HW#6 – İbrahim Çelik

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

Manual testing is testing of the software where tests are executed manually by the test engineers or QA analysts to discover bugs in software or development applicatıon. In manual testing, the tester checks all the requirement features. In this process, the software testers execute the test cases and generate the test reports without the help of any automation software testing tools.

Automation testing is the process of creation and execution of test scripts with the help of a tool and verifying the results against the expected behavior. It has become popular and inevitable in most of the organizatıon due to the ease and advantages it provides over the manual tests.

2 – What does Assert class ?

The assert keyword has been a feature of the Java programming language since Java 1.4. Assertion allows developers to test the assumptions in their programs to troubleshoot and fix bugs. The Assert statement is the best test-phase catch-up form to use instead of throwing rare errors that won't be used much from the Throwable class. Exceptions with few errors make programs very slow. Therefore, it makes more sense to use the assert word and only find bugs that can be avoided during testing.

3 - How can be tested 'private' methods ?

If we are talking about spring boot, it has nothing special about it: Private methods should not be tested it's an internal "how" of the class and you should mainly test the API of the class - the "capabilities" that it exposes to the user of the class via non-private methods.

But if we are using JUnit or SuiteRunner, we have the same four basic approaches to testing private methods:

- Don't test private methods.
- Give the methods package access.
- Use a nested test class.
- Use reflection.

4 – What is Monolithic Architecture ?

Monolith means composed all in one piece. The Monolithic application describes a single-tiered software application in which different components combined into a single program from a single platform. Components can be:

- Authorization — responsible for authorizing a user
- Presentation — responsible for handling HTTP requests and responding with either HTML or JSON/XML (for web services APIs).
- Business logic — the application’s business logic.
- Database layer — data access objects responsible for accessing the database.
- Application integration — integration with other services (e.g. via messaging or REST API). Or integration with any other Data sources.
- Notification module — responsible for sending email notifications whenever needed.

Consider an example of Ecommerce application, that authorizes customer, takes an order, check products inventory, authorize payment and ships ordered products. This application consists of several components including e-Store User interface for customers (Store web view) along with some backend services to check products inventory, authorize and charge payments and shipping orders.

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

- Unit Tests Should Be Trustworthy
- Unit Tests Should Be Maintainable and Readable
- Unit Tests Should Verify a Single Use Case
- Unit Tests Should Be Isolated
- Unit Tests Should Be Automated
- Use a Good Mixture of Unit and Integration Tests
- Unit Tests Should Be Executed Within an Organized Test Practice

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

There is no particular answer for this question, the main reason it does it like this to make the testing process simple and easy. Even though the failures are more than one it only reports the first failure, resolving which may implicitly resolve other unreported failures too.

7 - What are the benefits and drawbacks of Microservices ?

Advantages:

- Microservices are self-contained, independent deployment module.
- The cost of scaling is comparatively less than the monolithic architecture.
- Microservices are independently manageable services. It can enable more and more services as the need arises. It minimizes the impact on existing service.
- It is possible to change or upgrade each service individually rather than upgrading in the entire application.
- Microservices allows us to develop an application which is organic (an application which latterly upgrades by adding more functions or modules) in nature.
- It enables event streaming technology to enable easy integration in comparison to heavyweight interposes communication.
- Microservices follows the single responsibility principle.
- The demanding service can be deployed on multiple servers to enhance performance.
- Less dependency and easy to test.
- Dynamic scaling.
- Faster release cycle.

Disadvantages:

- Microservices has all the associated complexities of the distributed system.
- There is a higher chance of failure during communication between different services.
- Difficult to manage a large number of services.
- The developer needs to solve the problem, such as network latency and load balancing.
- Complex testing over a distributed environment.

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

Spring Boot Actuator is available from the very first release of Spring Boot. It proves several features like health check-up, auditing JVM metrics, log information, caching statics, etc. We have the option to use JMX or HTTP end points to manage and monitor our applications on the production environment.

Actuator also make it easy to integrate with external monitoring systems like Prometheus, Graphite, DataDog, New Relic, etc. The Actuator uses for Micrometer, an application metrics facade to support the integration with external monitoring systems.Actuator make it super easy to integrate our application with any external system with a very minimal configurations.

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

- Data Consistency
- Distributed Tracing
- Network failure
- Operation Overhead
- Testing
- Complex Team Communications
- Technical Debt

10 - How independent microservices communicate with each other?

- Calling each other via Http (REST, GraphQL), gRPC, Thrift, or via queuing like RabbitMQ
- Publishing events, commands and other data items that other microservices or lamdas can subscribe to, using RabbitMQ, Kafka, AWS SQS, Kinesis, Redis, Hazelcast, or even using databases and potentially cursors
- Writing to a store like AWS S3 and having the environment, AWS in this case, trigger an event that can again be subscribed to and consumed by microservices or lamdas
- BPM: using a BPM engine like Camunda to orchestrate workflows, with services being called in various stages

11 - What do you mean by Domain driven design ?

Domain-Driven Design is a software design discipline centred on the principles that:

- Software for a complex domain requires all designers (engineers, testers, analysts, …) to have a deep, shared understanding of the domain, guided by domain experts
- That understanding is rooted in language: the domain language should be formalised into a Ubiquitous Language (shared, agreed upon, unambiguous)
- The understanding is expressed in a model, shared between experts and designers, which express the problem space (as opposed to the solution space)
- The model must not shy away from explicitly expressing the essential complexity of the domain
- A complex domain can not be efficiently expressed as a single universal model and language, and must therefore be separated into Bounded Contexts (ie. an internally consistent language and model) by the system designers

- The model and language should be in sync with our understanding of, and changes in the domain, through continuous refactoring towards deeper insight

12 – What is container in Microservices ?

Microservices architecture needs container ingress.

Applications require a set of services from their infrastructure—load balancing, traffic management, routing, health monitoring, security policies, service and user authentication, and protection against intrusion and DDoS attacks. These services are often implemented as discrete appliances. Providing an application with these services required logging into each appliance to provision and configure the service.

This process was possible when managing dozens of monolithic applications, but as these monoliths become modernized into microservices based applications it isn't practical to provision hundreds or thousands of containers in the same way. Observability, scalability, and high availability can no longer be provided by discrete appliances.

The advent of cloud-native applications and containers created a need for a service mesh to deliver vital application services, such as load balancing. The service mesh handles east-west services within the datacenter, with container ingress handling north-south into and out of the datacenter. By contrast, trying to place and configure a physical hardware appliance load balancer at each location and every server is overly challenging and expensive. And require businesses need to deploy microservices to keep up with application demands and multi-cloud environments.

13 - What are the main components of Microservices architecture ?

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

14 - How does a Microservice architecture work?

Microservices architecture focuses on classifying the otherwise large, bulky applications. Each microservice is designed to address an application's particular aspect and function, such as logging, data search, and more. Multiple such microservices come together to form one efficient application.

This intuitive, functional division of an application offers several benefits. The client can use the user interface to generate requests. At the same time, one or more microservices are commissioned through the API gateway to perform the requested task. As a result, even larger complex problems that require a combination of microservices can be solved relatively easily.