**HOMEWORK-6**

**RÜMEYSA TÜRKER**

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

| | |
|---|---|
| Manual testing is not accurate at all times due to human error, hence it is less reliable. | Automated testing is more reliable, as it is performed by tools and/or scripts. |
| Manual testing is time-consuming, taking up human resources. | Automated testing is executed by software tools, so it is significantly faster than a manual approach. |
| Investment is required for human resources. | Investment is required for testing tools. |
| Manual testing is only practical when the test cases are run once or twice, and frequent repetition is not required. | Automated testing is a practical option when the test cases are run repeatedly over a long time period. |
| Manual testing allows for human observation, which may be more useful if the goal is user-friendliness or improved customer experience. | Automated testing does not entail human observation and cannot guarantee user-friendliness or positive customer experience. |

**2 – What does Assert class ?**

Assert class provides a set of assertion methods useful for writing tests.

Assert.assertEquals() methods checks that the two objects are equals or not. If they are not, an AssertionError without a message is thrown. Incase if both expected and actual values are null, then this method returns equal. In the below example, the first Test (mySimpleEqualsTest()) compares two strings. The second test (myObjectEqualsTest()) we are comparing two different user defined objects. The assertEquals() method calls equals method on each object to check equality.

**3 - How can be tested 'private' methods ?**

The best way to test a private method is via another public method. If this cannot be done, then one of the following conditions is true:

- The private method is dead code
- There is a design smell near the class that you are testing
- The method that you are trying to test should not be private

**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. According to the Cambridge dictionary, the adjective monolithic also means both too large and unable to be changed.

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

1. Adopt a well-organized test practice.
2. Name your test well.
3. Write reliable and trustworthy unit tests.
4. Make automated unit testing a rule.
5. Focus on single use-case at a time.
6. Minimal assertion per test.
7. Unit test should be isolated.
8. Truly unit, not integration.
9. Aim for 100% code coverage.
10. Start using headless testıng in the cloud.

**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 and 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.

**7 - What are the benefits and drawbacks of Microservices ?**

***Benefits:***

- Improved fault isolation: Larger applications can remain mostly unaffected by the failure of a single module.
- Eliminate vendor or technology lock-in: Microservices provide the flexibility to try out a new technology stack on an individual service as needed. There won't be as many dependency concerns and rolling back changes becomes much easier. With less code in play, there is more flexibility.
- Ease of understanding: With added simplicity, developers can better understand the functionality of a service.
- Smaller and faster deployments: Smaller codebases and scope = quicker deployments, which also allow you to start to explore the benefits of Continuous Deployment.
- Scalability: Since your services are separate, you can more easily scale the most needed ones at the appropriate times, as opposed to the whole application. When done correctly, this can impact cost savings.

***Drawbacks:***

- Communication between services is complex: Since everything is now an independent service, you have to carefully handle requests traveling between your modules. In one such scenario, developers may be forced to write extra code to avoid disruption. Over time, complications will arise when remote calls experience latency.
- More services equals more resources: Multiple databases and transaction management can be painful.

- Global testing is difficult: Testing a microservices-based application can be cumbersome. In a monolithic approach, we would just need to launch our WAR on an application server and ensure its connectivity with the underlying database. With microservices, each dependent service needs to be confirmed before testing can occur.
- Debugging problems can be harder: Each service has its own set of logs to go through. Log, logs, and more logs.
- Deployment challengers: The product may need coordination among multiple services, which may not be as straightforward as deploying a WAR in a container.
- Large vs small product companies: Microservices are great for large companies, but can be slower to implement and too complicated for small companies who need to create and iterate quickly, and don't want to get bogged down in complex orchestration.

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

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 ?**

Challenges of microservices architectures:

- Design.
- Security.
- Testing.
- Increased operational complexity.
- Communication.
- Your defined domain is unclear/uncertain.
- Improved efficiency isn't guaranteed.
- Application size is small or uncomplex.

**10 - How independent microservices communicate with each other?**

Some of the ways in which services can communicate in a microservices architecture:

***HTTP communication***

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

In such cases when one service is dependent on another service, the services complete their task cycle and then meet the requests received from another service. This is called Synchronous HTTP calls between services. There is no coupling between services here. But those services that placed requests have to wait till they get the response, in order to perform any action.

HTTP asynchronous call is another option between two services. Here, the service takes request from the first service in case of multiple requests and immediately responds with a URL. This URL is used to check on the progress of the request. The services need not wait for their response as this happens instantly as coupling is loose. The services are isolated.

***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. The message broker is the point of contact between all services. This reduces complexity and increases efficiency.

However, there is still some coupling between services using this approach. Both or all the services must agree on the message structure and components involved before the workflow.

***Event-driven communication***

An 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. Communication happens through events that individual services generate.

A message broker is still needed here as individual services will write their events to it. However, the participating services need not know the details of the event. They only respond to the event that is happening and not any message that the event would deliver.

**11 - What do you mean by Domain driven design ?**

Domain-driven design is a software engineering approach to solving a specific domain model. The solution circles around the business model by connecting execution to the key business principles.

Common terminology between the domain experts and the development team includes domain logic, subdomains, bounded contexts, context maps, domain models, and ubiquitous language as a way of collaborating and improving the application model and solving any domain-related challenges.

**12 – What is container in Microservices ?**

Containers are a lightweight, efficient and standard way for applications to move between environments and run independently. Everything needed (except for the shared operating system on the server) to run the application is packaged inside the container object: code, run time, system tools, libraries and dependencies.

**13 - What are the main components of Microservices architecture ?**

5 core components of microservices architecture:

1. Microservices. Microservices make up the foundation of a microservices architecture.
2. Containers.
3. Service mesh.
4. Service discovery.
5. API gateway.

**14 - How does a Microservice architecture work?**

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.