Theoretical Homework#3

1) **What is the difference between manual testing and automated testing?**
   Otomatik testler, araçlar ve/veya komut dosyaları tarafından gerçekleştirildiği için daha güvenilirdir. Manuel testler ise insan hatası nedeniyle her zaman doğru değildir, dolayısıyla daha az güvenilirdir.

   Manuel test, yalnızca test senaryoları bir veya iki kez çalıştırıldığında pratiktir ve sık tekrarlama gerekli değildir. Otomatik test ise, test senaryoları uzun bir süre boyunca tekrar tekrar çalıştırıldığında pratik bir seçenektir.

   Manuel testler insan eliyle değiştirilebildiği için müşteri isteklerine göre şekillendirilebilir. Ancak otomatik testlere müdahale şansı olmadığından farklı spesifik senaryoları test etmek mümkün değildir.

2) **What does Assert class?**

   Assert sınıfı bir test senaryosunun başarı durumunu sorgulamak için kullanılan ve java.lang.Object sınıfından türemiş bir sınıftır. Assert metotlarından test edilmek istenilen senaryoya uygun olanı seçilerek testlerin geçip geçmediği yani istenilen değeri verip vermediği sorgulanabilir.

3) **How can be tested “private” methods?**
   Standart olarak private metotları test etmemek bir yaklaşım olarak kabul edilebilir, ancak test etmek istediğimiz private metotları da bazı yaklaşımlarla test etmek mümkün olabilir. Bunlardan ilki metotlara kendi package yapısında erişim izni vermektir. Bunun zararlarından biri kapsüllemeyi zedeliyor oluşudur ancak, göz ardı edilen durumlarda denenebilir. Diğer bir yaklaşım ise static olarak sınıfın içine bir test metodu yerleştirmektir. Aynı sınıf içerisinde test metodu barındırmak karışıklığa sebep olabileceğinden çok da tercih edilmemektir. Diğer bir yöntem ise yansıtma yöntemidir. Özel sınıfın içindeki objeler dışarıya alınır ve taklit edilir. Yansıtma sınıfı için yazılan testler aslında private olan sınıfın da çıktılarıdır.

4) What is Monolithic Architecture?

Monolitik mimari, yazılım uygulamaları için geleneksel yapıdır. Analistler genellikle bunu uygulama geliştirme için daha yeni bir model olan mikro hizmetlerle karşılaştırır. Monolitik mimarinin uzun bir geçmişi olmasına rağmen, bazen mikro hizmet modelinden daha üstündür. Monolitik, yazılımın tüm yönlerinin tek bir birim olarak çalıştığı, hepsi bir arada bir mimaridir. Mikro hizmet modelinde bileşenler modülerdir, bağımsız olarak çalışır ve optimum işlevsellik için gerektiği şekilde birbirine bağlanır.

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

- Bir testte tek bir iddiada bulunmak
- Test bağımlılıklarını en aza indirmek
- Otomatize edilmiş test yöntemleri kullanmak
- Tutarlı isimlendirmeler kullanmak
- Testlerin belirleyici olduğundan emin olmak
- While, if gibi logic yapılarını kullanmaktan kaçınmak
- Geliştirme sırasında testlerini de yasmak
- Gerçek tarayıcılarla ve araçlarla çalışmak

6) Why does Junit only report the first failure in a single test?

Tek bir testte birden fazla hatanın rapor edilmesi, genellikle testin çok fazla şey yaptığının ve bir birim testinin çok büyük olduğunun bir işaretidir. JUnit, bir dizi küçük testle en iyi şekilde çalışacak şekilde tasarlanmıştır. Her testi, test sınıfının ayrı bir örneği içinde yürütür. Her testte başarısızlığı bildirir.

7) What is the role of actuator in spring boot?

Spring Boot Actuator, Spring Boot Framework'ün bir alt projesidir. Spring Boot uygulamasını izlememize ve yönetmemize yardımcı olan bir dizi ek özellik içerir. Aktüatör uç noktalarını (kaynakların yaşadığı yer) içerir. Spring Boot uygulamasını yönetmek ve izlemek için HTTP ve JMX uç noktalarını kullanabiliriz. Bir uygulamada üretime hazır özellikler almak istiyorsak Spring Boot actuator kullanmalıyız.

8) What are the benefits and drawbacks of Microservices?

Microservisler yapılan işleri küçük parçalar halinde bölüştürme ve farklı dilleri aynı proje içinde kullanma imkanı sağlayan yapılardır. JSON formatında veri alışverişi yapabilen bu küçük servisler, bir çok konuda avantaj sağlarken bu çoklu yapıyı yönetmek haliyle daha zordur. Özellikle oluşan çoklu yapılar arasında haberleşmenin yönetilmesi ve oluşan bu çoklu yapıya uygun testlerin yazılması, monolotik mimariye sahip bir projeye göre çok daha zordur.

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

Security sağlanması, servisler arası iletişimin yönetilmesi ve gerekli testlerin yazılması mikroservis mimarisinin en zorlu kısımları olarak adlandırılabilir.

10) How independent microservices communicate with each other?

Mikroservisler dağıtılır ve mikroservisler ağ düzeyinde hizmetler arası iletişim yoluyla birbirleriyle iletişim kurar. Her mikro hizmetin kendi örneği ve süreci vardır. Bu nedenle hizmetler, HTTP, gRPC veya mesaj aracılarının AMQP protokolü gibi hizmetler arası iletişim protokollerini kullanarak etkileşime girmelidir.

11) What do you mean by Domain driven design?

Etki alanına dayalı tasarım (DDD), uygulamayı temel iş kavramlarının gelişen bir modeline derinden bağlayarak karmaşık ihtiyaçlar için yazılım geliştirmeye yönelik bir yaklaşımdır.

İlk başta kulağa çok karmaşık geliyor, ancak DDD uygulamasının amacı, karmaşık senaryoları sistematik bir yaklaşımla, alan uzmanları ve geliştirme ekibinin minimum sürtünme ile etkili bir şekilde işbirliği yapabileceği şekilde ele almaktır. Bunu başarmak için hem stratejik hem de taktik araçlar ve kalıplar kullanılır. En önemli araçlardan biri iletişimdir. İlgili taraflar arasındaki iletişimi yönetmek için DDD, Ubiquitous Language (UL) kullanır.

12) What is container in Microservices?

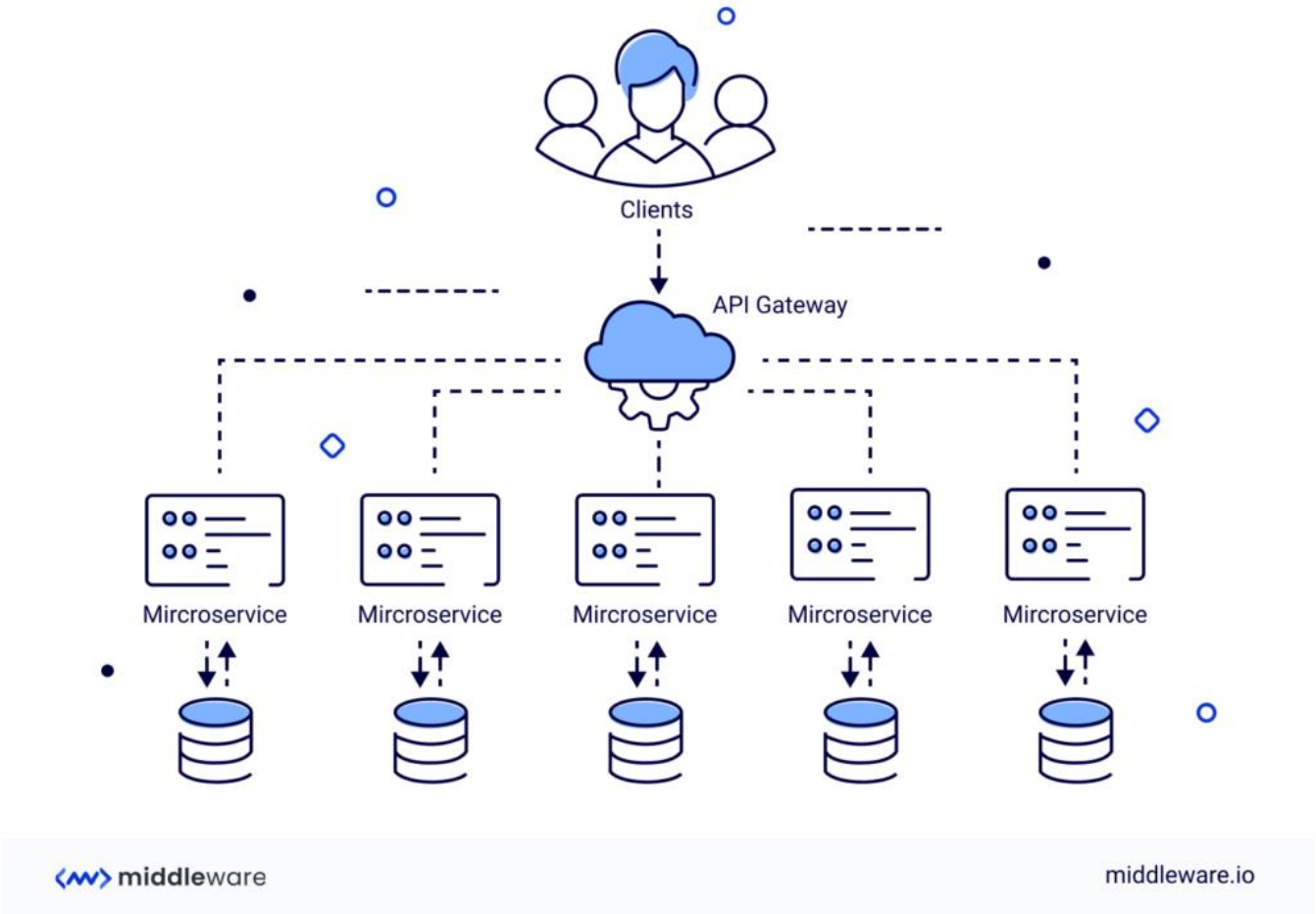
Konteynerler, bir işletim sistemi sanallaştırma biçimidir. Küçük bir mikro hizmet veya yazılım sürecinden daha büyük bir uygulamaya kadar her şeyi çalıştırmak için tek bir konteyner kullanılabilir. Bir konteynerin içinde gerekli tüm yürütülebilir dosyalar, ikili kod, kitaplıklar ve yapılandırma dosyaları bulunur. Ancak, sunucu veya makine sanallaştırma yaklaşımlarıyla karşılaştırıldığında, konteynerler işletim sistemi görüntüleri içermez. Bu, önemli ölçüde daha az ek yük ile onları daha hafif ve taşınabilir hale getirir. Daha büyük uygulama dağıtımlarında, birden çok konteyner bir veya daha fazla kapsayıcı kümesi olarak dağıtılabilir. Bu tür konteynerler, Kubernetes gibi bir kapsayıcı düzenleyici tarafından yönetilebilir.

13) What are the main components of Microservices architecture?

Mikroservis mimarisinin temel sekiz bileşeni var diyebiliriz.

- Müşteriler
- Kimlik Sağlayıcılar
- API Ağ Geçidi
- Mesajlaşma Formatları
- Veritabanları
- Yönetim
- Statik İçerik
- Servis Keşfi

14) How does a Microservice architecture work?



Yukarıda da gösterildiği gibi mikroservisler belli bir işi yapmak üzere tasarlanmış, kapsamı dışında başka bir hizmeti olmayan yapılardır. Her mikroservisin kendine ait database'i olduğundan sistemler arası geçişlere, veri karmaşıklığına ihtimal verilmeden yapılmak istenilenler gerçekleştirilebilir. Oluşturulan mikroservisler arası iletişimin ortak bir dille sağlanıyor olması da her mikroservise kendi içinde bağımsız olabilme fırsatı sunar. Sistemin bir parçası A proglamlama diliyle yazılmışken diğer bir parçası B programlama dili kullanılarak yazılmış olabilir. Bu iki sistem de kendi mikroservisinin dışına data göndereceğinde yine belirlenmiş ortak bir haberleşme dili kullanacakları için, kendi içlerinde hangi dille yazıldıkları sistemde sorun oluşturmaz.

Ayrıca microservisler gelen yük baz alınarak kopyalanabilir, bu da daha az hafızayla çok iş başarabilmeye olanak sağlar. Dikey değil yatay hiyerarşi içinde olan sistemlerde bir mikroservisin aksaması sistemin diğer ucunu etkilemez. Sadece kendi kapsamında sorun oluşturur ve sistem tamamen yerle bir olmadığından kayıp çok daha azdır.