T.C.
EGE ÜNİVERSİTESİ
Fen Bilimleri Enstitüsü

BLOKZİNCİRLERİ VE MİKROSERVİSLERİN TÜMLEŞİMİ ÜZERİNE BİR ÇALIŞMA

Naim Yürek

Danışman: Dr. Öğr. Üyesi Birol ÇİLOĞLUGİL
2.Danışman: Dr. Önder GÜRCAN

Bilgisayar Mühendisliği Anabilim Dalı

İzmir
2023

Naim Yürek tarafından Yüksek Lisans tezi olarak sunulan “Blokzincirleri ve Mikroservislerin Tümleşimi Üzerine Bir Çalışma” başlıklı bu çalışma EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliği ile EÜ Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi’nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 20/09/2023 tarihinde yapılan tez savunma sınavında aday oybirliği/oyçokluğu ile başarılı bulunmuştur.

Jüri üyeleri **İmza**

Jüri Başkanı : Dr. Öğr. Üyesi Birol ÇİLOĞLUGİL

Raportör Üye : Prof. Dr. Geylani KARDAŞ

Üye : Prof. Dr. Doğan AYDIN

EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

ETİK KURALLARA UYGUNLUK BEYANI

EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliğinin ilgili hükümleri uyarınca Yüksek Lisans Tezi / Doktora Tezi olarak sunduğum "Blokzincirleri ve Mikroservislerin Tümleşimi Üzerine Bir Çalışma." başlıklı bu tezin kendi çalışmam olduğunu, sunduğum tüm sonuç, doküman, bilgi ve belgeleri bizzat ve bu tez çalışması kapsamında elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara atıf yaptığımı ve bunları kaynaklar listesinde usulüne uygun olarak verdiğimi, tez çalışması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını, bu tezin herhangi bir bölümünü bu üniversite veya diğer bir üniversitede başka bir tez çalışması içinde sunmadığımı, bu tezin planlanmasından yazımına kadar bütün safhalarda bilimsel etik kurallarına uygun olarak davrandığımı ve aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul edeceğimi beyan ederim.

20/09/2023

İmzası

Ad Soyad

ÖZET

Blokzincirleri ve Mikroservislerin Tümleşimi Üzerine Bir Çalışma

YÜREK, Naim

Bilgisayar Mühendisliği Anabilim Dalı Bilgisayar Mühendisliği Yüksek Lisans Programı

Tez Danışmanı: Dr. Öğr. Üyesi Birol ÇİLOĞLUGİL, Dr. Önder GÜRCAN

20/09/2023, 53 sayfa

Mikroservis mimarisi, büyük yazılım projelerini daha yönetilebilir ve ölçeklenebilir hale getirmek için tasarlanmış bir yaklaşımdır. Bu mimari yapının avantajları arasında ölçeklenebilirlik, bağımsız servis dağıtımı ve daha hızlı kod paketleme ve taşıma süreçleri bulunmaktadır. Ancak, bu mimari yapının da bazı dezavantajları vardır. Özellikle, servisler arası iletişim karmaşıklığı, veri tutarlılığı sorunları ve potansiyel olarak daha yüksek hata oranları bu dezavantajlardan bazılarıdır.

Blockchain teknolojisi, dağıtık defter yapısıyla bilinen bir teknolojidir ve bu yapısıyla veri güvenliği ve şeffaflığı sağlar. Mikroservis mimarisiyle entegre edildiğinde, Blockchain'in onay yapısı, mikroservis mimarisi dezavantajlarını azaltabilir veya avantajlarını artırabilir. Özellikle, Blockchain'in sağladığı şeffaflık ve güvenlik, mikroservis mimarisiyle birleştiğinde, servisler arası iletişimde ve veri tutarlılığında potansiyel iyileştirmeler sağlanabilmektedir.

Sonuç olarak, Mikroservis mimarisi ve Blockchain'in birleştirilmesi, her iki teknolojinin de avantajlarından yararlanarak daha güçlü, güvenli ve ölçeklenebilir bir mimari yapının ortaya çıkmasını sağlayabilmektedir. Bu kombinasyon, özellikle büyük ölçekli projelerde ve karmaşık sistemlerde, performansı ve güvenliği artırarak yeni nesil yazılım çözümlerinin geliştirilmesine olanak tanımaktadır.

Anahtar sözcükler: Mikroservis, Monolitik, SOA (Service-Oriented Architecture), Blockchain, Discovery Servis, API Gateway, Load Balancer, Servisler arası iletişim, Event-driven, RESTful servisler, Veri tutarlılığı, Veritabanı izolasyonu, Circuit Breaker, Orkestrasyon, Koralasyon, Config Server, Servis Mesh, Granularity (Tanelilik), Veri replikasyonu, Dağıtık defter (Distributed Ledger).

ABSTRACT

A Study on the Integration of Blockchains and Microservices

YÜREK, Naim

Master in Department of Computer Engineering
Supervisor: Dr. Öğr. Üyesi Birol ÇİLOĞLUGİL, Dr. Önder GÜRCAN
20/09/2023, 53 pages

The microservice architecture is designed as an approach to make large software projects more manageable and scalable. Among the advantages of this architectural structure are scalability, independent service deployment, and faster code packaging and transportation processes. However, this architectural approach also has some disadvantages. Specifically, the complexity of inter-service communication, data consistency issues, and potentially higher error rates are some of these disadvantages.

Blockchain technology is known for its distributed ledger structure, providing data security and transparency with this structure. When integrated with the microservice architecture, the validation structure of Blockchain can mitigate the disadvantages of the microservice architecture or enhance its advantages. In particular, the transparency and security provided by Blockchain, when combined with the microservice architecture, can offer potential improvements in inter-service communication and data consistency.

In conclusion, the combination of Microservice architecture and Blockchain technology can bring forth a more robust, secure, and scalable architectural structure by leveraging the advantages of both technologies. This combination, especially in large-scale projects and complex systems, provides the opportunity to develop next-generation software solutions by enhancing performance and security.

Keywords: Microservice, Monolithic, SOA (Service-Oriented Architecture), Blockchain, Discovery Service, API Gateway, Load Balancer, Inter-service communication, Event-driven, RESTful services, Data consistency, Database isolation, Circuit Breaker, Orchestration, Correlation, Config Server, Service Mesh, Granularity, Data replication, Distributed Ledger.

ÖNSÖZ

Teknolojiyi hem hayatımda hemde işimde kullanıp takip ediyorum. Şirketlerin daha yaygın ve dünya üzerinde daha ulaşılabilir olması müşterilerinin sayısının ve isteklerinin eski kullandığımız teknolojiler ile sağlanmakta zorluk çektiğini yüksek kullanıcılı geliştirdiğim uygulamalardan ve yüksek lisans tezim için yaptığım araştırmalarda tanık oldum. Ancak eski mimari ve teknolojilerinde eksiklikleri hatta eskiye göre dezavantajlı olduğu yönünde yazılan araştırmaları okuyunca tez hocalarımın yönlendirmeleri ile bu yönde bir çalışma yapmaya karar verdim. Bu yolda mikroservisler konusunda yardımcı olan ve bu konuda araştırmalar yapmış olan danışmanım Dr. Ögr. Birol Çiloğlugil ve blokzincir konusunda araştırmalar yapmış olan danışmanım Dr. Önder Gürcan'ın bu iki konu üzerinde bir çalışma yapmamı kolaylaştırdı. Bu süreçte katkılarından dolayı hocalarıma teşekkür ederim. Mikroservis ve blokzincir üzerine yapılan araştırmaları inceledim. Hem ikisinin birlikte kullanılan hemde sadece mikroservisleri konu alan araştırmaları incelemem tezim için yol gösterici oldu. Bu bilgiler ışığında kendi mimari yapımı oluşturdum. Buna uygun diyagramlar ve simulasyonlar yazdım. Elde ettiğim sonuçları makalemde sizlere sunuyorum. Keyifli okumalar dilerim.

İZMİR
20/09/2023

Naim Yürek

İÇİNDEKİLER

Sayfa

ŞEKİLLER DİZİNİ

Şekil **Sayfa**

TABLOLAR DİZİNİ

Çizelge **Sayfa**

SİMGELER VE KISALTMALAR DİZİNİ

Simgeler	Açıklama
SOA	Servis Odaklı Mimari (Service Oriented Architecture)
MS	Mikroservis
APP	Uygulama (Application)
API	Uygulama Programlama Arabirimi

1 GİRİŞ

Günümüz dünyasında, büyük ekiplerle kesintisiz servis sunarak yazılımları, sistemleri, kod entegre etme ve dağıtma yapılarını oluşturmak; şirketlerin ürünlerinin yapabildiklerinin artmasıyla sistemlerin büyümesi ve karmaşıklığıyla birlikte bir meydan okuma haline gelmiştir. Ayrıca, bu karmaşık yapıların her zaman erişilebilir, hızlı ve istikrarlı bir servis sunarak müşteri memnuniyetini sağlaması, rekabetçi ortamda neredeyse bir zorunluluk haline gelmiştir. Yazılım geliştirme ekiplerinin küçük, yönetilebilir ve kendi alanlarına hakim olmalarını, birbirlerinden soyutlanmış ve çevik bir yapıya sahip olmalarını sağlamak için sistem ve altyapının bu süreci desteklemesi gerekmektedir. Mikroservis mimarisi kullanımı zorunlu olmamakla birlikte, birbirinden ayrı çalışan çevik ekiplerin sayısının artmasıyla, mikroservis mimarisi, birbirine bağlı yazılım geliştirme süreçlerinin daha da zorlaşmasını önleyebilir (Bakshi, 2017). Bu bağlamda, bu zorluklarla başa çıkmak amacıyla, Servis Odaklı Mimarisi (SOA) önerilmiştir (Huhns and Singh, 2005).

SOA'nın yapısında Şekil 2.3 gösterildiği üzere, veri erişimi için ayrı bir katman, iş geliştirmeleri için ayrı bir katman, model için ayrı bir katman ve uygulamanın dışa açılması için ayrı bir katman bulunmaktadır. Veritabanı genellikle tekildir, çünkü veritabanı işlem bütünlüğünün garantisini sağlar. Öte yandan, mikroservis mimarisinde Şekil 2.4 gösterildiği üzere, her iş varlığının bir mikroservis olarak kendi veritabanı bulunmaktadır. Varlıkların ve çalışma alanlarının her biri için bir servis oluşturulmuştur.

Mikroservis, bağımsız olarak konuşlandırılabilen ve karmaşık olmayan küçük uygulamalar tarafından oluşturulan başarısızlık riskinin sisteme dağıtıldığı bir uygulama geliştirme metodolojisidir[1]. Monolit işletme uygulamalarının incelenmesi ve ihtiyaçlarının anlaşılması, mikroservislerin anlaşılmasına yönelik bir başlangıç noktası olmaktadır. Bir işletmenin monolit bir uygulamada

[1]Microservices. (n.d.). Retrieved August 15, 2022, from `https://martinfowler.com/articles/microservices.html`.

taleplere tek bir işlemle yanıt verdiği durumda, karşılaşılan ölümcül bir hata, kullanılan sisteme veya bir geliştirici hatasına bağlı olabilir. Bu tür bir durum, sistemin işlemez hale gelmesine ve her şeyden sorumlu olan tek işlemin çalıştırılmasının karmaşıklığı ve zorluğu gibi sorunlara yol açabilir. Robert C. Martin'in SOLID yazılım geliştirme ilkelerinin ilk harfi olan "S", mimari açıdan servisleri daha basit kılan ve hem kod yazma hem de sistem yönetimi açısından özerk ve daha kolay hale getirilebilen "Tek Sorumluluk" kavramını ifade eder.

2 MİKROSERVİSLERİN EVRİMİ: BİR GEÇİŞ ANALİZİ

Mikroservisler uzun bir süredir mevcut olmakla birlikte, ancak yakın dönemde bir yazılım mimarisi stili olarak popülerlik kazanmıştır. 'Mikroservisler' terimi ilk defa Dr. Peter Rogers cloud computing konferansında 2005 yılında bahsedilmiştir.(Jawaddi et al., 2022) Rogers, bir mikroservisi 'a service which is fine-grained, self-contained, loosely coupled, and independently deployable' olarak tanımlamıştır. Mikroservisler konsepti, 2010'ların başlarında, daha fazla şirketin bu mimari tarzı benimsemesiyle ivme kazanmıştır. Bu geçişi gerçekleştiren ilk şirketlerden biri Netflix olup, 2009 yılında monolitik bir mimariden mikroservis mimarisine geçiş yapmıştır. Mikroservis mimarisi, her işlevsellik öğesini ayrı bir servise yerleştirir ve bu servisleri sunucular arasında dağıtarak, gerektiğinde çoğaltarak ölçeklendirir[2]. Bu dönemden sonra, Amazon, eBay, Twitter ve LinkedIn gibi birçok diğer şirket de benzer bir yaklaşımı benimsemiştir.

Geçmişten günümüze, yazılım geliştirme metodolojileri genellikle birbirlerinin üzerine eklenmeye devam etmiştir. Her geliştirilen yeni yazılım geliştirme metodolojisi ve stratejisi, eski olanların eksikliklerini gidermek amacıyla ortaya çıkmıştır.

Aşağıda, avantajları ve dezavantajları Tablo 2.3'de özetlendiği şekilde, her mimariyi kronolojik bir sırayla tartışmaktayız. Her mimari, belirli bir problemi çözmek için tasarlanmıştır ve bu, performans, bakım, modülerlik gibi çeşitli faktörlere göre değişkenlik gösterebilir. Tablo, her mimarinin bu faktörlere nasıl yanıt verdiğini göstererek, bir yazılım projesi için hangi mimarinin en uygun olduğuna karar vermede yardımcı olmaktadır.

[2]Microservices. (n.d.). Retrieved August 15, 2022, from `https://martinfowler.com/articles/microservices.html`.

	Monolith	**Layered**	**SOA**	**Microservice**
The problem it is solving	Application development with less infrastructure setup (Gos and Zabierowski, 2020).	n-Tier Architecture allows businesses to select the components they need.	SOA brings a new level of abstraction(**?**).	Used in high load systems where continuous service is required.
Simplicity of Development	Easy (Gos and Zabierowski, 2020)	Normal	Normal	Hard (Baraki, 2018)
Performance	Varies with request number (Gos and Zabierowski, 2020)	Easy to optimize	Varies with input validation	Varies with request number (Gos and Zabierowski, 2020)
Maintain	Complex (Gos and Zabierowski, 2020)	High cost	Complex	Easy (Gos and Zabierowski, 2020)

Tablo 2.1: Mimarilerin karşılaştırılmasıdır. Bu tablo, farklı yazılım mimarilerinin (Monolith, Layered, SOA ve Microservice) temel özelliklerini ve avantajlarını karşılaştırmaktadır.

	Monolith	Layered	SOA	Microservice
Governance	Manual management			Easy with new tools (Soldani et al., 2018)
Network Latency	Low	Normal	Normal	High due to many services
Separation of Concerns	Hard	Not for complex apps	Not for complex apps	Easy
Scalability and Resource Utilization	Hard to scale (Gos and Zabierowski, 2020)	Supports fewer users	Normal	Easy to scale (Gos and Zabierowski, 2020)
Choice of Tech Stack	Very hard	Hard	Hard	Easy
Fault Tolerance	One fault affects all	Hard	Hard	Affects only that microservice

Tablo 2.2: Mimarilerin karşılaştırılmasının devamıdır.

2.1 Monolit Mimarisi

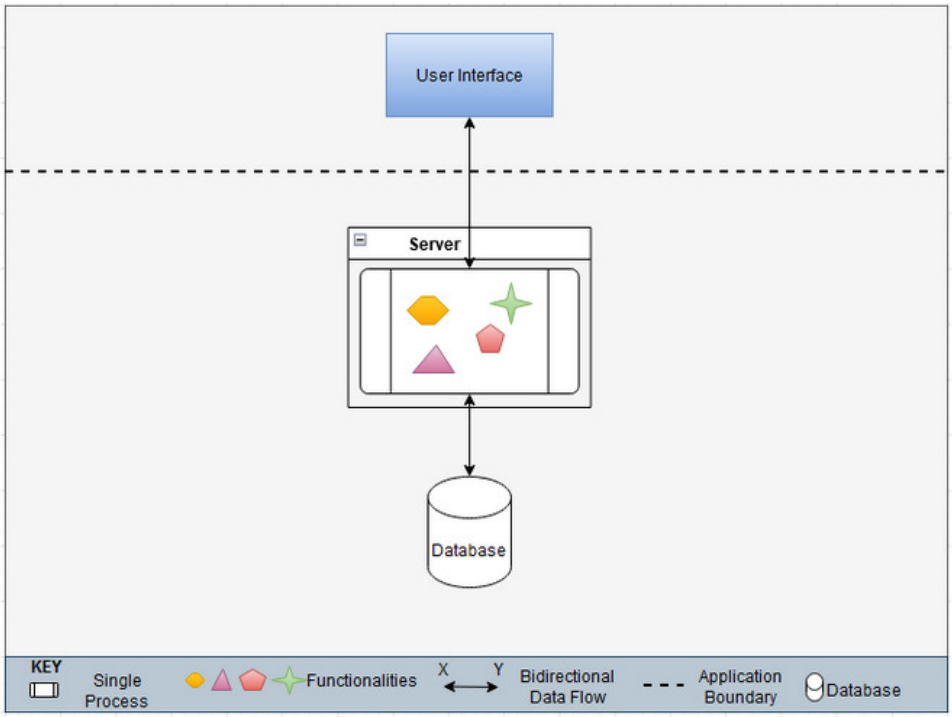
Monolit, aynı işlemde birden fazla modül ve farklı işlemlerde çalışan servis modülleri içerir.

Monolit yazılım mimarisi, Şekil 2.1'de olduğu gibi bir sistemin tüm işlevselliğini içeren tek, büyük bir uygulamadır. Bu tür bir mimari, genel-

	Monolith	Layered	SOA	Microservice
Fault Tolerance	One fault affects all	Hard	Hard	Affects only that microservice
Data consistency	Easy	Easy	Easy	Hard (Soldani et al., 2018)(Baraki et al., 2018)
Data duplication	Single db	Single db	Single db	Multiple db
Monitor	Easy	Easy	Easy	Requires distributed tools
Independent Dev/Deploy	Dependent apps affected	Possible independent deploy	Dependent apps affected	Easy due to structure (Soldani et al., 2018)
Flexibility	Hard	More flexible	More flexible	Easy (Gos and Zabierowski, 2020)
Extensibility	Hard	Manual extensibility	Manual extensibility	Easy (Gos and Zabierowski, 2020)
High Availability	Hard			Easy (Taibi et al., 2017) (Vayghan et al., 2019)
Fault Isolation (Netflix / chaosmonkey)	Hard	Hard	Hard	Easy (Taibi et al., 2017)

Tablo 2.3: Mimarilerin karşılaştırılmasının devamıdır.

likle küçük sistemler veya aynı anda sistemle az sayıda etkileşime sahip kullanıcısı olan sistemler için kullanılır. Monolit yazılım mimarisinin avantajı, geliştirmenin ve sunucu üzerinde kurulumunun kolay olmasıdır. Monolit



Şekil 2.1: Monolit Mimarisi

yazılım mimarisinin dezavantajı ise ölçeklendirmenin zor olmasıdır (Gos and Zabierowski, 2020).

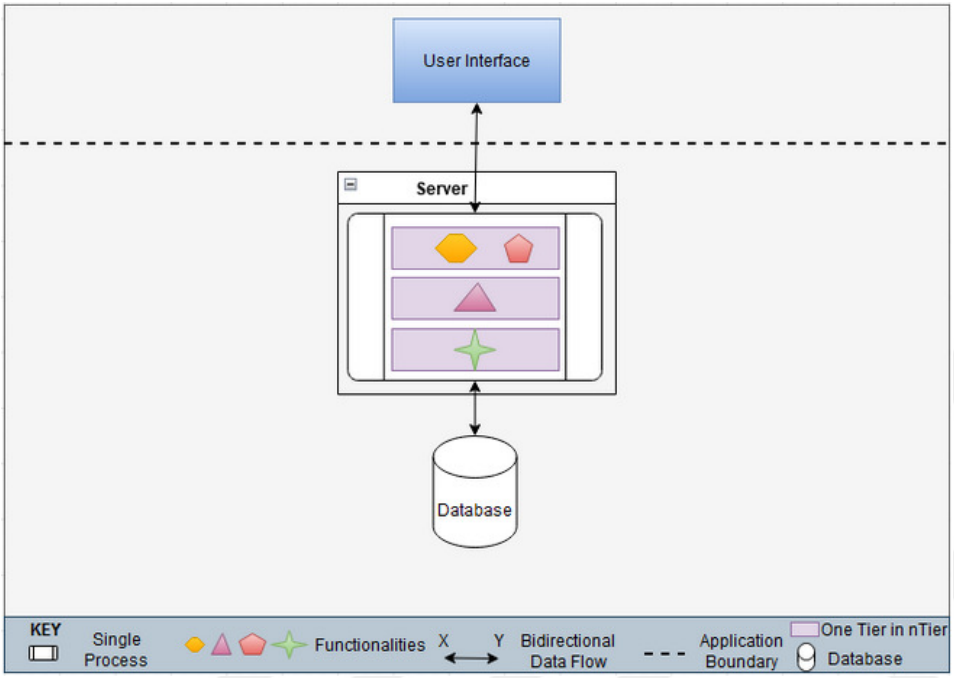
Monolitik yazılım mimarisi, tüm bileşenlerin sıkı bir şekilde bağlı olduğu ve aynı veri alanını paylaştığı bir yazılım mimarisidir. Bu tür bir mimari, tüm bileşenlerin tek bir ekip tarafından kolayca yönetilebildiği küçük uygulamalarda tipik olarak kullanılır.

Monolitik yazılım mimarisinin ana avantajı, geliştirmenin ve yayınlamanın kolay olmasıdır. Tüm bileşenler birlikte konuşlandırılır ve aynı veri alanını paylaşır, bu yüzden uyumluluk sorunları konusunda endişelenmeye gerek yoktur.

Monolitik yazılım mimarisinin ana dezavantajı ise ölçeklenebilir olmamasıdır. Uygulamanın ölçeklendirilmesi gerektiğinde, tüm bileşenlerin birlikte yeniden dağıtılması gerekir, bu da zaman alıcı ve maliyetli bir süreç olmaktadır. Ayrıca, tüm bileşenler sıkı bir şekilde bağlı olduğundan, bir bileşenin değiştirilmesi durumunda, tüm uygulamanın yeniden dağıtılması gerekmektedir.

2.2 n-Katmanlı Mimari

n-Tier Mimarisi, işletmeler için tek seçenek olan ana bilgisayarların kullanıldığı bilgisayarın ilk dönemlerine dayanır. Bu sistemler, maliyetli ve karmaşık olup, bakımı yüksek derecede uzmanlık gerektirir. Bu nedenle, sadece birkaç büyük şirket tarafından kullanılabilir.



Şekil 2.2: N-Tier Mimarisi

n-Tier Mimarisi, ana bilgisayar modelinin eksikliklerini ele almak üzere tasarlanmıştır. Her büyüklükteki işletmeler tarafından kullanılabilen daha ölçeklenebilir ve esnek bir yaklaşım sunar. Bu mimari Şekil 2.2'de görünmektedir.

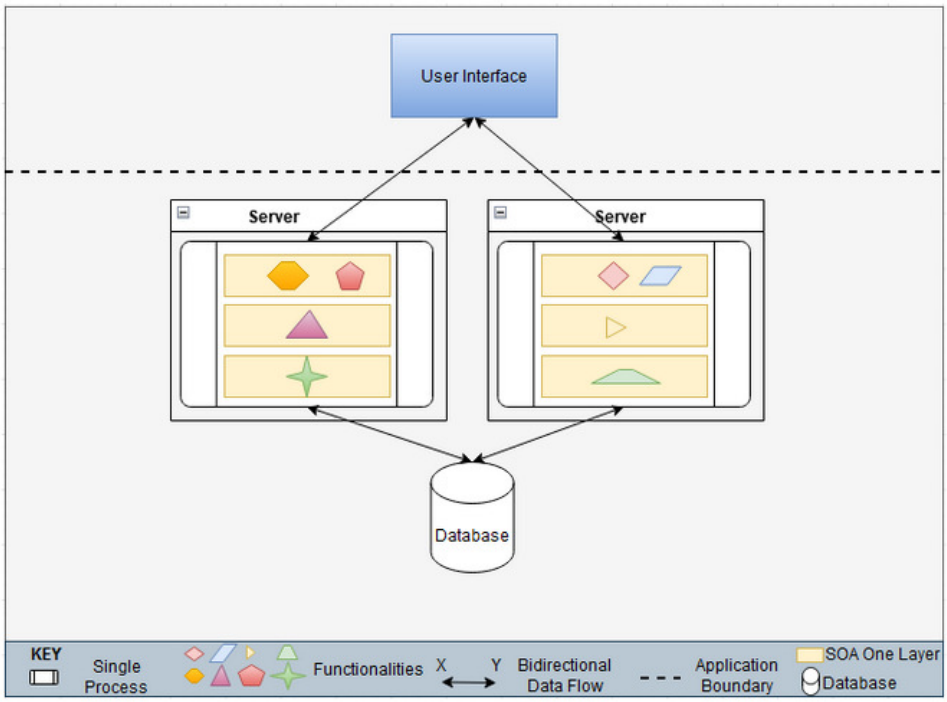
n-Tier Mimarisi, bir sistemin katmanlara veya seviyelere ayrılması konseptine dayalıdır. Her katman belirli bir işlevden sorumludur(Romhányi and Vámossy, 2021). Katmanlar, birden fazla sunucuya dağıtılabilir veya Şekil 2.2'de gösterildiği gibi tek bir sunucuda çalıştırılabilir. n-Tier Mimarisi'nin avantajı, işletmelerin belirli bir uygulama için ihtiyaç duydukları bileşenleri seçebilmeleridir. Bu esneklik, sistemin işletmenin özel ihtiyaçlarına göre uyarlanabilmesini sağlar.

2.3 Servis Odaklı Mimari

SOA, yazılım uygulamalarının düşünülme ve tasarlanma şeklini etkilemiştir. Yazılım geliştirmeye yeni bir soyutlama seviyesi getirmiş ve uygulamaların servislerin bir bileşimi olarak oluşturulabilmesini sağlamıştır.

SOA anlayışı, internetin yükselişi ve daha ölçeklenebilir ve esnek uygulamalara olan ihtiyaç gibi faktörler tarafından teşvik edilmiştir. Geleneksel "Waterfall" yazılım geliştirme modeli, internetin hızla değişen yapısına uygun değildi, ve SOA, işletmenin sürekli değişen gereksinimleriyle ayak uydurabilecek daha çevik bir yaklaşım sunmuştur.

2000'lerin başlarında, SOA ana akım benimsemeye başlamış ve bugün kurumsal uygulamalar için en yaygın kullanılan mimarilerden biri haline gelmiştir. SOA'nın faydaları arasında artan esneklik, bileşenlerin daha iyi yeniden kullanılabilirliği ve gelişmiş ölçeklenebilirlik bulunmaktadır.(Rallapalli, 2011)



Şekil 2.3: Servis Odaklı Mimari

2.4 Mikroservis Mimarisi

Mikroservislerin son yıllarda giderek daha popüler hale gelmesinin birkaç nedeni bulunmaktadır. İlk olarak, yazılım geliştirirken daha büyük esneklik ve çeviklik sağlarlar. Monolitik mimarilerin aksine, yerleştirildikten sonra değiştirmek çok zor olabilirken, mikroservisler gerektiği gibi kolayca eklenebilir veya kaldırılabilir. Bu, yazılım tasarımları üzerinde yinelemeyi ve kullanıcı geri bildirimine hızla yanıt vermek üzere değişiklikler yapmayı çok daha kolay kılar. Farklı çözümler için farklı programlama dilleri kullanma gibi bir çözüm, dil ve platforma bağlı olduğundan uygulanamaz. Uygulamanın boyutu ve yeni gelen değişiklik talebi daha azdı.

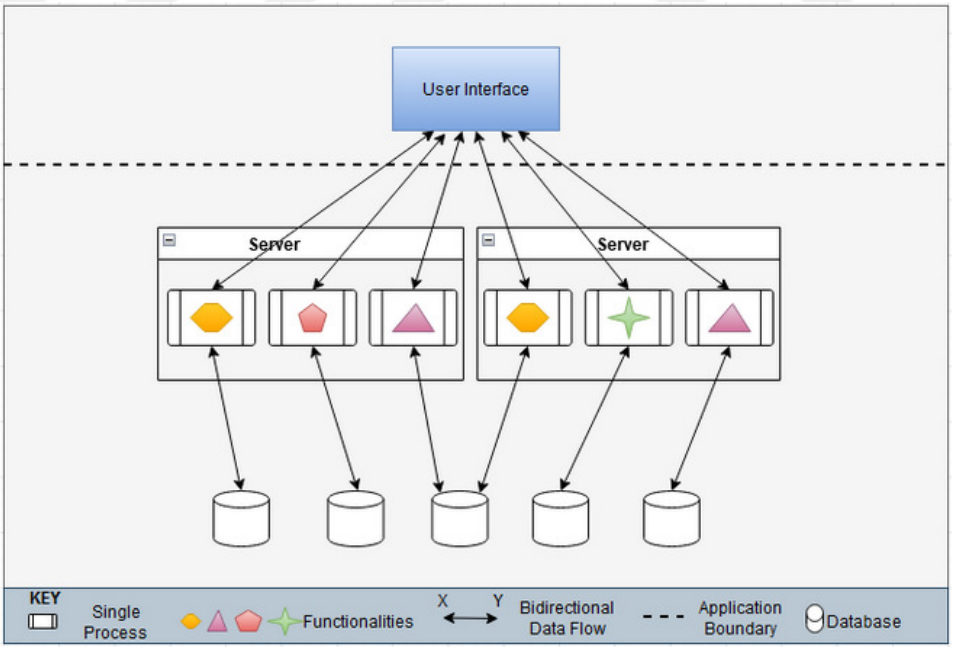
İkinci olarak, mikroservisler monolitik mimarilere göre daha iyi bir şekilde ölçeklenebilir. Monolit mimarisiyle tasarlanmış bir uygulamanın ölçeklendirilip çoğaltılması istendiğinde, tüm uygulama bölünmeden çoğaltılmalıdır. Monolitik uygulamanın ölçeklendirilmesi gerektiğinde, tüm uygulama yeni donanıma yüklenmek üzere çoğaltılmalıdır. Bu, maliyetli ve zaman alıcı olmaktadır. İlk geliştirilen web uygulamalarında, mimari monolitikti ve API sayısı azdı. Tüm uygulama tek büyük serviste çalışıyordu. Bu tek ve büyük servislerle olan mimari, monolit mimari olarak tanımlanır. Monolit mimari uygulaması, tüm işlevselliği tek bir işleme koyar. n-tier uygulamalar oluşturulmuştur. n-tier mimarisinde, belirli katmanlara bölünmüş uygulamaların bölünmesi ve ihtiyaç duyulan katmanın çoğaltılmasıyla katmanlar arası çağrılarla sınırlı bir servis ölçeklendirmesi yapılmaya çalışılmıştır. Mikroservislerle, her servis kendi bağımsız donanım örneğinde konuşlandırılabilir. Bu, gerektiği gibi bir uygulamanın ölçeklendirilmesini çok daha kolay ve ekonomik kılar.

Üçüncü olarak, mikroservisler kodun ve bileşenlerin daha iyi yeniden kullanılmasını teşvik eder. Monolitik bir mimaride, farklı uygulamalar veya aynı uygulamanın farklı kısımları arasında kod modüllerinin yeniden kullanılması zor olmaktadır. Zamanla, uygulamalarımızın yapabileceği şeylerin sayısı arttı. Müşteri ihtiyaçları ve istekleri arttı. Yazılım geliştiricileri ve mimarlar olarak, yeni müşteri talepleri için yazılımın yeniden kullanılabilirliğini kullanarak SOA yazılım mimarileri gibi çözümler oluşturuldu. Aynı zamanda, müşteri sayısının

ve sistemlerdeki sunucularda çalışan servislerin artmasıyla, servisler katmanlara bölünerek farklı katmanların farklı sunucularda çalıştırılması sağlandı. SOA mimarisi, yazılan kodun kalitesinin bozulmadan yeni iş taleplerinin sisteme dahil edilmesi ve kod çoğaltmasının önlenmesiyle daha çok ilgilenir. Mikroservislerle, her servis kendi içinde bağımsızdır ve diğer bağlamlarda kolayca yeniden kullanılabilir.

Dördüncü olarak, mikroservisler hata izolasyonunu ve kullanılabilirliği artırır. Monolitik bir mimaride bir servis çökerse, tüm uygulamanın da çökmesi muhtemeldir. Mikroservislerle, her servis bağımsızdır ve diğer servisler kullanılamaz durumda olsa bile çalışmaya devam edebilir. Bu, sistemin genel direncini artırır.

Son olarak, mikroservisler, izinler ve güvenlik üzerinde daha ince taneli kontrol sağlar. Monolitik bir mimaride, uygulamanın farklı kısımlarına erişimi kontrol etmek veya belirli işlemleri belirli kullanıcılara sınırlamak zor olmaktadır. Mikroservislerle, her servisin ihtiyaca göre özelleştirilebilen kendi güvenlik modeli vardır. Bu, yöneticilere, bir uygulama içindeki verilere ve işlevselliğe kimin erişebileceği üzerinde daha fazla kontrol sağlar.



Şekil 2.4: Mikroservis Mimarisi

Mikroservisler bölündüğünde daha az sunucu maliyeti ortaya çıkar, ancak monolit uygulamalarda yeni bir işlem gerektiğinde tüm proje çalışır ve çalışması gerekmeyen çok fazla modül için kaynak harcanır.

3 BLOKZİNCİR

Blokzincir, genellikle "blockchain" olarak bilinen, dağıtık bir defter teknolojisidir. Bu teknoloji, herkese açık, sadece ekleme yapılabilen, değiştirilemez ve sıralı bir işlem kaydını sürdürme kapasitesiyle dikkat çeker. Bu, herkesin erişebileceği denetlenebilir bir defteri garanti eder. Blok zincir, başlangıçta 2008'de Bitcoin kripto parası altında oluşturuldu ve o zamandan beri blok zincir ekosistemi oldukça büyüdü. Bugün, farklı algoritmalar ve izinlere dayalı birçok blok zincir varyantı bulunmaktadır (Ammous, 2017).

Blok zincir sistemleri, katılımcılarının birbirlerine tamamen güvenmeden, güvendiği üçüncü bir tarafa dayanmadan veya sistemin küresel bir görünümüne sahip olmadan doğrulanmış işlemler gerçekleştirmelerine olanak tanır (Ruoti et al., 2017). Bu, katılımcıların blok zincirini bir işlem hizmeti olarak kullanmaları ve bu hizmeti sağlamak için katkıda bulunmaları anlamına gelir.

Blok zincir teknolojisi, genel olarak birçok alanda büyük bir potansiyele sahiptir. Özellikle, bu teknoloji kriptografi, çok etmenli sistemler, dağıtık sistemler, sosyal sistemler, ekonomi ve finans gibi çeşitli alanları birleştirir. Bu, blok zincir sistemlerinin doğası gereği disiplinlerarası olduğu anlamına gelir (Roussille et al., 2020).

Blokzincir, sadece kripto para ile ilgili işlemleri sağlamakla kalmaz, aynı zamanda Turing-tamamlayıcı Akıllı Sözleşmeleri destekler. Bu, blok zincirlerin sadece kripto para platformlarından dağıtık işlemsel ve mantıksal sistemlere evrildiği anlamına gelir (Pierro, 2017).

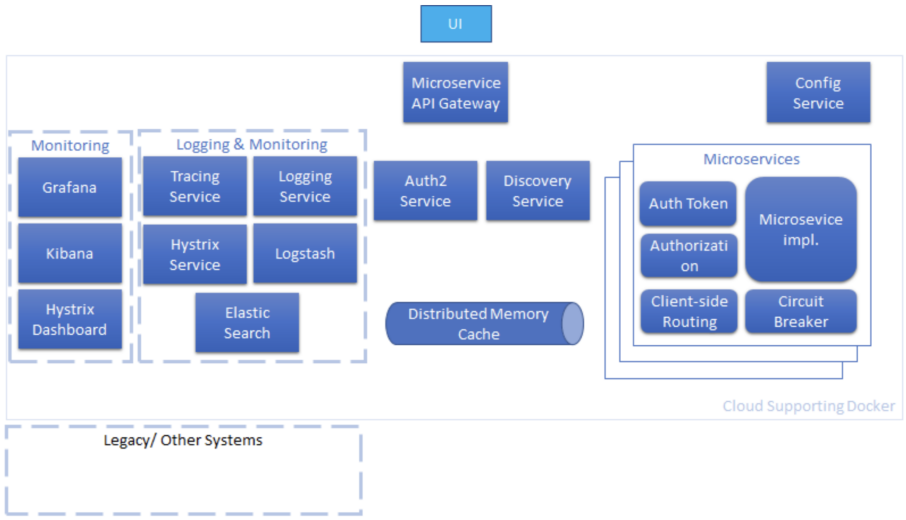
4 MİKROSERVİS MİMARİSİ

Mikroservisler, büyük uygulamaların küçük, bağımsız servisler olarak inşa edildiği bir yazılım mimarisi türüdür. Her servis, benzersiz bir işlemi çalıştırır ve diğer servislerle iyi tanımlanmış bir arayüz aracılığıyla iletişim kurar. Bu mimari, yazılım uygulamalarının esnekliğini, ölçeklenebilirliğini ve bakımını artırmak için kullanılır.

Mikroservis mimarisi, son yıllarda monolitik mimarilere kıyasla sağladığı avantajlar nedeniyle popüler hale gelmiştir. Bu avantajlar arasında artan esneklik, gelişmiş ölçeklenebilirlik ve daha iyi hata izolasyonu bulunmaktadır. Ayrıca, mikroservisler daha sık yazılım sürüm dağıtımlarına izin verir ve daha hızlı yenilik yapılmasını sağlar.

4.1 Araçlar ve Bileşenler

Avantajlara rağmen, mikroservis mimarisi bazı zorluklar da getirir. Bu zorluklar arasında artan karmaşıklık, dağıtılmış veri ve iletişim yükü bulunmaktadır. Aşağıdaki bölümler, bir mikroservis mimarisi oluşturmak için gerekli olan araçlar ve bileşenler hakkında detaylı bir şekilde incelenecektir. Mikroservis içerisinde kullanılan yapılar örneklerle beraber Şekil 4.1 de sunulmuştur.



Şekil 4.1: Mikroservis Mimarisi Sistem Diyagramı

4.1.1 Servis Keşfi(Service Discovery)

Servis Keşfi, mikroservis mimarisinin ana bileşenlerinden biridir. Servis Keşfi, servislerin birbirlerini bulmalarına ve iletişim kurmalarına olanak tanır. Kaşif servisi, diğer servislerin adreslerini ve hangi servisin hangi yola karşı çalıştığını saklar. Mikroservis mimarisindeki bir servisin, kullanılabilir servislerin yerlerini saklayan bir kayıt kullanarak başka bir servisle buluşup iletişim kurmasını sağlar. Bu önemlidir çünkü servislerin yerleri dinamik olarak değişebilir ve sistemden yeni servisler eklenebilir veya çıkarılabilir. Servis Kaşif'i uygulamanın çeşitli yolları vardır, bunlar arasında özel bir sunucu kullanmak, bir yük dengeleyici veya proxy kullanmakta veya dağıtılmış bir anahtar-değer deposu kullanmaktır.(Rotter et al., 2017)

4.1.2 Yük Dengeleyici(Load Balancing)

Mikroservis mimarisinde, Servis Keşfi belirli bir servis örneğinin yerini belirleme sürecidir. Yük dengelemesi, Servis Keşfi ile birlikte kullanılarak, bir servisin birden fazla örneğine gelen ağ trafiğini dağıtmak ve servis örneklerinin eklenmesi veya çıkarılmasını otomatik olarak yönetmek için kullanılır.(Ribbon, 2022)

Bir istemci bir servise erişmek istediğinde, önce Service Discovery mekanizmasını kullanarak servisin sağlıklı bir örneğinin yerini belirler. Yük dengeleyici, bu bilgiyi kullanarak istemcinin isteğini uygun servis örneğine yönlendirir. Eğer bir servis örneği kullanılamaz hale gelirse, Service Discovery mekanizması yer bilgisini günceller ve yük dengeleyici trafiği otomatik olarak farklı bir örneğe yönlendirir.

Bu yaklaşım, mikroservislerin ölçeklendirilmesi ve kullanılabilirliğinin otomatik ve dinamik bir şekilde yönetilmesine olanak tanır ve yeni örneklerin eklenmesi veya mevcut olanların çıkarılması için manuel yapılandırma değişiklikleri yapılmasına gerek kalmaz.

4.1.3 İzleyici(Tracing)

Mikroservis mimarisindeki servis izleme, dağıtılmış bir sistem içinde birden fazla mikroservis arasındaki etkileşimleri izleme ve analiz etme sürecini ifade eder. Bu, her isteğin süresi, dahil olan uç noktalar ve oluşan hatalar gibi bilgileri içerebilir. Servis izlemenin ana amacı, sistemin genel davranışını anlamak ve ortaya çıkan sorunları tanımlayıp teşhis etmektir.(OpenZipkin, 2022)

OpenTracing gibi popüler servis izleme araçları, geliştiricilerin kodlarına izleme bilgileri eklemelerine olanak tanır. Bu bilgiler, Jaeger gibi bir izleme arka ucu tarafından toplanabilir ve mikroservisler arasındaki etkileşimlerin ayrıntılı görselleştirmelerini oluşturmak için kullanılabilir.

Servis izleme, performans darboğazlarını belirlemeye, servis bağımlılıklarını izlemeye ve sistemin genel sağlığı ve performansı hakkında görünürlük sağlamaya da yardımcı olmaktadır. Bu, sorunları tanımlamak ve performans ayarı, hata ayıklama ve mikroservis mimarisinin izlenmesi için yararlı olmaktadır.

4.1.4 Devre Kesici (Circuit Breaker)

Circuit Breaker, bir mikroservis mimarisinde başarısız olan bir servisin diğer servislere yayılmasını önlemek için kullanılabilecek bir tasarım desenidir. Arayan servis ile çağrılan servis arasına bir proxy ekleyerek çalışır ve çağrılan servisin sağlığını izler. Eğer çağrılan servis başarısız olmaya başlarsa, Circuit Breaker istek akışını kesintiye uğratabilir, böylece arayan servisin hızla başarısız olmasına ve potansiyel olarak daha hızlı toparlanmasına olanak tanır.

4.1.5 Konfigrasyon Sunucusu (Config Server)

Mikroservis mimarisinde bir yapılandırma servisi, diğer mikroservisler için yapılandırma bilgilerini saklayan ve yöneten merkezi bir servistir. Yapılandırma verilerinin mikroservislerin kodundan ayrılmasına olanak tanır, böylece yapılandırmayı yeniden dağıtmadan yönetmek ve güncellemek daha kolay olur.(Spring Cloud Config, 2022)

Config servisi, yapılandırma bilgilerini bir anahtar-değer deposunda saklar ve diğer mikroservislerin yapılandırma verilerini almak için bir API sunar. Bu, mikroservislerin çalışma zamanında gerekli yapılandırma bilgilerini almasına ve ortam değişikliklerine kod değişiklikleri veya yeniden dağıtımlar olmadan uyum sağlamasına olanak tanır.

Config servisi ayrıca sürümleme, şifreleme ve erişim kontrolü gibi özellikler sunabilir. Ayrıca, dev, test, preprod ve prod gibi farklı ortamların yapılandırmasını yönetmeye yardımcı olur.

4.1.6 API Uç Kapısı(API Gateway)

Mikroservis mimarisinde bir API Gateway, bir istemci uygulaması ile bir dizi mikroservisler arasında aracı olarak hareket eden bir sunucudur. Bu servisi hastane güvenliği ve resepsiyonuna benzetebiliriz. İstemciye servisini alacağı ve ulaşabileceği adresi o adreste istenilen gereklilikleri bildirip yönlendirebilmektedir. Ana sorumluluğu, istemciye bir dizi uç nokta sunmak ve bu istekleri uygun mikroservise yönlendirmektir (Baeldung, 2022).

API Gateway, şunları yapabilir:

- Yönlendirme: Gelen istekleri, uç nokta ve/veya istek verilerine dayalı olarak uygun mikroservise yönlendirme.
- Kimlik Doğrulama ve Yetkilendirme: İstemcinin istenen kaynağa erişim için gerekli kimlik bilgilerine sahip olup olmadığını doğrulama.
- Önbellekleme: Mikroservislere yapılan isteklerin sayısını azaltmak için sıkça istenen verileri bellekte saklama.
- Oran Sınırlama: Mikroservisleri aşırı yüklenmeye karşı koruma amacıyla gelen isteklerin oranını kontrol etme.
- Dönüşüm: Gelen isteklerin veya yanıtların formatını, mikroservislerin veya istemcinin beklediği formata uyacak şekilde dönüştürme.
- API Gateway, dış müşterilerin mikroservislere erişimini kolaylaştırabilir, bu da istemcinin kodunu basitleştirebilir ve sistem iç detaylarını dış istemcilerden gizleyerek güvenliği artırabilir.

4.1.7 Yetki Sunucusu(Authorization Server OAuth2)

Mikroservis mimarisinde bir Authorization Server, sadece yetkili erişime açık kaynaklara erişmek isteyen istemcilere kimlik doğrulama yapma ve access token düzenleme sorumluluğuna sahip özel bir servistir. OAuth 2.0 ve OpenID Connect gibi yaygın olarak kullanılan yetkilendirme ve kimlik doğrulama protokollerinin ana bileşenidir (Baeldung, 2022).

Authorization Server, istemcilerin kaydını yönetmek ve kaynak sahiplerinin (kullanıcılar) kaynaklarına erişimlerini onaylamak istediklerinde kimliklerini doğrulamakla sorumludur. Ayrıca, korumalı kaynaklara erişim için yetkilendirilen istemcilere erişim belirteçleri düzenler.

Bir istemci sadece yetkili erişime açık bir kaynağa erişmek istediğinde, access-token ile kaynak sunucusuna bir istek gönderir. Kaynak sunucusu, access-token'ı Authorization Server ile doğrular. Eğer access-token geçerliyse, kaynak sunucusu korumalı kaynağa erişimi sağlar. Authorization Server, yeni access-token almak için kullanılabilecek access-token de düzenleyebilir, böylece uzun ömürlü oturumlar sağlanabilir.

Authorization Server, LDAP, Active Directory ve sosyal girişler gibi farklı kimlik doğrulama mekanizmalarıyla entegre edilebilir, bu da daha esnek olmasını ve aynı sistemde farklı kimlik doğrulama seçeneklerini kullanmasını sağlar.

Kaynak sunuculardan ayrı bir Authorization Server olması, daha güvenli ve ölçeklenebilir bir mimari sağlar ve farklı kaynaklar ve istemciler için erişim kontrolünü daha kolay yönetilebilir bir hale getirmektedir.

5 İLGİLİ ÇALIŞMALAR

Literatürdeki birçok çalışma, blokzincirin veri tutarlığı ve dağıtık yapıda veri yayma kapasitesine odaklanmıştır. Ancak bu tezde gerçekleştirilen çalışma, bu konseptleri farklı bir perspektiften ele almaktadır. Özellikle mikroservis mimarisi içerisinde, blokzinciri sadece loglama amacıyla gateway servise entegre edilmiştir. Bu yaklaşım, sistemdeki her bir mikroservisi bir etmen olarak ele alarak, agent-based simülasyon ortamında etkili bir şekilde modellememizi sağlamıştır.

Bu çalışmanın literatürdeki diğer çalışmalardan ayırt edici bir özelliği, blokzincirin sadece loglama amacıyla sınırlı bir şekilde kullanılmasına rağmen, bu entegrasyonun potansiyel genişlemeler için bir temel oluşturmasıdır. Özellikle, gelecekteki geliştirmelerde, bütün mikroservislerin blokzincire transaction ekleyerek olası sistem sızmalarına karşı ek bir güvenlik katmanı oluşturması hedeflenmektedir. Bu yaklaşım, blokzincirin mikroservis mimarisi ile daha geniş kapsamlı bir entegrasyonunun nasıl gerçekleştirilebileceğine dair bir yol haritası sunmaktadır.

İncelenen literatürde belirtilen çalışmalarda, blokzincirin sağladığı veri tutarlığı ve değiştirilemez yapının önemine dikkat çekilmiştir. Bu özellikler, blokzincirin temel avantajları arasında yer almaktadır. Özellikle dağıtık sistemlerde veri tutarlığını sağlamak ve veriyi sisteme yaymak, blokzincirin sunduğu en kritik özelliklerden biridir. Bu bağlamda, örnek çalışmalarda, blokzincirin bu benzersiz özelliklerinin, veri tutarlığını nasıl garanti altına aldığı ve dağıtık bir yapıda verinin nasıl etkin bir şekilde yayıldığı detaylı bir şekilde incelenmiştir.

Blokzincir ve mikroservislerin bir araya getirilmesi konusunda yapılan çalışmalar, bu teknolojilerin birleşiminin özellikle sağlık sektöründe büyük bir potansiyele sahip olduğunu göstermektedir(Santos et al., 2023).

Bu çalışmalar, blokzincir ve mikroservislerin bir araya getirilmesinin potansiyel avantajlarını ve uygulama alanlarını daha iyi anlamamıza yardımcı olmaktadır. Özellikle, bu teknolojilerin bir araya getirilmesinin sağlık sektöründeki

uygulamalarını derinlemesine incelemekte ve bu alandaki açık araştırma fırsatlarını belirlemektedir.

"How Blockchain and Microservices are Being Used Together: a Systematic Mapping Study" başlıklı bir çalışmadır ve Rayane Santos, Pamella Soares, Evandro Rodrigues, Paulo Henrique M. Maia ve Amanda Silveira tarafından yazılmıştır. Çalışma, beş önemli araştırma veritabanında yapılan bir araştırma sonucunda 40 makaleyi seçerek bilgi çıkarmıştır.

Makale, blockchain ve mikroservislerin bir araya getirilmesinin nasıl yapıldığını ve bu iki teknolojinin nasıl birlikte kullanıldığını incelemektedir. Çalışma, bu iki teknolojinin nasıl entegre edildiğini, uygulama durumlarını, çözüm önerilerini ve bu alandaki mevcut araştırma durumunu incelemek için bir sistematik haritalama çalışması gerçekleştirmiştir.

Sonuçlar, blokzincir ve mikroservislerin birleşiminin en çok sektörel olarak sağlık sektöründe ele alındığını göstermektedir. Öte yandan, çoğu çalışma ya değerlendirilmemiş ya da sadece bir konsept kanıtı olarak sunulmuştur, bu da önerilerin henüz olgunlaşmadığını ve hala açık araştırma fırsatları olduğunu göstermektedir. Makalede belirtilen yüzdesel oranlar ve sayılar aşağıdaki gibidir:

- Sağlık (Healthcare): %15 (En yaygın özel alan)
- Akıllı Sözleşme (Smart Contract): %10
- Akıllı Şehir (Smart City): %7.5
- Eğitim (Education): %7.5
- Osmotik Hesaplama (Osmotic Computing): %5
- Havacılık Sistemleri (Aviation Systems): %5
- IoV (Internet of Vehicles): %5
- Tedarik Zinciri (Supply Chain): Daha az temsil edilen alanlar arasında
- Çevrimiçi Ticaret/Pazar Yeri (Online Commerce/Marketplace): Daha az temsil edilen alanlar arasında

- Gözetim Sistemleri (Surveillance Systems): Daha az temsil edilen alanlar arasında
- Noter Ofisi (Notary's Office): Daha az temsil edilen alanlar arasında
- Genel alan: %17.5 (Daha spesifik teknolojilerle ilgilenen veya yalnızca bir alan türü belirtmeyen çalışmalar)

Ayrıca, makalede belirtilen ampirik stratejilerin nasıl değerlendirildiği hakkında da bilgi verilmiştir:

- Deney (Experiment): 19 makale (%47.5)
- Vaka Çalışması (Case Study): 5 makale (%12.5)
- Anket (Survey): 1 makale (%2.5)
- Deney yapan makaleler arasında, 12 makale (%17.5) kalite değerlendirme sürecimize göre %50'nin üzerinde bir kalite indeksine sahiptir. Vaka çalışmalarının yalnızca %2.5'i %50'nin üzerinde bir not aldığı görülmüştür.

Son yıllarda, blokzinciri teknolojisi ve mikroservisler, güvenli olan ve ölçeklenebilir uygulamaların geliştirilmesinde önemli bir rol oynamaktadır. Wang göre (Wang et al., 2020) mikroservislerin akıllı sözleşmelerle nasıl entegre edilebileceğini incelemişlerdir. Bu çalışma, mikroservislerin akıllı sözleşmelerle entegrasyonunun, uygulamaların güvenliğini ve ölçeklenirliğini artırabileceğini göstermektedir. Xu göre (Xu et al., 2019) ise blokzinciri tabanlı bir mikroservisler mimarisi önermişlerdir. Bu mimari, kamusal güvenlik uygulamaları için blokzinciri tabanlı bir mikroservisler mimarisi sunmaktadır. Makalede, blokzincir ve mikroservislerin bir araya getirilmesinin avantajları ve zorluklarına dair örnekler sunulmaktadır. Örneğin, Xu ve arkadaşları, IoT tabanlı kamu güvenliği sistemlerindeki zorlukları ele almak için mikroservisler ve blokzincir teknolojisini bir araya getiren BlendMAS adlı bir mimari önermişlerdir. Ayrıca, Wang ve arkadaşları, araç İnterneti (IoV) cihazları arasında verimli ve güvenli bir şekilde mikroservis yüklemesi ve doğrulama sağlamak için blokzincir tabanlı bir blok akışı servisi önermişlerdir.

Dilshan göre mikroservisler için blokzinciri tabanlı bir sertifika şeffaflığı önermişlerdir(Dilshan et al., 2020). Bu çalışma, mikroservislerin güvenliğini ve doğrulanabilirliğini artırmak için blokzinciri tabanlı bir sertifika şeffaflığı sunmaktadır. Staples göre blokzinciri uygulamaları için bir mimari sunmuşlardır (Staples et al., 2019). Bu mimari, blokzinciri uygulamalarının tasarım ve geliştirilmesi için bir framework sağlamaktadır. Wüst ve Gervais blokzinciri teknolojisinin gerçekten gerekli olup olmadığını sorgulamışlardır(Wüst and Gervais, 2018). Bu çalışma, blokzinciri teknolojisinin ne zaman ve nerede kullanılması gerektiğine dair kılavuzlar sunmaktadır. Bandara göre ölçeklenebilir bir akıllı sözleşme mimarisi önermişlerdir. Bu mimari, servis olarak yazılım (SaaS) tabanlı mikroservisler için ölçeklenebilir bir akıllı sözleşme mimarisi sunmaktadır (Bandara et al., 2020).

Stefan açık öğrenme toplulukları için blokzinciri teknolojileri ve mikroservisler sunmuştur. Bu çalışma, açık öğrenme toplulukları için blokzinciri teknolojileri ve mikroservislerin nasıl kullanılabileceğine dair bir bakış açısı sunmaktadır (Stefan, 2020). Soares de Sousa göre noter ofisleri için mikroservisler ve blokzinciri teknolojileri temelli bir prototip önermişlerdir (Soares de Sousa et al., 2020). Bu prototip, noter ofislerinin işlemlerini daha güvenli ve verimli bir şekilde gerçekleştirmelerine olanak tanımaktadır. Noter ofisleri ve diğer kurumlar arasındaki entegrasyonu sağlamak için mikroservisler ve blokzincir tabanlı bir yaklaşım önermektedir. Bu yaklaşım, akıllı sözleşmeleri belirli mikroservislerde kapsüllemeyi mümkün kılar. Çalışma, önerilen iş modelini entegre eden bir yaklaşım, bir yaratma dökümanı oluşturabilen ve bir blokzincir üzerinde kaydedebilen bir prototip uygulama ve bu projenin geliştirilmesindeki tasarım kararları ve çıkarılan dersler hakkında bir deneyim raporu sunmaktadır.

6 BLOKZİNCİR TABANLI BİR MİKROSERVİS MİMARİSİ

Örnek çalışma örneğinde etmen tabanlı bir sistemde simule edilmesinin sebepleri Önder Gürcan'ın "Using Agent-based Modeling and Simulation for Studying Blockchain Systems"(Gürcan, 2020) makalesinde ele aldıklarını çıkarımda bulunulursa şunlardır:

Karmaşık Etkileşimleri Modelleme: Mikroservis mimarisi, birçok bağımsız servisin bir araya gelerek bir bütün olarak çalıştığı dağıtık bir sistemdir. Etmen tabanlı simülasyon, bu tür karmaşık etkileşimleri modelleyerek mikroservisler arasındaki dinamikleri anlamamıza yardımcı olabilir.

Dağıtık Sistemlerin Özellikleri: Blockchain gibi, mikroservis mimarisi de doğası gereği dağıtıktır. EBMS, dağıtık sistemlerin özelliklerini (örneğin, hata toleransı, yüksek kullanılabilirlik) simüle ederek mikroservis mimarisinin bu özellikleri nasıl ele aldığını anlamamıza yardımcı olabilir.

Sosyal ve Ekonomik Dinamikler: Mikroservisler arasındaki etkileşimler sadece teknik değil, aynı zamanda sosyal ve ekonomik dinamiklere de dayanabilir. EBMS, bu sosyal ve ekonomik dinamikleri modelleyerek mikroservislerin nasıl bir araya geldiğini ve birlikte nasıl çalıştığını anlamamıza yardımcı olabilir.

Esneklik ve Ölçeklenebilirlik: Mikroservis mimarisi, esnek ve ölçeklenir olma avantajları sunar. EBMS, farklı ölçeklendirme senaryolarını simüle ederek mikroservis mimarisinin bu avantajları nasıl sağladığını ve potansiyel zorlukları neler olduğunu gösterebilir.

Dinamik Değişikliklere Yanıt: Mikroservislerin dinamik bir şekilde eklenmesi, kaldırılması veya değiştirilmesi gerekebilir. EBMS, bu tür dinamik değişikliklere sistem genelinde nasıl yanıt verildiğini simüle ederek bu değişikliklerin genel sistem üzerindeki etkisini anlamamıza yardımcı olabilir.

Detaylı Analiz: EBMS, mikroservis mimarisinin her bir bileşeninin detaylı bir şekilde analiz edilmesine olanak tanır. Bu, potansiyel sorunları tespit etmek ve performans iyileştirmeleri yapmak için kritiktir.

Sonuç olarak, etmen tabanlı simülasyonun mikroservis mimarisi içinde kullanılması, mikroservisler arasındaki etkileşimleri, dinamikleri ve potansiyel zorlukları anlamamıza yardımcı olabilir. Bu, daha sağlam, ölçeklenebilir ve etkili mikroservis sistemleri tasarlamanın anahtarı olabilir.

Etmen tabanlı modelleme ve simülasyon (EBMS), blockchain sistemleri gibi karmaşık sistemleri incelemek için güçlü bir araçtır. EBMS'nin blockchain sistemlerini incelemek için uygun olduğunu gösteren (Gürcan, 2020) bazı nedenler:

Dağıtık Yapı: Blockchain sistemleri doğuştan dağıtıktır ve çok sayıda katılımcı merkezi olmayan bir şekilde etkileşimde bulunur. EBMS, bireysel etmenlerin etkileşimlerini ve davranışlarını yakalayarak bu tür dağıtık sistemlerin modellemesine olanak tanır.

Sosyal Organizasyonlar: Blockchain sistemleri, katılımcıların farklı hedeflere ve amaçlara sahip olduğu sosyal organizasyonlar olarak görülebilir. EBMS, etmenler arasındaki etkileşimleri modelleyerek bu sosyal dinamikleri simüle edebilir.

Ekonomik Sistemler: Blockchain sistemlerinin ekonomik sonuçları vardır ve katılımcılar ekonomik teşviklere dayanarak kararlar alır. EBMS, bu ekonomik etkileşimleri simüle ederek farklı ekonomik senaryolar altında sistem davranışına dair içgörüler sunabilir.

Dinamik Ekosistem: Blockchain ekosistemi sürekli olarak gelişmektedir, yeni platformlar ve algoritmalar geliştirilmektedir. EBMS, gerçekçi bir şekilde hipotezleri test etmeye olanak tanıyarak hızlı prototipleme ve fizibilite analizi yapılmasını sağlar.

Detaylı İnceleme: EBMS, etmen seviyesinde çalışır, bireysel etmen davranışlarının ve genel sistem üzerindeki etkisinin detaylı bir analizini sağlar. Bu detaylı inceleme, blockchain sistemlerinin nüanslarını anlamak için kritiktir.

Organizasyon Merkezli Yaklaşım: Makale, blockchain sistemlerinin gereksinimlerini yakalamak için etmen merkezli bir yaklaşımdan ziyade organizasyon merkezli bir yaklaşımın daha uygun olduğunu öne sürmektedir. EBMS, bu yaklaşımı takip edecek şekilde özelleştirilebilir, blockchain dinamiklerinin daha

doğru bir temsilini sağlar.

Sonuç olarak, etmen tabanlı modelleme ve simülasyon, bireysel etmenler arasındaki etkileşimleri ve sonuçta ortaya çıkan sistem davranışını yakalayarak blockchain sistemlerinin karmaşık dinamiklerini incelemek için kapsamlı bir çerçeve sunmaktadır.

Blok zincir ve mikroservisler, modern yazılım mimarilerinde önemli bileşenler olarak kabul edilir. Her biri kendi başına birçok avantaja sahipken, bir araya getirildiğinde sistemlerin güvenilirliği, şeffaflığı ve ölçeklenebilirliği üzerinde önemli bir etkisi olduğu görülür.

Bu tezde sunulan yaklaşımda, belirli sürelerle blok üretimi API Gateway servisi tarafından gerçekleştirilir. Bu, blok zincirin değiştirilemez kayıtlar oluşturma özelliğiyle uyumludur. Öte yandan, gerçekleştirilen işlemler, içsel (internal) servisler tarafından gerçek zamanlı olarak bu bloklara kaydedilir. Bu işlemlerin şeffaf, denetlenebilir ve değiştirilemez bir şekilde saklandığı anlaşılır.

Bu entegrasyonla, her bir mikroservisin kendi işlevselliği sürdürülürken, tüm sistemdeki işlemlerin merkezi bir defterde (blok zincirinde) saklandığı görülür. Hataların veya kötü niyetli aktivitelerin tespiti bu sayede kolaylaştırılır ve sistemdeki her bir işlemin doğruluğu ve bütünlüğü garanti edilir.

Ayrıca, bu yaklaşımın mikroservislerin ölçeklenebilirliği ile blok zincirin güvenilirliği ve şeffaflığı arasında bir denge kurduğu gözlemlenir. Bu, hem performansın hem de güvenliğin optimize edildiği büyük ölçekli dağıtık sistemler için ideal bir çözüm olarak kabul edilir.

Mikroservis mimarisinin işleyişini detaylı olarak anlamak için örnek senaryolar oluşturulmuştur.

Sistemde, mikroservisler iki türe ayrılmıştır. Bunlardan ilki iç servisler ve ikincisi kullanıcı tanımlı servislerdir.

6.1 Mikroservis Keşfi ve Servis Kaydı

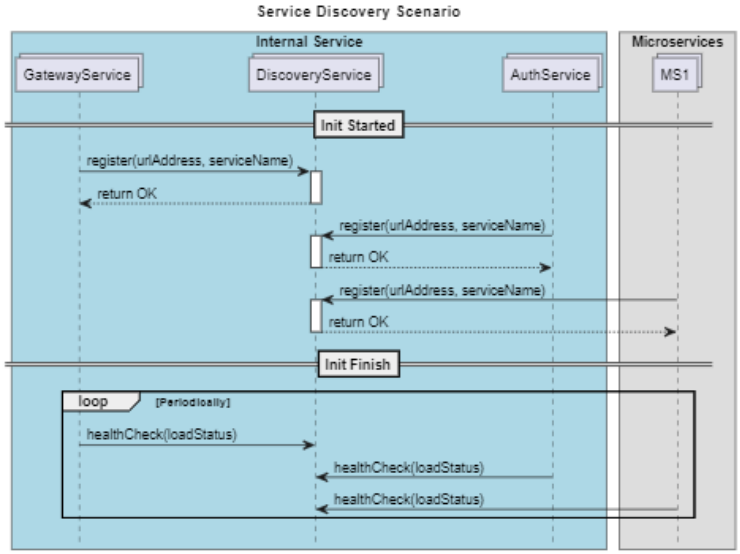
Mikroservis mimarisinde servis keşfinin nasıl çalışabileceğine dair bir örnek Şekil 6.1'da gösterilmektedir.

Bu senaryoda, GatewayService, AuthService ve MS1 adında üç farklı

servis bulunmaktadır. Bu servislerin her biri, DiscoveryService adlı merkezi bir servise kayıt olmaktadır. Kayıt sırasında, her servis kendi URL adresini ve servis adını DiscoveryService'e ileterek kendini tanıtmaktadır.

Kayıt işlemi tamamlandıktan sonra, belirli aralıklarla servislerin sağlık durumları kontrol edilmektedir. Bu, servislerin DiscoveryService'e periyodik olarak sağlık durumları ve yük durumları hakkında bilgi göndermesiyle gerçekleştirilir. Bu sayede, DiscoveryService hangi servislerin aktif olduğunu ve hangi servislerin yük altında olduğunu belirleyebilir.

Bu tür bir yapı, mikroservis mimarisinde servislerin dinamik olarak keşfedilmesini ve servisler arasında esnek bir iletişim kurulmasını sağlar. Ayrıca, servislerin sağlık durumlarının sürekli olarak izlenmesi, olası hataların veya kesintilerin erken tespit edilmesine olanak tanır.



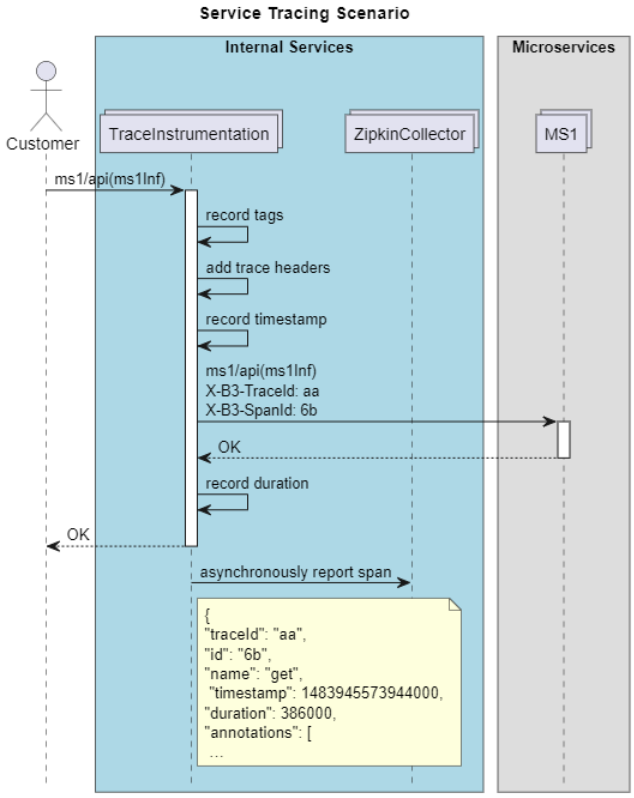
Şekil 6.1: Servis Keşfi Senaryosu Bu diyagram, mikroservis mimarisinde servislerin DiscoveryService adlı merkezi bir servise nasıl kayıt olduğunu ve bu servise periyodik olarak sağlık durumlarını nasıl bildirdiğini göstermektedir.

Bu örnekte, servis kaydı, müşterinin sunucu mikroservisini bulmak için kullanabileceği bir tür "rehber" olarak işlev görür. Müşteri, döndürülen adresi kullanarak sunucu mikroservisi ile doğrudan iletişim kurabilir. Bu, müşteri mikroservisinin, sunucu mikroservisinin yeri zaman içinde değişse bile sunucu mikroservisi ile iletişim kurmasına olanak tanır.

6.2 Mikroservis Servis İzleme

Mikroservis mimarisinde, servisler arası iletişimin sürekli olarak izlenmesi ve bu izleme bilgilerinin toplanması büyük bir öneme sahiptir. Şekil 6.2 bu süreci göstermektedir.

Bu senaryoda, bir Customer (Müşteri), MS1 adında bir mikroservise bir istekte bulunmaktadır. Ancak bu istek doğrudan MS1'e gitmek yerine, önce TraceInstrumentation adlı bir izleme servisine yönlendirilir. Bu servis, isteğin detaylarını kaydeder, iz sürme başlıkları ekler ve zaman damgası ekler. Ardından, bu istek MS1'e yönlendirilir. MS1 isteği başarıyla işledikten sonra, bir yanıt geri gönderir.



Şekil 6.2: Servis İz Sürme Senaryosu Bu diyagram, mikroservis mimarisinde bir isteğin nasıl izlendiğini ve bu izleme bilgilerinin nasıl toplandığını göstermektedir.

TraceInstrumentation servisi, isteğin ne kadar süreyle işlendiğini kaydeder ve bu bilgiyi ZipkinCollector adlı bir toplayıcıya asenkron bir şekilde

raporlar. Bu rapor, iz sürme bilgilerini, zaman damgasını, süreyi ve diğer detayları içerir.

Bu tür bir yapı, mikroservis mimarisinde izleme ve iz sürme işlemlerinin nasıl gerçekleştirildiğini gösterir. İzleme ve iz sürme, servisler arası iletişimin sağlıklı bir şekilde gerçekleşip gerçekleşmediğini kontrol etmek, hataları erken tespit etmek ve performansı optimize etmek için kritik öneme sahiptir.

Müşteri tarafından MS1 mikroservisine yapılan bir istek, öncelikle TraceInstrumentation servisi tarafından ele alınır. Bu servis, isteğin detaylarını, iz sürme başlıklarını ve zaman damgasını kaydeder. Ardından, bu istek MS1 mikroservisine yönlendirilir ve işlenir. İşleme tamamlandığında, TraceInstrumentation servisi, işlemenin ne kadar sürdüğünü kaydeder ve bu bilgiyi ZipkinCollector servisine asenkron bir şekilde raporlar. Bu yapı, servisler arası iletişimin sürekli olarak izlenmesini ve izleme bilgilerinin merkezi bir noktada toplanmasını sağlar.

Mikroservis mimarisinde bir Servis İzleme sunucusu için bir kullanım durumu, aşağıdaki adımları içermektedir:

1. Bir istemci, bir servis uç noktasına, örneğin bir web sayfası veya API uç noktasına bir istek yapar.
2. Servis uç noktası, istek için benzersiz bir izleme kimliği oluşturur ve bunu istek başlıklarına ekler.
3. Servis uç noktası, isteği uygun mikroservise yönlendirir.
4. Mikroservis, isteği işler ve ek izleme verileri oluşturur, örneğin isteğin zaman damgası ve süresi.
5. Mikroservis, bu izleme verilerini Servis İzleme sunucusuna gönderir, istek başlıklarından izleme kimliği ile birlikte.
6. Servis İzleme sunucusu, izleme verilerini bir veritabanında saklar ve istek ve yanıt akışının süresi, her adımın süresi ve zaman damgaları gibi bilgilerle ayrıntılı görselleştirmeler oluşturur.
7. Mikroservis, yanıtı servis uç noktasına geri gönderir, bu da onu istemciye geri gönderir.

8. İstemci yanıtı alır ve izleme verileri, sistemin performansını izlemek ve sorun gidermek için kullanılır.

Bu kullanım durumunun sistem dizin diyagramı, sistemin farklı bileşenlerini ve etkileşimlerini, istemci, servis uç noktası, mikroservisler, Servis İzleme sunucusu ve veritabanı dahil gösterecektir. Şekil 6.2'deki diyagram, isteklerin ve yanıtların akışını ve izleme verilerinin nasıl toplandığını ve sistemin performansını izlemek için nasıl kullanıldığını anlamaya yardımcı olacaktır.

6.3 Mikroservis Devre Kesici

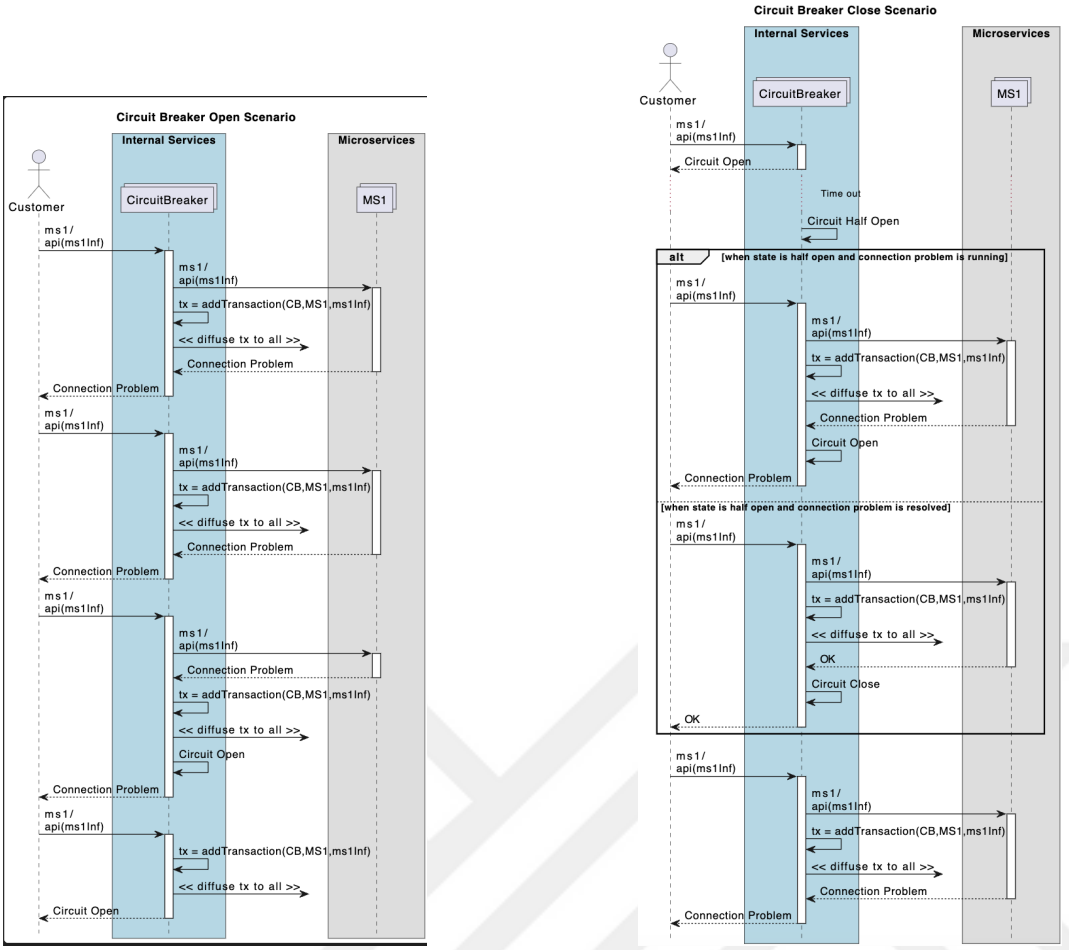
Mikroservis mimarisinde, servisler arası iletişimin sürekli olarak izlenmesi ve hataların erken tespiti büyük bir öneme sahiptir. Şekil 6.3a bu süreci göstermektedir.

Bu senaryoda, bir Customer (Müşteri), MS1 adında bir mikroservise bir istekte bulunmaktadır. Ancak bu istek, CircuitBreaker adlı bir devre kesici servisi tarafından ele alınır. Eğer devre kesici açık durumdaysa, istek doğrudan reddedilir ve "Circuit Open" yanıtı döner.

Belli bir süre sonra, devre kesici "Half Open" (Yarı Açık) durumuna geçer. Bu durumda, devre kesici servisi, MS1 mikroservisine yapılan istekleri denemeye başlar. Eğer bu istekler başarılı bir şekilde işlenirse, devre kesici "Close" (Kapalı) durumuna geçer. Ancak, isteklerde bir problem yaşanırsa, devre kesici tekrar "Open" (Açık) durumuna geçer.

Müşteri tarafından MS1 mikroservisine yapılan istekler, CircuitBreaker adlı devre kesici servisi tarafından ele alınır. Bu servis, isteklerin başarılı bir şekilde işlenip işlenmediğini kontrol eder ve buna göre devre kesicinin durumunu güncellenir. Açık pozisyondan kapalı pozisyona geçiş seneryosu Şekil 6.3b bu süreci göstermektedir.

Bu senaryoda, bir Customer (Müşteri), MS1 adında bir mikroservise bir istekte bulunmaktadır. Bu istek, CircuitBreaker adlı bir devre kesici servisi tarafından ele alınır. Eğer devre kesici kapalı durumdaysa, istek MS1 mikroservisine yönlendirilir. Ancak, MS1 mikroservisinden alınan yanıtta bir



(a) Devre Kesici Açık (b) Devre Kesici Kapalı

Şekil 6.3: Devre Kesici

"Connection Problem" (Bağlantı Problemi) olduğu tespit edilirse, devre kesici bu durumu kaydeder.

Eğer belirli bir süre zarfında belirli sayıda ardışık "Connection Problem" yanıtı alınırsa, devre kesici "Open" (Açık) durumuna geçer ve sonraki istekleri doğrudan reddeder.

Mikroservis mimarisinde bir devre kesicinin nasıl çalışabileceğine dair bir örnek, mesajların çeşitli bileşenler arasında akışını göstermek için bir akış diyagramı aşağıdaki gibi olmaktadır:

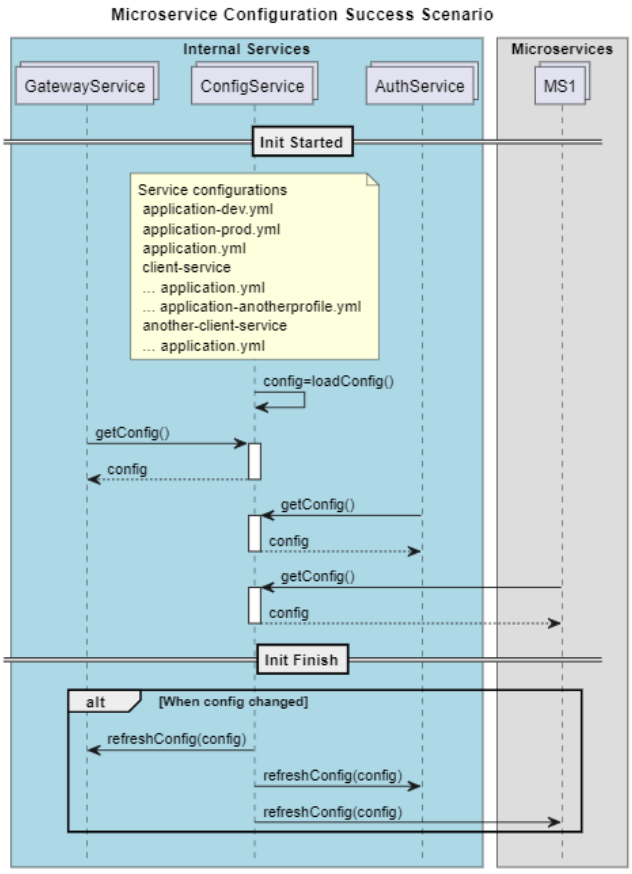
1. Bir kullanıcı, mikroservise bir istek gönderir.
2. Mikroservis, isteği yerine getirmek için sunucu mikroservisi ile iletişim kurması gerekmektedir.
3. Mikroservis, devre kesici proxy'ye bir istek gönderir.
4. Devre kesici proxy, sunucu mikroservisinin sağlıklı olduğuna karar verir ve isteği sunucu mikroservisine yönlendirir.
5. Sunucu mikroservisi, isteği işler ve yanıtı devre kesici proxy'ye döndürür.
6. Devre kesici proxy, yanıtı mikroservise döndürür.
7. Mikroservis, yanıtı kullanıcıya döndürür.
8. Eğer devre kesici "açık" durumdaysa (Circuit Open), kullanıcıya doğrudan bir hata mesajı döndürülür.
9. Belirli bir süre sonra devre kesici "yarı açık" (Half Open) duruma geçer ve gelen istekleri tekrar işlemeye başlar.
10. Eğer devre kesici "yarı açık" durumda ve bağlantı problemi devam ediyorsa, kullanıcıya "Bağlantı Problemi" mesajı döndürülür ve devre kesici tekrar "açık" duruma geçer.
11. Eğer devre kesici "yarı açık" durumda ve bağlantı problemi çözülmüşse, kullanıcıya başarılı bir yanıt döndürülür ve devre kesici "kapalı" (Close) duruma geçer.
12. Devre kesici proxy, her işlemde bir işlem (transaction) oluşturur ve bu işlemi blokzincir (blockchain) üzerine ekler. Bu işlem, devre kesici proxy'den tüm katılımcılara yayılır.

Eğer sunucu mikroservisi başarısız olmaya başlarsa, devre kesici proxy isteği kesintiye uğratabilir ve mikroservise bir hata yanıtı döndürebilir, başarısız olan sunucu mikroservisine isteği yönlendirmek yerine. Bu, mikroservisin hızla başarısız sonucunu dönmesini ve potansiyel olarak servisin yoğunluğunu azaltıp servisin daha hızlı toparlanmasına olanak tanır, başarısız olan sunucu mikroservisinden istemci uygulamanın bir yanıt almak için bekletilmesini engellemektedir.

6.4 Mikroservis Konfigürasyonu

Sistem başlarken servislerin konfigürasyonları okuması için bir senaryo yazılmıştır.

Şekil 6.4 belirtilen diyagramda, "Microservice Configuration Success Scenario" (Mikroservis Konfigürasyon Başarı Senaryosu) başlığını taşımaktadır. Bu diyagram, mikroservis mimarisi kapsamında konfigürasyonun nasıl ele alındığını ortaya koymaktadır. Diyagramda, iç servisler (Internal Services) ve mikroservisler (Microservices) olmak üzere iki temel bölüm bulunmaktadır. İç servisler kısmında uçların kapı servisi (GatewayService), konfigrasyon servisi (ConfigService) ve yetki servisi (AuthService) gibi servisler bulunurken, mikroservisler kısmında MS1 gibi servisler yer almaktadır.



Şekil 6.4: Mikroservis Konfigürasyon Başarı Senaryosu Bu diyagram, mikroservis mimarisinde servis konfigürasyonlarının nasıl yüklendiğini ve güncellendiğini göstermektedir.

Şekil 6.4 belirtilen diyagramda "Init Started" bölümünde, ConfigService'in çeşitli konfigürasyon dosyalarını yüklemekte olduğu belirtilir. İlk aşamada, ConfigService farklı konfigürasyon dosyalarını yükler. Daha sonra, diğer servisler bu konfigürasyonları ConfigService'ten alır. Eğer bir konfigürasyon değişikliği meydana gelirse, bu değişiklikler ilgili servislere ConfigService tarafından bildirilir. Bu dosyalar arasında "application-dev.yml", "application-prod.yml" ve "application.yml" gibi farklı profiller için konfigürasyon dosyaları mevcuttur. Ardından, diğer servislerin (GatewayService, AuthService, MS1) ConfigService'ten konfigürasyon bilgilerini aldığı görülmektedir.

Şekil 6.4 belirtilen diyagramda "When config changed" bölümünde, konfigürasyonun değiştiği bir durum ele alınır. Bu durumda, ConfigService'in

konfigürasyon değişikliklerini diğer servislere (GatewayService, AuthService, MS1) ilettiği gösterilir.

Bu diyagramda, mikroservis mimarisinde konfigürasyon yönetiminin ne denli önemli olduğunu vurgulamaktadır. Mikroservislerin bağımsız bir şekilde işleyebilmesi için her bir servisin kendi konfigürasyonuna sahip olması zorunludur. Ancak, bu konfigürasyonların merkezi bir servis tarafından yönetilmesi, konfigürasyon değişikliklerinin etkin bir şekilde yönetilmesini ve tüm servislere hızla uygulanmasını garantilemektedir.

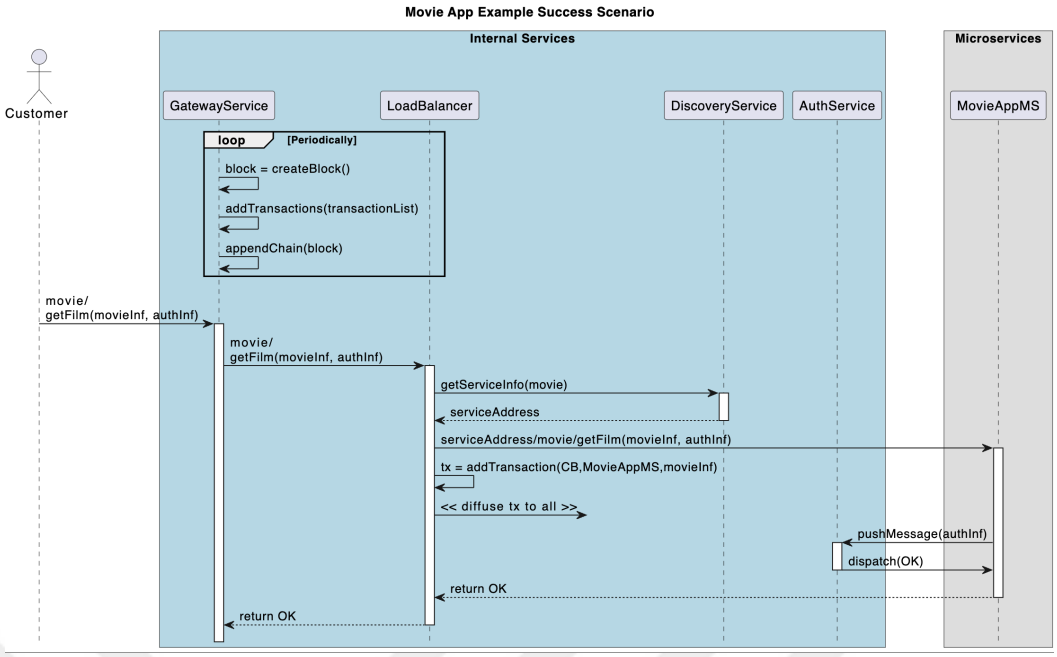
6.5 Müşteri MS1 Örnek Süreci

Bu bölümde, bir müşterinin sisteme kaydolma sürecini gösteren bir mikroservis senaryosunu inceleyeceğiz. Şekil 6.5 bu süreci detaylı bir şekilde göstermektedir.Müşteri, başlangıçta GatewayService'e bir istekte bulunarak süreci başlatır. Ardından, bu istek çeşitli servisler ve bileşenler arasında yönlendirilerek işlenir. Senaryo, mikroservis mimarisindeki bileşenlerin nasıl birbiriyle etkileşimde bulunduğunu ve bir isteğin nasıl işlendiğini detaylı bir şekilde göstermektedir. Diyagramda, yetkilendirme, yük dengeleme, servis keşfi ve veritabanı işlemleri gibi işlemlerin nasıl gerçekleştirildiği açıkça belirtilmiştir.

Müşteri, sisteme kaydolmak istediğinde öncelikle GatewayService'e bir istekte bulunur. Bu istek, hem müşteri bilgilerini (ms1Inf) hem de yetkilendirme bilgilerini (authInf) içerir. GatewayService, bu isteği LoadBalancer'a yönlendirir. LoadBalancer, hangi servisin bu isteği işleyeceğini belirlemek için DiscoveryService'e başvurur. DiscoveryService, istenen servisin (bu durumda MS1) adres bilgisini (serviceAddress) LoadBalancer'a geri döndürür.

LoadBalancer, bu adres bilgisini kullanarak isteği doğru mikroservise (MS1) yönlendirir. MS1, yetkilendirme bilgilerini (authInf) bir mesaj kuyruğuna (MessageQueueBroker) ekler. MessageQueueBroker, bu mesajı AuthService'e yönlendirir. AuthService, yetkilendirme bilgilerini doğrulamak için AuthDB'ye başvurur. Eğer yetkilendirme bilgileri doğruysa, AuthService bu bilgiyi mesaj kuyruğuna geri gönderir.

MS1, yetkilendirme bilgilerinin doğruluğunu aldıktan sonra, müşteri



Şekil 6.5: Müşteri Kayıt Senaryosu Bu diyagram, bir müşterinin sisteme kayıt olma sürecini göstermektedir.

bilgilerini (ms1Inf) kendi veritabanına (MS1DB) ekler. Tüm bu işlemler başarıyla tamamlandığında, müşteriye başarılı bir şekilde kaydedildiğine dair bir geri dönüş yapılır.

Bu senaryo, mikroservis mimarisindeki bileşenler arasındaki etkileşimi ve iş akışını göstermektedir. Özellikle yetkilendirme ve veritabanı işlemleri gibi kritik işlemlerin nasıl gerçekleştirildiğini anlamak için bu tür görselleştirmelere ihtiyaç duyulmaktadır. Ayrıca, bu senaryo mikroservis mimarisinin esnekliğini ve ölçeklenebilirliğini de göstermektedir. Farklı servislerin birbiriyle nasıl etkileşimde bulunduğunu ve bir isteğin nasıl işlendiğini anlamak, sistemin genel işleyişini ve performansını optimize etmek için kritik öneme sahiptir.

7 DURUM ÇALIŞMASI: MİKROSERVİS UYGULAMALARI İÇİN PERFORMANS MODELLEMESİ

Örnek mikroservis uygulaması, deneysel bir konfigürasyonda bir araya getirilmiş dört mikroservis ve bir MongoDB'den oluşmaktadır.(Jindal et al., 2019)

Bu simülasyonda, bir mikroservis tabanlı etmen sisteminin davranışını incelenmiştir. Sistem, belirli bir görevi yerine getirmekten sorumlu birkaç etmen içerir. Bu etmenler, bir dizi mikroservis aracılığıyla birbirleriyle bilgi alışverişinde bulunur ve eylemlerini koordine ederler.

Simülasyon, dağıtık bir ortamda birlikte çalışan birden fazla etmenin gerçek dünya senaryosunu taklit etmek üzere tasarlanmıştır. Etmenlerin farklı rolleri ve sorumlulukları vardır ve ortak bir hedefe ulaşmak için birbirleriyle etkileşimde bulunurlar. Simülasyon günlükleri, sistemin davranışına ve farklı koşullara nasıl yanıt verdiğine dair içgörüler sunar.

Simülasyon, etmenlere destek olan mikroservisleri başlatarak başlatılır. Her mikroservis ayrı bir sunucuda çalışır ve etmenlerden gelen istekleri işlemekten sorumludur. Etmenlar, kendi makinelerinde çalışan ayrı süreçler olarak uygulanır. Bu etmenler, mesaj kuyruğu kullanarak mikroservislerle iletişim kurar.

Simülasyon adımları ilerledikçe, etmenler birbirleriyle etkileşime başlar. Mikroservisler aracılığıyla mesajlar alışverişinde bulunurlar ve bu bilgileri eylemlerini koordine etmek için kullanırlar. Bu noktada, her iletişimde API Gateway tarafından oluşturulan bloğa transactionlar kaydedilir. Bu, blokzincirin sağladığı güvenli ve şeffaf bir kayıt mekanizması sayesinde mümkündür. Blokzincir, tüm işlemlerin doğrulanabilir ve değiştirilemez bir şekilde saklanmasını sağlar. Bu, sistemdeki herhangi bir hata veya kötü niyetli aktivitenin kolayca tespit edilmesine olanak tanır.

Logları incelediğimizde, sistemin başarısızlıklara karşı son derece dirençli

olduğu görülebilmektedir. Bir veya daha fazla etmen başarısız olsa bile, sistem düzgün bir şekilde çalışmaya devam edebilir. Bu, sistemin merkezi olmayan doğasından kaynaklanmaktadır; her etmen kendi görevinden sorumlu olup bağımsız olarak çalışmaya devam edebilir.

Loglar ayrıca sistemin son derece ölçeklenebilir olduğunu da ortaya koymaktadır. Etmen sayısı arttıkça, sistem artan iş yükünü performans üzerinde önemli bir etki olmadan işleyebilir. Bu, iş yükünün birden çok sunucu ve süreç arasında dağıtıldığı sistemin dağıtık doğasından kaynaklanmaktadır.

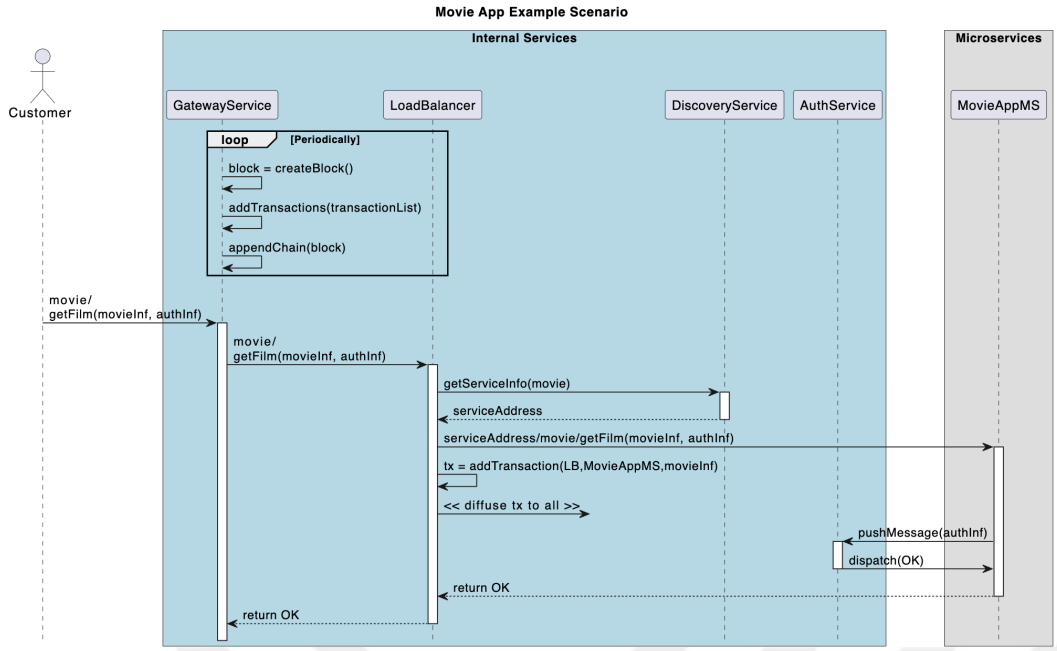
Loglardan başka bir gözlemimiz, sistemin değişen koşullara iyi bir şekilde uyum sağlaması yönündedir. Örneğin, iş yükü arttığında, etmenler davranışlarını ayarlayabilir ve görevlerine daha fazla kaynak ayırabilirler. Bu, iş yükü yüksek olduğunda bile sistemin yüksek performans düzeyini sürdürmesine olanak tanır.

Sonuç olarak, simülasyon, bir mikroservis tabanlı etmen sisteminin davranışına dair bilgiler sunmaktadır. Loglar, sistemin son derece dirençli, ölçeklenebilir ve uyumlu olduğunu göstermektedir. Bu, birden çok etmenin ortak bir hedefe ulaşmak üzere birlikte çalışması gereken dağıtık ortamlar için ideal bir çözüm olduğunu bize göstermektedir.

7.1 MovieApp Uygulaması

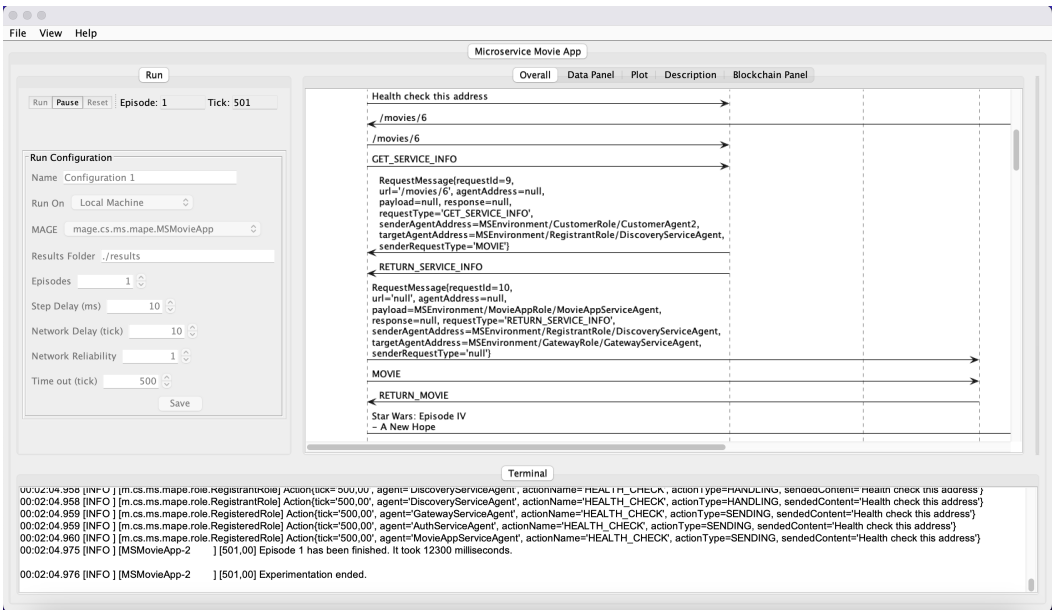
Deneysel uygulama, film uygulamasına bağlı dört mikroservis ve bir InmemoryDB veritabanından oluşmaktadır. Mikroservis mimarileri, kullanıcı taleplerini hızla ve etkili bir şekilde işleyebilmek için servisler arası etkileşimi optimize eder. Bu bölümde, bir film uygulamasının mikroservis mimarisi üzerinde nasıl çalıştığını gösteren Şekil 7.1 akış senaryosunu inceleyeceğiz.

Şekil 7.2 diyagramda, bir CustomerAgent'ın, GatewayServiceAgent aracılığı ile film bilgilerini sorguladığı görülmektedir. Bu sorgulama, DiscoveryService-Agent adlı bir servis keşif etmeni aracılığıyla yönlendirilmekte ve sonuç olarak MovieAppServiceAgent tarafından işlenmektedir.



Şekil 7.1: MovieApp uygulamasının simülasyondan önce mimari önerimize göre tasarlanan diyagramdır.

- Kullanıcı Sorgusu: Kullanıcı, belirli bir film bilgisini sorgular ("/movies/1, /movies/2 vb."). Bu sorgu GatewayServiceAgent'a yönlendirilir.
- Blokların Oluşturulması: GatewayService periyodik olarak yeni bloklar oluşturur, işlemleri ekler ve bu blokları blok zincirine ekler. Bu, blokzincirin sürekli güncellenmesini sağlar.
- Servis Keşfi: GatewayServiceAgent, hangi servisin bu sorguyu işleyeceğini belirlemek için DiscoveryServiceAgent'a bir "GETSERVICEINFO" talebinde bulunur.
- Yük Dengesi: GatewayServiceAgent, sorguyu LoadBalancer'a yönlendirir. LoadBalancer, DiscoveryService'den servis bilgisini alır ve bu bilgiye dayanarak sorguyu ilgili MovieAppMS'ye yönlendirir.
- İşlem Ekleme: LoadBalancer, işlemi işlem listesine ekler ve bu işlemi tüm katılımcılara yayınlar.
- Kimlik Doğrulama: MovieAppMS, kullanıcının kimlik doğrulama bilgile-



Şekil 7.2: Film mikroservisinden veri okuma seneryosun akış diagramı

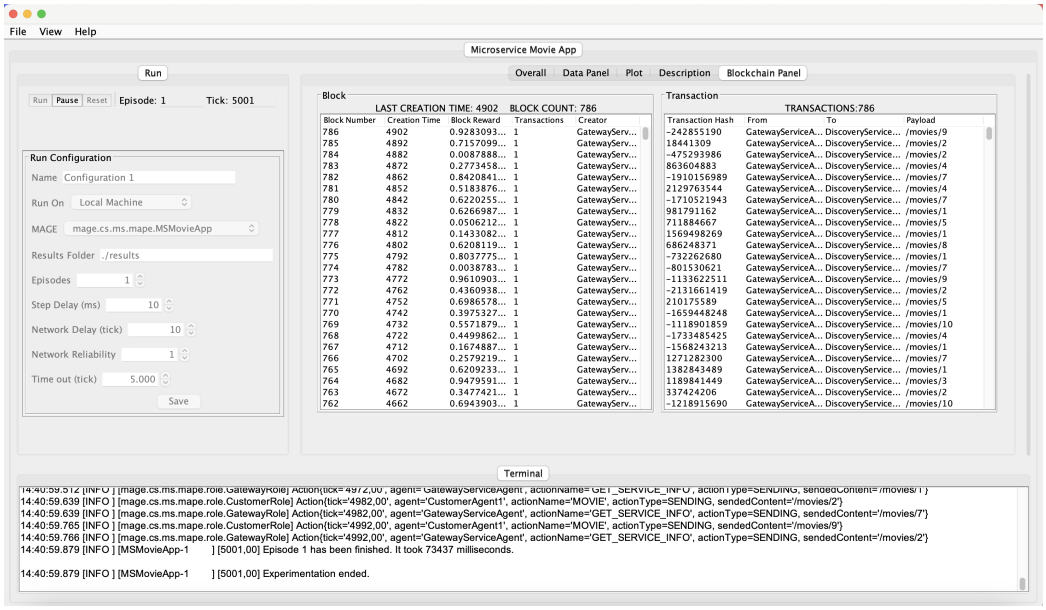
rini AuthService'e gönderir ve doğrulama sonucunu alır.

- Sonuç Dönüşü: MovieAppServiceAgent, sorgulanan film bilgisini GatewayServiceAgent'a geri gönderir. GatewayServiceAgent bu bilgiyi CustomerAgent'a ileterek kullanıcının sorgusuna yanıt verir.

7.2 PrimeApp Uygulaması

Simülasyon, bir müşterinin belirli bir üst sınır içinde asal sayıları hesaplamak üzere Gateway servisine bir istek göndermesiyle başlar. Şekil 7.5 bulunan diyagramı özeti şu şekildedir:

- Blokların Oluşturulması: GatewayService belirli aralıklarla yeni bloklar oluşturur. Bu bloklar, işlem listesinden alınan işlemlerle doldurulur ve daha sonra blok zincirine eklenir. Bu adım, blokzincirin sürekli olarak güncellenmesini sağlar.
- Kullanıcı Sorgusu: Kullanıcı, belirli bir üst sınır için asal sayıları hesaplamak istediğini belirten bir sorgu yapar. Bu sorgu GatewayService'a yönlendirilir.

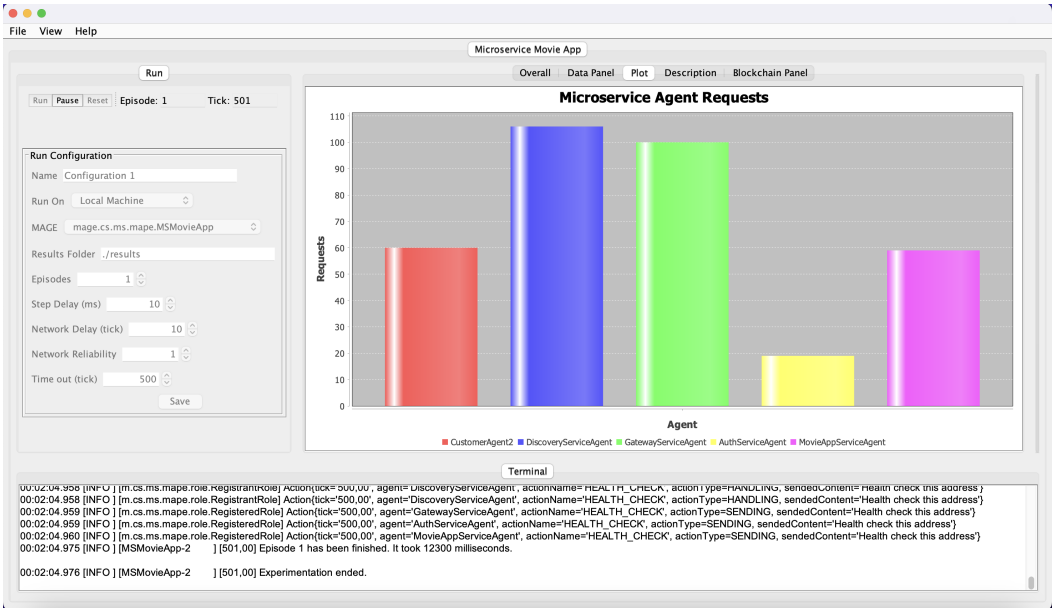


Şekil 7.3: MovieApp uygulamasında blokzincir kullanımı ve oluşturulan işlem ve blokların listelendiği similasyon ekranıdır.

- Yük Dengesi ve Servis Keşfi: GatewayService, sorguyu LoadBalancer'a yönlendirir. LoadBalancer, DiscoveryService'den ilgili servisin adres bilgisini alır.
- Asal Sayı Hesaplama: LoadBalancer, alınan servis adresi üzerinden PrimeAppMS'ye asal sayı hesaplama sorgusunu yönlendirir. Bu sorgu sırasında, LoadBalancer aynı zamanda işlemi işlem listesine ekler ve bu işlemi diğer katılımcılara yayınlar.
- Kimlik Doğrulama ve Sonuç: PrimeAppMS, kullanıcının kimlik doğrulama bilgilerini AuthService'e gönderir. AuthService, kimlik doğrulama işlemini gerçekleştirir ve sonucu PrimeAppMS'ye geri gönderir. Ardından, hesaplanan asal sayılar GatewayService'a geri gönderilir ve bu bilgi kullanıcıya iletilir.

7.3 Webacapp Uygulaması

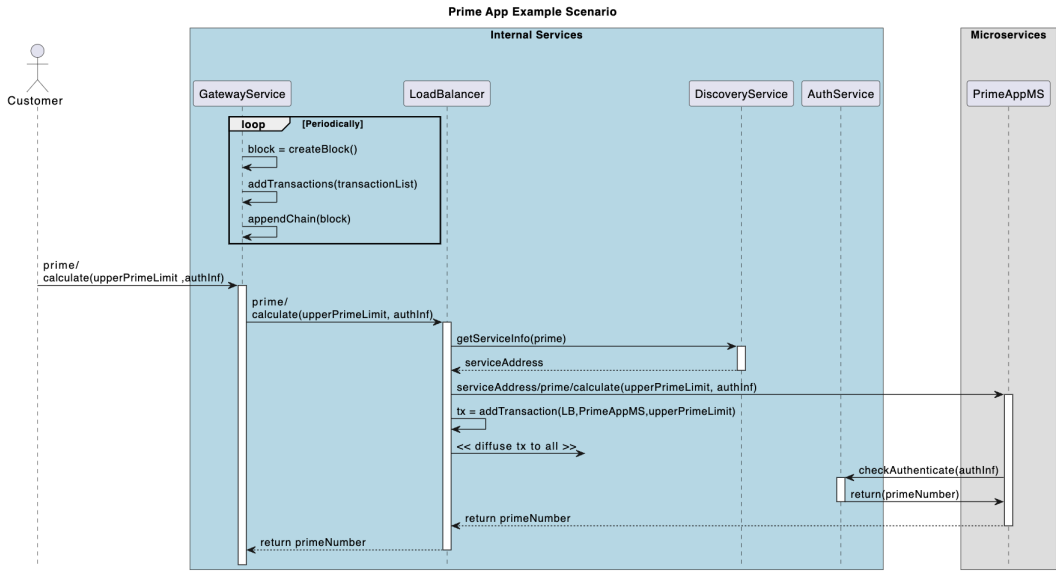
Webacapp, her isteğe anında "merhaba dünya" metni içeren bir mesaj göndererek yanıt vermek üzere tasarlanmıştır.



Şekil 7.4: MovieApp uygulamasında işlem sayısının mikroservis bazında sayıları gösterilmektedir.

Şekil 7.6 bulunan diyagramı özeti şu şekildedir:

- Blokların Oluşturulması: GatewayService belirli aralıklarla yeni bloklar oluşturur. Bu bloklar, işlem listesinden alınan işlemlerle doldurulur ve daha sonra blok zincirine eklenir. Bu adım, blokzincirin sürekli olarak güncellenmesini sağlar.
- Kullanıcı Sorgusu: Kullanıcı, belirli bir web erişim bilgisi ile selamlaşma işlemi yapmak istediğini belirten bir sorgu yapar. Bu sorgu GatewayService'a yönlendirilir.
- Yük Dengesi ve Servis Keşfi: GatewayService, sorguyu LoadBalancer'a yönlendirir. LoadBalancer, DiscoveryService'den ilgili servisin adres bilgisini alır.
- Web Erişim İşleme: LoadBalancer, alınan servis adresi üzerinden WebAccessAppMS'ye web erişim sorgusunu yönlendirir. Bu sorgu sırasında, LoadBalancer aynı zamanda işlemi işlem listesine ekler ve bu işlemi diğer katılımcılara yayınlar.
- Kimlik Doğrulama ve Sonuç: WebAccessAppMS, kullanıcının kimlik



Şekil 7.5: PrimeApp uygulamasının simülasyondan önce mimari önerimize göre tasarlanan diyagramdır.

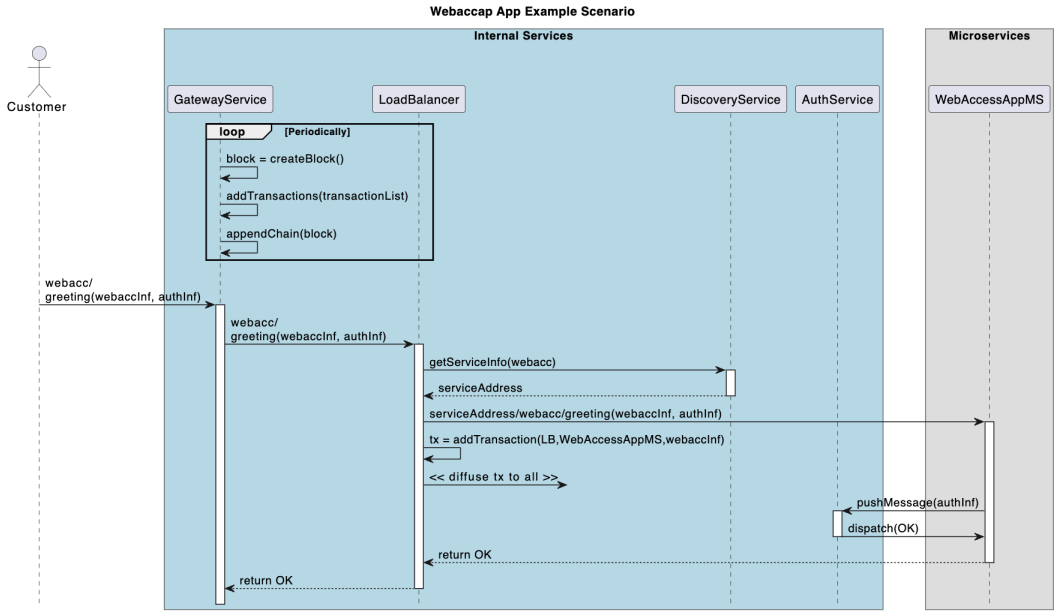
doğrulama bilgilerini içeren veriyi AuthService'e gönderir. AuthService, kimlik doğrulama işlemini gerçekleştirir ve sonucu WebAccessAppMS'ye geri gönderir. Ardından, selamlaşma işlemi sonucu GatewayService'a geri gönderilir ve bu bilgi kullanıcıya iletilir.

7.4 Serveapp Uygulaması

Serveapp, kullanıcılardan gelen istekleri uygun bağımlı mikroservislere (primeapp, movieapp ve webacapp) yönlendirir. Daha sonra, bu mikroservislerden her birinden yanıtları toplar ve bunları tek bir sonuçta birleştirir, ardından kullanıcıya döndürür.

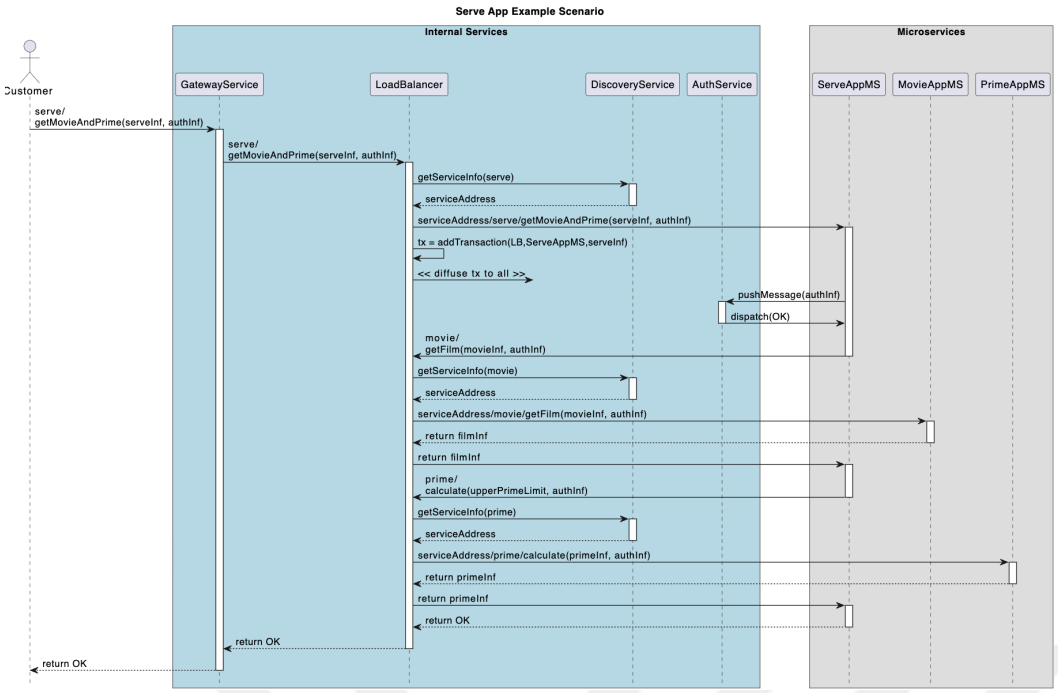
Şekil 7.7 bulunan diyagramı özeti şu şekildedir:

- Kullanıcı Sorgusu: Kullanıcı, belirli bir hizmet bilgisi ile hem film bilgisi hem de asal sayı bilgisi almak istediğini belirten bir sorgu yapar. Bu sorgu GatewayService'a yönlendirilir.
- Yük Dengesi ve Servis Keşfi: GatewayService, sorguyu LoadBalancer'a yönlendirir. LoadBalancer, DiscoveryService'den ilgili servisin adres bilgisini alır.



Şekil 7.6: Webacapp uygulamasının simülasyondan önce mimari önerimize göre tasarlanan diyagramdır.

- ServeAppMS İşleme: LoadBalancer, alınan servis adresi üzerinden ServeAppMS'ye hizmet sorgusunu yönlendirir. Bu sorgu sırasında, LoadBalancer aynı zamanda işlemi işlem listesine ekler ve bu işlemi diğer katılımcılara yayınlar.
- Kimlik Doğrulama: ServeAppMS, kullanıcının kimlik doğrulama bilgilerini AuthService'e gönderir. AuthService, kimlik doğrulama işlemini gerçekleştirir ve sonucu ServeAppMS'ye geri gönderir.
- Film Bilgisi İşleme: ServeAppMS, LoadBalancer üzerinden MovieAppMS'ye film bilgisi sorgusu yapar. LoadBalancer, DiscoveryService'den film servisinin adres bilgisini alır ve bu bilgi MovieAppMS'ye yönlendirilir. Film bilgisi alındıktan sonra ServeAppMS'ye geri döner.
- Asal Sayı Bilgisi İşleme: ServeAppMS, LoadBalancer üzerinden PrimeAppMS'ye asal sayı bilgisi sorgusu yapar. LoadBalancer, DiscoveryService'den asal sayı servisinin adres bilgisini alır ve bu bilgi PrimeAppMS'ye yönlendirilir. Asal sayı bilgisi alındıktan sonra ServeAppMS'ye geri döner.



Şekil 7.7: Serveapp uygulamasının simülasyondan önce mimari önerimize göre tasarlanan diyagramdır.

- Sonuç Dönüşü: Hem film bilgisi hem de asal sayı bilgisi ServeAppMS tarafından işlendikten sonra sonuç GatewayService'a geri gönderilir ve bu bilgi kullanıcıya iletilir.

8 DEĞERLENDİRME

Bu çalışmada, etmen tabanlı bir sistemde mikroservis mimarisi ile blokzincirin nasıl entegre edilebileceği üzerinde durulmuştur. Önder Gürcan'ın "Using Agent-based Modeling and Simulation for Studying Blockchain Systems" başlıklı makalesinde belirtildiği gibi, etmen tabanlı simülasyon, karmaşık etkileşimleri modelleme, dağıtık sistemlerin özelliklerini simüle etme, sosyal ve ekonomik dinamikleri anlama, esneklik ve ölçeklenebilirlik avantajlarını gösterme, dinamik değişikliklere yanıt verme ve detaylı analiz yapma gibi birçok avantaj sunmaktadır.

Bu tezde, API Gateway servisi tarafından belirli aralıklarla blok üretimi gerçekleştirilmektedir. Bu bloklara kayıtların eklenmesi işlemi, gerçekleştirilen örnekte sadece Discovery servisi tarafından yapılmaktadır. Ancak ideal bir uygulamada, bu kayıtların bütün mikroservisler tarafından eklenmesi gerekmektedir.

Gerçekleştirilen bu entegrasyon, blok zincirin değiştirilemez kayıtlar oluşturma özelliğiyle uyumludur. İçsel servisler, gerçekleştirilen işlemleri gerçek zamanlı olarak bu bloklara kaydetmektedir. Bu entegrasyon, her bir mikroservisin işlevselliğini sürdürürken, tüm işlemlerin merkezi bir defterde saklandığı bir yapı sunmaktadır. Bu, hataların veya kötü niyetli aktivitelerin tespitini kolaylaştırır ve işlemlerin doğruluğunu ve bütünlüğünü garanti eder. Bu yaklaşım, mikroservislerin ölçeklenebilirliği ile blokzincirin güvenilirliği ve şeffaflığı arasında bir denge kurar. Bu, büyük ölçekli dağıtık sistemler için ideal bir çözüm olarak kabul edilir. Örnek senaryolar aracılığıyla mikroservis mimarisi işleyişi daha iyi anlaşılmıştır. Sistemde tanımlanan iki tür mikroservis; iç servisler ve kullanıcı tanımlı servisler, bu entegrasyonun nasıl çalıştığını göstermek için kritik öneme sahiptir.

9 SONUÇ

Genel Bakış: Bu çalışma kapsamında, mikroservislerin ve ilgili araçların detaylı bir şekilde incelendiği görülmüştür. Blokzincirin ne olduğu ve nasıl çalıştığına dair temel bilgiler sunulmuş, bu iki teknolojinin bir araya getirilmesi konusundaki literatür değerlendirilmiştir.

Literatürdeki Çalışmalar: İncelenen literatürde, mikroservislerin ve blokzincirin bir araya getirilmesiyle ilgili yapılan çalışmaların, bu alandaki potansiyel uygulama alanlarına ve avantajlarına dikkat çektiği belirlenmiştir. Özellikle bu teknolojilerin birleşiminin nasıl ele alındığına dair derinlemesine incelemeler yapılmıştır.

Önerilen Mimari: Bu tezde, mikroservisler ve blokzincir teknolojilerinin entegrasyonuna yönelik bir mimari önerisi sunulmuştur. Bu mimari, agent base simülasyon kullanılarak gerçekleştirilmiştir. Sunucular arası iletişimin nasıl olabileceği ve blokzincirin nasıl entegre edilebileceği simüle edilmiştir.

Örnek Uygulamalar: Örnek olarak seçilen mikroservis projesi, literatürdeki örneklerle desteklenmiştir. Bu projede, mikroservislerin blokzincirle nasıl bir araya getirilebileceği üzerine kapsamlı bir analiz yapılmıştır.

Sonuç ve Öneriler: Mikroservis mimarisinin blokzinciri ile entegrasyonu, birçok sektörde yeni ve etkili çözümler sunma potansiyeline sahiptir. Bu çalışma, bu entegrasyonun nasıl gerçekleştirilebileceği konusunda önemli bir adım atmıştır. Gelecekteki çalışmaların, bu entegrasyonun daha spesifik uygulama alanlarına odaklanması önerilmektedir.

Kaynaklar

Bakshi, K. (2017). Microservices-based software architecture and approaches. IEEE. http://dx.doi.org/10.1109/aero.2017.7943959

Huhns, M.N. and Singh, M.P. (2005). Service-oriented computing: key concepts and principles. IEEE Internet Computing, 9(1), 75-81. https://doi.org/10.1109/MIC.2005.21

Jawaddi, S.N.A., Johari, M.H., and Ismail, A. (2022). A review of microservices autoscaling with formal verification perspective. Software: Practice and Experience, 52(11), 2476-2495. Wiley Online Library.

Gos, K. and Zabierowski, W. (2020). The comparison of microservice and monolithic architecture. In 2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH). https://doi.org/10.1109/memstech49584.2020.9109514

Welke, R., Hirschheim, R., and Schwarz, A. (2010). Service Oriented Architecture Maturity. *Computer*, *43*(10), 61-67. http://dx.doi.org/10.1109/mc.2010.271

Taibi, D., Lenarduzzi, V., Pahl, C., & Janes, A. (2017). Microservices in Agile Software Development: a Workshop-Based Study into Issues, Advantages, and Disadvantages. Proceedings of the ACM. http://dx.doi.org/10.1145/3120459.3120483

Romhányi, Á. and Vámossy, Z. (2021). Benefits of layered software architecture in machine learning applications. In Proceedings of the International Conference on Image Processing and Vision Engineering. https://doi.org/10.5220/0010424500660072

Rallapalli, M. V. (2011). SOA in enterprise architectures: Overlooked misperceptions, pitfalls and tangible benefits. INCOSE International Symposium, 21(1), 1260-1271. https://doi.org/10.1002/j.2334-5837.2011.tb01283.x

Rotter, C., Illes, J., Nyiri, G., Farkas, L., Csatari, G., and Huszty, G. (2017). Telecom strategies for service discovery in microservice environments. In 2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN). https://doi.org/10.1109/icin.2017.7899414

Singh, N., Hamid, Y., Juneja, S., Srivastava, G., Dhiman, G., Gadekallu, T. R., and Shah, M. A. (2023). Load balancing and service discovery using Docker Swarm for microservice based big data applications. Journal of Cloud Computing Advances Systems and Applications, 12(1). https://doi.org/10.1186/s13677-022-00358-7

Client Side Load Balancer: Ribbon. (2022). https://cloud.spring.io/spring-cloud-netflix/multi/multi_spring-cloud-ribbon.html

OpenZipkin. A distributed tracing system. (2022). https://zipkin.io/

Spring Cloud Config. (2022). https://cloud.spring.io/spring-cloud-config/reference/html/

Using Spring Cloud Gateway with OAuth 2.0 Patterns. (2022). https://www.baeldung.com/spring-cloud-gateway-oauth2

Router and Filter: Zuul. (2022). https://cloud.spring.io/spring-cloud-netflix/multi/multi__router_and_filter_zuul.html

Circuit Breaker: Hystrix Clients. (2022). https://cloud.spring.io/spring-cloud-netflix/multi/multi__circuit_breaker_hystrix_clients.html

Huhns, M. N. and Singh, M. P. (2005). Service-oriented computing: key concepts and principles. IEEE Internet Computing, 9(1), 75-81. https://doi.org/10.1109/MIC.2005.21

Baraki, H., Schwarzbach, C., Fax, M., & Geihs, K. (2018). Architectural Patterns for Microservices: A Systematic Mapping Study. In *CLOSER*

2018: Proceedings of the 8th International Conference on Cloud Computing and Services Science (pp. 136-147). https://doi.org/10.5220/0006701101360147

Soldani, J., Tamburri, D. A., & Van Den Heuvel, W.-J. (2018). The pains and gains of microservices: A Systematic grey literature review. Journal of Systems and Software. http://dx.doi.org/10.1016/j.jss.2018.09.082

Vayghan, L. A., Saied, M. A., Toeroe, M., & Khendek, F. (2019). Microservice based architecture: Towards high-availability for stateful applications with kubernetes. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)* (pp. 176-185). IEEE.

Ruoti, S., Kaiser, B., Yerukhimovich, A., Clark, J., and Cunningham, R. (2017). Blockchain Technology: What Is It Good For?. IEEE Security & Privacy, 15(4), 76-81. https://doi.org/10.1109/MSP.2017.3080831

Roussille, H., Gürcan, Ö., and Michel, F. (2020). AGR4BS: A Generic Multi-Agent Organizational Model for Blockchain Systems. Information, 11(5), 278. https://doi.org/10.3390/info11050278

Ammous, S.H. (2017). Blockchain Technology: What is it Good for?. SSRN Journal. http://dx.doi.org/10.2139/ssrn.2832751

Pierro, M.D. (2017). What Is the Blockchain?. Computing in Science & Engineering, 19(5), 92-95. http://dx.doi.org/10.1109/mcse.2017.3421554

Santos, R., Soares, P., Rodrigues, E., Maia, P. H. M., & Silveira, A. (2023). How blokzincir and microservices are being used together: a systematic mapping study. In *ACM*. http://dx.doi.org/10.1145/3528226.3528371

Jindal, A., Podolskiy, V., and Gerndt, M. (2019). Performance modeling for cloud microservice applications. In Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering, 25-32.

Wang, S., Zhang, X., Yu, W., Hu, K., & Zhu, J. (2020). Smart contract microservitization. In *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)* (pp. 1569–1574). IEEE. https://doi.org/10.1109/COMPSAC48688.2020.00-83

Xu, R., Nikouei, S. Y., Chen, Y., Blasch, E., & Aved, A. (2019). Blendmas: a blockchain-enabled decentralized microservices architecture for smart public safety. In *2019 IEEE International Conference on Blockchain (Blockchain)* (pp. 564–571). IEEE. https://doi.org/10.1109/Blockchain.2019.00078

Dilshan, D., Piumika, S., Rupasinghe, C., Perera, I., & Siriwardena, P. (2020). Mschain: blockchain based decentralized certificate transparency for microservices. In *2020 Moratuwa Engineering Research Conference (MERCon)* (pp. 1–6). IEEE. https://doi.org/10.1109/MERCon48737.2020.9141720

Staples, M., Xu, X., & Weber, I. (2019). *Architecture for blockchain applications*. Springer. https://doi.org/10.1007/978-3-030-03035-3

Wüst, K., & Gervais, A. (2018). Do you need a blockchain? In *Anais... Crypto Valley Conference on Blockchain Technology (CVCBT), 2018, Zug, Switzerland* (pp. 45–54). IEEE. https://doi.org/10.1109/CVCBT.2018.00011

Bandara, E., Liang, X., Foytik, P., Shetty, S., Ranasinghe, N., Zoysa, K. D., & Ng, W. K. (2020). Saas-microservices-based scalable smart contract architecture. In *SSCC* (pp. 228–243).

Stefan, L. (2020). Blockchain technologies and microservices for open learning communities. a software architecture perspective. *eLearning & Software for Education*, 3.

Soares de Sousa, P., Parente Nogueira, N., Celestino dos Santos, R., Maia, P. H. M., & de Souza, J. T. (2020). Building a prototype based on microservices and blockchain technologies for notary's office: an academic experience

report. In *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)* (pp. 122–129). IEEE. https://doi.org/10.1109/ICSA-C49205.2020.00028

Gürcan, Ö. (2020). On using agent-based modeling and simulation for studying blockchain systems. In *JFMS 2020 - Journées Francophones de la Modélisation et de la Simulation*. Cargèse, France. *cea-03134094*.

TEŞEKKÜR

Bu çalışma süresince araştırmalarım sırasında kolaylık gösteren TebArf Teknoloji şirketi yöneticilerimden Olcay Yüce ve Mehmet Zahid Kılıç'a, tezin biçimlenmesinde araştırmalarımda her türlü değerli katkılarını aldığım Dr. Önder Gürcan'a ve bölüm öğretim üyelerinden Dr. Öğ. Gör. Birol Çiloğlugil'e teşekkürü bir borç bilirim.

../../20..

İmzası

Ad Soyad

ÖZGEÇMİŞ

Naim Yürek eğitim hayatına Ege Üniversitesi Bilgisayar Mühendisliği bölümünde lisans eğitimine başlamıştır. 2018 yılında bu prestijli kurumdan mezun olurken, aynı zamanda Semafor Teknoloji'de part-time olarak çalışmaktaydı. Mezuniyet tezi olarak "Pilsiz Algılayıcılar ile Park Yeri Takip Sistemi" üzerine kapsamlı bir çalışma gerçekleştirdi.

Semafor Teknoloji'de görev alırken, Sağlık Bakanlığı'nın API uçlarını çağıran ve veritabanı tasarımını ve yazılımını gerçekleştirdiği "Aile Hekimliği Bilgi Sistemi" projesinde önemli bir rol üstlendi.

Profesyonel kariyerine hızla atılan Naim, Java EE, Oracle veritabanı ve TEB özelinde kullanılan Star Framework üzerinde uygulamalar geliştirdi. Özellikle Java, Spring Framework, React.js gibi teknolojilerde derinlemesine bilgi sahibi oldu. TEB Arf ve TEB gibi bankacılık sektörünün önde gelen kuruluşlarında çeşitli görevlerde bulundu. Bu süre zarfında, mikroservislerle smart key uygulaması, FCR (Full Credit Report) projesi ve Mobil Bireysel İnternet projesi gibi birçok önemli projede aktif roller üstlendi.

Naim'in kariyerindeki bu başarılı projelerin yanı sıra, yazılım dünyasında güvenlik konusunda da sertifikaları bulunmaktadır. Ayrıca, yazılım mimarisi tasarımı, nesne yönelimli programlama ve tasarım desenlerinin uygulanması konularında da derinlemesine bilgiye sahiptir.