boot Application

You can shortcut the steps below by going to [start.spring.io](#) and choosing the “Web” starter from the dependencies searcher. Doing so generates a new project structure so that you can [start coding right away](#). Check the [start.spring.io user guide](#) for more details.

Prerequisites

Before we begin, open a terminal and run the following commands to ensure that you have a valid version of Java installed:

```
$ java -version
openjdk version "17.0.4.1" 2022-08-12 LTS
OpenJDK Runtime Environment (build 17.0.4.1+1-LTS)
OpenJDK 64-Bit Server VM (build 17.0.4.1+1-LTS, mixed mode, sharing)
```

NOTE

This sample needs to be created in its own directory. Subsequent instructions assume that you have created a suitable directory and that it is your current directory.

Maven

If you want to use Maven, ensure that you have Maven installed:

```
$ mvn -v
Apache Maven 3.8.5 (3599d3414f046de2324203b78ddcf9b5e4388aa0)
Maven home: usr/Users/developer/tools/maven/3.8.5
Java version: 17.0.4.1, vendor: BellSoft, runtime: /Users/developer/sdkman/candidates/java/17.0.
```

Gradle

If you want to use Gradle, ensure that you have Gradle installed:

```
$ gradle --version

-----
Gradle 8.1.1
-----

Build time:   2023-04-21 12:31:26 UTC
Revision:     1cf537a851c635c364a4214885f8b9798051175b

Kotlin:      1.8.10
Groovy:      3.0.15
Ant:         Apache Ant(TM) version 1.10.11 compiled on July 10 2021
JVM:         17.0.7 (BellSoft 17.0.7+7-LTS)
OS:          Linux 6.2.12-200.fc37.aarch64 aarch64
```

Setting Up the Project With Maven

We need to start by creating a Maven `pom.xml` file. The `pom.xml` is the recipe that is used to build your project. Open your favorite text editor and add the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>myproject</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.5.0</version>
  </parent>

  <!-- Additional lines to be added here... -->

</project>
```

The preceding listing should give you a working build

- Spring Boot
- 3.5.0
- ...
- Q Search
- CTRL + k
- Overview
- Documentation
- Community
- System Requirements
- Installing Spring Boot
- Upgrading Spring Boot
- Tutorials
- Developing Your First Spring Boot Application
- Reference
- How-to Guides
- Build Tool Plugins
- Spring Boot CLI
- Rest APIs
- Java APIs
- Kotlin APIs
- Specifications
- Appendix

Developing Your First Spring Boot Application

- Prerequisites
- Maven
- Gradle
- Setting Up the Project With Maven
- Setting Up the Project With Gradle
- Adding Classpath Dependencies
- Maven
- Gradle
- Writing the Code
- The @RestController and @RequestMapping Annotations
- The @SpringBootApplication Annotation
- The "main" Method
- Running the Example
- Maven
- Gradle
- Creating an Executable Jar
- Maven
- Gradle

- Edit this Page
- GitHub Project
- Stack Overflow

The preceding listing should give you a working build.

You can test it by running `mvn package` (for now, you can ignore the “jar will be empty - no content was marked for inclusion!” warning).

NOTE

At this point, you could import the project into an IDE (most modern Java IDEs include built-in support for Maven). For simplicity, we continue to use a plain text editor for this example.

Setting Up the Project With Gradle

We need to start by creating a Gradle `build.gradle` file. The `build.gradle` is the build script that is used to build your project. Open your favorite text editor and add the following:

```
plugins {  
    id 'java'  
    id 'org.springframework.boot' version '3.5.0'  
}  
  
apply plugin: 'io.spring.dependency-management'  
  
group = 'com.example'  
version = '0.0.1-SNAPSHOT'  
sourceCompatibility = '17'  
  
repositories {  
    mavenCentral()  
}  
  
dependencies {  
}
```

The preceding listing should give you a working build. You can test it by running `gradle classes`.

NOTE

At this point, you could import the project into an IDE (most modern Java IDEs include built-in support for Gradle). For simplicity, we continue to use a plain text editor for this example.

Adding Classpath Dependencies

Spring Boot provides a number of starters that let you add jars to your classpath. Starters provide dependencies that you are likely to need when developing a specific type of application.

Maven

Most Spring Boot applications use the `spring-boot-starter-parent` in the `parent` section of the POM. The `spring-boot-starter-parent` is a special starter that provides useful Maven defaults. It also provides a `dependency-management` section so that you can omit `version` tags for “blessed” dependencies.

Since we are developing a web application, we add a `spring-boot-starter-web` dependency. Before that, we can look at what we currently have by running the following command:

```
$ mvn dependency:tree  
  
[INFO] com.example:myproject:jar:0.0.1-SNAPSHOT
```

The `mvn dependency:tree` command prints a tree representation of your project dependencies. You can see that `spring-boot-starter-parent` provides no dependencies by itself. To add the necessary dependencies, edit your `pom.xml` and add the `spring-boot-starter-web` dependency immediately below the `parent` section:

```
<dependencies>  
  <dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-web</artifactId>  
  </dependency>  
</dependencies>
```

If you run `mvn dependency:tree` again, you see that there are now a number of additional dependencies, including the Tomcat web server and Spring Boot itself.

Gradle

Most Spring Boot applications use the `org.springframework.boot` Gradle plugin. This plugin provides useful defaults and Gradle tasks. The `io.spring.dependency-management` Gradle plugin provides `dependency management` so that you can omit `version` tags for “blessed” dependencies.

Since we are developing a web application, we add a `spring-boot-starter-web` dependency. Before that, we can look at what we currently have by running the following command:

```
$ gradle dependencies  
  
> Task :dependencies  
  
-----  
Root project 'myproject'  
-----
```

The `gradle dependencies` command prints a tree representation of your project dependencies. Right now, the project

has no dependencies. To add the necessary dependencies, edit your `build.gradle` and add the `spring-boot-starter-web` dependency in the `dependencies` section:

```
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-web'  
}
```

If you run `gradle dependencies` again, you see that there are now a number of additional dependencies, including the Tomcat web server and Spring Boot itself.

Writing the Code

To finish our application, we need to create a single Java file. By default, Maven and Gradle compile sources from `src/main/java`, so you need to create that directory structure and then add a file named `src/main/java/com/example/MyApplication.java` to contain the following code:

```
package com.example;  
  
@RestController  
@SpringBootApplication  
public class MyApplication {  
  
    @RequestMapping("/")  
    String home() {  
        return "Hello World!";  
    }  
  
    public static void main(String[] args) {  
        SpringApplication.run(MyApplication.class, args);  
    }  
}
```

Although there is not much code here, quite a lot is going on. We step through the important parts in the next few sections.

The `@RestController` and `@RequestMapping` Annotations

The first annotation on our `MyApplication` class is `@RestController`. This is known as a *stereotype* annotation. It provides hints for people reading the code and for Spring that the class plays a specific role. In this case, our class is a web `@Controller`, so Spring considers it when handling incoming web requests.

The `@RequestMapping` annotation provides “routing” information. It tells Spring that any HTTP request with the `/` path should be mapped to the `home` method. The `@RestController` annotation tells Spring to render the resulting string directly back to the caller.

TIP

The `@RestController` and `@RequestMapping` annotations are Spring MVC annotations (they are not specific to Spring Boot). See the [MVC section](#) in the Spring Reference Documentation for more details.

The `@SpringBootApplication` Annotation

The second class-level annotation is `@SpringBootApplication`. This annotation is known as a *meta-annotation*, it combines `@SpringBootConfiguration`, `@EnableAutoConfiguration` and `@ComponentScan`.

Of those, the annotation we’re most interested in here is `@EnableAutoConfiguration`. `@EnableAutoConfiguration` tells Spring Boot to “guess” how you want to configure Spring, based on the jar dependencies that you have added. Since `spring-boot-starter-web` added Tomcat and Spring MVC, the auto-configuration assumes that you are developing a web application and sets up Spring accordingly.

Starters and Auto-configuration

Auto-configuration is designed to work well with starters, but the two concepts are not directly tied. You are free to pick and choose jar dependencies outside of the starters. Spring Boot still does its best to auto-configure your application.

The “main” Method

The final part of our application is the `main` method. This is a standard method that follows the Java convention for an application entry point. Our main method delegates to Spring Boot’s `SpringApplication` class by calling `run`. `SpringApplication` bootstraps our application, starting Spring, which, in turn, starts the auto-configured Tomcat web server. We need to pass `MyApplication.class` as an argument to the `run` method to tell `SpringApplication` which is the primary Spring component. The `args` array is also passed through to expose any command-line arguments.

Running the Example

Maven

At this point, your application should work. Since you used the `spring-boot-starter-parent` POM, you have a useful `run` goal that you can use to start the application. Type `mvn spring-boot:run` from the root project directory to start the application. You should see output similar to the following:

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

```
$ mvn package

[INFO] Scanning for projects...
[INFO]
[INFO] -----
```

If you look in the `target` directory, you should see `myproject-0.0.1-SNAPSHOT.jar`. The file should be around 18 MB in size. If you want to peek inside, you can use `jar tvf`, as follows:

SHELL

To run that application, use the `java -jar` command, as follows:

SHELL

As before, to exit the application, press `ctrl-c`.

To create an executable jar, we need to run `gradle bootJar` from the command line, as follows:

SHELL

If you look in the `build/libs` directory, you should see `myproject-0.0.1-SNAPSHOT.jar`. The file should be around 18 MB in size. If you want to peek inside, you can use `jar tvf`, as follows:

SHELL

SHELL

As before, to exit the application, press `ctrl-c`.