

Spring Boot

Hands On Lab 1

Table of Contents

Lab 1 - A Simple RESTful API in Spring Boot.....	3
--	---

Lab 1 - A Simple RESTful API in Spring Boot

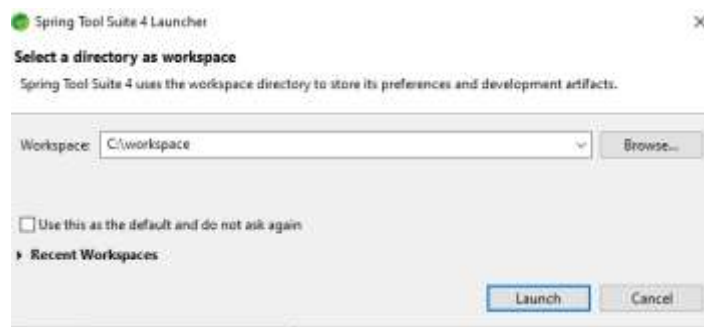
In this lab we're going to build a simple "Hello World" API using Spring Framework and Spring Boot. The API will implement a single resource, "/hello-message" that returns a JSON object that contains a greeting.

Part 1 - Create a Maven Project

We're going to start from scratch on this project, with an empty Apache Maven project, and add in the dependencies that will make a Spring Boot project with a core set of capabilities that we can use to implement our "Hello World" API.

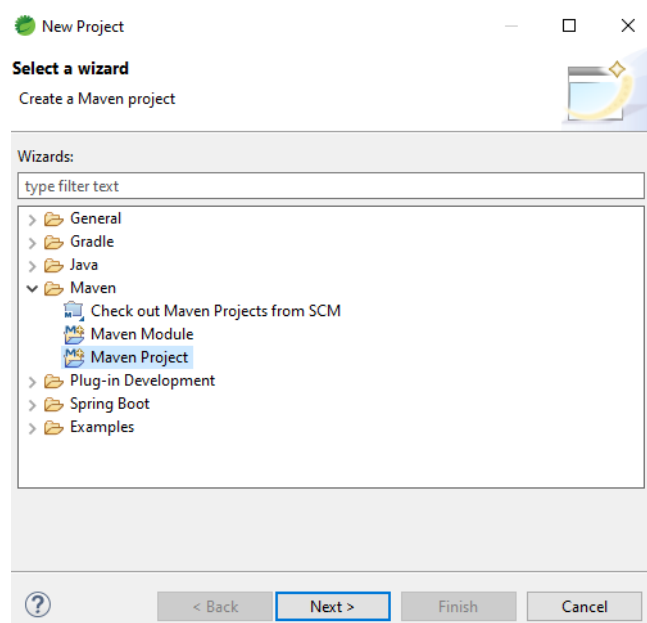
__1. Open Spring Tool Suite by navigating to the directory where you unzipped the distribution and double-clicking on **SpringToolSuite4.exe** (note that the '.exe.' extension may not be shown, depending on the view options that have been set).

__2. In the **Workspace Launcher** dialog, enter 'C:\Workspace' in the **Workspace** field, and then click **OK**. You may also create another workspace for the labs.

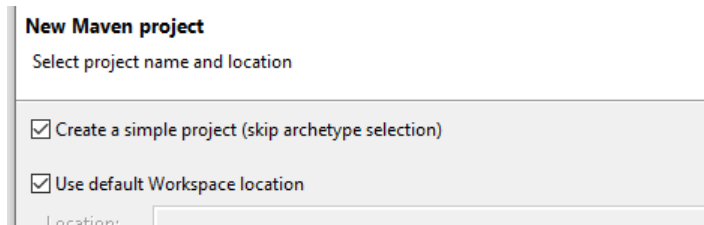


__3. If present, close the **Welcome** panel by clicking on the 'X':

__4. From the main menu, select **File** → **New** → **Project** → **Maven Project**.



__5. In the **New Maven Project** dialog, click on the checkbox to select "Create a simple project (skip archetype selection)", and then click **Next**.



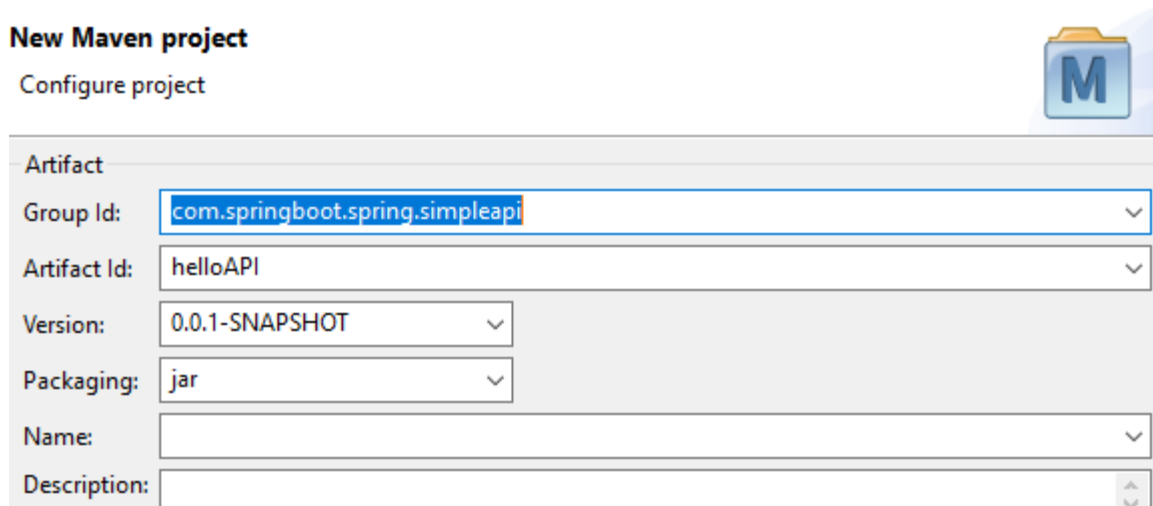
__6. Enter the following fields:

Group Id: com.springboot.spring.simpleapi

ArtifactId: helloAPI

Leave all the other fields at their default values.

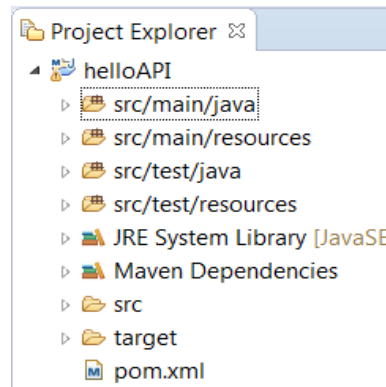
__7. When the dialog looks like below, click **Finish**.



Part 2 - Configure the Project as a Spring Boot Project

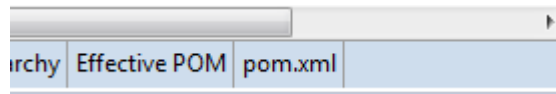
The steps so far have created a basic Maven project. Now we'll add the dependencies to make a Spring Boot project.

__1. Expand the **helloAPI** project in the **Project Explorer**.



__2. Double-click on **pom.xml** to open it.

__3. At the bottom of the editor panel, click the **pom.xml** tab to view the XML source for **pom.xml**.



__4. Insert the following text after the "<version>...</version>" element, and before the closing "</project>" tag.

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.5.5</version>
</parent>
<properties>
  <java.version>11</java.version>
</properties>
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

The entries above call out the Spring Boot Starter Parent project as the parent to this project, then call out the Spring Boot Starter Web dependencies. Finally the <build> element configures the Spring Boot Maven Plugin, which will build an executable jar file for the project.

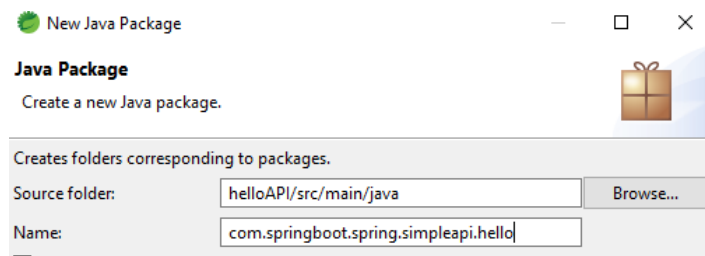
__5. Save the file by pressing **Ctrl-S** or selecting **File -> Save** from the main menu.

Part 3 - Create an Application Class

Spring Boot uses a 'Main' class to startup the application and hold the configuration for the application. In this section, we'll create the main class.

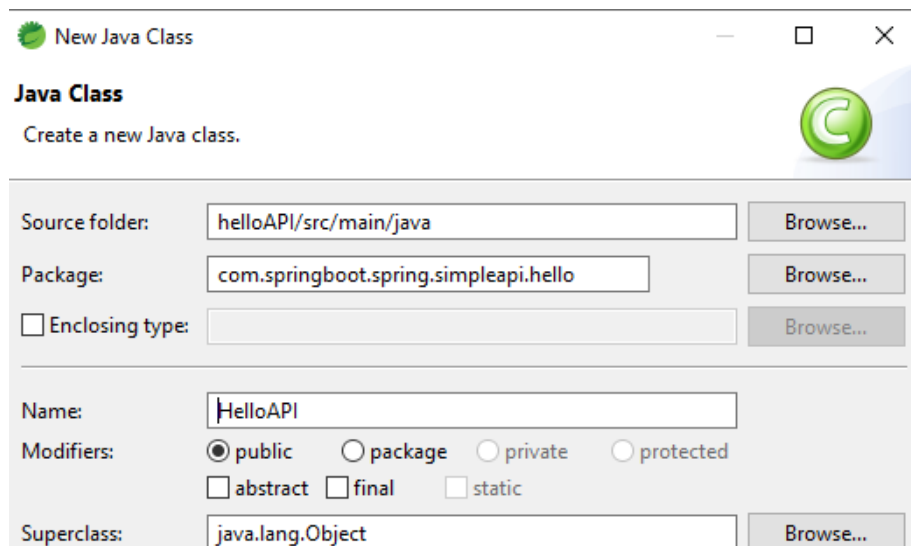
__1. In the **Project Explorer**, right-click on **src/main/java** and then select **New -> Package**.

__2. Enter `com.springboot.spring.simpleapi.hello` in the **Name** field, and then click **Finish**.



__3. In the **Project Explorer**, right-click on the newly-created package and then select **New -> Class**.

__4. In the **New Java Class** dialog, enter 'HelloAPI' as the **Name**, and then click **Finish**.



__5. Add the '@SpringBootApplication' annotation to the class, so it appears like:

```
@SpringBootApplication
public class HelloAPI {
```

__6. Add the following 'main' method inside the class:

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

__7. The editor is probably showing errors due to missing 'import' statements. Press **Ctrl-Shift-O** to organize the imports.

__8. Save the file.

__9. The **helloAPI** project node may show a small red 'x' to indicate an error. If so, right-click on the **helloAPI** project and then select **Maven** → **Update Project**, and then click **OK** in the resulting dialog.

__10. In the **Project Explorer**, right-click on either the **helloAPI** project node or the 'pom.xml' file and then select **Run As** → **Maven Install**.

Note. If fails building try again and the second time should works.

The console should show a successful build. This ensures that we don't have any typos in the pom.xml entries we just did.

```
[INFO] Downloaded from : https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven
[INFO] Downloaded from : https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-ut
[INFO] Installing C:\workspace\Assignments\helloAPI\target\helloAPI-0.0.1-SNAPSHOT.jar to C
[INFO] Installing C:\workspace\Assignments\helloAPI\pom.xml to C:\Users\steve\.m2\repositor
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 01:35 min
[INFO] Finished at: 2021-10-02T20:31:29+05:30
[INFO] -----
```

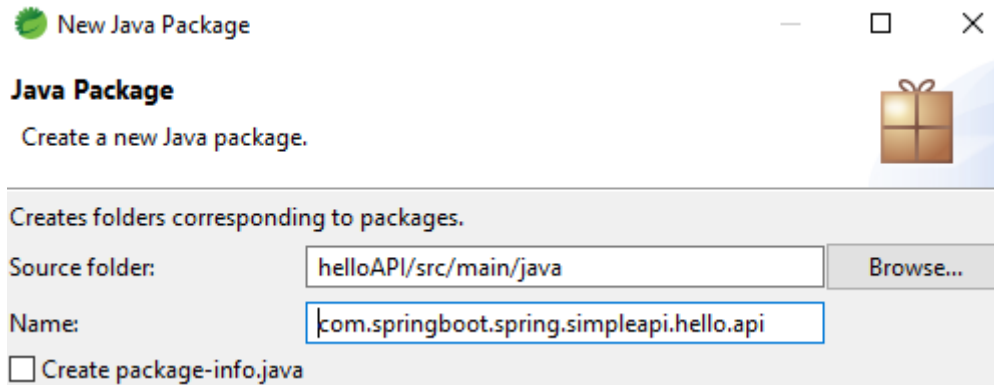
Now all we need to do is add a resource class and a response class.

Part 4 - Implement the RESTful Service

In this part of the lab, we will create a response class and a RESTful resource class,

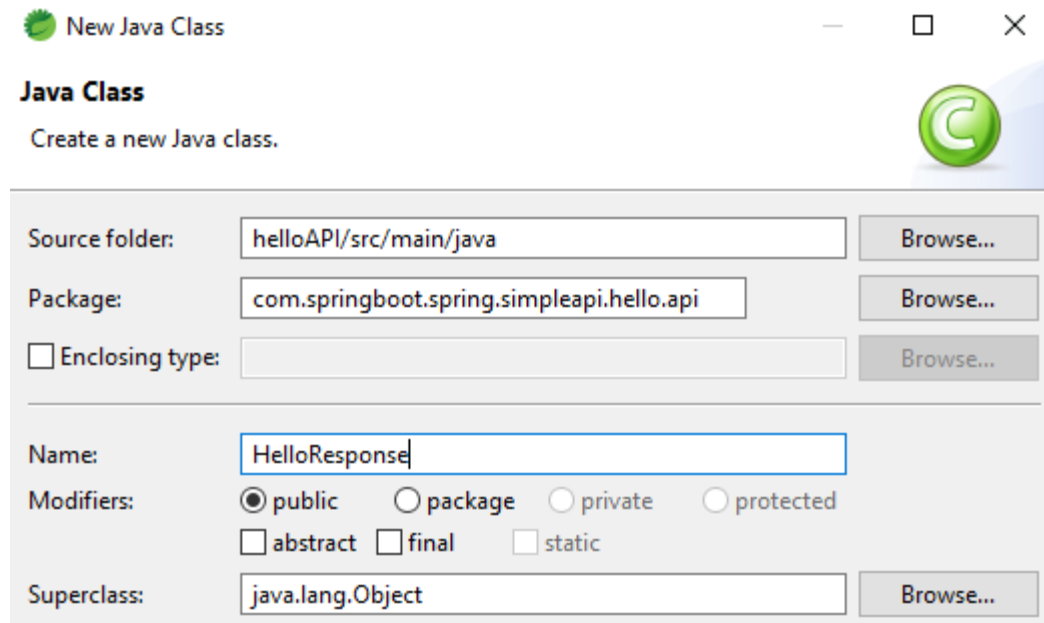
__1. In the **Project Explorer**, right-click on **src/main/java** and then select **New → Package**.

__2. Enter `com.springboot.spring.simpleapi.hello.api` in the **Name** field, and then click **Finish**.



__3. In the **Project Explorer**, right-click on the newly-created package and then select **New → Class**.

__4. In the **New Java Class** dialog, enter 'HelloResponse' as the **Name**, and then click **Finish**.



__5. Edit the body of the class so it reads as follows:

```
package com.springboot.spring.simpleapi.hello.api;

public class HelloResponse {
    String message;

    public HelloResponse(String message) {
        super();
        this.message = message;
    }

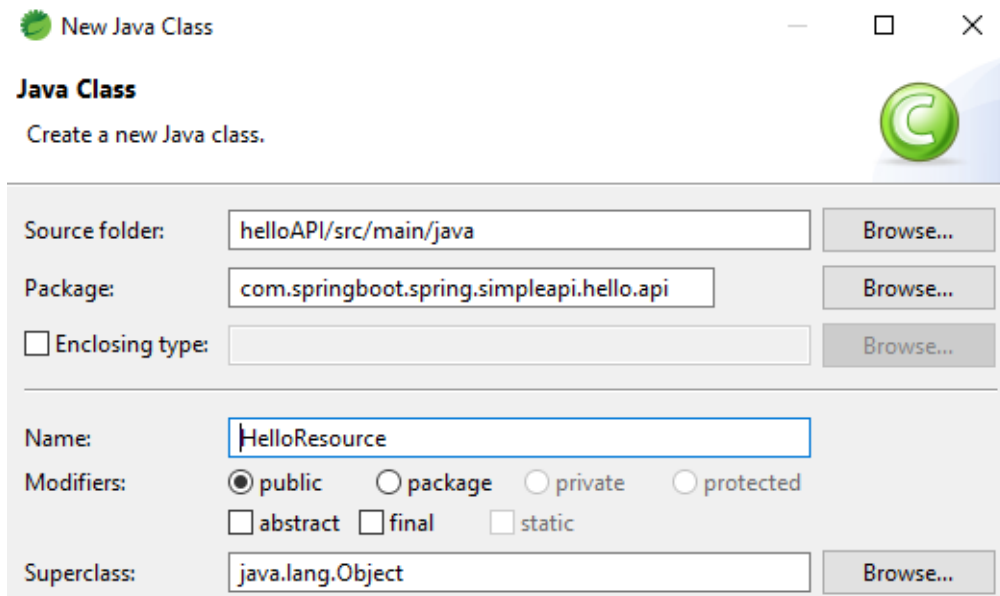
    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

__6. Save the file.

__7. In the **Project Explorer**, right-click on the com.springboot.spring.simpleapi.hello.api package and then select **New** → **Class**.

__8. In the **New Java Class** dialog, enter 'HelloResource' as the **Name**, and then click **Finish**.



New Java Class

Java Class
Create a new Java class.

Source folder: helloAPI/src/main/java Browse...

Package: com.springboot.spring.simpleapi.hello.api Browse...

☐ Enclosing type: Browse...

Name: HelloResource

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

__9. Add the following 'getMessage' method inside the new class:

```
public HelloResponse getMessage() {  
    return new HelloResponse("Hello!");  
}
```

Spring Boot recognizes and configures the RESTful resource components by the annotations that we're about to place on the resource class that we just created.

__10. Add the '@RestController' annotation to HelloResource, so it looks like:

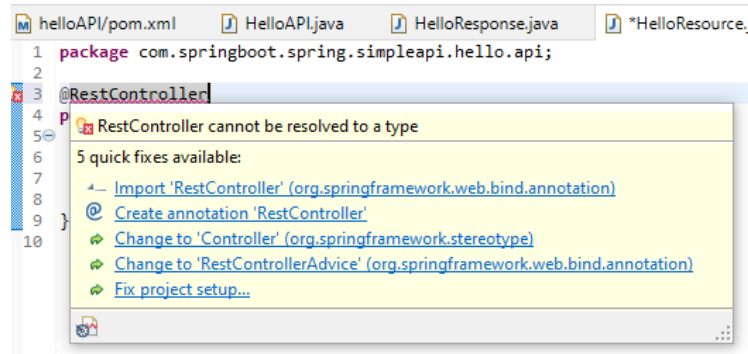
```
@RestController  
public class HelloResource {
```

__11. Add the '@GetMapping' annotation to the 'getMessage' method, so it looks like:

```
@GetMapping("/hello-message")  
public HelloResponse getMessage() {
```

__12. Organize the imports by pressing **Ctrl-Shift-O**.

Better yet, hover over the red squiggly line under the annotation and choose the required import.



__13. Save all files by pressing **Ctrl-Shift-S**.

__14. In the **Project Explorer**, right-click on either the **helloAPI** project node or the 'pom.xml' file and then select **Run As** → **Maven Install**.

Note. If fails building try again and the second time should works.

The console should show a successful build.

```
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ helloAPI ---  
[INFO] Installing C:\workspace\Assignments\helloAPI\target\helloAPI-0.0.1-SNAPSHOT-  
[INFO] Installing C:\workspace\Assignments\helloAPI\pom.xml to C:\Users\steve\.m2\re  
[INFO] BUILD SUCCESS  
[INFO] Total time: 26.488 s  
[INFO] Finished at: 2021-10-02T20:50:56+05:30
```

Part 5 - Run and Test

10

That's all the components required to create a simple RESTful API with Spring Boot. Now let's fire it up and test it!

__1. In the **Project Explorer**, right-click on the **HelloAPI** class and then select **Run as** → **Java Application**.

__2. If the **Windows Security Alert** window pops up, click on **Allow Access**.

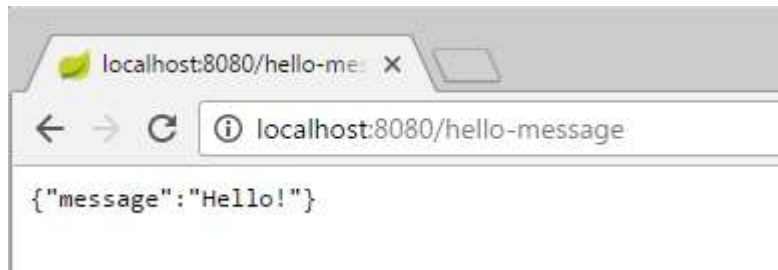
__3. Watch the **Console** panel. At the bottom of it, you should see a message indicating that the 'HelloAPI' program has started successfully:

```
2021-10-02 21:00:14.883 INFO 9956 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2021-10-02 21:00:14.910 INFO 9956 --- [main] c.s.spring.simpleapi.hello.HelloAPI : Started HelloAPI in 7.36 seconds (JVM running for 8.556)
```

__4. Open the **Chrome** browser and enter the following URL in the location bar:

```
http://localhost:8080/hello-message
```

__5. You should see the following response:



Notice that the response is in the form of a JSON object whose structure matches the 'HelloResponse' class contents.

__6. Close the browser.

__7. Click on the red 'Stop' button on the **Console** panel to stop the application or close it from the Boot Dashboard.

__8. Close all open files.

Part 6 - Review

In this lab, we setup a rudimentary Spring Boot application. There are a few things you should notice:

- There was really very little code and configuration required to implement the very simple RESTful API.
- The resulting application runs in a standalone configuration without requiring a web or application server. It opens its own port on 8080 (we'll see later how to configure this port to any value you want).
- Although the Eclipse IDE is providing some nice features, like type-ahead support and automatic imports, the only tool we really need is a build tool that does dependency management (e.g. Apache Maven).