**Spring Boot Interview Questions and Answers**

Last Updated : 02 Jan, 2025

Spring Boot is a Java-based framework used to develop stand-alone, production-ready applications with minimal configuration. Introduced by Pivotal in 2014, it simplifies the development of Spring applications by offering embedded servers, auto-configuration, and fast startup. Many top companies, including Accenture, Netflix, Amazon, and Google, rely on it for its performance and ease of development.

In this guide, we provide comprehensive **Spring Boot interview questions** for both freshers and experienced developers. It covers essential topics such as Core Spring concepts, REST APIs, microservices, auto-configuration, embedded servers, monitoring, and error handling, helping you prepare for any Spring Interview Questions with confidence.

**Table of Content**

* [Spring Boot Interview Questions for Freshers](https://www.geeksforgeeks.org/spring-boot-interview-questions-and-answers/#spring-boot-interview-questions-for-freshers)
* [Spring Boot Intermediate Interview Questions](https://www.geeksforgeeks.org/spring-boot-interview-questions-and-answers/#spring-boot-interview-questions-for-intermediate)
* [Spring Boot Interview Questions For Experienced](https://www.geeksforgeeks.org/spring-boot-interview-questions-and-answers/#advanced-spring-boot-interview-questions)
* [Bonus Spring Boot Interview Questions and Answers](https://www.geeksforgeeks.org/spring-boot-interview-questions-and-answers/#bonus-spring-boot-interview-questions-and-answers)

**Spring Boot Interview Questions for Freshers**

**1. What is Spring Boot?**

Spring Boot is built on top of the Spring framework to create stand-alone RESTful web applications with very minimal configuration and there is no need of external servers to run the application because it has embedded servers like Tomcat and Jetty etc.

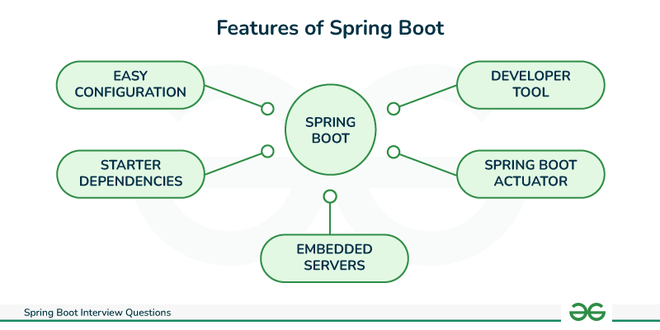
* Spring Boot framework is independent.
* It creates executable spring applications that are production-grade.

**To know more about Spring Boot, refer to this article**– [Introduction to Spring Boot](https://www.geeksforgeeks.org/introduction-to-spring-boot)

**2. What are the Features of Spring Boot?**

There are many useful features of Spring Boot. Some of them are mentioned below:

* **Auto-configuration -**Spring Boot automatically configures dependencies by using **@EnableAutoconfiguration**annotation and reduces boilerplate code.
* **Spring Boot Starter POM**- These Starter POMs are pre-configured dependencies for functions like database, security, maven configuration etc.
* **Spring Boot CLI (Command Line Interface)**- This command line tool is generally for managing dependencies, creating projects and running the applications.
* **Actuator -**Spring Boot Actuator provides health check, metrics and monitors the endpoints of the application. It also simplifies the troubleshooting management.
* **Embedded Servers -**Spring Boot contains embedded servers like Tomcat and Jetty for quick application run. No need of external servers.



**3. What are the advantages of using Spring Boot?**

Spring Boot is a framework that creates stand-alone, production grade Spring based applications. So, this framework has so many advantages.

* **Easy to use:**The majority of the boilerplate code required to create a Spring application is reduced by Spring Boot.
* **Rapid Development:**Spring Boot's opinionated approach and auto-configuration enable developers to quickly develop apps without the need for time-consuming setup, cutting down on development time.
* **Scalable:**Spring Boot apps are intended to be scalable. This implies they may be simply scaled up or down to match your application's needs.
* **Production-ready:**Metrics, health checks, and externalized configuration are just a few of the features that Spring Boot includes and are designed for use in production environments.

**4. Define the Key Components of Spring Boot.**

The key components of Spring Boot are listed below:

* Spring Boot starters
* Auto-configuration
* Spring Boot Actuator
* Spring Boot CLI
* Embedded Servers

**5. Why do we prefer Spring Boot over Spring?**

Here is a table that summarizes why we use Spring Boot over Spring framework.

| **Feature** | **Spring** | **Spring Boot** |
| --- | --- | --- |
| **Ease of use** | More complex | Easier |
| **Production readiness** | Less production-ready | More production-ready |
| **Scalability** | Less scalable | More scalable |
| **Speed** | Slower | Faster |
| **Customization** | Less Customizable | More Customizable |

**To know more, refer to the article –**[**Difference between Spring and Spring Boot**](https://www.geeksforgeeks.org/difference-between-spring-and-spring-boot)

**6. Explain the internal working of Spring Boot.**

**Here are the main steps involved in how Spring Boot works:**

* Start by creating a new Spring Boot project.
* Add the necessary dependencies to your project.
* Annotate the application with the appropriate annotations.
* Run the application.

**To know more about internal working of spring boot application, refer to this article –**[How Spring Boot Application works Internally?](https://www.geeksforgeeks.org/how-spring-boot-application-works-internally)

**7. What are the Spring Boot Starter Dependencies?**

Spring Boot provides many starter dependencies. Some of them which are used the most in the Spring Boot application are listed below:

* Data JPA starter
* Web starter
* Security starter
* Test Starter
* Thymeleaf starter

**8. How does a spring application get started?**

A Spring application gets started by calling the **main()**method with **@SpringBootApplication**annotation in the **SpringApplication**class. This method takes a SpringApplicationBuilder object as a parameter, which is used to configure the application.

* Once the SpringApplication object is created, the **run()**method is called.
* Once the application context is initialized, the run() method starts the application's embedded web server.

**Example:**

**import** **org.springframework.boot.SpringApplication**;

**import** **org.springframework.boot.autoconfigure.SpringBootApplication**;

@SpringBootApplication

**public** **class** **MyApplication**

{

**public** **static** void main(String[] args) {

SpringApplication.run(MyApplication.class, args);

}

}

**9. What does the @SpringBootApplication annotation do internally?**

The **@SpringBootApplication**annotation combines three annotations. Those three annotations are: **@Configuration, @EnableAutoConfiguration,**and **@ComponentScan**.

* **@AutoConfiguration**: This annotation automatically configuring beans in the class path and automatically scans the dependencies according to the application need.
* **@ComponentScan**: This annotation scans the components (@Component, @Service, etc.) in the package of annotated class and its sub-packages.
* **@Configuration:**This annotation configures the beans and packages in the class path.

@SpringBootApplication automatically configures the application based on the dependencies added during project creation and bootstraps the application by using run() method inside the main class of an application.

*@SpringBootApplication = @Configuration + @EnableAutoConfiguration + @ComponentScan*

**10. What is Spring Initializr?**

**Spring Initializer**is a tool that helps us to create skeleton of spring boot project or project structure by providing a maven or gradle file to build the application. It set up the framework from scratch.

**11. What are Spring Boot CLI and the most used CLI commands?**

**Spring Boot CLI**is a command-line tool that can be used to **create, run,**and **manage**Spring Boot applications. It is a powerful tool that can help us to get started with Spring Boot quickly and easily. It is built on top of the Groovy programming language.

Most used **CLI commands**are:

* -run
* -test
* -jar
* -war
* --init
* -help

**Spring Boot Intermediate Interview Questions**

**12. What are the basic Spring Boot Annotations?**

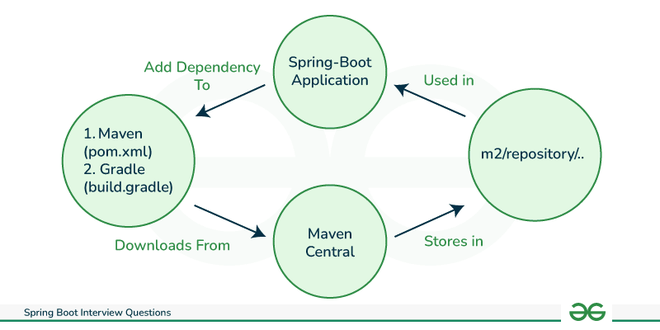
* **@SpringBootApplication:**This is the main annotation used to bootstrap a Spring Boot application. It combines three annotations: **@Configuration**, **@EnableAutoConfiguration**, and **@ComponentScan**. It is typically placed on the main class of the application.
* **@Configuration:**This annotation is used to indicate that a class contains configuration methods for the application context. It is typically used in combination with @Bean annotations to define beans and their dependencies.
* **@Component:**This annotation is the most generic annotation for any Spring-managed component. It is used to mark a class as a Spring bean that will be managed by the Spring container.
* **@RestController:**This annotation is used to define a RESTful web service controller. It is a specialized version of the @Controller annotation that includes the @ResponseBody annotation by default.
* **@RequestMapping:**This annotation is used to map HTTP requests to a specific method in a controller. It can be applied at the class level to define a base URL for all methods in the class, or at the method level to specify a specific URL mapping.

**To know more about Spring Boot Annotations, refer to this article –**[Spring Boot - Annotations](https://www.geeksforgeeks.org/spring-boot-annotations)

**13. What is Spring Boot dependency management?**

**Spring Boot dependency management**makes it easier to manage dependencies in a Spring Boot project. It makes sure that all necessary dependencies are appropriate for the current Spring Boot version and are compatible with it.

*To create a web application, we can add the S* ***pring Boot starter web dependency*** *to our application.*



**To know more about Spring Boot Dependency Management, refer to this article –**[**Spring Boot - Dependency Management**](https://www.geeksforgeeks.org/spring-boot-dependency-management)

**14. Is it possible to change the port of the embedded Tomcat server in Spring Boot?**

Yes, it is possible to change the port of the embedded Tomcat server in a Spring Boot application.

The simple way is to set the **server. port**property in your application's **application.properties**file. For example, to set the port to 8081, add the following property to the application.properties file:

server.port=8081

**15. What is the starter dependency of the Spring boot module?**

**Spring Boot Starters**are a collection of pre-configured maven dependencies that makes it easier to develop particular types of applications. These starters include,

* Dependencies
* Version control
* Configuration needed to make certain features.

To use a **Spring Boot starter dependency**, we simply need to add it to our project's **pom.xml**file. For example, to add the Spring Boot starter web dependency, add the following dependency to the pom.xml file:

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

**To know more about Spring Boot Starters, refer to this article –**[Spring Boot - Starters](https://www.geeksforgeeks.org/spring-boot-starters/?ref=)

**16. What is the default port of Tomcat in spring boot?**

The default port of the embedded Tomcat server in Spring Boot is **8080**. We can change the default port by setting the **server.port**property in your application's **application.properties**file.

**17. Can we disable the default web server in the Spring Boot application?**

Yes, we can disable the default web server in the Spring Boot application. To do this, we need to set the **server.port**property to "-1" in the application's **application.properties**file.

**18. How to disable a specific auto-configuration class?**

To disable a specific auto-configuration class in a Spring Boot application, we can use the **@EnableAutoConfiguration**annotation with the " **exclude"**attribute.

@EnableAutoConfiguration(exclude = {//classname})

**19. Can we create a non-web application in Spring Boot?**

Yes, we can create a non-web application in Spring Boot. Spring Boot is not just for web applications. Using Spring Boot, we can create applications like Microservices, Console applications, and batch applications.

**20. Describe the flow of HTTPS requests through the Spring Boot application.**

The flow of HTTPS requests through a Spring Boot application is as follows:

* First client makes an **HTTP request**( **GET, POST, PUT, DELETE**) to the browser.
* After that the request will go to the controller, where all the requests will be mapped and handled.
* After this in Service layer, all the **business logic**will be performed. It performs the business logic on the data that is mapped to **JPA (Java Persistence API)**using model classes.
* In repository layer, all the **CRUD**operations are being done for the **REST APIs**.
* A **JSP page**is returned to the end users if no errors are there.

**21. Explain @RestController annotation in Spring Boot.**

**@RestController**annotation is like a shortcut to building RESTful services. It combines two annotations:

* **@Controller**: Marks the class as a request handler in the Spring MVC framework.
* **@ResponseBody**: Tells Spring to convert method return values (objects, data) directly into HTTP responses instead of rendering views.

It enables us to Define endpoints for different **HTTP methods (GET, POST, PUT, DELETE),**return data in various formats (JSON, XML, etc.) and map the request parameters to method arguments.

**22. Difference between @Controller and @RestController**

| **Features** | **@Controller** | **@RestController** |
| --- | --- | --- |
| **Usage** | It marks a class as a controller class. | It combines two annotations i.e. @Controller and @ResponseBody. |
| **Application** | Used for Web applications. | Used for RESTful APIs. |
| **Request handling and Mapping** | Used with @RequestMapping annotation to map HTTP requests with methods. | Used to handle requests like GET, PUT, POST, and DELETE. |

*Note: Both annotations handle requests, but @RestController prioritizes data responses for building API.*

**23. What is the difference between RequestMapping and GetMapping?**

| **Features** | **@RequestMapping** | **@GetMapping** |
| --- | --- | --- |
| **Annotations** | @RequestMapping | @GetMapping |
| **Purpose** | Handles various types of HTTP requests (GET, POST, etc.) | Specifically handles HTTP GET requests. |
| **Example** | @RequestMapping(value = "/example", method = RequestMethod.GET) | @GetMapping("/example") |

**24. What are the differences between @SpringBootApplication and @EnableAutoConfiguration annotation?**

| **Features** | **@SpringBootApplication** | **@EnableAutoConfiguration** |
| --- | --- | --- |
| **When to use** | When we want to use auto-configuration | When we want to customize auto-configuration |
| **Entry point** | Typically used on the main class of a Spring Boot application, serving as the entry point. | Can be used on any configuration class or in conjunction with @SpringBootApplication. |
| **Component Scanning** | Includes **@ComponentScan**annotation to enable component scanning. | Does not perform component scanning by itself. |
| **Example** | @SpringBootApplication public class MyApplication { public static void main(String[] args) { SpringApplication.run(MyApplication.class, args); } } | @Configuration @EnableAutoConfiguration public class MyConfiguration { } |

**25. What are Profiles in Spring?**

**Spring Profiles**are like different scenarios for the application depending on the environment.

* You define sets of configurations (like database URLs) for different situations (development, testing, production).
* Use the **@Profile**annotation to clarify which config belongs to where.
* Activate profiles with **environment variables**or **command-line**options.

To use Spring Profiles, we simply need to define the **spring.profiles.active**property to specify which profile we want to use.

**26. Mention the differences between WAR and embedded containers.**

| **Feature** | **WAR** | **Embedded containers** |
| --- | --- | --- |
| **Packaging** | Contains all of the files needed to deploy a web application to a web server. | It is a web application server included in the same JAR file as the application code. |
| **Configuration** | Requires external configuration files (e.g., web.xml, context.xml) to define the web application. | Uses configuration properties or annotations within the application code. |
| **Security** | Can be deployed to a web server that is configured with security features. | Can be made more secure by using security features that are provided by JRE. |

**Spring Boot Interview Questions For Experienced**

**27. What is Spring Boot Actuator?**

**Spring Boot Actuator**is a component of the Spring Boot framework that provides production-ready operational monitoring and management capabilities. We can manage and monitor your Spring Boot application while it is running.

*Note: To use Spring Boot Actuator, we simply need to add the* ***spring-boot-starter-actuator*** *dependency to our project.*

**To know more about Actuator, refer to this article –**[Spring Boot Actuator](https://www.geeksforgeeks.org/spring-boot-actuator/?ref=)

**28. How to enable Actuator in the Spring boot application?**

Below are the steps to enable actuator in Spring Boot Application:

* Add Actuator dependency.
* Enable endpoints in application.properties.
* Run your Spring Boot app.

Now we can access Actuator endpoints at URLs on the management port.

**29. What is the purpose of using @ComponentScan in the class files?**

**@ComponentScan**annotation is used to tell Spring to scan a package and automatically detect Spring components, configurations, and services to configure. The @ComponentScan annotation can be used in the following ways:

* **Without arguments**
* **With basePackageClasses**
* **With basePackages**

**To know more about @ComponentScan annotation, refer to this article –**[Spring @ComponentScan Annotation with Example](https://www.geeksforgeeks.org/spring-componentscan-annotation-with-example/?ref=)

**30. What are the @RequestMapping and @RestController annotations in Spring Boot used for?**

**@RequestMapping:**@RequestMapping is used to map HTTP requests to handler methods in your controller classes. It can be used at the class level and method level. It supports mapping by:

* HTTP method - GET, POST, PUT, DELETE
* URL path
* URL parameters
* Request headers

**@RestController:**@RestController is a convenience annotation that combines **@Controller**and **@ResponseBody**. It indicates a controller where every method returns a domain object instead of a view.

*@RestController = @Controller + @ResponseBody*

**31. How to get the list of all the beans in your Spring boot application?**

* Using the **ApplicationContext**object in Spring Boot, we can retrieve a list of all the beans in our application.
* The ApplicationContext is responsible for managing the beans and their dependencies.

**32. Can we check the environment properties in your Spring boot application explain how?**

Yes, we can check the environment properties in our Spring Boot Application. The Environment object in a Spring Boot application can be used to check the environment's properties.

Configuration settings for the application, includes:

* property files
* command-line arguments
* environment variables

We can get the Environment instance by calling the **getEnvironment()**method.

**33. How to enable debugging log in the spring boot application?**

To enable debugging log in Spring Boot Application, follow the below steps:

* **Add the logging level property to application.properties.**
* **Configure the log pattern to include useful information.**
* **Run the Spring Boot application.**

Using the actuator endpoint, the log level can also be changed at runtime.

Curl -X POST \http://localhost:8080/actuator/loggers/<logger-name>   
\ -H 'content-type: application/json' \-d '{"configuredLevel": "DEBUG"}'

**34. What is dependency Injection and its types?**

**Dependency Injection**(DI) is a design pattern that enables us to produce loosely coupled components. In DI, an object's ability to complete a task depends on another object. There three types of dependency Injections.

* **Constructor injection:**This is the most common type of DI in Spring Boot. In constructor injection, the dependency object is injected into the dependent object's constructor.
* **Setter injection:**In setter injection, the dependency object is injected into the dependent object's setter method.
* **Field injection**: In field injection, the dependency object is injected into the dependent object's field.

**To know more about Dependency Injection, refer to the article –**[Spring Dependency Injection with Example - GeeksforGeeks](https://www.geeksforgeeks.org/spring-dependency-injection-with-example/?ref=)

**35. What is an IOC container?**

An **IoC (Inversion of Control)**Container in Spring Boot is essentially a central manager for the application objects that controls the creation, configuration, and management of dependency injection of objects (often referred to as beans), also referred to as a DI (Dependency Injection) container.

**To know more about IOC Container, refer to the article –**[Spring - IoC Container](https://www.geeksforgeeks.org/spring-ioc-container/?ref=)

**36. What is the difference between Constructor and Setter Injection?**

| **Features** | **Constructor Injection** | **Setter Injection** |
| --- | --- | --- |
| **Dependency** | Dependencies are provided through constructor parameters. | Dependencies are set through setter methods after object creation. |
| **Immutability** | Promotes immutability as dependencies are set at creation. | Dependencies can be changed dynamically after object creation. |
| **Dependency Overriding** | Harder to override dependencies with different implementations. | Allows easier overriding of dependencies using different setter values. |

**Bonus Spring Boot Interview Questions and Answers**

**1. What is Thymeleaf?**

**Thymeleaf**is a Java-based server-side **template engine**used in Java web applications to render dynamic web pages. It is a popular choice for server-side templating in the Spring ecosystem, including Spring Boot.

**To know more about Thymeleaf, refer to this article -**[Spring Boot - Thymeleaf with Example](https://www.geeksforgeeks.org/spring-boot-thymeleaf-with-example/?ref=)

**2. Explain Spring Data and What is Data JPA?**

**Spring Data**is a powerful framework that can be used to develop data-oriented applications. It aims to simplify the development of data-centric applications by offering abstractions, utilities, and integration with various data sources.

* **Spring Data JPA:**This project provides support for accessing data from relational databases using JPA.

**3. Explain Spring MVC**

**MVC**stands for **Model, View,**and **Controller. Spring MVC**is a web MVC framework built on top of the Spring Framework. It provides a comprehensive programming model for building web applications.

**4. What is Spring Bean?**

An object that is managed by the Spring IoC container is referred to as a spring bean. A Spring bean can be any Java object.

**5. What are Inner Beans in Spring?**

An **Inner Bean**refers to a bean that is defined within the scope of another bean's definition. It is a way to declare a bean inside the configuration of another bean, without explicitly giving it a unique identifier.

To define an Inner Bean in Spring, we can declare it as a nested <bean> element within the configuration of the enclosing bean.

**6. What is Bean Wiring?**

**Bean wiring**is a mechanism in Spring that is used to manage the dependencies between beans. It allows Spring to inject collaborating beans into each other. There are two types of Bean Wiring:

* Autowiring
* Manual wiring

**To know more about Autowiring, refer to the article –**[Spring - Autowiring](https://www.geeksforgeeks.org/spring-autowiring/?ref=)

**7. What Are Spring Boot DevTools Used For?**

**Spring Boot DevTools**provides a number of development-time features and enhancements to increase developers' productivity and can be used for the following purposes:

* Automatic application restart
* Fast application startup:
* Actuator endpoints
* Additional development utilities

**To know more about Spring Boot DevTools, refer to the article –**[Spring Boot - DevTools](https://www.geeksforgeeks.org/spring-boot-devtools/?ref=)

**8. What error do you see if H2 is not present in the class path?**

Below is the error we see if H2 is not present in the class path:

java.lang.ClassNotFoundException: org.h2.Driver

**9. Mention the steps to connect the Spring Boot application to a database using JDBC.**

To connect an external database like MySQL or Oracle to a Spring Boot application using JDBC, we need to follow below steps:

* Add the dependency for the JDBC driver of the database.
* Create an application.properties file.
* Configure the database connection properties.
* Create a JdbcTemplate bean.
* Use the JdbcTemplate bean to execute SQL queries and statements.

**To know more, refer to this article –**[Spring Boot - CRUD Operations using MySQL Database](https://www.geeksforgeeks.org/spring-boot-crud-operations-using-mysql-database/?ref=)

**10. Mention the advantages of the YAML file over than Properties file and the different ways to load the YAML file in Spring boot.**

Advantages of YAML file over Properties file:

* Easy to edit and modify.
* Conciseness
* Supports Complex data types.

Different ways to load YAML file in Spring Boot:

* Using the @ConfigurationProperties annotation
* Using the YamlPropertiesFactoryBean class

**11. What Do you understand about Spring Data Rest?**

**Spring Data REST**is a framework that exposes Spring Data repositories as RESTful web services. It allows us to expose repositories as REST endpoints with minimal configuration by following Spring Data REST Technologies like **Spring Data**and **Spring MVC**.

**To know more about Spring Data REST, Please Refer to this article-**[Spring - REST Controller](https://www.geeksforgeeks.org/spring-rest-controller/?ref=)

**12. Why is Spring Data REST not recommended in real-world applications?**

Here are the reasons why not to choose Spring Data REST:

* **Performance**- Performance may not be optimal for very large-scale applications.
* **Versioning**- It can be difficult to version the REST APIs exposed by Spring Data REST.
* **Relationships**- Handling relationships between entities can be tricky with Spring Data REST.
* **Filtering**- There are limited options for filtering the results returned by the endpoints.

**13. How is Hibernate chosen as the default implementation for JPA without any configuration?**

Spring Boot automatically configures **Hibernate**as the default JPA implementation when we add the **spring-boot-starter-data-jpa**dependency to our project. This dependency includes the Hibernate JAR file as well as the Spring Boot auto-configuration for JPA.

**To know more about Hibernate and JPA, Refer to below articles:**

* [Hibernate Architecture](https://www.geeksforgeeks.org/hibernate-architecture/?ref=)
* [Java - JPA vs Hibernate](https://www.geeksforgeeks.org/java-jpa-vs-hibernate/?ref=)

**14. Explain how to deploy to a different server with Spring Boot?**

Below are the steps on how to deploy to a different server with Spring Boot:

* Step 1: **Build your Spring Boot application.**
* Step 2: **Create a deployment package.**
* Step 3: **Deploy the deployment package to the server.**
* Step 4: **Start the server.**