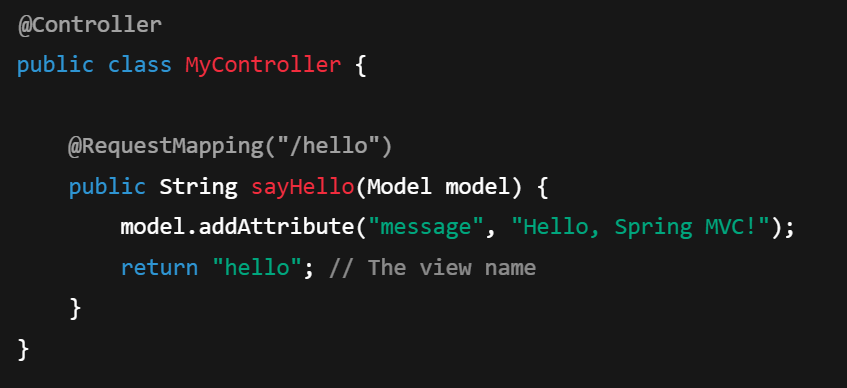
What is the difference between @Controller and @RestController in Spring MVC?

In Spring MVC, both @Controller and @RestController are used to define Spring MVC controllers, but they serve different purposes and have different behaviors when handling HTTP requests.

**1. @Controller**

* **Purpose**: @Controller is used to define a controller class that handles web requests and returns **views** (HTML, JSP, etc.) as responses, typically for traditional **server-side rendering** applications.
* **Response Type**: The methods inside @Controller typically return **view names** (like "home", "login") or a **model object** (like ModelAndView).



**2. @RestController**

* **Purpose**: @RestController is a specialized version of @Controller that is used to create **RESTful web services**. It combines @Controller and @ResponseBody annotations.
* **Response Type**: Unlike @Controller, methods in a @RestController return **response bodies** directly (e.g., JSON or XML) instead of view names. The returned objects are automatically serialized into the specified response format (like JSON).

A screen shot of a computer program

AI-generated content may be incorrect.

The @EnableAutoConfiguration annotation in Spring Boot plays a crucial role in **automatic configuration** of Spring Beans based on the libraries present in the classpath and the application's environment. It is a core feature of Spring Boot that helps reduce the need for manual configuration, enabling **convention over configuration**.

In **Spring Boot**, a **starter** is a **pre-configured set of dependencies** that you can add to your project to easily set up a specific functionality or feature in your application. Starters are part of the Spring Boot ecosystem, and they simplify the process of integrating commonly used libraries and technologies by providing a convenient way to include all necessary dependencies in a single dependency.

**@RequestMapping:** Maps HTTP requests to handler methods. It is used to define the URL patterns and HTTP methods (GET, POST, etc.) for the methods in the controller.

**@ResponseBody**: Tells Spring to write the return value of a method directly to the response body, rather than interpreting it as a view name. It is commonly used in REST APIs to send serialized objects (e.g., JSON or XML) as the response.

**An idempotent API is** an API that can be called multiple times with the same parameters and always produces the same result. This means that making the same request multiple times will not cause any side effects beyond the first request. The state of the system remains consistent and unchanged after repeated calls with identical inputs.

**Example of Idempotency:**

For instance, a **DELETE** operation on a resource (such as deleting a user) should be idempotent. If you call it once, the user is deleted. If you call it again, it won't cause any error or change because the user is already deleted, and the system remains in the same state.

REST APIs: RequestMapping, PathVariable vs RequestParams

In REST APIs, **RequestMapping**, **PathVariable**, and **RequestParams** are commonly used annotations or concepts in frameworks like Spring to handle incoming HTTP requests. Let's break them down:

**1. @RequestMapping (or specialized mapping annotations like @GetMapping, @PostMapping)**

@RequestMapping is an annotation used to map HTTP requests to handler methods of MVC and REST controllers in frameworks like Spring. It is used to define the URL pattern, HTTP method type, and other conditions for handling requests.

* **@RequestMapping** is a general-purpose annotation. It can be used with different HTTP methods like GET, POST, PUT, DELETE, etc.
* Specialized variants such as **@GetMapping**, **@PostMapping**, **@PutMapping**, and **@DeleteMapping** are used to make the code more readable and to directly indicate the HTTP method for the request.

@RestController

public class MyController {

@RequestMapping("/api/hello") // General mapping

public String hello() {

return "Hello, World!";

}

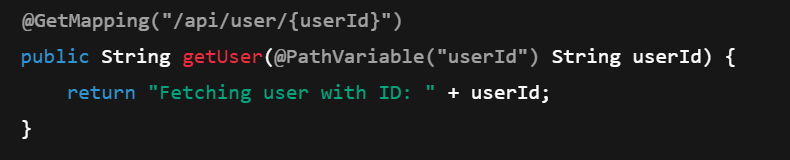
@GetMapping("/api/user/{id}") // Get request with dynamic path parameter

public String getUser(@PathVariable String id) {

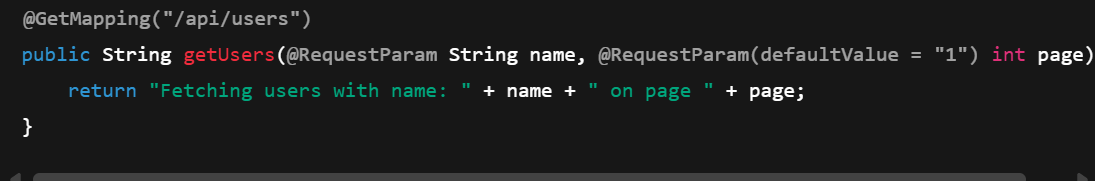
return "User ID: " + id;

}

**@PathVariable** is an annotation used to extract values from the URL path and pass them to the controller method as method parameters. It is typically used for **dynamic URL patterns** where the value can vary.



**@RequestParam** is used to extract values from the query string (i.e., parameters in the URL following the ?). It is commonly used for handling **optional parameters** or filtering and pagination in RESTful services.



Statelessness in REST APIs

**Statelessness** is one of the core principles of REST (Representational State Transfer) and is a key design concept for RESTful APIs. It means that each **HTTP request** from a client to a server must be independent and contain all the necessary information to understand and process the request. The server **does not store any state** about the client between requests.