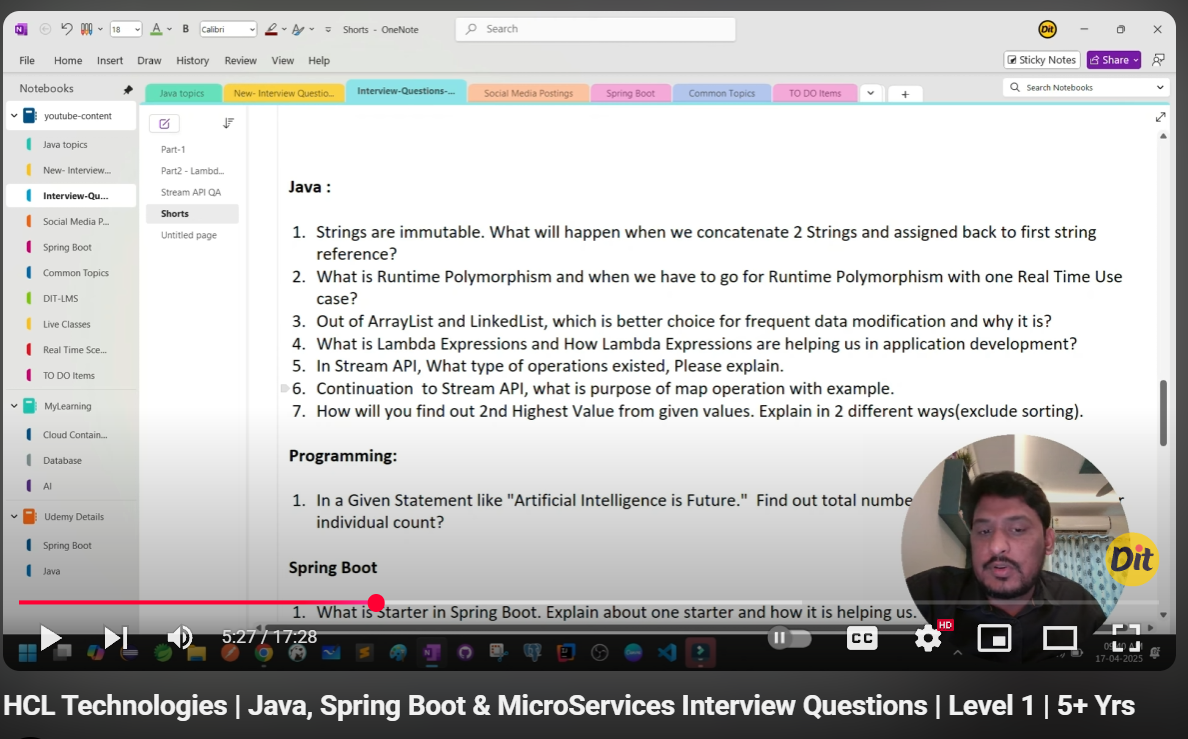
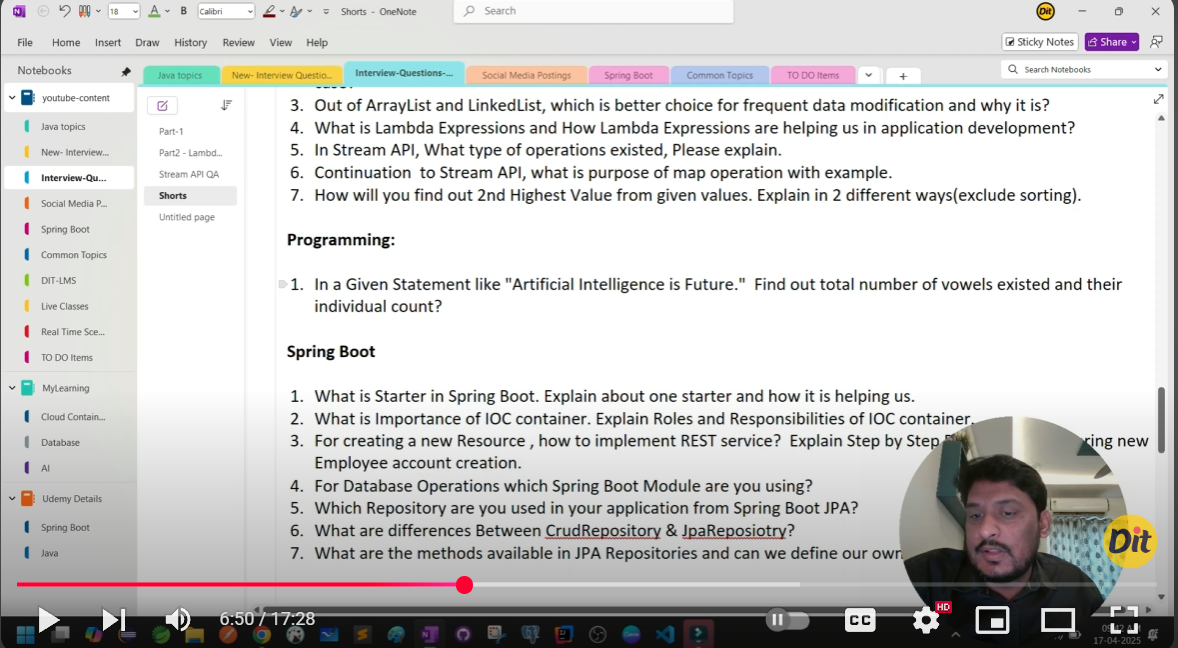
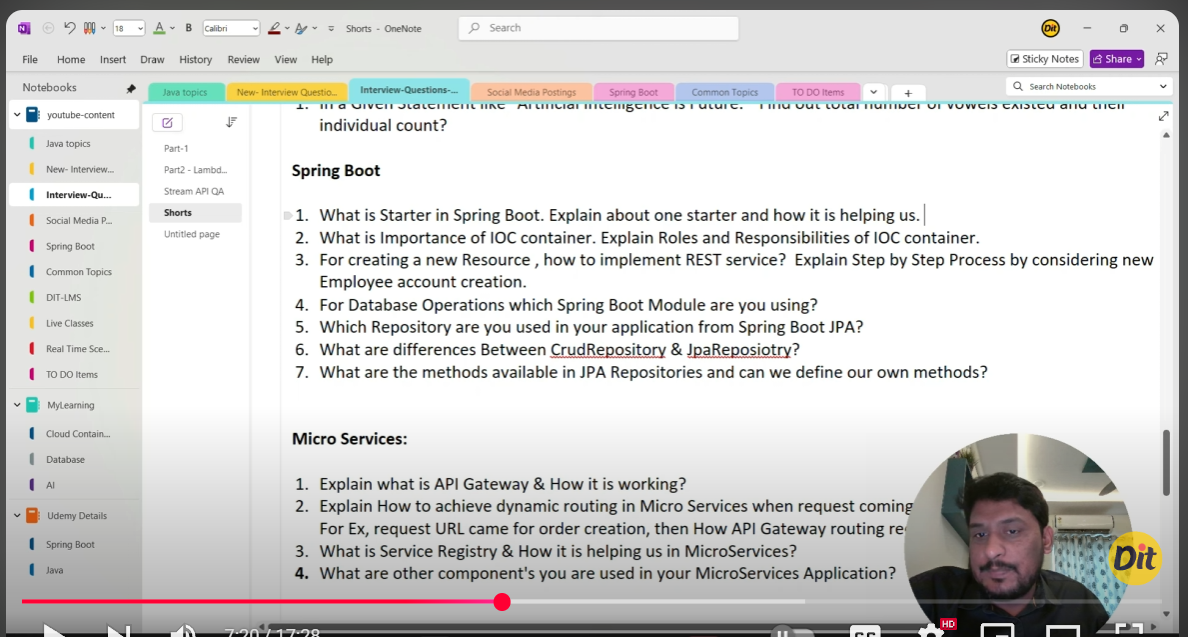
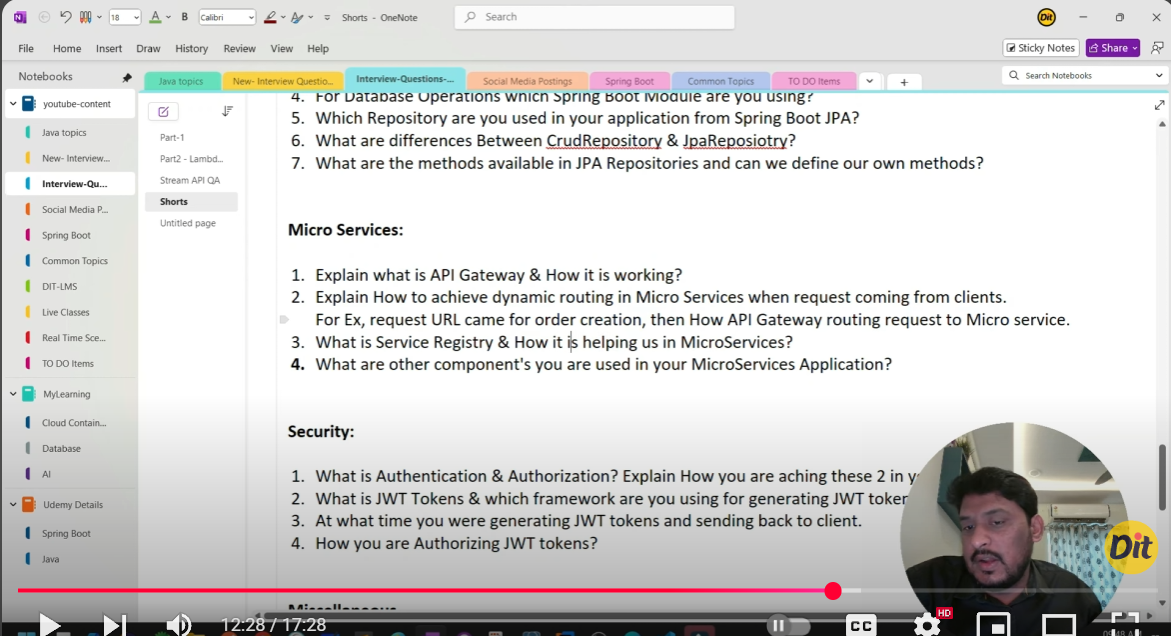
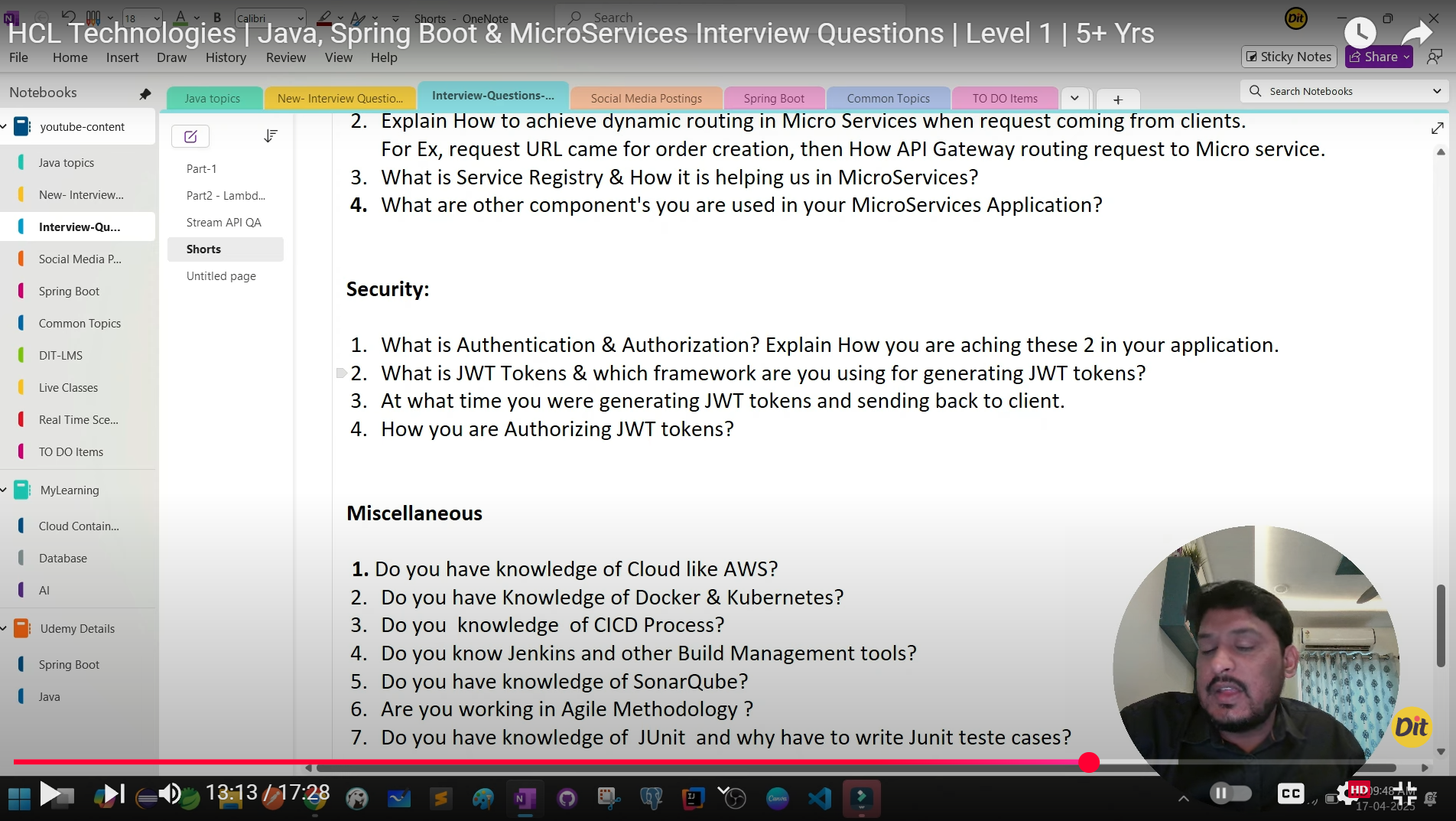
****

****

****

****

****

**✅ Java**

**1. Strings are immutable. What will happen when we concatenate 2 Strings and assign back to the first string reference?**

Yes, that's correct. In Java, String is immutable, so when we concatenate two strings and assign the result to the original reference, a **new String object is created in memory**, and the original one becomes eligible for garbage collection if no other reference points to it.

Example:

String str = "Hello";

str = str + " World";  *// New object "Hello World" created*

**2. What is Runtime Polymorphism and when do we have to go for it? One real-time use case?**

Runtime Polymorphism refers to **method overriding**, where the decision about which method to call is made at runtime based on the actual object.  
**Runtime Polymorphism** allows **method overriding** where a **child class provides a specific implementation** of a method already defined in its **parent class**.

Example Use Case:  
In a **Payment Gateway**, you might have:

class Payment { void process() {} }

class CreditCardPayment extends Payment { void process() { */\* CC Logic \*/* } }

class UpiPayment extends Payment { void process() { */\* UPI Logic \*/* } }

Client code works with Payment reference, actual logic is **decided at runtime** based on the object type.

**3. Out of ArrayList and LinkedList, which is a better choice for frequent data modification and why?**

For frequent insertions and deletions, **LinkedList is a better choice** because:

* It provides constant-time add() and remove() from the beginning or middle,
* Whereas ArrayList shifts elements, which is costly (O(n)).

**4. What is a Lambda Expression and how does it help in application development?**

Lambda expressions are a concise way to represent **functional interfaces** using an inline function.**Lambda Expressions** provide a **concise way** to write **anonymous functions** or **functional interfaces**.  
They help in **cleaner, more readable code**, especially with **streams**, **event handling**, and **collection processing**.

Example:

List<String> list = Arrays.asList("a", "b", "c");

list.forEach(item -> System.out.println(item));

**Helps by**:

* Reducing boilerplate code
* Improving readability
* Enabling functional programming (Streams, Map, Filter)

**5. In Stream API, what type of operations exist? Please explain.**

There are two types of operations in Stream API:

* **Intermediate operations** like map(), filter(), and sorted(). These are lazy and don’t process data until a terminal operation is invoked.
* **Terminal operations** like collect(), count(), and forEach(), which trigger the stream pipeline.

**6. What is the purpose of map() operation in streams? Give an example.**

map() is used to **transform elements** in a stream.  
For instance, if I have a list of names and want their lengths:

java

CopyEdit

List<Integer> lengths = names.stream().map(String::length).collect(Collectors.toList());

This is useful when we need to convert or restructure data.

**7. How will you find the 2nd highest value from given values? Explain in 2 different ways (excluding sorting).**

Yes, I can do this in two ways:

**Method 1: Using two variables**

int first = Integer.MIN\_VALUE, second = Integer.MIN\_VALUE;

for (int num : arr) {

    if (num > first) {

        second = first;

        first = num;

    } else if (num > second && num != first) {

        second = num;

    }

}

import java.util.Scanner;

public class SecondLargestInArray {

public static void main(String[] args) {

// Step 1: Read the size and elements of the array from the user

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the size of the array: ");

int size = scanner.nextInt();

int[] array = new int[size];

System.out.println("Enter the elements of the array:");

for (int i = 0; i < size; i++) {

array[i] = scanner.nextInt();

}

// Step 2: Find the second largest number

int largest = Integer.MIN\_VALUE;

int secondLargest = Integer.MIN\_VALUE;

for (int i = 0; i < size; i++) {

if (array[i] > largest) {

secondLargest = largest;

largest = array[i];

} else if (array[i] > secondLargest && array[i] != largest) {

secondLargest = array[i];

}

}

// Step 3: Display the result

if (secondLargest == Integer.MIN\_VALUE) {

System.out.println("There is no second largest number in the array.");

} else {

System.out.println("The second largest number is " + secondLargest);

}

}

}

[**https://www.rameshfadatare.com/java-programming/java-program-to-find-the-second-largest-number-in-an-array/**](https://www.rameshfadatare.com/java-programming/java-program-to-find-the-second-largest-number-in-an-array/)

**Method 2: Using TreeSet**

TreeSet<Integer> set = new TreeSet<>(List.of(arr));

set.pollLast(); *// removes highest*

int secondHighest = set.last();

import java.util.\*;

public class SecondHighestUsingTreeSet {

    public static void main(String[] *args*) {

        int[] arr = {5, 3, 8, 6, 8, 4, 9, 7};  *// Example input*

        TreeSet<Integer> set = new TreeSet<>();

        for (int num : arr) {

            set.add(num);  *// TreeSet removes duplicates and sorts automatically*

        }

        if (set.size() < 2) {

            System.out.println("No second highest value available.");

        } else {

            set.pollLast();  *// Remove highest*

            int secondHighest = set.last();  *// Get second highest*

            System.out.println("Second Highest: " + secondHighest);

        }

    }

}

Second Highest: 8

* **TreeSet** automatically **removes duplicates** and **sorts the elements in ascending order**.
* **pollLast()** removes the **largest (highest) element**.
* **last()** then **gives the second highest**.

import java.util.\*;

public class SecondHighestFinder {

    public static void main(String[] *args*) {

        int[] arr = {5, 3, 8, 6, 8, 4, 9, 7};  *// Example input*

*// Call separated logic method*

        Optional<Integer> secondHighest = findSecondHighest(arr);

*// Display result*

        if (secondHighest.isPresent()) {

            System.out.println("Second Highest: " + secondHighest.get());

        } else {

            System.out.println("No second highest value available.");

        }

    }

*/\*\**

*\* This method returns the second highest number using TreeSet.*

*\* @param numbers The array of integers*

*\* @return Optional containing second highest if available, otherwise empty*

*\*/*

    public static Optional<Integer> findSecondHighest(int[] *numbers*) {

        TreeSet<Integer> set = new TreeSet<>();

        for (int num : numbers) {

            set.add(num);  *// TreeSet auto-sorts and removes duplicates*

        }

        if (set.size() < 2) {

            return Optional.empty();  *// Not enough unique elements*

        }

        set.pollLast();  *// Remove the highest*

        return Optional.of(set.last());  *// Return second highest*

    }

}

**✅ Programming**

**1. In the statement "Artificial Intelligence is Future.", find the total number of vowels and their individual count.**

Yes. Here's a simple Java logic:

Map<Character, Integer> count = new HashMap<>();

String str = "Artificial Intelligence is Future.".toLowerCase();

for (char ch : str.toCharArray()) {

    if ("aeiou".indexOf(ch) >= 0) {

        count.put(ch, count.getOrDefault(ch, 0) + 1);

    }

}

This gives total and individual counts.

✅ **Cleaned Version Using Only Loop-Based Logic**

import java.util.\*;

public class VowelCounter {

    public static void main(String[] *args*) {

        String sentence = "Artificial Intelligence is Future.";

        countVowels(sentence);

    }

    public static void countVowels(String *input*) {

        Map<Character, Integer> vowelCounts = new HashMap<>();

        String vowels = "aeiou";

        int totalVowelCount = 0;

        for (char c : input.toLowerCase().toCharArray()) {

            if (vowels.indexOf(c) != -1) {

                vowelCounts.put(c, vowelCounts.getOrDefault(c, 0) + 1);

                totalVowelCount++;  *// Count each vowel occurrence*

            }

        }

        System.out.println("Total Vowels: " + totalVowelCount);

        System.out.println("Individual Counts: " + vowelCounts);

    }

}

✅ **Complete Java Program with Separated Logic**

import java.util.\*;

public class VowelCounter {

    public static void main(String[] *args*) {

        String sentence = "Artificial Intelligence is Future.";

*// Call reusable method*

        Map<Character, Integer> result = countVowels(sentence);

*// Display result*

        int totalVowels = result.values().stream().mapToInt(i -> i).sum();

        System.out.println("Total Vowels: " + totalVowels);

        System.out.println("Individual Counts: " + result);

    }

*// ✅ Reusable Method to Count Vowels*

    public static Map<Character, Integer> countVowels(String *input*) {

        Map<Character, Integer> vowelCounts = new HashMap<>();

        String vowels = "aeiou";

        for (char c : input.toLowerCase().toCharArray()) {

            if (vowels.indexOf(c) != -1) {

                vowelCounts.put(c, vowelCounts.getOrDefault(c, 0) + 1);

            }

        }

        return vowelCounts;

    }

}

**✅ Spring Boot**

**1. What is a Starter in Spring Boot? Explain one and how it helps.**

A Starter in Spring Boot is a **pre-configured dependency bundle** for specific functionality.  
For example, spring-boot-starter-web includes **Spring MVC**, **Tomcat**, and **Jackson**—everything needed to build a REST API.

This helps reduce configuration and dependency management overhead.

**2. What is the importance of the IoC container? What are its roles and responsibilities?**

The IoC (Inversion of Control) container:

* **Manages bean lifecycle**
* **Resolves dependencies via constructor or field injection**
* **Provides loose coupling**

It lets developers focus on business logic instead of object creation or wiring.

**3. How do you implement a REST service to create a new resource (e.g., an Employee)?**

Here’s how I do it:

1. **Define the Employee entity** with fields like name, dept, etc.
2. **Create an EmployeeRepository** extending JpaRepository.
3. **Implement a service layer** to handle business logic.
4. **Expose a POST endpoint** in the controller using @PostMapping.
5. Use @RequestBody to accept JSON input.

**4. For database operations, which Spring Boot module do you use?**

I use **Spring Data JPA**, which abstracts boilerplate JDBC code and integrates with Hibernate ORM.

**5. Which repository do you use in Spring Boot JPA?**

I typically use **JpaRepository<Employee, Long>** for full CRUD + pagination support.

**6. What are the differences between CrudRepository and JpaRepository?**

* CrudRepository provides basic CRUD (save, delete, findById)
* JpaRepository extends CrudRepository and adds **pagination, batch operations**, and **JPQL support**

**7. What methods are available in JPA Repositories and can you define your own?**

Yes. We get built-in methods like:

* findById, save, delete, findAll

We can also define custom methods like:

* findByEmail(String email)
* findByStatusAndDept(String status, String dept)

Spring parses method names and generates queries automatically.

**✅ Microservices**

**1. What is API Gateway and how does it work?**

An API Gateway is the **entry point for all client requests** in a microservices architecture.  
It handles:

* **Routing**
* **Authentication**
* **Rate limiting**
* **Load balancing**

We used **Spring Cloud Gateway**, which routes requests to downstream services dynamically using **service discovery**.

**2. How do you achieve dynamic routing in microservices? For example, routing order creation requests.**

We use **Spring Cloud Gateway** + **Eureka Service Registry**.

When a client sends a request to /order/create, the Gateway:

* Looks up the **Order Service** in Eureka.
* Forwards the request to the actual **Order Service instance** dynamically.

**3. What is a Service Registry and how does it help in microservices?**

A **Service Registry** like **Eureka** maintains a **list of available microservices**.  
It allows services to **discover each other** without hardcoding URLs.  
It enables **load balancing** and **dynamic scaling**.

**4. What other components do you use in your microservices application?**

I’ve used:

* **Spring Cloud Config** for centralized configuration
* **Kafka** for asynchronous messaging
* **Resilience4j** for circuit breaker
* **Zipkin** for distributed tracing
* **Spring Boot Admin** for service monitoring

**✅ Security**

**1. What is Authentication and Authorization? How do you achieve these in your application?**

* **Authentication** = Verifying who the user is (e.g., JWT login)
* **Authorization** = What the user is allowed to access (e.g., role-based)

We achieve this using:

* **Spring Security + JWT tokens**
* Custom UserDetailsService for loading users
* Secure endpoints with @PreAuthorize("hasRole('ADMIN')")

**2. What is JWT and which framework do you use to generate it?**

JWT (JSON Web Token) is a compact token for **stateless authentication**.

I use:

* **jjwt** or **Spring Security OAuth2** to generate and validate tokens.

**3. When do you generate JWT tokens and send to client?**

After **successful login**, the server creates a JWT and sends it to the client via:

* HTTP Response Header or
* JSON body

Client uses this token in future requests.

**4. How do you authorize JWT tokens?**

In each request:

* We **intercept the token using a filter**
* **Validate the token signature and expiry**
* **Extract roles/claims**
* Then allow or deny access using SecurityContextHolder

**✅ Miscellaneous**

**1. Do you have knowledge of AWS?**

Yes, I’ve worked with:

* **EC2** for hosting
* **S3** for object storage
* **RDS** for databases
* **EKS** for Kubernetes-based microservice deployment

**2. Do you have knowledge of Docker and Kubernetes?**

Yes. I containerize services using **Docker**,  
and manage deployments using **Kubernetes**, including:

* YAML configurations
* Secrets management
* Rolling updates

**3. Do you know about CI/CD?**

Yes, I’ve implemented **CI/CD pipelines** using:

* **GitLab CI**
* **Jenkins**
* Automated build, test, and deploy to staging/prod environments

**4. Do you know Jenkins and build management tools?**

Yes. I use:

* **Jenkins for pipeline automation**
* **Maven and Gradle** for dependency and build management

**5. Do you have knowledge of SonarQube?**

Yes, I’ve used **SonarQube** in CI pipelines to check:

* Code quality
* Duplications
* Vulnerabilities

**6. Are you working in Agile methodology?**

Yes. I work in a **Scrum team** with:

* **Daily standups**
* **Sprint planning**
* **Retrospectives**
* **JIRA** for tracking stories
* **backlog grooming**.

**7. Do you use JUnit and why do we write test cases?**

Yes. I write:

* **JUnit tests** for unit testing and Integration Testing
* **Mockito** for mocking dependencies

Writing tests:

* Helps in **early bug detection** and **maintaining code quality**.