Pro Spring Boot 3

An Authoritative Guide with Best Practices

Third Edition

Felipe Gutierrez

Apress®

Pro Spring Boot 3

An Authoritative Guide with Best Practices

Third Edition

Felipe Gutierrez

Apress®

Pro Spring Boot 3: An Authoritative Guide with Best Practices, Third Edition

Felipe Gutierrez
4109 Rillcrest Grove Way Fuquay Varina, NC 27526-3562
Albuquerque, NM, USA

ISBN-13 (pbk): 978-1-4842-9293-8 ISBN-13 (electronic): 978-1-4842-9294-5
https://doi.org/10.1007/978-1-4842-9294-5

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Melissa Duffy
Development Editor: Jim Markham
Coordinating Editor: Gryffin Winkler

Cover image designed by kentaro-tachikawa on Unsplash (https://unsplash.com/)

Distributed to the book trade worldwide by Springer Science+Business Media LLC, 1 New York Plaza, Suite 4600, New York, NY 10004. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at https://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on the GitHub repository. For more detailed information, please visit https://www.apress.com/gp/services/source-code.

If disposing of this product, please recycle the paper

Dedicated in loving memory of my aunt, Fabiola Cerón,
and Simón Cruz, my uncle; I miss you so much!
Thanks uncle, for everything you taught me!

Table of Contents

About the Author

Felipe Gutierrez is a solutions software architect with bachelor's and master's degrees in computer science from Instituto Tecnologico y de Estudios Superiores de Monterrey Campus Ciudad de Mexico. Felipe has over 25 years of IT experience and has developed programs for companies in multiple vertical industries, such as government, retail, healthcare, education, and banking. He is currently working as Staff Engineer for VMware, specializing in content development for Tanzu Learning and the new Spring Academy learning site, Spring Framework, Spring Cloud Native Applications, Groovy, and RabbitMQ, among other technologies. He has also worked as a solutions architect for big companies like Nokia, Apple, Redbox, IBM, and Qualcomm. He is the author of *Spring Boot Messaging* (Apress, 2017) and *Spring Cloud Data Flow* (Apress, 2020).

About the Technical Reviewer

Manuel Jordan Elera is an autodidactic developer and researcher who enjoys learning new technologies for his own experiments and creating new integrations. Manuel won the Springy Award – Community Champion and Spring Champion 2013. In his little free time, he reads the Bible and composes music on his guitar. Manuel is known as dr_pompeii. He has tech-reviewed numerous books, including *Pro Spring MVC with WebFlux* (Apress, 2020), *Pro Spring Boot 2* (Apress, 2019), *Rapid Java Persistence and Microservices* (Apress, 2019), *Java Language Features* (Apress, 2018), *Spring Boot 2 Recipes* (Apress, 2018), and *Java APIs, Extensions and Libraries* (Apress, 2018). You can read his detailed tutorials on Spring technologies and contact him through his blog at `www.manueljordanelera.blogspot.com`. You can follow Manuel on his Twitter account, `@dr_pompeii`.

Acknowledgments

Thanks to the Spring team for creating such an amazing Java Framework!

Thanks to my technical reviewer, Manuel Jordan, for always excelling in his reviews! I also want to thank the Apress editorial team for their patience and excellent work.

Finally, I want to thank all of my family for their support and a special dedication to my loving aunt Fabiola and my super awesome uncle Simon (wife and husband), who passed away too soon! We miss you.

PART I

Introductions

CHAPTER 1

Spring Boot Quick Start

Welcome to the first chapter of the book, which will quickly immerse you in Spring Boot and demonstrate how easy it is to use by walking you through a simple project that exposes an API over the Web. If you are new to Spring Boot, this chapter will help you to rapidly familiarize yourself with the framework. If you are an experienced developer, feel free to quickly review the setup of the project (which will be referenced throughout this book) and move on to the next chapter.

Project: Users App

The project that we are going to build, named Users App, will expose a simple CRUD (create, read, update, and delete) API over the Web. These are the requirements for the Users App project:

- A user must have a name and an email address.
- A map is used to hold the information, using the email address as the key.
- It exposes an API that uses CRUD over the Web.

Initial Setup

To start with Spring Boot, you need to have the following installed:

- *Java*: You can install, for example, OpenJDK (`https://jdk.java.net/archive/`) or Eclipse Temurin (`https://adoptium.net/temurin/releases/`).
    - If you are Unix user, you can use SDKMAN! (`https://sdkman.io/`), which works for Linux and macOS.

© Felipe Gutierrez 2024
F. Gutierrez, *Pro Spring Boot 3*, https://doi.org/10.1007/978-1-4842-9294-5_1

  - If you are Windows user, you can use Chocolatey (https://chocolatey.org/).

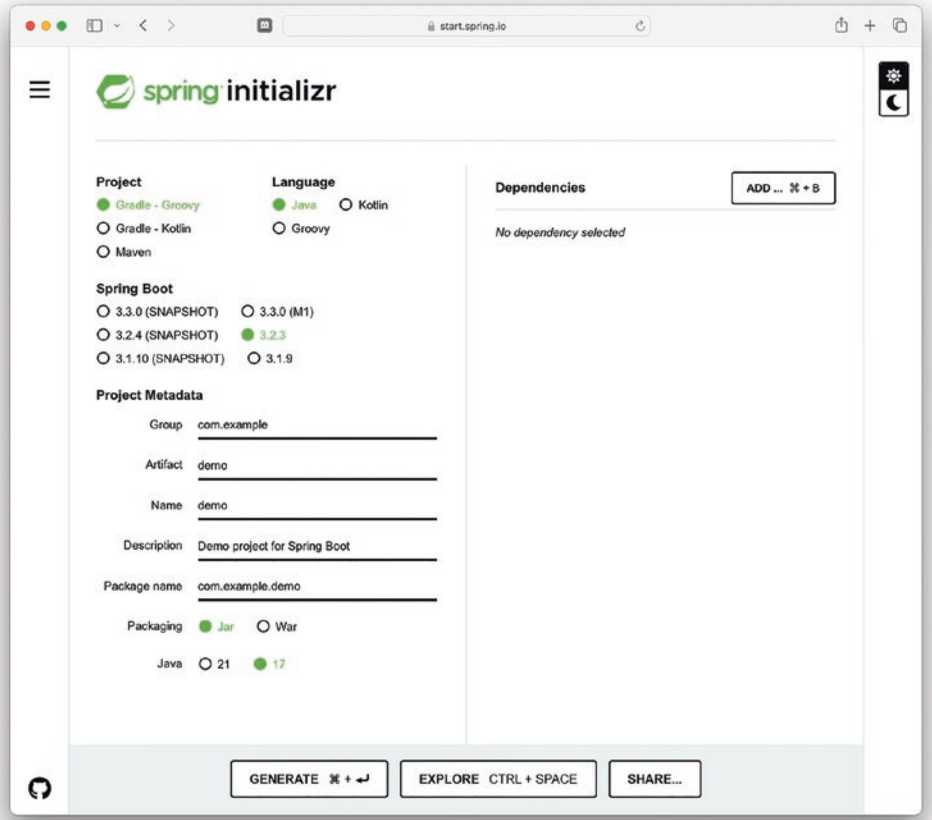
- *An integrated development environment (IDE)*: As a suggestion, you can use Microsoft Visual Studio Code (https://code.visualstudio.com/download), the Community edition of IntelliJ IDEA from JetBrains (https://www.jetbrains.com/idea/download/), or Spring Tools (https://spring.io/tools).
- *The* `curl` *or* `http` *command*: For `http`, you can install HTTPie (https://httpie.io/). Both commands are demonstrated later in this chapter.
- *The* `jq` *command*: You can install it using the instructions at https://stedolan.github.io/jq/.

Start @ start.spring.io

start.spring.io is the official web-based tool for generating Spring Boot projects. It provides a user-friendly interface to quickly set up a new Spring Boot project with your desired dependencies and configurations. Key features and benefits:

- **Streamlined project creation**: Eliminates the need to manually configure a project structure and dependencies.
- **Curated dependencies**: Offers a selection of common libraries and frameworks to easily add to your project.
- **Customization**: Allows you to choose the build tool (Maven or Gradle), language (Java, Kotlin, Groovy), and Spring Boot version.
- **Downloadable project**: Generates a ZIP file containing the configured project ready to be imported into your IDE.
- **Spring Boot integration**: Leverages Spring Boot's auto-configuration and convention-over-configuration principles for rapid development.

Open a browser and go to https://start.spring.io . You should see the home page of the Spring Initializr, as shown in Figure 1-1.



start.spring.io

spring initializr

Project
[✓ Gradle - Groovy]
Gradle - Kotlin
Maven

Language
[✓ Java] Kotlin
Groovy

Spring Boot
3.3.0 (SNAPSHOT) 3.3.0 (M1)
3.2.4 (SNAPSHOT) [✓ 3.2.3]
3.1.10 (SNAPSHOT) 3.1.9

Project Metadata

Group	com.example
Artifact	demo
Name	demo
Description	Demo project for Spring Boot
Package name	com.example.demo
Packaging	[✓ Jar] War
Java	21 [✓ 17]

Dependencies

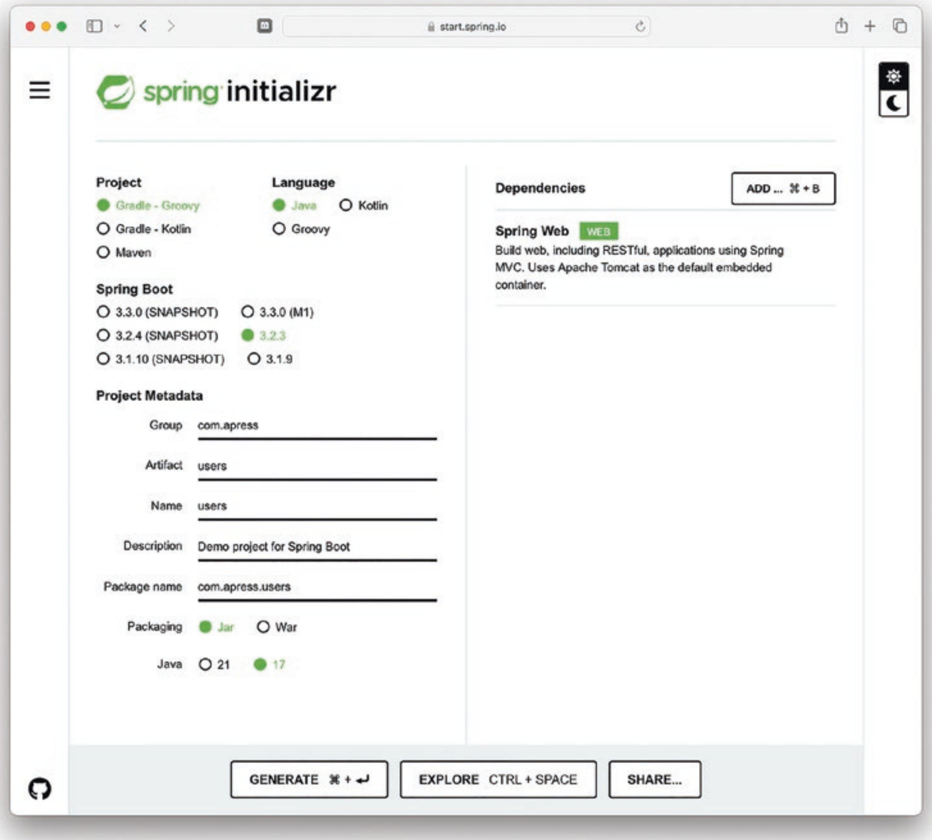
No dependency selected

Figure 1-1. *Spring Initializr home page*

Notice that by default the Spring Initializr uses Gradle – Groovy as the project builder, Java as the programming language, JAR for packaging, Java 17, and Spring Boot 3 (at the time of this writing, I'm using Spring Boot 3.2.3).

Modify the Project Metadata section with the following values, as shown in Figure 1-2 (The value of the Package name field will change automatically based on the values of the Group and Artifact fields).

- Group: `com.apress`
- Artifact: `users`
- Name: `users`
- Dependencies: Spring Web (click Add to find it)



Project
[✓ Gradle - Groovy]
Gradle - Kotlin
Maven

Language
[✓ Java] Kotlin
Groovy

Spring Boot
3.3.0 (SNAPSHOT) 3.3.0 (M1)
3.2.4 (SNAPSHOT) [✓ 3.2.3]
3.1.10 (SNAPSHOT) 3.1.9

Project Metadata

Group	com.apress
Artifact	users
Name	users
Description	Demo project for Spring Boot
Package name	com.apress.users
Packaging	[✓ Jar] War
Java	21 [✓ 17]

Dependencies

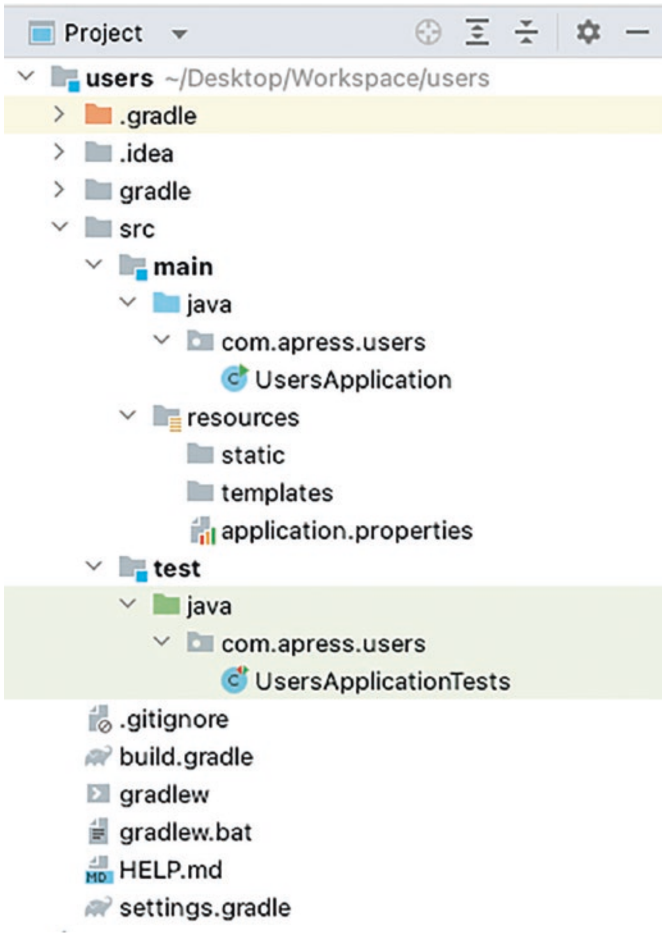
Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Figure 1-2. *Spring Initializr: Users App project*

Figure 1-2 shows all the necessary information to create the Users App project. Click the Generate button to zip the project and save it to your computer. Then, unzip it and import it to your favorite IDE. (I am using IntelliJ IDEA Community Edition, so that is what you will see in the screenshots in this book.)

Note You can download or fork the source code from the Apress GitHub site.

When you open the project in your IDE, you should see the project structure shown in Figure 1-3.



```
Project
users ~/Desktop/Workspace/users
  .gradle
  .idea
  gradle
  src
    main
      java
        com.apress.users
          UsersApplication
      resources
        static
        templates
        application.properties
    test
      java
        com.apress.users
          UsersApplicationTests
  .gitignore
  build.gradle
  gradlew
  gradlew.bat
  HELP.md
  settings.gradle
```

Figure 1-3. *Users App project structure*

Figure 1-3 shows the following three folders in the project structure:

- `src/main/java`: This folder contains all the source code. By default, the Spring Initializr creates the `UsersApplication.java` file. This has the main entry point where the application will start.
- `src/main/resources`: This folder contains one of the most important files, `application.properties`, which is used to modify configuration. We will use this file repeatedly throughout the book; for now, just know that it's located in this folder. This folder also

contains subfolders that normally hold assets such as HTML pages, images, JavaScript, etc. (more details are provided in upcoming chapters).

- `src/test/java`: This folder contains everything related to the unit and integration testing that you can perform to ensure that your project does what you expect it to do. By default, the Spring Initializr creates the `UsersApplicationTests.java` file.

Let's take a look at some of the files generated by the Spring Initializr, starting with the `build.gradle` file, shown in Listing 1-1.

Listing 1-1. build.gradle - File Generated by Spring Initializr

```
plugins {
    id 'java'
    id 'org.springframework.boot' version '3.2.3'
    id 'io.spring.dependency-management' version '1.1.4'
}

group = 'com.apress'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '17'

repositories {
    mavenCentral()
    maven { url 'https://repo.spring.io/milestone' }
    maven { url 'https://repo.spring.io/snapshot' }
}

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

tasks.named('test') {
    useJUnitPlatform()
}
```

The build.gradle file is important because it contains all the details about our project, some attributes, and the dependencies that will be used to generate everything we need, from compiling our code to creating an executable JAR for the Java virtual machine (JVM). This file first declares the plugins we are going to use and the dependencies repository (in this case, Maven Central, the milestone and snapshot). It then declares the dependencies (in this case, spring-boot-starter-web and spring-boot-starter-test; one for all related to a web application and the unit/integration tests, respectively). Don't worry about these dependencies for now; we are going to discuss them in detail in the following chapters.

The build.gradle file also comes with a few wrappers (gradlew for Unix users and gradlew.bat for Windows users). These wrappers will bring the Gradle engine without any prior installation of such builder. We will be using these files when running our application.

Before we continue to review the files generated by the Spring Initializr, modify the build.gradle file as shown in Listing 1-2 and described next.

Listing 1-2. build.gradle Modified

```
//...
dependencies {
    // ... previous dependencies
    implementation 'org.webjars:bootstrap:5.2.3'
}

//...

test {
    testLogging {
        events "passed", "skipped", "failed"
        showExceptions true
        exceptionFormat "full"
        showCauses true
        showStackTraces true
        showStandardStreams = false
    }
}
```

First, add the bootstrap dependency that will help us to create a nice style for a home page; we are going to use bootstrap (https://getbootstrap.com/). At the time of writing, the version is 5.2.3, but you should choose the latest version in the maven repository. Then, add the test section, which tells Gradle to show the keywords passed, skipped, or failed when running a test. If you set showStandardStreams to true, you can use System.out.println statements in the tests.

Next, open the UsersApplication.java file generated by the Spring Initializr and view the contents, shown in Listing 1-3.

Listing 1-3. src/main/java/com/apress/users/UsersApplication.java

```java
package com.apress.users;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class UsersApplication {

    public static void main(String[] args) {
        SpringApplication.run(UsersApplication.class, args);
    }

}
```

UsersApplication.java is the main file for this project because it contains the main(String[] args) method that is necessary to run any Java application. As shown in Listing 1-3, it uses the @SpringBootApplication annotation and uses the static SpringApplication class that invokes the run method that accepts two parameters, one if the configuration class, and the arguments that we can pass at time of executing as parameters.

Next, add a new class that will hold the information of our user. Create the User.java file in the src/main/java/com/apress/users folder with the content shown in Listing 1-4.

Listing 1-4. src/main/java/com/apress/users/User.java

```
package com.apress.users;

public class User {
    private String email;
    private String name;

    public User() {
    }

    public User(String email, String name) {
        this.email = email;
        this.name = name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

As Listing 1-4 shows, the `User.java` class has two fields/properties, `Email` and `Name`. Notice that we have some constructors and the getters and setters for these fields/properties.

Next, let's add another class that will expose our API and manage the CRUD for our users. In the same folder, `src/main/java/com/apress/users`, create the `UsersController.java` file with the content shown in Listing 1-5.

Listing 1-5. src/main/java/com/apress/users/UsersController.java

```
package com.apress.users;

import org.springframework.web.bind.annotation.*;

import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

@RestController
@RequestMapping("/users")
public class UsersController {
    private Map<String,User> users = new HashMap() {{
        put("ximena@email.com",new User("ximena@email.com","Ximena"));
        put("norma@email.com",new User("norma@email.com","Norma"));

    }};

    @GetMapping
    public Collection<User> getAll(){
        return this.users.values();
    }

    @GetMapping("/{email}")
    public User findUserByEmail(@PathVariable String email){
        return this.users.get(email);
    }

    @PostMapping
    public User save(@RequestBody User user){
        this.users.put(user.getEmail(),user);
        return user;
    }

    @DeleteMapping("/{email}")
    public void save(@PathVariable String email){
        this.users.remove(email);
    }
}
```

The `UsersController.java` class will expose the API. This class is marked with the annotation `@RestController`, telling Spring Boot that this class is responsible for accepting any incoming requests. It's also marked with `@RequestMapping("/users")`, which tells Spring Boot that it will have a based `/users` endpoint for every request made with any of the HTTP methods (`GET`, `POST`, `PUT`, `DELETE`, `PATCH`, etc.). This class also has some methods that are marked with special annotation such as `@GetMapping`, `@PostMapping`, and `@DeleteMapping`, which are used to tell Spring Boot that those methods will be executed once a request is made.

This class also has a `java.util.Map` interface that is being initialized with some users. That will be covering one of the requirements.

Now, let's take a look at the following endpoints:

- `/users`: This will handle the read part of CRUD. The method that will be executed is `getAll()`. This method is executed when an HTTP `GET` request is made. This endpoint is also used when an HTTP `POST` request is made (can be taken as the create and update parts of CRUD); the `save(@RequestBody User user)` method will be executed and it will add or update a user based on the user's email address. Note that the argument of this method has an annotation, `@RequestBody`, meaning that for every `POST` request, Spring Boot will convert the data sent over into the class type, in this case the `User` class. The data sent is in JSON format by default, unless you modify the HTTP Header `Content-Type` and inform Spring Boot of that change.
- `/users/{email}`: This will also be our read, but it will look for an email in the `Map` instance. The method executed will be `findUserByEmail(@PathVariable String email)`, which will return the user found. This method also is called when an HTTP `GET` request is made. This endpoint is also used by the `deleteByEmail (@PathVariable String email)` method when an HTTP `DELETE` request is made. This will remove the user from the map. This method contains the `@PathVariable` annotation that will translate the path that matches the name, in this case `{email}` match, with the `String email` parameter. This means that when there is a request,

either the GET or DELETE such as /users/ximena@email.com for example, it will execute any of this methods (depending on the HTTP method request) and it will assign ximena@email.com to the email variable.

Next, let's create a landing page that will open when we run this project. For this purpose, add an index.html file in the src/main/resources/static folder with the content shown in Listing 1-6.

Listing 1-6. src/main/resources/static/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <link rel="stylesheet" type="text/css"
         href="webjars/bootstrap/5.2.3/css/bootstrap.min.css">
    <title>Welcome - Users App</title>
</head>
<body class="d-flex h-100 text-center">

<div class="cover-container d-flex w-100 h-100 p-3 mx-auto flex-column">
    <header class="mb-auto">
        <div>
            <h3 class="float-md-start mb-0">Users</h3>
        </div>
    </header>

    <main class="px-3">
        <h1>Simple Users Rest Application</h1>
        <p class="lead">This is a simple Users app where you can access any
        information from a user</p>
        <p class="lead">
            <a href="/users">Get All Users</a>
        </p>
    </main>
    <footer class="mt-auto text-black-50">
        <p>Powered by Spring Boot 3</p>
    </footer>
</div>
</body>
</html>
```

As you can see, we are using `bootstrap` (`https://getbootstrap.com/`) to style our home page; we are passing the path for the CSS that begins with `webjars`. There is a convention to this, but we are going to talk about it later. For now, you can take this as a recipe. If you want to use any `webjars` tech, such as `jQuery` or any other, you must provide the path starting with `webjars`.

So, you now have in place for the Users App project everything that you need to run a web API using Spring Boot. Next, we'll look at how to test the application to make sure that it performs as expected.

Testing the Users App Project

Let's test our code using the Testing framework that Spring Boot provides. In the `src/test/java` folder structure, open the `UserApplicationTests.java` file. Listing 1-7 shows its content.

Listing 1-7. src/test/java/com/apress/users/UsersApplicationTests.java

```
package com.apress.users;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class UsersApplicationTests {

    @Test
    void contextLoads() {
    }

}
```

Note in Listing 1-7 that `UserApplicationTests.java` uses the `@SpringBootTest` annotation, which prepares your environment for executing any unit or integration testing; for every test that you want to conduct, you only need to use the `@Test` annotation for the method that you want to use to execute that test.

Next, replace all the code in `UserApplicationTests.java` with the content shown in Listing 1-8.

Listing 1-8. Modified src/test/java/com/apress/users/UsersApplicationTests.java

```java
package com.apress.users;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.web.client.TestRestTemplate;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.http.ResponseEntity;

import java.util.Collection;

import static org.assertj.core.api.Assertions.assertThat;

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
public class UsersApplicationTests {

    @Value("${local.server.port}")
    private int port;

    private final String BASE_URL = "http://localhost:";
    private final String USERS_PATH = "/users";

    @Autowired
    private TestRestTemplate restTemplate;

    @Test
    public void indexPageShouldReturnHeaderOneContent() throws Exception {
        assertThat(this.restTemplate.getForObject(BASE_URL + port,
                String.class)).contains("Simple Users Rest Application");
    }
    @Test
    public void usersEndPointShouldReturnCollectionWithTwoUsers() throws
    Exception {
        Collection<User> response = this.restTemplate.
                getForObject(BASE_URL + port + USERS_PATH,
                Collection.class);

        assertThat(response.size()).isEqualTo(2);
    }

    @Test
    public void userEndPointPostNewUserShouldReturnUser() throws
Exception {
        User user =  new User("dummy@email.com","Dummy");
        User response =  this.restTemplate.postForObject(BASE_URL + port +
        USERS_PATH,user,User.class);

        assertThat(response).isNotNull();
        assertThat(response.getEmail()).isEqualTo(user.getEmail());

        Collection<User> users = this.restTemplate.
                getForObject(BASE_URL + port + USERS_PATH,
                Collection.class);

        assertThat(users.size()).isGreaterThanOrEqualTo(2);

    }


    @Test
    public void userEndPointDeleteUserShouldReturnVoid() throws Exception {
        this.restTemplate.delete(BASE_URL + port + USERS_PATH + "/norma@
        email.com");

        Collection<User> users = this.restTemplate.
                getForObject(BASE_URL + port + USERS_PATH,
                Collection.class);

        assertThat(users.size()).isLessThanOrEqualTo(2);
    }
```