Spring Boot @Controller Annotation with Example

Last Updated : 23 Jul, 2025

Spring is one of the most popular frameworks for building enterprise-level Java applications. It is an open-source, lightweight framework that simplifies the development of robust, scalable, and maintainable applications. Spring provides various features such as Dependency Injection (DI), Aspect-Oriented Programming (AOP), and support for Plain Old Java Objects (POJOs), making it a preferred choice for Java developers.

In this article, we will focus on the **@Controller annotation in Spring**, which is a key component in building web applications using the Spring MVC framework.

@Controller Annotation in Spring Boot

The **@Controller annotation** is a specialization of the @Component annotation. It is used to mark a class as a controller in a Spring MVC application. Controllers are responsible for handling incoming web requests, processing them, and returning the appropriate response.

Key Points about @Controller:

- It is typically used in combination with the @RequestMapping annotation to map web requests to specific handler methods.
- It can only be applied to classes.
- It is part of the Spring MVC framework and is used to create web controllers.
- The @Controller annotation allows the Spring framework to detect the class as a controller during component scanning.

Steps to Use the @Controller Annotation

Let's consider a simple example to understand how to use the @Controller annotation in a Spring Boot application.

Step 1: Create a Simple Spring Boot Project

Refer to this article Create and Setup Spring Boot Project in Eclipse IDE and create a simple spring boot project.

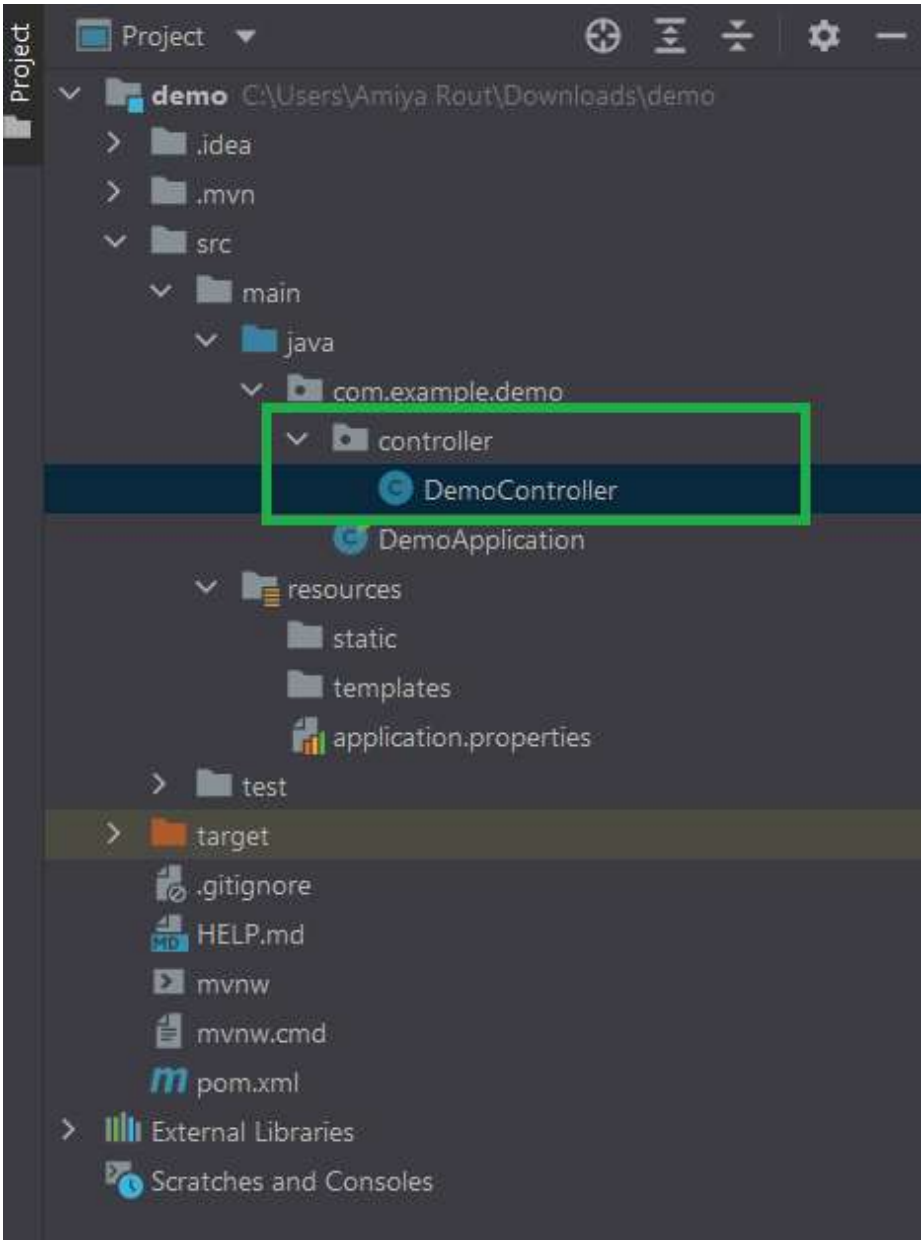
Step 2: Add the spring-web dependency

In your pom.xml file, inside your project and add the following **spring-web dependency.**

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Step 3: Create a Controller Package

Create a package named controller. This package will contain your controller classes. This is going to be our final project structure.



Step 4: Create a Controller Class

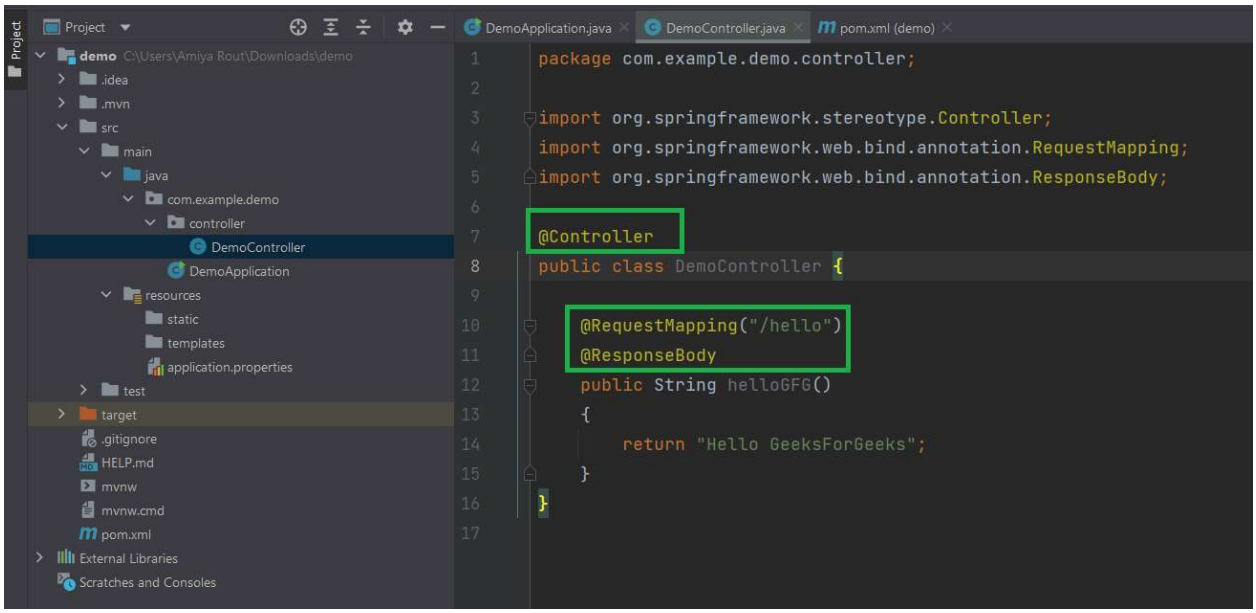
Inside the controller package, create a class named DemoController. This class will act as the controller for handling web requests.

```
// Java Program to Illustrate DemoController File

// Importing package in this code module
package com.example.demo.controller;
// Importing required classes
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
```

```
// Annotation
@Controller
// Main class
public class DemoController {

    @RequestMapping("/hello")
    @ResponseBody

    // Method
    public String helloGFG()
    {
        return "Hello GeeksForGeeks";
    }
}
```

We have used the below annotations in our controller layer. Here in this example, the URI path is ***/hello***.

- **@Controller:** This is used to specify the controller.
- **@RequestMapping:** This is used to map to the Spring MVC controller method.
- **@ResponseBody:** Ensures the returned value is sent as an HTTP response instead of resolving a view name.



Step 5: Run the Application

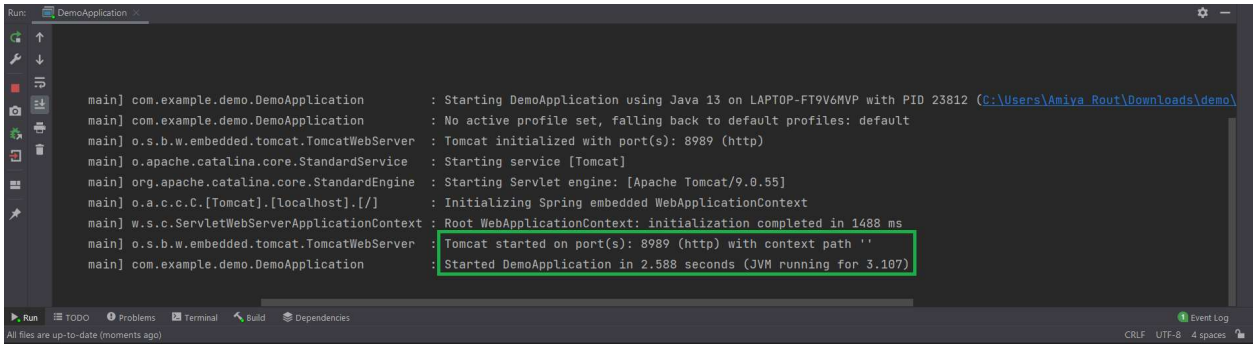
To run the application, navigate to the main class (usually named DemoApplication.java) and execute it. The Spring Boot application will

start, and you can access the **/hello** endpoint in your browser or using a tool like Postman.

```java
// Java Program to Illustrate DemoApplication File

// Importing package in this code module
package com.example.demo;
// Importing required classes
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

// Annotation
@SpringBootApplication

// Main class
public class DemoApplication {

    // Main driver method
    public static void main(String[] args)
    {

        SpringApplication.run(DemoApplication.class, args);
    }
}
```

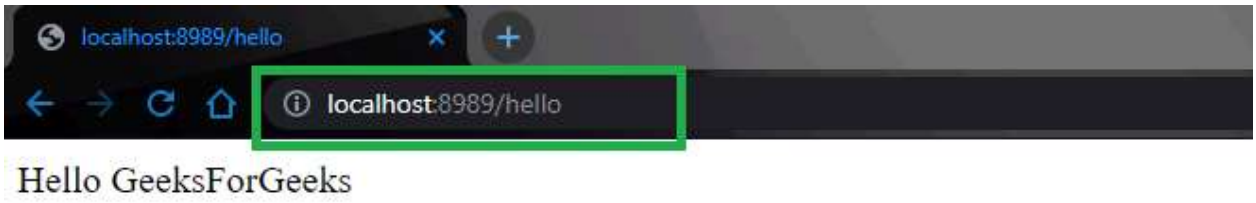
"@SpringBootApplication" enables component scanning, auto-configuration, and Spring Boot features, ensuring all annotated classes within the package hierarchy are detected.

Step 6: Test the Application



Note: Note: By default, Spring Boot runs on port 8080. If not changed, access the endpoint at http://localhost:8080/hello. If you have configured a

custom port (e.g., 8989), update "server.port=8989" in "application.properties".



Comment | More info | Advertise with us