Spring Boot Interview Questions & Answers

Core Concepts

1. What is Spring Boot?

Spring Boot is a powerful framework that streamlines the development, testing, and deployment of Spring applications. It eliminates boilerplate code and offers automatic configuration features to ease the setup and integration of various development tools. It is ideal for microservices, supporting embedded servers and providing a ready-to-go environment that simplifies deployment processes and improves productivity.

2. What are the Features of Spring Boot?

Key features of Spring Boot include auto-configuration, which automatically configures application components based on libraries present; embedded servers like Tomcat and Jetty to ease deployment; starter kits that bundle dependencies for specific functionalities; Spring Boot Actuator for complete monitoring; and extensive support for cloud environments, simplifying deployment of cloud-native applications.

3. What are the advantages of using Spring Boot?

Spring Boot makes Java application development easier by providing a ready-made framework with built-in servers, so developers don't have to set up everything from scratch. It reduces the amount of code needed, boosts productivity with automatic configurations, and works well with other Spring projects. It also supports creating microservices, has strong security features, and helps with monitoring and managing applications efficiently.

4. Define the Key Components of Spring Boot

The key components of Spring Boot are: Spring Boot Starter Kits that bundle dependencies for specific features; Spring Boot AutoConfiguration that automatically configures applications based on included dependencies; Spring Boot CLI for developing and testing Spring Boot apps from the command line; and Spring Boot Actuator, which provides production-ready features like health checks and metrics.

5. Why do we prefer Spring Boot over Spring?

Spring Boot is preferred over traditional Spring because it requires less manual configuration and setup, offers production-ready features out of the box like embedded servers and metrics, and simplifies dependency management. This makes it easier and faster to create new applications and microservices, reducing the learning curve and development time.

6. Explain the internal working of Spring Boot

Spring Boot works by automatically setting up default configurations based on the tools a project uses. It includes built-in servers like Tomcat to run applications. Special starter packages make it easy to connect with other technologies. Settings can be customized with simple annotations and properties files. The Spring

Application class starts the app, and Spring Boot Actuator offers tools for monitoring and managing it.

7. What are the Spring Boot Starter Dependencies?

Spring Boot Starter dependencies are pre-made packages that help easily add specific features to Spring Boot applications. For example, spring-boot-starter-web helps build web apps, spring-boot-starter-data-jpa helps with databases, and spring-boot-starter-security adds security features. These starters save time by automatically including the necessary libraries and settings.

8. How does a Spring application get started?

A Spring application typically starts by initializing a Spring ApplicationContext, which manages the beans and dependencies. In Spring Boot, this is often triggered by calling SpringApplication.run() in the main method, which sets up the default configuration and starts the embedded server if necessary.

9. What does the @SpringBootApplication annotation do internally?

The @SpringBootApplication annotation is a convenience annotation that combines @Configuration, @EnableAutoConfiguration, and @ComponentScan. This triggers Spring's auto-configuration mechanism to automatically configure the application based on its included dependencies, scans for Spring components, and sets up configuration classes.

10. What is Spring Initializr?

Spring Initializr is a website that helps to start a new Spring Boot project quickly. Developers choose project settings, like dependencies and configurations, using an easy interface. It then creates a ready-to-use project that can be downloaded or imported into a development tool, making it faster and easier to get started.

Spring Core Concepts

11. What is a Spring Bean?

A Spring Bean is an object managed by the Spring framework. The framework creates, configures, and connects these beans, making it easier to manage dependencies and the lifecycle of objects. Beans can be set up using simple annotations or XML files, helping build applications in a more organized and flexible way.

12. What is Auto-wiring?

Auto-wiring in Spring automatically connects beans to their needed dependencies without manual setup. It uses annotations or XML to find and link beans based on their type or name. This makes it easier and faster to develop applications by reducing the amount of code needed for connecting objects.

13. What is ApplicationRunner in Spring Boot?

ApplicationRunner in Spring Boot lets developers run code right after the application starts. A class is created that implements the run method with custom logic. This code runs automatically when the app is ready. It's useful for tasks like setting up data or resources, making it easy to perform actions as soon as the application launches.

14. What is CommandLineRunner in Spring Boot?

CommandLineRunner and ApplicationRunner in Spring Boot both let developers run code after the application starts, but they differ slightly. CommandLineRunner uses a run method with a String array of arguments, while ApplicationRunner uses an ApplicationArguments object for more flexible argument handling.

15. What is Spring Boot CLI and the most used CLI commands?

Spring Boot CLI (Command Line Interface) helps quickly create and run Spring applications using simple scripts. It makes development easier by reducing setup and configuration. Common commands are 'spring init' to start a new project, 'spring run' to run scripts, 'spring test' to run tests, and 'spring install' to add libraries. These commands make building and testing Spring apps faster and simpler.

16. What is Spring Boot dependency management?

Spring Boot dependency management makes it easier to handle the dependencies that a project requires. Instead of manually keeping track of them, Spring Boot manages them automatically. It uses tools like Maven or Gradle to organize these dependencies, ensuring they work well together. This saves developers time and effort, allowing them to focus on writing code without getting bogged down in managing dependencies.

Configuration & Server Settings

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

Yes, it's possible to change the default port of the embedded Tomcat server in Spring Boot by setting the server.port property in the application.properties or application.yml file to the desired port number.

18. What happens if a starter dependency includes conflicting versions of libraries with other dependencies in the project?

If a starter dependency includes conflicting versions of libraries with other dependencies, Spring Boot's dependency management resolves this using "dependency resolution." It ensures that only one version of each library is included in the final application, prioritizing the most compatible version. This helps prevent runtime errors caused by conflicting dependencies and ensures smooth application functioning.

19. What is the default port of Tomcat in Spring Boot?

The default port for Tomcat in Spring Boot is 8080. When a Spring Boot application with an embedded Tomcat server is run, it will, by default, listen for HTTP requests on port 8080 unless configured otherwise.

20. Can we disable the default web server in a Spring Boot application?

Yes, the default web server in a Spring Boot application can be disabled by setting the spring.main.web-application-type property to none in the application.properties or application.yml file. This will result in a non-web application, suitable for messaging or batch processing jobs.

21. How to disable a specific auto-configuration class?

Specific auto-configuration classes in Spring Boot can be disabled by using the exclude attribute of the @EnableAutoConfiguration annotation or by setting the spring.autoconfigure.exclude property in the application.properties or application.yml file.

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

Absolutely, Spring Boot is not limited to web applications. Standalone, non-web applications can be created by disabling the web context. This is done by setting the application type to 'none', which skips the setup of web-specific contexts and configurations.

Web Development

23. Describe the flow of HTTPS requests through a Spring Boot application

In a Spring Boot application, HTTPS requests first pass through the embedded server's security layer, which manages SSL/TLS encryption. Then, the requests are routed to appropriate controllers based on URL mappings. Controllers process the requests, possibly invoking services for business logic, and return responses, which are then encrypted by the SSL/TLS layer before being sent back to the client.

24. Explain @RestController annotation in Spring Boot

The @RestController annotation in Spring Boot is used to create RESTful web controllers. This annotation combines @Controller and @ResponseBody, which means the data returned by each method will be written directly into the response body as JSON or XML, rather than through view resolution.

25. Difference between @Controller and @RestController

The key difference is that @Controller is used to mark classes as Spring MVC Controller and typically return a view. @RestController combines @Controller and @ResponseBody, indicating that all methods assume @ResponseBody by default, returning data instead of a view.

26. What is the difference between RequestMapping and GetMapping?

@RequestMapping is a general annotation that can be used for routing any HTTP method requests (like GET, POST, etc.), requiring explicit specification of the method. @GetMapping is a specialized version of @RequestMapping designed specifically for HTTP GET requests, making the code more readable and concise.

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

The @SpringBootApplication annotation combines @Configuration, @EnableAutoConfiguration, and @ComponentScan annotations. It marks the main class of a Spring Boot application and triggers auto-configuration and component scanning. The @EnableAutoConfiguration specifically enables Spring Boot's auto-configuration mechanism, which automatically configures the application based on jar dependencies. It is included within @SpringBootApplication.

28. How can you programmatically determine which profiles are currently active in a Spring Boot application?

In a Spring Boot application, active profiles can be determined using the Environment interface. By including Environment in code using @Autowired, and then using the getActiveProfiles() method, a list of all active profiles can be obtained as an array of strings.

```
@Autowired
Environment env;
String[] activeProfiles = env.getActiveProfiles();
```

29. Mention the differences between WAR and embedded containers

Traditional WAR deployment requires a standalone servlet container like Tomcat, Jetty, or WildFly. In contrast, Spring Boot with an embedded container allows packaging the application and the container as a single executable JAR file, simplifying deployment and ensuring that environment configurations remain consistent.

Monitoring & Management

30. What is Spring Boot Actuator?

Spring Boot Actuator provides production-ready features to help monitor and manage applications. It includes built-in endpoints that provide vital operational information about the application (like health, metrics, info, dump, env, etc.) which can be exposed via HTTP or JMX.

31. How to enable Actuator in Spring Boot?

To enable Spring Boot Actuator, simply add the spring-boot-starter-actuator dependency to the project's build file. Once added, its endpoints and their visibility properties can be configured through the application properties or YAML configuration file.

32. How to get the list of all the beans in our Spring Boot application?

To list all beans loaded by the Spring ApplicationContext, inject the ApplicationContext into any Spring-managed bean and call the getBeanDefinitionNames() method. This returns a String array containing the names of all beans managed by the context.

33. Can we check the environment properties in our Spring Boot application? Explain how

Yes, environment properties in Spring Boot can be accessed via the Environment interface. Inject the Environment into a bean using the @Autowired annotation and use the getProperty() method to retrieve properties.

```
@Autowired
private Environment env;
String dbUrl = env.getProperty("database.url");
System.out.println("Database URL: " + dbUrl);
```

34. How to enable debugging log in the Spring Boot application?

To enable debugging logs in Spring Boot, set the logging level to DEBUG in the application.properties or application.yml file by adding a line such as logging.level.root=DEBUG. This provides detailed logging output, useful for debugging purposes.

35. Explain the need of dev-tools dependency

The dev-tools dependency in Spring Boot provides features that enhance the development experience. It enables automatic restarts of applications when code changes are detected, which is faster than restarting manually. It also offers additional development-time checks to help catch common mistakes early.

Testing

36. How do you test a Spring Boot application?

To test a Spring Boot application, different tools and annotations are used. For testing the whole application together, @SpringBootTest is used. When testing just a part of the application, like the web layer, @WebMvcTest is used. For testing database interactions, @DataJpaTest is used. JUnit helps check if things are working as expected, and Mockito allows replacing some parts with dummy versions to focus on what's being tested.

37. What is the purpose of unit testing in software development?

Unit testing checks if small parts of a program work as they should. It helps find mistakes early, making them easier to fix and keeping the program running smoothly. This makes the software more reliable and easier to update later.

38. How do JUnit and Mockito facilitate unit testing in Java projects?

JUnit and Mockito help test small parts of Java programs. JUnit checks if each part works correctly, while Mockito creates fake versions of components not being tested. This allows focusing on testing one thing at a time.

39. Explain the difference between @Mock and @InjectMocks in Mockito

In Mockito, @Mock creates a fake version of an object to test it without using the real one. @InjectMocks puts these fake objects into the class being tested. This helps see how the class works with the fakes, ensuring everything fits together correctly.

40. What is the role of @SpringBootTest annotation?

The @SpringBootTest annotation in Spring Boot is used for integration testing. It loads the entire application context to ensure that all components work together as expected. This helps test the application in an environment similar to production, where all parts (like databases and internal services) are active, allowing detection and fixing of integration issues early in development.

Exception Handling & Project Structure

41. How do you handle exceptions in Spring Boot applications?

In Spring Boot, exceptions are handled by creating a class with @ControllerAdvice or @RestControllerAdvice. This class has methods marked with @ExceptionHandler that deal with different types of errors. These methods ensure that when something goes wrong, the application responds in a helpful way, like sending a clear error message or a specific error code.

42. Explain the purpose of the pom.xml file in a Maven project

The pom.xml file in a Maven project tells Maven how to build and manage the project. It lists dependencies (libraries and tools) and instructions (like file locations and build processes). This helps Maven automatically handle tasks like building the project and adding the right libraries, making developers' work easier.

43. How does auto configuration play an important role in Spring Boot applications?

Auto-configuration in Spring Boot makes setting up applications easier by automatically configuring system components. For example, if a database tool is added, it will set up the database connection automatically. This means less time is spent on setup and more on creating actual application features.

44. Can we customize a specific auto-configuration in Spring Boot?

Yes, specific auto-configurations in Spring Boot can be customized. Although Spring Boot automatically configures components based on the environment, these settings can be overridden in application properties or YAML files, or by adding custom configuration beans. The @Conditional annotation can be used to include or exclude certain configurations under specific conditions. This flexibility allows tailoring auto-configuration to better fit specific application needs.

45. How can you disable specific auto-configuration classes in Spring Boot?

Specific auto-configuration classes in Spring Boot can be disabled using the @SpringBootApplication annotation with the exclude attribute. For example, @SpringBootApplication(exclude = {DataSourceAutoConfiguration.class}) disables the DataSourceAutoConfiguration class. Alternatively, the spring.autoconfigure.exclude property in application.properties or application.yml can list classes to exclude.

46. What is the purpose of having a spring-boot-starter-parent?

The spring-boot-starter-parent in a Spring Boot project provides default configurations for Maven. It simplifies dependency management, specifies common properties like Java version, and includes useful plugins. This parent POM ensures consistent versions of dependencies and plugins, reducing manual configuration needs and helping maintain uniformity across Spring Boot projects.

47. How do starters simplify the Maven or Gradle configuration?

Starters simplify Maven or Gradle configuration by bundling common dependencies into a single package. Instead of manually specifying each dependency for a particular feature (like web development or JPA), developers can add a starter (e.g., spring-boot-starter-web), which includes all necessary libraries. This reduces configuration complexity, ensures compatibility, and speeds up the setup process, allowing more focus on coding and less on dependency management.

REST APIs & Documentation

48. How do you create REST APIs?

To create REST APIs in Spring Boot, annotate a class with @RestController and define methods with @GetMapping, @PostMapping, @PutMapping, or @DeleteMapping to handle HTTP requests. Use @RequestBody for input data and @PathVariable or @RequestParam for URL parameters. Implement service logic and return responses as Java objects, which Spring Boot automatically converts to JSON. This setup handles API endpoints for CRUD operations.

49. What is versioning in REST? What are the ways that we can use to implement versioning?

Versioning in REST APIs helps manage changes without breaking existing clients. It allows different versions of the API to exist simultaneously, making it easier for clients to upgrade gradually.

REST APIs can be versioned in several ways: including the version number in the URL (e.g., /api/v1/resource), adding a version parameter (e.g., /api/resource?version=1), using custom headers (e.g., Accept: application/vnd.example.v1+json), or using media types (e.g., application/vnd.example.v1+json).

50. What are the REST API Best practices?

Best practices for REST APIs include using appropriate HTTP methods (GET, POST, PUT, DELETE), keeping requests independent (stateless), naming resources clearly, handling errors consistently with clear messages and status codes, using versioning to manage updates, securing APIs with HTTPS and input validation, and using pagination for large datasets to make responses manageable.

51. What are the uses of ResponseEntity?

ResponseEntity in Spring Boot customizes responses by setting HTTP status codes, adding custom headers, and returning response data as Java objects. This flexibility creates detailed and informative responses. For example, new ResponseEntity<>("Hello, World!", HttpStatus.OK) sends back "Hello, World!" with a status code of 200 OK.

52. What should the delete API method status code be?

The DELETE API method should typically return a status code of 200 OK if the deletion is successful and returns a response body, 204 No Content if the deletion is successful without a response body, or 404 Not Found if the resource to be deleted does not exist.

53. What is Swagger?

Swagger is an open-source framework for designing, building, and documenting REST APIs. It provides tools for creating interactive API documentation, making it easier for developers to understand and interact with the API.

54. How does Swagger help in documenting APIs?

Swagger helps document APIs by providing a user-friendly interface that displays API endpoints, request/response formats, and available parameters. It generates interactive documentation from API definitions, allowing developers to test endpoints directly from the documentation and ensuring accurate, up-to-date API information.

Embedded Servers

55. What servers are provided by Spring Boot and which one is default?

Spring Boot provides several embedded servers, including Tomcat, Jetty, and Undertow. By default, Spring Boot uses Tomcat as the embedded server unless another server is specified.

56. How does Spring Boot decide which embedded server to use if multiple options are available in the classpath?

Spring Boot decides which embedded server to use based on the order of dependencies in the classpath. If multiple server dependencies are present, it selects the first one found. For example, if both Tomcat and Jetty are present, it will use the one that appears first in the dependency list.

57. How can we disable the default server and enable a different one?

To disable the default server and enable a different one in Spring Boot, exclude the default server dependency in the pom.xml or build.gradle file and add the dependency for the desired server. For example, to switch from Tomcat to Jetty, exclude the Tomcat dependency and include the Jetty dependency in the project configuration.

Important Annotations

58. @SpringBootApplication

This annotation combines @Configuration, @EnableAutoConfiguration, and @ComponentScan annotations with their default attributes.

59. @EnableAutoConfiguration

Tells Spring Boot to automatically configure the application based on included libraries, setting up the project with default settings likely to work well for the setup.

60. @Configuration

Marks a class as a source of bean definitions for the application context, allowing it to define and configure beans that Spring manages.

61. @ComponentScan

Tells Spring where to look for components, services, and configurations, automatically discovering and registering beans in specified packages.

62. @Bean

Marks a method in a configuration class to define a bean that Spring manages, handling its lifecycle and dependencies.

63. @Component

Marks a class as a Spring-managed component, allowing automatic detection and registration as a bean in the application context.

64. @Repository

Marks a class as a data access component for database operations, providing benefits like exception translation for easier database access management.

65. @Service

Marks a class as a service layer component for business logic, creating Spring-managed beans for better service organization.

66. Can we use @Component instead of @Repository and @Service? If yes, why use @Repository and @Service?

Yes, @Component can be used instead of @Repository and @Service since all three create Spring beans. However, @Repository and @Service make code clearer by showing each class's purpose. @Repository also helps manage database errors better. These specific annotations make code easier to understand and maintain.

67. @Controller

Marks a class as a web controller handling HTTP requests, defining methods that respond to web requests, show web pages, or return data.

68. @RestController

Marks a class as a RESTful web service controller, combining @Controller and @ResponseBody to automatically return JSON or XML responses.

69. @RequestMapping

Maps HTTP requests to handler methods in controller classes, specifying URL paths and HTTP methods that methods should handle.

70. @Autowired

Enables automatic dependency injection by telling Spring to find and inject required beans into classes, simplifying dependency management.

71. @PathVariable

Extracts values from URI templates and maps them to method parameters, capturing dynamic URL parts for request processing.

72. @RequestParam

Binds method parameters to web request parameters, extracting query parameters, form data, or URL parameters for handler methods.

73. @ResponseBody

Tells controller methods to return results directly as response bodies instead of rendering views, commonly used for REST APIs.

74. @RequestBody

Binds HTTP request bodies to method parameters by converting request bodies into Java objects for handling JSON or XML data.

75. @EnableWebMvc

Activates default Spring MVC configuration, setting up components like view resolvers and message converters for web applications.

76. @EnableAsync

Enables asynchronous method execution, allowing methods to run in background threads to improve performance.

77. @Scheduled

Triggers methods to run at fixed intervals or specific times based on cron expressions, fixed delays, or fixed rates.

78. @EnableScheduling

Enables scheduling capabilities for methods, allowing @Scheduled methods to execute based on time intervals or cron expressions.