



MÜHENDİSLİK VE MİMARLIK FAKÜLTYESİ
BİLGİSAYAR MÜHENDİSLİĞİ
BLOKZİNCİR PROJESİ

Dersin Adı: Blokzincir

Dersin Hocası: Dr. Öğr. Üyesi Nursena BAYĞIN

Konu: Blokzincir Tabanlı Oy Kullanma Sistemi

Tarih: 28.05.2024

Proje Ekibi:

180303007-Samet İPEK

180303026-Furkan DOĞAN

2023-2024

İçindekiler Tablosu

GİRİŞ	3
PROJE AMACI VE AŞAMALARI	5
Amaç	5
Proje Aşamaları	5
AKILLI SÖZLEŞME (SMART CONTRACT) GELİŞTİRME	5
KULLANICI ARAYÜZÜ GELİŞTİRME	10
Arayüz	10
KULLANILAN UYGULAMALAR	12
Ganache	12
Metamask	12
Metamask Ve Ganache Entegrasyonu	13
NOT	14
KURULUM İŞLEMLERİ	15
Projenin çalışması için kurulması gereken programlar	15
Kurulması gereken paketler	15
Çok önemli not	16
KAYNAKÇA	17

GİRİŞ

Oylama, ister geleneksel oy pusulasıyla ister elektronik yöntemlerle yapılsın, demokrasinin temelini oluşturur. Elektronik Oylama, oy verme sürecinin maliyetini azaltmayı ve gizlilik, güvenlik ve uyumluluk gereksinimlerini karşılayarak seçim bütünlüğünü sağlamayı amaçlayan araştırmaların odak noktası olmuştur. Mevcut yöntemler, elektronik olsun veya olmasın, şeffaflık açısından yetersiz kalmıştır. Seçmenlerin, seçim sırasında verdikleri oyun gerçekten seçim sonucuna yansıyor yansımadığını anlamaları oldukça zor olabilir. Doğrudan Kayıt Elektroniği (DRE) kullanan elektronik oylama sistemleri, oyların başarılı bir şekilde verilmesi durumunda makbuz üretmezler. Hükümet tarafından yalnızca oy sayımının kamuoyuna açıklanması dışında, herhangi bir seçim kaydı kamuya sunulmaz, bu da seçmenlerin hükümetin oy sayımını yeniden yapması durumunda herhangi bir dış müdahale olup olmadığından emin olamamalarına yol açar. Hükümet dışı varlıklar da seçimlere fiziksel güç, sözlü tehdit veya hile yoluyla müdahale edebilir ve bu da oyların yanlış verilmesine veya sayılmasına neden olabilir.

Güçlü özgür ve adil seçim geleneklerine sahip ülkelerde bile, seçim sahtekarlığını izlemek ve en aza indirmek sürekli bir görevdir. Geleneksel oylama yönteminin yerini Blockchain teknolojisini kullanan elektronik yöntemle değiştirmek, seçim sırasında meydana gelebilecek potansiyel sahtekarlıkları önleyebilir.

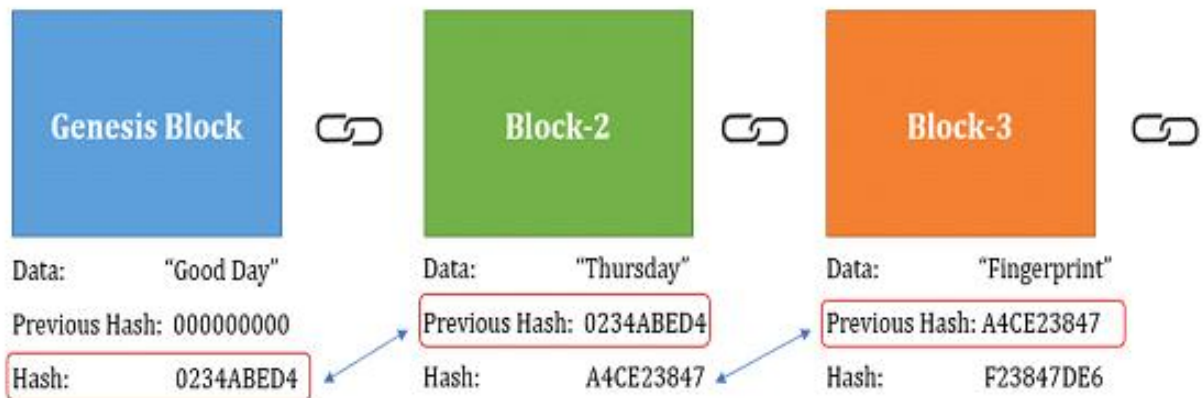
Bu proje kapsamında, Blockchain tabanlı bir elektronik oylama sistemi geliştirerek seçimlerin daha şeffaf, güvenli ve güvenilir hale getirilmesi amaçlanmaktadır. Blockchain teknolojisinin sağladığı değiştirilemezlik ve merkeziyetsizlik özellikleri sayesinde, oyların güvenli bir şekilde kaydedilmesi ve saklanması sağlanacaktır. Bu sayede, seçmenler oylarının gerçekten sayıldığından emin olacak ve seçim sonuçları üzerinde herhangi bir manipülasyonun önüne geçilecektir.

Blockchain teknolojisi, birbirine bağlı düğümlerden oluşan dağıtık bir ağıdır. Her bir düğüme, dağıtık defterin bir kopyası atanır ve bu defter, ağ tarafından işlenen tüm işlemlerin tam geçmişini içerir. İşlenen her işlem bir hash oluşturur. Oluşturulan hash, yalnızca mevcut işlemine değil, aynı zamanda önceki işlemin hash'ine de bağlıdır. Bu nedenle, verilerde yapılacak küçük bir değişiklik bile işlemin hash'ini etkiler. İşlem bir bloğa yazılır. Bu, kullanıcıların sistemi kullanırken özerk kalmalarını sağlar. Blockchain'in temel bir analizi, oylama sürecini daha güvenli ve güvenilir hale getirme potansiyeli sunduğunu göstermektedir.

Blockchain teknolojisinin sağladığı bu yapı, verilerin değiştirilemez ve güvenli bir şekilde saklanmasını sağlar. Her bir işlemin hash ile bağlantılı olması, zincirin herhangi bir noktasında yapılacak bir değişikliğin hemen tespit edilmesini mümkün kılar. Bu özellik, özellikle oylama süreçlerinde manipülasyonun önüne geçilmesi için büyük bir avantaj sağlar.

Ayrıca, kullanıcıların kendi verileri üzerinde tam kontrol sahibi olmalarını sağlayarak merkezi otoritelerin müdahalesini ortadan kaldırır. Bu da seçimlerin daha şeffaf ve adil olmasını sağlar.

Bu projede, Blockchain teknolojisinin bu özelliklerinden yararlanarak, daha güvenli ve güvenilir bir elektronik oylama sistemi geliştirilmesi hedeflenmektedir. Blockchain'in dağıtık yapısı sayesinde, oyların güvenli bir şekilde kaydedilmesi ve saklanması sağlanacak, böylece seçmenler oylarının gerçekten sayıldığından emin olabilecektir. Bu da seçim sonuçlarının manipülasyona karşı korunmasını sağlayacaktır.



PROJE AMACI VE AŞAMALARI

Amaç

Blokzincir teknolojisini kullanarak seçmen gizliliğini koruyan ve erişilebilirliği artıran bir oylama sistemi oluşturmayı amaçlamaktadır. Sistem, oylama sürecini şeffaf, güvenli ve maliyet etkin hale getirir. Ethereum blokzincirinde akıllı sözleşmeler kullanarak seçmen yönetimi ve denetlenebilir oylama kayıtları sağlanır.

Projenin backend kısmı Solidity, Node.js ve Ganache ile yapılmıştır.

Projenin fronted kısmı da React.js, Web3.js, HTML/CSS ve JavaScript ile yapılmıştır.

Proje Aşamaları

1. Akıllı Sözleşme (Smart Contract) Geliştirme: Oylama sürecini yönetecek akıllı sözleşmelerin Solidity dilinde yazılması.
2. Kullanıcı Arayüzü Geliştirme: Oylama sistemi için kullanıcı arayüzünün React.js kullanarak geliştirilmesi ve Web3.js ile Ethereum blokzinciri ile entegrasyon sağlanması.
3. Metamask ve Ganache Entegrasyonu: Metamask hesabının oluşturulması ve Ganache ile bağlantı kurulması. Test için faucet (musluk) araştırılarak test coinlerinin yüklenmesi.

AKILLI SÖZLEŞME (SMART CONTRACT) GELİŞTİRME

Akıllı sözleşmeler, blockchain üzerinde çalışan programlardır ve Solidity dili ile yazılır. Oylama sürecini yönetecek bir akıllı sözleşme oluşturmak için herhangi bir Solidity derleyicisi kurmalısınız. VS Code ile kodunuzu yazıp derleyebilir, Ganache ile yerel bir blockchain ağı oluşturabilir ve Metamask kullanarak sözleşmenizi dağıtabilirsiniz.

Bu akıllı sözleşme, seçim sahibini tanımlamak, adayları eklemek ve seçmenleri yetkilendirmek için gerekli fonksiyonları içerir. Seçmenler yetkilendirilmişse oy kullanabilir ve her seçmen yalnızca bir kez oy kullanabilir. Oylama tamamlandığında, toplam oy sayısı ve adayların aldığı oylar kaydedilir.

Sonuç olarak, bu akıllı sözleşme yapısı, oylama sürecinin güvenli ve şeffaf bir şekilde gerçekleştirilmesini sağlar. Seçmenlerin kimlikleri doğrulanır, oylar anonim olarak kaydedilir ve seçim sonuçları güvenli bir şekilde yönetilir. Bu sistem, dijital oylama uygulamaları için güvenli ve etkili bir çözüm sunar.

```
pragma solidity >=0.5.0 <0.9.0;
contract Election{
    address public manager;
    struct Candidate {
        uint id;
        string CfirstName;
        string ClastName;
        string CidNumber;
        uint voteCount;
    }
    mapping (address => bool) public voters;
    mapping (uint => Candidate) public candidates;
    uint public candidatesCount;
    event votedEvent (
        uint indexed_candidateId
    );
    constructor () public {
        manager = msg.sender;
    }
    function addCandidate(string memory _CfirstName,string memory _ClastName,string
memory _CidNumber) public onlyAdmin{
        candidatesCount++;
        candidates[candidatesCount] = Candidate(candidatesCount, _CfirstName, _ClastName,
_CidNumber, 0);
    }
    modifier onlyAdmin () {
        require(msg.sender == manager);
        _;
    }
    function vote (uint _candidateId) public {
        require(!voters[msg.sender]);
        require(_candidateId > 0 && _candidateId <= candidatesCount);
        voters[msg.sender] = true;
        candidates[_candidateId].voteCount ++;
        uint candidateId = _candidateId;
        emit votedEvent(_candidateId);
    }
    struct User {
        string firstName;
```

```

    string lastName;
    string idNumber;
    string email;
    string password;
    address add;
}
mapping (uint => User) public users;
uint public usersCount;
function addUser(string memory _firstName,string memory _lastName,string memory
_idNumber,string memory _email,string memory _password) public{
    usersCount++;
    users[usersCount] = User(_firstName, _lastName, _idNumber, _email, _password,
msg.sender);
}
}

```

Bu Solidity sözleşmesi, bir seçim sistemini yönetmek için tasarlanmıştır. Sözleşme, adayların eklenmesi, kullanıcıların kaydedilmesi ve oy kullanma işlemlerini içerir. Sözleşmenin ana bileşenlerini ve işlevlerini şu şekilde anlatabilirim:

Öncelikle, ``pragma solidity >=0.5.0 <0.9.0;`` ifadesiyle, sözleşmenin Solidity'nin 0.5.0 ile 0.9.0 arasındaki sürümleriyle uyumlu olmasını sağladım.

Sözleşme ``Election`` adını taşıyor ve seçim sürecini yönetmek için gerekli olan fonksiyonları içeriyor. ``manager`` adlı bir adres değişkeni belirledim ve bu değişken, sözleşmeyi dağıtan kişi yani yönetici olacak şekilde ayarlandı.

Adayları tanımlamak için ``Candidate`` adında bir yapı oluşturdum. Bu yapı, adayların kimlik numarası, adı, soyadı ve aldığı oy sayısını içeriyor. Adaylar ``candidates`` adlı bir mapping yapısında saklanıyor ve her bir aday kimlik numarası ile erişiliyor. Aday sayısını ``candidatesCount`` değişkeni ile takip ediyorum.

Oy verenleri ``voters`` adlı bir mapping yapısında saklıyorum. Bu mapping, bir adresin oy verip vermediğini kontrol etmek için kullanılıyor.

Yeni adaylar eklemek için ``addCandidate`` fonksiyonunu oluşturdum. Bu fonksiyon sadece yönetici tarafından çağrılabilir ve adayların adını, soyadını ve kimlik numarasını alarak yeni bir aday ekler.

``onlyAdmin`` adında bir modifier tanımladım. Bu modifier, sadece yöneticinin belirli fonksiyonları çağırabilmesini sağlıyor. ``require(msg.sender == manager);`` ifadesi ile çağıranın yönetici olup olmadığını kontrol ediyorum.

``vote`` fonksiyonu, kullanıcıların bir adaya oy vermesini sağlıyor. Bu fonksiyon, kullanıcının daha önce oy vermemiş olduğunu ve geçerli bir aday kimliği girip girmediğini kontrol eder. Kullanıcı oy verdikten sonra, oy verdiği adayın oy sayısını artırır ve ``votedEvent`` olayı tetiklenir.

Kullanıcıları yönetmek için `User` adında bir yapı tanımladım. Bu yapı, kullanıcının adı, soyadı, kimlik numarası, e-posta adresi, şifresi ve adresini içeriyor. Kullanıcılar `users` adlı bir mapping yapısında saklanıyor ve her bir kullanıcıya bir kimlik numarası atanıyor. Kullanıcı sayısını `usersCount` değişkeni ile takip ediyorum.

Yeni kullanıcılar eklemek için `addUser` fonksiyonunu oluşturdum. Bu fonksiyon, kullanıcının adı, soyadı, kimlik numarası, e-posta adresi ve şifresini alarak yeni bir kullanıcı ekler.

Bu sözleşme ile, yönetici yeni adaylar ekleyebilir, kullanıcılar oy kullanabilir ve kullanıcılar sisteme kaydedilebilir. Oy verme işlemi sırasında, her bir kullanıcının sadece bir kez oy vermesi sağlanmıştır ve adayların aldığı oy sayıları güncellenir.

```
pragma solidity >=0.5.0 <0.9.0;
contract Migrations {
    address public owner;
    uint public last_completed_migration;
    modifier restricted() {
        if (msg.sender == owner) _;
    }
    constructor() public {
        owner = msg.sender;
    }
    function setCompleted(uint completed) public restricted {
        last_completed_migration = completed;
    }
    function upgrade(address new_address) public restricted {
        Migrations upgraded = Migrations(new_address);
        upgraded.setCompleted(last_completed_migration);
    }
}
```

Bu kod dosyasını Ethereum blockchain üzerinde bir akıllı sözleşme olarak geliştirdim. Sözleşmemin adı "Migrations" ve temel amacı, Truffle projesi içerisindeki dağıtım sürecini yönetmek ve izlemek.

Öncelikle, sözleşmenin sahibi olacak bir owner adresi tanımladım ve bu adres, sözleşmeyi dağıtan kişinin adresi olacak. Ayrıca, son tamamlanan göçün kimlik numarasını tutmak için last_completed_migration adında bir değişken tanımladım.

Kodun güvenliğini sağlamak için bir restricted modifiyeri ekledim. Bu modifiyer, yalnızca sözleşmenin sahibinin belirli işlevleri çağırabilmesini sağlıyor. Bu sayede, yetkisiz kişilerin kritik işlemleri gerçekleştirmesini engelliyoruz.

Sözleşme ilk oluşturulduğunda çalışacak olan yapıcı fonksiyonu (constructor) yazdım. Bu fonksiyon, sözleşmeyi dağıtan kişinin adresini owner değişkenine atıyor.

Sözleşmeye eklediğim setCompleted fonksiyonu, son tamamlanan göçün kimlik numarasını güncellemek için kullanılıyor ve yalnızca owner tarafından çağırılabilir. Bu sayede, hangi adımların tamamlandığını takip edebiliyoruz.

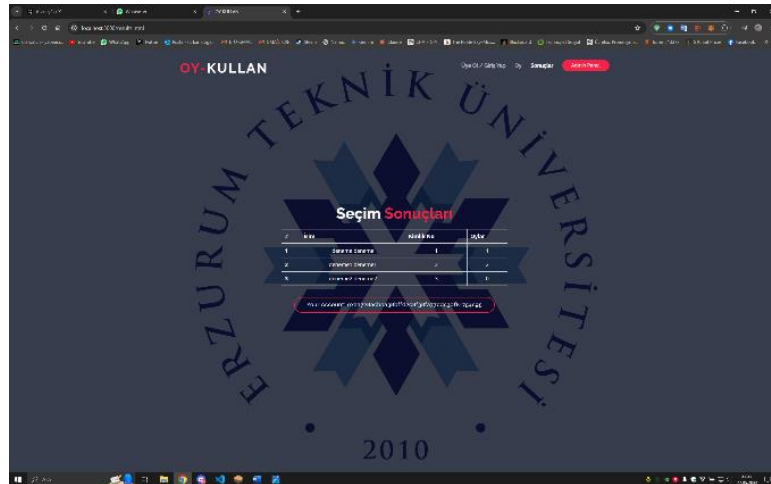
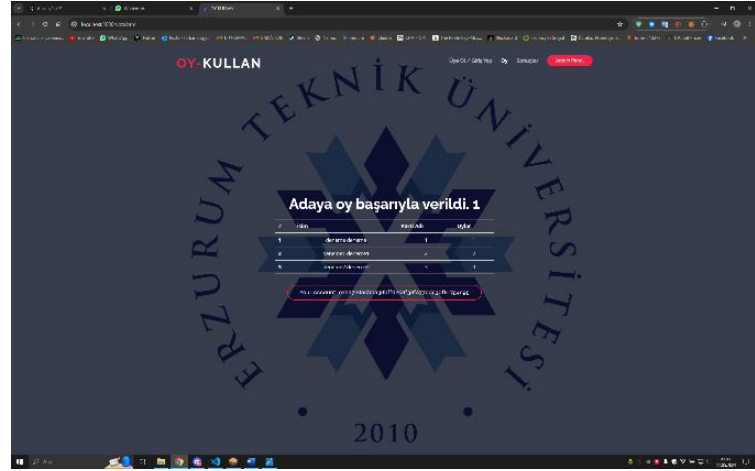
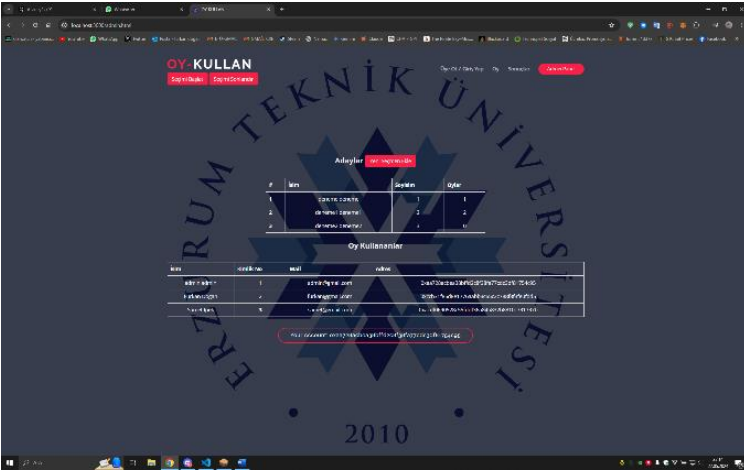
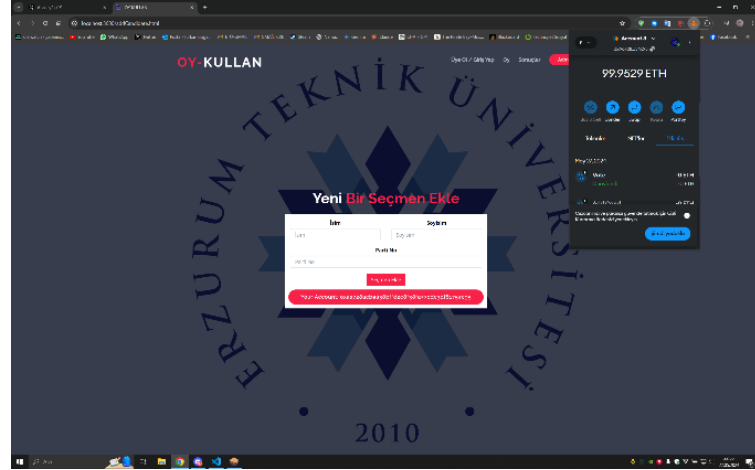
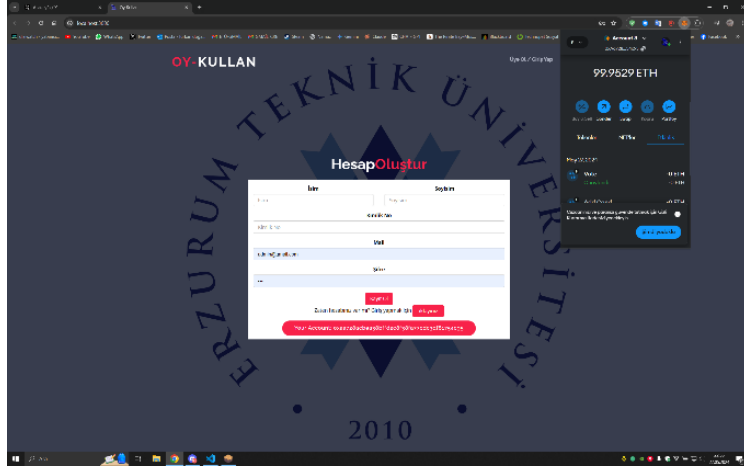
Son olarak, upgrade fonksiyonu ile sözleşmeyi yeni bir adrese yükseltebiliyoruz. Bu fonksiyon, yeni sözleşme adresini alır ve setCompleted fonksiyonunu çağırarak son tamamlanan göçü yeni adrese aktarır. Bu, projemizdeki değişiklikleri veya güncellemeleri kolayca yönetmemizi sağlıyor.

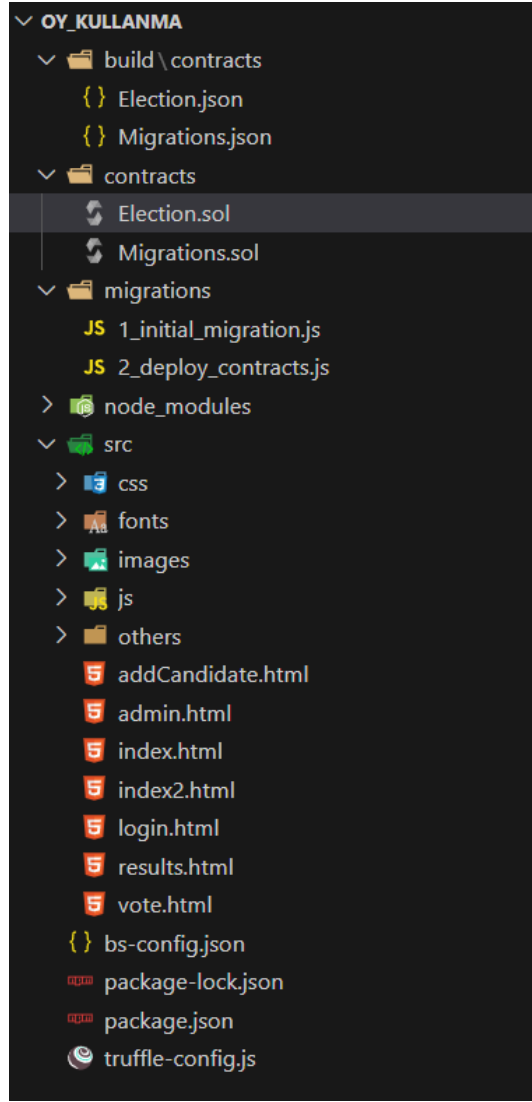
Bu sözleşme, projemizdeki dağıtım süreçlerini etkili bir şekilde izlemek ve yönetmek için geliştirilmiştir.

KULLANICI ARAYÜZÜ GELİŞTİRME

Kullanıcı arayüzü kısmı oldukça karmaşık, fazlaca kod dosyası, satırı içerdiği ve çoğunun da açıklanması oldukça gereksiz olduğu için arayüz görselleri üzerinden aşağıda kısaca açıklanmıştır.

Arayüz





truffle-config.js dosyası:

Bu dosya, Truffle yapılandırma dosyasıdır ve Ganache ile bağlantıyı sağlar. İçerisinde networks kısmında Ganache'ın RPC URL'si ve port bilgileri bulunur.

contracts/Migrations.sol ve contracts/Election.sol dosyaları:

Bu dosyalar, projenin akıllı sözleşmelerini içerir. Migrations.sol sözleşme dağıtımını yönetirken, Election.sol seçim işlemlerini ve oy kullanma fonksiyonlarını içerir.

build/contracts/Election.json dosyası:

Bu dosya, derlenmiş akıllı sözleşme bilgilerini ve ABI'yi içerir. Metamask ve diğer frontend uygulamaları ile etkileşimde kullanılır.

src/vote.html dosyası:

Bu HTML dosyası, kullanıcıların oy kullanabilmeleri için bir arayüz sağlar. Ayrıca, oy sonuçlarını görüntüleme ve admin paneline erişim için bağlantılar içerir.

KULLANILAN UYGULAMALAR

Ganache

Ganache, Truffle Suite'in bir parçası olarak Ethereum geliştiricileri için tasarlanmış bir araçtır. Geliştiricilerin yerel bir blockchain ağı üzerinde hızlı ve güvenli bir şekilde akıllı sözleşmelerini test etmelerini sağlar. Ganache, bir Ethereum blockchain'in tam kopyasını yerel makinenizde çalıştırarak, geliştiricilere gerçek dünya senaryolarını taklit etme imkanı sunar.

Ganache, hızlı bir şekilde blok üretir, bu da işlemlerin anında onaylanmasını sağlar ve bu nedenle geliştirme sürecini hızlandırır. Ayrıca, kullanıcıların özel blockchain parametrelerini ayarlamasına olanak tanır, örneğin, gas ücretleri ve blok süreleri gibi.

Ganache, iki farklı modda çalışabilir: Ganache UI (grafik arayüz) ve Ganache CLI (komut satırı arayüzü). Ganache UI, kullanıcıların blockchain'i görsel olarak yönetmelerini sağlar, işlemleri, hesap bakiyelerini ve blokları görselleştirir. Ganache CLI ise daha ileri düzeyde kullanıcılar için komut satırı üzerinden kontrol imkanı sunar. Projemizde Ganache UI kullandık.

Metamask

MetaMask, Ethereum ve diğer blockchain ağlarıyla etkileşimi kolaylaştıran, güvenli ve kullanıcı dostu bir cüzdan ve tarayıcı uzantısıdır. Geliştiricilere ve kullanıcılara, merkeziyetsiz uygulamalarla etkileşime geçme, dijital varlıklarını yönetme ve blockchain tabanlı işlemleri güvenli bir şekilde gerçekleştirme imkanı sunar.

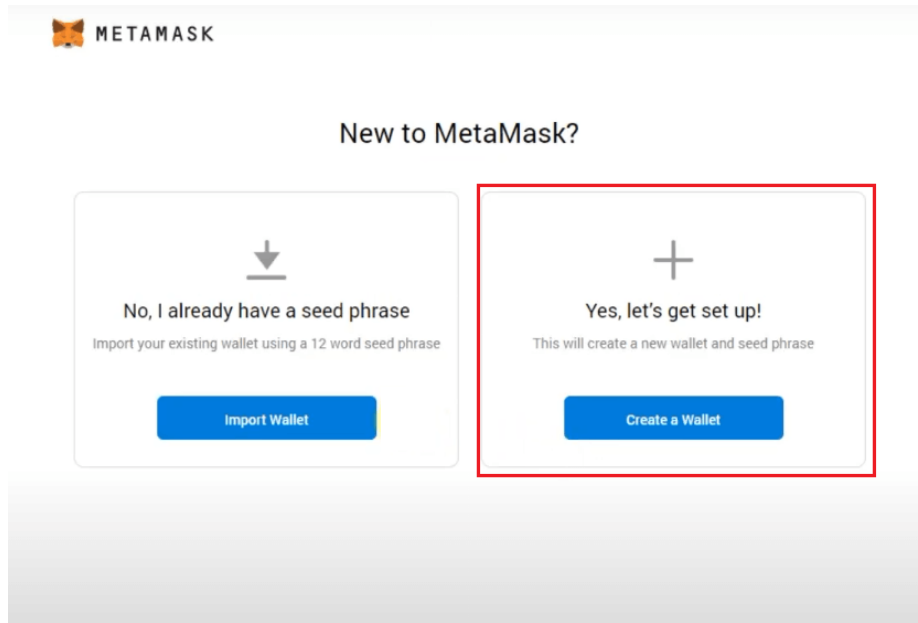
MetaMask'in sağladığı diğer önemli özellikler şunlardır:

- Güvenli Oturum Açma: Kullanıcıların Ethereum tabanlı dApp'lere güvenli bir şekilde giriş yapmalarını sağlar.
- Anahtar Yönetimi: Kullanıcıların özel anahtarlarını güvenli bir şekilde saklamalarını ve yedeklemelerini sağlar.
- İşlem Onaylama: Kullanıcıların her işlemi manuel olarak onaylamalarını sağlayarak, yetkisiz işlemleri engeller.
- DApp Tarayıcısı: Kullanıcıların Ethereum blockchain'i üzerinde çalışan dApp'lere doğrudan erişimlerini ve bu uygulamalarla etkileşimde bulunmalarını sağlar.

Metamask Ve Ganache Entegrasyonu

Metamask hesabı oluşturma:

Metamask hesabı oluşturmak için İlk olarak, Metamask'ın web sitesine gidip tarayıcına uygun olan eklentiye indiriyoruz. Kurulumdan sonra, tarayıcının sağ üst köşesinde beliren Metamask ikonuna tıklıyorsun ve açılan pencerede "**Get Started**" butonuna basıyoruz. Yeni bir cüzdan oluşturmak için "**Create a Wallet**" seçeneğine tıklıyoruz ve ardından güçlü bir şifre belirliyoruz. Şifreyi iki kez girdikten sonra, Metamask sana 12 kelimelik gizli bir kurtarma cümlesi gösterecek. Bu cümleyi güvenli bir yerde saklıyoruz. çünkü bu cümle cüzdanı kurtarmak için gerekli. Son olarak, bu kelimeleri doğrulayıp hesabını oluşturuyoruz.



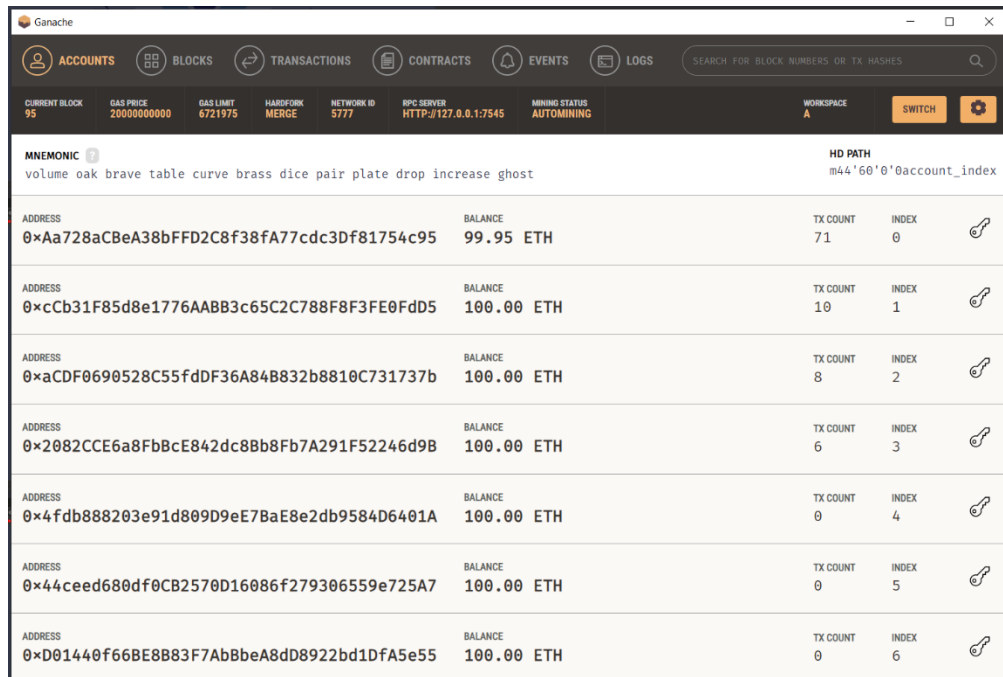
Metamask hesabını Ganache ile bağlantılandırma ve test coinlerini yükleme:

Öncelikle, Ganache'ı açıp "**New Workspace**" seçeneğine tıklayarak yeni bir proje oluşturuyoruz. Projemize bir ad veriyoruz ve Ganache uygulamasına proje konumunda bulunan "**truffle-config.js**" dosyasını import ediyoruz. Bu aşamadan sonra açılan ekranda "**RPC SERVER**" sekmesine gidiyoruz ve buradaki "**Host Name**" (**http://127.0.0.1**) ve "**Port Number**" (**7545**) bilgilerini not ediyoruz.

Ardından, Metamask'ı açıyoruz ve sağ üst köşedeki profil simgesine tıklayarak "Ayarlar" bölümüne gidiyoruz. Buradan "**Ağlar**" sekmesine tıklayıp "**Ağ Ekle**" butonuna basıyoruz. Ağ ismi olarak "Ganache" ya da herhangi bir isim yazıyoruz ve yeni RPC URL kısmına Ganache'dan kopyaladığımız URL'yi yapıştırıyoruz (**http://127.0.0.1:7545**). Zincir kimliği

(Chain ID) olarak **1337** yazıyoruz ve para birim simgesi olarak "**ETH**" seçiyoruz. Bu adımları tamamladıktan sonra ağı kaydediyoruz.

Sonrasında, Ganache'da oluşturulan hesaplardan birinin özel anahtarını kopyalıyoruz. Metamask'a geri dönüp profil simgesine tıklayarak "Hesap İçer Aktar" seçeneğine gidiyoruz ve kopyaladığımız özel anahtarı yapıştırıyoruz. Bu işlemle, Ganache'da oluşturulan hesabı Metamask'a eklemiş oluyoruz ve bu hesapta bulunan test coinlerini görebiliyoruz. Bu şekilde, yerel blockchain üzerinde test yaparken Metamask üzerinden işlemler gerçekleştirebiliyoruz ve geliştirdiğimiz uygulamaları test edebiliyoruz. Bu yöntemle Metamask ve Ganache arasında güvenli ve etkili bir bağlantı kurarak, blockchain uygulamalarını test edebiliriz.



ADDRESS	BALANCE	TX COUNT	INDEX
0xAa728aCBeA38bFFD2C8f38fA77cdc3Df81754c95	99.95 ETH	71	0
0xCb31F85d8e1776AABB3c65C2C788F8F3FE0FdD5	100.00 ETH	10	1
0xCDF0690528C55fdDF36A84B832b8810C731737b	100.00 ETH	8	2
0x2082CCE6a8FbBcE842dc8Bb8Fb7A291F52246d9B	100.00 ETH	6	3
0x4fdb888203e91d809D9eE7BaE8e2db9584D6401A	100.00 ETH	0	4
0x44ceed680df0CB2570D16086f279306559e725A7	100.00 ETH	0	5
0xD01440f66BE8B83F7AbBbeA8dD8922bd1DfA5e55	100.00 ETH	0	6

NOT: Özel anahtar kopyalama işlemi için yukarıda görülen görselde sağ kısımda bulunan anahtar sekmelerinden birisine tıklayarak açılan ekrandan **PRIVATE KEY** kodunu kopyalayarak yapılır. MetaMask içerisindeki her hesap için ayrı bir key alınmalı.

ACCOUNT INFORMATION

ACCOUNT ADDRESS
0xAa728aCBeA38bFFD2C8f38fA77cdc3Df81754c95

PRIVATE KEY
0xa7eaabf79ca4bc5199668bc44d8b36100a4a43bf1e3848bf34817ec0a40fd600
Do not use this private key on a public blockchain; use it for development purposes only!

DONE

KURULUM İŞLEMLERİ

Projenin çalışması için kurulması gereken programlar

- Visual Studio Code (<https://code.visualstudio.com/download>)
- Node.js / npm (<https://nodejs.org/en/download>)
- Truffle (<https://trufflesuite.com/docs/truffle/how-to/install/>)
- Ganache (<https://trufflesuite.com/ganache/>)
- MetaMask Eklentisi
(<https://chromewebstore.google.com/detail/metamask/nkbihfbeogaeaoehlefnkodbepgpgknn>)

Kurulması gereken paketler

- Web3.js: **npm i web3**
- Solidity : **npm i solc**

Adım 1:

MetaMask ve Ganache bağlantısını sağlayın. Proje kullanım süreci boyunca Ganache programını kapatmayın. (İlgili konuya erişmek için [tıklayınız.](#))

Adım 2:

Truffle-config.js dosyasını ganache ile kontrol edin. Ganache-cli (cli kullananlar **8545** portunu almalı) kullanmadığımız için port numarası **7545** olarak ayarlanmalıdır.

Adım 3:

İşlemleri proje terminalinde yaptığınızdan emin olun. Bunun için **cd Oy_Kullanma** komutunu terminalde çalıştırınız. Zaten bu konumda iseniz hata çıkabilir bu durumu umursamayın.

Adım 4:

npm install (ya da **npm i**) komutunu vs code terminalinde çalıştırarak projenin çalışması için gerekli olan node.js paketleri yükleyin.

Adım 5:

Projeyi ilk defa çalıştırıyorsanız **truffle compile** komutunu terminalde çalıştırın. Daha önce projeyi çalıştırıp kullandıysanız **truffle migrate --reset** komutunu kullanın. Böylece sözleşmenin dağıtıldığından emin olun ve Ganache üzerinden kontrol edin.

Adım 6:

npm start komutunu terminale yazarak projeyi çalıştırın.

Artık proje arayüzüne erişim sağlamış olacaksınız. Burada bağladığınız ilk MetaMask hesabı sizin **admin** hesabınız olacaktır. Proje localhost:3000 üzerinde başarılı bir şekilde çalıştıktan sonra Yönetici Paneline erişebilir ve aday ekleyebilirsiniz. Adayları ekledikten sonra “Seçimi Başlat” düğmesine tıklayın.

Çok önemli not

Farklı kullanıcılar/seçmenler eklemek istiyorsanız, farklı hesapların özel anahtarlarını Ganache'den MetaMask'a aktarın ve bu hesapları mevcut siteye bağlayın ve oy kullanmak için farklı bir kullanıcı/seçmen olarak kaydolun.

KAYNAKÇA

1. Hjálmarsson, F. Þ., Hreiðarsson, G. K., Hamdaqa, M., & Hjalmtýsson, G. (2018, July). Blockchain-based e-voting system. In *2018 IEEE 11th international conference on cloud computing (CLOUD)* (pp. 983-986). IEEE.
2. Al-Maaitah, S., Qatawneh, M., & Quzmar, A. (2021, July). E-voting system based on blockchain technology: A survey. In *2021 International Conference on Information Technology (ICIT)* (pp. 200-205). IEEE.
3. <https://medium.com/coinmonks/e-voting-system-based-on-blockchain-75570ef92c4>
4. Fusco, F., Lunesu, M. I., Pani, F. E., & Pinna, A. (2018, September). Crypto-voting, a Blockchain based e-Voting System. In *KMIS* (pp. 221-225).
5. Pathak, M., Suradkar, A., Kadam, A., Ghodeswar, A., & Parde, P. (2021). Blockchain based e-voting system. *International Journal of Scientific Research in Science and Technology*, 8, 134-40.
6. Tanwar, S., Gupta, N., Kumar, P., & Hu, Y. C. (2024). Implementation of blockchain-based e-voting system. *Multimedia Tools and Applications*, 83(1), 1449-1480.