TEORİK ÖDEV-3

1. manual testing ve automated testing arasındaki fark nedir?

Manuel testte, bir insan testleri test komut dosyaları olmadan adım adım gerçekleştirir. Otomatik testlerde ise; diğer araçlar ve yazılımlarla birlikte test otomasyon çerçeveleri aracılığıyla otomatik olarak yürütülür.

	Manual	Automation
Test Uygulaması	QA test cihazlarıyla manuel olarak yapılır.	Otomasyon araçları ve komut dosyaları kullanılarak otomatik olarak yapılır.
Test Verimliliği	Zaman alır ve verimliliği daha az.	Daha kısa sürede daha fazla test ve daha yüksek verimlilik
İşleyişi	Tamamen manuel	Çoğu görev otomatikleştirilebilir, gerçek kullanıcı simülasyonlarını içerir.
Test Kapsamı	Yeterli test kapsamını sağlamak zor.	Daha fazla test kapsamı sağlamak kolaydır.

2. Assert classı ne yapar?

Java 1.4'den beri assert anahtar kelimesi Java'nın bir özelliğidir. Assertion, geliştiricilerin hataları gidermek ve düzeltmek için programlarındaki varsayımları test etmelerini sağlar. Exception'lar programı yavaşlatacağı için assert kullanarak test aşamasında sadece önlenebilecek hataları bulmak daha mantıklıdır.

- Kullanımı: Assert expression;

  Assert expression1 : expression2;

Kullanımdaki expression1 boolean olmalıdır; expression2, bir değer döndürmemelidir(void).

3.Private metodlar nasıl test edilir?

TDD yaklaşımı ile class üretimi; önce test classı üretilip üyelerin bu test classından üretilmesi demektir. Bunun için Access modifier olarak public yapılması gerekir.

Bu durumda private olan bir metodu test edebilmek için; metod tekrar daha basit bir şekilde public olarak kopyalanır ve sadece yazılan yeni metod test edilir. Böylece kopya metod çalışırsa asıl metod da çalışacaktır.

4.Monolitik Mimari nedir?

Monolithic Architecture yani Monolitik Mimari, uygulamanın tüm parçalarının bir arada tutulması olarak tanımlanabilir. İşlemler, tek bir servis üzerindedir. Birden fazla modül içeren tek bir kod tabanına sahiptir. Modüller, fonksiyonel veya teknik özelliklerine göre ayrılmıştır. Tüm uygulamayı build eden tek bir derleme sistemine sahiptir. Ayrıca tek bir çalıştırılabilir veya deploy edilebilir dosyaya sahiptir.

5.Unit test case yazmak için en iyi yol nedir?

- Her zaman tek bir şeyi kontrol edin.
- AAA(Arrange, Act, Assert) patternini takip edin. Daha okunaklı ve kullanışlı testler yazmak için kullanılır. Önce test edilecekler düzenlenir, sonra test edilen metod çağırılır, son olarak test sonucunun beklenen sonuç olduğunu doğrulamak için test framework'u çağırılır.
- Güvenilir olmalı. Birim testleri, yalnızca test edilen sistemde bir hata varsa başarısız olmalıdır.
- Tekrarlanabilir olmalı ve çevre veya çalışma düzeni gibi dış etkenlerden bağımsız olmalıdır.
- Testler, herhangi bir makinede, herhangi bir sırada, birbirini etkilemeden çalıştırılabilir olmalıdır.
- Birim testlerini günlük veya her saat gibi otomatik bir süreçte veya Sürekli Entegrasyon veya Teslimat sürecinde çalıştırmalıyız.
- Uygulama değiştiğinde birim testlerini güncel tutarak yeniden düzenlememiz gerekiyor, aksi takdirde değerlerini kaybederler.

6.Neden JUnit, bir single test içinde sadece ilk hatayı raporlar?

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.Spring Boot içindeki actuator'lerin rolü nedir?

HTTP endpoint'ler veya JMX yardımıyla Spring Web uygulamalarını izlemek ve yönetmek içini kullanılır.

8.Microservicelerin faydaları ve sakıncaları nelerdir?

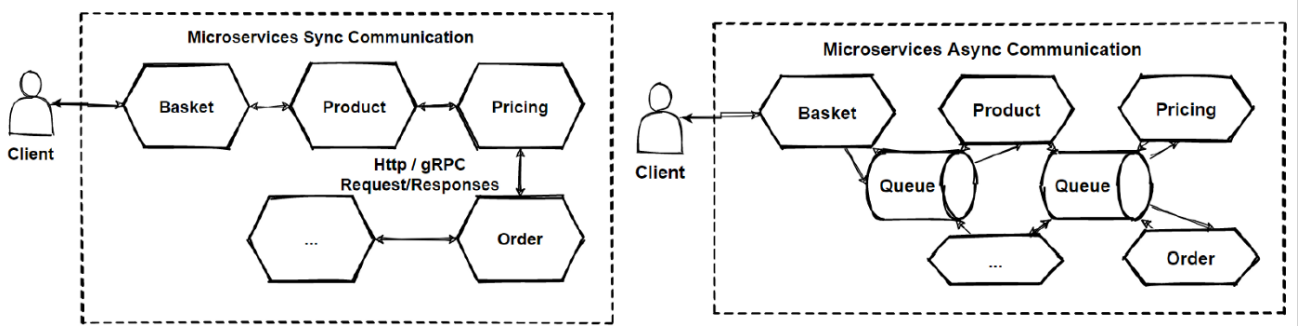
- Çeşitli diller ortak projed kullanılabilir.
- Dirençlilik(Resillince) dediğimiz kavram hataların servislerin ayrı olmasından mütevellit bütün sistemi etkilemeyeceğini ifade eder.
- Ölçeklenebilirlik(Scaleble): Mikroservis mimarisinde yüksek istek alan servislerimizi istediğimiz zaman birden fazla hostta(portta) ayağa kaldırabilir, load balancer algoritmaları ile yoğunluk olduğu durumda gelen istekleri bu instance'lara dağıtabilir uygulamamızı tamamen ölçeklenebilir yapıda geliştirmiş oluruz.
- Kolay Yayınlama(Ease of Deployment): Mikroservis mimaride yardımcı toollar ile developerlar kendi servislerini düşük riskler ile hızlı bir şekilde geliştirme yapabilir.
- Organizasyonel Uyum: Mikroservisler daha ufak ve uyumlu ekipler oluşturmamıza izin verirler. Bu aynı zamanda agile ilerlememize daha uygun bir yapıdır. Ekipler kendi mikroservislerinden sorumludur ve ihtiyaç dâhilinde sadece onlarda geliştirme yapar, hızlı aksiyon alabilirler.
- Bu mimarinin büyük avantajları olsa da diğer mimariler gibi dezavantaları da bulunmaktadır. Bu yüzden oluşturacağınız projelerde tüm durumları göz önüne alarak doğru seçimi yapmak durumundayız. Mikroservisler monolotik uygulamalara göre geliştiriciler için farklı karmaşık sorunlar getirir.
- Servisler arası iletişim ve uyumlu şekilde çalışması zorlayıcıdır. Yüzlerce mikroservisiniz olabilir bunların iletişimi güvenli ve sorunsuz olmalıdır. Mikro projelerde debug etmek zorlaşır bu sorunu çözmek için güçlü bir loglama alt yapısı ve monitoring araçları kullanılmalıdır.
- Unit test mikroservisler için basitken integration test daha zor hale gelir.
- Her mikrosersvis kendi API endpointlerine sahiptir. Değişiklik yapmak kolay olmasına rağmen bu endpointlere istek atılan başka servisleri göz önüne almalıyız. Kısacası sahip olduğunuz mikroservisleri kimlerin kullandığını ve business logice hâkim olmanız gerekir.
- Mikro servis mimarisinin verimli çalışması için, güvenlik ve bakım desteği ile yeterli barındırma altyapısına ve tüm hizmetleri anlaya, yönetebilen nitelikli geliştirme ekiplerine ihtiyacınız vardır.

9.Microservis kullanmanın zorlukları nelerdir?

- Her bir mikroservisin boyutunu, en uygun sınırlarını ve her bir mikroservis ile servisleri entegre etmek için çerçeve arasındaki bağlantı noktalarını belirleyebilirler. Mikroservisleri tasarlamak, bunların sınırlı bir bağlamda oluşturulmasını gerektirir. Bu nedenle her bir mikroservisin belirli bir sorumluluğu netleştirilmelidir.
- Veriler dağıtıldığından ve kullanıcı verilerinin gizliliğini ve bütünlüğünü korumak çok zordur.
- Her mikroservisin bağımsız yapısı nedeniyle, her bağımlı hizmetin testi zor olur.
- Mikroservis tabanlı uygulamaları başarıyla ölçeklendirmek zordur.

10.Microservisler arası bağlantı nasıl bağımsız olur?

Mikroservisler, bir Uygulama Programlama Arayüzü (API'ler) yardımıyla iletişim kurar. İki teknik kullanılır: eşzamanlı **(**synchronous) ve eşzamansız **(**unsynchronous).



Mikroservisler tabanlı uygulamaya geçerken en büyük zorluklardan biri iletişim mekanizmasını değiştirmektir. Çünkü mikroservisler dağıtılır ve mikroservisler birbirleriyle ağ düzeyinde servisler arası iletişim ile haberleşir. Her mikroservis 'in kendi örneği ve süreci vardır. Bu nedenle hizmetler, HTTP, gRPC veya mesaj aracılarının AMQP protokolü gibi hizmetler arası iletişim protokollerini kullanarak etkileşime girmelidir.

Mikroservisler, bağımsız olarak geliştirilmiş ve dağıtılmış *servisler halinde karmaşık bir yapı olduğundan, iletişim türlerini değerlendirirken dikkatli olmalı ve bunları tasarım aşamalarında yönetmeliyiz.

11.Domain Driven Design nedir?

Bir etki alanının süreçleri ve kuralları hakkında zengin bir anlayışa sahip bir etki alanı modelinin programlanmasına odaklanan bir yazılım geliştirme yaklaşımıdır.

12.Microservislerdeki container nedir?

Uygulamaların veya uygulama bölümlerinin çalışabileceği yarı yalıtılmış ortamlardır. Tamamen ayrı işletim sistemleri çalıştıran VM'lerin aksine, kapsayıcılar kaynakları doğrudan kapsayıcıları barındıran sunucunun işletim sistemiyle paylaşır. Hızlı start, security, service discovery, toollar gibi faktörler nedeniyle mikroservislerde de containerlar kullanlır.

13.Microservis Mimarisindeki temel componentler nelerdir?

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

14.Microservis Mimarisi nasıl çalışır?

Microservice mimarisi, büyük, hacimli uygulamaları sınıflandırmaya odaklanır. Her microservice, bir uygulamanın günlüğe kaydetme, veri arama ve daha fazlası gibi belirli yönünü ve işlevini ele almak üzere tasarlanmıştır. Bu tür birden çok microservice, verimli bir uygulama oluşturmak için bir araya gelir.