ANKARA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

MİKROSERVİS MİMARİSİ İÇİN VERİ TAŞINMASI

Ibrahim Bakarr JALLOH

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

ANKARA
2020

ÖZET

Yüksek Lisans Tezi

MİKROSERVİS MİMARİSİ İÇİN VERİ TAŞINMASI

Ibrahim Bakarr JALLOH

Ankara Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Dr. Öğr. Üyesi Ömer Özgür TANRIÖVER

Mikroservis mimarisinin; daha iyi sürdürülebilirlik, pazara ve üretime daha kısa sürede ulaştırma, ölçeklenebilirlik ve çok dilli programlama gibi önemli avantajları ile ortaya çıkması yazılım endüstrisinin tercihlerini mikroservis mimarisi lehine yönlendirmiştir. Bununla birlikte, monolit uygulamasının bir mikroservis uygulamasına dönüştürülmesi ile ilgili yerleşik bir metodoloji eksikliği vardır. Bu nedenle, bu tez çalışması, monolitik uygulamaların mikroservis uygulamalarına dönüştürülmesi ve veri taşınması için bir metot sunmaktadır. Önerilen metodu doğrulamak için bir vaka çalışması yapılmıştır. Bu vaka çalışması mevcut monolitik bir kütüphane sisteminin RESTful bitiş noktaları üzerinden iletişim kuran üç bağımsız hizmetle mikroservis tabanlı bir mimariye dönüştürülmesini içermektedir. Vaka çalışmasında NoSQL veritabanı teknolojisine sahip hayali monolitik bir kurumsal kütüphane sistemi ele alınmıştır. Önerilen yöntem, dönüşüm süreci boyunca kullanılmıştır. Süreç, monolitik uygulamanın parçalanması, mikroservis uygulaması için bir mimari tasarlanması, kodlama ve başarılı testlerinden oluşmaktadır. İşlem, verilerin herhangi bir veri tutarsızlığı olmadan monolitik uygulamadan mikroservis mimarisine taşınması ile sona ermiştir.

Mayıs 2020, 81 Sayfa

Anahtar Kelimeler: mikroservis mimarisi, monolitik mimari, veri, taşı

ABSTRACT

Master Thesis

DATA MIGRATION FOR MICROSERVICE ARCHITECTURE

Ibrahim Bakarr JALLOH

Ankara University
Graduate School of Natural Applied Sciences
Department of Computer Engineering

Supervisor: Asst. Prof. Dr. Ömer Özgür TANRIÖVER

The emergence of microservice architecture has its significant advantages; such as better maintainability, shorter time to market, scalability and allowing polyglot programming. For this reason it have driven the preferences of software industry in favour of Microservice Architecture. However, there seem to be a lack of an established methodology, related to transformation of monolith application to a microservice application. Therefore, this thesis work presents a proposed methodology for the transformation and data migration of monolithic applications to microservices applications. A case study was conducted to validate the proposed methodology. It involved the transformation of an existing monolithic library system to a microservice-based architecture with three independent services that communicate via RESTful endpoints. The case study was a fictitious monolithic enterprise library system with a NoSQL database technology. The proposed methodology was used throughout the transformation process. The process included breaking down of the monolithic application, designing an architecture for the microservice application, coding, and successful testing. The process ended with the migration of data from the monolithic application to the microservice architecture without any data inconsistencies.

May 2020, 81 Pages

Key Words: microservice architecture, monolithic architecture, data, migration

TEŞEKKÜR

Bana verdiği tüm nimetler için Yüce Allah'a minnettarım. Teşekkür ederim Allah'ım.

Ankara Üniversitesi-Bilgisayar Mühendisliği bölümünden danışmanım Dr. Öğretim Üyesi Ömer Özgür Tanrıöver'e, beni mikroservis mimarisi ile tanıştırdığı ve bu kritik yolculuk boyunca sabırla yol gösterdiği için minnettarım. Teşekkür ederim.

Bu tez çalışmasını İngilizce'den Türkçe'ye titizlikle çeviren Sinem Sahin'e teşekkür ediyorum.

Bu güzel ülkede eğitim almam için bana kapsamlı bir burs verdiği için Türk Hükümetine minnettarım. Teşekkürler, YTB

Bu çalışma dönemi boyunca hayallerimi sürdürmem konusunda beni teşvik edip desteklediği ve evdeki yokluğumu tolere ettiği için aileme minnettarım, teşekkürler sevgili ailem.

Bu tez çalışmasını anneme adıyorum. Bana eğitim kapısını gösterdiğin ve araladığın için çok teşekkür ederim. Sen olmadan yapamazdım, teşekkürler Neneh.

Ibrahim Bakarr JALLOH

Ankara, Mayıs 2020

İÇİNDEKİLER

KISALTMALAR DİZİNİ

SOA	Hizmet Odaklı Mimari
HTTP	Hiper Metin Aktarım Protokolü
REST	Temsilcisi Devlet Transferi
RQ	Araştırma Sorusu
ORDBMS	Nesne İlişkisel Veritabanı Yönetim Sistemleri
OODBMS	Nesneye Dayalı Veritabanı Yönetim Sistemleri
NoSQL	İlişkisel Olmayan
SQL	Yapısal Sorgu Dili
WSDL	Web Hizmeti Açıklama Dili
XML eXtensible	İşaretleme Dili
SOAP	Basit Nesne Erişim Protokolü
API	Uygulama Programlama Ara yüzü
JAD	Ortak Uygulama Geliştirme
2PC	İki Fazlı Taahhüt

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

1. GİRİŞ

1.1 Motivasyon

Günümüz iş ve organizasyon ortamlarında yazılımın rolü ve yazılım teknolojisinin dijital dönüşümdeki etkisi önemlidir (Duarte ve Bank, 2016)(Ebert ve Duarte, 2018). İşletmelerin dijitalleştirilmesinde yazılım mühendisliğine olan ihtiyaç, yazılım geliştirme için mükemmel bir çözüm arama ihtiyacını devam ettirmiştir. Bu durum yazılım mimarisinde, monolitik mimaride, Servis Odaklı Mimaride (SOA) ve alanın yeni gelişmekte olan metodu, mikroservis mimarisinde görülmüştür.

Günümüzde hem akademik anlamda hem de yazılım endüstrisinde kabul görmüş mikroservis mimarisi yazılım geliştirmede tercih edilen mimarilerden biri haline gelmiştir. Mikroservis mimarisinin performans (Heinrich vd., 2017)(Niu, Liu, ve Li, 2018) güvenlik (Yarygina ve Bagge, 2018)(Yu, Jin, Zhang, ve Zheng, 2017) noktasındaki kalite faktörleri hakkında birçok çalışma bulunmaktadır. Literatürde monolitik uygulamaların mikroservis mimarilerine farklı nedenlerle dönüştürülmesi konusunda kayda değer birçok çalışma vardır (Knoche ve Hasselbring, 2019). Buna karşın, monolitik uygulamaların kullandığı verilerin hareketine yönelik çabaların geri planda kaldığı görülmektedir (Di Francesco, Lago, ve Malavolta, 2018). Özellikle, mevcut eski verilerin monolitik bir uygulamadan mikroservis tabanlı bir uygulamaya nasıl taşınacağı üzerine çalışmalar literatürde sınırlıdır. Bu nedenle, bu tez çalışmasındaki motivasyon, eski bir monolitik uygulamayı ve verilerini mikroservis mimarilerine taşınması konusuna odaklanmaktır.

1.2 Problem Tanımı

Bu tez çalışmasında, mevcut uygulama ve verilerini mevcut bir monolitik uygulamadan mikroservis tabanlı bir uygulamaya veri taşınması sorunu ele alınmıştır (Di Francesco vd., 2018).Bu nedenle, bu tezin problem açıklaması tam olarak: Eskiden kalan uygulamalar ve verilerinin mikroservis mimarisine taşımanın nasıl gerçekleştirileceğidir.

1.3 Amaç

Bu tez çalışmasının amacı monolitik uygulamalardan mikroservis uygulamalarına veri taşıma problemi ile karşı karşıya olan mühendisler için bir çözüm sunmaktır.

1.4 Kapsam

Bu tezin çalışması, monolitik eskiden kalan uygulamanın ve verilerinin mikroservis tabanlı uygulamalara taşınması sorunudur; bundan fazlası değildir. Verileri kullanan veya işleyen uygulamayı dikkate almadan verileri taşımak imkansız değildir fakat zor olduğunu belirtmek gerekmektedir. Bu nedenle, bu tezdeki çalışma hem monolitik uygulamanın hem de mevcut eskiden kalan verilerin taşınmasını içermektedir. Buna ek olarak, eski monolitik uygulamanın ve verisinin mikroservis tabanlı uygulamalara taşınması için gerekli fizibilite çalışması ve diğer organizasyonel problemler ve bu tür sorunların çözümü bu tezin kapsamı dışındadır.

1.5 Anahat

Bu tez çalışmasının geri kalanı şu şekildedir: ikinci bölümde – problem tanımı ile ilgili literatür, ilgili arka plan ve temel kavramlar sunulmuş ve tartışılmıştır. Üçüncü bölümde, bu çalışmayı yürütmek için kullanılan araştırma yöntemi sunulmuştur. Dördüncü bölümde, bulguları doğrulamak için kullanılan örnek bir olay sunulmuştur. Son olarak beşinci bölümde, benzer bilimsel bir çalışmanın sonucu ile yaptığım çalışmanın sonucunun karşılaştırması sunulmuştur.

2. KAYNAK TARAMASI VE ÖZETİ

2.1 Giriş

Yazılım ürünlerini etkin ve verimli bir şekilde geliştirme ve sunma hedefi, mevcut yöntemleri ve araçları geliştirme çabasını sürekli olarak güdülemiştir. Mikroservis mimarisi (Jamshidi, Pahl, Mendonca, Lewis, ve Tilkov, 2018)(Di Francesco vd., 2018)(Alshuqayran, Ali, ve Evans, 2016), yazılım sistemlerini geliştirmek için hem akademide hem de endüstride yaygınlaşan yeni bir yaklaşımdır.

Geleneksel mimariden farklı olarak - (monolitik ve katmanlı) (Gouigoux ve Tamzalit, 2017) - mikroservis mimarisi, bir uygulamanın belirli bir grup fonksiyonu sağlamaya adanmış hizmetlere bölünmesini gerektirmektedir. Bu bireysel hizmetler, diğer hizmetlerden bağımsız olarak geliştirilip dağıtılabilir ve diğer mikroservislerin sağladığı hizmetlere hafif iletişim mekanizmaları (HTTP RESTful Web hizmetleri [(Rodriguez, 2008)(Pautasso ve Wilde, 2010)) aracılığıyla erişebilir. Mikroservis mimarisinin geleneksel yöntemden farklı olarak birçok alanda faydaları vardır. Bu faydalardan bazıları: yüksek ölçeklenebilirlik, sürdürülebilirlik, pazara kısa sürede ulaşım, çok dilli geliştirme teknolojileridir. Mikroservis mimarisinin sağladığı bu faydalar bu alanın gelişmesinin ve ilerlemesinin en önemli etkenleridir(Knoche ve Hasselbring, 2019).

Mikroservis mimarisi ile ilgili çalışmalar sıklıkla literatürde yer almaktadır. Bu tür çalışmalar mikroservis temelli bir uygulamanın geliştirilmesini, mevcut monolitin modernizasyonunu, endüstriyel kabul ve zorlukları ile mikroservisin ortaya çıkan bilgi işlem alanlarına uygulanmasını incelemektedir. Literatür (Escobar vd., 2016)'te yapıldığı gibi mevcut uygulamanın dönüşümü ile ilgili çalışmalar sunmaktadır; monolitik uygulamanın dönüşümü için mevcut yöntem ve teknikler (Kazanavicius ve Mazeika, 2019)'te sunulmaktadır. Ayrıca bazı çalışmalarda monolitik mimariler ile mikroservis mimarisi arasındaki performans farkları(Al-debagy ve Martinek, 2018)'da sunulmaktadır. Mikroservis mimarisi tabanlı

sistemlerle ilgili avantajlar ve dezavantajlar (Soldani, Tamburri, ve Van Den Heuvel, 2018)'de tartışılmıştır. Bununla birlikte, mikroservis mimarisine veri taşınması konusu, mevcut mikroservis çalışmalarında çok az ilgi görmektedir veya hiç dikkate alınmamaktadır (Di Francesco vd., 2018).

Veri, günümüz iş ortamının en değerli varlığı olmasa da en önemlilerinden biridir (Khatri, 2020). Yazılım ve bilgi sistemleri veriyi yakalamak, işlemek ve saklamak için geliştirilmekte ve kurulmaktadır. Bu nedenle, bu tez, monolitik bir uygulamayı ve verilerini mikroservis mimarisine taşınması ile ilgili bilginin mevcut durumunu incelemektedir.

Tezin geri kalanı şu şekilde organize edilmiştir: ikinci bölümde araştırma metodolojisi sunulmuştur, üçüncü bölüm araştırma sorularına yönelik sonuçları kapsamaktadır**,** dördüncü ve beşinci bölümler sırasıyla sonuçlar üzerine tartışmayı ve çıkarımları içermektedir.

2.2 Araştırma Yöntemi

Bu tez, (Kitchenham ve Charters, 2007)'da belirtilen ve yazılım mühendisliğinde literatür taramasının nasıl yapılacağına dair bir kılavuz olan yazılım mühendisliği yönergelerini izlemiştir.

Çizelge 2.1 (Schön, Thomaschewski, ve Escalona, 2017)'de kullanılan referansların bir özetini sunmaktadır.

Çizelge 2.1 Kaynak taraması için yazılım mühendisliği kılavuzlarının özeti

1. Planlama	2. Uygulama	3. Raporlama
İnceleme ihtiyacının belirlenmesi	Araştırma	Sonuçları çıkarma ve tartışma
İnceleme sorusunun özelleştirilmesi	Çalışma seçimi	Rapor yazma
Araştırma sorusunu oluşturma	Kalite değerlendirmesi	Format raporu
İnceleme Protokolü'nün değerlendirilmesi	Veri çıkarma ve analizi	Değerlendirme raporu

2.2.1 Amaç ve araştırma soruları

Bu kaynak incelemesinin amaçlarından biri, mikroservis mimarisi ve veri/veritabanı taşınması konusundaki mevcut bilgi durumunu sunmaktır.

Arama dizelerinin formülasyonu araştırma sorularını yansıtmıştır, böylece her bir dizeyi araştırma sorularının genel hedeflerine hizalamıştır. Araştırma soruları ve her sorunun arkasındaki motivasyon için **çizelge 2.2**'ye bakılmalıdır. **Çizelge 2.3**, arama aşamasında kullanılan anahtar kelimeleri ve kategorilerini göstermektedir.

Boolean OR ve AND benzer sonuçlar üretmek ve sonuçları araştırma sorularıyla sınırlandırmak için kullanılmıştır.

- (Mikroservis VEYA mikroservis mimarileri) VE yazılım geliştirme
- (Veri taşınması VEYA veritabanı taşınması) VE (mikroservis VEYA mikroservis mimarisi)
- (Veri taşınması VEYA veritabanı taşınması) VE bulut bilişim
- (Veri taşınması VEYA veritabanı taşınması) VE yazılım geliştirme/mühendislik

- (Eskiden kalan verinin taşınması VEYA eskiden kalan veritabanının taşınması) VE bulut bilişim
- (Eskiden kalan verinin VEYA eskiden kalan veritabanının taşınması) VE yazılım geliştirme/mühendislik

Çizelge 2.2 Araştırma Soruları ve Soruların Motivasyonları

Araştırma Sorusu	**Amaç**
S1. Mikroservis mimarisi ve taşınmasına dair mevcut eğilimler nelerdir?	Buradaki amaç mikroservis mimarileri içindeki alt başlıkları incelemektir.
S2. Veritabanı taşınması için mevcut yöntemler nelerdir?	Bu soru, veri ve veritabanı taşınmasında kaydedilen ilerlemeyi incelemektedir.

Birincil ve İkincil Araştırma

Birincil araştırma, online kütüphaneler ve Web arama motorları aracılığıyla elektronik kaynaklara başvurulmasını içermektedir.

İkincil araştırma ise araştırma sürecinin ilk aşamasında oluşturulan referansları ve alıntıları incelemektedir.

Çizelge 2.3 Araştırma anahtar kelimeleri ve kategorileri

Kategori	Anahtar Kelimeler
Mikroservis Mimarisi	Mikroservisler, mikroservis mimarisi, mikroservisteki eğilimler, yazılım geliştirme, yazılım mühendisliği
Veri/Veritabanı Taşınması	Veri taşınması, veritabanı taşınması, yazılım geliştirme, yazılım mühendisliği

2.2.2 Çalışma seçimi

Seçim stratejisi, dahil etme-hariç tutma kıstasları ile beraber kalite değerlendirme kriterlerini de içine alan iki aşamalı bir süreci içermektedir. Strateji, bu tezde kullanılan her bir makaleyi kapsamaktadır. Mikroservis mimarisi üzerine yapılan ilk arama 221 sonuç vermiştir. Bu araştırma için belirlenen yıl aralığı 2014 ve 2019'dur. Bu yıl aralığı, mikroservis teriminin son literatürde ifade ettiğinden farklı bir anlamda kullanılabileceği düşüncesine dayanarak belirlenmiştir.

Veri taşınması araştırması alanında, ilki 1996'da ve sonuncusu makale 2018'de yayımlanan 737 makale ortaya çıkmıştır. **Çizelge 2.5** bu tezde referans alınan makaleleri taramak için kullanılan dahil etme ve hariç tutma kriterlerini sunmaktadır.

Çizelge 2.4 Seçilmiş yayınlar (Riaz, Mendes, ve Tempero, 2009)

YAY. NO	MAKALE	ALAN	YIL
SP1	Servis Bilişim İçin Mikroservis Mimarisi Tabanlı Bulut Bilişim Dağıtımı	Bulut Bilişim	2016

Çizelge 2.4 Seçilmiş yayınlar (Riaz, Mendes, ve Tempero, 2009)(devamı)

SP2	Bulut Uygulamaları İçin Taşıyıcı Tabanlı Mikroservis Mimarisi	Bulut Bilişim	2017
SP3	Bulut Mikroservisinin Mimarisel Çıkarımları	Bulut Bilişim	2018
SP4	Bulut Ortamı Proje Geliştirmede Mikroservis Mimarisi Uygulaması	Bulut Bilişim	2018
SP5	Monolitik ve Mikroservis Mimari Desenlerinin Buluttaki Web Uygulamalarına Dağıtımının Değerlendirilmesi	Bulut Bilişim	2015
SP6	Mikroservis Mimarisi ile Nesnelerin İnterneti Ortamında Akıllı Bir Şehir Tasarlama	Nesnelerin İnterneti	2015
SP7	Mikroservis Mimarisine Dayanan Açık Bir Iot Sistemi	Nesnelerin İnterneti	2017
SP8	Web Nesnelerindeki Mikroservislerin Iot Çevrelerine Yeniden Kullanabilirliği Arttırması İçin Olanak Sağlaması	Nesnelerin İnterneti	2018
SP9	Nesnelerin İnterneti İçin Mikroservisin Geliştirilmesi	Nesnelerin İnterneti	2018
SP10	Mikroservis Mimarilerinde Güvenlik Sorunlarının Üstesinden Gelmek	Güvenlik	2018
SP11	Uygulamalar İçin Etkinleştirilen Mikroservislerin Hizmet İletişimindeki Güvenlik Sorunları Üzerine Bir Anket	Güvenlik	2017
SP12	Mikroservislerde Yük Dengeleme	Performans	2018

Çizelge 2.4 Seçilmiş yayınlar (Riaz, Mendes, ve Tempero, 2009) (devamı)

SP13	Mikroservis Araştırmaları İçin Performans Mühendisliği: Araştırma Zorlukları ve Yönlendirmeleri	Veritabanı	2017
SP14	Mikroservis Mimarisinde Veri Tutarlılığını Sürdürme Yöntemi	Yazılım Mimarisi	2018
SP15	Mikroservisin Edinimi İçin Sürücüler ve Engeller-Almanya'daki Profesyoneller Arasında Bir Anket	Bulut Bilişim	2019
SP16	Sunucusuz Bilişim: Mikroservis Performansını Etkileyen Faktörlerin İncelenmesi	Bulut Bilişim	2018
SP17	SQL Veritabanının Nosql'e Dönüştürme Şema Modeli	Veritabanı	2014
SP18	Nosql Sütunsal Veritabanları Arasında Uyumlu Veri Taşınması	Veritabanı	2014
SP19	İlişkisel Veri Kümelerinin Nosql'e Taşınması İçin Bir Çerçeve	Veritabanı	2015
SP20	İlişkiselden Nesneye Dayalı Veritabanlarına Taşınma	Veritabanı	1996
SP21	İlişkisel ve Nesneye Dayalı Veritabanları Arasında Veri Taşınması Desteklenmesi	Veritabanı	2000
SP22	İlişkisel Veritabanı Taşınması: Bir Görüş	Veritabanı	2008
SP23	Anlamsal Zenginleştirme: İlişkisel Veritabanı Taşınmasının İlk Aşaması	Veritabanı	2010
SP24	Verilerin İlişkisel Veritabanından (RDB) Nesne İlişkisel (ORDB) Veritabanına Taşınması	Veritabanı	2013
SP25	Cmotion: Uygulamaların Bulutlar Arasında Taşınması İçin Bir Çerçeve	Bulut Bilişim	2011
SP26	Bulut Veritabanları Arasında Veri Taşınması Sağlamak İçin Örüntü Modelleri	Bulut Bilişim	2012

Çizelge 2.4 Seçilmiş yayınlar (Riaz, Mendes, ve Tempero, 2009) (devamı)

SP27	Uygulamalı Veritabanı Katmanının Buluta Taşınması İçin Karar Desteği	Bulut Bilişim	2013
SP28	Eskiden Kalan Bilgi Sisteminin Taşınmasına Genel Bakış	Veritabanı	1997
SP29	Heterojen Veritabanında Veri Taşınması Sisteminin Tasarımı ve Uygulaması	Veritabanı	2010
SP30	Nesne İlişkisel Veritabanından Dönüşüm	Veritabanı	2003
SP31	Verilerin İlişkisel Veritabanından (RDB) Nesne İlişkisel (ORDB) Veritabanına Taşınması	Veritabanı	2013

Çizelge 2.5 Dahil etme ve hariç tutma kriterleri (Schön, Thomaschewski, ve Escalona, 2017)

Dahil etme	Hariç tutma
Aşağıdakileri içeren bir yayın:	Aşağıdakileri içermeyen bir yayın:
Mikroservis mimarisinin bir yazılım geliştirme paradigması olarak tanımlanması	Yazılım geliştirme paradigması olarak mikroservis mimarisine odaklanılması
Mikroservis mimarisi ile doğrudan ilgili alt konuların bulunması	Raporun tüm metnini kullanılabilir yapılması
	Çalışmayı bildirmek için İngilizcenin kullanılması

(Lewis, James; Fowler, 2014)'de belirlenmiş kalite kriterleri bu çalışma için yeniden tanımlanmıştır. Bu kriterler, seçilen her bir makalenin araştırma sorularıyla ilgili bilgiler

sunduğundan ve bu incelemenin kalite ihtiyaçlarını yansıttığından emin olmak için dikkatle uyarlanmıştır.

Bu çalışmaya dahil edilen her yayın **çizelge 2.6**'da belirtilen üç kritere göre değerlendirilmiştir. Her yayın için en yüksek puan dörttür, en düşük puan ise ikidir. Minimum puanı karşılamayan yayınlar bu çalışmaya dahil edilmemiştir. **Çizelge 2.7,** çalışmaya dahil edilen her bir makalenin puanını özetlemektedir.

Çizelge 2.6 Kalite değerlendirme kriterleri (Schön vd., 2017)

Soru	Değerlendirme sorusu	Skor	Tanımlama
QA1	Yayın, yaklaşımın tanımını detaylı bir şekilde sunar mı?	-1	Hayır, detaylar mevcut değildir.
		0	Kısmen, eğer biri bu yaklaşımı kullanmak isterse referansları okuması gerekmektedir.
		1	Evet, yaklaşım sunulan detaylarla kullanılabilir niteliktedir.

Çizelge 2.6 Kalite değerlendirme kriterleri (Schön vd., 2017)(devamı)

QA2	Yayın önyargılı bir bakış açısı içerir mi?	-1	Evet, içermektedir.
		0	Kısmen; ilgili çalışma açıklanmış ve makale daha özel bir içerik üzerine kurulmuştur.
		1	Hayır, yayın araştırmaya dayandırılmıştır.
QA3	Yayın, çalışmanın amacını açık bir şekilde ifade ediyor mu?	-1	Hayır, hedefler açıkça tanımlanmamıştır.
		0	Kısmen; hedefler tanımlanmış ama açık nitelikte değildir.
		1	Evet, hedefler açıkça tanımlanmıştır.

Çizelge 2-7 Seçilmiş yayınların skoru (Riaz vd., 2009)

YAYIN NO	SKOR	YAYIN NO	SKOR
SP1	4	SP17	4
SP2	4	SP18	4
SP3	4	SP19	4

Çizelge 2.7 Seçilmiş yayınların skoru (Riaz vd., 2009) (devamı)

SP4	2	SP20	4
SP5	4	SP21	4
SP6	4	SP22	4
SP7	4	SP23	4
SP8	4	SP24	4
SP9	2	SP25	4
SP10	3	SP26	4
4SP11	4	SP27	4
SP12	3	SP28	4
SP13	4	SP29	4
SP14	2	SP30	4
SP15	4	SP31	4
SP16	4		

2.2.3 Veri özütleme ve analizi

Bu tezde veri analizi çıkarmak, kaydetmek ve gerçekleştirmek için Mendeley, MS Word ve Excel uygulama ve yazılımları kullanılmıştır. Dahil edilen veriler aşağıda listelenmiştir:

- Başlık
- Öz
- Yazar (lar)
- Alıntılar
- Şekiller ve çizelgeler

2.3 Araştırma Sorularına Cevaplar

Sonuçlar ve Katkılar

AS1: Mikroservis mimarisi ve taşınmasına dair mevcut eğilimler nelerdir?

Mikroservis, yazılım mühendisliğinde yeni bir paradigmadır. Terim 2012 yılında (Lewis, James; Fowler, 2014) tarafından popülerleştirilmiştir. Ancak, akademiye yoğun ilgi 2014 yılında ivme kazanmaya başlamıştır (Balalaie, Heydarnoori, ve Jamshidi, 2016).

(Escobar vd., 2016)'te eskiden kalan sistemlerin mikroservisler yoluyla modernleştirilebilmesi ile ilgili bir yaklaşım sunulmuştur. Yaklaşım iki önemli adımdan oluşmaktadır. İlk adım, eskiden kalan sistemi anlamaya yardımcı olan görselleştirme modelleri üretmektir. Bu adımda eskiden kalan sistemin güçlü ve zayıf bağlı elementlerini gösteren bir dizi diyagram oluşturulmaktadır. İkinci adım, görsel model elementlerinin bağlanma ilişkisine dayanarak, gevşekçe bağlanmış bileşenleri ayrı mikroservislere dağıtan bir dizi diyagram üretmektir.

Buna karşılık, güçlü bağı olan elementler ise aynı mikroservise bağlanmaktadır. Sunulan yaklaşım eskiden kalan sistemin anlaşılması ile sınırlıdır. Eskiden kalan sistemin modernleştirilme aşaması bu çalışmanın kapsamı dışındadır.

Otomatik ayrışmış sistemlerin mikroservislere dönüştürülmesinin performansı ve ölçeklendirilebilirliği (Abdullah, Iqbal, ve Erradi, 2019)'te ayrıntılı olarak incelenmiştir. Yazarlar, otomatik ayrıştırılmış eskiden kalan sistemin, manuel ayrıştırılmış eskiden kalan sistemden daha iyi performans gösterdiğini ve daha iyi ölçeklenebildiğini iddia etmektedirler. Bu nedenle, makale, eskiden kalan sistemlerin mikroservislere otomatik olarak ayrıştırılması için bir yöntem sunmaktadır. Yöntem, verilen bir monolitik sistemin Tekdüzen Kaynak Tanımlayıcıları'nı (URI) kullanmaktadır. Teknik, bir dizi URI'yi tanımlamaktadır; tanımlanmış olan URI'ler otomatik olarak farklı mikroservislere bölünmektedir. Ve sonra her mikroservis, performansı en üst düzeye çıkarmak için uygun sanal makine türü kullanılarak dağıtılmaktadır. Veritabanı katmanı bu çalışmada dikkate alınmamıştır.

Monolitik sistemlerin mikroservislere ayrıştırılması için, veri akışı temelli tekniklere dayanan, yarı otomatik bir yaklaşım (Li vd., 2019)'te ayrıntılı olarak açıklanmıştır. Yöntem,

monolitik sistemin kullanım senaryosu tanımlamasını ve iş mantığını anlamakla başlamaktadır. Bunu; bir önceki adımda kullanıcı tanımlamasına ve analiz edilen iş mantığına dayanan veri akışı diyagramlarının oluşturulması ve bunu veri akışı diyagramının bir işlem-veri deposu versiyonunun elle geliştirilmesi takip etmektedir. Son olarak hem veri akışı hem de işlem-veri deposu diyagramları aday mikroservisler oluşturmak için otomatik olarak ayrıştırılmaktadır. Çalışma, monolitik sistemin ayrışmasının ötesine geçmememektedir.

Eskiden kalan monolitik sistemden mikroservis sistemine olan farklı taşınma metotları (Kazanavicius ve Mazeika, 2019)'de özetlenmiştir. Çalışma, farklı taşınma yöntemlerini özetlemeden önce hem monolit hem de mikroservis sistemleriyle ilgili genel farklılıkları, avantajları ve dezavantajları hakkında genel bir tartışma ile başlamıştır. Mevcut taşınma teknikleri iki gruba ayrılmaktadır: yeniden düzenleme ve yeniden inşa etme. Yeniden düzenleme, mevcut monolitlerin, uygulamayı sıfırdan yeniden oluşturmaya gerek kalmadan tek tek mikroservislere ayrılmasını ifade ederken, yeniden inşa etme, tüm sistemleri sıfırdan geliştirmeyi ifade etmektedir. Yazarlar, eskiden kalan sistemlerde kullanılan boyut, karmaşıklık ve teknolojilerin bu sistemleri ya yeniden inşa etme ya da yeniden düzenleme için bir aday haline getirdiğini savunmaktadırlar.

Mikroservisler hakkındaki mevcut kaynakların çoğu, sıklıkla bulut bilişim üzerine çalışmaları içermektedir. Bulut için mikroservis tabanlı uygulamaların geliştirilmesi ve konuşlandırılması (Guo, Wang, Zeng, ve Wei, 2016)'da ele alınmaktadır, mikroservislerin dağıtımında taşıyıcıların kullanımı (Singh ve Peddoju, 2017)'de sunulurken (Gan ve Delimitrou, 2018)'de mikroservislerin bulut üzerindeki etkisi incelenmektedir. Bulut için mikroservis tabanlı uygulamanın teknik ve ekonomik analizleri (Zheng ve Wei, 2018)'da sunulmaktadır. (Villamizar vd., 2015)'da ise mikroservis uygulamalarının buluta dağıtımının faydaları ve zorlukları tartışılmaktadır.

Mikroservislerin Nesnelerin İnterneti'ne(IoT) uygulanması ivme kazanmaya başlamıştır. Nesnelerin İnterneti(IoT) uygulamasında akıllı bir şehir inşa etmek için mikroservislerin kullanımı (Krylovskiy, Jahn, ve Patti, 2015)'de sunulmaktadır. (Sun, Li, ve Memon, 2017)'de Nesnelerin İnterneti'nin uygulanmasıyla ilgili farklı gereksinimlerin, birlikte çalışabilirliğin, özelleştirmenin ve ölçeklenebilirliğin zorluklarını çözmek için mikroservislere dayalı bir açık ortam sunulmaktadır. (Jarwar, Kibria, Ali, ve Chong, 2018)'te Nesnelerin İnterneti uygulamasında Web nesnelerinin yeniden kullanılabilirlik prensibi görülmektedir. (Al-Masri, 2019)'te ise Nesnelerin İnterneti uygulamalarının işlevsel olmayan niteliklerinin genel hizmet kalitesinin geliştirilmesi için detaylı bir çerçeve çizilmektedir.

Mikroservis uygulamalarının güvenliği yavaş yavaş akademik çalışmalarda yerini almaktadır. Örneğin, (Yarygina ve Bagge, 2018)'te güvenlik sorunlarının taksonomisi ve mikroservis iletişimini güvence altına alan bir çerçeve sunulmaktadır. Mikroservislerin güvenlik zorlukları ve burada belirlenen boşlukları kapsayan bir çözüm (Yu vd., 2017)'da detaylandırılmıştır.

Mikroservislerin performansı araştırmacılar için de önemli bir konudur. (Heinrich vd., 2017)'te mevcut tekniklerin yetersiz olduğuna vurgu yapılarak mikroservis mimarilerinin performansıyla ilgili daha fazla araştırmaya ihtiyaç duyulduğundan bahsedilmektedir. (Niu vd., 2018)'te bir mikroservis mimarisinde performansı artırmak için bir yük dengeleme algoritması tartışılmıştır. Sunucusuz bir bulut platformunda mikroservislerin performansını etkileyen faktörler (Lloyd, Ramesh, Chinthalapati, Ly, ve Pallickara, 2018) 'te tartışılmıştır. (Fan, Han, Zhang, ve Wang, 2018)'da mikroservislerde veri tutarlılığının artırılmasına yönelik bir metodoloji önerilmektedir. (Knoche ve Hasselbring, 2019)'de mikroservislerin benimsenmesine yönelik engellerin ve sürücülerin araştırılması detaylandırılmıştır.

Bu nedenle, birinci araştırma sorusunun yanıtı, hem mevcut sistemlerin ve bu sistemlerin modernizasyonunun; hem de yeni ortaya çıkan alanlara (Nesnelerin İnterneti ve Bulut

bilişim) uygulanmasının ötesine geçen mevcut mikroservis mimarisi çalışmalarıdır. Mikroservis mimarisinin işlevsel olmayan yönüyle ilgili çalışmalar da artmaktadır.

AS2: Veritabanı Taşınması İçin Mevcut Yöntemler Nelerdir?

Mikroservisten farklı olarak, veri/veritabanı taşınması konusunda oldukça zengin bir literatür mevcuttur. Veri/veritabanı üzerine çalışmalar 1990'lara kadar uzanmaktadır. (Youn ve Ku, 1992); bu nedenle, mikroservise kıyasla veri taşınması, olgun bir çalışma alanıdır. Veritabanı taşınması ile ilgili çalışmalar aşağıdaki gruplara ayrılabilmektedir:

(Zhao, Lin, Li, ve Li, 2014)'de ilişkisel bir veritabanının NoSQL'e dönüştürülmesi için bir model sunulmuştur, (Scavuzzo, Nitto, ve Ceri, 2014) satıcıya bağımlılık kilidini önlemek için veritabanları arasında veri taşınması ile ilişkili bir metamodel sunmaktadır. (Rocha, Vale, Cirilo, Barbosa, ve Mourão, 2015)'da ilişkiselden NoSQL'e veri taşınması için otomatik bir çerçeve sunulmaktadır.

(Monk, Mariani, Elgalal, ve Campbell, 1996)'de ilişkisel veritabanı şemasını nesne yönelimli veritabanına dönüştürmek için bir algoritma sunulmaktadır. Ardından veri taşınmasının; hedef nesne tabanlı bir veritabanı ve kaynak ilişkisel veritabanıyla eşleştirilmesi için gereken metoda ayrıntılı bir şekilde (Hohenstein, 2000)'de değinilmiştir. İlişkisel veritabanı modelini alan ve onu birden fazla hedef modele dönüştüren bir çözüm (Maatuk, Ali, ve Rossiter, 2008)'te sunulmuştur. Çözüm, nesne tabanlı, XML, nesne ilişkisel veri modellerine hitap etmektedir.

(Rossiter, 2010)'te ilişkiselden nesne-ilişkisel veritabanına veri taşınması yöntemi sunulmaktadır. Yöntem, aynı zamanda şema dönüşümü ve veri örneği yapmaktadır. İlişkiselden nesne-ilişkisel veritabanına otomatik veri taşınması için bir yöntem önerilmektedir (Bahaj ve El Alami, 2013). (Bahaj ve El Alami, 2013)'te yapılan çalışmaların biraz daha ilerletilmiş hali sunulmuştur (Elalami, 2014).

Buluta olan veri taşınması dikkat çekici boyuttadır. (Binz, Leymann, ve Schumm, n.d.)'de hem uygulamanın hem de onunla ilişkili verilerin buluta taşınması için bir çerçeve sunulmaktadır. (Shirazi, 2012)'de verilerin buluta taşınması için Laszewski ve Nadduom tarafından taşınma metodolojisinin uyarlaması olan bir tasarım deseni (Strauch vd., 2013) sunulmuştur.

(Laszewski ve Nauduri, 2011), (Bisbal vd., 1997), (Xing ve Li, 2010), (Niyomthum ve Chittayasothorn, 2003)'te bahsedilen diğer taşıma modellerini yukarıdaki kategorilere dahil etmek neredeyse imkansızdır.Diğer taşıma modelleri, veri ve veritabanı taşınması çalışmaları literatüründe ele alınmıştır.

Özetle, veri ve veritabanı taşınması üzerine çalışmalar; verinin ilişkisel, nesne tabanlı, nesne ilişkisel ve SQL'den NoSQL sistemlerine geçişi alanlarına yönlendirilmiştir. Veri ve veritabanının buluta taşınması üzerine çalışmalar ve teknikler de yükseliştedir.

2.4 Tartışma

2.4.1 Bulguların yansıması

Mikroservis mimarisinde mevcut yönelim nedir?

Mikroservis mimarisi, yazılım mühendisliği alanında nispeten yeni bir paradigma olmasına rağmen hem endüstride hem de akademide çok büyük ilgi çekmektedir. Sonucun gösterdiği gibi, mikroservis çalışmaları mevcut veya eski sistemlerin modernizasyonuna (Knoche ve Hasselbring, 2019), yani monolit bir sistemi mikroservis tabanlı bir sisteme (Di Francesco vd., 2018)(Jamshidi vd., 2018) dönüştürmeye yöneliktir. Ayrıca, mikroservis mimarisinin literatürde yaygın olan bulut bilişim (Balalaie vd., 2016)(Lloyd vd., 2018) ve Nesnelerin İnterneti (Krylovskiy vd., 2015)(Sun vd., 2017) gibi gelişmekte olan alanlara uygulanmasına yönelik önemli çalışmalar yürütülmektedir.

Öte yandan, mikroservis mimarisinin kalite nitelikleri (Ampatzoglou, Frantzeskou, ve Stamelos, 2012) üzerine çalışmalar artmaktadır. Mikroservis performansı (Heinrich vd.,

2017)(Niu vd., 2018) ve güvenlik sorunları (Yarygina ve Bagge, 2018)(Yu vd., 2017) alanlarında devam eden kayda değer çalışmalar bulunmaktadır.

Özetle, mikroservisler üzerindeki çalışmalar iki yönlüdür. Birincisi mikroservis mimarisinin diğer bilgi işlem alanlarına katkısı, ikincisi işlevsel olmayan nitelikler üzerinden (Ampatzoglou vd., 2012), mikroservis mimarilerinin kalite özelliklerinin (Yarygina ve Bagge, 2018)(Yu vd., 2017) değerlendirilmesidir.

Veritabanı Taşınması İçin Mevcut Yöntemler Nelerdir?

Şimdiye kadar veri/veritabanı taşınması ile ilgili çalışmalar, yazılım sistemlerinin geliştirilmesinde kullanılan metodolojiler doğrultusunda evrilmiştir. Çok sayıda çalışma, ilişkisel, nesne-ilişkisel ve nesneye dayalı veritabanı yönetim sistemlerinden yine bu sistemlere veri taşınmasına yönelik olarak yapılmıştır.

Yukarıda bölüm 2.3'te sunulduğu gibi, NoSQL'e ilişkisel bir veritabanı için bir dönüşüm modeli (Zhao vd., 2014)'de tartışılmıştır. Satıcı kilitlenmesini önlemek için veritabanları arasında veri taşınması için bir metamodel (Scavuzzo vd., 2014)'da görülmektedir. Verinin ilişkiselden NoSQL'e taşınması için otomatik bir çerçeve (Rocha vd., 2015)'da sunulmuştur.

(Monk vd., 1996)'de ilişkisel veritabanı şemasından nesneye dayalı bir veritabanına dönüşümsel bir algoritma görülmektedir. Ardından veri taşınması için hedef nesne tabanlı bir veritabanıyla; eşleşmesi için gerekli kaynak ilişkisel veritabanı metodu ayrıntılı bir şekilde (Hohenstein, 2000)'de belirtilmiştir. (Maatuk vd., 2008)'te ilişkisel veritabanı modelini alan ve onu nesne yönelimli, XML ve nesne ilişkisel veri modellerine dönüştüren çok modelli bir çözüm görülmektedir.

İlişkiselden nesne-ilişkisel veritabanına veri taşınması için (Rossiter, 2010)'te bir yöntem sunulmuştur; çözüm hem şema dönüşümünü hem de veri örneği oluşturma işlemini

gerçekleştirmektedir. Laszewski ve Nadduonm'un buluta veri taşınması için bir tasarım deseni olan çalışmalarının (Strauch vd., 2013) uyarlanması, (Shirazi, 2012)'de sunulmaktadır. Özetle, veritabanı taşınmasının evrimi, yazılım geliştirmedeki ilerlemeyi yansıtmıştır.

Bulut ve Nesnelerin İnterneti gibi gelişmekte olan alanlara mikroservis mimarisinin uygulanması ile ilgili araştırmalarla veritabanı yönetim sistemlerinin entegrasyonu, araştırmacılar için çalışmaya değer olabilecek niteliktedir. Veri ve veritabanı yönetim sistemlerinin gelecekte bilişim ve dijital dönüşümdeki yeri yadsınamaz niteliktedir.

2.5 Bulgular ve Tartışma

Açıkça görülüyor ki, veritabanı sistemleri arasında veri taşınması literatürde iz bırakmıştır ve verilerin veritabanı sistemlerinden birbirine taşınmasına yönelik farklı teknikler vardır.
Bu teknikler, yazılım geliştirme yöntemleri için daha önce bulunan benzer teknikleri izlemiştir.

Mikroservis, literatürdeki yerini bulmuştur. Mevcut çalışmalar, var olan sistemlere ve yeni yazılım projelerine uygulanmasının yanı sıra; gelişmekte olan bilgi işlem alanlarına ve mikroservis mimarilerinin niteliğine yönelik çabalarını da göstermektedir.
Ancak, bu çalışma, literatürde var olan çalışmaların eskiden kalan bir monolitik sistemin uygulama katmanlarının anlaşılması ve dönüşümü ile sınırlı olduğunu göstermiştir. Bu nedenle, bu çalışmada, uygulama katmanının anlaşılması ve dönüştürülmesiyle sınırlı olmayan, ayrıca verilerin mikroservis mimarisine daha sonra taşınması için veritabanı katmanının dönüştürülmesini de içeren bir yöntem önerilecektir.

Bu yüzden, yakın gelecekte yapılacak olan çalışmalar, bir mikroservis mimarisine veri taşınmasını incelemek amacıyla (Di Francesco vd., 2018)'de yapılan öneriyi takip edecektir. Verinin mevcut bir sistemden mikroservislere taşınması ve endüstriyel uygulayıcıların bunu deneyimlemesi için bir metodoloji geliştirilmelidir. Veri toplamak için görüşme ve anketleri

kullanacak deneysel araştırmalar yapılabilir. Önerilen çözümü doğrulamak için vaka çalışması şeklinde bir deney yapılacaktır.

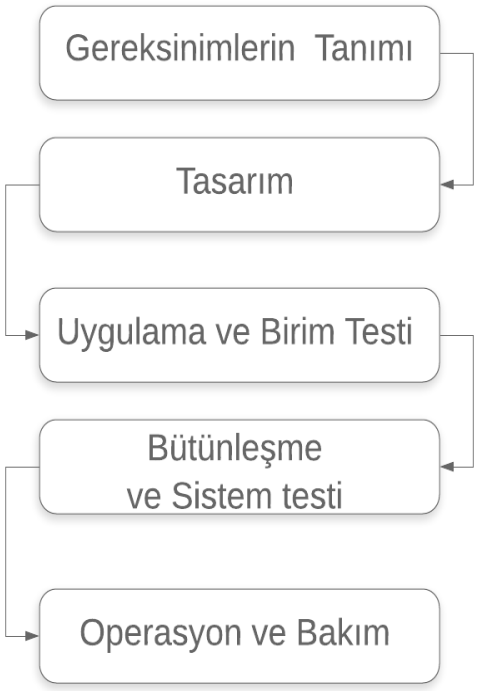
3. MATERYALLER VE YÖNTEMLER

3.1 Giriş

Tezin bu noktasında, bazı önemli kavramların sunulması gereklidir. Bu kısım iki alt bölümden oluşmaktadır. İlk bölümde, yazılım mühendisliği yaşam döngüsü ile başlayan ve bu tezin ana konusu olan yazılım mimarisine kadar uzanan temel yazılım kavramları açıklanmaktadır. İkinci bölüm, bu tez çalışmasını yürütmek için kullanılan materyalleri, araçları ve araştırma yöntemini açıklamaktadır.

3.2 Yazılım Mühendisliğinde Aşama Temelli Yaklaşım

Yazılım mühendisliği (Bourque, Dupuis, Abran, Moore, ve Tripp, 1999), Ada Lovelace (Charman-Anderson, 2015) günlerinden bugüne çok yol kat etmiştir. Bu evrim; süreçler, yöntemler veya tekniklerle sınırlı değildir; yazılım mühendisliğinin tanımı da sürekli gelişmiştir. Bu tez çalışmasında (Sommerville, 2010)'de verilen: "*Sistem spesifikasyonunun ilk aşamalarından sistem kullanıma girdikten sonra sistemin bakımına kadar yazılım üretiminin tüm yönleriyle ilgilenen bir mühendislik disiplini olarak yazılım mühendisliği*" tanımı dikkate alınmıştır. Tanımda, eğer detaylandırılırsa bu çalışmanın kapsamı dışına çıkacak çok şey vardır. Bununla birlikte, tanımda geçen "*aşamalar*" ilgi çekicidir, bu da bizi yazılım mühendisliğinin-sistem geliştirme yaşam döngüsünün ve çalışmasının temel kavramlarından birine götürmektedir. Yazılım mühendisliği yaşam döngüsü veya yazılım geliştirme aşamaları; gereksinimlerinin tanımlanması, sistem ve yazılım tasarımı, uygulama ve birim testi, entegrasyon ve sistem testi ile işletme ve bakım olarak ayrılabilir (Sommerville, 2010).



Şekil 3.1 Yazılım Geliştirme Yaşam Döngüsü (Sommerville, 2010).

Yukarıda **şekil 3.1**'de gösterilen aşamalar, kullanılan işlem modeline bakılmaksızın yazılım mühendisliğinin gelişimi için temel olan adımları göstermektedir. Başka bir deyişle, gereksinim toplama, tasarım, uygulama, test etme ve işlemler, tüm yazılım geliştirme çabalarının bir parçasıdır. Bununla birlikte, bu aşamaların her birinin gerçekleştirilme şekli veya yöntemi, bir işlem modelinden diğerine ve bir yöntemden diğerine farklı olabilir.

3.3 Yazılım Tasarım ve Mimarisi

Bu tezin çalışması öncelikle hem akademiden hem de uygulayıcılardan muazzam bir çaba gören yazılım mühendisliğinin tasarım aşamasına odaklanmıştır. Tasarım aşaması, gereksinim aşamasını uygulama aşamasına bağlar; bu nedenle, işlevsel programlama (Hughes, 1990, nesne yönelimi (Zalewski, 2003) gibi kavramlar, teknik olarak yazılım mühendisliğinin tasarım sürecinde kapsüllenebilir.Tartışmanın bu aşamasında, bir adım geri

çekilip daha büyük resme (yazılım mimarisinden tasarıma) kısaca değinmek faydalı olacaktır. (Sommerville, 2010)'de açıklandığı gibi "Yazılım mimarisi, bir sistemin nasıl organize edilmesi gerektiğini anlamak ve bu sistemin genel yapısını tasarlamak ile ilgilidir. Mimari tasarım, yazılım tasarımı sürecinin ilk aşamasıdır. Bir sistemdeki ana yapısal bileşenleri ve aralarındaki ilişkiyi tanımlar. Mimari tasarım sürecinin çıktısı, sistemin bir dizi iletişim bileşeni olarak nasıl organize edildiğini açıklayan mimari bir modeldir." Özetle, tanım, mimariyi inşa edilecek yazılımın bir planı olarak tanımlamaktadır. Bu, bina tasarlayan bir mimarın bir ev inşaatına başlamadan önce çizdiği planla eşanlamlıdır. Bir ev planı sadece odaların ve binanın duvarlarının yapısını ve boyutunu değil, aynı zamanda su, elektrik, gaz ve telekomünikasyon teknolojileri için rotalar ve geçitler içermektedir. Bu farklı bileşenler, bina inşa edildiğinde, ihtiyaç duyulan her şey için bir arada tasarlanmıştır.

Yukarıdaki örneğe benzer şekilde, yazılımın mimarisi, yazılımın genel yapısını, katmanlarını veya bileşenlerini göstermektedir. Her katmanın veya bileşenlerin nasıl çalıştığını ve bileşen katmanları arasındaki ilişkileri ve en önemlisi, uygulamanın genel hizmetlerini sağlamak için farklı parçaların nasıl iletişim kurduğunu göstermektedir. Mimarinin, yazılım mühendislerinin ince taneli olarak adlandırdığı daha alt düzey ayrıntılara giremediği belirtilmelidir. Bunun yerine, mimari model, yazılımın üst düzey yapısı olan kaba tanelerle sınırlıdır. Bir yazılım tasarımının daha düşük seviyedeki detayı, bu tezin konusu olmayan yazılım tasarım kalıplarının çatısı altındadır (Ampatzoglou vd., 2012).

Sihirli çözüm arayışında, yazılım geliştirme yaygın olarak bilinen üç yazılım mimarisi sunmuştur: Monolit (Chen, Li, ve Li, 2018), servis odaklı (SOA) (Rosen, 2009) ve mikroservis (Larrucea, Santamaria, Colomo-Palacios, ve Ebert, 2018) mimarileri. Önceki cümle için "yaygın" kelimesi kritiktir, çünkü yazılım mimari çalışması, belirtilen üç mimariyle sınırlı değildir.

(Sommerville, 2010)'de mimari, kalıplar bağlamında da tartışılmaktadır; bu mimari desenler, yazılım tasarımında tekrar eden sorunlara farklı çözümler olarak sunulmaktadır. Bu mimari

desenler arasında katmanlı mimari, depo mimarisi, kullanıcı-sunucu mimarisi ile pipe and filter mimarisi bulunmaktadır.

Yazılım mimarisi çalışması kendi başına bir derstir ve bu alanda yayınlanmış birkaç cilt kitap vardır. Ancak, bu tezin çalışmalarını ileriye taşımak için monolit, SOA ve mikroservis mimarilerinin tartışılması tarafımdan yeterli görülmektedir.

3.4 Monolitik Mimari

Monolit kelimesi sözlüklerde aranmıştır: Merriam-WebsterDictionary: "genellikle bir dikilitaş veya sütun şeklinde tek bir büyük taş." devasa bir yapı." tek etkili güç olarak hareket eden organize bir bütün. " Macmillan sözlüğü: "değişmek istemeyen büyük ve çok güçlü bir organizasyon veya sistem. " taş veya benzeri bir maddeden yapılmış çok büyük bir bina veya yapı. " eski zamanlarda yerleştirilmiş ve bir uçta duran büyük bir taş parçası." Bu nedenle, bir nesneyi veya şeyi tanımlamak için monolit kelimesinin tek/bütün veya büyük olarak kullanımını anlamak güvenilirdir. Bu nedenle, monolit mimari, gerekli hizmetleri sağlamak için tek/bütün paket olarak oluşturulmuş bir uygulamayı açıklamaktadır.

Yazılım mimarları, monolit mimariyi, sistemi oluşturan tüm bileşenler için tek bir kod tabanını paylaşan bir sistem olarak tanımlamaktadır. Bu bileşenler, gerekli hizmetleri sağlayan tek bir sistem olarak teknik görüşü kullanmak için birbirine sıkıca entegre edilmiştir veya yüksek oranda birleştirilmiştir (Candela, Bavota, Russo, ve Oliveto, 2016).

3.4.1 Katmanlı mimari

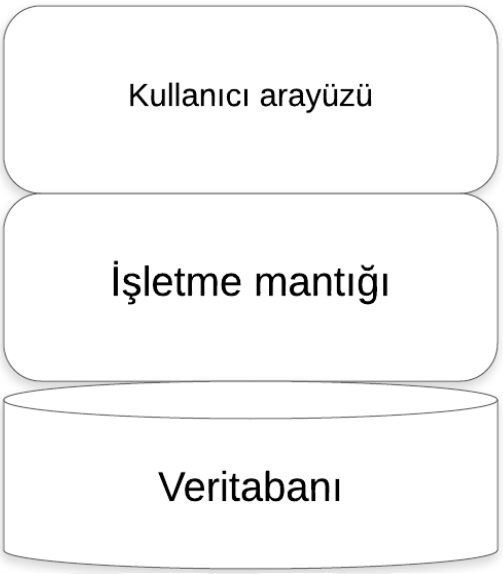
Monolitik mimarinin farklı mimari desenleri vardır. Monolitik mimariyi tartışmak için, bu tezde kurumsal uygulamalarda yaygın olan katmanlı mimari deseni kullanılacaktır. Katmanlı mimari, uygulamanın her bileşenini, her katmanın belirli bir işlev kümesinden sorumlu olduğu bir katman olarak değerlendirilmektedir. Genellikle, bileşenler üç katman halinde tasarlanmıştır: Kullanıcı ara yüzü, iş mantığı ve veri. Bkz. **şekil 3.2.** Her katmanı kısaca

tartışmadan önce, bileşenler katman olarak oluşturulmuş olsa da bir katmandaki bir değişikliğin tüm uygulamayı etkileyecek şekilde sıkıca bağlandığını belirtmek gerekir.

Kullanıcı ara yüzü, kullanıcı ve sistem arasındaki bağlantıyı sağlamaktadır. Kullanıcının sistemin çalışması için sağladığı hizmetlere erişmesine yardımcı olmaktadır. Web tabanlı sistemler için, genellikle, kullanıcı arabiriminin teknoloji yığını HTML, CSS ve JavaScript içermektedir.

Orta katman, iş kurallarına dayanarak iş mantığının yürütülmesinden sorumludur. Uygulamanın bir bordro sistemi olduğunu varsayılırsa, o zaman bu katman, maaş skalaları ve uygulanan vergi basamaklarına dayalı personel maaşları ve vergi kesintileri ödemek için bordronun hesaplanmasından ve işlenmesinden sorumludur. Teknoloji seçenekleri Node.js, PHP, Java Spring, .NET teknolojilerini içermektedir.

Kalıcılık katmanı olarak da bilinen veritabanı katmanı, iş verilerinin depolanmasından sorumludur. Kurumsal verileri gerektiğinde depolar, işler ve çıktılamaktadır. Orta katman, genel uygulama hizmetleri sağlamak için genellikle veritabanı katmanı ile iletişim kurmaktadır. İki popüler veritabanı modeli SQL ve NoSQL'dir.



Şekil 3.2 Katmanlı mimari

3.4.2 Monolitik mimarinin dezavantajları

Yazılım mühendisliğinde bir teknoloji veya yöntemle ilgili her zaman yan etkiler veya endişeler mevcuttur. Monolitik mimarilerin kendileriyle ilişkili endişeleri ve sorunları vardır. Geleneksel ve monolitik mimarilerin temel sorunlarının çoğu, uygulamanın tüm bireysel bileşenlerinin aynı kod tabanını paylaşması gerçeğine dayanmaktadır.Monolitik mimarilerde bakım önemli bir sorundur (Salonen ve Deleryd, 2011) ve bakımın sadece maliyeti yüksek değildir aynı zamanda yavaştır (Bennett ve Rajlich, 2000). Teknik açıdan, tek bir bileşenin bakımı tüm uygulamayı etkilemektedir ve hatta kodun diğer alanlarına hatalar (Rothermel ve Harrold, 1996) ekleyebilecek niteliktedir.

Yeni bileşenler eklemek tüm uygulamayı etkilemektedir. Yönetimin Büyük Veri'nin potansiyel faydalarını en üst düzeye çıkarmak için SQL veritabanından NoSQL model veritabanı sistemlerine geçmek istediği bir durum göz önünde bulundurulmalıdır. Bu tür bir değişiklik yalnızca uygulamanın kalıcılık katmanını etkilemez, aynı zamanda uygulama mantık kodunu da etkilemektedir, çünkü uygulama veritabanına bağlanmaktadır, bu da

doğrudan kullanıcı arabirim kodunu etkilemektedir -böylece tüm uygulamayı üretimden çıkarmaktadır. İş maliyetinin büyüklüğü aşikardır.

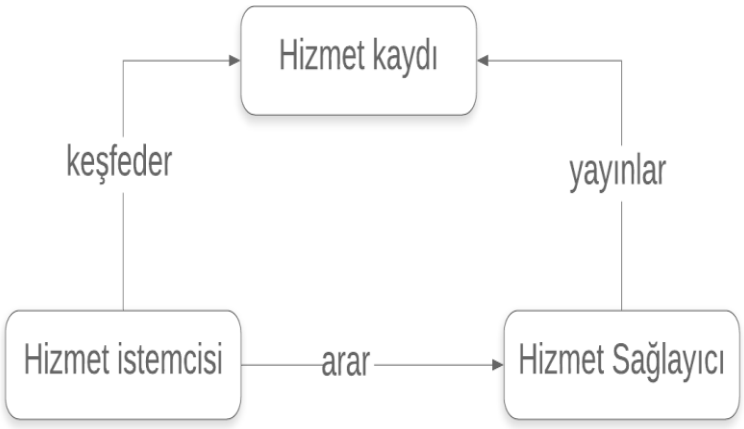
Bakım sorunu, piyasa endişesi ile sıkı bir şekilde bağlantılıdır, bir yazılım şirketinin pazara yazılım veya güncelleme sunması için gereken zaman bu bağlantının sebebidir. Bu durum, inovasyonun hayatta kalma ve büyümenin birincil kaynağı olduğu günümüzün küresel rekabetçi dünyasında büyük bir sorundur.

3.5 Servis Odaklı Mimari (SOA)

Monolitik mimari ve diğer endüstriyel yazılım talepleriyle ilgili problemler, yazılım geliştirmeye mükemmel bir çözüm bulmak amacıyla uzun bir araştırma yapılmasına sebep olmaktadır. Bu durum Servis Odaklı Mimari'nin (SOA) oluşmasına ve gelişmesine yol açmıştır (Mohammadi ve Mukhtar, 2018).

SOA, monolitik mimarinin aksine, yazılım sistemlerine tüm uygulamayı oluşturan ayrı bir hizmet parçası olarak yaklaşır; ancak, bu hizmetler yerel olarak bulunabilen veya harici olarak diğer kuruluşlar tarafından sağlanabilen ayrı birimlerdir. Temel olarak, SOA yazılım gereksinimlerini karşılamak için ayrı Web hizmetlerini bağlayarak kurumsal bir bilgi sistemi geliştirme ilkesini belirlemektedir. Bu servislere Web-Web servisi aracılığıyla erişilebilmektedir. Her hizmet, uygulamayı oluşturan diğer hizmetlerin teknoloji yığını hakkında endişelenmeden farklı bir teknoloji seti tarafından oluşturulabilmektedir. Bu hizmetlerin bireysel özelliklerinin bir sonucu olarak, SOA'daki hizmetlerin bileşimini ve iletişimini yönlendirmek için bir dizi standart (Curbera, Duftler, Khalaf, ve Nagy, 2002) geliştirilmiştir. SOA tabanlı Web hizmetleri genellikle bir hizmet bulma aracısı tarafından kataloglanmaktadır ve elbette bir hizmet sağlayıcısı mevcuttur. Şekilde gösterildiği gibi istekte bulunan hizmet, SOA tabanlı Web hizmetlerinin, Web Hizmetlerinden yararlandığı bir kataloğa sahiptir. Web Tanımlayıcı Dil (WSDL) (Curbera vd., 2002) Web servisleri arasında iletişimi kolaylaştırmaktadır. XML tabanlı bir WSDL dosyası, diğer bilgilerin yanı sıra bir Web hizmetinin sağladığı hizmetleri, bu hizmetlere nasıl erişileceğini ve Web

hizmetinin adresini içermektedir. SOA Web hizmetleri arasındaki iletişim, Basit Nesne Erişim Protokolü (SOAP) (Senagi, Okeyo, Cheruiyot, ve Kimwele, 2016)'nü kullanan kurumsal veri yolu üzerinden gerçekleştirilmektedir. Kurumsal veri yolu veri alır; bazı durumlarda, veri dönüşümünü iletmeden önce yapar. Bu da akıllı veri yolunu SOA tabanlı sistemler için önemli bir dezavantaj haline getirir.



Şekil 3.3 Hizmet Odaklı Mimari ("Technical Know-How: Service Oriented Architecture," n.d.).

3.5.1 Hizmet odaklı mimari'yle (SOA) ilişkili dezavantajlar

SOA'nın popülaritesi hem akademik hem de endüstride kesinlikle görülebilmektedir; ancak, bu popülerlik istenmeyen dezavantajlarla çevrelenmiştir. Web hizmetleri arasında veri bağlayan, dönüştüren ve veri aktaran kurumsal veri yolu anlamına gelen "akıllı iletişim hattı" önemli bir sorun olarak karşımıza çıkmaktadır. Bu bir sorun olarak tanımlanır çünkü veri yolunun akıllı olması mantığın veri yoluna odaklandığı anlamına gelmekte ve bu da özerklik kavramını baltalamaktadır. Sonuç olarak, bir hizmetteki değişiklikler, sistemin üretimi veya piyasaya çıkış süresi üzerinde doğrudan olumsuz bir etkiye sahip olabilen veri yolu da dahil olmak üzere sistemin diğer bölümlerini etkileyebilmektedir.

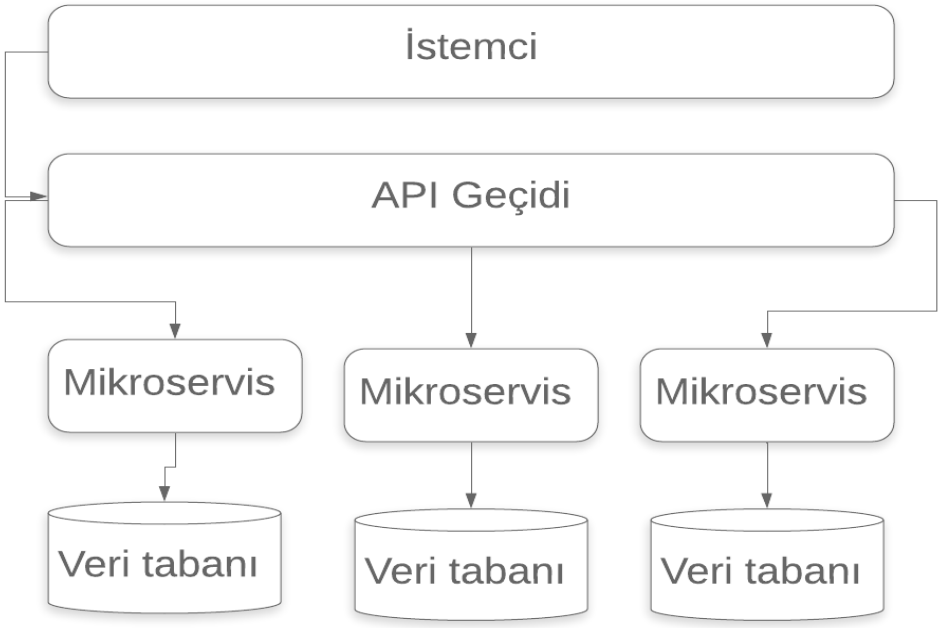
Bir diğer önemli endişe, çoğunlukla Web hizmetleri arasındaki iletişimi güçlendirmek için kullanılan SOAP protokolünün ağır yapısıdır. SOAP diğer benzer teknolojilere kıyasla

yavaştır. Küçük bir mesajın yükü büyüktür (ağırdır), bu nedenle sistemi yavaşlatır ve bu da geniş çaplı istemci kullanımı ile ilgili birkaç Web hizmeti olduğunda büyük oranda performansı etkilemektedir.

3.6 Mikroservis Mimarisi

Yazılım mimarisi dünyasında, mikroservis mimarisi gün geçtikçe popülerlik kazanmaktadır. Peki, mikroservis mimarisi nedir? Mikroservis mimarisinin, bu yazının yazıldığı tarih itibariyle, genel olarak kabul edilmiş bir tanımı yoktur. Bununla birlikte, bu tez için ("Mikroservis" n.d.)'de sunulan tanım şu şekildedir: "Kısacası, bir mikroservis mimarisi, her biri kendi sürecinde çalışan ve hafif mekanizmalarla iletişim kuran küçük bir hizmet paketi olarak tek bir uygulama geliştirmeye yönelik bir yaklaşımdır. Genellikle bir HTTP kaynak API'sidir."

Açıkça, tanımlarda fark edildiği gibi SOA ve mikroservis kavramları arasında önemli bir benzerlik vardır. Her iki mimaride de bir uygulamanın ayrı hizmetlere bölünmesine değinildiğinden ve her biri farklı bir hizmet sağladığı ve hizmetlere genellikle HTTP tabanlı iletişim yoluyla erişilebildiğinden; bu hizmetler, uygulamadaki diğer hizmetlerin altında yatan teknoloji yığını hakkında endişe duymadan farklı teknolojiler kullanılarak geliştirilebilir. Peki, SOA ve Mikroservis Mimarisi arasındaki fark nedir? Kime sorulduğuna veya ne okunduğuna bağlı değişmektedir. Aslında, bu iki mimarlık arasındaki farklılıkları belirlemek devam etmekte olan bir tartışmadır ("Mikroservis vs SOA | What's the Difference | Edureka," n.d.), ("Mikroserviss vs. SOA — Is There Any Difference at All?," n.d.), ("SOA versus mikroserviss: What's the difference? - Cloud computing news," n.d.) .



Şekil 3.4 Mikroservis Mimarisi (“Mikroserviss Architecture Design and Best Practices - XenonStack,” n.d.)

3.6.1 Servis odaklı mimari ve mikroservis mimarisi arasındaki farklar

SOA ve Mikroservis mimarisi arasındaki farkları aşağıdaki faktörlerde yatmaktadır: Hizmetin sahipliği, kalıcılık katmanı ve uç nokta mimarisinin yapılandırılması.

SOA genellikle hem iç hem de dış kuruluşlara ait olabilecek hizmetlerden oluşmaktadır. Sonuç olarak hizmet, talepte bulunan kuruluşun başvuru gereksinimlerini karşılaması durumunda kullanılabilirken, mikroservis mimarisindeki hizmetler, ekibine atanan her hizmetle aynı kuruluşa aittir. Bu ekip, hizmeti geliştirmek, dağıtmak ve sürdürmek için gerekli personellerden oluşmaktadır.

Bir mikroservis mimarisindeki her hizmetin kendine özgü bir veritabanı mevcuttur. Hizmet için gerekli veri, bu veritabanında depolanmaktadır. Prensip olarak, hiçbir hizmet aynı veritabanını paylaşamaz; diğer bir deyişle, SOA’nın aksine veritabanı paylaşımı yoktur. Web servisleri veritabanlarını paylaşabilir. SOA'da her Web hizmeti için ayrı bir veritabanı kavramı yoktur. Mikroservis mimarisinin uç noktasıyla ilgili olarak, genellikle Bağlantılı

Metin Aktarım Protokolü (HTTP) üzerinden çalışan Temsili Durum Transferi (REST) API'leri kullanılmaktadır. Bilindiği gibi RESTful API'leri SOAP'a göre daha hafiftir. Öte yandan, SOA Web hizmetleri uç nokta iletişimi için SOAP protokolünü kullanmaktadır. Daha önce de gördüğümüz gibi, SOAP, SOA tabanlı Web hizmetleri için ağır bir XML tabanlı protokoldür ve bu yüzden SOA için bir dezavantajdır.

3.6.2 Mikroservis mimarisi sürücüleri

Mikroservis mimarisi, hem akademi hem de endüstride aranan sihirli çözüm olmasa da, yazılım mühendisleri, mikroservis mimarisinin kendine has avantajları olduğunu düşünmektedirler. Bu önemli avantajlarından bazıları şunlardır: Sürdürülebilirlik, pazara sürüm süresi, ölçeklenebilirlik, çok dilli programlama ve organizasyonel uyum.

Pazara sunma süresi, bir uygulamanın geliştirilmesinin başlangıcından üretimine/müşteriye gönderilmesine kadar geçen süreyi ifade etmektedir. Uygulamanın her hizmetine farklı ekiplerin ayrılması ve atanması, hizmetin daha hızlı geliştirilmesini ve konuşlandırılmasını kolaylaştırmaktadır. Doğal olarak, her hizmetin bozulması ve özerkliği mikroservis mimarisinde ölçeklenebilirliği kolaylaştırır; uygulamanın boyutu, kuruluş tarafından gerektiğinde ölçeklendirilebilmektedir. Ayrıca, her hizmet, uygulama içindeki diğer hizmetlerin altında yatan teknoloji yığınını düşünmeden farklı bir teknoloji yığını kullanılarak geliştirilmekte ve oluşturulabilmektedir. Bu çok büyük bir iştir. Çünkü gelişmekte olan kuruluşun teknoloji yığınını genişletmektedir, böylece belirli satıcıya bağımlılıktan kurtarmaktadır. Başka bir deyişle, bir yazılımın düzeltilmesi, arıza durumunda veya gerektiğinde yazılımın bakımının yapılması, her hizmetin özerkliği nedeniyle nispeten basittir. İlkesel olarak, başvurunun dökümü, organizasyonun süreci ve işlevleri ile uyumlu olmalıdır; bu durum yazılım uygulamalarının rolünü organizasyonun stratejik seviyesine getirmektedir.

3.7 Yöntem Geliştirme Süreci

3.7.1 Kaynak tarama ve mülakat aşaması

Süreç mikroservis mimarisi hakkındaki kaynakların gözden geçirilmesiyle başlamıştır. Amaç, mikroservis mimarisi alanındaki başlıca çalışmalardan haberdar olmaktı. Bu araştırmalar sonucunda (Di Francesco vd., 2018):

1. Araştırmacılar için, ana eylem noktası belirlenmiştir.
2. Eskiden var olan verilerin mikroservise nasıl taşınacağı problemine yönelinmiştir.
3. Bu tez çalışmasının doğası olan bu sorun ele alınmıştır.

Takip eden adımda, tüm süreç boyunca yol gösterecek bir araştırma planı hazırlanmıştır. Plan, aşağıdaki eylemlerden oluşmuştur:

- Kapsamlı literatür taraması yapmak,
- Endüstri uygulayıcıları ile bir röportaj yapmak,
- Sorunlara olası çözümlere dair bir fikir edinmek için toplanan verileri analiz etmek ve çözümü doğrulamak için bir vaka çalışması kullanmak.

- Burada taslağının önerildiği gibi doğrusal bir şekilde uygulanmadığını vurgulamak gerekir. Aslında, planlama aşamasından son aşama raporlama aşamasına kadar yinelemeli bir süreç gerçekleşmiştir.

Kaynak taramasındaki iki amaç şunlardır: bu sorunun zaten çözülüp çözülmediğini görmek; cevap evet ise, projeyi sonlandırmak; hayır ise, mevcut olan ilgili çalışmaları keşfetmek amacıyla kapsamlı bir inceleme yapmak.

Kaynaklar, mevcut herhangi bir çalışma ortaya koymamıştır; bu yüzden proje sonuna kadar sürdürülmüştür. Ancak, ön izleme bölümünde hazırladığım için proje ile ilgili çalışmalar bulunmuştur. Yazılım mühendisliği uygulayıcıları için mülakat soruları (bkz. Ek 1) hazırlanmıştır; Mikroservis mimarisi tabanlı projelerde çalışan veya bu projelerde yer alan mühendislere ulaşmak için özel bir çaba harcanmıştır. Amaç yüz yüze görüşmek veya telefon görüşmeleri yapmaktı. Ancak, katılımcıların çoğu farklı zaman dilimlerinde bulunan yerlerde

oldukları için zaman farkı sorununun aşmak mümkün olmamıştır. Bu yüzden, soruları yazılı olarak gönderme ve yazılı cevap alma yoluna başvurmak zorunda kalınmıştır; yanıtlar bloglar ve makaleler olarak ("The Hardest Part About Mikroserviss: Your Data – Software Blog," n.d.), ("Database per service," n.d.), ("Patterns for distributed transactions within a mikroserviss architecture - Red Hat Developer," n.d.), ("Decompose by subdomain," n.d.)'da değinilen çevrimiçi sitelerde yer almaktadır.

Referans verilen bu materyaller, (Creswell, 2009)'de sunulan nitel veri analizi sürecine dayanarak analiz edildiğinde yeterli veri üretmiştir. Analiz sonucunda mühendislerin mevcut bir veritabanını mikroservis tabanlı bir servis uygulamasına taşımak için kullandıkları belirli karar noktaları ortaya çıkmıştır. Bu karar noktaları, bir sonraki bölümde sunulan detaylandırılmış metodolojiyi geliştirmek için kullanılmıştır (Vaka Çalışması bölümüne bakınız).

3.7.2 Vaka çalışması aşaması

Bu aşamanın amacı bulguları ve metodolojiyi doğrulamaktı; bu doğrulama, vaka çalışması bölümünde rapor edildiği gibi bir vaka çalışması biçimini almıştır. Mevcut bir monolitik uygulama alınmış ve metodoloji uygulanmıştır. Monolitik uygulama ayrı ayrı veritabanlarını çalıştıran otonom hizmetlerle mikroservis tabanlı bir uygulamaya dönüştürülmüştür. Tasarım, veri tutarlılığını sağlamak ve uygulama genelinde ayrı veri tabanlarında veri çoğaltılmasını önlemek için gerekli önlemleri almıştır. Sonra tüm uygulama kodlanmış ve başarılı bir sonuç için farklı testler yapılmıştır.

3.8 Önerilen Mikroservis Taşınma Metodolojisi

Burada önerilen metodolojinin hem uygulamanın taşınmasını hem de veri taşınmasını dikkate aldığını belirtmek gereklidir. Aynı verileri kullanan veya işleyen uygulamayı dikkate almadan eskiden var olan verinin taşınması zor veya neredeyse imkansızdır.

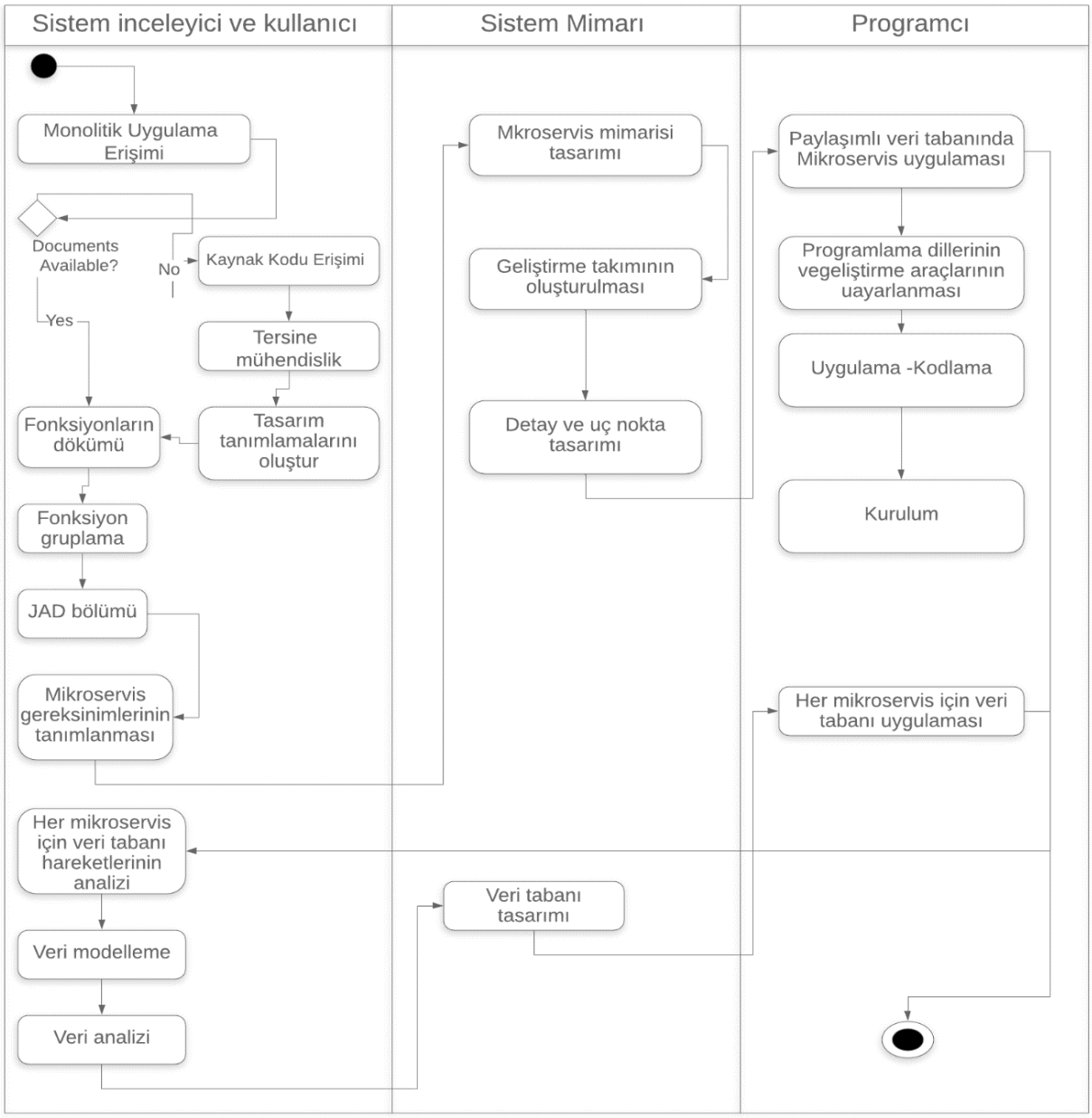
Farklı kaynaklardan toplanan bilgiler üzerinden yapılan veri analizine dayanarak yukarıdaki Araştırma Metodolojisi bölümünde açıklandığı gibi, verilerin monolitik bir uygulamadan bir mikroservis uygulamasına taşınmasını içeren geliştirme projelerinde yaygın olarak görülen

bazı karar noktaları ortaya çıkmıştır. Bu karar noktaları, monolitik bir uygulamadan mikroservis uygulamalarına veri aktarımı için tarafımdan geliştirilen metodolojinin temellerini oluşturmuştur. Metodoloji aşağıdaki aşamalardan oluşmaktadır, bkz. **şekil 3.5**.

- Monolitik Uygulama Erişimi
- Mikroservis Mimari Tasarımı
- Paylaşılan Veritabanında Hizmetlerin Uygulanması
- Mikroservis Başına Veritabanı İşlemlerinin Analizi
- Ayrı Veri Tabanlarının Uygulanması

Metodoloji alt fazlardan oluşmaktadır, **şekil 3.5**'teki her bir ana fazın alt fazları için bkz. **şekil 3.6**.


Monolitik
Uygulama erişimi
Mikroservis mimari
tasarımı
Mikroservislerin
paylaşımlı very
tabalarında
uygulanması
Her mikroservisin
very tabanı
hareketlerinin
analizi
Her Hizmet
Tasarımının
Veritabanı
Ayrı very
tabanlarının
uygulanması



Şekil 3.6 Önerilen Metodolojinin Alt Aşamaları

3.8.1 Monolitik uygulama erişimi

Bu aşamadaki faaliyetler, monolitik sistem dokümantasyonunun kullanılabilirliğini sorgulamakla başlamaktadır. Eğer böyle bir dokümantasyon mevcut ise çalışan bir alt aşama olan Fonksiyonların Dağılımı aşamasına geçilmektedir. Bu aşamanın amacı, uygulamanın benzer gruplar içerisinde toplanmış olan uygulamalarını detaylandıran bir çıktı üretmek, uygulama tarafından sağlanan farklı fonksiyonları anlamak için mevcut sistemi incelemektir. 'Hizmetler' kelimesi önceki ifadede kritik öneme sahiptir. İşlevleri incelemenin amacı, bir organizasyon bağlamında sağladıkları hizmetleri anlamaktır. Örnek olarak, iki benzer işlev ele alınacak olursa: İşlev-Bir, bir çalışanın gelir vergisini azaltmakta ve İşlev-İki ise bir ürün müşteriye gönderildiğinde/kargolandığında stokları azaltmaktadır. İşlev-Bir, İnsan Kaynakları Hizmeti olarak sınıflandırılırken, İşlev-İki, Stok Yönetimi Hizmeti olarak sınıflandırılabilmektedir.

Fonksiyon Gruplamaları

Fonksiyonlar bozulduktan ve sistemin belirli servislerine sınıflandırıldıktan sonra, bu fonksiyonların belirli gruplara sınıflandırılması, bir sonraki aşama olan Fonksiyon Gruplamalarında dikkate alınmalıdır. Hizmet grupları arasındaki sınırların mümkün olduğunca teknik ve net bir şekilde tanımlanmış olması önemlidir. Bu görevin, alana- dayalı gelişime benzer bir zihniyet içinde ele alınması önerilmektedir (Marzullo, Souza, Blaschek, ve Janeiro, 2008), her hizmetin geçici olarak ayrı bir alan adı olduğu kabul edilir ve böylece hizmet grupları aralarındaki sınırları açıkça korurken kapsamlı bir hizmet grubunun detaylı bir şekilde yapılması sağlanmaktadır.

Genel yazılım mühendisliğinin tek sorumluluk ilkesi sınırlı bağlamlar (bounded context) hedefini yönlendirmelidir.

Mevcut sistemin aşağıdaki önemli eserleri incelenmelidir:

- Gereksinim özellikleri
- Mimari tasarım
- Detay tasarım örüntüleri
- Kaynak kodu

- Kullanım kılavuzu vb.

Kaynak Kodu Alma ve Tersine Mühendislik

Belgelerin olmadığı durumlarda tasarım ve şartname belgelerinin bir kısmının veya tamamının, Kaynak Kod Alımının alt aşaması işleme konulmalıdır. Kaynak kodun, Tersine Mühendislik İşlemleri yapmak için kullanılması önerilmektedir (Robbes, Oliveto, ve Di Penta, 2016). Tersine mühendislik işlemleri, adından da anlaşılacağı gibi, yazılım mühendislerinin gereksinim özelliklerini, tasarımını ve mevcut kaynak kodunun benzerini geliştirmek ve eserlerini/modellerini oluşturmak için kullandıkları bir tekniktir. Bu nedenle, bu alt aşamanın amacı monolitik sistemlerin tasarım özelliklerini oluşturmaktır.

JAD Oturumu ve Mikroservis Gereksinimlerinin Belirlenmesi

Bir önceki alt aşamada oluşturulan tasarım özelliklerinin doğru olması, tüm çabanın başarısı için son derece önemlidir. Bu nedenle, üretilen şartnamenin ve benzer belgelerin mevcut uygulamayı bütünüyle temsil ettiklerini belirlemek için test edilmeleri gerekmektedir. Testi yürütmek için Ortak Uygulama Geliştirme (JAD) (Becker, Carmel, ve Hevner, 1993) yönteminin kullanılması önerilmektedir. JAD, gereksinim şartnamelerini ve ilgili oluşturulan belgeleri geliştirmek ve test etmek için paydaşları ortak oturumlara dahil etmek amacıyla kullanılan bir yöntemdir. Bu aşamada JAD oturumu kullanmanın başlıca nedeni, oluşturulan belgelerin söz konusu monolit uygulamasını temsil edip etmediğini açıklığa kavuşturmak ve bunu onaylamak; mevcut monolitik uygulamaların son kullanıcıları olan hissedarlar aracılığıyla doğrulanması ve anlaşılması gereken gereksinim sorularının açıklığa kavuşturulmasında sistem analistleri ve mimarlarına yardımcı olmaktır.

3.8.2 Mikroservis mimari tasarımı

Mikroservisin genel mimarisi bu aşamada tasarlanmıştır. Mimari hesaplar hem sistemin yapısını hem de iletişim mekanizmasını açıklamaktadır.

Geliştirme Takımı Kurulumu

Bu aşamanın amacı, her bir mikroservisin (hizmetin) hem mimari tasarımını hem de ayrıntılı tasarımını üretmektir. Aşama tipik olarak geliştirilecek her bir mikroservis için bir geliştirme ekibi oluşturulması ile başlamaktadır. Her hizmet, başlangıçtan sona kadar her bir mikroservisin geliştirme taleplerini yerine getirmek için gerekli tüm uzmanlardan ve personelden oluşan bir geliştirme ekibine atanmaktadır. Mikroservis mimarisinde, uygulama, "yap ve gönder" şeklinde değil; "oluştur ve sahip ol" şeklindedir; başka bir deyişle, her takım ömrü boyunca uygulamaya sadık kalmaktadır.

Detay ve Uç Nokta Tasarımı

Bu alt safhanın ayrıntılı tasarımı, her bir mikroservisin yapısını bütünüyle detaylandıran bir tasarım üretmeyi amaçlamalıdır. Her bir mikroservis için mimari modeller ve ilgili kararlar dikkate alınmalı ve gerekli tasarım işlemleri yapılmalıdır. Geleneksel yazılım geliştirme süreçleri, metodolojileri ve geliştirme araçları, her bir ekip tarafından ayrı mikroservislerini sunmak için uygun görüldüğü gibi kullanılabilir. Mikroservis mimarisinin temel ilkesi olarak her bir sisteme ayrı bir varlık olarak yaklaşılsa da tasarım eserleri için genel uygulama mimarisinde bulunan mikroservisler arasındaki gerekli iletişim mekanizmalarının açıkça hesaba katılması çok önemlidir. Mikroservis mimarisindeki işlemler genellikle tamamlanmadan önce birden fazla mikroservise yayılabilir.

İletişim mekanizması, servisler arasında iletişim modelini şart koşmalıdır. Mikroservis mimarisinde, HTPP üzerinden RESTFUL API'ler (Zhou vd., 2014) her yerde bulunmaktadır. Hizmetler arasında senkronize ("US6377640B2 - Means and method for a synchronous network communications system - Google Patents," n.d.) ve asenkron ("US8838808B2 - Asynchronous communication in Web applications - Google Patents," n.d.) olmak üzere iki popüler iletişim stili mikroservis mimarilerinde de yaygındır. Sam Neuman'ın ders kitabında hizmetler arasında senkronize ve asenkronize iş birliği modellerinin dikkate alınması konusunda açık bir tartışma ele alınmıştır. Her bir

senkronizasyon modelinin anlamı da vurgulanmıştır. Karşılaştırmalı olarak, bu bölümde mikroservis iletişimi hakkında yaptığım öneriler, ders kitabındaki tartışma ile iyi bir şekilde senkronize olmaktadır. Bir ağ geçidi üzerinden iletişimi kanalize etmek, mikroservis mimarisinin bir özelliğidir. Bir ağ geçidi, iletişim hizmetleri arasında bir köprü görevi görebilir, iletişimi kolaylaştıran ve uygulamanın servisleri arasında konuşmayı /gereksiz konuşmayı azaltan Ağ Geçidi Birleştirme Deseni ("Gateway Aggregation pattern - Cloud Design Patterns | Microsoft Docs," n.d.) tipik bir desendir.

3.8.3 Mikroservisin paylaşılan veritabanına uygulanması

Programlama Dillerinin Uyarlanması ve Geliştirilmesi

Bu aşama hem genel mimari hem de iletişim tasarımını karşılayan her hizmet için çalışan bir sistem üretmelidir. Programlama dilleri ve ilgili geliştirme araçları ve ortamları bu aşamada devreye girmektedir. Her hizmet bağımsız olarak uygulanabildiği için, her ekip, özellikle ihtiyaçlarını karşılayan uygun programlama dilini ve araçlarını seçme ayrıcalığına sahiptir; bu nedenle, çok dilli bir teknoloji yığını kavramı mikroservis mimarisi ile ilişkilidir.

Uygulama (Kodlama) ve Kurulum

Mimarinin kaynak koduna uygulanması sürecin bu aşamasında yapılır:
Kaynak kodu yazma, kaynak kodu dosya bağlılığı yapılandırması, birim sınama bütünleştirme.
Buradaki amaç, her bir mikroservis mimarisini karşılayan bir çalışma kodu üretmektir.

Son olarak, her bir mikroservis, bu yöntemin bir sonraki aşamasında gerekli görevleri kolaylaştırmak için mevcut paylaşılan verilerle bağlantılı olmalıdır- Her Mikroservisin Veritabanı İşlemini Analiz Etme.

3.8.4 Mikroservis başına veritabanı işlemlerinin analizi

Buradaki amaç, uygulamanın her servisi için mevcut veritabanını otonom veritabanı sistemlerine ayrıştırmak amacıyla her bir mikroservisin veri gereksinimini tanımlamaktır.
Her hizmetin veri gereksinimlerini anlamak için her bir mikroservisin okuma ve yazma veritabanı işlemleri ayrıntılı olarak incelenmelidir. Toplanan bilgiler daha sonra her bir mikroservisin veritabanı gereksinimlerini modellemek için kullanılmalıdır.

3.8.5 Her mikroservis tasarımı için veritabanı

Veri Modelleme ve Tasarımı

Bu aşamanın amacı, mikroservis mimarisinin her servisi için daha sonra geliştirilebilecek ayrı bir veritabanı tasarım spesifikasyonu sağlamaktır. Her hizmetin veritabanı modeli yaklaşımının [88]- SQL ve NoSQL- kullanımı, söz konusu hizmetin veri gereksinimleri tarafından belirlenmelidir. Bağımsız veritabanlarındaki normalizasyonlar ve veri tutarlılığı gereklilikleri, her veritabanı tasarım özelliklerinin tasarım modelinde dikkate alınmalıdır.

Monolitik bir sistemde, veri tutarlılığı sorunu, yalnızca ilgili tüm görevlerin başarılı olması durumunda bir işlemin gerçekleştirildiği atomik geçiş kavramı (ACID) ile çözülür; aksi halde, bir geri alma başlatılır- böylece veritabanının tutarlılığı sağlanmaktadır.
Mikroservis mimarisinde, işlemler uygulamanın farklı hizmetlerine dağıtılmaktadır. Diğer tüm dağıtılmış sistemlerde olduğu gibi, uygulama genelinde veri tutarlılığı sorunu ortaya çıkmaktadır. Bu aşamadaki görev veri tutarlılığı sorununu tüm uygulama genelinde çözmektir. Mikroservis mimarileri genelinde İki Fazlı İşlem (2pc) (Lampson, Lampson, Lomet, ve Lomet, 1993) ve Saga deseni (“Sagas,” n.d.), veri tutarlılığı sorunlarını çözmek için kullanılan iki geleneksel yaklaşımdır. İki Fazlı Taahhüt (Two-Phase Commit), dağıtılmış sistemler arasında veri tutarlılığını sağlayan bir algoritmadır. Algoritma iki aşamaya ayrılmıştır. Birinci aşamada, başlangıç düğümü, bir işleme katılan tüm düğümlerin, taahhütte bulunma veya geri alma sözü vermesini sağlamaktadır. İkinci aşamada, başlatma düğümü, katılan tüm düğümlerin başarılı bir şekilde taahhütte bulunup bulunamayacağını kontrol etmekte, cevap olumlu ise, işlem yapılmaktadır. Bununla birlikte, katılan herhangi bir düğüm

başarıyla işleyemezse, tüm düğümlerden geri dönmeleri istenmekte; böylece sistem genelinde veri tutarlılığı korunmaktadır.

Saga Deseni, veri tutarlılığını korumak için mikroservis mimarisinde yaygın olarak kullanılmaktadır. Saga, işlemlerin yönetiminde olaylardan yararlanmaktadır. Bir işlemdeki mikroservisler, diğer katılımcı mikroservislerin işlemi yürütmek veya zaten taahhüt edilmiş bir işlemi geri almak için telafi edici işlemleri yapmakta kullandığı bir olayı veri yolu aracılığıyla yaymaktadır. Örnek olarak, bir sipariş servisi, alınan bir sipariş için bir müşteriden ücret almak üzere finans hizmetine bir olay göndermekte ve aynı olay, siparişi müşteriye göndermek için pazarlama hizmeti tarafından da alınabilmektedir. Bununla birlikte, müşteri sipariş için ödeme yapmazsa hem sipariş hem de pazarlama hizmetleri tarafından yapılan taahhütleri geri almak için telafi işlemleri gerçekleştirilecek ve böylece mikroservis uygulaması genelinde veri tutarlılığı sağlanacaktır.

Örnek olarak Kayıtlı Öğrenci, Dönem Kredi Saatleri ve Modül Seçimi'ne sahip bir üniversite kayıt sistemini ele alınabilir. Bir öğrencinin bir dönemin başında belirli bir modüle kayıt yapabilmesi için, başvuruda öğrencinin üniversiteye kayıtlı olduğunun (Kayıtlı Öğrenci) onaylaması gerekir; bu onaylandıktan sonra öğrenci dersi seçebilir. Ancak, modül için başarılı bir şekilde kaydolmadan önce, uygulama öğrencinin dönem için mümkün olan azami kredi saatlerini (Dönem Kredi Saatleri) doldurup doldurmadığını teyit etmelidir; aynı zamanda, uygulamanın, modül için o dönemde kaydolabilecek maksimum öğrenci sayısına (Modül Seçimi) ulaşıp ulaşmadığını kontrol etmesi de gerekmektedir.

3.8.6 Ayrı veri tabanlarının uygulaması

Bu aşamanın amacı, önceki aşamalarda tanımlanan tasarım ürünlerine dayalı olarak her bir hizmet için farklı bir veritabanı sistemi geliştirmek ve sunmaktır. Veritabanı motorları görevleri, araçları, kodlaması ve uygulamanın ayrı hizmetlerine bağlantı seçmesi bu aşamada devreye girmektedir. Her bir mikroservis için ayrı veritabanları, tamamlandıkça yavaş yavaş sisteme bağlanmaktadır. Her bir mikroservisin tüm bağımsız veritabanları sisteme başarıyla yüklendikten sonra, paylaşılan veritabanı kullanımdan kaldırılabilir.

4. VAKA ÇALIŞMASI

4.1 Giriş

Bu bölümde, bu tezde ortaya çıkan ana sorunun çözümü olan metodolojinin geçerliliğini değerlendirmek için yürütülen vaka çalışması raporlanmaktadır. Vaka çalışmasının birincil amacı, sunulan metodolojinin mevcut verilerin monolitik mimari tabanlı bir uygulamadan mikroservis mimarisi tabanlı uygulamalara taşınması sorununu çözüp çözemediğini belirlemektir. Bu çalışma (Runeson ve Höst, 2009)'de sunulan yönergelere dayanılarak yapılmıştır.

Vaka çalışması, monolitik kütüphane uygulamasına bağlı mevcut bir veritabanının mikroservis tabanlı bir uygulamaya aktarılmasını sağlamaktadır. Seçilen monolitik uygulamanın, önerilen metodolojinin çözmesi için tasarlanmış problemleri temsil eden bir örnek olduğu düşünülmektedir.

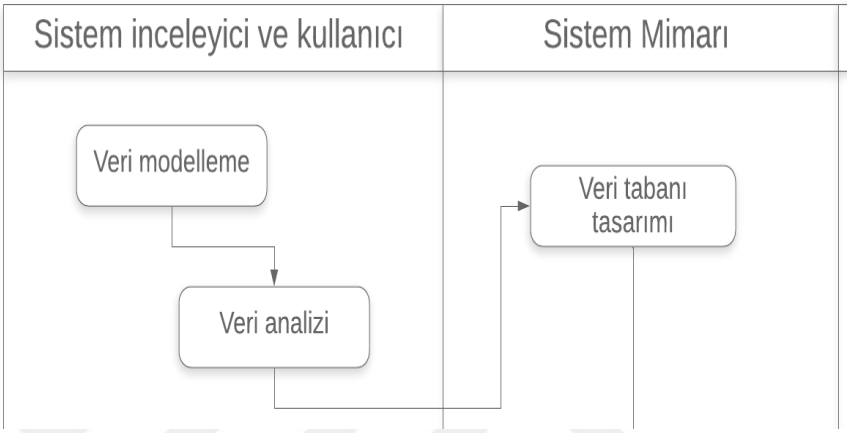
Yukarıda metodoloji bölümünde sunulduğu gibi, vaka çalışmasının uygulanması için çok sayıda rol gerekmektedir. Bununla birlikte, vaka incelemesinde gerçek kişilere erişilemediği için, sistem rolleri, sistem analistleri, sistem mimarları, programcıları ve son kullanıcılara ait temel rolleri tarafımdan üstlenilmiştir.

4.2 Amaç

Bu vaka çalışmasının amacı, önerilen metodolojinin mevcut monolitik uygulamalardan mikroservis tabanlı uygulamalara veri aktarımı uygulamasını değerlendirmektir. Başka bir deyişle, bu vaka çalışması ile, önerilen metodolojinin yazılım mühendislerinin verileri monolitik bir uygulamadan mikroservis tabanlı uygulamalara taşınmasına yardımcı olup olmayacağının belirlenmesidir.

Vaka çalışması monolitik kütüphane uygulamasının mikroservis uygulamasına taşınmasıdır. Vakanın amacı veriyi çoğaltmadan mikroservis uygulamasına taşınmasıdır. Aynı zamanda veri tutarlığının mikroservis uygulaması boyunca korumaktır.

Monolitik kütüphane sistemi, veri tabanına kitap ve sipariş ekleme, görüntüleme, güncelleme ve silme gibi hizmetlerin kablosudur. Uygulama üç katmandan oluşmaktadır: ön katman- kullanıcı ara yüzü, orta katman- işleme katmanı ve kalıcı katman- veritabanı, bkz. **şekil 4.1**, **şekil 4.2** ve **şekil 4.3**.



Şekil 4.1 Monolitik kütüphane sistemi

```
Const mongoose = require(‘mongoose’);

const bookSchema = mongoose.Schema({
  title: {
     type: String,
     required: true
  },

   author: {
     type: String,
      required: true
   },

numberOfPages: {
     type: Number,
      required: true
},

   publisher: {
     type: String,
      required: true
   }
});

const Book = mongoose.model(‘book’, bookSchema);

module.exports = Book;
```

Şekil 4.2 Monolitik sistem (Kitap şeması)

```
const mongoose = require('mongoose').
const orderSchema = mongoose.Schema({
  name: {
     type: String,
     required: true },
   book: {
     type: String,
      required: true },
   age: {
     type: Number,
   },
   address: {
     type: String,
      required: true
   },
initialDate: {
     type: Date,
      default: Date.now()
   },
deliveryDate: {
     type: Date,
      default: Date.now()},});
const Order = mongoose.model('order',
orderSchema);
module.exports = Order;
```

Şekil 4.3 Monolitik sistem (Sipariş şeması)

4.3 Yöntemin Uygulaması

Bu bölümde hem vaka çalışmasında önerilen metodolojinin pratik uygulaması hem de önerilen metodolojinin uygulanması boyunca karşılaşılan zorluklar ve çözümleri sunulmaktadır.

4.3.1 Monolitik uygulama erişimi

Bu tezde önerilen metodolojiye dayanarak, monolitik kütüphane uygulamasını toplanmış ve sunduğu farklı hizmetleri ve bu hizmetlerden sorumlu ilgili işlevleri belirlenmiştir. Yukarıda sunulduğu gibi, uygulamanın kitap ve müşteri siparişleri ekleme, görüntüleme, güncelleme ve silme hizmetlerini sunduğunu belirtilmiştir. Daha sonra, benzer fonksiyonları üçlü gruplar halinde gruplanmış, böylece sonraki mikroservisler için etkili bir şekilde sınırlar çizilmiştir.

4.3.1.1 Gözlemler

Pratik olarak, aşağıdaki teknik zorlukları not edilmiştir:
Benzer işlevleri birlikte tanımlamak ve gruplandırmak direkt geçilebilecek bir aşama değildir, ayrışma sıkıntıları olarak uygulama sorunları ve bu işlevler arasında sınırlı bağlam çizmenin doğru yapılabilmesi ileri-geri hareketler ortaya koyan yinelemeli bir süreçtir. Ayrıca, bu faaliyetleri geliştirme çabası metodolojinin tasarım aşamalarına ilerledikten sonra, bu başlangıç aşamasının çıktısı tasarım aşamalarına girdi olarak yeniden ele alınacaktır. Tekrarlama süreci aşamalar arasında ileri geri giderken kaos ve karışıklıktan kaçınmaya özen gösterilmelidir, çünkü bu sanki ilerleme kaydedilmemiş gibi düşündüren bir zorluktur.

4.3.2 Mikroservis mimarisi tasarımı

Sistemlerin gereksinim özellikleri hakkında yeterli bilgiye sahip olarak, mikroservis mimarisi için mimari hazırlanmaya devam edilmiştir. Bu tasarımı daha üst düzey sonuçları/ürünleri açıklayan üç hizmet sistemi oluşturacaktır. Bunlar kitap, müşteri ve sipariş işlemleriyle ilgili hizmetlerden kitap hizmeti, müşteri hizmeti ve sipariş hizmetidir.

Ayrıntılı tasarım hemen takip edilmiştir/uygulanmıştır, burada birincil görev, her mikroservisin sağladığı hizmetlere erişim sağlamak için hizmet API tasarımını hesaba

katmaktı. Bu projede, sipariş hizmetinin siparişleri görüntülemek amacıyla belirli kitap ve müşteri verilerini sorgulamasını sağlamak ve hem kitap servisi hem de müşteri hizmetleri için bağlantı uç noktaları sunulmuştur. RESTful API'ler için asenkronize programlama teknikleri kullanarak HTTP protokolü üzerinden yerleştirilmiştir.

Detay tasarımını ince ayarlamak için programlama dillerini ve geliştirme araçlarını seçme ihtiyacı mevcuttu. Karar verdiğim teknoloji yığını:

- Node.js (yürütme ortamı)
- Express.js (sunucu)
- Mongoose (Nesne Veri Eşleyici)
- MongoDB (Veritabanı Yönetim Sistemi)
- Postacı (API testi)
- WebStorm (Entegre Geliştirme Ortamı)

Yukarıdaki yığın seçiminin sebepleri şunlardır:

1. Teknolojilerin mikroservis tabanlı uygulamaların geliştirilmesine uygun olması,
2. Java- Spring gibi teknolojilerin daha önce kullanılmış olması.

NoSQL veritabanı modeli, yalnızca veri şemasını vurgulamadığı veya verileri mevcut bir şemaya bağlamadığı için seçilmiştir. Bu sayede veritabanı tasarımında zaman harcamak yerine problemin mantıksal yönüne odaklanılabilmiştir. Çünkü problem kullanımdaki temel veritabanı teknolojisine bakılmaksızın, uygulanması gereken mantıklı bir çözüme ihtiyaç duymaktadır. Seçimi etkileyen bir başka neden de tarayıcıdan JavaScript'in çalışma ortamı olan Nodejs'i kullanmanın getirdiği kolaylıktır ki Node.js'deki nesneler, mantıksal düşünme sürecini ve kod uygulamasını basitleştirmek için NoSQL'deki belgeler gibi ek bir avantaj sağlar.

4.3.2.1 Gözlemler

Pratik olarak, her bir rotanın tek bir görev yaptığından emin olmak için ve parametrelerini tanımlamak ve isimlerin (tekil ve çoğul formlar) fiillerin kullanımını karıştırmamak için tek

sorumluluk ilkesini uygulamak çok önemlidir. Örneğin hem müşteri hem de kitap mikro hizmetlerinden belirli bir müşteri ve kitap bulmak için hem müşterilere hem de kitap hizmetleri API'larına çağrı yollarını uygularken tasarım aşamasını tekrar ziyaret etmek zorunda kalınmıştır. Ayrıca, ilk tasarımların hataları, özellikle HTTP tabanlı bağlantıları, hesaba katmadığı için kodlama aşamasında hayal kırıklıkları yaşanmıştır. Yine de, programlama ve HTTP bağlantı kodlama aşamalarında ilerleyebilecek hataları hesaba katmak için tüm tasarım elden geçirilmiştir.

4.3.3 Paylaşılan veritabanında hizmetlerin uygulanması

Bir sonraki aşama tasarımları çalışma koduna uygulamaktır. Nesneye Dayalı İlkeler her hizmet için kodlamaya rehberlik etmiştir; önemli kompozisyonları kullanılmıştır çünkü iyi bir kodlama uygulamasıdır ("Reasons to use Composition over Inheritance in PHP – Amit Merchant – A blog on PHP, JavaScript and more," n.d.) ve kaynak dosyaları mantıksal gruplara ayırmaya yardımcı olmuştur.

4.3.3.1 Gözlemler

Teknik açıdan, bu aşamadaki zorluklar genellikle spesifik/özel programlama diline ve teknoloji yığınına bağlı olacaktır. Bu vaka incelemesinde, mikroservisi komutları, sipariş servisi tarafından arayan istemciye sunulacak olan hem kitaptan hem de müşteri hizmetinden veri toplama zorluğunu sunmuştur. İlk olarak, her iki hizmetten de veri almak için eş zamansız programlama kullanılmıştır. Ancak, her iki hizmetten de veri başarılı bir şekilde alınabilse de de onları arayan istemciye anlamlı bir sunum için bir araya getirilememiştir.
Bunu yapmak için JavaScript'in asenkronize özelliğini kullanılmış ve düzgün çalışması beklenmiştir. Yine de başka bir küçük teknik sorun ise geri dönüş verileri arasındaki sorunları kapsamaktadır; bu sorunu çözmek için istenen geri arama zincirini kullanarak her iki API çağrısını da aynı deneme ve yakala bloğuna dahil edilmiştir.

Az önce özetlenen kodun ayrıntıları için bkz. **şekil 4.5**.

Kaynak kodların geri kalanı için **Ek B**'ye bakınız.

```
// list an order
app.get('/orders/:id', async (req, res) => {
const id = req.params.id;
try {
const getOrder = await Order.findById(id);
const customerID = getOrder.customerID;
const bookID = getOrder.bookID;

const customerUrl = 'http://localhost:5555/customers/';

const bookUrl = 'http://localhost:4545/books/';

      request({url: customerUrl + customerID, json: true}, async (error, response) => {
const customer = await response.body.name;

         request({url: bookUrl + bookID, json: true}, async (error, response) => {
const book =  await response.body.title;
const orderedObject = {Customer: customer, Book: book, Order_Id: id};
            res.send(orderedObject);
         });
      });
   } catch (error) {
      res.send(error);

});
```

Şekil 4.4 Mikroservis REST API Siparişi

Yine, daha önce de değinildiği gibi, bu zorluklar doğrudan programlama diline ve geliştirme ortamına bağlıdır. Başka bir deyişle, geliştiriciler, kodlama sırasında ortaya çıkabilecek programlama dilinin ve geliştirme ortamının bu tür teknik zorluklarıyla başa çıkmak için yeterli beceri ve deneyime sahip olduklarından emin olmalıdır.

4.3.4 Ayrı veritabanlarının uygulaması

Bir sonraki aşamada ise her bir mikroservis için veritabanı tasarımı yürütülmüştür. Tasarım şemasını ve model tasarımı Node.js koduyla iyi bir şekilde biçimlenen Mongoose'da yapılmıştır. Bu tasarımla, her bir mikroservisin kalıcı katmanında Oluştur, Oku, Güncelle ve Sil (CRUD) işlemlerini gerçekleştirmek için veritabanı modelinin nesneleri somutlaştırılmıştır.

4.3.4.1 Gözlemler

Pratik olarak, bu Nodejs'de Mongoose kullanarak kod yazmayı öğrenmek dışında büyük bir zorluğu yoktur. Mongoose paketi, Node.js'de hem veri şemasını hem de veri modellemeyi kolaylaştırmaktadır. Bu nedenle, sunduğum vakada, veri haritalama işi basit bir aşamadır. Ancak, diğer veritabanı yönetim sistemlerinde, özellikle SQL tabanlı model sistemlerde durum böyle olmayabilir. Bu yüzden, geliştiriciler, seçtikleri veritabanı yönetim sistemleriyle ilgili ortaya çıkabilecek teknik sorunları çözmek için gerekli becerilere sahip olmalıdırlar.

4.4 Bulgular ve Sonuç

Uygulamada, önerdiğim metodolojinin uygulanması doğrusal değildir ve doğrusal olması mümkün değildir. Hem üst düzey hem de detay dizaynın tasarım aşamaları, gerekli son ürünleri elde etmek için tekrarlanan süreçlerdir. Metodolojinin kullanılması mühendislik yaratıcılığına olan ihtiyacı ortadan kaldırmamaktadır; aslında bunun tam tersi geçerlidir. Hem kitap hizmetini hem de müşteri hizmetini görüntülemek ve sipariş vermek için başarılı bir şekilde sorgulamak için asenkronize kodun alınması çok uğraştırmıştır.

Veri çoğaltma ve tutarlılık konusundaki endişeler, kritik bir mühendislik değerlendirmesi gerektiren mikroservis tabanlı bir uygulamanın doğasında mevcuttur. Uygulama genelinde

verileri çoğaltmadan veri tutarlılığını korumak, tasarım ve kodlama faaliyetleri sırasında önemli bir problem oluşturmuştur. Müşteri ve kitap- sadece iki alan depolamak için sipariş hizmeti veritabanını yeniden düzenlemek zorunda kalınmıştır. Her iki hizmet de veriyi okumak için RESTful API'leri aracılığıyla sipariş hizmeti hem müşteri hem de kitap hizmetlerine erişim sağlamaktadır.

Olumlu olarak, metodoloji, veriyi mikroservis uygulamalarına taşıyan mühendisler için kılavuzluk etmektedir. Her aşamada gerekli faaliyetlerin sağlanması ve açıklanması, dijital dönüşümü etkileyen ve gelişmekte olan teknolojilerde zemin kazanan mikroservis mimarisi alanında son derece önemli olan geliştirme süreci boyunca düzen ve sanitasyon/temizlik sağlamaktadır.

Yine de metodoloji, geliştirme sürecinin her aşamasında gerekli olan çeşitli faaliyetlerin ve görevlerin yürütülmesine yardımcı olmak için yazılım mühendisliğinde farklı mevcut teknikleri işaret etmekte ve önermektedir. Yazılım mühendisliğinde sihirli değnek olmadığı ve metodolojilerin, tekniklerin ve teknolojilerin artıları ve eksileri olduğu gerçeği göz önüne alındığında, bu tezde önerilen metodoloji, verileri mikroservis uygulamalarına veri taşınması için pratiktir.

4.5 Geçerliliğe Yönelik Tehditler

Bu bölümde, bu çalışmanın geçerliliği üzerine testler sunulacaktır. Yapı geçerliliği, iç geçerliği ve dış geçerliliği değerlendirilecektir.

Yapı geçerliliği bir deneyin çalışmanın iddiasını nasıl ölçtüğünü ifade etmektedir. Analizin araştırmacının aklında olanları ne kadar iyi karşıladığını veya deneyin araştırma soruları ile ne kadar iyi eşleştiğini ve bu soruları hangi oranda karşıladığını belirlemek için kullanılmaktadır.

Bu nedenle, yürütülen vaka çalışması ile araştırma soruları arasındaki uyumsuzluk potansiyel bir tehdittir. Böyle bir uyumsuzluk, bir araştırma çalışmasının geçerliliğini ortadan kaldırmaktadır. Bununla beraber, vaka çalışması ile araştırma soruları arasında uyumsuzluk olup olmadığı test edilmiştir.

4.5.1 İçsel geçerlilik

İç geçerlilik, deneyde kullanılan yöntemlerin ölçümüne veya gücüne karşılık gelmektedir. Geçerlilik ne kadar yüksek olursa deney o kadar geçerli olur. Geçerlilik ne kadar düşük olursa araştırma çalışmasına yönelik tehdit de o kadar yüksek olur.

Bu nedenle, vaka çalışmasına katılmış olan metodoloji ve yöntemlerin yanlış kullanımı veya yanlış uygulanması ise potansiyel bir tehdittir. Bu deney, tezin araştırma sorusuna çözüm olarak önerdiğim metodolojiye dayanmaktadır. Bu tez çalışması, yukarıda Önerilen Metodoloji bölümünde sunulan metodolojiyi sıkı bir şekilde takip ederek test edilmiştir. Bu da araştırma çalışmasının iç geçerliliğinin daha yüksek olmasını sağlamıştır. Ancak, birden fazla vaka çalışması yapılmış olsaydı çok daha etkili bir iç tehdit olurdu/ oluştururdu.

4.5.2 Dışsal geçerlilik

Dış geçerlik, araştırmanın benzer sorunlara ve projelere uygulanıp uygulanamayacağını ölçmektedir. Eğer erişim gerçek dünyadaki projelere başarılı bir şekilde uygulanabilirse, daha yüksek bir dış geçerliliğe sahipken, uygulanamayan bir araştırma daha düşük bir dış geçerliliğe sahiptir ki bu da istenmeyen bir durumdur. Seçilen vaka çalışmasının temsil niteliği ve yapılan deneyin sonucu göz önüne alındığında, bu araştırmanın benzer yazılım mühendisliği projeleri için de geçerli olduğunu iddia edilebilmektedir.

5. TARTIŞMA ve SONUÇ

5.1 Giriş

Bu bölümde, monolitik uygulamanın bir mikroservis uygulamasına dönüştürülmesi ile ilgili daha önce yapılmış çalışmalar ve bu tez çalışmasının bir sonucu sunulmaktadır.

5. 2 Önceki Çalışmalarla Tezin Karşılaştırılması

Kaynak Taraması bölümünde sunduğum gibi, mikroservis mimarisi hem akademik çevrede hem de endüstride önemli ilgi görmüştür. Bu nedenle, monolitik uygulamanın mikroservis mimarisinin uygulanması konusunda ve mikroservis mimarisinin bulut bilişim, Nesnelerin İnterneti (IoT) gibi gelişmekte olan teknolojilere uygulanması konularında da önemli çalışmalar bulunmaktadır.

Bu tezin literatüre katkısını belirtmek için bu tez çalışmasına benzer çalışmalar seçilmiştir; özellikle, monolitik uygulamaların mikroservis tabanlı uygulamalara dönüştürülmesine yönelik çalışmalar ele alınmıştır.

Çizelge 5.1 Önceki Çalışmalar ve Tezin Karşılaştırılması

Önceki Konular	Bu Tezden Farklılıkları
Monolitik uygulamanın mikroservise dönüşümünün benzer bir çalışması (Escobar vd., 2016)'te sunulmuştur. Çalışma, mevcut bir uygulamayı anlamak için uygulama geliştirme diyagramları oluşturmak üzere kaynak kodlarını kullanan bir yaklaşımı, mikroservislere bölmek amacıyla tartışmaktadır.	Çalışma sadece mevcut monolitik uygulamayı anlamakla sınırlıdır. Ancak bu tez, uygulama katmanının ötesinde her bir mikroservisin kalıcı katmanının(veritabanının) ötesine geçmektedir.

Çizelge 5.1 Önceki Çalışmalar ve Tezin Karşılaştırılması (devamı)

(Kazanavicius ve Mazeika, 2019)'de, eskiden kalan sistemin mikroservis uygulamasına taşınması için mevcut teknik ve yöntemlerle ilgili avantajların ve dezavantajların karşılaştırmalı bir analizi sunulmaktadır.	Burada, mevcut taşıma tekniklerinin karşılaştırmalı bir analizi, literatüre yeni bir şey eklemeden hem avantajları hem de dezavantajları ile birlikte sunulmaktadır. Öte yandan, bu tez çalışması, mevcut monolitik bir uygulamanın bir mikroservis mimarisine dönüştürülmesi konusunda, literatürdeki mevcut bilgi kütlesine yeni bir metodoloji ile katkıda bulunmuştur.
Monolitik mimari ve mikroservis mimarisinin performansları, yazarların farklı değişkenlere bağlı olarak karşılaştırdıklarında farklılık gösterdiği (Al-debagy ve Martinek, 2018)'da sunulmuştur.	Aradaki fark, (Lampson vd., 1993)'da mikroservis mimarisinin performans ölçümü sunulurken, bu tezde mevcut bir monolitik uygulamadan mikroservise veri taşınması için bir metodoloji sunulmaktadır.

Çizelge 5.1 Önceki Çalışmalar ve Tezin Karşılaştırılması (devamı)

(Gouigoux ve Tamzalit, 2017)'de bir endüstriyel projeden öğrenilen teknik dersler sunulmaktadır. Proje, monolitik bir işletme sisteminin üç yıl içinde 17 300 iş-gününü içeren bir mikroservis uygulamasına taşınmasıdır. Ders, mikroservislerin en uygun ayrıntı düzeyi, en uygun dağıtım ve en verimli düzenleme ile ilgili teknik değerlendirmelerine dayanmaktadır.	Bu tezde, mevcut bir monolitik uygulamadan veri taşınması için bir metodoloji sunulmaktadır; metodoloji, veri taşınması uygulamasının dağılımını içeren faaliyetleri kapsamaktadır. Bununla birlikte, bu çalışmadaki rapor öncelikle her bir mikroservisin boyutunun (ayrıntı düzeyinin) ne kadar uygun olduğuna, bunların mikroservis uygulamasını oluşturan tüm hizmetlere dağıtımına ve kontrolüne (orkestrasyonuna) dayanmaktadır.
Bir tez çalışmasında (Faradj, 2018), ayrışmış monolitik uygulamanın mikroservis mimarisine dayalı ayrı hizmetlerle çalışma sürecine etkisi değerlendirilmiştir. Yazar, ayrışmanın hem çalışma zamanını hem de dış iletişimini etkilediği sonucuna varmıştır.	Monolitik uygulamanın çalışma süresi etkisi, ayrışan mikroservis uygulaması ile karşılaştırılmaktadır. Aynı zamanda, bu tez mevcut bir monolitik uygulamadan bir mikroservis uygulamasına veri taşınması için gerekli olan metodolojiyi sunmaktadır.

Çizelge 5.1 Önceki Çalışmalar ve Tezin Karşılaştırılması (devamı)

Saga mimarisi, (“Sagas,” n.d.)'da, daha fazla yayılan bir işlem içeren veritabanlarından birinde yerel bir işlemin başarısız olması durumunda, telafi edici özelliklerin uygulama boyunca veri tutarlılığını korumak için ters işlevler olduğu hizmetlere telafi edici işlevlerin sağlandığı (“Sagas,” n.d.)' da sunulmaktadır.	Burada, bir mikroservis uygulamasında veri tutarlılığını korumak için bir mimari (Saga) sunulmaktadır. Bu tezin diğerlerinden farklı olmasının sebebi çalışmanın kapsamıdır. Veri tutarlılığı, önerilen yöntemin Hizmet Başına Veritabanı Tasarımı içindeki bir faaliyettir. Yine de Saga mimarisi sunulan metodolojide de önerilmektedir, bu nedenle Saga tekniklerini önerilen metodoloji içinde uygulamak mümkündür.
Monolitik Web uygulamasının mikroservis uygulamasına otomatik olarak bölünmesi (Abdullah vd., 2019)'de, minimum maliyetle performansı en üst düzeye çıkarmak için uygun kaynakları sağlamaya yönelik olarak sunulmaktadır.	Çalışma, monolitin otomatik olarak ayrıştırılmasını ve performansı arttırmak ve para maliyetini azaltmak için ayrıştırılmış hizmetlere kaynakların dinamik olarak tahsis edilmesini sunmaktadır. Bununla birlikte, bu tez bir mikroservis uygulamasına veri taşınması için önerilen metodoloji ile ilgilidir.

Çizelge 5.1 Önceki Çalışmalar ve Tezin Karşılaştırılması (devamı)

Mikroservis gelişmeleriyle ilişkili avantajlar ve dezavantajlar tam olarak belgelenmiştir (Soldani vd., 2018).	Bu çalışma, literatüre yeni bilgi eklemeden avantaj ve dezavantajları belgelemektedir. Öte yandan, bu tez, mevcut mikroservis literatürüne önerilen bir metodolojiyle katkıda bulunmaktadır.
Yarı otomatik veri akışı odaklı yaklaşım (Li vd., 2019)'te sunulmaktadır. İş mantığını veri akışı yaklaşımı yönlendirmektedir	Burada monolitik bir uygulamayı bir mikroservis uygulamasına ayırmak için bir yöntem sunulmaktadır. Ancak, prosedür iş mantığının veri akışı ile sınırlıdır. Bu tezde sunulan yöntem, metodoloji boyunca gerekli görevleri yerine getirmek için çeşitli yöntemler ve ortaya koymaktadır. Dolayısıyla, tek bir yaklaşım ile sınırlı değildir.

Çizelge 5.1 Önceki Çalışmalar ve Tezin Karşılaştırılması (devamı)

Mikroservis ve Docker konteynırı kullanarak bir uygulamanın buluta dağıtımının optimizasyonu (Wan, Guan, Wang, Bai, ve Choi, 2018)'te sunulmaktadır. Optimizasyon hem dağıtım hem de işletim maliyetine odaklanmaktadır.	Uygulama dağıtımının ekonomik maliyetini azaltmak için optimizasyon, mikroservis mimarisi ve konteynerizasyon teknolojisi kullanılarak sunulmaktadır. Diğer yandan tez, mevcut bir monolitik uygulamanın ve bununla ilişkili veri tabanının bir mikroservis kullanımına taşınması için önerilen bir metodoloji sunmaktadır.
Burada veritabanının kendisi, iş mantığını kendi içinde sarmalayan mikroservis olarak kabul edilmektedir ve veritabanı bir hizmet kalıbıdır (Messina, Rizzo, Storniolo, ve Urso, 2016).	Çalışmayı gerçekleştirmek için taranan kaynaklar ve okunan çalışmalar içerisindeki en ilginç çalışma, bu tezde sunulan yöntemdir. Veritabanının kendisinin uygulama mantığını kapsayan, böylece bir uygulama katmanını ortadan kaldıran bir yaklaşım önermektedir. Diğer taraftan, bu tez, uygulama katmanı ve her hizmete özel kalıcı bir katmanla ile ilgilenen bir metodoloji sunmaktadır.

5.3 Sonuç

Kaynak bölümünde sunulduğu üzere, mikroservis mimarisinin; bulut mühendisliği, Nesnelerin İnterneti, yazılım mühendisliği ve yazılım tabanlı gelişen teknolojiler üzerinde kullanımı artmaktadır. Diğer nedenlerin yanı sıra, eskiden kalan yazılım sistemlerinin modernizasyonu mikroservis mimarisinin gelişmesi anlamında en önemli motivasyonlardan biridir. Eskiden kalan monolitik sistemlerin modern mikroservislere dönüştürülmesi ve modernizasyonu konusunda kayda değer çalışmalar bulunmaktadır. Bu tez çalışmasında, bu

tür monolitik eski sistemlerin, veritabanı (kalıcı) katmanı içererek uygulama katmanının ötesine geçecek şekilde dönüşümünü genişletme sorununa çözüm üretilmiştir. Bu amaçla, hem monolitik eskiden kalan bir uygulamanın hem de verilerinin mikroservis mimarisine taşınması için bir yöntem ortaya koyulmuş ve bu yöntemin doğruluğunun ispatı için bir vaka çalışması sunulmuştur.

Vaka çalışması, NoSQL veritabanı teknolojisini kullanan monolitik bir mimariye sahip hayali bir kütüphane sistemidir. Monolitik kütüphane uygulamasını, mikroservis mimarisi boyunca veri tutarlılığını korurken veri çoğaltması olmadan üç mikroservise dönüştürmek için önerilen yöntem başarıyla uygulanmıştır. Her mikroservis kalıcı katmanda MongoDB veritabanı teknolojisini çalıştırır.

Ancak, mevcut yazılım teknolojileri ve metodolojileri gibi, önerilen metodolojinin de sınırları vardır. Önerilen metodolojinin kullanımı, monolitik kurumsal eski sistemlerin mikroservis mimarisine dönüştürülmesi ve modernizasyonu ile sınırlıdır.

KAYNAKLAR

Abdullah, M., Iqbal, W., ve Erradi, A. (2019). Unsupervised learning approach for Web application auto-decomposition into mikroserviss. *Journal of Systems and Software*, *151*, 243–257. https://doi.org/10.1016/j.jss.2019.02.031

Al-debagy, O., ve Martinek, P. (2018). A Comparative Review of Mikroserviss and Monolithic Architectures. *18th IEEE International Symposium on Computational Intelligence and Informatics*, 149–154.

Al-Masri, E. (2019). Enhancing the Mikroserviss Architecture for the Internet of Things. In *Proceedings - 2018 IEEE International Conference on Big Data, Big Data 2018*. https://doi.org/10.1109/BigData.2018.8622557

Alshuqayran, N., Ali, N., ve Evans, R. (2016). A systematic mapping study in mikroservis architecture. In *Proceedings - 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications, SOCA 2016*. https://doi.org/10.1109/SOCA.2016.15

Ampatzoglou, A., Frantzeskou, G., ve Stamelos, I. (2012). A methodology to assess the impact of design patterns on software quality. *Information and Software Technology*. https://doi.org/10.1016/j.infsof.2011.10.006

Bahaj, M., ve El Alami, A. (2013). The migration of data from a Relational Database (RDB) to an Object Relational (ORDB) Database. *Journal of Theoretical and Applied Information Technology*.

Balalaie, A., Heydarnoori, A., ve Jamshidi, P. (2016). Mikroserviss Architecture Enables DevOps: Migration to a Cloud-Native Architecture. *IEEE Software*, *33*(3), 42–52. https://doi.org/10.1109/MS.2016.64

Becker, S. A., Carmel, E., ve Hevner, A. R. (1993). Integrating joint application development (JAD) into cleanroom development with ICASE. *Proceedings of the Annual Hawaii International Conference on System Sciences*, *3*, 13–21. https://doi.org/10.1109/HICSS.1993.284290

Bennett, K. H., ve Rajlich, V. T. (2000). Software maintenance and evolution. In *Proceedings of the conference on The future of Software engineering - ICSE '00*. https://doi.org/10.1145/336512.336534

Binz, T., Leymann, F., ve Schumm, D. (n.d.). CMotion : A Framework for Migration of Applications into and between Clouds. *2011 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, 1–4. https://doi.org/10.1109/SOCA.2011.6166250

Bisbal, J., Lawless, D., Wu, B., Grimson, J., Wade, V., Richardson, R., ve Sullivan, D. O. (1997). An Overview of Legacy Information System Migration, 529–530.

Bourque, P., Dupuis, R., Abran, A., Moore, J. W., ve Tripp, L. (1999). Guide to the software engineering body of knowledge. *IEEE Software*. https://doi.org/10.1109/52.805471

Candela, I., Bavota, G., Russo, B., ve Oliveto, R. (2016). Using Cohesion and Coupling for Software Remodularization. *ACM Transactions on Software Engineering and Methodology*. https://doi.org/10.1145/2928268

Charman-Anderson, S. (2015). Ada lovelace: Victorian computing visionary. *Ada User Journal*.

Chen, R., Li, S., ve Li, Z. (2018). From Monolith to Mikroserviss: A Dataflow-Driven Approach. In *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*. https://doi.org/10.1109/APSEC.2017.53

Creswell, J. W. (2009). *Research Design*. (3 rd Edtion, Ed.). Calif: SAGE.

Curbera, F., Duftler, M., Khalaf, R., ve Nagy, W. (2002). Unraveling the Communication : SOAP. *Ieee Internet Computing*, (April), 86–93. https://doi.org/10.1109/4236.991449

Database per service. (n.d.). Retrieved November 21, 2019, from https://mikroserviss.io/patterns/data/database-per-service.html

Decompose by subdomain. (n.d.). Retrieved January 9, 2020, from https://mikroserviss.io/patterns/decomposition/decompose-by-subdomain.html

Di Francesco, P., Lago, P., ve Malavolta, I. (2018). Migrating Towards Mikroservis Architectures: An Industrial Survey. *Proceedings - 2018 IEEE 15th International Conference on Software Architecture, ICSA 2018*, (Section VII), 29–38. https://doi.org/10.1109/ICSA.2018.00012

Duarte, C. H. C., ve Bank, B. D. (2016). Requirements Engineering for the Digital Transformation. *2016 IEEE 24th International Requirements Engineering Conference (RE)*, 4–5. https://doi.org/10.1109/RE.2016.21

Ebert, C., ve Duarte, C. H. C. (2018). Digital Transformation. *IEEE Software*, *35*, 16–21. https://doi.org/10.1109/MS.2018.2801537

Elalami, A. (2014). The Road to a Full Migration of Relational Database (RDB) to Object Relational Database (ORDB): Semantic Enrichment , Target Schema , Data Mapping, *30*(30).

Escobar, D., Diana, C., Amarillo, R., Castro, E., Garc, K., Parra, C., ve Casallas, R. (2016). Towards the Understanding and Evolution of Monolithic Applications as Mikroserviss. *XLII Latin American Computing Conference (CLEI)*. https://doi.org/10.1109/CLEI.2016.7833410

Fan, W., Han, Z., Zhang, Y., ve Wang, R. (2018). Method of Maintaining Data Consistency in Mikroservis Architecture. In *Proceedings - 4th IEEE International Conference on Big Data Security on Cloud, BigDataSecurity 2018, 4th IEEE International Conference on High Performance and Smart Computing, HPSC 2018 and 3rd IEEE International Conference on Intelligent Data and Securit*. https://doi.org/10.1109/BDS/HPSC/IDS18.2018.00023

Faradj, R. (2018). *The run-time impact of business functionality when decomposing and adopting the mikroservis architecture*. KTH Royal Institute of Technology. Retrieved from http://kth.diva-portal.org/smash/record.jsf?pid=diva2%3A1253472vedswid=2322

Gan, Y., ve Delimitrou, C. (2018). The architectural implications of cloud mikroserviss. *IEEE Computer Architecture Letters*. https://doi.org/10.1109/LCA.2018.2839189

Gateway Aggregation pattern - Cloud Design Patterns | Microsoft Docs. (n.d.). Retrieved November 11, 2019, from https://docs.microsoft.com/en-us/azure/architecture/patterns/gateway-aggregation

Gouigoux, J. P., ve Tamzalit, D. (2017). From monolith to mikroserviss: Lessons learned on an industrial migration to a Web oriented architecture. In *Proceedings - 2017 IEEE International Conference on Software Architecture Workshops, ICSAW 2017: Side Track Proceedings*. https://doi.org/10.1109/ICSAW.2017.35

Guo, D., Wang, W., Zeng, G., ve Wei, Z. (2016). Mikroserviss architecture based cloudware deployment platform for service computing. In *Proceedings - 2016 IEEE Symposium on Service-Oriented System Engineering, SOSE 2016*. https://doi.org/10.1109/SOSE.2016.22

Heinrich, R., van Hoorn, A., Knoche, H., Li, F., Lwakatare, L. E., Pahl, C., … Wettinger, J. (2017). Performance Engineering for Mikroserviss, 223–226. https://doi.org/10.1145/3053600.3053653

Hohenstein, U. (2000). Supporting Data Migration between Relational and Object-Oriented Databases Using a Federation Approach. *Database Engineering and Applications Symposium*, 371–379.

Jamshidi, P., Pahl, C., Mendonca, N. C., Lewis, J., ve Tilkov, S. (2018). Mikroserviss: The journey so far and challenges ahead. *IEEE Software*. https://doi.org/10.1109/MS.2018.2141039

Jarwar, M. A., Kibria, M. G., Ali, S., ve Chong, I. (2018). Mikroserviss in Web objects enabled IoT environment for enhancing reusability. *Sensors (Switzerland)*. https://doi.org/10.3390/s18020352

Kazanavicius, J., ve Mazeika, D. (2019). Migrating Legacy Software to Mikroserviss Architecture. *2019 Open Conference of Electrical, Electronic and Information Sciences (EStream)*, 1–5.

Khatri, V. (2020). Managerial work in the realm of the digital universe : The role of the data triad. *Business Horizons*, *59*(6), 673–688. https://doi.org/10.1016/j.bushor.2016.06.001

Kitchenham, B., ve Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering. *Engineering*. https://doi.org/10.1145/1134285.1134500

Knoche, H., ve Hasselbring, W. (2019). Drivers and Barriers for Mikroservis Adoption - A Survey among Professionals in Germany. *Enterprise Modelling and Information Systems Architectures (EMISAJ) - International Journal of Conceptual Modeling*, *14*(1), 1–35. https://doi.org/10.18417/emisa.14.1

Krylovskiy, A., Jahn, M., ve Patti, E. (2015). Designing a Smart City Internet of Things Platform with Mikroservis Architecture. In *Proceedings - 2015 International Conference on Future Internet of Things and Cloud, FiCloud 2015 and 2015 International Conference on Open and Big Data, OBD 2015*. https://doi.org/10.1109/FiCloud.2015.55

Lampson, B. W., Lampson, B. W., Lomet, D., ve Lomet, D. (1993). A New Presumed Commit Optimization for Two Phase Commit. *19th VLDB Conference*.

Larrucea, X., Santamaria, I., Colomo-Palacios, R., ve Ebert, C. (2018). Mikroserviss. *IEEE Software*. https://doi.org/10.1109/MS.2018.2141030

Laszewski, T., ve Nauduri, P. (2011). Migrating Applications to the Cloud. In *Migrating to the Cloud*. https://doi.org/10.1016/b978-1-59749-647-6.00008-9

Lewis, James; Fowler, M. (2014). Mikroserviss - A definition of this new architectural term. Li, S., Zhang, H., Jia, Z., Li, Z., Zhang, C., Li, J., … Shan, Z. (2019). A dataflow-driven approach to identifying mikroserviss from monolithic applications. *Journal of Systems and Software*, *157*. https://doi.org/10.1016/j.jss.2019.07.008

Lloyd, W., Ramesh, S., Chinthalapati, S., Ly, L., ve Pallickara, S. (2018). Serverless computing: An investigation of factors influencing mikroservis performance. In *Proceedings - 2018 IEEE International Conference on Cloud Engineering, IC2E 2018*. https://doi.org/10.1109/IC2E.2018.00039

Maatuk, A., Ali, A., ve Rossiter, N. (2008). Citation : Maatuk , Abdelsalam , Ali , Akhtar and Rossiter , Nick (2008) An Integrated Approach to Relational Database Migration . In : International Conference on Information and Communication Technologies (IC-ICT 2008), 27 August 2008 , Bannu , Paki, (August).

Marzullo, F. P., Souza, J. M. De, Blaschek, J. R., ve Janeiro, R. De. (2008). A Domain-Driven Development Approach for Enterprise Applications , using MDA , SOA and Web Services COPPE / Sistemas , Federal University of Rio de Janeiro , 432–437. https://doi.org/10.1109/CEC/EEE.2008.40

Messina, A., Rizzo, R., Storniolo, P., ve Urso, A. (2016). (A) A Simplified Database Pattern for the Mikroservis Architecture. *The Eighth International Conference on Advances in Databases, Knowledge, and Data Applications*, (June), 35–40. https://doi.org/10.13140/RG.2.1.3529.3681

Mikroserviss. (n.d.). Retrieved January 7, 2020, from https://martinfowler.com/articles/mikroserviss.html

Mikroserviss Architecture Design and Best Practices - XenonStack. (n.d.). Retrieved January 10, 2020, from https://www.xenonstack.com/insights/mikroserviss/

Mikroserviss vs. SOA — Is There Any Difference at All? (n.d.). Retrieved January 7, 2020, from https://medium.com/@kikchee/mikroserviss-vs-soa-is-there-any-difference-at-all-2a1e3b66e1be

Mikroserviss vs SOA | What's the Difference | Edureka. (n.d.). Retrieved January 7, 2020, from https://www.edureka.co/blog/mikroserviss-vs-soa/

Mohammadi, M., ve Mukhtar, M. (2018). Service-oriented architecture and process modeling. *2018 International Conference on Information Technologies, InfoTech 2018 - Proceedings*, (46116), 1–4. https://doi.org/10.1109/InfoTech.2018.8510730

Monk, S., Mariani, J. A., Elgalal, B., ve Campbell, H. (1996). Migration from relational to object-oriented databases. *Information and Software Technology*, *38*(7), 467–475. https://doi.org/10.1016/0950-5849(95)01090-4

Niu, Y., Liu, F., ve Li, Z. (2018). Load Balancing Across Mikroserviss. In *Proceedings - IEEE INFOCOM*. https://doi.org/10.1109/INFOCOM.2018.8486300

Niyomthum, K., ve Chittayasothorn, S. (2003). A TRANSFORMATION FROM AN OBJECT DATABASE TO AN OBJECT RELATIONAL DATABASE. *Proceedings IEEE Southeast Conference*, 7–11.

Patterns for distributed transactions within a mikroserviss architecture - Red Hat Developer.

(n.d.). Retrieved January 9, 2020, from https://developers.redhat.com/blog/2018/10/01/patterns-for-distributed-transactions-within-a-mikroserviss-architecture/

Pautasso, C., ve Wilde, E. (2010). RESTful Web services. In *Proceedings of the 19th international conference on World wide Web - WWW '10*. https://doi.org/10.1145/1772690.1772929

Reasons to use Composition over Inheritance in PHP – Amit Merchant – A blog on PHP, JavaScript and more. (n.d.). Retrieved January 11, 2020, from https://www.amitmerchant.com/reasons-use-composition-over-inheritance-php/

Riaz, M., Mendes, E., ve Tempero, E. (2009). A Systematic Review of Software Maintainability Prediction and Metrics. *Third International Symposium on Emprical Software Engineering and Measurement.*, 367–377.

Robbes, R., Oliveto, R., ve Di Penta, M. (2016). Guest editorial: special section on software reverse engineering. *Empirical Software Engineering*, *21*(3), 749–752. https://doi.org/10.1007/s10664-016-9433-9

Rocha, L., Vale, F., Cirilo, E., Barbosa, D., ve Mourão, F. (2015). A framework for migrating relational datasets to NoSQL. *Procedia Computer Science*, *51*(1), 2593–2602. https://doi.org/10.1016/j.procs.2015.05.367

Rodriguez, A. (2008). Restful Web services: The basics. *Online Article in IBM DeveloperWorks Technical Library*.

Rosen, M. (2009). Service-oriented architectures. In *The Decision Model: A Business Logic Framework Linking Business and Technology*. https://doi.org/10.1201/9781420082821
Rossiter, N. (2010). Converting Relational Databases into Object- relational Databases. *Journal of Object Technology*, *9*, 145–161.

Rothermel, G., ve Harrold, M. J. (1996). Analyzing regression test selection techniques. *IEEE Transactions on Software Engineering*. https://doi.org/10.1109/32.536955

Runeson, P., ve Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 131–164. https://doi.org/10.1007/s10664-008-9102-8

Sagas. (n.d.). Retrieved November 12, 2019, from https://mikroserviss.io/patterns/data/saga.html

Salonen, A., ve Deleryd, M. (2011). Cost of poor maintenance. *Journal of Quality in Maintenance Engineering*. https://doi.org/10.1108/13552511111116259

Scavuzzo, M., Nitto, E. Di, ve Ceri, S. (2014). Interoperable data migration between NoSQL columnar databases. *2014 IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations*, 154–162. https://doi.org/10.1109/EDOCW.2014.32

Schön, E. M., Thomaschewski, J., ve Escalona, M. J. (2017). Agile Requirements Engineering: A systematic literature review. *Computer Standards and Interfaces*, *49*, 79–91. https://doi.org/10.1016/j.csi.2016.08.011

Senagi, K. M., Okeyo, G., Cheruiyot, W., ve Kimwele, M. (2016). An aggregated technique for optimization of SOAP performance in communication in Web services. *Service Oriented Computing and Applications*, *10*(3), 273–278. https://doi.org/10.1007/s11761-015-0186-x

Shirazi, M. N. (2012). Design Patterns to Enable Data Portability between Clouds ' Databases. *2012 12th International Conference on Computational Science and Its Applications*, 117–120. https://doi.org/10.1109/ICCSA.2012.29

Singh, V., ve Peddoju, S. K. (2017). Container-based mikroservis architecture for cloud applications. In *Proceeding - IEEE International Conference on Computing, Communication and Automation, ICCCA 2017*. https://doi.org/10.1109/CCAA.2017.8229914

SOA versus mikroserviss: What's the difference? - Cloud computing news. (n.d.). Retrieved January 7, 2020, from https://www.ibm.com/blogs/cloud-computing/2018/09/06/soa-versus-mikroserviss/

Soldani, J., Tamburri, D. A., ve Van Den Heuvel, W. J. (2018). The pains and gains of mikroserviss: A Systematic grey literature review. *Journal of Systems and Software*, *146*, 215–232. https://doi.org/10.1016/j.jss.2018.09.082

Sommerville, I. (2010). *Software Engineering, Ninth Edition*. *Software Engineering*. Massachusetts: Addison-Wesley.

Strauch, S., Andrikopoulos, V., Bachmann, T., Karastoyanova, D., Passow, S., ve Vukojevic-haupt, K. (2013). Decision Support for the Migration of the Application Database Layer to the Cloud. *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, *1*, 639–646. https://doi.org/10.1109/CloudCom.2013.90

Sun, L., Li, Y., ve Memon, R. A. (2017). An open IoT framework based on mikroserviss architecture. *China Communications*. https://doi.org/10.1109/CC.2017.7868163

Technical Know-How: Service Oriented Architecture. (n.d.). Retrieved January 7, 2020, from http://techknowhowforyou.blogspot.com/2011/05/service-oriented-architecture.html?m=0

The Hardest Part About Mikroserviss: Your Data – Software Blog. (n.d.). Retrieved January 9, 2020, from https://blog.christianposta.com/mikroserviss/the-hardest-part-about-mikroserviss-data/

US6377640B2 - Means and method for a synchronous network communications system - Google Patents. (n.d.). Retrieved November 11, 2019, from https://patents.google.com/patent/US6377640B2/en

US8838808B2 - Asynchronous communication in Web applications - Google Patents. (n.d.). Retrieved November 11, 2019, from https://patents.google.com/patent/US8838808B2/en

Villamizar, M., Garces, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., ve Gil, S. (2015). Evaluating the monolithic and the mikroservis architecture pattern to deploy Web applications in the cloud. In *2015 10th Colombian Computing Conference, 10CCC 2015*. https://doi.org/10.1109/ColumbianCC.2015.7333476

Wan, X., Guan, X., Wang, T., Bai, G., ve Choi, B. Y. (2018). Application deployment using Mikroservis and Docker containers: Framework and optimization. *Journal of Network and Computer Applications*, *119*(July), 97–109. https://doi.org/10.1016/j.jnca.2018.07.003

Xing, L., ve Li, Y. (2010). Design and application of data migration system in hetertogeneous database. *Proceedings - 2010 International Forum on Information Technology and Applications, IFITA 2010*, *2*, 192–195. https://doi.org/10.1109/IFITA.2010.81

Yarygina, T., ve Bagge, A. H. (2018). Overcoming Security Challenges in Mikroservis Architectures. In *Proceedings - 12th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2018 and 9th International Workshop on Joint Cloud Computing, JCC 2018*. https://doi.org/10.1109/SOSE.2018.00011

Youn, C., ve Ku, C. S. (1992). Data migration. In *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*. https://doi.org/10.1109/ICSMC.1992.271615

Yu, D., Jin, Y., Zhang, Y., ve Zheng, X. (2017). A survey on security issues in services communication of Mikroserviss-enabled fog applications. *Concurrency Computation* . https://doi.org/10.1002/cpe.4436

Zalewski, J. (2003). Object-oriented software engineering. A use case driven approach. *Control Engineering Practice*. https://doi.org/10.1016/0967-0661(93)90235-j

Zhao, G., Lin, Q., Li, L., ve Li, Z. (2014). Schema Conversion Model of SQL Database to NoSQL. *2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, 355–362. https://doi.org/10.1109/3PGCIC.2014.13

Zheng, L., ve Wei, B. (2018). Application of mikroservis architecture in cloud environment project development. *MATEC Web of Conferences*. https://doi.org/10.1051/matecconf/201818903023

Zhou, W., Li, L., Luo, M., ve Chou, W. (2014). REST API design patterns for SDN northbound API. In *Proceedings - 2014 IEEE 28th International Conference on Advanced Information Networking and Applications Workshops, IEEE WAINA 2014*. https://doi.org/10.1109/WAINA.2014.153

EK 1 MİKROSERVİS UYGULAMALARI İÇİN RÖPORTAJ SORULARI

1. Mikroservis mimarisini ne zaman duydunuz?
2. Bunu nereden ve nasıl duydunuz?
3. Mevcut bir sistemi mikroservis tabanlı bir sisteme dönüştürme projesinde yer aldınız mı? Kaç defa?
4. Mevcut verilerin mikroservise taşınması ile ilgili hangi başlangıç planı veya değerlendirme dikkate alınmıştır?
5. Plan veya değerlendirme mevcut bir referanstan mı kaynaklandı - kitap, araştırma çalışması, metodoloji, vb.? Varsa, lütfen belirtin.
6. Veriler mevcut sistemden mikroservise nasıl taşındı?
7. Deneyim nasıldı?
8. Benzer bir projeyi tekrar yapacak olsaydınız - mevcut bir veriyi bir mikroservise taşısaydınız - bunu önceki projelerde yaptığınız gibi yapar mıydınız? Evet ise, neden? Değilse, neden yapmazdınız?

EK 2 Sipariş Mikroservisi

```
//dependencies configurations
const express          = require('express');
const mongoose         = require('mongoose');
const request          = require('request');
const bodyParser = require("body-parser");
const path             = require('path');
Order                  = require('./database-configuration/database-schema');
const app              = express();
// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');

// path and data conversion configurations
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(express.static(path.join(__dirname, 'public')));
app.use(bodyParser.json());

// database connection
mongoose.connect('mongodb://127.0.0.1:27017/orderService', {useUnifiedTopology:
true, useNewUrlParser: true, useCreateIndex: true });
// post an order

app.post('/orders', async (req, res) => {
try {
const order = await Order.create({
customerID: mongoose.Types.ObjectId(req.body.customerID),
bookID: mongoose.Types.ObjectId(req.body.bookID),
```

```
});
        res.send(order);
    } catch (error) {
        res.send(error);
    }

});

// list an order

app.get('/orders/:id', async (req, res) => {
const id = req.params.id;
try {
const getOrder = await Order.findById(id);
const customerID = getOrder.customerID;
const bookID = getOrder.bookID;

const customerUrl = 'http://localhost:5555/customers/';
const bookUrl = 'http://localhost:4545/books/';

request({url: customerUrl + customerID, json: true}, async (error, response) => {
const customer = await response.body.name;
console.log(customer);
request({url: bookUrl + bookID, json: true}, async (error, response) => {
const book =  await response.body.title;
const orderedObject = {
Customer: customer,
Book: book,
Order_Id: id
};
```

```
            res.send(orderedObject);
          });
        });
    } catch (error) {
        res.send(error);
    }
});

// delete an order
app.delete('/orders/:id', async (req, res) => {
const id = req.params.id;
try {
const deletedOrder = await Order.findByIdAndDelete(id);
        res.send(deletedOrder);
console.log(deletedOrder);
    } catch (error) {
        res.send(error);
    }

});

// port configuration
app.listen(7777, () => {
console.log('order service is up and running on port 7777');
});
```

Müşteri Mikroservisi

```
// dependencies configurations
const express          = require('express');
const mongoose         = require('mongoose');
const path             = require('path');
Customer               = require('./database-configurations/database-schema');
const app              = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');

// path and data conversion configurations
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(express.static(path.join(__dirname, 'public')));


// database connection
mongoose.connect('mongodb://127.0.0.1:27017/customerService',
{useUnifiedTopology: true, useNewUrlParser: true, useCreateIndex: true });

// create a customer
app.post('/customers', async (req, res) => {
try {
const newCustomer = await Customer.create(req.body);
   res.send(newCustomer);
console.log(newCustomer);
  } catch (error) {
   res.send(error);
```

```
  }
});

// get all customers
app.get('/customers', async (req, res) => {
try {
const allCustomers = await Customer.find({});
    res.send(allCustomers);

  } catch (error) {
    res.send(error);
  }
});

// get a single customer

app.get('/customers/:id', async (req, res) => {
const id = req.params.id;
try {
const customer = await Customer.findById(id);
    res.send(customer);
  } catch (error) {
    res.send(error);
  }
});

// update a single customer
app.patch('/customers/:id', async (req, res) => {
const id = req.params.id;
try {
```

```
const updateCustomer = await Customer.findByIdAndUpdate(id, req.body, {new: true,
runValidators: true});
   res.send(updateCustomer);
console.log(updateCustomer);
  } catch (error) {
   res.send(error);
  }
});

// delete a single user
app.delete('/customers/:id', async (req, res) => {
const id = req.params.id;
try {
const deletedCustomer = await Customer.findByIdAndDelete(id);
   res.send(deletedCustomer);
console.log(deletedCustomer);
  } catch (error) {
   res.send(error);
console.log(error);
  }
});

// port configuration
app.listen(5555, () => {
console.log('Server is up and running on port 5555')
});
```

Kitap Mikroservisi

```
// dependencies configurations
const express          = require('express');
const mongoose         = require('mongoose');
const path             = require('path');
Books                  = require('./database-configuration/database-Schema');
const app              = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');

// path and data conversion configurations
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(express.static(path.join(__dirname, 'public')));

// database connection
mongoose.connect('mongodb://127.0.0.1:27017/bookService', {useUnifiedTopology:
true, useNewUrlParser: true, useCreateIndex: true });
// post a new book
app.post('/books', async (req, res) => {
try {
const newBook = await Books.create(req.body);
   res.send(newBook);
console.log(newBook);
  } catch (error) {
   res.send(error);
  }
});
```

```
// retrieve all books
app.get('/books', async (req, res) => {
try {
const getAllBooks = await Books.find({});
   res.send(getAllBooks);

  } catch (error) {
   res.send(error);
  }
});

// retrieve a single book
app.get('/books/:id', async (req, res) => {
const id = req.params.id;
try {
const getBook =  await Books.findById(id);
   res.send(getBook);
console.log(getBook);
  } catch (error) {
   res.send(error);
  }
});

// update a book
app.patch('/books/:id', async (req, res) => {
const id = req.params.id;
try {
const updatedBook = await Books.findByIdAndUpdate(id, req.body,{new: true,
runValidators: true});
   res.send(updatedBook);
```

```
  } catch (error) {
    res.send(error);
  }
});

// delete a single book
app.delete('/books/:id', async (req, res) => {
const id = req.params.id;
try {
const deletedBook = await Books.findByIdAndDelete(id);
    res.send(deletedBook);
console.log(deletedBook);
  } catch (error) {
    res.send(error);
  }
});

// port configuration
app.listen(4545, () => {
console.log('Book services is up and running on port 4545');
});
```

ÖZGEÇMİŞ

Adı Soyadı:	Ibrahim Bakarr JALLOH.
Doğum Yeri:	Freetown, Sierra Leone.
Doğum Tarihi:	28 Eylül 1990
Medeni Durumu:	Evli
Yabancı Dili:	İngilizce ve Türkçe

Eğitim Durumu (Kurum ve Yıl)

Lise:	Sierra Leone Müslüman Kongresi Lisesi - Batı Afrika Okul Sertifikası(2007).
Lisans:	Siera Leone Üniversitesi - Fen Fakültesi Diploması ve Onur Belgesi (2013).
Yuksek Lisans:	Ankara Universitesi Fen Bilimleri Enstitusu Bilgisayar Mühendisliği Anabilim Dalı (2020)

Çalıştığı Kurum ve Yıl

Standard Charted Bank Sierra Leone (2012)