FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

MİKROSERVİS TABANLI ÖDEME SİSTEMİ TASARIMI VE GERÇEKLENMESİ

YÜKSEK LİSANS TEZİ

YUSUF KOCAMAN

tarafından

YÜKSEK LİSANS

derecesi şartını sağlamak için hazırlanmıştır.

Ocak 2018

Program: Bilgisayar Mühendisliği

MİKROSERVİS TABANLI ÖDEME SİSTEMİ TASARIMI VE GERÇEKLENMESİ

YÜKSEK LİSANS TEZİ

YUSUF KOCAMAN

tarafından

OKAN ÜNİVERSİTESİ

Bilgisayar Mühendisliği Anabilim Dalına

Yüksek Lisans

derecesi şartını sağlamak için sunulmuştur.

Onaylayan:

Danışman
Prof. Dr. Bekir Tevfik AKGÜN

Üye
Prof. Dr. Semih BİLGEN

Üye
Doç. Dr. Erchan APTOULA

Ocak 2018

Program: Bilgisayar Mühendisliği

ÖZET

Bu tezde, daha çok büyük ölçekli oyunlara yönelik, satın alınan jetonların tutulması ve harcanması gibi hizmetleri destekleyen bir ödeme sistemi yapılmıştır. Temel problemlerden ilki; başlangıçta az sayıda olan oyuncuların, oyunun popülerleşmesinden sonra kısa sürede artmasıdır. Bu önemli bir ölçekleme problemi yaratır. İkincisi ise; büyük ölçekli oyunlarda işlem hızı, oyunun sürdürülebilmesi için oldukça önemli olduğundan kısa zamanda çok sayıda üretilen hareketlerin (transaction), kayıp vermeden uygun maliyet ve insan kaynağı ile karşılanmasıdır. Şirketlerin hızlı ve yüksek kazanç elde edilebilmesi için oyunlarının keşfedilmesi önemlidir. Bu durum özellikle birkaç kişilik küçük oyun şirketlerinde daha iyi gözlemlenebilmektedir. Çok sayıda oyun sunulduğundan ve oyuncuların ilgisi de kolayca değiştiğinden dolayı, fırsatları değerlendirmek için uzun süreler yoktur.

Sonuç olarak; ihtiyaçlar doğrultusunda açık kaynak yazılım teknolojileri dahil, çok sayıda teknoloji ve yazılım paketi incelenmiş, konuya en uygun teknolojiler ele alınmış ve bazıları kullanılarak uygulama örneği geliştirilmiştir. Amaç; ölçeklenebilir, hızlı yanıt veren, oyun uygulamalarına kolay uyarlanabilir bir sistem sunulmasıdır. Gerçeklenen uygulamanın kullanıcılar tarafından sınanması mümkün olmadığından, testler simülasyon ortamında gerçekleştirilmiştir.

Anahtar Kelimeler: Ödeme sistemi, mikroservis, uygulama ölçeklendirme

ABSTRACT

In this thesis, a payment system was built to support services such as keeping track of points earned, scoring points and purchasing additional shares. The main problems are; At the beginning, few player users are artwork shortly after the game coalesces. This creates a significant scalability problem. The second is; In large-scale games, the processing speed is very important for the game to be sustained in a short time and the very high transaction are met with reasonable cost and human resource in time without giving loss. This situation can be better observed in small game companies that have not played once. It is important for users to discover games, so that companies can get fast and high profits. You can not constantly change the offerings of sellers with very acclaimed users and change the player interest, there is no long time to evaluate the opportunities.

As a result, a large number of software packages, including open source software technologies, have been examined in accordance with the needs, the most appropriate technologies have been discussed and some of them have developed an application example to accommodate them. The goal is providing a system which scalable, able to fast response and easily adaptable for game applications. Since it is not possible for users to test the application, the tests were performed in the simulation.

Keywords: Payment system, microservices, application scaling

TEŞEKKÜR

Çalışmam süresince bana yol gösteren, sahip olduğu deneyimleri benimle paylaşan tez danışman hocam Sayın Prof. Dr. Bekir Tevfik AKGÜN'e teşekkür ederim.

Bu çalışma boyunca yardımlarını ve manevi desteklerini esirgemeyen aileme, arkadaşlarıma ve yöneticilerime minnet ve teşekkürlerimi sunarım.

Yusuf KOCAMAN

Ocak 2018

İÇİNDEKİLER

TABLO LİSTESİ

ŞEKİL LİSTESİ

KISALTMALAR

API	**:**	Uygulama Programlama Arayüzü	Application Programming Interface
CSS	**:**	Basamaklı Biçim Sayfaları	Cascading Style Sheets
HTML	**:**	Hiper Metin İşaretleme Dili	Hypertext Markup Language
JSON	**:**	JavaScript Nesne Gösterimi	JavaScript Object Notation
JWT	**:**	JSON Web Jetonu	JSON Web Token
MSTOS	**:**	Mikroservis Tabanlı Ödeme Sistemi	
NPM	**:**	Node.js Paket Yöneticisi	Node.js Package Manager
REST	**:**	Temsili Durum Transferi	Representational State Transfer
URL	**:**	Tekdüzen Kaynak Bulucu	Uniform Resource Locator

1. GİRİŞ

Uygulama geliştiren şahıs veya firmalar, varlıklarını sürdürebilmeleri için belirledikleri bir yöntem ile gelir sağlamak zorundadırlar. Ücretsiz oyunlar için reklam yayını yapmak genel uygulanan bir kazanç faaliyeti olabilir. Ancak anlamlı gelir düzeyine gelmek için çok sayıda oyuncunun, uzun sürelerde oynaması gerekmektedir. Örneğin oyuncudan eşya veya puan karşılığı ödeme alınması daha yüksek kazançlar sağlayabilir. Kapsamlı oyun projelerinde satın alma ve ödeme prosedürleri ayrıca ele alınıp, oyuna dahil edilebilir. Bu imkanı bulamayan, reklam yayını ve benzeri düşük kazanç yöntemleri üzerinden ilerleyen veya ücretsiz hizmete sunulan bir oyun projesinin, aniden beklenenden daha fazla talep görmesi durumunda, oyun sahibinin bunu yüksek kazanca dönüştürmesi kolay olmamaktadır. Dolayısı ile oyun yazılımının etkin bir ödeme sistemine dahil olması gerekir. Oyuncunun ise oyunun kesintisiz, hızlı ve hatasız hizmet alması beklenir. Günümüz kullanıcıları için zamanlamanın değerli olması ve oyun alternatiflerinin kolaylıkla bulunabilmesi sebebi ile üreticilerin, bu süreçleri problemsiz atlatması gerekmektedir.

Önemli insan gücü ve zaman ile üretilen bir oyun projesinin beklenmeyen yüksek oranda bir kullanıcı ilgisi karşısında veya belirli bir sadık kullanıcı kitlesi edinmesi sonucunda hızlıca oyuncularından kazanç ve sermaye artırımı sağlayabilmesi için ölçeklenebilir bir ödeme sistemi ile desteklenmesi gerekmektedir. Çalışmada gerçeklenmiştir. Bu tezde ölçeklenebilir ve uygulamalara kolay uyarlanabilir MSTOS ismi verilen bir “Mikroservis Tabanlı Ödeme Sistemi” geliştirilmiştir. MSTOS’nin

oyuncu ve firma müşterilerine yönelik uygulamaları da geliştirilmiş ve simülasyon ile tüm sistemin başarımı izlenmiştir.

Bu çalışmada, oyuncuların çok sayıda olabilen oyun firmalarına güvenilir ortak bir sistem ile hızlıca ödeme yapılabilmesi hedeflenmiştir. Oyuncular, ödemeleri karşılığında harcayabilecekleri jetonlar satın alır. Bu jetonlara "MSTOS Jetonu" denilir. Oyuncunun kullandığı bir jeton firma tarafından oyun puanına dönüştürülerek, oyuna (oyuncuya) yansıtılır. Jetonlar, oyuncular veya oyunlar arasında transfer edilebilir. Oyuncuların, oyun anında oyunu terk etmeden ve zaman kaybetmeden hızlıca ödeme yapabilmeleri için bir çözüm üretilmiştir. Oyuncuların ve firmaların bağlığını kazanabilmek için avans jeton verilmesi gibi öneriler de tez kapsamında bulunmaktadır. Firmalardan, sunulan hizmet karşılığı olarak bir komisyon talep edilmektedir. Komisyon yüzdesi, oyuncuların firmalar için yaptığı ödemelerin işlem adetlerine göre belirlenmektedir. Tasarlanan sistem, kazançların toplanmasını ve dağıtılmasını kolaylaştırmaktadır. Oyuncular, ödemelerini bir sisteme yapar. Oyun firmaları, ödeme işlemi için bir firmadan hizmet almış olur. Böylece mali açıdan çoktan - çoğa ilişkisi yerine tekten - teke yapısı gelir. Kazançlar için kanun gereği vergilendirme mevzuatları ile ilgili konular tam olarak teze dahil edilmemiştir.

Bu çalışmada teknolojiler olarak Go'da makine diline derlenen bir web ödeme sistemi, dağıtık çalışan NoSQL türündeki Cassandra veritabanı, Docker sanallaştırma platformunda ölçeklendirme, Rancher ile dağıtma ve minimum ağ/kaynak trafiği için JWT ve Rest tabanlı arayüz kullanılmıştır. Kullanılan sanallaştırma teknolojisi

sayesinde sistem kaynaklarının verimli kullanımına ek olarak, güvenlik zafiyetlerini en aza indirmeyi sağlayan servis yalıtımı da sağlanmıştır.

1.1. Önceki Çalışmalar

Son kullanıcılara hizmet veren bazı uygulamalar tespit edilmiştir. Bu uygulamaların, sadece ödeme sistemi veya ödeme sistemini barındıran platformlar oldukları görülmüştür. Amaçlarına göre, oyunlara yönelik ve genel amaçlı olarak iki ayrı bölümde incelenmiştir. Genel yapıları ve özellikleri hakkında kısaca bilgiler verilmiştir. Bunlar arasında Steam, oyuncular arasında oldukça popüler ödeme sistemi de içeren bir oyun platformudur [1]. Genel amaçlı ödeme yöntemleri, örnek olarak Bitcoin, oyunlar içerisinde ödeme kanalı olarak kullanılabildiği için değerlendirilmeye alınmıştır.

1.1.1. Oyunlara Yönelik Ödeme Sistemleri

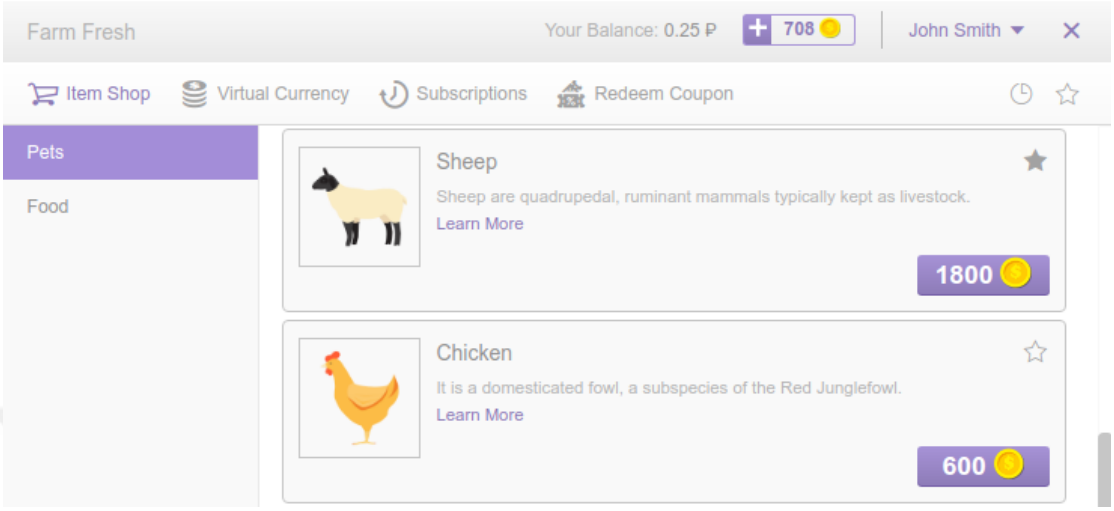
Steam, Valve tarafından 2003 yılında yayına alınan oyuncu platformudur. Oyuncular, yeni oyunları satın alabilmektedir. Oyuncu profilinde seviye, puan, rozet ve eşyaları listelenmektedir. İhtiyaçları halinde rozet ve eşya alım-satımını yapabilmektedirler. Oyuncular tarafından üretilen yeni oyun içi ögelerin, örnek olarak arkaplan görseli, alışverişi de yapılabilmektedir. Oyuncu eşyalarına gelen ödeme tekliflerini görüp, değerlendirebilmektedir. Oyuncular birbirleri arasında oyun ve eşyaları hediye olarak gönderebilmektedir. Valve haricinde diğer oyun yapımcıları da oyunlarının satışını Steam üzerinden yapabilmektedir. Bunun için öncelikle oyuncuların, Steam'e dahil

olmasını istedikleri oyunlar için yaptıkları oylamada başarılı olmaları gerekmektedir [2].

Şekil 1.1. Steam üye oyuncu arayüzü

GameSparks, oyun firmalarının sunucu üzerindeki standart arkaplan işlemlerini servis olarak sunan bir platformdur. Puan tablosu, sanal para, rozet ve eşya yönetimi için uygulama programlama arayüzü servisi vermektedir. Eşya alım-satım ve ödeme için desteği vardır. Oyuncuların yönetimi, bildirim gönderimi ve market oluşturma vb. diğer önemli yetenekleri de içermektedir. Oyun kodlarının çalıştırılması, ölçekleme ve veritabanı gibi altyapısal hizmetleri bulunmaktadır [3].

Xsolla, 2006 yılında kurulmuştur. Oyun firmaları için ödeme, faturalandırma, dağıtım ve pazarlama araçları sunmaktadır. Oyun firmasının belirlediği kriterlere göre sanal para, eşya, aylık abonelik ve lisans satışı veya belirlenen tutarda ödeme alma hizmetleri vermektedir. Oyuncu seviye veya puan bilgilerini saklamamaktadır [4] [5] .



Şekil 1.2. Xsolla'nın resmi demosu

Apple App Store, iOS işletim sistemi yüklü olan mobil cihaz, tablet ve ek olarak OS X işletim sistemi yüklü bilgisayarlar için hizmet vermektedir. Oyun satın alma ve erişim hizmeti vermektedir. Oyuncuların profil bilgilerini de tutmaktadır. Oyun ile ilgili yorum ve puanlandırma özellikleri vardır. Oyun içerisinde eşya veya hizmet satışına destek vermektedir. Android yüklü olan mobil cihaz, tabletlerde ise Google Play platformu aynı özellikler ile hizmet vermektedir [6]–[8].

Facebook Games, Facebook tarafından oyunların sunulduğu bir platformdur. Oyun ve oyun içi satın alma ile oyuncular arası hediyeleşme imkanı vermektedir. Oyuncunun iznine bağlı olarak oyuna, özel bilgilerine erişim ve duvarının kontrolü yetkisini vermektedir [9] [10].

itch.io, dağıtıcı firmalardan bağımsız olarak geliştirilen oyunların, barındırıldığı ve dağıtımının yapıldığı pazar olarak çalışan bir platformdur. Oyun firması yüklediği oyunu için sabit bir fiyat belirleyebilir ve indiren oyuncu başına kazanç sağlayabilir. itch.io oyunun indirilmesi sonrasında herhangi bir destek vermemektedir [11].

1.1.2. Diğer Yaygın Ödeme Yöntemleri

Uluslararası para transferi için yaygın olarak kullanılan yöntemler aşağıda anlatılmıştır. Bu sistemler oyun harcamaları için tasarlanmadığından dolayı oyuncular için oyun esnasında beklenen kolaylıkları sunamazlar.

Western Union, 1851 yılında kurulan uluslararası para transferi servisidir. 200'den fazla ülke ve bölgede bulunan 500,000'den fazla acenteye sahiptir. Dünya çapında hızlı ve güvenli olarak para transferi hizmeti vermektedir. Belirli miktar aralıkları için sabit gönderim ücreti bulunmaktadır. Her transfer işlemi için otomatik olarak Para Transfer Kontrol Numarası (MTCN) oluşturur. Alıcının acenteden parasını alabilmesi için bu numarayı ve resmi kimliğini ibraz etmesi gerekir [12] [13].

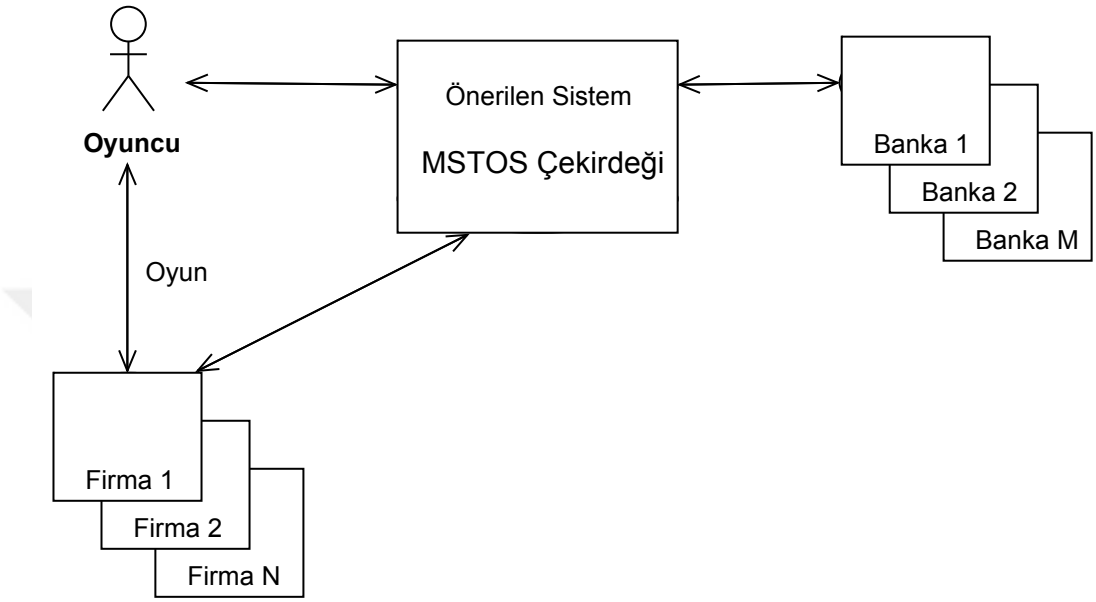
Swift, küresel çapta üyeleri bulunan bir kooperatif olarak dünyanın önde gelen güvenli finansal mesajlaşma hizmeti sağlayıcısıdır. 200'den fazla ülkede 11,000'den fazla banka kuruluşu ve organizasyon arasında finansal bağlantıyı sağlamaktadır. Bu kuruluş ve organizasyonlar arasında para transferi masraflarının düşük ve hatasız olması hedeflenmektedir. Swift, müşterilerinin hesaplarını barındıran veya yöneten bir kuruluş değildir. BIC veya Swift kodu ismi verilen standardı sayesinde üyesi olan

her bankanın şube ve konumuna göre özel kodu vardır. Bu kod ve hesap numarası ile dünya çapında para transferi yapılabilmektedir [14] [15].

Paypal, kimlik bilgilerini paylaşmadan, hızlı ve güvenli olarak ödeme yapmayı sağlayan çevrim-içi ödeme hizmetidir. 192 milyon aktif kullanıcısı bulunmaktadır. Para transferi için alıcının e-posta hesabını bilmek yeterlidir. Ayrıca ödeme almak isteyen web siteleri için çevrim-içi ödeme formu hizmeti vermektedir. Çok sayıda para birimi arasındaki dönüşümleri, döviz kurlarına göre otomatik olarak yapar. Bu sayede uluslararası alışveriş için kolaylık sağlamaktadır [16].

Bitcoin, sahibi veya merkezi yönetimi bulunmayan bir dijital paradır. Dünyanın her yerine, zaman ve miktar limiti olmadan transferi yapılabilir. Tüm transfer işlemleri "blok zinciri" adı verilen ve tüm ağa açık olan bir ana hesap defterine kaydedilmektedir. Bitcoin sahibi kişilerin, sanal imzalarını içeren cüzdanları bulunur. Bitcoin transferi için göndericinin, sanal imzası ile işlemi imzalaması gerekir. SHA256 şifreleme ile yapılan imzalama işlemi, ağdaki tüm sunucular tarafından teyit edilir. Onay alması durumunda, transfer işlemi blok zincirine kaydedilir. Sahip olunan miktarın itibari paralara dönüşümü için takas yapılmalıdır. Dönüşüm için aracılık yapan çok sayıda web sitesi vardır. Değişen alım-satım talebine göre Bitcoin'in değeri değişebilmektedir. Bitcoin yazılımının kaynak kodları, ücretsiz olarak herkesin erişimine açıktır [17].

1.2. Önerilen Mikroservis Tabanlı Ödeme Sistemi



Şekil 1.3. Önerilen sistem mimarisi

MSTOS, oyuncu ve firma arasında ödeme işlemlerinde aracılık yapmaktadır. Oyuncu tanımadığı oyun firmasına, kredi kartı bilgisi gibi hassas bilgilerini vermek istemeyebilir. Firmanın sunacağı ödeme yöntemleri sınırlıysa oyuncu açısından itici olabilir. Örnek olarak, oyuncu kredi kartı sahibi olmayabilir, sonuç olarak alışverişinden vazgeçebilir. Şekil 1.3'de gösterildiği gibi MSTOS, bu noktada çok sayıda ödeme yöntemini tek çatı altında birleştirmektedir. Oyuncular için çok sayıda oyun firmasına güvenilir ortak bir sistem ile ödeme imkanı sunar. Firma açısından ödeme mekanizmasını MSTOS'ne devretmesi ile bu konuya iş gücü ve zaman ayırmasına gerek kalmaz. Oyuncu ile oyun firması arasında sadece oyun ilişkisi

kalmış olur. Firma, artan iş gücünü oyun içeriğinin zenginleştirilmesine harcar ve MSTOS ile ani artan oyuncu sayısında kayıpsız ve esnek ödeme alabilir.

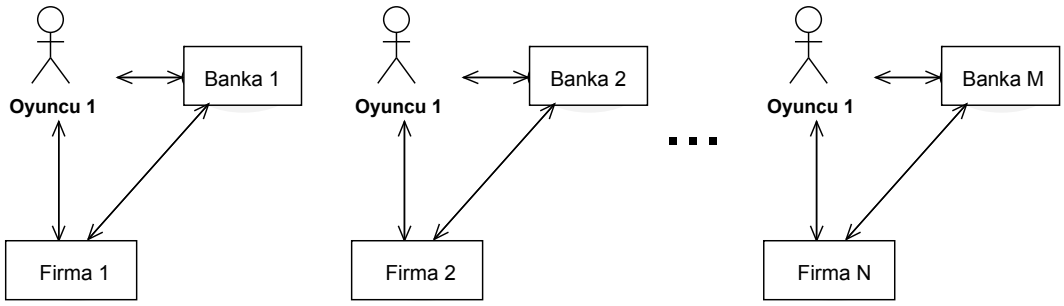
Aşağıda MSTOS sisteminin oyuncu ve firmaya sağladığı faydalar listelenmiştir.

Oyuncunun kazançları:

1. Firmalar ile sadece oyun, puan ve ödül ilişkisi kurması
2. Farklı firmalar olsa da tek ödeme kanalı kullanılabilmesi
3. Çok sayıda oyun için tek jeton sistemi
4. Avans olarak negatif jeton bakiyesi sunulması
5. Oyuncuların kendi aralarında jeton transferi yapabilmesi

Oyun firmasının kazançları:

1. Oyun içi hızlı ödeme sistemi
2. Mali kolaylıklar
3. Ödeme sistemi yazılımı geliştirmesine gerek kalmaması
4. Hızlı gelişen oyuncu sayısının hızlı uyum sağlayan alt yapı ile karşılanması
5. Basit / Ayrıntılı ödeme sistemi kolaylığı



Şekil 1.4. Tek oyuncu – çoklu oyun/banka sistemi

MSTOS'nin diğer ödeme sistemlerine karşı avantajı, oyuncuların hesabındaki jetonların tüm üye firmaların oyunlarında geçerli olmasıdır. Bu sayede oyuncu, MSTOS üzerindeki jetonunu, bir oyundan farklı bir oyuna transfer edebilir. Şekil 1.4'de gözüktüğü gibi oyun firmaları farklı bankaların çeşitli ödeme yöntemlerini kullanabilir. Bu yöntemler kredi kartı, havale veya diğer çevrim-içi ödeme sistemleri üzerinden olabilir. Oyuncular, tüm firmalardan alışveriş yapabilmek için gerekli tüm ödeme yöntemlerine sahip olmayabilir. Çeşitli ödeme yöntemleri, MSTOS'nin bünyesine alternatif ödeme yöntemi olarak dahil edilmesi hedeflenmiştir. Bu sayede oyuncu ve firmalar arasındaki ödeme yönteminde uzlaşma problemine çözüm getirecektir. Diğer sunduğu bir hizmeti ise oyunculara, Oyuncunun ödeme geçmişine veya firmanın üyelik türüne bağlı olarak belirlenen kriterlere uyması durumunda avans olarak negatif jeton bakiyesi vermesidir. Oyuncunun jeton bilgileri MSTOS üyeliğinde saklanmaktadır. Oyuncular, belirlenen limitler çerçevesinde jetonlarını kullandıktan sonra ödemelerini yapabilirler. Bakiyesi olan oyuncular, istedikleri herhangi bir oyuncuya jetonlarını transfer edebilir. Ayrıca oyuncu, üye firmalardan birine ait başka bir oyuna, sahip olduğu jetonları aktarabilir. Bu sayede oyuncu bağlılığı oluşturulması hedeflenmektedir.

1.1 Önceki Çalışmalar başlığında bahsedilen Steam, Xsolla ve GameSparks işleyiş olarak MSTOS'ne yakın gözükmektedirler. Fakat MSTOS, amacı ve hedef kitlesi ile bu sistemlerden farklılaşmaktadır. MSTOS'nin avans jeton, oyun duraksamadan ödeme yapılabilmesi, hızlı kurulum ve kullanım gibi başlıca farklılıkları vardır. Ücretsiz hizmet veren bir oyunun, ödeme almak istemesi durumunda gelir

sağlayabilmesi için dinamik bir altyapının tasarlanması ve gerçeklenmesidir. Diğer Google Play Game ve Apple App Store Games gibi oyun marketlerinde sadece oyunun satışı ve oyun içi sabit ödemelerin alınması konusunda oyun firmalarına destek vermektedirler [6] [8]. Ek olarak sadece mobil oyunlar için kullanılabilmektedir. Bu konuda oyun firmaları için özelleşmiş bir ödeme sistemi sunamamaktadırlar. Facebook Games ise sadece Facebook üyelerine hizmet verilmesini zorunlu tutmaktadır. Bu kısıtlama Facebook hesabı olmayan oyuncuların kaçırılması anlamına gelmektedir [18].

Firma ve oyuncu arasındaki alışverişte ödemeyi MSTOS alır. MSTOS aldığı ödemeye karşılık firmaya onay verir. Firma satışı tamamlar ve üye satın aldığı eşya veya puanı kullanabilir. MSTOS firma adına aldığı ödemeler içerisinden yüzde olarak komisyonunu alır. Dönem sonunda firma adına topladığı ödemeleri, firmaya aktarır. Firmanın ödemeyi doğrudan oyunculardan alması durumunda mali bağlantısı, tekten çoğa şeklindedir. Kanunen kesmesi gereken fatura sayısı çok sayıda olacaktır. Firmanın MSTOS'dan hizmet alması durumunda ise mali bağlantısı tekten teke şekline iner.

MSTOS çekirdeği, üyelik ve ödeme servisi olarak iki ayrı parçadan oluşmaktadır. MSTOS'nin gerçeklenmesi için ihtiyaç duyulan iki adet yan uygulama olarak da banka ve e-posta gönderim servisleri oluşturulmuştur. Uygulamada bankacılık işlemleri için oyuncunun kartından ödeme alınmasını simüle eden bir sanal pos servisi hazırlanmıştır. E-posta gönderim servisi ise oyuncuya gönderilecek olan 3D PIN'i

gibi e-posta içeriklerinin gönderimini yapar. Bunun dışında oyuncular MSTOS ile doğrudan sistem üzerinden haberleşirler. Oyuncu ve firmanın MSTOS üyeliklerini kontrol edebildikleri web tabanlı MSTOS paneli bulunmaktadır.

Tez ile ilgili özet bilgiler, "Giriş" bölümünde sunulmuştu. MSTOS'nin gerçeklenmesi süresince kullanılan teknolojiler ve standart ile ilgili ön bilgiler 2. bölümde verilmiştir. MSTOS'nin tasarımı, kullanıcı türleri, kullanım senaryosu ve gelir paylaşım modeli ile ilgili bilgiler 3. "Mikroservis Tabanlı Ödeme Sistemi Tasarımı" bölümünde sunulmuştur. MSTOS'ini oluşturan uygulamaların gerçeklenmesi süresince izlenen adımlar 4. "Mikroservis Tabanlı Ödeme Sistemi Gerçeklenmesi" bölümünde anlatılmıştır. MSTOS'nin sunduğu altyapıyı kullanan örnek bir oyun uygulaması 5. "Örnek Bir Uygulama" bölümünde anlatılmıştır. Tez ile ilgili genel değerlendirmeler ve öneriler ise 6. "Sonuçlar" bölümünde altında yapılmıştır.

2. KULLANILAN TEKNOLOJİLER

MSTOS'nin gerçeklenmesi esnasında kullanılan teknolojiler ile ilgili bilgiler, bu bölümde anlatılmıştır. Genel bilinirliği olan ve uzun yıllardır kullanılan teknolojilerin adlarına yer verilmesi yeterli görülmüştür. Güncel teknolojiler ise ilerleyen paragraflarda açıklanmıştır. Bu çalışmada, Web [19] tabanlı teknolojiler yer almıştır. İletişim protokolü HTTP'dir [20]. Web sayfalarına URL [21] adresleri ile erişilmektedir. Kullanıcı bilgisayarlarındaki HTML [22] standardındaki sayfalar, CSS [23] ile arayüzü düzenlenmekte ve JavaScript [24] ile programlama dili yeteneklerine kavuşmaktadır. Kullanılan bazı teknolojiler Linux [25] işletim sistemi altında çalışmakta, kabuk (shell) [26] ile sistem programlaması yapılmakta ve SSH [27] üzerinden haberleşmektedir. Ayarlar YAML [28] dosyalarında tutulmaktadır. Genelde Linux ve türevi işletim sistemlerinde program derleme Make [29] ile yapılır. Kaynak kod paylaşımının yapıldığı en güçlü sürüm yönetim sistemi Git [30] kullanılmaktadır.

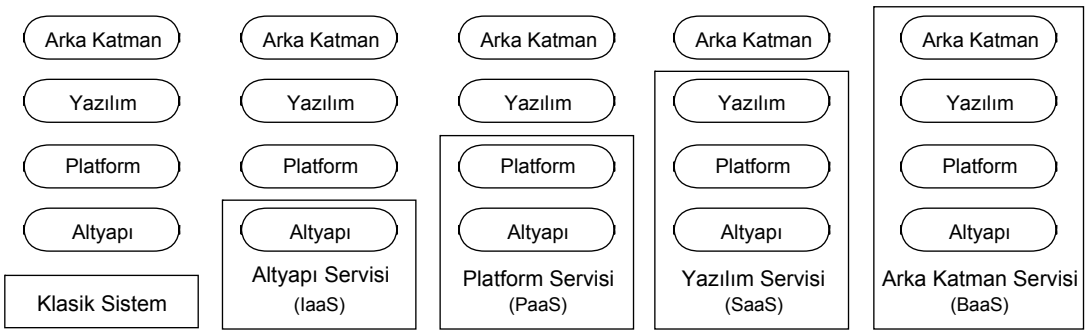
Gerçeklenen MSTOS çekirdeğinin yazılımı, web programlama için kullanılan PHP gibi betik dilleri ve Apache gibi klasik web sunucu yazılımları ile de yapılabilirdi. Fakat ani olarak gerçekleşen son kullanıcı artışlarında klasik mimarinin ölçeklenme işleminde ihtiyaç duyulan; yüksek donanım maliyeti, platformun hazırlanması için tüm yazılımların birebir kurulması, ölçekleme yapılandırmaları gibi problemlerin bulut bilişim üzerinden çözülmesi günümüzde bir zorunluluktur. Eğer bulut bilişimde, 2.1. bölümde bahsedilen servis tiplerinden olan altyapı veya platform servisi kullanılırsa, sadece donanım yönetimi ve işletim sistemi aşaması çözülmüş olur.

Yazılım veya arka katman servisinde ise her servis sağlayıcıya uygun ve farklı programlama yapılması gerekebilmektedir.

Çözüm olarak bu çalışmada kullanılan işletim sistemi seviyesinde sanallaştırma yöntemi, yazılımların özgün ve esnek olarak dağıtım ve ölçeklendirilmesini sağlamaktadır. Bu sanallaştırma teknolojilerinin herhangi bir işlemi yalıtılmış olarak çalıştıran parçasına konteyner denilmektedir. Konteynerlerin oluşturulması, yapılandırılması, çalıştırılması, birbirleriyle iletişimi ve orkestrasyonu gibi ihtiyaçlar için çok sayıda yazılım vardır. Bu yazılımlar içerisinden yüksek performans, kullanıcı desteği ve yaygınlığa sahip olan özgür yazılımlar arasından 2.2.2.1. bölümde Docker yazılımı ve 2.2.2.2. bölümde Rancher konteyner yönetim yazılımı anlatılmıştır. Oluşturulacak çok sayıda konteyner arasında yük dağılımının yapılabilmesi için 2.2.3. bölümdeki HAProxy yük dengeleyicisi kullanılmıştır. Bahsedilen işletim sistemi seviyesinde sanallaştırma için olabildiğince az kaynak kullanılması amacı ile en hafif web uygulamalarının yazılabileceği diller incelenmiş ve 2.3.4. bölümdeki Go programlama dili seçilmiştir. Asenkron yazılımın çalışabilmesi için 2.3.5. bölümde bahsedilen NSQ mesaj kuyruğu servisi kullanılmıştır. Tasarlanan ve gerçeklenen mimarinin performans testi için gerçek hayattaki kullanıcılar üzerinde denemeler yapılıp bilgi toplanamayacağı için sanal olarak bu yükler 2.2.4. bölümde anlatılan Gatling yük testi aracı ile oluşturulmuştur. Son günlerde popülerleşen ve çok sayıda araç ile yaygın bir kullanıma sahip olan 2.5. bölümde detaylıca bahsedilen Javascript ve diğer bileşenlerinden olabildiğince faydalanılmıştır. Hazırlanan uygulama programlama arayüzü için kullanılan REST arayüz mimarisi 2.6.2. bölümde, arayüz

dökümantasyonu için kullanılan Swagger 2.6.2. bölümde, arayüz ağ geçidi olarak kullanılan Kong 2.6.4. bölümde anlatılmıştır. Halka topolojisine benzer çalışan bağlantı yapısı ile hızlıca ölçeklenebilen Apache Cassandra veritabanı ise 2.7.1. bölümde tanıtılmıştır. MSTOS için 2.1.1'de bahsedilen ve hizmet olarak alınan Altyapı servisi, gerçeklenme sonucunda 2.1.4. Arka katman servisi olarak firmalara sunulmuştur. Bahsedilen teknolojilerin seçim nedenleri, alternatifleri ve uygulama mimarisindeki yerleri için tüm açıklamalar 4. bölümde yer almaktadır. Bu bölümde sadece teknolojiler hakkında temel bilgiler verilmiştir.

2.1. Bulut Bilişim



Şekil 2.1. Bulut bilişim ve servis tipleri

Bulut bilişim (cloud computing), ihtiyaç halinde, herhangi bir yerde ve zamanda, belirli bilgisayar kaynaklarının oluşturduğu paylaşım havuzundan ağ, sunucu, depolama, uygulama ve uygulama servislerinin minimum etkileşim veya yönetim iş gücü ile hızlıca kullanılabilmesi ve gereksinim fazlası kaynağın havuza geri iade edilebilmesini sağlayan teknolojidir [31]. Bulut bilişim sağlayıcılarının sunduğu

servislerde dört temel model şunlardır: altyapı servisi (IaaS), platform servisi (PaaS), yazılım servisi (SaaS) ve arka katman servisi (BaaS). Bu servis tipleri, servis kapsamlarına göre Şekil 2.1 üzerinde gösterilmiştir.

2.1.1. Altyapı servisi

Altyapı servisi (IaaS – Infrastructure as a Service) alan müşteriler, uygulamaları çalıştırabilmek için makinelere öncelikle işletim sistemini, ardından ise kullanacakları uygulama yazılımlarını kendileri yüklemelidirler. Bu modelde yüklenen, işletim sistemi de dahil olmak üzere tüm yazılımlarının onarımından, problemlerinden müşteri sorumludur [32].

2.1.2. Platform servisi

Platform servisinde (PaaS - Platform as a Service), bulut sağlayıcıları işletim sistemi, program çalıştırma ortamı, veri tabanı ve web sunucularını içeren kullanıma hazır bir platform sunmaktadırlar. Müşteri, işletim sistemi ve alt seviye yazılımsal sorunlarla uğraşmamaktadır. Kullanmak istediği yazılımını hızlıca çalıştırabileceği ortama sahip olabilmektedir [32].

2.1.3. Yazılım servisi

Yazılım servisinde (SaaS - Software as a Service) ise, bulut sağlayıcıları, uygulama yazılımını yüklemeyi ve çalışabilirliğinin devamını sağlar. Bakım ve destek de, sağlayıcı tarafından yapılmaktadır. Müşteri ise herhangi bir altyapı veya platformu yönetmesine gerek kalmadan sadece kullanmak istediği uygulama yazılımına genellikle web platformu ile erişir ve kolayca kullanım sağlar. Bu sayede kullanıcılar,

eski nesil uygulamalardaki, kurulu olma ve çalıştırılma zaruretinden kurtulmuş olurlar. Bulut uygulamasını eski nesil uygulamalara göre avantajı esnekliğidir. Artan iş taleplerini karşılamak için çoklu sanal makineler oluşturulabilir ve görev paylaşımı sağlanabilir [32].

2.1.4. Arka katman servisi

Arka katman servisi (BaaS - Backend as a Service) veya diğer bir adı ile mobil arka katman servisi (Mobile Backend as a Service – mBaaS), web uygulaması ve mobil uygulama geliştiricilerine uygulamalarını, kullanıcı yönetimi, anlık bildirimler ve sosyal paylaşım servisleri ile entegrasyon gibi arka katman işlemleri için ihtiyaç duydukları, bulut depolamada alan ve uygulama geçiş arayüzüne bağlayabilmelerini sağlayan modeldir. Bu servisler, özelleştirilmiş uygulama geliştirme kitleri ve programlama arayüzleri sayesinde sunulmaktadır [33].

Yukarıda da bahsedildiği gibi web ve mobil uygulamalarda, arka katmanda benzer özellikler gerekebilmektedir. Bu model sayesinde geliştirilen arka katman yazılımı, iki bağımsız kullanıcı arayüzü için de ortak havuzdan kullanılabilmektedir. Bu da kullanılmayan arka katman servisinin, kullanımın yoğun olduğu kullanıcı arayüzüne kolayca aktarılması anlamına gelmektedir. Ek olarak uygulamanın istenilen platformda kullanıcılara sunulabilmesi için sadece platforma özel kullanıcı arayüzünün kodlanması yeterli olmaktadır. Çoğu BaaS girişimlerinin 2011 ve sonrasında ortaya çıktığı düşünülürse oldukça yeni bir modeldir [34].

2.2. Altyapı ve Yönetimi

MSTOS ve tez içerisinde tasarlanan diğer uygulamaların, altyapılarının oluşturulması için yapılan incelemelerde hız, çeviklik ve açık kaynak kodlu yazılımlara öncelik verilmiştir. İşletim sistemi olarak Linux tabanlı Ubuntu dağıtımı tercih edilmiştir. Ubuntu, sunucu ve kişisel bilgisayarlarda ücretsiz kullanılabilmektedir. Kolay kurulum ve kullanımı sebebiyle yeni başlayanlar tarafından çoğunlukla tercih edilen Linux dağıtımıdır [35]. Linux, çalıştırdığı işlemleri izole edilebilmesi, sınırlandırılabilmesi ve yetkilendirebilmesi gibi işletim sistemi seviyesinde sanallaştırma için sunduğu özellikler sayesinde 2.2.2.1. bölümde anlatılan Docker yazılımı gibi hızlı sanallaştırma projelerinin hayata geçirilmesine imkan vermektedir. Tez içerisinde, bu hızlı sanallaştırma ve ölçeklendirme araçları kullanılmış ve yönetimi yapılmıştır.

2.2.1. Geliştirme ve yayına alma

Geliştirme ve yayına alma (DevOps), altyapı değişimi veya yazılım geliştirme sonucu değişikliklerin yayına alınmasını, yazılım geliştirici ve sistem yöneticileri tarafından işbirliği ve iletişim içerisinde otomatize edilmesini tanımlamak için kullanılır [36]. Derleme, test etme ve yayına alma işlemlerinin hızlıca, sıklıkla ve güvenilir olmasını hedeflemektedir [37]. Manuel yapılan yayına alma işlemlerinde kontrolün insanlarda olması hataya ve ihmale imkan vermektedir. Belirli prosedür ve kalıplar uygulanarak yayına alma işlemi yapılması insan faktörünü devreden çıkarmaya imkan vermektedir. DevOps'un ortaya çıkmasını da, bu gelişmeler sebep olmuştur.

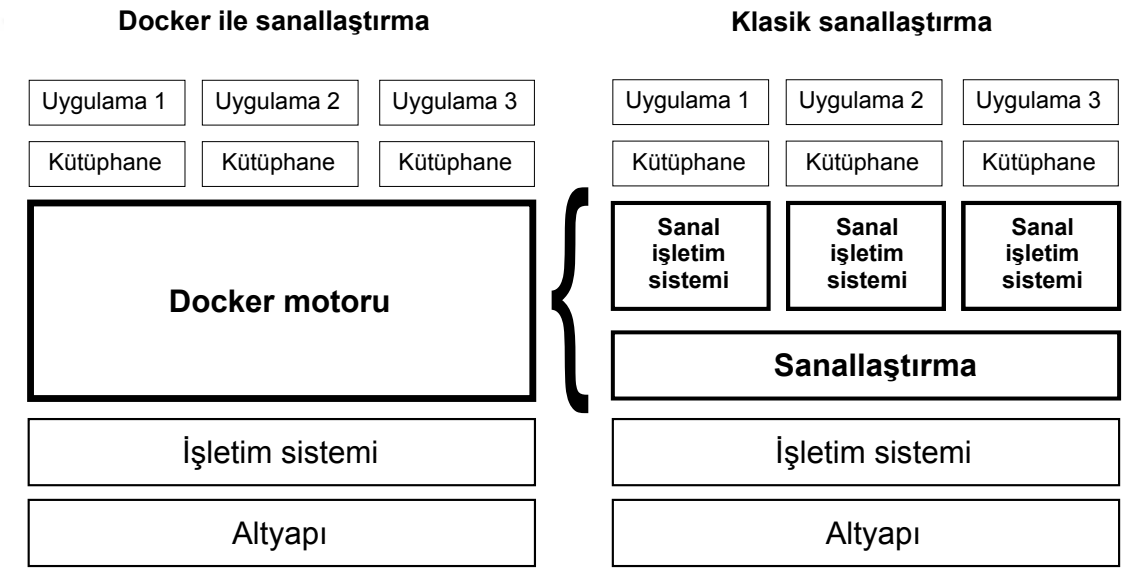
2.2.2. İşletim sistemi seviyesinde sanallaştırma

İşletim sistemi seviyesinde sanallaştırma (LXC – Linux Containers), Linux çekirdeğini kullanan bir sunucuda, birden fazla yazılımı yalıtılmış olarak çalıştırmayı sağlayan sanallaştırma teknolojisidir. Linux, cgroups (işlem kontrol grupları), namespaces (tanım uzayları) isimleri verilen özellikleri sayesinde koşturduğu programlara işlemci, bellek ve ağ bağlantısı gibi kaynakları engellenmeyi, sınırlandırılmayı veya belirli kurallara göre paylaştırmayı yapabilmektedir [38]. Yönetim, işletim sistemi çekirdeği (Linux) tarafından yapıldığı için herhangi bir sanal makineye ihtiyaç duyulmaz. Bu özelliğin seçilen programa uygulanması, işletim sistemi parametrelerinin (environment) özelleştirilmesi ve kullanılan dosya sisteminin sınırlandırılması gibi ek işlemler ile sanallaştırma yapılabilmektedir. Çalışan uygulamanın istekleri kurgulanan işletim sistemi tarafından yapay ve kontrollü olarak gerçekleştirilmektedir. Bu sebeplerle de tam anlamı ile işletim sistemi seviyesinde yalıtım, güvenlik ve gizlilik sağlanabilmektedir [39]. İşletim sistemi seviyesinde sanallaştırmayı basit ve etkin kullanmayı sağlayan araçlardan birisi de Docker'dır [40].

2.2.2.1. Docker yazılımı

Docker, yazılımın ve ihtiyaç duyduğu tüm yazılımsal bağımlılıkların, kalıbının çıkartılması ve çalıştırma anında kalıptaki tüm içeriklerin onu taşıyan bir konteyner ile herhangi bir sunucu üzerinde çalışabilmesini sağlayan bir özgür yazılım projesidir. Konteynere çevrilen yazılımın, her koşul ve sistemde aynı, güvenli ve stabil çalışabilmesini garanti eder. Linux, Windows ve OS X işletim sistemleri üzerinde

çalışabilmektedir. Şekil 2.2'de gözüktüğü üzere sanal makineler gibi tüm işletim sistemi paketlenemediği için sunucu işletim sistemi ile ortak sistem kütüphaneleri kullanılır, bu sebeple çok az yer kaplamaktadır. Sanallaştırma için fazladan işletim sistemi çalıştırılmadığı için sanal makinelere kıyasla çok yüksek performans ve açılış hızı sunmaktadır [41].



Şekil 2.2. Docker ve klasik sanallaştırma karşılaştırması

Tüm bunlar, yatay ölçeklenebilirlik için gereken yeni sunucu ekleme veya çıkarma işlemlerini hızlıca ve basit olarak yapılmasını sağlamaktadır [42]. Docker'ın kullanımı ve yönetimi ile ilgili diğer araçlar şunlardır;

Docker Compose; birden fazla Docker konteynerini aynı anda tanımlama ve çalıştırmak için kullanılan araçtır. Compose dosyaları ile uygulama için gerekli servisler YAML formatında tanımlanıp, yapılandırma ayarları ile

kaydedilebilmektedir. Gerektiği anda tek bir komut ile tanım dosyasına göre sistemler oluşturulabilmektedir [43].

Docker Machine; istenilen sunucuya, Docker ve servislerinin kurulmasını, komut satırı aracılığıyla platformdan bağımsız olarak Docker konteynerlerinin yönetilmesi sağlamaktadır. Docker Machine, uzaktaki sunucularda çalışan Docker servislerine, yerelde çalışıyormuş gibi erişimi mümkün kılmaktadır. Docker, Linux tabanlı işletim sistemlerinde yüksek verimlilik göstermektedir. Docker Machine sayesinde, düşük kapasiteli ve Linux harici işletim sistemine sahip bilgisayarlar istemci olarak çalışabilmektedir. Docker servisi çalıştıran Linux sunucularına protokol veya platform farkından dolayı oluşan dezavantajlara maruz kalmadan erişebilmektedir [44].

Docker Hub; kalıp dosyalarının oluşturulması, test edilmesi, depolanması ve kullanılacak olunan sunuculara gönderilmesi için bulut bilişim alt yapısı ile oluşturulan çevrim içi kütüphanedir. Kalıp dosyalarının konteynerler için keşfedilmesi, dağıtılması ve kalıptaki güncellemelerin takip edilemesi için merkezi olarak hizmet vermektedir. Takım çalışması ve otomatik yayına alma işlemleri içinde kullanılmaktadır [45].

Swarm; Docker'ın, orkestrasyon -konteynerlerin ihtiyaca göre kombin edilmesi- işlemi için hazırladığı resmi yazılım projesidir. Ölçeklendirme, yük dengeleme, ağ ve servis tespiti gibi yönetim problemlerine yeni çözümler getirmiştir. Swarm, konteyner isteklerini, oluşturduğu görev havuzunda tutmaktadır. Docker sunucuları tek bir

komut ile Swarm'ın havuzuna katılıp, konteynerleri çalıştırabilmektedir [46]. Kubernetes ve Apache Mesos gibi diğer orkestrasyon araçları ile karşılaştırıldığında Swarm daha sade ve verimli gözükmektedir. Küçük ekipler ve yüksek sayıda konteyner yönetimi için daha idealdir [47].

2.2.2.2. Rancher konteyner yönetim yazılımı

Rancher, konteynerlerin canlı sistemlerde yayına alınmasını ve yönetilmesi sağlayan açık kaynaklı platformdur. Önceki bölümlerde Docker konteynerlerini kullanabilmek için farklı görevlere sahip önemli araçlardan bahsedilmiştir. Rancher ise bahsedilen araçların bütün yeteneklerini kendisinde toplamayı hedeflemektedir. Yönetim için oluşturulan görsel arayüzü web altyapısı ile çalışmaktadır. Kurulum sonrasında arayüze erişim için web tarayıcısı gerekmektedir. Rancher, Şekil 2.4'te görülen Docker Swarm ve muadilleri olan MESOS, Kubernetes gibi orkestrasyon araçları ile uyumlu çalışmaktadır. Amazon Web Services, Digital Ocean veya Microsoft Azure gibi bulut bilişim servis sağlayıcılarında veya sanal olmayan sunucularda, konteynerleri yönetebilmektedir. Sahip olduğu yazılım kataloğu, tek tık ile çalıştırılabilen konteyner paketlerinden oluşmaktadır. Topluluk tarafından düzenli olarak eklemeler ve güncellemeler yapılmaktadır [48].

2.2.3. HAProxy yük dengeleyicisi

HAProxy, TCP ve HTTP tabanlı uygulamalar için yüksek erişilebilirlik, yük dengeleme ve vekil sunucu oluşturmayı sağlayan ücretsiz, çok hızlı ve güvenilir bir çözümdür. Linux, Solaris ve BSD platformlarında çalışabilmektedir. 3 sene boyunca

kesintisiz açık kalan ve işletim sisteminin radikal güncelleme yamasından dolayı mecburi olarak yeniden başlatılması gereken bir sunucuda hatasız olarak günlük milyonlarca sayfa gösterimi için çalıştırılmıştır. Dünyanın en çok ziyaret edilen çoğu web sitesi tarafından da kullanılmaktadır [49].

2.2.4. Gatling yük testi aracı

Gatling, kullanımı kolay ve yüksek performanslı bir yük testi aracıdır. Yüksek seviyede HTTP protokolü desteğine sahip olduğu için herhangi bir HTTP sunucusunun yük testi için öncelikli olarak tercih edilebilmektedir. Oluşturulan senaryo dosyalarına göre test adımlarını gerçekleştirmektedir. Senaryo dosyaları, bakımı ve anlaşılması kolay olması için özel olarak tasarlanan yazım kurallarına (domain-specific language) göre oluşturulmaktadır. Gatling mimarisi gereği asenkron olarak çalışabilmektedir. Bu sayede düşük sistem kaynakları ile eş zamanlı olarak çok sayıda sanal kullanıcı simüle edebilmektedir [50].

2.3. Projede Kullanılan Programlama Teknikleri ve Teknolojileri

MSTOS ve diğer uygulamaların programlanmasında kullanılan ve tez içerisinde bahsedilen programlama dili, araçları ve önemli terimleri aşağıdaki bölümlerde açıklanmıştır. Tercih edilme sebeplerinden de bahsedilmiştir.

2.3.1. Eş zamanlı çalışma

Eş zamanlı çalışma (concurrency), bilgisayar bilimlerinde bir programın veya algoritmanın tamamının veya bir kısmının parçalara ayrılıp, aynı anda ya da sıradan

bağımsız olarak çalıştırılabilme yeteneğine sahip olmasıdır [51]. Eş zamanlı bir sistemdeki işlemler, yürütülürken birbirleriyle etkileşim kurabileceğinden, sistemdeki olası yürütme yollarının sayısı çok büyük olabilir ve sonuçta belirsiz olabilir. Paylaşılan kaynakların eş zamanlı kullanımı, kilitlenmeler ve ek kaynak ihtiyaçları gibi sorunlar gereken kaynak tespitini zorlaştırabilir [52].

Genellikle "eş zamanlı çalışma" denildiği zaman akıllara "paralel çalışma" gelebiliyor, fakat ikisi birbirinden tamamen farklı iki kavramdır. Programlamada, eş zamanlılık bağımsız olarak yürütülen işlemlerin bileşimidir; paralellik bir işlemin aynı anda birden fazla kolda yürütülmesidir. Eş zamanlılık aynı anda birçok şeyle uğraşmakla ilgilidir. Paralellik, bir şeyi aynı anda çok defa yapmakla ilgilidir [53].

2.3.2. Eş zamanlı programlama

Eş zamanlı programlama, algoritmadaki işlem parçalarının ardışık olarak işletilmesi yerine aynı zaman aralığında çalıştırılmasıdır. İşlem parçacıklarına ayrılan algoritmanın, birden fazla başlatma noktası veya her parçacık için farklı kontrol mekanizması olması sistemin bir özelliğidir. Eş zamanlı programlama, diğer işlemlerin tamamlanmasını beklemeden herhangi bir işlemin ilerleyebileceği bir sistemdir [54]

2.3.3. Asenkron çalışma

Asenkron çalışma (asynchrony) bilgisayar programlamada, ana program akışından bağımsız olarak işlemlerin çalıştırılmasını ve bu işlemlerin karşılaşılabilinecek

problemleri ile başa çıkmanın yollarını ifade eder. Asenkron işlemler, eş zamanlı olarak başka bir program tarafından tetiklenen ve tamamlanması beklenilmeyen eylemler veya gönderilen sinyaller olabilir. Örnek olarak; asenkron giriş/çıkış işlemlerinde, işlemcinin herhangi bir programı çalıştırırken aynı anda depolama ve ağ cihazlarına komutlar göndermesi veya onlardan gelen istekleri yerine getirmesi düşünülebilir. Asenkron programlama bu yönüyle kısmi olarak paralel programlama ile de benzerlik gösterebilir [55].

2.3.4. Go programlama dili

Go programlama dili veya Golang, Google tarafından 2009'da özgür yazılım lisansı ile duyrulan sistem programlama dilidir. Go kodları, platformdan bağımsız olarak Linux, OSX, Windows veya çeşitli UNIX ve BSD dağıtımları için derlenebilir veya platforma kurulan yorumlayıcı ile çalıştırılabilir. C ve Algol türevi yazım kuralları bulunmaktadır. Hafıza koruması (memory safety) ve artık bellek denetleyicisine (garbage collection) sahiptir. Tam anlamı ile nesnel tabanlı programlama dili değildir. Hantal olmamak için ihtiyaç duyulan özelliklere destek sunulmaktadır. Eş zamanlı programlamanın problem ve zorluklarına çok sayıda güncel çözümler sunmaktadır. Tony Hoare'nin eş zamanlı sistemlerde etkileşim kalıplarını açıklayan "Communicating sequential process" modeli temel alınmıştır. Goroutine ismi verilen özelliği sayesinde tanımlı herhangi bir fonksiyon sadece "go" komutu ile eş zamanlı olarak çalışabilmektedir. Eş zamanlı çalışan işlemler "channel" ismi verilen sistem sayesinde kolayca haberleşmektedir. Test, hata ayıklama ve kod inceleme için çok sayıda araca sahiptir. Çevrimiçi çalışan paket yöneticisi ile yazılan kaynak kodlarının

kütüphane oluşturmak için paketlenmesini, dağıtılmasını ve açık olarak isteyen herkes tarafından kullanılmasını kolaylaştırmıştır [56].

2.3.5. Mesaj kuyruğu servisi

Mesaj kuyruğu servisi, uygulamalar ve diğer servisler arasında, güvenilir şekilde birbirlerine mesajlar gönderebilmesi ile asenkron olarak etkileşime geçebilmelerini sağlayan servistir. Servisler arasındaki iletilerin taşınmasının kolaylaştırmaktadır. Erişim için bir protokol veya arayüz sağlar. Bu arayüz, mesajlar oluşturan üreticileri ve onları işleyen tüketicileri birbirine asenkron olarak bağlar. Mesaj kuyruğu servisi için açık kaynak olarak Go programlama dili ile geliştirilen NSQ tercih edilmiştir.

Mesaj kuyruğu servisinin önemini anlatabilmek için örnek olarak standart bir web uygulamasını düşünecek olursak, mesaj üretici, kullanıcıdan gelen isteklere (yeni üye kayıdı gibi) göre yeni mesajlar oluşturur. Artık web uygulamasının görevi bitmiştir. Bir sonraki web isteğini karşılayabilir. Mesaj tüketiciler, ayrı bir işlem olarak arka planda sürekli çalışmaktadır. Belirli zamanlarda gelen mesaj kutusunu tarar ve içeriğine göre işlemleri yapmaktadır. Diğer türlü, kullanıcının tüm işlemler tamamlanana kadar beklemesi gerekmektedir [57].

NSQ, dağıtık mimari ile çok sayıda mesaj kuyruğunu yüksek trafiğe rağmen taşıyabilmeyi hedefleyen popüler yazılım projesidir. Üretici ve tüketici servisler için basit kullanım arayüzleri sunmaktadır. Dağıtık yapısı sayesinde veri tutarlılığını ve performansı garanti etmektedir. Dağıtık sistemlerde merkezi bir bileşenin çalışamaz

hale gelmesi (Single point of failure) ile tüm sistemin durması söz konusu olabilmektedir. NSQ, bu probleme çözüm olarak, mesaj kuyruğu servislerinde aracılık (broker) yapan bileşeni kaldırıp, tüm tüketicilerin tüm üreticilere bağlantı kurabilmesi üzerine mimarisini kurmuştur. NSQ'nin geliştirme ekibi, projenin yazımında Go dilini kullanmıştır. Ekip, Go dili ile performans isteği ve mimarinin oluşturduğu alt seviye teknik problemleri (eş zamanlılık gibi), kolayca çözebildiklerini belirtmektedir [58].

2.4. Web Kullanıcı Arayüzü

Web kullanıcı arayüzü, web altyapısı ile indirilen metinsel ve görsel içeriklerin, belirli bir şablona göre sunulmasını, rahat ve düzenli kullanımı sağlayan teknolojiler bütünüdür. Geliştirmeleri hala devam eden belirli standartlar ile oluşturulmuştur. Bu standartlardan en yaygın olanları HTML, CSS ve JavaScript'tir. Web tarayıcıları, kullanıcı arayüzünü oluştururken, bu standartlara göre çalışmaktadırlar [59]. Web kullanıcı arayüzünde; HTML, CSS ve JavaScript'e ek olarak Twitter Bootstrap kullanılmıştır.

Twitter Bootstrap, sık kullanılan CSS tanımlamalarını içeren stil şablonu çatısı olarak Twitter tarafından 2011 yılında duyurulmuştur. Çoğu web tarayıcısı ile uyumlu olarak çalışır ve güncel teknikleri de içermektedir. Web uygulamalarının kullanıcı arayüzlerini hızlıca geliştirmeyi sağlar. Buton, başlık, tablo ve panel gibi bileşenlerin mobil veya tablet gibi her boyuttaki cihaz ekranları için esnek (responsive) ve basit görsel tasarımları içermektedir. Sayfanın ekranlara uyumlu olarak bölümlendirilmesi için de geliştiricilere avantajlar sağlamaktadır [60].

MSTOS panelinin tez içerisindeki işlevi ikinci planda olduğu için gerçeklenmesi ile ilgili performans ve benzeri kaygılar önemsenmemiştir. Tez içerisinde özgür yazılımların kullanımına önem verildiğinden dolayı panelin gerçeklenmesi için yaygın olarak kullanılan web programlama dili PHP tercih edilmiştir. PHP kütüphanesi olarak ise Laravel kullanılmıştır. Laravel, nesne tabanlı programlama paradigmalarına uygunluğu ve basit geliştirmeye imkan vermesi ile son yıllarda hızlıca popüler hale gelen bir kütüphanedir [61]. Panelin hızlı gerçeklenebilmesinde de önemli katkısı olmuştur.

2.5. Javascript ve Bileşenleri

Javascript, tüm tarayıcılar tarafından desteklenen yaygın bir dil haline gelmesi ECMA standardizasyonu sonucu olmuştur. Google'ın Chrome tarayıcısı için Javascript yorumlayıcı olarak geliştirdiği V8 motoru verimsizlik ve istikrarsızlığı çok düşük seviyelere indirmesi dikkatleri üzerine çekmiştir. V8'in başarısı üzerine sunucu, masaüstü, gömülü sistem ve mobil programlama platformlarına JavaScript kullanımı için gönüllü ekipler tarafından çalışmalar yapılmıştır. Tüm platformlar üzerinde aynı yazım kuralına sahip olması (isomorphic), geliştiriciler için platform geçişlerinde kolaylık sağlamıştır. Web kullanıcı arayüzü geliştiren bir programcı, NodeJS ile sunucu uygulamasını da geliştirebilmiştir. Doğal olarak geliştiriciler arasında 2016 yılında en çok tercih edilen programlama dili olmuştur. Gelişimine paralel olarak aktarıldığı platformlar içinde önemli kütüphane ve sistemlerin gelişmesine sebep

olmuş veya altyapısını oluşturmuştur [62]. Alt bölümlerde JavaScript dünyasını anlamak için bilinmesi gereken en önemlilerinden bahsedilmektedir.

2.5.1. Ajax yazılım teknolojisi

Web uygulamalarının, mevcut sayfanın görüntüsünü ve yapısını bozmadan, sunucuya asenkron olarak (arka planda) veri transferini sağlayan sistemdir. Ajax, verilerin görünüm katmanından ayrılmasını sağlamaktadır. Sayfaların içeriklerini dinamik olarak görünümlerine uygulamalarını mümkün kılmaktadır. Bu sayede veri trafiği azaltılmış ve içeriğe ulaşım da hızlandırılmış olmaktadır. Veri transfer şablonu olarak JSON ve XML kullanılabilmektedir [63].

2.5.2. JSON veri formatı

JSON basit, metin tabanlı, bağımsız bir veri transfer formatıdır. Yapılandırılmış verilerin taşınabilirliği için basit yapılandırma kuralları tanımlamaktadır. Tanımlamalar için JavaScript programlama dilini temel almaktadır. JSON, kullanılacağı yazılım dilinden tamamen bağımsızdır ancak; C++, C#, Java, JavaScript gibi C ve türevi programlama dillerini bilen kişilere yazım kuralları aşina gelmektedir. İnsanların okuyabileceği kadar düzenli gözükmektedir. Makinalar tarafından yorumlanması, oluşturulması da hızlı ve kolaydır [64].

2.5.3. JWT standardı

JWT (JSON Web Token), bir JSON nesnesi ile servisler arasında güvenli bilgi aktarımı için çevik bir format tanımlayan açık bir standarttır. Taşınan bilgi, dijital olarak şifrelenmiş olduğundan doğrulanabilir ve güvenilir olmaktadır. JWT'ler, gizli

HMAC veya RSA anahtar çifti ile şifrelenmektedir. Şifreleme teyit edilebildiğinden dolayı değişime ve suistimale kapalı hale gelmektedir. JWT'ler boyutlarının küçük olmasından dolayı URL, POST parametresi veya bir HTTP başlık bilgisi ile gönderilebilmektedir. Bu sayede kullanıcı hakkında veri tabanına defalarca sorgu yapmaya gerek kalmadan gereken bilgiler sayfalar arasında taşınabilmektedir. Bu sebeple etkin kullanımı ile yüksek performans kazancı elde edilebilmektedir. Genel olarak kullanıcı oturum bilgileri (session) ve web servislere erişimde standart haline gelen, tek kullanımlık erişim biletlerini (token based authentication) taşımak için kullanılmaktadır [65] [66].

2.5.4. Node.js programlama dili

Ryan Dahl tarafından 2009'da icat edilen açık kaynak kodlu Node.js, hızlı ve ölçeklenebilir ağ uygulamaları oluşturmak için Chrome'un JavaScript yorumlama motoru (V8) üzerine kurulmuş bir platformdur. Dağıtık yapıda yüksek trafik altında çalışan sistemler için hafif ve verimli bir şekilde olaya dayalı, engelleyici olmayan bir girdi/çıktı modeli kullanır. Linux, Windows ve OSX üzerinde çalışabilmektedir. Node.js'nin en önem verdiği ağ protokolü HTTP'dir. Bu sebeple web sunucusu yazmak için çeşitli JavaScript modüllerinden oluşan zengin bir kütüphane ve bileşenler sunmaktadır. Yüksek işlemci gücü gerektiren uygulamalarda kullanılması önerilmemektedir [67].

2.5.5. NPM paket yöneticisi

NPM, JavaScript kodlarını paylaşmayı, yeniden kullanmayı ve paylaşılan kodu güncellemeyi sağlayan paket yöneticisidir. Node.js projesi geliştirmek veya geliştirilmiş projede kullanmak için gereken ortamı hazırlayabilmektedir. Paylaşılan paketlerin versiyon ve etiketlerine göre arşivini oluşturmaktadır. Bu sayede, paketin değişim takibine imkan sağlamaktadır. İhtiyaç halinde güncel olmayan projeler için gereken geriye dönük uyumluluk sorununu çözebilmektedir [68].

2.5.6. Webpack modül paketleyicisi

Webpack, açık kaynak kodlu bir JavaScript modülü paketleyicisidir. Ana amacı, Node.js ortamında çalışan JavaScript dosyalarını bağımlı olduğu modüller ile tarayıcıda çalışabilecek şekilde paketlemektir. Farklı formatlardaki dosyalarda da web için dönüştürme veya paketleme yapabilmektedir. Webpack paketleyicisi, komut satırından kullanılabilir veya webpack.config.js adlı bir yapılandırma dosyası kullanılarak yapılandırılabilir [69] [70].

2.6. Uygulama Programlama Arayüzü

Uygulama programlama arayüzü (API), programcıların geliştirdikleri uygulamanın diğer uygulamalar ile etkileşim sağlamalarını kolaylaştırmaktadır. Arayüz servisi verecek olan uygulama, sadece geliştiricinin ihtiyaç duyduğu nesneleri veya eylemleri erişilebilir yapmaktadır. Kendisini soyutlaması sayesinde programcı için karmaşıklığı azaltmaktadır. Bir e-posta istemcisi, görsel kullanıcı arayüzü sayesinde, yeni e-posta göndermek için bir tuşa tıklanması sonucu arka planda yapılması gereken tüm işleri

halletmektedir. Aynı şekilde uygulama programlama arayüzü de yazılım geliştirici içinde arka planda hangi adımlar gerçekleştiğini bilmeden herhangi bir işlemi yapabilmesini sağlayabilmektedir [71].

2.6.1. Web uygulama programlama arayüzü

Web uygulama programlama arayüzü (Web API), internet üzerinden uygulama programlama arayüzlerine erişmek için önemli bir bağlantı mekanizması sağlamaktadır. Arayüze web üzerinden erişim için kullanılan URL'e "Endpoint" denilmektedir. Arayüz oluşturulurken, HTTP altyapısında JSON veri transfer formatı ile çalışan RESTful gibi çeşitli protokoller kullanılmaktadır. Hizmet veren uygulama, izin verdiği düzeyde veri ve yeteneklerini diğer yazılımlara kullanım için sunabilmektedir. Arayüz ile gelişmiş uygulamalardan alınan hizmetler, kullanıcılar için daha düşük maliyetle daha iyi hizmet verebilmektedir. Örnek olarak, belirlenen bir adresin yakınındaki işletmeler, Google Maps API'si ile kolayca listelenmektedir [72].

2.6.2. REST arayüz mimarisi

REST, web üzerindeki kaynaklara erişim için çoğunlukla HTTP protokolü ile kullanılan uygulama programlama arayüzü mimarisidir. Sunucu ve istemci modelinde durum-bağımsız çalışmaktadır. HTTP protokolü ile kullanımı tüm kaynaklara tekil URI tanımlanması ve HTTP komutları ile erişebilmesi anlamına gelmektedir. HTTP komutlarından GET kayıda erişimi, POST yeni kayıt eklemeyi ve DELETE ise kaydı silmeyi temsil etmektedir. REST, web projesinin kullanımını, geliştirilmesini ve

dağıtımını zorunlu olarak basitleştirmektedir. Standartlara uyumlu tasarlanan web uygulama programlama arayüz sistemlerine RESTful denilmektedir [73].

2.6.3. Arayüz dokümantasyonu

Arayüz dokümantasyonu, arayüzün hangi servisleri sunduğu ve bu servislerin nasıl kullanılacağını detaylı olarak tanımlayan dokümanlardan oluşmaktadır. Programcının arayüzü kullanması için bilmesi gereken her şeyi kapsaması beklenmektedir. Eksik dokümantasyon, arayüzün verimsiz veya yanlış kullanılmasına sebep olabilmektedir. Dokümantasyon, arayüz kullanan uygulamaların geliştirilebilmesi ve bakımı için büyük önem taşımaktadır [74]. Tez içerisinde arayüz dokümantasyonu için Swagger tercih edilmiştir.

Swagger, arayüzün yapısı ve özellikleri ile ilgili tanımlamaları OpenAPI standartlarına uygun olarak YAML veya JSON formatında dosyaya çevirmektedir. Oluşturulan Swagger dosyası farklı amaçlar için defalarca kullanılabilmektedir. Arayüzün istemci kodları, çok sayıda programlama dili için taslak seviyesinde otomatik olarak oluşturabilmektedir. HTML tabanında etkileşimli test ekranı sunan arayüz dokümantasyonunu otomatik oluşturabilmektedir. Gerekli yapılandırmaların yapılması durumunda, proje kaynak kodları ve yorum satırlarından Swagger dosyasını da oluşturabilmektedir [75].

2.6.4. Arayüz ağ geçidi

Arayüz ağ geçidi, karmaşık ve çok sayıda alt arayüz sistemlerinin daha basit arayüzlere dönüştürülmesini, kontrol ve takip edilmesini sağlamaktadır. Arayüze gönderilen istekler, ilk olarak arayüz ağ geçidine ulaşmaktadır. Arayüz ağ geçidi, gelen istekleri tanımlamalara göre arayüz servislerine yönlendirmektedir. Arayüze dışarıdan erişimde kimlik doğrulama, istek sınırlandırma, istatistik tutma veya işlem kaydı oluşturma gibi temel gereksinimleri eklemeyi ve merkezi kontrolü sağlamaktadır [76]. Tez içerisinde arayüz ağ geçidi için Kong tercih edilmiştir.

Kong, açık kaynaklı ve ölçeklenebilir bir arayüz ağ geçididir. Kong, herhangi bir RESTful arayüzün girişinde çalışabilir. Platformu oluşturan çekirdek sistem, eklentiler aracılığıyla geliştirilebilir ve özelleştirilebilir olarak tasarlanmıştır. Kong, ilk olarak Mashape firması tarafından 200.000 müşterisinin 15.000 adet arayüz ve mikro servisini kullanması ile yarattığı aylık milyarlarca isteği karşılamak için oluşturulmuştur. Sonraları açık kaynak kodlu olarak ücretsiz kullanım için genel kullanıma sunulmuştur. Bugün Kong, küçük ve büyük organizasyonlarda kritik pozisyonlarda kullanılmaktadır [77].

2.6.5. Mikroservis mimarisi

Mikroservis mimarisi, dağıtılmış yazılım sistemleri oluşturmak için kullanılan servis odaklı mimarilerin, modern bir yorum ile karmaşık uygulamaların birbirleriyle iletişim kurduğu, bağımsız ve küçük uygulamalardan meydana gelen yazılım mimarisidir. Birbirlerine mesaj gönderimi ile etkileşime giren küçük bağımsız

uygulamalara mikroservis denilmektedir. Mikroservis, bağımsız bileşenler ve izole veri tabanları kullanabilmektedir. Servislerin birbirinden ayrılması ile herhangi birinde oluşacak problemin tüm sistemi etkilemesi engellenmektedir. Mikroservis, ihtiyaç durumunda sistemden bağımsız ölçeklenebilmektedir. Bu sayede tüm sistemin, tüm kaynakları tüketmesi yerine yoğunluğun olduğu mikroservise daha fazla kaynak ayırabilmektedir. Mikroservis, ihtiyaç olmadığı durumda tamamen kapatılabilir veya farklı platform ve dilde tasarlanan mikroservis ile değiştirilebilmektedir [78].

2.7. Veritabanı

Veritabanı yönetim sistemleri (DBMS) tarafından belirli yapı ve kurallara bağlı olarak muhafaza edilebilen veri kümelerine veritabanı denilmektedir. Veri tabanlarına yeni kayıtlar eklenebilir, güncellenebilir veya silinebilmektedir. Kullanıcılar oluşturdukları veritabanındaki kayıtları belirli filtrelere göre sorgulayabilmektedir. Çökme veya hata anında veri kaybedilmemesi için kurtarma önlemleri bulunmaktadır. Hataya karşı yedekli çalışabilme (Atomicity), tutarlılık (Consistency), bağımsız çalışabilme (Isolation) ve sağlamlık (Durability) ile açıklanan ACID kriterlerini taşıması gerekmektedir. İlişkisel veritabanı sistemlerinde veri yaratmak, işlemek ve almak için kullanılan yapılandırılmış sorgu dili SQL olarak adlandırılmaktadır. İlişkisel veri tabanları için SQL standart dildir [79]. İlişkisiz, açık kaynaklı, dağıtık çalışabilen ve yatay ölçeklenebilen yeni nesil veri tabanları NoSQL olarak adlandırılmaktadır [80]

2.7.1. Apache Cassandra veritabanı

Cassandra, çok büyük miktarda yapılandırılmış veriyi bir çok sunucuda yüksek erişilebilirlik ile yönetmeyi sağlayan tek düğümün çöküşünden etkilenmeyen (no single point of failure) dağıtık bir veri tabanıdır. Önemli bir Apache projesidir. Amazon'un Dynamo ve Google'ın BigTable veri tabanları örnek alınarak ilk olarak Facebook tarafından geliştirilmiştir. Sürekli erişilebilirlik, ayarlanabilir veri tutarlılığı seviyesi, operasyonel basitlik, çoklu veri merkezi ve bulut bilişim arasında kolay veri dağıtımı gibi ilişkisel ve NoSQL veri tabanlarının basitçe sunamayacağı özelliklere sahiptir. Mimarisi, kolay ve doğrusal ölçeklendirme, yüksek verim ve çalışma süresi hedefi ile oluşturulmuştur. Birincil-ikincil olarak çalışan klasik mimari yerine tüm noktaların eşit olarak çalıştığı halka topolojisini kullanmaktadır. Veritabanında sorgulamalar, Cassandra Query Language (CQL) ismi verilen kendine has sorgulama dili ile yapılmaktadır. Keyspace olarak adlandırılan tabloların şemaları kullanım öncesinde kolon tiplerine göre tanımlanmalıdır. SQL'e benzerliğinden dolayı kolayca öğrenilebilmektedir. Apple, eBay, Instagram, Spotify, Uber, Netflix ve bir çok firma tarafından problemsiz olarak kullanılmaktadır. Son kullanıcı ortamında kullanılan bir veri kümesinde 75.000 adet fazla düğüm ile çalışmaktadır [81].

2.7.2. MariaDB veritabanı

MariaDB, dünyanın en popüler veritabanı sistemlerinden MySQL'in, Oracle tarafından satın alınması sonucu, şirket ile anlaşmazlık yaşayan MySQL'in çekirdek yazılımcı takımı tarafından MySQL kaynak kodlarının çatallanması ile oluşturulmuştur. Projenin açık kaynak olarak MySQL'i ikame edebilmesi

hedeflenmiştir. Wikipedia, WordPress ve Google önemli kullanıcıları arasında bulunmaktadır. İlişkisel veritabanı yöneticisi olarak veri manipülasyonu için SQL'e destek sunmaktadır. Son versiyonu ile artık JSON ve GIS veri türlerine de tam destek sunmaktadır. Zengin eklenti, depolama motoru ve araçları ile hızlı, çevik ve ölçeklenebilir olması yaygın kullanılmasına sebep olmaktadır [82] [83]. Yatay ölçeklendirmeyi MariaDB Galera Cluster sayesinde çok sayıda dağıtık sunucuyu kümeleme ile senkron olarak çalıştırabilmektedir. Yalnızca Linux'ta ve XtraDB ile InnoDB depolama motorlarını desteklemektedir [84].

3. MİKROSERVİS TABANLI ÖDEME SİSTEMİ TASARIMI

MSTOS'nin hedef kitlesi, verdiği hizmet karşılığında kullanıcılarından ödeme almak isteyen amatör ve girişimci oyun firmalarıdır. Genelde oyunlar tanıtım amacı ile ücretsiz sürümlerle oyuncuları kendine çekmeye çalışır. Ancak gelir kazanmasını sağlayacak oyun yetenekleri ve özellikleri, oyuncu ve firma arasında bir ödeme sistemi ile yapılır. Örnek olarak 1 milyon indirim sağlayan bir mobil oyunun, 5 bin ödeme yapan üyeye dönüşümü verilebilir. Ayrıca oyuncular, tanımadıkları firmaların oyunlarına ödeme yaparken şüphe ile yaklaşabilirler. Kredi kartı veya özel bilgilerinin güvenliğinden endişe duyabilirler. Bu sebeplerden dolayı girişimci firmaların oyunları, ödeme almada kayıp yaşarlar. MSTOS, bu noktada oyuncu ile firma arasında güvenli bir ödeme sistemi olarak aracılık yapmaktadır. MSTOS sayesinde, firmaların rutin ödeme alma süreçleri ile uğraşmaları gerekmez. İş gücünü daha verimli üretim yapabilmek için oyunlarına harcayabilirler. Bu tez çalışması ile oyuncu ve firmalar arasında tespit edilen bu problemlere çözüm üretilmesi hedeflenmiştir. Geliştirilen ödeme sisteminin tasarım hedefleri şunlardır:

1. Ani oluşan kullanıcı istek yüklerinin karşılanabilmesi
2. Maliyeti en aza indiren yüksek verimli yazılım mimarisi
3. Oyuncuların, MSTOS aracılığı ile farklı firmaların ücretli oyunlarını oynayabilmesi

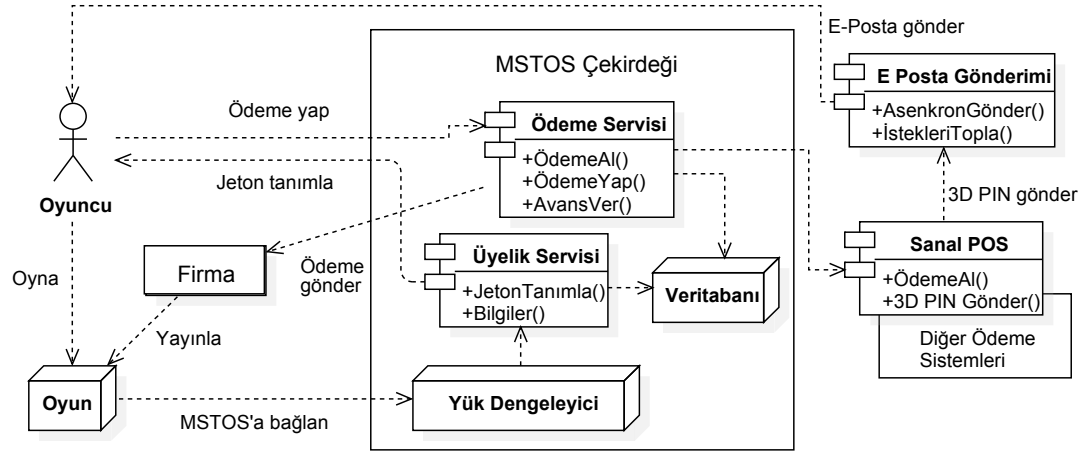
4. Firmalar için kolay ödeme alma ve raporlama
5. Oyuncuların üye oyunlarda kullanabileceği tek hesap

Oyuncu ve firma sadakatini sağlayabilmek için negatif jeton imkanı veren bir avans sistemi, tasarlanmış ama gerçeklenmemiştir. Detaylı anlatımı 3.6 Avans Sistemi altında açıklanmıştır. Bu özellik, yazılımda 2. versiyon olarak geliştirilecektir.

3.1. Genel Sistem Mimarisi

MSTOS çekirdeğinde üyelik ve ödeme işlemleri yapılmaktadır. Banka hizmeti ve e-posta gönderim servisi ise MSTOS'nin gerçeklenebilmesi için oluşturulan yan uygulamalardır. MSTOS, oyuncu ve firma arasındaki bağlantılar Şekil 3.1'de gösterilmiştir. Oyuncuların sistemi kullanabilmeleri için MSTOS'ne üye olmaları gerekir. Oyuncular firmalara ödemelerini MSTOS üzerinden yapmaktadırlar. Oyuncuların, ödemeleri karşılığında elde ettikleri ve oyunlarda harcayabilecekleri jetonlara "MSTOS Jetonu" denilir. MSTOS üyelikleri ile ödeme geçmişlerini ve bilgilerini MSTOS paneli üzerinden yönetebilirler. Firmalar, MSTOS'ne üye oyuncuların bilgilerine ulaşabilmek ve ödeme alabilmek için MSTOS'ne üye olmak zorundadırlar. Belirtilen uygulama programlama arayüzünün kurulumunu gerçekleştirdikten sonra sisteme dahil olabilirler. Oyunları için aldıkları ödemeleri MSTOS üyelikleri ile kontrol edebilirler. Dönemsel olarak kazançlarını raporlayabilirler. Oyuncu, jeton kullanarak oyun anı harcayacakları puanları firmadan

edinmiş olur. Tez içerisinde bahsedilen MSTOS, firma ve oyuncu rolleri detaylı olarak 3.2. Kullanıcı Türleri bölümünde anlatılmıştır.



Şekil 3.1. Genel sistem mimarisi

3.1.1. MSTOS çekirdeği

MSTOS çekirdeği üyelik servisi ve ödeme servisi olmak üzere iki bölümden oluşmaktadır. Üyelik servisinde oyuncuların jeton ve üyelik bilgileri depolanır. Ödeme servisinde, ödeme işlemleri yapılır ve takibini sağlanır. İki servis de Restful arayüz ile üye firmaların kullanımına sunulur. Oyuncu ve firma kendi üyeli bilgilerini ve işlem geçmişlerini MSTOS'nin sunduğu web paneli ile görüntüleyebilir. Sanal POS hizmeti veren banka servisi, MSTOS'nin ödeme almak için bağlantı kurduğu sistem dışı bir bankacılık hizmeti olduğu için MSTOS çekirdeğine dahil edilmemiştir. E-posta gönderim servisi ise banka servisi tarafından kullanıldığı için MSTOS çekirdeğine dahil edilmemiştir. Geliştirilen avans sistemi ve hızlı ödeme ile

oyunculara oyunlarından ayrılmadan harcama yapma imkanı sunmaktadır. 3.1.4 Oyun anında jeton satın alınması başlığı altında detaylı olarak anlatılmıştır.

3.1.2. Banka servisi

MSTOS Çekirdeği tarafından gelen ödeme işlemi taleplerine sanal POS hizmeti sunan uygulamadır. Kredi kartı ve miktar bilgisine göre müşterisine ödeme işlemleri için bakiye oluşturur ve beklemeye alır. Kullanıcıya ödeme güvenliği için e-posta ile 3D pini göndermektedir. 3D onayı kullanıcıya gönderilen pin ile aynı olması durumunda ödemeyi onaylamaktadır.

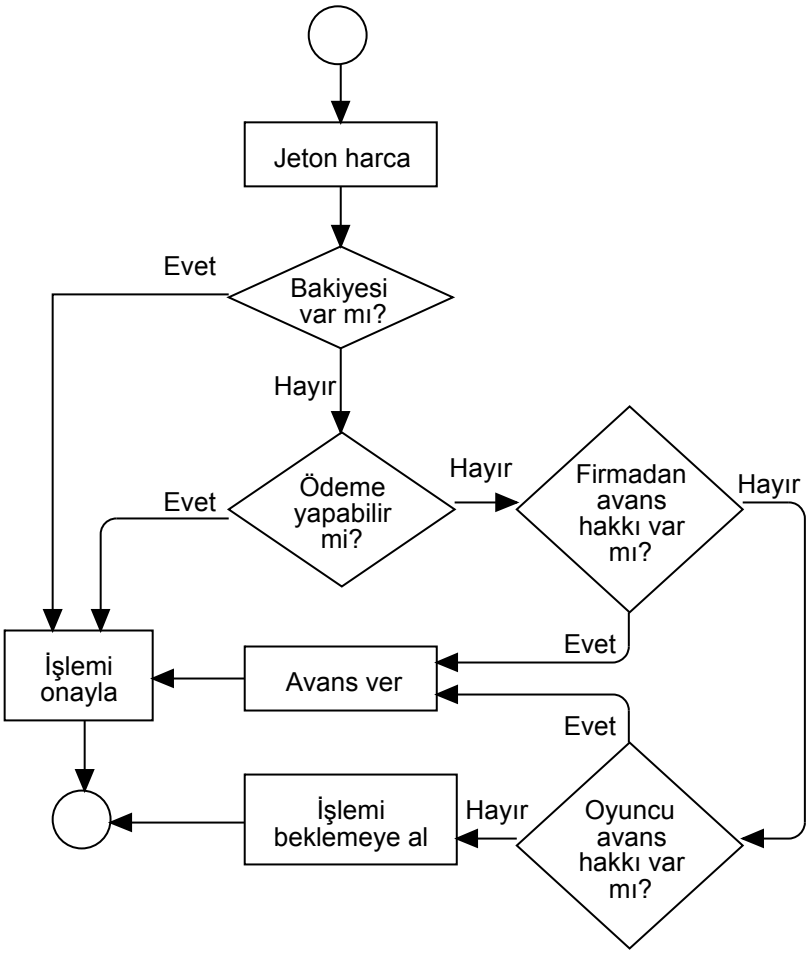
3.1.3. E-Posta gönderim servisi

E-posta gönderim isteklerini gerçekleştiren asenkron çalışan uygulama bütünüdür. İki çeşit uygulamadan oluşmaktadır. Birinci uygulama e-posta gönderim isteklerini toplamaktadır. E-posta gönderim işleminin sonuçlanması isteği yapan uygulama tarafından beklenmez ve zaman kazanılır. İkinci uygulama ise biriktirilen bilgilere göre asenkron olarak e-posta gönderme işlemi yapar. E-posta gönderim işlemi uzun sürmesi son kullanıcının etkileneceği bir zincirleme süre uzamasına sebep olmaz. Hata durumunda kuyrukta duran istek tekrar işleme alınır ve kullanıcının işlemi hata sebebi ile yarıda kesilmesi engellenmektedir.

3.1.4. Oyun anında jeton satın alınması

Burada 3 tür jeton alımı vardır. Bunlar; kredi kartlı, hızlı ve avans ile jeton alımlarıdır. Oyuncu genellikle, MSTOS uygulaması üzerinden kredi kartı ile toplu jeton alır. Oyun oynanma esnasında jeton satın alınması gerektiğinde, MSTOS arayüzüne

ulaşılması ve işlem yapılması zaman kaybına sebep olacaktır. Bu da oyuncunun oyunu yarıda bırakması veya bekleyerek oynaması anlamına gelir. Bu durumda hızlı satın alma işlemi gerçeklenir. Hızlı satın almada sadece sabit adet jeton alınabilir. MSTOS tanımlanmış olan kart bilgileri kullanarak tutarı çeker. Ancak kullanıcı onayı gerekmeyen bu yöntemde, provizyonun iptali ve benzeri durumlar söz konusu olabileceği için işlemin tamamlandığı garanti edilemez. Bu nedenle MSTOS, oyuncunun kesintisiz oynayabilmesi için küçük bir tutarı riske eder.



Şekil 3.2. Oyun anında jeton harcama algoritması

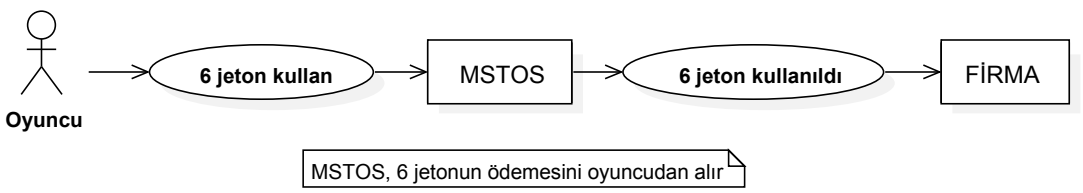
Kredi kartı bakiyesi bulunmaması ihtimaline karşılık olarak 3.6. nolu bölüm altında açıklanan avans sistemi geliştirilmiştir. Şekil 3.2 üzerinde jeton harcama algoritması gösterilmiştir. Sadece oyuncunun ödeme bakiyesi olmaması ve kullanabileceği avansı kalmaması halinde, jeton satın alma işlemi beklemeye alınmaktadır.

3.1.5. Ödeme tutarını hesaplama türleri

Toplam yapılacak ödeme tutarının belirlenmesinde firmalar açısından kolaylık olması için iki yöntem geliştirilmiştir. Birincisinde jeton adedi firma tarafından belirlenir. İkincisinde ise oyun içerisinde satın alınacak eşyalar, içerik ve adetleri ile MSTOS'ne gönderilir ve gereken jeton adedi MSTOS tarafından belirlenir. Firma, ödeme alma yöntemini, yönetim paneli üzerinden seçebilir ve seçimine göre kurulum işlemi yapar. Gerçeklenen uygulamada basit hesaplama yöntemi tercih edilmiştir.

3.1.5.1. Basit hesaplama

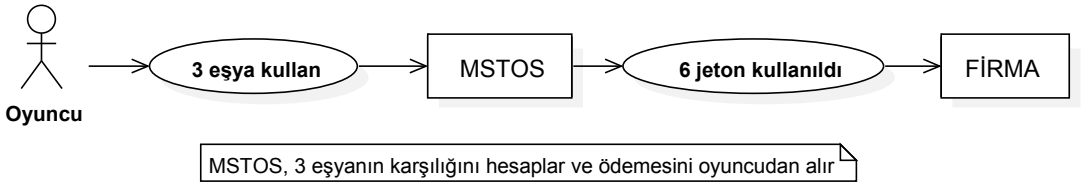
Basit hesaplama yönteminde firma sadece MSTOS'ne jeton tutarını göndermektedir. Firma tarafında eşyaların ve değerlerinin önceden belirli olması ve alışveriş öncesinde hesaplanması gerekmektedir. MSTOS gönderilen tutara göre oyuncudan ödeme tahsil eder. Firma oyuncusuna, isterse kendi iç hesabı sırasında indirim yapabilir. Şekil 3.3 üzerinde kullanım senaryosu gösterilmiştir.



Şekil 3.3. Basit hesaplama

3.1.5.2. Ayrıntılı hesaplama

Bu yöntemde, öncelikle firma, oyunu içerisindeki tüm eşyaları ve jeton olarak değerlerini yönetim paneli üzerinden MSTOS'ne kaydeder. Ödeme esnasında firma satın alınan eşyaları ve adetlerini gönderir. MSTOS alışverişin içeriğine göre hesaplama yapar ve tutarı oyuncuya yansıtır. Bu yöntemde, firmanın ekstradan eşya listesi tutma ve tutar hesaplama yeteneklerini oyununa eklemesine gerek kalmaz. Eşyaların jeton karşılığı MSTOS üzerinde sabit tanımlı olduğu için indirim uygulanması mümkün olmaz. Şekil 3.4 üzerinde kullanım senaryosu sunulmuştur.



Şekil 3.4. Ayrıntılı hesaplama

3.1.6. Oyuncu paketleri

Oyuncuların jeton satın alma miktarlarına göre dahil oldukları paket belirlenmektedir. Paket değişimi için fazladan ücret ödemeleri beklenmez. Bunun ile ilgili belirlenen jeton alım adetleri Tablo 3.1'de belirtilmiştir. Bu paketlere göre oyuncunun günlük hızlı ödeme özelliğini kullanmak hakkı sınırlandırılmıştır. Hızlı ödemelerde, oyuncunun kredi kartından 3D ödeme onayı olmadan ödeme alınmaktadır. İleriye dönük itirazlar ile ödemeler iptal olabileceği için belirlenen paketler halinde

sınırlandırılmıştır. Ayrıca bu paketler, 3.6.3. Avans risk yönetimi bölümünde bahsedilen avans miktarlarının belirlenmesinde de kriter olarak kullanılmıştır.

Tablo 3.1. Oyuncu paketleri

Paket Adı	Jeton Adedi	Geçerlilik Süresi	Günlük Hızlı Ödeme Hakkı
Standart	0 - 49	Süresiz	10
Bronz	50 - 99	1 Ay	20
Gümüş	100 - 499	3 Ay	30
Altın	500+	6 Ay	50

3.2. Kullanıcı Türleri

Tez için oluşturulan kullanıcı türleri 3 rol altında toplanmıştır. Bunlar; oyuncu, firma ve MSTOS'dir. MSTOS ise yönetici ve destek elemanı olmak üzere 2 adet alt role sahiptir. Oyuncu, firmada harcamak amacıyla jeton satın almak için MSTOS'ne ödeme yapar. MSTOS, ödemeleri firmaya dönemsel olarak aktarır. Oyuncu, ödeme yapma haricinde tüm işlemlerinde firmaya bağlıdır. Oyuncu ek olarak sahip olduğu profilini de MSTOS'ne bağlanarak yönetebilir. Firmanın yaşayacağı teknik problemler veya ilk MSTOS'ne üyeliği için gereken provizyon işlemleri MSTOS destek elemanı tarafından yapılır. MSTOS'nin fiyat politikası, donanım ihtiyacı ve firmaya ödemelerin aktarılması gibi yönetimsel süreçler MSTOS yöneticisi tarafından gerçekleştirilir.

3.2.1. Oyuncu

Firma tarafından yayına alınan oyunlara talep oluşturan son kullanıcı grubudur. Oyunları kullanımları sonunda beğenmeleri durumunda arkadaşlarına önermeleri ile başarılı oyunlar üzerinde hızlıca büyüme ve yoğunluk oluşturacakları varsayılmaktadır. Ödeme ve üyelik hesap bilgileri MSTOS tarafından tutulur. Oyun içerisinde herhangi bir satın alma yapacağı zaman jeton almak için gereken ödemeyi MSTOS'ne yapar. Oyuncular, deneyecekleri yeni oyunlarda diğer oyunlardan kalan jetonlarını kullanabilirler. Hesap hareketlerini, MSTOS profil sayfalarından kontrol edebilirler. MSTOS tarafından oyuncuya, her ödeme işlemi tamamlandıktan sonra fatura kesilir.

3.2.2. Firma

Yayına almak istedikleri oyun projeleri için üyelik ve ödeme ile ilgili konularda MSTOS'dan hizmet almaktadır. MSTOS tarafından sunulan 3.1.5. Ödeme tutarını hesaplama türleri başlığı altındaki ödeme hesaplama türüne göre kurulum yapmaktadır. Oyuncu bağımlılığı edinmek isterse sorumluluğunu almak şartı ile oyuncularına avans olarak negatif jeton bakiyesi verebilir. Dönemsel olarak MSTOS üzerinden kapasitesine göre aldığı hizmete karşılık olarak seçtiği pakete göre komisyon ödemesi yapar. Dönem içerisindeki ödeme işlemlerinin raporlarını görüntüleyebilir.

3.2.3. MSTOS kullanıcı türleri

MSTOS'nin yönetimsel ve ödemeler ile ilgili süreçleri yönetici tarafından kontrol edilir. Yeni firma üyeliği için provizyon ve yaşanılan problemlerin çözümü gibi rutin operasyonel işlemlerin sürdürülmesinde ise destek personeli görev almaktadır.

3.2.3.1. Yönetici

MSTOS'nin yönetim, altyapı ve kar/zarar yatırım oranları ile ilgilenir. MSTOS'u oluşturan uygulamaların kaynak ihtiyacı kontrolünü yapar. Aktif kullanıcı sayısı artışı ile sistemin kaynak tüketimi ihtiyacı paralel olarak artar. Bu sebeple havuza yeni sunucular eklemesi gerekir. Yeni sunucuların sisteme dahil olması ise maliyet kaleminin artışına sebep olmaktadır. Maliyet kaleminin aktif kullanıcı sayısına göre artışını göz önüne alarak Firmalar için ödeme paketleri oluşturur. Oyuncu avansı ile ilgili inceleme ve değişiklikleri yapar. Firma, Oyuncu ve MSTOS arasındaki karşılıklı sorumlukların belirlendiği prosedürleri oluşturur. Oyuncular ile Firmalar arasında MSTOS üzerinden gerçekleşen ödeme ve işlem raporlarını kontrol eder. MSTOS'nin operasyonel olarak büyümesi durumunda teknik yönetim sorumluklarını "Sistem Yöneticisi" benzeri yeni bir role aktarması düşünülebilir.

3.2.3.2. Destek elemanı

MSTOS için rutin teknik işlemlerin yürütülmesi ve takibi sağlar. MSTOS'ne yeni firma başvurusunu tanımlar ve gereken provizyon işlemlerini tamamlar. Firma ve oyuncu tarafından gönderilen şikayetlere geri dönüş yapar. Sistem hata kayıtlarını kontrol eder. Hata tespiti sonrasında mağduriyet oluşmaması adına gerekli

müdahaleyi sağlar. Havuz içerisinde bulunan sistem kaynaklarının durumlarını takip eder. MSTOS'u oluşturan servis bileşenlerinden hangisinde yoğunluk varsa kullanılmayan sistem kaynaklarından aktarım yapar ve maksimum düzeyde verimliliğe ulaşılmasını sağlar. Konteyner mimarisi kullanıldığı için MSTOS bileşenlerinin dağıtım, ölçekleme ve izleme (monitoring) işlemlerini bir arada yürütebilmektedir. Yetersiz sistem kaynağı kaldığında MSTOS Yöneticisi'ne sistem kaynağı havuzunun genişletilebilmesi için satın alma talebi oluşturur.

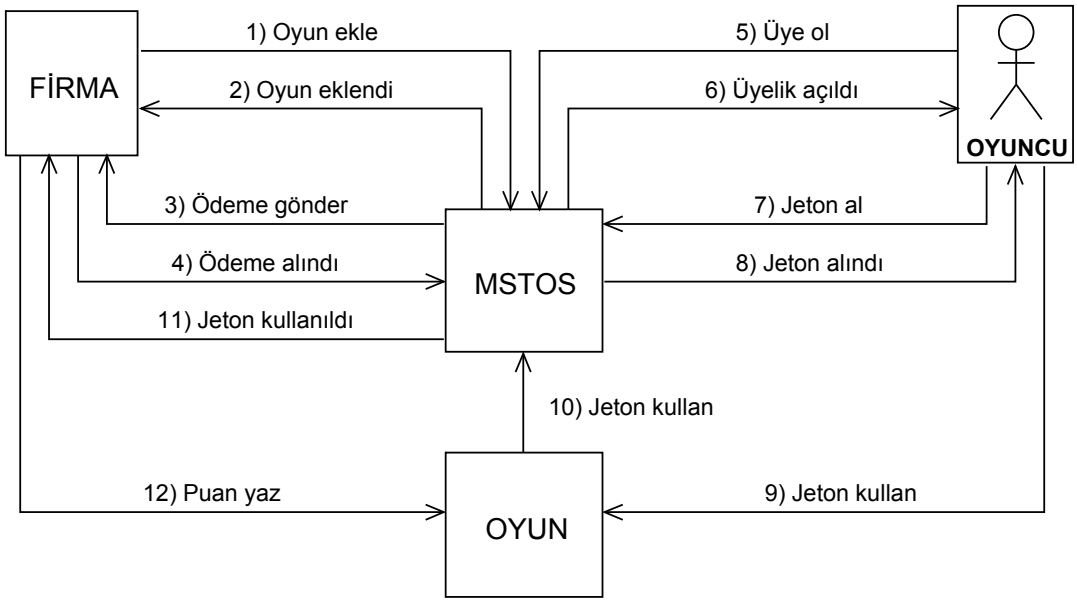
3.3. İşlem Akışı

İşlem akışı; oyuncu ile MSTOS arasında jeton alımı, firma ile MSTOS arasında jeton harcaması ve ödemenin firmaya aktarılması olarak 3 alt işleme ayrılmıştır. Banka üzerinden ödeme çekimi dahil edilmemiştir. İşlem akışını oluşturan adımlar, aşağıdaki Şekil 3.5'de gösterilmiştir. Adımların gruplandırılması şu şekildedir;

1 – 4 arası: Firma ile MSTOS ilişkisi

5 – 8 arası: Oyuncu ile MSTOS ilişkisi

9 – 12 arası: Oyuncu, firma, oyun ve MSTOS ilişkisi



Şekil 3.5. İşlem akışı

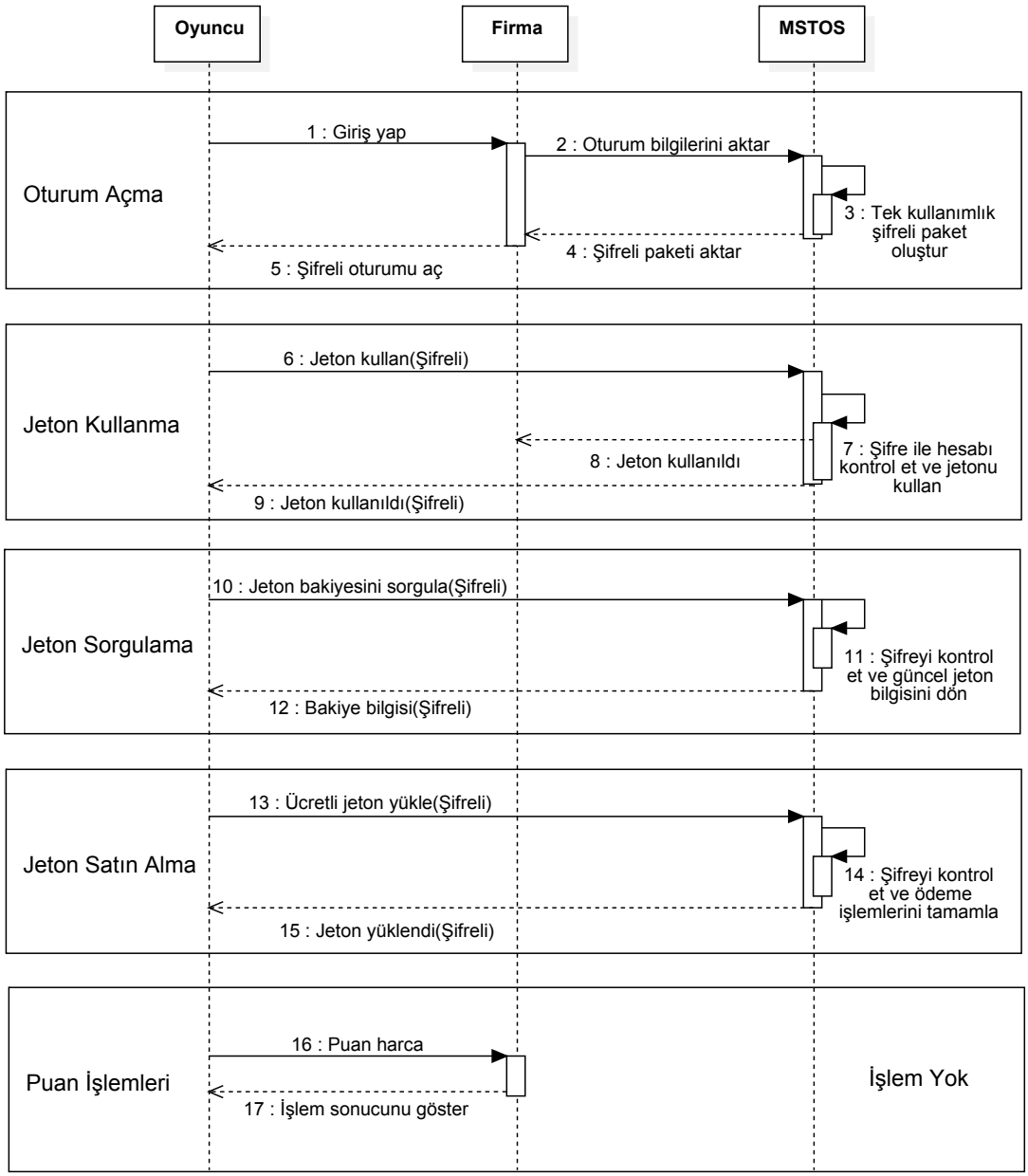
Yukarıda sunulan Şekil 3.5 üzerinde anlatılan işlemler, Tablo 3.2 üzerinde ise işlem türüne göre gruplandırılmış olarak açıklanmıştır. İşlem akışının başlangıcı olarak 1 ve 2. adımlarda, firmanın üyelik açması ve oyununu MSTOS'ne eklemesi belirlenmiştir. MSTOS ile firma arasındaki dönemsel ödemelerin yapılma işlemi 3. ve 4. adımlar ile gösterilmiştir. Sonraki adımlar olarak; oyuncunun MSTOS'ne üyelik açması 5. ile 6. adımlarda, jeton satın alma ve harcaması 7. ile 10. adımlar arasında ve oyuncunun oyun içerisinde puan kazanması 11. ile 12. adım olarak sayılmıştır.

Tablo 3.2. İşlem akışı adımları

No	Aktör	İşlem	İşlem Türü
1	Firma	MSTOS'ne oyununu ekler	Üyelik
2	MSTOS	Firma sisteme dahil edilir	Üyelik
3	MSTOS	Firmaya ödemeler dönemsel olarak yapılır	Firmaya ödeme
4	Firma	Yapılan ödemeleri kontrol eder	Firmaya ödeme
5	Oyuncu	MSTOS'ne üye olur	Üyelik
6	MSTOS	Oyuncu sisteme dahil edilir	Üyelik
7	Oyuncu	Jeton satın alır	Jeton alımı
8	MSTOS	Oyuncunun hesabına jetonu yükler	Jeton alımı
9	Oyuncu	Oyun içerisinde jeton kullanmak ister	Jeton kullanımı
10	Oyun	Jeton kullanılmasını talep eder	Jeton kullanımı
11	MSTOS	Jeton kullanıldığı ile ilgili bilgi verir	Jeton kullanımı
12	Firma	Jetona karşılık olarak puan tanımlar	Jeton kullanımı

3.4. Oyuncu, firma ve MSTOS arasındaki iletişim

Oyuncu, firma ve MSTOS arasındaki iletişim trafiği Şekil 3.6'da sunulmuştur. Burada gösterilen işlemler; oturum açma, jeton kullanma, jeton sorgulama ve jeton satın alma işlemleridir. Sunulan şekil üzerindeki puan işlemlerinde ise MSTOS ile puan işlemleri arasında herhangi bir bağ olmadığının vurgusu yapılmıştır. MSTOS ve firma arasındaki iletişimin güvenli olarak gerçekleşebilmesi için her oyuncu için farklı tek kullanımlık JWT protokolü ile oluşturulmuş şifreli paket oluşturulmaktadır. Firmanın oyunu, bu şifreli paket ile oyuncu adına işlem yapabilmektedir. Oyucunun her işleminden sonra, bu şifre değiştirilmektedir. Oyuncudan gelecek her işlem için güncel şifreli paket de gönderilmelidir. Sadece oturum açma işleminde, istek firma üzerinden aktarılmaktadır. Diğerlerinde istekler, oyun üzerinden direkt olarak MSTOS'ne yapılmaktadır. Bu sayede firma sunucusu yükten kurtarılmış olmaktadır.



Şekil 3.6. Oyuncu, firma ve MSTOS arasındaki iletişim

Oluşturulan şifreli paket içerisine oyun sunucusunun IP adresi de eklenmektedir. MSTOS'ine yapılan isteklerde, şifrenin doğruluğu teyit edildikten sonra şifre içerisindeki IP adresi ile isteği yapanın IP adresi karşılaştırılmaktadır. Bu sayede

“Ortadaki Adam Atağı (Man-the-middle attack)” [85] benzeri saldırılar için önlem alınmıştır. Şifreli paket ile güvenliğin sağlanması sayesinde veri tabanına isteklerin geldiği kaynağın teyidi için ek yük oluşturulmamış olmaktadır. Bu her istek için ciddi performans kazancı sağlamaktadır. Puan kullanımı işlemleri ise firmanın oyunu ve oyuncu arasında gerçekleşmektedir. Bu konuda MSTOS ile herhangi bir iletişim kurulmamaktadır.

3.5. MSTOS Gelir Paylaşımı

MSTOS’nin kazancı, oyuncuların firmalara yaptığı ödemelerden aldığı komisyon ile oluşmaktadır. Kazancın ne olabileceği konusunda fikir verebilmesi için bir deneysel çalışma yapılmış ve bu bölümde verilmiştir. Tablo 3.3‘de sunulan 3 adet paket ile işlem adetleri gruplandırılmıştır. Firma için aracılık ettiği işlem adedi arttığında komisyon yüzdesi azalır. MSTOS’nin masraf kalemi, altyapısını oluşturan sunucu giderleri ve personel giderleridir. Bunların ödemesi, aylık sürelerde ya da belirlenen dönemlerde hizmet alınan firmaya yapılmaktadır. İlk dönemlerde masraflarını amorti edecek adette işleme aracılık etmesi gerekir. Bu sebeple, yüksek işlem adedine sahip firmaların MSTOS kullanımına devam etmesi için komisyon oranı düşük tutulur. Oyuncudan çekilen tutar içerisinden alınan komisyon, oyunu geliştiren şirketten kesildiği için oyun hak edişleri jeton tutarı, komisyon ve vergiler düşülmüş olarak dönem sonlarında oyunu geliştiren şirketlere aktarılır.

Tablo 3.3. MSTOS'nin firmalar için komisyon paketleri

Paket Adı	**İşlem Adedi**	**Komisyon Yüzdesi**
Bronz	100	% 20
Gümüş	1000	% 10
Altın	1000+	% 5

Firmalara sağlanan hizmet sonucunda kullanılan sistem kaynaklarının maliyetlerinin fikir vermesi amacı ile ortalama bir hesap yapılmıştır. Altyapı hizmeti alınan firma olarak Amazon Web Services tercih edildi. Tez içerisinde Tablo 3.4'de belirtilen özelliklere sahip bir sunucu ile sistemin canlıya alındığı varsayılmaktadır. Aylık masraf yaklaşık olarak 1.200 TL çıkmaktadır [86].

Tablo 3.4. Altyapı maliyet kalemleri

Sunucu	**Özellik**	**Ücret**
T2.2xlarge-EC2	8 CPU, 32 GB RAM, Linux/GNU OS	316,23 $
S3 Storage	10 GB SSD	0,25 $
ElastiCache (Redis)	1 CPU, 1 GB RAM	13,91 $
CassandraDB	-	0 $
Toplam		**330,39 $**
Güncel Kur		**3,54 TL**
Genel Toplam		**1.169,58 TL**

Sistem ihtiyaçlarına uygun olarak NoSQL veritabanı olan 10 GB SSD depolama alanına ve 1000 Okuma/Saniye hızına sahip DynamoDB'nin eklenmesi durumunda 679.16 $ fiyatı ile maliyet yaklaşık olarak 3 katına çıkmaktadır. Bu sebeple listeden çıkartılmıştır. İkame olarak EC2 sunucusu üzerine ölçekleme ve dağıtık çalışma

konularında son derece başarılı bir NoSQL veritabanı olan Cassandra kurulmuştur. Bu sayede yaklaşık olarak %66 oranında maliyet tasarrufu sağlanmıştır.

Tablo 3.3 üzerinde tanımlanan paketler içerisinden Paket 2 baz alındığında ödeme işlemi başına, tutarın %10'luk kısmı MSTOS tarafından komisyon olarak kesilmektedir. Sunucu altyapı masraflarının karşılanabilmesi için aylık en az 12.000 TL'lik ödeme alması gerekmektedir. Yüzeysel bir hesap ile 12.000 TL'lik ödemenin tahsilatı için ortalama olarak sıradan bir oyuncunun 10 TL'lik bir ödeme yaptığı varsayılır ise 1.200 işlem sonrasında tamamlanmış olmaktadır. Tasarlanan ve satın alınan sistem kaynakları ile MSTOS saniyede yaklaşık 1000 adet işlemi (request) karşılayabilmektedir. 1.200 işlemin MSTOS tarafından tamamlanması yaklaşık olarak 1,2 saniye sürmektedir. Bu sebeple MSTOS'nin masraflarına göre kar oranı oldukça yüksek olmasına rağmen kazanç sağlayabilmesi için sistemini kullanan firmaların oyuncularını harcama yapmaya ikna etmelerine bağımlıdır.

3.6. Avans Sistemi

MSTOS'nin oyuncularının memnuniyetlerini ve bağımlılıklarını artırabilmek için sunduğu bir hizmettir. Oyuncuların satın alma geçmişlerine veya firmaların sorumluluğunu aldıkları tutara göre, belirlenen limitler dahilinde, oyunculara avans olarak negatif jeton bakiyesi tanımlanmaktadır. Avans sisteminin amacı oyuncuların ödeme yapamadıkları zamanlarda borç karşılığında harcamaya devam edebilmelerini sağlamaktadır. Oyuncuya sunulan iki çeşit avans türü bulunmaktadır. Bunlar; firma avansı ve oyuncu avansıdır. Şekil 3.2'de gösterildiği gibi öncelikle oyun için firma

avansı eklenmişse, oyuncuya sunulur. Eğer ki firma avansı yoksa, MSTOS tarafından oyuncu avansı sunulur. Avanslar birleştirilemez. Oyuncunun, ilk jeton satın alışı sonrasında bakiyesi mahsup edilir.

3.6.1. Firma avansı

Oyuncularının bağlılığını artırmak veya harcamalarını teşvik etmek isteyen firmaların talebi ile oluşturulur. Firmalar belirledikleri tutarı, MSTOS yönetim panelleri üzerinden bildirirler. Firmanın yeni jeton almak isteyen oyuncuları, belirlenen tutar kadar negatif jeton avanslarından kullanabilirler. Oyuncu avansını kullandıktan sonra yeni jeton almazsa, negatif jetonlar firma onayı ile verildiği için sorumluluk firmaya aittir.

3.6.2. Oyuncu avansı

MSTOS tarafından verilen avanstır. Oyuncunun geçmiş jeton satın alma tutarlarına göre hesaplanır. Hesaplama ile ilgili bilgiler 3.6.3. Avans risk yönetimi‘nde verilmiştir. Firma avansı sunulmayan oyunlarda verilmektedir. MSTOS’nin oyunculara, diğer ödeme sistemlerine karşı rekabet ortamında sunduğu önemli avantajlardan birisidir. Oyuncu avansını kullandıktan sonra yeni jeton almazsa, zararın sorumluluğu MSTOS’ne aittir. Oyuncunun, oyunda kullandığı negatif jetonların ödemesi de, dönem sonunda firmaya yapılır. Yeni oyuncuları sisteme dahil etmek için promosyon olarak, hiç jeton almadan avanslı olarak başlatılması yapılabilir. Promosyon konusundaki çalışma genişletilmemiştir. Örneğin promosyonlar yeni oyuncularda referanslarına göre olabilir.

3.6.3. Avans risk yönetimi

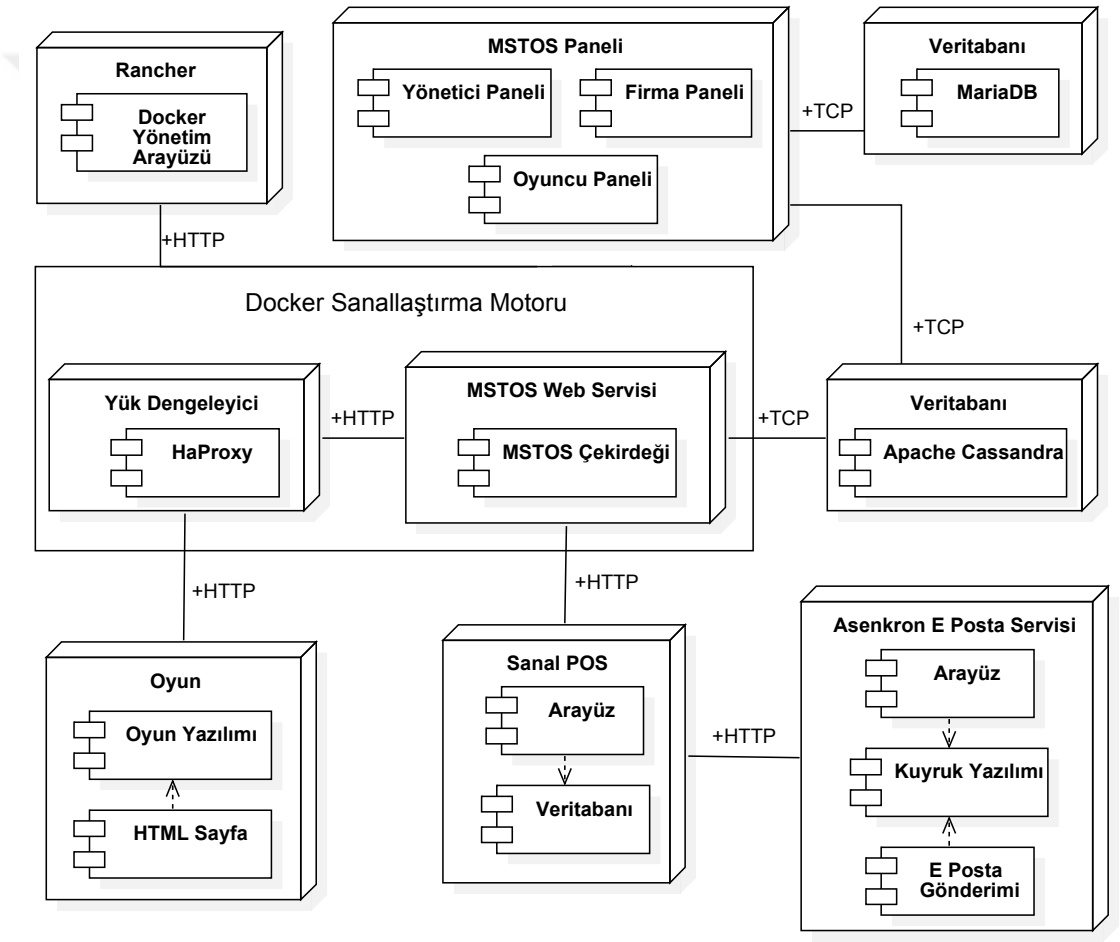
Her oyuncunun ödeme geçmişindeki toplam jeton satın alma tutarına göre avans limiti oluşturulmaktadır. Bu yüzden her oyuncu için farklı tutarda avans çıkabilir. Oyuncuların harcamalarına göre belirlenen negatif jeton adedi Tablo 3.5'de verilmiştir. Tutarların belirlenmesinde Tablo 3.3'de firmalara sunulan komisyon yüzdeleri içerisinde en düşüğü olan %5 kar oranını aşmamasına dikkat edilmiştir. Tahmini olarak %10 oranında verilen avansların geri ödenmeyeceği varsayılmıştır. Bu varsayıma göre kar edilen tutarın %10'u avans sistemi için riske edilmiştir. Dönemsel olarak edilen kar tutarı, bir sonraki dönem için riske edilerek verilecek avanslar için havuzun limitini belirlemektedir. Limit aşımı durumunda, tüm oyuncular için avans sistemi askıya alınır ve yeni avans verilmez.

Tablo 3.5. Oyuncuya verilecek avans jeton adetleri

Dahil Olduğu Paket Adı	**Oyuncunun Satın Aldığı Toplam Jeton Adedi**	**Avans Verilecek Jeton Adedi**
Standart	0 - 49	0
Bronz	50 - 99	2
Gümüş	100 - 499	5
Altın	500+	10

4. MİKROSERVİS TABANLI ÖDEME SİSTEMİ GERÇEKLENMESİ

MSTOS, yönetim paneli ve yardımcı sistemlerin birbirleriyle olan bağlantıları Şekil 4.1'de gösterilmiştir.



Şekil 4.1. MSTOS yazılım mimarisi

MSTOS'nin sanallaştırma işlemi için Docker kullanılmıştır. Docker ile ölçeklendirme için gerektiği kadar konteyner üretimi yapılabilmektedir. Konteynerler içerisinde çalışan mikroservisler arası iletişim ise RESTful web arayüzü ile sağlanmaktadır.

Docker sanallaştırma motorunun yönetimi için Rancher kullanılmıştır. Mimari detaylı olarak 4.1 MSTOS Yazılım Mimarisi başlığında anlatılmıştır. Gerçeklenme ile ilgili bilgleri ise 4.2 MSTOS'nin Gerçeklenmesi bölümünde sunulmuştur.

4.1. MSTOS Yazılım Mimarisi

MSTOS çekirdeğinin gerçeklenmesi süresince, güncel teknolojilerin kullanılmasına önem verilmiştir. Yapılan karşılaştırmalar sonucunda bu süreyi kısaltan, verimliliği artıran ve yaygın kullanım oranlarına sahip olan teknolojiler tercih edilmiştir. Bu teknolojiler, kullanıldığı gerçeklenme esnasındaki süreçlere göre aşağıdaki gibi gruplandırılmış ve sebepleri ile anlatılmıştır.

4.1.1. Dağıtım ve ölçeklendirme

Kullanıcıların hizmetine sunulmak için yayına alınacak web uygulamalarının dağıtımı ve ölçeklendirmesi için Docker kullanılmıştır. İstenilen uygulamanın, gereksinimleri ile paketlenmesini ve böylece çalıştırılacağı sistemden bağımsız olarak her an kullanıma hazır kalıplarının oluşturulmasını sağlamaktadır. Bu kalıpların sunucular üzerinde koşturulan kopyalarına konteyner denilmektedir. Uygulamanın ihtiyaç duyacağı kütüphane, tanımlamalar veya dosyalar konteyner kalıbı içerisine ihtiyaç duyulduğu hali ile eklenebilir. Oluşturulan kalıplar Docker kurulu olan herhangi bir sunucu üzerinde aynı kararlılık ve davranışlar ile çalışmayı garanti etmektedir. Kalıbın çalışan konteyner adedi artırılarak ise yatay ölçekleme yapılabilmektedir. Her konteyner, birbirinden ve sunucu sistemden yalıtılmış olduğu için yüksek güvenlik düzeyi ile çalışmaktadır. Docker yerine kullanılabilecek alternatif teknolojiler olarak;

LXD [87], rkt [88], Solaris Containers [89], BSD Jails [90] ve Windows Containers [91] sayılabilir. Fakat bu teknolojiler arasında Docker kullanım kolaylığı ve topluluk desteği sayesinde öne çıkmaktadır. 2017 yılında yapılan bir araştırmaya göre kullanım popülerliği ve pazar payı olarak %78 ile Docker lider konumdadır [92]. Bu sebeplerle, tez içerisinde hazırlanan uygulamanın paketlenmesi, yayına alınması ve ölçeklendirilmesi Docker ile yapılmıştır.

4.1.2. Ölçeklendirme yönetimi

Ölçeklendirme sonucu ortaya çıkan çok sayıda konteynerin terminal üzerinden Docker komutları ile yönetilmesi zaman kaybına ve takibinin zorlaşmasına sebep olabilmektedir. Web tabanlı görsel arayüzü sayesinde hızlı durum tespiti ve aksiyon almaya olanak sağlayan Rancher, konteyner orkestrasyon aracı olarak tercih edilmiştir. Rancher, Docker konteynerlerini çalıştıracak sunucuların eklenilmesini, yüklerine göre konteyner dağıtılmasını ve ayrı sunucular üzerinde çalışan konteynerlerin aynı sunucuda çalışıyor gibi haberleşebilmesini sağlamaktadır. Yeni kurulmuş bir sunucunun Rancher kontrolüne girebilmesi için sadece Docker servisi kurulması ve Rancher konteynerinin bir defaya mahsus olarak ilgili sunucuda çalıştırılması yeterli olmaktadır. Rancher sayesinde oluşturulan havuza eklenen sunucularda sistem kaynaklarının sınırları içerisinde istenilen adette Docker konteyneri oluşturulabilmektedir.

4.1.3. Programlama dili

UNIX felsefesinde olduğu gibi konteyner mimarisinde de bir konteyner içerisinde çalışan bir tane uygulama olması ve bu uygulamanın işini en iyi şekilde yapması gerekmektedir [93]. Uygulamanın verimliliğini, mümkün olduğu kadar sistem kaynağı ihtiyacının az olması ve hızı belirlemektedir. Yorumlanan dillerden olan PHP'nin çalışabilmesi için PHP yorumlayıcısına ve bir web sunucu uygulamasına minimum gereksinim olarak ihtiyaç duyulmaktadır. Go programlama dili, sahip olduğu basit ve güçlü HTTP kütüphanesi sayesinde her web isteği öncesinde yorumlanmak zorunda olan betik dillerine alternatif olarak makine diline çevrilmiş doğal web uygulamaları yazmayı sağlamaktadır. Go programlama dili; yapılan çoğu arama, sıralama ve benzeri algoritma performansı karşılaştırmalarında açık ara öndedir. Bu karşılaştırma testlerinde kullanılan dillere; PHP [94], NodeJS [95], Python [96] ve Java [97] örnek olarak verilebilir. Bu sebeplerle bağımsız, hızlı ve minimal sunucu uygulamaları oluşturmaya imkan veren Go programlama dili, geliştirme dili olarak tercih edilmiştir.

4.1.4. Web uygulaması mimarisi

Web sayfa görüntüleme isteklerinin sunuculara iletilmesi ve işlenen verilerin sonucunun istemciye geri dönülmesi ile oluşan süreçte her defasında sayfanın tamamının kodları üretilmektedir. Eğer ki sadece küçük bir bölüm değiştirmek veya işlev tetiklenmek isteniyorsa, yeni bir kullanıcıya sunuluyor gibi tüm sayfanın oluşturulması, mükerrer iş ve iş gücü kaybına sebep olmaktadır. Bu klasik yöntem yerine Ajax ile web arayüzü üzerinden spesifik amaçlı istekler web sunucularına

iletilebilir ve JSON sonuçları istemcinin tarayıcısında sadece değişmesi gereken kısımlar güncellenerek gerçekleştirilebilmektedir. Ajax sadece az veri trafiği ürettiği için sunucunun bant genişliğinden, tam olarak eksik veriyi veya işlemi temsil etmesinden dolayı da işlemci ve diğer sistem kaynaklarından tasarruf etmeyi sağlamaktadır.

Geliştirilen uygulamalardan, web platformu haricinde mobil platformunda da karakteristiği haricinde görevleri yerine getirmesi beklenmemektedir. Mobil uygulamalar geliştirebilmek için cep telefonu veya tabletin işletim sistemine uygun kodlanması gereken arayüzler dışında uygulamanın özünü oluşturan hareketleri web uygulamasından farklı olmamaktadır. Bu sebeple, web uygulaması için oluşturulan bir uygulama programlama arayüzü, mobil uygulama içinde sunucu ile iletişime geçmeyi, standart ve tekil sunucu uygulamasına sahip olmayı sağlamaktadır. Web programlama arayüzü kullanıldığı için uygulama oluşturulmak istenen herhangi bir platform için sadece platform arayüzü oluşturulması yeterli olmaktadır [98]. Bu sebeplerle sunucu uygulaması web uygulama programlama arayüzü tabanlı tasarlanmıştır.

4.1.5. Yönetim paneli

MSTOS içerisinde tanımlı olan oyun, oyuncu ve firmalar için yönetim paneli oluşturulmuştur. Bu panel aracılığı ile oyuncuların ve firmaların başvuruları da alınmaktadır. Oyuncu ve firmaların da kendi hesaplarını yönetebileceği ek arayüzler tasarlanmıştır. Oyuncular, kendi panelleri üzerinden kredi kartı ile jeton alabilmektedirler. Jeton bakiyelerini görüntüleyip, harcamalarını kontrol

edebilmektedirler. Firmalar, panelleri aracılığı ile yeni oyun tanımlayıp, oyunları içerisinde gerekli yapılandırmayı sağladıklarında MSTOS'ni kullanmaya başlayabilmektedirler. Hangi oyundan ne kadar ödeme aldığı gibi temel detayları görebilmektedirler. Panelin amacı sadece veri yönetimi amaçlı oluşturulduğu için MSTOS çekirdeğinin çok sayıda olabilecek işlemler karşısında performansına bir etkisi olmamaktadır. Bu sebeple panel, PHP ile ayrı bir web yazılımı olarak oluşturulmuştur. Son derece popüler olması ve sağladığı kolaylıklar sebebi ile Laravel isimli PHP kütüphanesi kullanılmıştır [61].

4.1.6. Uygulama programlama arayüz dokümantasyonu

Küçük, kısa zamanlı veya kişisel projelerde web arayüz tasarımı, kullanımı ve dokümantasyonu önemsenmezse pek bir kayıp olacağı iddia edilemez. Fakat bir ekip tarafından geliştirilen, uzun vadede ayakta tutulması planlanan ve farklı ekipler ile etkileşimi söz konusu olabilecek uygulamalar için önem taşımaktadır. Uygulama gerçeklendikten sonra yazılım dokümantasyonun hazırlanması geliştirme sırasında yol haritasının belirsizliği anlamına gelmektedir. Her yazılımcının dokunduğu alanda kendi görüşüne göre projenin farklı biçimlenmesine sebep olabilmektedir. Arayüz dokümantasyonu ekiplerin belirlenen kalıplara göre uygulamaları kodlamasını sağlayan ve karşılıklı sorumlukları garanti altına alan bir sözleşmedir. Bu sebeplerden dolayı öncelikli olarak arayüz taslağı belirlenmeli ve dokümantasyonu hazırlandıktan sonra yazılımcılar tarafından uygulama geliştirilmesi önerilmektedir. Tez uygulamasında arayüz dokümantasyonu oluşturmak, taslak belirlemek, sunucu ve

istemci için başlangıç kaynak kodlarını üretebilmek için OpenAPI standardı ile Swagger kullanılmıştır [75].

4.1.7. Dağıtık oturum yönetimi

Web uygulamalarında kullanıcıları, üyelik bilgileri ile eşleştirebilmek için çerezler (cookie) kullanılarak oturumlar (session) oluşturulmaktadır. Yetki ve bilgi güvenliği sağlamak için oturum bilgileri sunucuda tutulmaktadır ve her ziyaretçiye açıkça dağıtılan çerezler içerisinde oturum bilgilerine işaret eden anahtarlar bulunmaktadır. Bu sayede yetkisiz kişiler sahip oldukları oturum anahtarını sunucu erişimine sahip olmadan manipüle edememektedir. Ek olarak oturum bilgileri sunucuda tutulduğu için uygulama sahibi tarafından zaman aşımı gerekçesi ile sonlandırma gibi işlemler için kolayca yönetilebilmektedir. Fakat bu yöntemde oturum açan kullanıcı sayısı arttıkça sunucu üzerinde oturum bilgileri de paralel olarak artarak depolama kaynaklarını tüketmektedir. Ölçeklendirme ihtiyacına uygun bir yöntemde değildir çünkü yük paylaşımı sunucuların yoğunluğuna göre yapılması gerekirken kullanıcının oturum bilgisi tutulan sunucuya yönlendirilmesi zorunluk kılmaktadır. Çözüm olarak ekstra kaynak tüketimi ile ortak oturum havuzu için özel sunucu veya servisler oluşturulmaktadır. Bunun yerine oturum bilgileri JWT (JSON Web Token) standardının önerdiği gibi sunucuda depolamaya gerek olmadan kullanıcılara açıkça dağıtılabilir ve tarayıcı çerezleri arasında depolanabilmektedir. Kötü niyetli kullanıcıların bilgileri manipüle etmesine engel olmak için belirlenecek gizli anahtar ile oturum bilgisi güçlü kriptoloji algoritmaları tarafından şifrelenmektedir. Kullanıcıya şifreli özeti ile oturum veya herhangi bir bilgi emanet edilebilir fakat geri

teslim alındığında gizli anahtar ile şifreleme tekrar edilerek manipülasyon durumu teyit edilmesi gerekmektedir. Gizli anahtara sahip herhangi bir sunucu ile emanet edilen bilgi teyit edilebilir ve güvenli olarak ölçeklendirme yapılabilmektedir [65]. JWT, mobil ve diğer platformlarda herhangi bir bilgiyi veya uzun süreli kalıcı oturum bilgilerini kullanıcıya emanet etmek için kullanılmaktadır. Bu sebeplerle ölçeklendirme ve kaynak tasarrufu avantajlarından dolayı tez uygulamasında oturum yönetiminde JWT kullanılmıştır. JWT'nin klasik sunucuda tutulan oturumlara karşı dezavantaj olarak değerlendirilebilinecek tek eksikliği oturumların istenildiği anda oturum dosyasının silinerek sonlandırılamamasıdır. Çözüm olarak kullanıcıya verilen JWT verisine mutlaka son kullanım tarihinin de eklenmesi ve kullanıcılardan JWT'ler geldikçe kontrol edilerek oturum kapanmış olarak değerlendirilmesi gerekmektedir.

4.1.8. Veritabanı

Cassandra, çok yüksek miktarda verinin dağıtık olarak saklamak için tasarlanmıştır. Ölçeklendirme ile kümeye yeni eklenen düğümleri otomatik olarak tanır ve yapılandırmasını yapmaktadır. Kümedeki bir düğümün çökmesi diğer düğümleri ve sistemin genel çalışmasını etkilememektedir. Kümedeki düğümlere verilerin dağıtımı, okumanın ve yazmanın yapılacağı düğüm adedi gibi veri tutarlılığı ile ilgili ayarlar performans beklentisine göre değiştirilebilmektedir. Cassandra yoğunluğuna göre gelen işlem isteklerini çemberdeki diğer düğümlere dağıtabilme yani yük dengeleme yeteneğine sahiptir. Bu özelliğinden dolayı erişim öncesinde harici bir yük dengeleme uygulamasına gerek bulunmamaktadır. NoSQL mimarisine sahip olmasına rağmen SQL veri tabanlarına benzerlikler göstermektedir. Bu sebeple SQL tecrübesi olan

geliştiriciler, NoSQL'e daha rahat geçiş yapabilmektedir [81]. Diğer NoSQL veritabanlarından; MongoDB, Couchbase ve Hbase ile yapılan performans testlerinde Cassandra performansı ile öne çıkmaktadır [99]. Bu sebeplerle uygulama veritabanı olarak Cassandra tercih edilmiştir.

4.1.9. Çalıştırma, geliştirme testi ve ön izleme

Uygulama kodlarının derlenmesi sırasındaki rutin işlemleri hızlandırabilmek için Make programı basit ve esnek yapısı ile çözüm sağlamaktadır. Derleme, test, dağıtım veya herhangi bir amaç için sıralı olarak yapılması gereken işlemler tanımlı Make veya kabuk (shell) komutları ile betik haline getirilebilmektedir. Make her defasında tüm betiği çalıştırmak yerine sadece değişiklik olan dosyaları tespit eder ve dosyayı ilgilendiren blokları çalıştırmaktadır. Yüksek boyutta ve sıralı olarak karmaşık işlem basamakları olan projeler için verimli bir araç olarak tercih edilmektedir. Tez içerisinde dosya değişikliğine göre derleme, çalıştırma, konteyner oluşturma ve dağıtım amacı ile kullanılmıştır.

Web uygulama programlama arayüzünün gerçeklenmesi esnasında yol gösterici olarak sık sık testlere ihtiyaç duyulmaktadır. Postman, test edilecek arayüzün parametre ve adres gibi bilgilerini Swagger dosyası üzerinden tanıyabilmektedir. İstenilen adette web isteğini iterasyon ile test edebilmektedir. İstek sonuçlarını yazılan koşullara göre yorumlayabilmektedir. Bu konuda kolay kullanımı ve basit arayüzü sebebi ile Restful destekli arayüz test aracı Postman kullanılmıştır.

JavaScript dünyası içerisinde son derece hızlı gelişmeler yaşanmasından dolayı yeni çıkan standart ve teknolojilerin tüm tarayıcılar tarafından güncel olarak desteklenmesi mümkün olmamaktadır. NodeJS ile sunucu uygulaması için geliştirilen genel kütüphanelerin veya BabelJs gibi desteklenmeyen diğer yeniliklerin web tarayıcılarında kullanılabilmesi için ekleme, çevirme ve paketleme gerekmektedir. En temel ihtiyaç olarak JavaScript veya CSS dosyalarının tek dosyada sıkıştırılmış olarak toplanabilmesi gerekebilmektedir. JavaScript'in Make uygulaması olarak adlandırabileceğimiz Webpack, tüm bu işlemleri belirlenen adımlara göre yapabilmektedir. Tez içerisinde React ile kullanıcı arayüzü geliştirme esnasında anlık derleme ve ön izleme yeteneklerinden dolayı tercih edilmiştir.

4.2. MSTOS'nin Gerçeklenmesi

MSTOS'nin gerçeklenmesi ile ilgili yapılan işlemler aşağıdaki bölümlerde adım adım açıklanmıştır. Bu bölümde sadece MSTOS'nin gerçeklenmesi ve firmalara kullanım için hazır hale getirilmesi sürecine kadar anlatılmıştır. Firmaların, MSTOS'ne dahil olmaları ile ilgili teknik konular ve adımları, 5. bölümde Örnek Bir Uygulama ve Simülasyon altında anlatılmıştır. Gerçeklenme iş akışı şu şekilde oluşmuştur.

A) Arayüzlerin gerçeklenmesi
    1) Swagger aracı ile web uygulama programlama arayüzünün oluşturulması
    2) Go programlama dili ile web uygulama programlama arayüzünün gerçeklenmesi
    3) Yönetim panelinin gerçeklenmesi
B) Yayına alma

1) Docker yazılımı ile konteyner kalıplarının oluşturulması

2) Docker Compose aracı ile konteynerlerin orkestrasyonu

3) Rancher yazılımı ile konteynerlerin dağıtımı ve yönetimi

C) Yardımcı uygulamalar

1) Banka servisinin gerçeklenmesi

2) E-posta gönderim servisinin gerçeklenmesi

4.2.1. Web uygulama programlama arayüzünün gerçeklenmesi

Şekil 4.2. Swagger kullanıcı arayüzü

Arayüzün taslağını oluşturmak için Swagger yazılımının Şekil 4.2'de görünen bulut bilişim tabanlı çevrimiçi editörü kullanılabilir veya açık kaynak kodlu olması sebebi ile istenilen sistem üzerinde çalıştırılabilir [100]. Editor, tanımlamaların yapıldığı ve

ön izlemenin takip edildiği olmak üzere iki bölümden oluşmaktadır. Tanımlamalar YAML veya JSON formatında Tablo 4.1'de gösterilen ana etiketlerin altında gruplandırılmalıdır. Daha anlaşılabilir bir dokümantasyon için her teknik tanıma gösterilen özen kadar açıklama alanı için gösterilmelidir.

Tablo 4.1. Swagger tanım dosyası etiketleri

Etiket	Açıklama
info	Projenin adı, tanımı ve versiyon numarası gibi genel bilgiler
securityDefinitions	Erişime uygulanacak olan güvenlik protokolü ve bilgileri
paths	URL'i, adı, açıklaması, metodu, güvenlik önlemi, gereken parametreler ve dönüş tipi bilgileri tanımlanarak oluşturulan uç noktaların (endpoint) listesi
definitions	Parametre ve dönüş tiplerinde kullanılması için modellenen nesneler listesi

Oluşturulan YAML dosyasına göre Swagger editörü üzerinden istenilen programlama dili için sunucu veya istemci başlangıç şablonu elde edilebilir. Tezdeki sunucu uygulamasının başlangıç şablonu için Go dili tercih edilmiştir. Kullanım amaçları ile birlikte uygulamanın URL listesi Tablo 4.2'de ve tanımlanan veri modelleri Tablo 4.3'de belirtilmiştir.

Tablo 4.2. MSTOS çekirdeğinin uç noktaları

Grup	Metot	URL	Açıklama
User	POST	/login	Üye girişi
User	GET	/me	Kullanıcı hesap bilgileri
Coin	POST	/coins	Jeton işlemleri
Cards	GET	/cards	Kullanıcının tanımlı kart listesi
Credit	POST	/transactions	Karttan ödeme talebi
Credit	POST	/fastpay	Hızlı satın alma
Credit	GET	/transactions	Ödeme listeleme

Grup	Metot	URL	Açıklama
Credit	PUT	/credits/{id}/verification	3D ödeme onayı

Tablo 4.3. MSTOS için tanımlanan modeller

Model Adı	Açıklama
Result	Temel işlem sonucu
Error	Temel işlem hatası
Auth	Üye girişi için gereken parametreler
User	Oturum açan kullanıcının hesap bilgileri
Coin	Jeton artırma/azaltma talebi için gereken parametreler
Cards	Kullanıcının tanımlı kart listesi
Credit	Karttan ödeme talebi için gereken parametreler
CreditResult	Ödeme talebine bankanın cevabı
Verification	3D ödeme onayı için gereken parametreler

Swagger üzerinden indirilen sunucu arayüz şablonunda, tanımlanan nesneler ve belirlenen metotlara göre go dilinde fonksiyon isimleri hazır olarak gelmektedir. Projenin başlangıç dosyası "main.go" isimli dosyadır. Diğer dosyaları "go" isimli klasör içerisindedir. Tablo 4.3'de verilen nesneler ve özellik tanımları "models.go" dosyası içerisindedir. Arayüz için uç noktaların hangi fonksiyon ile eşleştirileceği "routers.go" içerisinde kayıtlıdır. Ayrıca bu dosya içerisinde orta katman olarak çalışması istenilen fonksiyonlar tanımlanabilir. MSTOS çekirdek yazılımı orta katmanında, işlem kayıtları tutulmakta ve yetki kontrolü yapılmaktadır. Yetki kontrolü için JWT kodu teyit edilmektedir. Tablo 4.4 içerisinde bulunan kod bloğu, "jwt-go" [101] kütüphanesi aracılığı ile doğrulama yapmaktadır. Eğer doğrulama başarısız olursa veya kullanıcının JWT kodu bulunmuyorsa, öncelikle Tablo 4.3'de bulunan üye girişi ile JWT kodu elde edilmelidir. Veritabanı bağlantısı için "gocql"

[102] kütüphanesi kullanıldı. Bu kütüphane Cassandra veritabanı kullanımı için gerekli işlemleri sunmaktadır. Bir istek boyunca veritabanı bağlantısının, açık tutulan tek bağlantı (singleton) üzerinden yürütülmesine dikkat edildi.

Tablo 4.4. JWT doğrulamasının yapıldığı kod bloku

```
token, _ := jwt.ParseWithClaims(jwt_token, &User{},
func(token *jwt.Token) (interface{}, error) {
if _, ok := token.Method.(*jwt.SigningMethodHMAC); !ok {
      return nil, errors.New("Unexpected signing method.")
      }
      return []byte(os.Getenv("SECRET")), nil
})
var claim *User
var ok bool
if token != nil {
      claim, ok = token.Claims.(*User)
}
if ok && token.Valid {
      ctx := context.WithValue(r.Context(), "user", claim)
      inner.ServeHTTP(w, r.WithContext(ctx))
} else {
      WriteUnauth(w, errors.New("Invalid token."))
}
```

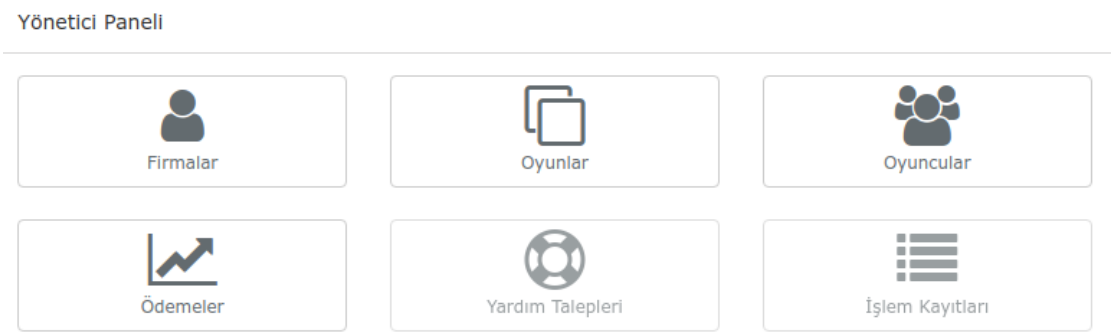
Son kullanıcılara sunulacak olan web arayüzü için HTML, CSS ve Javascript kodlarının da, bu HTTP sunucusu üzerinden dağıtımı sağlandı. Yazılım derlemesi için "Make" kullanıldı. Otomatik derleme için "Makefile" ile oluşturulan gerekli yapılandırma Tablo 4.5'de sunulmuştur. Derlenen yazılım, Linux tabanlı "Alpine" [103] isimli işletim sistemi üzerinde sanallaştırılacağı için bu dosyadaki parametreler önemlidir.

Tablo 4.5. Derleme için kullanılan “Makefile” içeriği

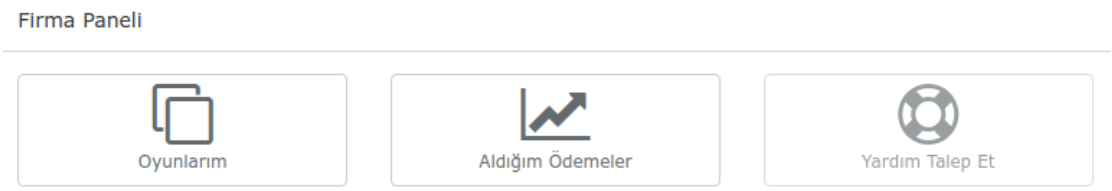
```
PWD:=$(shell pwd)
PACKAGE:=$(shell basename $(PWD))
SOURCES:=$(shell find $(PWD) -name '*.go')
BUILD:=$(PWD)/bin
BINARY:=$(BUILD)/$(PACKAGE)
GO_BUILD_ENV:=GOOS=linux GOARCH=amd64 CGO_ENABLED=0

$(BINARY): $(SOURCES)
	mkdir -p $(BUILD)
	$(GO_BUILD_ENV) go build -v -o $@ main.go
```

4.2.2. Yönetim panelinin gerçeklenmesi

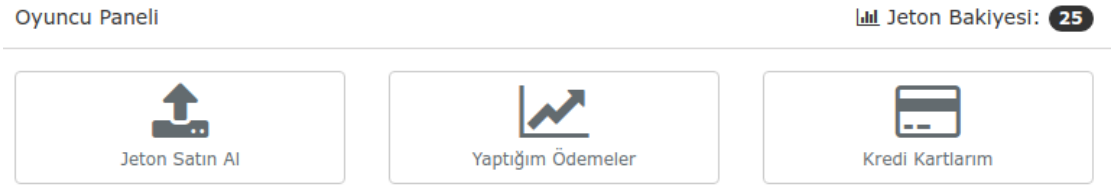
Oyuncu ve firmaların; üyelik açabilmeleri, hesaplarını yönetebilmeleri ve ödemelerini kontrol edebilmeleri için gerekli arayüz hazırlanmıştır. Web tabanlı hazırlanan MSTOS panelinin gerçeklenmesi için Laravel isimli PHP kütüphanesi kullanılmıştır. MSTOS yöneticisi, firma ve oyuncu olmak üzere üç farklı rol için üç farklı panel hazırlanmıştır. Bootstrap arayüz kütüphanesi ile oluşturulan kullanıcı arayüzünden, tanımlı yetkilere göre tanımlı yetkilere göre her rol farklı işlemler yapabilmektedir. Firma ve oyuncular, yen üyelik açma, mevcut üyelik parolalarını sıfırlama gibi temel işlemleri de bu panel üzerinden yapmaktadırlar. MSTOS yöneticisine ait tam yetkili arayüz Şekil 4.3‘de, firma için olan arayüz Şekil 4.4‘de ve oyuncu için oluşturulan arayüz Şekil 4.5‘de sunulmuştur.



Şekil 4.3. MSTOS yönetici paneli



Şekil 4.4. MSTOS firma paneli



Şekil 4.5. MSTOS oyuncu paneli

Oluşturulan panel arayüzü için MariaDB veritabanı kullanılmıştır. Oyuncu ve firma bilgileri burada saklanmakta ve belirli aralıklarla MSTOS çekirdeğinin işlem yaptığı Apache Cassandra veritabanı ile eşitlenmektedir. Bu sayede çok sayıda işlem yükünü taşımak zorunda olan MSTOS çekirdeğinin veritabanı ile tanımlama işlemlerinin tutulduğu panel veritabanı, yoğunluk anlarında birbirlerini etkilememektedir.

4.2.3. Konteyner oluşturma, dağıtım ve yönetimi

Hazırlanan MSTOS çekirdeği ve web kullanıcı arayüzü ile Docker yazılım kalıbı oluşturulması gerekmektedir. Bunun için Tablo 4.6'de bulunan içerik ile "Dockerfile" isimli dosya oluşturulmuştur. Bu dosya içerisinde "FROM" kısmı, temel alınacak kalıp dosyasını belirtmektedir. "COPY" komutu, belirtilen dosya ve klasörleri kalıp içerisine kopyalamaktadır. "EXPOSE" komutu, çalıştırılacak konteynere erişim için açılması gereken port numaralarını tanımlamaktadır. Son olarak ise "CMD" komutu, belirtilen dosyayı çalıştırmaktadır [104]. MSTOS çekirdeği için Error: Reference source not found. bölüm altında derlenen uygulama "/opt/handler" olarak yüklenmesi sağlanmıştır. Bu dosyaya göre kalıp oluşturulması için "docker build -t **etiket-ismi** ." komutu kabuk üzerinde çalıştırılmalıdır. "-t" parametresi ile kalıba özel bir isim verilmelidir. Örnek verilen komutta, kalın olarak tanımlanan "etiket-ismi" kısmına yazılmalıdır [105]. Verilen etiket ismine göre sonraki dağıtım ve ölçeklendirme işlemlerinde, bu kalıp kullanılabilecektir.

Tablo 4.6. MSTOS çekirdeği için kalıp oluşturma kodları

```
FROM alpine:latest

COPY bin/mstos-core /opt/mstos-core
COPY .env /opt/.env
COPY api-docs /opt/api-docs

EXPOSE 80

CMD /opt/mstos-core
```

Oluşturulan kalıpların, konteyner olarak çalıştırılması ve birbirleri ile belirlenen kurallara göre iletişime geçebilmeleri için orkestrasyonu yapılması gerekmektedir.

Bunun için Tablo 4.7‘de verilen yapılandırma gibi tanımlamalar yapılmalıdır. MSTOS çekirdeği, birden fazla sayıda konteyner içerisinde sanallaştırılacağı için isteklerin karşılanması ve yönlendirilmesi gereken bir yük dengeleyicisine ihtiyaç vardır. HAProxy, yük dengeleyicisi olarak tanımlanmıştır. Bu yapılandırmalar, her dizin için farklı “docker-compose.yml” dosyası içerisine kaydedilmelidir. İlgili dizindeki kabukta çalıştırılacak “docker-compose up” komutu ile tanımlanan birden fazla kalıp, belirlenen yapılandırmaya göre tek konteyner olarak çalıştırılacaktır. Bu yöntem, ölçeklendirme ve yönetim konusunda uğraştırıcıdır [106].

Tablo 4.7. MSTOS çekirdeğinin yük dengeleyicisi ile orkestrasyonu

```
version: '2'
services:
 mstos:
  image: mstos-core
  environment:
   TCP_PORTS: '80'
 lb:
  image: rancher/lb-service-haproxy:latest
  ports:
  - 80:80/tcp
```

Hızlı konteyner yönetimi için Rancher yazılımı kullanılmaktadır. Görsel kullanıcı arayüzü sayesinde, daha hızlı ve kolay yönetim yapılabilmektedir. Rancher yazılımı ile yönetilecek ve Docker sanallaştırma motorunu çalıştıracak ana sunucular, kolayca Rancher’ın havuzuna eklenebilmektedir. Rancher içerisine “docker-compose.yml” isimli orkestrasyon dosyaları aktarılabilmektedir. Bu dosyalara göre oluşturulan konteynerlerin, Şekil 4.6‘deki görsel arayüz üzerinden adetleri artı ve eksi butonları üzerinden hızlıca değiştirilebilmektedir.

Şekil 4.6. Rancher konteyner ölçekleme arayüzü

MSTOS çekirdeği ve diğer yazılımlar için oluşturulan orkestrasyon dosyalarına göre sanallaştırma yapılmıştır. Fakat, veritabanı olarak kullanılan Cassandra, Docker içerisinde dinamik olarak ölçeklenirse performans kayıpları yaşayabilmektedir [107]. Bu sebeple, orkestrasyon içerisine dahil edilmemiştir. Cassandra, sahip olduğu halka topolojisine dayalı otomatik ölçeklendirme yeteneği sayesinde, ana sunucular üzerinde çalıştırılmıştır. Cassandra, Docker içerisinde çalıştırılmak istenilirse, **MAX_HEAP_SIZE="128M"** ve **HEAP_NEWSIZE="24M"** parametreleri ile mutlaka Java Sanal Makinesi'nin belleği kısıtlanmalıdır [108]. Yoksa ana sunucular üzerinde sistem belleği kalmayacaktır. Sonuç olarak sistemin komple tıkanmasına sebep olacaktır.

4.2.4. Banka ve e-posta gönderim servislerinin gerçeklenmesi

MSTOS'nin gerçeklenmesi esnasında, kredi kartı ile çekimin simüle edilebilmesi için sanal pos arayüzü hazırlandı. Bu servisin Go programlama dilinde yazılan HTTP istekleri için eklenen uç noktaları Tablo 4.8'da verilmiştir. Bu uç noktalara yapılacak isteklerde kullanılması gereken nesne isimleri ise Tablo 4.9'da bulunmaktadır. Bu

servisin veri transfer formatı JSON'dır. Servise erişim yetkisinin kontrolü ve adres tanımlaması için Kong isimli arayüz ağ geçidi kullanılmıştır. Kong, MSTOS ödeme isteklerini gerekli yalıtım ile "http://banka/" adresi üzerinden aktarmaktadır. Sanal pos, gelen ödeme isteğine göre, 3.1.3. bölümde anlatılan servisi ile Tablo 4.10'de bir örneği sunulan 3D kodunu e-posta aracılığı ile göndermektedir. Bu esnada ödeme bekletilmektedir. Kullanıcı 3D kodunu doğru gönderdiği anda, ödeme onaylanmaktadır.

Tablo 4.8. Sanal posun uç noktaları

Grup	Metot	URL	Açıklama
Ödeme	GET	/	Tüm ödemelerin listesi
Ödeme	POST	/	Yeni ödeme isteği
Ödeme	GET	/{islem-no}	{islem-no} alanına yazılan işlemin detayı
3D Onayı	PUT	/{islem-no}/verification	{islem-no} alanına yazılan işlemin 3D onayı

Tablo 4.9. Sanal pos için tanımlanan modeller

Model Adı	Açıklama
Card	Ödeme alınacak kart bilgisi
Payment	Alınacak ödeme bilgileri
Verification3D	3D kodu
Transaction	Ödeme işlem sonucu

Tablo 4.10. 3D kodu taşıyan e-postalardan bir örnek

Gönderen: E-Posta Servisi
Konu: Ödeme İşleminiz

Merhaba,

2017-05-20 17:30 tarihli ödeme işleminiz ile ilgili detaylar aşağıdadır.

İşlem no: 52716
Kart no: 1234-1234-1234-1234
Tutar: 15.00 TL
3D Onay kodu: 810547

İyi günler.

Web uygulama programlama arayüzünü sunan HTTP servisi, konteyner içerisinde çalışabilir olarak tasarlanmış ve paketlenmiştir. İhtiyaca göre gereken adet konteyner çalıştırılabilmektedir. Çok sayıda konteynere erişim sağlanabilmesi için hepsinin önünde HAProxy yük dengeleyicisi çalışmaktadır. Veritabanı olarak Mariadb kullanıldı. Docker içerisinde problemsiz olarak işletim sistemi seviyesinde sanallaştırılmaktadır. Oluşabilecek iş yüküne göre MariaDB'nin çalıştırıldığı kümeye, yeni düğümler eklenebilir. Bu sayede hızlıca ölçeklendirmesi yapılabilmektedir. MariaDB'nin küme yapısı, birincil (master) ve ikincil (slave) olarak sınıflandırılan konteynerlerden oluşmaktadır. Birincil konteynerler, veri yazma ve okuma yapabilmektedir. İkincil konteynerler ise sadece okuma yapabilmektedir. Bu yüzden, uygulama içerisinden veri tabanına iki adet bağlantı açılmaktadır. Veri yazmak için birincil konteynerler, okumak için ise ikincil konteynerler kullanılmaktadır. Bu sayede dengeli iş yükü dağıtılmış olmaktadır.

E-posta gönderim servisi Go programlama dilinde yazılmıştır. E-posta gönderim isteklerinin toplanması ve asenkron olarak gönderilebilmesi için NSQ mesaj kuyruk sistemi kullanılmıştır. Web uygulama programlama arayüzüne gelen istek, NSQ üzerinde depolanır. Bu isteklerin tamamlanması için isteği yapan işlem beklemek zorunda kalmaz. Diğer oluşturulan bir servis ise, NSQ mesaj kuyruğunda biriken işlemleri tüketmek için çalışmaktadır. Bu tüketme işlemini yapan tüketici konteyner ve gelen istekleri toplayan konteyner, birbirinden bağımsızdır. Böylece e-posta gönderimi asenkron olarak yapılabilmektedir. Bu servise erişim, yine Kong arayüz ağ geçidi aracılığı ile "http://eposta/" adresi üzerinden olabilmektedir. Kong tarafından yönlendirme, yalıtım ve oturum güvenliği kontrol edilmektedir. Servis üzerindeki yoğunluğa göre mesaj kuyruğunu oluşturan, üretici veya tüketici konteyner adetleri hızlıca değiştirilebilmektedir. Bu sayede sistem belleği, ihtiyaç duyulan işleme tahsis edilmiş olmaktadır.

Tablo 4.11. E-posta gönderim servisi uç noktası

Grup	Metot	URL	Açıklama
Gönderim	POST	/	Yeni e-posta gönderimi

Tablo 4.12. E-posta gönderim servisi için tanımlanan model

Model Adı	Açıklama
Mail	E-posta gönderim bilgileri

5. ÖRNEK BİR UYGULAMA VE SİMÜLASYON

MSTOS gerçeklenmesi ardından, işleyişin tam olarak açıklanabilmesi için bu bölüm üzerinden örnek bir uygulama ile anlatım yapılmıştır. Örneklendirme, firma ve oyuncu açısından ayrı olarak ele alınmıştır. Firma açısından ise örnek uygulama ve MSTOS'ne dahil olma süreçlerinden de bahsedilmiştir. MSTOS çekirdeğinin sunduğu web uygulama programlama arayüzü, direk olarak HTTP ve JSON desteği sunan herhangi bir platform veya programlama dili ile kullanılabilir. Fakat örnek uygulama için PHP ve JavaScript dilleri ile web platformu tercih edilmiştir. Bu platform ve diller için de örnek kütüphaneler oluşturulmuştur. Bölüm sonunda ise MSTOS çekirdeğine çok sayıda oyun sunucusundan istek gelmesi durumunun simülasyonu yapılmıştır. Yapılan simülasyonda, artan istek adetleri ve başarı oranları tablo olarak sunulmuştur.

5.1. Örnek Uygulama

Uygulamada iki arayüzden oluşmaktadır. İlki oyun arayüzüdür ve MSTOS'nin sunduğu servisleri kullanmaktadır. İkincisi ise MSTOS'nin oyuncuya sunduğu özelliklerin olduğu arayüzdür. Belirlenen temel işlemlerin gösterimine önem verildiği için görsel tasarım basit tutulmuştur. Uygulama arayüzü, HTML ile oluşturulmuş bir sayfadır. Oyun arayüzü Şekil 5.1'de, MSTOS arayüzü ise Şekil 5.2'de sunulmuştur. Oyun sunucusunda PHP dilinde yazılmış, oyun arayüzünden gelen istekleri karşılayan bir yazılım çalışmaktadır. Bu yazılım, herhangi bir firmanın oluşturabileceği basit

yapıda olması için MSTOS PHP kütüphanesi haricinde farklı bir kütüphane veya teknik kullanılmamıştır.

Bu oyun içerisinde anlatım için temel alınan ve Şekil 5.1'de ekran görüntüsü sunulan üç işlem şunlardır;

1. Oturum açma
2. Jeton kullanma
3. Hızlı satın alma

Şekil 5.1. Örnek uygulamada oyun arayüzü

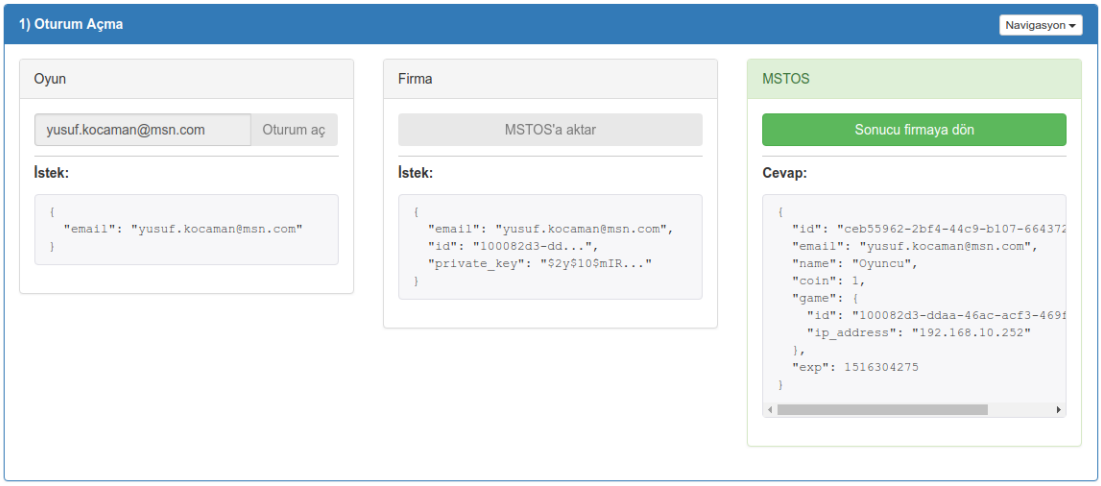
Şekil 5.2. Örnek uygulamada MSTOS arayüzü

Bu işlemler kısaca şu şekilde özetlenebilir:

1. Oturum açma işleminde, istek oyun arayüzünden firmanın oyun sunucusuna gelir, oyun sunucusu MSTOS kütüphanesi ile MSTOS çekirdeğine isteği aktarır. MSTOS çekirdeğinden, işlem onaylanması durumunda tek kullanımlık bir şifre ile oturum anahtarı dönülür. Bu anahtar, oyun sunucusu üzerinden oyuncunun tarayıcısına aktarılır ve çerezlerde tutulur. Sonraki işlemlerinde de şifre ile oturumunu doğrulatması beklenir.
2. Jeton kullanma isteğinde, önceden oyuncuya bırakılan tek kullanımlık şifre ile istek arayüzünden oluşturulur. Oyun arayüzünden MSTOS kütüphanesi ile MSTOS çekirdeğine istek iletilir. İşlem sonucunda, MSTOS çekirdeği tarafından tekrar tek kullanımlık bir şifre ve işlem sonucu, oyun arayüzüne geri dönülür. Oyuncu arayüzü elindeki tek kullanımlık şifreyi, sonraki istek için tekrar oyuncunun arayüzünde saklar. İşlem sonucu ayrıca oyun sunucusuna da iletilir.
3. Hızlı satın alma işleminde, yine aynı şekilde oyun arayüzü üzerinden MSTOS çekirdeğine iletilmektedir. Hızlı satın almada satın alınabilinecek jeton adedi 1'dir. Sınırlandırma ile ilgili detaylı bilgi 3.1.4. Oyun anında jeton satın alınması bölümünde açıklanmıştır. Oyuncu isterse daha fazla adet için direkt MSTOS paneli üzerinden satın alma yapabilir.

Bu işlemlerin akışı Şekil 3.6'da sunulmuştur. Gerçeklenme için kullanılan kodların bir kısmı, bu başlıklara göre gruplandırılarak bir sonraki bölümde paylaşılmış ve anlatımı yapılmıştır. Yukarıda bahsedilenlerin haricinde puan işlemlerinde,

MSTOS'nin herhangi bir müdahalesi olmadığı için oyuna dahil edilmemiş ve anlatımı yapılmamıştır. Ayrıca ek olarak Şekil 5.3 üzerinde sunulan arayüz oluşturulmuştur. Bu arayüz, oyun, firma ve MSTOS arasındaki iletişimin simülasyonunun anlatımı için oluşturulmuştur. Adım adım olarak oyun arayüzünden başlatılan bir isteğin, firmaya ulaşan veri paketi ve firmanın MSTOS'ne aktarımı gibi detayların gösterimi amaçlanmıştır.

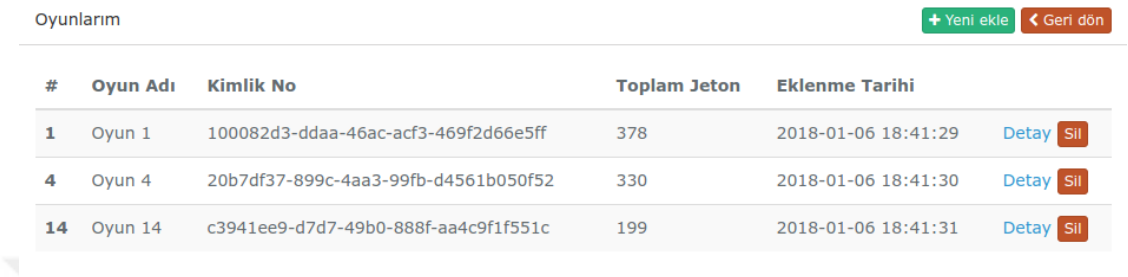


Şekil 5.3. Oyun, firma ve MSTOS arasındaki iletişimin simülasyonu

5.2. Firma Açısından Örnek Kullanım

Herhangi bir oyun firmasının MSTOS altyapısını kullanabilmesi için öncelikle MSTOS üyeliği açması gerekmektedir. Açılan üyelik ile hızlıca MSTOS paneli üzerinden, oyun adı ve URL bilgisi ile oyun tanımlanabilmektedir. Eklenen her oyun için MSTOS tarafından benzersiz "Kimlik no" ve "Gizli anahtar" oluşturulur. Bu

bilgiler, gerçeklenme esnasında oyunun kimlik doğrulaması için kullanılacaktır. Firmanın oyunlarını yönetebildiği ekran Şekil 5.4'de sunulmuştur.



Şekil 5.4. Bir firmanın örnek olarak oyun listesi

Oyun tanımın yapılmasının ardından gerçeklenme işlemlerine başlanılmalıdır. Oyun ile MSTOS çekirdeği arasındaki bağlantının kurulması gerekmektedir. MSTOS'nin sunduğu web uygulama programlama arayüzüne, HTTP ve JSON desteği sunan istenilen programlama dili ve platform üzerinden bağlantı kurulabilir. Fakat 5.1. Örnek Uygulama bölümünde anlatılan tasarıma uygun olarak iki ayrı kütüphane oluşturulmuştur. Oluşturulan kütüphaneler, 4.2. MSTOS'nin Gerçeklenmesi bölümünde anlatılan web programlama arayüzüne göre oluşturulmuştur. İlki oyun sunucusu üzerinde çalıştırılacak PHP kütüphanesi, ikincisi ise oyun arayüzünde çalıştırılacak olan JavaScript kütüphanesidir. Ayrıca bu kütüphanelerin, diğer diller için de oluşturulabilecek kütüphaneler için örnek ve temel olması amaçlanmıştır. Oluşturulan kütüphanelere göre örnek kodlar iki ayrı başlık altında verilmiştir.

5.2.1. Oyun programında MSTOS kullanımı

Oyun arayüzünün tasarımı, tamamen firmaların isteklerine göre oluşması gerektiği için örnek olarak temel işlemleri içeren bir sade bir arayüz tasarlanmıştır. Arayüz ile

oyun sunucusu arasında iletişim Ajax ile sağlanmaktadır. Oluşturulan MSTOS JavaScript kütüphanesini kullanabilmek için öncelikle HTML dosyasına tanımlanmalıdır. Sonrasında ise "mstos" nesnesinde tanımlı olan ilgili fonksiyonların çağrılması gerekmektedir. Sadece oturum açma işleminde ilk parametreye, oyun sunucusunda isteği karşılayacak olan PHP dosyasının adı yazılmalıdır. Diğer işlemler için çağrılan fonksiyonlarda adet ve benzeri parametreler kullanılmaktadır. Son parametre ise istek tamamlandığı zaman çalıştırılacak olan JavaScript fonksiyonu işaret etmektedir. Firma, bu sayede istek öncesi ve sonrasında oyun arayüzünde ihtiyacı olan işlemleri yapabilmektedir. Bir önceki bölümde de bahsedilen üç temel işlemin kodları, aşağıdaki listeye göre belirtilen tablolarda sunulmuştur.

1. Oturum açma: Tablo 5.1
2. Jeton kullanma: Tablo 5.2
3. Hızlı satın alma: Tablo 5.3

Tablo 5.1. Oturum açma için gereken JavaScript fonksiyonu

```
var eposta = document.getElementById("eposta").value;

mstos.login("login.php", eposta, function (data) {
    var bakiye = document.getElementById("mstos_bakiye");
    bakiye.innerText = data.coin;
});
```

Tablo 5.2. Jeton kullanma için gereken JavaScript fonksiyonu

```
var adet = document.getElementById("adet").value;

mstos.decCoin(adet, function (data) { //Jeton harcama
  var bakiye = document.getElementById("mstos_bakiye");
  bakiye.innerText = data.coin;
});
```

Tablo 5.3. Hızlı satın alma için gereken JavaScript fonksiyonu

```
var miktar = document.getElementById("miktar").value;

mstos.fastpay(miktar, function () {
  alert("Satın alma işlemi tamamlandı");
});
```

Oluşturulan kütüphane içerisinde 3D onayı ile jeton satın alma, oyuncunun güncel bilgisinin çekilmesi, 3D onayı bekleyen işlemlerin listelenmesi ve kodun gönderilmesi, oyuncu oturumunun kapatılması gibi temel fonksiyonlar tanımlanmıştır. Yukarıda sadece örnek olması için üç temel fonksiyon içeriği paylaşılmıştır. Oyun arayüzünün tasarlanması ve oluşturulması; oyunun içeriği, firmanın isteği, oyunun platformuna ve benzeri çok sayıda parametreye göre değişiklik gösterebileceği için belirli bir kalıp uydurulması mümkün olmamaktadır. Bu yüzden oluşturulan JavaScript kütüphanesinde, firmaya işlemlerinde kolaylık sağlayabilecek en geniş düzeyde tanımlamalar yapılmıştır. Fakat, Ajax ile alınan değerin ekranda gösterilmesi veya listelenmesi gibi işlemler konusunda destek sunulamamıştır. Yukarıda verilen kodların, oyun arayüzünde ilgili tetikleyicilere tanımlanması ile oyun arayüzü gerçeklenmesi tamamlanmış olmaktadır.

5.2.2. Oyun sunucusu üzerindeki MSTOS kullanımı

Oyun arayüzünden gelen istekler, MSTOS sunucusu tarafından karşılanmaktadır. Sadece oturum açma işleminde firmanın istekleri karşılaması, MSTOS kütüphanesi ile MSTOS çekirdeğine aktarması gerekmektedir. Bunun sebebi, oturum açma işlemlerinde firmanın istekten haberdar olabilmesi içindir. Tablo 5.4'de kullanılan "Kimlik no" ve "Gizli anahtar", MSTOS panelinden alınmalıdır. Oluşturulan örnek

oyun sunucusunda, oyun arayüzünden gelecek oturum açma isteklerini karşılayabilmesi için PHP sayfası oluşturulmuştur.

1. Oturum açma: Tablo 5.4
2. Jeton kullanma: İstek MSTOS sunucusu tarafından karşılanır.
3. Hızlı satın alma: İstek MSTOS sunucusu tarafından karşılanır.

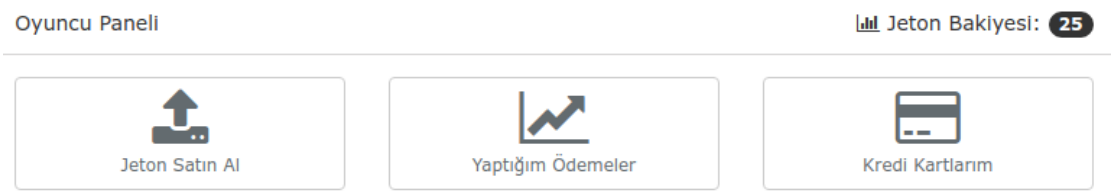
Tablo 5.4. Oturum açma için oluşturulan “login.php” içeriği

```
require_once 'mstos.php';
if (empty($_POST['email']) == false) {
  $mstos->login(
 '100082d3-ddaa-46ac-acf3-469f2d66e5ff', //Kimlik no
 '$2y$10$mIR6AYzlV8WdzCb2QTcQreo8eAEHe5', //Gizli anahtar
    $_POST['email'] //Oyuncunun e-posta adresi
  );
  echo $mstos->getResponse();//Cevabı dön
} else {
  echo 'Eksik e-posta adresi';
}
```

Puan işlemleri, firma sorumluluğunda olduğu için herhangi bir örneklemesi yapılmamıştır. Bu bölüm içerisinde örnek kodu verilmeyen fakat kütüphane içerisinde tanımlanmış olan fonksiyonlar bulunmaktadır. Bunlar; kart ile jeton satın alma, oyuncunun güncel bilgisinin çekilmesi, 3D onayı bekleyen işlemlerin listelenmesi ve kodun gönderilmesi gibi ödeme sistemi için önemli fonksiyonlardır. Yukarıda sadece örnek olması için üç temel fonksiyon içeriği paylaşılmıştır. Verilen kodların, oyun yazılımına eklenilmesi ile oyun sunucusu üzerindeki gerçeklenme tamamlanmış olmaktadır.

5.3. Oyuncu Açısından Örnek Kullanım

Oyuncunun öncelikle, MSTOS paneli üzerinden üye olması gerekmektedir. Açılan üyelik ile hesabına jeton yükleyebilir. İhtiyacı halinde yeni satın alma işlemleri için kredi kartı bilgilerini tanımlayabilir. Önceden yaptığı ödemelerin, hangi oyuna ve miktarının dökümünü görebilmektedir. MSTOS oyuncu panelinin ekran görüntüsü Şekil 5.5'de sunulmuştur.

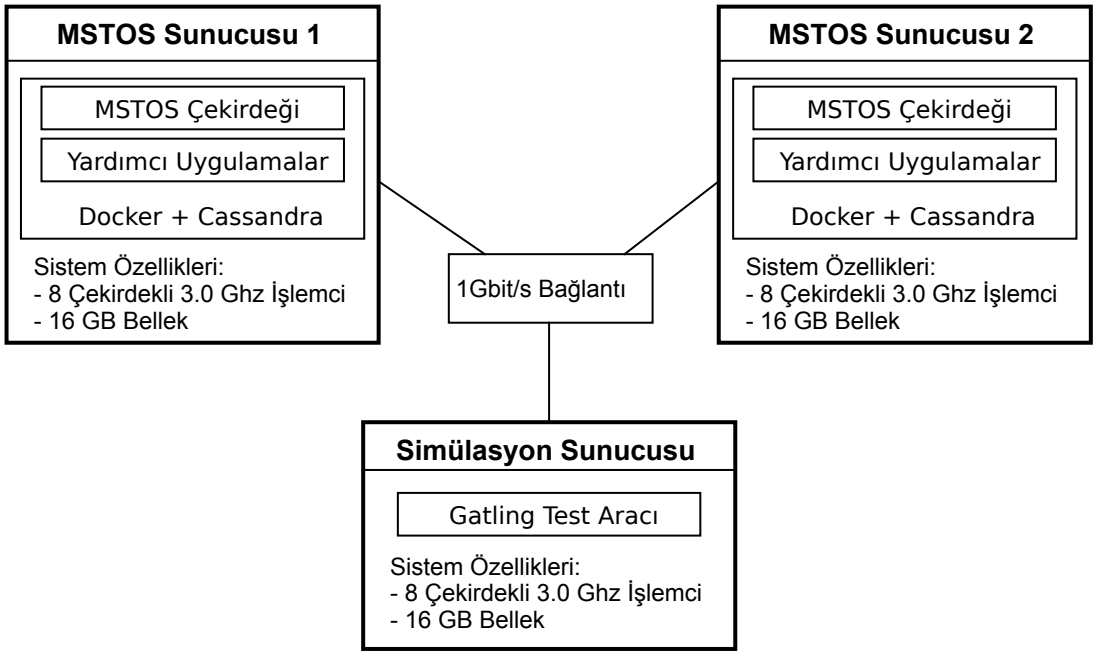


Şekil 5.5. Oyuncunun MSTOS paneli

Oyuncu, üyeliğinden sonra örnek oyun içerisinde kullanım yapabilmektedir. Firma, oyuncunun e-posta adresi üzerinden, oyuncu adına işlem yapabilmektedir. Oyuncunun oyun üzerindeki hareketleri, oyunun akışına göre belirlenebilir. Bu yüzden oyuncunun MSTOS üzerinden üyelik açması ve jeton satın alması haricinde, MSTOS ile etkileşime geçeceği bir süreci yoktur. İşlemlerin akışı, oyun ile oyuncu arasında şekillenmektedir. Örnek olarak tasarlanan oyunda ise web tabanlı oluşturulan HTML sayfayı, tarayıcısında açması gerekmektedir. Örnek oyunda oluşturulan, jeton kullanma ve hızlı satın alma işlemleri için ilgili butonlar ile işleyişi yönlendirebilmektedir.

5.4. Simülasyon

Temsili oyuncu davranışlarını simüle edebilmek ve hazırlanan yazılımın performansını test edebilmek için Gatling performans test aracı kullanılmıştır. Kütüphanesinde tanımlı fonksiyonlardan oluşan senaryo dosyasına göre istenilen oyuncu davranışları simüle edilebilmektedir. Simülasyon için kullanılan sistem şeması ve bileşenleri Şekil 5.6'da sunulmuştur.



Şekil 5.6. Simülasyon sistem şeması ve bileşenleri

Gatling yazılımı, MSTOS'nden ayrı olarak 3.0 Ghz frekansında 8 çekirdek ve 16 GB sistem belleğine sahip bir sunucu üzerinde çalıştırılmıştır. MSTOS sunucuları ve Gatling'in çalıştığı sunucu, birbirlerine 1 Gbit/s kapasitesindeki ağ ile bağlıdır. Test için sunucular üzerindeki yazılımsal limitlerin de kaldırılması veya yükseltilmesi gerekmektedir. İşletim sistemi güvenliği için uygulanan, aynı anda açık olabilecek en

fazla dosya adedi gibi kısıtlamalar, testin hatalı sonuçlanmasına sebep olmaktadır. Gatling tarafından gönderilen parametreler ile ideal yapılandırmaya sahip olunabilir [109]. Test sonucu yapılan tüm istekler, süreleri, sonuçları ve birbirleri ile olan çapraz karşılaştırmaları HTML tabanlı olarak rapor edilmektedir.

5.4.1. Simülasyon senaryosu ve uygulanması

MSTOS'nin performans testi için oluşturulan senaryoya göre toplam 5 adet HTTP isteği çalıştırılmaktadır. İlki ana sayfanın görüntülenmesi, sonrasında kullanıcı girişi, satın alma, hızlı satın alma, jeton sorgulama ve jeton harcama isteklerinden oluşmaktadır. Belirlenen kullanıcı bilgisi ile üyelik oturumu açılmaktadır. Oturum ile elde edilen şifre paketi, sonraki 3 istekte de doğrulama için gönderilmektedir. 60 saniye içerisinde ivmeli artış ile belirlenen miktarlarda kullanıcı gönderilmesi tanımlanmıştır. Ana sunucu adedi 1 iken bin, 10 bin, 20 bin, 30 bin, 40 bin ve 50 bin adet test edilmiştir. Yatay ölçeklendirme için olabildiğince konteyner adedi artırılmaya çalışılmıştır. Fakat sistem kaynaklarının yetersiz kalması sonucu dikey ölçeklendirme ile sunucu adedi 2'ye çıkarıldıktan sonra konteyner adetleri tekrar artırılmış ve 30 bin, 40 bin, 50 bin adet kullanıcı gönderilmiştir. Böylece MSTOS'nin yük altında hızlıca ölçeklenmesi yapılmış ve tepkisi test edilmiştir. Testin başarı yüzdesini belirlemek için yapılan isteklere dönülen cevaplara bakılmıştır. Gönderilen istekler içerisinde, hata veren veya zaman aşımını dolduranlar başarısız olarak değerlendirilmiştir. Beklenen yanıtları verenler ile toplam istek adedinin oranlanması ile başarı yüzdesi hesaplanmıştır.

Gatling kullanımı için öncelikle test senaryosunun, Scala dilinde hazırlanmış olan kütüphanesindeki fonksiyonlara göre yazılması gerekmektedir. Senaryoya göre hangi HTTP uç noktasına, hangi verilerin gönderileceği ve beklenen cevaplar tanımlanabilmektedir. Daha kolay bir yöntem olarak, Gatling yazılımı içerisinde gelen "bin/record.sh" dosyası ile tarayıcı hakaretlerine göre senaryo tanımlanabilmektedir. Bunun için tarayıcı web trafiğinin, Gatling vekil sunucusu üzerinden aktarılması yeterlidir. Bu işlemin sonunda, "user-files/simulations" dizinine tarif edilen senaryonun dosyası oluşmuş olacaktır. Senaryo dosyası içerisinde kullanıcıların, simetrik ya da ivmeli artan sayıda olması belirlenebilmektedir. Toplam simüle edilecek kullanıcı adedi tanımlanabilmektedir. Sağlıklı bir test için işletim sistemi ve ağ üzerindeki güvenlik kaynaklı oluşturulan limitlerin devre dışı bırakılması gerekmektedir. Belirlenen sunucu üzerinde Gatling yazılımının çalıştırılması ve senaryo dosyasının seçilmesi ile performans testi başlatılır ve beklenen oyuncu davranışları simüle edilmiş olur.

5.4.2. Simülasyon sonuçları

MSTOS'nin yapılan simülasyona göre performans sonucu Tablo 5.5'te verilmiştir. Tabloda belirtilen "Süre" kolonu, gönderilen tüm isteklerin tamamlanma süresini göstermektedir. Bir istek için zaman aşımı 60 saniyedir. "Başarı" kolonu ise toplam gönderilen istekler ile hatasız olarak, zaman aşımını geçmeyen ve beklenen cevabı veren isteklerin oranlanması sonucu elde edilen yüzdesel değerdir. Öncelikle sistemin karşılayabileceği kullanıcı adedi ile başlayıp, belirli aralıklar ile artırılmıştır. Başarı yüzdesi düştükçe, istekleri karşılayan konteyner adedi artırılmıştır. Konteyner

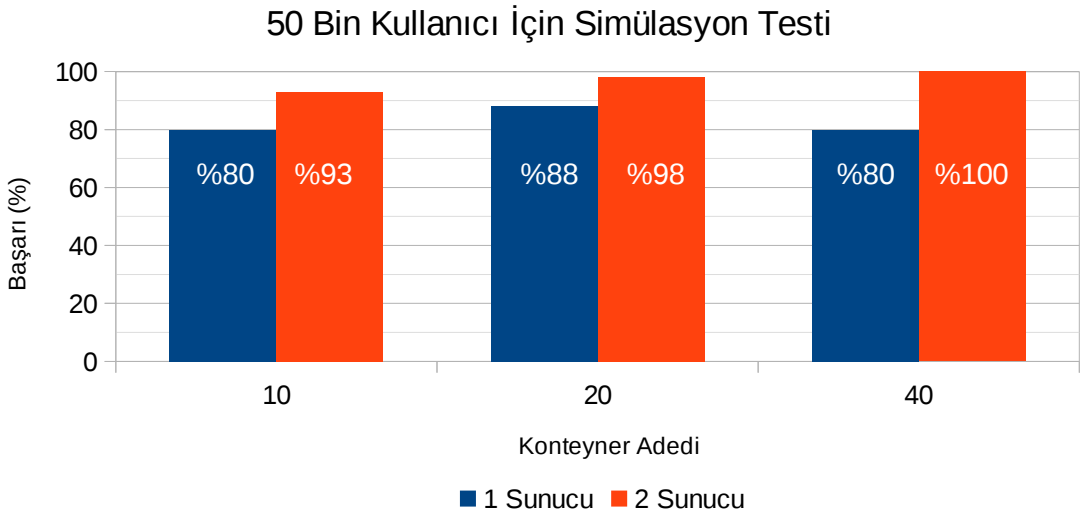
adedinin artmasına rağmen başarı yüzdesi artmadığı ve başarı yüzdesini ters orantılı olarak azalttığı 6. ve 7. testlerde görülmüştür. Buna istinaden yeni kaynak ekleyebilmek için işletim sistemi seviyesinde sanallaştırmanın yapıldığı sunucu sayısı 2'ye çıkartılmış ve tekrar test edilmiştir.

Tablo 5.5. MSTOS performans testi sonuçları ve ölçeklendirmenin etkisi

Test No	Kullanıcı adedi (1000)	Süre (saniye)	Sunucu adedi	Konteyner adedi	Başarı (%)
1	10	60	1	10	100
2	20	60	1	10	99
3	30	146	1	10	96
4	40	163	1	10	92
5	50	185	1	10	80
6	50	183	1	20	88
7	50	188	1	40	80
8	50	179	2	10	93
9	50	162	2	20	98
10	50	155	2	40	100

Yapılan testler içerisinde kullanıcı sayısı 50 bin olarak uygulanmıştır. Her kullanıcının 5 adet istek yapacağı senaryoya göre toplam 250 bin istek karşılanmaktadır. İsteklerin karşılanamama durumlarına göre konteyner adetleri artırılmıştır. Bu testlerde 10, 20 ve 40 konteyner adetlerinde ve sunucu adedine göre başarı yüzdesinin değişimini gösteren grafik Şekil 5.7'te sunulmuştur. Mavi renkli çubuklar 1 sunucuyu, kırmızı renkli çubuklar ise 2 sunucu adedini göstermektedir. Grafikten de görüleceği üzere ilk 50 bin kullanıcı testinde -5. test-, 1 sunucu vardır ve konteyner adedi 10'dur. Başarı yüzdesi %80 olduğu için konteyner adedi 20'ye

çıkartılmıştır ve başarı yüzdesi %88 olarak iyileştirilmiştir. 5. ve 6. testler ile yatay ölçeklemenin başarılı olduğu görülmüştür. Bir sonraki 7. testte ise konteyner adedi 40'a çıkartıldığında ise konteyner oluşturma maliyetinin kazançtan fazla gelmesi sebebi ile başarı yüzdesi tekrar %80'e düşmüştür. Yetersiz sistem kaynağından dolayı yatay ölçeklendirme kayıplara sebep olmuştur. Dikey ölçekleme ile eklenen yeni kaynaklar sonucunda testler tekrar edilmiştir. Son testlerde gözüktüğü gibi başarı yüzdesi %100'e ulaşmıştır. Testler sonucunda eklenen sistem kaynağı ile başarı yüzdesinin doğru orantılı olarak arttığı ve ölçeklendirmenin başarılı olduğu anlaşılmıştır.



Şekil 5.7. MSTOS performans testi sonuç grafiği

6. SONUÇLAR

Oyunculara yönelik ödeme sistemlerinde beklenmeyen bir anda, yüksek sayıda işlem isteği gelir. Bu durumu karşılamak için ödeme sisteminin ölçeklenebilir yapıda olması gerekir. Bu tezde tasarlanan ve gerçeklenen "Mikroservis Tabanlı Ödeme Sistemi" (MSTOS); ölçeklenebilir, hızlı yanıt veren ve oyun uygulamalarına kolay uyarlanabilir bir sistemdir. Ayrıca, bir ödeme sistemi olarak oyun firmalarına yönelik finansal kolaylıklar sağlayan, oyuncular açısından ise ödemelerini kolay ve hızlı yönetebildiği bir platform sunulmuştur. MSTOS, oyun firmaları için karmaşık ve güvenliğinin sağlanabilmesi gibi ödeme sistemi zorluklarını ortadan kaldıran bir hizmet olarak sunulmaktadır. MSTOS ile firmaların, oyunlar üzerinden kolayca satış işlemi yapabilmeleri ve yeni oyun üretiminde ihtiyaç duyacakları sermaye için de kaynak yaratabilmeleri mümkün olacaktır.

MSTOS'nin oyun anında duraksamadan hızlı ödeme yapılabilmesi, oyuncu – firma - MSTOS arasındaki jeton transferleri, hızlı kurulum ve kullanım gibi başlıca özellikleri gerçeklenmiştir. Firmalara, ücretsiz yayınladığı bir oyundan ödeme almak istemesi durumunda MSTOS ile kolayca oyunlarını güncelleyip hızlıca gelir elde etmesini sağlayan bir yapı sunulmuştur. Oyuncu açısından ise hesabındaki tanımlı jetonların, MSTOS'ne üye firmaların tüm oyunlarında geçerli olabilmesidir. Jetonlarını istediği oyunlar içerisinde harcayabilmektedir.

MSTOS yazılımı, oyuncu ve firmalara yönelik arayüzler gerçeklenmiştir. Firma ve oyun tarafından kullanılacak kütüphaneler ve uygulama programlama arayüzleri oluşturulmuştur. Ayrıca tüm sistemi deneyebilmek için oyun ve firma açısından denemelerde kullanılan basit uygulamalar gerçeklenmiştir.

Yapılan simülasyon testlerinde MSTOS'nin gelen istekleri, belirli bir yükten sonra doğal olarak karşılamakta yetersiz kaldığı görülmüştür. MSTOS, 20 bin kullanıcı ile %99 başarı gösterirken, 50 bin kullanıcı da %80'e kadar düşmüştür. Yetersiz kaldığı durumlarda, MSTOS hızlıca ölçeklendirilmiş ve simülasyon testleri tekrar edilmiştir. Başarı yüzdesi düştükçe, istekleri karşılayan konteyner adedi artırılarak istekleri karşılayan uygulama adedi de artırılmış ve böylece ve başarı yüzdesi iyileştirilmiştir. Yüksek kullanıcı sayısında konteyner adedinin artmasına rağmen başarı yüzdesi artmadığı tersine düşmesine neden olduğu görülmüştür. Örneğin 50 bin kullanıcı için 20 konteyner %88 başarı sağlarken 40 konteyner ile başarı ancak %80 olmuştur. Bunun nedeni konteyner işleminin yarattığı ek işlem yükünün getirdiği faydadan daha fazla olmasıdır. Konteyner artırmanın yararı olmadığı durumda dikey ölçekleme yapılmalıdır. Simülasyonda sunucu sayısı dikey ölçeklendirme ile 2'ye çıkartılması ile 50 bin kullanıcıda, başarı oranı tekrar %100'e ulaşmıştır. Testlerin sonucunda, beklenmeyen bir anda oluşabilecek yüksek sayıda işlem isteklerini, hızlıca ölçeklendirilebilmesi sayesinde karşılayabilir hale geldiği görülmüştür.

MSTOS mimarisi içerisinde tasarlanan ancak henüz gerçekleştirilmeyen konular olarak; avans sistemi, oyuncular arasında jeton transferi, detaylı hesaplama, alternatif

ödeme alma yöntemleri ve oyuncu paketleri vardır. Sonraki çalışmalarda, önerilen tasarımının tümünü gerçekleyen bir ödeme sistemini elde etmeyi planlıyoruz.

7. KAYNAKLAR

[1] “Steam: Game and Player Statistics”. http://store.steampowered.com/stats/?l=turkish. Erişim Tarihi: 17 Eylül 2017

[2] “Steam Community :: Guide :: Steam Topluluk Temelleri ve Tanıtım”. http://steamcommunity.com/sharedfiles/filedetails/?id=440790789. Erişim Tarihi: 17 Eylül 2017

[3] “The complete server side solution for game developers | GameSparks”. https://www.gamesparks.com/product/. Erişim Tarihi: 17 Eylül 2017

[4] “Xsolla”. https://xsolla.com/en/. Erişim Tarihi: 17 Eylül 2017

[5] “Xsolla Documentation - Start Page”. https://developers.xsolla.com/index.html. Erişim Tarihi: 17 Eylül 2017

[6] “Uygulama içi satın alımlar ve bunların nasıl faturalandığı konusunda bilgi edinin”, *Apple Support*. https://support.apple.com/tr-tr/HT202023. Erişim Tarihi: 17 Eylül 2017

[7] “Android uygulamalarında uygulama içi satın alma işlemleri gerçekleştirme - Google Play Yardım”. https://support.google.com/googleplay/answer/1061913?hl=tr. Erişim Tarihi: 17 Eylül 2017

[8] “Oyuncu Kimliği ile oyun oynama - Google Play Yardım”. https://support.google.com/googleplay/answer/2954594?hl=tr. Erişim Tarihi: 17 Eylül 2017

[9] “Games - Documentation”. https://developers.facebook.com/docs/games/. Erişim Tarihi: 17 Eylül 2017

[10] “Game Payments - Documentation”. https://developers.facebook.com/docs/games_payments/. Erişim Tarihi: 17 Eylül 2017

[11] “About itch.io”, *itch.io*. https://itch.io/docs/general/about. Erişim Tarihi: 17 Eylül 2017

[12] “About Us | Western Union”. https://corporate.westernunion.com/index.html. Erişim Tarihi: 17 Eylül 2017

[13] “What’s a Western Union MTCN / tracking number? | Western Union”. https://www.westernunion.com/us/en/what-is-mtcn.html. Erişim Tarihi: 17 Eylül 2017

[14] “Discover SWIFT | SWIFT”. https://www.swift.com/about-us/discover-swift. Erişim Tarihi: 17 Eylül 2017

[15] “Swift Code & BIC Code for all Banks in the World”. https://www.theswiftcodes.com/. Erişim Tarihi: 17 Eylül 2017

[16] “FAQ - PayPal Credit”. https://www.paypal.com/us/webapps/mpp/paypal-credit/faq. Erişim Tarihi: 17 Eylül 2017

[17] “SSS - Bitcoin”. https://bitcoin.org/tr/sss. Erişim Tarihi: 17 Eylül 2017

[18] “Games on Facebook”. https://developers.facebook.com/docs/games/gamesonfacebook. Erişim Tarihi: 17 Eylül 2017

[19] “The World Wide Web”. http://www.biology.wustl.edu/class/www.html. Erişim Tarihi: 18 Eylül 2017

[20] R. Fielding ve J. Reschke, “Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing”. https://tools.ietf.org/html/rfc7230. Erişim Tarihi: 18 Eylül 2017

[21] “Uniform Resource Locators (URL)”. https://www.w3.org/Addressing/URL/url-spec.txt. Erişim Tarihi: 18 Eylül 2017

[22] “W3C HTML”. https://www.w3.org/html/. Erişim Tarihi: 18 Eylül 2017

[23] “Cascading Style Sheets”. https://www.w3.org/Style/CSS/. Erişim Tarihi: 18 Eylül 2017

[24] D. Flanagan, *JavaScript: the definitive guide*, 6th ed. Beijing ; Sebastopol, CA: O’Reilly, 2011

[25] “gnu.org”. https://www.gnu.org/gnu/linux-and-gnu.html. Erişim Tarihi: 18 Eylül 2017

[26] tutorialspoint.com, “Unix / Linux Quick Guide”, *www.tutorialspoint.com*. https://www.tutorialspoint.com/unix/unix-quick-guide.htm. Erişim Tarihi: 19 Eylül 2017

[27] T. Ylonen ve C. Lonvick, “The Secure Shell (SSH) Protocol Architecture”. https://tools.ietf.org/html/rfc4251#page-3. Erişim Tarihi: 19 Eylül 2017

[28] “YAML Ain’t Markup Language (YAML™) Version 1.2”. http://www.yaml.org/spec/1.2/spec.html. Erişim Tarihi: 19 Eylül 2017

[29] “GNU make”.

https://www.gnu.org/software/make/manual/make.html#Introduction. Erişim Tarihi: 19 Eylül 2017

[30] "Git - Book". https://git-scm.com/book/en/v2/. Erişim Tarihi: 19 Eylül 2017

[31] P. Mell ve T. Grance, "The NIST Definition of Cloud Computing", *Natl. Inst. Stand. Technol. Spec. Publ.*, 2011

[32] M. Hamdaqa, T. Livogiannis, ve L. Tahvildari, "A Reference Model for Developing Cloud Applications.", içinde *CLOSER*, 2011, ss. 98–103 https://pdfs.semanticscholar.org/d45c/6e091a29d8e7d0ebfb1c2a9bf9bb034688c3.pdf. Erişim Tarihi: 17 Eylül 2017

[33] "Rise of Mobile Backend as a Service (MBaaS) API Stacks". https://apievangelist.com/2012/06/03/rise-of-mobile-backend-as-a-service-mbaas-api-stacks/. Erişim Tarihi: 18 Eylül 2017

[34] "MobStac Raises $2 Million In Series B To Help Brands Leverage Mobile Commerce | TechCrunch". https://techcrunch.com/2014/01/07/mobstac-raises-2-million-in-series-b-to-help-brands-leverage-mobile-commerce/. Erişim Tarihi: 18 Eylül 2017

[35] "What is Linux?", *Linux.com | The source for Linux information*. https://www.linux.com/what-is-linux. Erişim Tarihi: 18 Eylül 2017

[36] M. Loukides, "What is DevOps?", *O'Reilly Radar*, 07-Haz-2012. http://radar.oreilly.com/2012/06/what-is-devops.html. Erişim Tarihi: 18 Eylül 2017

[37] "The Rise of DevOps". http://www.somic.org/2010/03/02/the-rise-of-devops/. Erişim Tarihi: 18 Eylül 2017

[38] R. Rosen, “Resource management: Linux kernel Namespaces and cgroups”, May-2013 http://www.haifux.org/lectures/299/netLec7.pdf. Erişim Tarihi: 18 Eylül 2017

[39] “Linux Containers - LXC - Introduction”. https://linuxcontainers.org/lxc/. Erişim Tarihi: 18 Eylül 2017

[40] “What is the difference between containerization (Docker) and virtualization (VMWare, VirtualBox, Xen)? - Quora”. https://www.quora.com/What-is-the-difference-between-containerization-Docker-and-virtualization-VMWare-VirtualBox-Xen. Erişim Tarihi: 24 Ekim 2017

[41] W. Felter, A. Ferreira, R. Rajamony, ve J. Rubio, “An updated performance comparison of virtual machines and Linux containers”, içinde *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2015, ss. 171–172

[42] “What is Docker”, *Docker*, 14-May-2015. https://www.docker.com/what-docker. Erişim Tarihi: 18 Eylül 2017

[43] “Overview of Docker Compose | Docker Documentation”. https://docs.docker.com/compose/overview/. Erişim Tarihi: 18 Eylül 2017

[44] “Docker Machine Overview”, *Docker Documentation*, 18-Eyl-2017. https://docs.docker.com/machine/overview/. Erişim Tarihi: 18 Eylül 2017

[45] “Overview of Docker Hub”, *Docker Documentation*, 18-Eyl-2017. https://docs.docker.com/docker-hub/. Erişim Tarihi: 18 Eylül 2017

[46] “Docker Swarm overview”, *Docker Documentation*, 18-Eyl-2017. https://docs.docker.com/swarm/overview/. Erişim Tarihi: 18 Eylül 2017

[47] A. Otto, “DOCKER, KUBERNETES, AND MESOS: COMPARED”, Rackspace https://www.socallinuxexpo.org/sites/default/files/presentations/SCALE_14_Docker_Kubernetes_Mesos_Compared_0.pdf. Erişim Tarihi: 18 Eylül 2017

[48] “Rancher Documentation”. http://rancher.com/docs/rancher/latest/en/. Erişim Tarihi: 18 Eylül 2017

[49] “HAProxy - The Reliable, High Performance TCP/HTTP Load Balancer”. http://www.haproxy.org/#desc. Erişim Tarihi: 18 Eylül 2017

[50] “Gatling Load and Performance testing - Open-source load and performance testing”, *Gatling Load and Performance testing*. http://gatling.io/. Erişim Tarihi: 19 Eylül 2017

[51] L. Lamport, “Time, Clocks and the Ordering of Events in a Distributed System”, *Microsoft Res.*, Ara. 2016 https://www.microsoft.com/en-us/research/publication/time-clocks-ordering-events-distributed-system/. Erişim Tarihi: 18 Eylül 2017

[52] R. Cleaveland ve S. Smolka, “Strategic Directions in Concurrency Research.”, *ACM Comput Surv*, c. 28, ss. 607–625, Ara. 1996

[53] “Concurrency is not parallelism - The Go Blog”. https://blog.golang.org/concurrency-is-not-parallelism. Erişim Tarihi: 18 Eylül 2017

[54] A. Silberschatz, “Operating System Concepts 9th edition, Chapter 4: Threads”, 2013 http://www-etud.iro.umontreal.ca/~dift2245/notes/2013/ch4.pdf. Erişim Tarihi: 18 Eylül 2017

[55] A. Davies, *Async in C# 5.0*. O'Reilly Media, Inc., 2012

[56] "Frequently Asked Questions (FAQ) - The Go Programming Language". https://golang.org/doc/faq. Erişim Tarihi: 18 Eylül 2017

[57] N. Esquenazi, "Asynchronous Processing in Web Applications, Part 2: Developers Need to Understand Message Queues", *CodePath*, 06-Oca-2013. https://blog.codepath.com/2013/01/06/asynchronous-processing-in-web-applications-part-2-developers-need-to-understand-message-queues/. Erişim Tarihi: 19 Eylül 2017

[58] "NSQ Docs 1.0.0-compat - Design". http://nsq.io/overview/design.html. Erişim Tarihi: 19 Eylül 2017

[59] I. Codesido, "What is front-end development?", *The Guardian*, 28-Eyl-2009 http://www.theguardian.com/help/insideguardian/2009/sep/28/blogpost. Erişim Tarihi: 18 Eylül 2017

[60] "Bootstrap from Twitter". https://blog.twitter.com/developer/en_us/a/2011/bootstrap-twitter.html. Erişim Tarihi: 18 Eylül 2017

[61] "Laravel - The PHP Framework For Web Artisans". https://laravel.com/. Erişim Tarihi: 02 Ocak 2018

[62] "22 answers: Why is JavaScript so popular? - Quora". https://www.quora.com/Why-is-JavaScript-so-popular. Erişim Tarihi: 19 Eylül 2017

[63] C. Ullman ve L. Dykes, *Beginning Ajax*. Indianapolis, IN: Wiley Pub, 2007

[64] "JSON". http://www.json.org/. Erişim Tarihi: 19 Eylül 2017

[65] J. Bradley, N. Sakimura, ve M. Jones, “JSON Web Token (JWT)”. https://tools.ietf.org/html/rfc7519. Erişim Tarihi: 19 Eylül 2017

[66] auth0.com, “JWT.IO - JSON Web Tokens Introduction”. http://jwt.io/. Erişim Tarihi: 19 Eylül 2017

[67] tutorialspoint.com, “Node.js Quick Guide”, *www.tutorialspoint.com*. https://www.tutorialspoint.com/nodejs/nodejs_quick_guide.htm. Erişim Tarihi: 19 Eylül 2017

[68] “01 - What is npm? | npm Documentation”. https://docs.npmjs.com/getting-started/what-is-npm. Erişim Tarihi: 19 Eylül 2017

[69] “webpack - Getting Started”. https://webpack.js.org/guides/getting-started/. Erişim Tarihi: 19 Eylül 2017

[70] *webpack: A bundler for javascript and friends. Packs many modules into a few bundled assets. Code Splitting allows to load parts for the application on demand. Through "loaders," modules.*. webpack, 2017 https://github.com/webpack/webpack. Erişim Tarihi: 19 Eylül 2017

[71] S. Clarke, “Measuring API Usability”, *Dr. Dobb’s*. http://www.drdobbs.com/windows/measuring-api-usability/184405654. Erişim Tarihi: 19 Eylül 2017

[72] S. M. Sohan, C. Anslow, ve F. Maurer, “A Case Study of Web API Evolution”, içinde *2015 IEEE World Congress on Services*, 2015, ss. 245–252

[73] S. Cholia, D. Skinner, ve J. Boverhof, “NEWT: A RESTful service for building High Performance Computing web applications”, içinde *2010 Gateway Computing Environments Workshop (GCE)*, 2010, s. 7

[74] U. Dekel ve J. D. Herbsleb, “Improving API documentation usability with knowledge pushing”, içinde *2009 IEEE 31st International Conference on Software Engineering*, 2009, ss. 320–330

[75] “What Is OpenAPI? - Swagger”. https://swagger.io/docs/specification/about/. Erişim Tarihi: 19 Eylül 2017

[76] “☆ What Is an API Gateway? - Quora”. https://www.quora.com/What-is-an-API-gateway. Erişim Tarihi: 19 Eylül 2017

[77] Mashape, “Open-Source API Management and Microservice Management”, *Kong*. http://getkong.org. Erişim Tarihi: 19 Eylül 2017

[78] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, ve L. Safina, “Microservices: yesterday, today, and tomorrow”, *ArXiv160604036 Cs*, Haz. 2016 http://arxiv.org/abs/1606.04036. Erişim Tarihi: 19 Eylül 2017

[79] “The worlds of Database Systems - Stanford InfoLab”. http://infolab.stanford.edu/~ullman/fcdb/ch1.pdf. Erişim Tarihi: 19 Eylül 2017

[80] “NOSQL Databases”. http://nosql-database.org/. Erişim Tarihi: 19 Eylül 2017

[81] “What is Apache Cassandra | DataStax Academy: Free Cassandra Tutorials and Training”. https://academy.datastax.com/planet-cassandra/what-is-apache-cassandra. Erişim Tarihi: 19 Eylül 2017

[82] “About MariaDB”, *MariaDB.org*. https://mariadb.org/about/. Erişim Tarihi: 19 Eylül 2017

[83] “Dead database walking: MySQL’s creator on why the future belongs to MariaDB”, *Computerworld*.

https://www.computerworld.com.au/article/457551/dead_database_walking_mysql_creator_why_future_belongs_mariadb/. Erişim Tarihi: 19 Eylül 2017

[84] "What is MariaDB Galera Cluster?", *MariaDB KnowledgeBase*. https://mariadb.com/kb/en/library/what-is-mariadb-galera-cluster/. Erişim Tarihi: 19 Eylül 2017

[85] "Man-in-the-middle attack - OWASP". https://www.owasp.org/index.php/Man-in-the-middle_attack. Erişim Tarihi: 18 Aralık 2017

[86] "Amazon Web Services Simple Monthly Calculator". https://calculator.s3.amazonaws.com/index.html. Erişim Tarihi: 25 Eylül 2017

[87] "Linux Containers - LXD - Introduction". https://linuxcontainers.org/lxd/introduction/. Erişim Tarihi: 31 Ekim 2017

[88] "CoreOS". https://coreos.com/rkt/. Erişim Tarihi: 31 Ekim 2017

[89] "Solaris Containers". http://www.oracle.com/technetwork/server-storage/solaris/containers-169727.html. Erişim Tarihi: 31 Ekim 2017

[90] "Chapter 14. Jails". https://www.freebsd.org/doc/handbook/jails.html. Erişim Tarihi: 31 Ekim 2017

[91] taylorb-microsoft, "About Windows Containers". https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/. Erişim Tarihi: 31 Ekim 2017

[92] "2017 Annual Container Adoption Survey: Huge Growth in Containers", *Portworx*, 12-Nis-2017. https://portworx.com/2017-container-adoption-survey/. Erişim Tarihi: 31 Ekim 2017

[93] "Run Multiple Processes in a Container", *Runnable Docker Guides*, 28-Haz-

2016. https://runnable.com/docker/rails/run-multiple-processes-in-a-container. Erişim Tarihi: 25 Eylül 2017

[94] "Go vs PHP (64-bit Ubuntu quad core) | Computer Language Benchmarks Game". https://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=go&lang2=php. Erişim Tarihi: 29 Ekim 2017

[95] "Go vs Node.js (64-bit Ubuntu quad core) | Computer Language Benchmarks Game". https://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=go&lang2=node. Erişim Tarihi: 29 Ekim 2017

[96] "Go vs Python 3 (64-bit Ubuntu quad core) | Computer Language Benchmarks Game". https://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=go&lang2=python3. Erişim Tarihi: 29 Ekim 2017

[97] "Go vs Java (64-bit Ubuntu quad core) | Computer Language Benchmarks Game". https://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=go&lang2=java. Erişim Tarihi: 29 Ekim 2017

[98] "To the Cloud: Why We've Moved to an API-Based Architecture – Part 1: Content as a Service". https://blogs.cisco.com/ciscoit/b-cd-01072015-api-based-architecutre-part1-content-as-a-service. Erişim Tarihi: 25 Eylül 2017

[99] "NoSQL Comparison Benchmarks", *DataStax: always-on data platform | NoSQL | Apache Cassandra*. https://www.datastax.com/nosql-databases/benchmarks-cassandra-vs-mongodb-vs-hbase. Erişim Tarihi: 31 Ekim 2017

[100] "Swagger Editor". http://editor.swagger.io/. Erişim Tarihi: 06 Kasım 2017

[101] D. Grijalva, *jwt-go: Golang implementation of JSON Web Tokens (JWT)*. 2017 https://github.com/dgrijalva/jwt-go. Erişim Tarihi: 03 Kasım 2017

[102] *Package gocql implements a fast and robust Cassandra client for the Go programming language*. gocql, 2017 https://github.com/gocql/gocql. Erişim Tarihi: 03 Kasım 2017

[103] "index | Alpine Linux". https://alpinelinux.org/. Erişim Tarihi: 03 Kasım 2017

[104] "Dockerfile reference | Docker Documentation". https://docs.docker.com/engine/reference/builder/. Erişim Tarihi: 04 Kasım 2017

[105] "docker build | Docker Documentation". https://docs.docker.com/engine/reference/commandline/build/. Erişim Tarihi: 04 Kasım 2017

[106] "Get started with Docker Compose | Docker Documentation". https://docs.docker.com/compose/gettingstarted/. Erişim Tarihi: 04 Kasım 2017

[107] "Running Cassandra on Rancher | Rancher Labs". http://rancher.com/running-cassandra-on-rancher/. Erişim Tarihi: 05 Kasım 2017

[108] "Cassandra and Docker Lessons Learned". https://www.slideshare.net/planetcassandra/cassandra-and-docker-lessons-learnt. Erişim Tarihi: 05 Kasım 2017

[109] "Gatling Load and Performance testing - Operations". https://gatling.io/docs/2.3/general/operations/. Erişim Tarihi: 06 Kasım 2017