Spring Core Questions

1. What is Spring Framework? Why is it used?

Answer: Spring is a Java framework for building enterprise applications.

It provides IoC (Inversion of Control) and DI (Dependency Injection), which make code loosely coupled, testable, and easier to maintain.

✅ Example:

```
@Component
class Engine {
   public void start() { System.out.println("Engine started"); }
}
@Component
class Car {
   private final Engine engine;
   @Autowired
   Car(Engine engine) { this.engine = engine; }
   public void drive() { engine.start(); System.out.println("Car is moving"); }
}
```

Spring injects Engine into Car → we don't create objects manually (new Engine()).

2. What is Dependency Injection? How is it implemented in Spring?

Answer: Dependency Injection is a design pattern in which an object (a client) receives the objects it depends on (its dependencies) from an external source rather than creating them itself.

Instead of:

```
public class OrderService {
   private PaymentService paymentService = new PaypalPaymentService(); // tight coupling
}
```

You do:

```
public class OrderService {
```

```
    private final PaymentService paymentService;

    public OrderService(PaymentService paymentService) { this.paymentService = paymentService; } //
dependency injected

}
```

Why DI?

Loose coupling — classes depend on interfaces, not concrete implementations.

Testability — you can inject mocks in tests.

Flexibility — choice of implementation can be changed/configured externally.

Single Responsibility — classes don't manage dependency creation.

How Spring implements DI (overview)

Spring provides an Inversion of Control (IoC) container (e.g., ApplicationContext) that manages objects called beans. The container:

Scans/reads bean definitions (via annotations, Java config, or XML).

Instantiates bean objects.

Resolves and injects dependencies (by type, qualifier, or name).

Manages bean lifecycle (init, destroy), scopes (singleton, prototype), proxies (AOP), etc.

Key components / annotations:

@Component, @Service, @Repository, @Controller — mark classes as beans.

@Autowired — requests injection (constructor/setter/field).

@Qualifier, @Primary — disambiguate when multiple beans of same type exist.

@Configuration + @Bean — Java-based bean definitions.

ApplicationContext / BeanFactory — the IoC container.

3. What is the difference between BeanFactory and ApplicationContext?

BeanFactory: Basic container, lazy loading.

ApplicationContext: Advanced container, supports AOP, internationalization, event handling, eager loading.

Almost always use ApplicationContext.

4. What are Spring Beans? How are they defined?

A Spring Bean is an object managed by the Spring IoC container.

Defined using:

@Component / @Service / @Repository / @Controller

@Bean inside @Configuration class.

✔ Example:

```
@Configuration
class AppConfig {
   @Bean
   public Engine engine() { return new Engine(); }
}
```

5. Bean Scopes in Spring?

singleton → one instance per container (default).

prototype → new instance every time.

request → one per HTTP request (web apps).

session → one per HTTP session.

6. What is Spring AOP? Where have you used it?

Aspect Oriented Programming → add cross-cutting concerns (logging, transactions, security) without changing business code.

✔ Example (Logging Aspect):

```
@Aspect
@Component
class LoggingAspect {
   @Before("execution(* com.example.service.*.*(..))")
   public void logBefore(JoinPoint jp) {
      System.out.println("Method called: " + jp.getSignature());
   }
}
```

7. What is Spring Boot? How is it different from Spring?

Spring Boot = Spring + auto-configuration + embedded server (Tomcat) + starter dependencies.

Removes boilerplate → just run main() and app starts.

✅ Example:

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

8. What are Spring Boot Starters?

Predefined dependency packages for common use cases.

Examples:

spring-boot-starter-web → web + REST

spring-boot-starter-data-jpa → JPA + Hibernate

spring-boot-starter-security → security

9. What is Spring Boot Auto-Configuration?

Spring Boot automatically configures beans based on dependencies.

Controlled using EnableAutoConfiguration (included in @SpringBootApplication).

Can exclude:

```
@SpringBootApplication(exclude = DataSourceAutoConfiguration.class)
```

10. What is Spring Boot Actuator?

Provides production-ready endpoints like /actuator/health, /actuator/metrics.

✅ Example:

Add dependency:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
```

```
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Then visit → http://localhost:8080/actuator/health

11. How do you connect to a database in Spring Boot?

Configure in application.properties:

```
spring.datasource.url=jdbc:mysql://localhost:3306/testdb
spring.datasource.username=root
spring.datasource.password=pass
spring.jpa.hibernate.ddl-auto=update
```

Use Spring Data JPA repository:

```
interface EmployeeRepository extends JpaRepository<Employee, Long> {}
```

12. What is @RestController vs @Controller?

@Controller → used in MVC, returns views (HTML, JSP).

@RestController → combination of @Controller + @ResponseBody. Returns JSON/XML.

✔ Example:

```
@RestController
@RequestMapping("/api")
class EmployeeController {
   @GetMapping("/hello")
   public String hello() {
      return "Hello World";  // JSON/text response
   }
}
```

13. How do you handle exceptions in Spring Boot?

Use @ControllerAdvice + @ExceptionHandler.

✔ Example:

```
@ControllerAdvice
```

```
public class GlobalExceptionHandler {
    @ExceptionHandler(Exception.class)
    public ResponseEntity<String> handleException(Exception ex) {
        return new ResponseEntity<>("Error: " + ex.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

14. What is @SpringBootApplication?

Combines 3 annotations:

@Configuration (for bean definitions)

@EnableAutoConfiguration

@ComponentScan (scans beans automatically)

15. How do you secure a Spring Boot application?

Using Spring Security (spring-boot-starter-security).

By default → adds basic authentication.

You can configure WebSecurityConfigurerAdapter to customize.

Real-Time Scenario Questions

16. How to handle transactions in Spring Boot?

Use @Transactional on methods.

```
@Service
class PaymentService {
    @Transactional
    public void processPayment() {
        // debit + credit logic → rollback if exception
    }
}
```

17. How do you improve performance in Spring Boot?

Use caching (@EnableCaching, @Cacheable).

Optimize queries with indexes.

Use pagination in repositories (Pageable).

18. What are Profiles in Spring Boot?

Profiles let you load different configurations for dev/test/prod environments.

application-dev.properties

server.port=8081

application-prod.properties

server.port=80

Activate with:

spring.profiles.active=dev

19. What is the difference between @Component, @Service, @Repository?

@Component → generic bean.

@Service → business logic layer.

@Repository → DAO layer, also translates DB exceptions.

20. What is the difference between @RequestParam and @PathVariable?

@RequestParam → extracts query parameter (?id=10).

@PathVariable → extracts value from URL path (/user/10).

✔ Example:

@GetMapping("/user")

public String getUser(@RequestParam int id) { return "User " + id; }

@GetMapping("/user/{id}")

public String getUserById(@PathVariable int id) { return "User " + id; }

Expect basic Spring Core (DI, IoC, beans, scopes).

Spring Boot basics (starters, auto-config, actuator).

REST APIs (@RestController, @RequestParam, @PathVariable).

Database (Spring Data JPA, transactions).

Exception handling & AOP basics.

Real-time scenarios (profiles, caching, performance).

1. What is the Spring Framework?

Spring is a Java framework for building enterprise applications. It provides IoC (Inversion of Control) and DI (Dependency Injection) to reduce tight coupling between classes.

2. What is Inversion of Control (IoC)?

IoC means the control of object creation is given to the Spring container, instead of creating objects manually using new.

3. What is Dependency Injection (DI)?

A design pattern where Spring injects required dependencies into a class.

Types: Constructor Injection (preferred), Setter Injection, Field Injection.

✔ Example:

@Autowired

public Car(Engine engine) { this.engine = engine; }

4. What is the difference between BeanFactory and ApplicationContext?

BeanFactory: Basic container, lazy loading.

ApplicationContext: Advanced, supports AOP, events, internationalization, eager loading (commonly used).

5. What are Spring Beans?

Spring Beans are objects managed by the Spring IoC container. Defined with annotations like @Component, @Service, @Repository, or using @Bean.

6. What are different Bean Scopes?

singleton (default) → one instance per Spring container.

prototype → new instance every time requested.

Web scopes: request, session, application.

7. What is Spring AOP?

Aspect Oriented Programming – used for cross-cutting concerns (logging, transactions, security). Implemented using @Aspect.

8. Difference between @Component, @Service, @Repository?

@Component: Generic bean.

@Service: Business logic layer.

@Repository: DAO layer, converts DB exceptions into Spring exceptions.

9. What is Spring Boot?

Spring Boot makes Spring development easier by providing:

Auto-configuration

Starter dependencies

Embedded server (Tomcat, Jetty)

10. What is @SpringBootApplication?

It is a combination of:

@Configuration

@EnableAutoConfiguration

@ComponentScan

11. What are Spring Boot Starters?

Predefined dependency bundles.

Examples:

spring-boot-starter-web → web & REST apps.

spring-boot-starter-data-jpa → DB with JPA & Hibernate.

spring-boot-starter-security → authentication & authorization.

12. What is Auto-Configuration in Spring Boot?

Spring Boot automatically configures beans based on dependencies present in the classpath. Can be excluded using:

@SpringBootApplication(exclude = DataSourceAutoConfiguration.class)

13. What is Spring Boot Actuator?

Provides production-ready endpoints like /actuator/health, /actuator/metrics to monitor and manage applications.

14. How do you handle exceptions in Spring Boot?

Use @ControllerAdvice + @ExceptionHandler.

✅ Example:

```
@ControllerAdvice
class GlobalExceptionHandler {
  @ExceptionHandler(Exception.class)
  public ResponseEntity<String> handle(Exception e) {
     return ResponseEntity.status(500).body("Error: " + e.getMessage());
  }
}
```

15. Difference between @Controller and @RestController?

@Controller → returns view (HTML/JSP).

@RestController = @Controller + @ResponseBody → returns JSON/XML.

16. Difference between @RequestParam and @PathVariable?

@RequestParam → extracts query parameters (?id=10).

@PathVariable → extracts values from URL path (/user/10).

17. How do you connect Spring Boot to a database?

Configure in application.properties:

```
spring.datasource.url=jdbc:mysql://localhost:3306/testdb
spring.datasource.username=root
spring.datasource.password=pass
```

spring.jpa.hibernate.ddl-auto=update

Then use JpaRepository.

18. What is Spring Data JPA?

A part of Spring that simplifies database access using repositories.

✅ Example:

interface EmployeeRepository extends JpaRepository<Employee, Long> {}

19. What is @Transactional?

Ensures database operations are executed in a single transaction (commit/rollback automatically).

@Transactional

public void transferMoney(...) { ... }

20. What are Spring Profiles?

Used to load environment-specific configurations (dev, test, prod).

spring.profiles.active=dev

21. What is the difference between Clustered and Non-Clustered Index?

Clustered Index: Data stored in sorted order (1 per table).

Non-Clustered Index: Separate index structure with pointers (many per table).

22. How do you optimize a slow query in Spring Boot?

Add proper indexes.

Avoid SELECT *.

Use pagination (Pageable).

Use JOIN instead of nested queries.

Check execution plan (EXPLAIN).

23. How do you secure a Spring Boot application?

Using Spring Security (spring-boot-starter-security). By default, it enables basic authentication. Custom roles can be defined in SecurityConfig.

24. What is REST API in Spring Boot?

REST = Representational State Transfer.

Spring Boot uses @RestController to build REST APIs.

✔ Example:

```
@RestController
@RequestMapping("/api")
class HelloController {
  @GetMapping("/hello")
  public String sayHello() { return "Hello!"; }
}
```

25. Real-time scenario: If an API is slow, how do you troubleshoot?

Check DB queries → optimize with indexes.

Enable caching with @Cacheable.

Use asynchronous processing (@Async).

Monitor with Actuator/Logs.

Scale application (load balancing).

✔ Tip for you (2.4 yrs experience):

Interviewers usually expect you to:

Know basics of Spring & Spring Boot.

Explain real-time usage (where you applied transactions, exception handling, AOP).

Write simple code snippets for REST APIs, DI, and JPA.

Q. What is SDLC?

SDLC

Ans. SDLC stands for Software Development Life Cycle, a process used to design, develop, and test software applications.

SDLC is a structured process that includes planning, designing, coding, testing, and deployment of software.

Common SDLC models include Waterfall, Agile, and DevOps.

Each phase of SDLC has specific goals and deliverables to ensure the quality and success of the software project.

What is a daemon thread?

Operating Systems

Ans. A daemon thread is a low priority thread that runs in the background and does not prevent the JVM from exiting when all user threads have finished.

Daemon threads are used for tasks that need to run continuously in the background, such as garbage collection.

They are automatically terminated by the JVM when all user threads have finished execution.

To create a daemon thread in Java, you can call the setDaemon(true) method on a Thread object before starting it.

Q. What is volatile?

Ans. Volatile is a Java keyword that ensures visibility and ordering of variables across threads.

Volatile variables are stored in main memory, ensuring visibility across threads.

Changes to a volatile variable are immediately visible to other threads.

Volatile does not guarantee atomicity; use synchronized for atomic operations.

Example: 'volatile int counter;' ensures all threads see the latest value of counter.

Q.What is JDK? Explain.

Ans. JDK stands for Java Development Kit, it is a software development kit used for developing Java applications.

JDK includes tools for developing, debugging, and monitoring Java applications.

It contains JRE (Java Runtime Environment) which is necessary for running Java programs.

JDK also includes a compiler, debugger, and other tools needed for Java development.

Examples of JDK versions include JDK 8, JDK 11, and JDK 15.

Q. What is JRE? Explain.

Ans. JRE stands for Java Runtime Environment, it is a software package that provides the necessary tools to run Java applications.

JRE includes Java Virtual Machine (JVM), class libraries, and other necessary files to run Java applications.

It does not contain development tools like compiler or debugger, which are included in JDK (Java Development Kit).

JRE is required to run Java applications on a computer, as it provides the environment for executing Java code.

Examples of JRE implementations include Oracle JRE, OpenJDK, and IBM JRE.

Q.What are the differences between HashMap and Hashtable in Java?

Ans. Hashtable is synchronized and does not allow null keys or values, while HashMap is not synchronized and allows null keys and values.

Hashtable is synchronized, while HashMap is not.

Hashtable does not allow null keys or values, while HashMap allows them.

Hashtable is thread-safe, while HashMap is not.

Q. How can you write a custom exception in Java?

Ans. To write a custom exception in Java, create a new class that extends Exception or a subclass of Exception.

Create a new class that extends Exception or a subclass of Exception.

Add a constructor to the custom exception class to pass a message to the superclass constructor.

Throw the custom exception using the 'throw' keyword in your code.

Handle the custom exception using try-catch blocks or propagate it up the call stack.

Q. Explain Normalization.

Ans. Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity.

1. First Normal Form (1NF): Ensures that all columns contain atomic values. Example: A table with a 'Phone Numbers' column should not have multiple numbers in one cell.

2. Second Normal Form (2NF): Achieves 1NF and removes partial dependencies. Example: If a table has 'StudentID' and 'CourseID', 'CourseName' should depend only on 'CourseID'.

3. Third Normal Form (3NF): Achieves 2NF and removes transitive dependencies. Example: If 'StudentID' determines 'AdvisorID', and 'AdvisorID' determines 'AdvisorName', 'AdvisorName' should not be in the Student table.

4. Boyce-Codd Normal Form (BCNF): A stronger version of 3NF. Example: If a table has a composite key, all non-key attributes must depend on the whole key.

Q. Explain Final, Finally, and Finalize() in Java.

Ans. Final, Finally, Finalize() are keywords in Java used for different purposes.

Final is used to declare a variable as constant.

Finally is used to execute a block of code after try-catch block.

Finalize() is a method used for garbage collection.

Final can be used with classes, methods, and variables.

Finally block is always executed whether an exception is thrown or not.

Q. What are constructors and what are the different types of constructors?

Ans. Constructors are special methods used to initialize objects in Java.

Constructors have the same name as the class and do not have a return type.

Types of constructors include default constructor, parameterized constructor, and copy constructor.

Default constructor is provided by Java if no constructor is defined.

Parameterized constructor takes parameters to initialize an object with specific values.

Copy constructor creates a new object by copying the values of an existing object.

Q. What are the advantages of Spring Boot?

Ans. Spring Boot simplifies Java development with rapid setup, embedded servers, and production-ready features.

Rapid Application Development: Spring Boot allows developers to create stand-alone applications quickly with minimal configuration.

Embedded Servers: It comes with embedded servers like Tomcat or Jetty, eliminating the need for external server deployment.

Convention over Configuration: Spring Boot follows this principle, reducing the need for extensive XML configuration files.

Production-Ready Features: It includes built-in features like health checks, metrics, and externalized configuration for easy deployment.

Microservices Support: Spring Boot is ideal for building microservices, with easy integration with Spring Cloud for distributed systems.

Q. What is the jQuery AJAX GET request syntax?

Ans. The jQuery ajax GET request syntax is used to send an HTTP GET request to a server and retrieve data.

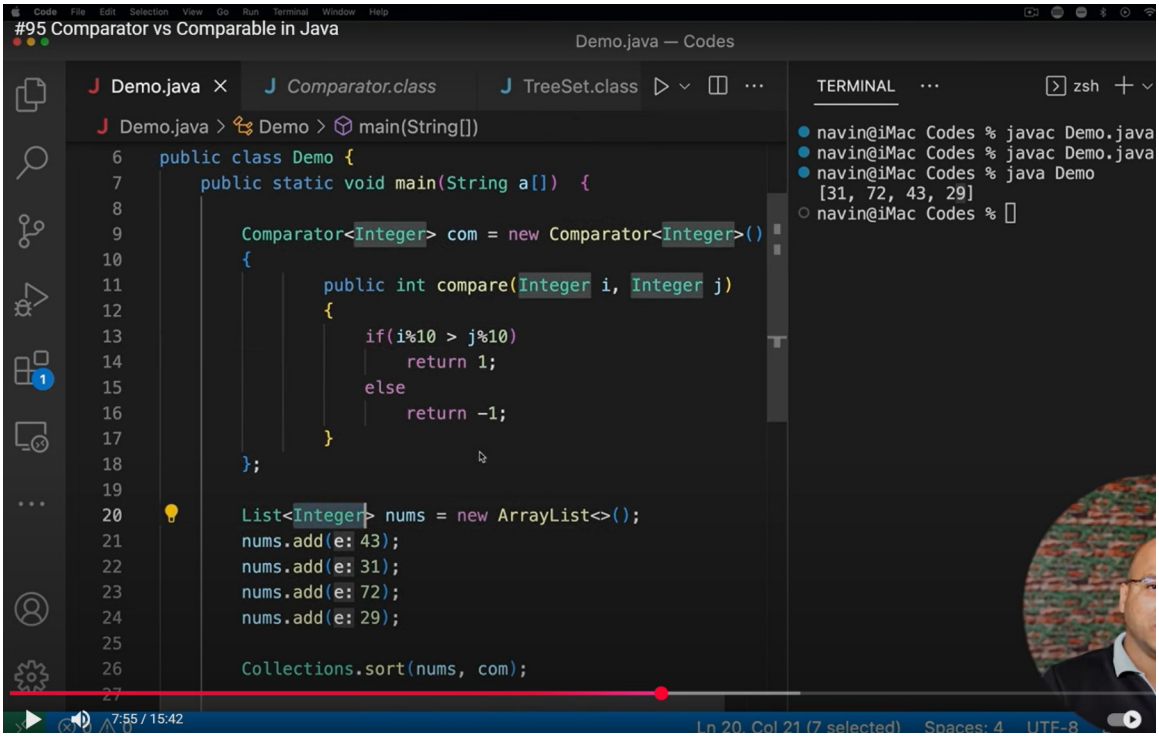
Use the $.ajax() method with the 'type' parameter set to 'GET'

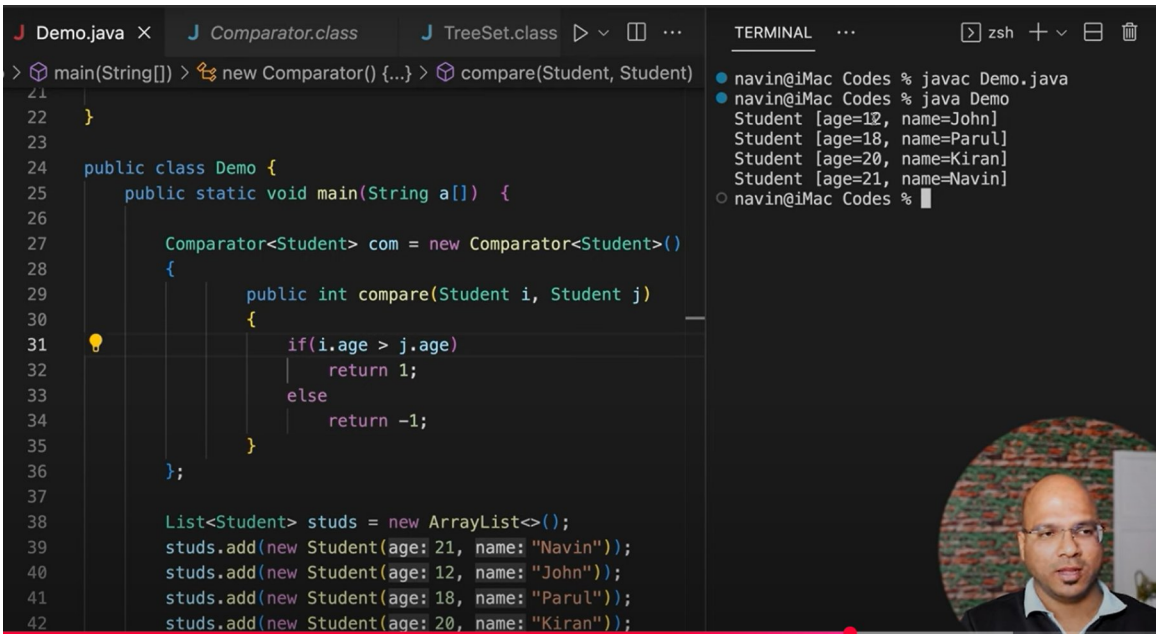
Specify the URL of the server endpoint as the 'url' parameter

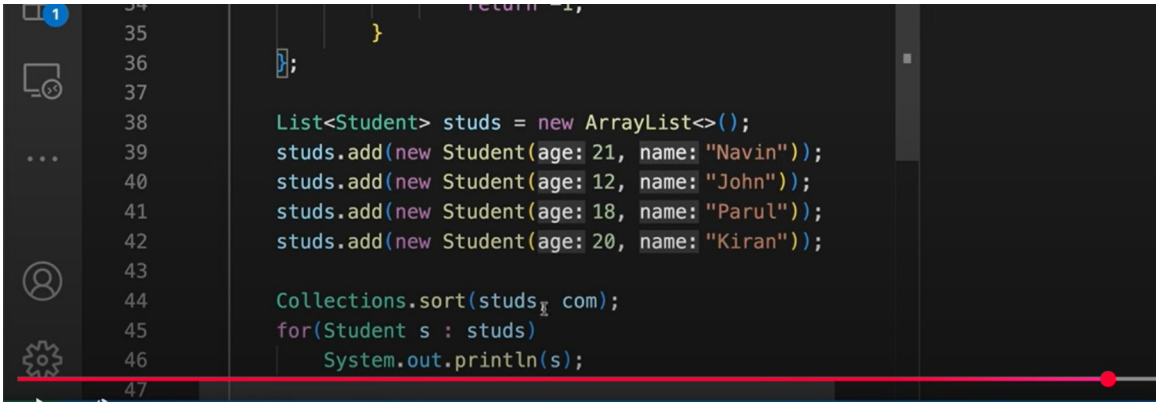
Handle the response using the 'success' callback function

Optionally, pass data to the server using the 'data' parameter

Q.What is camapator?







1) Spring Security

Q1. What is Spring Security and why is it used?

Answer:

Spring Security is a powerful framework that provides authentication, authorization, and protection against common attacks (CSRF, session fixation, clickjacking). It integrates seamlessly with Spring Boot and supports multiple authentication mechanisms like Basic Auth, JWT, OAuth2, LDAP, SAML.

Example:

/admin/** → only accessible to ADMIN

/user/** → accessible to USER role

```
@Override
protected void configure(HttpSecurity http) throws Exception {
   http.authorizeRequests()
      .antMatchers("/admin/**").hasRole("ADMIN")
      .antMatchers("/user/**").hasAnyRole("USER", "ADMIN")
      .antMatchers("/public/**").permitAll()
      .anyRequest().authenticated()
      .and().httpBasic();
}
```

Q2. Explain Authentication vs Authorization in Spring Security.

Answer:

Authentication = Who are you? → verifying identity (username/password, token, etc.)

Authorization = What can you access? → checking user's role/privileges

Example:

A user logs in with credentials → authentication.

The same user tries to access /admin/dashboard but has role USER → denied → authorization.

Q3. What is CSRF protection in Spring Security?

Answer:

CSRF (Cross-Site Request Forgery) is when an attacker tricks a logged-in user into performing unwanted actions.

Spring Security enables CSRF tokens by default for state-changing requests (POST, PUT, DELETE).

Example:

When you send a form request, Spring Security expects a hidden token like:

<input type=→hidden→ name=→_csrf→ value=→abc123xyz→>

This ensures requests are genuine.

Q4. How does password encryption work in Spring Security?

Answer:

Spring Security uses PasswordEncoders like BCryptPasswordEncoder to hash passwords instead of storing plain text.

Example:

```
@Bean
public PasswordEncoder passwordEncoder() {
   return new BCryptPasswordEncoder();
}
```

Password "12345" → stored as $2a$10$QvO3qf... (hashed).

Even if DB leaks, passwords remain safe.

Q5. How do you implement JWT authentication in Spring Security?

Answer (high-level steps):

User logs in → server validates → generates JWT token.

Token is sent in response.

For every request → client sends token in header.

Server validates token → allows or denies access.

Example header:

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR...

2) Encryption & Decryption in API

Q1. Why do we need encryption in API communication?

Answer:

To protect sensitive data (passwords, credit card numbers, personal info).

Even with HTTPS, some companies use payload-level encryption for extra security.

Prevents MITM (Man-in-the-Middle) attacks.

Q2. What are common algorithms used for encryption in Java APIs?

Answer:

AES (Advanced Encryption Standard) → symmetric (same key for encrypt & decrypt).

RSA (Rivest–Shamir–Adleman) → asymmetric (public/private keys).

SHA-256/512 → hashing (one-way, for password storage).

Q3. Example of AES Encryption & Decryption in Java API.

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import java.util.Base64;
```

```
public class AESUtil {
    private static SecretKey secretKey;

    static {
        try {
            KeyGenerator keyGen = KeyGenerator.getInstance("AES");
            keyGen.init(128); // 128-bit AES
            secretKey = keyGen.generateKey();
        } catch (Exception e) { throw new RuntimeException(e); }
    }

    public static String encrypt(String data) throws Exception {
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        return Base64.getEncoder().encodeToString(cipher.doFinal(data.getBytes()));
    }

    public static String decrypt(String encryptedData) throws Exception {
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        return new String(cipher.doFinal(Base64.getDecoder().decode(encryptedData)));
    }
}
```

Q4. What's the difference between Hashing vs Encryption?

Answer:

Encryption = two-way (you can encrypt & decrypt).

Hashing = one-way (cannot be decrypted, only compared).

Example:

Password storage → use hashing (BCrypt, SHA).

Secure message transfer → use encryption (AES, RSA).

Q5. How does HTTPS secure APIs?

Answer:

Uses TLS/SSL protocol for encryption.

Data between client & server is encrypted with public/private keys.

Prevents sniffing & tampering.

3) Global Exception Handling

Q1. What is Global Exception Handling in Spring Boot?

Answer:

Instead of writing try-catch inside every controller, we use @ControllerAdvice or @RestControllerAdvice to handle exceptions in a centralized way.

Q2. How do you implement a global exception handler?

Answer:

```
@ControllerAdvice
public class GlobalExceptionHandler {
```

```
    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<String> handleNotFound(ResourceNotFoundException ex) {
        return new ResponseEntity<>(ex.getMessage(), HttpStatus.NOT_FOUND);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<String> handleGlobal(Exception ex) {
        return new ResponseEntity<>("Something went wrong: " + ex.getMessage(),
                                    HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Q3. Difference between @ControllerAdvice and @RestControllerAdvice?

Answer:

@ControllerAdvice → used with @ResponseBody for JSON responses.

@RestControllerAdvice = @ControllerAdvice + @ResponseBody (shortcut).

Q4. How do you customize error responses?

Answer:

By creating an error response model.

```
public class ErrorDetails {
    private Date timestamp;
    private String message;
    private String details;
```

```
    // constructor + getters
}
```

Then return ErrorDetails from @ExceptionHandler.

Q5. Can you extend Spring's ResponseEntityExceptionHandler?

Answer:

Yes. You can extend ResponseEntityExceptionHandler and override methods like handleMethodArgumentNotValid() to handle validation errors globally.

Example:

```
@Override
protected ResponseEntity<Object> handleMethodArgumentNotValid(
        MethodArgumentNotValidException ex,
        HttpHeaders headers, HttpStatus status, WebRequest request) {
    Map<String, String> errors = new HashMap<>();
    ex.getBindingResult().getFieldErrors().forEach(error ->
        errors.put(error.getField(), error.getDefaultMessage()));
    return new ResponseEntity<>(errors, HttpStatus.BAD_REQUEST);
}
```

✓ Summary for Interview Prep

Spring Security → Authentication, Authorization, CSRF, JWT, password hashing.

Encryption/Decryption → AES, RSA, Hashing vs Encryption, HTTPS importance.

Global Exception Handling → @ControllerAdvice, @ExceptionHandler, custom error response, validation handling.