Spring Boot Interview Questions & Answers with Code Examples

1. What is Spring Boot? How is it different from the Spring Framework?  
Explanation:  
Spring Boot is a framework built on top of Spring that simplifies the setup, configuration, and development of new Spring applications. It provides defaults for code and annotation configuration to reduce boilerplate. Unlike the traditional Spring Framework, Spring Boot offers embedded servers, opinionated starter dependencies, and auto-configuration to get apps running quickly without manual XML or annotation config.

Snippet:  
@SpringBootApplication // Combines @Configuration, @EnableAutoConfiguration, @ComponentScan  
public class MyApp {  
 public static void main(String[] args) {  
 SpringApplication.run(MyApp.class, args);  
 }  
}

2. What are the main features of Spring Boot?  
- Auto-configuration  
- Embedded web servers (Tomcat, Jetty)  
- Starter dependencies (to easily add libraries)  
- Production-ready metrics and health checks via Actuator  
- Externalized configuration  
- Opinionated defaults

3. What is the role of the @SpringBootApplication annotation?  
It’s a convenience annotation that includes:  
- @Configuration — marks the class as a source of bean definitions  
- @EnableAutoConfiguration — enables Spring Boot’s auto-configuration mechanism  
- @ComponentScan — enables component scanning from the package of the class

4. How does Spring Boot simplify configuration compared to traditional Spring?  
Spring Boot uses auto-configuration to automatically set up beans based on the classpath and properties, reducing or eliminating the need for manual XML or Java config. It also provides starter POMs for common dependencies and embeds servers to run apps without deploying WARs.

5. What are starters in Spring Boot? Give examples.  
Starters are dependency descriptors that bundle commonly used libraries for a specific feature, so you just include one starter instead of multiple dependencies.

Examples:  
- spring-boot-starter-web for building web applications  
- spring-boot-starter-data-jpa for JPA & Hibernate  
- spring-boot-starter-security for security

6. What is an embedded server in Spring Boot? Which servers are commonly used?  
An embedded server runs your web application inside the app itself (no need to deploy to an external server). Spring Boot commonly uses embedded Tomcat by default but supports Jetty and Undertow.

7. How does Spring Boot auto-configuration work?  
Spring Boot scans the classpath for libraries and checks your configuration to apply sensible default configurations automatically. For example, if spring-boot-starter-web is on the classpath, it auto-configures Tomcat and Spring MVC beans.

8. What are Spring Boot Actuators? What useful endpoints does it provide?  
Actuator adds production-ready features like health checks, metrics, info, and environment details. Common endpoints:  
- /actuator/health — app health  
- /actuator/metrics — application metrics  
- /actuator/info — general info about the app

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

9. What is the use of the application.properties or application.yml file?  
Used to externalize configuration — define app properties like server port, datasource URL, logging levels, etc., outside of the code for easy modification without rebuilding.

10. How do you externalize configuration in Spring Boot?  
By using application.properties or application.yml, environment variables, command line arguments, or profiles to override default settings.

11. What are the scopes of Spring Beans?  
- Singleton (default): One instance per Spring container  
- Prototype: New instance every time requested  
- Request: One per HTTP request (web apps)  
- Session: One per HTTP session (web apps)  
- Application: One per ServletContext (web apps)

12. Difference between @Component, @Service, @Repository, and @Controller  
- @Component: Generic stereotype for any Spring-managed component  
- @Service: Business service layer  
- @Repository: Data access layer (also translates exceptions)  
- @Controller: MVC controller for handling HTTP requests

13. How does Spring Boot handle component scanning?  
By default, it scans all classes in the package where the main app class is located and its sub-packages for classes annotated with @Component or its stereotypes.

14. How do you create a custom configuration class in Spring Boot?  
Example:  
@Configuration  
public class AppConfig {  
 @Bean  
 public MyService myService() {  
 return new MyServiceImpl();  
 }  
}

15. How do you configure Spring Data JPA with Spring Boot?  
Add dependency:  
<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>  
Configure datasource in application.properties:  
spring.datasource.url=jdbc:mysql://localhost:3306/mydb  
spring.datasource.username=root  
spring.datasource.password=password  
spring.jpa.hibernate.ddl-auto=update

16. Role of @Entity and @Id in JPA?  
@Entity marks a class as a JPA entity mapped to a DB table.  
@Id marks the primary key of the entity.

17. How do you create a simple CRUD repository using Spring Data JPA?  
Example:  
public interface UserRepository extends JpaRepository<User, Long> {  
}

18. Different ways to query the database with Spring Data JPA?  
- Using method names: findByName(String name)  
- Using @Query annotation with JPQL or native SQL  
- Using QueryDSL or Specifications for complex queries

19. How does Spring Boot handle database connection pooling?  
By default, Spring Boot uses HikariCP for connection pooling, which is fast and efficient.

20. How do you create a REST controller in Spring Boot?  
Example:  
@RestController  
@RequestMapping("/api/users")  
public class UserController {

@GetMapping("/{id}")  
 public User getUser(@PathVariable Long id) {  
 // fetch user logic  
 return user;  
 }  
}

21. Explain @RequestMapping vs @GetMapping, @PostMapping, etc.  
@RequestMapping is generic and can map any HTTP method.  
@GetMapping, @PostMapping, etc., are shortcuts for @RequestMapping(method = RequestMethod.GET), etc.

22. How do you handle exceptions in Spring Boot REST APIs?  
Use @ControllerAdvice or @RestControllerAdvice to define global exception handlers with @ExceptionHandler.

23. Purpose of @RestControllerAdvice?  
It handles exceptions globally across all @RestController s, returning proper HTTP responses.

24. How do you validate request parameters or body in Spring Boot?  
Use JSR-303 annotations like @Valid with DTOs and add @NotNull, @Size, etc. on fields.

Example:  
@PostMapping("/add")  
public ResponseEntity<?> addUser(@Valid @RequestBody UserDto user) {  
 // ...  
}

25. How to handle CORS in Spring Boot?  
Using @CrossOrigin annotation on controllers or global CORS config with WebMvcConfigurer.

26. What is Spring Security? How to enable basic authentication?  
Spring Security adds authentication and authorization. Basic auth can be enabled with starter dependency and configuration.

27. How do you configure custom user authentication?  
By implementing UserDetailsService and configuring AuthenticationManager in a security config class.

28. How do you write unit tests for Spring Boot applications?  
Using @SpringBootTest or @WebMvcTest with JUnit and Mockito.

29. What annotations to test Spring Boot applications?  
- @SpringBootTest for full context  
- @MockBean for mocking dependencies  
- @WebMvcTest for controller layer tests

30. Difference between @MockBean and @Autowired in tests?  
@MockBean creates a mock and injects it; @Autowired injects real beans from the context.

31. How do you package and run a Spring Boot application?  
Package with Maven or Gradle (mvn clean package), then run the executable JAR:  
java -jar target/myapp.jar

32. How do you enable logging in Spring Boot?  
By default, Spring Boot uses Logback. Configure in application.properties:  
logging.level.org.springframework=DEBUG

33. What are profiles in Spring Boot and how do you use them?  
Profiles allow different configs for different environments. Use application-{profile}.properties and activate via spring.profiles.active.

34. How does Spring Boot support externalized config for different environments?  
By profiles, environment variables, command line args, or config servers.

35. Difference between @ComponentScan and default component scan?  
Default component scan scans from the package of the main app class. @ComponentScan lets you specify base packages explicitly.

36. How to override auto-configuration?  
Define your own bean with the same name or use @ConditionalOnMissingBean to prevent auto-config beans from loading.

37. Lifecycle of a Spring Bean?  
Instantiate → Populate properties → @PostConstruct → Ready for use → Destroy (@PreDestroy)

38. How does Spring Boot support microservices architecture?  
Spring Boot apps are lightweight and self-contained, making it easy to create independently deployable services that can register with service discovery tools like Eureka.