Beykent Üniversitesi Fen ve Mühendislik Bilimleri Dergisi 2024, 17(1), 70-89
Beykent University Journal of Science and Engineering 2024, 17(1), 70-89

Makale Türü: Araştırma Makalesi
Paper Type: Research Article

Geliş tarihi/Received:26/01/2024
Kabul tarihi/Accepted:06/07/2024

MONOLİTİK VE MİKRO HİZMET MİMARİSİNİN PERFORMANS AÇISINDAN KARŞILAŞTIRILMASI

COMPARISON OF MONOLITHIC AND MICRO SERVICES ARCHITECTURE IN TERMS OF PERFORMANCE

DOI: 10.20854/bujse.1426301

Süleyman ŞANVER[1], Talat FIRLAR[2]*

Öz

Çalışmanın temel amacı yazılım dünyasında kullanılan mimari yöntemlerden monolotik ve mikrohizmet mimarilerini teorik ve uygulamalı olarak karşılaştırmaktır. Günümüz yazılım uygulamalarında monolotik mimariden, mikrohizmet mimariye geçiş süreçlerini görülme birlikte, mikrohizmet mimariden monolitik mimariye geri dönüşlerde olmaktadır. Monolotik ve mikrohizmet mimariyi performans açısından test edilmiştir. Bu çalışmada, monolitik ve mikrohizmet mimarilerin mantığına uygun iki farklı uygulama yapılıp, ölçümleri gerçekleştirilmiş, bu iki mimarinin çeşitli avantajları yanında, farklı dezavantajlarıda bulunmaktadır. Bu çalışmadan her iki mimarinin olumlu ve olumsuz yanları değerlendirilmiştir.

Abstract

The main purpose of the study is to theoretically and practically compare monolithic and microservice architectures, which are architectural methods used in the software world. In today's software applications, there is a transition from monolithic architecture to microservice architecture, and there is also a return from microservice architecture to monolithic architecture. Monolithic and microservice architectures have been tested for performance. In this study, two different applications were made and measured in accordance with the logic of monolithic and microservice architectures. These two architectures have various advantages as well as different disadvantages. In this study, the positive and negative aspects of both architectures were evaluated.

Anahtar Kelimeler: Mikro Hizmet, Monolitik Mimari, Yazılım Mimarileri, Performans Geliştirme, Jmeter.

Keywords: Microservice, Monolithic Architecture, Software Architectures , Performance Improvement, Jmeter.

[1] İstanbul Beykent Üniversitesi Lisansüstü Eğitim Enstitüsü Bilgisayar Mühendisliği Mezunu, suleymansanver@gmail.com, orcid.org/0009-0009-8689-4116

[2] *Sorumlu Yazar: İstanbul Beykent Üniversitesi Mühendislik Mimarlık Fakültesi, Bilgisayar Mühendisliği Bölümü, talatfirlar@beykent.edu.tr, orcid.org /0000-0002-0399-3955

1. GİRİŞ

Yazılım dünyası, günümüz projelerinin büyümesi, bakım ve yeni taleplerin kolayca eklenebilmesi veya bir modülün kolayca eklenip çıkarılabilmesi, uygulamayı ürün yapma süreçleri gibi taleplerin sonucunda dağıtık mimarilere yönelmiştir. Kurumsal projelerin yönetilmesi, ihtiyaçların belirlenmesi zorlu süreçlerdir. Bu yüzden yazılım projeleri geliştirilirken geniş yelpazeden bakılmalı, proje sonu görülmelidir ve seçilecek mimariye önceden karar vermek daha doğru olacaktır. Mikro hizmet mimarileri programlama dili açısından, bağımsız uygulamalar geliştirmeye fırsat tanımlamaktadır. Bu mimaride her servis/program kendi içinde bir uygulamadır ve bu servisleri birbirinden bağımsız yönetir. Mikro hizmet mimarileri, geleneksel yöntemlerle geliştirilen yazılımlardaki birçok problemi çözmeye odaklanır. Amazon, Google, Netflix gibi dünya çapında firmalar monolitik mimarilerini, mikro hizmet mimarilerine çevirdiler. Bu çalışmada; mikro hizmetlerin monolotik ve hizmet odaklı mimariler karşısındaki avantajları ve dezavantajları incelenmiştir. Her mimari türünün avantajları ve dezavantajları vardır ve doğru seçim projenin başarısını etkileyebilir.

Yazılım mimarileri bölümünde, yazılım mimarilerinin temel yapı taşlarını ve bileşenlerini, bu bileşenler arasındaki ilişkileri belirleyen, sistemin işlevselliğini, performansını, güvenilirliğini, ölçeklenebilirliği ve bileşenleri incelenmiştir. Yine bu bölümde tasarım desenleri hakkında bilgi verilmiştir. Monolotik mimari bölümde monolotik mimari incelenmiş, avantaj ve dezavantajlarından bahsedilmiştir. Bu mimarinin katmanları ve katmanların özelliklerinden bahsedilmiş, uygulamalardaki güçlü ve zayıf yanları incelenmiştir. Daha sonraki bölümde, mikro hizmet mimarisinin özellikleri anlatılmış ve en son kısımda seçilen iki mimariye göre hız ve performanslarına göre karşılaştırmalı olarak uygulama yapılmıştır.

2. YAZILIM MİMARİLERİ

Mikro hizmetler terimi ilk defa 2005 yılında Dr. Peter Rodgers tarafından bulut bilişim üzerine bir sunum sırasında “Mikro İnternet Hizmetleri” olarak kullanıldı. Rodger’ın çalışması 1999 yılında, kodu daha az kırılgan hale getirmek ve büyük ölçekli, karmaşık yazılım sistemlerini değişime dayanıklı hale getirmek olan Hewlett Packard Labs’daki Dexter araştırma projesi ile başladı (Russell ve ark., 2004).

2007 yılında Juval Löwy, her sınıfın bir hizmet olduğu sistemler inşa edilmesi çağrısında bulundu. Löwy hizmetlerin bu kadar parçalı kullanımını destekleyecek bir teknolojinin kullanılması gerektiğini fark etti ve WCF(Windows Communication Foundation)’i bunu yapacak şekilde genişletti.

2011 yılında Venedik’te Yazılım Mimarları etkinliğinde yazılımcılarının çoğunun ortak bir mimari yaklaşım olarak gördükleri bu stile Mikro Hizmet adını verdiler (Dragoni vd., 2017). 2012’nin Mayıs ayında aynı grup Mikro Hizmetler olarak adını değiştirdiler. James Lewis Mart 2012 de Krakow’da, Fred George’da bir vaka çalışması olarak ayrı ayrı sunumlar yaptılar. Netflix’te bulut sistemlerinin eski yöneticisi Adrian Cockcroft kendisiyle yapılan bir görüşmede bu yaklaşımı “ayrıntılı hizmet odaklı mimari” olarak isimlendirdi (Farrow, 2012).

Dünya çapında büyük şirketler eski mimari yaklaşımlardan bu mimariye doğru geçmişlerdir. Bunların arasında Netflix, Amazon ve Uber sayılabilir.

Yazılım mimarisi, bir yazılım sisteminin temel yapı taşlarını ve bileşenlerini tanımlayan, bu bileşenler arasındaki ilişkileri belirleyen, sistemin işlevselliğini, performansını, güvenilirliğini,

ölçeklenebilirliğini ve diğer önemli nitelikleri tasarlayan disiplindir. Bu mimari , bir projenin geliştirilmesi sürecinde tüm ekip üyeleri arasında bir rehberlik sağlar. "Yazılım mimarisi , yazılım bileşenlerini, ilişkileri ve özellikleri kavramak için kullanılan yapılandırılmış bir çerçeveyi temsil eder. Bu kavramlaştırma , sistem tasarım sürecini basitleştirir ve farklı projeler için tasarım bileşenleri ve deseni, tekrar kullanımı ile ilgilenen bir soyutlamaya olanak tanır" (Bass ve ark., 2012).

Yazılım mimarileri genellikle , karmaşık sistemlerin tasarımları ve sistem içerisindeki parçaların birbirleriyle olan iletişimleri ile ilgilenir. Ayrıca bu parçaları daha sonra başka uygulamalara hizmet vermeleri için kullanılır. Bu çalışmada görüldüğü gibi mikrohizmet mimarisinin temel amaçlarından biri de budur. "Yazılım mimarilerini belgelemek , paydaş iletişimini kolaylaştırır, tasarım ve tasarım bileşenleri ile ilgili , erken ve önemli kararları belgeleyerek ve farklı projeler için yeniden kullanılabilir bir bilgi tabanı olarak işlev görmesi önemli bir süreçtir" (Clements ve ark., 2010).

Tasarım desenleri uygulanması tekrarlayan işlerin engellenmesi ve buna göre süreçleri yönetme prensibini kavramaktır. Süreçlerinizde tekrarlayan işlemler varsa süreçlerinizi gözden geçirmeniz gerekebilir. "Bir tasarım deseni , nesne yönelimli sistemlerde tekrarlanan bir tasarım problemine çözüm sunan genel bir tasarımı adlandırır , gerekçelendirir ve açıklar" (Buschmann ve ark. , 1996).

Uygulamalarımızı mimari üzerine inşa ettiğimizde aynı zamanda böldüğümüz parçalar anlam ve görev kazanırlar. Her parçanın kendine ait sorumluluğu vardır. Bu yaklaşım parçaya anlam kazandırır. "Yazılım mimarisi tasarımı , yazılım gereksinimlerini yazılımın yapısını ve davranışını açıklayan bir mimariye dönüştürme sürecidir" (Albin ve Pautasso, 2008).

Yazılım Mimarileri, yazılım dünyasında birçok farklı yazılım mimarisi yaklaşımı bulunmaktadır. Bu yaklaşımlar, projenin gereksinimlerine, boyutuna, özelliklerine ve hedeflere göre seçilir. Temel yazılım mimarisi türleri olarak; Katmanlı (Layered Architecture) , Mikrohizmet (Microservices Architecture), MVC (Model-View-Controller), MVVM (Model-View-ViewModel), Monolitik, Dağıtık, Olay Odaklı (Event-Driven Architecture), N-Tier (Çok Katmanlı) Mimariler olarak söylenebilir. Projenin gereksinimlerine ve hedeflere bağlı olarak, bu yaklaşımlardan biri veya birkaçının birleşimi tercih edilebilir. Her mimari türünün avantajları ve dezavantajları vardır ve doğru seçim projenin başarısını etkileyebilir.

3. MONOLİTİK MİMARİ

Monolitik mimariler tek veya birden fazla katmandan oluşan mimarilerdir. Birden fazla katmandan oluşsa bile gün sonunda tek parça haline gelen yapılardır. Uygulamaların tüm parçalarının tek bir çatı altında geliştirilmesi ve yönetilmesini sağlayan bir yazılım stilidir. Uygulama tek bir çatı altından sunuculara dağıtılır. Yatay ölçeklendirme yapılamaz eğer yapılmak istenirse uygulama bütünüyle sunuculara kopyalanmalıdır.

Monolitik uygulamaların avantajları basitlik, performans ve başlangıçta düşük karmaşıklık olarak öne çıkar. Ancak, büyüdükçe ve karmaşıklık arttıkça ölçeklenebilirlik , bakım zorlukları ve teknoloji çeşitliliği gibi dezavantajlar ortaya çıkabilir.

Monolitik mimari , yazılım geliştirme alanında sıkça kullanılan bir tasarım yaklaşımıdır. Bu yaklaşımda , bir uygulama tüm bileşenleriyle tek bir büyük ve bütünleşik yapı içerisinde

bulunur. Kullanıcı arayüzü , iş mantığı ve veritabanı gibi farklı işlevler, genellikle aynı kod tabanı içerisinde yer alır ve tek bir çalıştırılabilir dosya olarak sunulabilir.

Monolitik mimari , yazılım geliştirme sürecini basit ve hızlı hale getirebilir. Başlangıçta, tüm ekip aynı kod tabanı üzerinde çalışarak işbirliği yapabilir ve geliştirmeyi kolaylaştırabilir. Ayrıca, monolitik uygulamalar genellikle hızlı bir şekilde dağıtılabilir, çünkü tüm bileşenler bir arada olduğu için dağıtım süreci daha az karmaşıktır.

Ancak, monolitik mimarinin bazı dezavantajları da vardır. Uygulama büyüdükçe, ölçeklenebilirlik sorunları ortaya çıkabilir. Bir bileşeni ölçeklendirmek, tüm uygulamayı etkileyebilir. Çünkü tüm bileşenler aynı kod tabanında yer alır. Aynı şekilde, farklı ekiplerin farklı teknolojiler kullanma esnekliği sınırlı olabilir. Bir bileşende yapılan değişiklikler diğer bileşenleri de etkileyebilir, bu da bakım süreçlerini karmaşıklaştırabilir.

Monolitik uygulama birden fazla modül içeren tek bir kod tabanına sahiptir. Tüm uygulamayı derleyen tek bir yapısı vardır ve 3 katmandan oluştur.

- Sunum Katmanı
- İş Katmanı
- Veri Erişim Katmanı

Çalışma yapısı, istemci sunucuya hangi veriyi görmek istediğine dair istek iletir, sunucuda ilgili isteği kontrol eder, güvenlik, doğrulama, yetkilendirme vb. kontroller yapılır ve istek doğruysa veri tabanında sorgulama yapar ve aynı hiyerarşide veri tabanından cevap alınır ve sunucuya bu cevabı istemciye veri olarak döner.

Monolitik mimariyi daha yakından inceleyebilmek için sunum katmanı incelenerek fikir sahibi olunabilir. İlk olarak sunum katmanının istemci ve veri tabanı arasında köprü olduğu görülür. Sunum katmanı makalenin kendisidir.

- Sunum Katmanı: İstemciden gelen “get”,”post” isteklerinin yönlendirildiği katmandır. Örnek: ‘User/1’ veya ‘UserSave/sare’ olarak istekler yapılır.
- İş Katmanı: Sunum katmanından gelen isteklere ait doğrulamalar, yetkilendirmeler, iş kuralları gibi kontrollerin yapıldığı yerdir.
- Veri Erişim Katmanı: İş katmanından gelen isteklerin veri tabanına işlendiği ya da gönderildiği yerdir

Büyük projelerde çalışırken veya yoğun trafikli projelerde, uygulamaların yükünü hafifletmek veya yönetmek için ‘Yük Dengeleyici’ kullanılması gerekekir. Ürün sayfasında yoğunluk var ve diğer sayfalar çok yoğun değildir. Ürün servisinde bir iyileştirme yapılmak isteniyor ancak yük dengeleyici üzerinde sadece ürün servisi için ayar yapılamıyor bütün projenin kopyalanması ve bu şekilde yönetilmesi gerekiyor.

Gerçek bir monolitik uygulamada ‘eShopOnWeb’ uygulaması ve uygulamaya ait “ApplicationCore” katmanı, yani makalenin iş katmanına denk geliyor. “Infrastructure” katmanı veri tabanı erişim katmanına karşılık geliyor. Web projesi de uygulamanın sunum katmanıdır.

- Web projesi “ApplicationCore” katmanını çağırır.
- “ApplicationCore” katmanı “Infrastructure” katmanını çağırır.

- Gün sonunda Web projesi oluşur ve diğer bütün katmanlar Web projesinin içerisinde derlenmiş olarak bulunur.

Yukarıdaki bilgiler ışığında iplik düzeyinde giysi modelleme (yarn level cloth) üzerine çalışmalar incelenmiş ve iplik düzeyinde giysi modellemesinin rendering işleminin hesaplama maliyetinin çok yüksek olduğu görülmüştür. İplik seviyesinde kumaş modellemesi, tek tek ipliklerin davranışını modelleyen bir bilgisayar tabanlı simülasyonudur. Bu, ipliklerin gerilme, deformasyon ve birbirleriyle etkileşim gibi özelliklerini simüle etmeyi de içerebilir. Ayrıca kumaşların drapaj ve

Monolitik Mimarinin Güçlü Yanları;

Kolay Dağıtım: Monolitik uygulamaların tek parça oluşu dağıtım için avantaj sağlar. Bütün projeyi tek bir seferde ilgili sunucuya dağıtabiliriz. Tek parçayı dağıtmak onlarca servisi dağıtmaktan çok daha kolaydır.

Hata Ayıklama: Tek bir proje olduğundan tek bir çalıştırma ile hataları kolayca yakalanabilir.

Test Kolaylığı: Bütün işlemler tek bir projede olduğundan test etmek kolaydır.

- Geliştirmesi basittir.
- Yatay ölçeklendirme daha kolay yapılıyor. Yük dengeleyicisi üzerinde projenin kopyalarını çalıştırmak yeterli oluyor.

Monolitik Mimarinin Zayıf Yanları;

- Proje büyüdükçe karmaşıklık ve geliştirme zorluğu ortaya çıkar.
- Uygulama karmaşık olduğunda herhangi bir değişiklik yapmak zorlaşır.
- Uygulamanın boyutu projenin sunucuda başlama süresini uzatabilir.
- En küçük bir değişiklikte uygulamanın bütün çalıştırılıp dağıtımı yapılmalıdır.
- Uygulama modülleri arasında farklı ölçeklendirme , kaynak artırımı yapılmak istenirse zordur.
- Herhangi bir modüldeki bir hata uygulamanın çalışmasını bozabilir. Ayrıca ölçeklendirme yapılmışsa bile aynı hata hepsinde bulunacağından tüm uygulamayı etkileyebilir.
- Yeni bir teknolojiye entegre etmek için uygulama bütünüyle değiştirmek gerekebilir.

Monolitik Mimari Performans Artırımı için; Kod Optimizasyonu, Veritabanı İyileştirmeleri, Önbellekleme, Paralelleştirme, Donanım İyileştirmeleri, Gereksiz Bağımlılıkları Azaltma, Gereksiz Kaynak Kullanımını Azaltma , Caching (Önbellekleme) Kullanımı, Monitörleme ve Analiz, Kod Profillemesi, Veritabanı Normalizasyonu, HTTP Caching gibi yöntemler bulunur. Unutulmamalıdır ki performans artırma stratejileri uygulamanın türüne, kullanım senaryolarına ve altyapıya bağlı olarak farklılık gösterebilir. Performansı artırmak için birçok farklı yaklaşımı deneyebilir ve gerektiğinde ölçümler yaparak en etkili çözümler belirlenebilir.

4. MİKRO HİZMET MİMARİSİ

Mikro Hizmet Mimari bağımsız çalışabilen servisler topluluğudur. Bu mimaride servisler gevşek bağlıdır ve Tek Görev/Cevap prensibine uygun olarak çalışmaktadırlar.

Mikro hizmetler SOA'nın yorumudur. SOA tabanlıdır. Her servis kendine ait bir dünyada çalışır. Her bir servis kendine ait sunucuda çalışır ve kendine ait veritabanı olabilir (Yerelbt, 2022).

Mikro hizmet küçük API üzerinden iletişim kurabilen ve bağımsız hizmetlerden oluşan servislerdir. Büyük uygulamanın küçücük kısımlara ayrılarak yönetilmesidir. Kullanıcın tek bir talebinde birden fazla Mikro hizmet çağırılabilir.

Mikro Hizmet, kendi içinde tek sorumluluğu olan veya tek iş yapan bağımsız projelerdir. Farklı programlama dillerinde geliştirilme imkanı ve farklı veritabanı teknolojileri kullanılabilir. Kısacası, mikro hizmet mimari stili , her biri kendi sürecinde çalışan ve genellikle bir HTTP kaynak API'si olan , hafif mekanizmalarla iletişim kuran , küçük hizmetler paketleri olarak uygulama geliştirmemizi sağlayan bir yaklaşımdır.

Yazılım geliştirme aşamasında da çok kolaylıklar sağlamaktadır. Her bir hizmet farklı yazılım takımlarına, ekiplerine bölünerek yönetilebilir. Bu yazılımı geliştiren kişi sadece ilgili kısmı bilir ve oraya odaklanır. Bu şekilde ki yazılım geliştirme ve bakım daha kolaydır.

Genel olarak API Gateway olarak bilinen ortak giriş noktaları vardır. İstemci yani web uygulaması ya da mobil uygulama buraya istek atar ve buradan ilgili mikro hizmete yönlendirilir. Gateway deki yönlendirme yazılım geliştirme esnasında gateway'e öğretilir. Örnek olarak kullanıcının yorumları için yorum mikro hizmetine gideceği bu tanımlamalarla öğretilir. Kullanıcı da ekrandan yorumlara bastığında ya da incelediğinde buradaki yorum mikro hizmetine yönlendiriliyor. Ayrıca yetkilendirme işlemleri burada yapılır. Yani kullanıcının ilgili hizmeti çağrımaya izni var mı yok mu soruları bu kısımda kontrol edilir. Burada birden fazla mikro hizmete istek gidebilir. Yorum mikro hizmetinden cevap alınır ve gateway'e gelir ve aynı şekilde istemciye yani aslında kullanıcıya sonuç gösterilir.

Mikro Hizmet Mimarisinin Faydaları;

Teknoloji Çeşitliliği; Mikro Hizmet mimarisinde hizmetler ayrı olduğu gibi dil bağımsızdır. Projelerde Net, PHP, Go, Ruby vb. istenilen bir programlama dilinde geliştirilebilir. Ekiplere yeni personel alınırken çok daha geniş teknoloji alanından insanlar seçebilme fırsatı sunuyor. Örnek olarak yeni alınan personel PHP biliyorsa farklı bir hizmeti, PHP ile geliştirmesi istenilebilir veya hangi dil o hizmete uygunsa bu şekilde de seçim yapılabilir. Mikro Hizmet ile teknoloji değişim kararları daha kolay verilebilir. Çünkü geliştirilen yer bütün projeye göre çok daha küçük olduğundan teknoloji denemeleri yapılabilir.

Bağımsız; Uygulamadaki bütün hizmetler birbirinden bağımsızdır. Geliştiriciler ve ekipler birbirinden bağımsızdır, böylelikle geliştirme ve test süreçleri ayrı ayrı yürütülebilir. Birbirlerini beklemeye gerek yoktur.

Çeviklik; Mikro Hizmetler ekiplerin çevik olmasına da katkıda bulunur. Büyük bir hizmetin küçük bir parçasını değiştirmeye çalıştığımızda avantajını görmüş oluruz. Yeni bir istek hızlıca geliştirilip projeye eklenebilir.

Anlaşılabilirlik; Proje küçük parçalara ayrıldığından anlaşılması kolaydır. Kod okuması çok daha kolay yapılabilir veya ekibe katılan bir kişi ilgili hizmeti çok daha kısa sürede öğrenebilir.

Bağımsız Dil; Standart bir teknolojiye veya bir programlama diline bağlı değiliz. İlgili hizmet için en iyi teknoloji hangisiyse o kullanılabilir.

Ölçekleme; Monolitik yapıda ölçeklendirme komple bütün projeye yapılır. Ancak mikro hizmetler küçük parçalara ayrıldıklarından sadece ilgili hizmete ölçeklendirme yapılabilir. Her bir servis bağımsız ölçeklendirilebilir. Yoğun bir şekilde kullanılan hizmeti ölçekleyip sadece o hizmetin sunucusuna iyileştirme yapmak bize zaman/bütçe maliyetini minimum düzeyde tutulmasını sağlar. Örnek olarak Twitter uygulamasının en yoğun kısmı yorumlar bölümüdür. Bu yorumlar kısmı mikro hizmet olarak geliştirilirse ve sadece bu hizmete ait kaynak artırımı yapabilme fırsatı bulunur.

Dağıtım; Mikro hizmet mimarisinde sadece geliştirme yapılan hizmetin canlıya alınması yeterlidir. Projenin sadece %5 in de geliştirme yaptıysak sadece yüzde %5 ini dağıtım yapıyoruz. Hizmetin hızlı bir şekilde dağıtımının yapılması yazılımcılar açısından da çok önemlidir. Eğer bir problem meydana gelirse sadece problemin olduğu hizmette geliştirme yapılır ve dağıtımı gerçekleştirilir.

Yeniden Kullanılabilirlik; Bağımsız geliştirilen bir hizmet de çok küçük değişiklikler yapılarak ya da hiç geliştirme yapılmadan ihtiyaç olan başka bir projeye hizmet olarak sunulabilir. Örnek olarak kullanıcı bilgileri için bir mikro hizmet olsun , bu hizmeti tüm uygulamalarda kullanabilir düzeye getirilir. Böylece yeni bir servis geliştirmek için oluşacak zaman ve para maliyetinden kurtula bilinir.

Test Kolaylığı; Hizmetlere ayrıldığından bütün uygulamanın bitmesini beklemeden hizmetler bağımsız olarak testleri gerçekleştirilebilirler.

Hata İzolasyonu; Herhangi bir hizmette hata alındığında bu bütün sistemi etkilemez.

Mikro Hizmet Mimarisinin Zayıf Yanları;

Karmaşıklık; Hizmetlerin sayısı arttıkça karmaşıklık artar. Hizmetler ayrı ayrı olduğundan hepsi için ayrı optimizasyonlar ve veritabanı oluyor. Hepsinin ayrı ayrı yapılandırılması gerekmektedir.

Test Etme Zorluğu; Hizmetler birbirinden bağımsız olduğu için uygulamaların test edilmesi için hepsinin ayağa kaldırılması gerekmektedir. Bu yazılımcı için zaman maliyetidir.

İzleme Zorluğu; Proje içerisinde kullanılan her hizmet bağımsız olacağından kendisine ait izleme ortamına ihtiyaç duyacaktır. İzleme işi içinde ek kaynak ayırmak gerekebilir.

Maliyet; Genellikle hizmetler monolotik mimariye göre daha maliyetlidir.

Güvenlik; Hizmetler arasında iletişim, veri alışverişi çok sık gerçekleştiğinden uygulama güvenliğini sağlamak için ayrıca bu konuya odaklanmak lazım. Bu hizmet başka bir uygulamaya veya dışarıya açıldığında bu hizmete ait yetki tanım ve kontrolleri yapılmalıdır. Mikro hizmetler birbirlerine çağırdıklarında ilgili hizmetin diğer hizmeti çağırabilmek için yetkisinin olması ve kimlik doğrulama işlemlerinden geçmesi gerekmektedir.

Dağıtık Sistem Zorlukları; Mikroservisler, farklı sunucularda veya konteynerlerde çalışabilir. Bu durum ağ hataları, veri bütünlüğü sorunları ve hata ayıklama zorlukları gibi dağıtık sistem zorluklarını beraberinde getirebilir.

Veri Bütünlüğü; Mikroservislerin ayrı ayrı veritabanlarını kullanması, veri bütünlüğü ve tutarlılığı sağlamayı zorlaştırabilir. Bu servisler arasında veri senkronizasyonu gerektirebilir.

Yüksek İletişim Trafik; Servisler arası yoğun iletişim, ağ trafiğini artırabilir. Bu, servisler arasındaki performansı olumsuz etkileyebilir.

Dağıtık Veritabanları; Mikroservislerin farklı veritabanları kullanması, veri yönetimini zorlaştırabilir. Dağıtık veritabanlarını yönetmek ve senkronizasyon sağlamak zor olabilir.

Servis Bağımlılıkları; Mikroservis mimarisi, servisler arası bağımlılıkların artmasına neden olabilir. Bir servisin aksaması, diğerlerini de etkileyebilir.

Geliştirme ve Dağıtım Süreci; Mikroservislerin ayrı ayrı geliştirilip dağıtılması, koordinasyon gerektiren bir süreçtir. Yeni özelliklerin tüm servislere entegrasyonu zor olabilir.

Maliyet Artışı; Birden çok servis ve altyapının yönetimi, maliyeti artırabilir. Altyapı, işletim ve güvenlik için ekstra kaynaklar gerekebilir.

Mikroservis mimarisinde performansı artırmak için aşağıdaki stratejileri ve yaklaşımları göz önünde bulundurulur. Mikroservisleri Ölçeklendirme, Servisler Arası İletişimi Optimize Etme, Hafif Konteynerler Kullanma, Düşük Gecikme Süreleri, Caching (Önbellekleme) Kullanımı, Performans Testleri, Optimize Edilmiş Veritabanı Kullanımı, Servis Bağımlılıklarını Azaltma, Monitörleme ve Analiz, Uygulama Optimizasyonu, Dağıtım ve Dağıtım Araçları, Hızlı Hata Tespiti ve Düzeltme gibi yöntemler bulunur. Unutulmamalır ki mikroservis mimarisinin karmaşıklığı ve dinamizmi, performansı artırma sürecini bazen zorlaştırabilir. Her bir stratejiyi ve yaklaşımı dikkatlice değerlendirilerek, belirli ihtiyaçlara ait en uygun çözümler seçilmelidir.

5. MİKROSERVİS İLE MONOLİTİK MİMARİLERİN KARŞILAŞTIRILMASI

Mikroservis mimarisi ve monolitik mimari, yazılım uygulamalarının farklı şekillerde yapılandırılmasını temsil eder. İşte mikroservis ve monolitik mimarilerin bazı temel karşılaştırmaları:

Bağımsızlık;

- **Monolitik Mimarisi:** Tek bir monolitik yapı olduğundan, herhangi bir bileşenin güncellenmesi veya değiştirilmesi tüm uygulamayı etkileyebilir.
- **Mikroservis Mimarisi:** Mikroservisler bağımsız olarak geliştirilir, dağıtılır ve ölçeklendirilir. Bu, bir mikroservisin güncellenmesinin diğerlerini etkilemediği anlamına gelir.

Ölçeklendirme;

- **Monolitik Mimarisi:** Monolitik uygulamaların tüm bileşenleri birlikte ölçeklenir. Yani tüm uygulama aynı oranda ölçeklenir.
- **Mikroservis Mimarisi:** Tekil servisler, ihtiyaca göre bağımsız olarak ölçeklenebilir. Sadece yoğun kullanılan servisler ölçeklenir.

Bakım ve Dağıtım;

- **Monolitik Mimarisi:** Yeniden dağıtım genellikle tek bir birimde yapılır. Tüm uygulama birlikte güncellenir.
- **Mikroservis Mimarisi:** Servislerin bağımsız olması, daha hızlı dağıtım ve güncelleme süreçlerini sağlar. Yalnızca değişiklik yapılan servisler güncellenir.

Teknoloji Çeşitliliği;

- **Monolitik Mimarisi:** Monolitik uygulamalarda tüm bileşenler aynı teknoloji yığını üzerine kurulur.
- **Mikroservis Mimarisi:** Her mikroservis kendi teknoloji yığınına sahip olabilir. Farklı diller ve teknolojiler kullanmak mümkündür.

Hata Yayılması;

- **Monolitik Mimarisi:** Bir bileşende meydana gelen hatalar tüm monoliti etkileyebilir.
- **Mikroservis Mimarisi:** Tek bir servisteki hata , diğer servisleri etkilemez. Ancak hataların yönetimi daha karmaşık olabilir.

İletişim ve Performans;

- **Monolitik Mimarisi:** İç içe geçmiş işlevselliğe sahip olduğundan iletişim daha kolaydır. Ancak ölçeklendirme ve performans yönetimi zor olabilir.
- **Mikroservis Mimarisi:** Servisler arası iletişim ağ üzerinden yapılır ve bazen daha karmaşık hale gelebilir. Ancak performansı belirli işlevselliğe özgü optimize etmek daha kolaydır.

Hangi mimarinin tercih edilmesi gerektiği, proje ihtiyaçlarına, ekibin yeteneklerine ve projenin özelliklerine bağlıdır. Her iki yaklaşımın da avantajları ve zayıf yönleri vardır ve seçim yaparken iyi bir değerlendirme yapmak önemlidir.

6. UYGULAMA

Bu çalışmada monolitik ve mikrohizmet mimarisinden oluşan 2 uygulama vardır. Uygulamalar kendi içerisinde servis methodları bulundurmaktadır. Monolitik ve mikrohizmet mimarisi içerisinde bulunan servis methodlarına birden fazla istekte bulunarak performans değerlendirmesi hedeflenmektedir. Testler Jmeter uygulaması ile yapıldı. Uygulama rapor kısmında Jmeter uygulamasına ait sonuç raporları bulunacaktır**.**

6.1. Monolotik Uygulama Örneği

Uygulama Monolotik mimari üzerine kurulmuştur. Uygulama tek katmandan oluşur. Bu katmanın adı API katmanıdır. Uygulama içerisinde “Dto”, “Servis”, “Controller” nesneleri bulunmaktadır. Bu nesnelere ait detayı proje yapısını incelenirken değinildi. Uygulama ile sql server’ın iletişim kurabilmesi için “entityframework” kullanılmıştır. “Entityframework” bir “ORM(Object Relational Mapping)” aracıdır. “ORM” araçları veritabanlarını nesneler halinde oluşturarak projeye dahil edebilebilmesini sağlarlar.

Uygulamada veri tabanı olarak Microsoft SQL server kullanılmıştır. Veritabanında kullanılmak üzere Microsoft tarafından geliştirilen “Adwenturework” veritabanı kullanılmıştır.

Projede “Aspnet Core Web Api” kütüphanesi kullanılmıştır. Projede “Controllers” klasörü bulunmaktadır. Onunda altındada “Salesordercontroller” bulunuyor. Monolotik uygulama için testler bu “controller” üzerinden yapıldı. Bu “controller” içerisindeki Get methodu çağırıldı. Get methodu “Adwenturework db” sine gidip satış bilgileri, satış detayı ve bunlara bağlı ürün bilgilerini tablolardan okunarak getirecektir.

Projede “Dto” klasörü bulunmaktadır. Bu klasör içerisinde “SalesOrderListDto” dosyası bulunuyor. Bu “Dto” nesnesi veritabanından gelen “entity” modelin istemciye gönderilirken belirlenen kullanıcıların bilmesi gereken özellikleri filtreleyip istemciye dönülecektir.

Projede Models klasöründe veritabanı nesneleri bulunmaktadır. Bu nesneler “Adwenturework” veritabanına aittir. Biz bu projemizde “SalesOrderHeader”, “SalesOrderDetail”, “Product” tabloları kullanıldı. Bu tablolardan en çok sipariş edilen 50 ürünü istemciye dönüldü. Veritabanını projeye ekleyebilmek için “Entityframework” e ait aşağıdaki kodun konsolda çalıştırılması gerekir.

Çalıştırılacak Kod Bloğu: dotnet ef dbcontext scaffold "Server=.\;Database= AdventureWorks2019;Trusted_Connection=True;" Microsoft.EntityFrameworkCore.SqlServer -o Models

Kod bloğu çalıştırılırken daha sonra “Models” klasörü aşağıdaki gibi oluyor. Artık veritabanı ve nesneler proje tarafından iletişim kurulabilir hale getirildi.

Servis klasörü altında bulunan “SalesOrderService” dosyasından veritabanında bulunan “SalesOrderHeader”, “SalesOrderDetail”, “Product” tablolarındaki kayıtları ilişkilendirerek en çok satılan ilk 50 ürün bilgisi çekilir. Aşağıdaki resimde görüldüğü üzere “GetMaxSalesOrders” metodu en çok satılan 50 ürün bilgisini getirdi. Veritabanından dönen modelin tamamını değil sadece “ProductName”, “ProductID” ve “SalesOrderCount” alanlarını oluşturulan “SalesOrderListDto” ya çevirip geri dönüldü. Veritabanı “AdwentureWorks2019Context” nesnesi üzerinden iletişim kuruldu.

“SalesOrderController” sınıfında oluşturulan servisi çağırıp istemciye cevap dönüldü. “SalesOrderController” veritabanını bilmiyor. Tek görevi servis nesnesini çağırmak oluyor. N-Katmanlı mimarilerde bu klasörler az bağımlı olarak tasarlanır. Ancak bu projede bu şekilde yapıldı.

Proje IIS üzerinde ayağa kaldırıldı. Kaldırıldıktan sonra browser üzerinden http istekleri atılıp veriler görüldü. “SalesOrderController” içerisindeki “GetMaxSalesOrders” methodu http://localhost:3801/salesorder/GetMaxSalesOrders olarak çağrıldı. Ekranda veritabanında bulunan en çok satılan 50 ürün bilgisi listelendi.

6.2. Monolitik Uygulama Jmeter ile Yük Testi

Monolitik uygulamanın yük testi Jmeter ile yapılmıştır. Bu bölümde yük testi yapıldı. Geliştirilen uygulamalar canlı ortama alınmadan önce yük testi yapılması büyük önem arz etmektedir. Uygulamaların farklı istekler üzerinde nasıl tepki verdiğinin görülmesi önemlidir. Geliştirmeler bu sonuçlara göre iyileştirebilir, farklı yöntemler düşünebilir, farklı çözümler üretilebilir. Yük testleri, uygulamaların kalitesini ve kullanılabilirliğini artırmada yardımcı olurlar.

Yük testi yaparken 3 saniye içerisinde 1000 kullanıcıya çıkıldı ve 1000 kullanıcı istekte bulunduğunda monolitik uygulamanın nasıl tepki verdiği görüldü. 1000 kullanıcı isteği yerel bilgisayarda Jmeter uygulaması üzerinde gerçekleştirildi.

Monolitik uygulamada jmeter gerekli konfigürasyon ayarları yapıldı.

Http Bilgisi, Sunucu İsmi, Port Numarası, Http İstek Türü, Yol bilgileri girilip test ayarları yapıldı.

Monolotik uygulamasında konfigürasyon ayarları yapıldıktan sonra rapor sonuçlarının görülebilmesi için Jmeter'ın farklı özellikleri kullanıldı. Jmeter'a ait Özet rapor ve Sonuç tablosunu gösteren özellikleri, test parçacığının içine eklendi.

6.3. Sonuç Tablosu Rapor

Jmeter'ın 1000 kullanıcı ile testine ait 1000 istek Tablo 1'de görüldüğü üzere başarıyla tamamlanmıştır. Monolotik uygulamada 1000 request için tek bir servis çağrılıp tek bir dönüş olmuştur. Sonuç tablosunda bütün istekler başarılı bir şekilde görülüyor. 3 saniye içerisinde 1000 kullanıcının istek atması sonucu elde edilen rapor bilgisine aşağıda yer verilmiştir.
Elapsed : Dönüş süresi, **Label :** İstek Türü, **Responce Code :** Cevap Kodu, **Data Type :** Metin Dili,**Url :** İstek Atılan Adres

Testin sonucunda 1000 istek ortalama **3,684** saniye sürmüştür.

Tablo 1: Monolith İstek Cevap Süreleri.

No	Süre	İstek	Cevap	Tip	URL
1	4178	HTTP	200	Metin	http://localhost:3801/salesorder/GetMaxSalesOrders
2	4180	HTTP	200	Metin	http://localhost:3801/salesorder/GetMaxSalesOrders
3	4179	HTTP	200	Metin	http://localhost:3801/salesorder/GetMaxSalesOrders
4	4179	HTTP	200	Metin	http://localhost:3801/salesorder/GetMaxSalesOrders
5	4179	HTTP	200	Metin	http://localhost:3801/salesorder/GetMaxSalesOrders
6	4178	HTTP	200	Metin	http://localhost:3801/salesorder/GetMaxSalesOrders
7	4179	HTTP	200	Metin	http://localhost:3801/salesorder/GetMaxSalesOrders
8	4180	HTTP	200	Metin	http://localhost:3801/salesorder/GetMaxSalesOrders
9	4179	HTTP	200	Metin	http://localhost:3801/salesorder/GetMaxSalesOrders
10	4179	HTTP	200	Metin	http://localhost:3801/salesorder/GetMaxSalesOrders
11	4178	HTTP	200	Metin	http://localhost:3801/salesorder/GetMaxSalesOrders
12	4155	HTTP	200	Metin	http://localhost:3801/salesorder/GetMaxSalesOrders
13	4165	HTTP	200	Metin	http://localhost:3801/salesorder/GetMaxSalesOrders
14	4205	HTTP	200	Metin	http://localhost:3801/salesorder/GetMaxSalesOrders
15	4204	HTTP	200	Metin	http://localhost:3801/salesorder/GetMaxSalesOrders
16	4203	HTTP	200	Metin	http://localhost:3801/salesorder/GetMaxSalesOrders
17	4204	HTTP	200	Metin	http://localhost:3801/salesorder/GetMaxSalesOrders
18	4205	HTTP	200	Metin	http://localhost:3801/salesorder/GetMaxSalesOrders
19	4212	HTTP	200	Metin	http://localhost:3801/salesorder/GetMaxSalesOrders
20	4248	HTTP	200	Metin	http://localhost:3801/salesorder/GetMaxSalesOrders
999	2404	HTTP	200	Metin	http://localhost:3801/salesorder/GetMaxSalesOrders
1000	2431	HTTP	200	Metin	http://localhost:3801/salesorder/GetMaxSalesOrders

6.4. Monolitik Özet Rapor

Jmeter uygulamasında özet rapor için istekler çalıştırıldığında aşağıdaki tablo ve sonuçlar elde edilir. 1000 istek üzerinden ortalama 2 saniyede dönüşler elde edilir. Monolitik uygulama performansı gayet yeterli olduğu görüldü. Ortaya çıkan performans izleme raporuna bakıldığında aşağıdaki sonuçlar elde edilmiştir (Tablo 2).

- **Etiket:** Bu sütun, ölçümün türünü veya kategorisini temsil eder. İlk sütun, "HTTP İsteği" ve "TOPLAM" olarak iki farklı etiket içeriyor. "HTTP İsteği" etiketi, belirli bir HTTP isteğinin performansını temsil ediyor olabilirken, "TOPLAM" etiketi tüm isteklerin toplam performansını gösteriyor olabilir.
- **Örnek:** Bu sütun, her bir etiket için kaç ölçüm yapıldığını gösterir. "HTTP İsteği" ve "TOPLAM" için her ikisinde de 1000 ölçüm yapıldığı görünüyor.
- **Ortalama:** Bu sütun, ölçülen değerlerin ortalamasını temsil eder. Örneğin, "HTTP İsteği" etiketi için ortalama değer 2'dir.
- **En Az:** Bu sütun, ölçülen değerler arasındaki en küçük değeri gösterir. "HTTP İsteği" etiketi için en düşük değer 0'dır.
- **En Çok:** Bu sütun, ölçülen değerler arasındaki en büyük değeri gösterir. "HTTP İsteği" etiketi için en yüksek değer 57'dir.
- **Hata %:** Bu sütun, ölçülen değerlerin ne kadarının hata içerdiğini gösterir. Tabloya göre, ölçülen değerlerin hiçbirinde hata olmadığı (%0 hata) görünüyor.
- **KB/sn (Kilobayt/saniye):** Bu sütun, belirli bir hızla veri transferini temsil eder. 1600.03 KB/sn, bir saniyede 1600.03 kilobayt veri transferi olduğunu gösterir.
- **Sent KB/sn (Gönderilen Kilobayt/saniye):** Bu sütun, ölçülen veri transfer hızının belirli bir gönderen tarafından sağlandığını gösterir. 41.37 KB/sn, belirli bir gönderen tarafından gönderilen verinin saniyede 41.37 kilobayt olduğunu gösterir.
- **Ortalama Byte:** Bu sütun, belirli bir işlem veya veri transferinin ortalama byte boyutunu gösterir. 4912, bu işlem veya veri transferinin ortalama olarak 4912 byte olduğunu gösterir.

Tablo 2 : Monolitik İstek Cevap Özet Raporu.

Etiket	Örnek	Ortalama	En Az	En Çok	Hata %	KB/sn	Sent KB/sec	Ort. Byte
HTTP İsteği	1000	3684	0	4589	0,00%	764,46	27,7	4057,3
TOPLAM	1000	3684	0	4589	0,00%	764,46	27,7	4057,3

7. MİKROSERVİS UYGULAMA ÖRNEĞİ

Uygulama mikroservis mimaris üzerine kurulmuştur. Uygulama toplamda 3 Api'den oluşmaktadır. "SalesApi", "ProductApi" ve 'GatewayApi'dir. Uygulamaların içerisinde "Dto", "Servis", "Controller" gibi nesneleri vardır. "SalesApi" ve "ProductApi" verileri çekmek için kullanılan "Api"lerdir. "GatewayApi" ise "Api"lerin arkasında olan istemcinin bileceği bir Api'dir. Bu Api üzerinden yönlendirmeler yapılır. Apilerin üzerinden SQL Server bağlanabilmek için "EntityFramework ORM"i kullanıldı. "ORM" araçları veritabanlarını nesneler halinde oluşturarak projelere dahil edebilmesini sağlar. Bu 3 Apiye ait solution (çözüm) aşağıda görünmektedir. "Microservice Gateway Api"si üzerinde herhangi bir işlem bulunmamaktadır. Bu Api den diğer Api'lere yönlendirme işlemi yapılmaktadır.

7.1. Sales Api

Projede "Aspnet Core" Web Api kütüphanesi kullanılmıştır. Projede "Controllers" klasörü bulunmaktadır. Onunda altındada "Salesordercontroller" bulunuyor. Gateway üzerinden yapılan istekler bu controllera gelip aynı şekilde sonuç döndürülmektedir. Bu controller içerisindeki "GetMaxSalesOrders" methodu çağırılır. "GetMaxSalesOrders" methodu "Adwenturework db" sine gidip "SalesOrderHeader", "SalesOrderDetail" tablosundaki verileri getirir.

Projede "Dto" klasörü bulunmaktadır. Bu klasör içerisinde "SalesOrderListDto" dosyası bulunuyor. Bu "Dto" nesnesini veritabanından gelen "entity" modelin istemciye gönderilirken belirlenen kullanıcıların bilmesi gereken özellikleri filtrelenip istemciye dönülmektedir.

Projede "Models" klasöründe veritabanı nesneleri bulunmaktadır. Bu nesneler "Adwenturework" veritabanına aittir. Bu projede "SalesOrderHeader", "SalesOrderDetail" tablosu kullanıldı. Daha sonra buradan gelen siparişe ait "Productid" ile beraber "Product" mikroservisine istekte bulunup ürüne ait isim bilgisi alındı. "Monolith" uygulamada hepsi aynı anda çekilmektedir ancak burada durum daha farklıdır. Veritabanını projeye ekleyebilmek için "entityframework" e aşağıdaki kodu konsolda çalıştırılması gerekiyor.

Çalıştırılacak Kod Bloğu: dotnet ef dbcontext scaffold "Server=.\;Database= AdventureWorks2019;Trusted_Connection=True;"Microsoft.EntityFrameworkCore.SqlServer -o Models

Kod bloğu çalıştıktan sonra veritabanı ve nesneler proje tarafından iletişim kurulabilir hale getirildi.

Servis klasörü altında bulunan "SalesOrderService" dosyasından veritabanında bulunan "SalesOrderHeader", "SalesOrderDetail" tablolarındaki en çok sipariş verilen 50 ürüne ait siparişler çekilir. "GetAll" metodu bütün satış kayıtlarını dönülüyor. Veritabanından dönen modelin tamamını değil de sadece "ProductID" ve "SalesOrderCount" alanlarını oluşturulan "SalesOrderDto" ya çevrilip geri dönülür. Veritabanına "AdwentureWorks2019Context" nesnesi üzerinden iletişim kurulur.

"SalesOrderController" sınıfında oluşturulan bu servisi çağrılıp istemciye cevap dönülür. "SalesOrderController" veritabanını bilmiyor. Tek görevi servis nesnesini çağırmak oluyor. N-Katmanlı mimarilerde bu klasörler az bağımlı olarak tasarlanır.

"SalesOrderApi'si" monolotik proje de "SalesOrder" sınıfına benziyor. Burada "SalesOrder" ve "Product'ı" mikroservis mimariyle beraber birbirinden ayrıldı. "SalesOrder" ve "Product"

geliştirmeleri birbirlerinden farklı bir dilde, bağımsız olarak ölçeklemede, sunucu yapılandırmalarında kullanılabilir hale getirildi. "GetMaxSalesOrders" methodu ilk önce satış bilgilerini alıyor sonrasında da "Product" mikroservisine gidip ürün bilgilerini çağırıyor. "SalesAPI" ye ait URL ve Port bilgisi; http://localhost:63761/

Gateway Api'sine "Salesorder" isteği geldiğinde gateway bunu http://localhost:63761/ adresine yönlendirecektir. İstemci direkt olarak bu Api'yi çağırmayacaktır. Gateway Api'sini çağıracaktır.

7.2. Product API

Proje de Aspnet Core Web Api kütüphanesi kullanılmıştır. Proje de Controllers klasörü bulunmaktadır. Onunda altındada productcontroller bulunuyor. Gateway üzerinden yapılan istekler bu controllera gelir. "SalesAPI" içerisinden bu mikroservis çağrılır ve ürün bilgilerini alıp istemciye gönderilir. Bu "controller" içerisindeki "GetByProductIds" methodu çağrıldı. "GetByProductIds" methodu "adwenturework db" sine gidip "product" tablosundaki verileri getirdi.

Proje de "Dto" klasörü bulunmaktadır. Bu klasör içerisinde "ProductDto" dosyası bulunuyor. Bu "Dto" nesnesini veritabanından gelen "entity" modelin istemciye gönderilirken belirlenen, kullanıcıların bilmesi gereken özellikleri filtreleyip istemciye dönüldü.

Proje de "Models" klasöründe veritabanı nesneleri bulunmaktadır. Bu nesneler "Adwenturework" veritabanına aittir. Bu projede "Product" tablosu kullanıldı. Veritabanını projeye ekleyebilmek için "entityframework" e aşağıdaki kod bloğu konsolda çalıştırıldı.

Çalıştırılacak Kod Bloğu: dotnet ef dbcontext scaffold "Server=.\;Database= AdventureWorks2019;Trusted_Connection=True;" Microsoft.EntityFrameworkCore.SqlServer -o Models

Kod bloğu çalıştıktan sonra , veritabanı ve nesneler proje tarafından iletişim kurulabilir hale getirildi. Servis klasörü altında bulunan "ProductService" dosyasından veritabanında bulunan "Product" tablosundaki bütün kayıtlar çekildi. Aşağıdaki resimde görüldüğü üzere "GetByProductIds" metodu bize bütün "Product" kayıtlarını dönüyor. Veritabanımıza "AdwentureWorks2019Context" nesnesi üzerinden iletişim kuruldu.

ProductController sınıfında oluşturulan bu servis çağrılıp istemciye cevap dönülür. ProductController veritabanını bilmiyor. Tek görevi servis nesnesini çağırmak oluyor. N-Katmanlı mimarilerde bu klasörler az bağımlı olarak tasarlanır. Ancak bu projede bu şekilde yapıldı.

ProductApi'si monolotik projesinde ki Product sınıfına benziyor. Product servisi o projeden tamamen ayrıldı. Product geliştirmesini artık farklı bir dilde, farklı bir teknoloji kullanılarak da yazılabilir. ProductApi ye ait URL ve Port bilgisi; http://localhost:27946/ , Gateway Api'sine product isteği geldiğinde gateway bunu http://localhost:27946/ adresine yönlendirecektir. İstemci direkt olarak bu Api'yi çağırmayacaktır. Gateway Api'sini çağıracaktır.

7.3. Gateway API

Gateway Api'sine istemciden gelen isteği ilgili mikroservise yönlendiriyor. Burada istenirse loglama, kimlik doğrulama, yetki kontrolü işleri yapılabilir. Bu Api'de controller "DTO" gibi nesneler bulunmuyor. İstenirse bu özelliklerde eklenip gelen giden datalarda manipülasyon yapılabilir. Bu Api'de "configuration.json" ve "launchSetting.json" dosyaları bulunmaktadır.

launchSetting.json: Bu dosyada apinin ayağa kalkarken hangi porttan ayağa kalkacağı bilgisini belirtilir.

Configuration.json: Bu dosyada Api'yi gateway olarak özelliklendirecek ocelot paketinin ayarları bulunuyor. Mikroservislerin hangi portlarda bulunduğu bilgisi ve gelen isteğin hangi durumlarda hangi mikroservise gideceği bilgisi tutuluyor.

Api'nin gateway olarak çalışabilmesi için "Ocelot" isimli paketi uygulamaya kuruldu. Sonrasında "configuration.json" dosyasında yönlendirme ayarlarını URL ve port bilgileri ile verildi. GatewayApi ye ait URL ve Port bilgisi http://localhost:65070/ 'dir.

Gateway Api'sine'e http://localhost:65070/salesorder/pmax diye bir istek gelirse http://localhost:63761/ salesapi mikroservisine yönlendirecektir.

Jmeter üzerinden test edilen adres gateway Api'sine ait olan http://localhost:65070/ Api'sidir.

7.4. Mikroservis Uygulama Jmeter ile Yük Testi

Mikroservis uygulamasının yük testi Jmeter ile yapılmıştır. Bu bölümde yük testi yapıldı. Geliştirilen uygulamalar canlı ortama alınmadan önce yük testi yapılması büyük önem arz etmektedir. Uygulamaların farklı istekler üzerinde nasıl tepki verdiği görüldü. Geliştirmeler bu sonuçlara göre iyileştirebilir, farklı yöntemler düşünebilir, farklı çözümler üretebilir. Yük testleri uygulamaların kalitesini ve kullanılabilirliğini artırmada yardımcı olurlar.

Yük testini yaparken 3 saniye içerisinde 1000 kullanıcıya çıkıldı ve 1000 kullanıcı istekte bulunduğunda monolitik uygulamasının nasıl tepki verdiğini görüldü. 1000 kullanıcı isteğini yerel bilgisayarda Jmeter uygulaması üzerinde gerçekleştirildi.

Mikroservis uygulaması jmeter gerekli konfigürasyon ayarlar yapıldı.

Http Bilgisi, Sunucu İsmi, Port Numarası, Http istek türü, Yol bilgilerini girip test ayarları girildi.

Mikroservis uygulamasında konfigürasyon ayarları yapıldıktan sonra rapor sonuçlarını görebilmesi için Jmeter'ın farklı özellikleri kullanıldı. Jmeter'a ait Özet rapor ve Sonuç tablosunu göster özelliklerini test parçacığının içine eklendi.

7.5. Sonuç Tablosu Rapor

Jmeter'ın 1000 kullanıcı ile testine ait 1000 istek Tablo3'te görüldüğü üzere başarıyla tamamlanmıştır. Mikroservis uygulasında 1000 request için tek bir servis çağrılıp tek bir dönüş olmuştur. Sonuç tablosunda bütün istekler başarılı bir şekilde görüldü. 3 saniye içerisinde 1000 kullanıcının istek atması sonucu elde edilen rapor bilgisine aşağıda yer verilmiştir. **Elapsed:** Dönüş süresi, **Label:** İstek Türü, **Responce Code:** Cevap Kodu, **Data Type:** Metin dili, **Url:** İstek atılan adres

Testin sonucunda 1000 istek ortalama 21,822 saniye sürmüştür.

Tablo 3: Mikrohizmet İstek Cevap Süreleri.

No	Süre	İstek	Cevap	Tip	URL
1	17724	HTTP	200	Metin	http://localhost:65070/salesorder/pmax
2	17726	HTTP	200	Metin	http://localhost:65070/salesorder/pmax
3	17726	HTTP	200	Metin	http://localhost:65070/salesorder/pmax
4	17700	HTTP	200	Metin	http://localhost:65070/salesorder/pmax
5	17750	HTTP	200	Metin	http://localhost:65070/salesorder/pmax
6	17776	HTTP	200	Metin	http://localhost:65070/salesorder/pmax
7	17616	HTTP	200	Metin	http://localhost:65070/salesorder/pmax
8	17775	HTTP	200	Metin	http://localhost:65070/salesorder/pmax
9	17658	HTTP	200	Metin	http://localhost:65070/salesorder/pmax
10	17654	HTTP	200	Metin	http://localhost:65070/salesorder/pmax
.					
.					
998	26725	HTTP	200	Metin	http://localhost:65070/salesorder/pmax
999	26050	HTTP	200	Metin	http://localhost:65070/salesorder/pmax
1000	25191	HTTP	200	Metin	http://localhost:65070/salesorder/pmax

7.6. Mikroservis Özet Rapor

Jmeter uygulamasında özet rapor için istekleri çalıştırıldığında aşağıdaki tabloyu ve sonuçlar elde edildi. 1000 istek üzerinden ortalama 21 saniyede dönüşler elde edildi. Monolitik uygulama performansı göre yetersiz göründü. Ortaya çıkan performans izleme raporuna baktığımızda aşağıdaki sonuçları elde ediyoruz (Tablo 4).

- **Etiket:** Bu sütun, ölçümün türünü veya kategorisini temsil eder. İlk sütun, "HTTP İsteği" ve "TOPLAM" olarak iki farklı etiket içeriyor. "HTTP İsteği" etiketi, belirli bir HTTP isteğinin performansını temsil ediyor olabilirken, "TOPLAM" etiketi tüm isteklerin toplam performansını gösteriyor olabilir.
- **Örnek:** Bu sütun, her bir etiket için kaç ölçüm yapıldığını gösterir. "HTTP İsteği" ve "TOPLAM" için her ikisinde de 1000 ölçüm yapıldığı görünüyor.
- **Ortalama:** Bu sütun, ölçülen değerlerin ortalamasını temsil eder. Örneğin, "HTTP İsteği" etiketi için ortalama değer 2'dir.
- **En Az:** Bu sütun, ölçülen değerler arasındaki en küçük değeri gösterir. "HTTP İsteği" etiketi için en düşük değer 0'dır.
- **En Çok:** Bu sütun, ölçülen değerler arasındaki en büyük değeri gösterir. "HTTP İsteği" etiketi için en yüksek değer 57'dir.

- **Hata %:** Bu sütun, ölçülen değerlerin ne kadarının hata içerdiğini gösterir. Tabloya göre, ölçülen değerlerin hiçbirinde hata olmadığı (%0 hata) görünüyor.
- **KB/sn (Kilobayt/saniye):** Bu sütun, belirli bir hızla veri transferini temsil eder. 1600.03 KB/sn, bir saniyede 1600.03 kilobayt veri transferi olduğunu gösterir.
- **Sent KB/sn (Gönderilen Kilobayt/saniye):** Bu sütun, ölçülen veri transfer hızının belirli bir gönderen tarafından sağlandığını gösterir. 41.37 KB/sn, belirli bir gönderen tarafından gönderilen verinin saniyede 41.37 kilobayt olduğunu gösterir.
- **Ortalama Byte:** Bu sütun, belirli bir işlem veya veri transferinin ortalama byte boyutunu gösterir. 4912, bu işlem veya veri transferinin ortalama olarak 4912 byte olduğunu gösterir.

Tablo 4 : Mikrohizmet İstek Cevap Özet Raporu.

Etiket	Örnek	Ortalama	En Az	En Çok	Hata %	KB/sn	Sent KB/sec	Ort. Byte
HTTP İsteği	1000	21822	0	27444	0,00%	142,01	4,7	4080
TOPLAM	1000	21822	0	27444	0,00%	142,01	4,7	4080

Kaynak: Yazar tarafından oluşturulmuştur.

8. SONUÇ

Monolotik mimari 3,684 saniyede isteğimize cevap verirken, mikroservis mimari 21,822 saniyede isteğe cevap dönmüştür. Bu sonuçlara Jmeter uygulaması ile otomatik istek yaparak varılmıştır. Test için atılan istekler 3 saniye içerisinde 1000 kullanıcıya çıkmaktadır. 3 saniye içerisinde 1000 istekte bulunup monolitik ve mikroservis mimarinin verdiği cevaplar sonucunda ölçümleler yapıldı.

Monolitik mimari bu örnekler ve test seneryolarına göre mikroservis mimariye göre daha hızlı çıkmaktadır. Bu örneklerde en çok satılan 50 ürün bilgisi elde edilmeye çalışıldı. Bu bilgiler için satış, sipariş ve ürün tablolarına ihtiyaç oldu. Satış,sipariş bilgileri için bir mikroservis, ürün için ayrı bir mikroservis yapıldı. Monolotik uygulamada ise hepsini aynı veritabanında ilişkilendirilerek çekildi.

Monolitik mimari ihtiyaç olan tablolarla direk olarak erişim sağlamasından dolayı daha hızlı cevap döndü. Mikroservis mimaride ise sipariş bilgileri bir servisten ürün bilgileri bir servisten ve bu servislerin önüne koyulan gateway olmasından kaynaklı performans kayıpları meydana gelmektedir. Her istek her cevap ayrı bir maliyettir. Sunucuya bir kere gidip verileri toplayıp gelmek daha performanslıdır.

Küçük ölçekli projelerde monolotik mimari kullanılması daha uygundur. Projelerin büyümesi ile monolitiğin yönetilmesi zorlaşıyor gibi görünsede monolitik mimari içinde çözümler vardır. Modüler monolitik buna örnek olabilir. Yüksek trafikli bir uygulama geliştirilecekse bu uygulamada mikroservis mimari tercih edilebilir. Mikroservis mimaride hizmetlerin bağımsız

olmaları onları ayrı ayrı ölçeklendirilmesine imkan sağlıyor. Aşırı trafik alan bir hizmetin sunucu kapasite, performans vb. özellikler artırılarak çözümler sağlanabilir.

Mikroservis mimaride çok fazla konfigürasyon ihtiyacı olması, izleme ve bakım maliyetleri oldukça fazla olduğundan mimariyi seçerken talebi, ihtiyacı iyi değerlendirip karar vermek gerekir.

9. KAYNAKÇA

Albin, S. T., Pautasso, C. (2008). The Art of Software Architecture: Design Methods and Techniques. Wiley.

Bass, L. , Clements, P. , Kazman, R. (2012). Software Architecture: Foundations, Theory, and Practice. Wiley.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., ve Stal, M. (1996). Pattern-Oriented Software Architecture: A System of Patterns. Wiley.

Clements, P., Bachmann, F., Bass, L., Garlan, D., ve Ivers, J. (2010). Documenting Software Architectures: Views and Beyond. Addison-Wesley Professional.

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., ve Safina, L. (2017). Microservices: Yesterday, today, and tomorrow. In Present and Ulterior Software Engineering (s. 195–216).

Farrow, R. (2012). Netflix Heads into the Clouds: Interview with Adrian Cockroft. In: ;login:, 37(1), 44-46. Retrieved from,https://www.usenix.org/system/files/login/articles/cockcroft_0.pdf

George, F. (2013, March 20). MicroService Architecture: A Personal Journey of Discovery [Slide presentation]. Retrieved February 9, 2020, from https://www.slideshare.net/fredgeorge/micro-service-architecure

Lewis, J. (2012, March). Microservices - Java, the Unix Way. Presentation at 33rd Degree Conference. Retrieved April 9, 2019, from http://2012.33degree.org/talk/show/67

Löwy, J. (2007). Programming WCF Services (1st ed.). O'Reilly Media.

Löwy, J. (Speaker). (2009, May). Every Class As a Service (Session SOA206) [Conference session]. Microsoft TechEd Conference. Archived from the original on 2010. Retrieved from http://channel9.msdn.com/ShowPost.aspx?PostID=349724

Rodgers, P. (2005). Service-Oriented Development on NetKernel: Patterns, Processes & Products to Reduce System Complexity. Web Services Edge 2005 East: CS-3 [Performance]. CloudComputingExpo 2005. SYS-CON TV.

Russell, P., Rodgers, P., ve Sellman, R. (2004). Architecture and Design of an XML Application Platform. HP Technical Reports, 62. Retrieved August 20, 2015.

Yerelbt (2022, July 4). *Monolitik, SOA ve Mikro Servis Mimari Nedir?.* 10 Mart 2024 tarihinde https://www.yerelbt.com/monolitik-soa-ve-mikro-servis-mimarileri-nelerdir/ adresinden edinilmiştir.