T.C.
İSTANBUL SABAHATTİN ZAİM ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ
ANABİLİM DALI
BİLGİSAYAR BİLİMLERİ VE MÜHENDİSLİĞİ
(%30 İNGİLİZCE)
BİLİM DALI

BLOK ZİNCİR AĞLARINDA AKILLI KONTRAT TASARIMI

YÜKSEK LİSANS TEZİ

İrem BEKMEZ

İstanbul
Temmuz-2025

T.C.
İSTANBUL SABAHATTİN ZAİM ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ
ANABİLİM DALI
BİLGİSAYAR BİLİMLERİ VE MÜHENDİSLİĞİ
(%30 İNGİLİZCE)
BİLİM DALI

BLOK ZİNCİR AĞLARINDA AKILLI KONTRAT TASARIMI

YÜKSEK LİSANS TEZİ

İrem BEKMEZ

Tez Danışmanı
Dr. Öğr. Üyesi Hakan GENÇOĞLU

İstanbul
Temmuz-2025

Lisansüstü Eğitim Enstitüsü Müdürlüğüne,

Bu çalışma, jürimiz tarafından Bilgisayar Bilimleri ve Mühendisliği Anabilim Dalı, Bilgisayar Bilimleri ve Mühendisliği Bilim Dalında YÜKSEK LİSANS TEZİ olarak kabul edilmiştir.

Danışman Dr. Öğr. Üyesi Hakan GENÇOĞLU

Üye Dr. Öğr. Üyesi Artrim KJAMILJI

Üye Dr. Öğr. Üyesi Selçuk ŞENER

Onay

Yukarıdaki imzaların, adı geçen öğretim üyelerine ait olduğunu onaylarım.

Prof. Dr. Erhan İÇENER
Enstitü Müdürü

BİLİMSEL ETİK BİLDİRİMİ

Yüksek lisans tezi olarak hazırladığım "**Blok Zincir Ağlarında Akıllı Kontrat Tasarımı"** adlı çalışmanın öneri aşamasından sonuçlandığı aşamaya kadar geçen süreçte bilimsel etiğe ve akademik kurallara özenle uyduğumu, proje içindeki tüm bilgileri bilimsel ahlak ve gelenek çerçevesinde elde ettiğimi, proje yazım kurallarına uygun olarak hazırladığımı, bu çalışmamda doğrudan veya dolaylı olarak yaptığım her alıntıya kaynak gösterdiğimi ve yararlandığım eserlerin kaynakçada gösterilenlerden oluştuğunu beyan ederim.

İrem BEKMEZ

ÖN SÖZ

Tez konusunun belirlenmesinden araştırma ve yazım sürecine kadar her aşamada desteğini esirgemeyen; bilgi ve tecrübesinden daima yararlandığım, yönlendirmeleriyle farklı bakış açıları kazanmama katkı sağlayan ve projeyi şekillendirmemde büyük rol oynayan değerli tez danışmanım Dr. Öğr. Üyesi Hakan GENÇOĞLU'na, lisans ve yüksek lisans eğitimim boyunca sabırları ve inançlarıyla her daim yanımda olan kıymetli aileme ve her koşulda desteğini hissettiren değerli dostum Emine BOZDEMİR KARACA'ya teşekkür ederim.

İrem BEKMEZ
İstanbul-2025

ÖZET

BLOK ZİNCİR AĞLARINDA KONTRAT TASARIMI

İrem BEKMEZ

Yüksek Lisans, Bilgisayar Bilimleri ve Mühendisliği (%30 İngilizce)

Tez Danışmanı: Dr. Öğr. Üyesi Hakan GENÇOĞLU

Temmuz, 2025 – 78 Sayfa

Bu çalışma, blok zincir ağlarında akıllı kontratların tasarımını ve uygulanabilirliğini araştırmaktır. Bu bağlamda, Ethereum ağı üzerinde çalışan akıllı kontratlar, Solidity programlama dili kullanılarak Remix IDE platformu aracılığıyla geliştirilmiştir. Bu çalışma sürecinde, akıllı kontratların teorik altyapısı, mimari yapısı ve dağıtık defter teknolojilerindeki işlevleri ele alınmıştır.

İlk aşamada blok zincir teknolojisinin temel yapısı ve Ethereum altyapısının akıllı kontrat geliştirme sürecindeki önemi açıklanmış; devamında ise kontratların iş mantıkları, koşullu işlem yürütme mekanizmaları ve sözleşmeler arası etkileşim modelleri kapsamlı bir şekilde incelenmiştir. Uygulama aşamasında geliştirilen örnek kontratlar üzerinden sistemin güvenlik düzeyi, işlem maliyetleri (gas tüketimi) ve performans durumlarının değerlendirmesi yapılmıştır.

Buna bağlı olarak, akıllı kontratların yazılım geliştirme süreçlerine sağladığı yenilikler ve bu kontratların teknik zorlukları bütüncül bir yaklaşımla ele alınmıştır. Bu çalışma literatüre, Ethereum ağı özelinde, blok zincir tabanlı uygulamalarda akıllı kontratları geliştirmeye yönelik uygulamalı bir katkı sağlamayı amaçlamaktadır.

Anahtar Kelimeler: Blok zincir, Ethereum, Akıllı Kontrat, Solidity, Remix IDE

ABSTRACT

SMART CONTRACT DESIGN IN BLOCKCHAIN NETWORKS

İrem BEKMEZ

Master, Computer Science and Engineering (%30 English)

Thesis Advisor: Asst. Prof. Dr. Hakan GENÇOĞLU

July, 2025 – 78 Pages

This study investigates the design and applicability of smart contracts in blockchain networks. In this context, smart contracts running on the Ethereum network were developed using the Solidity programming language via the Remix IDE platform. In this study, the theoretical infrastructure, architectural structure and functions of smart contracts in distributed ledger technologies were discussed.

In the first stage, the basic structure of blockchain technology and the importance of the Ethereum infrastructure in the smart contract development process were explained; then, the business logic of the contracts, conditional transaction execution mechanisms and interaction models between contracts were examined comprehensively. In the application stage, the security level of the system, transaction costs (gas consumption) and performance status were evaluated through sample contracts developed.

Accordingly, the innovations that smart contracts provide to software development processes and the technical difficulties of these contracts were addressed with a holistic approach. This study aims to make a practical contribution to the literature, speci fically for the Ethereum network, towards developing smart contracts in blockchain-based applications.

Keywords: Blockchain, Ethereum, Smart Contract, Solidity, Remix IDE

İÇİNDEKİLER

TABLOLAR LİSTESİ

ŞEKİLLER LİSTESİ

KISALTMALAR LİSTESİ

PoW	: İş Kanıtı
PoS	: Bahis Kanıtı
EOA	: Harici Sahipli Hesap
MSP	: Üyelik Hizmet Sağlayıcısı
EVM	: Ethereum Sanal Makinesi
IoT	: Nesnelerin interneti
dApps	: Merkezi Olmayan Uygulamalar
ETH	: Ether
IPFS	: InterPlanetary File System
CID	: Content Identifier

BİRİNCİ BÖLÜM
GİRİŞ

Günümüzün dijital veri yönetimi süreçlerinde, dağıtık sistemler giderek daha da önemli bir rol üstlenmektedir. Özellikle güven, şeffaflık ve otomasyon gibi kriterlerin öncelik kazandığı alanlarda, blok zincir teknolojisinin sunduğu mimari çözümler büyük oranda ilgi görmektedir. Merkezi bir otoriteye ihtiyaç duymaksızın işlem doğrulama ve veri bütünlüğü sağlama kapasitesiyle blok zincir sistemleri, yalnızca finansal uygulamalarda değil, aynı zamanda hukuk, sağlık, lojistik ve dijital mülkiyet yönetimi gibi farklı disiplinlerde de kullanım alanı bulmuştur.

Blok zincir ağlarının sunduğu bu yapısal olanaklar, programlanabilir işlem modellerine yönelik ihtiyacı da beraberinde getirmiştir. Bu ihtiyaca binaen geliştirilen akıllı kontratlar; blok zincir ağlarında çalışan, dış müdahalelere kapalı ve önceden tanımlı koşullar gerçekleştiğinde otomatik bir şekilde işlem yürütebilen yazılım bileşenleri olarak ifade edilmektedir. Tanımlanmış iş kuralları çerçevesinde hareket eden bu yapılar, taraflar arasında güvenilir ve izlenebilir işlemler gerçekleştirilmesine imkân sağlamakta, böylece geleneksel sözleşmelerde sıklıkla karşılaşılan operasyonel risklerin de büyük ölçüde önüne geçebilmektedir.

Bu tez çalışması, Ethereum blok zinciri üzerinde akıllı kontratların tasarımı ve uygulanmasına odaklanmıştır. Çalışmada, Solidity programlama dili kullanılarak Remix IDE ortamında çeşitli akıllı kontratlar geliştirilmiş ve bunlar üzerinden sistemin işleyiş mekanizmaları, kullanıcı etkileşimleri, gas maliyetleri ile güvenlik dinamikleri ayrıntılı biçimde incelenmiştir. Ayrıca, akıllı kontrat geliştirme sürecinin yapısal ve mantıksal gereksinimleri irdelenmiş; bu teknolojinin yazılım geliştirme yaşam döngüsüne entegrasyonu sistematik bir bakış açısıyla ele alınmıştır.

İKİNCİ BÖLÜM
BLOK ZİNCİR TEKNOLOJİSİ VE AKILLI KONTRATLAR

2.1. Mutabakat Protokolleri

Blok zincir teknolojisinin sürdürülebilirliği ve güvenilirliği, ağdaki tüm katılımcılar arasında ortak bir veri bütünlüğü anlayışının tesis edilmesine dayanmaktadır. Bu amaç doğrultusunda geliştirilen mutabakat protokolleri, dağıtık yapının doğası gereği merkezi bir otoritenin bulunmaması nedeniyle, işlemlerin geçerliliğini sağlamak ve blokların doğrulanarak zincire eklenmesini temin etmek açısından önemli bir rol üstlenmektedir. Mutabakat mekanizmaları yalnızca işlem doğruluğunu güvence altına almakla kalmaz; aynı zamanda ağı kötü niyetli saldırılara ve veri manipülasyonu girişimlerine karşı da dirençli hale getirir.

Blok zincir platformlarının farklı ihtiyaçlarına ve güvenlik önceliklerine bağlı olarak geliştirilmiş çeşitli mutabakat yöntemleri bulunmaktadır. Özellikle Proof of Work (PoW) ve Proof of Stake (PoS) gibi protokoller, blok zincir ağlarında öne çıkan temel mutabakat modelleri arasında yer almaktadır (Bekmez ve Gençoğlu, 2024). Bu bölümde, söz konusu protokollerin teorik temelleri, işleyiş mekanizmaları ve blok zincir ekosistemine sağladıkları katkılar ele alınacaktır.

2.1.1. Proof of Work (PoW)

Blok zincir teknolojisinin ilk ve en yaygın kullanılan mutabakat mekanizmalarından biri olan Proof of Work (PoW), ağ katılımcıları arasında veri bütünlüğü ve işlem güvenliğini tesis etmek amacıyla geliştirilmiştir. Bu yöntemde, bir bloğun geçerli kabul edilerek blok zincirine eklenebilmesi için, SHA-256 algoritması kullanılarak üretilen hash değerinin, önceden belirlenen bir hedef değerin altında olması gerekmektedir. İlgili hedefe ulaşabilmek için madencilerin yüksek miktarda hesaplama gücü harcaması zorunludur; bu da ağın güvenlik seviyesini önemli ölçüde artırmaktadır. Ortalama olarak, bir bloğun işlenmesi ve zincire eklenmesi süreci yaklaşık on dakika sürmektedir.

PoW mekanizmasında, madenciler nonce adı verilen rastgele bir değeri değiştirerek, istenen hash çıktısını elde etmeye çalışmaktadır. Bu çıktıyı bulan madenci, önceki bloğun hash değeri ile bağlantılı bir şekilde yeni bir blok oluşturarak, ağ üzerinde bu

bloğun geçerliliğini sağlar. Bu yöntem, blok zincir üzerinde geçmişe dönük değişiklik yapmayı teknik olarak oldukça zorlaştırmakta ve böylece veri manipülasyonuna karşı yüksek bir direnç oluşturmaktadır.

Bitcoin ağı özelinde, mutabakat süreci en fazla hesaplama çabası gerektiren ve dolayısıyla en uzun kabul edilen blok zinciri üzerinden sağlanmaktadır. Madencilik sürecinde başarılı olan düğüm, yeni bir blok ekleme hakkı kazanmakta ve bunun karşılığında sistem tarafından Bitcoin ile ödüllendirilmektedir. Ancak, PoW mekanizması yüksek enerji tüketimi ve yoğun bilgi işlem kaynakları gerektirmesi nedeniyle ciddi ekonomik ve çevresel maliyetler doğurmaktadır. Tüm bu dezavantajlara karşın, PoW tabanlı sistemler dağıtık bilgi depolama çözümleri ve dijital kimlik doğrulama sistemleri gibi pek çok farklı alanda uygulama alanı bulmaya devam etmektedir (Ouyang, Shao ve Zeng, 2021).

2.1.2. Proof of Stake (PoS)

Proof of Work (PoW) tabanlı sistemlerin yüksek enerji tüketimi ve yoğun bilgi işlem gücü gereksinimi, blok zincir teknolojilerinde daha verimli alternatiflerin geliştirilmesini zorunlu kılmıştır. Bu çerçevede ortaya çıkan Proof of Stake (PoS) protokolü, Bitcoin ve Ethereum gibi mevcut sistemlerdeki enerji verimliliği sorunlarına çözüm sunmak amacıyla geliştirilmiştir. PoS yapısında, yeni blokların oluşturulabilmesi için yoğun madencilik faaliyetlerine gerek duyulmamakta; böylece süreç, çok daha az kaynak kullanılarak sürdürülebilmektedir.

Bu mekanizmada, kullanıcıların blok zincir üzerindeki etkinliği, sahip oldukları kripto para miktarına bağlıdır. Kullanıcının zincirde sahip olduğu pay oranı, yeni blok oluşturma sürecinde belirleyici bir faktör olarak öne çıkmakta ve mutabakat, bu paylara dayanarak sağlanmaktadır. Kripto para barındırmayan sistemlerde ise, mutabakatın tesis edilmesi amacıyla oylama gibi farklı yöntemler tercih edilmektedir.

Proof of Stake mekanizması, hesaplama yükünü önemli ölçüde azaltarak enerji tüketimini düşürmesine rağmen, ağın merkeziyetsizlik düzeyinde bazı riskleri de beraberinde getirmektedir. Özellikle "Hiçbir Şey Tehlikede Değil" (Nothing at Stake) problemi ve "Uzun Menzilli Saldırı" (Long-Range Attack) gibi güvenlik açıkları, PoS protokolünün zayıf yönleri arasında yer almaktadır. Buna karşın, Ethereum gibi büyük çaplı kripto para platformlarında ve yüksek işlem verimliliği gerektiren Araçların

İnterneti (Internet of Vehicles - IoV) gibi yeni nesil uygulamalarda, PoS mimarisinin tercih edildiği gözlemlenmektedir (Nguyen vd., 2019).

2.2. Blok Zincir Ağları

Son yıllarda dijitalleşmenin toplumsal ve kurumsal yapılar üzerindeki etkileri derinleşmiş ve bu dönüşüm, veri güvenliği, şeffaflık ve bütünlük gibi temel kavramlara yönelik yeni çözüm arayışlarını da beraberinde getirmiştir. Bu çerçevede ön plana çıkan blok zincir teknolojisi, yalnızca teknik bir yenilik olarak değil, aynı zamanda veri yönetimi paradigmasında köklü bir değişimin habercisi olarak değerlendirilmektedir. Dağıtık yapıyı ve merkeziyetsizliği esas alan bu yaklaşım, kriptografik yöntemlerle güvence altına alınmış olup, değiştirilemez ve geri dönüştürülemez şekilde yapılandırılan veri blokları üzerinden işlenmektedir. Böylece bu yapı, işlem doğruluğunu sağlayan şeffaf bir altyapı sunarken, üçüncü taraflara duyulan güven ihtiyacını da ortadan kaldırarak yeni bir dijital güven rejiminin temelini oluşturmaktadır. Blok zincir, bu özellikleriyle yalnızca bilgi teknolojileri alanında değil; ekonomi, hukuk, kamu yönetimi ve sağlık gibi çeşitli disiplinlerde de uygulama alanı bulmakta ve işleyişi dönüştüren kapsayıcı bir araç hâline gelmektedir.

Blok zincir teknolojisinin tarihsel gelişimi, bu dönüşümün dijital para birimleri üzerinden görünürlük kazandığı bir sürece işaret etmektedir. Kavramsal düzlemde 2000'li yılların başlarında tartışılmaya başlanan blok zincir, 2008 yılında Satoshi Nakamoto tarafından yayımlanan "Bitcoin: A Peer-to-Peer Electronic Cash System" adlı çalışma ile geniş kitlelerin ilgisini çekmiştir. Bu çalışma, blok zinciri yalnızca kripto para birimlerinin temelini oluşturan teknik bir altyapı olarak değil, aynı zamanda merkeziyetsiz finansal sistemlerin yapı taşı olarak da tanımlamıştır. 2009 yılında Bitcoin'in uygulamaya geçirilmesiyle birlikte, blok zincir teknolojisi daha geniş kapsamda dijital sistemlerin temelini oluşturmaya başlamıştır.

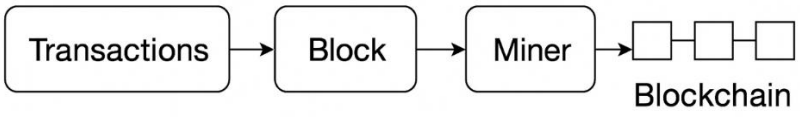
Bu tarihsel gelişimin somut bir sonucu olarak Bitcoin, blok zincir teknolojisinin en dikkat çekici uygulama örneği olarak ortaya çıkmıştır. Bitcoin, kullanıcıların herhangi bir aracıya ihtiyaç duymaksızın doğrudan dijital para transferi yapabilmesine imkân tanıyan açık kaynaklı bir yazılımdır (Zhao ve Zhang, 2021). Bitcoin ağına katılım, yalnızca Bitcoin istemcisinin indirilmesiyle sağlanmakta olup; kullanıcılar, dijital

cüzdanlarında oluşturdukları açık ve özel anahtar çiftleri aracılığıyla gönderim ve alım işlemlerini gerçekleştirmektedir.

Bitcoin ağı, işlemlerin güvenliğini sağlamak amacıyla gelişmiş kriptografik yöntemlere dayanmaktadır. Özellikle SHA-256 algoritması, yeni blokların oluşturulmasında kullanılmakta ve çift harcama gibi güvenlik tehditlerine karşı ağı korumaktadır. Bunun yanı sıra, Bitcoin işlemleri Elliptic Curve Digital Signature Algorithm (ECDSA) kullanılarak dijital olarak imzalanmakta ve yalnızca gerçek sahiplerin Bitcoin'lerini kullanabilmesi sağlamaktadır (Kowalski, 2022).

Blok zincir teknolojisi, yalnızca finans sektörüyle sınırlı kalmayıp, eğitim ve sağlık gibi kritik alanlarda da etkili uygulama örnekleri sunmaktadır. Eğitim kurumlarında, idari işlevlerin daha verimli ve güvenilir bir biçimde organize edilmesine olanak sağlayarak, eğitim yönetimi alanında dikkate değer bir kapasite oluşturduğu gözlemlenmektedir (Pal, Tiwari ve Haldar, 2021). Sağlık sektöründe ise, Dünya Sağlık Örgütü, IBM, Microsoft, Çin Ulusal Sağlık Komisyonu ve Johns Hopkins Üniversitesi gibi kuruluşların katkılarıyla geliştirilen MiPasa projesi (MiPasa, 2020), Covid-19 pandemisiyle mücadelede etkin bir araç olarak kullanılmıştır (Aydar ve Çetin, 2020).

Teknik açıdan ele alındığında ise blok zincir sistemleri, kriptografik yöntemlerle birbirine bağlı işlem bloklarından oluşan merkeziyetsiz bir kayıt defteri yapısına sahiptir. Her blok, içerdiği verilerin yanı sıra, bir önceki bloğun özet değerini (hash) de barındırmakta; bu sayede lineer ve birbirine bağlı bir veri zinciri meydana getirmektedir. Şekil 2.1'de gösterilen yapıdan da anlaşılacağı üzere, blok zincir temel olarak kullanıcılar tarafından başlatılan işlemler (transactions) ve bu işlemleri organize eden bloklardan (blocks) oluşmaktadır. İşlemler belirli bir mantıksal yapı çerçevesinde bloklara dönüştürülmekte, ardından madenciler (miners) tarafından doğrulandıktan sonra zincire eklenmektedir. Madenciler, her zaman en uzun ve geçerli kabul edilen zincir üzerinde çalışarak, sistemin bütünlüğünü ve sürekliliğini koruma görevini üstlenmektedir. Bu hash tabanlı bağlantılar, veri bütünlüğünü ve değiştirilemezliği garanti altına almakta, böylece sistemin dış müdahalelere karşı yüksek bir direnç göstermesini sağlamaktadır.



Şekil 2.1: Blok Zincir Veri Yapısı Modeli

Geleneksel merkezi veri tabanlarından farklı olarak, blok zincir mimarisi, ağdaki her bir düğümde (node) eş zamanlı ve senkronize şekilde güncellenen bir yapı sunmaktadır. Bu dağıtık sistem, işlemlerin şeffaf ve doğrulanabilir olmasına imkân tanırken, merkezi bir otoriteye duyulan gereksinimi de ortadan kaldırmaktadır. Ağ üzerinde gerçekleştirilen işlemlerin geçerlilik kazanabilmesi için, düğümler tarafından önceden tanımlanan mutabakat algoritmaları (örneğin Proof of Work veya Proof of Stake) çerçevesinde onaylanması gerekmektedir.

Blok zincir teknolojisinin dikkat çeken bir diğer yönü, yalnızca veri saklamaya değil, aynı zamanda belirli işlemleri programlamaya ve otomatikleştirmeye olanak tanımasıdır. Bu amaçla geliştirilen akıllı sözleşmeler (smart contracts), önceden tanımlı koşulların sağlanması hâlinde otomatik olarak işlemleri gerçekleştiren yapılar olarak tanımlanmaktadır (Wu vd., 2022). Akıllı sözleşmelerin detaylı yapıları ve geniş uygulama alanları, blok zincir teknolojisinin önemli bir uzantısı olarak değerlendirilmekte olup, bu konu ilerleyen bölümlerde ayrıntılı şekilde ele alınacaktır.

Bunun yanında, blok zincir sistemleri yüksek hata toleransı sunan yapılar olarak da öne çıkmaktadır. Özellikle Bizans hata toleransı (Byzantine Fault Tolerance - BFT) gibi mekanizmalar sayesinde, bazı düğümlerin hatalı ya da kötü niyetli davranması durumunda dahi ağın genel işleyişi sekteye uğramadan devam edebilmektedir (Spasovski ve Eklund, 2017).

Blok zincir mimarisi, güvenlik açısından açık anahtarlı kriptografi temelli bir model kullanmaktadır. Her kullanıcıya atanan açık anahtarlar, ağ üzerindeki işlemlerde dijital kimlik işlevi görmekte; yalnızca kullanıcının sahip olduğu özel anahtarlar ise veri yönetimini ve dijital varlıklara erişimi mümkün kılmaktadır. Bu yapı hem kullanıcı mahremiyetinin korunmasına hem de işlemlerin güvenilirliğinin teminat altına alınmasına hizmet etmektedir.

Bu bağlamda, blok zincir teknolojisinin yalnızca dijital finansal sistemlerle sınırlı kalmadığı; tedarik zinciri yönetimi, seçim sistemleri, tapu kayıtları ve dijital kimlik doğrulama gibi pek çok alanda da dönüşümsel etkiler oluşturma kapasitesine sahip olduğu görülmektedir. Güvenlik, şeffaflık ve merkeziyetsizlik ilkeleri üzerine inşa edilen bu yapı, dijital çağın ihtiyaç duyduğu yeni nesil veri yönetimi modelleri için güçlü ve sürdürülebilir bir alternatif sunmakta; gelişimini ise çok boyutlu bir perspektiften sürdürmektedir.

2.2.1. Hyperledger Fabric

Hyperledger Fabric, Linux Vakfı tarafından desteklenen, kurumsal uygulamalar için tasarlanmış açık kaynaklı ve izinli bir blok zinciri platformudur. Temel amacı, özellikle iş dünyasında duyulan güvenlik, gizlilik, esneklik ve yüksek performans gereksinimlerine yanıt verecek bir yapı sunmaktır. Modüler bir mimariye sahip olan Hyperledger Fabric, üyelik yönetimi ve fikir birliği mekanizmalarının kolayca özelleştirilebilmesine olanak tanımaktadır. Sistem içerisinde akıllı sözleşmeler, "zincir kodu" (chaincode) adı verilen kapsayıcılar içerisinde yürütülmekte olup; bu yapı, uygulama mantığının güvenli bir ortamda çalıştırılmasını sağlamaktadır (Yewale, 2018).

Fabric'in en dikkat çekici özelliklerinden biri, işlem gizliliğini korumak amacıyla "kanal" yapısını kullanmasıdır. Kanallar, ağ üzerindeki belirli kullanıcı gruplarının birbirinden bağımsız şekilde işlem gerçekleştirebilmesine imkân tanımakta; böylece her grup kendi özel defterini koruyabilmektedir. Bu yapı, hassas iş verilerinin yalnızca yetkili taraflar arasında paylaşılmasına olanak sağlamaktadır.

Hyperledger Fabric ağındaki düğümler, "doğrulayan eş" (validating peer) ve "doğrulamayan eş" (non-validating peer) olmak üzere iki kategoriye ayrılmaktadır. Doğrulayan eşler, işlemlerin yürütülmesi, doğrulanması ve deftere kaydedilmesinden sorumlu iken; doğrulamayan eşler, istemciler ile doğrulayan eşler arasındaki iletişimi yönetmektedir. Üyelik Hizmet Sağlayıcısı (MSP) ise ağda kimlik doğrulama ve erişim yönetimini gerçekleştiren birimdir.

Fabric'in mutabakat mekanizması, geleneksel blok zincir sistemlerinden farklı bir yaklaşım benimsemekte; işlemlerin sıralanması ve doğrulanması süreçlerini birbirinden ayırarak çalışmaktadır. Bu yöntem, sistemin ölçeklenebilirliğini artırmakta ve yüksek işlem hacimlerine karşı daha dayanıklı bir yapı sunmaktadır.

Performans analizlerine göre Hyperledger Fabric, düşük gecikme süresi ve yüksek işlem hacmi avantajları sağlamaktadır. Özellikle yoğun işlem yükü altında dahi, Ethereum gibi diğer blok zinciri platformlarına kıyasla daha yüksek işlem verimliliği sunabilmektedir (Pongnumkul, Siripanpornchana ve Thajchayapong, 2017). Ayrıca, zincir kodlarının Docker kapsayıcılarında güvenli şekilde çalıştırılması, sistem güvenliğini ve uygulama esnekliğini artıran bir diğer önemli unsurdur.

Sonuç olarak, Hyperledger Fabric; güçlü kimlik yönetimi, özelleştirilebilir modüler yapısı, işlem gizliliğini destekleyen kanal mimarisi ve yüksek performansı sayesinde, günümüz kurumsal blok zinciri çözümleri arasında ön plana çıkmaktadır. Özellikle tedarik zinciri yönetimi, finansal hizmetler ve sağlık sektörü gibi alanlarda yaygın olarak tercih edilen Fabric, güven, şeffaflık ve hesap verebilirlik ilkelerini iş süreçlerine entegre etme imkânı sunmaktadır.

2.2.2. Ethereum

Günümüzde blok zincir teknolojisinin kullanım alanları hızla çeşitlenirken, Ethereum, akıllı sözleşme yapılarını pratikte hayata geçiren ve bu alanda en yaygın kullanılan platformlardan biri olarak dikkat çekmektedir. Ethereum, yalnızca akıllı sözleşmelerin oluşturulması ve çalıştırılması için uygun bir altyapı sunmakla kalmayıp, aynı zamanda Ethereum Sanal Makinesi (EVM) ve Solidity programlama dili aracılığıyla geliştiricilere yüksek düzeyde esneklik sağlamaktadır. Bu yapı sayesinde zincir üzerinde koşul tabanlı işlem yönetimi, erişim denetimi ve veri bütünlüğü gibi birçok işlev doğrudan kod düzeyinde güvence altına alınabilmektedir.

Ethereum ağı, akıllı sözleşmelerin yazımından dağıtımına kadar olan tüm yaşam döngüsünü destekleyen kapsamlı bir mimariye sahiptir. Solidity diliyle geliştirilen sözleşmeler, derlenerek EVM üzerinde çalıştırılabilir hâle getirilmekte ve ardından işlem havuzuna gönderilmektedir. Burada seçilen işlemler, madenciler tarafından bloklara dâhil edilmekte ve mutabakat süreci tamamlandıktan sonra kalıcı biçimde blok zincirine kaydedilmektedir. Ethereum'un gas modeli, ağdaki kaynak kullanımını optimize ederek işlem önceliklendirmesine ve genel ağ verimliliğinin artmasına katkı sunmaktadır.

Kimlik doğrulama, veri paylaşımı, varlık yönetimi ve oy kullanımı gibi pek çok alanda geliştirilen akıllı sözleşmeler, Ethereum ağı üzerinde veri bütünlüğünü ve süreçlerin denetlenebilirliğini güvence altına almaktadır. Platformun halka açık yapısı ve açık kaynaklı ekosistemi, Ethereum'un ölçeklenebilirlik, gizlilik ve işlem verimliliği gibi konularda sürekli gelişimini sürdürmesine olanak tanımaktadır.

Bu çalışma kapsamında, akıllı sözleşme uygulamalarının geliştirilmesi amacıyla Ethereum platformu tercih edilmiştir. Sonraki bölümde, Ethereum'un mimari yapısı,

ağ işleyişi ve teknik özellikleri kapsamlı bir şekilde ele alınarak, platformun akıllı sözleşme geliştirme süreçlerindeki rolü detaylı biçimde incelenecektir.

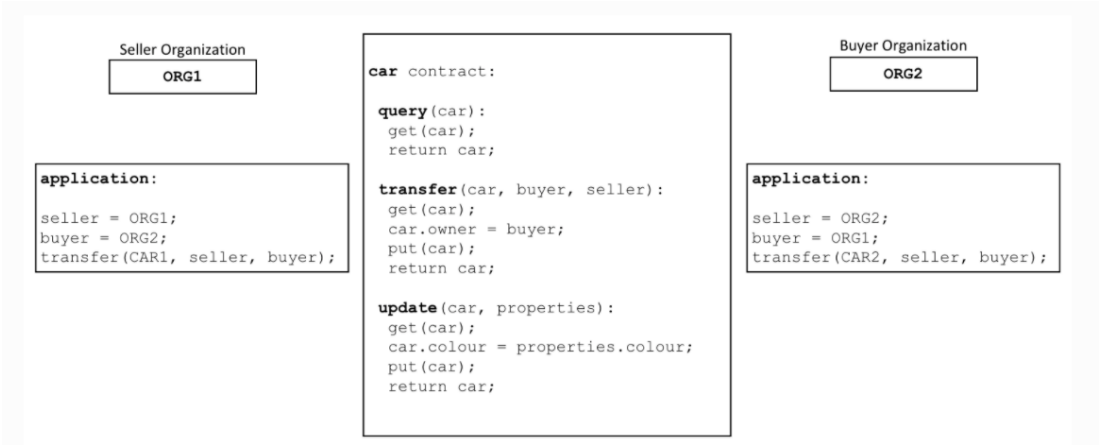
2.3. Akıllı Kontratlar

Dijital dönüşüm süreci, yalnızca ekonomik yapıları değil; yönetim, güvenlik ve işlem güvenilirliği gibi kritik sistem bileşenlerini de yeniden yapılandırmaktadır. Bu dönüşümde merkeziyetsiz sistem mimarileri ve özellikle akıllı sözleşmeler (smart contracts), önemli bir rol üstlenmektedir. İlk kez Nick Szabo tarafından 1990'ların sonlarında kavramsallaştırılan akıllı sözleşmeler, blok zincir teknolojisinin gelişmesiyle birlikte özellikle Ethereum ağı üzerinde programlanabilir dijital sözleşmeler olarak uygulanabilir hâle gelmiştir (Dongfang ve Wang, 2022).

Teknik olarak akıllı sözleşmeler, belirli koşullar sağlandığında kendiliğinden çalışan, işlemleri otomatik olarak başlatan ve sonuçlarını blok zincire kaydeden yazılım bileşenleridir. Bu sözleşmeler, geleneksel hukuki anlaşmaların dijital eşleniği olarak düşünülebilir; ancak herhangi bir aracıya gerek duymazlar. İşlem kuralları doğrudan kod biçiminde tanımlanır ve sistemin tüm katılımcıları tarafından denetlenebilir hale getirilir.

Örneğin bir akıllı sözleşme, belirli bir ödeme gerçekleştikten sonra otomatik olarak dijital bir varlığın transferini gerçekleştirebilir ya da tedarik zincirinde belirli koşullar yerine geldiğinde ürün sevkiyatını başlatabilir. Bu sayede, manuel işlem ihtiyacı ortadan kalkarken, insan hatası ve suistimal riskleri de minimize edilmiş olur.

Bu yapının kurumsal düzeydeki işleyişini daha somut biçimde açıklamak adına, Şekil 2.2'deki senaryo örneklendirilebilir:



Şekil 2.2: ORG1 ve ORG2 Arasında Akıllı Kontrat Etkileşimi

Blok zincir ağı üzerindeki yetkili aktörler olarak tanımlanan ORG1 ve ORG2, kimlik doğrulama süreçleriyle sisteme dâhil olan ve işlem gerçekleştirme yetkisine sahip iki bağımsız kurumu temsil etmektedir. Her iki kuruluş, ağ üzerinde kendi düğümlerine (node) sahip olup; işlemleri doğrulamak, akıllı sözleşmeleri çalıştırmak ve yeni blokların oluşturulmasına katkıda bulunmakla yükümlüdür.

Bu kurumlar, örneğin bir taşıtın mülkiyet devrini içeren bir işlem senaryosunda, aralarındaki etkileşimi yönetmek amacıyla ortak bir akıllı sözleşme oluşturabilir. Tanımlanan bu sözleşme; aracın belirli bir tarih aralığında teslim edilmesini zorunlu kılabilir ya da mülkiyetin yalnızca ödeme gerçekleştiğinde devredilebileceğini şart koşarak işlem güvenliğini ve taraflar arası güveni sistem düzeyinde sağlamayı hedefleyebilir.

Söz konusu işlem başladığında, ORG1'in sisteme yaptığı çağrı ile sözleşme yürürlüğe girer ve sistem, tüm şartların sağlanıp sağlanmadığını denetleyerek yalnızca koşullar tamamlandığında işlemi gerçekleştirir. Bu süreçte, arabanın durumu, sahiplik verisi ve işlem zamanı gibi bilgiler blok zincire kalıcı olarak yazılır.

Böylece hem ORG1 hem de ORG2, dış müdahalelere kapalı, tarafsız ve geri alınamaz bir işlem geçmişine sahip olur. İş akışındaki tüm taraflar işlemleri doğrudan blok zincir üzerinden izleyebilir.

Bu iş modeline ilişkin şematik gösterimler, çeşitli platformlarda "organization-to-organization smart contract interaction" başlığı altında sunulmakta ve dijital sözleşmelerin çok taraflı ticari yapıların temel taşı hâline geldiğini ortaya koymaktadır.

Bu tür akıllı sözleşmeler, yalnızca ticarette değil, aynı zamanda birçok sektörde de etkin biçimde kullanılmaktadır:

Dijital Kimlik ve Erişim Denetimi: Özellikle IoT sistemlerinde, cihazların birbirine güvenli şekilde veri aktarması için kullanıcı rolleri ve yetkileri akıllı sözleşmelerle yönetilmektedir (Ma vd., 2022).

Tedarik Zinciri: Ürünlerin üretimden son kullanıcıya ulaşana kadar geçtiği tüm aşamalar blok zincir ile kayıt altına alınarak sahtecilik ve uygunsuzluk riski azaltılmakta, tedarik güvenliği artırılmaktadır (Abuhashim ve Tan, 2020).

Sigorta, Lisanslama ve Tapu Yönetimi: Poliçelerin yalnızca belirlenen şartlar oluştuğunda ödenmesi, mülkiyet devrinin doğrulama süreçleri veya dijital içerik haklarının sahipliğinin korunması gibi alanlarda, işlem güvenliği akıllı sözleşmelerle sağlanmaktadır (Srinivasan vd., 2020).

Seçim Sistemleri (E-Voting): Oy hakkı kontrolü, tekrarsız oy kullanımı ve şeffaf sonuç üretimi gibi süreçler, merkeziyetsiz biçimde yürütülebilir hâle gelmiştir. Bu bağlamda yapılan örnek çalışmalardan biri, Tunus'ta yürütülen blok zincir temelli öğrenci oylama sistemidir. Bu sistemde, öğrencilere dağıtılmış kimlikler atanmış, her bir kullanıcı yalnızca bir kez oy kullanabilecek şekilde tanımlanmış ve seçim sonuçları doğrudan Ethereum ağı üzerinde yayımlanmıştır (Ezzeddini vd., 2022). Benzer bir sistem ise Malezya'da bir üniversitede gerçekleştirilmiş; burada MetaMask entegrasyonu ile sisteme giriş yapılmış ve seçim süreci Solidity ile geliştirilen akıllı sözleşmeler aracılığıyla yürütülmüştür (Rosasooria vd., 2020).

Literatürdeki uygulamalara bakıldığında, akıllı sözleşmeler yalnızca teknik bir yenilik değil; aynı zamanda yönetimsel ve kurumsal düzeyde şeffaflık, hesap verebilirlik ve güvenliğin yeniden tanımlandığı bir sistem önerisi sunmaktadır. Bu sistemler, yalnızca büyük ölçekli işlemler için değil, aynı zamanda yerel yönetim, akademik kurumlar ve sivil toplum örgütleri gibi mikro düzeyde yönetişim süreçleri için de uygulanabilir ve ölçeklenebilir çözümler üretmektedir.

ÜÇÜNCÜ BÖLÜM
ETHEREUM AĞI

3.1. Ethereum

Dağıtık defter teknolojilerindeki gelişmeler, finansal işlemlerin yanı sıra farklı sektörlerde de merkeziyetsiz uygulamaların geliştirilmesine zemin hazırlamıştır. Bu çerçevede, Ethereum mimarisi, programlanabilir yapısıyla söz konusu teknolojik dönüşümün öncülerinden biri olarak dikkat çekmektedir.

Ethereum, merkeziyetsiz uygulamaların (dApps) geliştirilmesine imkân tanımak amacıyla tasarlanmış, Turing-tam programlama desteğine sahip açık kaynaklı bir blok zinciri platformudur. Platform, yalnızca dijital para transferlerini değil, aynı zamanda koşul tabanlı karmaşık işlemleri de destekleyen bir altyapı sunmaktadır. Ethereum Sanal Makinesi (EVM) üzerinde çalışan akıllı sözleşmeler aracılığıyla geliştiriciler, kendi uygulamalarını oluşturabilmekte ve bu uygulamaları güvenli bir biçimde dağıtık ağ üzerinde çalıştırabilmektedir.

Ethereum, yerel kripto para birimi Ether ile EVM'yi birleştirerek hem ekonomik işlemleri hem de programlanabilir sözleşmeleri destekleyen bütünleşmiş bir yapı sunmaktadır. EVM, Ethereum blok zinciri üzerinde çalışan programların yürütülmesini sağlamakta olup; akıllı sözleşmeler çoğunlukla Solidity programlama dili kullanılarak geliştirilmekte ve sonrasında bayt kodu (bytecode) biçimine derlenmektedir. Dağıtım süreci sonrasında bu bayt kodu, harici olarak yönetilen hesapların (Externally Owned Accounts, EOA) adres alanında saklanmakta, böylece insan tarafından kontrol edilen hesaplar ile akıllı sözleşme hesapları teknik açıdan ayırt edilemez hâle gelmektedir (Giesen vd., 2022).

Ethereum ağı üzerinde gerçekleştirilen her etkileşim, bir işlemin oluşturulmasını ve yürütülmesini zorunlu kılmaktadır. Kullanıcılar, akıllı sözleşme işlevlerini çağırmak veya ağ üzerinde başka işlemler gerçekleştirmek istediklerinde, gaz (gas) mekanizması aracılığıyla yürütme maliyetlerini karşılamak durumundadırlar. Gaz ücretleri, EVM tarafından yürütülen talimatların türüne ve işlemlerin karmaşıklık derecesine bağlı olarak değişkenlik göstermektedir. Ayrıca, bir akıllı sözleşmenin çağrılması sırasında, bu sözleşmenin başka akıllı sözleşmeleri tetikleyebilmesi de mümkündür. Bu durum,

Ethereum ağı üzerinde zincirleme ve çok katmanlı işlem yapılarına imkân tanımakta, platformun işlevselliğini artırmaktadır.

3.1.1. Ethereum Yapısı

Ethereum ağı üzerindeki her hesap, belirli temel unsurlar çerçevesinde yapılandırılmaktadır. Bir Ethereum hesabı dört ana bileşenden oluşur:

1. Nonce: Ethereum blok zincirinde nonce, her hesabın gönderdiği transferlerin sırasını belirlemek ve aynı verinin yeniden yürütülmesini önlemek amacıyla kullanılan, sürekli artan bir tamsayı olarak ifade edilir. Her yeni gönderim, ilgili hesabın güncel nonce değeriyle tam olarak eşleşmek zorundadır. Bu sayede ise aktarım kayıtları doğru bir sıra ile blok zincire dahil edilir ve her bir transfer yalnızca bir kez onaylanır. Matematiksel açıdan değerlendirildiğinde bu yaklaşım, veri bütünlüğünü garanti altına alır ve tekrar eden işlemleri engeller. Buna ilaveten nonce değeri, ağın genel güvenliğini destekler ve gönderimlerin doğrulama sürecinde önemli bir rol oynar. Bu sebeple bu mekanizma, blok zincir sistemlerinin istikrarlı ve güvenilir biçimde çalışmasının temel bileşenlerinden biri olarak kabul edilmektedir.

Matematiksel olarak bu durum şu şekilde ifade edilir:

$\forall\ tx \in T:\ tx.nonce = Account.nonce$

Bu koşul sağlanmadığı takdirde işlem, ağ tarafından geçersiz kabul edilerek reddedilir (Ethereum Yellow Paper, 2025).

Ethereum ağı üzerinde gerçekleştirilen transferler, gönderen hesabın özel anahtarı kullanılarak dijital şekilde imzalanır. Bu imzalama işlemi, Elliptic Curve Digital Signature Algorithm (ECDSA) yöntemiyle yapılır ve secp256k1 adı verilen eliptik eğriye dayanır. İşleme ait veriler, imza oluşturulmadan önce Keccak-256 algoritması yardımıyla kriptografik olarak özetlenir. Ardından üretilen dijital imza, doğrulayıcı düğümler tarafından çözümlenerek hem gönderenin kimliği doğrulanır hem de verinin değiştirilmediği garanti altına alınır. Bu süreç, yalnızca yetkili kullanıcıların işlemlerinin kabul edilmesini sağlayan ve ağın genel güvenliğini destekleyen kritik bir güvenlik katmanı olarak işlev görür.

2. Balance: Hesabın sahip olduğu Ether (ETH) miktarını gösterir.

3. CodeHash: Sadece akıllı sözleşme hesaplarında bulunan bu alan, ilgili sözleşmeye ait EVM kodunun kriptografik hash değerini barındırır.

4. StorageRoot: Hesaba ait kalıcı verilerin saklandığı Merkle Patricia Trie veri yapısının kök hash değerini içerir.

Bu bileşenler, Ethereum'un Execution Layer bileşeninde tanımlanan ve ağı oluşturan durum nesnesinin (state object) birer parçası olarak konumlandırılmaktadır (Ethereum Yellow Paper, 2025).

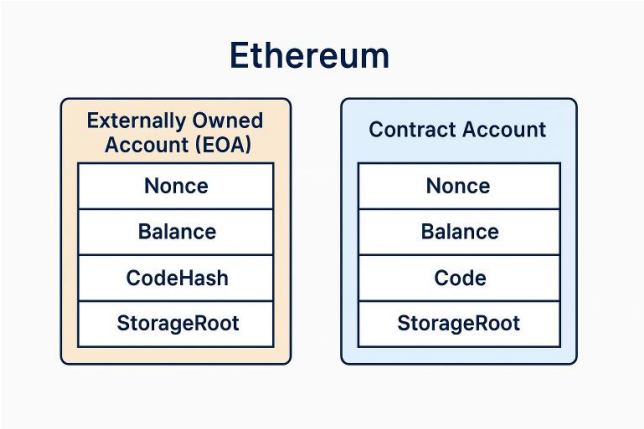
Ethereum ağı, iki temel hesap türü barındırmaktadır:

1. Externally Owned Account (EOA): Bir özel anahtar tarafından kontrol edilen bu hesap türü, doğrudan Ether gönderebilir ve işlemleri başlatabilir. Ancak akıllı sözleşme kodu içermez.

2. Contract Account: Bir akıllı sözleşme barındıran bu hesap türü, EVM üzerinde çalışan kodlar aracılığıyla kontrol edilir ve yalnızca başka bir işlem veya mesaj yoluyla tetiklenebilir. Aynı zamanda, kalıcı veri depolama (storage) kapasitesine sahiptir (Ethereum.org, 2024).

Ethereum'un Execution Layer düzeyinde tanımlı olan bu hesap yapısı, ağın Proof of Work (PoW) dönemindeki hâliyle korunmuş ve Proof of Stake (PoS) geçişi sonrasında da değişikliğe uğramamıştır. Yani, mutabakat süreci ve blok üretimi yalnızca Consensus Layer üzerinde evrilmiş; hesap yapısı, işlem mantığı ve akıllı sözleşme yürütme süreçleri mevcut formunu muhafaza etmiştir.

Şekil 3.1'de de gösterildiği üzere, Ethereum ağı üzerindeki tüm hesaplar bu iki temel formdan birine aittir. Her iki hesap türü de durum nesnesi (state object) içinde benzer yapısal özelliklere sahip olmakla birlikte, kontrol ve işleyiş bakımından farklılık göstermektedir.



Şekil 3.1: Ethereum Yapısı

Externally Owned Account (EOA) ile Contract Account arasındaki temel farklar Tablo 3.1'de özetlenmiştir.

Tablo 3.1: EOA ve Contract Account Arasındaki Temel Farklılıklar

Özellik	EOA	Contract Account
Kontrol	Özel anahtar (dış kullanıcı)	Kod (akıllı sözleşme)
Kod içerir mi?	Hayır	Evet (EVM bytecode)
Başlatıcı olur mu?	Evet	Hayır (yalnızca tetiklenir)
Veri saklar mı?	Genelde hayır	Evet (örneğin mapping, counter, vs.)

Bu çerçevede, Ethereum'un mimari özellikleri ve akıllı sözleşme altyapısı, merkeziyetsiz uygulamaların gelişiminde kritik bir rol üstlenmektedir ve ilerleyen bölümlerde bu mimarinin teknik detayları kapsamlı biçimde incelenecektir.

3.1.2. Ethereum Blok Yönetimi ve İki Katmanlı Mimari

Ethereum'un blok yapısı, zincirleme bir veri organizasyonuna dayanmaktadır. Her blok, kendisinden önce gelen bloğa kriptografik bir referans içererek bağlanmakta ve böylece bloklar arasında doğrulanabilir bir süreklilik sağlamaktadır. Her yeni blok,

içerdiği işlemler ile, ağın güncel durumunun (state) kriptografik bir özetini sunmakta ve bu sayede veri bütünlüğü ve işlem geçmişi güvence altına alınmaktadır. Bloklar arasındaki bu bağlantı yapısı, zincirin geriye dönük olarak değiştirilmesini teknik olarak son derece zor hâle getirmekte ve ağı dış müdahalelere karşı dirençli kılmaktadır.

Ethereum, Proof of Stake (PoS) mekanizmasına geçişle birlikte, daha gelişmiş bir iki katmanlı mimari yapı benimsemiştir. Bu yeni mimari, platformun hem ölçeklenebilirliğini hem de güvenliğini artırmayı hedeflemektedir. Söz konusu iki ana katman şu şekilde tanımlanmaktadır:

a. Execution Layer (Yürütme Katmanı)

Bu katman, Ethereum ağı üzerinde gerçekleştirilen işlemlerin yürütüldüğü, akıllı sözleşmelerin çalıştırıldığı ve ağın durumunun güncellendiği bölümdür. Kullanıcı işlemleri burada işlenmekte, sözleşme etkileşimleri gerçekleştirilmekte ve her yeni işlemin sonuçları ağın genel veri tabanına yansıtılmaktadır. Execution Layer, Ethereum Sanal Makinesi (EVM) aracılığıyla işlemekte ve ağın programlanabilir özelliklerini mümkün kılmaktadır.

Ethereum ağında gerçekleştirilen işlemler sırasında veri bütünlüğünü sağlamak amacıyla Keccak-256 algoritması kullanılmaktadır. Keccak, Sponge Function mimarisi üzerine inşa edilmiş olup, günümüzde SHA-3 kriptografik standardının temelini oluşturur (Bertoni vd., 2011). Bu algoritma, blok üretimi esnasında, yürütülen her işlemin sonucunda oluşan stateRoot, transactionsRoot ve receiptsRoot kriptografik özetlerini hesaplar ve bunları blok başlığına kaydeder. Böylece her bloğun içeriği, üzerinde mutabık kalınan bu özet değerleri aracılığıyla doğrulanabilir hale gelir. Bu yaklaşım, zincirin değiştirilemezliğini ve veri tutarlılığını güvence altına alır. Keccak algoritmasının bu özellikleri, Ethereum'un güvenlik modelinin temel unsurlarından birini teşkil eder.

Blok İçeriği;

Her Execution Payload (eski "blok") aşağıdaki alanları içerir:

parentHash: Önceki bloğun hash değeri

stateRoot: Küresel durumun Merkle Patricia Trie kökü

transactionsRoot: İşlem listesinin Merkle kökü

receiptsRoot: Makbuzların Merkle kökü

gasLimit, gasUsed: Blok için tanımlanan ve kullanılan gaz

timestamp: Zaman damgası

baseFeePerGas: EIP-1559 ile belirlenen taban ücret

logsBloom: Olay filtrelemede kullanılan özet

transactions: İşlem listesi

extraData: Blok üreticisine ait bilgi (opsiyonel)

Not: difficulty, nonce ve iş ispatı ile ilgili alanlar PoS geçişiyle birlikte kaldırılmıştır.

b. Consensus Layer (Mutabakat Katmanı)

Bu katman ise blokların oluşturulmasından, onaylanmasından ve ağ genelinde mutabakatın sağlanmasından sorumludur. Blok üretim süreci, doğrulayıcılar (validators) aracılığıyla yürütülmekte ve ağ üzerindeki işlemlerin sıralanması ile onaylanması Consensus Layer içerisinde gerçekleşmektedir. Bu yapı, ağın güvenliğini ve istikrarını sağlamanın yanı sıra, saldırılara karşı dayanıklılığını da artırmaktadır.

Bu iki katmanlı yaklaşım, Ethereum'un hem teknik hem de yapısal açıdan daha esnek, ölçeklenebilir ve sürdürülebilir bir blok zincir altyapısına kavuşmasını sağlamıştır. Özellikle yüksek işlem hacmine sahip uygulamalarda, bu ayrım sayesinde Execution ve Consensus işlevleri birbirinden ayrılarak ağın performansı optimize edilebilmektedir.

Blok geçerlilik süreci ise şu adımlarla belirtilebilir:

Ethereum ağında bir bloğun geçerli kabul edilmesi, belirli bir dizi kontrol adımının başarıyla tamamlanmasına bağlıdır. Proof of Stake (PoS) mekanizması kapsamında, blok geçerliliği süreci şu aşamalardan oluşmaktadır:

Önceki Blok Doğrulaması: Yeni bloğun referans verdiği önceki bloğun geçerli olup olmadığı kontrol edilir. Bu adımda, parent_root alanının doğruluğu onaylanır.

Zaman Damgası Kontrolü: Blokta belirtilen zaman damgasının (timestamp), slot bazlı zaman sistemi ile uyumlu olup olmadığı incelenir.

İşlem Yürütme: Execution Payload içerisinde yer alan işlemler sırayla yürütülerek ağın güncel durumu elde edilir.

Durum Kökü Doğrulaması: İşlemlerin yürütülmesi sonucu ortaya çıkan yeni stateRoot değeri, blok başlığında belirtilen stateRoot ile karşılaştırılır. Elde edilen değerler eşleşmelidir.

Doğrulayıcı Oylamaları (Attestations): Blok zincirinin ağırlığı, doğrulayıcılar tarafından verilen oylar ile belirlenir. Bu süreçte ağırlıklı oylama mekanizması uygulanır.

Finalization: Blok, doğrulayıcıların en az üçte ikisinin (%2/3) desteğini alması hâlinde finalized (kesinleşmiş) statüsüne ulaşır.

Proof of Stake mekanizmasında geleneksel anlamda "iş ispatı" (Proof of Work) bulunmamaktadır. Bunun yerine, ağı koruma ve yeni blokları doğrulama işlemleri stake edilen Ether (ETH) miktarına dayalı doğrulayıcı oylamaları aracılığıyla gerçekleştirilmektedir (Ethereum.org, 2024).

Ethereum'un PoS geçişiyle birlikte blok yönetimi yapısal açıdan önemli değişikliklere uğramıştır:

Blok yapısı iki katmana ayrılmıştır: Execution Layer ve Consensus Layer. İşlemler Execution katmanında yürütülürken, blok üretimi ve mutabakat süreçleri Consensus katmanında gerçekleşmektedir.

İş ispatı (Proof of Work), zorluk seviyesi (difficulty) ve nonce gibi kavramlar sistemden çıkarılmıştır.

Blok geçerliliği, işlemlerin yürütülmesi, durum verisinin (state) hash'lenmesi ve ardından doğrulayıcı oylamaları ile sağlanmaktadır.

Bu yapı, Ethereum ağının hem enerji verimliliğini artırmış hem de mutabakat süreçlerini daha hızlı ve esnek bir hale getirmiştir.

3.1.3. Ethereum Blok Görüntüleme

Ethereum ağında her blok, yalnızca içerdiği işlemleri barındırmakla kalmaz; aynı zamanda bu işlemlerin yürütülmesi sonucunda oluşan küresel durumun (global state) kriptografik kök hash değerini de kapsar. Bu özellik, Ethereum'un hesap modeline özgü bir yapısal yaklaşımdır ve ağ üzerindeki durumun şeffaf ve doğrulanabilir olmasını sağlamaktadır.

Bloklar arasındaki ilişki, her bloğun başlığında bulunan parentHash alanı üzerinden kurulan kriptografik referanslarla sağlanmaktadır. Böylece, her yeni blok, kendisinden önce gelen bloğun bir özetini içererek zincir boyunca doğrulanabilir bir süreklilik oluşturur.

Ethereum'da her işlem yürütüldüğünde, ağın mevcut durumu güncellenmekte ve bu güncellemenin sonucunda bir durum geçişi meydana gelmektedir. Her blokta, bu durum geçişinin sonucu, stateRoot olarak adlandırılan bir kök hash değeriyle temsil edilmektedir. Bu kök değer, Merkle Patricia Trie adlı özel bir veri yapısı kullanılarak oluşturulmaktadır. Trie yapısının önemli bir özelliği, yalnızca değişen düğümlerin güncellenmesidir; böylece hem veri verimliliği sağlanmakta hem de durum geçişleri minimum veri kullanımıyla doğrulanabilmektedir.

Ethereum Execution Layer, bu durum bilgisini Merkle Patricia Trie içinde organize ederken; işlemler (transactionsRoot) ve işlem makbuzları (receiptsRoot) için de benzer Merkle tabanlı yapıların kullanılması, ağ üzerinde yapılan işlemlerin ve sonuçlarının bütünlüğünü garanti altına almaktadır.

Bu yapı sayesinde, bir blok başlığındaki kök hash değerleri kullanılarak ağın mevcut durumu veya belirli işlemler minimum veri gereksinimiyle, hızlı ve güvenilir biçimde doğrulanabilmektedir. Böylece Ethereum, büyük veri hacmine rağmen yüksek verimlilikte ve güvenilir bir doğrulama mekanizması sunmaktadır.

Proof of Stake (PoS) geçişi sonrasında, Ethereum Execution Layer'daki blok yapısı korunmuş; yalnızca mutabakat mekanizması Consensus Layer'a taşınmıştır. Bu değişimle birlikte, blokların üretim sıralaması ve kesinlik kazanması (finality) artık Consensus Layer tarafından yönetilmekte, ancak işlemler ile durum verileri hâlen Execution Layer içerisinde oluşturulup yürütülmeye devam etmektedir.

Bu mimari yapı, Ethereum'un hem veri tutarlılığını hem de işlem güvenliğini artırarak, dağıtık sistemler arasında şeffaf ve güvenilir bir işlem altyapısı sunmasına katkı sağlamaktadır.

Ethereum ağı, PoS geçişi sonrasında geliştirdiği yapı ve blok görüntüleme mekanizmasını, belirli teknik prensipler doğrultusunda tasarlamış ve uygulamaya koymuştur. Tablo 3.2'de bu teknik ilkeler belirtilmiştir.

Tablo 3.2: Ethereum'da Blok Yapısı ve Görselleştirme İlkeleri

Özellik	Açıklama
Turing-tam VM	Execution Layer, akıllı sözleşmelerin yürütüldüğü EVM ile çalışır
Hesap tabanlı model	UTXO yerine hesap ve bakiye modeli kullanılmaktadır
Durum geçişi	APPLY(S,TX) → S' fonksiyonu Execution Layer'da işlem bazlı yürütülür
State Root	Her blokta durumun Merkle Patricia Trie kökü yer alır
Görüntüleme	Blok başlıkları ve kök hash'ler üzerinden durum kontrol edilebilir
Yönetim	Blok üretimi ve mutabakat işlemleri Consensus Layer'daki doğrulayıcılar tarafından yürütülür

3.1.4. Ethereum'da Madencilik ve Mutabakatın Dönüşümü

The Merge güncellemesi ile Ethereum ağı, madenciliğe dayalı Proof of Work mekanizmasından, doğrulayıcı merkezli Proof of Stake sistemine geçiş yapmıştır. Bu geçiş sürecinde meydana gelen yapısal değişiklikler, ilerleyen bölümlerde ayrıntılı olarak ele alınacaktır.

a. İlk Sistem: Proof of Work (PoW)

"The Merge" güncellemesinden önce Ethereum ağı, mutabakat için Proof of Work (PoW) mekanizmasını kullanmaktaydı (Buterin, 2014). Bu sistemde:

Bloklar, yüksek hesaplama gücü tüketen madenciler tarafından üretilmekteydi.

Madenciler, iş ispatı (Proof of Work) algoritması aracılığıyla karmaşık hash bulmacalarını çözerek geçerli bir blok oluşturmaktaydı.

Geçerli bir blok bulan madenci, blok zincirine bu bloğu ekleme hakkı kazanmakta ve karşılığında blok ödülü ile işlem ücretlerini elde etmekteydi.

b. Geçiş Süreci: The Merge

15 Eylül 2022 tarihinde gerçekleştirilen "The Merge" güncellemesi ile Ethereum ağı, PoW mekanizmasından Proof of Stake (PoS) sistemine geçiş yapmıştır. Bu geçişin temel sonuçları şunlardır:

Ethereum ağında madencilik faaliyetleri tamamen sona ermiştir.

Blok üretimi, artık madenciler tarafından değil, stake etmiş doğrulayıcılar (validators) tarafından gerçekleştirilmektedir.

c. Yeni Sistem: Proof of Stake (PoS) ve Beacon Chain

The Merge sonrasında Ethereum ağı, "Beacon Chain" adı verilen mutabakat katmanı üzerinde Proof of Stake mekanizmasıyla çalışmaya başlamıştır. Bu yeni yapıda:

Bloklar, "doğrulayıcı" (validator) olarak adlandırılan katılımcılar tarafından üretilmektedir.

Bir katılımcının doğrulayıcı olabilmesi için minimum 32 Ether (ETH) stake etmesi gerekmektedir.

Doğrulayıcılar, rastgele seçim mekanizmalarıyla blok üretimi ve zincir üzerindeki oylama görevlerine atanmaktadırlar.

Önceden madenciler tarafından yürütülen tüm görevler, artık doğrulayıcılar tarafından üstlenilmiştir.

d. Madencilik Sonrası Durum

The Merge sonrası Ethereum ağı üzerinde madencilik faaliyeti teknik olarak mümkün değildir.
Bu değişimin etkileri şunlardır:

Madenciler, artık blok önermek, zincir üzerinde mutabakata katılmak veya ödül kazanmak gibi haklara sahip değildir.

PoW madenciliği için kullanılan donanımlar (ASIC, GPU vb.), Ethereum ağı üzerinde işlevsiz hâle gelmiştir.

e. Proof of Stake Sisteminin Teknik İşleyişi

Ethereum'un PoS tabanlı mutabakat mekanizması, ağın sürekliliğini ve güvenliğini doğrulayıcıların etkin katılımı yoluyla sağlamaktadır. Sistem, blok önermeden oylamaya ve kötü niyetli davranışların tespit edilerek cezalandırılmasına kadar çeşitli işlevleri kapsamaktadır.

Validator Seçimi: Blok Önericisi (Proposer) Ataması:

Her "slot" döneminde bir doğrulayıcı, yeni bir blok önermek amacıyla atanır. Bu seçim süreci aşağıdaki adımları içerir:

Seçim, RANDAO (Randomness DAO) ve Gecikmeli Doğrulama Fonksiyonu (VDF) tabanlı rastgelelik mekanizmaları kullanılarak gerçekleştirilir.

Her slot için yalnızca bir adet blok önericisi (proposer) atanır.

Bu işlem, Beacon Chain üzerinde get_beacon_proposer_index fonksiyonu aracılığıyla yürütülmektedir.

Teknik işlem şu şekildedir:

```
def get_beacon_proposer_index(state: BeaconState, slot: Slot) ->
ValidatorIndex:
# Committees are shuffled per slot using a RANDAO-based seed
active_validator_indices=get_active_validator_indices(state,
get_current_epoch(state))
seed = get_seed(state, get_current_epoch(state),
DOMAIN_BEACON_PROPOSER)
return compute_proposer_index(active_validator_indices, seed, slot)
```

Attestation Süreci (Onaylama):

Attestation, doğrulayıcıların (validators) belirli bir blok ve zincir durumu hakkında oy kullandıkları kritik bir mutabakat işlemidir. Bu işlem, ağın güvenliğinin sağlanması ve doğru blok zincirinin sürdürülmesi açısından temel bir rol oynamaktadır. Her doğrulayıcı, her epoch döneminde bir kez attestation sunarak, ağdaki zincir durumunu desteklediğini veya doğrulamakta olduğunu beyan eder.

Bu oylama süreci, doğrulayıcıların katılımı oranında ağın mutabakat ağırlığını etkiler ve blokların finalized (kesinleşmiş) hâle gelmesine katkı sağlar.

Attestation, doğrulayıcıların zincir üzerindeki mutabakata katkı sağladıkları ve ağın güncel durumuna ilişkin oy kullandıkları bir işlemdir. Tablo 3.3'te gösterildiği üzere her attestation, üç temel bileşen üzerine yapılandırılmıştır: doğrulayıcının bir blok üzerinde oy kullanması, doğru epoch zamanlamasının teyit edilmesi ve zincirin doğru kontrol noktası (checkpoint) üzerinde ilerlediğinin onaylanması. Bu üç ana başlık, attestation işleminin hem blok üretiminin hem de zincir bütünlüğünün korunmasında merkezi bir rol oynamasını sağlamaktadır.

Tablo 3.3: Attestation Sürecinde Kullanılan Üç Temel Oylama Bileşeni

Alan	**Açıklama**
source	Son finalized checkpoint
target	Geçerli epoch'un checkpoint'i
head	LMD GHOST kuralına göre seçilen blok

Attestation işlem süreci, doğrulayıcıların ağ üzerindeki mutabakata etkin katılımını sağlamak amacıyla belirli adımlar doğrultusunda yürütülmektedir:

1. Komite Belirlenmesi: Her epoch başlangıcında, doğrulayıcılardan rastgele seçim yoluyla bir attestation komitesi (committee) oluşturulur. Bu komite, belirli bir epoch süresince oy kullanmakla görevlendirilir.

2. Görev Dağılımı: Belirlenen komiteler, epoch içindeki farklı slotlara atanır. Her komite, kendisine tahsis edilen slotta oy kullanma ve ağdaki zincir durumuna ilişkin attestation gönderme sorumluluğunu üstlenir.

3. Attestation Sunulması: Komite üyeleri, kendi görevli oldukları slot zamanında attestation işlemini gerçekleştirir. Bu işlem sırasında üyeler, zincir durumu ve blok doğruluğu hakkında oy kullanarak ağı destekler ve mutabakatın ilerlemesine katkıda bulunur.

Attestation içeriği şu şekildedir:

Attestation:

aggregation_bits: Bitlist

data: AttestationData

signature: BLSSignature

AttestationData ise şöyledir:

AttestationData:

slot: Slot

index: CommitteeIndex

beacon_block_root: Root

source: Checkpoint

target: Checkpoint

Slashing – Cezalandırma Mekanizması:

Slashing, Ethereum ağı üzerinde kötü niyetli davranış sergileyen veya protokole aykırı şekilde hareket eden doğrulayıcıların (validators) cezalandırılmasını amaçlayan bir güvenlik mekanizmasıdır. Bu süreç, doğrulayıcıların ağdan çıkarılması ve stake ettikleri ETH miktarının belirli bir oranının kesilmesi (slashing) işlemlerini içerir. Slashing, ağın güvenliğini korumak ve doğrulayıcıların doğru ve dürüst davranmasını teşvik etmek açısından kritik bir rol oynamaktadır.

Ethereum ağı üzerinde iki temel slashing senaryosu tanımlanmıştır:

1. Proposer Slashing: Proposer Slashing, bir doğrulayıcının aynı slot için iki farklı blok önermesi durumunda uygulanan bir cezalandırma yöntemidir. Eğer bir doğrulayıcı, belirlenen slot süresi içinde birden fazla blok üretir ve bu blokları ağa sunarsa, ağ protokolünü ihlal etmiş sayılır. Bu ihlal tespit edildiğinde, ilgili doğrulayıcının stake edilmiş ETH miktarının bir kısmı kesilir ve doğrulayıcı sistem dışına çıkarılır. Bu mekanizma, blok zincirinin tutarlılığını ve mutabakatın güvenilirliğini korumayı amaçlamaktadır.

Koşul:

proposer_slashing.signed_header_1.message.slot
proposer_slashing.signed_header_2.message.slot

2. Attester Slashing: Attester Slashing, bir doğrulayıcının aynı hedef epoch (hedeflenen zaman dilimi) için iki çelişkili attestation (oylama) sunması durumunda uygulanan bir cezalandırma mekanizmasıdır. Bu ihlal, doğrulayıcının zincir üzerinde iki farklı blok veya zincir durumu hakkında tutarsız oy kullanması anlamına gelir. Böyle bir durumda, doğrulayıcının ağ protokolünü ihlal ettiği kabul edilir ve stake ettiği ETH miktarının belirli bir kısmı kesilerek slashing işlemi gerçekleştirilir. Attester Slashing mekanizması, doğrulayıcıların ağ bütünlüğünü bozan tutarsız davranışlarını önlemek ve mutabakat sürecinin güvenilirliğini korumak amacıyla hayata geçirilmiştir.

Koşul:

is_slashable(attestation_1.data.source.epoch, attestation_2.data.source.epoch)

and

attestation_1.data != attestation_2.data

Slashing uygulama fonksiyonu ise şu şekildedir:

def slash_validator(state: BeaconState, index: ValidatorIndex, whistleblower_index: Optional[ValidatorIndex]):

initiate_validator_exit(state, index)

state.validators[index].slashed = True

state.validators[index].withdrawable_epoch = ...

decrease_balance(state,index,effective_balance//
SLASHING_PENALTY_QUOTIENT)

Slashing sonucunda validator:

Hemen aktif durumdan çıkarılır.

Ceza kesintisine uğrar.

Belirli bir epoch sonrasında stake'ini geri alabilir.

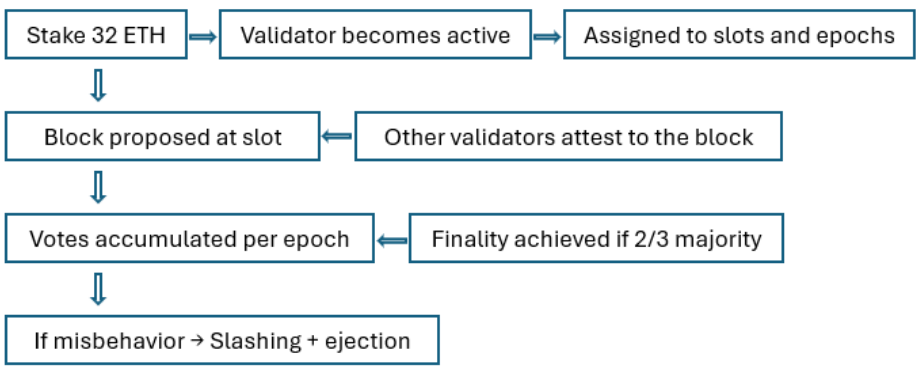
İlgili teknik özet ve işlemlere dair açıklamalar, Tablo 3.4. içerisinde bütüncül bir yaklaşımla ele alınmıştır.

Tablo 3.4: Ethereum PoS Sürecinde Teknik İşlem Özeti

İşlem	Açıklama	Fonksiyon
Validator seçimi	RANDAO + epoch seed	get_beacon_proposer_index
Attestation	Oy verme: source, target, head	AttestationData
Proposer Slashing	Aynı slotta iki blok önerme	proposer_slashing
Attester Slashing	Aynı epoch için çelişkili oylama	attester_slashing
Ceza uygulanması	Stake kesintisi, ihraç	slash_validator

f. Ethereum'un Mutabakat Yapısı

The Merge güncellemesi sonrasında, Ethereum ağı madencilere dayalı Proof of Work mekanizmasından, doğrulayıcıların etkin katılımına dayanan Proof of Stake yapısına geçiş yapmıştır. Şekil 3.2'de, yeni mutabakat mekanizmasında doğrulayıcıların üstlendiği görevlerin ve süreç akışının şematik bir gösterimi sunulmaktadır.



Şekil 3.2: Ethereum PoS Mekanizmasında Doğrulayıcı Görev Akışı

The Merge güncellemesi sonrasında Ethereum'un mutabakat mekanizması, Casper FFG ve LMD GHOST protokollerine dayanacak şekilde yapılandırılmış olup; ağ güvenliği ile katılımı artırmaya yönelik çeşitli ekonomik ve yapısal unsurlar barındırmaktadır. Söz konusu unsurların temel özellikleri ise Tablo 3.5'te özetlenmiştir.

Tablo 3.5: Ethereum Ağı: PoS Mutabakat Modeli Özeti

Bileşen	Açıklama
Mutabakat modeli	Proof of Stake
Temel protokol	Casper FFG + LMD GHOST
Blok zamanı	12 saniye (slot)
Finality süresi	~6.4 dakika (epoch başına)
Katılımcı tipi	Validator (32 ETH stake)
Ceza	Slashing – stake kesintisi ve çıkış
Güvenlik	Ekonomik teşvik + protokol denetimi

3.2. Solidity Programlama Dili

Dağıtık ve güvene dayalı dijital sistemlerin dünya genelinde hızla benimsenmesi, yalnızca finans sektörüyle sınırlı kalmayıp aynı zamanda yazılım mimarisi, bilgi sistemleri ve diğer teknolojik alanlarda da köklü dönüşümlere zemin hazırlamıştır. Bu yenilikçi yaklaşımın temelinde yer alan merkeziyetsizlik ilkesi, güven ilişkisini aracı kurumlara ihtiyaç duymadan, kod tabanlı otomasyon mekanizmaları aracılığıyla tesis etmektedir. Özellikle Ethereum'un geliştirilmesiyle birlikte bu dijital sistemler yalnızca değer transferi sağlayan bir sistem olmaktan çıkarak, programlanabilir bir altyapıya da kavuşmuştur. Bu durum ise akıllı sözleşmelerin ortaya çıkmasını mümkün kılmıştır. Akıllı sözleşmeler, blok zincir üzerinde değiştirilemez şekilde depolanan ve koşullara bağlı olarak otomatik yürütülen kod bloklarıdır. Bu tür uygulamaların geliştirilmesi, güvenli, verimli ve sürdürülebilir bir programlama dilini zorunlu kılmış ve bu bağlamda Solidity öne çıkan bir araç olmuştur.

Solidity, 2014 yılında Ethereum'un kurucu geliştiricilerinden Dr. Gavin Wood tarafından tasarlanmış ve Ethereum Foundation'ın desteğiyle açık kaynaklı olarak geliştirilmeye başlanmıştır. Başlangıçta sınırlı işlevselliğe sahip olan bu dil, zamanla topluluk katkıları, güvenlik denetimleri ve Ethereum Improvement Proposal (EIP) süreçleriyle daha kararlı ve kapsamlı bir yapıya dönüşmüştür. Solidity, Ethereum ağında çalışmak üzere optimize edilmiş olsa da günümüzde Binance Smart Chain, Polygon, Avalanche gibi Ethereum Virtual Machine (EVM) uyumlu blok zincirlerde de yaygın biçimde kullanılmaktadır. Bu yaygınlık, dili yalnızca Ethereum'a özgü bir araç olmaktan çıkararak daha geniş bir merkeziyetsiz uygulama ekosistemine hizmet edebilen genel geçer bir standart haline getirmiştir.

Teknik açıdan değerlendirildiğinde, Solidity statik tip tanımlı ve sözleşme yönelimli bir programlama dili olarak yapılandırılmıştır. Bu programlama dili JavaScript, Python ve C++ gibi yaygın dillerden etkilenmiş olup, geliştiricilere tanıdık bir ortam sunmaktadır (Antonopoulos & Wood, 2018). Ancak, blok zincir üzerindeki işlemlerin geri alınamaz olması ve her işlemin "gas" adı verilen işlem maliyetleriyle ilişkilendirilmesi, Solidity ile yazılım geliştirmenin yüksek dikkat ve optimizasyon gerektirdiğini göstermiştir. Değişken türlerinin de derleme aşamasında açıkça tanımlanması zorunludur. Sözleşmelerin erişim türleri ise public, private, internal, external gibi anahtar kelimelerle kontrol edilir. Ayrıca, fonksiyonların bellek kullanımı gibi teknik ayrıntılar da geliştiriciler tarafından dikkatle yönetilmelidir.

Bu bağlamda, Solidity dilinin temel sözdizimi ve işlevselliği, blok zincir üzerinde bir metin mesajını depolayan ve bu mesajın güncellenmesine olanak tanıyan basit bir akıllı sözleşme örneği üzerinden açıklanabilir.

```
pragma solidity ^0.8.0;

contract MerhabaDunya {

string public mesaj = "Merhaba, Blockchain!";

function degistir(string memory yeniMesaj) public {

mesaj = yeniMesaj;   }

}
```

Bu örnekte yer alan mesaj adlı değişken, sözleşme üzerinde herkese açık bir biçimde depolanmakta ve degistir fonksiyonu aracılığıyla güncellenebilmektedir. public

anahtar kelimesi değişkenin dışarıdan erişilebilir olmasını sağlarken, memory ifadesi fonksiyon parametresinin yalnızca geçici olarak bellekte tutulacağını belirtir. Her ne kadar bu sözleşme oldukça basit görünse de Solidity'nin temel yapı taşlarını anlamak açısından açıklayıcıdır.

Solidity'nin teknik gücüne rağmen, özellikle güvenlik alanında ciddi sorumluluklar barındırdığı da açıktır. Akıllı sözleşmeler blok zincir üzerine bir kez kaydedildiğinde geri döndürülemez olmaları nedeniyle, içeriklerinde yapılan hatalar sistem düzeyinde geri dönüşü olmayan zararlara yol açabilmektedir. Bu durumun en bilinen örneği, 2016 yılında gerçekleşen ve Ethereum tarihine DAO saldırısı olarak geçen olaydır. Bu saldırı, bir yeniden giriş (reentrancy) açığı kullanılarak sözleşmenin beklenmeyen şekilde sömürülmesiyle milyonlarca dolar değerindeki Ether'in kötü niyetli aktörlerce ele geçirilmesine neden olmuştur. Benzer şekilde integer overflow/underflow hataları, erişim kontrol zafiyetleri ya da dış sözleşmelere yapılan güvensiz çağrılar da Solidity sözleşmelerinde sıklıkla karşılaşılan güvenlik açıkları arasında yer almaktadır. Bu bağlamda, geliştirilen sözleşmelerin otomatik testlerden geçirilmesi, manuel kod denetimi yapılması ve tercihen bağımsız güvenlik firmaları tarafından denetlenmesi, bu tür saldırıların önüne geçilmesi açısından kritik öneme sahiptir.

Solidity'nin başarısında yalnızca teknik yapısı değil, aynı zamanda çevresinde oluşan güçlü geliştirici ekosistemi de önemli bir rol oynamaktadır. Web tabanlı bir geliştirme ortamı olan Remix IDE, kullanıcıların hızlı prototipleme ve hata ayıklama işlemlerini gerçekleştirmesine olanak tanımaktadır. Bu platform bir sonraki bölümde daha ayrıntılı biçimde ele alınacaktır. Daha büyük ve profesyonel projeler için ise Truffle ve Hardhat gibi geliştirme çerçeveleri, ölçeklenebilirlik ve çoklu test ortamı desteği sunmaktadır. Ayrıca, OpenZeppelin gibi açık kaynak kütüphaneler ise güvenlik denetiminden geçmiş ve endüstri standardı hâline gelmiş sözleşme şablonları ile geliştiricilere büyük bir kolaylık sağlamaktadır (Ethereum Foundation, n.d.). Bu araçlar yalnızca geliştirme sürecini hızlandırmakla kalmayıp, aynı zamanda sistem güvenliğini de artırmaktadır.

Öte yandan, Solidity dilinin bazı sınırlılıkları da bulunmaktadır. Dilin hata toleransının düşük olması, sürüm güncellemelerinin zaman zaman geriye dönük uyumsuzluklara yol açması ve yüksek gas ücretleri nedeniyle kodların optimize edilme zorunluluğu, geliştiricilerin karşılaştığı temel zorluklardandır. Ayrıca, mevcut blok zincir altyapısının sınırlı işlem kapasitesi, akıllı sözleşmelerin daha kompleks uygulamalarda

etkin biçimde kullanılmasını zaman zaman güçleştirebilmektedir. Bu zorluklara rağmen, Ethereum 2.0'a geçiş süreci ve katman-2 (Layer-2) çözümleri ile Solidity'nin ölçeklenebilirlik, işlem maliyeti ve verimlilik gibi alanlarda daha etkin hale gelmesi beklenmektedir. Özellikle sharding, rollup teknolojileri ve Proof of Stake mekanizması gibi yapısal değişiklikler, Solidity ile geliştirilen sözleşmelerin gelecekte daha geniş çaplı uygulamalara hizmet etmesine olanak tanıyacaktır (Buterin, 2020).

Tüm bu yönleriyle Solidity, merkeziyetsiz yazılım geliştirme ekosisteminde güçlü bir yer edinmiştir. Gerek topluluk desteği gerekse teknik yetkinliği sayesinde günümüz Web3 dünyasının temel yapı taşlarından biri hâline gelen bu programlama dili, önümüzdeki dönemde farklı platformlarla daha fazla entegre olması ve yeni nesil blok zincir mimarileriyle daha uyumlu çalışmasıyla birlikte önemini daha da artıracaktır. Özellikle güvenli, şeffaf ve özerk dijital uygulamaların artış gösterdiği bir çağda, Solidity ile geliştirilen akıllı sözleşmeler, blok zincir teknolojisinin sunduğu vaatleri somutlaştıran temel araçlardan biri olarak değerlendirilmektedir.

3.3. Remix IDE

Ethereum üzerinde akıllı sözleşme geliştirme sürecinde sıklıkla tercih edilen araçlardan biri olan Remix IDE (Integrated Development Environment), web tabanlı yapısıyla doğrudan tarayıcı üzerinden çalıştırılabilen bir geliştirme ortamıdır. Kurulum gerektirmemesi, çok sayıda eklentiyle genişletilebilir olması ve temel sözleşme işlevlerini doğrudan test etme imkânı sunması nedeniyle, farklı deneyim seviyelerinden geliştiriciler tarafından kullanılmaktadır.

Remix IDE, akıllı sözleşmelerin yazılması, derlenmesi, dağıtılması ve test edilmesi gibi temel yazılım geliştirme adımlarını tek bir platform üzerinden gerçekleştirmeye olanak tanır. Geliştirici, Solidity dilinde hazırladığı bir sözleşmeyi yerleşik simülasyon ortamında çalıştırabildiği gibi, Goerli ya da Sepolia gibi test ağlarını kullanarak sözleşmenin ağ üzerindeki davranışını da gözlemleyebilir. Bu yapı, geliştirme sürecinde güvenlik ve işlevsellik açısından erken aşamada doğrulama yapmayı mümkün kılmaktadır (Remix IDE Documentation, n.d.).

Platform, eklenti (plugin) tabanlı bir yapıda tasarlanmıştır. Bu sayede kullanıcılar yalnızca ihtiyaç duydukları araçları etkinleştirerek çalışma ortamını özelleştirebilir. Yaygın olarak kullanılan eklentiler arasında; Solidity Compiler, Deploy & Run

Transactions, Static Analysis, Gas Usage Analyzer ve Terminal modülleri yer almaktadır. Bu eklentiler aracılığıyla sözleşme kodları üzerinde derinlemesine analizler gerçekleştirilebilir, işlem maliyetleri değerlendirilebilir ve potansiyel güvenlik açıkları tespit edilebilir.

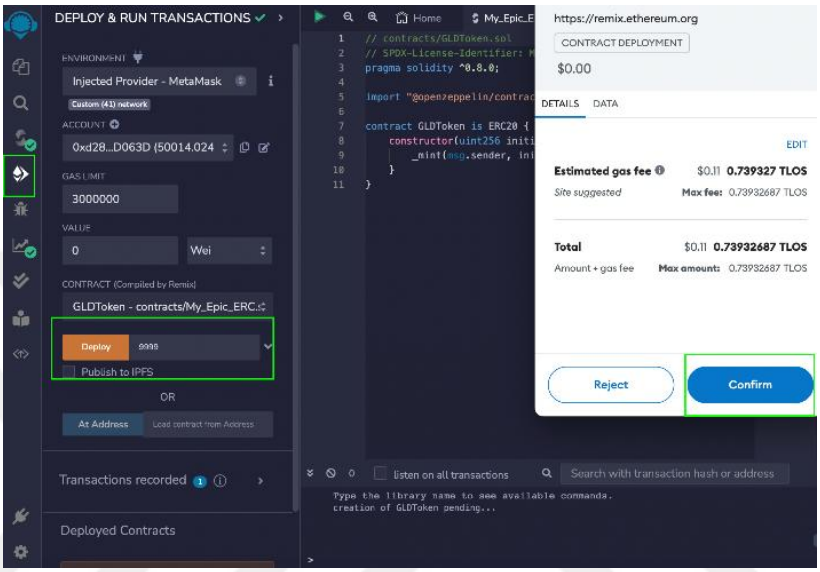
Tablo 3.6'da Remix IDE bünyesinde yaygın olarak kullanılan temel eklentiler ve bu eklentilerin akıllı sözleşme geliştirme sürecindeki işlevlerine ilişkin özet bilgiler verilmiştir.

Tablo 3.6: Remix IDE Eklentileri ve Geliştirme Sürecindeki İşlevleri

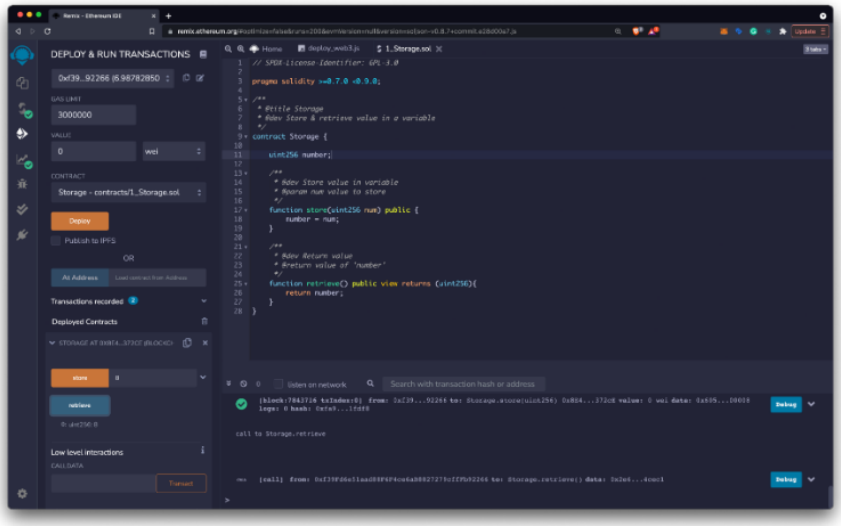
Eklenti Adı	Temel İşlevi
Solidity Compiler	Akıllı sözleşmeleri derler; farklı Solidity sürümleriyle uyumluluk testleri yapılabilir.
Deploy & Run Transactions	Sözleşmelerin yerel simülasyon veya test ağları üzerinde dağıtılmasını ve çalıştırılmasını sağlar.
Static Analysis	Kod üzerinde potansiyel güvenlik açıklarını ve yapısal hataları analiz eder.
Gas Usage Analyzer	Fonksiyon bazında işlem maliyetlerini hesaplar, performans optimizasyonuna yardımcı olur.
Terminal	Geliştirici ve ağ arasında komut tabanlı etkileşim sağlar; dağıtım ve test sürecinde geribildirim sunar.

Kod düzenleme sürecini desteklemek amacıyla Remix IDE içerisinde bütünleşmiş bir editör bulunmaktadır. Bu editör; sözdizimi renklendirme, otomatik tamamlama, hata bildirimi gibi işlevler aracılığıyla yazım sürecini kolaylaştırmakta ve kodun yapısal bütünlüğünün korunmasına katkı sunmaktadır. Derleme işlemleri sırasında kullanıcılar, farklı Solidity sürümlerini seçerek çeşitli sürüm senaryolarında kod uyumluluğunu da test edebilmektedir. Bu, özellikle sürüm değişikliklerinin projeye etkisini önceden değerlendirmek açısından önem taşımaktadır.

Şekil 3.3 ve Şekil 3.4'te Remix IDE'nin kullanıcı arayüzünü gösteren ekran görüntüleri verilmiştir. Bu görseller, geliştirme ortamının genel yapısını ve bileşenlerin konumlandırılmasını yansıtmaktadır. Sol panelde dosya gezgini ve eklentiler; orta panelde kod editörü, sağ panelde ise derleme, dağıtım ve test işlemleri için kullanılan modüller yer almaktadır.



Şekil 3.3: Remix IDE'nin Kullanıcı Arayüzünde Temel Bileşenlerin Konumlandırılması



Şekil 3.4: Remix IDE Üzerinden Akıllı Sözleşme Geliştirme Arayüzü

Remix IDE'nin Web3 entegrasyonu da dikkat çeken bir özelliğidir. Geliştiriciler, MetaMask gibi cüzdan uygulamalarıyla Remix arayüzünü bağlayarak Ethereum test ağlarına veya ana ağa doğrudan erişim sağlayabilmektedir. Bu bağlantı sayesinde, akıllı sözleşmeler yalnızca teorik düzeyde değil, gerçek ağ üzerinde de test edilebilmektedir. Bu, sözleşmenin ağdaki davranışının erken aşamada gözlemlenmesini mümkün kılar.

Sonuç olarak, Remix IDE; akıllı sözleşme geliştirme sürecinde deneysel prototipleme, test ve dağıtım aşamalarını bir arada yürütebilmeyi mümkün kılan bütünleşik bir yazılım geliştirme platformu olarak değerlendirilebilir. Bu çalışma kapsamında geliştirilen örnek uygulamalar da Remix IDE kullanılarak derlenmiş, test edilmiş ve çeşitli ağ ortamlarında değerlendirilmiştir.

DÖRDÜNCÜ BÖLÜM
E-OYLAMA İÇİN ETHEREUM TABANLI AKILLI SÖZLEŞME TASARIMI

4.1. Tezin Konusu

Bu tez, blok zincir tabanlı teknolojilerin elektronik oylama sistemleriyle entegrasyonu çerçevesinde geliştirilen akıllı sözleşme temelli bir uygulamayı incelemektedir. Çalışmada, Ethereum altyapısı kullanılarak, seçim süreçlerinde doğrulama, kayıt, oy verme ve sonuç değerlendirme işlemlerini dijital ortamda gerçekleştirebilen işlevsel bir sistem modeli oluşturulmuştur. Yapılan bu uygulama, farklı sözleşme modülleri aracılığıyla aday yönetimi, seçmen kaydı ve oy kullanımına ilişkin süreçleri birbirinden ayrıştırılmış biçimde yürütmekte; aynı zamanda dış veri kaynaklarıyla (IPFS) uyumlu bir yapıda çalışarak yazılım mimarisi açısından çok katmanlı bir yapı sunmaktadır. Geliştirme süreci boyunca Hardhat, Node.js, Remix IDE gibi araçlardan yararlanılmış ve tüm bu işlemler programatik olarak test edilebilir bir yapıda tasarlanmıştır.

4.2. Tezin Amacı

Bu çalışmanın amacı, Ethereum ağı üzerinde geliştirilen sözleşme temelli bir sistem aracılığıyla elektronik oylama süreçlerinin dijital ortama taşınmasına yönelik bir uygulama örneği ortaya koymaktır. Geliştirilen bu oylama platformunda, sadece sistem yöneticisi tarafından yetkilendirilmiş adreslerin oy kullanmasına izin verilirken, her katılımcının yalnızca bir kez oy kullanabilmesi teknik olarak garanti altına alınmıştır. Ayrıca aday bilgileri zincir dışı kaynaklardan sisteme entegre edilmiş ve sonuçların gerçek zamanlı olarak erişilebilir olması sağlanmıştır. Bu model, benzer projeler için temel teşkil edebilecek bir yapı sunmaktadır. Farklı ortamlara ise uyarlanabilecek şekilde modüler biçimde tasarlanmıştır. Bu sayede, oylama sistemlerinin dijital dönüşümüne ilişkin yazılım düzeyindeki olanaklar somutlaştırılmıştır.

4.3. Genel Sistem Mimarisi

Bu tez çalışması kapsamında, blok zincir teknolojisinin elektronik oylama sistemlerine entegrasyonu üzerine odaklanan bir prototip model geliştirilmiştir. Çalışmanın temel amacı bir önceki başlıkta da belirtildiği üzere, merkezi otoritelerin kontrolüne ihtiyaç duymaksızın işleyebilen, şeffaflık, değiştirilemezlik ve denetlenebilirlik ilkeleriyle uyumlu bir seçim altyapısı tasarlamaktır. Bu doğrultuda, Ethereum blok zinciri üzerinde çalışan akıllı sözleşmeler aracılığıyla yapılandırılmış ve Web3 uyumlu istemci taraflı uygulamalar ile entegre olabilen bir sistem mimarisi geliştirilmiştir.

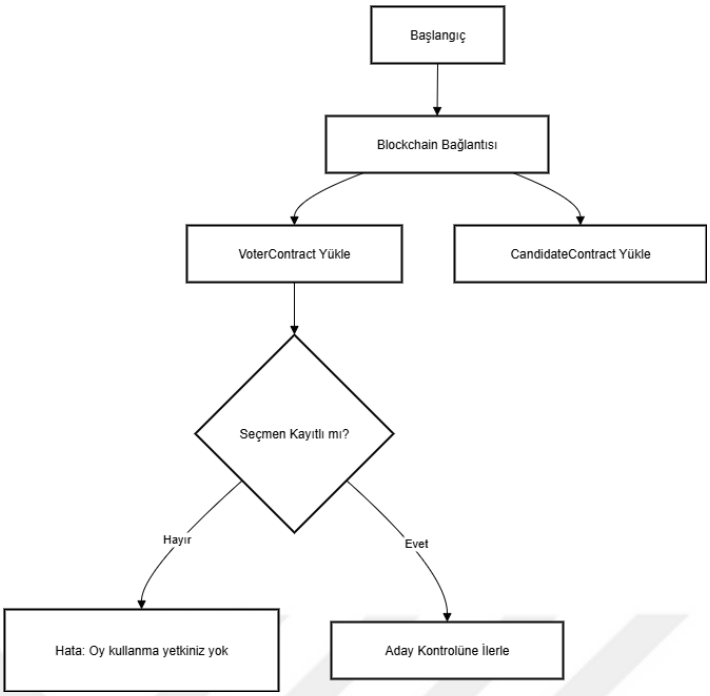
Geliştirilen bu mimari üç temel akıllı sözleşmeden oluşmaktadır:

1. CandidateContract: Sistemde oylamaya katılan adaylara ilişkin bilgilerin yönetildiği sözleşmedir. Adayların kimlik bilgileri ve ek belgeler, IPFS (InterPlanetary File System) üzerinden erişilebilir dosya bağlantıları aracılığıyla bu kontratta depolanmaktadır.

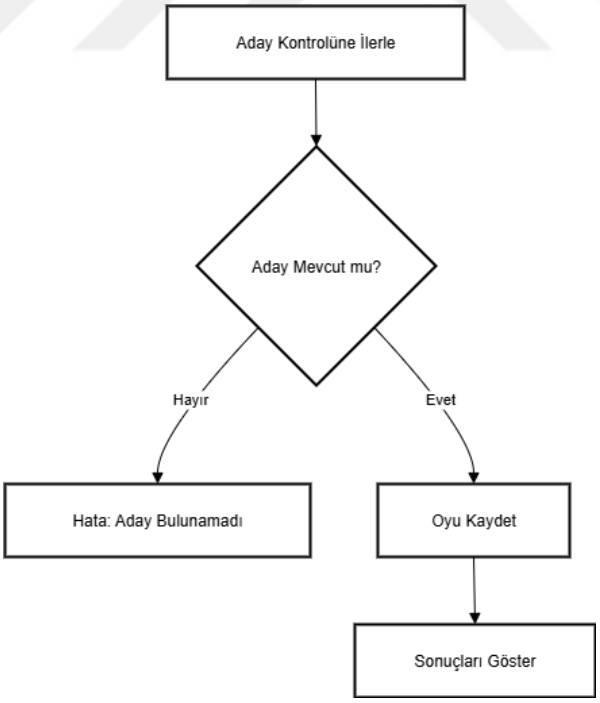
2. VoterContract: Oy kullanma yetkisi bulunan seçmenlerin tanımlandığı ve kimlik doğrulama işlemlerinin yürütüldüğü sözleşmedir. Seçmen adreslerinin önceden kaydedilmesi ve bu adreslerin geçerliliğinin sağlanması bu yapı üzerinden gerçekleştirilir. Ayrıca, toplu seçmen verileri IPFS üzerinden çekilerek sistemde otomatik kayıt yapılmasına da olanak tanınmaktadır.

3. ElectionContract: Seçim sürecinin işleyişinden doğrudan sorumlu olan bu yapı, oy verme işlemlerini başlatma, geçersiz oyları engelleme, geçerli oyları sayma ve sonuçları hesaplama gibi işlevleri yürütmektedir. Aynı zamanda, sistem güvenliğini denetleyen ve sözleşmeler arası koordinasyonu sağlayan merkezi yapı konumundadır.

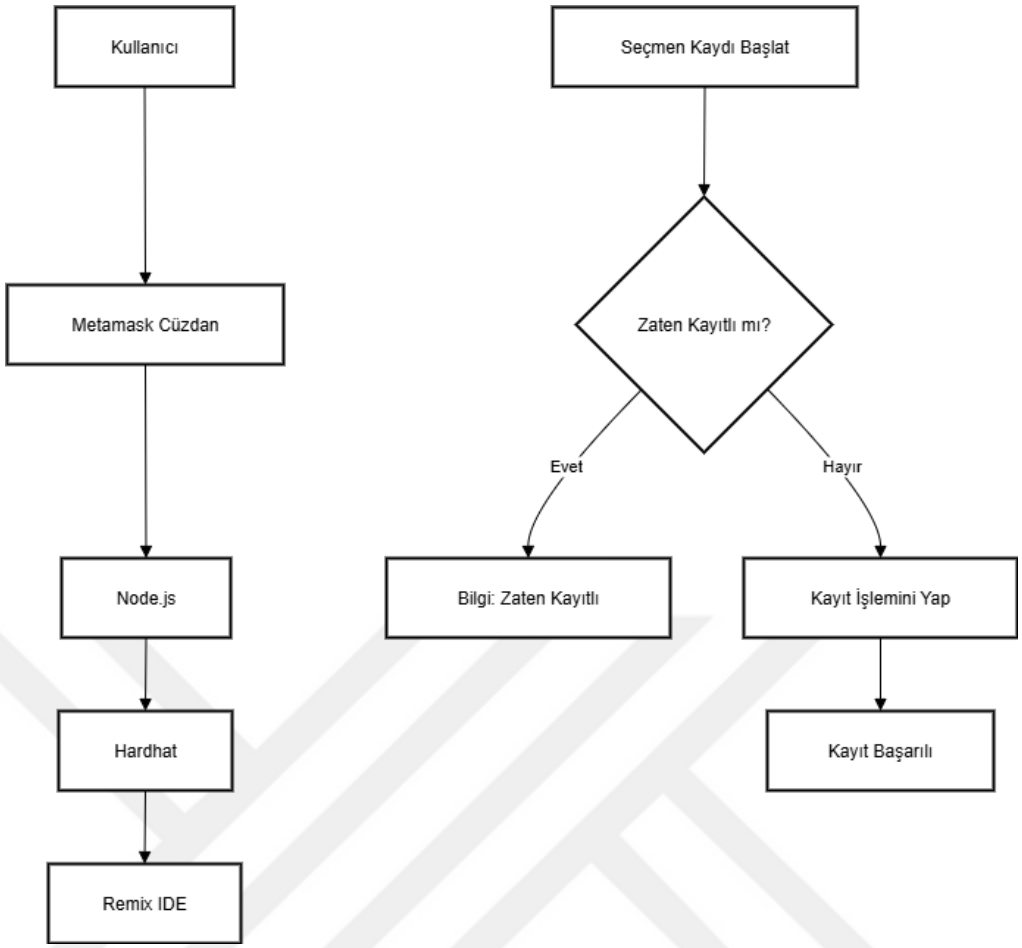
Bu üç sözleşme, sistemin genel işleyişi içinde birbirleriyle bütünleşmiş ve etkileşimli bir biçimde çalışacak şekilde tasarlanmıştır. Yetkilendirme kontrolleri, işlem sıralamaları ve veri güncellemeleri her sözleşme arasında belirlenen mantıksal sınırlar içerisinde gerçekleştirilmekte, böylece sistem bütünlüğü ve güvenilirliği korunmaktadır.



Şekil 4.1: Blok Zincir Bağlantısı ve Seçmen Doğrulama Süreci



Şekil 4.2: Aday Kontrolü ve Oy Kullanma İşlemi



Şekil 4.3: Seçmen Kaydı ve Sistem Bileşenleri

Şekil 4.1, Şekil 4.2 ve Şekil 4.3'teki diyagramlar, kullanıcı arayüzü, akıllı sözleşmeler (ElectionContract, VoterContract, CandidateContract), IPFS entegrasyonu ve Web3 altyapısı arasında kurulan ilişkileri göstermekte; sistemin veri akışı ve bileşenler arası etkileşimini şematik olarak sunmaktadır.

Bu çalışmanın geliştirme sürecinde ise Ethereum tabanlı uygulama geliştirmede yaygın olarak kullanılan çeşitli araç ve teknolojilerden yararlanılmıştır. Bunlar arasında Remix IDE sözleşme testleri için, Hardhat geliştirme çerçevesi ve dağıtım süreçleri için, Node.js istemci taraflı işlemler için ve IPFS ise dağıtık dosya depolama amacıyla kullanılmıştır. Web3.js kütüphanesi ile istemci tarafı, Ethereum ağıyla etkileşimli hale getirilmiştir.

Sistemin öncelikli hedefi, yalnızca önceden tanımlanmış ve tekil cüzdan adreslerinden gelen oyların kabul edildiği, birden fazla oy kullanımının teknik olarak engellendiği,

aday verilerinin dağıtık biçimde IPFS ağı üzerinden saklandığı ve nihai seçim sonuçlarının tüm katılımcılar tarafından okunabilir biçimde yayınlandığı bir seçim altyapısı sunmaktır. Bu mimariyle, yalnızca işlevsel değil; aynı zamanda güvenli, şeffaf ve müdahaleye kapalı bir dijital oylama deneyimi oluşturulması amaçlanmıştır.

4.4. Kontratların Tasarımı (Solidity Katmanı)

Bu bölümde Solidity ile Remix IDE platformunda tasarlanan akıllı kontratlar kapsamlı bir şekilde anlatılacaktır.

4.4.1. Candidate Contract

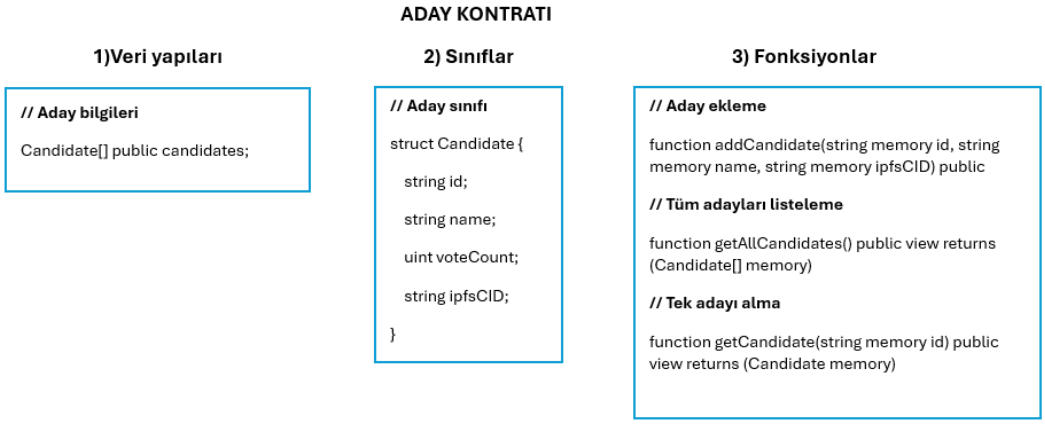
CandidateContract, sistem içerisinde seçim adaylarının tanımlanması, saklanması ve yönetilmesi amacıyla yapılandırılmış temel veri sözleşmesidir. Bu kontratta her aday, kendisine özgü bir kimlik bilgisi (ID) ve bu adaya ait detaylı bilgilerin saklandığı, IPFS (InterPlanetary File System) üzerinden erişilen bir içerik tanımlayıcısı (Content Identifier – CID) ile sistemde temsil edilmektedir. Bu yaklaşım, zincir üzerinde büyük boyutlu veri barındırmaktan kaçınılarak, belgelerin dağıtık bir dosya sisteminde tutulmasını ve bu yolla hem şeffaflığın hem de dışarıdan denetlenebilirliğin sağlanmasını mümkün kılmaktadır.

Bu sözleşme, iki temel işlev aracılığıyla çalışmaktadır:

getCandidate(string memory candidateId): Belirtilen kimliğe sahip adaya ilişkin verileri geri döndürür. Eğer ilgili ID'ye karşılık gelen bir kayıt sistemde bulunmuyorsa, işlem başarısız olur ve kontrol dışı erişim engellenir.

getAllCandidates(): Sistemde kayıtlı olan tüm adayları içeren bir liste döndürerek, bu verilerin gerek oy verme arayüzünde görüntülenmesi gerekse sonuçların hesaplanması süreçlerinde kullanılmasına olanak tanır.

Adaylara ilişkin veriler, Solidity dilinde tanımlı struct yapısı kullanılarak saklanmakta; aday kimliğini anahtar olarak kullanan bir mapping yapısı aracılığıyla sisteme erişim sağlanmaktadır. Bu yapı sayesinde hem veri okuma süreçlerinde performans avantajı elde edilmekte hem de sistem genelinde mantıksal bütünlük korunmaktadır.



Şekil 4.4: CandidateContract Veri Yapısı ve Fonksiyonları

Şekil 4.4'teki diyagramda, CandidateContract sözleşmesinde tanımlı temel veri yapıları, adaylara ilişkin sınıf tanımı (Candidate struct) ve sözleşme kapsamında sunulan fonksiyonel işlemler sınıflandırılarak sunulmuştur. Aday bilgileri mapping veri yapısı ile depolanmakta, addCandidate, updateCandidate, removeCandidate gibi fonksiyonlar aracılığıyla bu verilere erişim ve yönetim sağlanmaktadır. Şekil, sözleşme mimarisinin işlevsel bileşenlerini özetleyerek, sistemin veri işleme kapasitesini görsel olarak açıklamayı amaçlamaktadır.

CandidateContract, sistemdeki diğer sözleşmeler tarafından yalnızca salt okunur biçimde kullanılacak şekilde tasarlanmıştır. Böylece adaylara ait veriler üzerinde doğrudan değişiklik yapılmasının önüne geçilmekte; bütünlük, IPFS üzerinden sağlanan dış bağlantılar aracılığıyla korunmaktadır. CID tabanlı bu yapı, dağıtık bir veri modelinin güvenilirliğini Ethereum ağının doğrulama mekanizması ile birleştirerek, seçim sistemine yönelik şeffaf ve güvenli bir veri altyapısı sunmaktadır.

4.4.2. Voter Contract

VoterContract, blok zincir tabanlı seçim sisteminde oy kullanma yetkisine sahip seçmenlerin adres bilgilerinin tanımlanması ve doğrulanması süreçlerini yöneten temel sözleşmedir. Bu sözleşmenin temel işlevi, yalnızca önceden yetkilendirilmiş Ethereum adreslerinin oy kullanabilmesini garanti altına alarak seçim güvenliğini sağlamaktır.

VoterContract'ı benzer yapılardan ayıran önemli özelliklerden biri, zincir dışı kimlik tanımlayıcılarını (örneğin öğrenci numaralarını) blok zincirle uyumlu adreslere dönüştürebilmesidir. Bu işlev, Ethereum ağı üzerinde benzersiz ve doğrulanabilir adresler üretilmesini mümkün kılmakta ve sisteme dış dünyadan kimlik entegrasyonu sağlamaktadır. Bu dönüştürme işlemi, generateAddress(string memory studentId) fonksiyonu ile gerçekleştirilmekte olup, SHA-3 (keccak256) algoritmasına dayalı bir özetleme yöntemi kullanılarak address(uint160(uint256(...))) formülü üzerinden Ethereum adres formatı elde edilmektedir.

Sözleşme kapsamında yer alan temel fonksiyonlardan bazıları şunlardır:

1. registerVoter(address _voter): Bu fonksiyon, yalnızca sistem yöneticisi (admin) yetkisine sahip adresler tarafından çağrılabilmekte olup, bireysel olarak belirli bir adresin seçmen listesine kaydedilmesini sağlar.

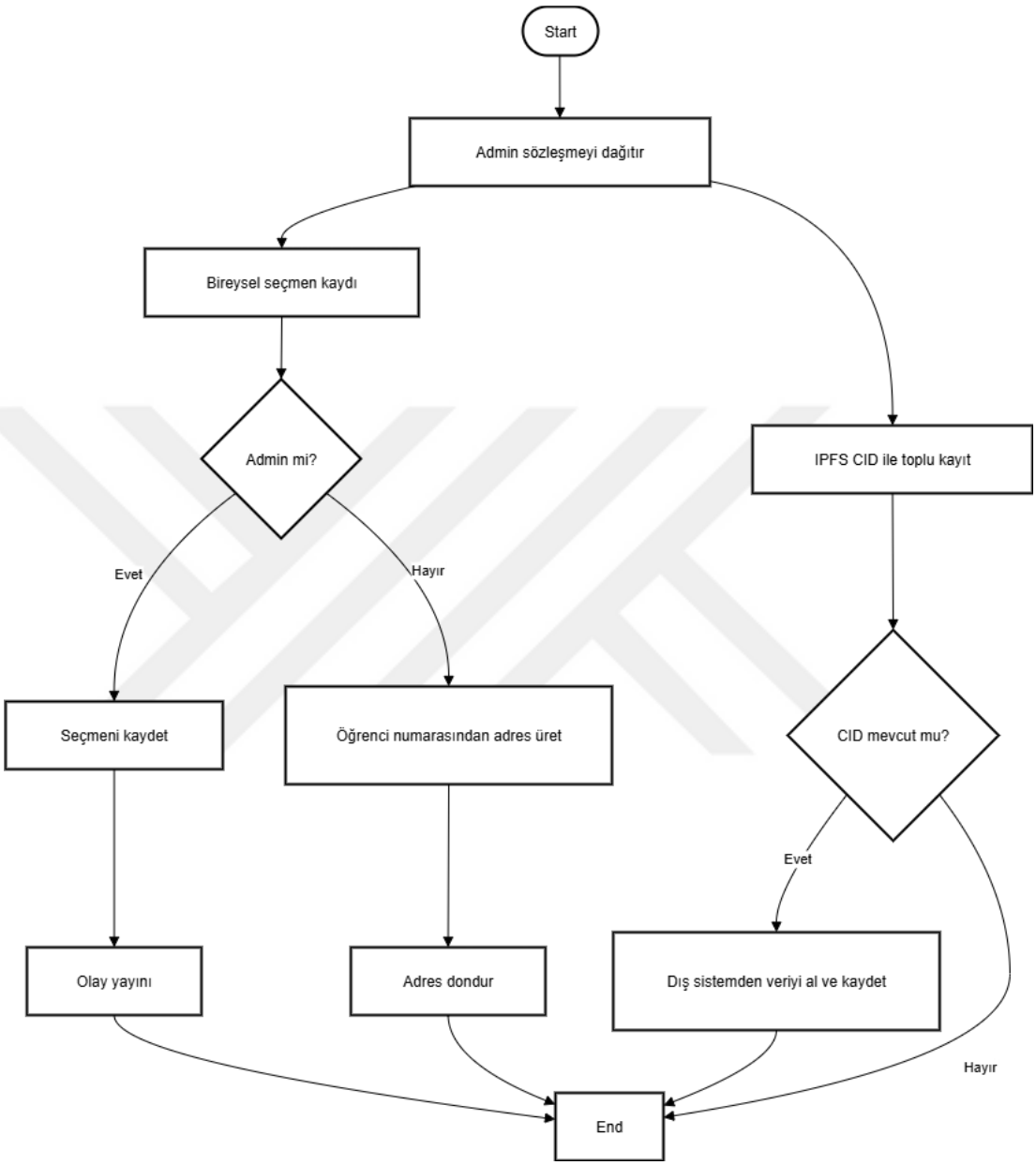
2. registerVotersWithCID(string memory _cid): Bu yapı, IPFS üzerinde saklanan ve içinde öğrenci numaralarının yer aldığı bir JSON dosyasına ait CID bilgisinin sisteme kaydedilmesini sağlar. Söz konusu dosyanın içeriği zincir dışı (off-chain) bir istemci aracılığıyla işlenmekte, adres üretimi ve kaydı daha sonra zincir üzerine yansıtılmaktadır. Bu durum, IPFS ile blok zincir arasında güvenli ve izlenebilir bir veri köprüsü kurulmasını sağlamaktadır.

3. isRegisteredVoter(address _voter): Bu fonksiyon, belirli bir Ethereum adresinin sistemde kayıtlı olup olmadığını kontrol etmek amacıyla kullanılır. Seçim sürecini yöneten sözleşme (ElectionContract), oy kullanma işlemlerinde bu fonksiyonu çağırarak yetkilendirme kontrolünü gerçekleştirir.

Sözleşme içerisinde, mapping(address => bool) veri yapısı kullanılarak her bir adresin kayıtlı olup olmadığı hızlı ve güvenilir bir biçimde kontrol edilebilmektedir. Bu yapı sayesinde sistem, her oy verme girişimini adres bazında değerlendirerek yalnızca önceden tanımlanmış seçmenlerin işlem yapmasına izin verir. Ayrıca onlyAdmin isimli özel yetki kontrol mekanizması (modifier), yalnızca yönetici düzeyindeki kullanıcıların kayıt işlemlerini gerçekleştirmesine olanak tanıyarak sisteme dış müdahale riskini azaltır.

VoterContract, zincir dışı kaynaklardan gelen kimlik verilerinin blok zincire entegre edilmesi sürecinde kritik bir rol üstlenmektedir. Özellikle Node.js gibi uygulamalar

aracılığıyla gelen öğrenci numaralarının Ethereum adreslerine dönüştürülerek sisteme kaydedilmesi, bu yapının blok zincir dışı dünyayla kurduğu işlevsel bağı güçlendirmektedir.



Şekil 4.5: VoterContract Sözleşmesine Ait Temel İşlem Akışı Diyagramı

Şekil 4.5'teki diyagram, VoterContract sözleşmesinin üç temel işlevine ilişkin işlem adımlarını göstermektedir. Bireysel seçmen kaydı sürecinde yalnızca sistem yöneticisinin işlem yapmasına izin verilirken, zincir dışı öğrenci numaralarının Ethereum adreslerine dönüştürülmesi ve IPFS üzerinden alınan toplu seçmen verisinin kaydedilmesi de sistemin dış veri kaynaklarıyla bütünleşmiş çalışmasını mümkün

kılmaktadır. Bu iş akış diyagramında üç sürecin de başlangıç ve bitiş noktaları ile kritik kontrol adımları (yetkilendirme, veri doğrulama) açık biçimde sunulmuştur.

Şekilde sunulan işlem akışına bağlı olarak, sistem yalnızca içsel doğruluğa değil; aynı zamanda dış sistemlerle uyumlu, ölçeklenebilir ve sürdürülebilir bir kimlik doğrulama altyapısına da kavuşmuş olmaktadır.

4.4.3. Election Contract

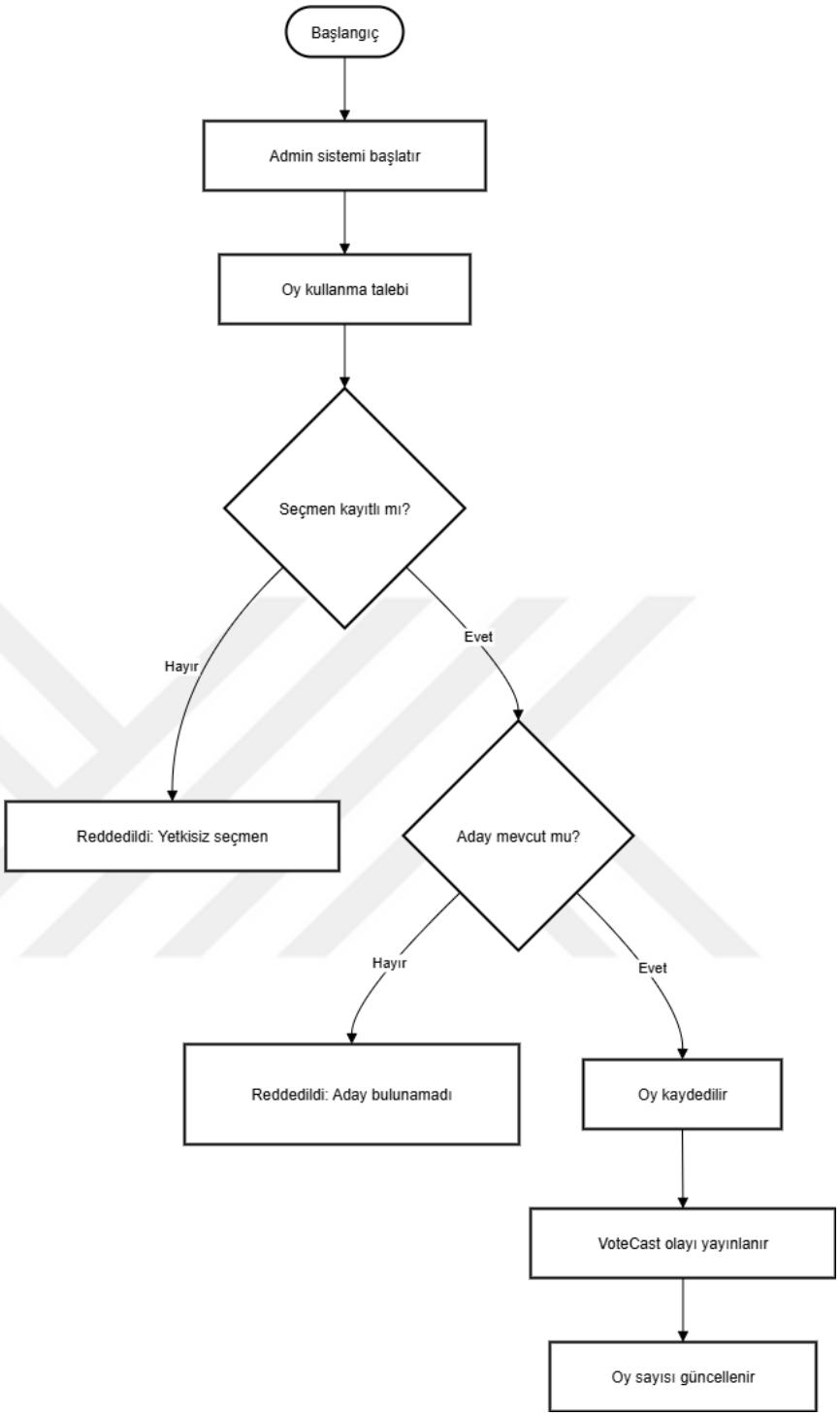
ElectionContract, blok zincir tabanlı seçim sisteminin merkezi bileşeni olup, oy verme işlemlerinin yürütülmesini, güvenlik kontrollerinin uygulanmasını ve seçim sonuçlarının hesaplanarak yayımlanmasını sağlayan temel sözleşmedir. Bu yapı hem sistem bütünlüğünü hem de seçim sürecinin şeffaflığını koruma amacı taşımaktadır. ElectionContract, diğer iki kritik sözleşme olan VoterContract ve CandidateContract ile bütünleşmiş biçimde çalışmakta; böylece aday bilgileri, seçmen doğrulaması ve oy kaydı gibi işlemler arasında güvenli bir iş akışı oluşturulmaktadır.

Sözleşme içerisinde yer alan başlıca işlevsel bileşenler aşağıdaki gibidir:

initialize(VoterContract_voterContract,CandidateContract_candidateContract: Seçim sisteminin başlatılması aşamasında çağrılan bu fonksiyon, ilgili yardımcı sözleşmelerin (seçmen ve aday modülleri) ana seçim sistemine bağlanmasını sağlar. Bu yapı, sistemin modülerliğini korurken, dış sözleşmelerle koordineli çalışmasına olanak tanır.

vote(string memory candidateId): Seçmenlerin belirli bir aday için oy kullanmasını sağlayan temel işlevdir. Bu fonksiyon, üç aşamalı bir kontrol mekanizması içerir: yalnızca sistem yöneticisi tarafından başlatılmış bir yapıda çalıştırılabilir; yalnızca daha önce kayıt altına alınmış seçmen adresleri bu işlemi gerçekleştirebilir ve her adres yalnızca bir kez oy kullanabilir. Bu kontroller, sırasıyla onlyAdmin, onlyRegisteredVoter ve onlyOnce isimli özel modifier'lar aracılığıyla uygulanmaktadır.

Şekil 4.6'daki diyagram, oy kullanma sürecinin sistem üzerindeki temel işlem akışını şematik olarak göstermektedir.



Şekil 4.6: ElectionContract Oy Verme Süreci Diyagramı

Bu diyagram, Ethereum tabanlı seçim sisteminde seçmenin bir adaya oy kullanması sırasında gerçekleşen işlem adımlarını özetlemektedir. Fonksiyonun yürütülmesi yalnızca kayıtlı seçmenlere açık olup, adayın sistemde mevcut olduğunun doğrulanmasının ardından oy kaydedilir ve VoteCast olayı yayınlanır. Bu yapı, seçim

güvenliğini sağlarken zincir üzerindeki işlemlerin izlenebilirliğini de garanti altına almaktadır.

Diğer yandan, sistemin şeffaflık ilkesine uygun olarak tasarlanan getResults() fonksiyonu, seçim sonuçlarının tüm kullanıcılarla doğrudan ve denetlenebilir biçimde paylaşılmasını mümkün kılmaktadır.

getResults(): Sisteme tanımlı tüm adayların kimlik bilgilerini ve aldıkları toplam oy sayısını dönen bu fonksiyon, sonuçların blok zincir üzerinden izlenebilirliğini sağlayarak sistemin şeffaflık ilkesini pekiştirir.

getCandidateCID(string memory candidateId): Belirtilen adaya ait IPFS tabanlı içerik tanımlayıcısını (CID) geri döndürür. Özellikle adaylara ait belge ya da görsel tabanlı materyallerin paylaşımı bu mekanizma aracılığıyla gerçekleştirilir.

Sözleşme bünyesinde tanımlı olan kontrol yapıları seçim güvenliği açısından kritik rol oynamaktadır:

onlyAdmin: Seçim sisteminin kurulumu ve yönetimi yalnızca yetkili yönetici (admin) adresi tarafından yürütülebilir.

onlyRegisteredVoter: Seçmen kimlik doğrulamasını gerçekleştiren bu yapı, yalnızca önceden tanımlanmış adreslerin oy kullanmasına izin verir.

onlyOnce: Her seçmenin yalnızca bir kez oy kullanmasını garanti altına alır. Bu kontrol, hasVoted isimli bir mapping yapısı aracılığıyla uygulanmaktadır.

ElectionContract, yalnızca oy kayıtlarını güvenli şekilde gerçekleştirmekle kalmaz; aynı zamanda seçim süreci boyunca olası sahtecilik girişimlerini engelleyen bir dizi teknik filtreleme mekanizmasını da aktif olarak devreye sokar. Her bir oy kullanımı, zincir üzerinde bir olay (event) olarak kaydedilmekte; böylece dış denetime açık, izlenebilir ve değiştirilemez bir kayıt altyapısı oluşturulmaktadır.

Sonuç olarak bu sözleşme, blok zincir temelli elektronik oylama sisteminin güvenilirlik, şeffaflık ve adil işlem ilkelerine uygun şekilde çalışmasını sağlayan yapısal bir çekirdek modül niteliğindedir.

4.5. Geliştirme Ortamı ve Teknik Entegrasyonlar

Merkeziyetsiz yapılar temel alınarak geliştirilen oylama sistemi, zincir içi (on-chain) ve zincir dışı (off-chain) bileşenlerin birlikte ve uyum içinde çalışabilmesini sağlamak amacıyla çeşitli yazılım araçları, geliştirme ortamları ve entegrasyon çözümleri kullanılarak yapılandırılmıştır. Sistem mimarisi, yalnızca akıllı sözleşmelerin dağıtık defter altyapısı üzerinde güvenli biçimde işlemesini değil, aynı zamanda kullanıcı verilerinin harici kaynaklardan alınarak doğrulanmasını mümkün kılan çok katmanlı bir yapıya da dayanmaktadır.

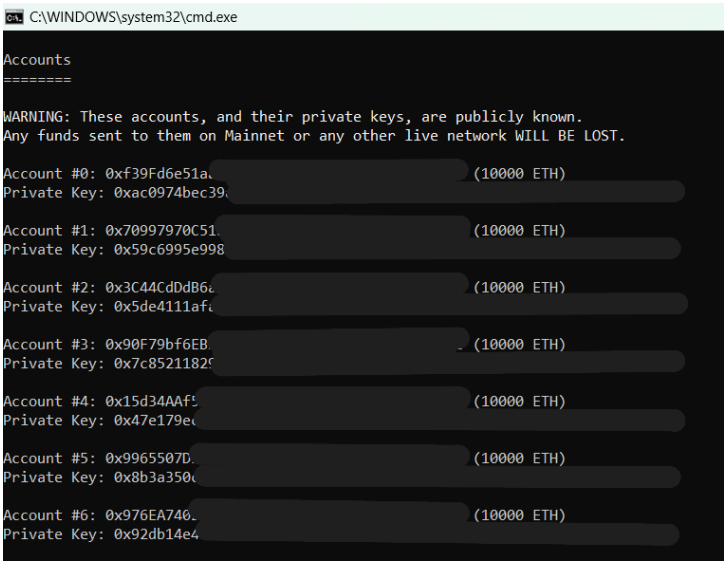
Bu bölümde; Hardhat geliştirme çatısı, IPFS dosya sistemi, Node.js tabanlı betikler ve Remix IDE platformu gibi unsurların proje kapsamında nasıl kullanıldığı detaylı biçimde açıklanmıştır.

4.5.1. Hardhat Ortamı

Hardhat, Ethereum tabanlı uygulamaların geliştirme sürecinde yaygın biçimde kullanılan, modüler ve esnek bir çalışma ortamı sunan açık kaynaklı bir geliştirme aracıdır. Yerel bir blok zincir ağı simülasyonu sağlayarak sözleşme derleme, test senaryoları oluşturma, dağıtım işlemlerini yönetme ve hata ayıklama gibi işlevleri tek bir platformda birleştirir. Özellikle çok bileşenli sistemlerde, sözleşmelerin hem zincir içi hem de zincir dışı etkileşimlerinin test edilebilmesini mümkün kılan yapısıyla, uygulama geliştirme süreçlerinin merkezinde yer alır.

Bu tez kapsamında Hardhat, seçim sistemi mimarisinin test edilmesi, çeşitli senaryolar altında fonksiyonel doğrulama yapılması ve zincir dışı veri kaynaklarıyla etkileşimlerin kontrol edilmesi amacıyla kullanılmıştır. Hardhat'in sunduğu başlıca işlevler aşağıdaki şekilde özetlenebilir:

Yerel blok zincir ortamı kurulumu: Hardhat Network kullanılarak, işlemlerin zincir üzerindeki etkilerinin simüle edilebildiği bir geliştirme ağı oluşturulmuş; sözleşmeler burada deploy edilerek işlem davranışları gözlemlenmiştir.



Şekil 4.7: Hardhat Yerel Ağı Üzerinde Oluşturulan Hesaplar ve Anahtarlar

Şekil 4.7'deki çıktı, Hardhat yerel ağı başlatıldığında otomatik olarak oluşturulan varsayılan Ethereum adreslerini ve ilgili özel anahtarları göstermektedir. Geliştirme sürecinde bu adresler test amaçlı kullanılmış olup, gerçek ağlarda kullanılmaları durumunda güvenlik riski taşımaktadır. Bu hesaplar, test işlemlerinde gaz ücreti gerektirmeyen ortamda işlem yapmak amacıyla yapılandırılmıştır.

Akıllı sözleşme dağıtımı ve testleri: npx hardhat test komutu ile chai ve mocha kütüphaneleri aracılığıyla yazılmış birim testler çalıştırılmış; böylece sözleşme fonksiyonlarının güvenilirliği ve doğru çalışıp çalışmadığı sistematik olarak değerlendirilmiştir.

Sepolia test ağı entegrasyonu: Gerçek Ethereum test ağı olan Sepolia'ya Infura RPC bağlantısı üzerinden erişim sağlanmış; sözleşmeler burada da test edilerek sistemin gerçek ağ koşullarında davranışı doğrulanmıştır.

Zincir dışı sistem etkileşimi: Node.js ile yazılmış index.js script'i, zincir dışındaki verileri işleyerek Hardhat üzerinden çalışan sözleşmelere kayıt işlemleri gerçekleştirmiştir. Böylece dış veri kaynaklarının blok zincirle entegrasyonu başarıyla sağlanmıştır.

Hardhat tabanlı projelerde yaygın olarak benimsenen klasör yapısı bu çalışmada da uygulanmıştır. Aşağıda örnek bir proje dizini yer almaktadır:

```
project-root/
├── contracts/
│   ├── CandidateContract.sol
│   ├── VoterContract.sol
│   └── ElectionContract.sol
├── scripts/
│   ├── deploy.js
│   └── interact.js
├── test/
│   └── election.test.js
├── .env
├── hardhat.config.js
└── index.js
```

Bu yapı sayesinde sözleşmelerin geliştirimi, dağıtımı, test edilmesi ve zincir dışı betiklerle etkileşimi birbirinden ayrılmış; modüler, okunabilir ve sürdürülebilir bir proje mimarisi oluşturulmuştur.

Projenin güvenlik açısından kritik olan yapılandırma bilgileri, .env dosyasında tutulmuştur. Bu sayede hassas bilgiler doğrudan kaynak kodda yer almamış hem geliştirici güvenliği sağlanmış hem de sürüm kontrol sistemlerinde bilgi sızıntısı önlenmiştir. Dosyada yer alan temel bilgiler şunlardır:

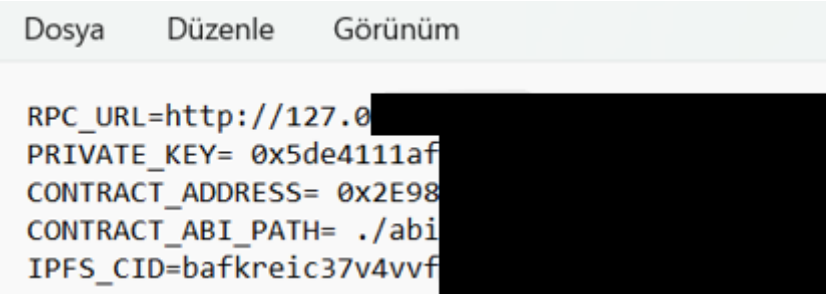
Kullanıcıya ait private key (maskelenmiş),

Infura RPC URL (Sepolia ağı için),

VoterContract sözleşme adresi,

IPFS CID değeri.

Şekil 4.8'de ifade edildiği üzere bu yapı sayesinde, sistem yalnızca zincir içi test süreçlerinin etkin biçimde yürütülmesini değil; aynı zamanda dış veri kaynaklarıyla bütünleşik çalışan, güvenilir ve senaryo tabanlı test edilebilir bir geliştirme ortamının oluşturulmasını mümkün kılmıştır.



Şekil 4.8: .env Yapılandırma Dosyası

4.5.2. IPFS Entegrasyonu

IPFS (InterPlanetary File System), verilerin merkeziyetsiz biçimde saklanmasını sağlayan açık kaynaklı bir dosya sistemidir. Blok zincir tabanlı sistemlerde, özellikle büyük boyutlu veri depolama işlemleri (örneğin kimlik belgeleri, aday bilgileri, seçmen listeleri) zincir üzerinde doğrudan gerçekleştirilemediğinden, IPFS gibi dış depolama çözümleri kritik önem taşımaktadır.

Bu projede seçmen bilgilerini içeren JSON formatındaki veri, Pinata platformu aracılığıyla IPFS ağına yüklenmiştir. IPFS'e yüklenen bu verinin CID (Content Identifier) değeri, VoterContract içinde referans olarak saklanmakta; böylece blok zincir üzerinde yalnızca veri kaynağına erişim noktası tutulmaktadır.

CID değeri, zincire aşağıdaki fonksiyon ile aktarılmıştır:

function registerVotersWithCID(string memory _cid) public onlyAdmin

Bu fonksiyon yalnızca veri kaydını gerçekleştirmekte; IPFS üzerindeki verinin içeriği ise Solidity ortamında doğrudan işlenememektedir. Bunun temel nedeni, Solidity'nin zincir dışı HTTP istekleri gerçekleştirememesi ve dış veri kaynaklarını native olarak okuyamamasıdır. Bu nedenle, IPFS üzerindeki JSON verisi, zincir dışı bir istemci (off-chain client) olan Node.js aracılığıyla okunarak işlenmiş ve uygun Ethereum adreslerine dönüştürülmüştür.

4.5.3. Node.js Tabanlı Otomatik Seçmen Kaydı

Dağıtık defter tabanlı sistemlerde zincir içi veriler genellikle sabit ve sınırlıdır. Ancak uygulamanın gerçek dünya verileriyle bütünleşmiş çalışabilmesi için, zincir dışı (off-chain) veri kaynaklarının güvenilir bir biçimde işlenip sözleşmelere entegre edilmesi

gerekmektedir. Bu bağlamda, çalışmada kullanılan VoterContract'a seçmen verilerinin kaydedilmesi amacıyla özel bir Node.js betiği geliştirilmiştir.

Bu betik, index.js dosyası içerisinde yapılandırılmış olup aşağıdaki iş akışına sahiptir:

a. IPFS Entegrasyonu

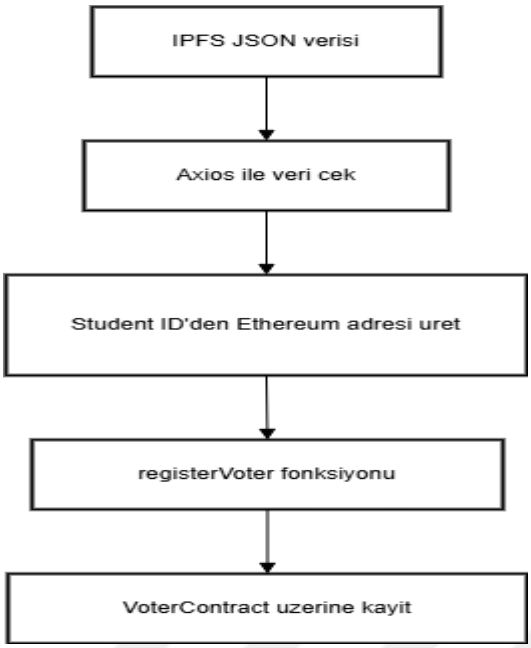
Sistemdeki işlemlerin başlatılabilmesi için öncelikle seçmen verilerine erişim sağlanmalıdır. İlk aşamada, Pinata platformu üzerinden IPFS ağına yüklenmiş olan seçmen listesi (JSON formatında) Axios kütüphanesi yardımıyla okunur. Burada kullanılan CID (Content Identifier), .env dosyası üzerinden betiğe aktarılmaktadır. Bu sayede kod ile yapılandırma dosyası arasında gevşek bağlılık (loose coupling) sağlanmıştır.

b. Adres Üretimi

function generateAddress(string memory studentId) public pure returns (address) {

return address(uint160(uint256(keccak256(abi.encodePacked(studentId))))); }

Yukarıda belirtilen bu yapı ile her öğrenci numarası, generateAddress() fonksiyonu aracılığıyla deterministik bir Ethereum adresine dönüştürülür. Bu fonksiyon SHA-3 (keccak256) algoritması kullanılarak hash işlemi gerçekleştirir ve sonuç, address(uint160(uint256(...))) zinciriyle Ethereum adresine çevrilir. Bu yöntem, dış kimliklerin blok zincir sistemine uyumlu hâle getirilmesini sağlamaktadır.



Şekil 4.9: Zincir Dışı Seçmen Verisinin Zincir İçi Kayıt Süreci

Şekil 4.9'da IPFS üzerinde saklanan öğrenci verisinin Node.js betiği ile alınması, generateAddress() fonksiyonu ile Ethereum adresine dönüştürülmesi ve registerVoter() çağrısı ile blok zincir üzerindeki VoterContract 'a kaydedilmesi süreci gösterilmektedir. Bu yapı, sistemin zincir dışı kaynaklarla bütünleşik şekilde çalışmasını sağlayan temel iş akışını temsil etmektedir.

Bu şematik gösterim, özellikle veri üretiminden kayıt işlemine kadar geçen sürecin modüler ve otomasyon tabanlı şekilde çalıştığını açıkça ortaya koymaktadır.

c. Seçmen Kaydı

Üretilen adresler, registerVoter() fonksiyonu ile VoterContract üzerine kaydedilir. Bu işlem yalnızca admin yetkisine sahip hesaplarca yürütülebilmektedir. Her kayıt işlemi sonucunda terminalde anlık bir çıktı üretilecek şekilde betik yapılandırılmıştır.

Script'in çalıştırılması sırasında geliştiriciye yönelik işlem sonuçları terminal ekranında aşağıdaki gibi net bir şekilde sunulmaktadır:

5203230001 → 0xA87c...12eF ==> Kayıt edildi

5203230002 → 0xB13d...9aD4 ==> Zaten kayıtlı

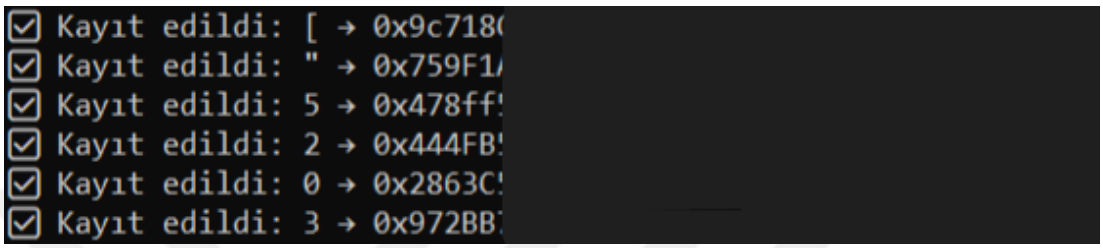
Bu çıktı sayesinde:

Her öğrencinin sisteme başarıyla kaydolup kaydolmadığı doğrulanabilmekte,

Çakışmalar veya mükerrer girişler hızlıca fark edilmekte,

Geliştirme ve hata ayıklama süreci kolaylaşmaktadır.

Görsel olarak bu sürece ait çıktı, Şekil 4.10'da sunulmuştur:



Şekil 4.10: Seçmen Kayıt Betiğinin Terminal Çıktısı

Görselde, IPFS üzerinden alınan öğrenci numaralarının Ethereum adreslerine çevrilerek sistemdeki VoterContract sözleşmesine başarıyla kaydedildiği sürece ait terminal çıktısı yer almaktadır. Bu yapı, zincir dışı sistem ile zincir içi sözleşme arasında doğrudan entegrasyonun gerçekleştiğini belgelemektedir.

Node.js betiği, Hardhat yerel ağı çalışırken aşağıdaki komut ile çalıştırılmıştır:

node index.js

Bu sırada Hardhat ağı (örneğin npx hardhat node) aktif olarak çalışmalıdır. Betik içerisinde sözleşme ile etkileşim kurulabilmesi için ABI dosya yolu ve kontrat adresi .env dosyasından çekilmektedir. Bu durum, sistemin hem taşınabilirliğini hem de güvenliğini artırmaktadır.

Sonuç olarak geliştirilen bu otomasyon sistemi, çok sayıda seçmenin blok zincir sistemine hızlı ve hatasız biçimde eklenmesini sağlayarak süreçteki manuel müdahale ihtiyacını ortadan kaldırmıştır. Ayrıca zincir dışı sistemler ile zincir içi sözleşmeler arasında senkronize bir iletişim kurulmuş; bu da uygulamanın ölçeklenebilir, tekrar edilebilir ve geliştirici dostu bir yapıya kavuşmasını mümkün kılmıştır.

4.6. Manuel Test Süreci: Remix IDE Üzerinde Fonksiyonel Doğrulama

Geliştirilen uygulamanın işlevselliğinin yalnızca otomatik testler üzerinden değil, doğrudan kullanıcı etkileşimiyle de değerlendirilmesi, sistemin bütüncül güvenilirliği açısından büyük önem taşımaktadır. Bu doğrultuda, proje kapsamında geliştirilen üç akıllı sözleşme, Remix IDE ortamında sırasıyla dağıtılarak manuel senaryolar üzerinden test edilmiş ve her bir modülün görevini beklenen şekilde yerine getirip getirmediği adım adım gözlemlenmiştir.

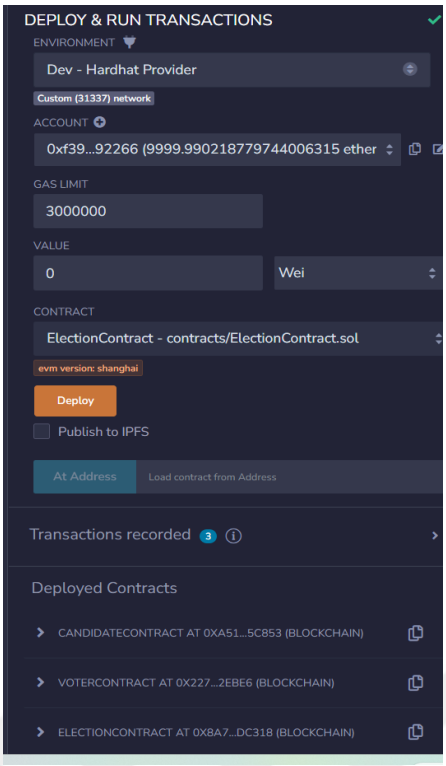
4.6.1. Sözleşmelerin Sıralı Şekilde Dağıtılması

İlgili uygulamanın manuel test süreci kapsamında ilk adım, proje içerisinde geliştirilen üç temel sözleşmenin (CandidateContract , VoterContract ve ElectionContract) Remix IDE aracılığıyla test ağına dağıtılmasıdır. Remix IDE, hem tarayıcı üzerinden erişilebilen web tabanlı bir geliştirme ortamı olması, hem de JavaScript VM, Injected Web3 ve Hardhat gibi çeşitli ağlara bağlanabilme özelliği ile sözleşme etkileşimlerinin test edilmesinde kullanıcıya büyük kolaylık sağlamaktadır.

Bu çalışmada, doğrudan Hardhat yerel ağı ile entegre çalışan Remix ortamı tercih edilmiştir. Hardhat ağı aktif durumdayken Remix arayüzü üzerinden Injected Web3 seçeneği aracılığıyla bağlantı kurulmuş ve sözleşmeler sırasıyla deploy edilmiştir. Deploy işlemleri tamamlandıktan sonra her bir sözleşme, Ethereum Virtual Machine üzerinde kendisine özgü bir adres ile temsil edilmeye başlanmıştır.

Özellikle ElectionContract yapısının fonksiyonel olarak devreye alınabilmesi için, sistemde daha önce oluşturulan CandidateContract ve VoterContract 'a ait adres bilgilerinin bu sözleşmeye tanımlanması zorunludur. Bu işlem, sözleşmenin içinde tanımlı olan initialize(VoterContract , VoterContract, fonksiyonu aracılığıyla gerçekleştirilmiştir. Fonksiyon çağrısı yalnızca sistem yöneticisine (admin) yetkili adresler tarafından yapılabildiğinden, dağıtım sırasının doğru kurgulanması ve sözleşme adreslerinin eksiksiz alınması test sürecinde kritik rol oynamaktadır.

Şekil 4.11'de söz konusu sözleşmelerin Remix IDE üzerinde deploy edilmesini ve adres üretimini gösteren örnek bir ekran görüntüsü sunulmaktadır:



Şekil 4.11: Remix IDE Üzerinden Akıllı Sözleşmelerin Dağıtımı ve Adres Ataması

Görselde, CandidateContract, VoterContract ve CandidateContract, dağıtımının tamamlandığı Remix IDE arayüzü yer almaktadır. Her bir sözleşmenin karşısında, Ethereum sanal makinesinde oluşturulmuş benzersiz bir adres görünmektedir. Bu adresler, ElectionContract içerisinde tanımlanan initialize(...) fonksiyonu ile ilişkilendirilmiş ve oylama sisteminin bütünsel çalışması sağlanmıştır.

Bu ekran çıktısı, test ortamında sözleşmelerin başarıyla deploy edildiğini ve birbirleriyle bütünleşik biçimde çalışabilmesi için gerekli olan adres bağlantılarının doğru şekilde sağlandığını görsel olarak belgelemektedir.

4.6.2. initialize(...) Fonksiyonu ile Sistem Bağlantılarının Kurulması

ElectionContract, sistemin işlevsel olarak çalışabilmesi için diğer iki temel modül olan VoterContract ve CandidateContract ile bütünleşik bir şekilde çalışmak zorundadır. Bu bütünleşme, doğrudan fonksiyon çağrısı üzerinden değil; bu sözleşmelerin sistemde oluşturulmuş adreslerinin ElectionContract içerisine atanması yoluyla sağlanmaktadır. Bu işlem, yalnızca sistem yöneticisi (admin) yetkisine sahip bir adres

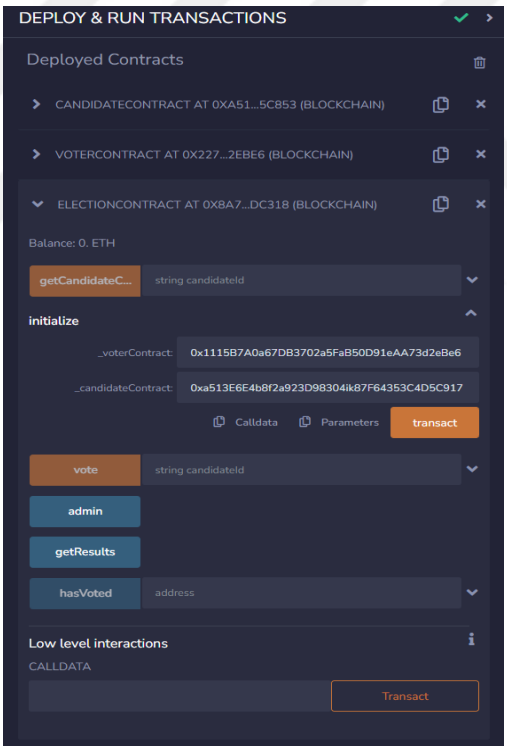
tarafından çağrılabilen initialize(VoterContract _voterContract, CandidateContract _candidateContract) fonksiyonu aracılığıyla gerçekleştirilir.

Bu fonksiyon, sistemin kurulum sürecinde kritik bir yapılandırma adımını temsil etmektedir. Fonksiyon çağrıldığında:

VoterContract'ın adresi, seçim sisteminin kimin oy kullanmaya yetkili olduğunu belirlemesini sağlar.

CandidateContract'ın adresi, hangi adayların yarıştığına dair referans veriyi sağlamış olur.

Bu iki referans olmadan ElectionContract, oyları doğru şekilde eşleştiremez veya doğrulama işlemlerini gerçekleştiremez.



Şekil 4.12: initialize(...) Fonksiyonunun Remix IDE Üzerinden Çağrılması

Şekil 4.12'de ElectionContract'ın initialize(...) fonksiyonu aracılığıyla, sistemde daha önce deploy edilmiş olan VoterContract ve CandidateContract sözleşmelerine ait adreslerin parametre olarak atanması süreci yer almaktadır. Bu işlem, yalnızca admin yetkisine sahip bir kullanıcı tarafından gerçekleştirilebilir ve seçim sistemi içinde yer

alan oy kullanma ve sonuç hesaplama işlemlerinin etkinleşebilmesi için zorunlu bir yapılandırma adımıdır.

Fonksiyonun yalnızca admin tarafından çağrılabilir olması, sistemin kötü niyetli bir kullanıcı tarafından manipüle edilmesini engellemeye yöneliktir. Eğer bu yetkilendirme mekanizması devre dışı bırakılmış olsaydı, herhangi bir adres sözleşmeler arası sahte bağlantılar kurabilir, seçim bütünlüğünü tehlikeye sokabilirdi.

Sistem mimarisi açısından düşünüldüğünde, initialize() fonksiyonu, ElectionContract'ın kendi başına çalışmadığını; diğer iki sözleşmenin içerdiği verilere bağlı olduğunu gösteren bir bağımlılık çözümleme noktası (dependency injection) işlevi görmektedir.

Bağlantılar başarıyla sağlanmadan vote(), getResults() veya getCandidateCID() gibi ana işlevlerin çalıştırılması mümkün değildir. Çünkü bu fonksiyonlar, bağlı sözleşmelerden veri çekmeden işlem gerçekleştiremez. Örneğin:

vote() fonksiyonu, VoterContract'taki seçmen kayıtlarını kontrol eder,

vote() ayrıca CandidateContract'taki aday kimliğini doğrular,

getResults() fonksiyonu da adaylara ait listeyi CandidateContract üzerinden çeker.

Dolayısıyla initialize(...) çağrısı, sadece bir bağlantı fonksiyonu değil; sistemin başlatılabilirliği açısından zorunlu bir yapısal önkoşul niteliğindedir.

4.6.3. Oy Verme İşlemleri ve Senaryo Tabanlı Hata Testleri

Seçim sisteminin adil, güvenli ve güvenilir bir biçimde işlemesini sağlamak amacıyla geliştirilen oy kullanma modülü, yalnızca yetkilendirilmiş kullanıcıların tek seferlik katılımını mümkün kılacak biçimde yapılandırılmıştır. Bu kapsamda, ElectionContract içerisinde tanımlı vote(string memory candidateId) fonksiyonu, çok katmanlı kontrol mekanizmaları içermekte; fonksiyon çağrıları hem seçmen kaydı hem de önceki oy kullanımı açısından değerlendirmeye tabi tutulmaktadır.

Yürütülen manuel test süreci, söz konusu kontrol yapılarını senaryo bazlı olarak sınamak amacıyla üç farklı kullanım durumunu içerecek biçimde planlanmıştır:

Senaryo 1: Yetkili Seçmen ile Oy Kullanımı

İlk senaryoda, daha önce index.js betiği ile sistemde kayıt altına alınmış bir adres, Remix IDE üzerinden seçilmiş ve vote("Aday1") komutu çağrılmıştır. Fonksiyon sorunsuz biçimde çalışmış, oy başarıyla işlenmiş ve sistemde tanımlı VoteCast olayı tetiklenmiştir.

Bu işlem sonrasında:

Belirtilen adaya ait oy sayısı bir artırılmış,

hasVoted[msg.sender] değeri true olarak güncellenmiş,

Seçim sonucu getResults() fonksiyonu ile doğrulanabilir hâle gelmiştir.

Bu durum, sistemin kayıtlı seçmenleri doğru biçimde tanıdığını ve işlem mantığını hatasız yürüttüğünü ortaya koymuştur.

Senaryo 2: Yetkisiz Adres ile Oy Kullanımı

İkinci testte, sistemde seçmen olarak tanımlanmamış bir Ethereum adresi kullanılarak vote() fonksiyonu çağrılmıştır. Remix arayüzü üzerinden işlem gönderildiğinde, akıllı sözleşme çağrıyı reddetmiş ve şu hata mesajı döndürülmüştür:

reverted with reason string "Oy kullanma yetkiniz yok!"

Bu hata mesajı, sözleşmede tanımlı onlyRegisteredVoter modifier'ının başarılı bir şekilde çalıştığını, sisteme dâhil olmayan bir adresin seçim sürecine müdahale edemeyeceğini göstermektedir. Bu durum, sistemin dış manipülasyonlara karşı dayanıklı olduğunu ve kayıtlı olmayan katılımcıların oy kullanmasının önlendiğini ortaya koymaktadır.

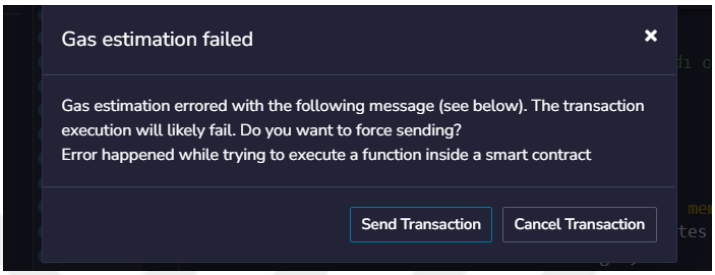
Senaryo 3: Aynı Adres ile Tekrar Oy Kullanma Girişimi

Üçüncü ve son senaryo, aynı adresin ikinci kez oy kullanmayı denemesi üzerine kurulmuştur. Daha önce oy kullanmış olan bir adres ile vote() fonksiyonu yeniden çağrıldığında, Remix IDE üzerinden şu uyarı ile karşılaşılmıştır:

[execution reverted] gas estimation failed

Bu hata, onlyOnce modifier'ının devreye girdiğini ve söz konusu adresin tekrar oy kullanmasının engellendiğini göstermektedir. Sözleşmede her seçmen adresi için tanımlı olan hasVoted kontrolü, bu adresin daha önce işlem gerçekleştirdiğini tespit etmiş ve çağrıyı iptal etmiştir.

Bu güvenlik kontrolünün başarısı, Remix IDE üzerinde karşılaşılan gaz tahmini hatasıyla doğrulanmıştır (bkz. Şekil 4.13).



Şekil 4.13: Aynı Adresin İkinci Kez Oy Kullanma Girişiminde Karşılaşılan Hata

4.6.4. getResults() Fonksiyonu ile Seçim Sonuçlarının Görselleştirilmesi

Seçim sisteminin en kritik aşamalarından biri olan sonuç hesaplama işlemi, doğrudan kullanıcıya sunulan çıktılar üzerinden sistemin doğruluğunu ve güvenilirliğini test etme imkânı sağlar. Bu bağlamda, seçim süreci tamamlandıktan sonra getResults() fonksiyonu çağrılarak, adayların aldığı toplam oy sayıları belirlenmiş ve söz konusu veriler detaylı biçimde analiz edilmiştir.

Fonksiyonun işleyiş mantığı, CandidateContract içerisinde kayıtlı bulunan aday listesini döngüsel olarak taramakta ve her aday kimliğine karşılık gelen oy sayısını votes[candidateId] mapping yapısı üzerinden sorgulamaktadır. Bu işlem sonucunda iki adet dizi (candidateIds[] ve voteCounts[]) oluşturulmakta, bu diziler eşleştirilerek her bir adayın aldığı oy sayısı kullanıcıya sunulmaktadır.

Yapılan test senaryoları doğrultusunda:

Kayıtlı bir seçmen tarafından oy verilen adayın oy sayısı bir artırılmış,

Sistem dışı kullanıcıların yaptığı başarısız işlem girişimlerinin oy sayıları üzerinde hiçbir etkisi olmadığı doğrulanmış,

Tekrar oy kullanma teşebbüslerinin de hasVoted kontrolü ile engellendiği belgelenmiştir.

Bu çıktılar, yalnızca getResults() fonksiyonunun teknik doğruluğunu değil, aynı zamanda ElectionContract içerisinde tanımlı kontrol mekanizmalarının zincir genelinde tutarlı şekilde çalıştığını da ortaya koymaktadır.

BEŞİNCİ BÖLÜM
SONUÇLAR VE DEĞERLENDİRME

Bu tez çalışması kapsamında, blok zincir teknolojisinin demokratik süreçlerdeki uygulanabilirliğini göstermek amacıyla Ethereum altyapısı kullanılarak merkeziyetsiz, güvenli ve şeffaf bir elektronik oylama sistemi prototipi tasarlanmış ve başarıyla hayata geçirilmiştir. Geliştirilen sistem, geleneksel oylama mekanizmalarının karşılaştığı güven eksikliği, merkezi müdahaleye açıklık ve denetlenemezlik gibi temel sorunlara çözüm üretmeyi amaçlamaktadır.

Önerilen çözüm, üç temel akıllı sözleşmeden oluşan modüler bir mimari üzerine inşa edilmiştir: CandidateContract, VoterContract ve ElectionContract. Bu sözleşmeler, seçim sisteminin farklı işlevsel ihtiyaçlarına karşılık verecek şekilde yapılandırılmıştır. VoterContract, yalnızca yetkilendirilmiş seçmenlerin oy kullanabilmesini sağlayarak kimlik doğrulama ve erişim kontrolü işlevini yerine getirmiştir. CandidateContract aracılığıyla adaylara ilişkin bilgiler, dağıtık dosya sistemi olan IPFS üzerinde saklanarak veri güvenliği ve bütünlüğü sağlanmıştır. ElectionContract ise oy kullanma, sonuç hesaplama ve sistem güvenliği gibi temel süreçleri yöneten çekirdek modül işlevi görmüştür.

Geliştirme sürecinde kullanılan Hardhat ortamı, yerel test senaryolarının gerçekleştirilmesine olanak tanımış; Node.js ile entegre edilen zincir dışı betikler, dış veri kaynaklarının sisteme entegrasyonunu sağlamıştır. Özellikle IPFS ile sağlanan entegrasyon, merkeziyetsiz veri saklama çözümleri açısından uygulamaya güncel bir boyut kazandırmıştır. Seçmen bilgilerine ilişkin verilerin zincir dışı olarak işlenip sisteme dâhil edilmesi, uygulamanın gerçek dünya senaryoları ile uyumluluğunu güçlendirmiştir.

Uygulamanın güvenlik ve doğruluk açısından değerlendirilmesi, Remix IDE üzerinden gerçekleştirilen manuel testlerle desteklenmiştir. Test senaryoları kapsamında, sistemin yalnızca önceden tanımlı adreslere oy kullanım izni verdiği, bir adresin yalnızca bir kez oy kullanmasına olanak tanıdığı ve tüm işlemlerin blok zincir üzerinde izlenebilir biçimde kaydedildiği gözlemlenmiştir. Ayrıca, getResults() fonksiyonu aracılığıyla sistemin toplam oy sayılarını doğru biçimde hesapladığı ve şeffaf bir şekilde sunduğu da teyit edilmiştir.

Genel değerlendirme itibarıyla, bu tez çalışması yalnızca teknik bir uygulama sunmakla kalmamış, aynı zamanda blok zincir temelli seçim sistemlerinin kavramsal, yapısal ve işlevsel olarak nasıl tasarlanabileceğini gösteren bütüncül bir model ortaya koymuştur. Modüler mimarisi sayesinde sistemin farklı uygulama senaryolarına uyarlanabilir olduğu ve daha kapsamlı dijital demokrasi çözümleri için bir temel teşkil edebileceği düşünülmektedir.

Sonuç olarak, elde edilen bulgular ve geliştirilen prototip; blok zincir teknolojisinin güven, denetim, şeffaflık ve merkeziyetsizlik ilkeleri doğrultusunda seçim süreçlerine entegre edilebileceğini ve bu alandaki sorunlara yenilikçi çözümler üretebileceğini göstermektedir.

ALTINCI BÖLÜM

GELECEK ÇALIŞMALAR İÇİN ÖNERİLER

Geliştirilen sistem, dağıtık yapılar temelinde kurgulanmış bir mimari yaklaşımın hem teknik hem de kuramsal düzeyde uygulanabilirliğini somut biçimde ortaya koymaktadır. İşlevsel bütünlüğü, bileşenler arası entegrasyon yapısı ve güvenlik odaklı tasarımı sayesinde, literatürde yer alan benzer çözümlerle karşılaştırıldığında özgün ve geliştirilebilir bir örnek teşkil etmektedir. Bununla birlikte, sistemin farklı kullanım senaryolarına ölçeklenebilirliğini artırmak ve uzun vadeli sürdürülebilirliğini sağlamak amacıyla, ileride yürütülebilecek çalışmalar için çeşitli geliştirme başlıkları da gündeme gelmektedir. Aşağıda bu doğrultuda öne çıkan başlıca araştırma alanları özetlenmiştir:

Kimlik Doğrulama Sistemlerinin Entegrasyonu: Mevcut sistemde seçmen kimlikleri zincir dışı kaynaklar üzerinden adresle ilişkilendirilmekte olup, daha güvenli bir yapı için ulusal dijital kimlik altyapıları (e-Devlet, eID, vb.) ile bütünleşmiş çözümler geliştirilebilir.

Mobil Uyumlu Arayüz Geliştirme: Uygulamanın son kullanıcıya ulaşan kısmı, hâlihazırda geliştirici düzeyinde çalıştırılmaktadır. Seçmenlerin kullanıcı dostu arayüzlerle oy kullanabilmesini sağlayacak mobil veya web tabanlı istemci uygulamaları üzerinde çalışılabilir.

Gizlilik Odaklı Oylama Protokolleri: Geliştirilen sistemde oylar adreslerle ilişkilendirilebilir durumdadır. Bu durum bazı durumlarda gizlilik ihlali yaratabilir. ZK-SNARKs (Zero-Knowledge Proof) gibi sıfır bilgi ispatı tabanlı teknolojilerin entegre edilmesiyle, seçmen gizliliği artırılabilir.

Gerçek Zamanlı İzleme ve Analitik Sistemleri: Oy verme sürecinde katılım oranları, sistem yükü veya güvenlik tehditleri gibi metriklerin anlık olarak izlenebileceği bir gözetim paneli (dashboard) geliştirilebilir.

Farklı Mutabakat Algoritmalarının Test Edilmesi: Ethereum ağı üzerinde çalışan mevcut sistem Proof of Stake (PoS) yapısını temel almakta; ancak farklı mutabakat mekanizmalarının (ör. Delegated Proof of Stake, Practical Byzantine Fault Tolerance) uygulamaya etkileri karşılaştırmalı olarak incelenebilir.

Bu öneriler doğrultusunda yürütülecek ileri düzey çalışmalar, blok zincir tabanlı seçim sistemlerinin yalnızca teknik yeterlilik açısından değil; ölçeklenebilirlik, erişilebilirlik ve kullanıcı deneyimi açısından da daha olgun bir hâle gelmesine katkı sağlayacaktır.

KAYNAKÇA

A. Abuhashim and C. C. Tan. (2020). Smart Contract Designs on Blockchain Applications. *2020 IEEE Symposium on Computers and Communications (ISCC)*, Rennes, France, 1-4. doi: 10.1109/ISCC50000.2020.9219622.

Antonopoulos, A. M., & Wood, G. (2018). *Mastering Ethereum: Building Smart Contracts and DApps*. O'Reilly Media.

Aydar, M., ve Çetin, S. C. (2020). Blok zincir Teknolojisinin Sağlık Bilgi Sistemlerinde Kullanımı. *Avrupa Bilim ve Teknoloji Dergisi*, (19), 533-538.

Bekmez, İ., & Gençoğlu, H. (2024). Blockchain Mutabakat Protokollerinin Karşılaştırılması. *Bilgisayar Bilimleri ve Teknolojileri Dergisi*, 5(1), 1-5. https://doi.org/10.54047/bibted.1424422

Bertoni, G., Daemen, J., Peeters, M., & Van Assche, G. (2011). The Keccak sponge function. Retrieved from https://keccak.team/files/Keccak-specifications.pdf (Erişim Tarihi: 20.04.2025)

Buterin, V. (2014). A Next-Generation Smart Contract and Decentralized Application Platform. *Ethereum White Paper*. https://ethereum.org/en/whitepaper/ (Erişim Tarihi: 20.04.2025)

Buterin, V. (2020). *Ethereum 2.0: The Roadmap*. https://ethereum.org (Erişim Tarihi: 20.04.2025)

C. T. Nguyen, D. T. Hoang, D. N. Nguyen, D. Niyato, H. T. Nguyen and E. Dutkiewicz. (2019). Proof-of-Stake Consensus Mechanisms for Future Blockchain Networks: Fundamentals, Applications and Opportunities. *IEEE Access*, 7, 85727-85745. doi: 10.1109/ACCESS.2019.2925010.

Ethereum Foundation. (n.d.). *Solidity Documentation*. https://docs.soliditylang.org (Erişim Tarihi: 20.04.2025)

J. Dongfang and L. Wang. (2022). Research on smart contract technology based on block chain. *2022 International Conference on Artificial Intelligence in Everything (AIE)*, Lefkosa, Cyprus, 664-668. doi: 10.1109/AIE57029.2022.00130.

J. Wu, J. Zhang, R. Guo and W. Tang. (2022). Points Transaction Mechanisms Based on Blockchain Technology. *2022 2nd International Conference on Computer Science and Blockchain (CCSB)*, Wuhan, China, 62-65. doi: 10.1109/CCSB58128.2022.00018.

J.-R. Giesen, S. Andreina, M. Rodler, G. O. Karame and L. Davi. (2022). Tutorial: Analyzing, Exploiting, and Patching Smart Contracts in Ethereum. *2022 IEEE Secure Development Conference (SecDev)*, Atlanta, GA, USA, 3-4. doi: 10.1109/SecDev53368.2022.00013.

L. Ezzeddini, J. Ktari, I. Zouaoui, A. Talha, N. Jarray and T. Frikha. (2022). Blockchain for the electronic voting system: case study: student representative vote in Tunisian institute. *2022 15th International Conference on Security of Information and Networks (SIN)*, Sousse, Tunisia, 01-07. doi: 10.1109/SIN56466.2022.9970543.

Pal, A., Tiwari, C. K., & Haldar, N. (2021). Blockchain for business management: Applications, challenges and potentials. *The Journal of High Technology Management Research*, 32(2), 100414.

Remix IDE Documentation. (n.d.). *Welcome to Remix's documentation!*. https://remix-ide.readthedocs.io/ (Erişim Tarihi: 20.04.2025)

S. Pongnumkul, C. Siripanpornchana and S. Thajchayapong. (2017). Performance Analysis of Private Blockchain Platforms in Varying Workloads. *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, Vancouver, BC, Canada, 1-6. doi: 10.1109/ICCCN.2017.8038517.

S. Srinivasan, R. Sundar, S. J. Herald Immanuel, R. Belvadi and M. Sathiyanarayanan. (2020). Hierarchical Design and Execution of Smart Contracts in Blockchain. *2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE)*, Bengaluru, India, 13-18. doi: 10.1109/ICSTCEE49637.2020.9276856.

Spasovski, J., & Eklund, P. (2017). Proof of Stake Blockchain. *Proceedings of the 9th International Conference on Management of Digital EcoSystems*. https://doi.org/10.1145/3167020.3167058

T. Kowalski, M. M. Chowdhury, S. Latif and K. Kambhampaty. (2022). Bitcoin: Cryptographic Algorithms, Security Vulnerabilities and Mitigations. *2022 IEEE International Conference on Electro Information Technology (eIT)*, Mankato, MN, USA, 544-549. doi: 10.1109/eIT53891.2022.9814066.

T. Zhao and T. Zhang. (2021). Revisiting Anonymity and Privacy of Bitcoin. *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, Shenyang, China, 1275-1280. doi: 10.1109/TrustCom53373.2021.00175.

Y. Ma, Y. Fu, L. Liu, Z. Du, JingMa and Y. Sun. (2022). A smart contract approach to access control based on distributed identities and roles. *2022 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS)*, Hengyang, China, 677-680. doi: 10.1109/ICITBS55627.2022.00147.

Y. Rosasooria, A. K. Mahamad, S. Saon, M. A. M. Isa, S. Yamaguchi and M. A. Ahmadon. (2020). E-Voting on Blockchain using Solidity Language. *2020 Third International Conference on Vocational Education and Electrical Engineering (ICVEE)*, Surabaya, Indonesia, 1-6. doi: 10.1109/ICVEE50212.2020.9243267.

Yewale, A. J. (2018). Study of Blockchain-as-a-Service Systems with a Case Study of Hyperledger Fabric Implementation on Kubernetes. *UNLV Theses, Dissertations, Professional Papers, and Capstones*, 3392. https://digitalscholarship.unlv.edu/thesesdissertations/3392

Z. Ouyang, J. Shao and Y. Zeng. (2021). PoW and PoS and Related Applications. *2021 International Conference on Electronic Information Engineering and Computer Science (EIECS)*, Changchun, China, 59-62. doi: 10.1109/EIECS53707.2021.9588080.

https://ethereum.github.io/yellowpaper/paper.pdf#page=5 (Erişim Tarihi: 20.04.2025)

https://ethereum.org/en/developers/docs/accounts/#types-of-accounts (Erişim Tarihi: 20.04.2025)