Java IQA Part 2

Q . Is password is mandatory to generate the JWT token ?

--------------------------------------------------------------------------------------

Nope — a password is not mandatory to generate a JWT token itself. But here’s the key idea:

🔐 JWT Generation Depends on Authentication

\* In most real-world apps, a user logs in with a username and password.

\* If the credentials are valid, the server generates a JWT token to represent that the user is authenticated.

\* The token contains user info (like ID or role), but not the password — and it’s signed with a secret key.

So while the token generation doesn’t require a password, the process that leads to generating it usually does, especially in login-based systems2.

Q . What are the dependency required to the generate the JWT token ?

------------------------------------------------------------------------------------------------------------

Maven dependency names:

\* jjwt-api

\* jjwt-impl

\* jjwt-jackson

Q)  Which token is required for sigup API ?

--------------------------------------------------------------

Spring Security Signup Flow

Signup API: No token needed — it's for new users

User submits info: Like username, email, password

Backend saves user: Password is encrypted

Login API: User logs in and gets a JWT token

Token used: For accessing protected endpoints

Q. What is the structure of JWT token ?

----------------------------------------------------------

xxxxx.yyyyy.zzzzz

xxxxx → Header

yyyyy → Payload

zzzzz → Signature

Q. What is Class Loader ?

----------------------------------------

A ClassLoader in Java is a special part of the Java Runtime Environment (JRE) that’s responsible for loading classes into memory when your program runs — kind of like a librarian fetching

the right book when you ask for it

Q . What is Spring Container ?

---------------------------------------------

Think of the Spring Container like a kitchen helper in a restaurant.

You (the chef) just tell it what ingredients you need — and it:

Brings the ingredients (creates objects)

Prepares them (sets them up)

Hands them to you when needed (connects everything)

You don’t have to go find things or mix them yourself — the helper does it all behind the scenes.

The Spring Container is the heart of the Spring Framework — it’s what powers the magic of dependency injection and bean lifecycle management in Java applications.

🧠 What Is It?

The Spring Container is responsible for:

Creating objects (called beans)

Configuring them based on metadata (XML, annotations, or Java config)

Managing their lifecycle (initialization, destruction, etc.)

Injecting dependencies between beans automatically

This is all part of the Inversion of Control (IoC) principle — instead of your code manually creating objects, the container does it for you.

🧰 Types of Spring Containers

Container Type    Description

BeanFactory == Basic container; lazy initialization; lightweight and suitable for simple apps

ApplicationContext == Advanced container; eager initialization; supports internationalization, event handling, and integration with other Spring modules2

🧪 How It Works

You define beans using annotations like @Component, @Service, or via XML/Java config.

The container reads this metadata.

It instantiates the beans and wires them together using Dependency Injection.

Beans are ready to use — fully configured and connected.

🧩 Step-by-Step Code Example

1. Create a Service Class

@Service

public class GreetingService {

    public String getGreeting() {

        return "Hello from Spring!";

    }

}

✅ @Service tells Spring: “Hey, this is a bean. Please manage it.”

2. Use the Service in a Controller

@RestController

public class GreetingController {

    @Autowired

    private GreetingService greetingService;

    @GetMapping("/greet")

    public String greet() {

        return greetingService.getGreeting();

    }

}

✅ @Autowired tells Spring: “Please give me the GreetingService bean.”

3. Run the Application

@SpringBootApplication

public class MySpringApp {

    public static void main(String[] args) {

        SpringApplication.run(MySpringApp.class, args);

    }

}

✅ @SpringBootApplication starts the Spring Container, which:

Finds all @Service, @Controller, etc.

Creates objects for them

Connects them automatically

🧠 What Just Happened?

You didn’t write new GreetingService() anywhere.

Spring Container did it for you.

It also injected it into GreetingController.