**Interview Question**

**Question 1**

**what if @controller @repository @service is chaned in all layer with @component . will project work**

Functionally, your application may continue to work, as **@Component** is a generic stereotype annotation and Spring will still recognize these components as beans. However, some features and optimizations provided by specialized annotations may not be available anymore. For example, **@Repository** provides exception translation for persistence exceptions, and **@Service** typically enables transaction management

**Question 2**

**How to make spring boot application secured ?**

**1---Add Spring Security Dependency**: Include the Spring Security dependency in your **pom.xml**

**<dependency>**

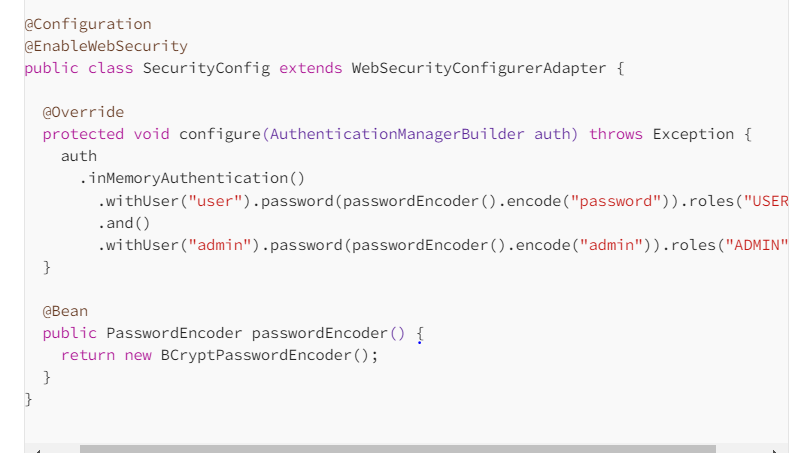
**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-starter-security</artifactId>**

**</dependency>**

# 2- Configuring Authentication

Spring Security supports various authentication mechanisms, such as in-memory, JDBC, LDAP, and OAuth2. To configure authentication, create a Java class that extends WebSecurityConfigurerAdapter and override the configure(AuthenticationManagerBuilder auth) method.

****

# 3-Configuring Authorization

To configure role-based authorization, override the configure(HttpSecurity http) method in your WebSecurityConfigurerAdapter class. Define access rules for specific endpoints based on user roles:

****

* Enable HTTPS by configuring an SSL certificate.
* Configure CORS and CSRF protection for cross-origin requests.
* Limit the number of login attempts to prevent brute force attacks.
* Implement proper password hashing and storage using strong password encoders like BCryptPasswordEncoder.

**CORS :--**

Cross-Origin Resource Sharing (CORS) is a W3C specification and mechanism that you can use to request restricted resources from a domain outside the current domain. In other words, CORS is a technique for consuming an API served from an origin different than yours.

**CSRF**

Cross-Site Request Forgery (CSRF) is a common web security vulnerability that allows attackers to make unauthorized actions on behalf of a user.

**brute force attacks.**

A brute force attack is a hacking method where an attacker tries to gain unauthorized access to a system or resource by systematically trying all possible combinations of usernames, passwords, or encryption keys until the correct one is found.

Question 3

**Different status code that u returned to the request in what case ?**

In RESTful APIs, different HTTP status codes are returned to indicate the outcome of a client's request. Here are some commonly used HTTP status codes and their meanings:

1. **200 OK**: The request was successful, and the server returned the requested resource.
2. **201 Created**: The request was successful, and a new resource was created as a result of the request. This is typically returned after a successful POST request to create a new resource.
3. **202** Accepted response status code indicates that the request has been accepted for processing, but the processing has not been completed
4. **204 No Content**: The request was successful, but there is no content to return. This is often used for successful requests that don't require a response body, such as successful DELETE requests.
5. **400 Bad Request**: The server could not understand the request due to invalid syntax or missing parameters.
6. **401 Unauthorized**: The request requires authentication, but the client has not provided valid credentials or has not authenticated.
7. **403 Forbidden**: The server understood the request, but the client does not have permission to access the requested resource.
8. **404 Not Found**: The server could not find the requested resource. This is often returned when the requested resource does not exist.
9. **405 Method Not Allowed**: The requested method (e.g., GET, POST, PUT) is not allowed for the requested resource.
10. **409 Conflict**: The request could not be completed due to a conflict with the current state of the resource, such as a version conflict or duplicate entries.
11. **500 Internal Server Error**: The server encountered an unexpected condition that prevented it from fulfilling the request.
12. **503 Service Unavailable**: The server is currently unable to handle the request due to temporary overloading or maintenance of the server.

These are just a few examples of HTTP status codes that can be returned in a RESTful API

**Question 4**

**What is required data is not passed in request . what response ?**

If required parameters are missing, the response might include a status code (**such as 400 Bad Request**) along with details about the validation error, which can guide you on what parameters are missing or incorrect.

Question 5

**what if required data is not present in db. what response will get ?**

the most common approach is to return a **404 Not Found status** code when the required data is not present in the database. This communicates to the client that the requested resource does not exist. However, depending on your API design and requirements, you may choose to use a different status code, such as 204 No Content or 200 OK with an empty response body.

**Question 6**

**how to test rest endpoint. how to make sure it works as expected ?**

Testing REST endpoints involves sending HTTP requests to the endpoint and verifying that the responses meet the expected behavior. Here's a general approach to testing REST endpoints and ensuring they work as expected:

**Unit Testing**: Write unit tests for each endpoint handler method in your application. Use testing frameworks like JUnit

**Integration Testing**: Perform integration tests to verify that the endpoints interact correctly with the rest of the application, including the database, external services, etc

**Security Testing**: Test the security of your endpoints to ensure they are protected against common security vulnerabilities such as injection attacks, XSS attacks, CSRF attacks, etc. Use tools like OWASP

**Performance Testing**: Perform performance tests to evaluate the performance of your endpoints under different load conditions. Use tools like JMete

**Monitoring and Logging**: Implement logging and monitoring in your application to track the behavior of the endpoints in production

**Continuous Integration and Deployment (CI/CD)**: CircleCI

**Question 7**

**What is component level testing ?**

Component testing is performed before Integration testing and after unit testing. Assume you are working on a project which has website development. Once the website's pages are developed, the development team will do a unit test and release the build for the testing team to test the component

The main objective of component testing is to verify the input/output behavior of the test object. It ensures that the test object's functionality works correctly and completely as per the desired specification. This type of testing focuses on performing separate tests on each component.

Question 8

**What is microservices architecture**

Microservice architecture is an architectural style that structures an application as a collection of loosely coupled, independently deployable services. Each service is focused on a specific business capability and can be developed, deployed, and scaled independently of other services. Microservices communicate with each other through well-defined APIs, typically over HTTP or messaging protocols.

Question 9

**How to do load balancing ?**

# [Spring Boot 2.4.2 Gateway API with Zuul](https://stackoverflow.com/questions/65924096/spring-boot-2-4-2-gateway-api-with-zuul)

# 

# Question 9 :

# how to join entity in JPA

In JPA (Java Persistence API), you can join entities using relationships defined between them. There are several types of relationships in JPA, including:

# **One-to-One Relationship**: In a one-to-one relationship, each record in one entity is associated with exactly one record in another entity. To establish a one-to-one relationship, you typically use @OneToOne annotation.

# **One-to-Many Relationship**: In a one-to-many relationship, each record in one entity can be associated with multiple records in another entity. To establish a one-to-many relationship, you typically use @OneToMany annotation.

# **Many-to-One Relationship**: In a many-to-one relationship, multiple records in one entity can be associated with a single record in another entity. To establish a many-to-one relationship, you typically use @ManyToOne annotation.

# **Many-to-Many Relationship**: In a many-to-many relationship, multiple records in one entity can be associated with multiple records in another entity. To establish a many-to-many relationship, you typically use @ManyToMany annotation.

# Question 10

# Define JPA and Different repository available in JPA ?

# JPA stands for Java Persistence API. It's a Java specification for managing relational data in Java applications. JPA provides a set of interfaces and annotations that allow developers to map Java objects to database tables and perform CRUD (Create, Read, Update, Delete) operations on those objects without having to write complex SQL queries. JPA implementations, such as Hibernate, EclipseLink, or Apache OpenJPA, handle the underlying database interactions, making it easier for developers to work with databases in Java applications.

1. **JpaRepository**: This is a repository interface provided by the Spring Data JPA framework. It extends the **CrudRepository** interface and provides additional methods for common data access operations, such as sorting, pagination, and custom queries. It's widely used in Spring-based applications for data access.
2. **CrudRepository**: Part of the Spring Data Commons module, this interface provides generic CRUD (Create, Read, Update, Delete) operations for entities. It's commonly used when working with Spring Data JPA repositories.

# Question 11

# What repository for pagination and sorting ?

# JpaRepository extends the PagingAndSortingRepository, which in turn extends the CrudRepository, adding methods to support pagination and sorting.

# 

# 

# Question 12

# Lazy fetch / eager fetch in hibernate

In Hibernate, lazy and eager fetching are strategies for loading associated entities or collections. These strategies determine when related entities or collections are loaded from the database.

1. **Eager Fetching**:
   * In eager fetching, associated entities or collections are loaded immediately along with the owning entity.
   * When you specify eager fetching for a particular association, Hibernate will automatically retrieve the associated entity or collection from the database when the owning entity is loaded.
   * This can lead to performance issues, especially if the associated entity or collection contains a large amount of data or if there are many associations to be eagerly fetched.
   * Eager fetching is specified using annotations like **@ManyToOne(fetch = FetchType.EAGER)** or **@OneToMany(fetch = FetchType.EAGER)**.
2. **Lazy Fetching**:
   * In lazy fetching, associated entities or collections are not loaded immediately when the owning entity is loaded.
   * Instead, the associated entities or collections are loaded from the database only when they are accessed for the first time.
   * Lazy fetching helps in avoiding unnecessary database queries and can improve performance, especially when dealing with large object graphs.
   * Lazy fetching is the default behavior for many-to-one and one-to-many associations in Hibernate.
   * Lazy fetching is specified using annotations like **@ManyToOne(fetch = FetchType.LAZY)** or **@OneToMany(fetch = FetchType.LAZY)**.

# Question 13

# What is spring boot starter pom ?

Parent project only having the version info i.e **spring-boot-starter-parent**

Will add all dependency due to **spring-boot-starter** in pom.xml

The libraries are pull what is listed in dependency section but version of those libraries is pulled by parent

**Version incompatibility problem resolved**