**Plan**

1 Core Java

- OOP Concepts

- Classes

- Objects

- Inheritance

- Polymorphism

- Abstraction

- Encapsulation

- Exception Handling

- Try-catch blocks

- Custom exceptions

- Checked vs. unchecked exception

- Exception hierarchy

- Usage with inheritance

- Multithreading

- Threads class

- Runnable interface

- Thread Life Cycle

- Synchronization

- Synchronization Basics: Ensuring thread safety using synchronized methods and blocks.

- Locks and Synchronization:

- Intrinsic Locks (Monitor Locks): Synchronizing on objects.

- Synchronized Methods: Using the synchronized keyword.

- Synchronized Blocks: Controlling access to code blocks with synchronized.

- Deadlock: Understanding and preventing deadlock situations (when two or more threads are blocked forever waiting for each other).

- Livelock and Starvation: Differences between deadlock, livelock, and starvation.

- Concurrency Utilities

- Java Concurrency Package: Using java.util.concurrent for advanced thread management.

- Executor Framework:

- ExecutorService: Managing and controlling thread execution.

- Executors: Factory methods to create thread pools (e.g., newFixedThreadPool, newCachedThreadPool).

- Callable and Future: Submitting tasks that return results and handling asynchronous computation.

- Scheduled Executors: Using ScheduledExecutorService to schedule tasks with fixed-rate or fixed-delay execution.

- Concurrent Collections

- Java Collections Framework: Using thread-safe collections from java.util.concurrent package.

- ConcurrentHashMap: A thread-safe implementation of Map.

- CopyOnWriteArrayList and CopyOnWriteArraySet: Collections with thread-safe modifications.

- BlockingQueue: Interfaces and implementations like ArrayBlockingQueue, LinkedBlockingQueue, PriorityBlockingQueue.

- Thread Communication

- Wait and Notify:

- wait(): Making a thread wait until it is notified.

- notify() and notifyAll(): Waking up threads waiting on an object's monitor.

- Condition Variables: Using Condition interface for advanced thread communication.

- Thread Safety

- Atomic Variables: Using AtomicInteger, AtomicLong, AtomicReference for lock-free thread-safe operations.

- Volatile Keyword: Ensuring visibility of changes to variables across threads.

- Fork/Join Framework

- Introduction: Parallel programming framework designed to take advantage of multi-core processors.

- ForkJoinPool: Executing tasks recursively and in parallel.

- RecursiveTask and RecursiveAction: Tasks that return results and tasks that don't return results.

- ThreadLocal

- Purpose: Providing thread-local variables that are isolated from other threads.

- Usage: Creating and using ThreadLocal objects for maintaining thread-specific context.

- Memory Management

- Java Memory Model (JMM)

- Purpose: Defines how threads interact through memory and what behaviors are allowed in concurrent executions.

- Memory Visibility: How changes made by one thread become visible to others.

- Atomicity: Ensures operations on shared variables are performed as a single unit of work.

- Memory Areas

- Heap:

- Purpose: Stores objects and their instance variables.

- Generations: Divided into Young Generation (Eden, Survivor Spaces) and Old Generation.

- Garbage Collection: Most GC activities occur in the heap.

- Stack:

- Purpose: Stores method frames, local variables, and call stacks.

- Frames: Created for each method invocation and destroyed after the method execution.

- Method Area:

- Purpose: Stores class definitions, method information, and static variables.

- Permanent Generation: In older JVM versions (replaced by Metaspace in newer versions).

- Native Method Stack:

- Purpose: Stores native method calls and their local variables.

- Garbage Collection (GC)

- Purpose: Automatically reclaims memory occupied by objects no longer in use.

- Garbage Collection Algorithms:

- Mark-and-Sweep: Marks live objects and sweeps away unreachable objects.

- Generational Garbage Collection: Divides heap into generations and performs GC based on object age.

- Tracing GC: Uses mark-and-sweep and/or mark-and-compact techniques.

- Copying GC: Uses two semi-spaces (Eden and Survivor spaces) to copy live objects.

- Garbage Collectors:

- Serial GC: Simple GC for single-threaded applications.

- Parallel GC: Uses multiple threads for minor and major GC.

- Concurrent Mark-Sweep (CMS) GC: Minimizes pause times by performing most work concurrently.

- G1 (Garbage-First) GC: Aims to balance throughput and pause times by dividing the heap into regions.

- Memory Leaks

- Definition: Situations where objects are no longer used but are still referenced, preventing GC.

- Causes:

- Unintentional References: Collections or caches holding references to objects.

- Static Fields: Holding references to large objects or collections.

- Inner Classes: Holding implicit references to outer classes.

- Detection:

- Profilers: Tools like VisualVM, YourKit, and JProfiler.

- Heap Dumps: Analyzing heap dumps to find excessive or unexpected object retention.

- Stack vs. Heap

- Heap:

- Usage: For dynamic memory allocation of objects.

- Characteristics: Managed by GC, larger and shared among threads.

- Stack:

- Usage: For method calls and local variables.

- Characteristics: Managed by JVM, smaller and thread-local.

- Strong vs. Weak References

- Strong References: Regular references that prevent objects from being collected.

- Soft References: Used for implementing memory-sensitive caches; objects are collected only when memory is low.

- Weak References: Objects are collected more eagerly; useful for canonicalizing or maintaining associations.

- Phantom References: Used for post-mortem cleanup; provides a way to know when an object has been finalized but before it is reclaimed.

- Object Lifecycles

- Creation: Allocation of memory for new objects.

- Use: Object references and interactions.

- Reclamation: When objects are no longer reachable, they become eligible for garbage collection.

- Tuning and Optimization

- Heap Size Configuration: Setting initial (`-Xms`) and maximum (`-Xmx`) heap sizes.

- GC Tuning: Adjusting GC parameters for performance (e.g., `-XX:+UseG1GC`).

- Memory Profiling: Analyzing memory usage and optimizing code based on profiling data.

- Metaspace (Java 8 and later)

- Purpose: Replaces PermGen to store class metadata.

- Size Management: Configurable via `-XX:MetaspaceSize` and `-XX:MaxMetaspaceSize`.

- Garbage Collection: Managed separately from heap GC.

- Java Heap Dumps

- Generating Heap Dumps: Using tools like `jmap` or JVM options (`-XX:+HeapDumpOnOutOfMemoryError`).

- Analyzing Heap Dumps: Tools like Eclipse MAT or VisualVM to analyze object retention and memory usage.

- String, StringBuilder, StringBuffer, String immutability

- Generics

- Collections Framework

- Java 8 Features

- Lambda Expressions

- Functional Interfaces

- Streams API

- Optional Class

- Default Methods in Interfaces

- Method References

- New Date and Time API

- LocalDate: Represents a date without time (e.g., 2024-08-07).

- LocalTime: Represents a time without date (e.g., 14:30:00).

- LocalDateTime: Combines date and time (e.g., 2024-08-07T14:30:00).

- ZonedDateTime: Represents a date and time with time zone.

- Duration: Represents a time-based amount of time.

- Period: Represents a date-based amount of time.

- Serialization and Reflection API

2. Spring Topics

- Core Container: Beans, Dependency Injection (DI), Bean lifecycle, Bean scopes.

- Spring Boot: Auto-configuration, Starter dependencies, Actuators, Spring Boot CLI, Microservices support.

- Spring MVC: Controllers, ViewResolvers, ModelAndView, Data binding, Form handling.

- Spring Data: Repository abstraction, Custom queries, Paging and sorting.

- Spring Cloud: Config Server, Eureka Discovery, Zuul/Gateway, Circuit Breaker (Hystrix/Resilience4j).

3. Spring AOP Topics

- Concepts: Aspect, Join Point, Advice, Pointcut, Weaving.

- Types of Advice: Before, After, Around, Throws.

- AspectJ vs. Spring AOP: Key differences, Use cases.

- Annotations: @Aspect, @Before, @After, @Around, @Pointcut.

4. Spring Security Topics

- Authentication and Authorization: User details service, Roles and permissions, Security context.

- Security Configuration: WebSecurityConfigurerAdapter, Method Security.

- OAuth2 / JWT: Token-based authentication, OAuth2 flows, JWT handling.

- Spring Security Features: CSRF protection, CORS, Session management.

5. Spring Data JPA Topics

- Repositories: CRUD operations, Query methods, Custom repository methods.

- Entity Mapping: @Entity, @Table, @Id, @Column, @ManyToOne, @OneToMany, @JoinColumn.

- JPQL: Query language, Named queries, Native queries.

- Criteria API: Building dynamic queries, CriteriaBuilder, CriteriaQuery.

6. Hibernate Topics

- Entity Lifecycle: Transient, Persistent, Detached, Removed.

- Mappings: Basic mappings, Associations (One-to-One, One-to-Many, Many-to-One, Many-to-Many).

- Caching: First-level cache, Second-level cache.

- HQL: Hibernate Query Language, Query examples.

- Transaction Management: Session management, Transactions, Isolation levels.

7. Microservices Topics

- Architecture: Service decomposition, Inter-service communication.

- Communication: RESTful APIs, gRPC, Message brokers (Kafka, RabbitMQ).

- Resilience: Circuit breakers, Retry mechanisms, Rate limiting.

- Service Discovery: Eureka, Consul.

- API Gateway: Routing, Aggregation, Security.

- Configuration Management: Spring Cloud Config, Consul.

8. CI/CD with Jenkins

- Jenkins Basics: Pipelines, Jobs, Build triggers.

- Pipeline as Code: Declarative vs. Scripted Pipelines, Jenkinsfile.

- Plugins: Common plugins (Maven, Git, Docker).

- Build and Deployment: Continuous Integration, Continuous Deployment, Automated tests.

9. AWS Topics

- Core Services: EC2, S3, RDS, Lambda, IAM.

- Deployment and Monitoring: CloudFormation, CloudWatch, Elastic Beanstalk.

- Networking: VPC, Security Groups, Load Balancers.

- Scaling and Management: Auto Scaling, Elastic Load Balancing.

10. Docker and Containers

- Container Basics: Docker vs. Virtual Machines, Containerization.

- Docker Commands: Building images, Running containers, Managing containers.

- Dockerfile: Creating and optimizing Dockerfiles, Multi-stage builds.

- Docker Compose: Defining and running multi-container applications.

- Container Orchestration: Basics of Kubernetes, Pod management, Service discovery.

- Image Management: Docker Hub, Private registries, Image versioning and tagging

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Core Java**

OOPS:

What are the main principles of object-oriented programming (OOP)?

What are the access modifiers in Java? Explain their significance and provide examples.

Explain the difference between an abstract class and an interface in Java.

How does Java handle multiple inheritance limitations?

Describe the difference between method overloading and method overriding.

What is the significance of the “static” keyword in Java? How does it affect variables and methods?

What is the purpose of the “final” keyword in Java? How is it different from “finally” and finalize?

Explain the concept of inner classes in Java. What are the different types of inner classes?

Describe the concept of anonymous inner classes in Java. How are they used?

Explain the difference between a shallow copy and a deep copy of an object in Java.

Explain the concept of class loading in Java. How does the JVM load and initialize classes?

Explain the concept of autoboxing and unboxing in Java. Provide examples.

Describe the purpose and usage of the “this” keyword in Java. Provide examples.

Explain the concept of method hiding in Java. How does it differ from method overriding?

Discuss the role of the “super” keyword in Java. When and how is it used?

Explain the concept of marker interfaces in Java. Provide examples of built-in marker interfaces.

Exception Handling:

Explain the concept of exception handling in Java. How does it help in managing errors?

What are the different types of exceptions in Java? Provide examples of each.

Explain the difference between checked and unchecked exceptions in Java. When would you use each one?

Explain the purpose and usage of the assert keyword in Java. How does it help in debugging?

Explain the concept of the try-with-resources statement in Java. How does it handle resource management?

Describe the differences between the throw and throws keywords in Java exception handling.

Multithreading

What is the purpose of the “synchronized” keyword in Java? How does it relate to multithreading?

How do you create and start a new thread in Java? What are the different ways to achieve this?

How does Java support multithreading? What are the different mechanisms provided by Java for concurrent programming?

Explain the purpose and usage of the transient and volatile keywords in Java.

Discuss the advantages and disadvantages of using the synchronized keyword in Java multithreading.

Memory Management:

How does garbage collection work in Java? What are the different types of garbage collectors?

Discuss the concept of Java memory management. What is the difference between stack and heap memory?

What are the Java Memory Model and its main principles? How does it ensure thread safety?

Discuss the purpose and usage of the finalize() method in Java. When would you use it?

String, StringBuffer and String builder

Describe the difference between StringBuffer and StringBuilder in Java.

Discuss the differences between String, StringBuilder, and StringBuffer in Java. When would you use each one?

Discuss the advantages of using the StringBuilder class over concatenating strings using the “+” operator.

Discuss the differences between the StringTokenizer and Split methods for string tokenization in Java.

Generics

Explain the concept of generics in Java. How does it ensure type safety?

Collections and Streams Java8

What are the advantages of using Java collections framework over traditional arrays?

What are the main differences between the “==”, “equals()”, and “hashCode()” methods in Java?

Discuss the concept of method references in Java 8 and how they simplify lambda expressions.

What are lambda expressions in Java? How do they improve code readability and conciseness?

What are functional interfaces in Java? Provide examples and explain their purpose.

Discuss the difference between the java.util.Date and java.time.LocalDate classes in Java 8.

Discuss the benefits of using the java.util.concurrent package for concurrent programming in Java.

What are the different ways to handle concurrent modification exceptions in Java collections?

Discuss the differences between the Comparable and Comparator interfaces in Java. When would you use each one?

What are the features introduced in Java 8 and Java 9? Explain their significance and usage.

Discuss the differences between HashMap and HashTable in Java. When would you use each one?

Describe the purpose and usage of the java.util.Optional class in Java 8.

Discuss the differences between the Vector and ArrayList classes in Java.

Discuss the differences between ArrayList and LinkedList in Java. When would you use each one?

Discuss the differences between the java.util.Stack and java.util.Queue interfaces in Java.

Discuss the differences between the java.util.HashSet and java.util.TreeSet classes in Java.

Describe the role and significance of the “default” keyword in Java interfaces

Describe the differences between HashSet and TreeSet in Java. When would you use each one?

Describe the purpose and usage of the java.util.Arrays class in Java.

Serialization and Deserialization

Describe the concept of serialization in Java. How can you make a class serializable?

How does Java support serialization and deserialization? Explain the Serializable interface.

Reflection API

Explain the concept of reflection in Java. How can you use it to inspect classes and objects?

Explain the concept of the “instanceof” operator in Java. How is it used for type checking?REFLECTION

Other

What are annotations in Java? Provide examples of built-in annotations and explain their usage.

Explain the concept of Java Native Interface (JNI) and its usage for integrating native code with Java programs.

Discuss the principles of immutability and how to create immutable objects in Java.

Discuss the differences between the FileInputStream and FileReader classes in Java.

Describe the purpose and usage of the java.util.Scanner class in Java.

**HashMap:**

How Hashmap is stored in memory — Memory Structure ?

What is Hash Collision during hashmap.put() operation ?

What is the significance of equals() and hashcode() in Hashmap ?

Why the searching time-complexity of HashMap is O(1) ?

What is Hashmap treefy threshold and why it is introduced in Java 8 ?

Difference betwenn Hashtable and Hashmap and also HashTable VS Concurrent HashMap ?

Why does concurrent hashmap and Hash table doesn't allow null keys or values ? refere: https://stackoverflow.com/questions/698638/why-does-concurrenthashmap-prevent-null-keys-and-values

refer this website for more questions: https://mirbozorgi.com/19-most-often-asked-java-hashmap-interview-questions/

4. How do you measure the performance of HashMap?

As a software engineer working in Java, it is essential you know how to measure the performance of HashMap.

When the interviewer asks you this question, you can mention the parameters you use for the performance measurement.

You can elaborate on what these are and how they help.

Example: 'You can measure the performance of HashMap by looking at the parameters that affect it.

These are initial capacity and load factor. Capacity is the number of buckets in the hash table has.

Initial capacity indicates the number of buckets the hash table had at the time of its creation. Load factor is the measure that determines how full the hash table is and when to increase the Map capacity.

The default load factor is 75% of the capacity.'

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Microservices + Spring boot interview 3 years experienced.**

1. What is the logging system do you use in your application ?

a. how do you configure it in your app from scratch ?

2. What is dependency management in spring boot and how does it help us ?

3. If you have mismatching dependencies in your application , what happens ?

4. How do you solve that manually ?

5. Can we disable the default server in the spring boot ? and how do you do that ?

Yes, it is possible to disable the default web server in a Spring Boot application.

By default, Spring Boot uses an embedded web server (like Tomcat), but it's possible to disable it by setting the property spring.main.web-application-type to none in the application.properties file.

Once we disable the webserver , our application can't recieve http requests.

6. How do you disable a specific class or bean from auto configuration ?

7. Difference between rest controller and controller ?

8. Which types of data formats we can return with Rest controller ? is it only JSON ?

9. How do you return XML format from rest controller ?

10. Difference b/w request mapping and get mapping ? and in which scenarios they should be used ?

11. Difference b/w war and embedded containers ?

12. Can we use environment variables in Spring boot application code level ?

13. What is the significance of spring boot dev tools property ?

14. How do you test the spring boot application ?

15. Can you explain the detailed steps of testing spring boot application from scratch ?

16. What is the purpose of unit testing in software development cycle ?

17. What is the code coverage rule in your organization ?

18. What is significance of @SpringBootTest annotation ?

19. What is purpose of pom.XML file in the spring boot application ?

20. What is the significance of @ComponentScan and @Bean annotations in spring boot ?

21. How can u programmatically determine which profile is currently active ?

22. What is significance of @Configuration annotation in spring boot ?

23. What are different log levels and significance of them ?

24. What is the use of spring boot starter parent ?

25. What are the REST API best practices ?

26. What will you keep the status code if you successfully delete a record in your DB ?

27. What is the significance of 400 series status codes in HTTP codes ?

28. Why do we need multi threading in some of our applications ?

29. What are some of best practices in Collections while you use them in your application ?

30. How to test a static method ?

31. What is the use of interface and how does it help us ?

32. How is multiple inheritance achieved in Java ? What is diamond problem ?

33. What is difference b/w abstract class and interface?

34. Did you ever use design patterns in your projects ?

35. Can you explain Leskov Substitution principle ?

36. Which one is better, constructor injection or setter injection ? and why ?

37. What is the static keyword ?

38. Can a static block throw exception ?

39. Can we override and overload a static method ?

40. What is the significance of final keyword in java ?

41. Can we modify a final object reference in java ?

42. How do you handle exceptions in your application ? What are strategies and best practices practices in your app ?

43. Can you share some custom exception names in your application ?

44. Can you brief on java memory model ?

**Some Q for Microservices**

Can you tell us about your experience and tech stacks you have used ?

can you brief us like how did you implement microservices in your project ?

How does the Eureka serve and registry improve the efficiency of Microservices ?

why do we really need service Discovery ?

How does Service discovery helps with cloud based applications compared to appilcations in local ?

If in a case our service discovery is down and all service registry is gone , how do you handle this scenario ?

What is the significance of API gateways ?

How did you implement sucurity in your application ?

How did you implement the role based security for methods ?

What is the method level security and @PreAuthorize annotation significance ?

How did you authenticate application ?

What is OIDC ?

Can u explain me how jwt is structured ?

What does payload part contains in jwt ?

How did your microservices communicate with each other ?

How will you improve communication b/w your microservices to be faster ?

1. use caching

2. use asynchronous mechanism

3. Data base indexing

How did you implement logging in your microservices ?

What is centralized logging and what tools are you using for visualizations in your applicataions ?

What is significance of prometheus ?

What is actuator and why do we need it in our applications ?

What caching techniques did you use in your applications and how did it help your systems ?

What are ORM frameworks did you use for database communications ?

What type of caching did you use for ORM based transactions ?

Lets suppose you have created a rest api and you need to handover it to consumer ? How did you communicate all the required rest api details with consumer, the details

like api url, request body, sample response body and headers ? How did you communicate with them ?

What if I want my consumer to see only 2-3 apis of my application and others need to be hidden ? How can you acheive this ?

Lets suppose your service is providing same rest api to microservices A and B , now B wants some major changes in it and A don't want them (because A's internal code will break)

now how would you achieve this with same API ?

How did you manage profilings in your spring boot application ?

How did you globalize your common properties b/w microservices ?

How does your application gets the latest changes in config files ?

What is the significance of @RefreshScope annotation ?

What CI/CD technique you have used for your applicatin ?

Coding Q's

1. Given an array of integers , remove duplicates and make that unique ?

2. How do you sort an array of integers ?

3. Which algorithm does collections.sort() internally ?

4. How do you find a number is prime or not in fastest way ?

5. Consider a array containing 0's and 1's and now move all 1's to right and zeros to left using java 8

How to create a spring boot project from scratch ?

What are the specific configuration settings required for spring boot application ?

HOw do you manage different profiles in your project ?

Internally how does spring boot manages dependencies ?

How do you integrate spring security from scratch ?

Can we include another spring boot project in our present spring boot app ?

What are the performance and other factors when we do that ?

What is the signifcance of @RestController annotation ?

Can we return custom formats other than json and xml from a rest controller ?

continuation: How do you return a custom format , explain the flow ?

What is the significance of http codes starting with 200, 300, 400 and 500 ?

continuation: What status code would you return if you delete an item successfully ?

What are the embedded servers in spring boot ?

How can improve the performance of embedded server in spring boot application ?

Why do we need swagger and what is the use of it ?

Can we customize the Auto configuration in spring boot ?

continuation: how do you do that ?

How do you design the transactional when working with multiple databases ?

How do you solve the N+1 select problem in hibernate ?

Do you know about json serialization and deserializationi and can we customize it ?

How do you debug when the app in prod level is slower than app deployed in dev level ?

Difference b/w jdk, jre, jvm and jvm architecture ?

Can we install jdk without jre ?

What are the tools avilable in jdk ?

What is memory storage architecture in jvm ?

what is young generation and old generation in jvm ?

What is internal working of hashmap and how it is different before java 8 ?

What is the performance improvement within the hashmap after java 8 ?

Can we run java application without main method?

**Microservices**

A screenshot of a computer

Description automatically generatedA close-up of a computer

Description automatically generatedA diagram of a software company

Description automatically generatedA close-up of a document

Description automatically generatedA screenshot of a computer

Description automatically generatedA close-up of a document

Description automatically generatedA white background with blue text

Description automatically generatedA close-up of a computer

Description automatically generatedA white paper with black text

Description automatically generatedA white background with blue text

Description automatically generatedA white background with blue text

Description automatically generatedA white paper with blue text

Description automatically generatedA white background with red text

Description automatically generated

**Communication in microservices:**

A close-up of a document

Description automatically generatedA close-up of a message

Description automatically generatedA white background with blue text

Description automatically generatedA screenshot of a computer

Description automatically generatedA close-up of a text

Description automatically generatedA close-up of a email

Description automatically generatedA close-up of a message

Description automatically generatedA close-up of a white card

Description automatically generatedA close-up of a message

Description automatically generatedA screenshot of a message

Description automatically generatedA close-up of a document

Description automatically generatedA close-up of a message

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

**SAGA Design pattern:**

A close-up of a document

Description automatically generatedA close-up of a message

Description automatically generatedA white background with blue text

Description automatically generatedA screenshot of a computer

Description automatically generatedA close-up of a text

Description automatically generatedA close-up of a email

Description automatically generatedA close-up of a message

Description automatically generatedA close-up of a white card

Description automatically generatedA close-up of a message

Description automatically generatedA screenshot of a message

Description automatically generatedA close-up of a document

Description automatically generatedA close-up of a message

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

**2types of SAGA:**

1. **Choreography:**

A screenshot of a white background

Description automatically generatedA white paper with blue text

Description automatically generatedA diagram of a message service

Description automatically generatedA screenshot of a white background

Description automatically generated

1. **Orchestrator SAGA:**

A white paper with blue text

Description automatically generatedA close-up of a paper

Description automatically generatedA diagram of a microservice process

Description automatically generatedA screenshot of a computer screen

Description automatically generated