Spring Boot Build RESTful API

Description: Spring Boot is an open source, micro service-based Java web framework. The Spring Boot framework creates a fully production-ready environment that is completely configurable using its prebuilt code within its own codebase.

Project: Build a simple RESTful API endpoint which any browser can connect to. The demo will contain an @RestController with one @GetMapping API. This project is based on Building a RESTful Web Service and just expands on the screenshots and step by step instructions.

Technology: This project uses the following technology:

- Integrated Development Environment (IDE):
  - Spring Tool Suite 4 (Version: 4.11.0.RELEASE)
  - Visual Studio Code (V8: 9.8.177.11)
- Java Development Kit (JDK):
  - Oracle's JDK 8 (1.8)

Table of Contents

Glossary of Terminology

For a list of key terms and definitions used throughout this and various Spring Boot demo documents see the document titled “Appendix 01 Glossary”.

Generate Spring Boot Download

Follow the instructions in the document title “Appendix 02 Spring Initializr” to generate a spring boot download for this project.

When the document talks about the items in the “**Project Metadata**” use the values shown below:

> “**Group**” use “**com.example**”
> “**Artifact**” use “**demo-spring-boot-restful**”
> “**Name**” use “**rest-service**”
> “**Package name**” use “**com.example.rest-service**”

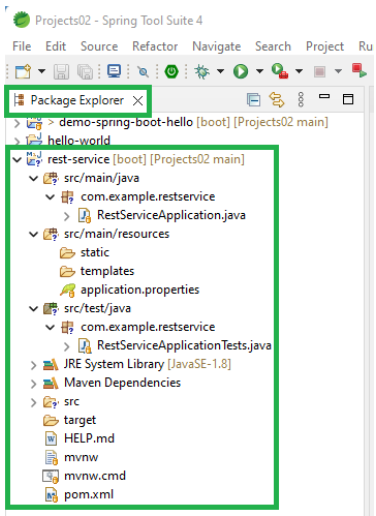
For other items in the “**Project Metadata**” use the defaults. Follow the instructions to extract the files from the zip file into the Sprint Tool Suite 4 workspace.

Import the Spring Boot Download

Open the Spring Tool Suite 4 IDE.

Import the project: “rest-service”

Follow the instructions in the document title “Appendix 03 Import Spring Tool Suite Project” and import the “**rest-service**” project that was created with Spring Initializr. After importing the project, it should look like the following using the IDE “Package Explorer”.



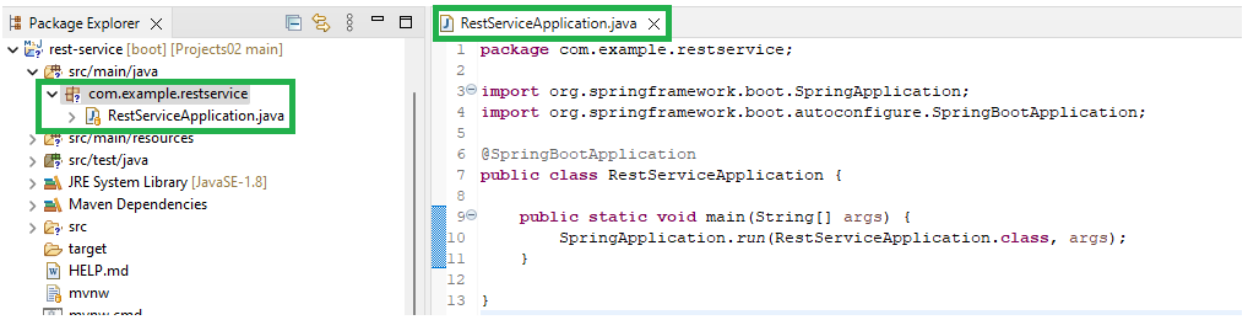
Demo Project rest-service Discussion

Create a Resource Representation Class

> "To model the greeting representation, create a resource representation class. To do so, provide a plain old Java object with fields, constructors, and accessors for the `id` and `content` data, as the following listing (from `src/main/java/com/example/restservice/Greeting.java`)." -- [Building a RESTful Web Service]

Class: RestServiceApplication

The RestServiceApplication class created when using the Spring Initializr is used without any additional modification from what was generated. This class contains the "`public static void main(String[] args) {}`" method that starts the application listening when the internal web server is started.



You can add the following block comment above the annotation to access the web URLs to use for testing when the project is finished.

```
/*
 * http://localhost:8080/greeting
 *              {"id":1,"content":"Hello, World!"}
 * http://localhost:8080/greeting?name=User
 *              {"id":1,"content":"Hello, User!"}
 */

@SpringBootApplication
```

The main class "RestServiceApplication" examined:

> "`@SpringBootApplication` is a convenience annotation that adds all of the following:
>
> - `@Configuration`: Tags the class as a source of bean definitions for the application context.
> - `@EnableAutoConfiguration`: Tells Spring Boot to start adding beans based on classpath settings, other beans, and various property settings. For example, if `spring-webmvc` is on the classpath, this annotation flags the application

as a web application and activates key behaviors, such as setting up a `DispatcherServlet.`

- `@ComponentScan`: Tells Spring to look for other components, configurations, and services in the `com/example` package, letting it find the controllers

The `main()` method uses Spring Boot's `SpringApplication.run()` method to launch an application. Did you notice that there was not a single line of XML? There is no `web.xml` file, either. This web application is 100% pure Java and you did not have to deal with configuring any plumbing or infrastructure." -- Building a RESTful Web Service

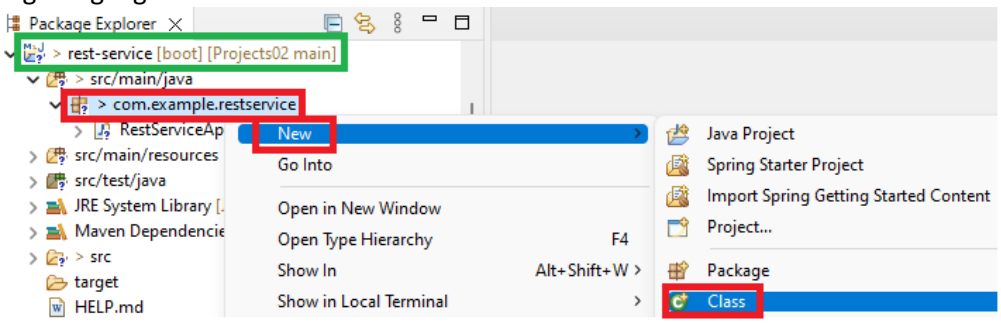
Two other classes are created to complete the project. The following sections detail creating these classes.

Create Model and Controller classes

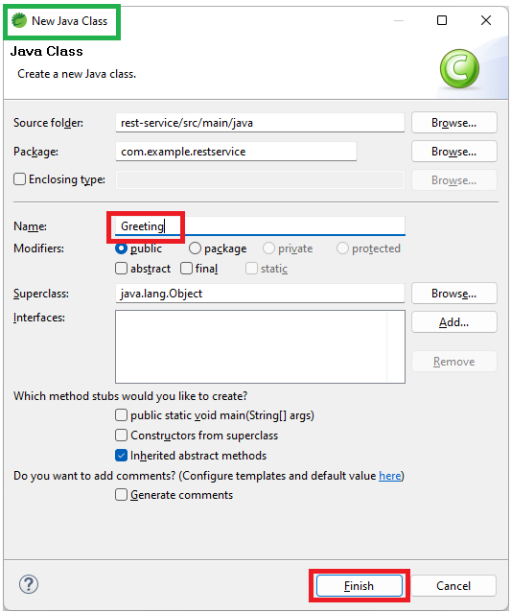
A model class is a generic class that defines an object, something tangible.
A controller class provide the entry points to the program.

Create Model Class: Greeting

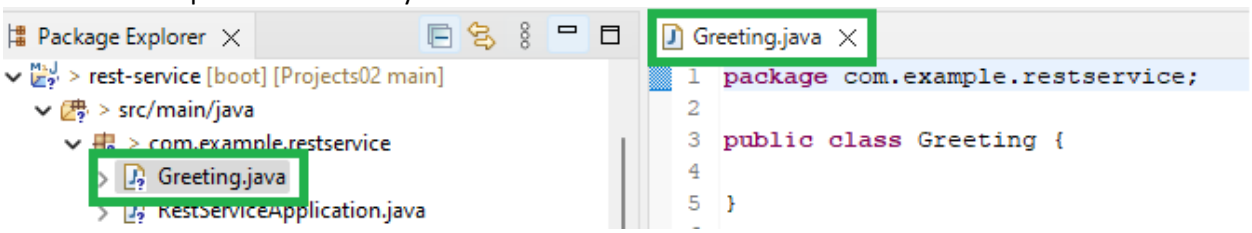
Right click on service package “com.example.restservice”
Right highlight “New” ➔ select “Class”



Enter class “Name:” “Greeting”
Click “Finish”



The new class opens automatically in an edit window.



Update model class: Greeting

Copy and paste the following code replacing the wizard generated class shell:

```
package com.example.restservice;

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;
        }
}
```

This is a basic model class and will have no import errors.

Greeting.java

```
1  package com.example.restservice;
2
3  public class Greeting {
4
5      private final long id;
6      private final String content;
7
8      public Greeting(long id, String content) {
9          this.id = id;
10         this.content = content;
11     }
12
13     public long getId() {
14         return id;
15     }
16
17     public String getContent() {
18         return content;
19     }
20 }
21
```

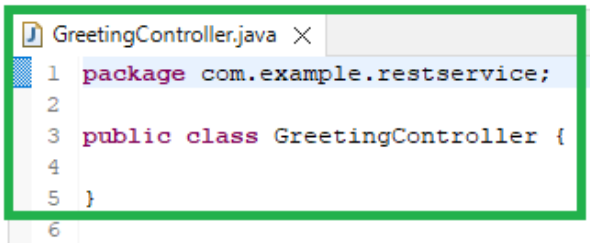
Create Controller Class: GreetingController

> "In Spring's approach to building RESTful web services, HTTP requests are handled by a controller. These components are identified by the `@RestController` annotation, and the `GreetingController` shown in the following listing (from `src/main/java/com/example/restservice/GreetingController.java`) handles `GET` requests for `/greeting` by returning a new instance of the `Greeting` class" -- [Building a RESTful Web Service]

Like before when creating the first class for this project.
Right click on service package "com.example.restservice"
Highlight "New" ➔ select "Class"

Enter Class "Name:" "GreetingController"
Click "Finish".

The class opens automatically in an edit window.



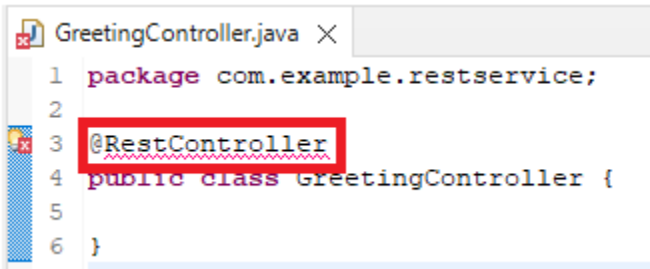
Update controller class: GreetingController

Add RestController class annotation

RestController - Used for making restful web services. This annotation is used at the class level and allows the class to handle the requests made by the client.

```
@RestController
```

Add the annotation above the class definition.



Correct the import error:
Use the IDE wizard to help correct the error.
Hover the mouse over the error.

Select the import link shown below.

```
GreetingController.java
1  package com.example.restservice;
2
3  @RestController
4  p
5
6  }
7
   RestController cannot be resolved to a type
   5 quick fixes available:
   Import 'RestController' (org.springframework.web.bind.annotation)
```

Save the file and see that the import is added and the error is cleared.

```
GreetingController.java
1  package com.example.restservice;
2
3  import org.springframework.web.bind.annotation.RestController;
4
5  @RestController
6  public class GreetingController {
7
8  }
```

Add class variables:

```
private static final String template = "Hello, %s!";
private final AtomicLong counter = new AtomicLong();
```

```
5  @RestController
6  public class GreetingController {
7      private static final String template = "Hello, %s!";
8      private final AtomicLong counter = new AtomicLong();
9
10 }
```

Use the IDE wizard to help correct the error.

```
7      private static final String template = "Hello, %s!";
8      private final AtomicLong counter = new AtomicLong();
9
10 }
11
   AtomicLong cannot be resolved to a type
   8 quick fixes available:
   Import 'AtomicLong' (java.util.concurrent.atomic)
```

Add GetMapping Annotation and Endpoint Method

GetMapping maps HTTP GET requests onto specific handler methods.
Add Endpoint Method: greeting

Add the annotation and method to the class shell:

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

```
7  @RestController
8  public class GreetingController {
9      private static final String template = "Hello, %s!";
10     private final AtomicLong counter = new AtomicLong();
11
12     @GetMapping("/greeting")
13     public Greeting greeting(@RequestParam(value = "name", defaultValue = "World") String name) {
14         return new Greeting(counter.incrementAndGet(), String.format(template, name));
15     }
16
17 }
```

Use the IDE wizard to help correct the errors.

```
11
12     @GetMapping("/greeting")
13     p
14
15     }
16
```

GetMapping cannot be resolved to a type

9 quick fixes available:

Import 'GetMapping' (org.springframework.web.bind.annotation)

```
13     @GetMapping("/greeting")
14     public Greeting greeting @RequestParam value = "name", defaultValue = "World"
15         return new Greeting(co
16     }
17
18 }
19
```

RequestParam cannot be resolved to a type

6 quick fixes available:

Import 'RequestParam' (org.springframework.web.bind.annotation)

Create annotation 'RequestParam'

The IDE wizard to helped to resolve the errors and produced the finished results.

GreetingController.java

```
1  package com.example.restservice;
2
3  import java.util.concurrent.atomic.AtomicLong;
4
5  import org.springframework.web.bind.annotation.GetMapping;
6  import org.springframework.web.bind.annotation.RequestParam;
7  import org.springframework.web.bind.annotation.RestController;
8
9  @RestController
10 public class GreetingController {
11     private static final String template = "Hello, %s!";
12     private final AtomicLong counter = new AtomicLong();
13
14     @GetMapping("/greeting")
15     public Greeting greeting(@RequestParam(value = "name", defaultValue = "World") String name) {
16         return new Greeting(counter.incrementAndGet(), String.format(template, name));
17     }
18
19 }
```

The GreetingController class examined:

"The `@GetMapping` annotation ensures that HTTP GET requests to `/greeting` are mapped to the `greeting()` method.

`@RequestParam` binds the value of the query string parameter `name` into the `name` parameter of the `greeting()` method. If the `name` parameter is absent in the request, the `defaultValue` of `World` is used.

The implementation of the method body creates and returns a new `Greeting` object with `id` and `content` attributes based on the next value from the `counter` and formats the given `name` by using the greeting `template`.

A key difference between a traditional MVC controller and the RESTful web service controller shown earlier is the way that the HTTP response body is created. Rather than relying on a view technology to perform server-side rendering of the greeting data to HTML, this RESTful web service controller populates and returns a `Greeting` object. The object data will be written directly to the HTTP response as JSON.

This code uses Spring `@RestController` annotation, which marks the class as a controller where every method returns a domain object instead of a view. It is shorthand for including both `@Controller` and `@ResponseBody`.

The `Greeting` object must be converted to JSON. Thanks to Spring's HTTP message converter support, you need not do this conversion manually. Because Jackson 2 is on the classpath, Spring's `MappingJackson2HttpMessageConverter` is automatically chosen to convert the `Greeting` instance to JSON." -- Building a RESTful Web Service

Copy and paste source code for class: GreetingController

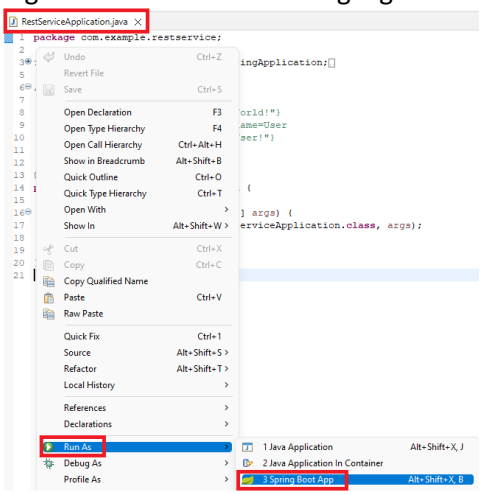
As an alternative you can copy the code below and replace all code in the class if you used the same package name.

```
package com.example.restservice;

import java.util.concurrent.atomic.AtomicLong;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@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));
        }
}
```

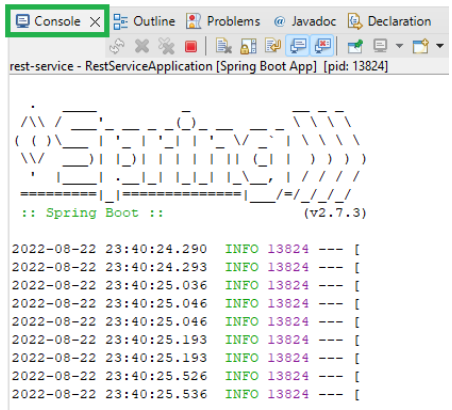
Quick Project Test

To do a quick test of the project use the Spring Boot bundled server. Later “Appendix 04 Run Spring Initializr Project” can be used to use a command line server version and to build an executable jar.

Open class RestServiceApplication.
Right click inside the class highlight “Run As” ➔ “3 Spring Boot App”



The Spring Boot server status is shown in the “Console” tab.

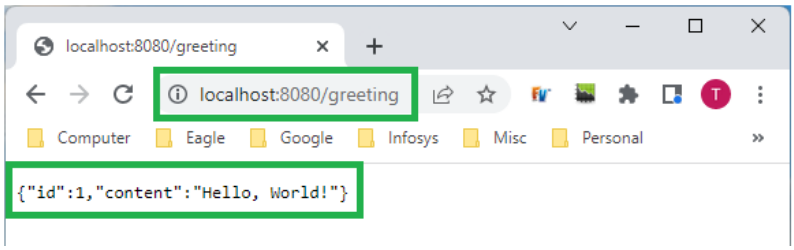


Test the Application’s URLs

Use the following URL in a browser.

```
http://localhost:8080/greeting
```

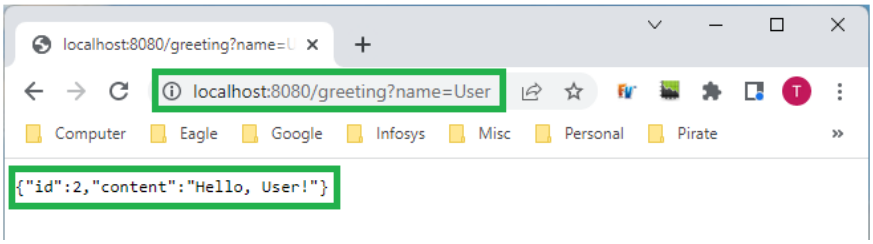
Expected results.



Use the next URL in a browser.

```
http://localhost:8080/greeting
```

Expected results.



Build and Run the Project

This project introduces using an executable Java Archive (JAR) file. The "Appendix 04 Run Spring Initializr Project" document section title "Executable JAR File Lifecycle" should be followed to learn the new concept. At a minimum follow section "Executable JAR File Lifecycle" in the Appendix 04 Run Spring Initializr Project" document.

Using the executable JAR file that is built for this project, the URLs in the previous section "Test the Application's URLs" should be used and the results should be the same.