Palash Bajpai

Spring Boot

Interview Questions

Table of Contents

[Q1. What is RestController and how it is different from controller? 2](#_Toc197068520)

[Q2. How to show html page via controller? 3](#_Toc197068521)

<https://codefarm0.medium.com/300-spring-boot-interview-questions-43393e1c2bef>

## Q1. What is the role of @SpringBootApplication annotation?

**@SpringBootApplication** is a convenience annotation that combines @Configuration, @EnableAutoConfiguration, and @ComponentScan to set up a Spring Boot application. It enables automatic configuration and component scanning, simplifying project setup.

Key roles:

**Auto-Configuration Enabler**

* Includes @EnableAutoConfiguration
* Automatically configures Spring beans based on the classpath.

**Component Scanning**

* Includes @ComponentScan
* Automatically scans the current package and sub-packages for components (@Component, @Service, @Repository, @Controller, etc.).

**Configuration Class**

* Includes @Configuration
* Marks the class as a source of bean definitions (@Bean methods allowed here).

**Bootstraps the Application**

* Used with SpringApplication.run(...)
* Starts the embedded server (like Tomcat) and initializes the Spring context.

**Simplifies Setup**

* Instead of writing 3 separate annotations (@Configuration, @EnableAutoConfiguration, and @ComponentScan), you just write one.

## Q2. What Are Spring Boot Starter Dependencies?

**Spring Boot Starter Dependencies** are a set of convenient dependency descriptors provided by Spring Boot that you can include in your application. They group commonly used dependencies into a single dependency, making it easier and faster to get started with Spring-based applications.

Spring Boot starters are **Maven POMs** that include transitive dependencies. Without starters, you’d have to manually add 5–10+ dependencies just to start using something like Spring MVC or Spring Data JPA. With starters, you just add **one** dependency like spring-boot-starter-web.

Starter dependencies are named with the prefix spring-boot-starter-\*. Examples:

| **Starter** | **Purpose** |
| --- | --- |
| spring-boot-starter-web | REST APIs |
| spring-boot-starter-data-jpa | JPA with Hibernate |
| spring-boot-starter-test | Testing |
| spring-boot-starter-security | Security layer |

They simplify Maven/Gradle dependencies.

## Q3. What is Auto-Configuration in Spring Boot?

**Spring Boot Auto-Configuration** tries to automatically configure your application with sensible defaults based on the libraries you have added and the environment you're in — so you don’t have to write a lot of boilerplate code. Eg:

If you add spring-boot-started-web:

Spring Boot:

* Detects spring-webmvc is present
* Auto-configures:
  + DispatcherServlet
  + RequestMappingHandlerMapping
  + Tomcat server
  + JSON converters (like Jackson)

You don’t need to define them yourself.

## Q4. What happens if I define my own bean that conflicts with auto-configuration?

Your bean takes precedence. Spring Boot respects user-defined beans and disables auto-config for that bean. We can define our own beans with @Bean, use exclude in @EnableAutoConfiguration, or configure via application.properties.

## Q1. What is RestController and how it is different from controller?

@RestController is a **convenience annotation** that is equal to:

@Controller + @ResponseBody

It tells Spring:

* This class is a controller.
* All methods inside will return data **directly in the response body** (usually as JSON) instead of views.

**@Controller**

* @Controller is used to define a Spring MVC Controller class.
* It's a specialization of @Component, so Spring will detect it during component scanning and manage it as a Spring Bean.
* Used to handle web requests.
* Typically returns views (JSP, Thymeleaf, etc.) using a ModelAndView or by returning a String (view name).

**@ResponseBody**

@ResponseBody is a Spring annotation used to bind the return value of a method directly to the HTTP response body, rather than rendering a view (like a JSP or HTML page).

Without @ResponseBody: If you return a String in a controller method, Spring assumes it's the name of a view (like home.jsp) to be rendered.

With @ResponseBody: Spring bypasses the view resolver and sends the return value directly in the HTTP response body, usually as JSON (if using Jackson) or XML (if using JAXB), depending on the configured message converters.

With RestController

@RestController  
public class HelloController {  
 @GetMapping("/hello")  
 public String hello(){  
 return " Hello World";  
 }  
}

Equivalent code with just Controller is

@Controller  
public class HelloController {  
 @GetMapping("/hello")  
 @ResponseBody  
 public String hello(){  
 return " Hello World";  
 }  
}

## Q2. How to show html page via controller?

If you're using @Controller (not @RestController) and want to return HTML pages, then you’ll be using Thymeleaf or another templating engine like JSP (though JSP is discouraged with Spring Boot). The default behavior for Spring Boot with Thymeleaf is as follows:

1. Add Thymeleaf dependency

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

2. Put your hellohello.html files inside this folder: src/main/resources/templates/

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
 <title>Hello Page</title>  
</head>  
<body>  
<h1>Hello, World!</h1>  
</body>  
</html>

3. Create Controller

@Controller  
public class HelloController {  
 @GetMapping("/hello")  
 public String hello(){  
 return "hello"; // Maps to hello.html in templates folder  
 }  
}

4. To pass data to html page via controller, we can use Model object.

import org.springframework.ui.Model;  
  
@Controller  
public class HelloController {  
 @GetMapping("/hello")  
 public String hello(Model model){  
 model.addAttribute("user","Palash");  
 return "hello"; // Maps to hello.html in templates folder  
 }  
}

in html:

</head>  
<body>  
 <h1 th:text="'Hello, ' + ${user} + '!'">Hello, Guest!</h1>  
</body>  
</html>

* th:text="'Hello, ' + ${username} + '!'"

Renders: Hello, Palash!

* If the model value is missing, the fallback content (e.g. Hello, Guest!) is displayed.

Uri vs url

# 1. Can we run SpringBoot application without @SpringBootApplication

2. Where war/jar is created for spring boot app , that is run by embedded server

3. Complete the JPA(mappings)

3. Profiling

4. Env based configuration

5. What type of application context is used by Spring boot

6. REST Template

7. how to securely save configurations

How SLF4J is better in performance than sout

1. **What is the use of @SpringBootApplication?**  
   It combines @Configuration, @EnableAutoConfiguration, and @ComponentScan.
2. **What is @EnableAutoConfiguration?**  
   It tells Spring Boot to guess and configure beans based on classpath settings.
3. **How does Spring Boot decide what to auto-configure?**  
   It uses spring.factories files and conditional annotations like @ConditionalOnClass.
4. **What is the role of application.properties or application.yml?**  
   It holds configuration values like port, DB credentials, etc.
5. **What is the use of @Value annotation?**  
   Injects values from properties or SpEL into Spring beans.
6. **What is @Profile?**  
   Enables beans only for specified active profiles (e.g., dev, prod).

