T.C.
DOĞUŞ ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

MİNİMUM UYGULANABİLİR ÜRÜN SÜREÇLERİ İÇİN UYGULAMA MİMARİSİ SEÇİM ÖNERİSİ

Yüksek Lisans Tezi

Tezi Hazırlayan

KORAY AYDIN

İstanbul, 2024

T.C.
DOĞUŞ ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

MİNİMUM UYGULANABİLİR ÜRÜN SÜREÇLERİ İÇİN UYGULAMA MİMARİSİ SEÇİM ÖNERİSİ

Yüksek Lisans Tezi

Tezi Hazırlayan

KORAY AYDIN

202191009001

0000-0002-5452-5481

Danışman

Dr. Öğr. Üyesi Mustafa Zahid GÜRBÜZ

İstanbul, 2024

YEMİN METNİ

Yüksek Lisans Tezi olarak sunduğum “Minimum Uygulanabilir Ürün Süreçleri İçin Uygulama Mimarisi Seçim Önerisi” başlıklı bu çalışmanın, bilimsel ahlak ve geleneklere uygun şekilde tarafımdan yazıldığını, bu tezdeki bütün bilgileri akademik ve etik kurallar içinde elde ettiğimi, yararlandığım eserlerin tamamının kaynaklarda gösterildiğini ve çalışmamın içinde kullanıldıkları her yerde bunlara atıf yapıldığını, patent ve telif haklarını ihlal edici bir davranışımın olmadığını belirtir ve bunu onurumla doğrularım. 10/01/2024

Koray AYDIN

	YÜKSEK LİSANS TEZ SINAV TUTANAĞI	*Doküman No*	FR.1.26
		Yürürlük Tarihi	1.11.2017
		Revizyon Tarihi	4.12.2020
		Revizyon No	2
		Sayfa	1 / 1

LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

Tarih : 09/05/2024

Anabilim/Anasanat Dalı : BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

Öğrencinin Adı Soyadı : KORAY AYDIN

Öğrenci No : 202191009001

Tez Danışmanının Adı Soyadı : Dr. Öğr. Üyesi M. Zahid Gürbüz

İkinci Tez Danışmanının Adı Soyadı :.....

Tezin Başlığı : MİNİMUM UYGULANABİLİR ÜRÜN SÜREÇLERİ İÇİN UYGULAMA MİMARİSİ SEÇİM ÖNERİSİ

Doğuş Üniversitesi Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliği'nin 32.Maddesi uyarınca yapılan değerlendirmeler sonunda;

☑ **tezin kabul edilmesine** ☐ **tezde düzeltme verilmesine** ☐ **tezin reddedilmesine**

oy birliği / ~~oy çokluğu~~ ile karar verilmiştir. Gereği için arz olunur.

Danışman Üye

Üye **Üye** **Üye**

Üye

Anabilim/Anasanat Dalı Başkanı Onayı:

Adı ve Soyadı : Koray Aydın
Danışmanı : Dr. Öğr. Üyesi Mustafa Zahid Gürbüz
Türü ve Tarihi : Yüksek Lisans Tezi, 10.01.2024
Alanı : Bilgisayar Mühendisliği
Anahtar Kelimeler : Yazılım mimarileri, altyapılar, minimum uygulanabilir ürün, analitik hiyerarşi süreci

ÖZET

MİNİMUM UYGULANABİLİR ÜRÜN SÜREÇLERİ İÇİN UYGULAMA MİMARİSİ SEÇİM ÖNERİSİ

Teknoloji gündelik hayatın ayrılamaz bir parçası haline gelmiştir. Özellikle süreçlerin dijitalleşmesi son kullanıcılara büyük konforlar sunmaktadır. Değişen ve sürekli artan talepler yazılım mimarilerinin de gelişmesine ve çeşitlenmesine zemin hazırlamaktadır. Tüm bunlarla beraber ürünün sunulacağı pazara hızlı ve sağlıklı bir şekilde çıkabilmesi de rakabet ve pazar adaptasyonu açısından önemli olmaktadır. Hal böyle olunca seçilecek yöntem ve mimarinin ürün, geliştirme ekipleri ve pazarın ihtiyaçları ile de uyumlu olması son derece önemlidir. Bu araştırmada özellikle ürünün ilk müşterilerine oluşturulacak çıktının geri bildirim alma, düzeltme yapma ve tekrar piyasaya sunma gibi süreçlerini tarifleyen minimum uygulanabilir ürün aşmasında, ihtiyaçları karşılayacak en optimum sonucu, çeşitli alternatif ve kriterler için sektör uzmanları tarafından yanıtlanan anket ile analitik hiyerarşi süreci kapsamında değerlendirilmiş ve öneri olarak sunucusuz yapılar tercihi makul bir çözüm olarak sunulmuştur.

Name and Surname : Koray Aydın
Supervisor : Assist.Prof. Mustafa Zahid Gürbüz
Degree and Date : Masters Degree, 10.01.2024
Major : Computer Engineering
Keywords : Software architectures, infrastructures, minimum viable product, analytical hierarchy process

ABSTRACT

APPLICATION ARCHITECTURE SELECTION PROPOSAL FOR MINIMUM VIABLE PRODUCT PROCESSES

Technology has become an integral part of daily life. In particular, the digitalization of processes offers great comfort to end users. Changing and ever-increasing demands lead up to the development and diversification of software architectures. In addition to all these, it is also important for the product to be introduced to the market quickly and safely in terms of competition and market adaptation. Therefore, it is extremely important that the method and architecture to be chosen to be accordant with the needs of the product, development teams and the market. In the minimum viable product, there are some stages such as receiving feedback, making corrections, remarketing. To find the most optimum result for these stages, surveys were conducted with sector experts, including various alternatives and criteria. The surveys were evaluated within the scope of analytical hierarchy process. In this research, it has been revealed that choosing serverless structures is a reasonable option.

İÇİNDEKİLER

Sayfa No.

TABLOLAR LİSTESİ

Sayfa No.

ŞEKİLLER LİSTESİ

Sayfa No.

KISALTMALAR

API	Application Programming Interface
AHP	Analytic Hierarchy Process
BAAS	Backend-as-a-Services
BFF	Backend for Frontend
CQRS	Command Query Responsibility Segregation
DDD	Domain Driven Development
DEVOPS	Development and Operation
FAAS	Function-as-a-Service
HTTP	Hypertext Transfer Protocol
IAAS	Infrastructure-as-a-Service
JAD	Joint Application Development
MVP	Minimum Viable Product
PAAS	Platform-as-a-Service
POC	Proof of Concept
RAD	Rapid Application Development
REST	Representational State Transfer
RTI	Rassal Tutarlılık İndeksi
S3	Simple Storage Service
SAAS	Software-as-a-Service
SDLC	Software Development Life Circle
SMTP	Simple Mail Transfer Protocol
TI	Tutarlılık İndeksi
TO	Tutarlılık Oranı

GİRİŞ

Günümüz dünyasında, teknoloji, hayatın bir parçası haline gelmiştir. Birçok sektörün hızla dijitalleştiğini görmek mümkündür. Bu gelişimi tetikleyen temel faktörlerden biri de kullanıcı talepleridir. Üretici gözüyle incelenirse, ürünün sağlıklı bir şekilde ve doğru zamanda kullanıcıya sunulması pazaryerinde önemli bir konumda olmayı sağlamak çok önemli olacaktır. Akıllı telefonların olmadığı ev artık yok gibi. Bu durum da pazarın hızla genişlediğini göstermektedir. Hâl böyle iken doğru ürünü geliştirmek kadar ürünü doğru bir şekilde geliştirmekte önem kazanmaktadır. Pazara sunulan ürünün dayanıklılığı, yeni özelliklerin geliştirilmesi ve kullanıcılara sunulabilme süreçleri de üretici gözüyle planlanan aşamalar arasında yerini almaktadır. Yazılım dünyası yaşanan süreç problemlerinden dersler çıkarıp gün sonunda yeni topluluklar ve bunların neticesinde de yeni metodolojiler üretmektedir. Süreç tasarımlarının başarılı uygulanabilmesi ürüne ve pazara bağlı olabileceği gibi aynı zamanda ekip yapısı, şirket kültürü gibi olgularla da doğrudan ilişkilidir. Tüm bunlarla beraber uygulama mimarisinin ortaya konulması yukarıda sayılan parametreler ile ele alınınca karar alma süreçlerinin daha da karmaşıklaşmasına sebebiyet vermektedir. Ürünün oluşması ve pazara sunuluşu her zaman önemli olmuştur ancak bir fikir değerlendirilmek üzere pazara sunulmuşsa, ilk geri bildirimler, müşteri adaptasyonun incelenmesi de ürünün yaşam süresi hakkında önemli ipuçları verecektir. Bu çalışmada literatür taraması olarak geliştirme metodolojilerinden başlayarak, mimariler ve dağıtımları karşılaştırmalı olarak ele alınmıştır. Sektör paydaşları ile yapılan anket çerçevesinde ise AHP yöntemi kullanılarak "Minimum Uygulanabilir Ürün" kavramı için geliştirme mimarisinin optimum verim alınabilecek şekilde seçilmesi amaçlanmıştır. Birinci bölümde literatür taramasına yer verilmiştir. İkinci bölümde ise mimari yapıların dağıtımları, altyapı ihtiyaçları ve çeşitli başlıklarda karşılaştırmalar ortaya koyulmuştur. Üçüncü bölümde analitik hiyerarşi süreçleri, minimum uygulanabilir ürün kavramlarının uygulanma yöntemleri, dördüncü bölümde ise araştırma sonuçları yorumlanmıştır.

BİRİNCİ BÖLÜM
LİTERATÜR TARAMASI

Mimari yapılar, geliştirme süreçlerinin tasarlandığı ve ürün, müşteri portföyü ve geliştirme ekiplerinin deneyim ve kabiliyetleri ile ele alınarak uygulanabilirliği değerlendirildikten sonra karar kılınan yapılardır. Uygulama bileşenlerinin birbirine entegre olduğu tek bir depo içerisinde bulunan yapılar "monolitik yapılar" olarak adlandırılır. Proje başlangıçları için tercih edilebilir olsa da uygulama büyüdükçe yapıyı anlamak ve yorumlamak zorlaşmaktadır (Seedat, v.d., 2023). Mikroservis mimarisinde, monolitik yapının aksine modüler bir kurgu mevcuttur. Daha önceden karar verilmiş bir mantık çerçevesinde her kod parçası ayrı geliştirilir ve birbirinden bağımsız şekilde sunucuya gönderilir. Geliştiriciler ürün oluşturmanın yanı sıra konteyner yapıları da oluştururlar ancak ana sunucu kaynakları yine de farklı ekiplerdedir. Mikroservisler ile geliştirilen yapılarda teknoloji değişimlerine daha kolay uyum sağlanabilmektedir. Böylelikle, zamanı geldikçe gerekli dönüşümler için çevik yapılar ortaya koyarlar. Bununla beraber üçüncü parti yazılım bileşenlerinin entegrasyonu da daha kolay uygulanabilmektedir. Mikroservislerin uygulanabilmesi şirket yapısıyla da ilgilidir. Bir servisin birden fazla istemciye hizmet verdiği durumlar için mikroservis çözümleri makul sonuçlar doğurabilmektedir. Böylelikle şirket yapısı alan bazlı dağıtılıp geliştiricilerin tek bir alana odaklanması sağlanabilir. Bir diğer yandan sunucu/bulut yapılarında birbirlerinden bağımsız geliştirilebilmeleri gibi birbirlerinden bağımsız konuşlanmaları da gerçekleşir. Bu da bağımsız dağıtımı sağlar. Bu kavramlar üzerine inşa edilmiş olması ürünün tamamında yaşanabilecek dayanıklılık sorunlarını minimize eder. Bir mikroservisin ne kadar küçük olacağı veya başka bir bakışla ne kadar büyüyebileceği bu mimarinin en temel tartışma konularından biridir (Jhingran ve Rakesh, 2023). Bir diğer yandan ise izleme açısından maliyetli olabilmektedir. Birçok servisin izlenmesi ve dayanıklılığından emin olunması gerekmektedir. Monolitik yapılara göre ağ gecikmeleri daha fazla görülebilmektedir. Bu durumda, yanıt süreleri açısından, servislerin konuşlandığı altyapılar (özellikle bulut sistemlerde) önem taşıyabilmektedir (Abd-Elwahab, v.d., 2023). Hızlı adapte olunabilecek ve canlı ortamda hazır hizmetler sunan

yapılar ise sunucusuz yapılar olarak adlandırılmaktadır. Geliştirilen kodlar hizmet sağlayıcısı tarafından sunucuya dahil olur. Böylelikle ürün geliştiricileri tarafından herhangi bir kimsenin altyapı bakımı ile uğraşmasına gerek kalmaz. Sunucusuz uygulamalar durum bilgisi olmayan (stateless) ortamlardır. Durum uygulama katmanında yönetilir. Nesne depolama protokoline (Örneğin: S3- Simple Storage Service) dosya yüklemek gibi olaylara yanıt verir. Bu mimaride geleneksel uygulama altyapı yönetimi yoktur. Ölçekleme konusunda da daha önceden belirlenmiş kurallara göre hareket eder. Böylelikle farklı bir planlama yapılmasına gerek kalmaz. Konteyner yapılara göre daha az esnektirler. Sağlayıcının çizdiği ve öngördüğü sınırlar içerisinde kalınır bu da verimliliği düşürebilir. Geliştirme perspektifinden ele alınırsa sağlayıcılar genellikle popüler dilleri destekler. Altyapı operasyon ihtiyaçları olmadığı için uygulama geliştirmeye odaklanmadaki verimi artırır (Veuvolu, v.d., 2023). Risk faktörü olarak ise bu mimarilerde sağlayıcıya bir bağımlılık mevcuttur. Servisler içerisinde geniş bir yelpaze olsa da sağlayıcının sunduğu çözüm ve kütüphaneler ile sınırlı bir dünya içerisine ürün inşa edilir. Bu durumda da olası bir taşınma söz konusu olduğunda çözülmesi gereken bir dizi sorun ile karşılaşılacaktır. Tüm bunlarla beraber uygulamanın ihtiyaç duyabileceği çeşitli servislerinde tüketilmesi mümkündür. Servis mesajlaşmalarında kullanılabilecek ve gönderme, alma gibi işlevleri yerine getirebilecek kuyruklama servisleri avantaj sağlayabilir (Miguel Rodriguez Cortes, v.d., 2022). Veritabanı için de ayrıca bakım ve yönetim gerektirmeyecek servislerde mevcuttur. Ürünün istemci ile arasına, giriş kapısı olarak adlandırılabilecek, API Gateway çözümleri de mevcuttur. Bu sayede HTTP üzerinden REST API çağrımları yapmakta bu mimaride mümkündür. Özetle bu ürünlerde kural tanımları hariç farklı bir yönetim ihtiyacı olmaz (Parres-Peredo, v.d., 2019).

Minimum uygulanabilir ürün (MVP- Minimum Viable Product) kavramı, müşteriden geri bildirim alabilmek ve ihtiyacı test etme amaçlı az efor ve temel özellik setiyle sunulan; geliştirme, ölçme ve öğrenmeye dayalı bir ürün geliştirme sürecini ifade eder. Bu döngüde yer alan kullanıcılar ürünü erken denemek isteyenler, ürünü erken deneyenlerden etkilenip denemek isteyenler ve ürünü kullanmaya direnenler şeklinde gruplanabilir. Karşılaştırmalı olarak ele alınırsa kavram kanıtı (POC- Proof of Concept), bir çalışmanın ya da fikrin uygulanabilirliğinin test edildiği bir gösterim sürecidir. Bu

süreçte projenin gerçekleşebilirliği, hangi teknolojinin kullanılacağı gibi parametreleri belirlemek için de önemlidir (Prasanna, v.d., 2021). Bu durumda ürünün olabilecek en az temel gereksinimleri ile kullanıcı ile buluşması ve oradan alınacak geri bildirimler ile iyileştirme yapılmasına veya geliştirmeden vazgeçilmesine karar verilmesi önemli adımlar olarak öne çıkmaktadır. Temel gereksinimler ele alınırsa, ürünün geliştirilmesi sürecinde seçilecek yazılım mimarisi buradaki çeviklik seviyesini ortaya çıkaracaktır. Temel çıkış noktası ise çözülmesi istenen problemin tespitidir. Fikir aslında çözülmesi gereken problemi temsil eder ve olabilecek en küçük çözüm ile sürece başlanır. Ürünü ilk kullanan kullanıcılar "ilk benimseyenler" olarak adlandırılır. Bu grup bazı temel ögeler ile ayrıştırılabilir. (Tripathi, 2019) Daha önce kullandığı bir üründen markaya sempatisi olanlar olabileceği gibi gerçekten sunulan çözüme ihtiyacı olan müşteriler de olabilir. Genel olarak ele alınacak olursa, yeni başlayan ürünler için önemli bir adım olabilmektedir. Bu sebeple fikrin ürüne dönüştüğü ilk aşamada bu kavramın uygulanması sıklıkla görülebilmektedir (Portman, 2022). Akış, fikrin ürüne dönmesi ile başlar ve geri bildirimler alır. Geri bildirimler doğrultusunda ürün/fikir sahibi geliştirmeye devam etme ya da geliştirmeyi durdurma kararları alabilir. Ürün geliştirmeye devam etme kararı alınırsa alınan geri bildirimler doğrultusunda geliştirmeler devam eder ve tekrar müşteriye sunulur. Bu durum karar vericinin ürünün olgunlaşmasına müsaade etmesine veya ürünü piyasadan çekmesine kadar devam eder (Prasanna,v.d., 2021). Benzer çalışmalar incelendiğinde (Hardy, 2023) henüz fikir aşamasında olan MVP süreçleri için monolitik yapıların kullanımının daha uygun olduğundan bahsedilmiştir.

Karar verme teknikleri, problem tanımında belirtilen öbeğin çözümü için, optimum performansı sunan alternatifleri olayın kriterleri ile değerlendiren ve sonuç olarak niceliksel veriler sunan metodolojilerdir. AHP diğer karar verme tekniklerinde olduğu gibi, karar probleminin tespit edilmesi, karar değişkenlerinin (kriterlerin) belirlenmesi ve alternatiflerin (seçeneklerin) ortaya koyulması ile başlar. Çok kriterli karar verme problemlerinde tutarlı olduğu gözlemlenmiştir. (Goyal, v.d., 2021; Wu, v.d., 2006)

İKİNCİ BÖLÜM
MİMARİ YAPILAR ve MVP

2.1 Mimari Yapılar

Teknoloji günümüz dünyasının vazgeçilmez unsuru haline gelmiştir. Hâl böyle iken teknoloji ürünü geliştirmek, hali hazırda pazarda olan ürünü iyileştirmek ve kullanıcı için sorunsuz bir deneyim sunma; ürün sahipleri için olmazsa olmaz unsurlar arasında yer almaktadır. Yazılım ile gelişen ve büyüyen teknoloji dünyasında ürünün mantığının yer aldığı dünya için, ihtiyaçlara göre, zaman içerisinde çeşitli mimariler ortaya çıkmıştır. Alt başlıklarda çeşitli perspektiflerden mimarilerin karşılaştırılmaları konu edinilmiş ve MVP gibi ürünün hızla canlı ortama çıkması gereken aşamalarında altyapı tasarımlarına daha az vakit ayırmanın efektif sonuçlar doğuracağı iddiası araştırılmıştır.

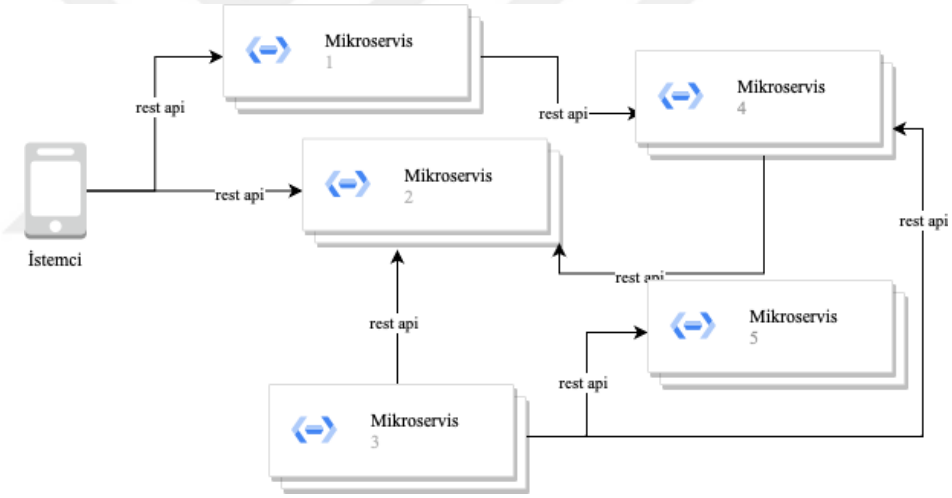
2.1.1 Monolitik Mimari

Ürünün tek bir uygulama paketinde toplandığı ve genel olarak geliştiriciler ile altyapı sorumlularının görünür bir şekilde ayrı çalıştığı yapılardır. (Pereira v.d., 2021) Geleneksel geliştirme paradigmasını temsil eder. Monolitik mimariye sahip bir uygulamada yapılacak değişiklik, uygulama içindeki bağlılıklar nedeniyle istenmeyen başka maliyetlerin ortaya çıkmasına sebebiyet verebilir. Yine aynı bağımlılıklar sebebiyle yeni teknolojilerin implementasyonu, dil güncellemeleri vb. faktörleri daha zorlaştırıcı bir hale getirebilir. Fikrin ilk ortaya çıkışının ürünleşmesi için daha az bağımlılık içermesiyle çevik bir yapı gibi görünse de uygulamanın büyümesi, müdahaleleri daha karmaşık hale getirebilmektedir. Büyüyen monolitik yapılar için müdahaleyi kolaylaştırmak için parçalı yapılara dönüşüm çalışmaları başlatılabilir. Bir diğer yandan, uygulamanın haberleşmesi, veri alışverişleri ve çerez yapılarının ortak kullanımları sebebiyle bazı avantajlarda taşımaktadır. Özellikle mikroservis mimarisi ile ortaya çıkan ve servis haberleşmelerinde oluşan ağ gecikmeleri bu yapılarda görünmez. (Mangwani, v.d., 2023)

2.1.2 Mikroservis Mimarisi

Mikroservisler; tek bir amaca hizmet eden, dil bağımsız geliştirilebilen ve tek başına dağıtıma çıkabilen küçük yapılardır. Bu yapılarda işlevleri izole etmek önemli bir avantaj oluşturabilir. Böylelikle birbirinden bağımsız fonksiyonlar, farklı ihtiyaçlara göre ölçeklenebilmeleri ve sorun çözümü daha basit hale gelebilir. Ayrıca her servis ayrı olarak dağıtılabildiği için sürekli teslimat süreçleri işletilebilir. Mikroservislerin geliştirildiği dil veya platform ilgili servisin çağrımı üzerinde herhangi bir bağımlılık oluşturmamalıdır. Genel olarak REST gibi yaygın kullanılan protokoller üzerinden çağrılır. (Tanasseri ve Rai, 2017)

Örnek bir akış modeli Şekil 1'de gösterilmiştir.



Şekil 1. Örnek Mikroservis Yapısı

Herhangi bir mimari yaklaşımı benimsemeden önce avantajlarını ve dezavantajlarını beraber değerlendirmek gerekir. Canlı ortama yeni özellikler veya bazı düzeltmeler çıkılacağı zaman kesinti süresinin çok düşük olması hizmet sağlayıcılar açısından kritik öneme sahiptir. Mikroservis yapısının birbirinden bağımsız çalışabiliyor oluşu; bir serviste bir sorun yaşandığında diğer hizmetlerin kesintiye uğramamasını sağlar. Bu durum genel bir kesintinin yaşanmasını engeller. Bir diğer kritik konu ise verilerin ayrıştırılabilmesi. Kritik bilgiler tutması gereken sektörler regülasyonları gereği ilgili bilgileri izole ederek ayrıca işlemeleri gerekebilir. Bu gibi durumlarda ayrı ve izole bir yapı kurmak çok önemli olabiliyor. Başka bir bakış açısıyla organizasyon yapıları arasında

bağımsız bir yapı kurulmasına olanak tanır. Farklı departmanlar altında geliştirme yapan ekipler birbirlerinden bağımsız bir şekilde ürün geliştirmeye devam edebilirler. Bununla beraber ekip seviyesinde ise daha küçük organizasyonlara ihtiyaç duyar. Böylelikle, geliştiriciler tek bir hizmete yönlenir ve üretkenliğin artmasına yardımcı olur. Teorik olarak mikroservis mimarisini tercih etmek basit bir karar olarak görünebilir ancak pratikte karmaşık bir yapıya bürünmesi de gayet olasıdır. Mikroservis yapıları bağımsız oluşları sebebiyle yaşam döngüleri içerisinde farklı araçlara ihtiyaç duyabilirler. Altyapı ekiplerinin farklı teknolojilere hâkim olması ve bakımını sürdürmesi anlamı çıkmaktadır. Bununla beraber geliştirilen mikroservislerin farklı teknolojiler kullanabilecek oluşu da ayrı bir bilgi birikime ihtiyaç duyulmasını sağlamaktadır. Bu durumu biraz daha açıklamak gerekirse; her bileşen geliştiricinin tercihlerine göre farklılaşabilir. Bu durumda ürün bakım aşamasına geçtiğinde, teknolojik çeşitlilik göz önüne alındığında daha fazla insan gücüne ihtiyaç duyulabilir. Özetle, hem bireysel becerilerin sürdürülebilir olması hem de teslimat gibi kısıtlayıcılar sebebiyle farklı maliyetlere de yol açabilir. Altyapı açısından ise mikroservisler birbirinden bağımsız dağıtılan ve nicelik olarak fazla sayıda olmaları sebebiyle bölünen modüler bir yapıyı ortaya çıkar. Bu durumda birden fazla altyapının bakımı ve güncel tutulması zorunluluğunu da beraberinde getirecektir. Bir diğer yandan, monolitik yapılarda sistem kendi içerisindeki haberleşmeyi bellek içinde sürdürebilirken, mikroservisler iletişimi bir ağ üzerinden sağlamak durumunda kalırlar. Bu durumda da mimarisi monolitikten mikroservise aynı şartlarda dönüşen bir ürün için yanıt sürelerinde uzamalar olasıdır. Sürdürülebilirlik, ölçeklenebilirlik, çeviklik, yönetilebilirlik gibi kavramlara uyum sağlamak istendiğinde, ürünün pazara çıkma süresinin kısaltılması ve maliyetlerin düşürülmesi gibi konular ön plana çıkmaktadır. Hali hazırda pazarda yer alan bir uygulamayı güncel bir teknoloji yığınıyla yazmak istendiğinde mikroservis yapısının tercih edilmesi rasyonel olabilir. Oturum açma, arama vb. birden çok kanaldan (mobil, internet gibi) çağrılabilecek merkezi işlevler var ise bu durumda da mikroservisler büyük avantaj sağlayacaktır. Uygulama üzerindeki modüler yapıların iş birimleri talepleri doğrultusunda yeni güncellemeler veya eski yapıların devre dışına alınması konusunda esnekliğe sahip olmak isteniyorsa yine mikroservis yapıları tercih edilebilir. Mikroservis yapılarının modüler bir yapıyı ortaya çıkardığı aşikardır. Bir

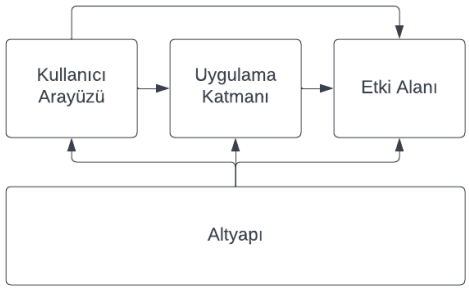
uygulamanın gerçekten modüler bir yapıya bölünmesinin bir ihtiyaç olup olmadığı sağlıklı bir şekilde sorgulanmalıdır. Mevcut kullanılan bir ürün yeteri kadar karmaşık değilse mikroservis yapıları doğru bir alternatif olamayabilir. Yeni bir özellik ekleme ihtiyacı sıklıkla doğmuyorsa ya da kritik ana işlevler sabit kalıyorsa mikroservis mimarisini tercih etmek daha fazla maliyet doğurabilir. Bir diğer yandan insan kaynakları olarak farklı programlama dillerinde veya teknoloji yığınlarında yetkin bir ekip yoksa bakım aşamasında ürünün sıklıkla kesilmesi de olasıdır. Özetle, bazı uygulamaların monolitik yapıda kalmaları, bir dönüşüm geçirmelerinden daha sağlıklı olabilir. Herhangi bir işlev sorun yaşadığında bütün süreç etkilenecekse, bu durumda da her fonksiyonu bir arada tutmak daha kıymetli olacaktır. Sonuç olarak, ürün geliştirme mimarisine karar vermek, ürünün tam olarak sorunsuz hale getirilebileceğinin kararını almak gözlemlenebilir bir sürece ihtiyaç duyacaktır. Dolayısıyla, mikroservis mimarisini tercih ederken gerçekten pazara çıkış süresini, ihtiyacı ve performans beklentisini iyi değerlendirmek gerekebilir. Günümüzde birçok kurumsal firma mikroservis mimarilerini tercih etmektedir. DevOps ekipleri mikroservislerin dağıtımını ve yönetimini sağlamaktadırlar (Preimesberger, 2016). Mikroservisler, bağımsız bir şekilde çalışabilecek yapılar şeklinde tasarlanırlar.

Mikroservis geliştirme desenleri; ayrıştırma, birleştirme ve veri tabanı desenleri olarak çeşitli ana başlıklarda değerlendirilebilir. Bu desenler ayrı ayrı düşünülebileceği gibi ihtiyaca göre bir arada kullanılması da mümkündür. Kullanılacak desene karar vermeden önce ihtiyaçlar ve sahip olunan altyapı tercihleri doğru sentezlenmelidir. Mikroservisler, yapıları gereği genel olarak sanal makineler içerisinde birbirlerinden bağımsız olarak konuşlanırlar. Bu izolasyonu sağlamak için ise konteyner sistemler ideal bir çözümdür. Böylelikle sanal makineleri daha verimli kullanmak için makul bir yapı ortaya çıkmış olur. Bununla beraber geliştirme tarafında ise ortaya çıkan ihtiyaçlara göre yıllara sâri değişimler söz konusu olmuştur. Böylelikle geliştirme desenleri de ortaya çıkmaktadır. (Norton Stanley ve Sebastian, 2023)

Etki Alanına Dayalı Tasarım (DDD- Domain-driven development) bu başlık altındaki en kritik desenlerden biridir. Bu yapıda ürünün kurum içinde ihtisas bazlı

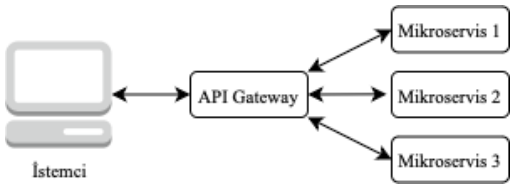
ayrışması esas alınır. Bu durumda eğer geliştirilen ürün bir bankacılık ürünü ise ödeme sistemleri ayrı bir yapı yatırım sistemleri ayrı bir yapıda yer alır. Böylelikle geliştirici ekiplerde de hem iş bazlı hem de teknoloji yetkinliği derinliği kazanılır. Bütün alt alanlar birbirleri ile servisler üzerinden haberleşirler. Birden fazla çözüm sunan ürünlerin etki alanları üzerinden modellenmesi için uygun bir yaklaşım olabilir. Yazılım mühendisliğinin karmaşıklığı, geleneksel yaklaşımları kullanarak ele alınması zor olan tasarımından kaynaklanmaktadır. Sunum, uygulama, etki alanı ve altyapı katmanı dahil olmak üzere DDD'nin dört katmanlı mimarisini önermiştir ve çok sayıda senaryoda yaygın olarak kullanılmaktadır. (Evans, 2004)

DDD, stratejik tasarım ve taktik tasarım olmak üzere iki süreç içerir. Stratejik tasarım, model ayrıştırma ve karmaşıklık kontrolüne odaklanarak problem alanını ayrıştırma yöntemidir. Örnek diyagram Şekil 2'de gösterilmiştir. Stratejik tasarımın önemli bir bileşeni olan sınırlı bağlam, çözüm alanının bir bölümüdür. Sınırlı bir bağlamdaki bileşenlerin ortak sorumlulukları vardır ve bu kapsamdaki anlamlar açıktır. Taktik tasarım, etki alanı modelinin her sınırlı bağlamda ayrıntılı tasarımı ve uygulanmasıdır. Varlık, kimliği olan bir nesnedir ve iş değişiklikleri taktik tasarım sürecinde izlenir. Etki alanı olayları, temel olarak model geçmişini ve sınırlar ötesi iletişimi belgelemek için alan uzmanlarının ilgisini çeken sorun alanında meydana gelen olayları temsil eder. Depolar, etki alanı varlıklarının kalıcılık işlemlerini altyapı katmanında kapsıyor ve etki alanı katmanını altyapı katmanından ayırıyor. (Marzullo, v.d., 2008)



Şekil 2. Mikroservis Desenleri- Ayrıştırma

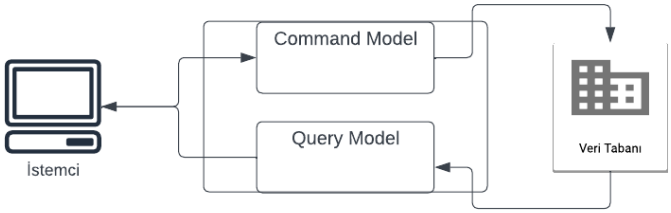
Bir uygulama küçük yapılardaki mikroservisler ile geliştirildiğinde bazı sorunlar ortaya çıkmaktadır. Ürün için birden fazla kullanım kanalı olduğunda birden fazla mikroservis için bu çağrıların gelmesi durumu trafiği yönetmek için bir soruna neden olabilir. API Gateway kullanımı bu desen için ön plana çıkan bir çözümdür. Bu durumda herhangi bir mikroservis çağrısı için tek giriş alanı API Gateway olur. Böylelikle, istekleri ilgili mikroservislere doğru trafik ile yönlendirmek için bir vekil (proxy) olarak konumlandırılabilir. Bununla beraber servis dönüşlerini de kullanıcıya tek elden yansıtabilir. Gösterimi Şekil 3'teki gibi oluşur. API Gateway konumlandırılırken fiyat, bakım ve performans gibi kriterler seçimlerde önemli rol oynayacaktır. (Zuo, v.d., 2020)



Şekil 3. Mikroservis Desenleri- Birleştirme

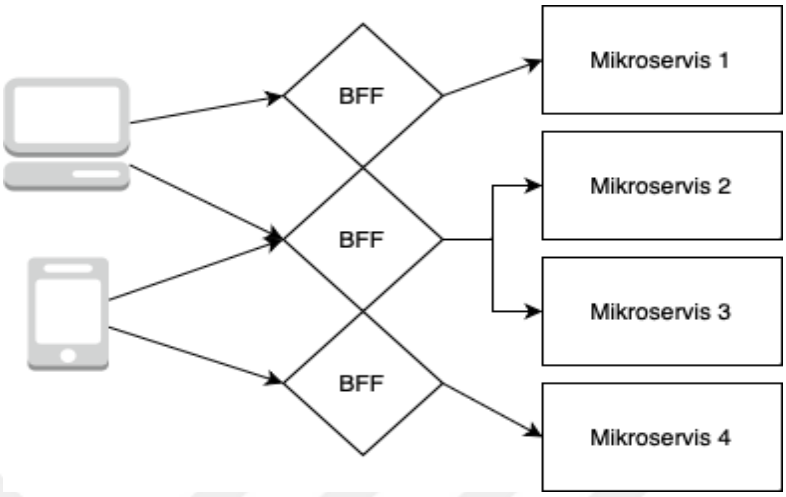
Her mikroservis için ayrı bir veri tabanı oluşturulduğunda geri dönüş sonuçları için verileri birleştiren bir sorgu ihtiyacı ortaya çıkabilir. Bu desendeki en kritik çözüm Komut

ve Sorgu Sorumluluklarının Ayrıştırılması (CQRS- Command Query Responsibility) olarak ortaya çıkmaktadır. Akış Şekil 4'te gösterilmiştir. Temel olarak güncelleme, ekleme veya silme gibi veri tabanı operasyonlarının komut modeli (command model) ile sorguların modelini (query model) ayırmayı önerir. Yazma ve okuma işlevleri için farklı veri tabanları seçimi bu modelin en belirgin örneklerinden biridir. (Debski, v.d., 2018)



Şekil 4. Mikroservis Desenleri- CQRS

Mikroservisler modern yazılım mimarilerinin kritik unsurları haline gelmeye başladı. Ön yüz için geliştirme deseninde (BFF- Backend for Frontend) ve (Şekil 5), verileri doğrudan geliştirme seviyesinden almak yerine bir API üzerinden iletişim kurmaya odaklanır. Böylelikle hem mobil hem de internet sayfası için tek bir mantık geliştirmek ve önyüzleri buna göre tasarlamak kolaylaşacaktır. Ön yüz için geliştirme deseni, ön yüz ile arkada çalışan yazılımın arasında bir arabirim olması temeline kurulur. Özetle, ön yüzden aldığı isteği arkadaki diğer servislere gönderir ve ön yüzün ihtiyaç duyduğu verileri toplayarak geri dönüş yapar. Böylelikle oluşan yeni arabirimle birden fazla ön yüz için aynı yazılım mantığı tercih edilebilir. (Rattanukul, v.d., 2023)



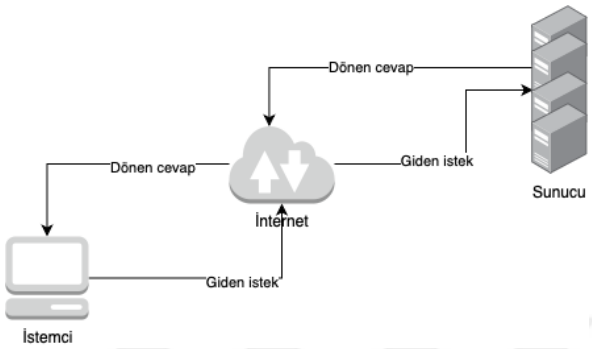
Şekil 5. Ön Yüz için Geliştirme Deseni

Mikroservis mimarilerinde koynteyner yapılar uygulama dağıtımı için önemli bir yer tutar. İşletim sistemleri içerisinde farklı bir katman oluşturarak kendi başlarına bir izolasyon konteyner yapıları ile sağlanabilir. Böylelikle bir işletim sistemi içerisinde de olsa izole ve küçük çaplı farklı bir katman içerisinde bir uygulama çalışabilir ve içerisinde bulunduğu altyapı hariç herhangi bir durumdan etkilenmezler. Bir uygulamayı konteyner içerisine yerleştirmek onu kendi içerisinde bağımlılıkları ile yöneterek tek başına çalışan bir sistem haline getirecektir. Birden fazla konteyner yönetmek için orkestrasyon desteğine ihtiyaç duyulur. (Liu, v.d., 2020)

2.2 Sunucusuz Mimari

Sunucusuz yapılar, sunucuların son kullanıcıdan tamamen soyutlandığı geliştirme ortamlarının sağlandığı alanlardır. Müşteriler için altyapıları kurmak, yönetmek ve bakımını sağlamak servis sağlayıcısının görevi olmakta; ürün geliştiren şirketler için tercih edilebilir bir bulut çözümü olarak ortaya çıkmaktadır. İstemci-Sunucu modeli uygulama altyapıları için tek çözüm olmasa da en yaygın tercihtir. Model özet olarak,

sunucuda yer alan bilgi ve hizmetleri talep eden istemci ile paylaşma üzere kurgulanır. Örnek bir diyagrama Şekil 6'da yer verilmiştir.



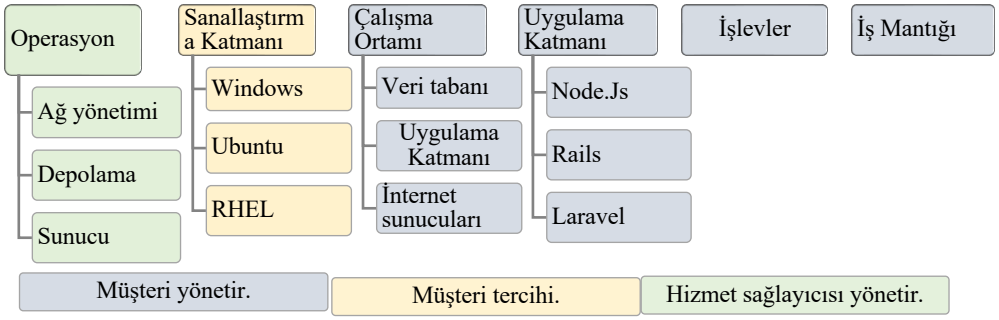
Şekil 6. Sunucu-İstemci Modeli

İstemciler yaygın olarak cep telefonları ve web uygulamaları olarak konumlandırabiliriz. Bunlarla beraber akıllı cihazlar, masaüstü uygulamalar da birer istemci örneğidir. Kullanıcının uygulamayı kullanmak veya hizmet almak için kullandığı cihaz istemci olarak isimlendirilir. Sunucu tarafı ise ürün ve hizmeti sunan kısımdır. Genellikle 4 ana başlıkta; dosya, veri tabanı, web ve internet sunucuları bulunmaktadır. Dosya sunucuları, diskleri üzerinde verileri kendisine erişebilme izni olan bütün ağlarla paylaşır. Bu tür sunucular genellikle üzerlerinde resim, ses, video ya da metinler tutarlar. Veri tabanı sunucuları ise üzerlerinde uygulamaya ait verilerin saklandığı veri tabanlarını bulundururlar. Web sunucular internete açık içeriklerin saklandığı yerler iken uygulama sunucularında ise uygulamaların çalıştığı sunucular denebilir. Her ürün sahibinin, geniş kaynaklara erişim imkânı olmayabilir. Bu gibi durumlarda internet üzerinden erişebilir bir sunucu hizmetleri makul bir çözüm olabilmektedir. İnternet üzerinde alınabilecek sunucu hizmetlerinde de çeşitli seçenekler olduğu görülmektedir. Aynı sunucunun birden fazla internet hizmetiyle paylaşılabileceği gibi, ürüne özel bir altyapı kurulması da mümkündür (Gupta, 2018). Yine de en yaygın kullanım sanallaştırılmış kaynakların kullanılmasıdır. Sanallaştırma ise bir fiziksel sunucunun kaynaklarının sanal sunuculara bölünmesi ve her bölümün birbirinden ayrı alanlara sahip olmasına odaklanır. Sanal makineler çeşitli esneklikler sağlayabilirler. Ürünü oluşturan uygulama parçaları için ihtiyaç olabilecek çeşitli işletim sistemi ve kütüphane ihtiyaçlarını karşılayacak altyapılar sanal makineler

üzerinde düzenlenmesi mümkündür. Fiziksel makineler kadar performanslı olmamaları ve her sanal makine için bakım maliyetleri de göz önüne alınırsa dezavantajları da mevcuttur. Google, Microsoft, Amazon gibi teknolojinin önde gelen şirketleri ürün geliştiricileri için altyapılar haricinde çeşitli hizmetler de sağlamaktadırlar. (Bass, 2019)

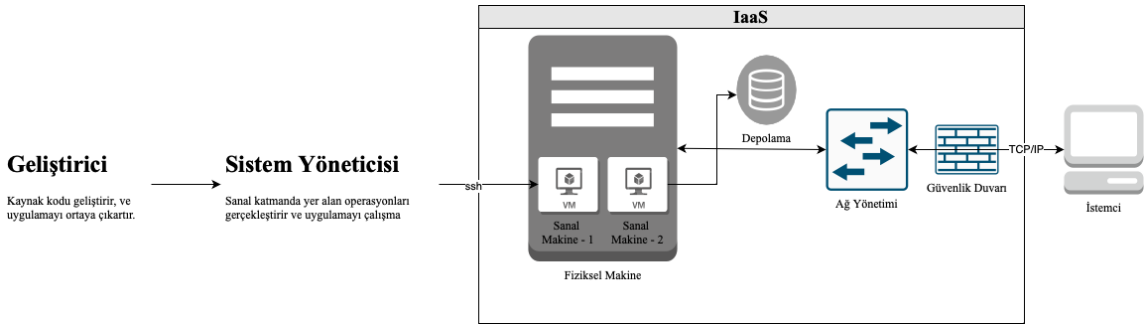
2.2.1 Hizmet Olarak Altyapı

Hizmet Olarak Altyapı (IaaS- Infrastacture-as-a-Service) temel olarak bulut üzerinde depolama, ağ gibi yapılar sunan bir servis hizmetidir (Şekil 7) ve "kullandığın kadar öde" mantığına sahiptir. Böylelikle firmalar, şirket içi fiziksel yapılar kurmak için satın alma yoluna gitmek ve bakımları için kaynak ayırmak yerine sunucular, disk kullanımı, ağ yapıları gibi özellikleri bulut yoluyla sağlamış olurlar. Bir diğer yandan prototip, oluşturma veya acil altyapı ihtiyaçları gibi durumlarda IaaS yapısı tercih edilebilir. Bu hizmet altyapı yönetimi için bir avantaj sağlasa da konfigürasyon ihtiyacının kullanıcının gidermesi gerekmektedir. Özetle IaaS yapısı, bulut üzerinde bakımının yapılması gereken bir sanal makine tahsis eder. Böylelikle fiziksel bir makine satın almaktan kaçınılabilir. IaaS için sağlayıcı depolama, ağ yönetimi, sanal makineler gibi temel altyapılar sağladığı için, müşteri, sağlayıcının sunduğu sanal makinede bir işletim sistemi üzerine uygulamasını yerleştirir. Böylelikle, fiziksel bir altyapı yönetme ihtiyacı ortadan kalksa da uygulama için çalışma ortamının (veri tabanları, izleme mekanizmaları vb.) sorumluluğu kullanıcıya aittir. Sanal makinenin konfigürasyonu ve tahsis süresi ücret konusunda belirleyicidir. (Ahluwalia, v.d., 2022)



Şekil 7. IaaS Yapısı

Geliştirici uygulamasını geliştirir ve oluşan çıktıyı sistem yöneticisine teslim eder. Sistem yöneticisi, IaaS içerisinde yetki sahibi olduğu sanal makine içerisinde uygulamasını yükler ve çalıştırır. İstemci bu uygulamaya internete açık bir ağ üzerinden erişir. Örnek akış Şekil 8'de ele alınmıştır.



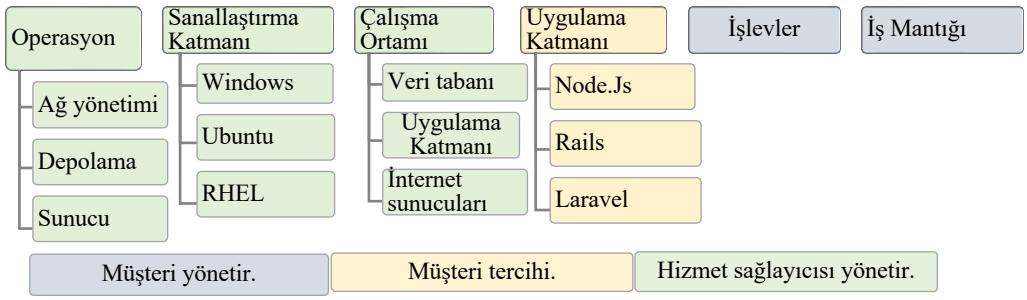
Şekil 8. IaaS Akışı

2.2.2 Hizmet Olarak Platform

Hizmet Olarak Platform (PaaS- Platform-as-a-Service), müşterilere, geliştirilen uygulamayı çalıştırmak ve yönetmek üzere; yazılım, donanım ve altyapılardan oluşan bir

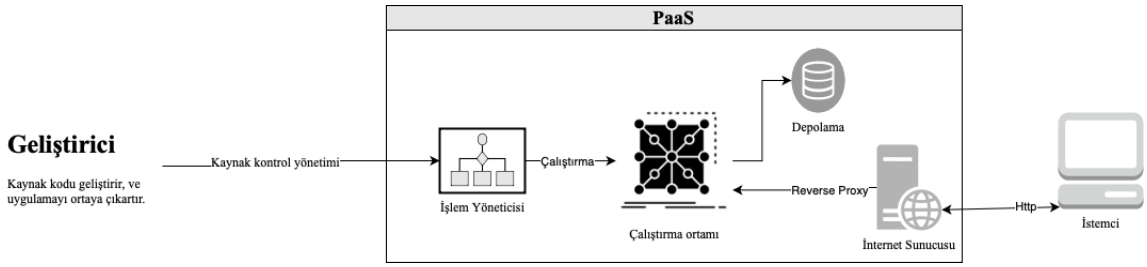
bulut çözüm sağlanmasıdır (Şekil 9). PaaS ağ yönetimi, disk kullanımı, işletim sistem, veri tabanı gibi unsurları içerisinde barındırır. Müşteriler, belirledikleri miktardaki kaynak için sabit ücret ödeyebildikleri gibi "kullandığın kadar öde" fiyatlandırmasını da seçebilirler. Özet olarak, belirli bir teknoloji ile yazılmış bir geliştirmeyi hızlıca devreye almaya olanak tanır. Örneğin bir Node.JS uygulamasını anlar ve "npm" bağımlılıklarını indirerek uygulamanın çalışması için gerekli koşulları sağlar. Geliştiricinin uygulamasına odaklandığı, herhangi bir altyapı yönetimi ihtiyacı duymayan ortamların sağlanması PaaS ile mümkün olabilmektedir. Geliştirici, uygulamanın çalışma ortamını minimum konfigürasyonla oluşturabildiği için, kod yazmaya daha çok odaklanabilir ancak altyapı sağlayıcı tarafından sağlandığı için ortamın kontrolünde daha az söz sahibidir. (Lawton, 2008)

Bazı PaaS örnekleri: Red Hat Openshift, Google App Engine, Heroku, Windows Azure



Şekil 9. PaaS Yapısı

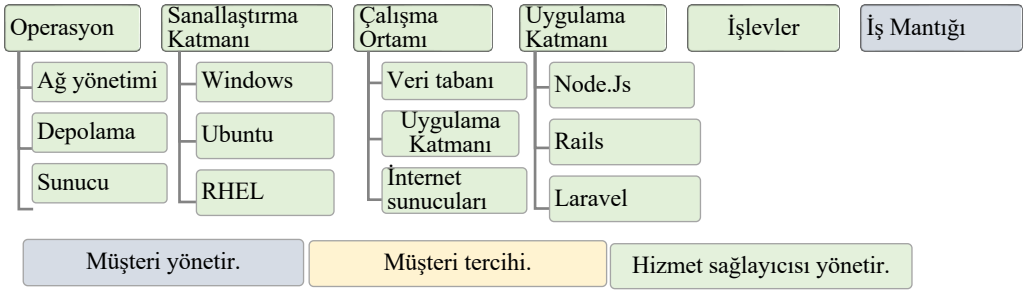
Geliştirici, kaynak kodlarını bir kaynak yönetim kontrolü vasıtası ile PaaS hizmetine gönderir. Sağlayıcı, uygulamanın türünü anlar ve kodu derleyerek ürettiği çıktıyı çalışma ortamına entegre eder ve ihtiyaç duyulan diğer kaynaklarla beraber uygulamayı ayağa kaldırır ve internet erişimine açar (Yasrab ve Gu, 2016). Örnek bir akış Şekil 10'da gösterilmiştir.



Şekil 10. PaaS Akışı

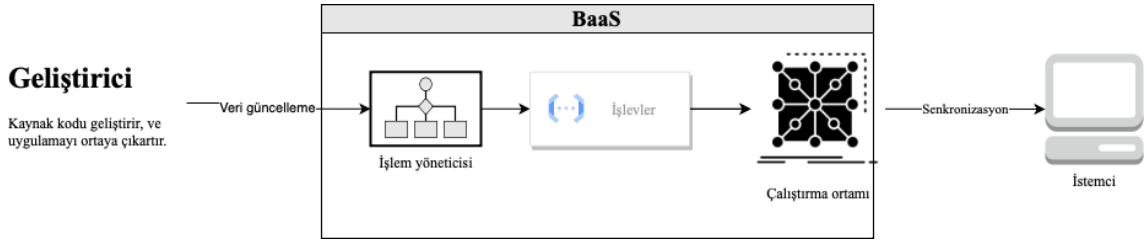
2.2.3 Hizmet Olarak Arka Uç

Hizmet Olarak Arka Uç (BaaS- Backend-as-a-service), geliştiricinin ortaya çıkardığı ürünün ihtiyaç duyduğu diğer araçların sağlandığı bir çözümdür (Şekil 11). Bulut depolama, barındırma, kimlik doğrulaması, veri tabanı yönetimi vb. sunucularda gerçekleşen eylemler için geliştiricilerin kullanımına API hizmetleri sağlar. Böylelikle geliştirici ürünün iş katmanına daha çok yoğunlaşabilir. Bununla beraber, geliştiricinin herhangi bir altyapı yönetmesi de gerekmez. API hizmetleri aracılığı ile kullanıcı yönetimi, bildirim gönderme gibi özellikler sunarak güçlü web veya mobil uygulama geliştirmeye olanak tanır. Müşteri sağlanan API hizmetleri sayesinde arka uçta farklı bir geliştirme ihtiyacı duymadan ön uçtan çeşitli işlevler kullanabilir. Bu durumda da arka uç geliştirme işleri daha çok iş mantığına odaklanmasına olanak tanır. Sağlanan hizmetler daha az esnektir ve geliştiricinin buna ayak uydurması gerekmektedir. Bununla beraber bu işlevlerin kullanılması ortama olan bağımlılığı da arttıracaktır ve farklı bir hizmet sağlayıcıya taşınması çok zor olacaktır (Navaneeth, v.d., 2022). Ayrıca (Prasanthan, v.d., 2023)'de görüldüğü üzere uygulama oluşturmayı daha basit hale getiren low-code platformları da BaaS örnekleri arasında yer alabilir. Böylelikle POC ortaya çıkarma, geliştirme süreçlerin karmaşıklığını azaltma ve daha az kod bilgisiyle ürün ortaya çıkarma gibi ihtiyaçlarda da bir seçenek olarak değerlendirilmesi mümkün olabilir.



Şekil 11. BaaS Yapısı

Geliştirici, hizmet sağlayıcısının sunduğu API hizmetleri aracılığı ile ön uçtan işlevler çağırabilir. Birçok durumda kod yazmak yerine basit konfigürasyonlar ile bu süreç tamamlanabilir. Örneğin, kullanıcı girişi doğrulamak için veya müşterilere gerçek zamanlı bildirimler gönderebilmek için tercih edilebilir. BaaS akışı Şekil 12'de gösterilmiştir.

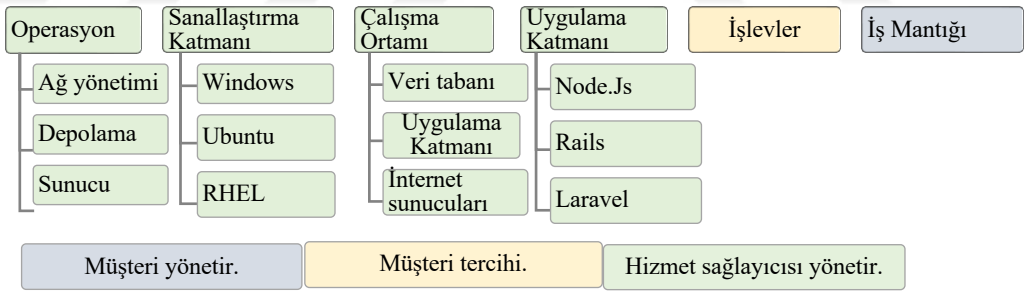


Şekil 12. Baas Akışı

2.2.4 Hizmet Olarak İşlev

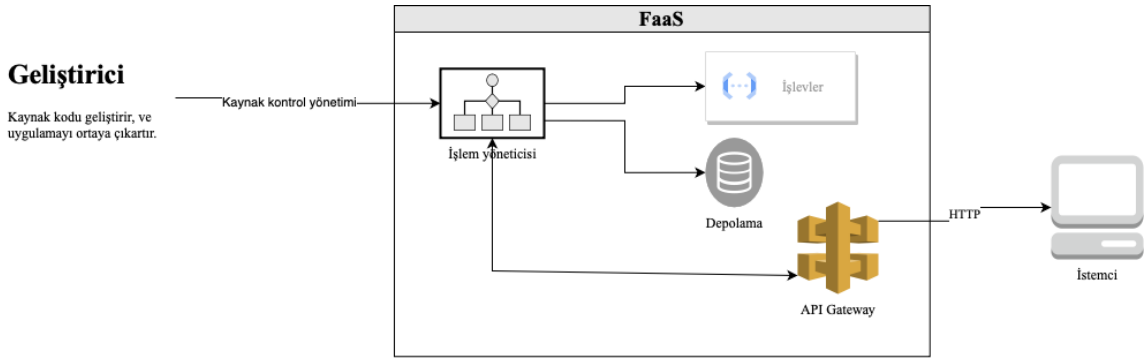
Hizmet Olarak İşlev (FaaS- başka deyişle Function-as-a-Service), geliştirilen ürünün olaya dayalı tetikleyicilerle isteklere yanıt verdiği sunucusuz yapılara ait bir çözümdür (Şekil 13). Yani, eğer bir istek yoksa; ilgili kaynaklar kapatılır ve diğer istekler için kullanılabilir hale getirilir. İstek gelince kaynaklar devreye alınır ve belirlenen eşik değerine istinaden yukarı veya aşağı doğru ölçeklenerek ilerler. Kaynaklar boşta

kaldığında ise herhangi bir maliyet oluşmaz. Basit bir mikroservis çağrısı FaaS'ı kullanmak için en ideal yoldur. Fazla kütüphane kullanımı veya bir fonksiyondan farklı çağrımlar yapılması yanıt süresini yavaşlatabileceği gibi maliyeti de arttıracaktır. Bu platformlar, uygulamaları depolar ve olaylara yanıt olarak çalıştırır. Müşteriler, veri tabanı gibi uygulamanın ihtiyaç duyduğu kaynaklara erişen basit kod parçacıkları yazar ve hangi olaylar karşısında çalıştırılacağını tayin eder. Bu model genellikle sunucusuz yapılar ile karşılaştırılır. Gerçekte olan ise FaaS hizmetinin sunucusuz yapıların bir parçası olduğudur. Bu hizmet hem altyapı hem de uygulama konfigürasyonun karmaşıklığını ortadan kaldırır ve bu hizmet sayesinde müşterinin sunulan kaynakları yönetmesine ihtiyaç kalmamaktadır. Müşterinin uygulamanın çalıştığı alana veya uygulama katmanlarından herhangi birine genellikle erişimi olmaz veya kısıtlıdır. Diğer servislerin aksine olay oluşmazsa hiçbir şey çalışmaz ve fiyatlandırma da buna dayanır. (Koschel, v.d., 2021)



Şekil 13. FaaS Yapısı

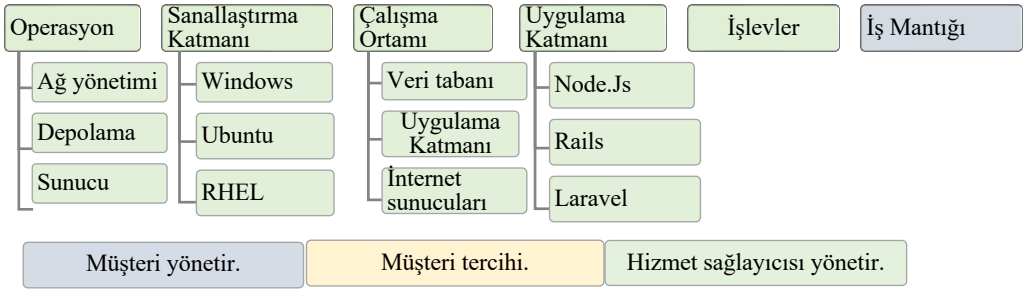
Geliştirici küçük kod parçacıkları yazarlar. Bunlar genellikle fonksiyon olarak isimlendirilir ve yapıları basittir; fazla kütüphane ihtiyacı duymazlar ve birbirlerini çağırmaları gerekmez, tek bir amaca hizmet ederler. Hizmet sağlayıcı bu hizmetlerin entegrasyonu, depolama ihtiyaçları ve altyapısal olarak izleme olanağı için farklı hizmetler sağlar. Örnek gösterim Şekil 14'te ifade edilmiştir.



Şekil 14. Faas Akışı

2.2.5 Hizmet Olarak Yazılım

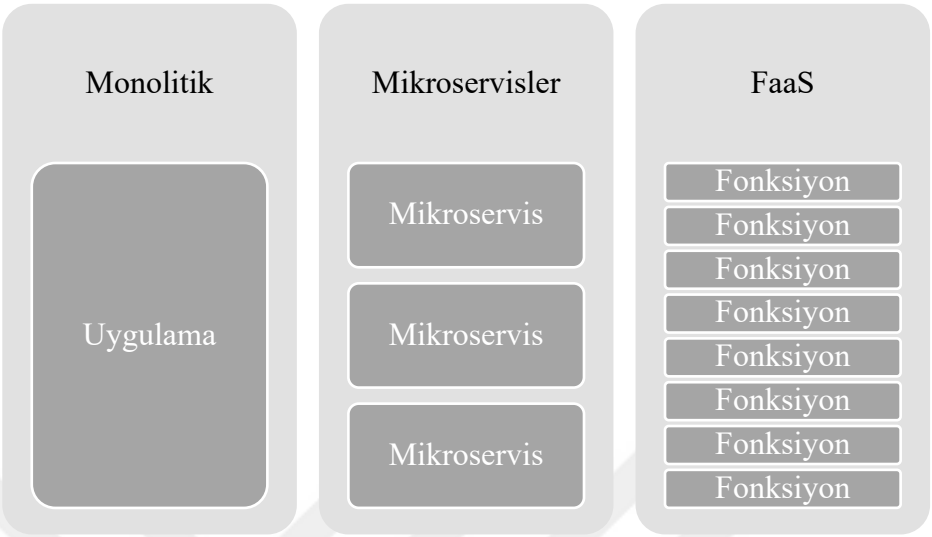
Hizmet Olarak Yazılım (SaaS- Software-as-a-Service), web tabanlı olarak; sunucuları, veri tabanları ve uygulamayı oluşturan kodun yönetilmesi çözümüdür (Şekil 15). Maliyet abonelik sistemine göre belirlenir. Böylelikle, şirket içerisinde bulunmaksızın çevrimiçi platformlarda geliştirme yapılır ve teslim edilir. (Aleem, v.d., 2021) Servis sağlayıcısı, donanım ve performans yönetimini sağladığı gibi uygulamanın güvenliğinden de sorumludur. Küresel çapta en bilinen örnekleri Office 365 ve Adobe Acrobat olarak verilebilir. E-posta uygulamaları ele alınırsa; alıcı, konu ve içerik ile bu hizmet rahatlıkla kullanılabilir. Eğer bu bir SaaS çözümü olmasaydı; iletişim sağlanacak sunucu ve istemcilere SMTP kurulması ve ölçeklendirilmesi ihtiyacı doğacaktı. (Johnston, 2014) SaaS, internet üzerinden ihtiyaca uygun bir yazılım imkânı sunmak için idealdir. Bu hizmette tüketilen yazılım sağlayıcı tarafından geliştirilir, dağıtılır ve çalıştırılır, böylelikle tüketiciye uygulamayı kullanmaktan başka bir iş kalmaz. Fiyatlandırma ise genellikle kullanıcı başı ve aylık periyotlardadır (Sun, v.d., 2011).



Şekil 15. SaaS Yapısı

2.3 Mimarilerin Karşılaştırılması

Uygulama mimariler ürün geliştirme süreçlerinde çeşitli sorunlara çözüm olabilmek üzere yaygın kullanım gören ve yapıları bu desenlere göre inşa edilmesine olanak tanıyan çözümlerdir. Başlıca mimariler; monolitik, mikroservis ve sunucusuz yapılardan oluşur. Genel görünümleri Şekil 16'da gösterilmiştir. Bu yapılar geliştirme süreçlerini başkalaştırsalar da aslında altyapı odaklı ayrımlar olduğundan söz etmek mümkündür. Geliştirme perspektifinden farklılaşmalar da uygulama depolarının ayrışması ile başlar. Buradaki temel unsur ürünün derleme sonucu oluşan çıktılarının nasıl konumlandırıldığıdır. Monolitik uygulamalarda tek bir depoda tutulan kodun derlenmesi sonucu oluşan ve bir bütün halinde sunuculara iletilmesi gereken yapılar mevcutken, mikroservis mimarilerinde ise farklı kod depolarından oluşan ve birbirileri arasında bağımlılık bulunmayan çıktılar oluşur ve sunuculara iletilir. Buradaki temel farklardan biri de mikroservisler tercih sebepleri gereği konteyner yapılarda yer alırlar. Böylelikle özellikle geliştirme sonrasında oluşan üretim hattı için DevOps araçlarından rahatlıkla faydalanabilirler. Konteynerler ise orkestrasyon araçları ile yönetilebilirler.

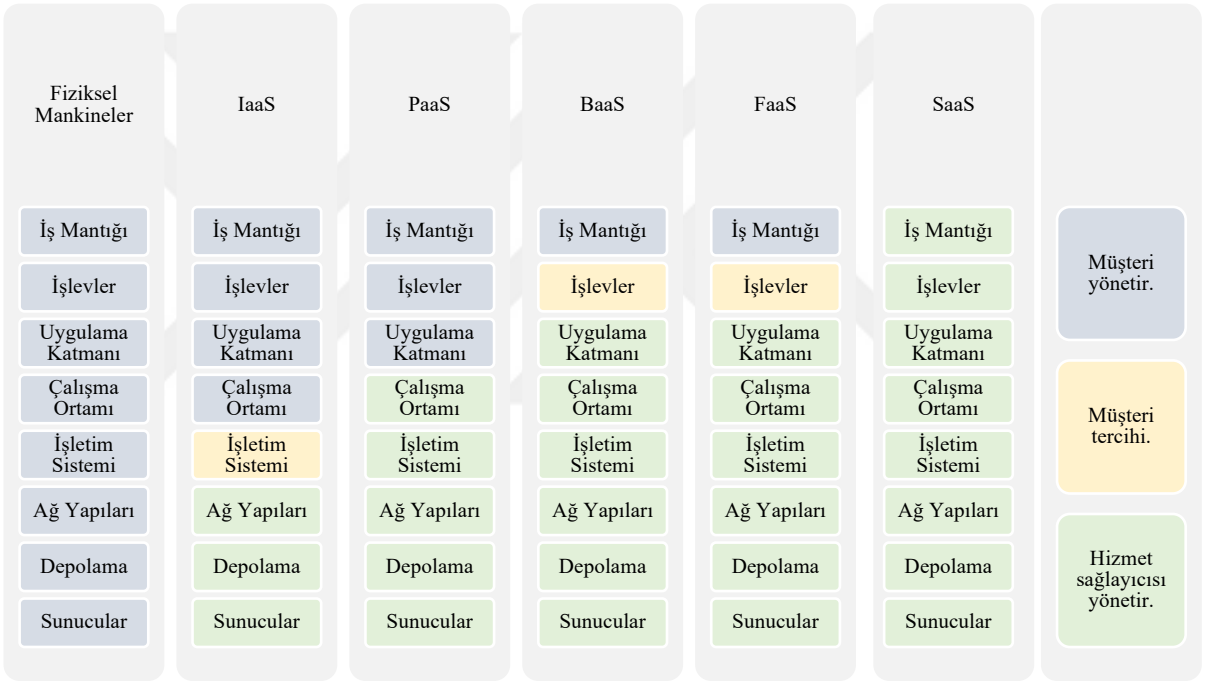


Şekil 16. Mimarilerin Genel Görünümü

Operasyon odaklı incelendiğinde ise uygulamanın yer aldığı altyapının ne olduğu operasyon ihtiyacını belirleyen temel faktördür. Monolitik mimariler genellikle fiziksel veya sanal makinelerde konumlanırlar. Bu durumda da fiziksel/sanal makinelerin bakımları, güvenlik güncellemeleri gibi ihtiyaçları ortaya çıkartır. Genel olarak bu yapılarda geliştirme ekipleri ve altyapı ekiplerinin de ayrı olduğu gözlemlenir. İnsan kaynağının bir kısmını altyapı bakımlarına ayırmak gerekir. Mikroservis mimarilerinde ise sanal makineler içinde konteyner teknolojiler kullanılabilir. Konteyner yapıları birbirilerinden izole bir şekilde konumlanabilir. Servisin ihtiyaç duyduğu işletim sistemi odaklı ihtiyaçlarından başlayarak farklı bağımlılıkları kendi içlerinde -sanal makineden bağımsız bir şekilde- yönetmek mümkündür. Konteyner içi ihtiyaçlar genellikle geliştirme ekipleri tarafından sağlanır ve operasyonel maliyet oluşturmaz ancak yine de bir sanal makine içinde yer alıyorsa konteynerler, ilgili makinenin bakım ihtiyaçları için operasyonel maliyet hala mevcuttur. Sunucusuz yapılarda ise kullanılan servise bağlı olarak altyapı operasyonlarını yönetmek gerekebilse de sağlayıcının kendi içinde sunduğu başka servislerle bu ihtiyacı en aza kadar indirmek mümkündür. Böyle bir durumda ise ürün için oluşan insan kaynağı yoğun bir şekilde geliştirmeye odaklanabilir ve sağlayıcının sunduğu hizmetleri anlaşma hüküm ve koşulları ile tüketebilir. Şekil 17'de

operasyon ihtiyaçlarının doğrusu Şekil 18'de ise altyapı ihtiyaçlarının kim tarafından ne kadar karşılandığının gösterimi ele alınmıştır.

Şekil 17. Operasyona Duyulan İhtiyaç



Şekil 18. Altyapı İhtiyaçlarının Detaylandırılması

Dağıtım ise yazılım yaşam döngüsünün bir parçasıdır. Uygulama geliştirilir ve derleme sonucu oluşan çıktılar canlı ortamlara alınır. Bu operasyonu sağlayabilecek çok sayıda yöntem bulunsa da anlamsal olarak ikiye ayrıldığı söylenebilir; manuel operasyonlar ve otomasyon aracılığı ile yapılan operasyonlar. Manuel operasyonlar; tercih edilen mimarinin desteklediği sürece geliştirme süreçlerinin sonucunda oluşan çıktıyı canlı ortama bir operasyon yöneticisi tarafından alınması sürecidir. Bu durumda geçişler genellikle yoğun olmayan saatlerde yapılır ve kesinti süresi fazladır. Ayrıca canlıdan önceki son test ortamında yer alan konfigürasyon dosyalarının, canlı ortamda, tekrar elden

geçirilmesi gerekebilir. Otomasyon çözümleri ile gerçekleşen operasyonların temelinde DevOps kavramı yer almaktadır. DevOps, bu süreçleri otomatik bir hale getirmeye odaklanırken aynı zamanda geliştirme süreçlerinin sürekli olmasını da sağlar. Böylelikle, test ortamları ve canlı ortam arasında bir entegrasyon sağlanmış olur. Mikroservis dağıtımlarında ise ilgili parçaların bağımsızlığını sağlayabilmek adına farklı ihtiyaçları olabilir.

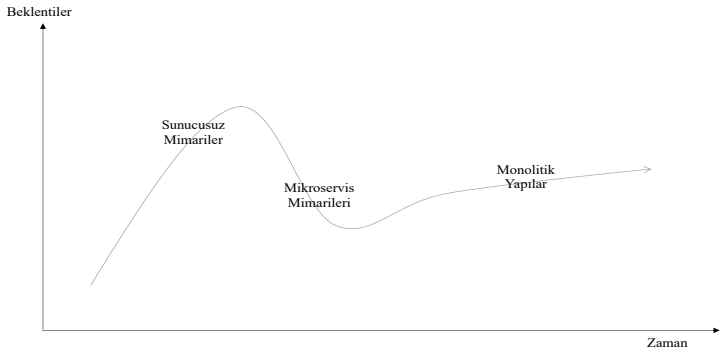
2.3.1 Performans Açısından Karşılaştırma

Canlı ortama alınan ürünlerin performans metrikleri, birden fazla parametreye göre şekillenebilir. Dayanıklılık ve performans açısından en kritik metriklerden biri "yanıt süresi"dir (Villamizar, v.d., 2017). Örneğin mikroservis mimarisinde tek bir noktadan istek girişi sağansa bile çeşitli servislerin senkron veya asenkron çağrımlar işlevin haricinde kalan network sürelerinin artışına sebep olacak ve yanıt süreleri uzayacaktır. Diğer yandan monolitik yapılar işlevlerin iç içe kurgulanması sebebiyle network süreleri eklenmeksizin daha hızlı yanıtlar verecektir. Son olarak ise sunucusuz yapılarda oluşabilecek iki senaryo mevcuttur; ilk gelen istek "soğuk başlangıç" denilen uygulamanın ayağa kalkmasını da içeren yavaş yanıt süresi ile karşı karşıya kalsa da hali hazırda yük alan bir işlev için bahsi geçen mimariler arasında en hızlı yanıtı sunacaktır. (Lenarduzzi, v.d., 2020) Bir diğer yandan teknik borçlar bir uygulamanın ömrünü belirleyen anlamlı metrikler arasındadır. Bu perspektif ile yapılabilecek en sağlıklı gözlemlerden biri aynı ürünün monolitik yapısı ile mikroservislere bölünmüş halinin karşılaştırılması olacaktır. Yapılan araştırmalar teknik borçların artma hızının monolitik yapılarda mikroservislere göre daha fazla olduğu gözlemlenmiştir. Bu da ürün ömrünün teknik açısından değerlendirilmesi için önemli bir parametre ortaya koymaktadır (Ouyang, v.d., 2023).

2.3.2 Maliyet Açısından Karşılaştırma

Ürün yaşamını etkileyen faktörlerden biri de maliyettir. Maliyet özünde parasal bir anlam taşıyor olsa da zaman, efor ve bakım gibi çeşitli alt başlıklar ile de ifade edilebilir. Monolitik yapılar için ürünün bakımının yapılması etkilenebilecek birçok olası işlev ve altyapı bileşeni de göz önüne alındığında içinden çıkılmaz bir kördüğüm oluşturabilir (Velepucha ve Flores, 2023). Bununla beraber tek parça yapılar için ölçeklenebilirlik de altyapısal açıdan maliyet yaratacaktır. Mikroservis dünyasında göz ardı edilemeyecek maliyetlerin başında aşırı ağ kullanımı olacaktır. Çünkü servisler birbirleriyle haberleşirken bir ağ içerisinde hareket etmeleri gerekecektir. Ayrıca, tasarıma göre değişse de her servisin ayrı bir veri tabanı olması önerildiğinden veri tekrarları ve depolaması da çeşitli maliyetlere yol açacaktır. Sunucusuz mimarilerde ise öncelikli maliyet ücret olarak görülmekte. Kullanılacak altyapının ve gelen istek sayısının da sağlayıcılar tarafından farklı fiyatlaması enflasyonist ortamlarda oluşacak ufak fiyat artışlarının bile yük alan ürünlerde maliyetin ön görülemez şekilde artmasına yol açabilir. Yoğun trafik gelen servislerin dışında; resim, ses veya video gibi disklerde kaplayacağı yerleri büyük olan yükleme ve indirme işlemlerinde de disk kapasiteleri alternatif çözümlere göre maliyetli olabilmektedir (Menéndez, v.d., 2023).

Şekil 19'da mimarilerin "Gartner Hype Döngüsü" üzerindeki yerleri gösterilmiştir. Sunucusuz mimariler teknoloji tetiği aşamasındayken, monolitik yapılar artık verimlilik platosuna girmiştir. Mikroservis mimariler ise aydınlanma eğiminde bulunmaktadır.



Şekil 19. Mimarilerin Gartner Hype Döngüsündeki Yeri

ÜÇÜNCÜ BÖLÜM
METODOLOJİ

3.1 Kriterler ve Ölçüm Yöntemi (AHP)

Karmaşık ve kesin sayılarla değerlendirilebilmesi mümkün olmayan kriterlerin ve alternatifleri ölçebilme kabiliyeti sebebiyle birçok karar verme çözümünden ayrışan bir tekniktir. Kriterlerin alternatifler karşısında değerlendirildiği gibi kendi aralarında da değerlendirilmelerine olanak tanır. Verilecek karardan daha öncesinde yapılan anketin de katkısıyla önyargıdan arındırılmış sonuçlar verir. AHP ile beraber Basit Toplamlı Ağırlıklandırma (SAW – Simple Additive Weighting) ve İdeal Çözüme Benzerliğe Göre Tercih Sıralaması Tekniği (TOPSIS - Technique for Order of Preference by Similarity to Ideal Solution) gibi karar alma yöntemleri mevcut olsa da AHP'nin daha objektif ağırlıklandırma sonucu verebilecek (Suartini, v.d, 2023) olması sebebiyle ölçme yöntemi olarak tercih edildi.

AHP Kendi içerisinde tek sayılardan oluşan bir standart tercih tablosuna sahiptir. Sayılar önem veya üstünlük derecesi belirtirken bu sayılara karşılık gelen tanımlar Tablo 1'de ifade edilmiştir. Anket sonucu oluşan çift sayılı değerler ise uzlaşma değerleri olarak kabul edilir.

Tablo 1. Standart Tercih Tablosu

1	Eşit önemli
3	Biraz Önemli
5	Önemli
7	Çok Önemli
9	Belirgin Önemli
2, 4, 6, 8	Uzlaşma Değerleri

Bir örnek üzerinden AHP süreçleri ele alınacak olursa; seçenekler tablosunun oluşturulması karşılaştırılabilir parametrelerin görünmesi için faydalı olabilir. (Tablo 2)

Tablo 2. Örnek Seçenekler Tablosu

Özellikler	Bilgisayar - A	Bilgisayar - B	Bilgisayar - C
Fiyat	2000 TL	1000 TL	1250 TL
Garanti Süresi	2 Yıl	1 Yıl	1.5 Yıl
Şarj Kapasitesi	15 saate kadar	5 saate kadar	10 saate kadar
Depolama	2 TB	500 GB	2 TB

Bir sonraki adım olarak kriterlerin karşılaştırma ve birbirlerine üstünlüklerini gösteren matrisler oluşturulur (Tablo 3) ardından hesaplamalara başlayabilmek için sütun toplamları bulunur (Tablo 4) ve buradaki değerler ile satır ortalamaları bulunur (Tablo 5).

Tablo 3. Örnek Hesaplama - Kriter için Karşılaştırma Matrisi

	Bilgisayar - A	Bilgisayar - B	Bilgisayar - C
Fiyat			
Bilgisayar - A	1	1/7	1/5
Bilgisayar - B	7	1	1
Bilgisayar - C	5	1	1
Garanti Süresi			
Bilgisayar - A	1	7	3
Bilgisayar - B	1/7	1	1/3
Bilgisayar - C	1/3	3	1
Şarj Kapasitesi			
Bilgisayar - A	1	7	3
Bilgisayar - B	1/7	1	1/3
Bilgisayar - C	1/3	3	1
Depolama			
Bilgisayar - A	1	5	1
Bilgisayar - B	1/5	1	1/5
Bilgisayar - C	1	5	1

Tablo 4. Örnek Hesaplama - Sütun Toplamları

Fiyat			
	Bilgisayar - A	Bilgisayar - B	Bilgisayar - C
Bilgisayar - A	1	1/7	1/5
	+	+	+
Bilgisayar - B	7	1	1
	+	+	+
Bilgisayar - C	5	1	1
Toplam	13	15/7	11/5
Garanti Süresi			
	Bilgisayar - A	Bilgisayar - B	Bilgisayar - C
Bilgisayar - A	1	7	3
	+	+	+
Bilgisayar - B	1/7	1	1/3
	+	+	+
Bilgisayar - C	1/3	3	1
Toplam	31/21	11	13/3
Şarj Kapasitesi			
	Bilgisayar - A	Bilgisayar - B	Bilgisayar - C
Bilgisayar - A	1	7	3
	+	+	+
Bilgisayar - B	1/7	1	1/3
	+	+	+
Bilgisayar - C	1/3	3	1
Toplam	31/21	11	13/3
Depolama			
	Bilgisayar - A	Bilgisayar - B	Bilgisayar - C
Bilgisayar - A	1	5	1
	+	+	+
Bilgisayar - B	1/5	1	1/5
	+	+	+
Bilgisayar - C	1	5	1
Toplam	11/5	11	11/5

Tablo 5. Örnek Hesaplama - Satır Ortalamaları Hesaplama Tablosu

Fiyat				Satır Ortalaması
	Bilgisayar - A	Bilgisayar - B	Bilgisayar - C	
Bilgisayar - A	1/13 = 0.08	1/7 / 15/7 = 0.06	1/5 / 11/5 = 0.10	(0.08 + 0.06 + 0.10) / 3 = 0,08
Bilgisayar - B	7 / 13 = 0.54	1 / 15/7 = 0.47	1 / 11/5 = 0.45	(0.54 + 0.47 + 0.45) / 3 = 0,49
Bilgisayar - C	5 / 13 = 0.38	1 / 15/7 = 0.47	1 / 11/5 = 0.45	(0.38 + 0.47 + 0.45) / 3 = 0,43
Toplam	1	1	1	1.00
Garanti Süresi				**Satır Ortalaması**
	Bilgisayar - A	Bilgisayar - B	Bilgisayar - C	
Bilgisayar - A	1 / 31/21 = 0,68	7 / 11 = 0,64	3 / 13/3 = 0,69	(0,68 + 0,64 + 0,69) / 3 = 0,67
Bilgisayar - B	1/7 / 31/21 = 0,10	1 / 11 = 0,09	1/3 / 13/3 = 0,08	(0,10 + 0,09 + 0,08) / 3 = 0,09
Bilgisayar - C	1/3 / 31/21 = 0,22	3 / 11 = 0,27	1 / 13/3 = 0,23	(0,22 + 0,27 + 0,23) / 3 = 0,24
Toplam	1	1	1	1.00
Şarj Kapasitesi				**Satır Ortalaması**
	Bilgisayar - A	Bilgisayar - B	Bilgisayar - C	
Bilgisayar - A	1 / 31/21 = 0,68	7 / 11 = 0,64	3 / 13/3 = 0,69	(0,68 + 0,64 + 0,69) / 3 = 0,67
Bilgisayar - B	1/7 / 31/21 = 0,10	1 / 11 = 0,09	1/3 / 13/3 = 0,08	(0,10 + 0,09 + 0,08) / 3 = 0,09
Bilgisayar - C	1/3 / 31/21 = 0,22	3 / 11 = 0,27	1 / 13/3 = 0,23	(0,22 + 0,27 + 0,23) / 3 = 0,24
Toplam	1	1	1	1.00
Depolama				**Satır Ortalaması**
	Bilgisayar - A	Bilgisayar - B	Bilgisayar - C	
Bilgisayar - A	1 / 11/5 = 0,45	5 / 11 = 0,45	1 / 11/5 = 0,45	(0,45 + 0,45 + 0,45) / 3 = 0,45
Bilgisayar - B	1/5 / 11/5 = 0,10	1 / 11 = 0,10	1/5 / 11/5 = 0,10	(0,10 + 0,10 + 0,10) / 3 = 0,10
Bilgisayar - C	1 / 11/5 = 0,45	5 / 11 = 0,45	1 / 11/5 = 0,45	(0,45 + 0,45 + 0,45) / 3 = 0,45
Toplam	1	1	1	1.00

Tablo 3-5 arası yapılan hesaplamaların sonuç tablosu Tablo 6'da verilmiştir.

Tablo 6. Örnek Hesaplama - Satır Ortalamaları Sonuç Tablosu

	Fiyat	**Garanti Süresi**	**Şarj Kapasitesi**	Depolama
Bilgisayar - A	0,08	0,67	0,67	0,45
Bilgisayar - B	0,49	0,09	0,09	0,10
Bilgisayar - C	0,43	0,24	0,24	0,45

Benzer aşamalar kriterlerin bir biri arasında önemini belirlemek için de Tablo 7'de sutün toplamları bulunur.

Tablo 7. Örnek Hesaplama - Kriterlerin Önem Matrisi

	Fiyat	**Garanti Süresi**	**Şarj Kapasitesi**	Depolama
Fiyat	1	1/3	1/9	1/7
Garanti Süresi	3	1	1/7	1/5
Şarj Kapasitesi	9	7	1	3
Depolama	7	5	1/3	1
Toplam	**20,00**	**13,33**	**1,59**	**4,34**

Tablo 8'de Tablo 7'de bulunan toplam değerlere göre satır ortalamaları bulunur.

Tablo 8. Örnek Hesaplama - Kriterlerin Önem Matrisi Satır Ortalamaları

	Fiyat	**Garanti Süresi**	**Şarj Kapasitesi**	Depolama	**Satır Ortalaması**
Fiyat	1/20	(1/3)/13,33	(1/9)/1,59	(1/7)/4,34	**0,04**
Garanti Süresi	3/20	1/13,33	(1/7) /1,59	(1/5)/4,34	**0,09**
Şarj Kapasitesi	9/20	7/13,33	1/1,59	3/4,34	**0,57**
Depolama	7/20	5/13,33	(1/3) /1,59	1/4,34	**0,29**
Toplam	**1**	**1**	**1**	**1**	**1,00**

Tablo 9'da kriterlerin kendi arasındaki önemini belirlemek için satır ortalamaları değerleri ayrı bir tabloya yazılır.

Tablo 9. Örnek Hesaplama - Kriterlerin Ağırlık Matrisi

Kriter	Ağırlık
Fiyat	**0,04**
Garanti Süresi	**0,09**
Şarj Kapasitesi	**0,57**
Depolama	**0,29**
Toplam	**1**

Tablo 10'da ise Tablo 6'da bulunan matris ile kriterlerin önem matrisi (Tablo 9) kendi aralarında çarpılır ve sonuç olarak hangi alternatifin tercih edilebileceği tablo üzerinden öneri olarak karar alıcıya sunulur. Yukarıda yapılan örnek anket sonucuna göre Bilgisayar – A diğer alternatiflere göre seçilebilir alternatif olarak öne çıkmıştır. Farklı anketlerle farklı sonuçlar da bulunabilir, okuyucular kendi örnekleri için bu hesaplamaları uygulayabilirler.

Tablo 10 Örnek Hesaplama - Sonuç Tablosu

Kriter	Ağırlık	Oran
Bilgisayar - A	**0,58**	**%58**
Bilgisayar - B	0,11	%11
Bilgisayar - C	0,31	%31
Toplam	**1**	**%100**

Örnek hesaplamada en yüksek oranı alan Bilgisayar-A alternatifi diğer alternatifler karşısında makul bir tercih olarak seçilebileceği görülmektedir. Matrislerin hesaplamaları yapıldıktan sonra λmax değeri ardından Tutarlılık İndeksi $TI = (\lambda max - n)/(n-1)$ Tutarlılık Oranı ise $TO = TI / RTI$ şeklinde hesaplanır (Peker ve Toksarı, 2007). Adımların detayı için ise (Doğan ve Gencan, 2015) kaynağından faydanılabilir.

Karar problemi olarak, MVP süreçlerinin çok vakit kaybetmeden pazara açılması ve alınacak bildirimler doğrultusunda ürünün tekrar pazara çıkışının sağlanması gerektiğinden, genel kullanımı olan mimariler arasında en ideal sonuca ulaşmak için "MVP için en iyi mimarinin seçilmesi." belirlenmiştir. Karar değişkenleri [Çeviklik,

karmaşıklık, veritabanı işlemleri, entegrasyon testi, bakım maliyeti, altyapı maliyeti, debugging, ölçekleme, geliştirme karmaşıklığı, bakım (Shahand, v.d., 2015), geliştirme hızı, birim test, dayanıklılık, geliştirme maliyeti (Khanam, v.d., 2017), gecikme (yanıt süresi), altyapı kısıtları, çalıştırma maliyeti (Christidis, v,d., 2020) Tablo 11'de alternatifiler ise Tablo 12'te belirtilmiştir.

Tablo 11. AHP için Karar Değişkenleri

Karar değişkenleri	Açıklamalar
Geliştirme hızı	Ürünün oluşması için harcanan geliştirme süresi.
Çeviklik	Değişim taleplerine adaptasyon.
Karmaşıklık	Geliştirme süreçlerinde, ekip içerisinde, altyapı ihtiyaçlarına göre yaşanabilecek genel komplesite durumudur.
Veritabanı İşlemleri	Veritabanına bağlanma, sorgulama ekleme ve çıkarma gibi işlemlerin tümü.
Birim Test	Fonksiyonların ve koşulların test edildiği bir beyaz kutu test türüdür.
Entegrasyon Testi	Bileşenlerin birbiri arasındaki etkileşiminin test edildiği bir beyaz kutu test türüdür.
Altyapı Maliyeti	Uygulamanın çalışması için ihtiyaç duyduğu altyapı maliyetlerinin tümü.
Altyapı Kısıtları	Uygulamanın ihtiyaç duyduğu veya altyapı sistemlerinin sunabildiği özelliklerin sınırı.
Hata Ayıklama	Yazılımın sağlıklı çalışması için hata ayıklama sürecidir.
Ölçekleme	Ürüne gelen yük karşısında oluşabilecek altyapı ihtiyaçlarının genişlemesini ifade eder.
Dayanıklılık	Yük ve felaket durumlarında ürünün sağlıklı ayakta kalabilmesi için gereken tüm kavramları ifade eder.
Gecikme (Yanıt Süresi)	Senkron isteklerdeki beklemeler, ağ çağrımları, güvenlik duvarlarında gibi bölümlerde yaşanabilecek ve olması gerekenin üstünde süre geçmesi durumu.
Geliştirme Karmaşıklığı	Geliştirme süreçlerinde veya iyileştirme çalışmalarında yaşanan veya yaşanabilecek komplekste.
Bakım	Ürünün performans iyileştirilmesi, altyapı güncellemelerinin gerçekleştirilmesi gibi durumları ifade eder.
Geliştirme Maliyeti	İhtiyaç duyulan yazılımcı, altyapı ihtiyaçları gibi süre ve sözleşme kısıtlarının da dahil edildiği maliyeti ifade eder.
Çalıştırma Maliyeti	Ürünün ayakta kalması ve müşteriye ulaşması için ihtiyaç duyulan maliyetleri ifade eder.

Tablo 12. AHP için Alternatifler

Alternatifler	Açıklamalar
Monolitik	Yekpare tasarlanan uygulamaları ifade eder.
Mikroservis	Anlamsal parçalara ayrılmış servislerin birbirleriyle bir mantık çerçevesinde haberleştiği mimariler olarak açıklanır.
Sunucusuz (Serverless)	Bazı işlevlerin ve/veya altyapı yönetimlerinin bulut sistemler üzerinde yer alan yapılar ile yönetilmesi. (IaaS, PaaS, BaaS, FaaS, SaaS)

Sektör paydaşlarına kriterlerin, alternatifler karşısında karşılaştırmalı olarak ve kriterleri de kendi içlerinde üstünlük olarak değerlendirmeleri istendiği bir anket örneği sunulmuştur.

3.2 Kullanılan Teknolojiler

Fikrin geliştirilmesi için monolitik, mikroservis ve sunucusuz mimariler incelenmiştir. AHP yöntemini uygulamak üzere yapılan anket .xlsx uzantılı dosyalar ile iletilmiş, gelen yanıtlar Excel üzerinden hesaplanarak sonuç elde edilmiştir.

3.3 MVP Örnek Uygulaması

Bir fikrin ürüne dönüşüp daha az efor ile müşterilerden daha hızlı geri bildirim toplanması için sunulan yöntemlerden bir tanesidir. Bir problemin çözümüne ulaşmak için oluşturulan senaryo ve bu senaryonun uygulanıp ilk müşterilerden geri bildirim toplanmasıyla beraber fikrin değerlendirilip olgunlaşması ya da fikrin geçersiz olduğuna kanaat getirip ürünü piyasadan çekilmesi gibi kararların alınabildiği bir süreçtir. Bu tezin konusu olarak bu süreçte seçilebilecek teknolojiler değerlendirilmiştir. Fikri test etmek için kurgulanan bu yaklaşım için iyileştirmeleri hızla piyasaya sürmek önemli olacağı gibi, sürecin sonunda ürünün oluşmaması ile de bitebileceği düşünülürse eforu doğru bir şekilde harcamak alınacak verimi arttıracaktır.

DÖRDÜNCÜ BÖLÜM
ARAŞTIRMA BULGULARI VE DEĞERLENDİRMELER

Tablo 11'de açıklamaları ile ifade edilen kriterler ile Tablo 12'de yer alan alternatifler en az 7 yıl tecrübesi olan ve bilişim sektöründe çalışan 8 temsilcinin katılımıyla yüz yüze yapılan anket ile değerlendirilmiştir. Değerlendirme için kriterlere doğru ağırlıklar tayin edebilmek adına "MVP süreçleri için kriterleri karşılaştırmalı olarak puanlayabilir misiniz?" sorusu sorulmuş ve alınan yanıtların geometrik ortalaması alınmıştır.

Tablo 13 Kriterlerin Önem Tablosu

Kriterler	Önem Sonuçları
Geliştirme hızı	0,09
Çeviklik	0,08
Karmaşıklık	0,04
Veritabanı İşlemleri	0,04
Birim Test	0,05
Entegrasyon Testi	0,04
Altyapı Maliyeti	0,06
Altyapı Kısıtları	0,06
Debugging	0,07
Ölçekleme	0,07
Dayanıklılık	0,07
Gecikme (yanıt süresi)	0,06
Geliştirme krmaşıklığı	0,06
Bakım	0,06
Geliştirme maliyeti	0,07
Çalıştırma maliyeti	0,08

Anket sonucunda kriterlerin alternatifler ile ele alındığında oluşan sonuçlar Tablo 13'de ifade edilmiştir. Karmaşıklık, gecikme (yanıt süresi), bakım ve geliştirme maliyeti kriterleri diğer kriterlerden farklı olarak negatif anlam taşır. Bu kriterler için yüksek puan pozitif sonuç olduğunu ifade eder.

Tablo 14. Kriterlerin Alternatifler ile Değerlendirilme Sonuçları

	Monolitik	**Mikroservis**	**Sunucusuz**
Geliştirme Hızı	0,26	0,25	**0,49**
Çeviklik	0,08	0,33	**0,59**
Karmaşıklık	**0,36**	0,32	0,32
Veritabanı İşlemleri	0,15	0,32	**0,53**
Birim Test	**0,41**	0,29	0,29
Entegrasyon Testi	**0,75**	0,12	0,13
Altyapı Maliyeti	0,09	**0,46**	0,45
Altyapı Kısıtları	0,07	0,22	**0,71**
Hata Ayıklama	**0,55**	0,29	0,17
Ölçekleme	0,06	0,22	**0,73**
Dayanıklılık	0,16	0,34	**0,50**
Gecikme (yanıt süresi)	0,09	0,39	**0,52**
Geliştirme karmaşıklığı	0,31	**0,39**	0,29
Bakım	0,08	**0,54**	0,39
Geliştirme maliyeti	**0,42**	0,24	0,34
Çalıştırma maliyeti	0,21	**0,44**	0,35

Tablo 14 detaylı incelendiğinde, geliştirme hızı ve çeviklik kriterleri, alternatifler karşısında değerlendirildiğinde sunucusuz mimarilerin bu konuda öne çıktığı gözlemlenmiştir. Karmaşıklık kriteri ele alındığında pozitif anlamda üstün çıkan mimari monolitik yapılar olarak öne çıkmıştır. Veritabanı işlemleri için sunduğu API çözümleri ile işleri biraz daha kolaylaştırabilen sunucusuz mimarilerin ön planda olduğu

görülmektedir. Test başlığında entegrasyon ve birim test kavramlarının değerlendirilmesi istenmiştir. Birim test yazım hızı ve entegrasyon testlerine duyulan minimum ihtiyaç sebebiyle monolitik mimariler test kriterlerinde iyi sonuç aldığı görülmektedir. Altyapı perspektifinden iki kriter anket değerlendirmesinde yer almıştır. Maliyet ve kısıtlar olarak oluşan bu seçenekler maliyet konusunda mikroservis yapılar ile sunucusuz yapıları hemen hemen aynı, kısıt konusunda da sunucusuz yapıların esnekliği sebebiyle diğer alternatifler karşısında öne çıkarmıştır. Hata ayıklama (debugging) geliştirme süreçleri için geliştirici adına önemli bir yer kaplar. Bu kriter değerlendirildiğinde monolitik yapıların diğer mimarilere göre pozitif ayrıştığı gözlemlenebilmektedir. Ölçekleme ve dayanıklılık kriterlerinde ise sunucusuz yapılarda, yönetimin sağlayıcılar tarafından karşılandığı düşünülürse, diğer mimarilere göre önde görülmesi makul bir sonuç gibi gözükmektedir. Gecikme (yanıt süresi) kriteri ise özellikle soğuk başlangıç (cold start) olarak adlandırılan ilk istekler, servisin tekrar ayağa kalktığı anlar gibi olaylar karşısında biraz daha geç yanıt verebilmesi mümkündür. Anket katılımları da bu konuda sunucusuz yapıları negatif ayrıştırmıştır. Geliştirme karmaşıklığı kriterini incelediğimizde ise yapısı gereği birden çok servisin ayrı ayrı yazılması ve birbirleriyle haberleşme gereği duyması gibi sebeplerle mikroservis mimarilerinin negatif olarak ön plana çıktığı görülmüştür. Bakım kriteri için ise ayrı yazılan servislere ayrı düzenleme yapılma ihtiyacı yine mikroservis yapılarının diğer yapılara göre biraz daha geride olduğu gözlemlenmiştir. Anket sonuçlarına göre geliştirme maliyeti en düşük mimari monolitik yapılar olarak görülmektedir. Bunun sebebi, tek bir depo üzerinden birçok geliştiricinin çalışabilmesi ve geliştiricilerden genellikle kod yazmak dışında farklı sorumluluklar beklenmemesi şeklinde açıklanabilir. Mikroservis mimarilerinde çalıştırma maliyetinin biraz daha fazla olduğu sonucu ortaya çıkmıştır.

Aynı anket içinde sektör katılımcılarına MVP süreçleri baz alındığında genel olarak kriterleri birbirleri arasında karşılaştırmaları istenmiştir. Karşılaştırma matrisi sonucu oluşan kriter önemleri Tablo 15'te ifade edilmiştir.

Tablo 15. Kriterlerin Önem Matrisi

Kriter	Önem Değeri
Geliştirme hızı	0,09
Çeviklik	0,08
Çalıştırma maliyeti	0,08
Dayanıklılık	0,07
Hata Ayıklama	0,07
Geliştirme maliyeti	0,07
Ölçekleme	0,07
Bakım	0,06
Altyapı Maliyeti	0,06
Gecikme (yanıt süresi)	0,06
Geliştirme karmaşıklığı	0,06
Altyapı Kısıtları	0,06
Birim Test	0,05
Karmaşıklık	0,04
Veritabanı İşlemleri	0,04
Entegrasyon Testi	0,04

Sonuç olarak alternatiflerin kriterlere göre karşılaştırıldığında elde edilen matris (Tablo 14), kriterlerin kendi içlerinde değerlendirildiği ve önem değeri kazandığı bir diğer matris (Tablo 15) beraber değerlendirildiğinde tezde örneği gösterilen AHP hesaplama teknikleri ile bir sonuca erişmek mümkün olabilir. Bu durumda ise sonuç matrisi Tablo 16'daki gibi oluşur.

Böylelikle, minimum uygulanabilir ürün kavramı için seçilebilecek en uygun mimariyi bulabilmek adına analitik hiyerarşi süreci ile bir değerlendirme yapıldığında, en yüksek skora sahip sunucusuz mimarilerin diğer alternatiflere karşı en makul seçenek olduğu görülmüştür.

Tablo 16. AHP Sonuç Matrisi

Mimariler	**Değer Ortalaması**	**Değer Yüzdesi**
Monolitik	0,24	23,67%
Mikroservis	0,33	32,75%
Sunucusuz	0,43	**43,58%**
Toplam	**1**	**100,00%**

SONUÇ

Minimum uygulanabilir ürün, bir fikrin veya problem çözümünün pazardaki müşteriler için anlamının test edildiği ve fikrin/çözümün iyileştirildiği bir döngüyü temsil eder. Her döngü sonunda karar verici fikrin geliştirilmesi veya iptal edilmesi gibi kararlar verir. Fikrin geliştirilmesi o kadar ileri boyutlara ulaşabilirki, müşteri ihtiyaçlarına göre ürünün başkalaşması dahi söz konusu olabilir çünkü burada müşteri ihtiyaçlarını doğru duyabilmek ve ihtiyaçları karşılayabilmek ürünün piyasa kabulü açısından son derece önemli olacaktır. Bu süreçler yeni girişimler açısından da oldukça önemlidir. Girişimde heyecanla oluşturulan ilk fikrin kullanıcı açısından ne ifade ettiği yine bu süreçler sonucunda somut bir çıktı oluşturabilir. Oluşan çıktı ile de ürünün/fikrin piyasada olgunlaşması sağlanabilir. Yazılım tarafında ise sürekli gelişen ve farklılaşan bir dünya mevcuttur. Yaşanan gelişmeler çeşitli avantaj ve dezavantajlar da taşıyabilmektedir. Dolayısıyla, doğru seçimler ile ürün geliştirmek piyasa kabulü içinde önemli olacaktır. Özellikle girişimlerde sıklıklıkla kullanılan minimum uygulanabilir ürün süreçlerinin tasarım ve olgunlaşma aşamalarının ihtiyaçları iyi irdelenmelidir. Eğer bu süreçlerde fikrin hızlı bir biçimde tekrar piyasaya sürülmesi gerekiyorsa burada seçecekleri yazılım mimarileri ve altyapılar da önemli rol oynayacaktır. Sonuçta odaklanılması gereken yer fikir ise bu fikrin nasıl sunulduğu süreci çok baltalamamalıdır. Bu çalışmada analitik hiyerarşi süreci ile uygulama mimarileri (monolitik, mikroservis ve sunucusuz yapılar) alternatifler olarak ele alınıp; geliştirme hızı, çeviklik, çalıştırma maliyeti, dayanıklılık, hata ayıklama, geliştirme maliyeti, ölçekleme, bakım, altyapı maliyeti, gecikme (yanıt süresi), geliştirme karmaşıklığı, altyapı kısıtları, birim test, karmaşıklık, veritabanı işlemleri ve entegrasyon testleri gibi kriterler birlikte değerlendirilmiştir. Kriter seçimlerinde ise literatür taraması ile birlikte elde edilen uygulama, altyapı ve test parametrelerinin; minimum uygulanabilir ürünün geliştirilme sürecini direkt veya dolaylı olarak etkileyebilecek olanların seçimine özen gösterilmiştir. Alanlarında tecrübeli sektör temsilcileri ile yüz yüze anketler yapılmış, kriterlere göre uygulama mimarilerinin değerlendirilmesi ve kriterlerin birbirleri arasında önem dereceleri belirlenmiş ve bunlar üstünden hesaplamalar yapılmıştır. Sonuç olarak MVP süreçlerinin henüz başında olan karar alıcılar için ihtiyaçlarını karşılayacak mimari, %43,58 ile sunucusuz yapılar olarak

ortaya konulmuştur. Bir sonraki aşamada, MVP ile olgunlaşan ürünün pazarda yüksek trafiklere ulaşmaya başlaması durumunda optimum sonuçlarla nasıl ölçeklenebileceği araştırılabilir.

KAYNAKÇA

Abd-Elwahab, A. M., Mohamed, A. G. ve Shaaban, E. M. (2023). Microservices-driven enterprise architecture model for infrastructure optimization. *Future Business Journal*, 9(1). doi:10.1186/s43093-023-00268-3

Ahluwalia, J. K., Mouradian, C., Alam, M. N. ve Glitho, R. (2022). A cloud infrastructure as a service for an efficient usage of sensing and actuation capabilities in internet of things. P. Varga vd. (Eds.) *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, *25-29 Nisan 2022*, bildiri kitabı (s. 1-6) içinde. doi:10.1109/noms54207.2022.9789805

Aleem, S., Ahmed, F., Batool, R. ve Khattak, A. (2021). Empirical investigation of key factors for SAAS architecture. *IEEE Transactions on Cloud Computing*, 9(3), 1037–1049. doi:10.1109/tcc.2019.2906299

Bass, D. (2019). *Advanced Serverless Architectures with Microsoft Azure : Design complex serverless systems quickly with the scalability and benefits of Azure.* USA: Packt Publishing Ltd.

Christidis, A., Moschoyiannis, S., Hsu, C., & Davies, R. (2020). Enabling serverless deployment of Large-Scale AI workloads. *IEEE Access*, *8*, 70150–70161. https://doi.org/10.1109/access.2020.2985282

Debski, A., Szczepanik, B., Malawski, M., Spahr, S., & Muthig, D. (2018). A scalable, reactive architecture for cloud applications. *IEEE Software*, 35(2), 62–71. doi:10.1109/ms.2017.265095722

Doğan, N., & Gencan, S. (2015). seyahat acentasi yöneticilerinin bakiş açisiyla en uygun otel seçimi: Bir analitik hiyerarşi prosesi (ahp) uygulamasi. *Erciyes Üniversitesi İktisadi Ve İdari Bilimler Fakültesi Dergisi*, (41), 69–88.

Evans, E. (2004). *Domain-driven design: tackling complexity in the heart of software.* Addison-Wesley Professional.

Goyal, S., Garg, D., & Luthra, S. (2021). Sustainable production and consumption: Analysing barriers and solutions for maintaining Green Tomorrow by using Fuzzy-AHP–Fuzzy-Topsis hybrid framework. *Environment, Development and Sustainability*, 23(11), 16934–16980. doi:10.1007/s10668-021-01357-5

Gupta, M. (2018). *Serverless Architectures with AWS: Discover how you can migrate from traditional deployments to serverless architectures with AWS*. Packt Publishing Ltd.

Hardy, T. (2023). Best architecture for an MVP: monolithic, microservices, SOA, or serverless? Erişim adresi: https://www.sparxitsolutions.com/blog/mvp-architecture/ adresinden alındı.

Jhingran, S. ve Rakesh, N. (2023). Performance analysis of microservices behavior in cloud vs containerized domain based on CPU utilization. *International Journal on Recent and Innovation Trends in Computing and Communication*, 11(6s), 509–516. doi:10.17762/ijritcc.v11i6s.6959

Johnston, S. (2014). Simple Workload &Amp; Application Portability (SWAP). *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS),* bildiri kitabı (s. 37-42) içinde. Toronto, Kanada. doi:10.1109/infcomw.2014.6849165

Khanam, Zeba & Ahsan, M.N.. (2017). Evaluating the effectiveness of test driven development: Advantages and pitfalls. *International Journal of Applied Engineering Research*. 12(18) (s. 7705-7716)

Koschel A., Klassen, S., Jdiya, K., Schaaf, M., ve Astrova, I. (2021). Cloud Computing: Serverless. *2021 12th International Conference on Information, Intelligence, Systems & Applications (IISA), 12-14 Temmuz 2021, bildiri kitabı* (s. 1-7) içinde. Girit, Yunanistan. doi: 10.1109/IISA52424.2021.9555534.

Lawton, G. (2008). Developing software online with Platform-as-a-service technology. *Computer*, 41(6), 13–15. doi:10.1109/mc.2008.185

Liu, G., Huang, B., Liang, Z., Qin, M., Zhou, H., ve Li, Z. (2020). Microservices: Architecture, container, and challenges. *IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, *11-14 Aralık 2020, bildiri kitabı* (s. 629-635) içinde doi:10.1109/qrs-c51114.2020.00107

Mangwani, P., Mangwani, N., ve Motwani, S. (2023). Evaluation of a multitenant SAAS using monolithic and Microservice Architectures. *SN Computer Science*, 4(2). doi:10.1007/s42979-022-01610-2

Marzullo, F. P., Souza, J. M., ve Blaschek, J. R. (2008). A domain-driven development approach for enterprise applications, using MDA, SOA and web services. *10th IEEE Conference on E-Commerce Technology and the Fifth IEEE Conference on Enterprise Computing, 21-24 Temmuz 2008, bildiri kitabı* (s. 432-437) içinde. doi:10.1109/cecandeee.2008.119

Menéndez, J. M., Gayo, J. E., Canal, E. R., ve Fernández, A. E. (2023). A comparison between traditional and Serverless technologies in a microservices setting. *Computer Science.* doi:10.48550/arXiv.2305.13933

Miguel Rodriguez Cortes, L., Paul Guillen, E., ve Rojas Reales, W. (2022). Serverless architecture: Scalability, implementations and open issues. *6th International Conference on System Reliability and Safety (ICSRS)*, *23-25 Kasım 2022, bildiri kitabı* (s. 331-336) içinde. doi:10.1109/icsrs56243.2022.10067577

Navaneeth, V., Krithik Vasan, B., Mageshwar, S., ve Mercy, W. (2022). Building a backend-as-a-service platform in Kubernetes. *3rd International Conference on Smart Electronics and Communication (ICOSEC), 20-22 Ekim 2022) bildiri kitabı* (s. 894-899) içinde. doi:10.1109/icosec54921.2022.9951895

Ouyang, R., Wang, J., Xu, H., Chen, S., Xiong, X., Tolba, A., ve Zhang, X. (2023). A microservice and serverless architecture for secure IOT System. *Sensors*, 23(10), 4868. doi:10.3390/s23104868

Parres-Peredo, A., Piza-Davila, I., ve Cervantes, F. (2019). Building and evaluating user network profiles for cybersecurity using serverless architecture. *2019 42nd International Conference on Telecommunications and Signal Processing (TSP)*, 1-*3 Temmuz 2019*, *bildiri kitabı* (s.164-167) içinde. doi:10.1109/tsp.2019.8768825

Peker, O., ve Toksarı, M. (2007). "Analitik hiyerarşi prosesi yaklaşımı kullanılarak mobilya sektörü için Ege Bölgesi'nde hedef pazarın belirlenmesi." *Yönetim Ve Ekonomi: Celal Bayar Üniversitesi İktisadi Ve İdari Bilimler Fakültesi Dergisi, 14*(1), 171–180. https://doi.org/10.18657/yecbu.10270

Pereira, I. M., de Senna Carneiro, T. G., ve Figueiredo, E. (2021). Understanding the context of IOT software systems in DevOps. *2021 IEEE/ACM 3rd International Workshop on Software Engineering Research and Practices for the IoT (SERP4IoT), 3 Temmuz 2021), bildiri kitabı* (s. 13-20) içinde. doi:10.1109/serp4iot52556.2021.00009

Portman, H. (2022). The Minimum Viable Product (MVP) unraveled; sensemaking in the Agile Forest. *PM World Journal*, 11(8) 1-5.

Prasanna, K., Ramana, K., Dhiman, G., Kautish, S., ve Chakravarthy, V. D. (2021). POC Design: A methodology for proof-of-concept (POC) development on internet of things connected dynamic environments. *Security and Communication Networks*, 2021, 1–12. doi:10.1155/2021/7185827

Prasanthan, A., Anand, K. S., Nair, B. P., Gautham Santhosh, K., ve Swaminathan, J. (2023). Low code backend as a service platform. *2023 World Conference on Communication & Computing (WCONF)*. doi:10.1109/wconf58270.2023.10234969

Preimesberger, C. (2016). How Enterprises Can Use Microservices to Their Advantage. Erişim adresi: https://arxiv.org/abs/2309.03796

Rattanukul, P., Makaranond, C., Watanakulcharus, P., Ragkhitwetsagul, C., Nearunchorn, T., Visoottiviseth, V., … ve Sunetnanta, T. (2023). Microusity: A testing tool for

backends for frontends (BFF) Microservice Systems. Erişim adresi: https://arxiv.org/pdf/2302.11150.pdf

Seedat, M., Abbas, Q., ve Ahmad, N. (2022). Systematic Mapping of Monolithic Applications to Microservices Architecture. Erişim adresi: https://arxiv.org/abs/2309.03796

Shahand, S., Van Duffelen, J., ve Olabarriaga, S. D. (2015). Reflections on science gateways sustainability through the business model canvas: case study of a neuroscience gateway. *Concurrency and Computation: Practice and Experience*, *27*(16), 4269–4281. https://doi.org/10.1002/cpe.3524

Suartini, Y., N. K., Hendra Divayana, D. G., ve Erawati Dewi, L. J. (2023). Comparison analysis of AHP-Saw, AHP-WP, AHP-Topsis Methods in private tutor selection. *International Journal of Modern Education and Computer Science*, *15*(1), 28–45. doi:10.5815/ijmecs.2023.01.03

Sun, A., Zhou, J., Ji, T., ve Yue, Q. (2011). CSB: Cloud service bus based public SAAS platform for small and median enterprises. *International Conference on Cloud and Service Computing*, *12-14 Aralık 2011, bildiri kitabı* (s.309-314) içinde. doi:10.1109/csc.2011.6138539

Tanasseri, N., ve Rai, R. (2017). *Microservices with Azure : Architect enterprise-grade, Microservice-based solutions using Microsoft Azure service fabric*. Birmingham: Packt Publishing Ltd.

Lenarduzzi, V., Lomio, F., Saarimäki, N., ve Taibi, D. (2020). Does migrating a monolithic system to microservices decrease the technical debt? *Journal of Systems and Software*, 169, 110710. doi:10.1016/j.jss.2020.110710

Tripathi, N., Oivo, M., Liukkunen, K., ve Markkula, J. (2019). Startup ecosystem effect on minimum viable product development in software startups. *Information and Software Technology*, 114, 77–91. doi:10.1016/j.infsof.2019.06.008

Velepucha, V. ve Flores, P. (2023). A survey on microservices architecture: Principles, patterns and migration challenges. *IEEE Access*, 11, 88339–88358. doi:10.1109/access.2023.3305687

Veuvolu, R., Suryadevar, A., Vignesh, T., ve Avthu, N. R. (2023). Cloud computing based (serverless computing) using serverless architecture for dynamic web hosting and cost optimization. *International Conference on Computer Communication and Informatics (ICCCI), 23-25 Ocak 2023, bildiri kitabı* (s. 1-6) içinde. doi:10.1109/iccci56745.2023.10128286

Villamizar, M., Garces, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., …ve Lang, M. (2016). Infrastructure cost comparison of running web applications in the cloud using AWS Lambda and monolithic and Microservice Architectures. *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 16-19 Mayıs 2016, bildiri kitabı* (s. 179-182) içinde. Cartagena, Kolombiya. doi:10.1109/ccgrid.2016.37

Wu, C.-R., Chang, C.-W., ve Lin, H.-L. (2006). Evaluating the organizational performance of Taiwanese hospitals using the analytic hierarchy process. *Journal of Statistics and Management Systems*, 9(3), 633–649. doi:10.1080/09720510.2006.10701227

Yasrab, R. ve Gu, N. (2016). Multi-cloud Paas Architecture (MCPA): A solution to Cloud Lock-in. *3rd International Conference on Information Science and Control Engineering (ICISCE), 8-10 Temmuz 2016, bildiri kitabı* (s. 473-477) içinde. doi:10.1109/icisce.2016.108

Zuo, X., Su, Y., Wang, Q., ve Xie, Y. (2020). An API gateway design strategy optimized for persistence and coupling. *Advances in Engineering Software*, 148, 102878. doi:10.1016/j.advengsoft.2020.102878

EKLER

Ek-1: Kriterlerin alternatiflere göre önem puanlaması anket örneği

"Geliştirme hızı" kriterine göre değerlendırıldığında ıkı alternatıften üstün olanı ve üstünlük derecesını puanlayabılır mısınız

	9 (Belirgin Üstün)	7 (Çok Üstün)	5 (Üstün)	3 (Biraz Üstün)	1 (Eşit)	3 (Biraz Üstün)	5 (Üstün)	7 (Çok Üstün)	9 (Belirgin Üstün)	
Mikroservis							5			Monolitik
Sunucusuz			5							Monolitik
Sunucusuz		7								Mikroservis

"Çeviklik" kriterine göre değerlendirildiğinde iki alternatiften üstün olanı ve üstünlük derecesini puanlayabilir misiniz?

	9 (Belirgin Üstün)	7 (Çok Üstün)	5 (Üstün)	3 (Biraz Üstün)	1 (Eşit)	3 (Biraz Üstün)	5 (Üstün)	7 (Çok Üstün)	9 (Belirgin Üstün)	
Mikroservis								7		Monolitik
Sunucusuz				3						Monolitik
Sunucusuz		7								Mikroservis

"Karmaşıklık" kriterine göre değerlendirildiğinde iki alternatiften üstün olanı ve üstünlük derecesini puanlayabilir misiniz?

	9 (Belirgin Üstün)	7 (Çok Üstün)	5 (Üstün)	3 (Biraz Üstün)	1 (Eşit)	3 (Biraz Üstün)	5 (Üstün)	7 (Çok Üstün)	9 (Belirgin Üstün)	
Mikroservis								7		Monolitik
Sunucusuz							5			Monolitik
Sunucusuz			5							Mikroservis

"Veritabanı İşlemleri" kriterine göre değerlendirildiğinde iki alternatiften üstün olanı ve üstünlük derecesini puanlayabilir m

	9 (Belirgin Üstün)	7 (Çok Üstün)	5 (Üstün)	3 (Biraz Üstün)	1 (Eşit)	3 (Biraz Üstün)	5 (Üstün)	7 (Çok Üstün)	9 (Belirgin Üstün)	
Mikroservis					1					Monolitik
Sunucusuz					1					Monolitik
Sunucusuz					1					Mikroservis

"Birim Test" kriterine göre değerlendırıldığında ıkı alternatıften üstün olanı ve üstünlük derecesını puanlayabılır mısınız?

	9 (Belirgin Üstün)	7 (Çok Üstün)	5 (Üstün)	3 (Biraz Üstün)	1 (Eşit)	3 (Biraz Üstün)	5 (Üstün)	7 (Çok Üstün)	9 (Belirgin Üstün)	
Mikroservis								7		Monolitik
Sunucusuz							5			Monolitik
Sunucusuz			5							Mikroservis

"Entegrasyon Testi" kriterine göre değerlendirildiğinde iki alternatiften üstün olanı ve üstünlük derecesini puanlayabilir mis

	9 (Belirgin Üstün)	7 (Çok Üstün)	5 (Üstün)	3 (Biraz Üstün)	1 (Eşit)	3 (Biraz Üstün)	5 (Üstün)	7 (Çok Üstün)	9 (Belirgin Üstün)	
Mikroservis								7		Monolitik
Sunucusuz							5			Monolitik
Sunucusuz		7								Mikroservis

"Altyapı Maliyeti" kriterine göre değerlendirildiğinde iki alternatiften üstün olanı ve üstünlük derecesini puanlayabilir misin

	9 (Belirgin Üstün)	7 (Çok Üstün)	5 (Üstün)	3 (Biraz Üstün)	1 (Eşit)	3 (Biraz Üstün)	5 (Üstün)	7 (Çok Üstün)	9 (Belirgin Üstün)	
Mikroservis								7		Monolitik
Sunucusuz		7								Monolitik
Sunucusuz	9									Mikroservis

"Altyapı Kısıtları" kriterine göre değerlendirildiğinde iki alternatiften üstün olanı ve üstünlük derecesini puanlayabilir misin

	9 (Belirgin Üstün)	7 (Çok Üstün)	5 (Üstün)	3 (Biraz Üstün)	1 (Eşit)	3 (Biraz Üstün)	5 (Üstün)	7 (Çok Üstün)	9 (Belirgin Üstün)	
Mikroservis						3				Monolitik
Sunucusuz		7								Monolitik
Sunucusuz	9									Mikroservis

"Debbugging" kriterine göre değerlendirildiğinde iki alternatiften üstün olanı ve üstünlük derecesini puanlayabilir misiniz?

	9 (Belirgin Üstün)	7 (Çok Üstün)	5 (Üstün)	3 (Biraz Üstün)	1 (Eşit)	3 (Biraz Üstün)	5 (Üstün)	7 (Çok Üstün)	9 (Belirgin Üstün)	
Mikroservis						3				Monolitik
Sunucusuz							5			Monolitik
Sunucusuz							5			Mikroservis

"Olçekleme" kriterine göre değerlendirildiğinde iki alternatiften üstün olanı ve üstünlük derecesini puanlayabilir misiniz?

	9 (Belirgin Üstün)	7 (Çok Üstün)	5 (Üstün)	3 (Biraz Üstün)	1 (Eşit)	3 (Biraz Üstün)	5 (Üstün)	7 (Çok Üstün)	9 (Belirgin Üstün)	
Mikroservis		7								Monolitik
Sunucusuz		7								Monolitik
Sunucusuz			5							Mikroservis

"Dayanıklılık" kriterine göre değerlendirildiğinde iki alternatiften üstün olanı ve üstünlük derecesini puanlayabilir misiniz?

	9 (Belirgin Üstün)	7 (Çok Üstün)	5 (Üstün)	3 (Biraz Üstün)	1 (Eşit)	3 (Biraz Üstün)	5 (Üstün)	7 (Çok Üstün)	9 (Belirgin Üstün)	
Mikroservis	9									Monolitik
Sunucusuz	9									Monolitik
Sunucusuz				3						Mikroservis

"Gecikme (Yanıt Süresi)" kriterine göre değerlendırıldığında ıkı alternatıften üstün olanı ve üstünlük derecesını puanlayabı

	9 (Belirgin Üstün)	7 (Çok Üstün)	5 (Üstün)	3 (Biraz Üstün)	1 (Eşit)	3 (Biraz Üstün)	5 (Üstün)	7 (Çok Üstün)	9 (Belirgin Üstün)	
Mikroservis		7								Monolitik
Sunucusuz						3				Monolitik
Sunucusuz							5			Mikroservis

"Geliştirme Karmaşıklığı" kriterine göre değerlendirildiğinde iki alternatiften üstün olanı ve üstünlük derecesini puanlayabı

	9 (Belirgin Üstün)	7 (Çok Üstün)	5 (Üstün)	3 (Biraz Üstün)	1 (Eşit)	3 (Biraz Üstün)	5 (Üstün)	7 (Çok Üstün)	9 (Belirgin Üstün)	
Mikroservis								7		Monolitik
Sunucusuz							5			Monolitik
Sunucusuz			5							Mikroservis

"Bakım" kriterine göre değerlendirildiğinde ıkı alternatiften üstün olanı ve üstünlük derecesini puanlayabilir misiniz?

	9 (Belirgin Üstün)	7 (Çok Üstün)	5 (Üstün)	3 (Biraz Üstün)	1 (Eşit)	3 (Biraz Üstün)	5 (Üstün)	7 (Çok Üstün)	9 (Belirgin Üstün)	
Mikroservis					1					Monolitik
Sunucusuz					1					Monolitik
Sunucusuz					1					Mikroservis

"Geliştirme Maliyeti" kriterine göre değerlendirildiğinde iki alternatiften üstün olanı ve üstünlük derecesini puanlayabilir mi

	9 (Belirgin Üstün)	7 (Çok Üstün)	5 (Üstün)	3 (Biraz Üstün)	1 (Eşit)	3 (Biraz Üstün)	5 (Üstün)	7 (Çok Üstün)	9 (Belirgin Üstün)	
Mikroservis								7		Monolitik
Sunucusuz								7		Monolitik
Sunucusuz					1					Mikroservis

"Çalıştırma Maliyeti" kriterine göre değerlendirildiğinde iki alternatiften üstün olanı ve üstünlük derecesini puanlayabilir m

	9 (Belirgin Üstün)	7 (Çok Üstün)	5 (Üstün)	3 (Biraz Üstün)	1 (Eşit)	3 (Biraz Üstün)	5 (Üstün)	7 (Çok Üstün)	9 (Belirgin Üstün)	
Mikroservis								7		Monolitik
Sunucusuz								7		Monolitik
Sunucusuz					1					Mikroservis

Ek-2: Kriterlerin kendi içinde üstünlük puanlaması örneği

Örnek: MVP süreçlerinde **Çeviklik** kriteri **Geliştirme Hızı**'na göre daha **"Çok Önemli"** ise tabloda d

MVP süreçleri için kriterleri karşılaştırmalı olarak puanlayabilir misiniz? Örnek: MVP süreçlerinde **Geliştirme Hızı** kriteri, **Veritabanı İşlemleri**'ne göre **"Biraz Önemli"** ise t

	9 (Belirgin Önemli)	7 (Çok Önemli)	5 (Önemli)	3 (Biraz Önemli	1 (Eşit)	3 (Biraz Önemli)	5 (Önemli)	7 (Çok Önemli)	9 (Belirgin Önemli)	
Çeviklik			5							Geliştirme Hızı
Karmaşıklık								7		Geliştirme Hızı
Veritabanı İşlemleri							5			Geliştirme Hızı
Birim Test						3				Geliştirme Hızı
Entegrasyon Testi						3				Geliştirme Hızı
Altyapı Maliyeti								7		Geliştirme Hızı
Altyapı Kısıtları								7		Geliştirme Hızı
Debugging						3				Geliştirme Hızı
Ölçekleme					1					Geliştirme Hızı
Dayanıklılık					1					Geliştirme Hızı
Gecikme (yanıt süresi)					1					Geliştirme Hızı
Geliştirme karmaşıklığı								7		Geliştirme Hızı
Bakım								7		Geliştirme Hızı
Geliştirme maliyeti					1					Geliştirme Hızı
Çalıştırma maliyeti					1					Geliştirme Hızı
Karmaşıklık									9	Çeviklik
Veritabanı İşlemleri							5			Çeviklik
Birim Test							5			Çeviklik
Entegrasyon Testi						3				Çeviklik
Altyapı Maliyeti								7		Çeviklik
Altyapı Kısıtları								7		Çeviklik
Debugging						3				Çeviklik
Ölçekleme					1					Çeviklik
Dayanıklılık					1					Çeviklik
Gecikme (yanıt süresi)					1					Çeviklik
Geliştirme karmaşıklığı								7		Çeviklik
Bakım								7		Çeviklik
Geliştirme maliyeti								7		Çeviklik
Çalıştırma maliyeti								7		Çeviklik
Veritabanı İşlemleri		7								Karmaşıklık
Birim Test			5							Karmaşıklık
Entegrasyon Testi			5							Karmaşıklık
Altyapı Maliyeti		7								Karmaşıklık
Altyapı Kısıtları		7								Karmaşıklık
Debugging		7								Karmaşıklık
Ölçekleme		7								Karmaşıklık
Dayanıklılık		7								Karmaşıklık
Gecikme (yanıt süresi)		7								Karmaşıklık
Geliştirme karmaşıklığı			5							Karmaşıklık
Bakım						3				Karmaşıklık
Geliştirme maliyeti		7								Karmaşıklık
Çalıştırma maliyeti		7								Karmaşıklık
Birim Test				3						Veritabanı İşlemleri
Entegrasyon Testi				3						Veritabanı İşlemleri
Altyapı Maliyeti		7								Veritabanı İşlemleri
Altyapı Kısıtları		7								Veritabanı İşlemleri
Debugging		7								Veritabanı İşlemleri
Ölçekleme		7								Veritabanı İşlemleri
Dayanıklılık		7								Veritabanı İşlemleri
Gecikme (yanıt süresi)		7								Veritabanı İşlemleri
Geliştirme karmaşıklığı			5							Veritabanı İşlemleri
Bakım				3						Veritabanı İşlemleri
Çalıştırma maliyeti		7								Veritabanı İşlemleri
Altyapı Maliyeti		7								Entegrasyon Testi
Altyapı Kısıtları			5							Entegrasyon Testi
Debugging		7								Entegrasyon Testi
Ölçekleme		7								Entegrasyon Testi
Dayanıklılık		7								Entegrasyon Testi
Gecikme (yanıt süresi)		7								Entegrasyon Testi
Geliştirme karmaşıklığı			5							Entegrasyon Testi
Bakım				3						Entegrasyon Testi
Geliştirme maliyeti		7								Entegrasyon Testi
Çalıştırma maliyeti		7								Entegrasyon Testi
Altyapı Kısıtları					1					Altyapı Maliyeti
Debugging					1					Altyapı Maliyeti
Ölçekleme				3						Altyapı Maliyeti
Dayanıklılık				3						Altyapı Maliyeti
Gecikme (yanıt süresi)				3						Altyapı Maliyeti
Geliştirme karmaşıklığı				3						Altyapı Maliyeti
Bakım							5			Altyapı Maliyeti
Geliştirme maliyeti			5							Altyapı Maliyeti
Çalıştırma maliyeti					1					Altyapı Maliyeti
Debugging					1					Altyapı Kısıtları
Ölçekleme				3						Altyapı Kısıtları
Dayanıklılık				3						Altyapı Kısıtları
Gecikme (yanıt süresi)				3						Altyapı Kısıtları
Geliştirme karmaşıklığı				3						Altyapı Kısıtları
Bakım							5			Altyapı Kısıtları
Geliştirme maliyeti			5							Altyapı Kısıtları
Çalıştırma maliyeti					1					Altyapı Kısıtları
Ölçekleme		7								Debugging
Dayanıklılık		7								Debugging
Gecikme (yanıt süresi)		7								Debugging
Geliştirme karmaşıklığı		7								Debugging
Bakım								7		Debugging
Geliştirme maliyeti		7								Debugging
Çalıştırma maliyeti			5							Debugging
Dayanıklılık						3				Ölçekleme
Gecikme (yanıt süresi)					1					Ölçekleme
Geliştirme karmaşıklığı					1					Ölçekleme
Bakım								7		Ölçekleme
Geliştirme maliyeti					1					Ölçekleme
Çalıştırma maliyeti					1					Ölçekleme
Gecikme (yanıt süresi)					1					Dayanıklılık
Geliştirme karmaşıklığı				3						Dayanıklılık
Bakım							5			Dayanıklılık
Geliştirme maliyeti		7								Dayanıklılık
Çalıştırma maliyeti			5							Dayanıklılık
Geliştirme karmaşıklığı				3						Gecikme (yanıt süresi)
Bakım							5			Gecikme (yanıt süresi)
Geliştirme maliyeti		7								Gecikme (yanıt süresi)
Çalıştırma maliyeti			5							Gecikme (yanıt süresi)
Bakım								7		Geliştirme karmaşıklığı
Geliştirme maliyeti			5							Geliştirme karmaşıklığı
Çalıştırma maliyeti					1					Geliştirme karmaşıklığı
Geliştirme maliyeti		7								Bakım
Çalıştırma maliyeti			5							Bakım
Çalıştırma maliyeti							5			Geliştirme maliyeti