

SOLiDİTY TUTORİAL

Ümmü Derya Çelik & Kaan Efe Öğüt

180509041

180509021

Fırat Üniversitesi, Adli Bilişim Mühendisliği Bölümü, Elazığ, Türkiye

Özet

Günümüz teknoloji çağında aklımıza gelen neredeyse her işlemi online olarak yapabilme imkanı bulmaktayız. Bu teknolojilerden biri olan Blokzincir yapısı topluma güvenilir internet kavramını göstermiş olup kazanç getirebileceğini kanıtlamış ve kanıtlamaya devam etmektedir. Bu proje de blokzincir ağının ikinci versiyonu niteliği taşıyan Ethereum Platformunda yaygın olarak kullanılan Solidity akıllı kontrat yazma dilinin yazım şeklini, kurallarını ve işimizi nasıl Ethereum Platformuna taşıyabileceğimizden bahsedeceğiz.

I. GİRİŞ

Solidity diline değinmeden önce üzerinde çalışacağımız blokzincir ağını, yapısını ve işlevini kavramak gerekir. Blokzincir hayatımıza ilk olarak 2009 senesinde kripto para birimi olan Bitcoin ile girmiştir. İlerleyen zamanlarda ise blokzincirin para alışverişinden daha fazlası olduğu fark edilip bu ağda aynı zamanda akıllı kontratlara ve kontratta tanımlı kurallara göre uygulamalar yapıp çeşitli iş sektörlerine hitap edebileceği görülmüştür.

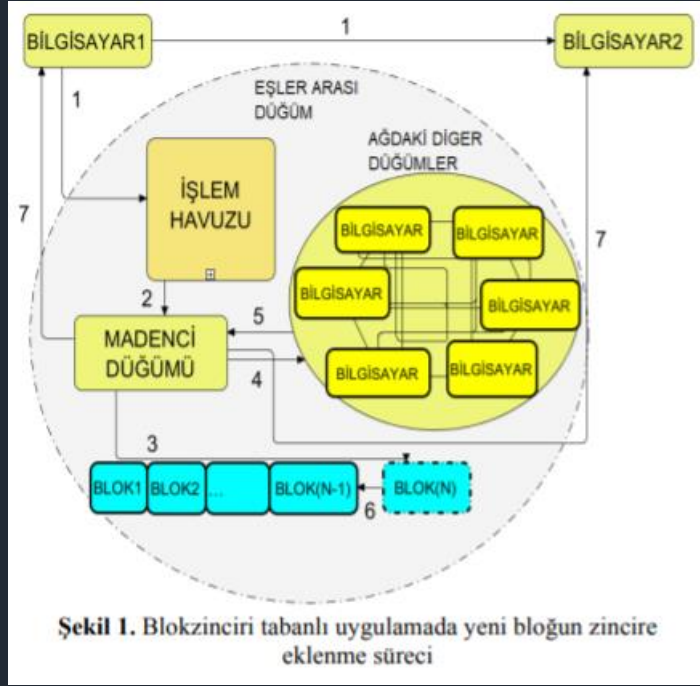
Bu makalenin, ikinci bölümünde blokzincir ve Ethereum ağının mimarisi ve özellikleri, üçüncü bölümünde blokzincir yapısının iş ağında kullanımı, dördüncü bölümünde siber güvenlik alanında blokzincir kullanımı, beşinci bölümünde akıllı kontratlar ve Solidity ele alınacaktır.

II. BLOKZİNCİR VE ETHEREUM

Blokzincir yapısı ilk olarak 2009'da Satoshi Nakamoto' nun ortaya attığı kripto para birimi olan Bitcoin fikri ile ortaya çıkmıştır. Bu yıllarda blokzincir yapısı geri planda kalırken ilerleyen zamanlarda blokzincirin kripto para alışverişi ortamından daha fazlası olduğu anlaşıp günümüz teknolojisinin büyüyen devi haline gelmeyi başarmıştır. Bu farkındalıkla beraber blokzincir ağının ikinci versiyonu sayılan Ethereum Platformu ortaya çıkmıştır.

Blokzincir, internet üzerinde herhangi bir merkeze bağlı kalmadan, verileri kripto şifreleme (SHA256 şifreleme standardı) kullanarak güvenliği sağlayan, işlem kayıtlarını şeffaflaştırarak aracısız ve güvenli bilgi transferi imkanı sunan, verileri dağıtık bir şekilde depolayan bilgi blokları bütünüdür. Ayrıca veri tabanı sondan eklemelidir. Bu özelliği ile işlenen veride değiştirme, silme gibi geriye dönük işlem yapılmasını önler ve böylece güvenli internet, güvenli veri kavramını yakalamış olur.

Dağıtık sistem, merkezi kurumsal mega bilgisayarların ve bulut sunucularının, dünyanın her yerinden gönüllüler tarafından çalıştırılan birçok küçük bilgisayardan oluşan büyük, merkezi olmayan bir ağıdır. Ağa dahil olan bilgisayarlar hala veri akışı sağladığı sürece veritabanı asla gerçekten kapatılamaz. Merkezi sistem de ise verilerin tutulduğu bir veritabanı bulunan bir bilgisayarda tutulur ve o bilgisayar hasar gördüğü anda tüm veriler kaybedilir.



Şekil 1. Blokzinciri tabanlı uygulamada yeni bloğun zincire eklenme süreci

Şekil1 'de, Bilgisayar1 'den Bilgisayar2 'ye gönderilen bir veri bloğunun blokzincire eklenme sürecini göstermektedir. Blokzincir yapısının merkezi veri tabanlarından farkını açıkça görmekteyiz. Blokzincir ile veri bütünlüğü (data integrity), kullanılabilirlik servisleri (availability), hata toleransı (fault tolerance) ve bütün kayıtlar ortada olsada kime ait olduklarının anonim olması

nedeniyle mahremiyet (privacy) gibi servisler verilebilmektedir.

Blokzincir yapısında her blok, bir önceki bloğun hash kodunu tutar. Sistemdeki bir işlemi değiştirmek, zincirdeki tüm blokları da hesaplamayı gerektirecektir. Bu da yoğun miktarda işlem gücü gerektirir. Zincirdeki tüm bloklara işlem yaptırabilmesi ve bunun için de PoW hesaplamalarını gerçekleştirebilmesi gerekir. Bunları yapabilmek için ağdaki bütün düğümlerin madencilik işlemci gücünün %51 'ine sahip olması gerekmektedir bu atağa “%51 saldırısı” denmektedir. Teorik olarak mümkün görünse de pratikte bu tür bir saldırı olasılığı olmadığı söylenmektedir.

Ethereum, blokzincir tabanlı yapısının yanı sıra akıllı kontrat olarak adlandırılan bilgisayar protokolünü de desteklemektedir. Sahip olduğu teknolojileri kullanarak merkezsiz dijital uygulamalar(dApp) üretmeyi ve çalıştırmayı mümkün kılar.

Geniş kullanım alanına sahip bu uygulamalar sayesinde kullanıcılar üçüncü kişi ,aracı, olmadan işlem yapabilir ve azami düzeyde güven ortamı sağlanır. Akıllı kontratlar yapısı gereğiyle dolandırıcılık, veri kaybı gibi siber zorbalık risklerini minimuma indirirken işlemleri şeffaf ve güvenli bir platformda gerçekleştirme imkanı sunar.

Ethereum Platformunun vizyonu; Herhangi bir yöneticiye gerek kalmadan, durdurulamaz, sansüre dirençli, kendi kendini idame ettiren merkezi olmayan bir dünya bilgisayarı yaratmaktır.

III.İŞ AĞI VE BLOKZİNCİR

Blokzincir, aşamalı ve yavaş bir şekilde iş hayatımıza da giriş yapmaya başlamıştır. Blokzincirin işimize getirebileceği artılar şu şekildedir:

- İş süreçlerini sadeleştirmek
- İş kayıtlarını şeffaflaştırmak
- Yönetimi katılımcı hale getirmek
- İşleyişi daha güvenilir hale getirmek (mahremiyet ile şeffaflık dengesi, kripto güvenlik)

Blokzincirin artıları olabildiği gibi eksileri de bulunmaktadır. Yapısı gereği yoğun güç ve maliyet gerektirdiği için işimizi ağa taşımadan önce biraz sorgulamak gerekir. Bunun için şu üç soruyu kendimize sormak gerekir;

- 1- Veriler varlıklar tutarlı olmalı mı?
- 2- Veriler yazıldıktan sonra değişmeden kalacak mı?
- 3- Katkıda bulunan birçok varlık var mı?

Eğer ilk soruya yönelik cevabınız “hayır” ise kalan iki soruyu cevaplamanıza gerek yok, işletmenizin blokzincire ihtiyacı yok. Bunun yerine sorunuz için çeşitli çözümler bulunabilir. Örneğin, veriler varlıklar arasında tutarlı olması gerekmiyorsa, bir işletme elektronik tabloları veya belgeleri kullanabilir.

Yöneticiler ve operatörlerin bir noktada yazılı verileri değiştirmesi gerekiyorsa, blokzincir uygun bir seçim olmaz çünkü veriler zincire girildiğinde herhangi bir değişikliğe izin verilmez. Bu durumda işletme, dağıtık defter teknolojisi yerine klasik bir veri tabanını tercih edebilir.

Üçüncü soruya gelince, bir kuruluş katkıda bulunan tek varlıksa veya birçok katkıda bulunan kuruluş arasında güven sorunu yoksa, verilerin dağıtılmasına da gerek yoktur. Dolayısıyla blokzinciri bu senaryo için de gereksiz görünmektedir.

IV.SİBER GÜVENLİK ALANINDA BLOKZİNCİR

Blokzincir yapısının siber güvenlik açısından önemini önceki bölümlerde de yer vermiştik. Blokzincir ile güvenli bir dijital kayıt defteri ve dağıtık otoriteye dayalı bir kimlik doğrulama sistemi güvenlik tehditlerine karşı alınabilecekler listesinin en üstünde yer alır. Ayrıca kişiye özel üretilmiş anahtarlar yani id'ler sayesinde kimlik hırsızlığı, dolandırıcılık, veri hırsızlığı, birden çok güvenlik katmanı (defense in depth) sayesinde DDOS ataklarını önleme gibi siber saldırılardan korunma imkanı verir.

Bu alanda kullanımı henüz çok yeni olsa da geleceği umut vadediyor. Bu konu hakkında araştırmalar yapan Projesium 'un kurucu ortağı Gökçe Philips 'in söylediklerine kulak verelim:

"Blokzinciri, hem veri tabanı kullanımında hem de kripto para aktarımında kullanılabilir. Sistem, zamandan tasarruf sağlayacak, bürokratik işlemler ve iş yükü azalacak. Kamuda verinin saklandığı birçok kurum var. Bunlar, kimlik, vergi, sigorta ve devlet hastanelerindeki veriler olabilir. A devlet hastanesinde yaptırdığımız MR sonuçları, B devlet hastanesine şu anda aktarılmıyor. Blokzincirin dağıtık veri yapısında bu bilgilere her yerden ulaşılabilir olacak. İzinli kurumlar ulaşabilecek ve izin verilmeyen kişiler de göremeyecek. Bunun yanında kamu kendi arasındaki veri geçişlerinde, bir kurum başka bir kuruma veri aktarıyorsa, yine blokzincirinin çok büyük getirisi olur. Veri tabanının kullanıldığı, tutulduğu ve aktarıldığı her alanda kamuda da değişiklikler göreceğimize inanıyorum. Dolayısıyla güvenlik anlamında blokzincirinin bize getireceği birçok yenilik var. Blokzinciri, zamandan tasarruf sağlayacak, bürokratik işlemler ve iş yükü azalacak. Kendinizle ilgili bir bilgi değişikliği olduğunda tek tek kurumları dolaşmanız gerekmeyecek. -

Bilgi bir yerde güncellendiği zaman diğer bütün kurumlarda da bu otomatik güncellenecek ve devlet bu veri doğru mu değil mi arayışı içinde olmayacak."

Cümleleriyle blokzincir teknolojisinin kolaylıklarına ve güvenilirliğine değinmiş oldu.

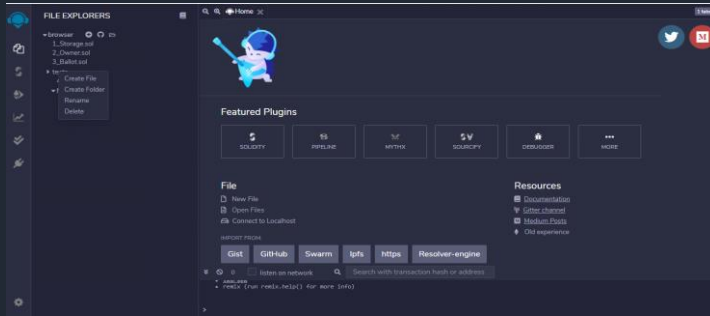
V.AKILLI KONTRATLAR VE SOLİDİTY

Akıllı kontratlar, alıcı ve satıcı arasındaki anlaşmanın doğrudan kod satırlarına yazılmasıyla Ethereum platformunda kendi kendini yürüten sözleşmelerdir. Akıllı kontratlar, otonom program, şeffaf, geri döndürülemez, para veya veri alın-gönderin ya da depolayan, internete ve diğer sözleşmelere bağlı sözleşmelerdir.

Solidity ise Ethereum platformunda kullanılan akıllı kontratları yazma dilidir. En basit ve en yaygın kullanılan dillerden biridir. GO dilinden esinlenilerek ortaya çıkmıştır. Solidity dilini bilen birinin GO dilinin yaklaşık %80'ini bildiğini söyleyebiliriz. Hazır fonksiyonlardan yararlanır. Yazımı her ne kadar kolay olsa da çalıştırması ve uygulamaya aktarması da bir o kadar zordur. İçerisinde bizim diğer yazılım dillerinden de aşına olduğumuz Class(Sınıf) yapısı gibi diyebileceğimiz Contract(Sözleşme)'lardan oluşur. Sözleşmelerde kullandığımız her fonksiyon için bu işlem karşılığında veritabanına 'GAS' yani depozito ücreti verilir. Bu yüzden çok fazla iş yaptıran fonksiyonlardan kaçınılır.

Sözleşmeleri genellikle kendimizin yazması önerilmez. Çünkü blokzincirde olabilecek en ufak bir açık bile bizim çok büyük kayıplar elde etmemize sebep olur. Genellik içinde TOKEN bulunduracak sözleşmeler için ERC-20 hazır sözleşmesi kullanılır. Ağ haberleşmesinde TCP/IP bizim için neyse SOLİDİTY içinde ERC-20'dir. Solidity için kurulum yapmamıza gerek yoktur.Browser

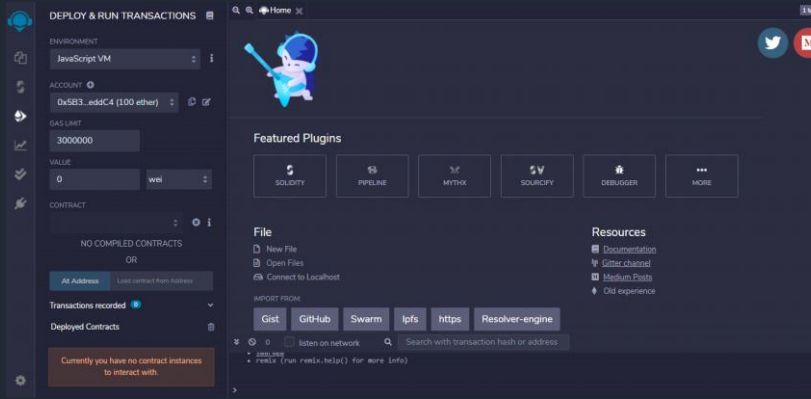
üzerinden sağlıklı bir şekilde çalışma yapabileceğimiz '<http://remix.ethereum.org/>' mevcuttur. Siteye giriş yaptığımızda böyle bi ekran bizi karşılar.



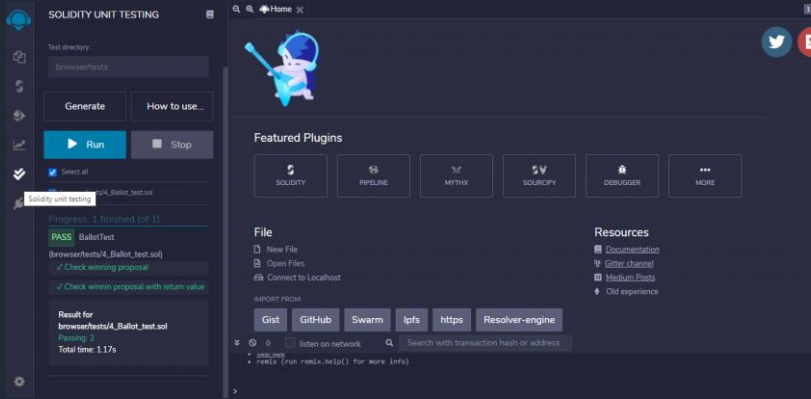
Hazır olan sözleşmeleri görebildiğimiz gibi 'Create File' sekmesine tıkladığımız da da kendi sözleşmemizi hazırlayabiliriz. Ayrıca 'Create Folder' sekmesinden de yeni klasörler oluşturabiliriz.



Bu sekmeden compile ayarlarımızı yapabileceğimizi görüyoruz.



Bu sekmeden ise sözleşmemizle ilgili ayrıntıları görüntüleyip değiştirebiliriz.



Bu sekmeden ise yazmış olduğumuz sözleşmeyi test edebiliriz.

Solidity Syntax Yapısı Anlatım İçeriği ;

1-)Pragma & Contract

5-)Dizi

9-)Keccak256

2-)Değişken tanımlama

6-)Struct

10-)Require & Modifier

3-)Operatörler

7-)Msg sender&Msg value

11-)Mapping & Events

4-)Function

8-)Memory/Storage

1-)Pragma &Contract

Öncelikle kodumuza bu satırlarla başlamamız gerekmektedir.

```
Calisma.sol x
1 pragma solidity ^0.4.18; //Çalıştığımız sürümü belirtmek için mutlaka yazılmalıdır.
2 contract Deneme {      //Kesinlikle ilk harfinin büyük olması gerekmektedir.
3
4 |
5 }
```

2) Değişken tanımlama

```
Calisma.sol x
1 uint a = 2; // Tanımlamasıyla rakam tutan bir değişken tutabiliriz.
2 //uint16,uint32,uint64,uint128,uint256 gibi türleride mevcuttur.Uint16 için 2^16 bit tutar diyebiliriz.
3
4 string b ="Deneme"; // Tanımlamasıyla ile harf tutan bir değişken tutabiliriz.
5
6 address owner; //İstedğimiz kullanıcının adres bilgilerini tutmak için kullanırız.
```

3) Operatörler

3.1-)Mantıksal Operatörler

! : Mantıksal olumsuzlama && : Mantıksal bağlantı, “ve”
|| : Mantıksal bağlantı kesilmesi , “veya” == : Eşitlik
!= : Eşitsizlik

3.2)Matematiksel Operatörler

+ : Toplama işlemi * : Çarpma işlemi
- : Çıkarma işlemi / : Bölme işlemi
% : Mod alma işlemi 5 ** 2 : 5 'in 2.kuvveti

4) Fonksiyon Tanımlama

Solidity dili fonksiyonlara bağlı olarak çalışan bir dildir. Bu yüzden fonksiyonlar büyük önem taşır.

```
Calisma.sol x
1 function deneme {
2     uint pp = 2;
3 }
```

Fonksiyon tanımlama örneği

Kurucu Fonksiyon Tanımlama

```
1 pragma solidity ^0.4.18;
2 contract Deneme {
3     string Pp;
4     function Deneme() //Buraya ekleyeceğimiz public - private kavramlarıyla fonksiyon erişimini kontrol edebiliriz.
5         //Eğer belirtmezsek default olarak public olur.
6     {
7         Pp = 'Merhabalar'; // Tanımladığım pp stringine bir yazı değişkeni atadım.
8     }
9 }
```

Bir fonksiyonun kurucu fonksiyon olarak çalışması için Contract ismiyle aynı ismi taşıması gerekmektedir. Kurucu fonksiyon isimleri her zaman büyük harf ile başlar. Kurucu fonksiyon her sözleşme çalıştığında çalışır.

Get & Set Kavramı

Dışarıdan eklenen veri boyutu arttıkça maliyette artar.

```
1 pragma solidity ^0.4.18;
2 contract Deneme {
3     string Pp;
4     function Deneme() //Buraya ekleyeceğimiz public - private kavramlarıyla fonksiyon erişimini kontrol edebiliriz.
5         //Eğer belirtmezsek default olarak public olur.
6     {
7         Pp = 'Merhabalar'; // Tanımladığım pp stringine bir yazı değişkeni atadım.
8     }
9
10    //Get fonksiyonu getirme işlemi gerçekleştirir.
11    //constant = Dışarıdan değişimi engeller. Veritabanı üzerinden değişim yapılmasını engeller.
12    function getP() public constant returns (string) {
13        return Pp ; //Bu fonksiyon çağırıldığında Pp değerini döndürür.
14    }
15    //public ile herkesin kullanımına açık bir hale getirdik.
16    function setPp(string newPp) public {
17        Pp = newPp ; //Pp değerinin yerine dışarıdan girilen bir değişkenin atanmasını sağladık.
18    }
19 }
20 }
```

Public & Private Kavramı

```
1
2 function deneme() public return (uint) { //Herkes tarafından kullanılabilen fonksiyonlardır.
3
4 }
5 function deneme2() private return (string) //Sadece belirli kişiler tarafından kullanılabilen fonksiyonlardır.
6
7 }
8 //return(Değişken türü) -> Fonksiyonun geri döndüreceği değişken türünü belirtiriz.
9
10
11
```

Payable Kavramı

```
1 function openAccount() payable public {
2
3     //Payable komutu ile para yatırma işlemlerine gerçekleştirme izni verilir.
4
5 }
```

5) Diziler

Dizi Tanımlama

```
2
3 uint [] ogrencinolari; //Uint tipinde bilgi tutan dizi tanımladık.
4
5 string[5] stringArray; //String tipinde bilgi tutan dizi tanımladık.
6
7 ogrencinolari.push(1);
8 stringArray.push("Deneme"); //Push komutu sayesinde içerisine bilgi ekleyebildim.
```

Değişken döndürme

Dizi veya string şeklinde tanımlanmış değişkenlerde geri döndürme işlemi yapılamaz. Bunun yerine 'idx' kullanılır.

```
pragma solidity ^0.4.18;
contract Deneme {
    string [] public greetings; //greetings isimli bir dizi tanımladık.

    function Deneme() public {
        greetings.push("Merhabalar...") //Bu Fonksiyon ile içerisine string push ettik.
    }
    function getGreeting(uint idx) public constant returns(string) //String değer döndüreceği için returns içine string yazılır.
    {
        return greetings[idx]; //idx olarak giriş yaptığımız değeri döndürür.
    }
    function setGreeting(string greetingMsg) public {
        greetings.push(greetingMsg); // Dışarıdan girilen greetingMsg değişkenini dizinin içersine push ettik.
    }
}
```

6) Struct

Structları birden fazla array bilgisi tutmak istediğimiz zaman kullanırız ve asla public olamaz. Amacımız düzenli bir şekilde tutmaktır.

```
3 pragma solidity ^0.4.18; //Sürümümüzü belirttik.
4 contract Calisma {
5
6 struct Deneme{ //Deneme isimli bir struct tanımladık.
7     string a;
8     uint b;
9     address c;
10 }
11 }
```

Struct
Tanımlama

7) Msg.sender & Msg.value

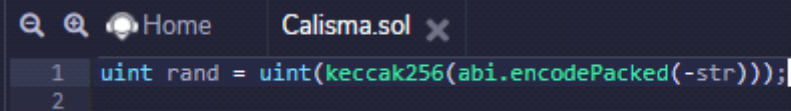
```
1 // msg.sender = Gönderen kişinin bilgisini tutar.
2 //msg.value =Dışarıdan girilen miktar bilgisini tutar.
3 pragma solidity ^0.4.18; //Sürümümüzü belirttik.
4 contract Deneme {
5 struct x {
6     address owner;
7     uint bakiye;
8 }
9 function Deneme() payable { //Kurucu fonksiyon //Payable komutu yazılmazsa msg.value kullanılamaz.
10     x.owner=msg.sender;
11     x.bakiye=msg.value;
12 }
13 }
```

8) Memory & Storage

```
function deneme() memory or storage public return (uint) {  
  //Memory : Sözleşmeyi çalıştırmanın da hafızasına kopyalama işlemi yapmak için kullanılır.BC yapısında bir değişiklik olmaz.  
  //Storage : Default olarak saklanma biçimi.  
}
```

9) Keccak 256

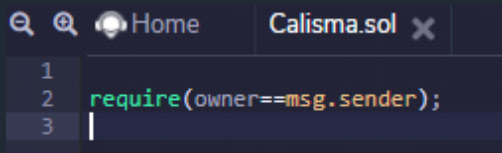
Şifreleme yöntemlerinden olan SHA-256 gibi düşünebiliriz



```
uint rand = uint(keccak256(abi.encodePacked(-str)));|
```

10) Require & Modifier

-Require

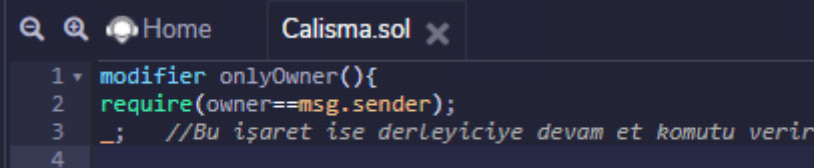


```
1  
2 require(owner==msg.sender);  
3 |
```

İçerisinde ki koşul sağlanırsa alt satırlarında ki işlemleri gerçekleştir. Genellikle sözleşmelerde sadece sözleşmeyi yazanın erişim sağlaması için kullanılır. Birden fazla değer tutarken sıkıntı oluşturabileceği gibi maliyette artar.

! NOT: 2 Adet public fonksiyon birbirini çağırmamalıdır. Bu bir güvenlik açığı oluşturur.

-Modifier



```
1 modifier onlyOwner(){  
2   require(owner==msg.sender);  
3   _; //Bu işaret ise derleyiciye devam et komutu verir  
4 }
```

*Require'nin yetersiz kaldığı yerlerde kullanılır.

*Kullanım amacı sadece sözleşmeyi yazanın erişim sağlamasının istendiği fonksiyonları yönetmektir.

11) Mapping & Events

-Mapping

```
Calisma.sol x
1 mapping(address=>AccountBalance)accounts; //Burada accounts deęiřkenine atadık.
2
3
```

Farklı iki sözleşme arasında bağlantı kurmak veya fonksiyon değerleri arasına bağlantı kurmak için kullanılır.

*Bağlantıladığımız değerleri tek bir değer olarak kullanmamızı sağlar.

*accounts[-----] şeklinde köşeli parantezle kullanılır.

-Events

```
Cal2.sol x
1 event Ad (uint a,string b,uint c);
```

*Dinleme, izleme işlemi yapılması ve ön yüze haber vermek için kullanılır.

Banka Akıllı Kontrat Örneęi

*Öncelikle bir struct yapısı, deęişkenler ve kurucu fonksiyon ile başlamak istiyorum.

```
pragma solidity ^0.4.18; //Sürümümüzü belirttik.
contract SharedAccount {
    struct AccountBalance {
        address addr; //Kullanıcının Adresi
        uint balance; //Kullanıcının bakiyesi
        bool isActive; //Kullanıcı aktif mi ?
    }
    uint public numberOfAccounts; //Yüklü account sayısını tutacağım deęişken
    uint maxAccountCount; //Oluřurulabilecek max. hesap sayısı
    function SharedAccount(uint _MaxAccountCount) public { //Kurucu Fonksiyon
        if(_MaxAccountCount != 0){
            maxAccountCount=_MaxAccountCount;
        }
        else {
            maxAccountCount = 128; //Bunu belirtmek çok önemlidir.Eęer bu sınır ařılırsa;
            //Hem sözleşmemiz bozulur hem de gas depozitomuz bořa gider.
        }
    }
}
```

Modifier yapısını teorik olarak anlatmıřtık burada uygulamalı olarak anlatalım. Artık istedięimiz fonksiyonda public kısmının gerisineonlyOwner ekleyerek sadece kurucu erişimine açabiliriz.

```
8 modifier onlyOwner(){
9     require(owner=msg.sender);
10    _; //Bu işaret ise derleyiciye devam etmesini sağlar
11 }
12 address owner; //Kurucu adresi burada tutulacak
13 uint public numberOfAccounts; //Yüklü account sayısını tutacağım deęişken
14 uint maxAccountCount; //Oluřurulabilecek max. hesap sayısı
15 function SharedAccount(uint _MaxAccountCount) public { //Kurucu Fonksiyon
16     owner=msg.sender; //Kurucu adresini ownera atıyorum.
17     if(_MaxAccountCount != 0){
18         //...
19     }
20 }
```

Ekleyeceğimiz mapping kodu ile accountları bir listede tutabileceğiz.

```
1 pragma solidity ^0.4.18; //Sürümümüzü belirttik.
2 contract SharedAccount {
3     struct AccountBalance {
4         address addr; //Kullanıcının Adresi
5         uint balance; //Kullanıcının bakiyesi
6         bool isActive; //Kullanıcı aktif mi ?
7     }
8     mapping(address => AccountBalance) accounts;
9
10
11
```

Öncelikle içinde account var mı yok mu onu kontrol edelim. Bu accountu olan birinin tekrar account oluşturmasını engeller.

```
    }
    function isAccountExists(address accountOwner) private return(bool){
        return accounts[accountOwner].addr!=address(0) && accounts[accountOwner].isActive |
    }
}
```

Daha sonrasında account açma fonksiyonunu yazalım.

```
    }
    function openAccount()payable public{ //payable Para yatırma işlemlerini sağlar.
        require(isAccountExists(msg.sender)); //Mesajı gönderenin hesabı var mı ?
        require(numberOfAccounts < maxAccountCount); //Hesap sınırı aşılmış mı ?
        accounts[msg.sender] = AccountBalance(msg.sender,msg.value,true); //AccountBalance içine değerler girilir.
    }
}
```

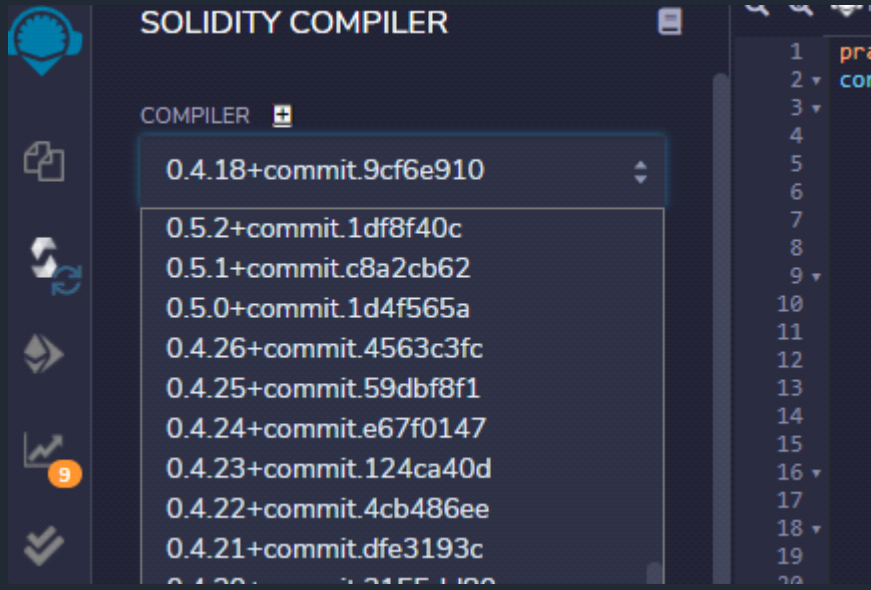
Şimdi para çekme fonksiyonunu yazalım fakat bu çekme işlemini sadece kurucu fonksiyon yapsın.

```
34     }
35     function withdrawMoney(uint amount) onlyOwner public{//Eklediğim onlyOwner sadece kurucu fonksiyonun yapabilmesini sağlar.
36         require(isAccountExists(msg.sender)); //Mesajı gönderenin hesabı var mı ?
37         require(accounts[msg.sender].balance>=amount); //Hesabında ki paradan fazla para çekmek istiyor mu ?
38
39         msg.sender.transfer(amount); //Hesabından transfer işlemi gerçekleşir.
40         accounts[msg.sender].balance -= amount; //Bakiyeden düşülür.
41
42     }
```

Son olarak para yatırma işlemi gerçekleştirelim.

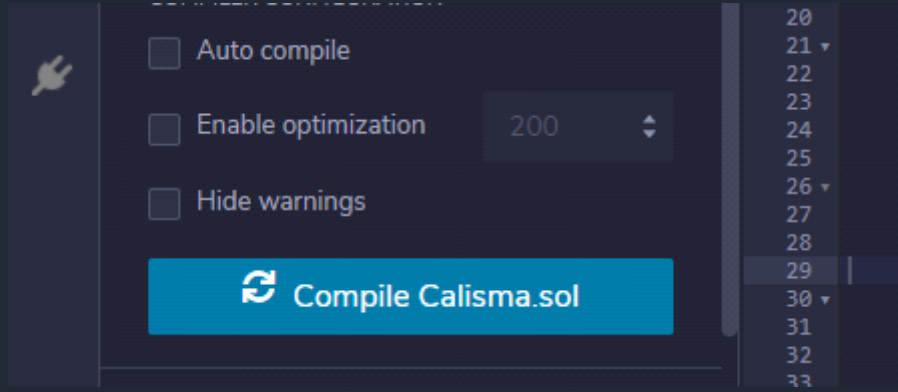
```
1     }
2     function depositMoney()payable public { //payable Para yatırma işlemlerini sağlar.
3         require (isAccountExists(msg.sender)); // Mesajı gönderenin hesabı var mı ?
4         accounts[msg.sender].balance +=msg.value; //Dışarıdan girilen değer kadar para hesaba eklenir.
5
6     }
7 }
```

COMPILE AŞAMALARI



Öncelikle sürüm kontrolü yapıyoruz.

Daha sonrasında compile ediyoruz.



Compile ettikten sonra 'Deploy' kısmına geçiş yapıyoruz.

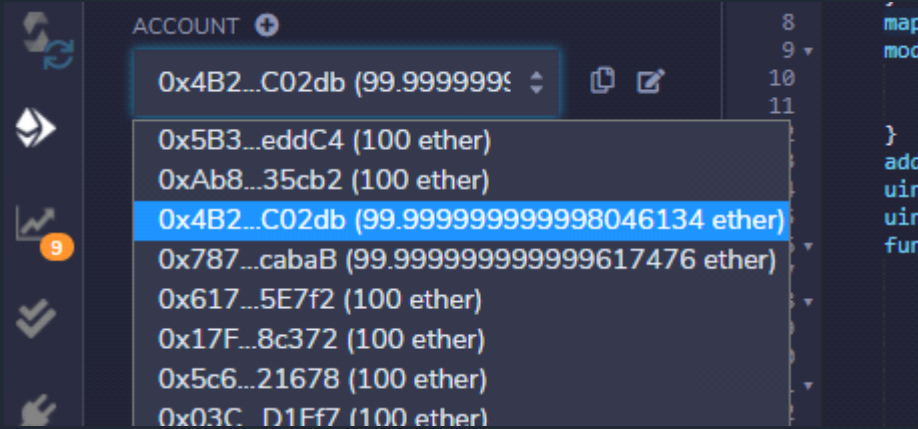
Java Script = Kendi browserımız üzerinden sabit accountlarla çalışmamızı sağlar.



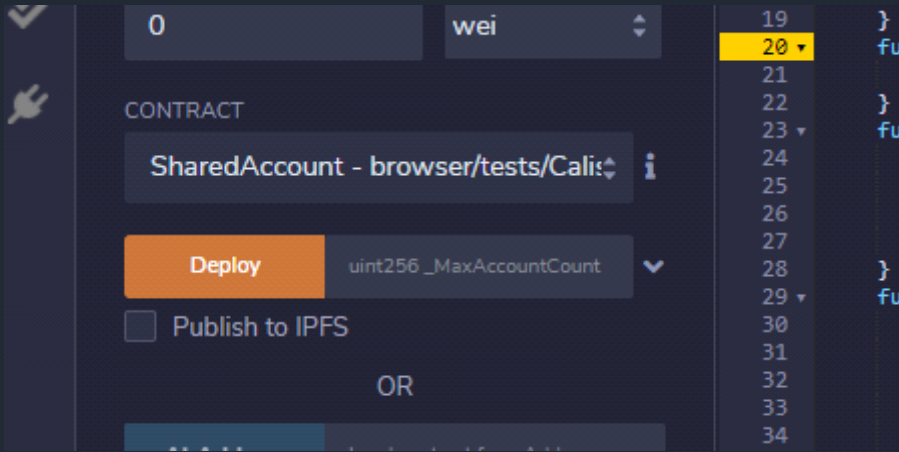
Injected Web3 = Chrome eklentisi olan Metamask üzerinden işlem yapmamızı sağlar.

Şimdi account bölümüne geliyorum. Buradan dilediğimizi seçebiliriz.

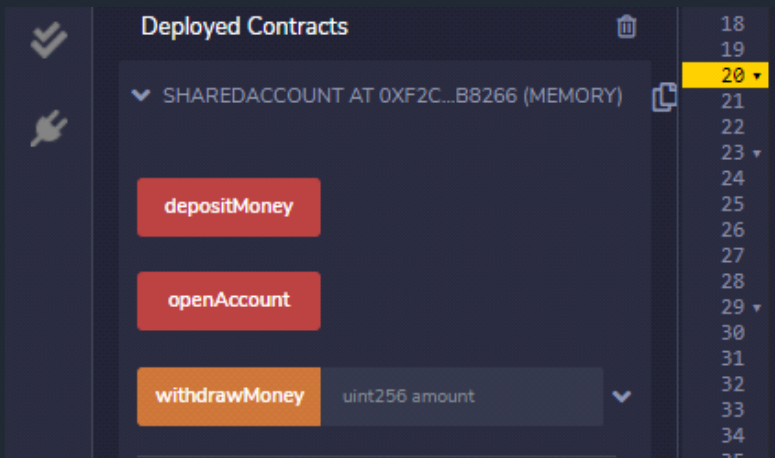
Bu accountlar SHA-256 ile şifrelenmiş biçimleridir.



'Deploy' kısmından kurucu fonksiyon içerisine atayacağımız değeri girmemiz lazım.



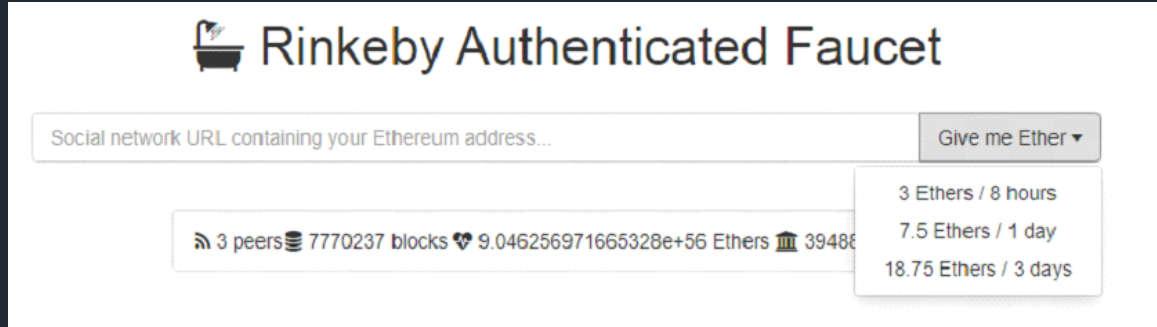
Deploy değeri de girildikten sonra aşağı tarafa kaydığımız da 'Deployed Contract' bölümüne sözleşmemizin eklendiğini görebiliyoruz. Buradan istediğimiz fonksiyonları dilediğimiz gibi kullanabiliriz.



-Sözleşmemizi JavaVM yani browser üzerinde çalıştırabildik, peki sözleşmemizi eğer paylaşmak diğer insanların da bu sözleşmeyle etkileşim içine girmesini isteseydik. O zaman devreye 'Metamask' eklentisi girecektir.

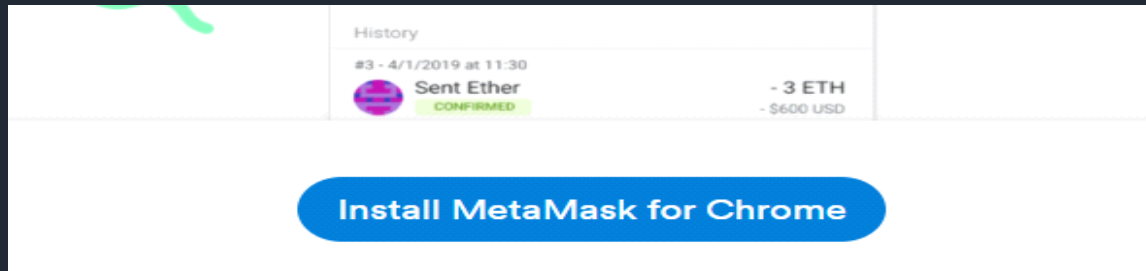
MetaMask

Bir chrome eklentisi olan metamask için bir online cüzdan diyebiliriz. Bu online cüzdanımızı ise belirli bir sosyal medyalarla paylaşım yaparak sanal para kazandıran siteler mevcuttur. Örnek olarak "https://faucet.rinkeby.io/" verebiliriz.

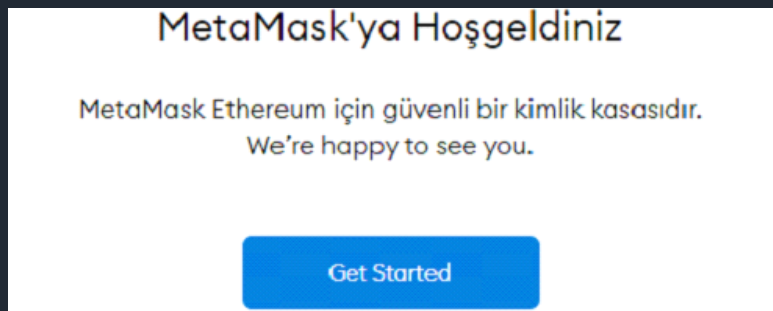
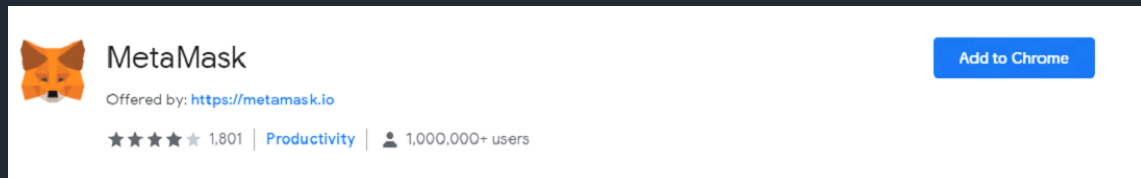


Site görünüm olarak bu şekildedir. Buraya metamask adresimizi girdiğimiz zaman bize istediğimiz koşullarda düzenli ether yüklemesi yapacaktır.

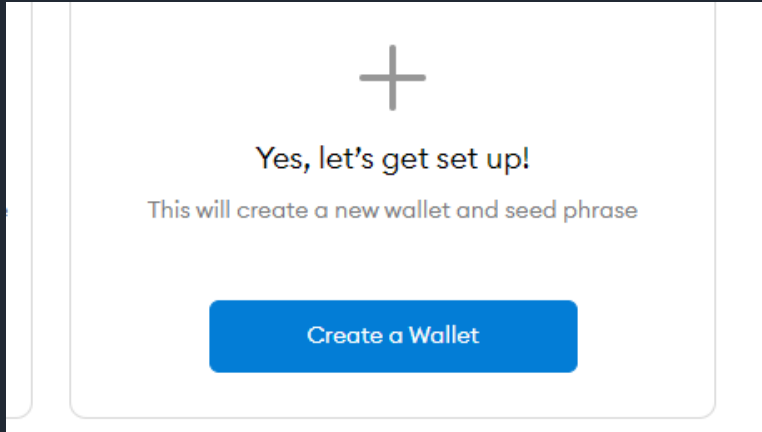
Peki MetaMask eklentisi nasıl yüklenir. "https://metamask.io/download.html" linkine girdiğimiz zaman; "Install MetaMask For Chrome" butonuna basıyoruz.



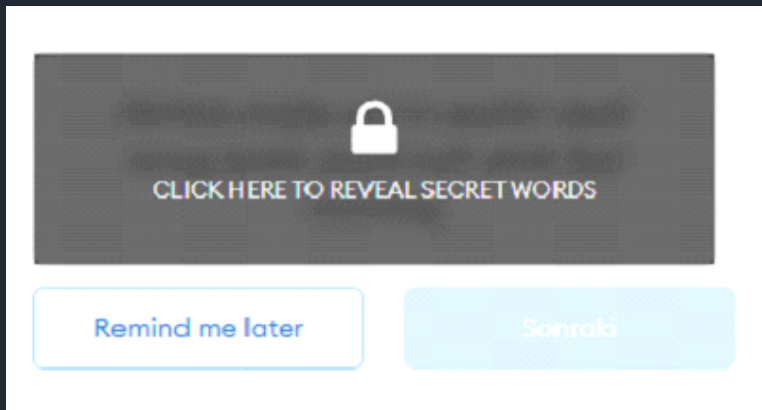
Bizi chrome' un sayfasına aktarıyor. Burada ise "Add to Chrome" butonuna tıklıyoruz.



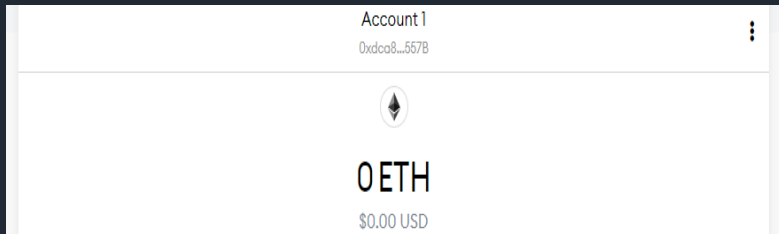
Gerekli yükleme işlemleri yapıldıktan sonra otomatik olarak bu sayfaya atacak. Buradan da "Get Started" butonuna tıklıyoruz.



Sağ tarafta bulunan "Create a wallet" butonuna tıklıyoruz.

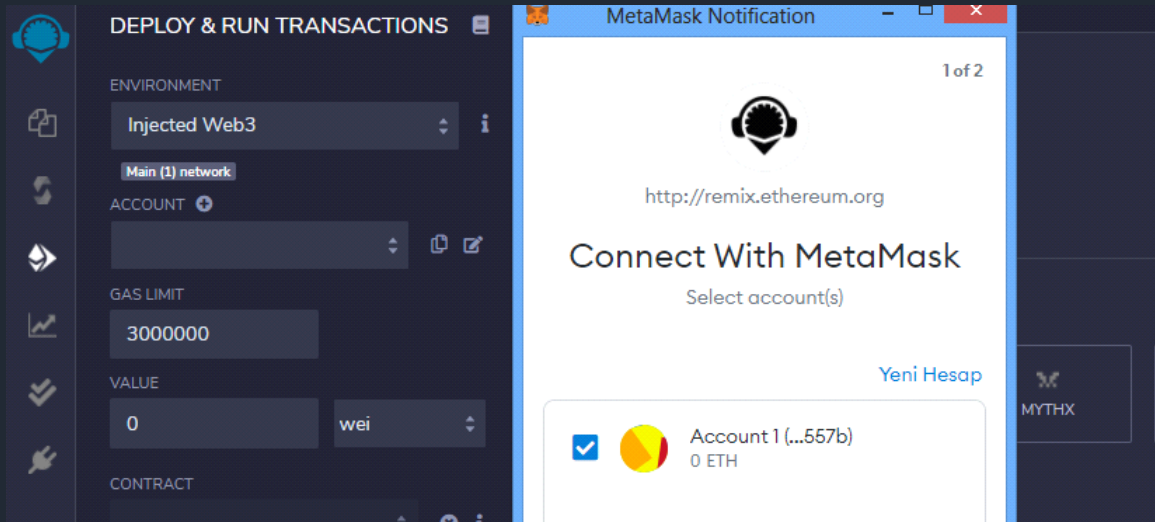


Gelen bağlantıyı "I Agree" butonuyla onayladıktan sonra şifre oluşturma ekranı gelecektir. Şifre oluşturduktan sonra bu sayfaya geçiş yapacaktır. Bu sayfada ise blurlu olan kısımda ki cümleyi bir sonraki sayfada doğru bir şekilde sıralamanız istenmektedir.



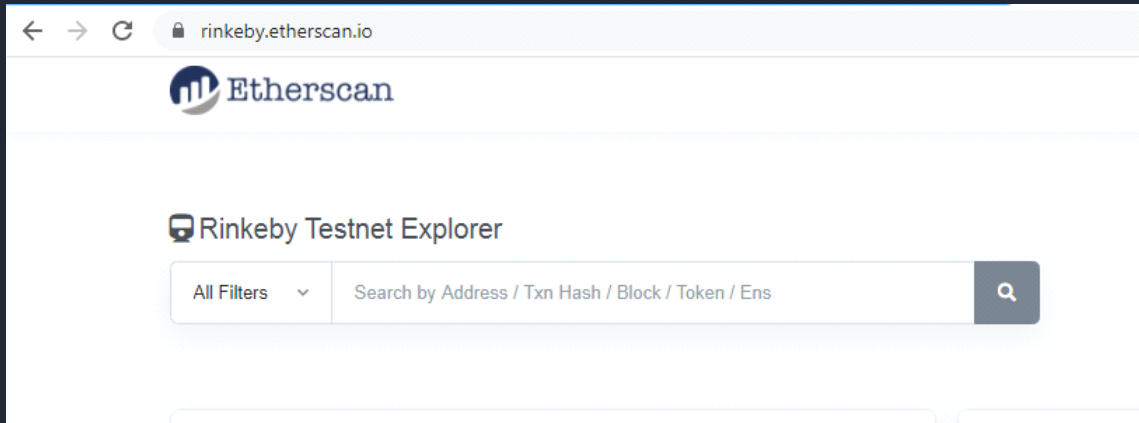
Hesabımızı oluşturmuş bulunuyoruz. Burada ise hesabımızın hash değerli adını görebiliyoruz.

Remix Ide browserımıza geri döndüğümüzde "Injected Web3" sekmesine bastığımızda;



Buradan ise istediğimiz hesabı aktif edebiliriz.

"https://rinkeby.etherscan.io/" sitesinden ise bu metamask adresini girerek yazmış olduğumuz sözleşmeleri burada da görüntüleyip kontrol sağlayabiliriz.



Solidity ile Zombi Oyunu

```
pragma solidity ^0.4.18; //Versiyonumuzu belirttik.
contract ZombieFactory { //Sözleşmemizi oluşturduk.
    struct Zombie{
        string name; //Zombimizin ismini tuttuk.
        uint dna; //Zombinin dna numarasını tuttuk.
    }
    uint dnaDigits =16; //Kaç adet dna rakamı olucak onu ayarladık.
    uint dnaModulus=10**dnaDigits; //Burada ise syntax yapısında öğrendiğimiz üst alma işlemini gerçekleştirdik.

    event NewZombie(uint zombieID,string name,uint dna); //Yeni zombi oluşturulduğunda dışarıya bilgi aktarmak için

    Zombie[] public zombies; // Public bir şekilde zombie dizisi oluşturduk.

    function _createZombie (string memory _name,uint _dna) private { // _ koymamızın sebebi private bir fonksiyon oluşturmak.
        // Bilgilerin kullanıcıda da tutulmasını istediğim için memory ekliyorum.
        uint id =zombies.push(zombie(_name,_dna));
        emit NewZombie(id,_name,_dna); //Yeni zombi oluşturulduğunu ön tarafa yayın yapmak için kullanılır.
    }

    function _generateRandomDna(string memory str) private view returns(uint){ //View ile sadece görüntülenme yapılması sağlanır.
        uint rand = uint(keccak256(abi.encodePacked(-str))); //Dışarıdan girilen string keccak256 ile rakamsal bir değer kazanır.
        //Bu değer rand isimli değişkene aktarılır.
        return rand % dnaModulus; //Bu işlem sonucunda sayımız 16 haneye düşürülüyor.
    }

    function createRandomZombie(string memory _name) public{
        uint randDna = _generateRandomDna(_name); //Oluşturduğumuz _generateRandomDna fonksiyonuna name ismini gönderdik.
        _createZombie(_name,randDna); //Fonksiyondan dönen değer ve isim ile zombimizi random bir şekilde oluşturduk.
    }
}
```

KAYNAKÇA

- BLOKZİNCİR VE KRİPTO PARALAR – BTK AKADEMİ
- ETHEREUM ÖĞRENME KLAVUZU - <https://medium.com/@denizozzgur/ethereum-%C3%B6%C4%9Frenme-k%C4%B1lavuzu-83b9734f0d1d>
- GETTING STARTED WITH METAMASK - <https://metamask.zendesk.com/hc/en-us/articles/360015489531-Getting-Started-With-MetaMask-Part-1->
- ETHEREUM IS GAME-CHANGING TECHNOLOGY, LITERALLY - <https://medium.com/@virgilqr/ethereum-is-game-changing-technology-literally-d67e01a01cf8>
- A BEGINNER'S GUIDE TO ETHEREUM - <https://blog.coinbase.com/a-beginners-guide-to-ethereum-46dd486ceecf>
- WHAT IS ETHEREUM - <https://education.district0x.io/general-topics/understanding-ethereum/what-is-ethereum/>
- HOW DOES ETHEREUM WORK, ANYWAY? - <https://preethikasireddy.medium.com/how-does-ethereum-work-anyway-22d1df506369>
- A GENTLE İNTRODUCTION TO ETHEREUM - <https://bitsonblocks.net/2016/10/02/gentle-introduction-ethereum/>
- INTRODUCTION TO SMART CONTRACTS - <https://docs.soliditylang.org/en/v0.5.3/introduction-to-smart-contracts.html>
- AKILLI KONTRAT NEDİR NASIL ÇALIŞIR? - <https://www.icrypex.com/tr/blog/akilli-kontrat-nedir-nasil-calisir#:~:text=Ak%C4%B1l%C4%B1%20kontratlar%2C%20al%C4%B1c%C4%B1%20ve%20sat%C4%B1c%C4%B1,bir%20blockchain%20a%C4%9F%C4%B1%20%C3%BCzerinde%20bulunmaktad%C4%B1r.&text=Burada%2C%20ak%C4%B1l%C4%B1%20kontratlar%C4%B1n%20bir%20%E2%80%9Ce%C4%9Fer..>
- BLOKZİNCİR TEKNOLOJİLERİ - <https://blokzincir.bilgem.tubitak.gov.tr/blok-zincir.html>
- ETHEREUM (ETH) NEDİR? - <https://www.btcturk.com/bilgi-platformu/ethereum-eth-nedir/>
- BLOKZİNCİR TABANLI SİBER GÜVENLİK SİSTEMLERİ - <https://dergipark.org.tr/en/download/article-file/396266>
- ÖĞRENCİLER İÇİN ETHEREUM VE SOLİDİTY EĞİTİMİ - <https://www.youtube.com/watch?v=Nsi82G2x3xk>
- KAMUDA VERİ GÜVENLİĞİ İÇİN BLOKZİNCİR - <https://www.haberturk.com/ankara-haberleri/16971931-kamuda-veri-guvenligi-icin-blokzinciri-onerisi>