Lab MVC: Spring Rest

Objectives

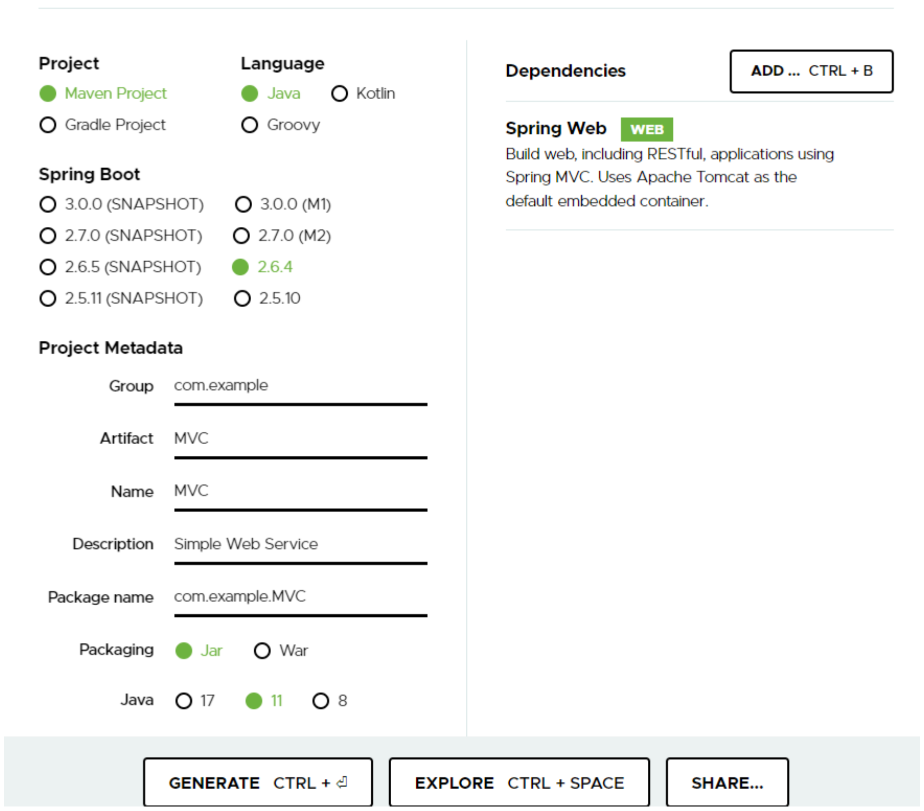
In this lab, you will use Spring MVC to implement a greeting Restful web service that will accept HTTP GET requests at: http://localhost:8080/greeting.

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,

Instructions

Step 1 Create the Spring Boot project

1. Go to https://start.spring.io.
2. We will be using the Spring Web starter. All we 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

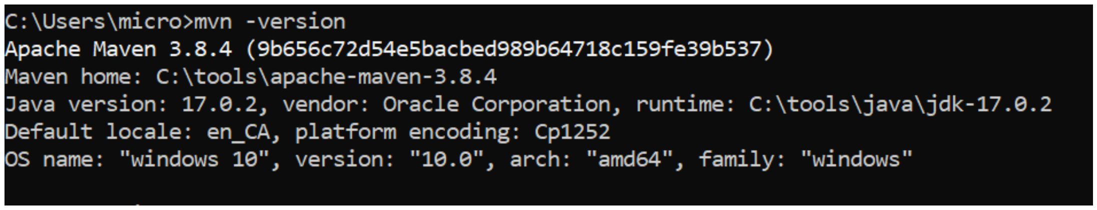
Group	com.example
Artifact	MVC
Name	MVC
Description	Simple Web Service
Package name	com.example.MVC
Packaging	[✓ Jar] War
Java	17 [✓ 11] 8

Dependencies

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

Step 2: Ensure Maven is installed

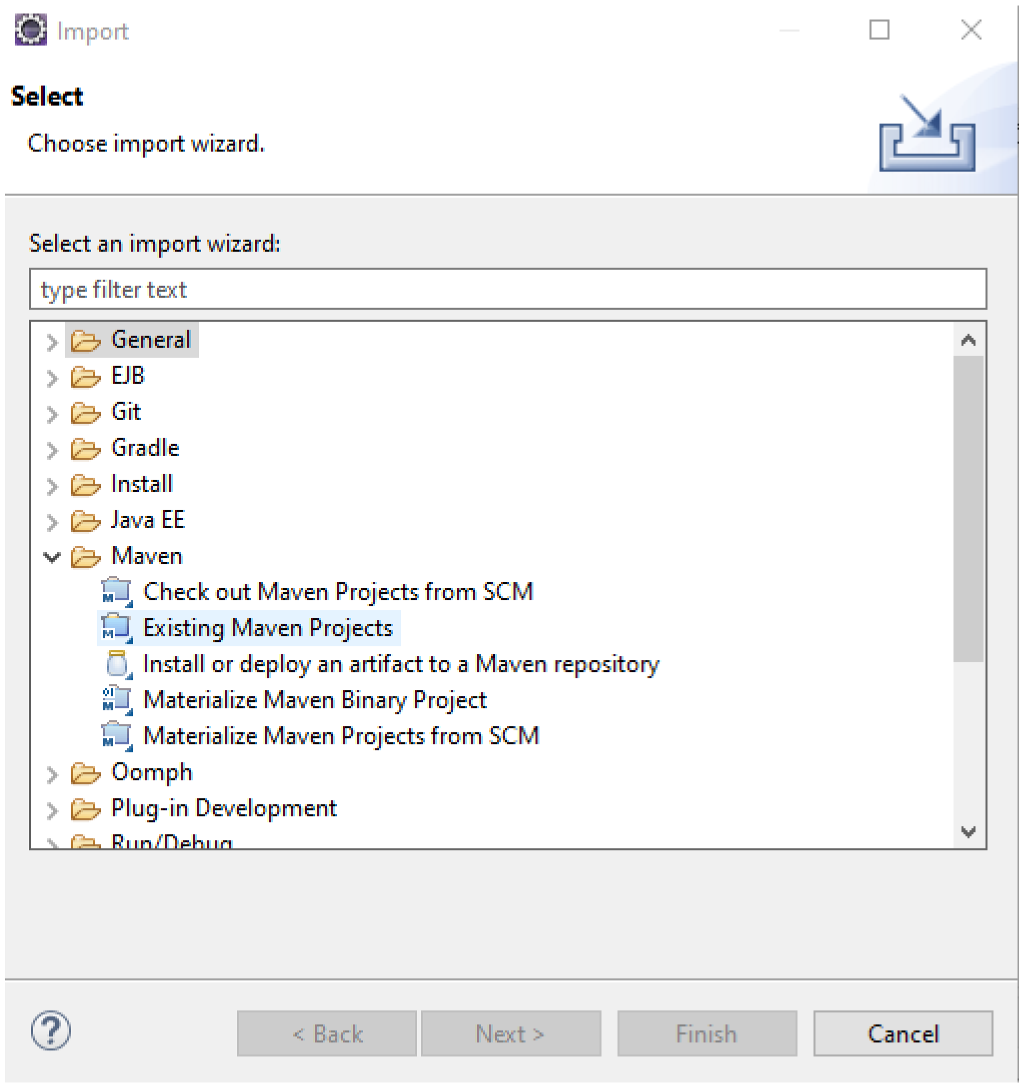
1. Check to see Maven is installed by opening a command window and executing “mvn -version. You should see the following



```
C:\Users\micro>mvn -version
Apache Maven 3.8.4 (9b656c72d54e5bacbed989b64718c159fe39b537)
Maven home: C:\tools\apache-maven-3.8.4
Java version: 17.0.2, vendor: Oracle Corporation, runtime: C:\tools\java\jdk-17.0.2
Default locale: en_CA, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

Step 3: Create the project

1. Create a new eclipse workspace
2. Use the file import option to import the unzipped director as a Maven project.



Import

Select

Choose import wizard.

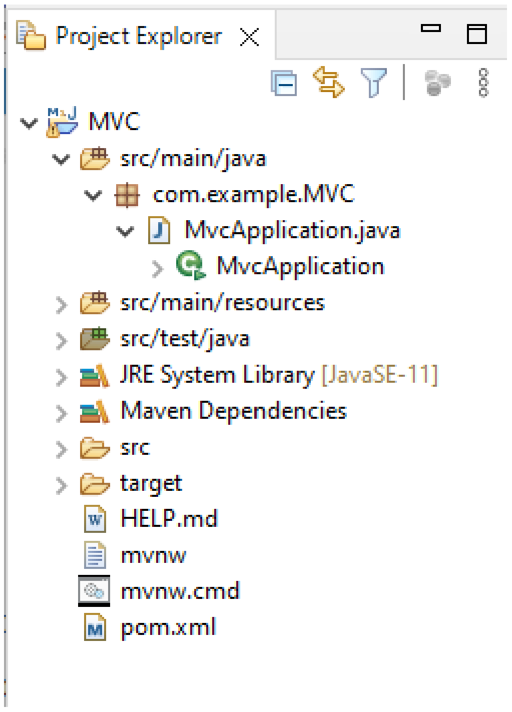
Select an import wizard:

type filter text

- General
- EJB
- Git
- Gradle
- Install
- Java EE
- Maven
  - Check out Maven Projects from SCM
  - Existing Maven Projects
  - Install or deploy an artifact to a Maven repository
  - Materialize Maven Binary Project
  - Materialize Maven Projects from SCM
- Oomph
- Plug-in Development
- Run/Debug

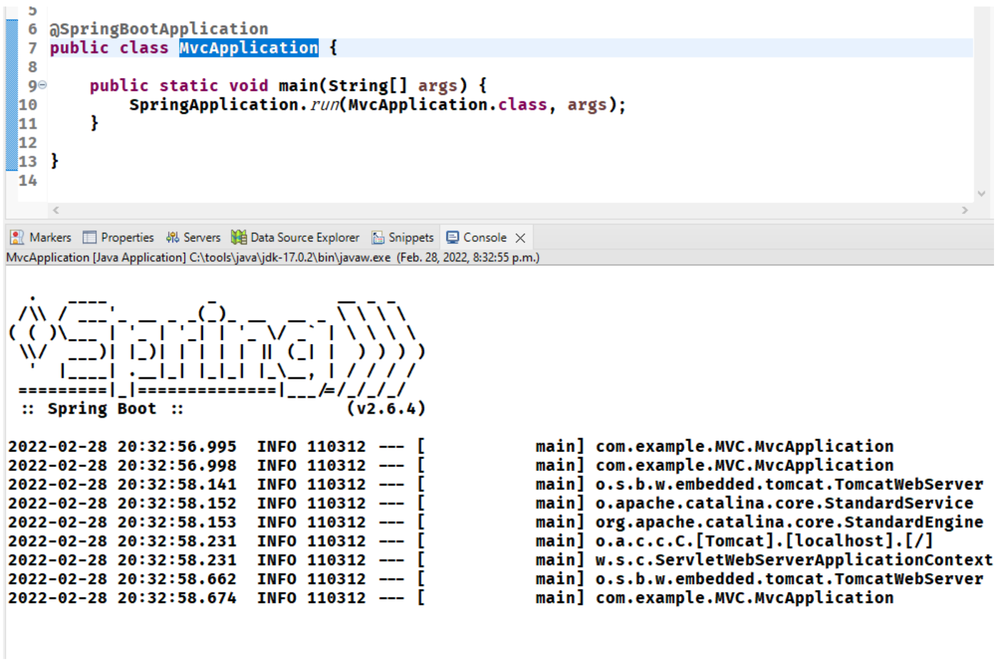
< Back | Next > | Finish | Cancel

3. One the project has been imported; Maven will build all the dependencies. This may take a while but eventually your project should look like this.



```
MVC
  src/main/java
    com.example.MVC
      MvcApplication.java
        MvcApplication
  src/main/resources
  src/test/java
  JRE System Library [JavaSE-11]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

4. Run the DataApplication.java main() method as a Java Application to ensure that everything is working.

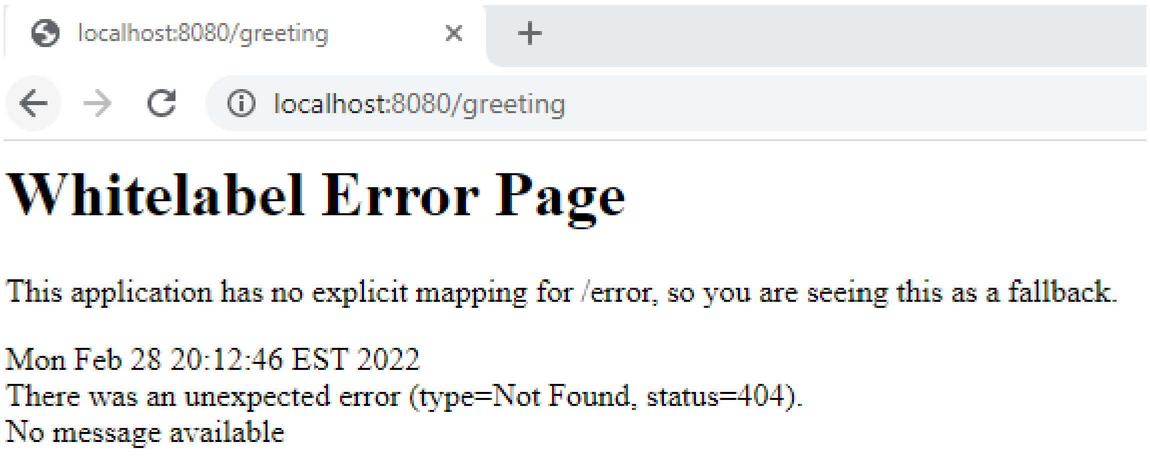


```java
@SpringBootApplication
public class MvcApplication {

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

}
```

MvcApplication [Java Application] C:\tools\java\jdk-17.0.2\bin\javaw.exe (Feb. 28, 2022, 8:32:55 p.m.)

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.6.4)

2022-02-28 20:32:56.995  INFO 110312 --- [           main] com.example.MVC.MvcApplication
2022-02-28 20:32:56.998  INFO 110312 --- [           main] com.example.MVC.MvcApplication
2022-02-28 20:32:58.141  INFO 110312 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2022-02-28 20:32:58.152  INFO 110312 --- [           main] o.apache.catalina.core.StandardService
2022-02-28 20:32:58.153  INFO 110312 --- [           main] org.apache.catalina.core.StandardEngine
2022-02-28 20:32:58.231  INFO 110312 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]
2022-02-28 20:32:58.231  INFO 110312 --- [           main] w.s.c.ServletWebServerApplicationContext
2022-02-28 20:32:58.662  INFO 110312 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2022-02-28 20:32:58.674  INFO 110312 --- [           main] com.example.MVC.MvcApplication
```

5. The web service is up and running, but we haven't added any code so it does nothing. You can see this by going to the URL for the service.

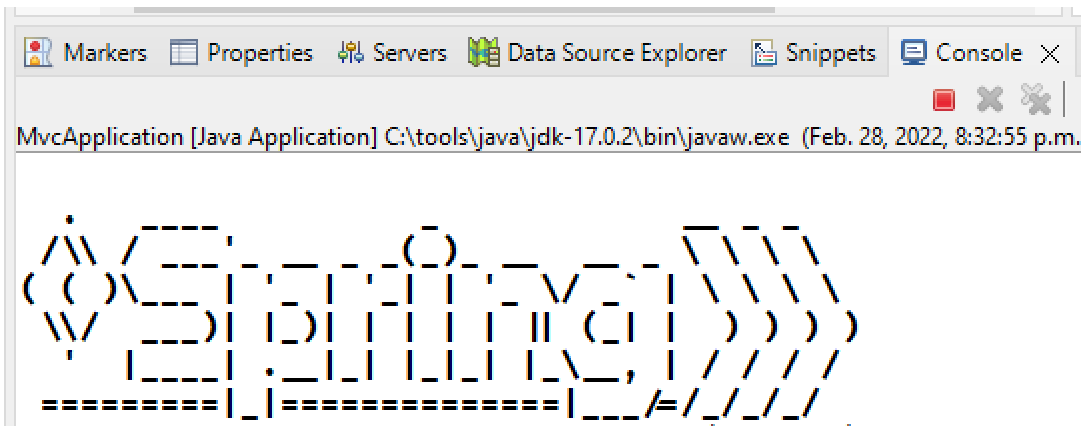


Whitelabel Error Page

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

Mon Feb 28 20:12:46 EST 2022
There was an unexpected error (type=Not Found, status=404).
No message available

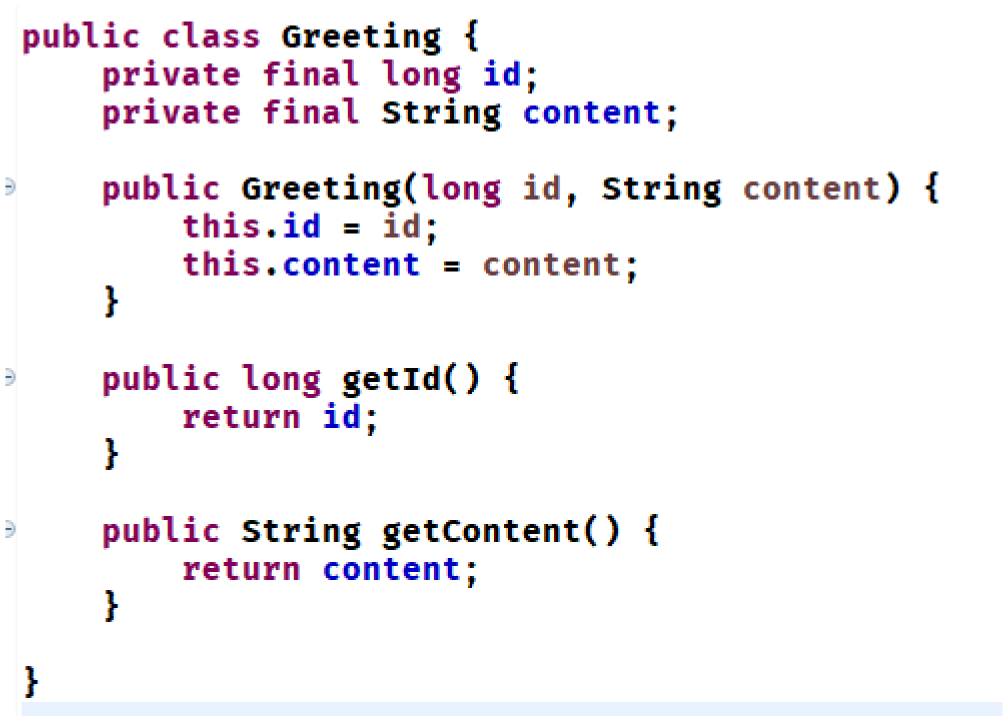
6. Once you have verified that the service runs, shut it down with the red square on the console



```
MvcApplication [Java Application] C:\tools\java\jdk-17.0.2\bin\javaw.exe  (Feb. 28, 2022, 8:32:55 p.m.

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

Step 3: Create the model class

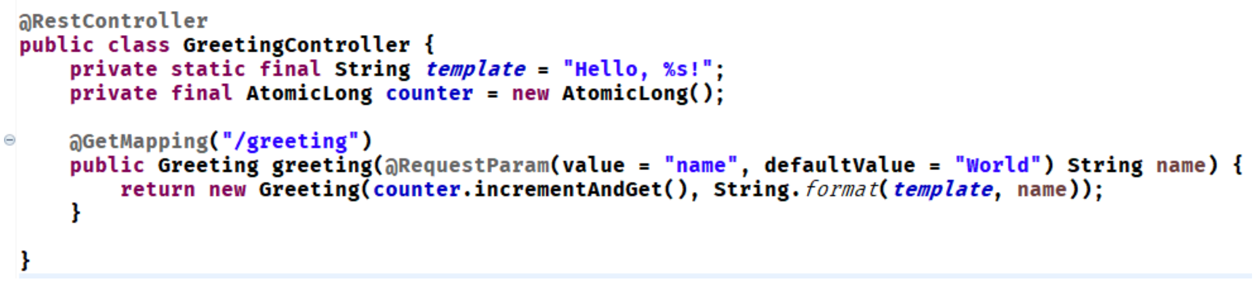
1. The resources that are being referenced by the web service are Greeting entities.
2. These are Pojos that need to know nothing about the controller or the views that are being used.
3. Notice that each resource created will have an id, a convention that is consistent with the REST approach



```
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;
    }

}
```

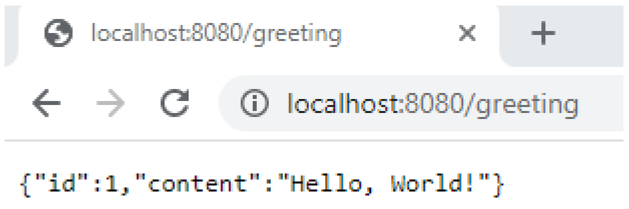
Step 4: Create the Controller

1. In MVC, the controller handles incoming requests and routes them to the correct model objects to me managed.
2. Spring Web provides a number of annotations to define how a controller class is to be managed.



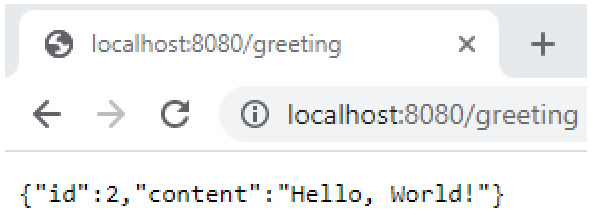
```java
@RestController
public class GreetingController {
    private static final String template = "Hello, %s!";
    private final AtomicLong counter = new AtomicLong();

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

}
```

3. The @RestController annotation tells Spring that this class is the controller.
4. 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.
5. 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.
6. 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”
7. Since we are not using an HTML page, what we will see is just a JSon object representing the Greeting resource.



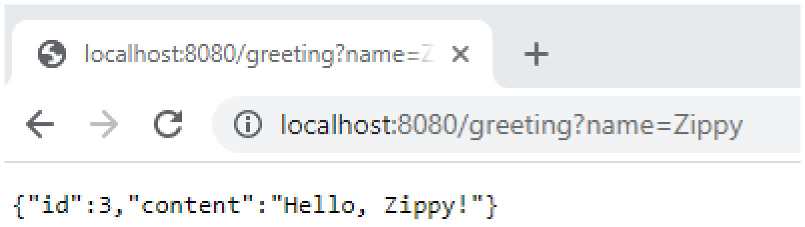
```
{"id":1,"content":"Hello, World!"}
```

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



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

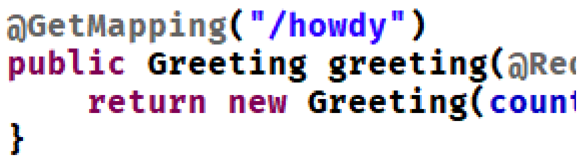
9. Calling with a parameter produces the following


localhost:8080/greeting?name=Z
localhost:8080/greeting?name=Zippy

```
{"id":3,"content":"Hello, Zippy!"}
```

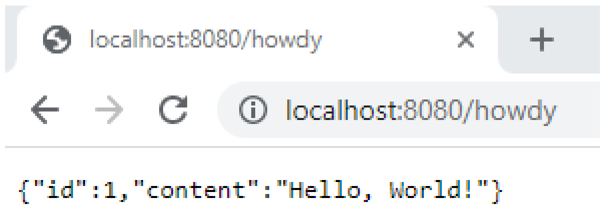
Step:5 Change the URL

1. Shut down the server using the red square button
2. Replace the mapping to /greeting to anything else, like below where "/howdy" is being used



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

3. Restart the application and test it out



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