Top 50 Spring Boot Interview Questions and Answers

1. What is Spring Boot, and why should I use it?

Spring Boot is a framework built on top of the Spring Framework. It simplifies the setup and development of new Spring applications by providing default configurations and embedded servers, reducing the need for boilerplate code.

2. How do I create a Spring Boot application?

You can create a Spring Boot application using Spring Initializr (start.spring.io), an IDE like IntelliJ IDEA, or by using Spring Boot CLI:

1. Go to Spring Initializr.
2. Select your project settings (e.g., Maven, Java, Spring Boot version).
3. Add necessary dependencies.
4. Generate the project and unzip it.
5. Open the project in your IDE and start coding.

3. What is the main class in a Spring Boot application?

The main class in a Spring Boot application is the entry point and is annotated with `@SpringBootApplication`. It includes the `main` method which launches the application using `SpringApplication.run()`.

```
@SpringBootApplication
public class MyApplication {
    public static void main(String[] args) {
        SpringApplication.run(MyApplication.class, args);
```

```
    }
}
```

4. What does the `@SpringBootApplication` annotation do?

`@SpringBootApplication` is a convenience annotation that combines three annotations: `@Configuration` (marks the class as a source of bean definitions), `@EnableAutoConfiguration` (enables Spring Boot's auto-configuration mechanism), and `@ComponentScan` (scans the package of the annotated class for Spring components).

5. How can you configure properties in a Spring Boot application?

You can configure properties in a Spring Boot application using `application.properties` or `application.yml` files located in the `src/main/resources` directory.

```
# application.properties
server.port=8081
spring.datasource.url=jdbc:mysql://localhost:3306/mydb
```

6. How do you handle exceptions in Spring Boot?

You can handle exceptions in Spring Boot using `@ControllerAdvice` and `@ExceptionHandler` annotations to create a global exception handler.

```
@ControllerAdvice
public class GlobalExceptionHandler {
    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<ErrorResponse>
handleResourceNotFoundException(ResourceNotFoundException ex) {
        ErrorResponse errorResponse = new ErrorResponse("NOT_FOUND",
```

```
ex.getMessage());
        return new ResponseEntity<>(errorResponse,
HttpStatus.NOT_FOUND);
    }
}
```

7. What is Spring Boot Actuator and what are its benefits?

Spring Boot Actuator provides production-ready features such as health checks, metrics, and monitoring for your Spring Boot application. It includes various endpoints that give insights into the application's health and environment.

8. How can you enable and use Actuator endpoints in a Spring Boot application?

Add the Actuator dependency in your `pom.xml` or `build.gradle` file:

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

Configure the endpoints in `application.properties`:

```
management.endpoints.web.exposure.include=health,info
```

9. What are Spring Profiles and how do you use them?

Spring Profiles allow you to segregate parts of your application configuration and make it only available in certain environments. You can activate profiles using the `spring.profiles.active` property.

```
# application-dev.properties
spring.datasource.url=jdbc:mysql://localhost:3306/devdb
```

```
# application-prod.properties
spring.datasource.url=jdbc:mysql://localhost:3306/proddb
```

10. How do you test a Spring Boot application?

Spring Boot supports testing with various tools and annotations like `@SpringBootTest`, `@WebMvcTest`, and `@DataJpaTest`. Use `MockMvc` to test MVC controllers without starting a full HTTP server.

```
@SpringBootTest
public class MyApplicationTests {
    @Test
    void contextLoads() {
    }
}
```

11. How can you secure a Spring Boot application?

You can secure a Spring Boot application using Spring Security. Add the dependency and configure security settings:

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

12. What is a Spring Boot Starter and why is it useful?

Spring Boot Starters are a set of convenient dependency descriptors you can include in your application. They provide a one-stop-shop for all the dependencies you need for a particular feature.

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

13. How can you configure a DataSource in Spring Boot?

You can configure a DataSource by adding properties in the `application.properties` file:

```properties
spring.datasource.url=jdbc:mysql://localhost:3306/mydb
spring.datasource.username=root
spring.datasource.password=secret
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

14. What is Spring Boot DevTools and how does it enhance development?

Spring Boot DevTools provides features to enhance the development experience, such as automatic restarts, live reload, and configurations for faster feedback loops. Add the dependency to your project:

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <optional>true</optional>
</dependency>
```

15. How can you handle different environments in a Spring Boot application?

You can handle different environments using Spring Profiles. Define environment-specific properties files like `application-dev.properties`, `application-prod.properties`, and activate a profile using `spring.profiles.active`.

16. What are the differences between `@Component`, `@Service`, `@Repository`, and `@Controller` annotations?

These annotations are specializations of `@Component`:

- `@Component`: Generic stereotype for any Spring-managed component.
- `@Service`: Specialization for service layer classes.
- `@Repository`: Specialization for persistence layer classes.
- `@Controller`: Specialization for presentation layer (MVC controllers).

17. How can you create a RESTful web service using Spring Boot?

Use `@RestController` and `@RequestMapping` annotations to create REST endpoints.

```
@RestController
@RequestMapping("/api")
public class MyController {

    @GetMapping("/greeting")
    public String greeting() {
        return "Hello, World!";
    }
}
```

18. What is Spring Boot CLI and how is it used?

Spring Boot CLI is a command-line tool that allows you to quickly prototype with Spring. It supports Groovy scripts to write Spring applications.

```
$ spring init --dependencies=web my-app
$ cd my-app
$ spring run MyApp.groovy
```

19. How can you connect to a database using Spring Data JPA?

Add the necessary dependencies and create a repository interface extending `JpaRepository`.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

```
public interface UserRepository extends JpaRepository<User, Long> {
}
```

20. How can you use the H2 Database for development and testing in Spring Boot?

Add the H2 dependency and configure the database settings in `application.properties`:

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.h2.console.enabled=true
```

21. What is the purpose of `@Autowired`?

`@Autowired` is used to inject beans (dependencies) automatically by Spring's dependency injection mechanism. It can be used on constructors, fields, or setter methods.

22. How can you customize the Spring Boot banner?

You can customize the Spring Boot startup banner by placing a `banner.txt` file in the `src/main/resources` directory. You can also disable it entirely using `spring.main.banner-mode=off` in the `application.properties` file.

23. How can you create a custom starter in Spring Boot?

To create a custom starter, you need to create a new project with the necessary dependencies and configuration, then package it as a JAR. Include this JAR as a dependency in your Spring Boot application.

24. How do you run a Spring Boot application as a standalone jar?

Spring Boot applications can be packaged as executable JAR files with an embedded server. You can run the JAR using the command `java -jar myapp.jar`.

25. What are the best practices for logging in Spring Boot?

Use SLF4J with Logback as the default logging framework. Configure logging levels in `application.properties` and use appropriate logging levels (DEBUG, INFO, WARN, ERROR) in your code.

```
logging.level.org.springframework=INFO
logging.level.com.example=DEBUG
```

26. How do you externalize configuration in Spring Boot?

Externalize configuration using `application.properties` or `application.yml` files, environment variables, or command-line arguments. This allows you to manage application settings without changing the code.

27. How can you monitor Spring Boot applications?

Use Spring Boot Actuator to monitor applications. It provides endpoints for health checks, metrics, and more. Integrate with monitoring tools like Prometheus, Grafana, or ELK stack for enhanced monitoring.

28. How do you handle file uploads in Spring Boot?

Handle file uploads using `MultipartFile` in a controller method. Ensure you configure the `spring.servlet.multipart` properties in `application.properties`.

```
@PostMapping("/upload")
public String handleFileUpload(@RequestParam("file") MultipartFile
file) {
    // handle the file
    return "File uploaded successfully!";
}
```

29. What is the purpose of `@ConfigurationProperties`?

`@ConfigurationProperties` is used to bind external configuration properties to a Java object. It's useful for type-safe configuration.

```
@ConfigurationProperties(prefix = "app")
public class AppProperties {
    private String name;
    private String description;

    // getters and setters
}
```

30. How do you schedule tasks in Spring Boot?

Schedule tasks using `@EnableScheduling` and `@Scheduled` annotations. Define a method with the `@Scheduled` annotation to run tasks at specified intervals.

```
@EnableScheduling
public class SchedulingConfig {
}

@Component
public class ScheduledTasks {
    @Scheduled(fixedRate = 5000)
    public void reportCurrentTime() {
        System.out.println("Current time is " + new Date());
    }
}
```

31. How can you use Spring Boot with Kotlin?

Spring Boot supports Kotlin. Create a Spring Boot application using Kotlin by adding the necessary dependencies and configuring the project. Kotlin's concise syntax can make the code more readable and maintainable.

32. What is Spring WebFlux?

Spring WebFlux is a reactive web framework in the Spring ecosystem, designed for building reactive and non-blocking web applications. It uses the Reactor project for its reactive support.

33. How do you enable CORS in Spring Boot?

Enable CORS (Cross-Origin Resource Sharing) using the `@CrossOrigin` annotation on controller methods or globally using a `CorsConfiguration` bean.

```
@RestController
@CrossOrigin(origins = "http://example.com")
```

```
public class MyController {
    @GetMapping("/greeting")
    public String greeting() {
        return "Hello, World!";
    }
}
```

34. How do you use Redis with Spring Boot?

Use Redis with Spring Boot by adding the `spring-boot-starter-data-redis` dependency and configuring Redis properties in `application.properties`.

```
spring.redis.host=localhost
spring.redis.port=6379
```

35. What is Spring Cloud and how is it related to Spring Boot?

Spring Cloud provides tools for building microservices and distributed systems on top of Spring Boot. It offers features like configuration management, service discovery, and circuit breakers.

36. How do you implement caching in Spring Boot?

Implement caching using the `@EnableCaching` annotation and a caching library like EhCache, Hazelcast, or Redis. Annotate methods with `@Cacheable`, `@CachePut`, and `@CacheEvict` for caching behavior.

```
@EnableCaching
public class CacheConfig {
}

@Service
public class UserService {
    @Cacheable("users")
```

```
    public User getUserById(Long id) {
        return userRepository.findById(id).orElse(null);
    }
}
```

37. How can you send emails with Spring Boot?

Send emails using Spring Boot by adding the `spring-boot-starter-mail` dependency and configuring email properties in `application.properties`.

Use `JavaMailSender` to send emails.

```
spring.mail.host=smtp.example.com
spring.mail.port=587
spring.mail.username=user@example.com
spring.mail.password=secret
```

```
@Service
public class EmailService {
    @Autowired
    private JavaMailSender mailSender;

    public void sendSimpleMessage(String to, String subject, String
text) {
        SimpleMailMessage message = new SimpleMailMessage();
        message.setTo(to);
        message.setSubject(subject);
        message.setText(text);
        mailSender.send(message);
    }
}
```

38. What is `@SpringBootTest`?

`@SpringBootTest` is an annotation that loads the full application context for integration tests. It is used to write tests that require Spring Boot's features, like dependency injection and embedded servers.

39. How do you integrate Spring Boot with a front-end framework like Angular or React?

Integrate Spring Boot with front-end frameworks by building the front-end project and placing the static files in the `src/main/resources/static` directory of your Spring Boot project. Configure Spring Boot to serve these files.

40. How do you configure Thymeleaf in Spring Boot?

Thymeleaf is a templating engine supported by Spring Boot. Add the `spring-boot-starter-thymeleaf` dependency and place your templates in the `src/main/resources/templates` directory.

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

41. What is the purpose of `@SpringBootApplication`?

`@SpringBootApplication` is a convenience annotation that combines `@Configuration`, `@EnableAutoConfiguration`, and `@ComponentScan`. It marks the main class of a Spring Boot application.

42. How do you use CommandLineRunner in Spring Boot?

CommandLineRunner is an interface used to execute code after the Spring Boot application starts. Implement the `run` method to perform actions on startup.

```
@Component
public class MyCommandLineRunner implements CommandLineRunner {
    @Override
    public void run(String... args) throws Exception {
        System.out.println("Hello, World!");
    }
}
```

43. How do you connect to an external REST API using Spring Boot?

Connect to an external REST API using `RestTemplate` or `WebClient`. `RestTemplate` is synchronous, while `WebClient` is asynchronous and non-blocking.

```
@RestController
@RequestMapping("/api")
public class ApiController {
    @Autowired
    private RestTemplate restTemplate;

    @GetMapping("/data")
    public String getData() {
        return
restTemplate.getForObject("https://api.example.com/data",
String.class);
    }
}
```

44. How do you implement pagination in Spring Boot?

Implement pagination using Spring Data JPA's `Pageable` interface. Define repository methods that accept `Pageable` parameters.

```
public interface UserRepository extends JpaRepository<User, Long> {
    Page<User> findByLastName(String lastName, Pageable pageable);
}
```

45. How do you document a Spring Boot REST API?

Document a Spring Boot REST API using Swagger. Add the `springfox-swagger2` and `springfox-swagger-ui` dependencies and configure Swagger.

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>
```

46. How do you handle validation in Spring Boot?

Handle validation using the `javax.validation` package. Use annotations like `@NotNull`, `@Size`, and `@Email` in your model classes, and `@Valid` in your controller methods.

```
public class User {
    @NotNull
    private String name;
    @Email
    private String email;
}
```

47. How do you set up Spring Boot with Docker?

Set up Spring Boot with Docker by creating a `Dockerfile` that specifies the base image and instructions to build and run the application.

```
FROM openjdk:11-jre-slim
COPY target/myapp.jar myapp.jar
ENTRYPOINT ["java", "-jar", "/myapp.jar"]
```

48. How do you deploy a Spring Boot application to AWS?

Deploy a Spring Boot application to AWS by using services like Elastic Beanstalk, ECS, or Lambda. Package your application as a JAR or Docker image and upload it to the chosen service.

49. What is the difference between Spring Boot and Spring MVC?

Spring Boot is a framework for quickly building Spring-based applications with minimal configuration. Spring MVC is a framework for building web applications using the Model-View-Controller design pattern. Spring Boot often uses Spring MVC as part of its web starter.

50. How do you migrate a legacy application to Spring Boot?

Migrate a legacy application to Spring Boot by incrementally introducing Spring Boot dependencies and configurations. Replace legacy configurations with Spring Boot's auto-configuration and starters, and gradually refactor the application to use Spring Boot features.

Spring Boot Interview Questions: Conclusion

Spring Boot is widely liked by developers because it's easy to use and powerful. Learning from these top 50 questions and answers helps you understand Spring Boot better. You

can solve many problems like setting up applications, connecting to databases, adding security, and putting your app on the cloud. Spring Boot makes these tasks simpler, helping you build better applications faster. Keep learning and enjoy coding with Spring Boot!