HW#6

1 - What is the difference between manual testing and automated testing?

In manual testing the tests are executed by humans. Tester executes the test cases and generates the test reports without the help of any automation software testing tools or scripts. We can say that it requires to be an experienced tester for to accomplish testing process successfully.

In automated testing test cases are executed with the assistance of software and scripts. It is faster than manual testing because it is done by some automation tools. And there is no chance of any human errors. Automated tests come handy especially while performing repetitive tests. Automation test process is more reliable than manual testing process because it is code and script based.

Manual testing is possible without programming knowledge whereas automation testing is not possible without programming knowledge. Manual testing allows random testing but automation testing doesn't allow it.

2 - What does Assert class do?

Assert class contains assertions which can be used to test our assumptions about the program. Assertions are a kind of statement in java. When an assertion is executed, it is assumed to be true. If the assertion is false, the JVM will throw an *AssertionError*. An assertion is achieved using the assert statement in Java. It is mainly used for testing purposes during development.

3 - How can be tested 'private' methods?

The general opinion about testing private methods is that we shouldn't test private methods, we just should test public methods. If it is necessary to test private methods the first thing we can use is reflection. The second way to test private methods is using a nested test class. And another suggestion for testing private methods is to create a new public class under the same package which contains body of the private method and to test this new class instead of the private method.

In Spring the easiest way to test a private method is to use this method:

ReflectionTestUtils.invokeMethod();

4 - What is Monolithic Architecture?

Monolithic architecture is the traditional unified model for the design of a software program. A monolithic application is built as a single and indivisible unit. It is unified and all the functions are managed and served in one place. In monolithic applications, all the of the components required to run the application are developed and deployed as a single unit, like JAR or WAR files.

Monolithic applications are simple to develop and simple to deploy. Monolithic applications are much easier to debug and test against microservices architecture. But when a monolithic application scales up, it becomes too complicated to understand. Also it is hard to make some changes in monolithic applications because of large and complex structure.

5 - What are the best practices to write a Unit Test Case?

General suggestions about best practice of unit tests are:

- Use a framework for unit testing
- Use automated testing
- Be careful about test case naming convention and use insightful test names
- Create simple, readable and fast tests
- Test only one code unit at a time
- Use assertions but avoid from making unnecessary assertions (use *expected and actual values*)
- Make every test independent
- Aim for the highest code coverage

6 - Why does JUnit only report the first failure in a single test?

The main reason of this is the first failure stops the test. It does it like this to make the testing process simple and easy. JUnit is designed to work best with a number of small tests. Reporting multiple failures in a single test is generally a sign that the test does too much and it is a too big unit test. Even though the failures are more than one JUnit only reports the first failure.

7 - What are the benefits and drawbacks of Microservices?

<u>Advantages:</u>

- Minimal work team
- Scalability
- Modular functionality, independent modules
- Developer freedom to develop and deploy services independently
- Use of containers, allowing for a quick deployment and development of the application

Disadvantages:

- High memory usage
- Time required to fragment different microservices
- Complexity of managing a large number of services
- Developer need to solve problems such as network latency or load balancing
- Complex testing over the distributed deployment

8 - What is the role of actuator in spring boot?

Actuator is a spring boot module that allows to monitor and manage application usages in a production environment, without coding and configuration for any of them. It uses HTTP endpoints to expose operational information about the running application. The actuator endpoints allow us to monitor and interact with the application.

9 - What are the challenges that one has to face while using Microservices?

Design: Organizations face increased complexity when designing microservices. At the start it is hard to define boundaries and connection points between each microservices and the framework to integrate services.

Security: Data security within a microservices-based framework is a well-known concern because of the vulnerable points.

Testing: The testing phase of any software development lifecycle is increasingly complex for microservices-based applications because individual services should be tested independently.

Increased operational complexity: Each service in a microservice application should be deployed and operated independently and maintaining operations creates greater complexity.

Communication: Independently deployed microservices act as miniature standalone applications that communicate with each other. For to create a proper communication between microservices developers have to configure infrastructure layers that enable resource sharing across services. Otherwise they get a non-optimized application with a slow response time.

10 – How independent microservices communicate with each other?

Microservices communicate with each other by inter-service communication on network level. So microservices interact using an inter-process communication protocol such as HTTP, AMQP or a binary protocol like TCP, depending on the nature of each service.

HTTP communication: HTTP calls between services is a feasible approach for service-to-service communication. HTTP has the total control while choosing how services should communicate with each other. HTTP calls between services could be synchronous or asynchronous.

Message communication: In message communication, the participating services do not communicate directly with each other. The services push messages via message broker in order to reach out to other services. So the message broker is the point of contact between all services. This reduces complexity and increases efficiency.

Event-driven communication: Event-driven pattern is another asynchronous approach where coupling between services is completely removed. In an event-driven approach the services need not know about any common message structure. A message broker is still needed here as individual services will write their events to it. Communication happens through events that individual services generate.

11 - What do you mean by Domain driven design?

Domain-driven design (DDD) is a software design approach that simplifies the complexity developers face by connecting the implementation to an evolving model. It is used to build systems that have a complex business domain.

DDD was created to make the domain the focus of design. According to DDD approach there are three principles:

- The primary focus of the project is the core domain and domain logic.
- Complex designs are based on models of the domain.
- Collaboration between technical and domain experts is crucial to creating an application model that will solve particular domain problems.

12 - What is container in Microservices?

Containers are packages of a software which include everything that it needs to run the software, including code, dependencies, libraries, binaries and more. Containers ensure the application to run quickly and reliably from one computing environment to another.

Containers can contain microservices. A microservices architecture does not dictate the use of containers. But using containers for to implement a microservice is a really easy way to use our microservice in a different computing environment in a short time period.

13 - What are the main components of Microservices architecture?

A typical Microservice Architecture should consist of the following components:

- *Clients*: Needs to handle multiple calls to microservice endpoints
- Identity Providers: Authenticates the requests of clients and communicates with API Gateway
- *API Gateway*: Is responsible for request routing, composition and protocol translation for to handle client requests
- *Messaging Formats*: Two types of messages which are Synchronize and Asynchronized
- *Databases*: Captures the data and implements the respective business functionality
- *Static Content*: Is stored on a cloud-based service that can deliver it directly to the client
- *Management*: Is responsible for balancing the services on nodes and identifying failures
- *Service Discovery*: Acts as a guide to microservices to find the route of communication between them as it maintains a list of services on which nodes are located

14 - How does a Microservice architecture work?

A microservices architecture works with a set of small services that run autonomously and independently. A microservice architecture breaks down an application into small parts to create independent APIs for each component, which are then hosted on their own virtual machines (VMs). Each service discovery within the microservice also needs its own database since a shared database is what defines a monolithic architectural style. Having independent databases is what allows these services to be loosely coupled. This kind of architecture is easier to manage and costs less to maintain than one big API.