Umut Yıldız

1) In manual testing, tests are executed by individuals without any program but in automated testing, tests are executed by software or tools. Manual testing is less reliable than automated testing because of human mistakes. Automated testing is faster than manual testing because it spends human resources. Manuel testing is practical when project needs to user friendly or user experiences.

2) An assert statement is defined by the Java keyword assert. In a software, an assert statement is used to specify an actual boolean condition. The condition is tested at runtime if the application is executing with assertions enabled. The Java runtime system throws an AssertionError if the condition is false.

3) It's not a good idea to test private methods independently. If you're attempting to test anything secret, you should first consider why it's private. The public methods to which they are linked invoke private methods. You can determine if its parent public method performs the private function by testing it.

4) Monolithic architecture can be seeming like only one part for apps. We can think like only one app is executed. It is easy to implement but when the project grows, it is not practical. In 'Joseph Ingeno Software Architect's Handbook' [1] the writer says that '*If the application is small, it can be easy to maintain due to the simplicity of the architecture. However, larger and more complex monolithic applications start to suffer in terms of maintainability*.'. The large codebase of a monolithic apps can make it difficult for team members to understand it. Another bad thing is that, when developer wants to deploy a monolithic app with small changes, developer must deploy all app. It wastes the sources of project.

5) We should strive for a code coverage of 100 percent. Our unit tests should be easy to maintain and understand. If the project is changed, the unit tests should be changed as well. We should name the conventions appropriately for optimal code readability. Furthermore, unit tests must be reliable. Our test case should also focus on a single scenario at a time. Our unit tests must be kept separate from other components and dependencies. Finally, automated unit testing should be used.

6) One of the unit test best practices is focus on only single scenario because of that Junit designed for this case. Multiple failures in a single test are usually an indication that the test is too large and accomplishes too much. JUnit is best used with a large number of short tests. Each test is run within its own instance of the test class. Each test results in a failure.

7)When developer deploys his codes, developer deploys only changed microservices because of that it saves the resources. Microservice Architecture is complex than Monolithic architecture that's why it must designed as experienced architects and developers. For example, one backend microservice is code with Java on the other hand another app is implemented with C#. In microservice architecture, it is possible. Because the apps communicate with a standard language and they don't care about another app programming language. Maintainability and scalability are main advantage against monolithic big projects. Another aspect, Microservice architecture needs to DevOps and CI/CD tools.

8) Actuator, in a word, gives our app production-ready functionalities. With this requirement, monitoring our app, getting analytics, comprehending traffic, and keeping a record of the condition of our database become simple. The key advantage of this library is that it allows us to access production-ready tools without having to create them ourselves. Actuator is mostly used to expose

operational data about a running application, such as health, metrics, info, dump, env, and so on. It allows us to connect with it via HTTP endpoints or JMX beans. Once this requirement is added to the class path, we have access to a number of endpoints right away. We can easily customize or expand to explain it in a variety of ways, as we do with most Spring modules.

9) Microservice is a complex system design architecture so requirement analysis and design must be optimal. Also testing of microservices is harder than monolithic. Management of microservices is hard because of that it contains very complex subsystems and also every software team is managed carefully such as agile. Communication is also one of challenges.

10) Microservices have communication types as AMQP, restful api or gRPC and we can divide communication types into two parts such as async and sync. HTTP or gRPC are synchronous messaging types. Client sends a request and server sends a response. In asynchronous messaging types such as RabbitMQ, Redis Pub/Sub, client sends a request and there is no guaranteed return a response.

11) Domain driven design is focus on domain of project. Domain mean is there that the main problems in projects. There are some properties in DDD such as Ubiquitous Language, Bounded Context, Refactoring etc. Ubiquitous Language defines that every developer use and speak same language that is technical term. At the beginning of the project, these terms are defined well and every developer must know that. To explain Bounded Context, domain of the project has many sub domains because DDD is used for very complex and big projects. Bounded Context tells that every sub domain must be divided and explained carefully.

12) Containers are a type of virtualization for operating systems. From a small microservice or software process to a huge application, a single container may operate it all. All of the essential executables, binary code, libraries, and configuration files are contained within a container. There are some advantages of container such as less overhead (less system resources), increased portability, more consistent operation, greater efficiency, better app development(containers supports agile and devops operations)

13)

* Microservices

* Containers

* Service Mesh (Communication)

* Service Discovery

* API Gateway

14) We can think that only one microservice is responsible from only one domain as a monolithic app. Many microservices come together and they build a very complex system. Every unit of this system that is microservices communicate with each other.

REFERENCES

1. Software Architect’s Handbook / Joseph Ingeno