İstanbul Beykent Üniversitesi Fen ve Mühendislik Bilimleri Dergisi 2024, 17(2), 75-86
İstanbul Beykent University Journal of Science and Engineering 2024, 17(2), 75-86

Makale Türü: Araştırma Makalesi
Paper Type: Research Article

Geliş tarihi/Received: 24/10/2024
Kabul tarihi/Accepted: 13/12/2024

MİKROSERVİS TABANLI E-TİCARET UYGULAMALARINDA SİPARİŞ ODAKLI ENDPOİNT YÖNETİMİ

ORDER-FOCUSED ENDPOINT MANAGEMENT IN MICROSERVICE-BASED E-COMMERCE APPLICATIONS

DOI: 10.20854/bujse.1573265

Hilal Şen[1], Talat Firlar[2,*]

Öz

Günümüz teknolojisindeki hızlı gelişmeler, bireylerin gereksinimlerini ciddi oranda dönüştürmektedir. Teknolojinin bireyler üzerindeki etkisi baz alınarak dünyada her gün yeni gelişmeler yaşanmaktadır. Modern yazılım projelerinde ihtiyaçlara sürekli yenilerinin eklenmesi, ölçeklenebilirlik, hata izolasyonu, teknoloji çeşitliliği, esneklik gibi konularda sorunlara yol açmaktadır. Monolitik Mimari'den günden güne mikroservis mimariye geçiş kaçınılmaz hale gelmektedir. Projelerin büyümesi ile mikroservis mimarilerin kullanım avantajlarından daha fazla yararlanılır hale gelinmiştir. Bu nedenle, monolitik mimari yerine mikroservis mimarisine geçiş, projelerin geleceğe yönelik hale getirilmesine imkân sunmaktadır. Tüketicilerin kolayca ihtiyaçları olan ürünlere ulaşımını kolaylaştırmak amacıyla, hızlı kargo, temassız teslimat ve temassız ödeme gibi seçenekler hayatımıza dahil olmuştur. Bu kapsamda tüketicilerin siparişlerinin hataya yer vermeden işlenilmesi ve teslimatının sağlanması büyük önem arz etmektedir.

Bu makalede amaç, e-ticaret uygulamalarında sipariş adımında meydana gelen birtakım hataların önüne geçilerek, maliyet, sistem karmaşıklığı, performans kaybı, veriler arası tutarlılık gibi konuların iyileştirilmesi sağlanıp tüketicilerin ihtiyaçlarına daha iyi yanıt vermek, daha optimize edilmiş bir deneyim sunulması amaçlanmıştır.

Çalışma kapsamında, mikroservis mimari ile oluşturulmuş e-ticaret uygulamasında yapılan bir istek üzerine servislerden alınabilecek olan bir hata üzerine, gerçekleştirilmek istenen istek başarısız olduğunda, bu e-ticaret uygulaması tüketiciler tarafından kabul edilmeyebilir. Oluşan problem üzerinde hata mekanizmalarının uygulanabilirliği, uygulamaların nasıl işleneceği, mekanizmaları kullanmanın kazanımları, kullanılması ve kullanılmaması durumlarında yaşanacak etkenlerin analizlerine yer verilmiştir.

Anahtar Kelime: Mikroservis Mimari, E-ticaret, Hata Mekanizmaları, Sipariş Hataları

Abstract

The rapid developments in today's technology are seriously transforming the needs of individuals. Based on the impact of technology on individuals, new developments are experienced every day in the world. In modern software projects, the constant addition of new needs leads to problems such as scalability, error isolation, technology diversity, and flexibility. The transition from monolithic architecture to microservice architecture is becoming inevitable day by day. With the growth of projects, the advantages of using microservice architectures have become more utilizable. For this reason, the transition from monolithic architecture to microservice architecture allows projects to be made future-oriented. In order to facilitate consumers' easy access to the products they need, options such as fast shipping, contactless delivery, and contactless payment have become part of our lives. In this context, it is of great importance that consumers' orders are processed and delivered without any room for error.

The aim of this study is to prevent some errors that occur in the ordering step in e-commerce applications, to improve issues such as cost, system complexity, loss of performance, and consistency between data, and to provide a better response to consumers' needs and to provide a more optimized experience.

Within the scope of the study, when a request made in an e-commerce application created with microservice architecture is received from the services, and the requested request fails, this e-commerce application may not be accepted by the consumers. The applicability of error mechanisms on the problem that occurs, how the applications will be processed, the benefits of using the mechanisms, the factors that will be experienced in case of use and non-use are analyzed.

Keywords: Microservice Architecture, E-commerce, Error Mechanisms, Order Errors

[1] İstanbul Beykent Üniversitesi, Lisansüstü Eğitim Enstitüsü, Bilgisayar Mühendisliği Mezunu, hilallaskin@gmail.com, orcid.org/0009-0008-0845-6981

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

1. GİRİŞ

Bu makalenin amacı, e-ticaret uygulamalarında sipariş aşamasında meydana gelen hataları önleyerek maliyet, sistem karmaşıklığı, performans kaybı ve veri tutarlılığı gibi konuları iyileştirip, tüketicilere daha optimize edilmiş bir deneyim sunmaktır. Mikroservis mimarisi ile oluşturulmuş e-ticaret uygulamasında, bir istek sırasında herhangi bir serviste oluşan hata, tüm isteği başarısız kılmaktadır ki bu, tüketiciler açısından kabul edilebilir bir durum değildir. Polly kütüphanesi ile mikroservisler arası iletişimde oluşan hatalar izlenmekte olup, kütüphanenin sunmuş olduğu hizmetleri kullanarak mikroservis tabanlı e-ticaret projelerinin daha tüketici dostu olmasına katkı sağlanmaktadır. Polly, geçici hataları ele almanıza ve uygulamalarınızın dayanıklılığını artırmanıza yardımcı olan güçlü bir .NET kütüphanesidir. Polly ile, arızaları ve yavaşlamaları akıcı ve iş parçacığı güvenli bir şekilde ele almak için Yeniden Deneme, Devre Kesici, Koruma, Zaman Aşımı, Hız Sınırlayıcı ve Geri Dönüş gibi stratejileri kolayca tanımlayabilir ve uygulayabilir.

Mikroservis tabanlı e-ticaret uygulamalarında, sipariş yönetim süreçlerinin daha verimli hale getirilmesi ve hataların minimum seviyeye indirilmesi üzerine yapılmış bir çalışmadır. Bu işlem sonucu mikroservis mimari yaklaşımını benimseyerek, e-ticaret projeleri geliştirilen firmaların, hata yönetimi için oluşturulmuş olan, kütüphane kullanımıyla elde edebilecekleri kazanımlar ele alınmıştır.

Bu makale kapsamında, hata mekanizmalarının uygulanabilirliği, uygulamaların nasıl işleneceği, bu mekanizmaların kullanılmasının kazandıracağı avantajlar ve kullanılmaması durumunda yaşanabilecek etkiler analiz edilmiştir. Hata yönetim kütüphanesi Polly'nin getirdiği çözümlerin kullanılması ile elde edilen kazanımlara da odaklanmaktadır. Böylelikle, günümüzde kullanımı giderek yaygınlaşan e-ticaret uygulamalarının, daha kullanılabilir sistemler olarak geliştirilebileceği elde edilmektedir. E-ticaret uygulamasının sipariş yönetimi modülünde Polly kütüphanesi içerisinde yer alan senaryoların uygulanması veya uygulanmaması durumu ele alınmıştır. Ayrıca, uygulandığı durumda sağladığı faydalar da analiz edilmektedir.

2. LİTERATÜRDE YER ALAN BENZER ÇALIŞMALAR

Dervişi (2022), bankacılık uygulamalarında monolitik mimariden mikroservis mimarisine geçişin zorlukları ve çözümleri incelenmiş, mikroservislerin ölçeklenebilirlik ve RAM tüketimi açısından avantajları vurgulanmıştır. Ayrıca, AWS'nin sunucusuz dağıtım modeli ve mikroservis mimarisinin maliyet etkinliği değerlendirilmiş, hata toleransı ve güvenlik gibi avantajlarına dikkat çekilmiştir. Monolitik ve mikroservis mimarileri kıyaslanarak, mikroservislerin bankacılık sistemlerindeki üstünlükleri grafiklerle sunulmuştur.

Altınkaya (2022), üniversite bilgi sistemleri için RESTful servisler kullanılarak sistem erişimi hızlandırılmış ve SOA ile REST mimarileri incelenmiştir. Proje, platform bağımsız iletişim, kullanıcı dostu ve güvenli yapı hedefleriyle yapılandırılmış, aylık yanıt süreleri analiz edilmiştir. REST tabanlı mimari, sistemin hızlı çalışmasını sağlayarak, kâğıt israfını önlemiş ve detaylı kazanımlar sunmuştur.

Gördesli ve Varol (2022), e-ticaret firmalarının mikroservis mimarisini hata toleransı ve ölçeklenebilirlik nedeniyle tercih ettikleri, iletişim tekniklerinin performans analizi yapılmıştır. Paralel yürütme ile sipariş sürecinin en hızlı şekilde tamamlandığı, ancak hata kontrolünün zor olduğu belirlenmiştir. Mimarinin seçiminde tüketici ihtiyaçlarının belirleyici olduğu vurgulanmıştır.

Zhou ve arkadaşlarının (2018), mikroservis mimarisinin endüstriyel alanda kullanımında ortaya çıkan hataların, analizi ve geliştiricilere yaşattığı zorluklar ele alınmıştır. Mikroservis tasarımı ve dağıtımı dışında, hata yönetimi üzerine az çalışma olduğundan, 46 mühendisle yapılan anket sonucu 22 hata durumu belirlenmiş ve TrainTicket projesinde bu hatalar görselleştirilmiştir. Hatalar, fonksiyonel ve işlevsel olmayan gibi kategorilere ayrılmış, REST istemcisi ve Docker konteyner teknolojilerinin kullanım kolaylıkları vurgulanmıştır. Hata tespit ve çözüm süreçleri 7 adımda açıklanmış, hata görselleştirme aracı ile analizler yapılmıştır. Çalışma, büyük endüstriyel uygulamalarda mikroservis mimarisinin tercih sebeplerini ve hata yönetiminin önemini detaylı şekilde incelemiştir.

Wang, Zhang, Xu ve Gu'nun (2020), mikroservis mimarisinde yazılmış projelerde yaşanan hataları, otomatik olarak tespit etmek için istatistiksel bir yaklaşım geliştirilmiştir. Çalışma, akıllı hata teşhisi, çevrimiçi hata teşhisi ve iş akışına duyarlı hata teşhisi olmak üzere üç kategoriye odaklanmıştır. Akıllı hata teşhisi, manuel müdahaleye gerek kalmadan otomatik tespit; çevrimiçi hata teşhisi, mikroservis güncellemelerinin ardından hataları dinamik olarak belirleme; iş akışına duyarlı hata teşhisi ise, mikroservislerin iş akışındaki farklılıkları tespit etme üzerine yoğunlaşmaktadır. Çalışmada, mikroservisler arası iletişim ağaç yapılarıyla görselleştirilmiş ve benzer içerikler kümelenmiştir. SockShop, PiggyMetrics ve TrainTicket projeleri üzerinde yapılan değerlendirmeler, ağ paket kayıpları, veritabanı yapılandırma hataları ve yanıt süresi gecikmeleri gibi performans sorunlarına odaklanmıştır. PerfCompass, PivotTracing ve LogAnalytics yöntemleriyle doğrulanan bu yaklaşım, iş akışı ve performans hatalarını başarılı bir şekilde tespit ederken, işletim sistemi hataları için aynı başarıyı göstermemiştir.

Asrowardi, Putra ve Subyantoro (2020), e-ticaret uygulamalarında, mikroservis mimarisinin performans avantajları incelenmiş ve sipariş oluşturma süreci üzerinden performans testi yapılmıştır. 300, 500 ve 1000 tekrar ile 22.000 veri üzerinde yapılan testler, mikroservis mimarisinin monolitik mimariye göre daha yüksek performans sağladığını göstermiştir. Çalışmada, RESTAPI gibi dil bağımsız web hizmetlerinin ve Nesne Yönelimli Programlama (OOP) kullanımının, geliştirme sürecini kolaylaştırdığı ve müşteri memnuniyetini artırdığı vurgulanmıştır.

Kocaman (2018) tarafından yapılan çalışmada, mikroservis mimarisi kullanılarak geliştirilen ödeme sistemi, popülerlik kazanmaya odaklanan oyun uygulamaları için tasarlanmıştır. Bu sistem, oyuncuların oyun içi ödemelerini hızlı ve güvenli bir şekilde gerçekleştirmelerini sağlar. MSTOS projesi, kullanıcıların kart bilgilerini güvenle saklayarak, farklı oyunlara ödemelerini tek bir platform üzerinden yapmalarını sağlar. Oyuncular ayrıca hesaplarında kalan bakiyeyi istedikleri takdirde, başka oyunculara aktarabilirler. Mikroservislerin ölçeklenebilirliği test edilmiş ve uygulamanın başarılı olduğu gösterilmiştir. Arayüzde HTML ve yazılım tarafında PHP kullanılmıştır. Uygulama, tek kullanımlık şifrelerle güvenli bir giriş sağlar ve jeton satın alma işlemleri gibi özellikleri içerir.

Hasselbring ve Steinacker (2017), otto.de e-ticaret sitesinin monolitik mimariden mikroservis mimarisine geçişini inceler. Mikroservis mimarisi sayesinde site, daha güvenilir, ölçeklenebilir ve hata toleransı yüksek hale gelirken, geliştirme süreçleri servis bazında ekiplere bölünerek iyileştirilmiştir. Mikroservis tabanlı mimarinin karmaşık olabileceği ancak tutarlılığı sürdürmek, alarm mekanizmaları ve hata toleransı gibi önemli konular için geniş bir operasyon ekibi gerektirdiği vurgulanır. Ölçeklendirme ve hata tespiti konusunda başarı elde edilmiş ve geliştirme sürelerinde azalma yaşanmıştır.

3. YAZILIM MİMARİLERİ

Yazılım mimarisi, bir projenin unsurlarının düzenlenmesini ve ilişkilerini açıklar, performans, güvenlik ve uzun vadeli bakım gibi kritik konuları ele alır. Projelerin teknolojik değişimlere uyum sağlayabilmesi için esnek bir yapı oluşturur ve ekibin katkılarıyla sürekli olarak geliştirilir. Yazılım mimarları, projenin ilerleyişiyle birlikte ortaya çıkan gereksinimlere göre yazılımın ihtiyaç duyduğu teknolojilerin seçiminde karar verir. Mimarinin oluşturulması sürecinde, ekip içi iletişim ve yetkinlik seviyesi önemlidir. Uygun tasarlanmamış olan yazılım mimarisi, projeye zarar verebilir ve uygulamalarda performans sorunlarına neden olabilir. Mimari, projenin karmaşıklığını yönetmek, güvenlik sağlamak ve teknolojik değişime ayak uydurmak için kritik bir öneme sahiptir.

Monolitik mimari, yazılım geliştiricilerin uzun yıllar kullandığı bir yaklaşımdır. Bu yaklaşım, farklı bileşenleri tek bir programa birleştirir ve uygulamayı tek bir bütün olarak dağıtır (Gos ve Zabierowski, 2020). Monolitik mimaride, üç temel katman bulunur: veritabanı, iş mantığı ve kullanıcı arayüzü (Kyryk vd., 2022). Bu katmanlar, uygulamanın veri saklama, iş mantığını yürütme ve kullanıcı arayüzü sunma işlevlerini yerine getirir. Monolitik mimari, anlaşılabilirliği ve kolay geliştirilebilirliği ile tercih edilirken, zamanla bakımı zorlaşabilir ve sistemler karmaşık hale gelebilir. Bu durumda, yazılım geliştiricilerin uzun saatler harcaması ve yeniden mimari oluşturması gerekebilir. Monolitik mimarinin dezavantajları, sistemdeki bağımlılıkların artması ve modülleri bağımsız olarak çalıştıramama gibi sorunlardır. Bu nedenle, projenin büyümesi ve geliştirici sayısının artmasıyla birlikte, monolitik mimarinin zorlukları da artabilir (Terra, Valente, Bigonha, 2012).

Monolotik ve Mikroservis mimarisinin tablo 1'deki karşılaştırmasına bakıldığında, yazılım geliştiricilerin küçük ve bağımsız hizmetler olarak uygulamaları bölerek yönetmelerini sağlayan bir yaklaşımdır. Bu mimari, her servisin belirli bir işlevselliği temsil ettiği ve bağımsız olarak geliştirilip dağıtılabildiği bir yapı sunar. Servisler arasındaki iletişim HTTP, RPC, RESTful API'ler gibi protokollerle sağlanırken, her bir servis kendi veritabanına sahip olabilir ve kendine özgü teknolojilerle çalışabilir. Mikroservis mimarisi, esneklik, ölçeklenebilirlik ve bakım kolaylığı gibi avantajlar sunarken, her servisin işlevselliğine dikkat edilmesi ve bağımlılıkların yönetilmesi önemlidir. Veri bütünlüğünün sağlanması için tutarlılık mekanizmaları kullanılabilir (Dragoni vd., 2017).

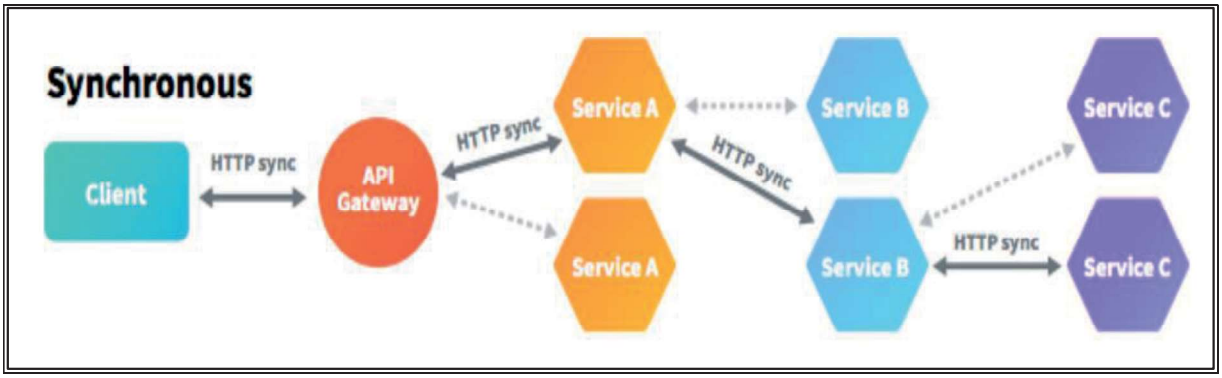
Tablo 1: Monolitik ve Mikroservis Mimari Karşılaştırması

Özellik	Monolotik	Mikroservis Mimari
Mimari Yapı	Tek bir büyük uygulama	Birden çok küçük uygulama
Esneklik	Daha az esneklik, değişiklikler genellikle zordur	Daha fazla esneklik, servisler bağımsız olduğu için değişiklikler daha kolaydır
DevOps Entegrasyonu	DevOps süreçleri daha kısıtlı olabilir	DevOps süreçleri daha esnek ve çeşitli olabilir

Bağımlılıklar	Yüksek bağımlılıklar ve entegrasyonlar	Düşük bağımlılıklar ve bağımsız servisler
Geliştirme Kolaylığı	Başlangıçta daha hızlı geliştirme	Her servisin bağımsız geliştirilmesi gerekebilir
Ölçeklenebilirlik	Dikey ölçeklenebilirlik	Yatay ölçeklenebilirlik
Bakım	Daha karmaşık bakım gereksinimleri	Daha az karmaşık bakım gereksinimleri
Teknoloji Seçimi	Teknoloji seçimi genellikle kısıtlıdır	Teknoloji seçimi genellikle daha esnektir
Dağıtım	Tüm uygulama tek bir yapıda olduğu için daha kolaydır	Servislerin dağıtımı karmaşıktır
Hata Yönetimi	Bir hata tüm sistem üzerinde etkili olabilir	Servisler birbirinden bağımsız olduğu için hata yönetimi daha kolay olabilir

Mikroservis Mimarisinde İletişim; Mikroservis mimarisi, büyük ve karmaşık uygulamaları esnek, ölçeklenebilir ve yönetimi kolay parçalara ayırarak modern teknolojilere uyum sağlar. Servisler arası iletişim, güvenlik, performans ve dayanıklılık açısından kritik olup, RESTful API'ler, gRPC, senkron ve asenkron sistemler gibi yöntemlerle sağlanır ve sürekli iyileştirilir.

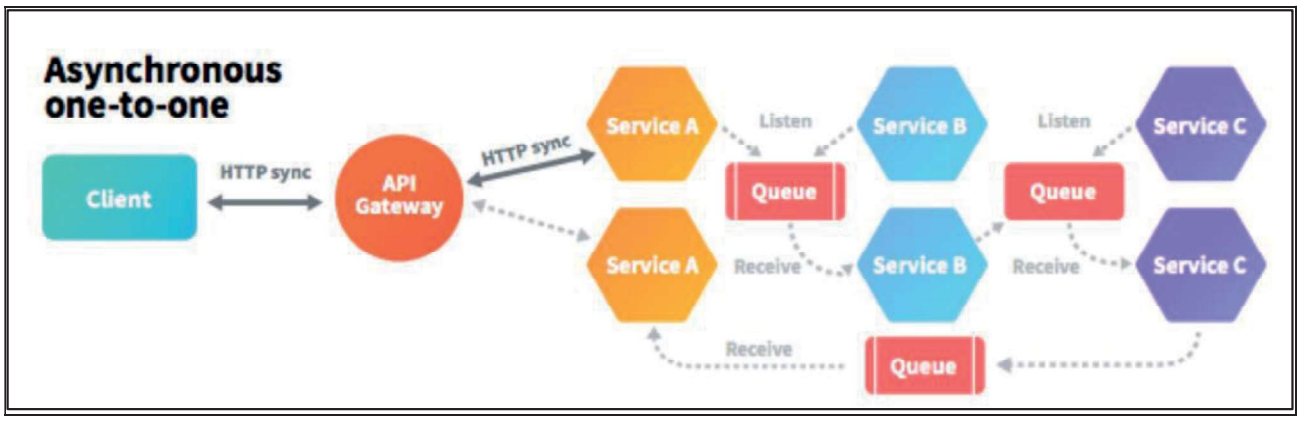
Mikroservis mimarisinde şekil 1 'de görülen senkron iletişim, istek/yanıt etkileşimi olarak sıkça tercih edilir ve bir servisin diğerine istekte bulunup yanıt alana kadar beklemesi şeklinde gerçekleşir. Monolitik uygulamalarda tek yapı üzerinden iletişim sağlanırken, mikroservisler bağımsız çalışır ve birbirlerinden veri bekler. Bu durum, dönüş sürelerinde gecikmelere ve sistemsel bloklamalara yol açabilir, maliyet ve kullanıcı kaybına neden olabilir. Örneğin, bir e-ticaret uygulamasında, senkron iletişim müşteri geri dönüşlerinin hızlı olması, stok ve fiyat güncellemelerinin anında yapılabilmesi ve ödeme işlemlerinin bankalardan anında yanıt alması gerektiği için tercih edilir. Bu nedenle, senkron iletişim türü dikkatli seçilmelidir (Shafabakhsh, Lagerström, Hacks, 2020).



Şekil 1. Senkron iletişim şeması (Shafabakhsh, Lagerström, Hacks, 2020)

Mikroservis mimarisinde asenkron iletişim, mesajı gönderen taraf geri dönüş beklemez. Bu tür iletişim, özellikle uzun sürecek işlemler sırasında kullanışlıdır; müşteri isteği gönderilir ve sunucu işlemi gerçekleştirirken müşteri başka işlemlerle meşgul olabilir. Asenkron iletişim, bağımlılıkları ortadan kaldırarak servislerin birbirinden bağımsız çalışmasını sağlar. Bu tür iletişimde, zaman farkına bakılmaksızın veri alışverişi yapılır ve hatalar sistemin kesintiye uğramasına neden olmaz. Kuyruk yapısı sayesinde mesajlar saklanabilir ve hata giderildiğinde tekrar gönderilebilir. Bu, mikroservis mimarisinde önemli bir avantajdır.

Asenkron iletişim hem bire bir hem de bire çok iletişim gerçekleştirilmesine olanak tanır. Kullanıcı bir bildirim gönderdiğinde, olay yayınlanır ve diğer işlemler paralel olarak devam edebilir. Örneğin, bir ürün sipariş edildiğinde, siparişin kargoya verilmesi ve stoktan düşülmesi eş zamanlı olarak yapılabilir. Asenkron iletişim, kaynakların ekonomik kullanımını sağlar ve yoğun zamanlarda değil, kullanımın az olduğu zamanlarda işlemleri gerçekleştirerek optimizasyon sağlar. RabbitMQ ve Kafka gibi sistemler, asenkron mesajlaşma teknolojileri olarak yaygın şekilde kullanılmaktadır (Van Steen ve Tanenbaum, 2017).



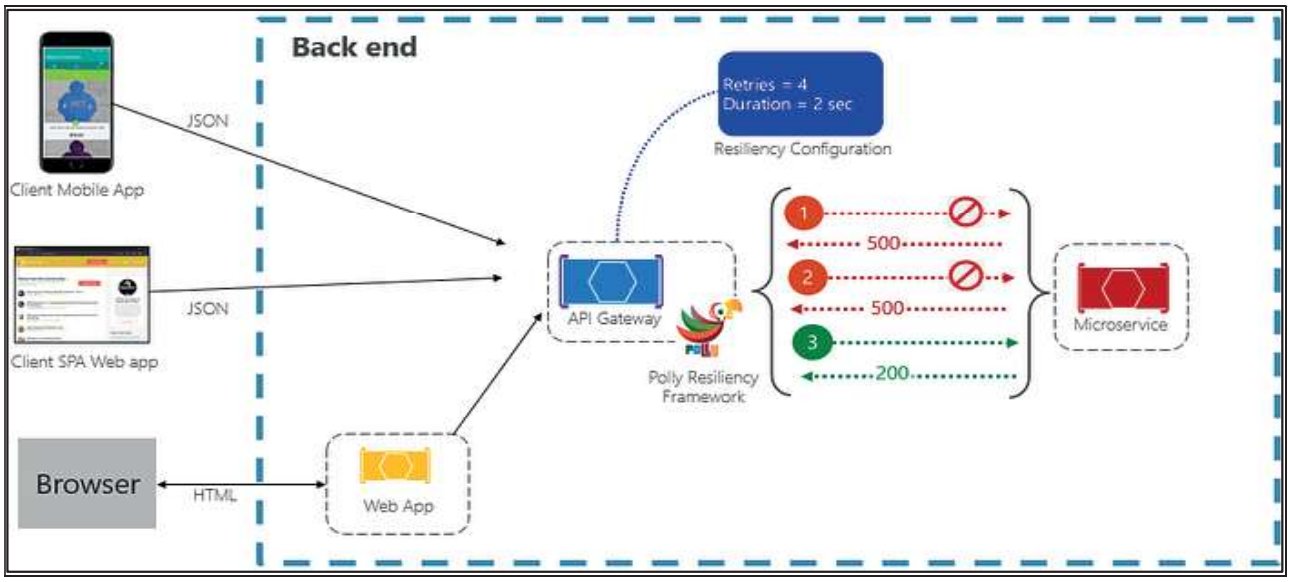
Şekil 2. Asenkron iletişim (Van Steen ve Tanenbaum, 2017).

Mikroservis Mimarisinde Hata Yönetimi; Mikroservis mimarisi, büyük projeleri küçük parçalara ayırarak yönetim avantajı sağlar; ancak bu süreçte hata yönetimi kritik bir rol oynar. Mikroservislerde hata yönetimi, hataların tespiti, kaydedilmesi, izlenmesi ve giderilmesini kapsar. Hatalar yazılım, veri, kullanıcı, altyapı ve işlemsel kategorilere ayrılır. Servisler arasındaki hata tolerans stratejileri, bir servisin diğerlerinden etkilenmeden çalışmasına olanak tanır. Otomatik yeniden yükleme ve tekrar deneme gibi teknikler kullanılarak, servis hatalarının yönetimi sağlanır. Ayrıca, kritik durumlarda alternatif çözümler sunularak, hata toleransı sağlanır ve kullanıcı hatalarına karşı da önlemler alınır.

Hata yönetiminde loglama ve izleme araçları (Elasticsearch, Logstash, Kibana, Splunk, Fluentd) kullanılarak hataların anlamlandırılması ve analiz edilmesi sağlanır. Monolitik mimariden mikroservis mimariye geçişte, dağıtık bileşenlerin karmaşıklığını yönetmek için Prometheus, Elastic APM ve Grafana gibi araçlar kullanılır. Servislerin birbirinden izole edilmesi, hata izolasyonu ve kullanıcı deneyimini olumlu yönde etkiler. Sistem kesintilerinde, hatalı servisler devre dışı bırakılır ve alternatif çözümler sunulur. Geri alma stratejileri (A/B testi, Gradual Rollout, Blue/Green Deployment) kullanılarak sistem güncellemeleri ve hata yönetimi yapılır. Mikroservis mimarisinin artan kullanımıyla birlikte otomatik hata yönetimi ve sürekli iyileştirme çalışmaları da artmaktadır.

Hata Yönetim Kütüphanesi; Mikroservis mimarisinin yaygınlaşmasıyla birlikte anlık hata müdahalesi önemli hale gelmiştir. Polly, NET tabanlı uygulamalarda hata yönetimi, retry (yeniden deneme) ve circuit breaker (devre kesici) gibi dayanıklılık stratejilerini uygulamak için açık kaynak ve yaygın olarak kullanılan bir kütüphanedir. Polly, bir .NET dayanıklılık ve hata yönetim kütüphanesidir. API istekleri sırasında karşılaşılan başarısızlıkları tekrar denemek, hataları yönetmek, gecikmelerle (backoff) tekrar denemek ya da devre kesici (circuit breaker) gibi stratejiler kullanarak uygulamalarımızın dayanıklılığını artırmak için kullanılır. Bu kütüphane, bir çok karmaşık hatayı basit bir şekilde yönetmemize yardımcı olur. Uygulama geliştirirken genellikle dış bir servisin bir anda yanıt vermemesi ya da gecikmesi durumunda uygulamamız kilitlenebilir,Polly burada devreye girer, belirli bir hatayı ya da istisnai durumu tekrar deneme, belirli bir süre bekleyip tekrar deneme ya da sistemde problem olduğunda devre kesici kullanarak bu problemi yönetme gibi seçenekler sunar. Kısacası Polly, bağlantı sorunlarında otomatik yeniden deneme gibi politikalarla kesintileri önler, güvenlik açıklarına karşı dayanıklılığı artırır ve aşırı yüklenme saldırılarına karşı sistemi korur.

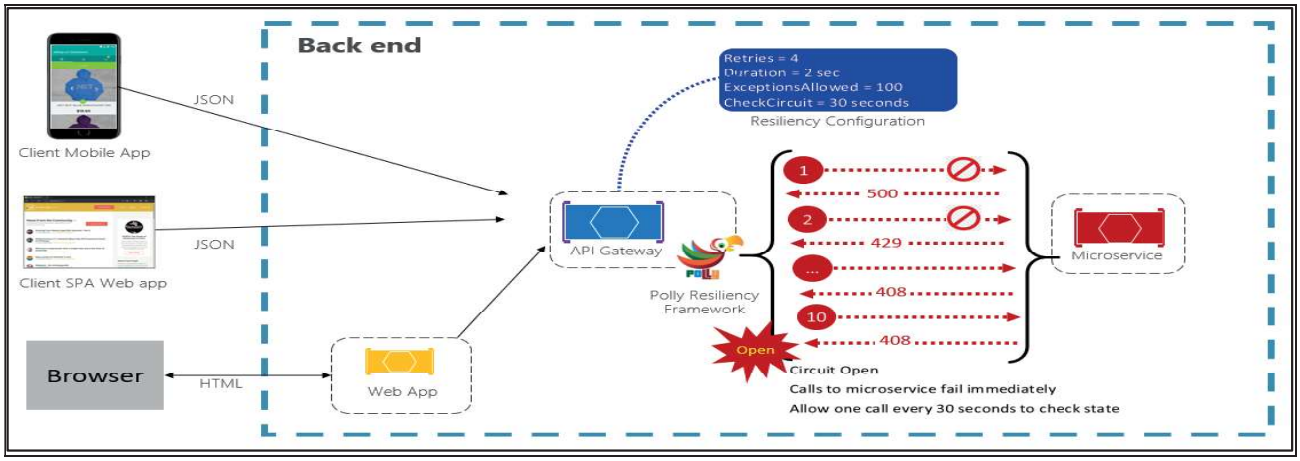
Retry Pattern; Retry (tekrar deneme) yöntemi, yapılan isteklerin sonucunun hatalı dönmesi durumunda tekrar denemeye imkân sağlar. Bu yöntem, istek gönderdiğimiz ve sonucunu beklediğimiz servisin küçük kesintiler yaşaması durumunda otomatik olarak düzeltilebilmesi için kullanılır. Belirli aralıklarla ardışık işlem yapılabilir. Retry pattern kullanımı, tek bir hata, koşullu hata veya birden fazla hata tipine göre uygulamanın çalışmasını sağlar. Başarılı bir sonuç alıncaya kadar tekrarlama yapılabilir veya belirli bir bekleme süresi konularak işlem başlatıldıktan sonra tekrar denemek için bir aralık bırakılabilir.



Şekil 3. Retry Pattern Şeması (Microsoft, 2024).

Circuit Breaker Pattern; Circuit Breaker (Devre kesici) kullanımı, hatalı işlemlerin tekrar denenmesi durumunda döngüye girip belirsiz bir şekilde devam etmesini engeller. Bu pattern, belirli bir süre bekledikten sonra sorguyu durdurarak işlev görür. Belirli hata tiplerinde, hatanın durumunda değişiklik olma ihtimaline karşı alınabilecek aksiyonlar belirlenebilir. Devre kesici kapalıyken tüm isteklerin geçmesine izin verir. İstekler başarısız olduğunda, belirlenen süre boyunca istekleri iletmemek için kendini açık konuma getirir. Daha sonra, tekrar deneme yaparak başarılı olursa kendini kapalı konuma geri getirir. Yarı açık durumda, isteklerin

dinlenmesi ve senaryoya göre durumun belirlenmesi sağlanır. Bu yöntem, zaman aşımlarını önler ve sunucunun aşırı yüklenmesini engeller.



Şekil *4*. Circuit Breaker Şeması (Microsoft, 2024).

Fallback Pattern; Fallback (Yedek plan) pattern kullanımı, hata alınması durumunda tekrar deneme politikalarından farklı olarak başka bir servise yönlendirilmesini sağlar. Farklı bir servisin tetiklenmesiyle karşılaşılan hataya çözüm sunulur. Bir servisten gelen hata durumunun başka bir serviste kontrol edilmesiyle başarısızlığın önüne geçilir. Bu yöntem, kullanıcı deneyimini iyileştirir ve uygulama kullanımını artırır. Örneğin, e-ticaret uygulamasında araçların satışının yapıldığı bir platformda, kullanıcıların ürünleri tanıtmak için fotoğraf yüklemesi zorunludur. Fotoğraf yükleme servisinin çalışmaması durumunda, sistem otomatik olarak boş bir resim yerleştirerek bu sorunu çözer.

Timeout Pattern; Timeout (Zaman aşımı) pattern, bir uygulamada istek gönderilen servisin zaman aşımına uğraması durumunda uygulanabilecek stratejileri içerir. Bu stratejiler optimistik ve pesimistik olarak ikiye ayrılır. Eşzamanlı işlemleri iptal etmek veya sonlandırmak için kullanılan 'CancellationToken', optimistik durumda asenkron işlemlerde kullanılan bir timeout stratejisidir. Pesimistik durumda ise CancellationToken olmadan senkron işlemlerde timeout durumunu yönetir. Yapılan isteklerin süresi dolduğunda hata göstererek servis sonucunu beklemeden sistemin devam etmesini sağlar.

Polly kütüphanesi, kullandığı patternler ile uygulamaların güvenliğini, sağlamlığını ve hata toleransını artırır. Beklenmedik hatalara uygun çözümler sunarak bu durumlara yanıt verir. Kod tekrarını azaltarak karışıklığı önler ve hızlı çözümler sunar. Bu nedenle, gerekli uygulamalarda hata yönetimi için tercih edilmelidir. Polly, yazılım geliştiricilere kod kullanışlılığı sağlayarak uygulamanın bakımını ve güncelleştirilmesini kolaylaştırır.

4. UYGULAMA

Uygulama, mikroservis mimarisi üzerine inşa edilmiştir, Sipariş ve Müşteri olmak üzere iki API'den oluşmaktadır. Sipariş API'si, mevcut müşterileri kontrol ederek sipariş oluşturulmasını sağlar ve olası problemler için çeşitli yöntemler sunar. Müşteri API'si ise müşteri bilgilerini yönetir, bu bilgileri kullanarak siparişlerin oluşturulmasına olanak tanır. Polly kütüphanesi, bu bağımsız API'lerde meydana gelen hataları yöneterek uygulamanın dayanıklılığını artırır.

Müşteri API'si, müşteri bilgilerini içeren bir sınıf oluşturur ve 6 farklı kullanıcı örneğiyle çalışır. HTTP GET isteğiyle müşteri numarası gönderilerek Swagger arayüzünde ilgili müşteri ismi

gösterilir. Swagger, API'lerin özelliklerini görüntüleme, test etme ve geri dönüş cevaplarını alma imkânı sunar. Hatasız çalıştığında, "/api/MusteriBilgileri/GetMusteriIsmi/{musteriNumarasi}" endpointine varsayılan müşteri numarası gönderilerek müşteri bilgileri alınır. Hata yönetimi, HTTP durum kodlarından alınan geri dönüşlerle uygulamanın problemleri ifade etmesini sağlar, böylece uygulamanın dayanıklılığı artar ve sorunlar hızlıca çözülebilir.

Sipariş API'si, siparişi veren müşterinin bilgilerini ve sipariş tarihini içeren verileri bir modelde barındırır. Bir müşteri birden fazla sipariş verebilir ve her sipariş eşsiz bir sipariş numarasıyla tutulur. Siparişler, ürünlerin id numarası ve isim bilgileriyle "içindekiler" sınıfında yer alır. E-ticaret uygulamalarında stokta olmayan ürünlerin siparişi, fiyat değişiklikleri, yetersiz bakiye, kargo hataları ve yanlış teslimat adresleri gibi senaryolar yaşanabilir. Her ürün kendi sipariş id'si ile değerlendirilir ve sipariş işlemleri müşteri memnuniyetini doğrudan etkiler. Ürün açıklamaları, resimleri ve müşteri yorumları, uygulamaya olan güveni artırır.

E-ticaret uygulamalarında hızlı ve etkili müdahale için hata izleme kütüphanelerinden faydalanmak önemlidir. Bu makalede, Serilog.AspNetCore kütüphanesi kullanılarak hataların izlenmesi, güvenlik açıklarının tespiti, hata takibi ve uygulama performansının iyileştirilmesi sağlanmaktadır.

Uygulamaya kütüphanenin eklenmesi ve appsetting.json dosyasının yapılandırılması gerekmektedir. Serilog.AspNetCore kütüphanesi eklenirken, test aşamasında "GetMusterininSiparisleri/{musteriNumarasi}" endpointi çalıştırılarak müşteriye ait siparişlerin, listelendiği kontrol edilir. Ancak, müşteri API'sinden geri dönüş alınamadığında HTTP durum kodu 500 alınır. Bu durumda bağlantı hatası veya ağ sorunları yaşanmış olabilir. Kimlik doğrulama veya yetkilendirme yöntemleri kullanılıyorsa, bu sorunlar da dikkate alınmalıdır. Serilog.AspNetCore kütüphanesi sayesinde yaşanan hatalar izlenir ve uygun alternatif stratejileri uygulanır. Bu şekilde geliştiriciler, sipariş sürecini etkili bir şekilde yönetebilirler.

Uygulama İçerisinde Hata Yönetimi ve İyileştirme Stratejileri;

Uygulama içinde Polly kütüphanesi entegre edilerek, SiparisAPI'nin MüsteriAPI'den dönüş değeri alamadığı durumlarda dayanıklılığı artırılmıştır. Retry Policy kullanılarak, istek olumsuz bir sonuç döndürdüğünde tekrar deneme işlemi yapılmıştır. Ancak, bu tekrar deneme işlemi sonsuz bir döngüye girme riskini önlemek için proje içinde 3 kere deneme yapılması prensibi benimsenmiştir.

SiparisAPI, MüsteriAPI'den olumlu yanıt alamadığı durumda deneme sayısının tamamlanmasıyla isteği olumsuz olarak sonlandırmaktadır. MüsteriAPI'ye yapılan istekte rastgele bir hata durumunun belirlenmesi için üretilen sayıların çift veya tek oluşuna göre 500 hatası veya olumlu bir yanıt döndürülmektedir. Bu yöntemle hataların denemesi gerçekleştirilmektedir

SiparisAPI'de Retry Policy kullanılarak ürün işlenmesi ardından yeniden deneme sağlanmaktadır. Müşterinin sipariş detaylarını almak için bir metod oluşturulmuş ve müşteri numarasına göre bu detayların görüntülenmesi sağlanmıştır. Bu süreçte, isteklerin başarılı olana kadar 3 kez tekrarlanması sağlanmış ve sonuç başarılı bir şekilde alındıktan sonra ilgili nesnenin gösterimi gerçekleştirilmiştir. Uygulama arayüzünde, müşteri numarasının girilmesiyle logların görüntülenebildiği App.log dosyası oluşturulmuştur. Bu sayede, isteklerin tekrar sayılarına ilişkin bilgilere erişilebilmektedir.

Uygulama, bir servisten belirlenen süre içinde yanıt alamadığında tekrar deneme işlemi yapar ancak sonsuz tekrarlardan kaçınmak için bir zaman aşımı süresi belirlenir. Örneğin, müşteri bilgilerinin alınması için yapılan istekte, 3 dakika gecikme sonrasında hala yanıt alınamazsa 30 saniye bekletilir ve başarılı bir sonuç alınamadığı için hata döndürülür. Bu sayede, uygulama sonsuz döngülere girmeden belirli bir süre içinde deneme yapar ve zaman aşımı durumunda kendisini durdurur.

Müşteri bilgilerine ulaşılamadığı durumda, API üzerinde hataya dair bilgilendirme mesajı gösterilir ve bu hata durumu Timeout Policy ile detaylı bir şekilde analiz edilebilir. Örneğin, Swagger arayüzünde müşteri numarası girildiğinde zaman aşımı sonrası hata gösterilir. Bu sayede, hataların belirli bir süre içinde yönetilmesi ve analiz edilmesi sağlanır.

Uygulama arayüzünde, müşterinin numarasının girilmesiyle beklenen süre içinde zaman aşımına uğraması durumu App.log dosyasında görüntülenir. Bu gecikmelerin önüne geçilmesiyle müşterilerin zaman kaybı minimum seviyeye indirilir ve sipariş verilmesi sırasında oluşabilecek hataların etkileri azaltılmış olur.

E-ticaret uygulamalarında, sipariş servisinde ürünlerin ait olduğu müşterinin bilgilerinin alınması sırasında hatalar meydana gelebilir. Kullanıcılar, bu hataları gidermek için tekrar tekrar istekte bulunabilir. Başarısız isteklerin belirli bir sayıyı aşması durumunda veya sürekli denemeler sonucunda sistemin kilitlenmesi riski ortaya çıkar. Bu durumu önlemek için Circuit Breaker Policy uygulanır. Örneğin, müşteri servisinden 3 defa hata dönmesi durumunda, sistem 1 dakika boyunca servise yeni istek göndermez.

Uygulama üzerinde, kullanıcıların karşılaştığı hataların, kullanıcı dostu arayüzler ile açıklayıcı hata mesajları olarak gösterilmesi sağlanır. Bu, kullanıcıların hataların nedenini anlamasını kolaylaştırır ve aynı hatanın sürekli tekrar edilmesinin önüne geçer. API'lerin çalıştırılmasının ardından açılan arayüzde, müşteri numarasının girilmesiyle oluşan hatalar, app.log dosyasında kaydedilir ve Swagger arayüzünde görüntülenir.

Sürekli hatalarla karşılaşan sistemlerin belirli bir süre yanıt vermemesini sağlayan Circuit Breaker Policy, sistemin kaynaklarını korur ve kullanıcıların zaman kaybını azaltır. Ayrıca, açıklayıcı hata mesajları ile kullanıcılar, müşteri hizmetlerine başvurmadan sorunlarını anlayabilirler. Bu, e-ticaret platformlarında müşteri hizmetlerine olan talebi azaltarak hem kullanıcı memnuniyetini artırır hem de işletmenin kaynaklarını daha verimli kullanmasını sağlar.

5. SONUÇ

Bu çalışmada, monolitik ve mikroservis mimarisinin özellikleri, avantajları ve dezavantajları detaylı olarak incelenmiştir. Mikroservis mimarisi, karmaşık yapıları belirli işlevlere göre ayırarak ölçeklendirme avantajı sağladığı için e-ticaret uygulamalarında sıklıkla tercih edilmektedir. Bu mimari, birbirinden bağımsız çalışan servisler sayesinde bir serviste meydana gelen hatanın diğer servislerin çalışmasını engellememesini sağlar.

Polly kütüphanesi, hata yönetimini kolaylaştırarak geliştiricilere ciddi anlamda zaman kazandırır. Retry stratejisi başarısızlıkları azaltır ve sistemi stabil hale getirir. Circuit Breaker, sisteminizin daha da zorlanmasını önler ve bir çökme durumunda kontrolü elimizde tutmamızı sağlar. Bu da uzun vadede daha az çökme ve daha yüksek kullanıcı memnuniyeti demektir. Polly, aynı zamanda uygulamaların dayanıklılığını artırarak hem geliştirici ekiplerine hem de şirketlere zaman kazandırır. Hatayı erken aşamada tespit etmek ve onu öngörülebilir hale getirmek gibi son kullanıcı memnuniyeti artırılır. Fallback stratejisi, bir hata meydana

geldiğinde alternatif bir yanıt sunulmasına olanak tanır. Veritabanı sorgusu başarısız olduğunda, kullanıcıya "şu an hizmet kullanılamıyor" mesajı sunmak yerine, Polly'nin yardımıyla belirli uygun bir mesaj veya veri sunulabilir. Bir işlemin belirli bir süre içinde sonuçlanmaması durumunda, Polly ile bu işlemi sonlandırılarak daha verimli bir sistem yaratılır. E-ticaret uygulamalarında sipariş veren müşteriye ait bilgilerin erişimi sırasında meydana gelen hataların önüne geçebilmek için .NET Polly kütüphanesi kullanılarak, sistem karmaşıklığı, veri tutarlılığı ve performans kayıpları gibi önemli konular etkin bir şekilde yönetilebilir. Bu sayede, müşteri memnuniyeti ve e-ticaret platformlarına olan güven artırılır.

Sonuç olarak, geliştirilen uygulama ile e-ticaret platformlarında sipariş sürecinde müşteri verilerine dair oluşan hataların önlenmesi için çözümler sunulmasıyla, gelecekteki çalışmalarda, yapay zekâ, veri analitiği ve makine öğrenmesi yöntemleri ile daha kişiselleştirilmiş deneyimlere odaklanarak müşteri deneyimini optimize etmeyi hedefleyebilir. Ayrıca, sanal ve artırılmış gerçeklik teknolojileri kullanılarak müşterilere daha esnek ve kontrollü alışveriş deneyimleri sunulabilir. Bu yenilikçi yaklaşımlar, müşteri memnuniyetinin artmasına ve e-ticaret platformlarının başarısının devamlılığına katkı sağlayacaktır.

6. KAYNAKÇA

Altınkaya, C. (2022). *Üniversite bilgi sistemleri için REST tabanlı bir web servis platformunun tasarımı ve geliştirilmesi.* [Yüksek lisans tezi]. Atatürk Üniversitesi.

Asrowardi, I., Putra, S, D. ve Subyantoro, E. (2020). *Designing microservice architectures for scalability and reliability in e-commerce.* (s. (Vol. 1450, No. 1, p. 012077). IOP Publishing). Journal of Physics: Conference Series.

Dervişi, F. (2022). *Açık bankacılık sistemlerinde monolitik mimariden mikroservis mimariye geçiş.* [Yüksek lisans tezi]. Trakya Üniversitesi.

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R. ve Safina, L. (2017). Microservices: yesterday, today, and tomorrow. Present and ulterior software engineering.

gRPC Authors. (2024). Introduction to gRPC, 5 Şubat 2024 tarihinde https://grpc.io/docs/what-is grpc/introduction/ adresinden edinilmiştir.

Gos, K. ve Zabierowski, W. (2020). The comparison of microservice and monolithic architecture. *2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH).*

Gördesli, M. ve Varol, A. (2022). Comparing interservice communications of microservices for e-commerce industry. (s. pp. 1-4). *2022 10th International Symposium on Digital Forensics and Security (ISDFS) IEEE.*

Hasselbring, W. ve Steinacker, G. (2017). Microservice architectures for scalability, agility and reliability in e-commerce (s. pp. 243-246). *2017 IEEE International Conference on Software Architecture Workshops (ICSAW).*

Kyryk, M., Tymchenko, O., Pleskanka, N. ve Pleskanka, M. (2022). *Methods and process of service migration from monolithic architecture to microservices.* (s. pp. 553-558). 2022

IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET).

Microsoft. (2024). Application Resiliency Patterns. 10 Şubat 2024 tarihinde https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/application-resiliency-patterns adresinden edinilmiştir.

Microsoft. (2024). Microservices architecture style. 24 Şubat 2024 tarihinde https://learn.microsoft.com/en-us/azure/architecture/microservices/design/ adresinden edinilmiştir.

Pandurang. (2022). *How does HTTP protocol work.* 20 Ocak 2024 tarihinde https://pandurangpatil.medium.com/how-does-http-protocol-work-426b1a4158f3 adresinden edinilmiştir.

Shafabakhsh, B., Lagerström, R., Hacks, S. (2020). Evaluating the impact of inter process communication in microservice architectures. QuASoQ@ APSEC.

Terra, R., Valente, M. T. ve Bigonha, R. S. (2012). *An approach for extracting modules from monolithic software architectures* (s. pp. 1-18). IX Workshop de Manutenção de Software Moderna (WMSWM).

Van Steen, M. ve Tanenbaum, A. (2017). Distributed systems. Leiden, The Netherlands: Maarten van Steen.

Wang, T., Zhang, W., Xu, J. ve Gu, Z. (2020). *Workflow-aware automatic fault diagnosis for microservice-based applications with statistics.* (s. 17(4), 2350-2363). IEEE Transactions on Network and Service Management.

Wang, X., Zhao, H., Zhu, J. (1993). *GRPC: A communication cooperation mechanism in distributed systems.* (s. 27(3), 75-86). ACM SIGOPS Operating Systems Review.

Wei, P., Hong, Z. ve Shi, M. (2016). Performance analysis of HTTP and FTP based on OPNET. 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS).

Zhou, vd., 2018). F*ault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study.* IEEE Transactions on Software Engineering,47(2), 243-260.

Microsoft. (2024). Application resiliency patterns. Microsoft Learn. 10 Şubat 2024 tarihinde https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/application-resiliency-patterns adresinden edinilmiştir.