M. Nardone, C. Scarioni, *Pro Spring Security*
https://doi-org.ezproxy.sfpl.org/10.1007/979-8-8688-0035-1_3

3. Setting up the Scene

Massimo Nardone[1] and Carlo Scarioni[2]
(1) HELSINKI, Finland
(2) Surbiton, UK

This chapter guides you through the process of building your first simple Spring Security 6 project using the IntelliJ IDEA Ultimate Edition 2023.1.2. This involves the following steps.

- Setting up the development environment
- Creating a new Java web application project without Spring Security
- Updating the project with Spring Security
- Running the example

Let's start with setting up the development environment.

Setting up the Development Environment

The following lists the software you need to download and install in the given order.

- Java SE Development Kit (JDK) 17+ (version 20 was the latest when writing this book)
- Maven 3.9.2
- IntelliJ IDEA Ultimate Edition 2023.1.2
- Apache Tomcat Server 10 (External)
- Windows OS (This book uses Windows 11.)

Let's go through the steps required to set up everything properly.

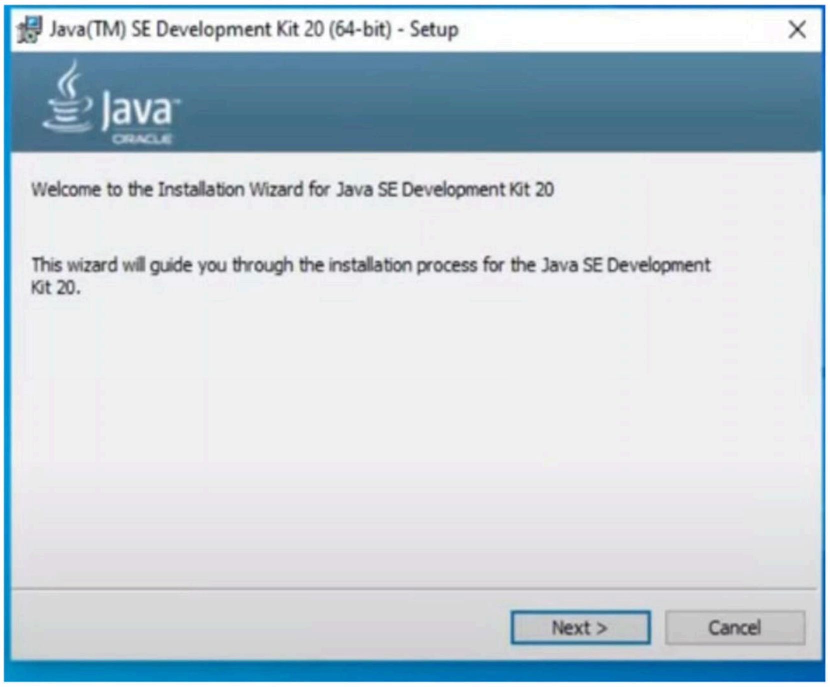
Your first step is to set up the JDK. It comes in an installer or package on most operating systems, so there shouldn't be any problems.

> **Note** Remember that the JDK and Java SE Runtime Environment (JRE) require, at minimum, a Pentium II 266 MHz processor, 128 MB of memory, and 181 MB disk for development tools for 64-bit platforms.

Download the JDK version specific to your Windows operating system from

```
www.oracle.com/technetwork/java/javase/downloads/jdk11-downloads-5066655.html.
```

JDK 20 installed on a Windows 11 machine is used in this book, as shown in Figure 3-1.



Java(TM) SE Development Kit 20 (64-bit) - Setup

Welcome to the Installation Wizard for Java SE Development Kit 20

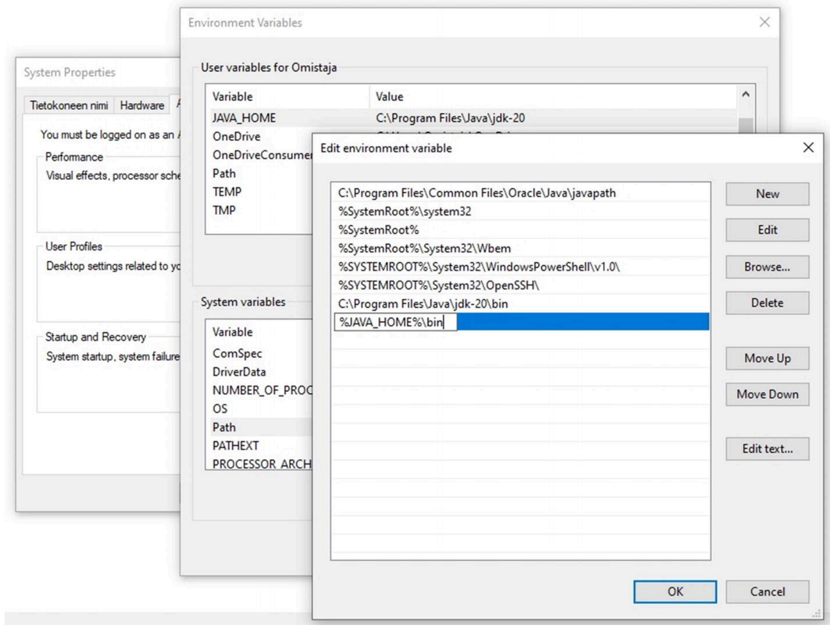
This wizard will guide you through the installation process for the Java SE Development Kit 20.

Figure 3-1 Installing JDK 20

Let's set a `JAVA_HOME` system variable by following these steps.

1. Open the Windows Environment Variables.
2. Add the `JAVA_HOME` variable and point it to the JDK installed folder (e.g., `C:\Program Files\Java\ jdk-20`).
3. Append `%JAVA_HOME%\bin` to the system PATH variable so that all the Java commands are accessible from everywhere.

The result is shown in Figure 3-2.



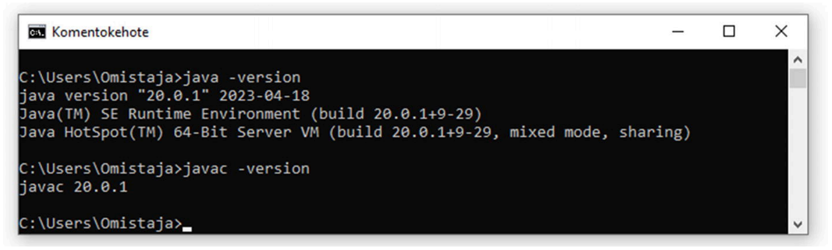
Variable	Value
JAVA_HOME	C:\Program Files\Java\jdk-20
OneDrive	
OneDriveConsumer	
Path	
TEMP	
TMP	

```
C:\Program Files\Common Files\Oracle\Java\javapath
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
C:\Program Files\Java\jdk-20\bin
%JAVA_HOME%\bin
```

System variables: ComSpec, DriverData, NUMBER_OF_PROC, OS, Path, PATHEXT, PROCESSOR_ARCH

Figure 3-2 Setting up the JAVA_HOME system variable

Let's test if the JDK installation was successful. Open a command prompt and type the code shown in Figure 3-3.



```
C:\Users\Omistaja>java -version
java version "20.0.1" 2023-04-18
Java(TM) SE Runtime Environment (build 20.0.1+9-29)
Java HotSpot(TM) 64-Bit Server VM (build 20.0.1+9-29, mixed mode, sharing)

C:\Users\Omistaja>javac -version
javac 20.0.1

C:\Users\Omistaja>_
```

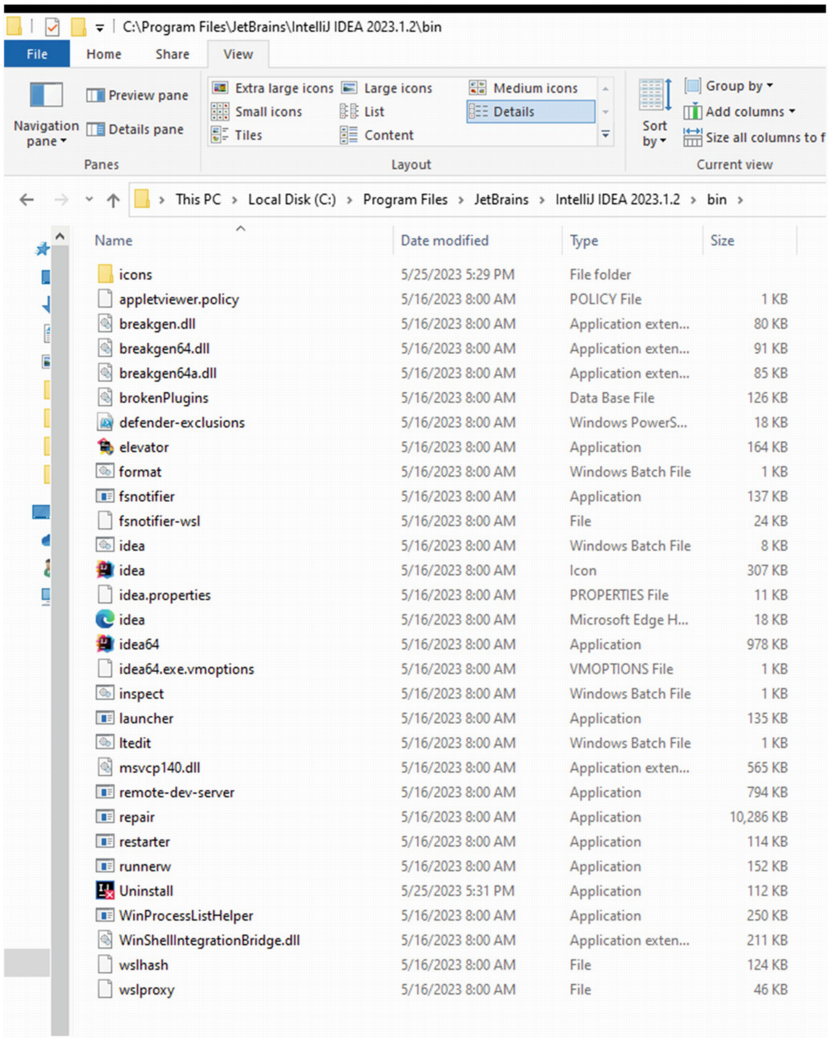
Figure 3-3 Testing the Java installation

Great! Java is now installed and ready to be used for the examples in the book.

Let's install the IntelliJ IDEA Ultimate Edition 2023.1.2 for web and enterprise development by following these steps.

1. Download the `.exe` file from **https://www.jetbrains.com/idea/download/?var=1&section=windows#section=windows**
2. Install the `.exe` file, which in our case is named `ideaIU-2023.1.2.exe`.

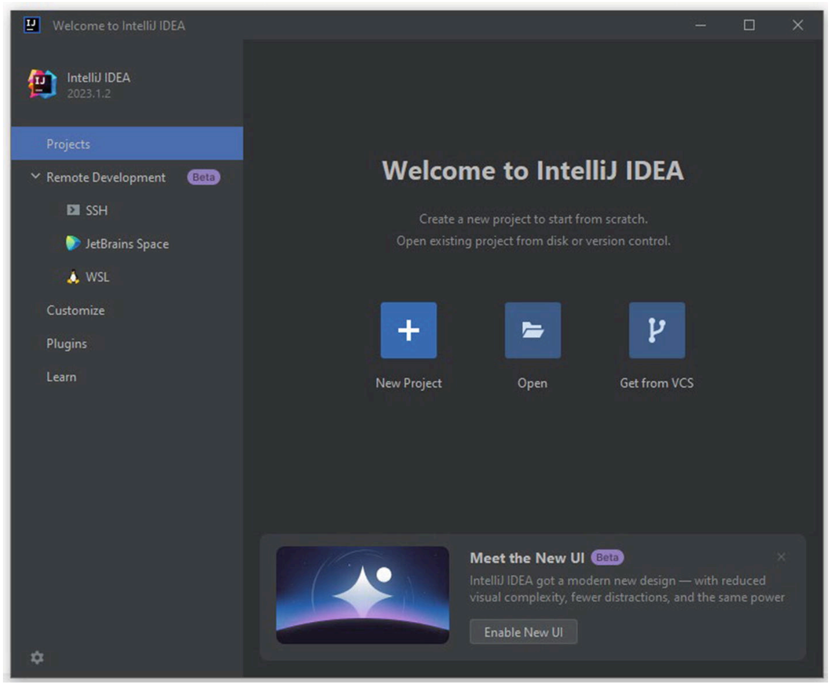
Once installed, the directory should look like Figure 3-4.



Name	Date modified	Type	Size
icons	5/25/2023 5:29 PM	File folder	
appletviewer.policy	5/16/2023 8:00 AM	POLICY File	1 KB
breakgen.dll	5/16/2023 8:00 AM	Application exten...	80 KB
breakgen64.dll	5/16/2023 8:00 AM	Application exten...	91 KB
breakgen64a.dll	5/16/2023 8:00 AM	Application exten...	85 KB
brokenPlugins	5/16/2023 8:00 AM	Data Base File	126 KB
defender-exclusions	5/16/2023 8:00 AM	Windows PowerS...	18 KB
elevator	5/16/2023 8:00 AM	Application	164 KB
format	5/16/2023 8:00 AM	Windows Batch File	1 KB
fsnotifier	5/16/2023 8:00 AM	Application	137 KB
fsnotifier-wsl	5/16/2023 8:00 AM	File	24 KB
idea	5/16/2023 8:00 AM	Windows Batch File	8 KB
idea	5/16/2023 8:00 AM	Icon	307 KB
idea.properties	5/16/2023 8:00 AM	PROPERTIES File	11 KB
idea	5/16/2023 8:00 AM	Microsoft Edge H...	18 KB
idea64	5/16/2023 8:00 AM	Application	978 KB
idea64.exe.vmoptions	5/16/2023 8:00 AM	VMOPTIONS File	1 KB
inspect	5/16/2023 8:00 AM	Windows Batch File	1 KB
launcher	5/16/2023 8:00 AM	Application	135 KB
ltedit	5/16/2023 8:00 AM	Windows Batch File	1 KB
msvcp140.dll	5/16/2023 8:00 AM	Application exten...	565 KB
remote-dev-server	5/16/2023 8:00 AM	Application	794 KB
repair	5/16/2023 8:00 AM	Application	10,286 KB
restarter	5/16/2023 8:00 AM	Application	114 KB
runnerw	5/16/2023 8:00 AM	Application	152 KB
Uninstall	5/25/2023 5:31 PM	Application	112 KB
WinProcessListHelper	5/16/2023 8:00 AM	Application	250 KB
WinShellIntegrationBridge.dll	5/16/2023 8:00 AM	Application exten...	211 KB
wslhash	5/16/2023 8:00 AM	File	124 KB
wslproxy	5/16/2023 8:00 AM	File	46 KB

Figure 3-4 The IntelliJ IDEA 2023.1.2 directory

Now IntelliJ IDEA Ultimate Edition 2023.1.2 for web and enterprise development tool is ready to be used. Figure 3-5 shows how the dashboard looks when executing it.



Welcome to IntelliJ IDEA

Create a new project to start from scratch.
Open existing project from disk or version control.

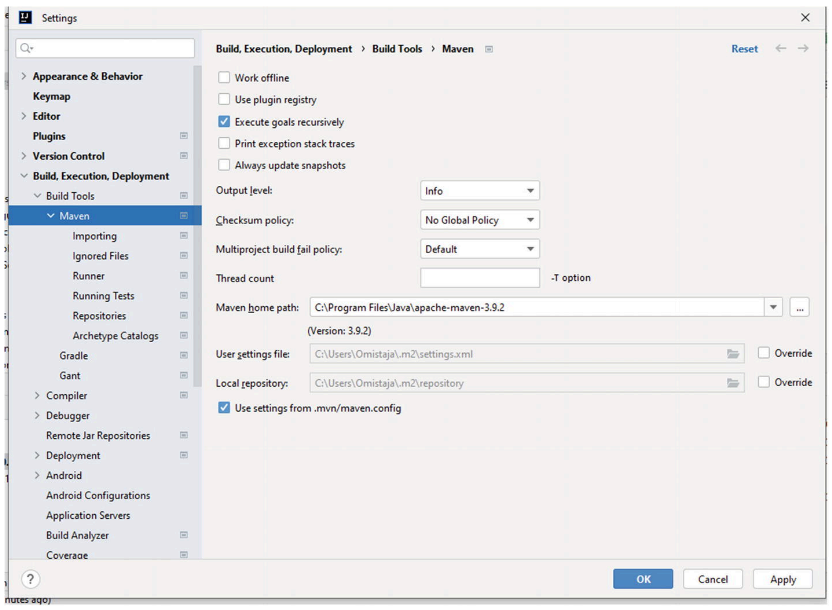
Meet the New UI

IntelliJ IDEA got a modern new design — with reduced visual complexity, fewer distractions, and the same power

Figure 3-5 The IntelliJ IDEA Ultimate Edition 2023.1.2 for web and enterprise development dashboard

The next step is to install Maven 3.9.2 by downloading the `.zip` file named `apache-maven-3.9.2-bin.zip` at **https://maven.apache.org/download.cgi**.

Now run the IntelliJ IDEA 2023.1.2 tool and configure Maven 3.6.1. as shown in Figure **3-6**.



Build, Execution, Deployment › Build Tools › Maven

- [ ] Work offline
- [ ] Use plugin registry
- [✓] Execute goals recursively
- [ ] Print exception stack traces
- [ ] Always update snapshots

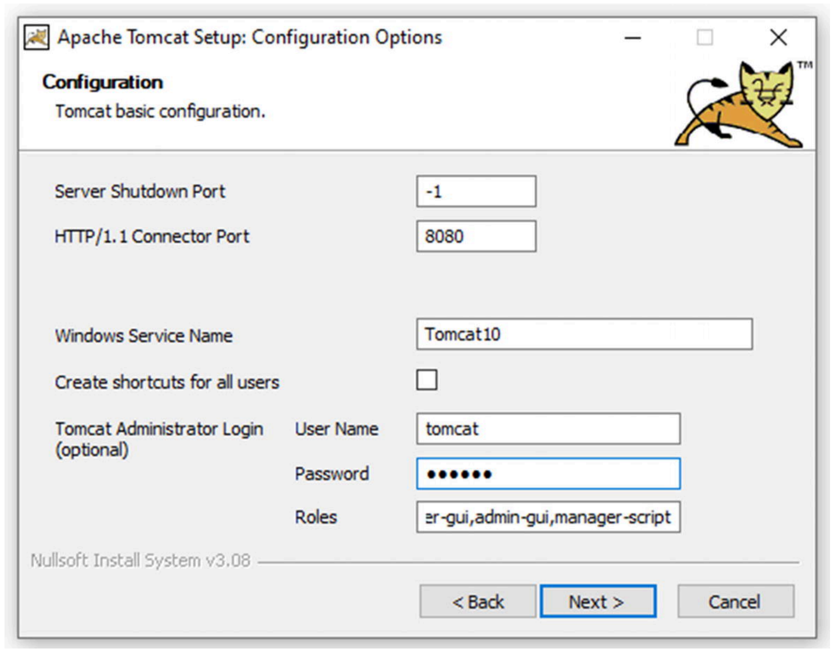
Output level: Info
Checksum policy: No Global Policy
Multiproject build fail policy: Default
Thread count: -T option
Maven home path: C:\Program Files\Java\apache-maven-3.9.2
(Version: 3.9.2)
User settings file: C:\Users\Omistaja\.m2\settings.xml [ ] Override
Local repository: C:\Users\Omistaja\.m2\repository [ ] Override
[✓] Use settings from .mvn/maven.config

Figure 3-6 Local Maven 3.9.2 is configured into IntelliJ IDEA 2023.1.2

Now, Maven 3.9.2 is ready to be used.

The last tool used in this book is the Apache Tomcat Server and plugin 10. The first step is to download and install the Apache Tomcat Server 10 `.exe` file named `apache-tomcat-10.1.9.exe` at **https://tomcat.apache.org/download-10.cgiInstall** the exe file to the default folder, which is C:\Program Files\Apache Software Foundation\Tomcat 10.1. Since you need to allow Spring projects to deploy to Tomcat Servers, you need to define Tomcat users to access Tomcat Manager. This can be done when installing Tomacat 10, as shown in Figure **3-7**, or by manually updating the file named `tomcat-users.xml` in the `conf` directory and adding the following XML fragment inside the `<tomcat-users>` element.

```
<role rolename="manager-gui"/>

<role rolename="manager-script"/>

<user username="tomcat" password="tomcat" roles="manager-gui, manager-script"/>
```



Apache Tomcat Setup: Configuration Options

Configuration
Tomcat basic configuration.

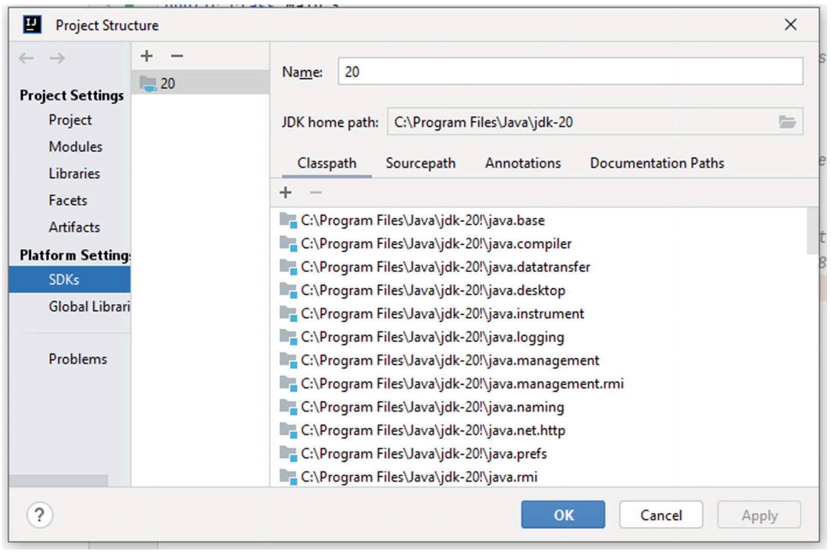
Server Shutdown Port		-1
HTTP/1.1 Connector Port		8080
Windows Service Name		Tomcat10
Create shortcuts for all users		☐
Tomcat Administrator Login (optional)	User Name	tomcat
	Password	••••••
	Roles	er-gui,admin-gui,manager-script

Figure 3-7 Installation of Tomacat 10 with new roles

Make sure you add the Tomcat Server to IntelliJ.

Now, the Apache Tomcat Server and plugin 10 are ready to be used.

Before starting a new Spring project, you want to make sure the right JDK package is installed into the IntelliJ IDEA 2023.1.2 IDE tool to compile your examples and avoid the typical compiling issue where the JRE is found instead of JDK. The configuration is shown in Figure 3-8.



Project Structure

Name: 20

JDK home path: C:\Program Files\Java\jdk-20

```
C:\Program Files\Java\jdk-20!\java.base
C:\Program Files\Java\jdk-20!\java.compiler
C:\Program Files\Java\jdk-20!\java.datatransfer
C:\Program Files\Java\jdk-20!\java.desktop
C:\Program Files\Java\jdk-20!\java.instrument
C:\Program Files\Java\jdk-20!\java.logging
C:\Program Files\Java\jdk-20!\java.management
C:\Program Files\Java\jdk-20!\java.management.rmi
C:\Program Files\Java\jdk-20!\java.naming
C:\Program Files\Java\jdk-20!\java.net.http
C:\Program Files\Java\jdk-20!\java.prefs
C:\Program Files\Java\jdk-20!\java.rmi
```

Figure 3-8 Configuring the JDK to compile your examples

So now the JDK compiler is set, and you are ready to start writing and running your first Spring web application example.

Creating a New Java Web Application Project

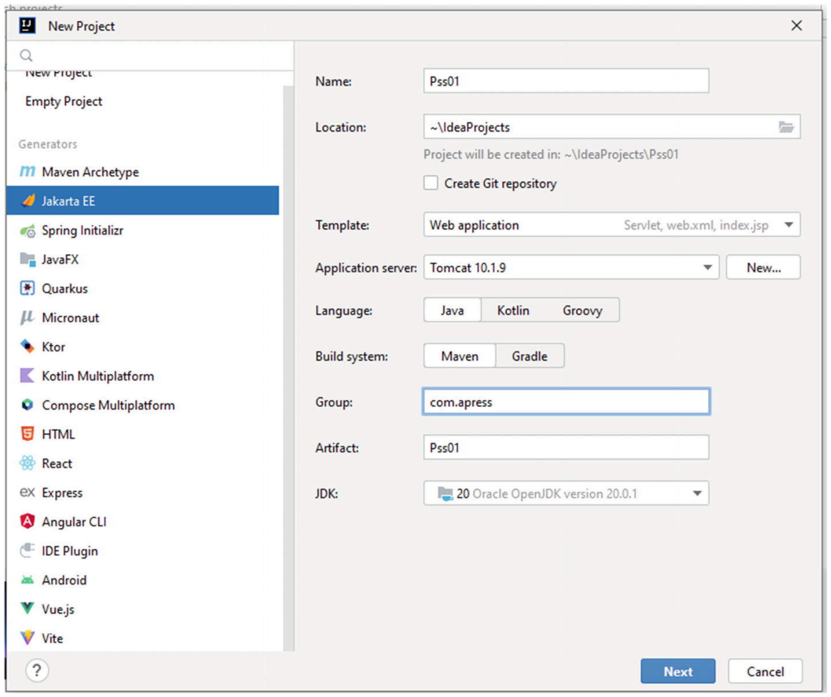
With your development tools set up, you can now create your first Java web application project using IntelliJ IDEA 2023.1.2. The built-in wizard makes creating a new Maven project very easy.

So, let’s create your first Java EE web application named Pss01, without security, which produces the following text: Hello Spring Security!

Here are the steps to build a simple Maven web application project.

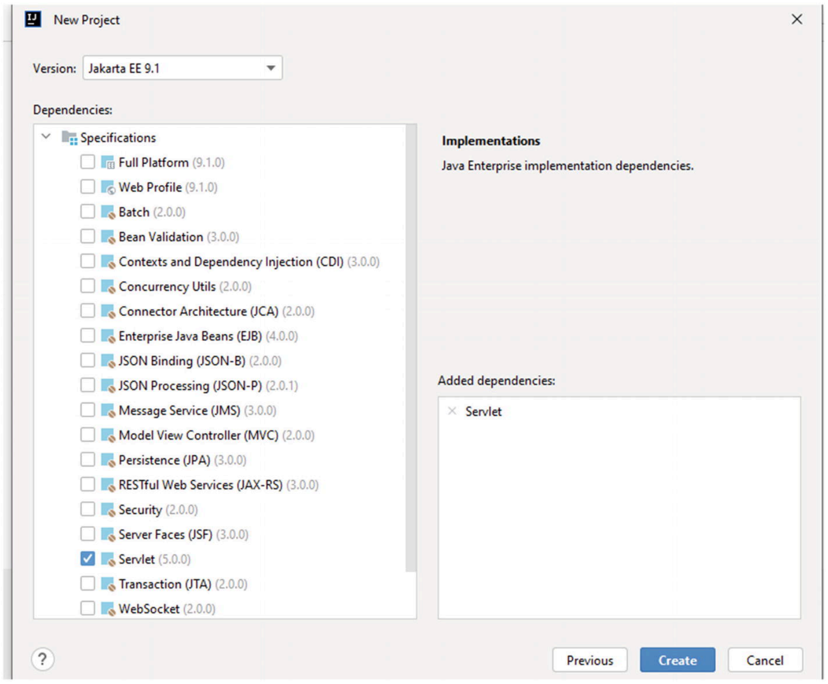
1. Create a Java EE web application.
2. Create and update the needed `.jsp` file.
3. Run the Java web application using the external Tomcat Server 10.

First, launch the IntelliJ IDEA tool and select File ➤ New ➤ Project ➤ Jakarta EE ➤ Web Application and fill in all information about the project, as shown in Figures 3-9 and 3-10.



Name: Pss01
Location: ~\IdeaProjects
Project will be created in: ~\IdeaProjects\Pss01
Create Git repository
Template: Web application — Servlet, web.xml, index.jsp
Application server: Tomcat 10.1.9 — New...
Language: Java | Kotlin | Groovy
Build system: Maven | Gradle
Group: com.apress
Artifact: Pss01
JDK: 20 Oracle OpenJDK version 20.0.1

Figure 3-9 Your first Java web application project



New Project

Version: Jakarta EE 9.1

Dependencies:

- Specifications
  - [ ] Full Platform (9.1.0)
  - [ ] Web Profile (9.1.0)
  - [ ] Batch (2.0.0)
  - [ ] Bean Validation (3.0.0)
  - [ ] Contexts and Dependency Injection (CDI) (3.0.0)
  - [ ] Concurrency Utils (2.0.0)
  - [ ] Connector Architecture (JCA) (2.0.0)
  - [ ] Enterprise Java Beans (EJB) (4.0.0)
  - [ ] JSON Binding (JSON-B) (2.0.0)
  - [ ] JSON Processing (JSON-P) (2.0.1)
  - [ ] Message Service (JMS) (3.0.0)
  - [ ] Model View Controller (MVC) (2.0.0)
  - [ ] Persistence (JPA) (3.0.0)
  - [ ] RESTful Web Services (JAX-RS) (3.0.0)
  - [ ] Security (2.0.0)
  - [ ] Server Faces (JSF) (3.0.0)
  - [✓] Servlet (5.0.0)
  - [ ] Transaction (JTA) (2.0.0)
  - [ ] WebSocket (2.0.0)

Implementations

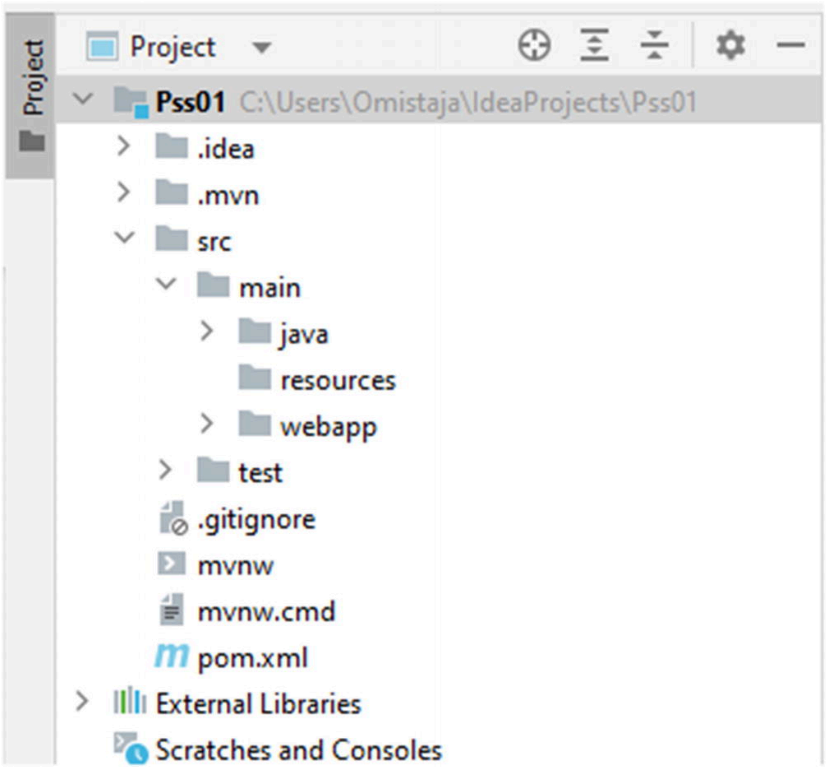
Java Enterprise implementation dependencies.

Added dependencies:

Servlet

Figure 3-10 Configuration for your first Java web application project

In Package Explorer, you should now see your Pss01 project. If you expand it and all its children, you'll see something like Figure **3-11**.



```
Pss01 C:\Users\Omistaja\IdeaProjects\Pss01
  .idea
  .mvn
  src
    main
      java
      resources
      webapp
    test
  .gitignore
  mvnw
  mvnw.cmd
  pom.xml
External Libraries
Scratches and Consoles
```

Figure 3-11 Your first Java web application project structure

In general, the structure of most Java web application projects contains the following.

- The target directory houses all the output of the build.
- The `src` directory contains all the source material for building the project, its site, etc.

- `src/main/java` contains the application/library sources.
- `src/main/resources` contains the application /library resources
- `web` contains the web application sources.
- `pom.xml` is a file that describes the project.

Your next step is to update the Java web application project's files needed for your first simple application. Please note that for this simple Java web application example, you do not need to add any specific dependency to the project file `pom.xml`, which looks initially like Listing 3-1.

```
<?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/mave

  <modelVersion>4.0.0</modelVersion>




  <groupId>com.apress</groupId>

  <artifactId>Pss01</artifactId>

  <version>1.0-SNAPSHOT</version>

  <name>Pss01</name>

  <packaging>war</packaging>
```

Listing 3-1
The pom.xml File with Servlet Dependencies

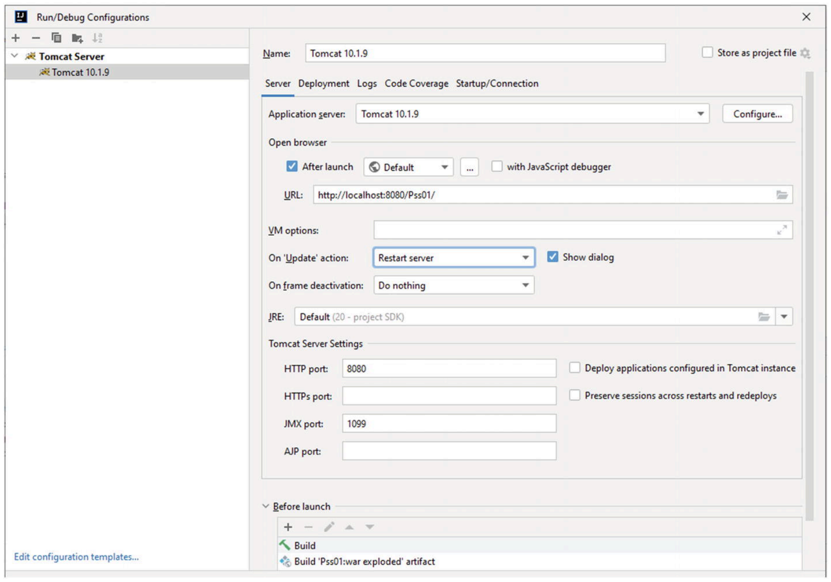
The project right now only contains one simple `.jsp` file named `index.jsp`, which you update to show the text you wish, as shown in Listing 3-2.

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
  <head>
    <title>$Title$</title>
  </head>
  <body>
     <h2>Hello Spring Security!</h2>
  </body>
</html>
```

Listing 3-2
The index.jsp File

Next, click the Add Configuration button at the top-right of the IntelliJ tool to configure how to run your first example.

You can run your project using the external Tomcat Server 10, as shown in Figure 3-12.


Run/Debug Configurations
Tomcat Server
Tomcat 10.1.9
Server Deployment Logs Code Coverage Startup/Connection
Configure...
Edit configuration templates...

Name: Tomcat 10.1.9
Store as project file: ☐
Application server: Tomcat 10.1.9
Open browser
After launch: ✓ Default
with JavaScript debugger: ☐
URL: http://localhost:8080/Pss01/
VM options:
On 'Update' action: Restart server
Show dialog: ✓
On frame deactivation: Do nothing
JRE: Default (20 - project SDK)
Tomcat Server Settings
HTTP port: 8080
Deploy applications configured in Tomcat instance: ☐
HTTPs port:
Preserve sessions across restarts and redeploys: ☐
JMX port: 1099
AJP port:
Before launch
Build
Build 'Pss01:war exploded' artifact

Figure 3-12 Configure the running steps of your first Maven project

Now you can open you web browser and type the web address `http://localhost:8080/Pss01`, as shown in Figure 3-13.

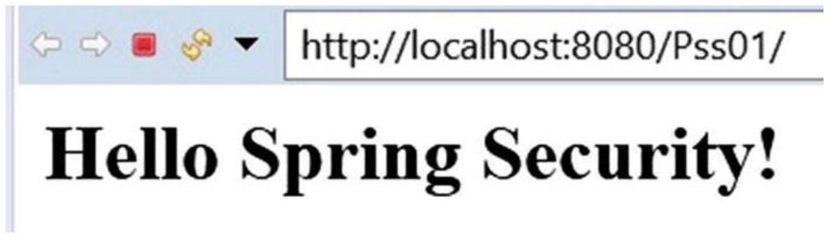


Figure 3-13 The Java web application project running in a web browser

Your first Java web application project was done now, so let's create a new Spring Security 6 project.

Adding Spring Security 6 to the Java Project

Spring Security builds upon the concepts defined in the previous section and integrates nicely into the general Spring ecosystem. You need to understand those concepts well to take maximum advantage of Spring Security 6. However, you can start using Spring Security without knowing all these details and then learn them as you progress and look to do more advanced things.

There are two ways to create a new Spring project.

You can create a Spring project via Spring Initializr, as discussed in Chapter 5, or via any IDE tool, which is IntelliJ IDEA 2023.1.2.

What kind of Spring Security Maven web application do you want to create?

Let's create a simple Spring Security example where the user must be authenticated as a user or an admin to access a certain secure project resource.

If you are using the stand-alone installation of Spring Security reference release and you decide not to use any IDE tool to build your Maven project, you will find many folders inside the installation directory. Most of the folders in the directory correspond to individual subprojects or modules that split the functionality of Spring Security into more discrete and specialized units.

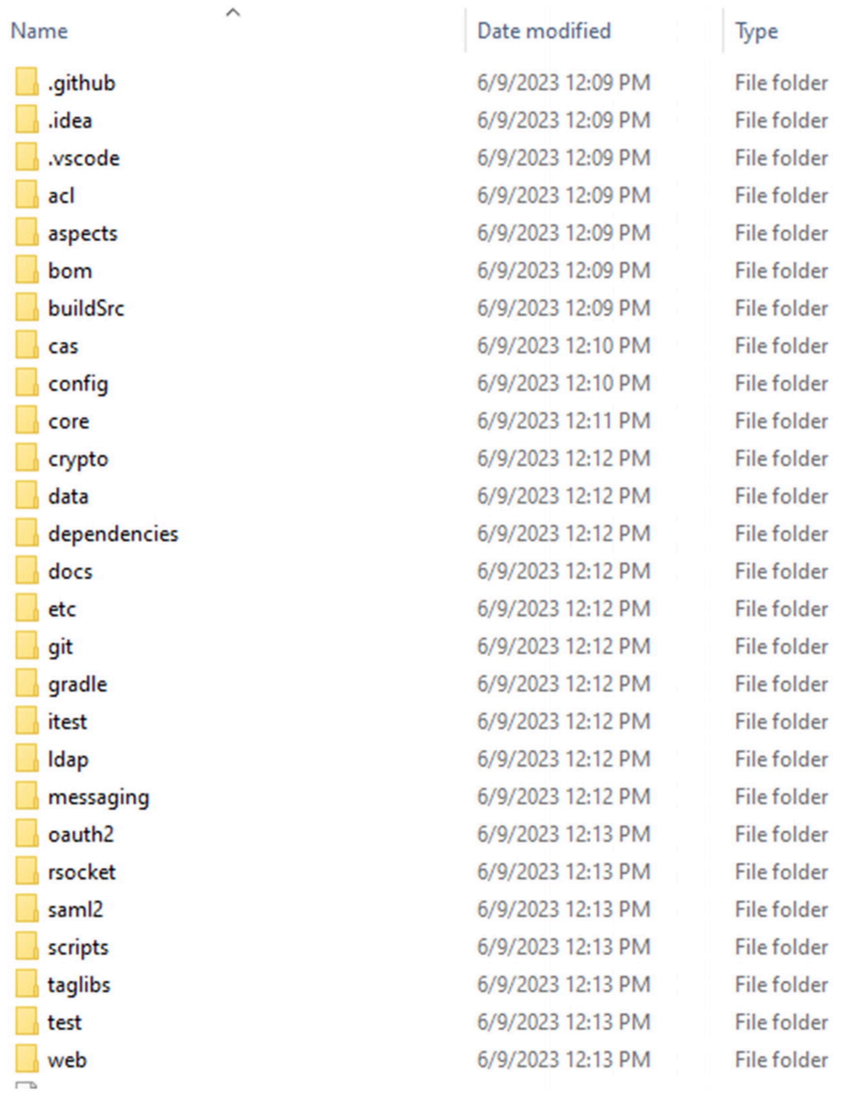
Spring Security 6 Source

Open source software has an invaluable characteristic for software developers: free access to all source code. With this, you can understand how tools and frameworks work internally. You can also learn a lot about how other (perhaps very good) developers work, including their practices, techniques, and patterns. Free access to source code also enables us, in general, to gather ideas and experience for our development. As a more practical matter, having access to the source code allows you to debug these applications in the context of our application; you can find bugs or simply follow your application's execution through them.

Currently, Spring Security and most Spring projects live on GitHub. You probably know about GitHub (`https://github.com/`). If you don't, you should look at it because it has become a standard public source-code repository for many open source projects in many programming languages.

GitHub is a repository and a hosting service for Git repositories with a very friendly management interface. The Spring Security project is in the SpringSource general GitHub section at `https://github.com/SpringSource/spring-security`. To get the code, just download and install it, as discussed earlier in this chapter.

Spring Security 6.1.0.RELEASE includes several modules and folders, as shown in Figure **3-14**.



Name	Date modified	Type
.github	6/9/2023 12:09 PM	File folder
.idea	6/9/2023 12:09 PM	File folder
.vscode	6/9/2023 12:09 PM	File folder
acl	6/9/2023 12:09 PM	File folder
aspects	6/9/2023 12:09 PM	File folder
bom	6/9/2023 12:09 PM	File folder
buildSrc	6/9/2023 12:09 PM	File folder
cas	6/9/2023 12:10 PM	File folder
config	6/9/2023 12:10 PM	File folder
core	6/9/2023 12:11 PM	File folder
crypto	6/9/2023 12:12 PM	File folder
data	6/9/2023 12:12 PM	File folder
dependencies	6/9/2023 12:12 PM	File folder
docs	6/9/2023 12:12 PM	File folder
etc	6/9/2023 12:12 PM	File folder
git	6/9/2023 12:12 PM	File folder
gradle	6/9/2023 12:12 PM	File folder
itest	6/9/2023 12:12 PM	File folder
ldap	6/9/2023 12:12 PM	File folder
messaging	6/9/2023 12:12 PM	File folder
oauth2	6/9/2023 12:13 PM	File folder
rsocket	6/9/2023 12:13 PM	File folder
saml2	6/9/2023 12:13 PM	File folder
scripts	6/9/2023 12:13 PM	File folder
taglibs	6/9/2023 12:13 PM	File folder
test	6/9/2023 12:13 PM	File folder
web	6/9/2023 12:13 PM	File folder

Figure 3-14 The Spring Security 6.1.0.RELEASE folder structure

The following are short descriptions of some of the most important modules included in Spring Security 6.1.0.RELEASE.

- Core (spring-security-core) is where Spring Security's core classes and interfaces on authentication and access control reside.
- Remoting (spring-security-remoting) is the module with the remoting classes.
- Aspect (spring-security-aspects) provides aspect-oriented programming support within Spring Security.
- Config (spring-security-config) provides XML and Java configuration support.
- Crypto (spring-security-crypto) provides cryptography support.
- Data (spring-security-data) supports integration with Spring Data.
- Messaging (spring-security-messaging) supports Spring Security messaging.
- OAuth 2.x provides support within Spring Security.
  - – Core (spring-security-oauth2-core)
  - – Client (spring-security-oauth2-client)
  - – JOSE (spring-security-oauth2-jose)

- CAS (spring-security-cas) (Central Authentication Service) supports client integration.
- TagLib (spring-security-taglibs) provides various Spring Security tag libraries.
- Test (spring-security-test) provides testing support.
- Web (spring-security-web) contains web security infrastructure code, such as filters and other Servlet API dependencies.

Note Remember that you are using the IntelliJ IDEA tool, where the Spring Security 6.1.0.RELEASE is integrated and configured in it. Spring Security is used via an XML link at the beginning of the `pom.xml` file.

Let's update our previous project or create a new one named Pss02 and add Spring Security 6.

Here are the steps to build a simple Spring Security Maven web application project.

1. Import the required Spring Framework and Spring Security 6 libraries into the project (into the pom.xml file).
2. Configure the project to be aware of Spring Security.
3. Configure the users and roles that will be part of the system.
4. Configure the URLs that you want to secure.
5. Create all needed Java and web files.
6. Run the Spring Security 6 project using the external Tomcat Server 10.

Note Implementing the Spring Security in a Spring application using XML- or Java-based configurations is possible. In this chapter, you use the Java configuration for your Spring Security web application since, in general, it is hardly suggested to use XML configuration as minimum as possible.

Since we are using Maven, the first step is to include Spring Security JARs dependencies in pom.xml.

- spring-security-core
- spring-security-config
- spring-security-web
- spring-webmvc

The following are the Maven dependencies you must add to the pom.xml file.

```
<dependency>

    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-core</artifactId>

    <version>6.1.0</version>

</dependency>

<dependency>

    <groupId>org.springframework.security</groupId>

    <artifactId>spring-security-config</artifactId>

    <version>6.1.0</version>

<dependency>

    <groupId>org.springframework.security</groupId>

    <artifactId>spring-security-web</artifactId>

    <version>6.1.0</version>

</dependency>

<dependency>

    <groupId>org.springframework</groupId>

    <artifactId>spring-webmvc</artifactId>

    <version>6.0.9</version>

</dependency>
```

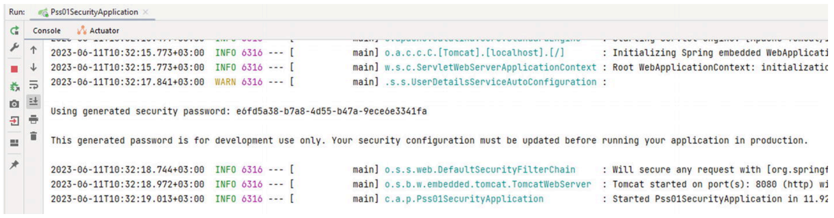
The new `pom.xml` file is generated when the new Spring Boot 3 and Spring Security 6 project is created, as shown in Listing 3-3.

```
<?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/mave

    <modelVersion>4.0.0</modelVersion>



    <groupId>com.apress</groupId>

    <artifactId>Pss02</artifactId>

    <version>1.0-SNAPSHOT</version>

    <name>Pss02</name>

    <packaging>war</packaging>



    <properties>

        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

        <maven.compiler.target>11</maven.compiler.target>
        <maven.compiler.source>11</maven.compiler.source>

        <junit.version>5.9.2</junit.version>

    </properties>




    <dependencies>

       <dependency>

            <groupId>org.springframework.security</groupId>

            <artifactId>spring-security-core</artifactId>

            <version>6.1.0</version>

        </dependency>

        <dependency>

            <groupId>org.springframework.security</groupId>

            <artifactId>spring-security-config</artifactId>

            <version>6.1.0</version>

        </dependency>

        <dependency>

            <groupId>org.springframework.security</groupId>

            <artifactId>spring-security-web</artifactId>
            <version>6.1.0</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>6.0.9</version>
        </dependency>
        <dependency>
            <groupId>jakarta.servlet</groupId>
            <artifactId>jakarta.servlet-api</artifactId>
            <version>5.0.0</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-api</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
        </dependency>
    </dependencies>


    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>3.3.2</version>
            </plugin>
        </plugins>
    </build>
```

```
</project>
```

Listing 3-3 pom.xml

Since Spring Security was added to the project, it secures the entire project by default. It gives a generated security password that is entered with “user” as the username, as shown in Figure 3-15.



```
2023-06-11T10:32:15.773+03:00  INFO 6316 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicati
2023-06-11T10:32:15.773+03:00  INFO 6316 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initializatic
2023-06-11T10:32:17.841+03:00  WARN 6316 --- [           main] .s.s.UserDetailsServiceAutoConfiguration :

Using generated security password: e6fd5a38-b7a8-4d55-b47a-9ece6e3341fa

This generated password is for development use only. Your security configuration must be updated before running your application in production.

2023-06-11T10:32:18.744+03:00  INFO 6316 --- [           main] o.s.s.web.DefaultSecurityFilterChain     : Will secure any request with [org.springf
2023-06-11T10:32:18.972+03:00  INFO 6316 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) wi
2023-06-11T10:32:19.013+03:00  INFO 6316 --- [           main] c.a.p.Pss01SecurityApplication           : Started Pss01SecurityApplication in 11.92
```

Figure 3-15 Running the new Spring project

This means that if you type localhost:8080, Spring requires you to provide the newly created username (user) and password (e6fd5a38-b7a8-4d55-b47a-9ece6e3341fa) to log in, as shown in Figure 3-16.

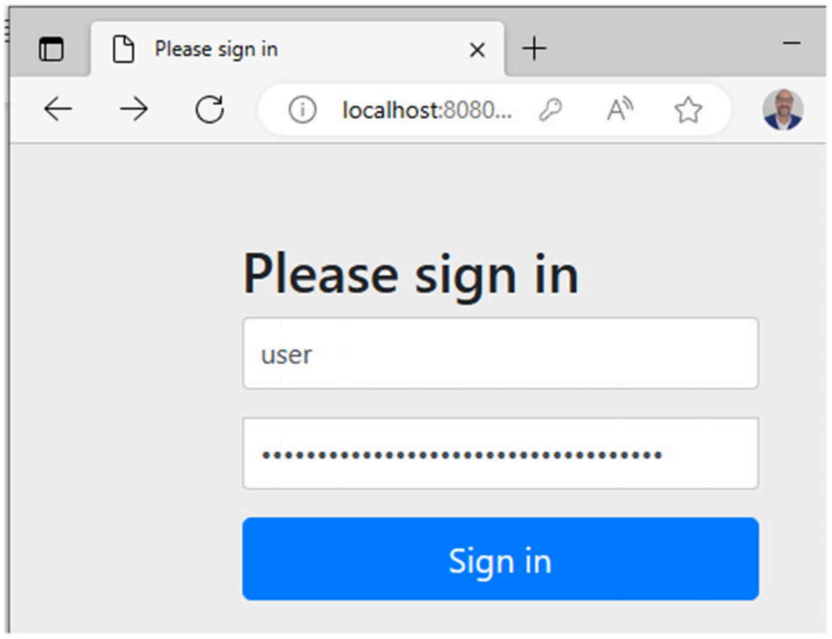
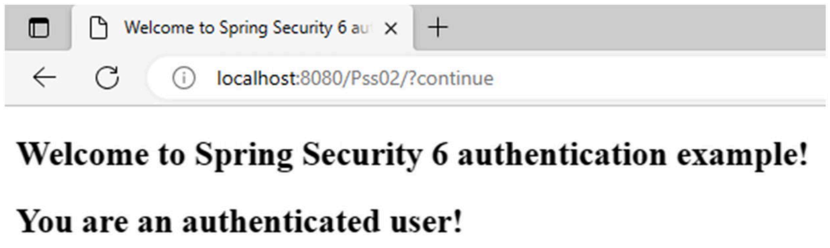


Figure 3-16 Secure Spring application with login page

If you enter the correct username and password, you get the “Welcome to Spring Security 6” message, as shown in Figure 3-17.


Welcome to Spring Security 6 au
localhost:8080/Pss02/?continue

Welcome to Spring Security 6 authentication example!

You are an authenticated user!

Figure 3-17 Successful login message

Configuring the Spring Security 6 Web Project

To activate Spring Security web project configuration in your Maven web application, you need to configure a particular Servlet filter that takes care of preprocessing and postprocessing the requests and managing the required security constraints.

Let's add now some more logic to our code.

First, let's create the Java package where all your Java classes will be located.

- com.apress.pss02.springsecurity.configuration

Then, you must define the Java classes needed for your example under package configuration.

- SecurityConfiguration
- AppInitializer
- SpringSecurityInitializer

In this example, you learn how to enable Spring Security 6 using the @EnableWebSecurity annotation without using the WebSecurityConfigurerAdapter class; however, this example is built on top of the spring-webmvc Hibernate integration example.

Let's create a new Java Spring Security configuration class named `SecurityConfiguration`, which utilizes the @EnableWebSecurity annotation to configure Spring Security–related beans such as WebSecurityConfigurer or SecurityFilterChain.

In the new Spring Security 6 `SecurityConfiguration` Java class (see Listing 3-4), you must do the following.

- Create two demo in-memory users named “user” and “admin”, authorized to access a secure project resource.
- Use BCryptPasswordEncoder to encode the user passwords for added security.
- Configure the SecurityFilterChain bean with the HTTP-based method login to the application as basic-auth.

```
package com.apress.pss02.configuration;


import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.core.userdetails.User;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.security.provisioning.InMemoryUserDetailsManager;

import org.springframework.security.web.SecurityFilterChain;


import static org.springframework.security.config.Customizer.withDefaults;
```

```
@Configuration

@EnableWebSecurity



public class SecurityConfiguration {



    @Bean

    public SecurityFilterChain filterChain1(HttpSecurity http) throws Exception {

        http

                .authorizeHttpRequests((authorize) -> authorize

                        .anyRequest().authenticated()



                )

                .formLogin(withDefaults());

        return http.build();

    }



    @Bean
    public UserDetailsService userDetailsService(){

        UserDetails user = User.builder()
                .username("user")
                .password(passwordEncoder().encode("userpassw"))
                .roles("USER")
                .build();

        UserDetails admin = User.builder()
                .username("admin")
                .password(passwordEncoder().encode("adminpassw"))
                .roles("ADMIN")
                .build();

        return new InMemoryUserDetailsManager(user, admin);
    }
    @Bean
    public static PasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }


}
```

Listing 3-4
SecurityConfiguration Java Class

Since Spring Security is implemented using DelegatingFilterProxy, the next step is to create a new Java class named `SpringSecurityInitializer` to initialize Spring Security using the AbstractSecurityWebApplicationInitializer class. This is done so that Spring can do the following.

- Detect the instance of this class during application startup
- Register the DelegatingFilterProxy to use the springSecurityFilter-Chain before any other registered Filter
- Register a ContextLoaderListener

The `SpringSecurityInitializer` Java class is shown in Listing **3-5**.

```
package com.apress.pss02.configuration;

import org.springframework.security.web.context.AbstractSecurityWebApplicationInitializer;
```

```
public class SpringSecurityInitializer extends AbstractSecurityWebApplicationInitializer {

    //no code needed

}
```

Listing 3-5
SpringSecurityInitializer Java Class

Next, include the SecurityConfiguration class to the new `AppInitializer` Java class, used to initialize the HibernateConfig, SecurityConfiguration, and WebMvcConfig classes, as shown in Listing **3-6**.

```
package com.apress.pss02.configuration;

import jakarta.servlet.ServletContext;

import org.springframework.security.access.SecurityConfig;

import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.ContextLoaderListener;

import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;

import org.springframework.web.filter.DelegatingFilterProxy;




public class AppInitializer implements WebApplicationInitializer {




    @Override

    public void onStartup(ServletContext sc) {




        AnnotationConfigWebApplicationContext root = new AnnotationConfigWebApplicationContext()

        root.register(SecurityConfiguration.class);




        sc.addListener(new ContextLoaderListener(root));




        sc.addFilter("securityFilter", new DelegatingFilterProxy("springSecurityFilterChain"))

                .addMappingForUrlPatterns(null, false, "/*");

    }
}
```

Listing 3-6
AppInitializer Java Class

Finally, update the index.jsp page, as shown in Listing 3-7.

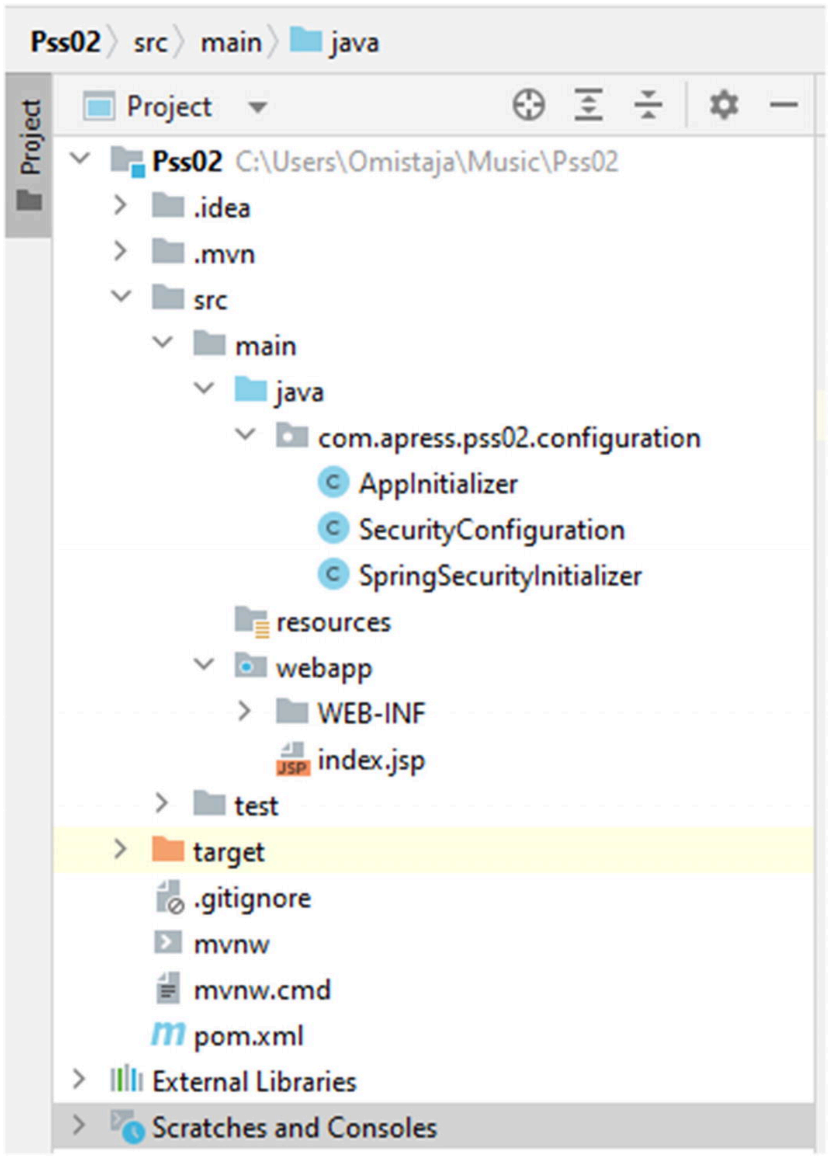
```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>

<!DOCTYPE html>

<html>

<head>

    <title>Welcome to Spring Security 6 authentication example!</title>

</head>

<body>

<h2>Welcome to Spring Security 6 authentication example!</h2>



<h2>You are an authenticated user!</h2>



</body>
```

```
</html>
```

Listing 3-7 index.jsp

The `index.jsp` page only displays a welcoming message if the user is authenticated.

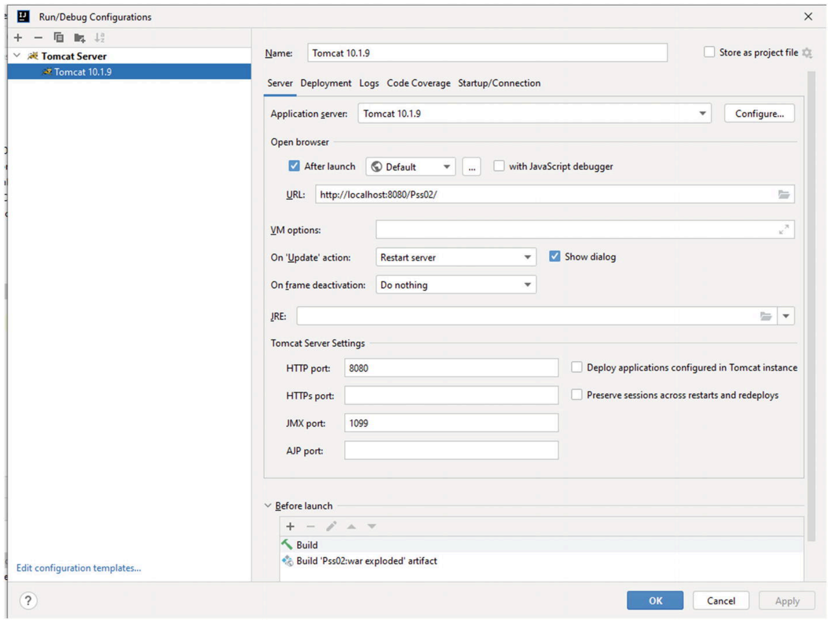
The structure of your new Spring Security 6 project should look like Figure 3-18.



```
Pss02 > src > main > java
Project
Pss02 C:\Users\Omistaja\Music\Pss02
  > .idea
  > .mvn
  v src
    v main
      v java
        v com.apress.pss02.configuration
            AppInitializer
            SecurityConfiguration
            SpringSecurityInitializer
        resources
      v webapp
        > WEB-INF
          index.jsp
    > test
  > target
    .gitignore
    mvnw
    mvnw.cmd
    pom.xml
> External Libraries
> Scratches and Consoles
```

Figure 3-18 New Spring Security 6 project structure

Next, build and run the Spring Security 6 project using Tomcat 10, as shown in Figure 3-19.



Run/Debug Configurations

Tomcat Server
Tomcat 10.1.9

Name: Tomcat 10.1.9 — Store as project file

Server | Deployment | Logs | Code Coverage | Startup/Connection

Application server: Tomcat 10.1.9 — Configure...

Open browser
[✓] After launch — Default — [ ] with JavaScript debugger
URL: http://localhost:8080/Pss02/

VM options:
On 'Update' action: Restart server — [✓] Show dialog
On frame deactivation: Do nothing
JRE:

Tomcat Server Settings
HTTP port: 8080 — [ ] Deploy applications configured in Tomcat instance
HTTPs port: — [ ] Preserve sessions across restarts and redeploys
JMX port: 1099
AJP port:

Before launch
Build
Build 'Pss02:war exploded' artifact

Edit configuration templates...

OK | Cancel | Apply

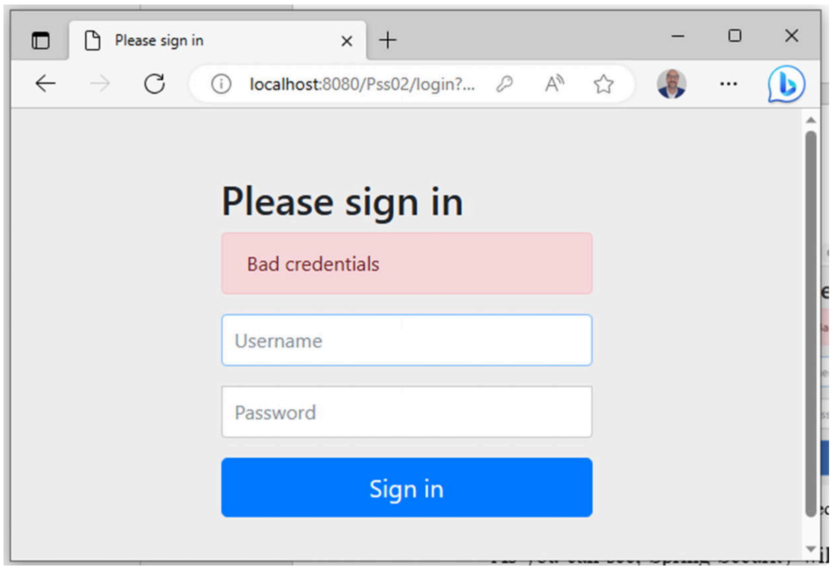
Figure 3-19 Project running configuration using Tomcat 10

You can now build the project, deploy the JAR file, start the application running on the stand-alone Tomcat Server 10, and deploy the JAR file automatically.

Your application is deployed successfully. The web browser automatically opens `http://localhost:8080/ /Pss02/login/`. The outcome is shown in Figure 3-20.

Figure 3-20 Accessing the Spring Security login web page

If you try to access using the wrong credentials, you receive an error message like the one shown in Figure 3-21.



Please sign in

Bad credentials

Username

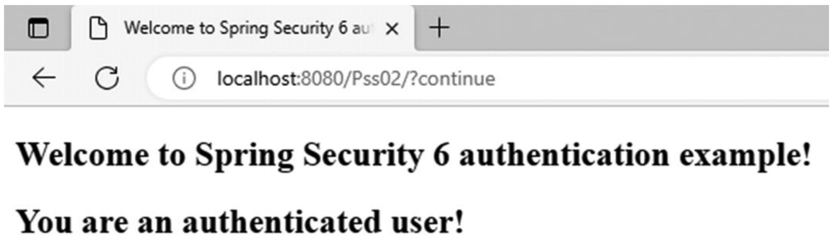
Password

Sign in

Figure 3-21 Accessing with the wrong login credentials

As you can see, Spring Security directly produces a login error and reminds the user that the credentials provided are incorrect.

If you provide the correct user or admin credentials, you receive the content defined in the `index.jsp` page, which identifies the credentials and displays a welcome message, as shown in Figure 3-22.


Welcome to Spring Security 6 au
localhost:8080/Pss02/?continue

Welcome to Spring Security 6 authentication example!

You are an authenticated user!

Figure 3-22 Accessing with the right admin credentials

The admin login iteration flow is shown in Figure 3-23.

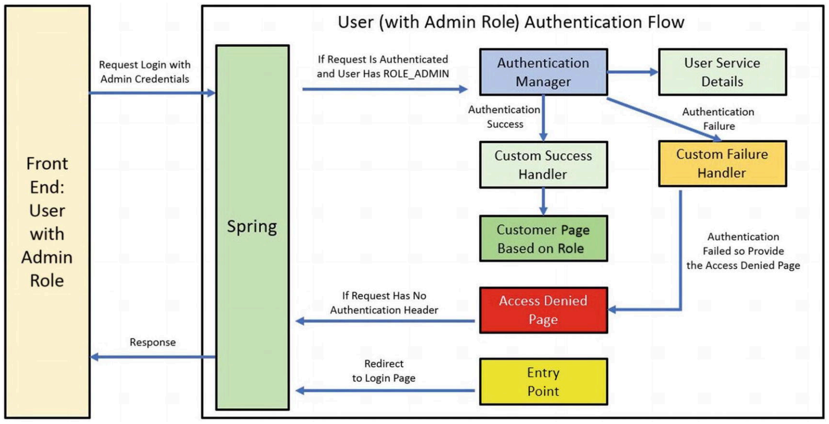

User (with Admin Role) Authentication Flow
Front End: User with Admin Role
Request Login with Admin Credentials
Spring
If Request Is Authenticated and User Has ROLE_ADMIN
Authentication Manager
User Service Details
Authentication Success
Authentication Failure
Custom Success Handler
Custom Failure Handler
Customer Page Based on Role
Authentication Failed so Provide the Access Denied Page
If Request Has No Authentication Header
Access Denied Page
Response
Redirect to Login Page
Entry Point

Figure 3-23 Spring Security user with admin role authentication request flow

Great! You have built your first Spring Security 6 web application.

The next chapter dives deeply into how this works internally by looking at the Spring Security architecture.

Summary

This chapter introduced all the tools needed to create the environment to develop Spring Security Java web applications. You learned how to install and configure all the tools needed for these examples, and you should have a good idea of what is needed to build a Spring Security 6 project. You learned how to build your first Java web application project without Spring Security, and then you added the security dependencies to update it as a Spring Security 6 application. The next chapter goes deeper into the Spring Framework architecture and design.