Full Stack Development
Spring MVC Lab

Creating a Webservice

1. Lab objectives

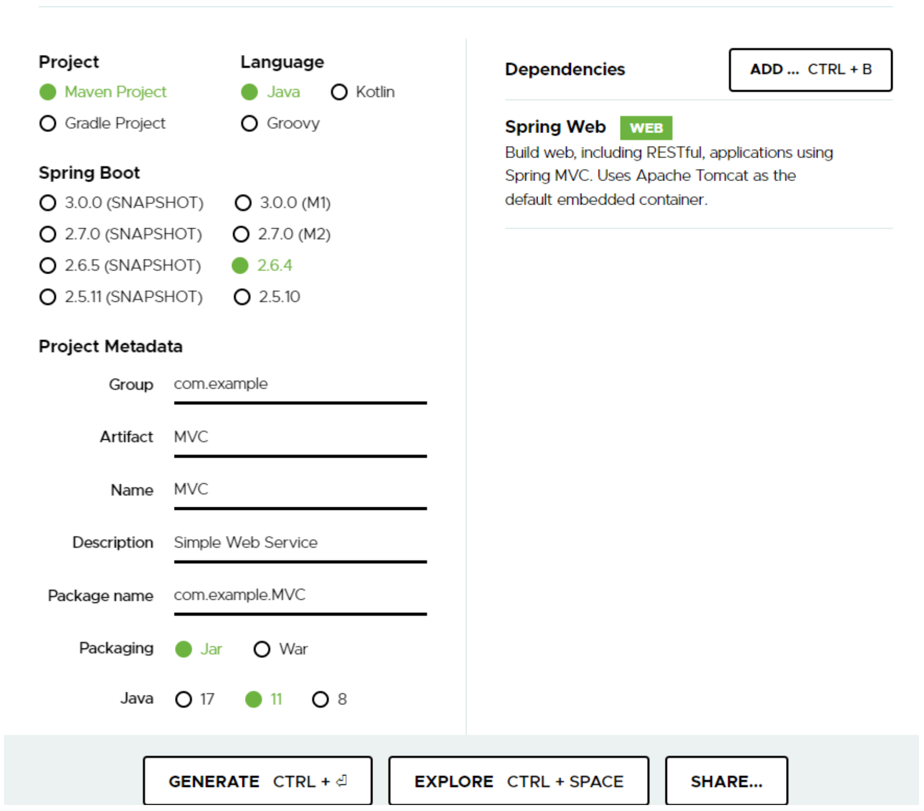
In this lab, you will use Spring MVC to implement a greeting Restful web service that will accept HTTP GET requests at: `http://localhost:8080/greeting`.

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.



Project
[✓ Maven Project] Gradle Project

Language
[✓ Java] Kotlin Groovy

Spring Boot
3.0.0 (SNAPSHOT) 3.0.0 (M1)
2.7.0 (SNAPSHOT) 2.7.0 (M2)
2.6.5 (SNAPSHOT) [✓ 2.6.4]
2.5.11 (SNAPSHOT) 2.5.10

Project Metadata

Group	com.example
Artifact	MVC
Name	MVC
Description	Simple Web Service
Package name	com.example.MVC
Packaging	[✓ Jar] War
Java	17 [✓ 11] 8

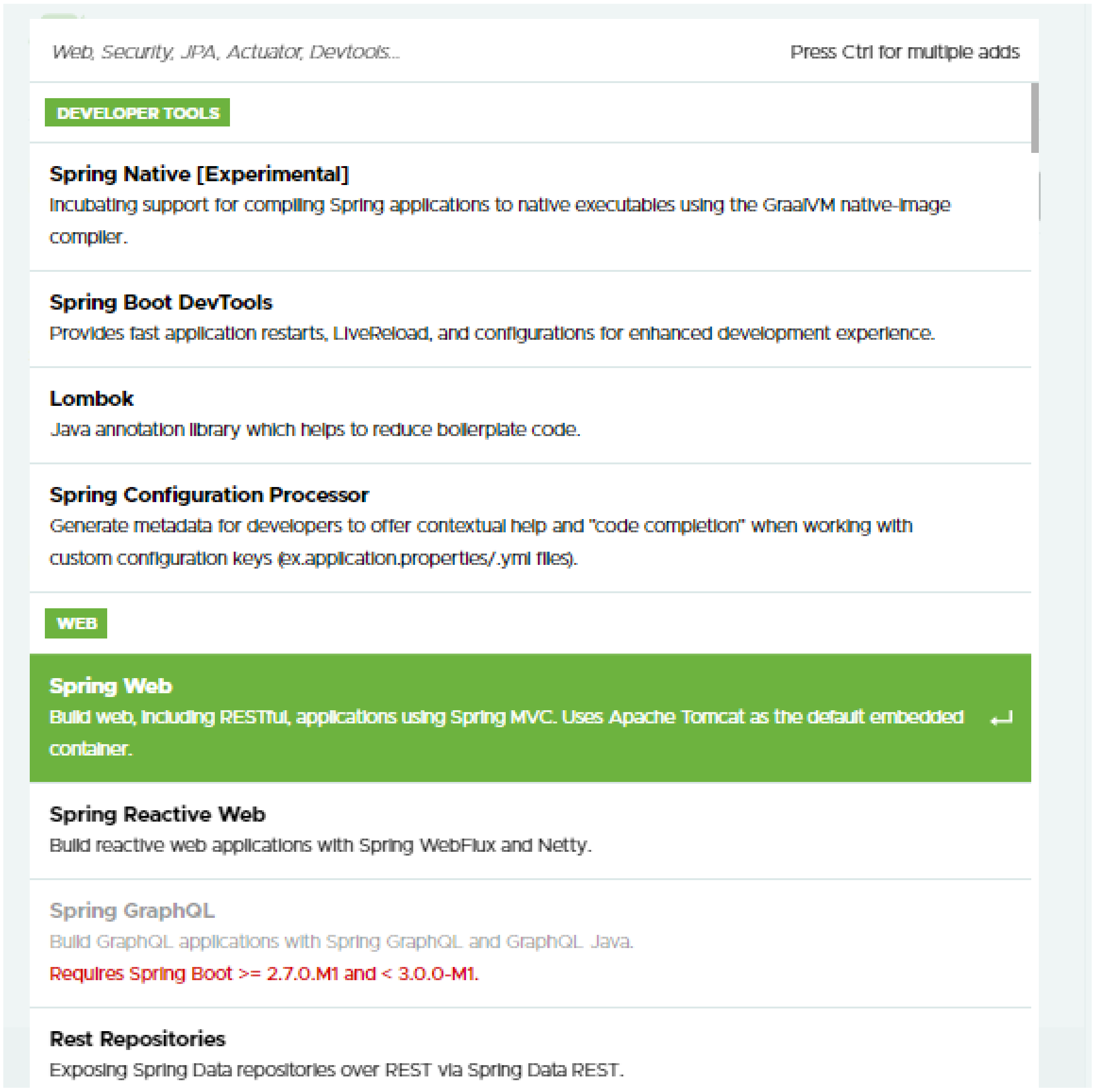
Dependencies

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

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


Web, Security, JPA, Actuator, Devtools...
Press Ctrl for multiple adds
DEVELOPER TOOLS

Spring Native [Experimental]
Incubating support for compiling Spring applications to native executables using the GraalVM native-image compiler.

Spring Boot DevTools
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Lombok
Java annotation library which helps to reduce boilerplate code.

Spring Configuration Processor
Generate metadata for developers to offer contextual help and "code completion" when working with custom configuration keys (ex.application.properties/.yml files).

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

Spring Reactive Web
Build reactive web applications with Spring WebFlux and Netty.

Spring GraphQL
Build GraphQL applications with Spring GraphQL and GraphQL Java.
Requires Spring Boot >= 2.7.0.M1 and < 3.0.0-M1.

Rest Repositories
Exposing Spring Data repositories over REST via Spring Data REST.

Click on “Generate” and download the resulting zip file

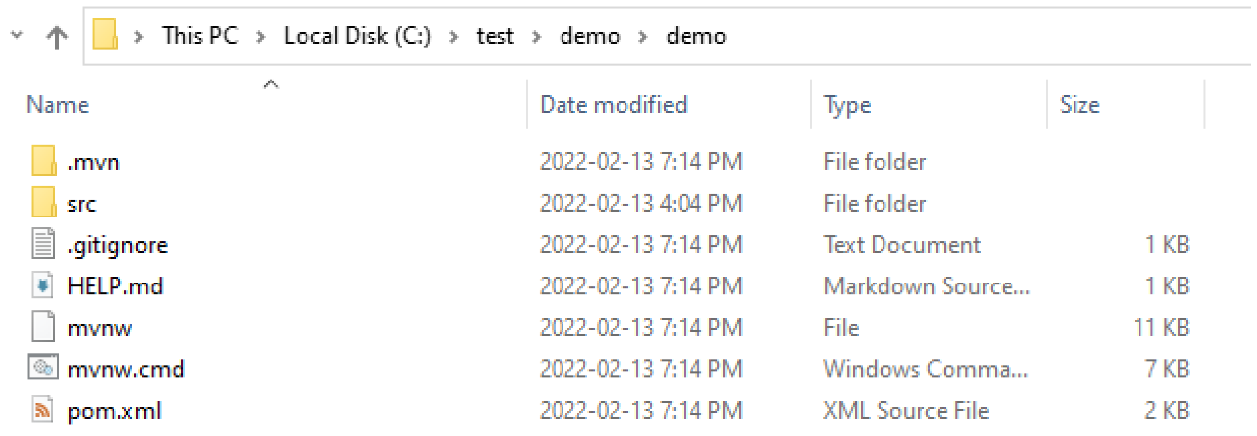
3. Importing the project

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.

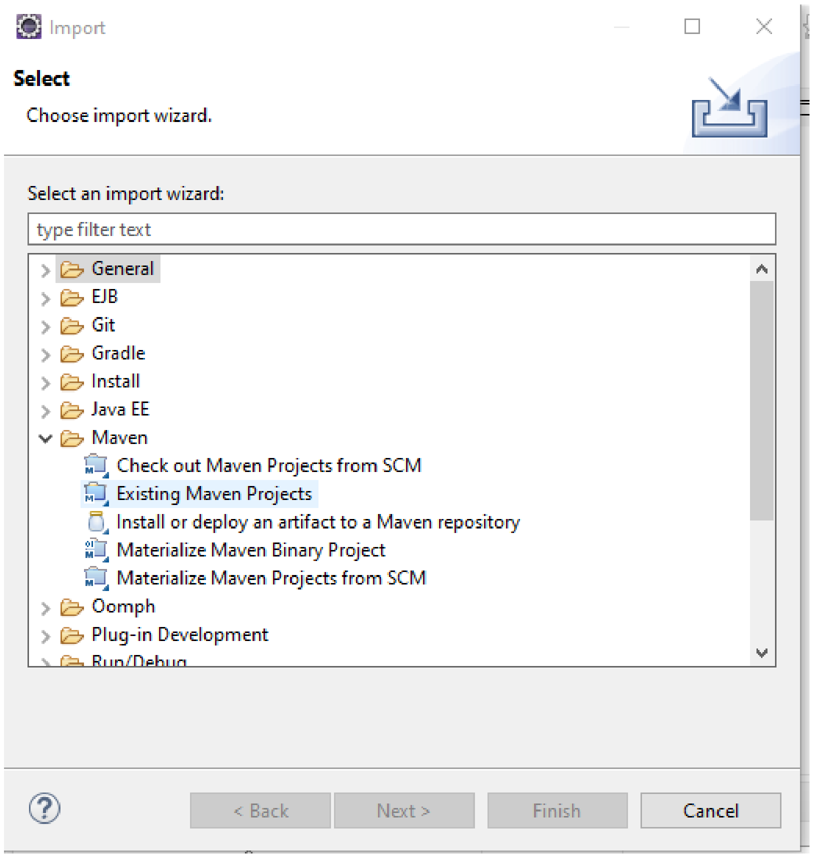
Move the zip file to a working directory (whatever you want) and unzip the archive

The resulting Maven project should



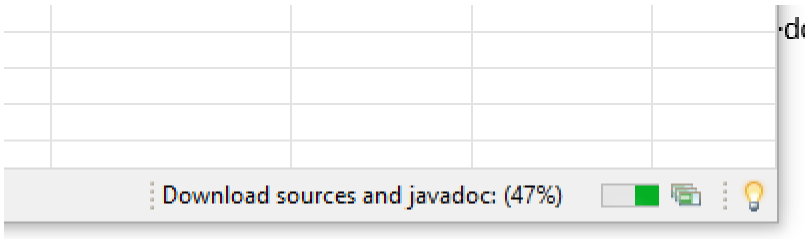
Name	Date modified	Type	Size
.mvn	2022-02-13 7:14 PM	File folder	
src	2022-02-13 4:04 PM	File folder	
.gitignore	2022-02-13 7:14 PM	Text Document	1 KB
HELP.md	2022-02-13 7:14 PM	Markdown Source...	1 KB
mvnw	2022-02-13 7:14 PM	File	11 KB
mvnw.cmd	2022-02-13 7:14 PM	Windows Comma...	7 KB
pom.xml	2022-02-13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

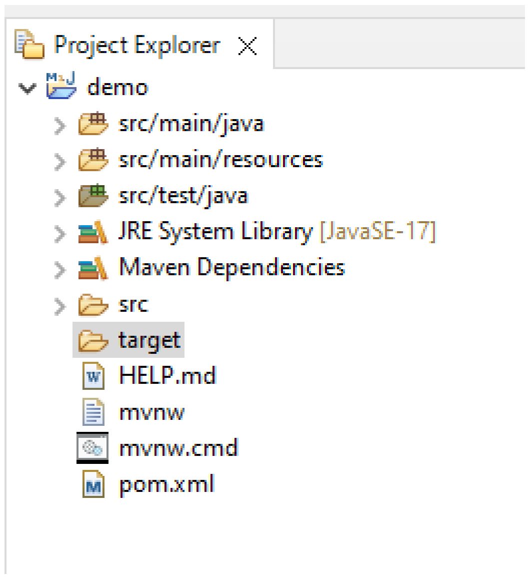


In the File selector, chose the directory of your downloaded project and then select finish.

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:

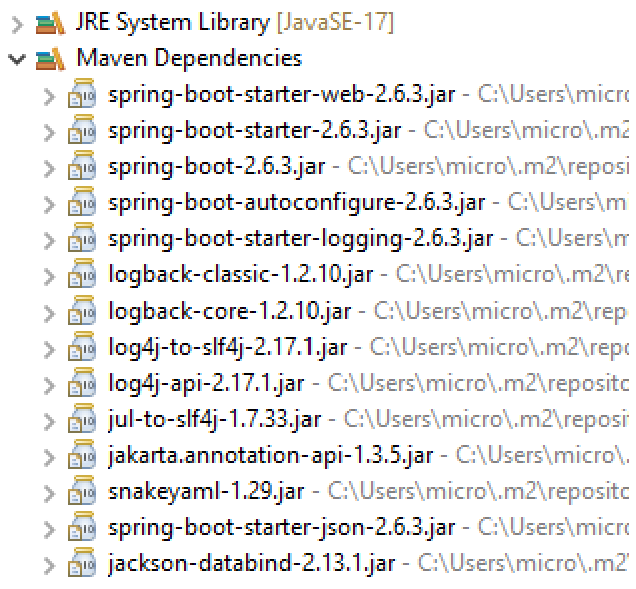


Once Maven is finished, the project should look like this:



```
Project Explorer
demo
  src/main/java
  src/main/resources
  src/test/java
  JRE System Library [JavaSE-17]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

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

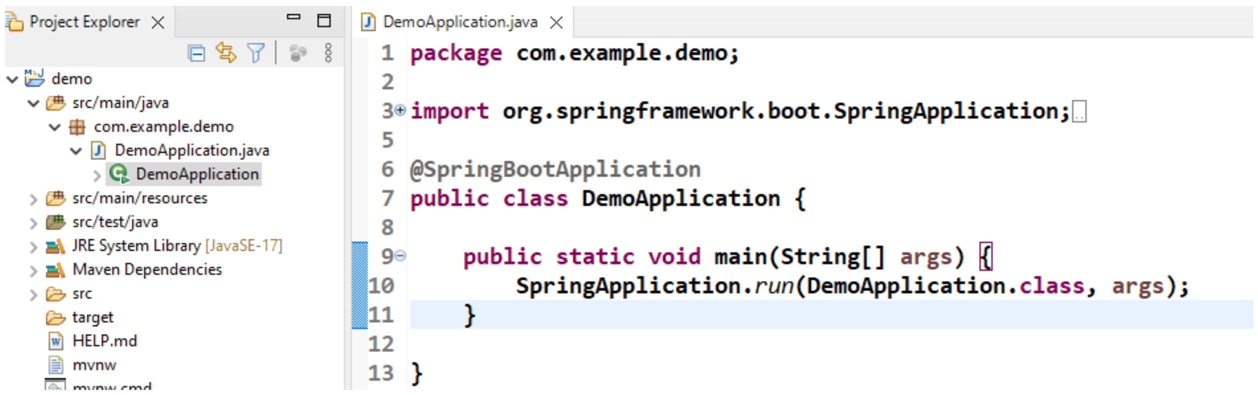


```
JRE System Library [JavaSE-17]
Maven Dependencies
  spring-boot-starter-web-2.6.3.jar - C:\Users\micr
  spring-boot-starter-2.6.3.jar - C:\Users\micro\.m2
  spring-boot-2.6.3.jar - C:\Users\micro\.m2\reposi
  spring-boot-autoconfigure-2.6.3.jar - C:\Users\m
  spring-boot-starter-logging-2.6.3.jar - C:\Users\m
  logback-classic-1.2.10.jar - C:\Users\micro\.m2\re
  logback-core-1.2.10.jar - C:\Users\micro\.m2\rep
  log4j-to-slf4j-2.17.1.jar - C:\Users\micro\.m2\rep
  log4j-api-2.17.1.jar - C:\Users\micro\.m2\reposit
  jul-to-slf4j-1.7.33.jar - C:\Users\micro\.m2\reposi
  jakarta.annotation-api-1.3.5.jar - C:\Users\micro\.
  snakeyaml-1.29.jar - C:\Users\micro\.m2\reposit
  spring-boot-starter-json-2.6.3.jar - C:\Users\micr
  jackson-databind-2.13.1.jar - C:\Users\micro\.m2
```

4. Testing the App

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

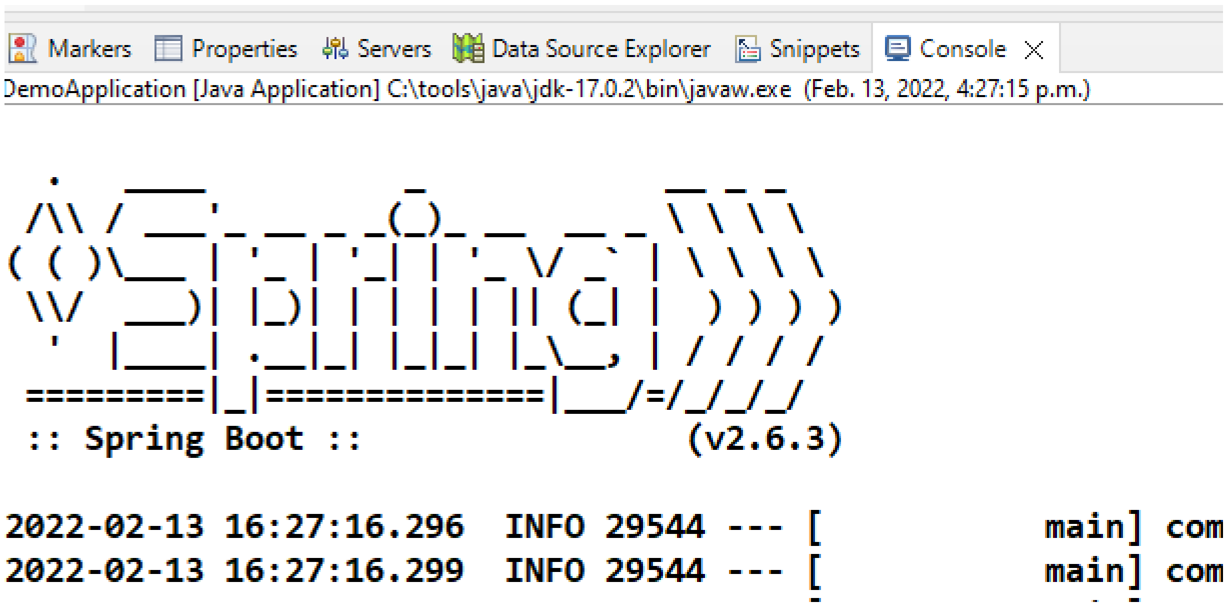
Open the DemoApplication.java file.



```java
package com.example.demo;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class DemoApplication {

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

}
```

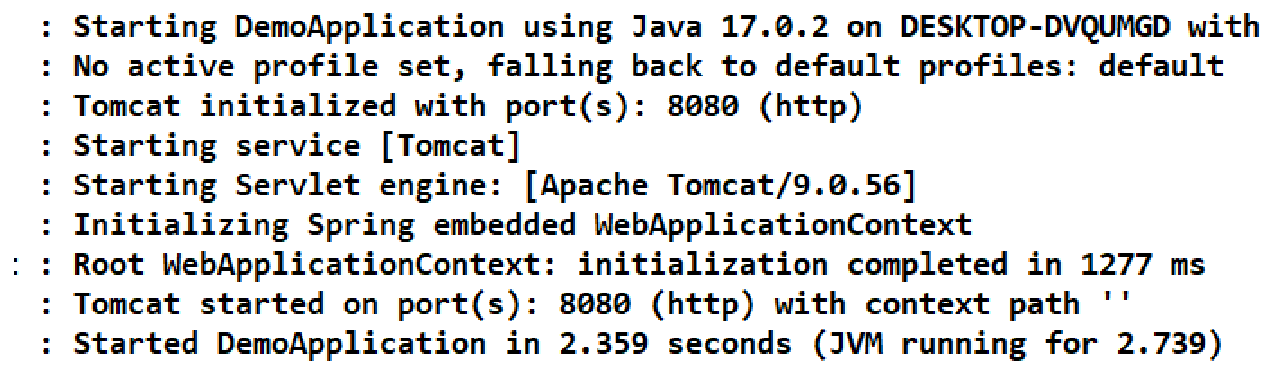
Just as in previous labs, run this as a Java application, do not run it on a server since Spring Boot has bundled a server as part of the Application.

You should see in the console the startup message



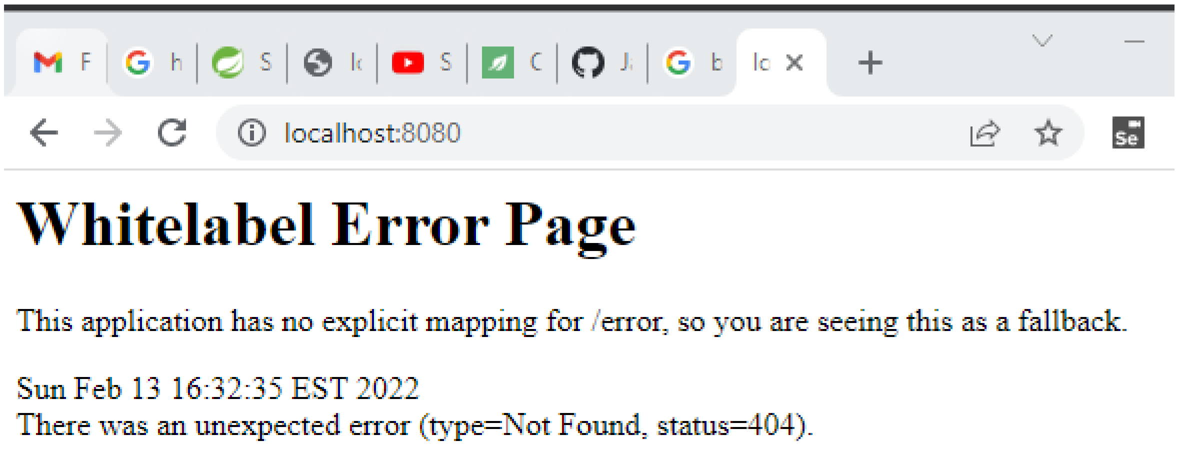
```
DemoApplication [Java Application] C:\tools\java\jdk-17.0.2\bin\javaw.exe  (Feb. 13, 2022, 4:27:15 p.m.)

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: 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.

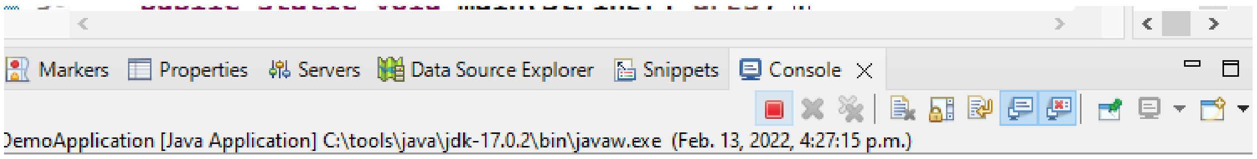


Whitelabel Error Page

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



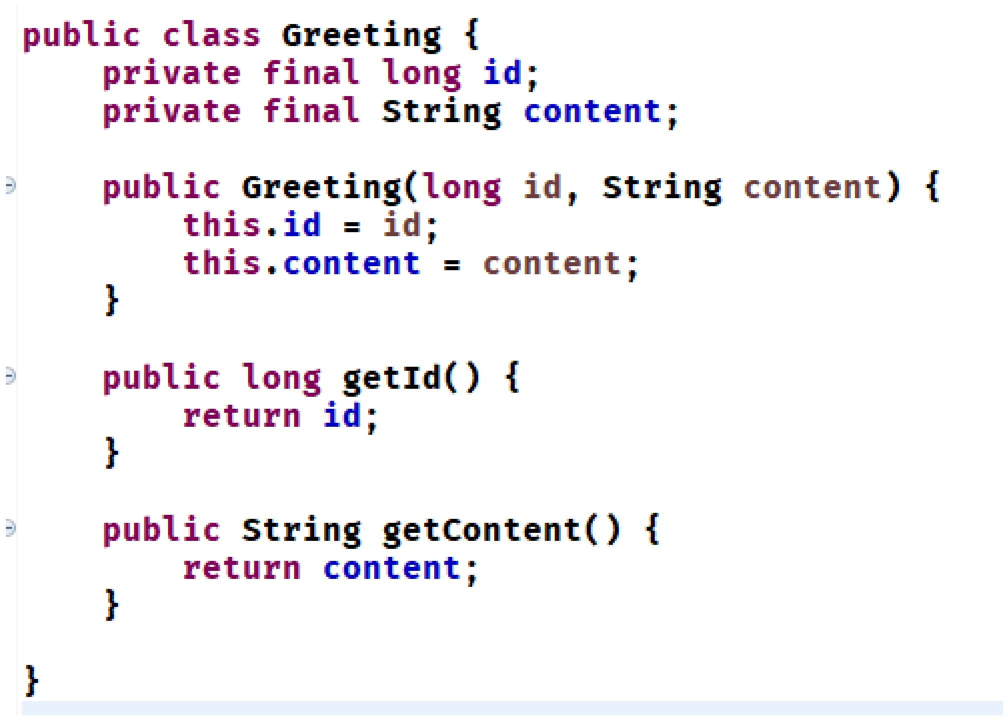
DemoApplication [Java Application] C:\tools\java\jdk-17.0.2\bin\javaw.exe (Feb. 13, 2022, 4:27:15 p.m.)

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

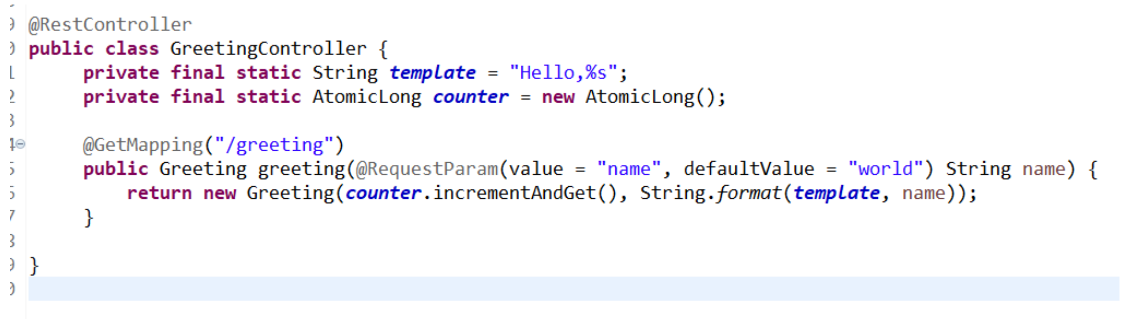
In MVC, the controller handles incoming requests and routes them to the correct model objects to be managed.

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 sophisticate application, we would use something like a templating engine like Thymeleaf to serve up HTML pages.

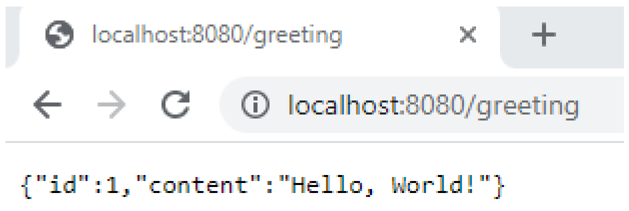
The @GetMapping(URL) maps specific URLs to processing objects. In this case the effect is to create a new model Greeting object and return it.



```java
@RestController
public class GreetingController {
    private final static String template = "Hello,%s";
    private final static 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));
    }

}
```

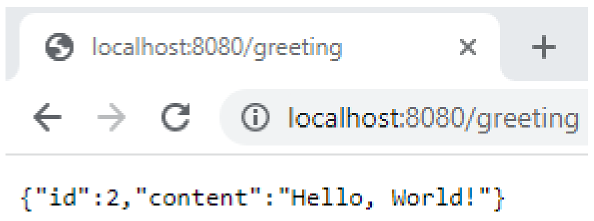
The @RequestParam() annotation looks for an argument in the URL and if there is one, assigns it to the string that appears as the value of the name parameter in the URL. If there is no parameter, it defaults to "World"

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



```
{"id":1,"content":"Hello, World!"}
```

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

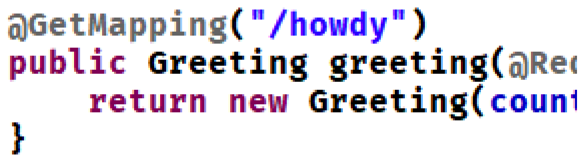


```
{"id":2,"content":"Hello, World!"}
```

7. Change the URL

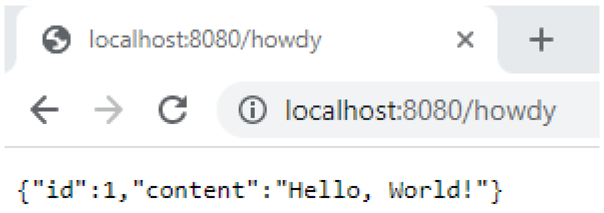
Shut down the server using the red square button

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



```
@GetMapping("/howdy")
public Greeting greeting(@Re
    return new Greeting(coun
}
```

Restart the application and test it out



```
{"id":1,"content":"Hello, World!"}
```

End Lab