𝗧𝗼𝗽 𝟯𝟬 𝗦𝗽𝗿𝗶𝗻𝗴 𝗕𝗼𝗼𝘁 𝗾𝘂𝗲𝘀𝘁𝗶𝗼𝗻𝘀 𝗰𝗼𝗺𝗺𝗼𝗻𝗹𝘆 𝗮𝘀𝗸𝗲𝗱 𝗶𝗻 𝗶𝗻𝘁𝗲𝗿𝘃𝗶𝗲𝘄𝘀 𝗳𝗼𝗿 𝗝𝗮𝘃𝗮 𝗱𝗲𝘃𝗲𝗹𝗼𝗽𝗲𝗿𝘀 𝗼𝗿 𝘀𝗼𝗳𝘁𝘄𝗮𝗿𝗲 𝗲𝗻𝗴𝗶𝗻𝗲𝗲𝗿𝘀 𝘄𝗶𝘁𝗵 𝟮+ 𝘆𝗲𝗮𝗿𝘀 𝗼𝗳 𝗲𝘅𝗽𝗲𝗿𝗶𝗲𝗻𝗰𝗲:  
1. What is Spring Boot, and how does it differ from the traditional Spring Framework?  
2. What is the purpose of @̲𝚂̲𝚙̲𝚛̲𝚒̲𝚗̲𝚐̲𝙱̲𝚘̲𝚘̲𝚝̲𝙰̲𝚙̲𝚙̲𝚕̲𝚒̲𝚌̲𝚊̲𝚝̲𝚒̲𝚘̲𝚗̲ annotation?  
3. Explain the concept of auto-configuration in Spring Boot.  
4. What is the role of [**application.properties**](http://application.properties/) or application.yml in Spring Boot?  
5. What are the advantages of using Spring Boot over traditional Spring applications?  
̲@̲𝚁̲𝚎̲𝚙̲𝚘̲𝚜̲𝚒̲𝚝̲𝚘̲𝚛̲𝚢̲,̲ and @̲𝙲̲𝚘̲𝚗̲𝚝̲𝚛̲𝚘̲𝚕̲𝚕̲𝚎̲𝚛̲ in Spring Boot? ̲@̲𝚂̲𝚎̲𝚛̲𝚟̲𝚒̲𝚌̲𝚎̲,̲ 6. What is the difference between @̲𝙲̲𝚘̲𝚖̲𝚙̲𝚘̲𝚗̲𝚎̲𝚗̲𝚝̲,̲  
7. How do you create a Spring Boot RESTful web service?  
8. What is Spring Boot Actuator, and what are its uses?  
9. What is the significance of @̲𝚁̲𝚎̲𝚜̲𝚝̲𝙲̲𝚘̲𝚗̲𝚝̲𝚛̲𝚘̲𝚕̲𝚕̲𝚎̲𝚛̲ and how is it different from @̲𝙲̲𝚘̲𝚗̲𝚝̲𝚛̲𝚘̲𝚕̲𝚕̲𝚎̲𝚛̲?  
10. How do you handle exceptions in Spring Boot applications?  
11. What is Spring Boot DevTools, and what benefits does it provide during development?  
12. How do you use Spring Boot to connect to a database (e.g., MySQL or PostgreSQL)?  
and @̲𝙲̲𝚘̲𝚖̲𝚙̲𝚘̲𝚗̲𝚎̲𝚗̲𝚝̲ in Spring Boot. 13. Explain the difference between @̲𝙱̲𝚎̲𝚊̲𝚗̲  
14. How do you configure logging in Spring Boot?  
15. What are Spring Boot profiles, and how do you manage them for different environments (dev, prod)?  
16. How do you implement security in a Spring Boot application?  
17. What is Spring Boot’s embedded server, and how does it work?  
18. What is the use of @̲𝙴̲𝚗̲𝚊̲𝚋̲𝚕̲𝚎̲𝙰̲𝚞̲𝚝̲𝚘̲𝙲̲𝚘̲𝚗̲𝚏̲𝚒̲𝚐̲𝚞̲𝚛̲𝚊̲𝚝̲𝚒̲𝚘̲𝚗̲ in Spring Boot?  
19. What is the difference between Spring Boot’s [**application.properties**](http://application.properties/) and application.yml files?  
20. What is Spring Boot's support for creating microservices?  
21. How does Spring Boot handle dependency injection?  
22. How do you test a Spring Boot application using @̲𝚂̲𝚙̲𝚛̲𝚒̲𝚗̲𝚐̲𝙱̲𝚘̲𝚘̲𝚝̲𝚃̲𝚎̲𝚜̲𝚝̲?  
̲@̲𝙿̲𝚘̲𝚜̲𝚝̲𝙼̲𝚊̲𝚙̲𝚙̲𝚒̲𝚗̲𝚐̲ and other HTTP method annotations. ̲@̲𝙶̲𝚎̲𝚝̲𝙼̲𝚊̲𝚙̲𝚙̲𝚒̲𝚗̲𝚐̲,̲ 23. Explain the difference between @̲𝚁̲𝚎̲𝚚̲𝚞̲𝚎̲𝚜̲𝚝̲𝙼̲𝚊̲𝚙̲𝚙̲𝚒̲𝚗̲𝚐̲,̲  
24. What are Spring Boot’s default error handling mechanisms, and how can they be customized?  
25. How do you perform batch processing in Spring Boot?  
26. What is Spring Data JPA, and how is it used in Spring Boot?  
27. How do you manage transaction handling in Spring Boot applications?  
28. What are the different ways to run a Spring Boot application?  
29. Explain the Spring Boot logging mechanism with default loggers (e.g., Logback).  
30. How do you configure Spring Boot to send an email (e.g., using JavaMailSender)?

Difference between Spring Framework and Spring Boot  
  
Spring Framework   
-> Allows to develop apps using 3 types of Configs i.e. xml driven configs, Annotations driven cofigs, 100% code driven configs  
-> Programmer explicitly created IOC Container except in Spring MVC application  
-> Does not give embedded server. So to run application we need to arrange web server explicitly.  
-> Does not give any in-memory databases  
-> We need to add dependencies (jar files) manually using gradle/maven  
-> Bit lightweight as compared of Spring Boot because of No AutoConfiguratin support  
  
Spring Boot  
-> Supports only one type of confgs that is through annotations and giving inputs for autoConfigutaration for [**application.properties**](http://application.properties/) file  
-> Programmer does not create IoC Container , rather he gets it by calling [**SpringApplication.run**](http://springapplication.run/)()  
-> Gives Tomcat, Jetty and etc servers as Embedded servers  
->Gives in-memory databases like h2  
-> Bit heavy weight due to AutoConfiguration support which created Objects which might not even needed  
-> Spring boot gives starters which provide main jars, dependent jars and related jars as well.

1. What is a microservice?  
   A microservice is a small, independently deployable service that focuses on a specific business functionality, communicating with other services via lightweight protocols like REST or messaging.  
     
   2. How does microservices architecture differ from monolithic architecture?  
   • Microservices: Decoupled, independently deployable, fault-tolerant, scalable.  
   • Monolithic: Tightly coupled, single deployable unit, harder to scale and maintain.  
     
   3. What is service discovery in microservices?  
   Service discovery allows services to find and communicate with each other dynamically. Tools like Eureka, Consul, and Zookeeper are commonly used.  
     
   4. How do you handle inter-service communication in microservices?  
   • Synchronous: REST APIs, gRPC.  
   • Asynchronous: Message brokers like Kafka, RabbitMQ.  
     
   5. What is API Gateway, and why is it important?  
   An API Gateway acts as a single entry point for all client requests, handling routing, authentication, rate limiting, and monitoring.  
     
   6. How do you ensure data consistency across microservices?  
   • Use sagas for distributed transactions.  
   • Implement eventual consistency with event-driven architectures.  
     
   7. What tools are used for monitoring microservices?  
   Prometheus, Grafana, ELK Stack, Zipkin, and Jaeger are commonly used for logging, monitoring, and distributed tracing.  
     
   8. What is circuit breaking in microservices?  
   Circuit breakers (e.g., Hystrix, Resilience4j) prevent cascading failures by halting requests to failing services and allowing fallback mechanisms.  
     
   9. How do you deploy microservices?  
   Use Docker for containerization and Kubernetes for orchestration. CI/CD pipelines ensure automated deployment and scaling.  
     
   10. What are the challenges of microservices?  
     
   • Managing distributed systems.  
   • Data consistency.  
   • Monitoring and debugging.  
   • Network latency and fault tolerance.

✅ API Gateway: A single entry point for routing, authentication, and more. Simplifies client interactions.  
✅ Database per Service: Promotes data encapsulation, enabling independent scalability and ownership.  
✅ Circuit Breaker: Enhances system resiliency by preventing cascading failures during service outages.  
✅ Service Registry and Discovery: Enables dynamic discovery of service instances for seamless communication.  
✅ Event Sourcing: Records state changes as events, ensuring traceability and consistency.  
✅ CQRS: Separates read and write models to optimize performance in high-demand systems.  
✅ Bulkhead: Isolates failures to ensure one service doesn’t take down the entire system.  
✅ Strangler Pattern: A gradual, safe migration from monolithic to microservices architecture.  
✅ Sidecar: Simplifies cross-cutting concerns (like monitoring) by running helper components alongside services.  
✅ Saga: Manages distributed transactions for eventual consistency using choreography or orchestration

Java HashMap Internal Working :  
  
A HashMap in Java is a part of the Java Collections Framework and is used to store data in key-value pairs. The internal working of a HashMap involves several core concepts such as hashing, buckets, and handling collisions. Below, I'll break down these concepts and explain how they contribute to the functionality of a HashMap.  
  
1. Hashing  
The primary concept in a HashMap is hashing, where the hash function is used to convert the large keys into smaller indices that represent indices in the array. A good hash function aims to distribute keys uniformly across the buckets.  
  
2. Array of Buckets  
A HashMap internally has an array of buckets, where each bucket is used to store entries (nodes containing key-value pairs). The size of the array grows dynamically as the number of entries in the HashMap increases.  
  
3. Structure of a Bucket  
Each bucket contains linked entries. Before Java 8, each bucket was structured as a linked list, but from Java 8 onwards, when the number of items in a bucket reaches a certain threshold, that bucket's linked list is replaced by a balanced tree, which improves the worst-case performance from O(n) to O(log n).  
  
4. Putting an Entry  
When the put(key, value) method is used:  
The HashMap computes the hash of the key.  
This hash is then used to find the bucket where the entry should be stored.  
If the bucket is found to have other entries (collision scenario), a new entry node will be added either to the linked list or tree (depending on the number of entries).  
  
5. Getting an Entry  
When the get(key) method is called:  
The HashMap again computes the hash of the key.  
It uses this hash to find the appropriate bucket.  
If the bucket contains multiple entries (due to previous collisions), the HashMap will either scan through a linked list or tree to find the matching key.  
  
6. Handling Collisions  
Collisions occur when multiple keys are hashed to the same bucket index. Initially, as mentioned, collisions are handled using linked lists. However, if the bucket size grows beyond a certain threshold and the overall treeify threshold is reached, the linked list is converted into a balanced tree to maintain efficient searching time.  
  
7. Resizing  
The HashMap automatically resizes itself when the number of entries in the map exceeds a certain threshold defined as the load factor multiplied by the current capacity. Resizing involves creating a new array of buckets larger than the original and then rehashing all existing keys to new bucket locations.  
  
8. Load Factor  
The load factor is a measure that decides when to increase the capacity of the HashMap to maintain the get/put operation's efficiency. The default load factor is 0.75, which is a good trade-off between time and space costs.