*Homework 6- Mustafa YAŞAR*

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

Manual testing and automated testing cover two vast areas. Within each category, specific testing methods are available, such as black box testing, white box testing, integration testing, system testing, performance testing, and load testing. Some of these methods are better suited to manual testing, and some are best performed through automation. Here's a brief comparison of each type, along with some pros and cons:

| Manual Testing | 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 is only practical when the test cases are run once or twice, and frequent repetition is not required. | Automated testing does not entail human observation and cannot guarantee user-friendliness or positive customer experience. |

## 2- What does Assert class?

An assertion allows testing the correctness of any assumptions that have been made in the program. An assertion is achieved using the assert statement in Java. While executing assertion, it is believed to be true. If it fails, JVM throws an error named AssertionError. It is mainly used for testing purposes during development.

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

For exhaustive unit testing of the method, we'd need to mock private methods. although private methods can be tested using PowerMock, we must be extra cautious while using this technique.Given the intent of our testing is to validate the behavior of a class, we should refrain from changing the internal behavior of the class during unit testing.Mocking techniques should be applied to the external dependencies of the class and not to the class itself.If mocking of private methods is essential for testing our classes, it usually indicates a bad design.

## 4- What is Monolithic Architecture?

A monolithic application describes a single-tiered software application in which the user interface and data access code are combined into a single program from a single platform.A monolithic application is self-contained and independent from other computing applications. The design philosophy is that the application is responsible not just for a particular task, but can perform every step needed to complete a particular function.Today, some personal finance applications are monolithic in the sense that they help the user carry out a complete task, end to end, and are private data silos rather than parts of a larger system of applications that work together. Some word processors are monolithic applications.These applications are sometimes associated with mainframe computers.A monolithic application describes a software application that is designed without modularity.

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

Add the following top unit-testing best practices in your development processes and ensure the desired output from your project development.

### 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.

Ideally, you should save your test cases in three parts:

- Save your test with the strategy being tested
- Save your test given the name of the environment under which it's being tested.
- Save your test with the proposed behavior when the scenario is requested.

The more readable your tests will be, the more it will be easy to fix a bug for not just the one who wrote it, but for other developers working on the project or will work on the project.

### 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.

Passing tests should produce no output while falling tests must produce clear output like unambiguous error messages.

## 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

Headless browsers are gaining popularity. Headless browsers or browser less testing do not involve a Graphical User Interface (GUI). In unit testing, adopting this simple practice of using browsers without GUI or UI makes your test much faster since it no longer needs to load the entire UI.Headless testing takes advantage of lightweight container instances in the cloud that allows you to write tests quickly and avail rapid feedback almost in real-time.

## 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, and when you decide to report multiple failures per test, you begin to fight against JUnit. This is not recommended.

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

Microservices follow the principles of service-oriented architecture (SOA) design. Although SOA has no official standards, principles defined in Thomas Erl's book entitled "SOA: Principles of Service Design" are often used as rules of thumb.

They are as follows:

a) Standardized service contract (services follow a standardized description)
b) Loose coupling (minimal dependencies)
c) Service abstraction (services hide their internal logic)
d) Service reusability (service structure is planned according to the DRY principle)
e) Service autonomy (services internally control their own logic)
f) Service statelessness (services don't persist state from former requests)
g) Service discoverability (services come with discoverable metadata and/or a service registry)
h) Service composability (services can be used together)

| Benefit | Drawbacks |
|---|---|
| Greater agility | Needs more collaboration (each team has to cover the whole microservice lifecycle) |
| Faster time to market | Harder to test and monitor because of the complexity of the architecture |
| Better scalability | Poorer performance, as microservices need to communicate (network latency, message processing, etc.) |
| Faster development cycles (easier deployment and debugging) | Harder to maintain the network (has less fault tolerance, needs more load balancing, etc.) |
| Easier to create a CI/CD pipeline for single-responsibility services | Doesn't work without the proper corporate culture (DevOps culture, automation practices, etc.) |
| Isolated services have better fault tolerance | Security issues (harder to maintain transaction safety, distributed communication goes wrong more likely, etc.) |
| Platform- and language agnostic services | |
| Cloud-readiness | |

## 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?

- Overcoming Design Complexity
- Achieving Data Consistency
- Need for Testing and Monitoring
- Debugging Issues
- Compromised Security
- Increased Operational Complexity
- Inter-Service Communication Breakdown
- Requires Team Expertise
- Maintenance of Microservices
- Network Management

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

Generally 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. We should be careful when considering communication types and manage them into design phases. Because microservices are complex structure into independently developed.

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

Domain Driven Design is an approach that tries to provide solutions to the basic problems that are frequently encountered in the development of complex software systems and in ensuring the continuity of our applications after these complex projects are implemented.

Generally, Domain Driven Design advocates a philosophy of how software should be modeled to adapt it to the digital world by creating real-world business models with a common language (Ubiquitous Language) that everyone can understand.

## 12-What is container in Microservices?

It is an approach that tries to solve the problems that are frequently experienced when developing complex software systems. In order to understand DDD, some basic concepts need to be mastered. We can achieve the DDD by these concepts.Ubiquitous Language: All the stakeholders of the software product should be able to speak the same language, which is naming for the business concepts should be the same, in order to deliver the desired output and ensure the quality of the output.Bounded Contexts: Complex structures should contain subdomains, and these subdomains should be grouped among themselves. These structures are called Bounded Context.

For example:

- Order Domain
- Stock Domain
- Product Domain

Entity & Value Object: DDD recommends that business logic should be built on business entities not on services.

Aggregate Root (AR): There should be aggregate roots where the entities associated with each other working together.

Repository: There should be repository levels which only responsible for the persistence of domain entities. Repository layers should not contain any other business logic.

Layered Architecture: Regardless of whether or not microservice architecture is used, there should be layers of the product architecture. Here is a well-known 4-tier architecture:

- Domain Layer
- Application Layer
- Presentation Layer
- Infrastructure Layer

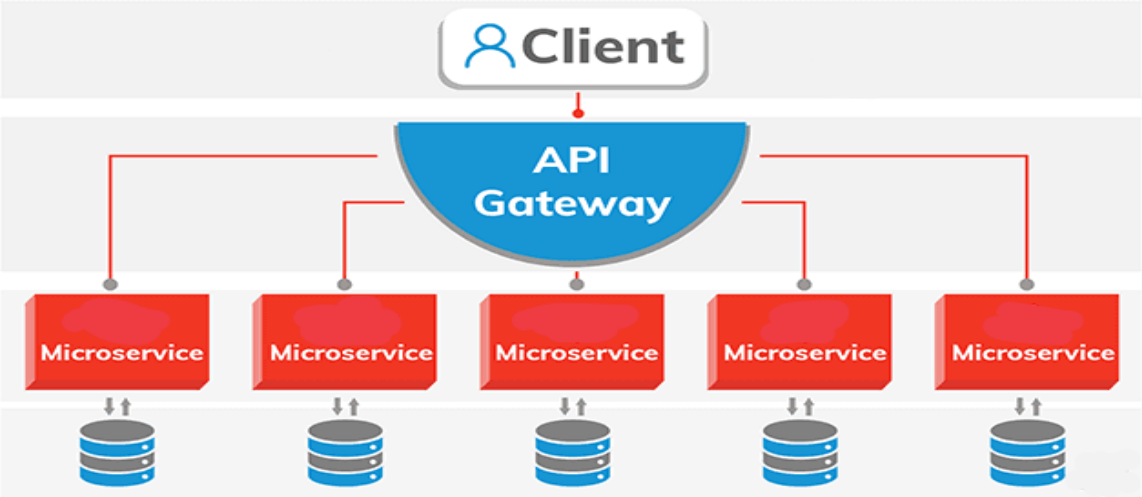
## 13-What are the main components of Microservices architecture?

There are 8 main components for Microservice architecture:

- Domain Layer
- Clients
- Identity Providers
- API Gateway
- Messaging Formats
- Databases
- Static Content
- Management
- Service Discovery

## 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.