Cloud Native Full Stack Application Development – II

VIVA Question:

UNIT - I

1. Q1: How can you create a new Spring Boot project using Spring Boot Initializer?
2. Q2: What is the purpose of the pom.xml file in a Spring Boot project, and how does it relate to project dependencies?
3. Q3: Explain the concept of Inversion of Control (IoC) and how it is implemented in Spring.
4. Q4: What is Dependency Injection (DI), and why is it essential in the context of Spring framework?
5. Q5: How do you configure the Spring container for Dependency Injection using XML configuration?
6. Q9: Compare and contrast Java-based configuration and XML-based configuration in Spring.
7. Q10: How do you apply all Java configuration to Spring Beans without using XML?
8. Q11: Demonstrate how to create a simple Spring Boot application with multiple beans wired together using annotations.
9. Q12: Explain a scenario where you would prefer Java-based configuration over XML configuration in a Spring Boot project.
10. Q13: What is the purpose of the @ComponentScan annotation in Spring Boot, and how does it relate to auto-scanning?
11. Q14: Discuss the role of the @Qualifier annotation in resolving bean autowiring conflicts.
12. Q15: How would you troubleshoot issues related to bean wiring and dependency injection in a Spring Boot application?
13. How does Spring Boot simplify the development of Spring applications?
14. Discuss some key features of Spring Boot that make it a popular choice for developers.

UNIT – II

1. Explain the role of @RestController in a Spring Boot application.
2. How does @RequestMapping annotation work in the context of a RESTful controller?
3. Discuss the differences between @RestController and @Controller in Spring MVC.
4. What is the role of validation in a RESTful API, and why is it important?
5. Discuss the annotations provided by Spring for basic validation in RESTful APIs.
6. How does validation contribute to the overall reliability of an API?
7. Explain how you would validate a request payload for creating a new post.
8. How can you customize validation constraints for updating a post?
9. Describe the process of customizing validation response messages in Spring Boot.
10. How can you handle validation errors and provide meaningful error messages to clients?
11. Apply the validation principles to create and update comment endpoints.
12. Discuss any specific challenges or considerations when validating comment-related requests.
13. How can you define users and their roles in a Spring Boot Security application?
14. Discuss the importance of roles in the context of access control.
15. What is JWT, and how does it relate to user authentication in Spring Boot Security?
16. How can you restrict access to specific URLs based on user roles using JWT?
17. Explain the concept of role-based authorization and how it is implemented in Spring Boot Security.
18. Provide an example scenario where role-based authorization is crucial in a RESTful API.

UNIT -III

1. Define persistence in the context of software development.
2. Why is persistence important in long-term data storage?
3. Briefly explain the concept of a relational database.
4. What are the key components of a relational database?
5. How can Java applications interact with a relational database using SQL?
6. Name a popular Java library for working with SQL databases.

7. Explain how object-oriented applications handle persistence.
8. Why is it important to bridge the gap between object-oriented models and relational databases?
9. Define ORM and its purpose.
10. Why is ORM considered a solution for the impedance mismatch problem?
11. Discuss some common challenges faced when working with Object/Relational Mapping.
12. How does ORM address these challenges?
13. Explain the advantages of using ORM in a software project.
14. Are there any drawbacks or limitations to using ORM?
15. What is Hibernate, and how does it relate to ORM?
16. How can Hibernate be integrated into a Java project?
17. What is the significance of annotating a domain class as an entity in JPA? - Provide an example of how to annotate a class as an entity in Spring Data JPA.
18. What is a JPA repository, and why is it useful? -
19. How do you declare a JPA repository in a Spring Data JPA application?

UNIT-IV

Containerization with Docker

1. Q: What is virtualization, and how does it differ from containerization?
2. Q: Can you name some popular virtualization technologies other than Docker?
3. Q: Explain the key components of Docker.
4. Q: What is the purpose of Docker Hub?
5. Q: How do you start a Docker container?
6. Q: What command is used to stop a running container?
7. Q: What is Docker Compose, and when is it useful?
8. Q: How do you define services in a Docker Compose file?
9. Q: Why would you containerize a Spring Boot application?
10. Q: What is the role of a Dockerfile in building a Docker image?
11. Q: How do you copy files into a Docker image using a Dockerfile?
12. Q: Explain the difference between an image and a container.
13. Q: How do you tag a Docker image?
14. Q: What command is used to run a Docker container from an image?
15. Q: How can you expose ports from a Docker container to the host machine?

Kubernetes

1. Q: How do you check the status of nodes in a Kubernetes cluster?
2. Q: Why is a multi-node cluster beneficial in Kubernetes?
3. Q: Can you name some common tools for setting up Kubernetes clusters?
4. Q: What is a Kubernetes namespace, and why is it useful?
5. Q: How do you scale a Deployment in Kubernetes?
6. Q: What is the purpose of an Ingress controller in Kubernetes?
7. Q: How do you define Ingress rules for routing traffic to different services?
8. Q: What is a StatefulSet, and in what scenarios would you use it?

9. Q: What are the advantages of deploying a Spring Boot application on Kubernetes?
10. Q: How do you update a deployed application in a Kubernetes cluster?

UNIT – V

1. **What is a microservice?**
    - *Answer:* A microservice is a small, independent, and self-contained service that performs a specific business function within a larger software application. Microservices architecture involves breaking down an application into a collection of loosely coupled, independently deployable services.
2. **What is Spring, and why is it relevant to microservices?**
    - *Answer:* Spring is a framework for building enterprise Java applications. It is relevant to microservices because it provides a comprehensive set of tools and libraries for developing scalable, maintainable, and loosely coupled components. Spring Boot, a part of the Spring framework, simplifies the process of building microservices by offering convention over configuration and automatic setup.
3. **Why change the way we build applications with microservices?**
    - *Answer:* Traditional monolithic applications can become complex and challenging to maintain as they grow. Microservices offer advantages such as improved scalability, flexibility, and ease of deployment. By breaking down applications into smaller services, development teams can work more independently, and applications become more modular.
4. **What exactly is the cloud?**
    - *Answer:* The cloud refers to a network of remote servers hosted on the internet that store, manage, and process data rather than on a local server or a personal computer. Cloud computing provides on-demand access to computing resources, including storage, processing power, and services, often with a pay-as-you-go model.
5. **Why the cloud and microservices?**
    - *Answer:* The cloud complements microservices by providing a scalable and flexible infrastructure. Microservices can be easily deployed and scaled horizontally in the cloud environment. Cloud services also offer various tools and services that enhance the development, deployment, and maintenance of microservices.
6. **What are the core microservice development patterns?**
    - *Answer:* Core microservice development patterns include:
        - **Service Independence:** Each microservice is independent and has its own database.
        - **Decentralized Data Management:** Each microservice manages its own data.
        - **Automated Deployment:** Microservices are deployed independently.
7. **Explain microservice routing patterns.**
    - *Answer:* Microservice routing patterns involve directing requests from clients to the appropriate microservices. Examples include API Gateway pattern, where a single entry point handles client requests and routes them to the corresponding microservices based on the request.
8. **Describe microservice client resiliency patterns.**
    - *Answer:* Microservice client resiliency patterns include strategies to handle failures gracefully. Examples include circuit breaker pattern, where a microservice can detect

failures and prevent cascading failures in the system by short-circuiting requests.

9. **What are microservice security patterns?**

    Answer: Microservice security patterns address security concerns in a distributed system. Examples include token-based authentication, where each microservice validates authentication tokens, and role-based access control to control access based on user roles.

10. **Explain microservice logging and tracing patterns.**

    Answer: Microservice logging and tracing patterns involve capturing and analyzing logs and traces to monitor and troubleshoot microservices. Examples include centralized logging and distributed tracing to gain insights into the behavior of microservices.

11. **What are microservice build/deployment patterns?**

    Answer: Microservice build/deployment patterns include techniques for building and deploying microservices. Examples include containerization with Docker and using container orchestration tools like Kubernetes for managing and scaling containers.

12. **How does Spring Cloud contribute to building microservices with Spring Boot?**

    Answer: Spring Cloud provides a set of tools and libraries that complement Spring Boot for building microservices. It includes features like service discovery, distributed configuration management, circuit breakers, and more to simplify the development of microservices.