

# Full Stack Development

## Spring MVC Lab

### Creating a Webservice

#### 1. Lab objectives

In this lab, you will use Spring MVC to implement a greeting Restful web service that will accept HTTP GET requests at: `http://localhost:8080/greeting`.

The service returns a web page that contains the greeting "Hello, World!" The request can include an optional name parameter in the query string like this `http://localhost:8080/greeting?name=User`. The name parameter value overrides the default value of World,

#### 2. Create the Spring Boot project

Go to <https://start.spring.io>.

You will be using the Spring Web starter. All you need is the Spring Web dependency.

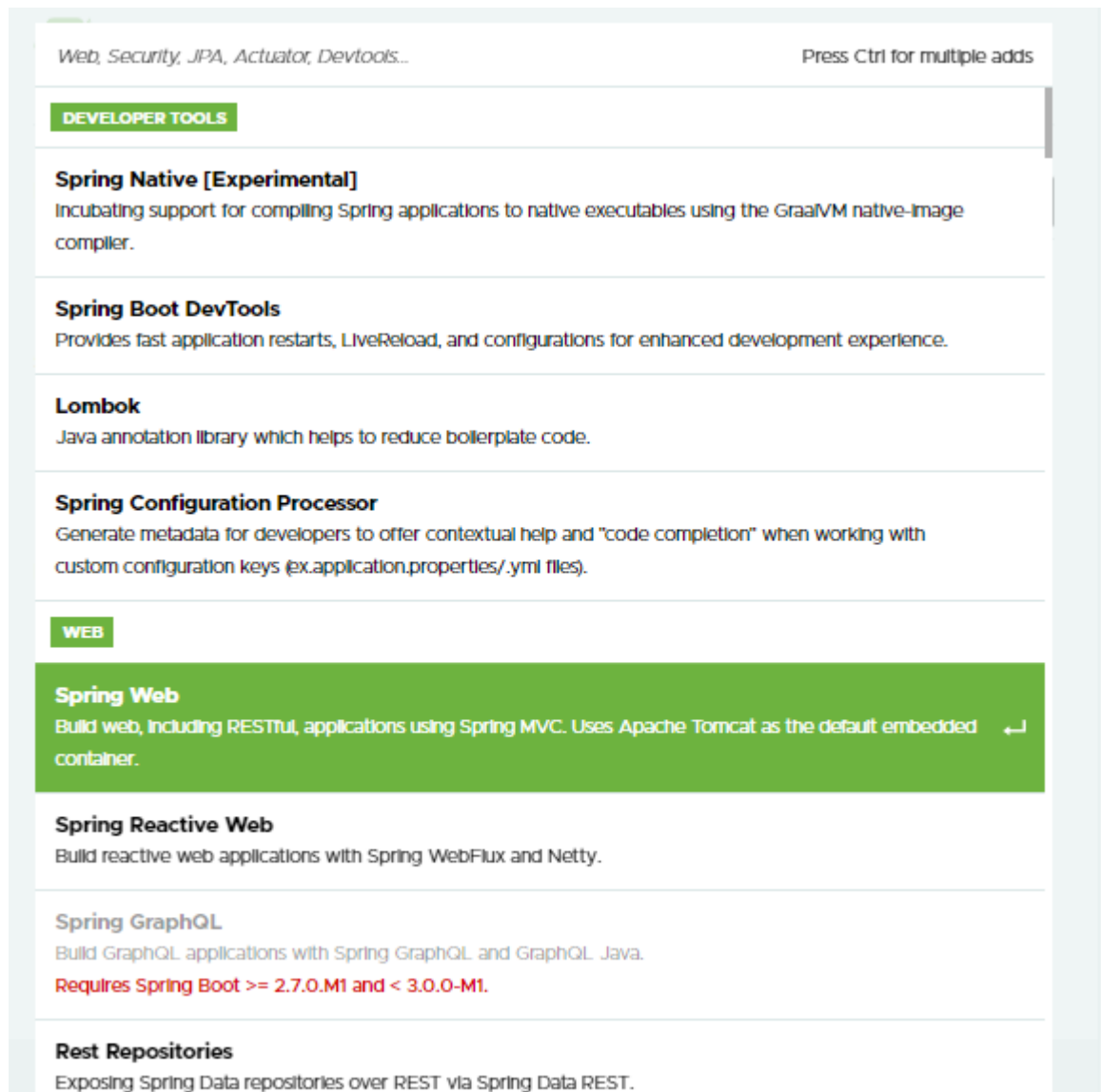
The screenshot shows the Spring Boot project generator interface. It is divided into several sections: Project, Language, Spring Boot, Project Metadata, Dependencies, and a bottom bar with action buttons.

- Project:** ☒ Maven Project, ☐ Gradle Project
- Language:** ☒ Java, ☐ Kotlin, ☐ Groovy
- Spring Boot:** ☐ 3.0.0 (SNAPSHOT), ☐ 2.7.0 (SNAPSHOT), ☐ 2.6.5 (SNAPSHOT), ☐ 2.5.11 (SNAPSHOT), ☒ 3.0.0 (M1), ☐ 2.7.0 (M2), ☒ 2.6.4, ☐ 2.5.10
- Project Metadata:**
  - Group:
  - Artifact:
  - Name:
  - Description:
  - Package name:
- Packaging:** ☒ Jar, ☐ War
- Java:** ☐ 17, ☒ 11, ☐ 8
- Dependencies:** 
  - Spring Web** ☒ WEB: Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
- Bottom Bar:**

The defaults “Java” and “Maven” should be selected as well as the highest numbered version that does *not* have (SNAPSHOT) or (M\*) following. What you see may not be exactly what the screen shot shows.

Ensure that you have selected the “Jar” packaging and the highest Java version that is less than yours (i.e. if you are running Java 12, then select Java 11.)

Select the dependencies box and then choose the Spring Web starter from the dropdown list



Click on “Generate” and download the resulting zip file

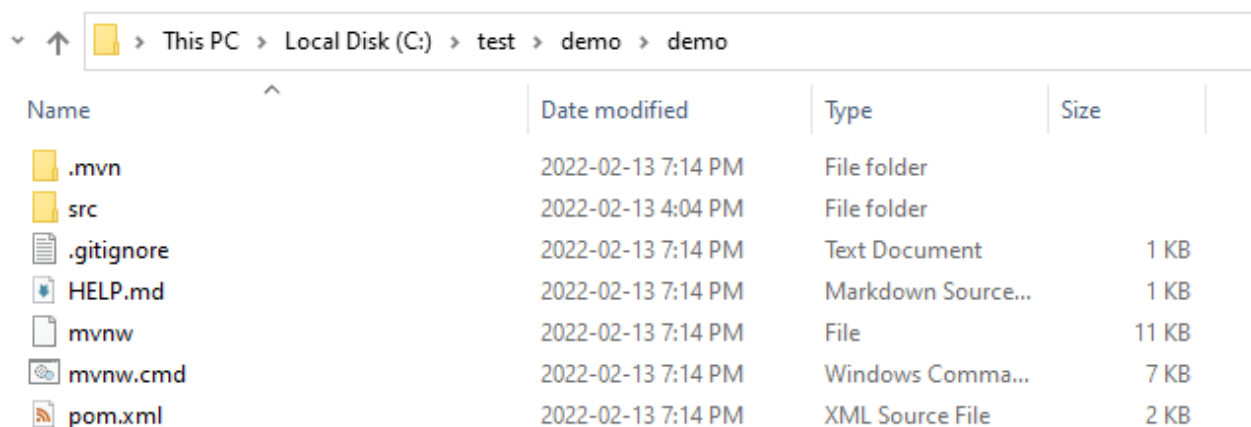
### 3. Importing the project

In this step, you will take the generated Maven project and import it into Eclipse to build the application.

Remember, Spring Boot just did all the configuration to create a Web App, but we still have to actually write the App code.

Move the zip file to a working directory (whatever you want) and unzip the archive

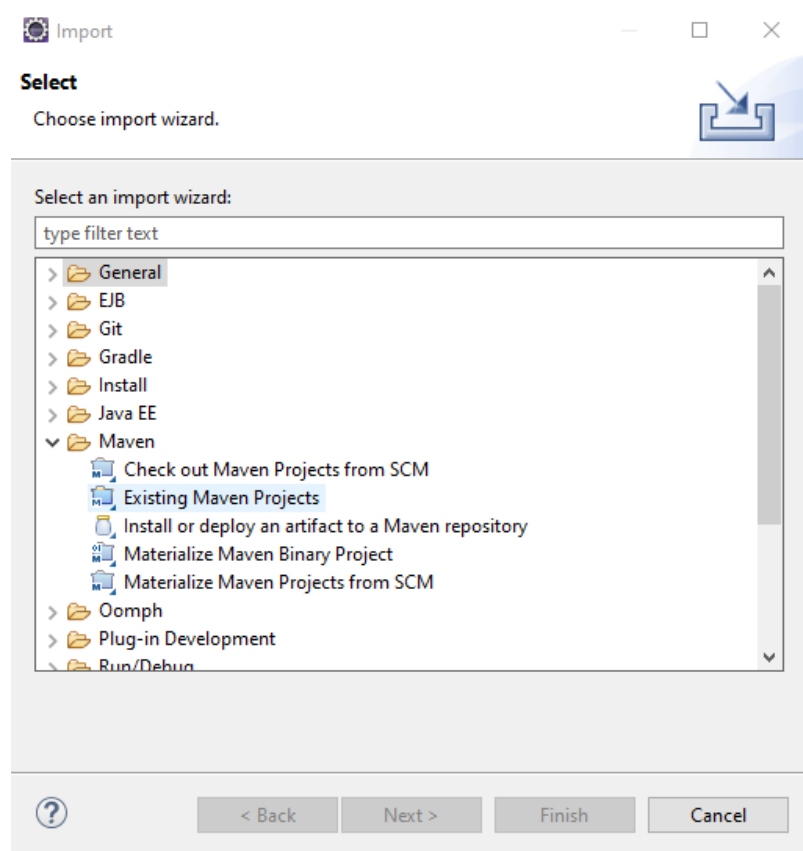
The resulting Maven project should



The screenshot shows a Windows File Explorer window with the address bar displaying the path: This PC > Local Disk (C:) > test > demo > demo. The main area shows a list of files and folders with columns for Name, Date modified, Type, and Size.

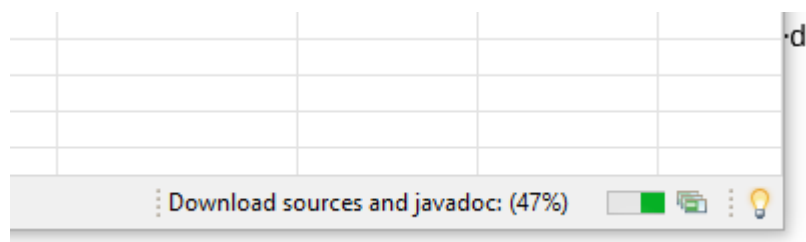
Name	Date modified	Type	Size
.mvn	2022-02-13 7:14 PM	File folder	
src	2022-02-13 4:04 PM	File folder	
.gitignore	2022-02-13 7:14 PM	Text Document	1 KB
HELP.md	2022-02-13 7:14 PM	Markdown Source...	1 KB
mvnw	2022-02-13 7:14 PM	File	11 KB
mvnw.cmd	2022-02-13 7:14 PM	Windows Comma...	7 KB
pom.xml	2022-02-13 7:14 PM	XML Source File	2 KB

In Eclipse, chose “Import” from the main File menu and select Maven Project and then “Import Existing Maven Project”.

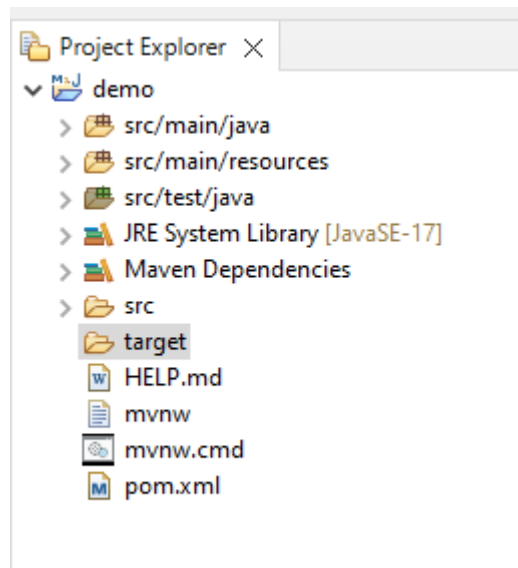


In the File selector, chose the directory of your downloaded project and then select finish.

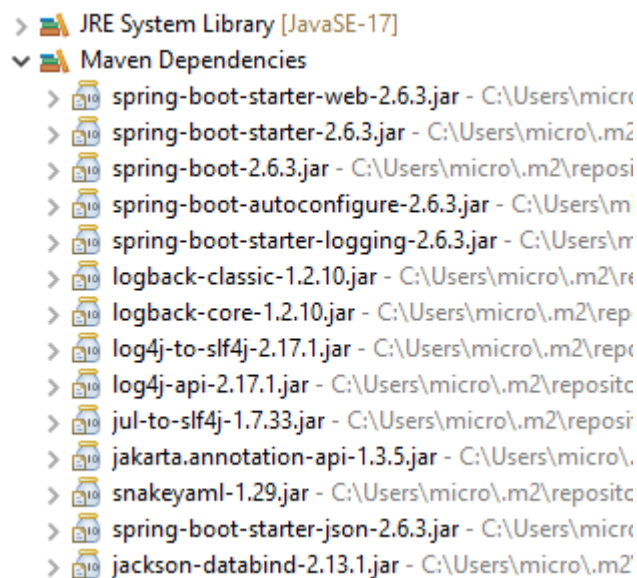
It will take some time to download all the Maven dependencies and build the project, but you can track the progress in the lower right hand corner:



Once Maven is finished, the project should look like this:



You can check in the Maven dependencies to see that a number of Spring Boot jars are now part of the project.



## 4. Testing the App

Remember that there is no actual code for the App to run, we still have to do some work ourselves.

Open the DemoApplication.java file.

```

1 package com.example.demo;
2
3 import org.springframework.boot.SpringApplication;
4
5
6 @SpringBootApplication
7 public class DemoApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(DemoApplication.class, args);
11     }
12
13 }

```

Just as in previous labs, run this as a Java application, do not run it on a server since Spring Boot has bundled a server as part of the Application.

You should see in the console the startup message

```

:: Spring Boot :: (v2.6.3)

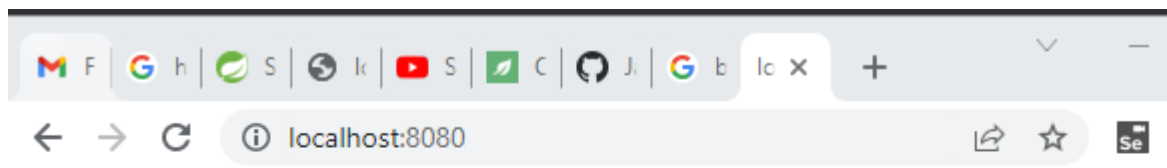
2022-02-13 16:27:16.296 INFO 29544 --- [main] com
2022-02-13 16:27:16.299 INFO 29544 --- [main] com

```

You should also be able to see (over to the right) messages about starting up the server and serving up the application on port 8080.

```
: Starting DemoApplication using Java 17.0.2 on DESKTOP-DVQUMGD with
: No active profile set, falling back to default profiles: default
: Tomcat initialized with port(s): 8080 (http)
: Starting service [Tomcat]
: Starting Servlet engine: [Apache Tomcat/9.0.56]
: Initializing Spring embedded WebApplicationContext
: : Root WebApplicationContext: initialization completed in 1277 ms
: Tomcat started on port(s): 8080 (http) with context path ''
: Started DemoApplication in 2.359 seconds (JVM running for 2.739)
```

Going to the webpage, we can see that the application is running but saying that there is no actual web content, including an error page.



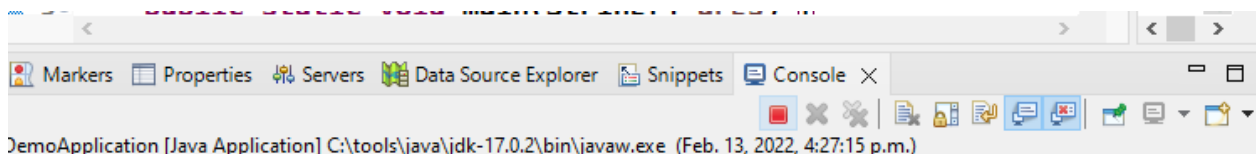
## Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sun Feb 13 16:32:35 EST 2022

There was an unexpected error (type=Not Found, status=404).

Shut down the application by pressing the red “terminate” button on the console toolbar



## 5. Create the Model Class

The resources that are being referenced by the web service are Greeting entities.

These are Pojos that need to know nothing about the controller or the views that are being used.

Notice that each resource created will have an id, a convention that is consistent with the REST approach

```
public class Greeting {
    private final long id;
    private final String content;

    public Greeting(long id, String content) {
        this.id = id;
        this.content = content;
    }

    public long getId() {
        return id;
    }

    public String getContent() {
        return content;
    }
}
```

## 6. Create the controller

1. In MVC, the controller handles incoming requests and routes them to the correct model objects to be managed.

```
@RestController
public class GreetingController {
    private static final String template = "Hello, %s!";
    private final AtomicLong counter = new AtomicLong();

    @GetMapping("/greeting")
    public Greeting greeting(@RequestParam(value = "name", defaultValue = "World") String name) {
        return new Greeting(counter.incrementAndGet(), String.format(template, name));
    }
}
```

Spring Web provides a number of annotations to define how a controller class is to be managed.



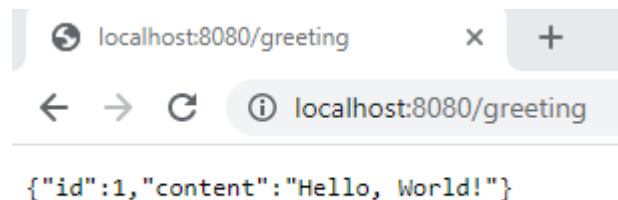
The `@RestController` annotation tells Spring that this class is the controller.

The view object is normally what is presented to the client. In this case the view is just a string so the template static variable is our model. In a more sophisticated application, we would use something like a templating engine like Thymeleaf to serve up HTML pages.

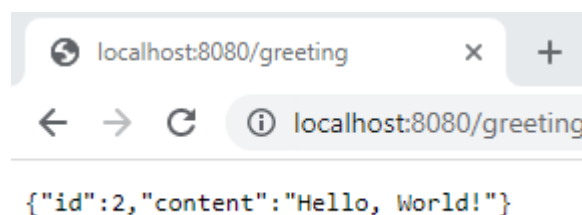
The `@GetMapping(URL)` maps specific URLs to processing objects. In this case the effect is to create a new model Greeting object and return it.

The `@RequestParam()` annotation looks for an argument in the URL and if there is one, assigns it to the string that appears as the value of the name parameter in the URL. If there is no parameter, it defaults to "World"

Since we are not using an HTML page, what we will see is just a JSON object representing the Greeting resource.



Reloading the page shows that a new request is generated and a new Greeting resource created in response.



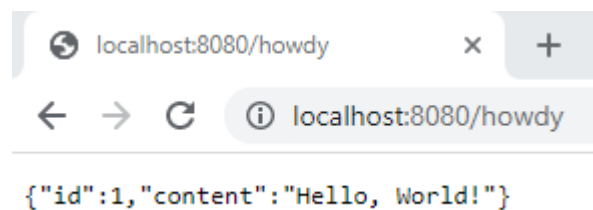
## 7. Change the URL

Shut down the server using the red square button

Replace the mapping to /greeting to anything else, like below where “/howdy” is being used

```
@GetMapping("/howdy")
public Greeting greeting(@RequestParam(required=false,defaultValue="World") String name) {
    return new Greeting("Hello, " + name + "!");
}
```

Restart the application and test it out



## End Lab

