

Spring Boot Interview Questions

1. What is Spring Boot?

- 1) Spring Boot is a **Spring module** that provides the RAD(**Rapid Application Development**) feature to the Spring framework
- 2) Spring Boot can be used to create **Web-Application**, **Enterprise Application**, **Rest-APIs**
- 3) It is used to create a **stand-alone**, production grade Spring-based application that **you can just run**. It needs minimal Spring configuration

2. What are the Features of Spring Boot?

- 1) Spring Boot provides **dependency injection** used to **achieve the loose coupling**(**@Autowired**)
Independent components (*Implementing an interface is also a way of achieving loose coupling*)
- 2) Spring Boot contains powerful **transaction Management** (payment)
- 3) Spring Boot simplifies **integration** of other java **frameworks** JPA/Hibernate (**ORM tools**), struts
- ~~5) **Embedded servers** Tomcat, Jetty or Undertow directly (*no need to deploy WAR files*)~~
- ~~6) Provide opinionated '**starter**' dependencies to simplify your build configuration~~
- ~~7) **Automatically configure** Spring and 3rd party **libraries** whenever possible(*dispatcher servlet*)~~
- ~~8) Provide **production ready features** such as metrics, health checks, and externalized configuration~~
- ~~9) It provides annotation based configuration, no need to write code for XML configuration~~
- 4) It **reduces cost** and **development time**

3. What are the advantages of using Spring Boot?

- 1) Spring Boot is used to create **standalone** applications
- 2) Spring Boot is used to easily test the application **embedded Servers** like tomcat, jetty, and undertow
- 3) There is **no need** to write **xml configuration**. (*Can use annotation based configuration*)
- 4) It **reduces** the amount of **boilerplate code**
- 5) Spring Boot provides **Production ready** features **health check**, **metrics** and externalized configuration
- 6) Spring Boot provides **starter POM.xml** (Project Object Model) to simplify the **maven configuration**
- 7) It offers a **maven** plugin
- 8) Spring Boot provides **auto-configuration**(Dispatcher servlet & View resolver)

4. What is an IOC container?

- 1) IOC stands for **Inversion of control**.
- 2) IOC container **creates the object**(bean), manages the entire **bean life cycle**, and configures the dependency management.
- 3) IOC container **configures Dependency injection** to manage their components.

Tasks of IOC container:

- i) **Instantiating beans**: **Creating objects** based on your configuration (XML or annotations).
- ii) **Configuring beans**: Setting properties and **injecting dependencies into the created objects**.
- iii) **Assembling beans**: Managing relationships between objects, ensuring they are properly wired together.
- iv) **Managing the bean lifecycle**: Providing **methods to initialize** and **destroy beans** when necessary.

5. What is @Component annotation?

- 1) @Component is a **class level** annotation.(*stereotype annotation*)

2) @Component used to make the java class as a **bean class**.

3) If you declare @Component in your class path, spring framework picks it up and makes that class spring bean.

This means that Spring will automatically **create** and **manage an instance of the annotated class** within its application context.

6. What is application context?

The Application Context is the central interface that **provides configuration information** to an application. It acts as a **container for managing beans (objects)** and their dependencies.

7. What is @Autowired annotation?

@Autowired annotation is used for **automatic dependency injection**. This means it tells Spring to automatically resolve and inject collaborating beans into your class, **eliminating the need for manual configuration**.

@Autowired used to wiring spring bean.

There are three types of injection:

- 1)field injection
- 2)setter injection
- 3)constructor injection

How it works:

👉 Spring looks for a bean in the **application context** that matches the type of the annotated field, **constructor**, **parameter**, or **setter** method argument.

👉 If there are **multiple beans of the same type**, Spring can also use the name of the field or parameter to resolve the dependency.

👉 You can use the **@Qualifier** annotation to further **narrow down the selection** if needed.

8. What is @Scope annotation?

The scope of a bean defines the **life cycle** and **visibility of that bean** in the contexts we use it.

@Scope annotations can only be used on the **concrete bean class** (for @Components) or the **factory method** (for @Bean methods).

Singleton scope:

Only one instance created for a bean in spring IOC container and that **shares all over object calls**.

By default springBoot follows a singleton scope **Eg:** @Scope("singleton")

ProtoType scope:

New instance is created every request and shares the new object for every object call.

A bean with the prototype scope will return a different(new) instance every time it is requested from the container. **Eg:** @Scope("prototype")

9. What is @Configuration annotation?

@Configuration is a **class level** annotation.

@Configuration annotated class is used by spring IOC container as a **source of bean definition method**.

@Configuration indicates it **contains bean definition methods**.

Eg:

@Configuration

```
public class AppConfig {
```

```

    @Bean
    public MyService myService() {
        return new MyServiceImpl();
    }
}

```

10. What is @Bean annotation in Spring Boot?

@Bean annotation is used to **explicitly declare a method as a bean definition**. This means that the method will **create** and configure **an object**, which will then be **managed by the Spring IoC container**.

11. What is the difference between Constructor Injection and Setter Injection?

Setter Injection Dependencies are provided through setter methods. Dependencies are **set after the bean** (object) **is created**, allowing for **flexibility in changing dependencies at runtime**. Allows the object's dependencies to be **changed** after the object is created thus promoting mutability. It is used for optional dependencies and when you need to change dependencies after object creation.

```

public class MyService {
    private MyRepository myRepository;
    @Autowired
    public void setMyRepository(MyRepository myRepository) {
        this.myRepository = myRepository;
    }
}

```

Constructor Injection Dependencies are provided through a class constructor. Dependencies are **injected at the time of bean creation**, ensuring that the bean is always in a valid state. Constructor Injection promotes **immutability** as the dependencies are **set once during object creation** and **cannot be changed later**. Generally considered more **thread-safe** because the state of the object does not change after creation. It is recommended for mandatory dependencies that are required for the bean to function properly.

```

public class MyService {
    private final MyRepository myRepository;
    @Autowired
    public MyService(MyRepository myRepository) {
        this.myRepository = myRepository;
    }
}

```

12. What is Spring Bean?

In Spring Boot, a bean is an object managed by the Spring framework. These beans are created and handled by the Spring IoC (Inversion of Control) container.

13. What is Bean Wiring?

Bean wiring, also known as **autowiring**, is a feature of the Spring Framework that allows the framework to **automatically create and inject dependent bean objects** at runtime. This eliminates the need for manual configuration.

14. Explain the internal working of Spring Boot.

From the run method, the main application context is started which in turn searches for the classes annotated with `@Configuration`, initializes all the declared beans in those configuration classes, and based upon the scope of those beans, stores those beans in JVM, specifically in a space inside JVM which is known as IOC container. After the creation of all the beans, it automatically configures the dispatcher servlet and registers the default handler mappings, messageConverters, and all other basic things.

Basically, spring boot supports three embedded servers:- Tomcat (default), Jetty and Undertow.

15. How does a spring application get started?

16. What does the `@SpringBootApplication` annotation do internally?

It is a class-level annotation. It is used to mark the **main configuration class** of a Spring Boot application. This annotation combines three other annotations: `@Configuration`, `@EnableAutoConfiguration`, and `@ComponentScan`.

1. **@Configuration**: Indicates that the class is a source of bean definitions for the application context.
2. **@EnableAutoConfiguration**: Enables Spring Boot's auto-configuration mechanism, which automatically configures the Spring application based on dependencies present in the classpath.
3. **@ComponentScan**: Tells Spring to scan the specified package and its sub-packages for components to register in the application context.

17. What is Spring Initializr?

Spring Initializr is a **web-based tool** that simplifies the process of creating Spring Boot projects. It allows you to **quickly generate a basic project structure** with all the necessary dependencies and configurations, saving you time and effort in the initial setup phase.

Key features:

- 1) **Project generation**: You can specify details like project name, group ID, artifact ID, packaging type (jar or war), Java version, and Spring Boot version.
- 2) **Dependency management**: Choose from a wide range of Spring Boot starters (dependencies) to include features like web development, data access, security, testing, and more.
- 3) **Easy customization**: You can easily add or remove features by including or excluding dependencies.

18. What is Spring Boot dependency management?

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

20. What is the difference between RequestMapping and GetMapping?

21. What is Spring Boot Actuator?

Spring Boot Actuator is a sub-project of Spring Boot. It adds several production-ready features to **monitor** and **manage** your Spring Boot application. It offers a set of **pre-defined endpoints** that provides various information about the application like:

- 👉 Overall **Health** of the application
- 👉 Metrics like **memory** usage, **CPU** usage
- 👉 Basic information about the application like **Version** and **Build**
- 👉 Displays the environment properties of the application

- 👉 **Loggers** to view and modify the logging levels of the application at runtime
- 👉 Generates a thread dump, which can be useful for debugging performance issues.
- 👉 **Tracks HTTP requests** and **responses**, which can be useful for troubleshooting issues.

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

23. What is dependency Injection and its types?

Dependency Injection is a fundamental aspect of the Spring framework, through which the Spring container “**injects**” **objects into other objects** or “dependencies”. It allows us to achieve **loose coupling**. To develop loosely coupled components.

Official Doc:

Dependency injection (DI) is a process whereby objects define their dependencies (that is, the other objects with which they work) only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is constructed or returned from a factory method. The container then injects those dependencies when it creates the bean. This process is fundamentally the inverse (hence the name, Inversion of Control) of the bean itself controlling the instantiation or location of its dependencies on its own by using direct construction of classes or the Service Locator pattern.

Dependency injection is a pattern we can use to implement IoC, where the control being inverted is setting an object's dependencies. Connecting objects with other objects, or “injecting” objects into other objects, is done by an assembler rather than by the objects themselves.

There are two types:

- 1) Field injection
- 2) Constructor injection

24. Explain @RestController annotation?

- 1) It is a specialized version of the **@Controller** annotation.
 - 2) When a method in a class annotated with **@RestController** returns an object, Spring automatically converts that **object into JSON** or **XML** response using **message converters**.
 - 3) This **eliminates the need for an additional @ResponseBody** annotation on each method.
- @RestController** is a convenience annotation that combines **@Controller** and **@ResponseBody**, making it easier to write **RESTful web services** in Spring MVC.

25. Difference between @RestController and @Controller

Web applications use HTML, CSS, JS to show the data to the user(*client*),
REST API uses the data in the format of JSON and XML.

@Controller is used for traditional Spring MVC controllers that **handle web page requests**. Methods in a **@Controller** class typically return a view name (a string) which is resolved to a JSP or Thymeleaf template to render the HTML response. (This annotation in web applications is used to **map the data or model object to view or template**). It searches for a view or template to map the object into that and show it to the client. You can use **@ResponseBody** annotation on individual methods to directly return **data** (e.g., JSON or XML) instead of a view.

@RestController combines the functionality of **@Controller** and **@ResponseBody**. All methods in a

@RestController class will automatically return data directly in the response body (e.g., JSON, XML) no need for @ResponseBody on each method. Primarily used for building REST APIs that return data to be consumed by other applications.

26. Difference between @RequestMapping and @GetMapping

@RequestMapping and @GetMapping are annotations used to map HTTP requests to specific controller methods.

@RequestMapping is a general-purpose annotation that can be used to map any HTTP method (GET, POST, PUT, DELETE, etc.) to a controller method. It can be applied at both the class and method level. At the class level, it defines a base path for all methods within that controller.

@GetMapping is a specialized annotation that is specifically designed to handle HTTP GET requests. It is essentially a shortcut for @RequestMapping(method = RequestMethod.GET). It is more concise and readable than @RequestMapping when dealing with GET requests

27. What is Hibernate?

Hibernate is a ORM tool/framework. 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.

- 1) Hibernate simplifies the development of Java applications to interact with databases.
- 2) It is an open source, lightweight ORM tool.
- 3) Hibernate simplifies data creation, data manipulation and data access (CRUD operations).
- 4) It is a programming technique that maps objects to the data stored in a database.

28. What is ORM?

ORM means Object Relational Mapping. The Objects in the Object Oriented Programming language are mapped with Relational Databases like MySQL, Oracle DB, etc.,

ORM is a technique in which a metadata descriptor is used to connect Object code to a relational database. Object code is written in Object Oriented Programming language like Java, C++, etc.,

Popular ORM Tools for Java:

- 1) Hibernate
- 2) Apache OpenJPA
- 3) EclipseLink
- 4) Oracle TopLink
- 5) Active JDBC
- 6) Ebean
- 7) Athena
- 8) EJB - Enterprise Java Bean
- 9) Cayenne
- 10) Carbonado
- 11) Data Nucleus
- 12) iBatis
- 13) JDO - Java Data Object
- 14) JPOX
- 15) Kodo
- 16) MyBatis

- 17) Object DB
- 18) Toplink
- 19) Torque
- 20) ORM lite

Advantages of using ORM:

- 1) It speeds up development time for teams.
- 2) Decreases the cost of development.
- 3) Handles the logic required to interact with databases.
- 4) Improves security. ORM tools are built to eliminate the possibility of SQL injection attacks.
- 5) You write less code when using ORM tools than with SQL.

29. What is JPQL?


JPQL is Java Persistence Query Language. It is a Java based query used to perform query operations.


30. What is JPA?

JPA is Java Persistence API. It is a **specification** that contains a **set of rules** and **guidelines** to set interfaces for implementing object-relational mapping. Dynamic and named queries can be included in JPA. It supports polymorphism and inheritance.

Hibernate is an implementation of the Java Persistence API (JPA) specification.

NOTE:

 *If we are using a framework, say Hibernate and if we want to change to some other framework, then we don't need to write a different code for it. JPA provides specifications(rules) for those implementations. It provides the same methods for all frameworks. The frameworks that implement JPA should follow the specifications, thus making it easier for us to implement those code into our project. It reduces the need for learning a new language everytime we switch to a new framework. We can write code in Java for mapping objects to DB fields.*

 *In JPA is an interface that contains methods, the class that implements those interfaces must define those methods. Hibernate in our case contains a class that implements those interfaces in JPA. We can use those methods in our program by implementing that JPA repository into our project.*

31. Explain UserDetailsService and UserDetails in Spring Security.

The **UserDetailsService** is an **interface** used for loading user-specific data.

- 1) It is used for **retrieving user information** from a backend data source, such as a database.
- 2) This **returns an instance** of the *UserDetails* interface.
- 3) It has a **single method** called **loadUserByUsername()**. It takes a username as a parameter, and it returns a UserDetails object (type).
- 4) The **UserDetails** object represents the authenticated user, and it contains all the details such as the username, password, authorities (roles) etc.

32. What is GrantedAuthority?

GrantedAuthority is an **interface** with a single method **getAuthority()**, which returns a String representation of the authority.

A GrantedAuthority represents a **privilege or permission granted to an authenticated user**. It defines

what actions the user is allowed to perform within the application.

The most common implementation is **SimpleGrantedAuthority**, which simply stores the authority as a String.

33. What is AuthenticationConfiguration?

It provides a way to easily access and configure the AuthenticationManager, which is the core component responsible for authenticating user credentials. Its primary purpose is to simplify the setup and customization of authentication in Spring Security applications.

34. What is AuthenticationManager?

It is an **interface** in Spring Security that handles authentication requests. It is a core part of the authentication process, **verifying credentials** and **providing access to users**.

~~This is a Spring Security class that provides a way to access the AuthenticationManager.~~ The **AuthenticationConfiguration** is used **to configure** and **customize** the **AuthenticationManager** to handle authentication processes like login, password verification, etc.

35. What is Maven?

Maven is an open-source **build automation** and **project management tool** that helps developers build, publish, and deploy projects.

Maven uses a **project object model (POM)** and plugins to build projects.

👉 It automates tasks such as **source code compilation**, **dependency management**(*download dependencies*), assembling **binary codes into packages**(*packing jar and war files to deploy*), and **executing test** scripts(*JUnit Tests*).

36. What is DTO?

DTO stands for Data Transfer Object. It's a design pattern used to **encapsulate data** that needs to be transferred between different layers of an application, such as between the controller and service layers, or between the service layer and the persistence layer.

This DTO class would be used to transfer user data between layers, **instead of directly exposing the entity** class.

37. Explain @PreAuthorize and @PostAuthorize annotations.

@PreAuthorize: It is used to secure a method before it is executed.

@PostAuthorize: It is used to secure a method after it is executed.

38. What is a dispatcher servlet?

Dispatcher is the front controller in Spring web applications. It **receives all the incoming requests** and **transfers them to all other components** of the application.

The job of DispatcherServlet is to take an incoming URI and find the right combination of **handlers** (Controller classes) and **views**. Finally, the DispatcherServlet returns the **Response Object back to the client**.

*DispatcherServlet handles an incoming **HttpRequest**, delegates the request, and processes that request according to the configured **HandlerAdapter** interfaces that have been implemented within the Spring*

application along with accompanying annotations specifying [handlers](#), [controller endpoints](#), and [response objects](#).

39. What is a view resolver?

A View Resolver is a component responsible for [mapping](#) view names returned from [controllers](#) to actual [view](#) implementations. It acts as a [bridge between the controller and the view](#) technology used to render the response.

Spring MVC defines the ViewResolver and View interfaces that let you [render models in a browser without tying you to a specific view technology](#). ViewResolver provides a mapping between view names and actual views. View addresses the preparation of data before handing over to a specific view technology.

40.