PATİKA TÖDEB
JAVA SPRİNG BOOTCAMP

ÖDEV 6

BATUHAN AYYILDIZ

1-

Manual test, kalite kontrol analizcileri tarafından istenen özelliklere uygunluk, yazılımda bug kontrolü gibi özellikleri kontrol etmek için yapılan testtir. Automated test, yukarıda bahsedilen aynı özellikleri gerçekleştirmek için scriptlerle,kodlarla veya bazı araçlarla otomatik bir şekilde testin yapılmasıdır. Manual testing'te insan kaynaklı hatalardan dolayı sıkıntılı sonuçlar alınabilir. Ek olarak manual test'in uygulanması daha çok zaman alır. Manual testler, rastgele test üretimi yani doğaçlamaya izin verir. Automated testlerde bu esneklik payı yoktur.

2-

Assert sınıfı test yazmak için faydalı olan assertion methodları barındırır. Bu methodlar sonucun istendiği gibi olup olmadığını anlamak için kullanılabilir. Mesela yazdığımız kodun "false" dönmesini istiyoruz. Bunu test etmek için assertFalse()'ı kullanabiliriz. Sadece boolean condition olarak da değil iki nesnenin birbirine eşitliğine ya da bir nesnenin null olup olmadığı vb. birden çok özelliğe bakmak istediğinde test için bu sınıf kullanılabilir.

3-

Reflection kullanılabilir. Private değişkenlere ulaşmak için yazılan getter setter mantığına benzer bir mantık da private method'lar için kullanılabilir.

4-

Monolitik yapı yazılımda kendi kendine yeten, birbirine sıkıca bağlı olan ilişkiler içeren (tightly coupled) bir yasarım biçimidir. Bu yapını her küçük birimi yazılımın çalışması için ya da kodun çalışması için hazır olmalıdır. Bir öğe değişirse diğer öğeler de değişime ihtiyaç duyabilirler. Ayrıca birden fazla öğenin tek bir öğeye bağlanması gibi durumlar çok büyük database'lerin oluşmasına sebep olur ve yönetimi de zorlaşır. Bunlarının yanında avantajlı durumu test ve debug kısmında ortaya çıkar. Farklı servislerin iletişimi vb gibi ek test case senaryoları olmadığı için daha kolay test edilebilir. Ayrıca farklı servislerin iletişimi gerekmediğinden daha hızlı işlem gerçekleştirebilirler.

5-

Testler, kolaylıkla okunabilir olmalı. Böylelikle geliştiriciler arasındaki yanlış anlaşılmaların önüne geçilmiş olunur. Birden ortaya çıkmış sihirli sayıları veya stringleri kullanmamak gerekir. Mesela çemberin çevresini bulurken 2*3.14*yarıçap yazmak yerine 3.14, pi diye bir değişkene atanıp öyle kullanılmalıdır (2*pi*yarıçap). Testler deterministic olmalı. Test edilen kod değişse de her testte aynı davranışı sergilemeliler. Testler arası dependency'ler önlenmeli. Testler birbirinden olabildiğince bağımsız olmalı. Testlerde mantıksal koşulların (if,else vb) kullanımı az olmalı. Bu koşulların fazla olması deterministikliği ve okunabilirliği azaltır. Bir

test, birden fazla assertionlar içermemelidir. Çünkü testin fail etmesindeki sebebi bulmak zorlaşır. Testler düzenli olarak update edilmeli.

6-

Testin geçmediği birden fazla durumun bildirilmesi unit test yapısına aykırıdır. Çünkü yazılan testin birden fazla assertion içerdiği anlamına gelir ve unit testten beklenenlerin üstünde bir durum oluşturur. Unit test, tek assertion'a odaklı olmalı ve ona göre yazılmalıdır. Junit'de unit test yapısını dikkate aldığı için sadece ilk hata raporunu gösterir.

7-

Http endpointlerin veya JMX'in yardımıyla Spring Web uygulamasını yönetmeye ve izlemeye yarayan bir dependency'dir. Ölçüm metriklerini almak için, trafiği veya database'in durumunu anlamak için kullanılabilir. Kullanıcıya çalışan uygulama hakkında işlevsel bilgiler verir.

8-

Avantajları

- Daha kolay genişletilebilir, büyütülebilir. Update'ler daha kolay olur.
- Hata toleransı yüksektir. Bir servis çalışamaz hale geldiğin uygulama çalışmaya devam edebilir.
- Modüler yapıya sahip olduğu içi kod tabanı daha rahat anlaşılabilir. Çünkü sdace o modülün işlevine dayalıdır.
- Farklı teknolojiler kullanılarak farklı servisler kurulabilir. Modüller arasa daha az teknolojik bağımlılık vardır.
- Her modül diğerinden ayrı incelenebilir. Tüm uygulamanın yeniden yapılanmasına gerek kalmaz.

Dezavantajları

- Servisler arası iletişim çok fazla karmaşıklaşabilir. Küçük modüllere böldükçe iletişme geçmesi gerekenlerin sayısı artar.
- Daha fazla kaynak ister. Birden fazla database ve logs gereklidir
- Test ve debug aşaması daha zordur. Teset ederken daha fazla case'in göz önüne alınması gerekir.
- Küçük uygulamalar için iyi değildir. Uygulaması zor olur ve daha çok zaman alır.

9-

- Tasarım aşamasında mikro servislerin boyutuna, bağlantı noktalarına karar verme
- Mikro servisler çeşitli cloud çevrelerine yüklendiği için oluşan güvenlik tehlikelerine (bulut sistemi ek açıkalr oluşturabilir) çözüm bulma
- Test aşaması. Servislerin birbirine entegre edilmesi ve bu durumların test edilmesi karmaşık durumalr doğurabilir
- Karmaşık operasyonlar. Geleneksel izleme yöntemleri işe yaramayabilir.
- Servisler arası iletişimin sağlanması. Kötü konfigürasyon, gecikmede artışa ve farklı servislerin çağrılarındaki hızın azalamsına sebep olabilir.

10-

HTTP ya da gRPC (senkron) gibi inter-service protokolleri veya AMQP protokolüne (asenkron) sahip message broker'lar (Kafka RabbitMQ gibi) kullanarak iletişime geçerler. Senkron iletişimde client istek yolladığında cevap beklenir. Yani client thread'i cevap gelene kadar engeller. Asenkron'da ise client istek yolladığında thread'i engellenmez. Yani istek atıldığında cevap için beklenmez.

11-

Dünyadaki iş modellerinin ortak bir dille dijital dünyaya uyarlanırken yazılımların nasıl modellenmesi gerektiğini anlatan bir felsefedir. Domain, yazılımın ilgi alanını belirtir. Spesifik ve biraz detaylı olması iş tanımı açısından daha iyidir. Domain Driven Design da , domain'in dikkate alınarak yazılım geliştirilmesi sürecidir. Büyük ve kompleks süreçlerin parçalara yani domainlere bölünerek sorunun kendi domaininde çözülmesi gerektiğini savunur.

12-

Konteynırlar, VM(virtual machine)'e alternatif olarak izole bir çalışma ortamı yaratırlar. Vm'ye kıyasla belirsiz inişli çıkışlı büyümelerin olduğu yerde uygulamanın daha hızlı durup ayağa kalkmasını sağlarlar. Çünkü kaynak soyutlamayı kullanırlar. Böylelikle uygulamanın gerçekten ihtiyacı olan ana işletim sisteminin kernel'ini paylaşırlar. Bu sayede sadece uygulamaya gereken installations, dependencies , kod vb. kaynakları sunar. Bu sebeplerden mikroservislerle kullanıldıklarında daha iyi bir performans ve daha düşük seviyede bir altyapı ayak izi sunarlar. Daha düşük maliyete sahiptirler. Kodu güncelleme veya sürüm yükseltme işlemleri her mikro servis için ayrı ayrı uygulamanın işleyişini sekteye uğratmadan yapılabilir. Eğer uygulamanın tamamı tek bir VM'ye yüklenirse ve o VM hasar alırsa uygulama komple kapanabilir. Ama konteynırlar kullanılarak küçük VM kümelerine mikro servislerin replikasyonu yapılabilir. Bu durumda VM çökerse uygulama, VM kümesindeki mikro servisleri kullanarak çalışmaya devam edebilir.

13-

- Clients --> farklı idaresel işleri istekler yollayarak yöneten yapılardır. (Seacrh build configure gibi)
- Identity Providers -- > Client'dan gelen istekleri doğrulama yaparak API gateway arasındaki iletişimi sağlar.
- API Gateway --> Mikroservisle doğrudan ulaşılamaması için istekleri karşılayan bir giriş noktası gibi hizmet veren bir yapdıır.
- Mesajlaşma formatı --> Çeşitli uygulamalr kullanarak servisler arası iletişimin sağlanmasıdır.
- Veri işleme --> Databaseler kullanılarak verilerin saklanması ve idare edilmesini sağlar.
- Statik içerik --> Servisler arası iletişim sağlandıktan sonra mikroservisler client'a ulaştırılmak üzere hafızaya sabit bir içerik yüklerler
- Yönetim --> Node'lar üzerinde servisler arası dengeyi sağlar. Hataları belirler.
- Service Discovery --> Mikroservise, node'lardaki mikroservislerle iletişim rotasını bulmasını sağlayan bir rehber olarak hizmet eder.

14-

Her mikroservis uygulamanın belirli bir işine, görünümüne odaklanır. (Logging , veri araması gibi.) Bu mikro servisler de çeşitli mesajlaşma sistemleri (RabbitMq gibi) kullanarak iletişim kurarlar ve bütün bir uygulamaya oluştururlar. Sistemin temel hatları ve işleyiş biçimleri soru 13'teki gibidir.