

Spring Boot

Spring Boot Interview Questions – FAQs

What will be the Spring Boot Interview Questions for 5 Years Experience?

In the interview, candidates with over 5 years of experience are primarily questioned about these concepts.

1. **Auto-configuration:** Spring Boot automatically configures beans based on project dependencies, saving time in setup.
2. **Starters :** Dependency management artifacts for easy integration of common Spring Boot features like web apps, data access, and security.
3. **Production-ready applications:** Spring Boot provides embedded servers, actuators, and metrics for creating production-ready apps.
4. **Best practices:** Use dependency injection, version control, and thorough testing when developing Spring Boot apps.
5. **Challenges :** Understanding auto-configuration and selecting appropriate dependencies may be challenging.
6. **Improving skills :** Enhance Spring Boot skills through documentation, conferences, and contributing to the project.

What are the most common Spring Boot interview questions?

The most common Spring Boot interview questions are:

- What is Spring Boot?
- What are the advantages of using Spring Boot?
- What are the features of Spring Boot?
- How to create a Spring Boot application?
- What is the difference between Spring Boot and Spring Framework?
- What are the starter dependencies in Spring Boot?
- What is the purpose of the `@SpringBootApplication` annotation?
- What is the purpose of the `@Configuration` annotation?
- What is the purpose of the `@Bean` annotation?
- What is the purpose of the `@Autowired` annotation?
- What is the purpose of the `@Value` annotation?
- What is the purpose of the `@Profile` annotation?
- What is the purpose of the `@EnableAutoConfiguration` annotation?

- *What is the default port of the embedded Tomcat server in Spring Boot?*
- *How to change the port of the embedded Tomcat server in Spring Boot?*
- *How to enable actuator in Spring Boot?*
- *How to access actuator endpoints in Spring Boot?*

How can I prepare for Spring Boot interview questions?

There are a few things you can do to prepare for Spring Boot interview questions:

- *Learn about Spring Boot*
- *Practice answering common Spring Boot interview questions*
- *Create a Spring Boot project and experiment with the different features*
- *Attend Spring Boot meetups and conferences*
- *Join the Spring Boot community on Stack Overflow and other forums*

What will be the Spring Boot Interview Questions for 2 Years Experience?

For candidates with up to 2 years of experience, interviews will typically focus on the core concepts of Spring Boot, such as auto-configuration, starters, actuator, and CLI. Questions may also be asked about how to create, configure, run, and deploy Spring Boot applications.

- *Basics of Spring Boot*
- *Components of Spring Boot*
- *Create a Spring Boot application*
- *Configure Spring Boot application*
- *Run a Spring Boot application*
- *Deploy a Spring Boot application*
- *Best practices for developing Spring Boot applications*

What will be the Spring Boot Interview Questions for 3 Years Experience?

In the interview, candidates with over 3 years of experience are primarily questioned about these concepts.

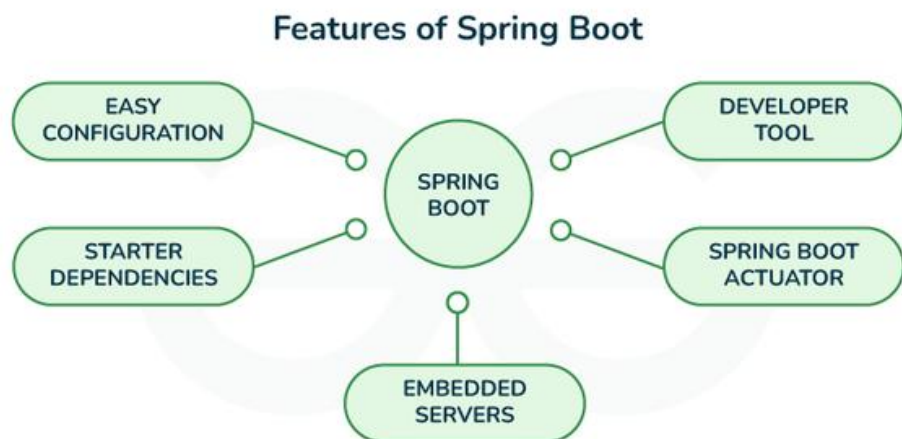
- *What are the different ways to start a Spring Boot application?*
- *What are the different ways to configure Spring Boot applications?*
- *How to use Spring Boot starters?*
- *How to use Spring Boot actuator?*
- *How to use Spring Boot CLI?*
- *How to use Spring Boot in a microservices architecture?*
- *How to secure a Spring Boot application?*
- *How to troubleshoot Spring Boot applications?*

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.

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?

<u>Feature</u>	<u>Spring</u>	<u>Spring Boot</u>
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

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.

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.

```
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.
 - $\text{@SpringBootApplication} = \text{@Configuration} + \text{@EnableAutoConfiguration} + \text{@ComponentScan}$

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?

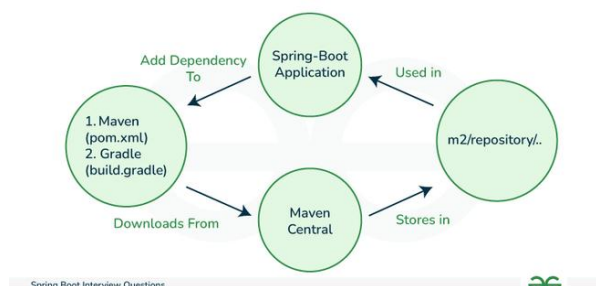
- **Spring Boot CLI** is a command-line tool that can be used to **create**, **run**, and **manage** Spring Boot applications. It is a powerful tool that can help us to get started with Spring Boot quickly and easily. It is built on top of the Groovy programming language.
- Most used **CLI commands**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

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.

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.



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>
```

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.

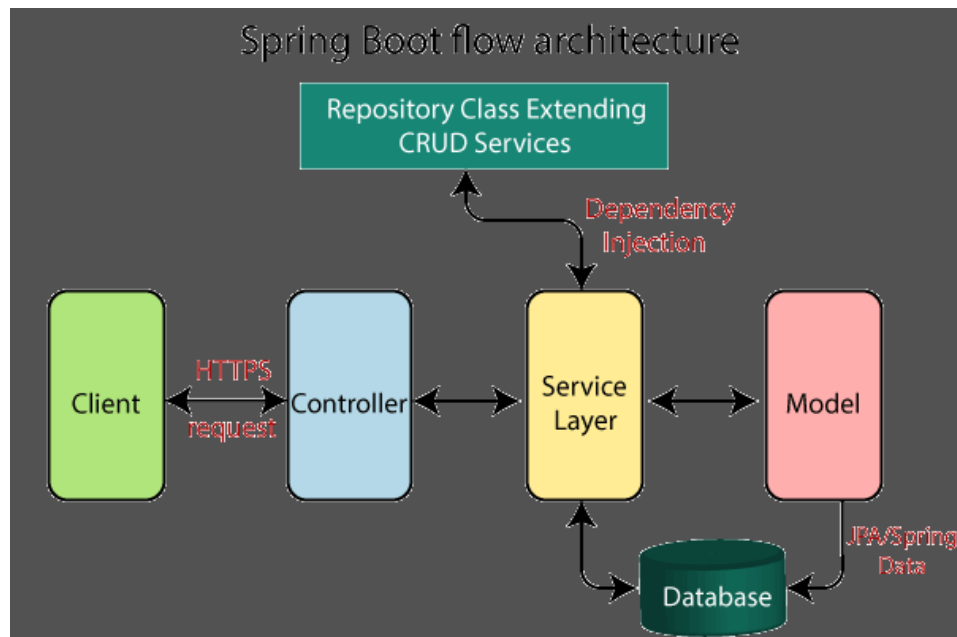
```
@EnableAutoConfiguration(exclude = { //classname })
```

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.

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	<code>@SpringBootApplication</code>	<code>@EnableAutoConfiguration</code>
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 <code>@SpringBootApplication</code> .
Component Scanning	Includes <code>@ComponentScan</code> annotation to enable component scanning.	Does not perform component scanning by itself.
Example	<pre><code>@SpringBootApplication public class MyApplication { public static void main(String[] args) { SpringApplication.run(MyApplicati on.class, args); } }</code></pre>	<pre><code>@Configuration @EnableAutoConfiguration public class MyConfiguration { }</code></pre>

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.

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**

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 are 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.

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.

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.

37. 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.

38. 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.

39. 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.

40. 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.

41. 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

42. 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

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

- `java.lang.ClassNotFoundException: org.h2.Driver`

44. 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.

45. 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

46. 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**.

47. 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.

48. 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.

49. 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.**

1. What CSRF ?

- **Cross-Site Request Forgery** is a web security vulnerability that allows an attacker to induce users to perform actions that they do not intend to perform. It allows an attacker to partly circumvent the same origin policy, which is designed to prevent different websites from interfering with each other.

2. What are the common defence against CSRF ?

- **CSRF tokens** - A CSRF token is a unique, secret, and unpredictable value that is generated by the server-side application and shared with the client. When attempting to perform a sensitive action, such as submitting a form, the client must include the correct CSRF token in the request. This makes it very difficult for an attacker to construct a valid request on behalf of the victim.
- **SameSite cookies** - SameSite is a browser security mechanism that determines when a website's cookies are included in requests originating from other websites. As requests to perform sensitive actions typically require an authenticated session cookie, the appropriate SameSite restrictions may prevent an attacker from triggering these actions cross-site. Since 2021, Chrome enforces Lax SameSite restrictions by default. As this is the proposed standard, we expect other major browsers to adopt this behavior in future.
- **Referer-based validation** - Some applications make use of the HTTP Referer header to attempt to defend against CSRF attacks, normally by verifying that the request originated from the application's own domain. This is generally less effective than CSRF token validation.

3. What is XSS ?

- **Cross-site scripting** (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. It allows an attacker to circumvent the same origin policy, which is designed to segregate different websites from each other. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data.

4. How to prevent XSS attack ?

- Preventing cross-site scripting is trivial in some cases but can be much harder depending on the complexity of the application and the ways it handles user-controllable data.
- In general, effectively preventing XSS vulnerabilities is likely to involve a combination of the following measures:
 - Filter input on arrival. At the point where user input is received, filter as strictly as possible based on what is expected or valid input.
 - Encode data on output. At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding.
 - Use appropriate response headers. To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type-Options headers to ensure that browsers interpret the responses in the way you intend.
 - Content Security Policy. As a last line of defence, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur.

5. What is the difference between XSS and CSRF?

- [Cross-site scripting](#) (or XSS) allows an attacker to execute arbitrary JavaScript within the browser of a victim user.
- [Cross-site request forgery](#) (or CSRF) allows an attacker to induce a victim user to perform actions that they do not intend to.
- The consequences of XSS vulnerabilities are generally more serious than for CSRF vulnerabilities:
- CSRF often only applies to a subset of actions that a user is able to perform. Many applications implement CSRF defences in general but overlook one or two actions that are left exposed. Conversely, a successful XSS exploit can normally induce a user to perform any action that the user is able to perform, regardless of the functionality in which the vulnerability arises.
- CSRF can be described as a "one-way" vulnerability, in that while an attacker can induce the victim to issue an HTTP request, they cannot retrieve the response from that request. Conversely, XSS is "two-way", in that the attacker's injected script can issue arbitrary requests, read the responses, and exfiltrate data to an external domain of the attacker's choosing.

6. How To work with JWT ?

- To work with JWT we need two dependencies
 - jjwt api
 - jjwt impl
 - jjwt Jackson (JJWT :: Extensions :: Jackson).
 - **The version have to be same version.**
-