Hafta 6

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

- Manuel Test, QA analisti (İnsan) tarafından manuel olarak yapılırken, Otomasyon Testi, bir test cihazı tarafından komut dosyası, kod ve otomasyon araçları (bilgisayar) kullanılarak yapılır.
- Manuel Test süreci, insan hataları nedeniyle doğru değildir, oysa Otomasyon süreci kod ve komut dosyası tabanlı olduğu için güvenilirdir.
- Manuel Test, zaman alan bir süreçtir, Otomasyon Testi ise çok hızlıdır.
- Manuel Test programlama bilgisi olmadan mümkündür, oysa Otomasyon Testi programlama bilgisi olmadan mümkün değildir.
- Manuel Test rastgele Teste izin verirken Otomasyon Testi rastgele Teste izin vermez.

2 – What does Assert class ?

fail(String)	Kodun tamamın çalışmadığını kontrol eder.
assertTrue(true);	İşlemin true değer döndürdüğünü kontrol eder.
assertsEquals(message, expected, actual)	expected ve actual değerlerinin aynı olduğunu kontrol eder. Değilse message'ı gösterir.
assertsEquals(message, expected, actual, tolerance)	expected ve actual değerlerinin belirli toleransta aynı olduğunu kontrol eder. Değilse message'ı gösterir.
assertNull(message, object)	Nesnenin null olduğunu kontrol eder.
assertNotNull(message, object)	Nesnenin null olmadığını kontrol eder.
assertSame(message, expected, actual)	expected ve actual nesnelerinin aynı nesneler olduğunu kontrol eder.
assertNotSame(message, expected, actual)	expected ve actual nesnelerinin aynı nesneler olmadığını kontrol eder.
assertTrue(message, boolean condition)	condition'ın true olduğunu kontrol eder. Değilse message'ı gösterir.

3 - How can be tested 'private' methods ?

Mockito + JUnit ReflectionUtils sınıfı ya da Spring ReflectionTestUtils sınıfı kullanılarak test edilebilir.

4 – What is Monolithic Architecture ?

Monolitik bir uygulama, 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 - What are the best practices to write a Unit Test Case ?

- Test sınıflarını ana kaynak kodundan ayrı tutmak iyi bir fikirdir
- Test sınıfının paketi, kaynak kod birimini test edeceği kaynak sınıfın paketiyle eşleşmelidir.

- Örneğin, eğer *Circle* class'ı *com.baeldung.math* paketindeyse, *CircleTest* classı da *com.baeldung.math* package içinde *src/main/test* altında olmalıdır.
- *Methodlarda "given_when_then" şeklinde isimlendirmeler yapılmalıdır.*
- Expected/Actual değerleri içermelidir.
- Kod blokları da given,when,then formatı şeklinde biçimlendirilmelidir.
- Basit test senaryoları tercih edilmelidir.
- %80 test coverage
- Tek bir belirli senaryoyu test etmek için bir birim testi yazılmalıdır.
- TDD yaklaşımı benimsenmelidir.

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 unit testi için çok büyük olduğunun bir işaretidir. JUnit, bir dizi küçük testle en iyi şekilde çalışacak şekilde tasarlanmıştır.

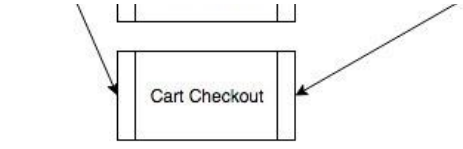
7 – What is the role of actuator in spring boot ?

Yaptığımız ya da yapacağımız Spring Boot uygulamalarımızın endpoint yardımı ile çalışan Spring Boot uygulaması hakkında bilgi almamızı sağlamaktadır.

8 - What are the benefits and drawbacks of Microservices ?

Avantajları:

Teknoloji Heterojenliği:
Monolotik uygulamalar tek bir yazılım dili ve veritabanı sistemi ile geliştiriliyordu. Yukarıda bir e-ticaret sisteminin altında bulunan özelliklerden bahsettik. Bu özellikleri ayrı mikro yazılımlar halinde geliştirdiğimizde farklı bir yazılım dili veya database seçebiliriz. Ödeme sistemleri java ile yazılırken, kategoriler .net, öneri sistemi python ile geliştirilebilir. Buda her yazılım dilinin avantajlarını kullanabilmemize olanak sağlar.



Your Entire Ecommerce App as a composition of multiple independent deployable units

Dirençlilik(Resillince):

Monolotik uygulamalarda projede var olan bir hata tüm sayfanın hata olarak karşımıza çıkmasına sebebiyet verir. Eminim hepimiz site patladı, açılmıyor gibi terimleri kullanmış veya yaşamışızdır. Mikroservis mimarisinde projede bulunan her bir alan farklı bir yazılımcık olduğu ve kendi alanında host edildiği için hata aldığımız yer tüm sistemi etkilemez. Mesela sepet sisteminde var olan bir hata kullanıcıların ürünleri incelemesini, kategorilerde dolaşmasını etkilemez.

Ölçeklenebilirlik(Scaleble):
Her bir mikroservisin kendi alanında host edildiğinden bahsetmiştik. Büyük bir e-ticaret sitesinin indirim günlerinde olduğunu düşünün ve saniyede onbinlerce istek alan servislerimiz

var. Örneğin ürünlerimizi listelediğimiz kategoriler sayfası. Monolotik uygulamada tüm proje tek bir hostta yayınlandığı için tüm sistemde iyileştirme yapmamız gerekir. Mikroservis mimarisinde yüksek istek alan servislerimizi istediğimiz saatlarde veya anlarda birden fazla hostta(portta) ayağa kaldırabilir, load balancer algoritmaları ile yoğunluk olduğu durumda gelen istekleri bu instancelara dağıtabilir uygulamamızı tamamen ölçeklenebilir yapıda geliştirmiş oluruz.

Kolay Yayınlama(Ease of Deployment):
Monolotik uygulamaların deploymont süreçleri daha uzun ve dikkat istiyordu.Production ortamına çıkmak belirli ekiplerin kontrolüyle haftalık/aylık şekilde yapılıyordu. Mikroservis mimaride yardımcı toollar ile developerlar kendi servislerini düşük riskler ile hızlı bir şekilde deployment yapabiliyor.

Organizasyonel Uyum:
Bir çoğumuz büyük takımlar ve fazla kod parçacıkları yüzünden sorunlarla karşılaşmışızdır. Bu problemler daha ufak ekiplere ayrıldığında kontrol edilebilir hale gelir. Mikroservisler daha ufak ve uyumlu ekipler oluşturmamıza izin verirler. Bu aynı zamanda agile ilerlememize daha uygun bir yapıdır. Ekipler kendi mikroservislerinden sorumludur ve ihtiyaç dahilinde sadece onlarda geliştirme yapar, hızlı aksiyon alabilirler.

Dezavantajlar:

Mikroservisler monolotik uygulamalara göre geliştiriciler için farklı kompleks sorunlar getirir.

- Servisler arası iletişim ve uyumlu şekilde çalışması zorlayıcıdır. 100 lerce 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 mikroservis 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 hakim 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 - What are the challenges that one has to face while using Microservices ?

- Dizayn
- Güvenlik
- Test süreci
- Haberleşme

10 - How independent microservices communicate with each other?

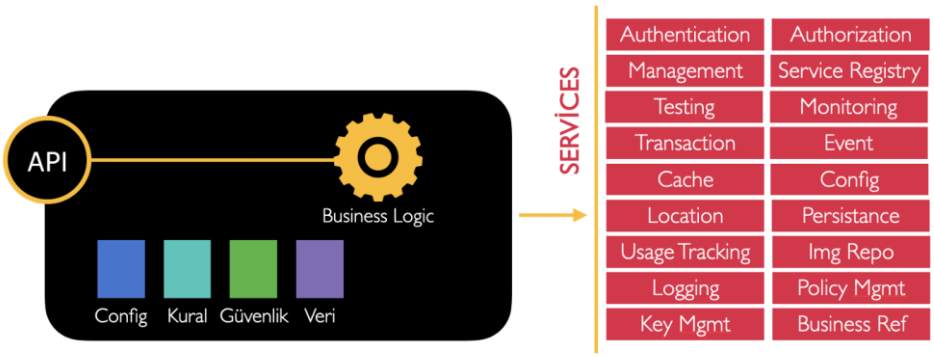
Servisler birbirleri ile HTTP istekleri (post, get, put, delete) ile haberleşirler. Yani bir data alışverişi ya da herhangi bir sebepten iletişim ihtiyacı olduğunda bir servis bir diğer servise

request atar, arkasından o servisten response bekler ya da ihtiyaç yoksa beklemeden devam eder.

11 - What do you mean by Domain driven design ?

DDD, geliştirilmiş bir teknoloji veya belirli yöntem değildir. DDD, karmaşık yazılım sistemlerinin geliştirilme aşamasında ve bu karşmaşık projeler hayata geçtikten sonra uygulamalarımızın sürekliliğini sağlamakta sıkça yaşanan temel problemlere çözümler getirmeye çalışan bir yaklaşımdır.

12 – What is container in Microservices ?

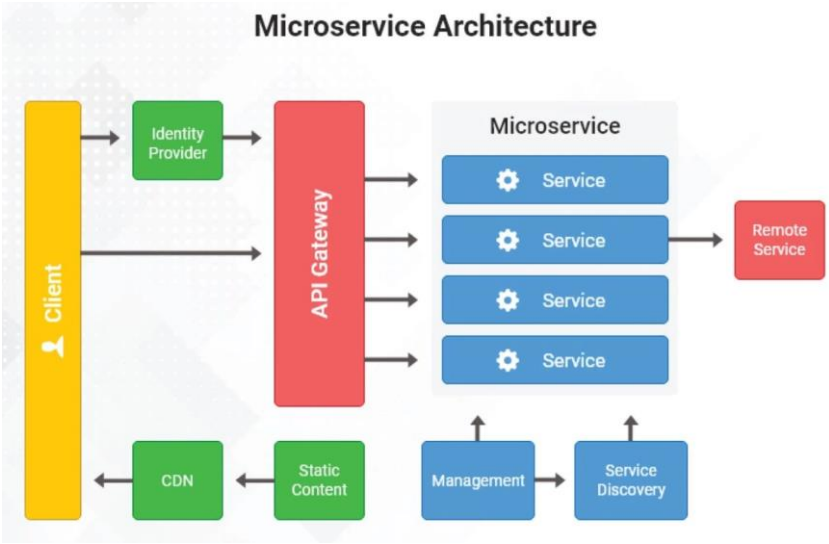


Microservis iş mantığı container içerisinde saklıdır. Container yapısı dışarıya servis sunar ve bir takım servisleri kullanır. İçerisinde Config, Policy(Kurallar), Security(Güvenlik) ve Veriyi birimlerini barındırır.

13 - What are the main components of Microservices architecture ?

- Microservisler
- Container
- Service Mesh
- Service Discovery
- API gateway

14 - How does a Microservice architecture work?


Microservice Architecture
Client
Identity Provider
API Gateway
Microservice
Service
Service
Service
Service
Remote Service
CDN
Static Content
Management
Service Discovery