W06 HomeWork – Talha Arıç, talharic@gmail.com

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

When manually testing, the tester validates the key features of a software application. Analysts execute test cases and develop summary error reports without specialized automation tools.

Automation testing helps testers execute more test cases and improve test coverage. When comparing manual vs. automation testing, manual takes longer. Automated testing is more efficient.

Manual testing is very hands-on. It requires analysts and QA engineers to be highly involved in everything from test case creation to actual test execution.

Automation testing involves testers writing test scripts that automate test execution. (A test script is a set of instructions to be performed on target platforms to validate a feature or expected outcome.)

2 – What does Assert class ?

Assertions are utility methods to support asserting conditions in tests; these methods are accessible through the Assert class, in JUnit 4, and the Assertions

In order to increase the readability of the test and of the assertions itself, it's always recommended to import statically the respective class. In this way, we can refer directly to the assertion method itself without the representing class as a prefix.

3 - How can be tested 'private' methods ?

Generally don't unit test private methods directly. Since they are private, consider them an implementation detail. Nobody is ever going to call one of them and expect it to work a particular way.

You should instead test your public interface. If the methods that call your private methods are working as you expect, you then assume by extension that your private methods are working correctly.

4 – What is Monolithic Architecture ?

A monolithic architecture is the traditional unified model for the design of a software program. Monolithic, in this context, means composed all in one piece.

Monolithic software is designed to be self-contained; components of the program are interconnected and interdependent rather than loosely coupled as is the case with modular software programs. In a tightly-coupled architecture, each component and its associated components must be present in order for code to be executed or compiled.

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

·ADOPT A WELL-ORGANIZED TEST PRACTICE

Some organizations write unit tests parallel to the production code of the application. The aim is to have test codes before the production code. When you adopt well-organized test practices like mutation testing, test-driven development, or behavior-driven programming, you can review tests

and production code together. It further helps you understand the code being written and improve the tests accordingly that gives you confidence in the quality of code being shipped.

The approach helps make your unit testing processes scalable and sustainable.

·NAME YOUR TEST WELL

Unit testing is more than ensuring production code works. They are the best documentation to track an application's behavior. Therefore, you must follow naming standards for each test that explicitly tell the intent of barely reading the names of test cases.

·WRITE RELIABLE AND TRUSTWORTHY UNIT TESTS

A simple yet rewarding practice is to write trustworthy and reliable tests. The trustworthiness of unit testing does not just rely on when test units pass the test, but when the test suite reports you about things that go wrong.

·MAKE AUTOMATED UNIT TESTING A RULE

You may feel automated unit testing the most challenging testing discipline, but it ensures rapid feedback. Automated unit testing is also a less expensive way to find errors and fix them.

Even if you can do automated unit testing on many levels, rapid feedback gives you insights about things that matter most to you like: code coverage, modified code coverage, performance, how many tests are being run, and so on. The insight helps you do in-depth analysis and enable you to work more effectively.

·FOCUS ON SINGLE USE-CASE AT A TIME

Good unit testing practices apply to test a single use-case at a time and validate its behavior against the expected output. Adhering to this simple practice helps understand and maintain code being written, easily.

·MINIMAL ASSERTION PER TEST

One requires something more than single use-case tests and good test names to figure out what's being wrong in the unit tests. And that is the use of logical assertion per test. Your test should contain minimal assertion, i.e., validate only output/side effect of the tested method.

·UNIT TEST SHOULD BE ISOLATED

Unit testing follows a standard of testing a single unit.

That means the unit under test needs to be completely isolated from any other unit or dependencies.

A unit should be considered only “testable” when its dependencies can be and are “faked” or “stub” and thus tested, without including its real dependencies or global/external state.

·TRULY UNIT, NOT INTEGRATION

Sometimes, tests that start as unit tests end up as integration tests and make it difficult to isolate issues that occur, calls for more resources to run, and significantly affect the pace of application delivery. Therefore, a unit test and the system under test must abstain from external factors.

·AIM FOR 100% CODE COVERAGE

In unit testing, it is possible to break down into smaller, reusable, and testable components to get 100% coverage. Automated unit testing further helps cover the entire code base. Thus you can expect 100% coverage.

Code coverage is a crucial metric, but be careful and not spend too many efforts for 100% coverage. Even 60% - 80% coverage is a worthy goal to achieve. 100% coverage is not necessary for every scenario. Here comes integration testing that should cover unit testing gaps.

·START USING HEADLESS TESTING IN THE CLOUD

Whether you accept it or not, unit testing simplifies the development and delivery of modern applications. You can start your journey for unit testing by following the tried and tested principles like keeping unit tests reliable, readable, and isolated and then step into newer ones, i.e., headless testing in the cloud.

By adopting the best practices for unit testing as outlined here, you not just enjoy a faster pace of deployment, but highly practical applications.6 - Why does JUnit only report the first failure in a single test ?

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

Reporting multiple failures in a single test is generally a sign that the test does too much, compared to what a unit test ought to do. Usually this means either that the test is really a functional/acceptance/customer test or, if it is a unit test, then it is too big a unit test.

JUnit is designed to work best with a number of small tests. It executes each test within a separate instance of the test class. It reports failure on each test. Shared setup code is most natural when sharing between tests. This is a design decision that permeates JUnit

7 - What are the benefits and drawbacks of Microservices ?

Microservices have become hugely popular in recent years. Mainly, because they come with a couple of benefits that are super useful in the era of containerization and cloud computing. You can develop and deploy each microservice on a different platform, using different programming languages and developer tools. Microservices use APIs and communication protocols to interact with each other, but they don’t rely on each other otherwise.

The biggest pro of microservices architecture is that teams can develop, maintain, and deploy each microservice independently. This kind of single-responsibility leads to other benefits as well. Applications composed of microservices scale better, as you can scale them separately, whenever it's necessary. Microservices also reduce the time to market and speed up your CI/CD pipeline. This means more agility, too. Besides, isolated services have a better failure tolerance. It's easier to maintain and debug a lightweight microservice than a complex application, after all.

As microservices heavily rely on messaging, they can face certain problems. Communication can be hard without using automation and advanced methodologies such as Agile. You need to introduce DevOps tools such as CI/CD servers, configuration management platforms, and APM tools to manage the network. This is great for companies who already use these methods. However, the adoption of these extra requirements can be a challenge for smaller companies.

Having to maintain a network lead to other kinds of issues, too. What we gain on the simplicity of single-responsibility microservices, lose on the complexity of the network. Or, at least a part of it. For instance, while independent microservices have better fault tolerance than monolithic applications, the network has worse.

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

In essence, Actuator brings production-ready features to our application.

Monitoring our app, gathering metrics, understanding traffic, or the state of our database become trivial with this dependency.

The main benefit of this library is that we can get production-grade tools without having to actually implement these features ourselves.

Actuator is mainly used to expose operational information about the running application — health, metrics, info, dump, env, etc. It uses HTTP endpoints or JMX beans to enable us to interact with it.

Once this dependency is on the classpath, several endpoints are available for us out of the box. As with most Spring modules, we can easily configure or extend it in many ways.

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

·Design

Compared to monolithic apps, organizations face increased complexity when designing microservices. Using microservices for the first time, you might struggle to determine:

Each microservice's size

Optimal boundaries and connection points between each microservice

The framework to integrate services

Designing microservices requires creating them within a bounded context. Therefore, each microservice should clarify, encapsulate, and define a specific responsibility.

To do this for each responsibility/function, developers usually use a data-centric view when modeling a domain. This approach raises its own challenge—without logic, the data is nonsensical.

·Security

Microservices are often deployed across multi-cloud environments, resulting in increased risk and loss of control and visibility of application components—resulting in additional vulnerable points. Compounding the challenge, each microservice communicates with others via various infrastructure layers, making it even harder to test for these vulnerabilities.

·Testing

The testing phase of any software development lifecycle (SDLC) is increasingly complex for microservices-based applications. Given the standalone nature of each microservice, you have to test individual services independently.

Exacerbating this complexity, development teams also have to factor in integrating services and their interdependencies in test plans.

10 - How independent microservices communicate with each other?

When we are talking about Monolithic applications, we said that the communication in Monolithic applications are inter-process communication. So that means it is working on single process that invoke one to another by using method calls. You just create class and call the method inside of target module. All running the same process.

This communication gives is very simple but at the same time components are highly coupled with each other and hard to separate and scale independently.

One of the biggest challenge when moving to microservices-based application is changing the communication mechanism. Because microservices are distributed and microservices communicate with each other by inter-service communication on network level. Each microservice has its own instance and process. Therefore, services must interact using an inter-service communication protocols like HTTP, gRPC or message brokers AMQP protocol.

11 - What do you mean by Domain driven design ?

Domain Driven Design (DDD) is a software development approach that deeply connects the ever-changing fundamental business rules in a world of complex requirements.

The main purpose of DDD is; is to produce easily scalable, flexible and highly available software that enables rapid adaptation to rapidly changing requirements. In DDD, the business rules included in the application are logically distributed to the domains. Responsibly, the most related units are kept on the same domain, while other units are moved to different domains. There are basic principles on which this approach is based. I say basic because not all.

12 – What is container in Microservices ?

Containers are a form of operating system virtualization. A single container might be used to run anything from a small microservice or software process to a larger application. Inside a container are all the necessary executables, binary code, libraries, and configuration files. Compared to server or machine virtualization approaches, however, containers do not contain operating system images. This makes them more lightweight and portable, with significantly less overhead. In larger application

deployments, multiple containers may be deployed as one or more container clusters. Such clusters might be managed by a container orchestrator such as Kubernetes.

13 - What are the main components of Microservices architecture ?

·Clients

The client apps usually need to consume functionality from more than one microservice. The client needs to handle multiple calls to microservice endpoints if that consumption is performed directly. Client apps need to be updated frequently, making the solution harder to evolve.

·Identity Providers

The services are fine-grained and lightweight. The Identity Microservice must allow both user-driven and server-to-server access to identity data. Microservices allow applications to be created using a collection of loosely coupled services.

·API Gateway

The API Gateway is responsible for request routing, composition, and protocol translation. It provides each of the application's clients with a custom API. For most microservices-based applications, implementation of an API Gateway is very important for a single-entry point into a system.

·Messaging Formats

Synchronize and Asynchronized are the 2 types of messages through which they communicate. Every microservice in order to communicate either synchronously or asynchronously with other microservices. "Synchronous – HTTP is a synchronous protocol. The client sends a request and waits for a response from the service. The client code or message sender usually does not wait for a response.

·Databases

Microservice owns a private database to capture their data and implement the respective business functionality. Microservices databases are updated through their service API. The services provided by Microservices are carried forward to any remote service which supports inter-process communication for different technology stacks.

·Static Content

After the microservices communicate within themselves, they deploy the static content to a cloud-based storage service that can deliver them directly to the clients via Content Delivery Networks (CDNs).

·Management

Management feature is a capability that allows operations and business users to configure services in run-time. For load balancers, the approaches to feature flags management in microservices architecture are a complex topic, especially when one business feature spans multiple microservices.

·Service Discovery

In a microservices application, the set of running service instances changes dynamically. Instances have dynamically assigned network locations. Consequently, for a client to make a request to service it must use a service-discovery mechanism. A key part of service discovery is the service registry.

14 - How does a Microservice architecture work?

Microservices is also called microservice architecture which is an organizational approach to software development where software is composed of small independent services that communicate over APIs which are well-defined.

Highly maintainable and testable

Loosely coupled

Independently deployable

Organized around business capabilities

Owned by a small team

The microservice architecture contains components depending on the business requirements.

API Gateway- Clients need API Gateway as it is an entry point, which forwards the call to the specific services on the back end. Here API gateway helps in collecting the responses from different services and returns the response to the client.

Microservices- As the name itself suggests that microservices are the services that help in dividing the service into small services that perform a certain business capability like user registration, current orders, or wish list.

Database- Microservices can either share the same database or an independent database.

Inter-microservices communication- REST or Messaging are the protocol to interact with each other.