



MAARİF MEKTEPLERİ 72

© Maarif Mektepleri Eğitim Basın Yayın Danışmanlık
Org. Araş. İnş. Tur. San. ve Tic. Ltd. Şti.
Maarif Mektepleri Yayınları, Ankara

Algorand SDK ve Pyteal İle Algorand Teknolojileri
Yazarlar: Aybars Göktuğ Ayan

Yayın Yönetmeni: Raşit Akyüz

Editör: İbrahim Sertkaya

Redaksiyon: Buğra Ayan

Tasarım ve Uygulama: GRIFFİN [0312 419 1619]

Kapak Tasarım: GRIFFİN

Birinci Baskı: Ekim 2022

Baskı: Tarcan Matbaa

Sertifika No: 47663

Adres:

İvedik Cad. No: 417 Yenimahalle/ANKARA

ISBN: 978-605-71507-3-8

Sertifika No: 44141

Adres:

Kızılay Mah. Gmk Bul. Fevzi Çakmak 2 Sok.

No: 37/1 Çankaya /Ankara

0312 419 16 19

www.maarifmektepleri.com.tr

info@maarifmektepleri.com.tr

maarifmektepleri@gmail.com

facebook.com/maarifmektepleri

instagram.com/maarifmektepleri



Bu kitaptaki hiçbir bilgi yatırım tavsiyesi değildir.

ALGORAND SDK VE PYTEAL İLE
ALGORAND TEKNOLOJİLERİ

AYBARS GÖKTUĞ AYAN



AYBARS GÖKTUĞ AYAN

Konya Teknik Üniversitesi Elektrik Elektronik Mühendisliği bölümünde 3. sınıf öğrencisidir. 2015 yılında profesyonel mobil uygulama geliştiriciliği ile yazılım konusundaki yolculuğuna başlayan Ayan, derin öğrenme, blokzincir ve bilgisayarlı görü teknolojileri ile devam etmektedir. 3 yıldır JavaScript, Python ve derin öğrenme teknolojileri üzerinde çalışmaktadır. 2020 Aralık ayından itibaren birçok blokzincir projesinde geliştirici ve moderatör olarak yer almıştır. 3 yıldır bulunduğu Konya Teknik Üniversitesi Yapay Zeka ve Görüntü İşleme Topluluğunda Başkan Yardımcısı görevinde 2 yıldır yer almaktadır. BTK Akademi kurumunda stajini gerçekleştiren Ayan blokzincir teknolojilerinin topluma yayılması konusunda çalışmalarına devam etmektedir.

*Her daim yanımda olup desteğini hiç esirgememiş İrem Karabaş'a
teşekkürlerim ile...*

İÇİNDEKİLER

BİRİNCİ BÖLÜM

ALGORAND NEDİR?.....	9
Algorand Dokümantasyon.....	9
Algorand SDK ve Pyteal.....	10
Gerekli Ortamların Kurulması.....	11
JavaScript Ortamlarının Kurulması.....	12
JavaScript Üzerinde Gerekli Dosya ve Paketlerin Kurulması.....	13
Python Ortamlarının Kurulması.....	14
Python'da Gerekli Ortamların Kurulması.....	16
JavaScript ve Algorand İle Yeni Hesap Açma.....	17
JavaScript ve Algorand İle Hesabın Bakiyesini Kontrol Etme.....	21
JavaScript ve Algorand İle Transfer İşlemi Gerçekleştirme.....	30
JavaScript ve Algorand İle ASA Token İşlemleri.....	42
ASA Oluşturulması İçin İlgili Hesapların ve Araçların Kurulması.....	42
ASA Token Oluşturma.....	48
Farklı Bir Hesaba ASA Ekleme ve Çıkarma.....	56
Token Bakiye Kontrolü.....	66
Adresler Arasında Varlık Transferi.....	70
Oluşturulan Token'i Ortadan Kaldırma.....	80

İKİNCİ BÖLÜM

PYTEAL İLE PYTHON ÜZERİNDE AKILLI KONTRAT GELİŞTİRME.....	91
Docker Kurulumu.....	91
Sandbox Kurulumu.....	92
Pyteal Araçlarının Kullanılması.....	96
Akıllı Kontratın Başlatılması.....	100

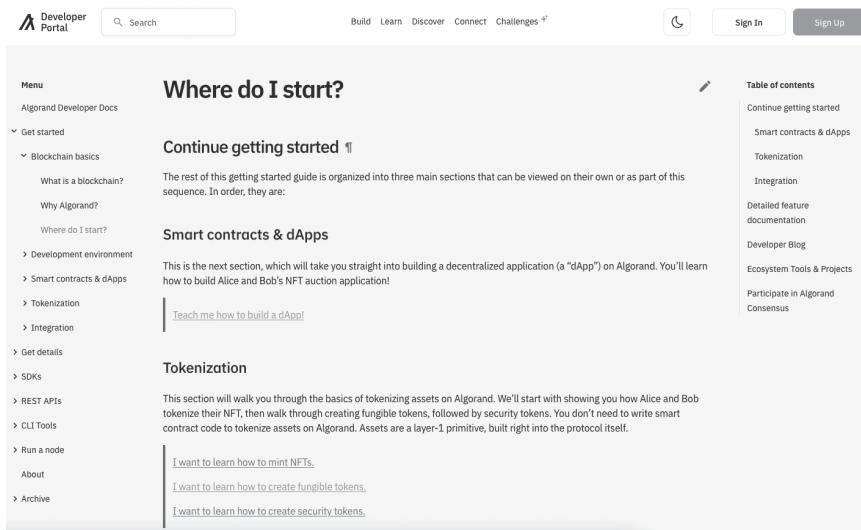
BİRİNCİ BÖLÜM

ALGORAND NEDİR?

Algorand, blokzincir üçgeni adını verdığımız hız, güvenlik ve merkezi-yetsizlik konularını tek başına en verimli şekilde çözmeyi hedefleyen merkezi-yetsiz ağıdır. 2017 yılında Micali tarafından yönetilen takım tarafından kurulmuştur. Silvio Micali kriptoloji üzerine çalışmalar yapan bir bilgisayar mühendisidir. MIT'de profesörlük yapmaktadır. Ayrıca 2012 yılında kripto-loji ile ilgili yaptığı çalışmalar nedeniyle Turing ödülüne layık görülmüştür. Algorand, Ethereum alternatif olarak tasarlanmıştır ancak Ethereum'un ak-sine neredeyse anlık tamamlanma hızına ulaşan işlem hızına sahiptir. Saniye-de 1000 işlem tamamlayabilmektedir. Gerçekleşen her transfer 5 saniyeden önce tamamlanmaktadır. Pure Proof of Stake (PPoS) adı verilen uzlaşma yöntemini kullanmaktadır. Şu anda 120 doğrulayıcı tarafından güvenlik altına alınmaktadır. ERC-20 protokolüne benzer olarak Algorand Standart Asset mekanizmalarına sahiptir. Bu mekanizma sayesinde zincir üzerinde akıllı kontrat gerektirmeden varlık oluşturmaya elverişlidir. Algorand aynı zamanda Co-chain yapıyı desteklemektedir. Bu yapı sayesinde kullanıcılar sadece belirli kişiler tarafından erişilebilen private blokzincirleri oluşturabilirler. Kitabımızda da bahsedeceğimiz üzere üzerinde akıllı kontrat geliştirmeyi sağlayan bir blokzincirdir.

Algorand Dokümantasyon

Algorand yapısı dahilinde geliştiricilere büyük önem vermektedir. Algorand Foundation vakfı bu anlamda protokol, açık kaynak, denetim ve eğitim gibi konularda çalışmaktadır. Blokzincir teknolojisi kısmen yeni gelişen bir teknolojidir. Bu nedenle teknolojiyi kullanmayı hedefleyen kişilerin kullanabileceği belgelendirmelerin bulunması gerekmektedir. Algorand vakfı bu duruma büyük önem vermektedir ve düzenli olarak güncellenen gelişen dokümantasyonlara sahiptir. Bu dokümantasyonların bir araya toplandığı ve kullanıcılarına sunulduğu site olan <https://developer.algorand.org/> sitesi kitap içerisinde de bolca kullanılacaktır.



Şekil 1

Şekil 1'de Algorand Developer Portal'ın kitabıne yazılma tarihine bağlı görünümü ifade edilmiştir. Yan kısmında bulunan menü'lere göz gezdirildiğinde birçok başlık olduğu gözükmemektedir. Algorand blokzinciri üzerinde büyük geliştirme yapmayı planlayan geliştiricilerin bu siteye büyük önem vermesi ve güncel olarak taraması gerekmektedir.

Algorand SDK ve Pyteal

Algorand bünyesinde iki adet ana geliştirme ortamı barındırmaktadır. Bu geliştirme ortamları SDK (Software Development Kit) ve Python üzerinde bulunan Pyteal kütüphanesi olmaktadır. Her ikisi de bazı alanlarda birbirlerini kullansalar da temel olarak görevleri birbirinden farklılık göstermektedir.

- **Algorand SDK:** Geliştiricilerin akıllı kontrat geliştirmeden zincir üzerinde bazı işlemler yapmasını sağlamaktadır. Bu işlemler adres oluşturma bakiye kontrol etme veya token oluşturma işlemleri olabilir. Algorand SDK şu anda 4 yazılım dili üzerinde desteklenmektedir. Bu yazılım dilleri Python, JavaScript, Go ve Java olmaktadır. Aslında bu dillerin her birinin yoğunluğu farklı kısımlar bulunmaktadır örneğin Python arka-uç (backend) programlamada kullanılarak JavaScript ön-uç (frontend) programlamada kullanılmaktadır. Bu alan farklılığı aslında Algorand SDK'nın çok farklı alanlarda farklı işlemlerin yapılmasına olanak sağladığından kaynaklanmaktadır.

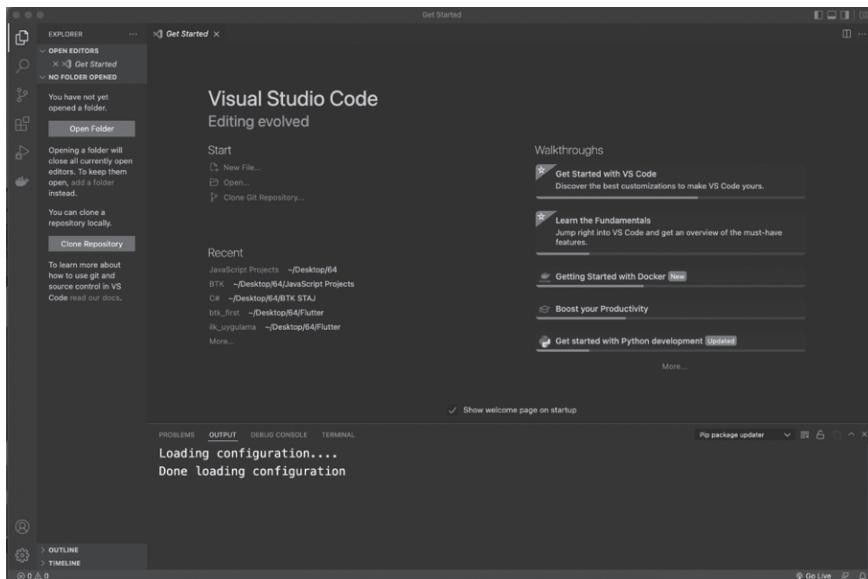
- **Pyteal:** Algorand tarafından geliştirilen Teal dilinin Python üzerinde erişilmesini sağlayan bir kütüphanedir. Sadece Python dili içerisinde erişilebilir. Amacı Algorand zincirinde çalışacak verimli ve etkili akıllı kontrat yazmayı kolay bir şekilde gerçekleştirmektir.

Kitap içerisinde iki ortamdan da bahsedilecek örnek projeler hayatı geçirilecektir.

Gerekli Ortamların Kurulması

Projelerimiz için öncelikli olarak Python ve JavaScript dilleri kullanılacaktır. Bu nedenle bu diller ile ilgili bazı programların kurulması gerekmektedir.

Projemiz içerisinde hem JavaScript için hem de Python için VS Code IDE'si tercih edilmiştir. Bu IDE'yi indirmek için <https://code.visualstudio.com/> sayfası ziyaret edilebilir. Kurulum dosyası indirildikten sonra işletim sistemine özgü kurulum aşamaları tamamlanır.



Şekil 2

Kurulum tamamlandığında Şekil 2'da ifade edildiği şekilde bir arayüz bizi karşılamaktadır. Bu arayüz içerisinde sol kısmında bulunan biri kopuk dört karenin olduğu seçenek seçilmelidir. Bu seçenek içerisindeki gerekli eklentiler VS Code içerisinde eklenebilmektedir.

JavaScript Ortamlarının Kurulması

JavaScript üzerinde geliştirmeler yapılmırken Node.js programına ihtiyaç duyulmaktadır. Node.js JavaScript kodlarını tarayıcı dışında çalıştırılmamızı sağlayan bir çalışma ortamıdır. Kurulum için [adresi](https://nodejs.org/en/) ziyaret edilmektedir.



Şekil 3

Şekil 3'te ifade edildiği üzere site içerisine girdiğimizde iki adet indirme seçeneği karşımıza çıkmaktadır. Bu sürümler LTS ile ifade edilen (Long Term Support) uzun süreli desteklenen sürüm ile “Current” yani güncel sürümü ifade etmektedir. Her ikisi de bir sıkıntısı çıkmadan kullanılabilse de projemiz için LTS sürümünü tavsiye etmekteyiz. Sürüm seçildikten sonra işletim sisteme uygun bir şekilde kurulumunun yapılması gerekmektedir.

Node.js kurulumu sonrasında VS Code üzerinden JavaScript ile ilgili bazı eklemeler yapılması gerekmektedir. Bu eklemeleri eklentiler sekmesinden yapmak mümkündür.



Şekil 4

İlk olarak eklenmesi gereken eklenti Şekil 4'te ifade edildiği üzere JavaScript dosyalarını daha doğru bir şekilde çalışmasını sağlayan “JavaScript (ES6)” eklentisidir. Eklentiler kısmından JavaScript araçları ile görüntülenebilir.

JavaScript Üzerinde Gerekli Dosya ve Paketlerin Kurulması

Uygun geliştirme ortamının kurulmasından sonra bu geliştirme ortamı üzerinden erişilecek klasörün oluşturulması gerekmektedir. Bu klasör üzerinde tüm geliştirme işlemlerini yapacak olmaktadır.

İlk olarak istediğimiz konumda oluşturduğumuz klasörü kod editöründe “Open Folder” tuşuna basarak açmamız gerekmektedir. Bu sayede klasörümüz editörümüz içerisinde rahatça erişilebilecektir. Sonrasında bu klasöre terminal üzerinden de erişmemiz gerekmektedir. Bu işlem bazı durumlarda klasör ekleme ile anında VS Code içinde gerçekleşebilse de proje klasörünüz editör üzerinden açığınız klasör içerisinde bulunan birçok klasörden biriyse otomatik olarak terminal klasöre bağlanmamaktadır. Bu durumda “cd”, “ls” ve “cd ..” komutları ile terminalden proje dosyanızı bulana kadar arama yapmanız gerekmektedir.

```
npm init -y
```

Tablo 1

Terminalden klasör içerisine girildikten sonra Tablo 1'de ifade edildiği şekilde komut çalıştırılmalıdır. Bu komut klasörü bir node.js klasörü haline getirecek ve lokal olarak makinemizde JavaScript dosyalarını çalıştırabileceğimiz klasör haline gelecektir. Bu işlem sonrası "package.json" adında bir dosya olduğu dikkatinizi çekebilir. Bu dosya projemizde kullanılan JavaScript paketlerini içermektedir.

JavaScript içerisinde kullanacağımız bazı paketler bulunmaktadır. Bu paketleri yine node.js ile yüklememiz mümkün olmaktadır.

```
npm install algosdk
```

Tablo 2

İlk ekleyeceğimiz paket Algorand zincire bağlanmamızı sağlayan ve kitap içerisinde öğrenimini gerçekleştireceğimiz "algosdk" paketi olmaktadır. Tablo 2'de ifade edilen komutu oluşturduğumuz proje klasörü içerisinde çalıştırırmamız durumunda Algorand SDK paketine erişim sağlayabiliriz.

```
npm install dotenv
```

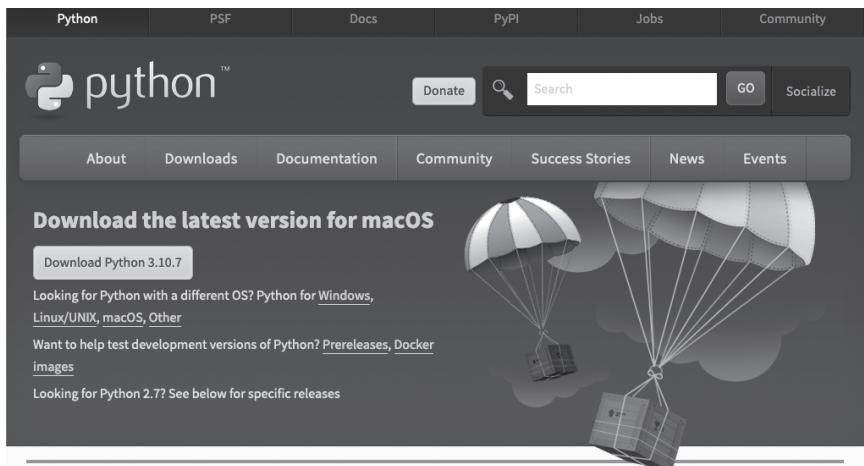
Tablo 3

Ekleyeceğimiz son paket proje klasörü içerisinde gizli bilgileri güvenli bir şekilde tutmamızı sağlayan paket olan "dotenv" paketinin eklenmesidir. Tablo 3'de ifade edilen komutu oluşturduğumuz proje klasörü içerisinde çalıştırırmamız durumunda ".env" ismine sahip dosyalara projemiz içerisinde erişim sağlayabiliriz.

Projemizde kullanılacak paketlerin kurulumu tamamlanmıştır. Bu paketleri kullanarak Algorand Blokzinciri üzerinde hesap oluşturacak, bakiye sorgulayacak, transfer yapacak ve token oluşturacağız.

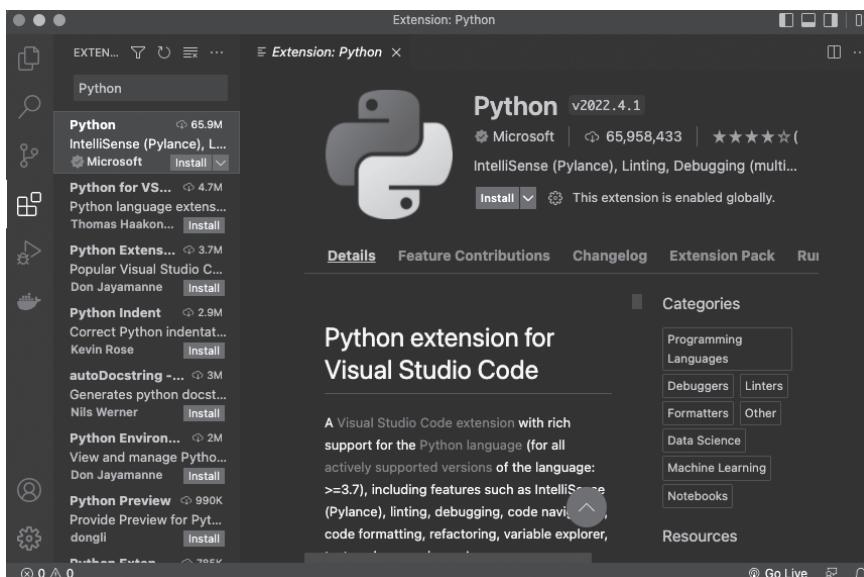
Python Ortamlarının Kurulması

Kitabımız üzerinde öğrenimini sağlayacağımız Pyteal ve Python Algosdk ortamlarının çalışması için bilgisayarımız üzerinde Python'ın kurulu olması gerekmektedir. Bu kurulumu <https://www.python.org/downloads/> adresi üzerinden gerçekleştirebiliriz.



Şekil 5

Şekil 5’de ifade edildiği üzere uygun işletim sistemi sürümüne göre Python’ı indirmek ve kurmak gerekmektedir.



Şekil 6

Python’ın kurulması sonrasında seçtiğimiz editör olan VS Code içerisinde Şekil 6’da ifade edildiği üzere Python dosyaları yazmamız için “Python” eklentisini kurmamız gerekmektedir. Bu eklenti sayesinde VS Code içerisinde Python dosyalarını otomatik tamamlama ve benzeri özellikler ile kullanabiliriz.

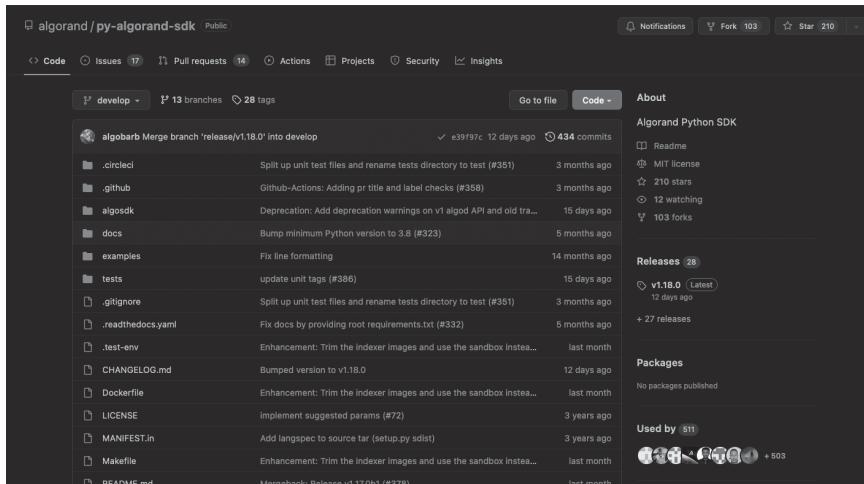
Python'da Gerekli Ortamların Kurulması

Python üzerinde geliştirmeye başlamak için ilk olarak proje dosyalarının bulunduğu bir klasörün oluşturulması gerekmektedir. Bu işlem terminal üzerinden veya kod üzerinden yapılabilir. Klasör oluşturulduktan sonra klasörün içeresine istediğimiz ismi verebileceğimiz “main.py” dosyası oluşturulmalıdır. Ayrıca Python içerisinde “pip” yapısı ile gelmektedir. Bu yapı Python dosyası üzerinde erişilebilecek kütüphanelerin oluşmasına imkân vermektedir.

```
pip install algosdk
```

Tablo 4

Python üzerinden Algorand SDK'ye erişebilmek için kütüphanenin indirilmesi gerekmektedir. Bu işlem Tablo 4'de ifade edildiği üzere “pip install” komutu ile gerçekleştirilebilir. Bazı durumlarda kütüphane ekleme işlemi düzgün gerçekleşmeyebilir. Bu gibi durumlarda kütüphane manuel olarak indirilebilir.



Şekil 7

Manuel ekleme işlemi Şekil 7'de ifade edildiği üzere Github klasörü üzerinden gerçekleştirilmektedir. Bu klasöre <https://github.com/algorand/py-algorand-sdk> linki üzerinden erişmek ve “pip install [dosya-yolu]” komutu ile yüklemek mümkün olmaktadır.

JavaScript ve Algorand İle Yeni Hesap Açma

JavaScript ile akıllı kontrat geliştirmeye başlamadan önce bu akıllı kontratların etkileşime geçeceği hesapların oluşturulması gerekmektedir. Bu hesapların içerisine kripto para gönderebilecek bakiye sorgulayabilecek ve transferlerini görüntüleyebileceğiz. Bu adresler içerisinde iki ana eleman taşımaktadır.

- **addr:** Adresi ifade etmektedir bu adresi hesabın ismi olarak da düşünebiliriz. Bu adresi istediğimiz kadar kişiyle güvenlik endişesi yaşamadan paylaşabiliriz. Bu adresе sahip olan herkes hesabın bulunduğu bakiyeyi ve gerçekleştirdiği transferleri görüntüleyebilir.
- **sk:** Açılımı “secret key” olan bu parametre aslında hesabın kendisi olmaktadır. Hesap üzerinde gerçekleştiren tüm işlemler bu anahtar sayesinde gerçekleştirir. İsminin de belirttiği üzere saklı ve güvende tutulmalıdır. Bu anahtara sahip olan herkes hesabın mutlak kontrolünü eline almış olur ve hesapta transfer ve benzeri işlemleri gerçekleştirebilir.

İlk olarak önceki bölümde gerekli paket kurulumlarını gerçekleştirdiğimiz index.js dosyasının bulunduğu klasöre VS Code içinde giriş yapmaktadır. Bu klasör içerisinde yeni bir JavaScript dosyası oluşturacağız. Bu durum kesin gerekli olmamak ile projenin zamanla büyümesi sonucunda bize hız kazandıracak ve daha derli toplu bir kod arayüzü yazmamızı sağlayacaktır. Girdiğimiz klasörün içerisinde “address.js” adında yeni bir klasör oluşturacağız. Yazdığımız kod parçası bu dosya içerisinde yer olacaktır.

JavaScript dosyasını oluşturduğumuzda karşımıza boş bir kod dosyası çıkmaktadır. İlk olarak bu dosya içerisinde kullanacağımız paketleri eklememiz gerekmektedir. Adres oluşturmak için sadece Algorand SDK'sının paket halini aldığı algosdk paketini yüklememiz yeterli olmaktadır.

```
const Algosdk = require("algosdk")
```

Tablo 5

Tablo 5'te paketin kod içerisinde eklenmesi gösterilmiştir. JavaScript üzerinde paketler “require()” fonksiyonu ile eklenmektedir. Paketi “require()” ile çağrıdıktan sonra kod içerisinde özgürce çağırılabilmemiz için bu paketi bir sabit veya değişkene eşitlememiz gerekmektedir. Bu nedenle Algosdk adında bir değişken oluşturup tanımlamayı gerçekleştiririz. Sabitin adının büyük olmasının nedeni kod içerisinde takibini kolaylaştırmaktır. Paket adından bağımsız bir şekilde istenilen tüm isimler bu sabite tanımlanabilir.

```
let hesapAdresi = ()=>{  
}
```

Tablo 6

Paketi ekledikten sonra dosyamızın içerisinde adres oluşturma işleminin yer alacağı bir fonksiyon oluşturmamız gerekmektedir. Bu fonksiyon yapısını Tablo 6'da gösterildiği gibi oluşturmamız mümkündür. Bu fonksiyon yapısına “Arrow Function” adı verilir ve gösterildiği şekilde tanımlanır. Başlangıçta bu fonksiyonu ifade edecek değişkenin oluşturulması gerekmektedir. Bu işlem için “const” adında bir değişken kullanmak yerine “let” ifadesini kullanmaktadır. Bu durumda neden fonksiyonun değişken değerler tutmasını istememizden kaynaklanmaktadır. Fonksiyon adı tamamen tercihe dayalı olsa da anlam bütünlüğü olması amacıyla “hesapAdresi” isimlendirmesi kullanılmıştır. Parametre kısmı olan parantez içerisinde herhangi bir parametre gerektirmemesinden dolayı boş bırakılmıştır.

```
let Hesap = Algosdk.generateAccount()  
console.log("Hesap Adresi: ", Hesap);
```

Tablo 7

Fonksiyon içerisinde girdiğimizde Tablo 7'de ifade edildiği şekilde adres oluşturması yapmamız gerekmektedir. Bu işlemi “let” ile oluşturduğumuz “Hesap” değişkeni üzerinde Algosdk paketinin “generateAccount()” metodu ile gerçekleştirebiliriz. Sonrasında bu değişkeni “console.log()” ile ekranımıza yazdırabiliriz. Eğer fonksiyonu bu şekilde çalıştırmak istersek fonksiyon dışına kodun en sonuna “hesapAdresi()” ibaresi ile gerçekleştirebiliriz. Sonrasında terminalden kodun bulunduğu klasör içerisinde “node address.js” kod satırı ile kodu çalıştırabiliriz.

```
Hesap Adresi: {  
  addr:  
    'TJA62NSXJ0A5SD5F6PWOH3RTILYHG2QIHKRKBX6RNPJPIEH6YV3ZC217E',  
    sk: Uint8Array(64) [  
      44, 69, 34, 14, 200, 215, 127, 227, 219, 165, 47,  
      108, 73, 211, 102, 116, 79, 235, 78, 45, 230, 191,  
      230, 229, 155, 182, 117, 66, 140, 44, 86, 4, 154,  
      65, 237, 54, 87, 74, 28, 14, 200, 125, 47, 159,  
      103, 31, 113, 154, 23, 131, 155, 80, 65, 213, 21,  
      6, 254, 139, 94, 151, 160, 135, 246, 43  
    ]  
}
```

Tablo 8

Kodu bu şekilde çalıştığımızda karşımıza Tablo 8'de ifade edildiği üzere bir obje türüne sahip verinin çıktığını gözlemeğekteyiz. Aslında bu obje hesabın ta kendisidir. Bu objenin içerisinde iki adet anahtar bulunmaktadır.

- addr anahtarı içerisinde hesabın adresini bulundurmaktadır. Bu adres hesabın kimliği olmaktadır herhangi bir transfer ve benzeri işlem gerçekleştiğinde zincirde adresi belirtmek için bu adres kullanılır ve paylaşılabilir.
- sk anahtarı içerisinde hesabın “secret key” değerini yani gizli anahtarını bulundurmaktadır. Bu anahtar hesabın mimarisi konumunda olmaktadır. Yapılacak tüm transferler bakiye tutma işlemleri ve benzeri işlemler bu anahtar üzerinde gerçekleşmektedir. Anahtara sahip olan kişinin căzdana da istediği gibi sahip olabilmesi durumundan dolayı paylaşımı son derece risklidir ve tavsiye edilmez. 64 integer değerinden oluşur. Okunması ve anlaşılması zor durumdadır. Güvenli tutulması gerekmektedir.

```
let MnemonicAnahtar = Algosdk.secretKeyToMnemonic(Hesap.sk)
console.log("Mnemonic Şifre: ", MnemonicAnahtar);
```

Tablo 9

Secret Key'in yani gizli anahtarın gizli tutulması gereğinden bahsettiğimiz ancak bu kadar karmaşık ve uzun bir integer değerinin kolay ve anlaşılır hale getirilmesi için Mnemonic şifrelerin kullanılması gereği belirlenmiştir. Bu Mnemonic şifreler secret key kullanılarak türetilmektedir. Tablo 9'da bu anahtarların tanımladığımız fonksiyon içerisinde algosdk paketinin “secretKeyToMnemonic()” metodu ile çağrılmışlığını gözlemeğekteyiz. Bu fonksiyon içerisinde secret key değerini almaktadır. Bu değere bir önceki satırlarda tanımladığımız Hesap objesinin “sk” anahtarı ile erişebilir ve metod içerisine verebiliriz. Sonrasında “console.log()” ile MnemonicAnahtar adı ile kaydettiğimiz değişkeni ekrana yazdırabiliriz.

Mnemonic Şifre: bean six digital arrange genre soap citizen ramp provide moon matrix random volume hill tissue expand toast client pizza hockey fortune hobby tonight absorb sure

Tablo 10

Tablo 10'da gösterildiği üzere secret-key'den türeyen Mnemonic şifre kolaylıkla okunabilen az heceli İngilizce kelimelerden oluşmaktadır. Ge-

nellikle bu anahtarın oluşturuluktan sonra fiziki ve sanal olarak bir yere kaydedilmesi tavsiye edilir. Bu anahtarın kaybolması sonucunda adresе dair her şeyi kaybetmiş olunur. Bu nedenle güvenli şekilde kaydedilmelidir.

```
module.exports= hesapAdresi
```

Tablo 11

Fonksiyonun tamamlanmasından sonra oluşturduğumuz address.js dosyasının çıktı verebilmesini sağlamak amacıyla fonksiyonumuzu “export” etmemiz gerekmektedir. Bu işlemi Tablo 11’de ifade edildiği üzere “module.export” fonksiyonu ile gerçekleştirmektedir. Fonksiyonumuzu tanımlarken değişken olmadığı için parantezsiz tanımlama yapabilmektedir.

```
const Algosdk = require("algosdk")

let hesapAdresi = ()=>{
let Hesap = Algosdk.generateAccount()
console.log("Hesap Adresi: ", Hesap);

let MnemonicAnahtar = Algosdk.secretKeyToMnemonic(Hesap.sk)
console.log("Mnemonic Şifre: ", MnemonicAnahtar);
}

module.exports= hesapAdresi
```

Tablo 12 address.js Tamamı

Tüm address.js dosyasının içerisindeki kodu Tablo 12’de ifade edildiği şekilde açıklamak gereklidir;

- İlk olarak paket tanımlarız.
- Sonrasında adresi içerisinde oluşturacağımız fonksiyonu tanımlarız.
- Fonksiyonun içerisinde adresi SDK ile oluşturup bir değişkene kaydederiz.
- Bu hesap objesini ekrana yazdırırız.
- Hesabın içerisinde yer alan Secret key anahtarı ile mnemonic anahtar oluştururuz.
- Mnemonic anahtarı ekrana yazdırırız
- Fonksiyon dışına çıkararak fonksiyonu dosya dışına export ederiz.

Dosyamız hazır konuma gelse de bu dosyaya index.js dosyası üzerinden bir değişkencesine erişmemiz gerekmektedir. Bu nedenle index.js içerisinde paket olarak address.js dosyasını eklememiz gerekmektedir.

```
const AdresYarat = require("./address")
```

Tablo 13

Dosyamızı “require()” fonksiyonu ile bir paketcesine eklememiz mümkün kündür. Bunu gerçekleştirmek için Tablo 13’te görüldüğü üzere “require” değişkeni içerisine dosya yolu ile ekleme sağlayabiliriz. Bu dosyayı sabit olarak “const” ile istediğimiz ad ile index.js dosyası içerisine kaydedebilir ve sonraki satırlarda çağrıabiliriz.

```
AdresYarat()
```

Tablo 14

Fonksiyonu çağrırmak için index.js içerisinde Tablo 14’de görüldüğü üzere “AdresYarat()” fonksiyonu kullanılabilir.

```
Hesap Adresi: {
  addr:
    'IL3IC7M32U775WXPJTUYODRHJ4QKSM4QQXSX37RHIAHGKNRMY3M4CS2KSU',
  sk: Uint8Array(64) [
    142, 103, 249, 121, 39, 133, 59, 126, 166, 255, 142,
    195, 140, 184, 120, 216, 79, 9, 52, 77, 111, 94,
    231, 12, 188, 11, 210, 46, 119, 40, 243, 134, 66,
    246, 129, 125, 155, 213, 63, 254, 218, 239, 76, 233,
    135, 14, 39, 79, 32, 169, 51, 144, 133, 229, 125,
    254, 39, 64, 14, 101, 54, 44, 198, 217
  ]
}
```

Mnemonic Şifre: vault tortoise rude fame jacket panel wood moment observe timber detail work barely escape kidney fury ostrich usual alarm unfold unusual skate daring above pact

Tablo 15

Kodumuzu bulunduğu klasör içerisinde “node index.js” komutu ile terminalden çalıştırıldığımızda Tablo 15’te bulunan çıktıyi vermektedir.

JavaScript ve Algorand İle Hesabın Bakiyesini Kontrol Etme

Var olan bir hesabın bakiyesini kontrol etmek aslında Algorand SDK ile mümkün olmaktadır. Ancak bu işlemi gerçekleştirmek için bir Algorand zincirinde verileri erişmemize olanak sağlayan API anahtarı gerekmektedir. Bu anahtarı <https://developer.purestake.io/> sitesinden temin edeceğiz. Site içerisinde girdikten sonra “Sign Up for Free” tuşuna basmamız gerekmektedir. Bu tuş bizi şekil 8’de gözüken <https://developer.purestake.io/signup> sitesine götürecektir.

First name *

Last name *

Email *

you@email.com

Password *

Confirm Password *

Passwords must be at least 12 characters long and contain a lowercase letter, and a number.

Terms and conditions

These Terms of Service govern the legal relationship between PureStake Inc. ("PureStake") and the customer identified on the initial Order Form ("Customer"). The parties acknowledge receipt and sufficiency of good and valuable consideration and agree as follows:

1. Definitions. Terms used in this Agreement with their initial letters capitalized have the meanings ascribed to them in this section or where they are elsewhere defined in this Agreement. Any term defined in the singular will have the corresponding definition in the plural (and vice versa). As used in this Agreement: (a) "Agreement" means these Terms of Service and all Attachments. (b) "Attachment" means all Order Forms and all other documents referred to therein, and all other written exhibits, statements of work, schedules, addenda or other similar documents executed by the parties pertaining to the Services. (c) "Confidential Information" means all non-public information disclosed by one party to the other in connection with this Agreement that the disclosing party marks as confidential or which the receiving party should reasonably know to be the confidential information of the other party. (d) "Customer Data" includes all data collected, used, processed, stored, or generated as the result of Customer's use of the Services. (e) "End User" means Customer's employees who are authorized to access and use the Services in accordance with this Agreement. (f) "Fees" are those amounts set forth in an Order Form that Customer agrees to pay to PureStake in accordance with this Agreement. (g) "Order Form" means the written documentation (whether electronic or hard copy), once accepted

I agree to the Terms and Conditions

Sign up!

Şekil 8

Gerekli bilgileri girmemiz ve "Sign up" tuşuna basmamız halinde site bizi "dashboard" adını verdiği ve birçok bilginin bulunduğu sayfaya yönlendirecektir. Bu sayfa içerisinde en üstte "Your API key" başlığı altında bulunan metin bizim API anahtarımız olacaktır. Bu anahtarları şimdilik kopyalayabiliriz.

Kodlama kısmına geçmeden önce bu API anahtarını güvenli bir şekilde kullanmak için birkaç ayar yapmamız gerekmektedir. Bu bağlamda index.js dosyamızın da bulunduğu klasör statüsüne terminal içerisinde girmemiz gerekmektedir.

```
npm install dotenv
```

Tablo 16

Klasör içerisine girdiğimizde Tablo 16'de gösterilen komut ile “dotenv” paketini eklemekteyiz. Bu paket gizli klasör olan “.env” klasörüne erişimimizi sağlayacaktır. “.env” klasörleri dosya paylaşımlarında paylaşılmayan kişisel ve gizli bilgilerin bulunduğu dosyadır. Bu dosya içerisine gizlemek istediğimiz dosyaları girebiliriz.

```
API = "EiSw8baM0m1bAZsKDi8ID3vpsRcr6arl8wVuOktH"
```

Tablo 17

Dosyanın içerisine API anahtarımızı Tablo 17'de görüldüğü üzere API anahtar kelimesi ile ekleyebiliriz.

Anahtarımızı oluşturdukta sona hesap oluşturma kodunda yaptığımiza benzer bir şekilde yeni bir JavaScript dosyası oluşturmak projenin organizasyonu ve düzeni için büyük önem arz etmektedir. “index.js” dosyasının bulunduğu klasöre sağ tıklayarak balance.js adında yeni bir JavaScript dosyası oluşturuyoruz.

```
const Algosdk = require("algosdk")
```

Tablo 18

Dosya içerisine girdiğimizde Tablo 18'de ifade edildiği üzere diğer JavaScript kodlarında yaptığımiza benzer şekilde “require()” komutu ile Algosdk sabiti içerisinde paketimizi eklemekteyiz. Bu sayede Algorand SDK'e tanımladığımız sabit ile kodumuzun içerisinde istediğimiz şekilde erişileceğiz.

```
let bakiyeKontrolEt = ()=>{  
}
```

Tablo 19

Bakiye kontrol işlemlerini gerçekleştirecek kodların yer alacağı bir fonksiyon tanımlamamız gerekmektedir. Bu tanımlamayı Tablo 19'da ifade edildiği üzere “Arrow Function” adı verilen tanımlama işlemi ile gerçekleştireceğiz. Arrow fonksiyonları yapıları ötürü bir değişkene veya sabite tanımlanmak isterler. Bu nedenle tanımlama yaparken değişken verilerin kontrolünü sağlamak amacıyla “let” anahtar kelimesi kullanılmıştır. Fonksiyonun adı tamamen kişisel tercihe bağlı olmaktadır ancak anlatım kolaylığı ve düzenlilik açısından “bakiyeKontrolEt()” isimlendirilmesi seçilmiştir. Fonksiyon parametresi olarak parantez içerisinde herhangi bir değer

girilmemiştir. Bu durumun nedeni fonksiyonun herhangi bir girdiye ihtiyaç duymamasıdır. Ancak farklı bir senaryoda kişiler fonksiyona kendi API anahtarlarını fonksiyon çağrıını sırasında besleyebilir.

```
const port=""
```

Tablo 20

Fonksiyonun içeresine girdiğimizde birkaç değişken tanımlamamız gerekmektedir. Bu değişkenlerden ilki Tablo 20'de ifade edildiği üzere “port” değeri olmaktadır. Port değeri kontrolünün gerçekleşeceği bağlantı portunu ifade etmektedir. Bu işlemin içi boş bırakılabilir veya 4001 değeri girilebilir.

```
const token={"x-api-key": process.env.API}
```

Tablo 21

Tanımlanması gereken bir sonraki değişken ise “Purestake” sitesinden elde ettiğimiz API anahtarıdır. Tablo 21'de görüldüğü üzere bu anahtarın kullanarak sorgularımızı gerçekleştireceğiz. Token değeri içeresine anahtarımızı kaydederken obje halinde kaydetmemiz gerekmektedir. Bu objenin içeresine anahtar olarak “x-api-key” string değeri girilmelidir. Bu sayede sonradan çağrıcağımız fonksiyon tam olarak hangi değerin anahtar olduğunu anlar. Tanımlamanın diğer tarafına da API anahtarımız gelmektedir. API anahtarımızı “.env” dosyası içeresine kaydettik ancak henüz herhangi bir şekilde dosyaya erişmemiştir. Erişimi gerçekleştirmek için “process.env” metodlarını çağırırız. Son noktanın devamında “.env” dosyasından hangi veriyi elde etmek istiyorsak o verinin kaydederken kullandığımız anahtarını girmemiz gerekmektedir. Biz dosya içeresinden API değerine erişmek istediğimiz için “process.env.API” komutu ile bu işlemi gerçekleştirebiliriz.

```
const TestSunucusu= "https://testnet-algorand.api.purestake.io/ps2"
```

Tablo 22

Tanımladığımız değişkenleri birleştirmeden önce son bir parametre tanımlaması yapmamız daha gerekmektedir. Bu tanımlama sorguyu gerçekleştireceğimiz testnetin adresi olmaktadır. Tablo 22'de ifade edildiği üzere bu adresi API anahtarını aldığımız “PureStake” sitesi üzerinden Algorand Testnet başlığı altında görüntülemek mümkündür.

```
let Kullanıcı = new Algosdk.Algodv2(token,TestSunucusu,port)
```

Tablo 23

Daha öncesinde tanımladığımız parametreleri “Kullanıcı” değişkeni üzerinde birleştirmektedir. İlk olarak “let” ile bir değişken tanımlamaktayız. Tablo 23’te gösterildiği üzere bu değişkene kodumuz içerisinde anlaşılırlığın artması amacıyla Kullanıcı adını verdik ancak yabancı kaynaklarda “client” olarak geçtiğini de görebilirsiniz. Devamında tanımın diğer taraflında “new” anahtar kelimesini görüyoruz. Bu anahtar kelime değişkenin yeni boş bir obje oluşturup içerisinde parametreler almasını sağlamaktadır. SDK içerisindeki “algosdk.Algodv2()” metodu ile obje oluşturma koduna erişmektedir. Bu metod içerisinde önceden oluşturduğumuz “port, token ve TestSunucusu” değişkenlerini almaktadır.

```
let Hesap = "IL3IC7M32U775WXPJTUYODRHJ4QKSM4QQXSX37RHIAHGKNRMY3M4CS2KSU";
```

Tablo 24

Kullanıcı oluşturulduktan sonra incelenmesini istediğimiz hesabın bilgilerinin girilmesi gerekmektedir. Tablo 24’te görüldüğü üzere herhangi bir adres “let” anahtar kelimesi ile tanımlanan değişkenin içerisinde kaydedilir. İsteğe göre bu değer fonksiyon girdisi olarak da alınabilir. Bu sayede fazlaca adresi “for” döngüleri ile kısa değişiklikler ile taramak mümkün olabilir.

```
(async()=>{  
})
```

Tablo 25

Tanımlamalar bittiğinden sonra fonksiyon içinde asıl sorgu işlemini yapacağımız kod yapısı yerini almıştır. Ancak bu işlem bir miktar zaman gerektirmektedir. Bu kısımda JavaScript’ın en büyük nimetlerinden olan asenkron kodlama yapısı kurtarıcı rol oynamaktadır. Asenkron işlemler yapısı bakımından fonksiyon içerisinde yer almıştır. Bu nedenle Tablo 25’te gösterildiği üzere boş ve anında çalışan bir “Arrow Function” yapısı oluşturulabilir. Bu fonksiyonun tanımlanmasında asenkron yapısını sağlayan anahtar kelime “async” olmaktadır. JavaScript’e bu satırda fonksiyonun asenkron çalışacağını bildirmiştir.

```
let hesapBilgisi = await kullanıcı.accountInformation(Hesap).do()
console.log("Bakiye: ", JSON.stringify(hesapBilgisi));
```

Tablo 26

Asenkron fonksiyonun içerisinde girildikten sonra Tablo 26'da gösterildiği gibi “let” ile tanımladığımız “hesapBilgisi” değişkeni içerisinde hesap bilgisi kaydedilir. Tanımlanmanın karşı tarafında asenkron yapıda satırda beklenmesini ifade eden “await” anahtar kelimesi bu satırın zaman alacağını JavaScript üzerinde ifade etmektedir. Sonrasında oluşturduğumuz kullanıcı üzerindeki “accountInformation()” metodunu çağırırız. Bu metod içerisinde bakiyesini sorgulamak istediğimiz Hesap değişkenini almaktadır. Son olarak “.do()” ile işlemi başlatırız ve console.log() ile bakiyeyi ekrana yazdırırız. Ekrana yazdırduğumuz bilgi verisi JSON formatında olmaktadır. Bu veriyi daha okunur bir şekilde görüntüleyebilmek için “JSON.stringify” işlemini gerçekleştiririz.

```
() .catch((err)=>{
  console.log(err);
})
```

Tablo 27

Tablo 27'de ifade edildiği üzere “async foksiyonunun” bitiş parantezinin yanına olası hataları yakalayarak ekrana yazdırın bir kod bloğu eklenebilir. Bu mekanizma “.catch” anahtar kelimesi ile çalışıyor olmaktadır.

```
module.exports = bakiyeKontrolEt
```

Tablo 28

Son olarak JavaScript kodunu farklı dosyalarda kullanmak için fonksiyonumuzu dışarı “export” eden bir yapı gerekmektedir. Tablo 28'de gösterildiği üzere “module.exports” işlemi ile istediğimiz fonksiyonu kod dosyası dışarısına çıkarabilmekteyiz.

```

const Algosdk = require("algosdk")
let bakiyeKontrolEt = ()=>{
  const port = ""
  const token = {
    "x-api-key": process.env.API
  };
  const TestSunucusu = "https://testnet-algorand.api.purestake.io/ps2"

  const kullanici = new Algosdk.Algodv2(token, TestSunucusu, port)
  let Hesap = "IL3IC7M32U775WXPJTUYODRHJ4QKSM4QQXSX37RHIHGKNRMY3M4CS2KSU";
  (async ()=>{
    let hesapBilgisi = await kullanici.accountInformation(Hesap).do()
    console.log("Bakiye: " + JSON.stringify(hesapBilgisi));
  })().catch((err)=>{
    console.log(err);
  })
}
module.exports = bakiyeKontrolEt

```

Tablo 29 *balance.js* Tamamı

Tablo 29'da kod bloğunun tamamı ifade edilmiştir. Yazdığımız sırasıyla neler yaptığına bakarsak:

- İlk olarak SDK ile bağlantımızı sağlayan algosdk paketini değişken ile dosya içerisinde erişilebilir yaparız.
- İşlemlerin yer alacağı fonksiyon tanımlamasını “Arrow Function” yapısı ile oluştururuz.
- İçi boş bir port değişkeni tanımlarız.
- API anahtarımızı “.env” dosyasından kodumuz içerisinde getiren bir değişken tanımlanır.
- Hangi zincirde arama yapacağımız ile ilgili bir bağlantı değişkeni kaydederiz.
- Tanımladığımız değişkenleri SDK üzerindeki Algov2 fonksiyonunda doğru sıralama ile birleştirir ve kullanıcı objesini oluştururuz.
- Tarama yapacağımız adresi değişkene kaydederiz.
- Asenkron bir fonksiyon tanımlarız.
- Fonksiyon içerisinde kullanıcı bilgileri ile tarama yapmayı sağlayan metodu await yapısı ile çağırırız.
- Değişkene kaydettiğimiz “hesapBilgisi” değişkenini string formatına çevirerek ekrana yazdırırız.
- Hataları yakalayıp ekrana yazdırması için “.catch()” fonksiyonunu çağırırız.
- Dosya çıktısı olarak “bakiyeKontrolEt()” fonksiyonunu dosya dışına export ederiz.

Dosyamızı *balance.js* adı altında yazdıktan sonra bu dosyaya *index.js* içerisinde erişmemiz ve gerekli paketler ile birleştirmemiz gerekmektedir.

```
const bakiyeSorgula = require("./balance")
```

Tablo 30

Hesap oluştururken yaptığımız dosya ekleme işlevinin aynısını balance.js dosyası için de gerçekleştirmekteyiz. Tablo 30'da görüldüğü üzere istediğimiz ismi verdigimiz değişkene “require()” işlevi ile balance.js dosyasını adeta bir paketmiş gibi dosyamıza eklememiz mümkündür.

```
require("dotenv").config()
```

Tablo 31

Oluşturduğumuz balance.js dosyası içerisinde “.env” dosyasına erişim sağlasak da bu işlemi gerçekleştirecek paketi ister balance.js dosyası içerisinde ister index.js dosyası içerisinde Tablo 31'de gösterildiği üzere ekleyebiliriz.

```
bakiyeSorgula()
```

Tablo 32

Gerekli kütüphaneleri ekledikten sonra dosyamızı kaydettiğimiz değişken adı ile fonksiyonu Tablo 32'de ifade edildiği üzere index.js dosyasında çalıştırabiliriz.

```
Bakiye: {"address":"IL3IC7M32U775WXPJTUYODRHJ4QKSM4QQXSX37RHIAHGKNRMY3M4CS2KSU","amount":0,"amount-without-pending-rewards":0,"apps-local-state":[],"apps-total-schema":{"num-byte-slice":0,"num-uint":0},"assets":[],"created-apps":[],"created-assets":[],"min-balance":100000,"pending-rewards":0,"reward-base":27521,"rewards":0,"round":24131895,"status":"Offline","total-apps-opted-in":0,"total-assets-opted-in":0,"total-created-apps":0,"total-created-assets":0}
```

Tablo 33

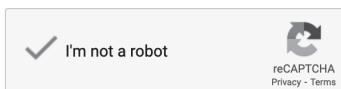
Dosyaların bulunduğu klasör içerisinde terminalde “node index.js” kodunu çalıştırduğumızda karşımıza Tablo 33'te ifade edilen çıktı çıkmaktadır. Bu çıktıının içerisinde birçok önemli değer bulunmaktadır. Bu değerlerin bazlarına daha ayrıntılı bakılması gereklir:

- **Address:** Sorgunun gerçekleştiği adresi ifade etmektedir.
- **Amount:** Sorgulanın hesabın içerisindeki bakiyeyi miktar olarak ifade etmektedir.
- **Amount-without-pending-rewards:** Beklemede olan ödüller hesaba geçirilmeden önceki bakiyeyi ifade etmektedir.

- **Apps-local-state:** Adres üzerinde local çalışan uygulamalar olup olmadığını kontrol etmektedir. Layer 2 mimarisinde büyük önem arz etmektedir.
- **Apps-total-schema:** Adres üzerinde tüm zaman birimlerinde çalışmış olan uygulamaların şemasını ifade etmektedir.
- **Assets:** Adres üzerinde yer alan varlıklarını ifade etmektedir. Bu varlıklar NFT ve benzeri varlıklar olabilir.
- **Created-apps:** Sorgulanan hesap üzerinde yaratılan uygulamaların bilgilerini göstermektedir.
- **Created-assets:** Sorgulanan hesap üzerinde yaratılmış varlıkların bilgilerini ifade etmektedir.
- **Min-balance:** Hesabın aktif olarak kalması için gerekli olan minimum bakiyeyi göstermektedir. Minialgo üzerinden hesaplanır.
- **Pending-rewards:** Beklemede olan ödüllerin miktarını ifade etmektedir.
- **Round:** Bakiyenin hangi birimde yuvarlandığını ifade etmektedir.
- **Status:** Hesabın zincirde aktif veya de aktif olma durumunu ifade etmektedir.

Hesabımızı kontrol ettiğimizde içerisinde herhangi bir miktar olmadığını görüntüledik. Ancak işlemlerimizde kullanmayı istediğimiz testnet coinlerine ihtiyaç duymaktayız. Test coinlerini almak için <https://bank.testnet.algorand.network/> sitesini ziyaret etmemiz gerekmektedir.

Algorand dispenser



The dispensed Algos have no monetary value and should only be used to test applications.

This service is graciously provided to enable development on the Algorand blockchain test networks.

Please do not abuse it by requesting more Algos than needed.

IL3IC7M32U775WXPJTYODRHJ4QKSM4QQXSX37RHIAHGKNRMY3M4CS2KSU	Dispense
---	----------

Status: Code 200 success: "4VN55JGRDJUDVWWSJYQZXSWGUHCST7JLSMMBZ56XQDNCLNZB4JHA"

Sekil 9

Sekil 9'da ifade edildiği üzere gerekli bilgileri sağlayarak istediğimiz adrese test coinlerini göndermek mümkün olmaktadır.

```
Bakiye: {"address":"IL3IC7M32U775WXPJTUYODRHJ4QKSM4QQXSX37RH
IAHGKNRMY3M4CS2KSU","amount":10000000,"amount-without-pending-
rewards":10000000,"apps-local-state":[],"apps-total-schema":{"num-byte-
slice":0,"num-uint":0},"assets":[],"created-apps":[],"created-assets":[],"min-
balance":100000,"pending-rewards":0,"reward-base":27521,"rewards":0,"round":2413
2247,"status":"Offline","total-apps-opted-in":0,"total-assets-opted-in":0,"total-created-
apps":0,"total-created-assets":0}
```

Tablo 34

Tablo 34'te ifade edildiği üzere “dispense” işlemi sonrasında bir kez daha dosyamızı çalıştırduğumızda artık adresimizin içerisinde 10000000 Test Coini bulunduğuunu görebiliriz. Her bir “dispence” işlemi hesap içerişine 10000000 test coinini eklemektedir. Bu coinlerin hiçbir değeri yoktur ve tamamen test amacıyla kullanılmaktadır.

JavaScript ve Algorand İle Transfer İşlemi Gerçekleştirme

Coin veya varlık transferleri iki hesap arasında gerçekleştirmektedir. Bu hesapların adresleri ve gönderici hesabın anahtarı bilinirse işlem herhangi bir sıkıntı olmadan gerçekleştirilebilir. Kullanılacak iki hesabın gerekliliklerinden ötürü ilk olarak iki hesap oluşturulması gerekmektedir. Bu işlemi index.js dosyamız içerisinde “AdresYarat()” adı ile tanımladığımız address.js fonksiyonunu çağırarak gerçekleştirebiliriz.

```
AdresYarat()
AdresYarat()
```

Tablo 35 index.js

Tablo 35'de görüldüğü üzere iki adet adres yaratmak için fonksiyonu iki kez alt alta çağırabiliz. İsteğe bağlı olarak tek fonksiyon çağrımları yapılmış iki kez kod baştan sona çalıştırılabilir.

```

Hesap Adresi: {
  addr:
  'YV5JTE3FJLCG7JVFFWZWH7TJ7BP43RHEXHUY4RUBUX3J342FVOIUCHFGY',
  sk: Uint8Array(64) [
    73, 210, 193, 126, 210, 247, 243, 185, 29, 63, 146,
    220, 184, 208, 166, 165, 97, 179, 152, 195, 51, 14,
    91, 100, 214, 228, 239, 239, 211, 56, 112, 42, 197,
    122, 153, 147, 101, 74, 196, 111, 166, 165, 45, 179,
    108, 31, 243, 79, 194, 254, 110, 39, 37, 207, 76,
    114, 52, 13, 47, 180, 239, 154, 45, 92
  ]
}

Mnemonic Şifre: empty also disorder large tree hover wedding mutual breeze drill
square bottom grocery tower own athlete merge flower venue year pond athlete fatal
ability anxiety

Hesap Adresi: {
  addr:
  'BJFNBQ7MQ6BU7PZAEEAET4C6BIOJF1YYSSBKQWI763YLPKYZXKAB5IK5U',
  sk: Uint8Array(64) [
    102, 2, 203, 55, 191, 81, 93, 114, 62, 116, 30,
    157, 170, 69, 223, 94, 136, 38, 195, 210, 176, 226,
    76, 18, 56, 131, 169, 193, 233, 48, 167, 143, 10,
    74, 208, 195, 236, 135, 131, 79, 191, 32, 33, 0,
    66, 79, 130, 240, 80, 228, 149, 24, 196, 164, 21,
    66, 200, 255, 183, 133, 189, 88, 205, 212
  ]
}

Mnemonic Şifre: error clown orbit brief rival orient peanut bunker father birth hurry
magnet escape radio make tired end theory screen box squeeze slow large above fruit

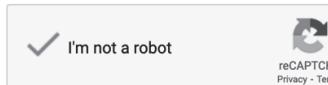
```

Tablo 36

Kod çıktısı Tablo 36'da verildiği üzere birçok veriye sahiptir. Bu veriler içerisinde Mnemonic Şifreler ve Adresler aktif olarak kullanılacaktır.

Hesapları oluşturma süreci tamamlanmış olsa da bu hesaplar arasında transfer yapılacak bakiyelere ihtiyaç olmaktadır. Bu durumu <https://bank.testnet.algorand.network/> sitesi ile gerçekleştirebiliriz. Simdilik sadece transferi gönderecek hesaba bakiye yüklemesi yapmak yeterlidir. İlk işlem için B harfi ile başlayan “BJFNBQ7MQ6BU7PZAEE...B5IK5U” adresi gönderici konumunda olacaktır. Y harfi ile başlayan “YV5JTE3FJLCG7...X3J342FVOIUCHFGY” adresi ise alıcı konumunda olacaktır. Kolay anlatım amacıyla bu adreslere baş harflerine karşılık gelecek şekilde Baran ve Yenal adı verilebilir.

Algorand dispenser



The dispensed Algos have no monetary value and should only be used to test applications.

This service is gracefully provided to enable development on the Algorand blockchain test networks.

Please do not abuse it by requesting more Algos than needed.

`BJFNBQ7MQ6BU7PZAEEAET4C6BIOJF1YYSSBKQWI763YLPKYZXKAB5IK5U` Dispense
Status: Code 200 success: "KMDHGKQSB4ZLABXR4JGQGDV37UII75VA3MVZT3F3Q7HBFQDSFQAA"

Şekil 10

Baran Hesabına Şekil 10'da ifade edildiği üzere Algorand dispenser üzerinden bakiye ekleme işlemini gerçekleştirdik. Bu işlemin gerçekleşip gerçekleşmediğini gözlemelemek amacıyla bakiye sorgulama kodumuzu çalıtırabiliriz. Ancak gözlemeleme kolaylığı açısından bu kodda birkaç düzenlemeye yapabiliriz.

```
Bakiye: {"address":"IL3IC7M32U775WXJPJUYODRHJ4QKSM4QQXSX37RH
IAHGKNRMY3M4CS2KSU","amount":10000000,"amount-without-pending-
rewards":10000000,"apps-local-state":[],"apps-total-schema":{ "num-byte-
slice":0,"num-uint":0}, "assets":[], "created-apps":[], "created-assets":[], "min-
balance":100000, "pending-rewards":0, "reward-base":27521, "rewards":0, "round":2413
2247, "status": "Offline", "total-apps-opted-in":0, "total-assets-opted-in":0, "total-created-
apps":0, "total-created-assets":0}
```

Tablo 37 balance.js Örnek Çıktı

Şu anda balance.js dosyasını çalıştırduğumızda Tablo 37'de görüldüğü üzere sorgulamak istediği adresin birçok verisini obje formatında ekrana yazdırmaktadır. Ancak bu veriler içerisinde sadece "amount" anahtar kelimesine sahip bakiyesini görmek istemekteyiz.

```
console.log("Bakiye: " + JSON.stringify(hesapBilgisi.amount));
```

Tablo 38 balance.js

Hazırladığımız balance.js dosyası üzerinde ekrana yazdırma komutunda "hesapBilgisi" objesinin içeresinden nokta operatörü ile istediğimiz anahtar kelimeyi alabiliriz. Bu durumda biz sadece amount parametresini almak istiyoruz. Bu nedenle Tablo 38'de ifade edildiği üzere "hesapBilgisi.amount()" ile istediğimiz veriye erişebiliriz.

Tamamladığımız balance.js dosyası üzerinde yapmamızın faydalı olacağı son değişiklik ise parametresiz bir fonksiyon olan bu değişkeni içerisinde “hesapAdresi” parametresini isteyen hale getirmek olacaktır.

```
let bakiyeKontrolEt = (hesapAdresi)=>{
  ...
  let Hesap = hesapAdresi;
  ...
}
```

Tablo 39

Tablo 39'da görüldüğü üzere gerekli modifikasyonları yapmak için balance.js içerisinde tanımladığımız fonksiyon olan “bakiyeKontrolEt()” fonksiyonunun parametrelerinin bulunduğu parametre alanına istediğimiz adı verdigimiz değişken gelir. Bu ad görsel açıdan kolaylık sağlamaşı amacıyla “hesapAdresi” olarak belirlenmiştir. Devamında Hesap değişkeni tanımlamasında karşılık gelecek şekilde “hesapAdresi” değişkeni atanmıştır.

```
bakiyeSorgula("IL3IC7M32U775WXPJTUYODRHJ4QKSM4QQXSX37RHIAHGKNRMY3M4CS2KSU")
```

Tablo 40 index.js

Yaptığımız değişiklikler ile index.js içerisinde Tablo 40'da gösterildiği üzere fonksiyonun içerisinde hesap adresi girilerek fonksiyon çağrılabiliriz.

```
bakiyeSorgula("BJFNBQ7MQ6BU7PZAEEAEET4C6BIOJF1YSSBKQWI763YLPKYZXKAB5IK5U")
bakiyeSorgula("YV5JTE3FJLCG7JVFFWZWYH7TJ7BP43RHEXHUY4RUBUX3J342FVOIUCHFGY")
```

Tablo 41 index.js

Tablo 41'de gösterildiği üzere Baran ve Yenal hesaplarının bakiyelerine kolaylıkla erişim sağlanır.

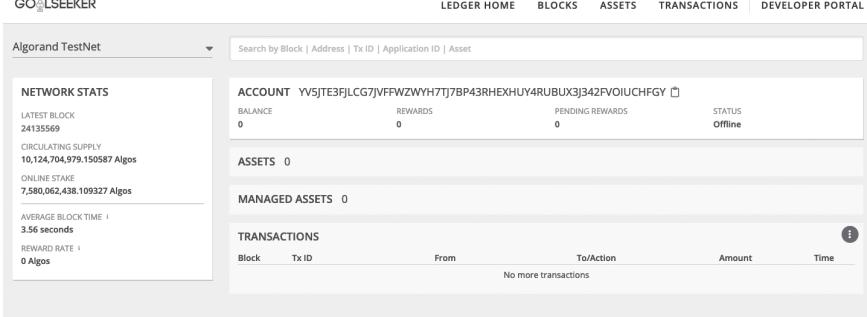
```
Bakiye: 10000000
Bakiye: 0
```

Tablo 42 index.js Çıktı

Dosayı bulunduğu klasördeki terminalden “node index.js” komutu ile çalıştığımızda karşımıza Tablo 42'de bulunan çıktı gelmektedir. Baran adresine yükleme yapıp Yenal adresine yükleme yapmadığımız için bu sonuç beklenen bir sonuçtur.

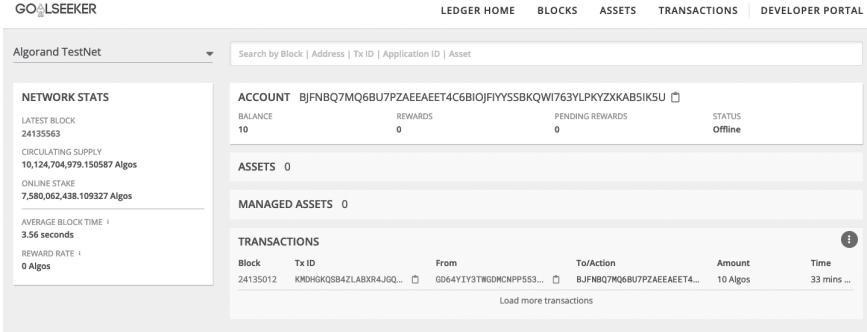
Ayrıca bakiyelere sadece kodlama içerisinde erişilmemektedir. Explo-

rer adı verilen siteler üzerinden de hesap bakiyeler kontrol edilebilmekte- dir. Bu sitelerden biri olan <https://goalseeker.purestake.io/algorand/testnet/> sitesini kullanacağımız.



The screenshot shows the Goalseeker TestNet interface. On the left, there's a sidebar with 'NETWORK STATS' showing the latest block (24135569), circulating supply (10,124,704,979.150587 Algos), and online stake (7,580,062,438 109327 Algos). Below that are average block time (3.56 seconds) and reward rate (0 Algos). The main area has tabs for 'LEDGER HOME', 'BLOCKS', 'ASSETS', 'TRANSACTIONS', and 'DEVELOPER PORTAL'. The 'TRANSACTIONS' tab is selected. It shows a single transaction for Yenal's account (YV5JTE3FJLCG7JVFFWYH7TJ7BP43RHEXUY4RUBUX3J342FVO1UCHFGY) with a balance of 0, rewards of 0, and pending rewards of 0. The status is 'Offline'. Below this, it shows 'ASSETS 0' and 'MANAGED ASSETS 0'. The 'TRANSACTIONS' table is empty with a message 'No more transactions'.

Şekil 11 Yenal Hesabı



The screenshot shows the Goalseeker TestNet interface. On the left, there's a sidebar with 'NETWORK STATS' showing the latest block (24135563), circulating supply (10,124,704,979.150587 Algos), and online stake (7,580,062,438 109327 Algos). Below that are average block time (3.56 seconds) and reward rate (0 Algos). The main area has tabs for 'LEDGER HOME', 'BLOCKS', 'ASSETS', 'TRANSACTIONS', and 'DEVELOPER PORTAL'. The 'TRANSACTIONS' tab is selected. It shows a transaction for Baran's account (BJFNBQ7MQ6BU7PZAEAEET4C6BIOJFYYSBKQWI763YLPKYZXKAB5IKU) with a balance of 10, rewards of 0, and pending rewards of 0. The status is 'Offline'. Below this, it shows 'ASSETS 0' and 'MANAGED ASSETS 0'. The 'TRANSACTIONS' table shows one transaction from block 24135012 (KMDHKGQS84ZLABXR4JQ...) to block 24135563 (GD64YIY3TNGDMCNPP553...) with an amount of 10 Algos and a time of 33 mins ... A 'Load more transactions' button is visible.

Şekil 12 Baran Hesabı

Testnet içerisinde olduğumuza emin olduktan sonra taramamızı ger- çekleştirdiğimizde Şekil 12'de yer alan Baran hesabının içerisinde 10 Algo olduğunu görüntülemektedir. Yenal hesabını Şekil 11'de olduğu şekilde kontrol ettiğimizde isteğimiz doğrultusunda boş olduğunu gözleme- teyiz.

Kodlama kısmına daha önce kullandığımız fonksiyonları birbirleri ile birleştirmemiz gerekmektedir. Yine aynı şekilde işlem kolaylığı olması açısından “index.js” dosyasının bulunduğu klasöre gidilerek “transection.js” dosyası oluşturulur. Transfer işlemleri bu dosya içerisinde yer alacaktır.

```
const Algosdk = require("algosdk")
require("dotenv").config()
```

Tablo 43

Başlangıçta her kod dosyamıza başladığımız gibi Tablo 43'de ifade edildiği üzere yani paket ekleme ile başlarız. JavaScript içerisindeki "require()" fonksiyonunu kullanarak çağrıdığımız paketi istediğimiz adı vereceğimiz sabite kaydederiz. Şu anlık bu sabitin adı Algosdk olarak belirlenmiştir. Devamında ".env" klasörlere erişimimizi sağlayan dotenv paketinin eklenmesi gerekmektedir.

```
const port=""
const token={"x-api-key": process.env.API};
const TestSunucusu= "https://testnet-algorand.api.purestake.io/ps2"
```

Tablo 44

Paket eklemesini gerçekleştirdikten sonra Tablo 44'te ifade edildiği üzere zincir üzerinde işlem yapabilmemizi sağlayan değerlerin girilmesi gerekmektedir. Bu değerler sırasıyla:

- **Port:** İşlemin gerçekleşeceği bağlantı portunu ifade etmektedir. Kullanım amacımızdan dolayı boş olarak bırakılabilir.
- **Token:** API anahtarımızı ifade etmektedir. "Purestake" sitesinden aldığımız API anahtarını güvenlik nedenleriyle .env klasörü içerisinde "process.env" metodu ile erişilmektedir.
- **TestSunucusu:** İşlemin gerçekleşmesini istediğimiz ağı ifade etmektedir. Bu işlemin test sunucusunda gerçekleşmesini istediğimizden ötürü "Purestake" sitesinde bulabileceğimiz zincir adresini yazmak tayız.

```
const kullanıcı = new Algosdk.Algodv2(token,TestSunucusu,port);
```

Tablo 45

İşlemleri yapacağımız özellikleri tanımladıktan sonra bu özellikleri birleştiren bir fonksiyon gerekmektedir. Tablo 45'te ifade edildiği üzere istediğimiz ismi kullanarak tanımladığımız değişkene "new" metodu ile boş bir obje oluşturup içerisine Algosdk kütüphanesinde bulunan Algodv2 metodu ile obje özellikleri ekleriz. Bu özellikler parantez içerisinde ifade edildiği üzere API anahtarı, sunucu adresi ve port bilgisi olmaktadır. Bu parametrelerin sıralaması son derece önemli olmaktadır.

```
( async () => {
})
```

Tablo 46

Gerekli parametreleri birleştirmemizden sonra işlemlere başlamak için asenkron bir yapı gerekmektedir. Bu yapıyı oluşturmak için boş bir “async” fonksiyonu oluşturulabilir. Tablo 46’da ifade edildiği şekilde tanımlanan asenkron “arrow function” yapısı içerisine herhangi bir değişken almaz ve çağrılamayı beklemeden kod satırı ulaşır ulaşmaz çalışır.

```
let parametreler = await kullanıcı.getTransactionParams().do();
```

Tablo 47

Asenkron fonksiyonun içerisine girdiğimizde ilk olarak elde etmemiz gereken parametrelerin tanımlandığı bir değişken oluşturulması gerekmektedir. Bu değişkeni istediğimiz ad ile tanımlayabiliriz. Tablo 47’de görüldüğü üzere tanımın karşı tarafında “await” ile başlandığı görüntülenmektedir. Bu durumun nedeni sonrasında gelecek fonksiyonun uzun zaman alması ihtimaline karşılık kodun beklemeye alınmasıdır. Alacağımız parametrelerin kullanıcı olarak tanımladığımız obje içerisinde olması nedeniyle obje içerisinde nokta operatörü ile “getTransactionParams()” metodu çağrılır. Bu metodun çalışması amacıyla satır sonuna .do(); metodу eklenir.

```
let mnemonic = "error clown orbit brief rival orient peanut bunker father birth hurry
magnet escape radio make tired end theory screen box squeeze slow large above fruit" ;
let çevrilenhesap = Algosdk.mnemonicToSecretKey (mnemonic) ;
```

Tablo 48

Parametreleri aldıktan sonra veya almadan önce gönderim yapacağımız hesabin erişimine sahip olmamız için birkaç tanımlama yapmamız gerekmektedir. Bu işlem için Tablo 48’de ifade edildiği üzere mnemonic adını verdigimiz değişkene gönderim işlemini yapacak olan hesabin anahtarını girmemiz gerekmektedir. Gönderme işlemini Baran yapacağı için, yani parannın çıkış yapacağı hesap Baran’ın hesabı olacağı için bu değişkene Baran hesabının anahtarı kaydedilir. Mnemonic olarak kaydedilen değişken insanlar tarafından okunabilir formattadır ancak makine tarafından anlamsız olmaktadır. Bu nedenle bu anahtarı Mnemonic değerden Secret Key değerine çevirmemiz gerekmektedir. Bu işlemi yaparken Algosdk paketi içerisinde bulunan “mnemonicToSecretKey” metodunu kullanmamız gerekmektedir. Bu değişken içerisine parametre olarak mnemonic değerini almaktadır ve “çevrilenhesap” adı altında kodumuz içerisinde kullanılmak üzere kaydedilebilir.

```
let miktar = 1000000;
const enc = new TextEncoder();
```

Tablo 49

Transfer işlemi içerisinde bazı değişkenleri tanımlamamız gerekmektedir. Bu değişkenleri tanımlamadan da kodumuz sıkıntısız şekilde çalışır ancak Tablo 49'da görüldüğü üzere anlatım kolaylığı açısından kodumuzda bu şekilde yer alacaktır. İlk tanımladığımız değişken miktar olmaktadır. Yapacağımız transfer bu miktar kadar olacaktır. Büyük bir sayı gibi görünüşde bu durum ondalık sistemden kaynaklanmaktadır. 1000000 birim 1 Algo'ya denk gelmektedir. Sonrasında tanımladığımız “enc” değişkeni yazarcağımız “string” değerini zincire yazılabilir hale getirmektedir. Bu işlemi gerçekleştirirken “new” işlemi ile bir obje oluşturmaktadır.

```
let transferislemi = {
  "from": "çevrilenesap.addr",
  "to" : "YV5JTE3FJLCG7JVFFWZWYH7TJ7BP43RHEXHUY4RUBUX3J342FVOIUCHFGY",
  "fee" : 1,
  "amount" : miktar,
  "firstRound" : parametreler.firstRound,
  "lastRound" : parametreler.lastRound,
  "genesisID" : parametreler.genesisID,
  "genesisHash" : parametreler.genesisHash,
  "note" : enc.encode("Merhabalar"),
}
```

Tablo 50

Transferi gerçekleştirmek için bu transferin özelliklerini gösteren bir değişken tanımlanmaktadır. Bu değişkene istenilen isim verilebilir. Şimdilik anlam karışıklığı yaratmaması amacıyla Tablo 50'de görüldüğü üzere “transferişlemi” isimlendirmesi verilmiştir. Aldığı parametrelere teker teker bakılması gerekirse:

- **From:** Hangi adresten miktarın gönderileceğini belirtmektedir. Daha öncesinde “Secret-Key” ile hesaba dönüştürüduğumuz “çevrilenesap” değişkeninin “addr” anahtarına erişerek işlemi tamamlar. Baran hesabının adresi olmaktadır.
- **To:** Miktarın hangi adrese gönderileceğini belirtmektedir. 64 karakterden oluşan hesap adresini almaktadır. Bu adres dışında miktarın gönderileceği hesabın anahtarı ve benzeri herhangi veriye ihtiyaç duyulmaz. Yenil hesabının adresi olmaktadır.
- **Fee:** İşlem ücretini ifade etmektedir. Yapılacak transfer sonucunda ne kadar bir işlem ücreti kesileceğini belirtmektedir. Sıfır değerine

sahip olmamak koşulu ile herhangi değere sahip olabilir. Testnet dısına çıķıldığında farklı değerler alabilir. Integer girdi istemektedir.

- **Amount:** Gönderilecek olan miktarı ifade etmektedir. Bu değeri miktar olarak üst satırlarda tanımladığımız için karşılığında amount değerini yazabiliz. İçerisine integer değeri almaktadır.
- **firstRound:** İşlemin geçerli olduğu ilk tur olmaktadır. İşlem bu turdan önce gönderilirse ağ tarafından reddedilecektir. Parametreler objesi içerisinde “firstRound” anahtarı ile erişilip anahtara eklenebilir.
- **lastRound:** İşlemin geçerli olduğu bitiş tur olmaktadır. Bu turdan sonra işlem ağ tarafından reddedilecektir. Parametreler objesi içerisinde “lastRound” anahtarı ile erişilip transfer objesi içerisinde ilgili anahtara kayıt edilebilir.
- **GenesisID:** İşlemin gerçekleşeceği ağı insanlar tarafından okunabilir şekilde ifade eden değişken olmaktadır. Parametreler içerisinde “genesisID” anahtarı ile erişilip transfer içerisinde ilgili anahtara kaydedilebilir.
- **GenesisHash:** İşlemin gerçekleşeceği ağıın makine tarafından okunabilir hash değerine karşılık gelmektedir. Parametreler içerisinde “genesisHash” değeri ile erişilip transfer objesi içerisinde ilgili anahtara kaydedilebilir.
- **Note:** İşlemi gerçekleştirirken eklemek istediğimiz notu ifade etmektedir. Uzun olmamak koşuluyla istediğimiz notu bu anahtar içersine ekleyebiliriz. Eklemei gerçekleştirirken notumuzu uygun şifreleme metodu ile düzenlememiz gerekmektedir. Bu nedenle daha önce tanımlamasını gerçekleştirdiğimiz “enc” değişkeni üzerinden “encode” metodunu çağırabilir, içersine parametre olarak istediğimiz notu girebiliriz.

```
let imzalanmışTransfer = Algosdk.signTransaction(transaction, çevrilenesap.sk);
```

Tablo 51

Transfer objesini oluşturduktan sonra bu objeyi gönderici kişinin imzalaması gerekmektedir. Tablo 51'de görüldüğü üzere istediğimiz isim ile değişkenimizi kaydederiz. Algorand SDK içerisinde eriştiğimiz “signTransaction()” metodu ile bu işlem imzalanır. Bu metot içersine iki adet parametre almaktadır. Bu parametreler:

- **Transferİşlemi:** Transferin değişkenlerinin girildiği objeyi ifade etmektedir.

- **Çevrilenhesap.sk:** Gönderimin yapılacak adresin Secret Key değerini yani gizli anahtarını ifade etmektedir. Bu anahtar ile işlemi imzalarız ve mümkün kılارız.

```
let gönderilenTransfer = await kullanıcı.sendRawTransaction(imzalanmışTransfer.blob).do();
console.log("Transfer : ", gönderilenTransfer.txId)
```

Tablo 52

İmzalanan transferi gerçekleştirmek için zincire göndermemiz gerekmektedir. Bu işlemi gerçekleştirmek amacıyla Tablo 52'de görüldüğü üzere istediğimiz isim ile kaydettiğimiz değişkenimizi asenkron yapı kazandıran “await” anahtarı ile tanımlarız. “Await” fonksiyonda bu kısımda durulup işlemin beklenmesi gerektiğini söylemektedir. Sonrasında kullanıcı objesi üzerindeki “sendRawTransaction()” metoduna erişilerek gönderilme hedeflenir. Bu metot içerisinde “imzalananTransfer” değişkeninin alır. Ancak bu değişken yalnız hali ile parametre olamaz. Bu değerin “blob” adını verdığımız metoda sokulması gerekmektedir. Blob metodu değerin herhangi bir program tarafından okunabilen “binary” haline getirilmesini sağlamaktaadır. Bu sayede zincir içerisinde kolaylıkla işleme alınabilir. Son olarak da metodu çalıştmak için “.do()” ibaresi gerekmektedir. Bu sayede metot çalışır ve “gönderilenTransfer” değişkeni içerisinde kaydedilir. Son olarak bu değişken “console.log()” fonksiyonu ile ekrana yazdırılır.

```
O.catch(err =>{
  console.log(err);
})
```

Tablo 53

Asenkron fonksiyonunun tam çıkış parantezine gelindiğinde Tablo 53'de verildiği üzere bir hata yakalama fonksiyonu eklenebiliniz. Bu sayede fonksiyonda oluşabilecek hatalar ekrana yazdırılır. Bu durumu sağlamak için “O.catch” Arrow Fonksiyonu ile yakaladığımız hatalar “console.log()” fonksiyonu ile ekrana yazdırılır.

```

const Algosdk = require("algosdk")
require("dotenv").config()
const port = ""
const token = {"x-api-key": process.env.API};
const TestSunucusu = "https://testnet.algorand.api.purestake.io/ps2"

const kullanıcı = new Algosdk.Algodv2(token, TestSunucusu, port);

(async () => {
  let parametreler = await kullanıcı.getTransactionParams().do();

  let mnemonic = "error clown orbit brief rival orient peanut bunker father birth hurry magnet escape radio make
tired end theory screen box squeeze slow large above fruit";
  let çevrilenhesap = Algosdk.mnemonicToSecretKey(mnemonic);

  let miktar = 1000000;
  const enc = new TextEncoder();
  let transferİşlemi = {
    "from": çevrilenhesap.addr,
    "to": "BJFNBQ7MQ6BU7PZAEEAET4C6BIOJF1YYSSBKQWI763YLPKYZXKAB5IK5U",
    "fee": 1,
    "amount": miktar,
    "firstRound": parametreler.firstRound,
    "lastRound": parametreler.lastRound,
    "genesisID": parametreler.genesisID,
    "genesisHash": parametreler.genesisHash,
    "note": enc.encode("Merhabalar"),
  }

  let imzalanmışTransfer = Algosdk.signTransaction(transferİşlemi, çevrilenhesap.sk);
  let gönderilenTransfer = await kullanıcı.sendRawTransaction(imzalanmışTransfer.blob).do();

  console.log("Transfer : ", gönderilenTransfer.txId)
})().catch(err => {
  console.log(err);
})

```

Tablo 54 transction.js Tamamı

Kod bloğumuz tamamlanmıştır ve Tablo 54'de ifade edilmiştir. Kodun işlevinden adım adım bahsedilmesi gereklidir:

- İlk olarak gerekli paketler require fonksiyonu ile istenilen değişkenlere eklenir ve kod içerisinde kullanıma hazır hale gelir.
- Sonrasında kullanıcıyı oluşturacak port, token ve sunucu adresi gibi değişkenler ilgili yerden farklı değişkenlere kaydedilir.
- Kaydedilen değişkenlerden SDK kullanılarak bir kullanıcı objesi oluşturulur.
- Asenkron bir fonksiyon tanımlanır.
- Fonksiyon içerisinde işlemi gerçekleştirecek parametreler kullanıcı objesinden türetilir.
- Gönderici hesabın yani Baran hesabının mnemonic anahtarının tanımlanması yapılır ve bu anahtardan hesabın kendisine erişilir.
- Transfer edilecek miktar değişken olarak kaydedilir.
- “Transferİşlemi” adlı bir obje oluşturularak yapılacak transferin tüm parametreleri uygun formatta ve anahtarlar ile kaydedilir.

- Gönderimi sağlayan Baran Hesabı tarafından işlem imzalanır ve de-ğişkene kaydedilir.
- Transfer ağa kullanıcı objesi tarafından asenkron bir şekilde gönde-rilir.
- Yapılan transferin adresi konsola yazdırılır.
- Olası hatalar “.catch()” fonksiyonu ile yakalanarak ekrana yazdırılır.

Transfer : 5NLBZNVTLNWLIVPOAWCVH3ZFHRGL5AQ7CHQFE7R52GRS7SV66WJQ

Tablo 55 balance.js çıktısı

Kodun bulunduğu klasöre terminalden erişim sağlanıp “node transac-tion.js” komutunun çağırılması sonucunda hatasız bir şekilde bağlantı kurulursa Tablo 55’te ifade edilen şekilde işlem adresi ekrana yazdırılmış olacaktır. Bu işlem adresini explorerlarda aratmamız mümkündür.

GOALSEEKER	LEDGER HOME	BLOCKS	ASSETS	TRANSACTIONS	DEVELOPER PORTAL
Algorand TestNet	Search by Block Address Tx ID Application ID Asset				
NETWORK STATS					
LATEST BLOCK 24141873	TRANSACTION 5NLBZNVTLNWLIVPOAWCVH3ZFHRGL5AQ7CHQFE7R52GRS7SV66WJQ				
CIRCULATING STAKE 10,124,704,857.969908 Algos	BLOCK 24141786	Fee 0.001 Algos	DATE Fri, 16 Sep 2022 18:06:51 GMT (5 mins ago)		
ONLINE STAKE 7,580,062,448.109327 Algos	PAYOUT				
AVERAGE BLOCK TIME : 3.56 seconds	SENDER BJFNBQ7MQ6BU7PZAEEAEET4C6BIOJFIYYSSBKQWI763YLPYZXKAB5IK5U				
Reward Rate : 0 Algos	RECEIVER YV5JTE3FJLCG7JVFFWZWHY7TJ7BP43RHEXHUY4RUBUX3J342FVOIUCHFGY				
AMOUNT 1 Algos	AMOUNT 1 Algos	SENDER REWARDS 0 Algos	RECEIVER REWARDS 0 Algos		
NOTE					
ASCII Merhabalar	Base64 TWVyaGFyWxhcg==	MessagePack 77			
ADDITIONAL INFO					
FIRST ROUND 24141784	LAST ROUND 24142784				
GENESIS ID testnet-v1.0	GENESIS HASH SG01GK5KyEPtEPhtTxCByw9x8FmrnCDex9/cOUjOii=				
SENDER REWARDS 0 Algos	RECEIVER REWARDS 0 Algos				

Şekil 13

Şekil 13’de ifade edildiği üzere <https://goalseeker.purestake.io/algorand/testnet> sitesi üzerinde tesnette olduğumuza emin olarak gerçekleştir-diğimiz transfer aramasında hesaplar yazdığını not işlem ücreti tarih ve benzeri açıklamaların yer aldığı görünmektedir.

Transferi gerçekleştirmemiz sonrasında yazdığını balance.js kodunu yeniden çalıştırarak kontrol edebiliriz.

```
bakiyeSorgula("BJFNBQ7MQ6BU7PZAEEAEET4C6BIOJFIYYSSBKQWI763YLPYZXKAB5IK5U")
bakiyeSorgula("YV5JTE3FJLCG7JVFFWZWHY7TJ7BP43RHEXHUY4RUBUX3J342FVOIUCHFGY")
```

Tablo 56 index.js

Tablo 56'da verildiği şekilde index.js içerisinde “bakiyeSorgula” fonksiyonunu farklı adresler için iki kez çağrılabiliriz.

Bakiye: 8999999
Bakiye: 1000000

Tablo 57 index.js Çıktı

Fonksiyonu bulunduğu klasörden terminal üzerinden “node index.js” komutu ile çağrıdığımızda her iki adresin de bakiyesi Tablo 57'de görüldüğü şekilde karşımıza çıkmaktadır. Baran Hesabının bakiyesi 1000001 azalarken Yenal Hesabının bakiyesi 1000000 artmıştır. Bu iki değer arasındaki fark işlem ücretinden kaynaklanıyor olmaktadır.

JavaScript ve Algorand İle ASA Token İşlemleri

Algorand yapısı dahilinde üzerinde token oluşturmaya izin veren bir blokzincirdir. Algorand bu Tokenlara ASA (Algorand Standard Assets) adını vermiştir. Algorand blokzinciri üzerinde bir ASA oluşturmak için ekstra bir akıllı kontrat oluşturmaya gerek bulunmamaktadır. Bu başlığımızda Algorand SDK ile JavaScript dili ile ASA'lar oluşturacağız, transfer edeceğiz ve yok edeceğiz.

ASA Oluşturulması İçin İlgili Hesapların ve Araçların Kurulması

JavaScript üzerinde daha öncesinden oluşturduğumuz index.js dosyasını içeren dosya üzerinde işlemlerinizi yapacağınız. İlk olarak transfer ve token oluşturma işlemlerini yapacağımız adresleri oluşturmamız gerekmektedir.

AdresYarat()
AdresYarat()

Tablo 58

Bu kısımda Tablo 58'de ifade edildiği üzere daha önceden yazdığımız ve index.js dosyasına eklediğimiz “address” kodunu tanımladığımız değişken adı ile çağrılabiliriz. İki adres tanımlanacağı için fonksiyonu art arda iki kez çağrılabiliriz.

```

Hesap Adresi: {
addr: 'IOIBYZPBAISY636GLWF2PRFXZCPVN2DKXKCCOUCANPA2BROUKV2UTOPVYA',
sk: Uint8Array(64) [
139, 85, 95, 75, 240, 65, 212, 43, 186, 198, 11,
75, 183, 95, 166, 243, 95, 248, 208, 247, 18, 196,
175, 233, 96, 145, 172, 111, 208, 171, 171, 95, 67,
144, 28, 101, 225, 2, 37, 143, 111, 198, 93, 139,
167, 196, 183, 200, 159, 86, 232, 106, 186, 132, 39,
80, 64, 107, 193, 160, 197, 212, 85, 117
]
}

Mnemonic Şifre: ramp later certain business possible memory cupboard armed truly learn spring
you vague vintage nut series curious lonely emerge width patch turtle wool able rice

Hesap Adresi: {
addr: 'BDG7MLDNCHDJH4XV7FS3YRQN2NTIUMWM4FKFF5Z5YQ4JZGQRXNEFD5L4TQ',
sk: Uint8Array(64) [
128, 169, 78, 245, 32, 229, 73, 96, 12, 225, 255,
27, 152, 252, 69, 16, 11, 254, 250, 62, 86, 115,
253, 223, 0, 250, 80, 166, 224, 228, 162, 201, 8,
205, 246, 44, 109, 17, 198, 147, 242, 245, 249, 101,
188, 70, 13, 211, 102, 138, 50, 204, 225, 84, 82,
247, 61, 196, 56, 156, 154, 17, 187, 72
]
}

Mnemonic Şifre: copy deny kingdom faith need blossom anchor zoo address venue carry rain
winter salt rapid infant zero address laptop govern scout rice escape absorb stove

```

Tablo 59

Index.js dosyasını oluşturduğumuzda karşımıza Tablo 59'da ifade edilen çıktı gelmektedir. Bu çıktı içerisinde bizlere sadece Mnemonic Şifre ve Adres değerleri gerekmektedir ve bu nedenle sonradan kullanmak üzere kaydedilmelidir. Secret Key adını verdigimizi "sk" değeri Mnemonic şifre içerisinden istenilen vakit oluşturulabildiği için ayrıca kaydetmeye gerek olmamaktadır.

Çıktıda ifade edilen 'IOIBYZPB...A2BROUKV2UTOPVYA' ve 'BDG7ML...Z5YQ4JZGQRXNEFD5L4TQ' adresleri projemizde kullanacağımız adresler olacaktır. Anlatım kolaylığı açısından bu adreslere anlatım içerisinde baş harflerine denk gelen isimler kullanılacaktır. 'IOIBYZPB...A2BROUKV2UTOPVYA' adresi için İrem adı kullanılacak olunurken 'BDG7ML...Z5YQ4JZGQRXNEFD5L4TQ' adresi için Beren adı kullanılacaktır. Transfer ve coin işlemleri belirli bir işlem ücreti (fee) gerekmektedir. Bu nedenle oluşturduğumuz hesaplara <https://bank.testnet.algorand.network/> adresi üzerinden Test coinleri eklemek mümkündür.

Algorand dispenser

I'm not a robot 
[Privacy](#) - [Terms](#)

The dispensed Algos have no monetary value and should only be used to test applications.

This service is gracefully provided to enable development on the Algorand blockchain test networks.

Please do not abuse it by requesting more Algos than needed.

Status: Code 200 success: "DGRD6SAVNWS4RKXIDAO6RWHA76IIA7KXTTO6TV43POTCOGMY2SAQ"

Şekil 14 Beren Hesabı Bakiye Ekleme

Algorand dispenser

I'm not a robot 
[Privacy](#) - [Terms](#)

The dispensed Algos have no monetary value and should only be used to test applications.

This service is gracefully provided to enable development on the Algorand blockchain test networks.

Please do not abuse it by requesting more Algos than needed.

Status: Code 200 success: "6B3GO5ISM27AMMVBIJ3YHBNXHY7I64PGCZOD46TLLBKB2L7EJA4A"

Şekil 15 İrem Hesabı Bakiye Ekleme

Şekil 14 ve 15'de görüldüğü üzere ilgili hesaplara site içerisinde “dispense” butonu sayesinde bakiye ekleme yapılır. Aralarında farklılık olması amacıyla İrem hesabına iki kez dispense yapılmıştır.

Bu adreslere bakiye eklenip eklenmemesi durumuna farklı yöntemler ile bakılması mümkündür. İlk olarak önceki bölümlerde yazdığımız bakiye sorgulama kodu ile bu işlemi gerçekleştirebiliriz.

```
const bakiyeSorgula = require("./balance")
require("dotenv").config()

bakiyeSorgula("IOIBYZPBAISY636GLWF2PRFXZCPVN2DKXKCCOUCANPA2BROUKV2UTOPVYA")
bakiyeSorgula("BDG7MLDNCHDJH4XV7FS3YRQN2NTIUMWM4FKFF5Z5YQ4JZGQRXNEFD5L4TQ")
```

Tablo 60

Kod içerisinde bakiye sorgulamak için index.js üzerinde require ile çağrıdığımız bakiye sorgulama dosyasını istediğimiz isimle kaydederiz ve bu değişkene parametre olarak bakiyelerini sorgulamak istediğimiz adresleri

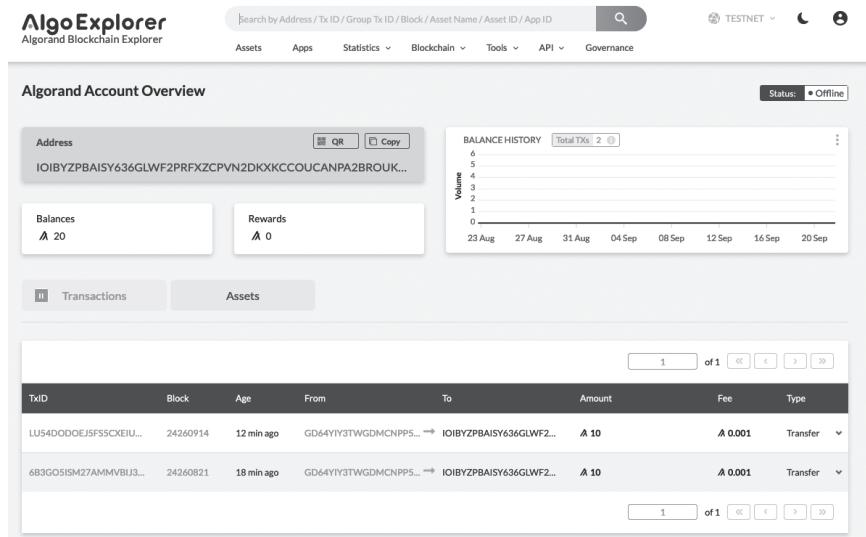
gireriz. İki adres inceleyeceğimizden dolayı Tablo 60'ta ifade edildiği üzere bu işlemi iki farklı adres için iki kere alt alta gerçekleyebiliriz.

Bakiye: 20000000
Bakiye: 10000000

Tablo 61

İlgili kodu bulunduğu dosya konumu üzerinden terminalden “node index.js” komutu ile çalıştırabiliriz. Bu kodun çalışması sonrasında Tablo 61'de ifade edildiği gibi bir çıktı ile karşılaşmaktadır. Daha önce de bahsedildiği gibi 20000000 Milialgo yani 20 Algo 'ya sahip hesap iki kez “dispense” işlemi yapılan İrem hesabı olmaktadır. 10000000 Milialgo yani 10 Algo'ya sahip olan hesap ise Beren hesabı olmaktadır.

Bakiye sorgu işlemini aynı zamanda <https://algoexplorer.io/> sitesinden de gerçekleştirebiliriz. Bu siteden aynı zamanda transferin kendini de görüntüleyebiliriz.



Şekil 16 İrem Hesabı

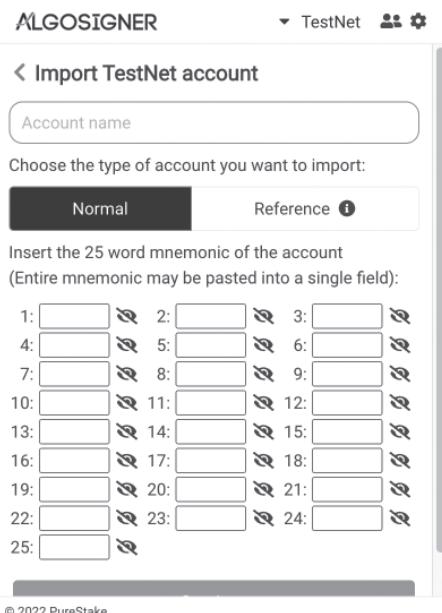
TxID	Block	Age	From	To	Amount	Fee	Type
DGRD65AVNW54RICKIDA...	24260831	19 min ago	GD64YIY3TWGDMCNPP553DZPPR6LDUSFQOIJVFDPPXWEG3F-VOJCCDBHU5A	BDG7MLDNCHDJH4XV7FS3YRQN2NTIUMWM4FKFF5Z5YQ4JZGQ...	A 10	A 0.001	Transfer

Şekil 17 Beren Hesabı

Şekil 16 ve 17'de görüldüğü üzere “algoexplorer” sitesinden de sağ üst köşede testnette olduğumuza emin olarak adres sorgusu gerçekleştirilir. Bu sitede yapılan transferin hangi adres üzerinden geldiği de görsel olarak kullanıcılar sunulmaktadır. Transferlerde iki transferin de ortak hesaptan yapıldığı göze çarpmaktadır. Bu Algorand’ın geliştiricilere test coinini yolladığı “GD64YIY3TWGDMCNPP553DZPPR6LDUSFQOIJVFDPPXWEG3F-VOJCCDBHU5A” adresi olmaktadır.

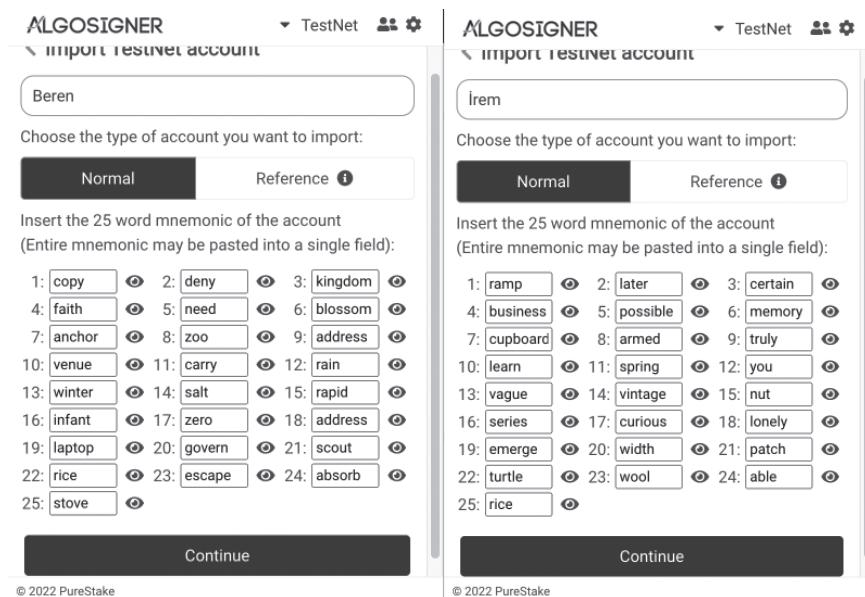
Oluşturulan hesapların görsel bir şekilde takibi için bir sıcak cüzdan kullanabiliriz. Bu işlem için Algosigner adlı eklenti kullanılabilir. Bu eklenti Chromium tabanlı tarayıcılarda <https://chrome.google.com/webstore/detail/algosigner/kmmolakhbglpkjkcjkebenjheonagdm> ile tarayıcıya eklenir. Bu kitabın yazıldığı tarihte Firefox için henüz Algosigner eklentisi bulunmamaktadır.

Eklenti eklendikten sonra eklenilen tarayıcı içerisinde yapboz parçası ile erişilebilir. Açıldıktan sonra ilk kez açlıyorsa bir hoş geldin ekranı ile karşılaşmaktayız. Bu ekranın “get started” seçeneğini seçerek ilerlememiz gerekmektedir. Sonrasında eklenti bu cüzdan içerisinde bulunacak tüm adresler için ortak kullanılacak bir şifre talep etmektedir. Bu şifre en az 8 karakterden oluşmaktadır. Şifreleri oluşturduktan ve “Create wallet” butonuna basıldıktan sonra karşımıza “Add account” yazısının olduğu bir ekran gelmektedir. İlk olarak bu ekranın sağ üst köşesinde bulunan “MainNet” yazısına basarak “Testnet” seçeneğini seçmemiz gerekmektedir. Sonrasında altında bulunan “Add account” düğmesine basmaktayız. Burada karşımıza 3 adet seçenek çıkmaktadır. Bu seçenekler arasından “Import existing account” seçeneği ile işlemlerimize devam edeceğiz.



Şekil 18

Açılan ekranda Şekil 18'de ifade edildiği şekilde bir ekran karşımıza çıkmaktadır. İlk olarak “Account name” kısmında cüzdanımıza bu cüzdanı tanımlayacak bir isim vermemiz gerekmektedir. Bu isim adres ile doğrudan bağlanmayacak olup sadece Algosigner üzerinde görsel olarak var olacaktır. Bu nedenle istenilen bir isim tercih edilebilir. İşmin eklemesi gerçekleştikten sonra hemen altında bulunan cüzdan tipi seçeneğinin “Normal” seçeneğinde olduğundan emin oluruz. Son olarak 25 karakterden bulunan Mnemonic anahtarı cüzdan içerişine girmemiz gerekmektedir. Bu anahtarı kelime kelime her satırda ekleye gitmemiz gerekmektedir. Ekleme işlemi tamamlandıktan sonra alt kısma kaydırarak “Continue” tuşuna basılarak devam edilir. Algosigner son adım olarak eklenti kurulumunda girilen şifreyi istemektedir.



Şekil 19

Şekil 19'da ifade edilen şekilde doldurulan bilgiler ile cüzdanlar Algo-signer üzerinde eklenmiş hal almaktadır.

ASA Token Oluşturma

Hesapların oluşumu sonrasında bu hesaplar içerisindeki biri seçili ve bu adres üzerinden token oluşturulabilir. Simdilik İrem hesabından token üreteceğiz. Bu nedenle index.js dosyasının bulunduğu klasöre gidilerek "asa_irem_create.js" adlı dosya oluşturulabilir. Token oluşturma kodları bu dosya içerisinde gerçekleştirilecektir.

```
const Algosdk = require('algosdk');
require("dotenv").config()
```

Tablo 62

Her JavaScript dosyasında başlamamız gereken şekilde Tablo 62'da belirtildiği gibi bu dosyada da paketlerin tanımlanması ile başlamamız gerekmektedir. Paketleri "require" fonksiyonu ile JavaScript içerisinde istediğimiz değişkene atayarak kullanabiliriz. Token oluşturma işlemi için şu anlık iki adet paket eklemektedir. Bu paketler:

- **Algosdk:** Algorand blokzincirine bağlanmamızı sağlayan SDK paketidir.

- **Dotenv:** Kodumuzu paylaşırken saklamak istediğimiz değerleri içe-resinde sakladığımız .env dosyasına erişimi sağlayan bir pakettir.

```
const server="https://testnet-algorand.api.purestake.io/ps2";
const port="";
const token={ "x-api-key": process.env.API };
```

Tablo 63

Paketlerin tanımlanmasından sonra işlemin gerçekleşeceği özellikleri tanımlamak gerekmektedir. Bu nedenle birtakım değişkenler oluşturulmaktadır. Tablo 63'te ifade edilen bu değişkenleri biraz daha ayrıntılı anlatmak gerekirse;

- **Server:** İşleminin yapılacakı ağı göstermektedir. Token oluşturma işlemini testnet üzerinden oluşturacağımızdan dolayı “Purestake” üzerinden alınan URL ile tanımlama yapılmaktadır.
- **Port:** İşlemin yapılacakı bağlantı portuna denk gelmektedir. Bazı uygulamalar için 4001 değeri de kullanabilir. Simdilik boş bırakılacaktır.
- **Token:** API anahtarını ifade etmektedir. Bu anahtar öncesinde de detaylı şekilde anlatıldığı üzere “Purestake” sitesinden elde edilmekte olup zincir üzerinde yapılan işlemlerin hesabımıza bağlanmasıını sağlar. Obje olarak tanımlanır ve “x-api-key” anahtarını alan bir değer atanır. Bu değer içerisinde direkt olarak API anahtarını string olarak girmek mümkündür. Ancak güvenlik ve profesyonellik açısından son derece önemli olan bu API anahtarını paylaşma gizli olan “.env” dosyasından almak daha işlevsel olacaktır. Önceden eklediğimiz paket üzerinden “process.env” fonksiyonu içerisinde API değişkenine erişmek için “process.env.API” şeklinde yazmamız gerekmektedir. Bu şekilde yazıldıkten sonra anahtar ile obje içerisinde token değişkeni olarak değişken kaydedilebilir.

```
let kullanıcı = new Algosdk.Algodv2(token, server, port);
```

Tablo 64

İşlem değişkenlerinin tanımlanmasından sonra bu değişkenlerin birleştilmesi gerekmektedir. Bu işlemi Tablo 64'te ifade edildiği üzere Algosdk paketinin “Algody2” metodu ile gerçekleştirmek mümkün olmaktadır. Değişkenlerin bir araya gelmesinden bir obje ortaya çıktıktan dolayı fonksiyon çağrılmadan önce başına “new” ibaresi eklenmelidir. Fonksiyon içeri-

sine öncesinde tanımladığımız token, server ve port değerlerini almaktadır. Sıralamaya dikkat edilmesi gerekmektedir. Yanlış sıralama ile parametreler çağırılırsa fonksiyon doğru bir şekilde çalışmayaacaktır.

```
var irem_mnemonic = "ramp later certain business possible memory cupboard armed
truly learn spring you vague vintage nut series curious lonely emerge width patch
turtle wool able rice";
var iremHesap = Algosdk.mnemonicToSecretKey(irem_mnemonic);
```

Tablo 65

Token'ların bir hesaptan türetilmesi gerekmektedir. Bu nedenle bu hesaba tokenlar bağlı hale gelmektedirler. Tablo 65'de ifade edildiği üzere kodumuza tokenların oluşturulacağı hesabın eklenmesi gerekmektedir. İlk olarak daha öncesinde oluşturduğumuz İrem hesabının mnemonic anahtarını bir değişkene kaydetmemiz gerekmektedir. Bu değişkenin adı tamamen tercihe bağlıdır ancak anlatım kolaylığı amacıyla "irem_mnemonic" isimlendirilmesi kullanılacaktır. Değişken tanımlanması yapıldıktan sonra bu anahtarın hesabın kendisine dönüştürülmesi yani "SecretKey" değerinin oluşturulması gerekmektedir. Bu işlemi gerçekleştirmek için Algosdk paketi içerisinde "mnemonicToSecretKey" metodu çağrılabılır. Bu metot içerisinde mnemonic anahtarını almaktadır. Çıktı olarak da hesabı obje olarak ifade etmektedir.

```
(async () => {
  ...
})
```

Tablo 66

Hesap ve kullanıcının oluşması sonrasında Token'ın içeriğini tanımlayacağımız fonksiyona geçiş yapılabilir. Token'ı oluşturacak parametreler zincir üzerinden elde edilirken belirli miktarlarda beklenmesi gerektiği için fonksiyon "async" olarak tanımlanacaktır. Tablo 66'da ifade edildiği üzere fonksiyon herhangi bir çağrılmamıştır. Ayrıca fonksiyonun içerisinde herhangi bir parametre almamasından dolayı parantezler boş bırakılmıştır. Tanimlarımız ve token oluşumumuz bu asenkron fonksiyon içerisinde yer olacaktır.

```

let params = await kullanıcı.getTransactionParams().do();
let note = undefined;
let adres = iremHesap.addr;
let defaultFrozen = false;
let decimals = 0;
let totalIssuance = 1000000;
let unitName = "JS COİN";
let assetName = "JS Coin";
let assetURL = "https://www.linkedin.com/in/aybarsayan/";
let assetMetadataHash = "01234567890123456789012345678901";
let manager = iremHesap.addr;
let reserve = iremHesap.addr;
let freeze = iremHesap.addr;
let clawback = iremHesap.addr;

```

Tablo 67

Token oluşturulurken belirli parametrelere ihtiyaç duyulmaktadır. Bu parametrelere Token'ı tanımlayan ve yapılandıran özellikler olmaktadır. Tablo 67'de ifade edildiği üzere birçok parametre Token'ı tanımlayan değişkenler içerisinde yer alabilir. Bu değişkenler transfer bilgilerini içerebilmektedir. Bu durumun nedeni Token oluşturma işleminin temelinde bir transfer işlemi olmasıdır. Bu değişkenlerin tanımlarına bakılacak olunursa;

- **Params:** Kullanıcı ile türetilen işlem parametrelerini oluşturan parametredir. "getTransactionParams()" metodu ile çalıştırılır ve fonksiyon olmasından dolayı .do() eklentisi ile çalıştırılır. İstenilen isim ile kayıt edilebilir.
- **Note:** Transferin gerçekleştirken içermesi istenilen not değişkenidir. Undefined yani boş olarak bırakılabilir. İçerisine girilecek değer transferörneğinde de ifade edildiği üzere "TextEncoder" ile şifrelenmelidir. Bu şifreleme sayesinde zincirde okunabilen bir hal alacaktır. İstenilen isim ile değişken üzerinde kaydedilebilir.
- **Adres:** Token oluşturma işleminin gerçekleşeceği hesabın adresini ifade etmektedir. Öncesi mnemonic kullanarak hesaba eriştiğimiz değişken içerisindeki ".addr" anahtarı ile erişilebilir. İstenilen isim ile değişken kaydedilebilir.
- **DefaultFrozen:** Oluşturulan Tokenların başlangıç durumunda dondurulmuş olup olmadığını ifade etmektedir. "Bool" değer almaktaadır. Şu anda projemizde dondurulmuş olmasını istemediğimizden dolayı False olarak atanılabilir. İstenilen değişken ismi ile kaydedilir.
- **Decimals:** Parametre tanımı içerisinde bakiye parametresinde yer alacak değerin ondalık değerinin ne olacağını ifade etmektedir. İçerisine integer değeri almaktadır. Eğer projemizde kullanacağımız

değeri olarak 0'değerinde bırakırsak herhangi bir basamak tanımaması yapmaz ve yazdığımız arz aynı şekilde aktarılır. İstedigimiz değişken isimlendirmesi ile kaydedilebilir.

- **TotalIssuance:** Oluşması istenen toplam arz'ı ifade etmektedir. İçeresine integer değerler almaktadır. Projemizde toplam 1 milyon Token oluşmasını istememizden dolayı 1000000 değeri atanabilir. İstenilen değişken ismi ile kaydedilebilir.
- **UnitName:** Token'ın tanımlanma birimi olmaktadır. Oluşturulurken belirlenir. Maksimum 8 byte değer alabilmektedir. İstenilen isim ile kaydedilebilmektedir.
- **AssetName:** Token'ın ismini belirtmektedir. 32 byte'a kadar istenilen isim verilebilir. İstenilen değişken isimlendirmesi ile kaydedilebilir.
- **URL:** Token hakkında daha fazla bilgi alınabilecek sitenin URL'sini ifade etmektedir. Maksimum 96 byte büyülüğünde veri kaydedilebilmektedir. İstenilen isim ile kaydedilebilir.
- **MetaDataHash:** Varlığınızla ve/veya varlık sahipleriyle ilgili bazı meta verilerin 32 baytlik bir karma olması amaçlanmıştır. Bu meta verinin formatı uygulamaya bağlıdır. İstenilen değer verilebilmektedir. İstenilen değişken ismi ile kaydedilir.
- **Manager:** Token'ı bir bakıma yönetecek kişinin adresini ifade etmektedir. Bu hesap Token'ı inşa edebilir ve gerekli şartlar sağlandığında Token'ı yok edebilir. Uygulamamızda İrem hesabının Token'ı oluşturulmasından dolayı yönetim de İrem adresine verilecektir. İstenilen değişken ismi ile kaydedilir.
- **Reserve:** Oluşması muhtemel token rezervlerinin hangi hesapta toplanacağını ifade etmektedir. Bu hesap tek başına zincir üzerinde herhangi bir yüküme sahip değildir. Dosya içerisinde İrem hesabının adresine bağlanabilir. İstenilen değişken ismi ile kaydedilir.
- **Freeze:** Dondurulan varlıkların tutulması istenilen adres olmaktadır. Boş bırakılırsa dondurma işlemine izin verilmez. Olası dondurulan varlıkların hesapta durmasını istedigimizden dolayı İrem hesabının adresi girilebilir. İstenilen değişken ismi ile kaydedilir.
- **Clawback:** Geri alma işlemlerinin gerçekleşebileceği hesabı ifade etmektedir. Boş bırakılırsa geri alma işlemine izin verilmez. İstenilen değişken ismi ile kaydedilir.

```
let txn = Algosdk.makeAssetCreateTxnWithSuggestedParams(adres, note,
totalIssuance, decimals, defaultFrozen, manager, reserve, freeze,
clawback, unitName, assetName, assetURL, assetMetadataHash, params);
```

Tablo 68

Parametrelerin tanımlanmasından sonra bu parametrelerin birer girdi olarak transfer fonksiyonuna eklenmesi gerekmektedir. Bu nedenle Tablo 68'de ifade edildiği üzere istedigimiz isimle kaydettiğimiz içerisinde Algosdk Kütüphanesini kullanarak “makeAssetCreateTxnWithSuggestedParams()” metodunu çağırabiliriz. Bu metot içerisine sırasıyla tüm Token parametrelerini almaktadır. Ancak sıralama çok önemlidir. Sıralamada yanlışlık yapılması halinde fonksiyon istenilen şekilde çalışmayaçaktır.

```
let rawSignedTxn = txn.signTxn(iremHesap.sk);
```

Tablo 69

Transfer kaydedildi ancak İrem hesabı ile henüz imzalanmadı. İmzalama işlemini gerçekleştirmek için Tablo 69'da ifade edildiği üzere daha öncesinde kaydettiğimiz transferin signTxn() metodu ile imzalanması gerekmektedir. Bu fonksiyon içerisinde daha öncesinde mnemonic anahtar içerisinde türetilen “iremHesap.sk” değerini almaktadır.

```
let tx = (await kullanıcı.sendRawTransaction(rawSignedTxn).do());
console.log("Transfer: : " + tx.txId);
```

Tablo 70

İmzalanan transfer gönderime hazır hale gelmektedir. Gönderim işleminin yapılması için Tablo 70'de ifade edildiği gibi asenkron olarak “await” anahtar kelimesi ile kullanıcı “sendRawTransaction” metodunu kullanarak zincire gönderim yapmaktadır. Metotun çalıştırılması gerektiği için sonuna “.do()” metodu eklenmelidir. Zincire gönderilen transfer geri dönüt olarak bir transfer numarası döndürmektedir. Bu değeri “console.log” ile terminalde yazdırabiliriz.

```
0.catch(e => {
  console.log(e);
});
```

Tablo 71

Asenkron fonksiyondan çıktıktan sonra Tablo 71'de ifade edildiği üzere olası hataların yazdırılması amacıyla “.catch()” metodu ile hatalar yakalanır ve console.log ile terminale yazdırılır. Bu şekilde kod tamamlanmış bir hal alır.

Transfer : HCKTPKBF2YEGDWOUUY22A7M2YILY6G3G73AXMDBXYKBL4JGRVRRA

Tablo 72 asa_irem_create.js Çıktısı

Kod dosyası bulunduğu klasörden “node asa_irem_create” komutu ile çalıştırıldığında karşımızda Tablo 72'de bulunan ifade çıkmaktadır. Bu şekilde transfer numarası çıktısı veriliyorsa kodumuz başarılı bir şekilde çalışmış demektir. Transfer işlemine Algoexplorer'dan veya Goalseeker siteleri üzerinden bakılması mümkün olmaktadır.

TRANSACTION	HCKTPKBF2YEGDWOUUY22A7M2YILY6G3G73AXMDBXYKBL4JGRVRRA		
BLOCK	24284516	Fee	0.001 Algos
DATE	Thu, 22 Sep 2022 18:06:31 GMT (7 mins ago)		
ASSET CONFIGURATION			
ADDRESS	IOIBYZPBAISY636GLWF2PRFXZCPVN2DKXCCOUCANPA2BROUKV2UT0PVYA		
ASSET NAME	JS Coin	UNIT NAME	JSCOIN
		URL	https://www.linkedin.com/in/aybarsayan/
MANAGER ADDRESS	IOIBYZPBAISY636GLWF2PRFXZCPVN2DKXCCOUCANPA2BRO...		
RESERVE ADDRESS	IOIBYZPBAISY636GLWF2PRFXZCPVN2DKXCCOUCANPA2BRO...		
FREEZE ADDRESS	IOIBYZPBAISY636GLWF2PRFXZCPVN2DKXCCOUCANPA2BRO...		
CLAWBACK ADDRESS	IOIBYZPBAISY636GLWF2PRFXZCPVN2DKXCCOUCANPA2BRO...		
ADDITIONAL INFO			
FIRST ROUND	24285514		
GENESIS ID	5G01GKs7e7iEPitxByw9xFmnrcDexi9cOUj0i=		
SENDER REWARDS	0 Algos		
RECEIVER REWARDS	0 Algos		

Şekil 20 Transfer Kontrolü

Şekil 20'de ifade edildiği üzere testnet üzerinde emin olunarak transfer detaylarına bakılırsa parametre olarak kaydettiğimiz arz, “asset name” gibi değerlerin Token içerisinde bulunduğu gözükmemektedir.

```

const Algosdk = require('algosdk');
require("dotenv").config()

const server="https://testnet-algorand.api.purestake.io/ps2";
const port="";
const token={ "x-api-key": process.env.API };
let kullanıcı = new Algosdk.Algodv2(token, server, port);

var irem_mnemonic = "ramp later certain business possible memory cupboard armed
truly learn spring you vague vintage nut series curious lonely emerge width patch
turtle wool able rice"; // fill in yours
var iremHesap = Algosdk.mnemonicToSecretKey(irem_mnemonic);

(async () => {
let params = await kullanıcı.getTransactionParams().do();
let note = undefined;
let adres = iremHesap.addr;
let defaultFrozen = false;
let decimals = 0;
let totalIssuance = 1000000;
let unitName = "JS COİN";
let assetName = "JS Coin";
let assetURL = " https://www.linkedin.com/in/aybarsayan/ ";
let assetMetadataHash = "01234567890123456789012345678901";
let manager = iremHesap.addr;
let reserve = iremHesap.addr;
let freeze = iremHesap.addr;
let clawback = iremHesap.addr;

let txn = Algosdk.makeAssetCreateTxnWithSuggestedParams(adres, note,
totalIssuance, decimals, defaultFrozen, manager, reserve, freeze,
clawback, unitName, assetName, assetURL, assetMetadataHash, params);

let rawSignedTxn = txn.signTxn(iremHesap.sk);
let tx = (await kullanıcı.sendRawTransaction(rawSignedTxn).do());
console.log("Transfer : " + tx.txId);
})().catch(e => {
  console.log(e);
});
});

```

Tablo 73 asa_irem_create.js Tamamı

Tablo 73'de yazmış olduğumuz asa_irem_create.js dosyasının tamamı bulunmaktadır. Bu kodun bir bütün olarak aşamaları ile neler yaptığına bakılırsa:

- İlk olarak gerekli kütüphaneler tanımlanır.
- Sonrasında sorguların yapılacakı kullanıcının API anahtarı portu ve ağ adresi değişkenler üzerinde kaydedilir.
- Kaydedilen kullanıcı değerleri "kullanıcı" adlı obje oluşturularak kaydedilir.

- Token'ın olusacağı adresin mnemonic adresi değişkene atanır.
- Mnemonic anahtardan hesabın kendisi SDK kullanılarak türetilir.
- Asenkron fonksiyon oluşturulur.
- Fonksiyonun içerisinde Token parametreleri tanımlanır.
- Tanimlanan Token parametreleri uygun sıra ile transfere hazırlamak üzere SDK ile “txd” değişkeni üzerinde kaydedilir.
- Transfer işlemi Token'ı oluşturan hesabın Secret-Key'i tarafından imzalanır.
- İmzalanan transfer kullanıcı tarafından zincire gönderilir.
- Gerçekleşen transferin adresi ekrana yazdırılır.
- Olası hatalar “.catch()” metodu tarafından yakalanarak terminale yazdırılır.

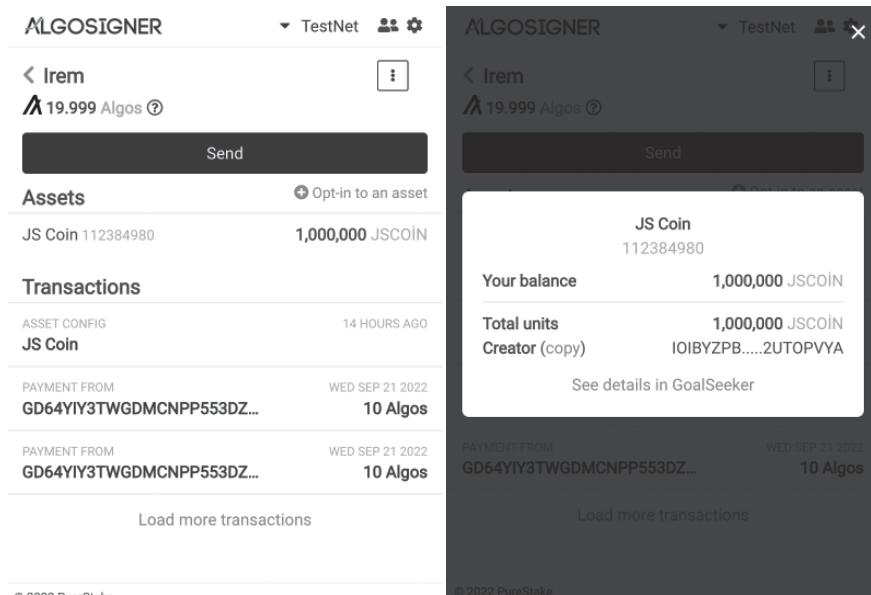
Şekil 21

Tamamlandıktan sonra Algosigner üzerinden hesaptaki ASA Şekil 21'de ifade edildiği üzere görüntülenebilir.

Farklı Bir Hesaba ASA Ekleme ve Çıkarma

Token oluşturma başarılı bir şekilde tamamlansa da bu tokeni farklı hesaplara atmamız gerekeceğinden dolayı çok bir işlevi bulunmamaktadır. Bu işlevin birçok yöntemi bulunmaktadır ancak projemiz dahilinde Algosigner ve JavaScript kullanarak bu ekleme işini gerçekleştireceğiz. Bu işleme aynı zamanda Opt-in denilmektedir.

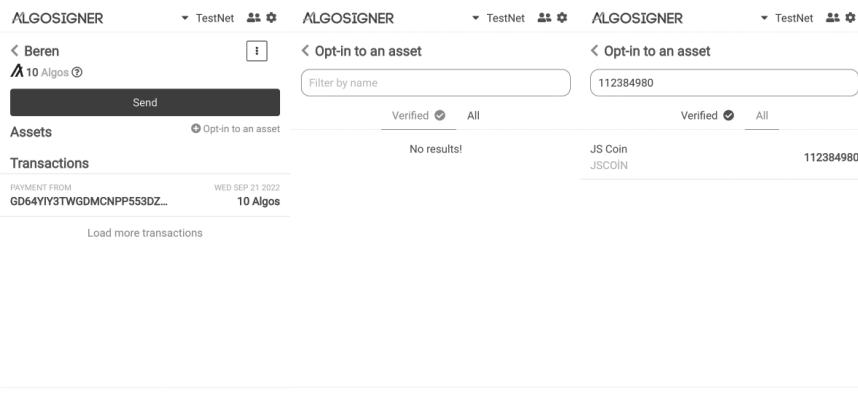
İlk olarak ekleme işlemini gerçekleştirirken bu Token'i tanımlayan bir tanımlayıcı olması gerekmektedir. Bu işlevi Algorand üzerinde AssetID'ler ile gerçekleştirmektedir. Bu AssetID'leri Algosigner üzerinde eklediğimiz hesaplardan gözlemlayabiliriz.



Şekil 22

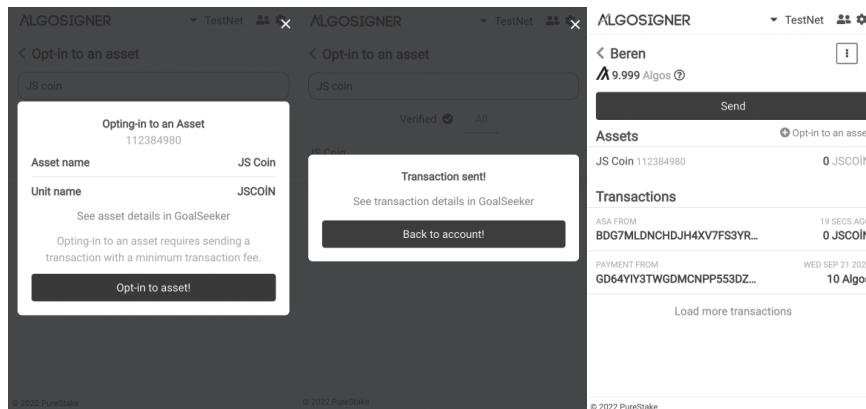
Şekil 22'de ifade edildiği üzere Token'i içeren hesabımıza Algosigner ile eriştiğimizde Assets kısmının altında belirlediğimiz Token adı yanında "AssetID" yer almaktadır. Bu değer zincir tarafından Token oluşturulurken atanmaktadır. Bu nedenle Token'i oluşturan hesap bu değeri değiştiremez. Ayrıca Token bilgilerinin üzerine basıldığında açılan "Popup" ekranda Token adımızın hemen altında aynı değer yer almaktadır. Bu değer oluşturduğumuz token için "112384980" olmaktadır.

Algosigner üzerinden ekleme yapılacakken ilk olarak eklenim yapılacak adı seçmemiz gerekmektedir. Bu işlem için Beren adresini seçeceğiz. Algosigner'in her browser açıldığında yeniden şifre istediği göz önünde bulundurulmalıdır. Ayrıca eklenti açıldığında önceden eklediğiniz hesapların görünmemesi ile karşı karşıya kalabilirsiniz. Bu durumun nedeni adreslerin testnet üzerinde yer almasıdır ve cüzdanın başlangıcında kendini otomatik olarak mainnet'de göstermesidir. Sağ üst kısımdan testnet kısmına basıldığında hesaplar önceden olduğu gibi karşımıza çıkacaktır.



Şekil 23

Açılan hesap sayfasında ilk olarak şekil 23'te ifade edildiği üzere turuncu ile yazılı “Opt-in to an asset” seçeneğini seçmemiz gerekmektedir. Sonrasında karşımıza bir arayüz çökmektedir. Bu arayüz içerisinde iki alt sekme bulundurmaktadır. İlk alt sekme Algorand tarafından doğrulanan Asset'leri gösterirken ikinci sekme ağ üzerindeki tüm Assetler'i ifade etmektedir. Oluşturduğumuz Token henüz doğrulanmadığı için “All” kategorisinde yer almaktadır. Kategori içerisinde girildiğinde isim ile arama mümkün olsa da karışıklık olmaması amacıyla “AssetID” ile arama gerçekleştirilebiliriz.



Şekil 24

Eklenmek istenen Token seçildikten sonra Şekil 24'te de ifade edildiği üzere “Opt-in to asset!” seçeneği seçilir ve hesap şifresi girilmesinden sonra Token Hesaba eklenmiş olunur.

Bazı durumlarda bakiyesi bitmiş olan Tokenlar'ı adresimizde yer klap-

maması için çıkartmak durumunda kalabiliriz. Bu işlemi de hem JavaScript hem de Algosigner ile gerçekleştirebiliriz.

Token’ı hesaptan çıkışma işlemine “Opt-out” işlemi denilmektedir. Bu işlemi Algosigner’da yapmak için ilk olarak hesabından çıkarılması istenilen adrese giriş yapılarak Assets kısmından çıkarılacak Token veya Asset seçilir. Bu asset’ın seçilmesinden sonra açılan Pop-up ekranından en alttaki seçenek olan “Opt-out of this asset” seçeneği seçilir. Algosigner bu kısımda şifre talep etmektedir. Şifrenin girilmesinden ve “Continue” seçeneğinin seçilmesi sonrasında Asset’den başarılı bir şekilde çıkışmış olur. Eklentiye çıkış yaparak bu işlemin gerçekleşip gerçekleşmediğini kontrol etmek mümkündür.

Algosigner üzerinden “Opt-in” ve “opt-out” işlemleri yapıldıktan sonra bu işlemlerin nasıl JavaScript ile yapılacağına bakabiliriz. Önceki sayfalar da Beren Hesabından Token’ı çıkardığımız için yeniden ekleme işlemini JavaScript ile yapabiliriz. Bu işlemi yapmak için index.js dosyasının bulunduğu dosyaya girip beren_opt_in.js dosyası oluşturabiliriz.

```
const Algosdk = require('algosdk');
require("dotenv").config()
```

Tablo 74

Kod içerişine girdiğimizde ilk olarak diğer tüm JavaScript dosyalarında yapıldığı üzere paketlerin tanımlanması gerekmektedir. Tablo 74’te ifade edilen şekilde JavaScript içerisinde kütüphaneler “require” fonksiyonu ile çağrılabılır. Sırasıyla eklenen paketlere bakılması gerekirse:

- **Algosdk:** Algorand blokzinciri ile bağlantı kurmamızı sağlayan SDK paketidir. İstenilen isimlendirme ile kod içerişine eklenebilir.
- **Dotenv:** Paylaşmak istenilmeyen gizli verilerin “.env” dosyasından çekilmesini sağlayan pakettir.

```
const server="https://testnet-algorand.api.purestake.io/ps2";
const port="";
const token={"x-api-key": process.env.API};
let kullanıcı = new Algosdk.Algodv2(token, server, port);
```

Tablo 75

Paketlerin eklenmesi sonrasında zincir üzerinde işlemlerin yapılabilmesi için bazı özelliklerin tanıtılması gerekmektedir. Bu özelliklere tablo 75’de ifade edildiği üzere bakılacak olunursa:

- **Server:** Yazdığımız kodun hangi zincir üzerinde çalışacağını ifade etmektedir. Testnet üzerinde çalışacağımız için “Purestake” sitesinden elde edilen testnet adresi bu değişken içerisinde kaydedilebilir. String değeri almaktadır.
- **Port:** Bağlantının yapılacak portu ifade etmektedir. Bu port bilgisi ile kodu farklı yerlerde çalıştırmak mümkündür. Kod içerisinde şimdilik boş bırakılacaktır ancak 4001 değeri girilerek de bazı durumlarda kullanılabilir. String değeri almaktadır.
- **Token:** Algorand blozkincirine bağlanırken tanımladığımız API anahtarıdır. Bu anahtar önceki bölümlerde ifade edildiği üzere Purestake sitesinden elde edilebilir. Obje değeri almaktadır. Obje içerisinde “x-api-key” anahtarı içerisinde API anahtarı yazılmalıdır. Ancak güvenlik ve profesyonellik açısından bu değeri “process.env” fonksiyonu ile “.env” dosyası içerisinde çekerememiz mümkündür.

```
let kullanıcı = new Algosdk.Algodv2(token, server, port);
```

Tablo 76

Tanımların yapılmasıından sonra bu değişkenlerin bir kullanıcı objesinde bir araya getirilmesi gerekmektedir. Bu durumu gerçekleştirmek için Tablo 76'da ifade edildiği üzere istediğimiz ad ile tanımladığımız değişken içeresine “new” anahtar kelimesi ile yeni bir obje oluştururuz ve bu obje içeresine gerekli bilgileri gireriz. Gerekli bilgileri Algosdk paketi içerisinde bulunan “Algody2” metodu ile elde edebiliriz. Metot içeresine önceden tanımladığımız değişkenler belirli sıra ile eklenmelidir. Aksi taktirde fonksiyon istenilen şekilde çalışmazacaktır.

```
var beren_mnemonic = "copy deny kingdom faith need blossom anchor zoo address
venue carry rain winter salt rapid infant zero address laptop govern scout rice escape
absorb stove";
var berenHesap = Algosdk.mnemonicToSecretKey(beren_mnemonic);
```

Tablo 77

Token ekleme görevi için token eklemek istediğimiz hesabın bilgileri gerekmektedir. Daha önce de belirttiğimiz üzere Mnemonic anahtarlar hesabı nitelendiren şifreler olmaktadır. Bu şifreler tek başına insan tarafından algılanabilir olsa da zincir için anlamsızdır. Bu nedenle Mnemonic anahtarının SecretKey değerine çevrilmesi gerekmektedir. Bu nedenle Tablo 77'de ifade edildiği üzere ilk olarak mnemonic anahtar bir değişken içerisinde kaydedilir. Sonrasında kaydedilen bu değişken Algosdk içerisindeki “mne-

monicToSecretKey()" metodu içerisinde eklenecek hesap oluşturulur. Hesap içerisinde Adresi, Secret Key değerini ve mnemonic anahtarları tutmaktadır. Bu hesap kullanılarak işlemler yapılacaktır.

```
(async () => {  
})
```

Tablo 78

Gerekli tanımlamaların yapılması sonrasında eklenecek Token varlığının değerlerini içerecek fonksiyonun tanımlanması gerekmektedir. Bu kısımda zincir ile iletişime geçilecektir. Bu nedenle sisteme olusabilecek beklemeleri karşılamak amacıyla asenkron fonksiyon tanımlanmaktadır. Bu fonksiyon Tablo 78'de ifade edildiği üzere “async” anahtar kelimesi ile tanımlanmakta olup içerisinde herhangi bir parametre istememektedir. Ayrıca herhangi bir tanımlama ismi kullanılmadığı için fonksiyon çağrııldığı zaman değil çalışma satırı ulaştığı an çalışacaktır.

```
let params = await kullanıcı.getTransactionParams().do();  
let assetID = 112384980;  
let sender = berenHesap.addr;  
let recipient = sender;  
let revocationTarget = undefined;  
let closeRemainderTo = undefined;  
let note = undefined;  
let amount = 0;
```

Tablo 79

Asenkron fonksiyon içerisinde girildiğinde bazı tanımlamaların yapılması gerekmektedir. Token ekleme yani “Opt-in” işlemi aynı zamanda bir transfer işlemi olduğu için bazı transfer parametrelerinin eklenmesi de gerekmektedir. Tablo 79'da ifade edilen bu parametrelere daha yakından bakılacak olunursa:

- **Params:** Kullanıcı olarak tanımladığımız zincir üzerinde işlemler yapabilen değişkenin transferler ile ilgili parametrelerini almaktadır. Bu bilgileri zincir üzerinden elde etmesinden dolayı zaman almaktadır bu nedenle “await” anahtar kelimesi ile çağrılmaktadır. Kullanıcı içerisinde bir metot çağrılmak üzere “.do()” metodu ile fonksiyon çalıştırılır ve gerekli değişkenler istenilen isim ile kaydedilen değişken içerişine eklenir.

- **AssetID:** Eklenmek istenilen Token’ın zincir üzerinde tanımlandığı numarayı ifade etmektedir. Bu numara Token’ı oluşturan hesap tarafından belirlenmemiş olunup otomatik olarak zincir tarafından tanımlanmıştır. Her Token’ın kendine özel bir “AssetID” değeri bulunmaktadır. Integer değeri almaktadır. İstenilen isim ile kaydedilebilir.
- **Sender:** Eklenilecek hesabı ifade etmektedir. Token ekleme işleminin aslında bir transfer olması nedeniyle tanımlanır. Eklenen Token’ı hesabın kendi kendisine göndermesi gibi düşünülebilir.
- **Recipient:** Token ekleme işleminin hangi adresе yapılacağını ifade eder. İçerisine bir önceki tanımladığımız “Sender” değişkenini alır. Kendi kendine gönderme işlemini tamamlar. İstenilen isim ile kaydedilebilir.
- **RevocationTarget:** Beklenmedik bir olay karşısında Tokenların gitmesini istediğimiz hesapları ifade etmektedir. Kodumuz içerisinde “undefined” yani tanımlanmamış olarak bırakılacaktır. İstenilen isim ile kaydedilebilir.
- **CloseRemainderTo:** Bazı işlemlerde olması muhtemel kalan değerlerin transfer edilmesi istenilen adresi ifade etmektedir. Kodumuz içerisinde “undefined” yani tanımlanmamış olarak bırakılacaktır. İstenilen isim ile kaydedilebilir.
- **Note:** Transfer işlemi yapılrken tanımlanmak istenilen notu ifade etmektedir. Zincir içerisinde kaydedilebilmek için doğru bir şekilde “encode” fonksiyonlarına sokulup şifrelenmelidir. Simdilik “undefined” yani tanımsız olarak bırakılacaktır.
- **Amount:** Kaç varlık ekleneceğini ifade etmektedir. Token’dan kaç birim ekleneceğini ifade etmemektedir. Integer değer almaktadır. Harici bir varlık eklemememiz dolayısıyla 0 değeri girilecektir.

```
let txn = Algosdk.makeAssetTransferTxnWithSuggestedParams(sender, recipient,
closeRemainderTo , revocationTarget,
amount, note, assetID, params);
```

Tablo 80

Token eklenmesi için eklenen gerekli parametreler sonrasında bu değişkenlerin bir transfer içerisinde buluşması gerekmektedir. Bu nedenle Tablo 80’de ifade edildiği üzere istediğimiz isimlendirme ile tanımladığımız fonksiyon içerisinde Algosdk paketi içerisinde bulunan “makeAssetTransferTxnWithSuggestedParams()” metodu girilerek bu tanımlama yapılabilir. Fonksiyon içerisinde daha öncesinde tanımladığımız transfer parametrelerini almaktadır. Bu parametrelerin sırası önemli olmaktadır. Yanlış sıralama girilmesi halinde kod istenilen şekilde çalışmayacaktır.

```
let rawSignedTxn = txn.signTxn(berenHesap.sk)
let tx = (await kullanıcı.sendRawTransaction(rawSignedTxn).do());
console.log("Transfer : " + tx.txId);
```

Tablo 81

Transferin oluşturulması sonrasında bu transferin imzalanması ve zincire gönderilmesi gerekmektedir. Bu işlem için Tablo 81'de ifade edildiği üzere ilk olarak transfer içerisinde bulunan “signTxn()” metodu içerisinde Token'in ekleneceği hesap olan Beren hesabının Secret Key değerini alarak imzalı bir değişken oluşturulmalıdır. Sonrasında bu işlem asenkron olarak “await” anahtar kelimesi ile kullanıcı objesinde bulunan “sendRawTransaction()” metodu sayesinde zincire gönderilir. Obje içerisinde bulunan bir metoda erişim sağlanmasından ötürü “.do()” metodu kullanılmaktadır. Son olarak bu transfer ekrana “console.log()” fonksiyonu ile yazdırılmaktadır.

```
O.catch(e => {
  console.log(e);
});
```

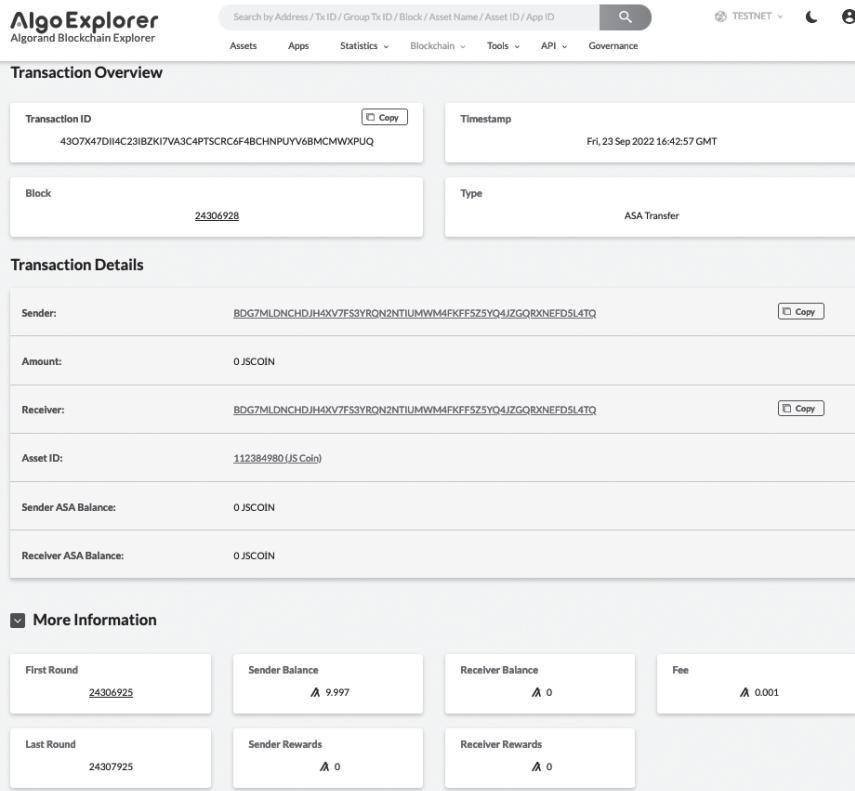
Tablo 82

Asenkron fonksiyon içerisinde çıkışması sonrasında oluşacak muhtemel sorunları çözmek için “.catch()” fonksiyonu kullanılmaktadır. Bulunan hatalar Tablo 82'de ifade edildiği üzere “console.log()” ile ekrana yazdırılacaktır.

```
Transfer : 43O7X47DII4C23IBZKI7VA3C4PTSRC6F4BCHNPUYV6BMCMWXPUQ
```

Tablo 83 beren_opt_in.js Kod Çıktısı

Yazdığımız kod tamamlandıktan sonra bulunduğu klasörde terminal aracılığı ile “node beren_opt_in.js” komutu ile çalıştırılabilir. Çalışma sonrasında terminalde Tablo 83'te ifade edilen çıktı gibi bir çıktı ile karşılaşacaktır. Bu çıktı oluşan transferin adresini ifade etmektedir ve AlgoExplorer gibi sitelerde kontrol edilebilir.



The screenshot shows the AlgoExplorer Transaction Overview page for a specific transaction. The transaction details are as follows:

- Transaction ID:** 4307147D14C23BZK17VA3C4PTSCRC6F4BCHNPUVY68MCMVXPUQ
- Timestamp:** Fri, 23 Sep 2022 16:42:57 GMT
- Block:** 24306928
- Type:** ASA Transfer
- Sender:** BDG7MLDNCHDJH4XV7FS3YRQN2NTIUMWM4FKFF5Z5Y04JZGQRXNEFD5L4TQ
- Amount:** 0 JS COIN
- Receiver:** BDG7MLDNCHDJH4XV7FS3YRQN2NTIUMWM4FKFF5Z5Y04JZGQRXNEFD5L4TQ
- Asset ID:** 112384980 (US Coin)
- Sender ASA Balance:** 0 JS COIN
- Receiver ASA Balance:** 0 JS COIN
- First Round:** 24306925
- Sender Balance:** ₣ 9.997
- Receiver Balance:** ₣ 0
- Fee:** ₣ 0.001
- Last Round:** 24307925
- Sender Rewards:** ₣ 0
- Receiver Rewards:** ₣ 0

Şekil 25

Şekil 25'te ifade edildiği üzere Algoexplorer'da transfer incelenebilir. Bu transfer içerisinde parametre olarak girdiğimiz birçok değer bulunmaktadır.

```

const Algosdk = require('algosdk');
require("dotenv").config()

const server="https://testnet-algorand.api.purestake.io/ps2";
const port="";
const token={"x-api-key": process.env.API};
let kullanıcı = new Algosdk.Algodv2(token, server, port);

var beren_mnemonic = "copy deny kingdom faith need blossom anchor zoo address
venue carry rain winter salt rapid infant zero address laptop govern scout rice escape
absorb stove";
var berenHesap = Algosdk.mnemonicToSecretKey(beren_mnemonic);

(async () => {
let assetID = 112384980;
let params = await kullanıcı.getTransactionParams().do();
let sender = berenHesap.addr;
let recipient = sender;
let revocationTarget = undefined;
let closeRemainderTo = undefined;
let note = undefined;
let amount = 0;

let txn = Algosdk.makeAssetTransferTxnWithSuggestedParams(sender, recipient,
closeRemainderTo, revocationTarget,
amount, note, assetID, params);

let rawSignedTxn = txn.signTxn(berenHesap.sk)
let tx = (await kullanıcı.sendRawTransaction(rawSignedTxn).do());
console.log("Transfer : " + tx.txId);
})().catch(e => {
  console.log(e);
});
});

```

Tablo 84 beren_opt_in.js Kodu Tamamı

Tablo 84'te "Opt-in" işlemini gerçekleştirmek için gerekli olan tüm kod satırları bütün olarak verilmiştir. Kodun işlevine parça parça bakılacak olunursa:

- İlk olarak gerekli paketler kod içerisinde kaydedilmektedir.
- Sonrasında zincir üzerinde işlem yapılabilmesini sağlayan server, token ve port değişkenleri tanımlanır.
- Tanımlanan işlem parametreleri yeni bir kullanıcı objesinde bir araya getirilir.
- Token'ın ekleneceği hesap mnemonic şifresinden türetilerek kullanıma hazır hale getirilir.
- Asenkron fonksiyon tanımı yapılır.

- Eklenecek Token'ın değerleri değişkenlere kaydedilir. Bu değerlerden biri de Token'ı zincir üzerinde tanımlayan “AssetID” değeri olmaktadır.
- Tanımlanan değişkenler bir transfer olarak kaydedilmek üzere txn değişkenine SDK fonksiyonu tarafından kaydedilir.
- Yapılan transfer eklenecek hesap üzerinde bulunan Secret Key kullanılarak imzalanır.
- İmzalanan transfer kullanıcı tarafından zincire gönderilir ve “transferId” terminale yazdırılır.
- Olası hatalar “.catch()” metodu kullanılarak kaydedilir.

Token Bakiye Kontrolü

Önceki bölümlerde nasıl Algo bakiyesine bakılacağı ifade edilmişti. Ancak bu bakiye sadece Algorand'ın kendi coin'i için çalışmaktadır. Bir he-sapta hangi Tokenların ve varlıkların bulunduğu kontrol etmek için Algosigner'i veya JavaScript ile yazacağımız kod mekanizmaları kullanılabilir.

Account	Algo Balance	Transactions
Beren	9.997 Algos	ASAS FROM ASA TO ASA PAYMENT
Irem	19.999 Algos	ASAS FROM ASA TO ASA PAYMENT
Irem	19.997 Algos	ASAS FROM ASA TO ASA PAYMENT

Şekil 26

Sorgu işlemini AlgoSigner üzerinde gerçekleştirmek istememiz durumunda Şekil 26'da ifade edildiği üzere Testnet'de olduğumuza emin olarak bu işlemi gerçekleştirebiliriz. Assets başlığı altında hesabın sahip olduğu bütün Tokenlar ve varlıklar gözükmek olacaktır.

Bu işlemi JavaScript ve Algosdk ile de gerçekleştirmek mümkündür. İlk olarak index.js dosyasının bulunduğu dosyaya giriş yaparak belirlediğiniz isme sahip bir JavaScript dosyası oluşturulması gerekmektedir. Anlatım kolaylığı açısından aynı anda hem İrem hem de Beren hesabının kontrolünün sağlayacak kod dosyasına ikili_kontrol.js ismi verilmiştir.

```
const Algosdk=require('algosdk');
require("dotenv").config()
```

Tablo 85

İlk olarak diğer JavaScript dosyalarına başlandığı gibi kütüphaneleri ekleyerek başlamamız gerekmektedir. Bu paketleri Tablo 85'te ifade edildiği üzere “require” yapısı ile eklemektedir. Eklenen paketlere bakılırsa:

- **Algosdk:** Algorand blokzinciri ile bağlantı kurulmasını sağlayan kütüphane olmaktadır. İstenilen isim ile kaydedilebilir.
- **Dotenv:** Saklanması istedigimiz paylaşmak istemedigimiz belgeleerin bulunduğu “.env” dosyası içerisinde verilere erişim yapılmasını sağlayan kütüphanedir. Herhangi bir değişken ile çağrılmaması gerekmektedir. Çağırıldıkten sonra “.config()” yapısı ile kurulumu yapılmaktadır.

```
const server="https://testnet-algorand.api.purestake.io/ps2";
const port="";
const token={"x-api-key": process.env.API};
```

Tablo 86

Paketlerin çağrılmamasından sonra zincir üzerinde işlem yapılması için gerekli parametrelerin tanımlanması gerekmektedir. Tablo 86'da ifade edildiği üzere bu değişkenler “const” yapısı ile sabit olarak tanımlanabilirler. Her bir parametrenin yapıtlarına ayrıntılı bakılırsa:

- **Server:** Sorgulama işleminin hangi zincir üzerinde gerçekleşeceğini ifade etmektedir. Purestake sitesinden alınmaktadır ve mainnet üzerinde işlem yapılmak istenilirse farklı bir adres kullanılmalıdır. String değeri kabul etmektedir.
- **Port:** Bağlantının yapılacak portu ifade etmektedir. Bu değişken değerini değiştirerek zincir üzerinde bulunan farklı bir düğümden işlem yapmak mümkündür. Simdilik kod içerisinde boş bırakılacaktır.
- **Token:** Purestake üzerinden alınan API anahtarını ifade etmektedir. API anahtarları zincir üzerinde yapılan sorgu, veri gönderme gibi işlemleri kullanıcıya bağlamaktadır. İçerisine obje girdisi istemektedir. Atanırken “x-api-key” anahtarı ile kaydedilmektedir. API anahtarı güvenli tutulması gereken bir anahtardır. Bu nedenle gizli tutulmalıdır. Bu işlemi gerçekleştirmek için “.env” dosyası içerisinde tutulan API anahtarını “process.env.API” fonksiyonu ile çağırmak mümkündür.

```
let kullanıcı = new Algosdk.Algodv2(token, server, port);
```

Tablo 87

Parametreler tanımlandıktan sonra bir obje içerisinde ek özellikler ile kaydedilmelidir. Bu nedenle kullanıcı değişkeni içerisinde kaydedilen ve “new” anahtar kelimesi ile oluşturulan bir obje yaratmaktadır. Bu obje Algosdk paketi içerisindeki “Algodv2” metodunu Tablo 87’de ifade edildiği şekilde kullanarak içerisinde girilen token, server ve port değişkenleri ile bir obje oluşturur ve değişkene kaydeder.

```
var iremAddress =
'IOIBYZPBAISY636GLWF2PRFXZCPVN2DKXXCCOUCANPA2BROUKV2UTOPVYA';
var berenAddress =
'BDG7MLDNCHDJH4XV7FS3YRQN2NTIUMWM4FKFF5Z5YQ4JZGQRXNEFD5L4TQ';
```

Tablo 88

Kullanıcı tanımlanması sonrasında Tablo 88’de ifade edildiği üzere Token varlıklarını kontrol edecek hesapların adresleri gerekmektedir. Bu adresler herkese açık adresler olmaktadır. Bu durumun nedeni bakiye ve varlık sorgulama işlemlerinin herkes tarafından sorgulanmaya açık olmasıdır. Adrese sahip olan bir birey tüm kontrolleri gerçekleştirebilir.

```
( async() => {
  })
})
```

Tablo 89

Sorgu işlemleri zincire bağlantı gerektirmektedir. Bu nedenle zaman alabilir. Bu durumda Tablo 89’da ifade edildiği üzere JavaScript içerisinde asenkron bir fonksiyon tanımlanmalıdır. Bu fonksiyon içerisinde herhangi bir parametre almamaktadır. Ayrıca herhangi bir tanımlayıcı değişkene ihtiyaç duymamaktadır. Bu nedenle çağrılmayı beklemeden çalışma satırı ulaşır ulaşmaz çalışır ve içerisindeki işlemleri asenkron olarak gerçekleştirir. Tanımlanması için “async” anahtarını gerekmektedir.

```
let irem_hesap_bilgisi = (await kullanıcı.accountInformation(iremAddress).do());
console.log("İrem Hesabı Varlıklar: " + irem_hesap_bilgisi.assets);

let beren_hesap_bilgisi = (await kullanıcı.accountInformation(berenAddress).do());
console.log("Beren Hesabı Varlıklar: " + beren_hesap_bilgisi.assets);
```

Tablo 90

Asenkron fonksiyonun içerisinde sorgu işlemleri yapılacaktır. Sorgu işlemleri için istenilen isim ile kaydedilen değişkenler oluşturulmaktadır. Bu işlem zincire bağlantı gerektirmektedir. Bu nedenle asenkron yapıda “await” anahtar kelimesi ile kodun bu satırda durması gerektiğini belirtmekteyiz. Sonrasında Tablo 90’da ifade edildiği üzere oluşturulan kullanıcı sayesinde “accountInformation()” metoduna erişilir. Bu metod içerisinde sorgulanmak istenilen adresi string olarak almaktadır. Obje içerisinde bir metod çağrılmamasından dolayı “.do()” anahtar kelimesi ile bu fonksiyonun çalıştırılması gerekmektedir. Fonksiyonun çalışması sonrasında kaydedilen değişken uygun etiketleme ile “console.log()” yapısı kullanılarak terminale yazdırılmaktadır. Bu işlem her iki hesap için de ayrı ayrı yapılmaktadır.

```
0.catch(e => {
    console.log(e);
})
```

Tablo 91

Asenkron fonksiyon içerisinde çıktıktan sonra olası hataların yakalanması için Tablo 91’de gösterildiği şekilde “.catch()” fonksiyonu ile bu işlemi gerçekleştirebiliriz. Yakalanan hatalar “console.log()” ile terminale yazdırılacaktır.

İrem Hesabı Varlıklar: [{ amount: 1000000, 'asset-id': 112384980, 'is-frozen': false }]
Beren Hesabı Varlıklar: [{ amount: 0, 'asset-id': 112384980, 'is-frozen': false }]

Tablo 92 ikili_kontrol.js Çıktı

Kodumuz tamamlandıktan sonra bulunduğu klasörde terminalden “node ikili_kontrol.js” komutu ile çalıştırılabilir. Çıktı Tablo 92’de ifade edildiği gibi olacaktır. Bazı durumlarda zincir ile bağlantı sorunları yaşanmasından dolayı birkaç kez çalışma gereklili olabilir. Görüldüğü üzere çıktı liste içerisinde bulunan objelerden oluşmaktadır. Her iki hesapta da tek varlık bulunduğu için listelerinde birer obje bulunmaktadır. Bu objeler içerisinde 3 adet anahtar taşımaktadır. Bu anahtarlarla sırası ile bakılması gerekirse:

- **Amount:** Token’ın miktarını ifade etmektedir.
- **Asset-id:** Varlığın zincir üzerinde tanımlanlığı numarayı ifade etmektedir.
- **Is-frozen:** Varlığın hesapta dondurulmuş olup olmadığını ifade etmektedir. Bool değerler almaktadır.

```

const Algosdk=require('algosdk');
require("dotenv").config()

const server="https://testnet-algorand.api.purestake.io/ps2";
const port="";
const token={"x-api-key": process.env.API};
let kullanici = new Algosdk.Algodv2(token, server, port);

var iremAddress =
'IOIBYZPBAISY636GLWF2PRFXZCPVN2DKXKCCOUCANPA2BROUKV2UTOPVYA';
var berenAddress =
'BDG7MLDNCHDJH4XV7FS3YRQN2NTIUMWM4FKFF5Z5YQ4JZGQRXNEFD5L4TQ';

(async() => {
  let irem_hesap_bilgisi = (await kullanici.accountInformation(iremAddress).do());
  console.log("İrem Hesabı Varlıklar: " , irem_hesap_bilgisi.assets);

  let beren_hesap_bilgisi = (await kullanici.accountInformation(berenAddress).do());
  console.log("Beren Hesabı Varlıklar: " , beren_hesap_bilgisi.assets);

})().catch(e => {
  console.log(e);
})

```

Tablo 93 ikili_kontrol.js Kodu Tamamı

Tablo 93'te yazdığımız kodun tamamı ifade edilmektedir. Bu kodun işlevlerine parça parça bakılması gerekirse:

- İlk olarak kütüphaneler tanımlanmaktadır.
- Zincir üzerinde işlem yapmamızı sağlayan parametreler tanımlanmaktadır.
- Bu parametreler yeni bir obje oluşturularak içerisinde farklı özellikler ile eklenmektedir.
- Sorgu yapılacak adresler tanımlanmaktadır.
- Asenkron işlemler gerçekleştireceğimiz için `async` anahtar kelimesi ile asenkron fonksiyon tanımlanmaktadır.
- Bu fonksiyonun içerisinde kullanıcı metodu ile gerekli adres sorguları gerçekleştirilir.
- Kaydedilen sorgu değişkenleri terminale yazdırılır.
- Asenkron fonksiyonda oluşabilecek olası hatalar `".catch()` yapısı ile ekrana yazdırılır.

Adresler Arasında Varlık Transferi

Önceki bölümlerde nasıl bakiye transferi yapılacağını gözlemlemiştik. Ancak varlık transferleri daha farklı yapılmaktadır. Bu işlemi gerçekleştirmek için Algosigner ve JavaScript dili üzerindeki Algosdk paketi kullanılabilir.

JavaScript üzerinden bu işlem gerçekleştirilmek istendiğinde ilk olarak index.js dosyasının bulunduğu klasöre yeni bir JavaScript dosyasının oluşturulması gerekmektedir. Bu belge istenilen isme sahip olabilir ancak anlatım kolaylığı açısından şimdilik “token_transfer.js” ismi tercih edilmiştir.

```
const Algosdk=require('algosdk');
require("dotenv").config()
```

Tablo 94

Her JavaScript dosyasına başlanıldığı gibi ilk olarak dosya içeresine paket tanımlarını yaparak başlanması gerekmektedir. Bu paketler Tablo 94’te ifade edildiği üzere JavaScript içerisinde bulunan “require” fonksiyonu ile eklenmektedir. Sırası ile paketlerin işlevlerine bakılması gerekirse:

- **Algosdk:** Algorand blokzinciri ile bağlantı yapılmasını sağlamakta-dır. İstenilen isim ile kaydedilebilir.
- **Dotenv:** Paylaşılması riskli olan bilgilerin kod içeresine güvenli bir şekilde eklenmesini sağlayan pakettir. Bu paket sayesinde klasörde oluşturduğumuz ve paylaşılarda transfer olmayan “.env” dosyası içeresine yazdığımız bilgileri dosyamıza ekleyebiliriz.

```
const server="https://testnet-algorand.api.purestake.io/ps2";
const port="";
const token={"x-api-key": process.env.API};
```

Tablo 95

Paket tanımlama sonrasında zincirde işlem yapılabilmesi için Tablo 95’de ifade edildiği üzere bazı parametrelerin tanımlanması gerekmektedir. Bu parametrelere zinciri ve işlemi yapan kişiyi tanımlamaktadır. Bu parametrelere ayrıntılı olarak bakılması gerekirse:

- **Server:** İşlemin gerçekleşmesini istediğimiz ağı ifade etmektedir. Bu işlemin test serverında gerçekleşmesini istediğimizden ötürü “Pures-
take” sitesinde bulabileceğimiz zincir adresini yazmaktayız.
- **Port:** İşlemin gerçekleşeceği bağlantı portunu ifade etmektedir. Kul-
lanım amacımızdan dolayı boş olarak bırakılabilir.
- **Token:** API anahtarımızı ifade etmektedir. Purestake sitesinden al-
diğimiz API anahtarını güvenlik nedenleriyle “.env” klasörü içerisinde-
den “process.env” metodu ile erişilmektedir.

```
let kullanıcı = new Algosdk.Algodv2(token, server, port);
```

Tablo 96

Zinciri ve bireyi tanımlayacak parametrelerin tanımlanması sonrasında kullanıcı adı ile kaydedilen değişenin Tablo 96'da gösterildiği üzere bir objeye eşitlenmesi gerekmektedir. Bu işlem için “new” anahtar kelimesi ile oluşturulan değişkenin içerisinde Algosdk içerisinde bulunan Algodv2 metodu çağırılır. Bu metot içerisinde önceden tanımladığımız token, server ve port parametrelerini almaktadır. Bu parametrelerin sırası büyük önem arz etmektedir. Aksi halde fonksiyon istenilen şekilde çalışmamayacaktır.

```
var irem_mnemonic = "ramp later certain business possible memory cupboard armed truly learn
spring you vague vintage nut series curious lonely emerge width patch turtle wool able rice";
var iremHesabı = Algosdk.mnemonicToSecretKey(irem_mnemonic);
```

Tablo 97

Kullanıcının tanımlanmasından sonra transferi yapacak kişinin bilgileri gerekmektedir. Transfer iki hesap arasında da yapılmaktadır ancak henüz Beren hesabı içerisinde bir bakiye bulunmadığı için transferi İrem Hesabından Beren hesabına doğru gerçekleştireceğiz. Bakiye azalması İrem hesabında olacağı için bu hesabın Secret Key verisine ihtiyaç bulunmaktadır. Tablo 97'da ifade edildiği üzere ilk olarak elimizde bulunan mnemonic değeri bir değişken içerisinde kaydederiz. Bu değer insanlar tarafından okunması kolay olsa da zincir tarafından anlamsızdır. Bu nedenle bu mnemonic şifreden Secret Key değeri türetilmelidir. Bu işlemi Algosdk paketi içerisinde bulunan “mnemonicToSecretKey” metodu ile gerçekleştirebiliriz. Bu metot içerisinde transferin yapılacak hesabın mnemonic anahtarını string olarak almaktadır.

```
var berenAdres = 'BDG7MLDNCHDJH4XV7FS3YRQN2NTIUMWM4FKFF5Z5YQ4JZGQRXNEFD5L4TQ';
```

Tablo 98

Transferler iki taraflı olmaktadır. Bir gönderici karşısında her zaman bir alıcı da bulunmalıdır. Örneğimizde alıcı taraf Beren Hesabı olduğu için bu hesabın adresi sonradan kullanılmak üzer Tablo 98'da ifade edildiği üzere kaydedilmelidir.

```
(async () => {
  })
```

Tablo 99

Transferi gerçekleştirmek için gerekli bazı parametrelerde zincire bağlantı söz konusu olmaktadır. Bu bağlantı zaman alabilmektedir. Bu nedenle bir asenkron yapı kullanmamız gerekmektedir. JavaScript üzerinde asenkron işlemler yaparken Tablo 99'da ifade edildiği üzere “async” yapısını kullanmaktadır. Fonksiyonumuz komut sırası geldiğinde direkt olarak çalışacağından dolayı herhangi bir tanımlayıcıya ihtiyaç duymamaktadır. Ayrıca herhangi bir parametre gerektirmemektedir.

```
let assetID = 112384980;
let params = await kullanıcı.getTransactionParams().do();
let sender = iremHesabı.addr;
let recipient = berenAdres;
let revocationTarget = undefined;
let closeRemainderTo = undefined;
let note = undefined;
let amount = 200000;
```

Tablo 100

Transferin özellikleri bazı değişkenler ile tanımlanmalıdır. Tablo 100'da ifade edilen şekilde her biri istenilen isim ile kaydedilmektedir ve farklı özellikler taşımaktadır. Bu özellikler Beren hesabına Token eklenirken kullanılan parametrelere benzerlik göstermektedir. Değişkenlerin her birine bakılacak olunursa:

- **AssetID:** Hangi varlığın transfer edileceğini ifade etmektedir. Bu değer varlıkların birbirinden ayırmaktadır ve zincir tarafından atanmaktadır. Token oluşturucusunun bu numara üzerinde değişiklik yapma hakkı bulunmamaktadır.
- **Params:** Kullanıcı olarak tanımladığımız zincir üzerinde işlemler yapabilen değişkenin transferler ile ilgili parametrelerini almaktadır. Bu bilgileri zincir üzerinden elde etmesinden dolayı zaman almaktadır bu nedenle await anahtar kelimesi ile çağrılmaktadır. Kullanıcı içerisinde bir metot çağırılacağı için “.do()” metodu ile fonksiyon çalıştırılır ve gerekli değişkenler istenilen isim ile kaydedilen değişken içerişine eklenir.
- **Sender:** Miktarın hangi hesap üzerinden gönderileceğini ifade etmektedir. İrem hesabından Beren hesabına doğru bir transfer yapılacağı için “iremHesabı” olarak oluşturduğumuz değişken içerisinde “addr” anahtar kelimesine sahip adresi tanımlamamız gerekmektedir.
- **Recipient:** Token ekleme işleminin hangi adrese yapılacağını ifade eder. İçerisine bir önceki tanımladığımız “Sender” değişkenini alır. Kendi kendine gönderme işlemini tamamlar. İstenilen isim ile kaydedilebilir.

- **RevocationTarget:** Beklenmedik bir olay karşısında Tokenların gitmesini istediğimiz hesapları ifade etmektedir. Kodumuz içerisinde “undefined” yani tanımlanmamış olarak bırakılacaktır. İstenilen isim ile kaydedilebilir.
- **CloseRemainderTo:** Bazı işlemlerde oluşması muhtemel kalan değerlerin transfer edilmesi istenilen adresi ifade etmektedir. Kodumuz içerisinde “undefined” yani tanımlanmamış olarak bırakılacaktır. İstenilen isim ile kaydedilebilir.
- **Note:** Transfer işlemi yapılrken tanımlanmak istenilen notu ifade etmektedir. Zincir içerisinde kaydedilebilmek için doğru bir şekilde “encode” fonksiyonlarına sokulup şifrelenmelidir. Simdilik “undefined” yani tanımsız olarak bırakılacaktır.
- **Amount:** Transfer yapılması planlanan miktarı ifade etmektedir. Integer değeri kabul etmektedir. Projemizde 200.000 Token transferi yapmayı istememizden dolayı 2000000 değeri girilebilir.

```
let txn = Algosdk.makeAssetTransferTxnWithSuggestedParams(sender, recipient,
closeRemainderTo, revocationTarget,
amount, note, assetID, params);
```

Tablo 101

Tanımlanan transfer parametrelerinin ham bir transfer olarak birleştirilmesi gerekmektedir. Bu işlemi gerçekleştirmek üzere Tablo 101’de ifade edildiği üzere Algosdk paketi içerisinde bulunan “makeAssetTransferTxnWithSuggestedParams()” metodu kullanılmaktadır. Bu metot içerisinde daha önceden tanımlanan transfer parametrelerini almaktadır. Bu parametrelerin sıralaması önemlidir ve doğru şekilde sıralanmaması halinde istenilen şekilde çalışmamayacaktır.

```
let rawSignedTxn = txn.signTxn(iremHesabi.sk);
```

Tablo 102

Transfer kaydedildi ancak İrem hesabı ile henüz imzalanmadı. İmzalama işlemini gerçekleştirmek için Tablo 102’de ifade edildiği üzere daha öncesinde kaydettiğimiz transferin “signTxn()” metodu ile imzalanması gerekmektedir. Bu fonksiyon içerisinde daha öncesinde mnemonic anahtar içerisinde türetilen iremHesap.sk değerini almaktadır.

```
let tx = (await kullanıcı.sendRawTransaction(rawSignedTxn).do());
console.log("Transfer: : " + tx.txId);
```

Tablo 103

İmzalanan transfer gönderime hazır hale gelmektedir. Gönderim işleminin yapılması için Tablo 103'da ifade edildiği gibi asenkron olarak await anahtar kelimesi ile kullanıcı "sendRawTransaction()" metodunu kullanarak zincire gönderim yapmaktadır. Metodun çalıştırılması gerektiği için sonuna ".do()" metodu eklenmelidir. Zincire gönderilen transfer geri döngüt olarak bir transfer numarası döndürmektedir. Bu değeri console.log ile terminale yazdırabiliriz.

```
 0.catch(e => {  
  console.log(e);  
});
```

Tablo 104

Transfer bilgilerini içeren asenkron fonksiyondan çıktıktan sonra bu fonksiyonda oluşabilecek olası sorunların kullanıcıya ifade edilmesi gerekmektedir. Bu nedenle Tablo 104'de ifade edildiği üzere ".catch()" yapısı ile olası sorunlar tespit edilir ve "console.log()" ile terminal üzerine yazdırılır.

```

const Algosdk=require('algosdk');
require("dotenv").config()

const server="https://testnet-algorand.api.purestake.io/ps2";
const port="";
const token={"x-api-key": process.env.API};
let kullanici = new Algosdk.Algodv2(token, server, port);

var irem_mnemonic = "ramp later certain business possible memory cupboard armed truly learn
spring you vague vintage nut series curious lonely emerge width patch turtle wool able rice";
var iremHesabi = Algosdk.mnemonicToSecretKey(irem_mnemonic);

var berenAdres =
'BDG7MLDNCHDJH4XV7FS3YRQN2NTIUMWM4FKFF5Z5YQ4JZGQRXNEFD5L4TQ';

(async () => {
let assetID = 112384980;
let params = await kullanici.getTransactionParams().do();
let sender = iremHesabi.addr;
let recipient = berenAdres;
let revocationTarget = undefined;
let closeRemainderTo = undefined;
let note = undefined;
let amount = 200000;

let txn = Algosdk.makeAssetTransferTxnWithSuggestedParams(sender, recipient,
closeRemainderTo, revocationTarget,
amount, note, assetID, params);

let rawSignedTxn = txn.signTxn(iremHesabi.sk)
let tx = (await kullanici.sendRawTransaction(rawSignedTxn).do());
console.log("Transfer : " + tx.txId);
})().catch(e => {
console.log(e);
});

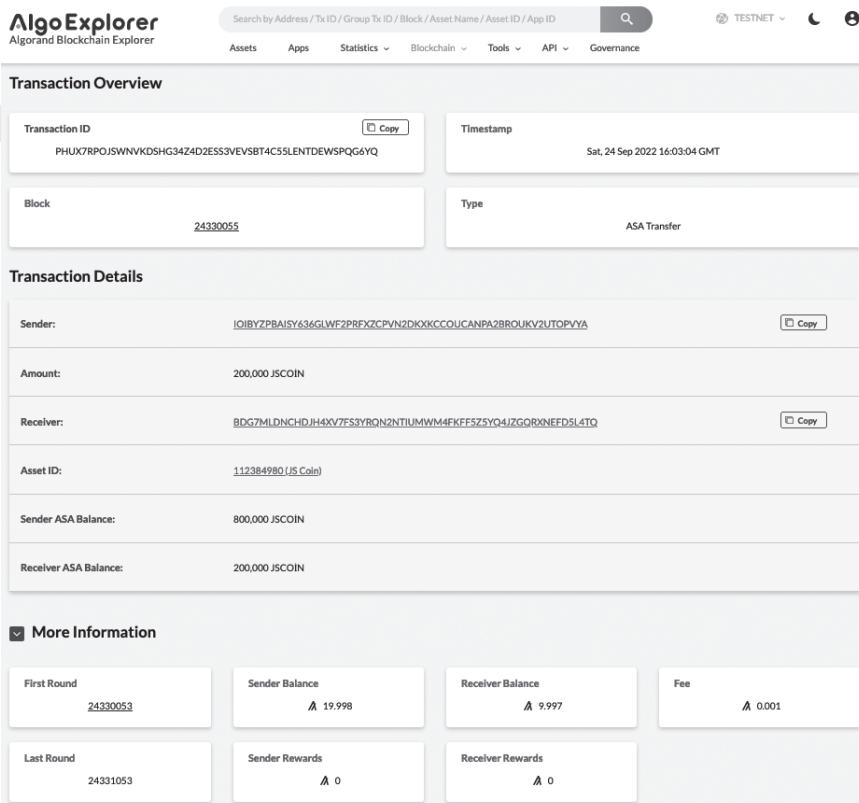
```

Tablo 105 token_transfer.js Kodunun Tamamı

Tablo 105'te transfer kodunun tamamı ifade edilmiştir. Bu koda parça parça bakılması gereklidir:

- İlk olarak kullanılacak paketler tanımlanmaktadır.
- Zincir üzerinde bilgi elde edinmeyi sağlayan API anahtarı, zincir adresi ve port değerleri değişkenlere kaydedilir.
- Kaydedilen zincir parametreleri Algorand SDK ile birleştirilir ve ek özellikler ile obje olarak kullanıcı değişkeninde kaydedilir.
- Gönderici hesabının mnemonic şifresi Algosdk kullanılarak hesabının kendisine çevrilir.
- Alıcı hesabının adresi kaydedilir.
- Asenkron fonksiyon tanımlanır.
- Fonksiyon içerişine token adresi ve miktarı gibi birçok değer istenilen isim ile değişkenlere kaydedilir.

- Transfer “txn” adında ham bir şekilde transfer olarak kaydedilir.
- Transfer gönderici hesabın Secret Key değeri tarafından imzalanır.
- İmzalanan transfer kullanıcı tarafından zincire gönderilir ve zincirden cevap alır.
- Transfer numarası terminale yazdırılır.



The screenshot shows the AlgoExplorer interface with the following details:

Transaction Overview

Transaction ID	PHUX7RPOJSWNVKDSHG34Z4D2ESS3VEVSBT4C55LENTDEWPSPQG6YQ	Copy	Timestamp	Sat, 24 Sep 2022 16:03:04 GMT
Block	24330055		Type	ASA Transfer

Transaction Details

Sender:	1QIBY7PBAISY636GLWF2PRFXZCPV/2DIXKCCOLICANPA2BROUKV2UTOPV/YA	Copy
Amount:	200.000 JSCOIN	
Receiver:	BDG7MLDNCHDjh4Xv7FS3YRQN2NTIUMWM4FKFF5Z5YQ4/ZGORXNEFD5l4TQ	Copy
Asset ID:	112384980 (5 Coin)	
Sender ASA Balance:	800.000 JSCOIN	
Receiver ASA Balance:	200.000 JSCOIN	

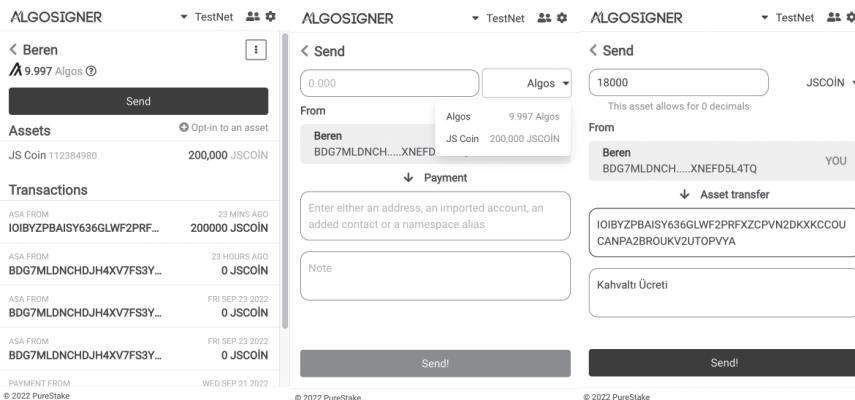
More Information

First Round 24330053	Sender Balance A 19.998	Receiver Balance A 9.997	Fee A 0.001
Last Round 24331053	Sender Rewards A 0	Receiver Rewards A 0	

Şekil 27

Sekil 27'de transfer AlgoExplorer üzerinde gözlenmektedir. Belirlediğimiz 200.000 Token İrem hesabından Beren hesabına başarılı bir şekilde transfer edilmiştir.

Transfer işlemi aynı zamanda Algosigner üzerinden de gerçekleştirilebilir. Bu işlem için ilk olarak kurulumunu gerçekleştirdiğimiz hesabı açarak şifremizi girmemiz gerekmektedir. Giriş yaptığımızda önceden kaydettiğimiz adresler görünür olmayabilir. Bu durumu düzeltmek için sağ üst köşeden testnet üzerinde olduğumuza dikkat etmemiz gerekmektedir.



Şekil 28

Algosigner'da transfer yapmak için ilk olarak transferi yapmak istediğimiz hesaba giriş yapmamız gerekmektedir. JavaScript üzerinde İrem hesabından Beren hesabına bir transfer gerçekleştirdiğimiz için bu sefer tam tersi olarak Şekil 28'de ifade edildiği üzere Beren hesabından İrem hesabına token aktaracağız. Yapmamız gereken adımları teker teker incelememiz gerekirse:

- Gonderici hesap içerisindeki “Send” butonuna basarız.
- Açılan ekrandan aktarmak istediğimiz Token’ı seçmemiz gerekmektedir. Projemizde bu oluşturduğumuz token olan JS Coin’e karşılık gelmektedir.
- Varlık seçiminin yan kısmında göndermek istediğimiz miktar yer almaktadır. Bu değere maksimum hesapta bulunan bakiyeye kadar tüm değerler girilebilir. Şimdilik 18.000 Token transfer edeceğiz.
- Sonrasında gönderilecek adres alt kısma eklenmektedir. Bizim için bu adres İrem hesabının adresi olmaktadır.
- İsteğe bağlı bir not alt kısma eklenir.
- Göndermek için “Send” tuşuna basılır.
- İstenilen zincir şifresi girilir ve transfer tamamlanmış olur.

Transfer sorunsuz bir şekilde tanımlanırsa ekrana bir transfer numarası gelecektir. Bu transfer numarasına yine Algoexplorer ve Goalseeker sitelerinden incelemek mümkündür.

JavaScript ve Algosigner üzerinden yapılan transferler hesapların adresleri ile takip edilebilirler. Bu takibi yine Algoexplorer ve Goalseeker gibi zincir gezginlerinden görüntülemek mümkündür.

AlgodeXplorer
Algorand Blockchain Explorer

Search by Address / Tx ID / Group Tx ID / Block / Asset Name / Asset ID / App ID

TESTNET ▾

Assets Apps Statistics ▾ Blockchain ▾ Tools ▾ API ▾ Governance

Algorand Account Overview

Address: IOIBYZPBAISY636GLWF2PRFXZCPVN2DKXKCCOUCANPA2BROUK...

Balances: A 19.998 Rewards: A 0

BALANCE HISTORY: Total TXs: 5

Transactions Assets

Filter: All

TxID	Block	Age	From	To	Amount	Fee	Type
FFHCXTYNFDRHAIGB5Q...	24330626	6 min ago	BDG7MLDNCHDJH4XV7F...	IOIBYZPBAISY636GLWF2...	18,000 JSCOIN	A 0.001	Transfer
PHUX7RPOJSWNVKDSH...	24330055	41 min ago	IOIBYZPBAISY636GLWF2...	BDG7MLDNCHDJH4XV7F...	200,000 JSCOIN	A 0.001	Transfer
HCKTPKB2YEGDWOUJ...	24284516	1 days ago	IOIBYZPBAISY636GLWF2...	Asset 112384980	A 0.001		Config
LUS4DODOEJSFSSCKEIJU...	24260914	2 days ago	GD64YY3TWGDMCNPP5...	IOIBYZPBAISY636GLWF2...	A 10	A 0.001	Transfer
6B3G05ISM27AMMVBJ3...	24260821	2 days ago	GD64YY3TWGDMCNPP5...	IOIBYZPBAISY636GLWF2...	A 10	A 0.001	Transfer

Filter: All

Şekil 29

AlgodeXplorer
Algorand Blockchain Explorer

Search by Address / Tx ID / Group Tx ID / Block / Asset Name / Asset ID / App ID

TESTNET ▾

Assets Apps Statistics ▾ Blockchain ▾ Tools ▾ API ▾ Governance

Algorand Account Overview

Address: BDG7MLDNCHDJH4XV7FS3YRQN2NTIUMWM4FKFF5Z5YQ4JZGQ...

Balances: A 9.996 Rewards: A 0

BALANCE HISTORY: Total TXs: 6

Transactions Assets

Filter: All

TxID	Block	Age	From	To	Amount	Fee	Type
FFHCXTYNFDRHAIGB5Q...	24330626	7 min ago	BDG7MLDNCHDJH4XV7F...	IOIBYZPBAISY636GLWF2...	18,000 JSCOIN	A 0.001	Transfer
PHUX7RPOJSWNVKDSH...	24330055	41 min ago	IOIBYZPBAISY636GLWF2...	BDG7MLDNCHDJH4XV7F...	200,000 JSCOIN	A 0.001	Transfer
43D7X47D14C23IBZK7U...	24306928	1 days ago	BDG7MLDNCHDJH4XV7F...	BDG7MLDNCHDJH4XV7F...	0 JSCOIN	A 0.001	Transfer
SQ7MRK6EUJ67Y14SBPL...	24303967	1 days ago	BDG7MLDNCHDJH4XV7F...	BDG7MLDNCHDJH4XV7F...	0 JSCOIN	A 0.001	Transfer
				BDG7MLDNCHDJH4XV7F...	0 JSCOIN	A 0.001	Close
OMYADXJ7JCJNO7IWSU...	24303871	1 days ago	BDG7MLDNCHDJH4XV7F...	BDG7MLDNCHDJH4XV7F...	0 JSCOIN	A 0.001	Transfer
DGRD6SAVNW54RKKXIDA...	24260831	2 days ago	GD64YY3TWGDMCNPP5...	BDG7MLDNCHDJH4XV7F...	A 10	A 0.001	Transfer

Filter: All

Şekil 30

Şekil 29'da ve Şekil 30'da oluşturulan hesapların işlem geçmişlerinin AlgoExplorer sitesi tarafından kontrol edildiği ifade edilmiştir. Görüldüğü üzere Token ve Coin transferleri Amount kısmında farklı olarak ifade edilmiştir.

Oluşturulan Token'i Ortadan Kaldırma

Algorand üzerinde oluşturulan Tokenların bazen ortadan kaldırılması gerekebilir. Bu işlem için birkaç kriterin gerçekleşmiş olması gerekmektedir. İlk olarak bir Token'in tamamen yok olabilmesi için tüm token sahiblerinin bulundurduğu miktarın Token oluşturucu hesaba göndermesi gerekmektedir. Bu gönderim sağlandıktan sonra tüm hesaplar ve oluşturucu hesap opt-out yaparak Token'i ortadan kaldırabilirler.

JavaScript üzerinde bu işlemin gerçekleştirilmesi için ilk olarak Beren hesabının tüm Tokenlarını oluşturucu hesap olan İrem hesabına göndermesi ve “opt-out” olması gerekmektedir. Bu işlemi sağlamak için bir transfer fonksiyonu kullanılmaktadır. Transferi gerçekleştirmek için “index.js” dosyasının bulunduğu dosyaya gidilerek “Beren_opt_out.js” adlı bir dosya oluşturulur. Bu dosya içeresine normal bir transfer fonksiyonu yazılır.

```
const Algosdk=require('algosdk');
require("dotenv").config()

const server="https://testnet-algorand.api.purestake.io/ps2";
const port="";
const token={"x-api-key": process.env.API};
let kullanici = new Algosdk.Algodv2(token, server, port);

var beren_mnemonic = "copy deny kingdom faith need blossom anchor zoo address venue carry rain winter salt
rapid infant zero address laptop govern scout rice escape absorb stove";
var berenHesap = Algosdk.mnemonicToSecretKey(beren_mnemonic);
var iremAddress = '1OIBYZPBAISY636GLWF2PRFXZCPVN2DKXKCCOUCANPA2BROUKV2UTOPVYA';

(async () => {
  let assetID = 112384980;
  let params = await kullanici.getTransactionParams().do();
  let sender = berenHesap.addr;
  let recipient = iremAddress;
  let revocationTarget = undefined;
  let note = undefined;
  let closeRemainderTo = iremAddress;
  let amount = 0;

  let txn = Algosdk.makeAssetTransferTxnWithSuggestedParams(sender, recipient, closeRemainderTo,
  revocationTarget,
  amount, note, assetID, params);
  let rawSignedTxn = txn.signTxn(berenHesap.sk)

  let tx = (await kullanici.sendRawTransaction(rawSignedTxn).do());
  console.log("Transfer : " + tx.txId);

})().catch(e => {
  console.log(e);
});
```

Tablo 107 beren_opt_out.js Tamamı

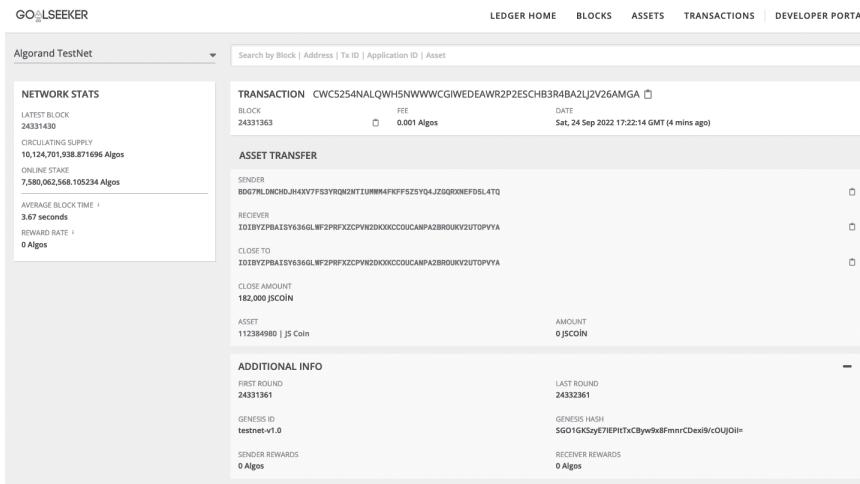
Tablo 107'de opt-out için özelleştirilmiş bir transfer fonksiyonu ifade edilmiştir. Bu kodun neler yaptığına ve normal bir fonksiyondan nerede ayrılarak opt-out olduğuna bakılacak olunursa:

- İlk olarak gerekli paketleri kod içeresine “require” fonksiyonu kullanarak aktarırız.
- Sonrasında zincir içerisinde işlemlerini gerçekleştirecek değişkenler tanımlanır.
- Bu parametreler yeni bir kullanıcı objesi oluşturularak Algosdk içerisindeki metodlar ile kaydedilir.
- Opt-out yapacak yani tokenları gönderecek hesabin mnemonic anahtarı bir değişken üzerine kaydedilir ve bu değişken ile hesabin kendisi türetilir.
- Alıcı hesabin yani oluşturucu hesabin adresi sonrasında kullanılmak amacıyla kaydedilir.
- Asenkron bir fonksiyon tanımlanır.
- Hesaptan çıkarılmak istenilen Token'ın zincir üzerinde kayıtlı olduğu adres "AssetID" değişkeni üzerinde kaydedilir.
- Transfer parametreleri kullanıcı objesinde "getTransactionParams()" metodu çalıştırılarak bir değişkene kaydedilir.
- Gönderici adres yani Beren hesabının adresi mnemonic anahtardan türetilen hesap objesinin içerisinde bulunan "addr" anahtarına eşitlenerek değişkene kaydedilir.
- Note kısmı ve işlemin gerçekleşmemesi halinde bakiyenin aktarılacağı hesap değişkenleri tanımlanır ve tanımsız (undefined) olarak bırakılır.
- Opt-out işleminin gerçekleşmesi için kalan miktarın nereye gideğini ifade eden "closeRemainderTo" parametresi İrem hesabının adresine hedeflenir. Bu sayede hesaptaki tüm Tokenlar İrem hesabına transfer edilir.
- Önceki satırda tüm Tokenları gönderdiğimiz için amount değeri 0 olarak tanımlanır.
- Tüm parametreler ham bir transfer değişkeninde kaydedilir.
- Transfer Beren hesabı tarafından imzalanır.
- İmzalanan transfer kullanıcı tarafından zincire gönderilir ve bir geri dönüş beklenir.
- Transfer numarası terminale yazdırılır.
- Olası bir hata asenkron fonksiyonun sonunda ".catch()" yapısı kullanılarak kaydedilir ve terminale yazdırılır.

Transfer : CWC5254NALQWH5NWWWCGIWEDEAWR2P2ESCHB3R4BA2LJ2V26AMGA

Tablo 108

Opt-out işlemini kodun bulunduğu konumda terminalden “node beren_opt_out.js” komutu ile çağrımadan sonra Tablo 108’de ifade edildiği gibi bir çıktı oluşmaktadır. Bu çıktı AlgoExplorer ve Goalseeker gibi sitelerde incelenebilir.



The screenshot shows the Goalseeker interface for the Algorand TestNet. The top navigation bar includes Ledger Home, Blocks, Assets, Transactions, and a Developer Portal. The main content area is divided into sections: Network Stats, Transaction Details, Asset Transfer, and Additional Info.

Network Stats:

- Latest Block: 24331363
- Circulating Supply: 10,124,701,938.871696 Algos
- Online Stake: 7,580,062,568.105234 Algos
- Average Block Time: 3.67 seconds
- Reward Rate: 0 Algos

Transaction Details:

- Block: 24331363
- Fee: 0.001 Algos
- Date: Sat, 24 Sep 2022 17:22:14 GMT (4 mins ago)

Asset Transfer:

- Sender: BD67MLDNICHQJH4XV7FSYRQK2NT1UWMM4FKFF5ZSYQ4JZGQXNEF05L4TQ
- Receiver: 1OIBYZPBAISY636GLWF2PRFXZCPVNW2D0XCCCOUCANPAZBROUKV2UTOPVYA
- Close To: 1OIBYZPBAISY636GLWF2PRFXZCPVNW2D0XCCCOUCANPAZBROUKV2UTOPVYA
- Close Amount: 182,000 JSCOIN
- Asset: 112384980 | JSCoin
- Amount: 0 JSCOIN

Additional Info:

- First Round: 24331361
- Genesis ID: testnet-v1.0
- Sender Rewards: 0 Algos
- Last Round: 24332361
- Genesis Hash: SGO1GK5syE7iEPit7xCByw9x8fmrnCDex9/cOUjOii=
- Receiver Rewards: 0 Algos

Şekil 31

Şekil 31’de bu transferin Goalseeker sitesi üzerinden incelenmesi ifade edilmiştir. Görüldüğü üzere kapanma miktarı 182.000 JSCOIN yani tüm bakiye olurken transfer edilen miktar 0 JSCOIN olmaktadır.

Account	ASAs	Algos
Beren	0 ASAs	9.995 Algos
Irem	1 ASAs	19.998 Algos

Transactions

ASA TO	ASA FROM	Time
IOIBYZPBAISY636GLWF2PRF...		6 MINS AGO
IOIBYZPBAISY636GLWF2PRF...		50 MINS AGO
	IOIBYZPBAISY636GLWF2PRF...	1 HOUR AGO
	BDG7MLDNCHDJH4XV7FS3Y...	FRI SEP 23 2022
	BDG7MLDNCHDJH4XV7FS3Y...	FRI SEP 23 2022

Şekil 32

Beren hesabının Opt-out işlemi sonrasında hesabında herhangi bir Asset kalmadığı Şekil 32'de Algosigner üzerinden de görünebilir.

Tüm Token tutucularının oluşturucu hesaba Tokenlarını kalan sistemi ile göndermesi sonrasında Irem hesabı Token'ı ortadan kaldırabilmektedir. Bu işlemi gerçekleştirmek için index.js dosyasının bulunduğu klasöre yeni bir dosya oluşturulması gerekmektedir. Bu dosyaya istenilen isim verilebilir ancak anlatım kolaylığı açısından “irem_token_destroy.js” ismi verilecektir.

```
const Algosdk = require('algosdk');
require("dotenv").config()
```

Tablo 109

Dosya başlangıcında ilk olarak gerekli paketlerin tanımlanması gerekmektedir. Bu paketler JavaScript içerisinde bulunan “require()” fonksiyonu ile kod içerisine eklenmektedir. Tablo 109'da ifade edildiği üzere paketlerin sırası ile neler yaptığına bakılacak olunursa:

- **Algosdk:** Algorand blokzinciri ile bağlantı kurulmasını sağlayan pakettir.
- **Dotenv:** Gizli olmasını istediğimiz güvenlik konusunda riskli bilgilerin saklandığı “.env” dosyasına erişimi sağlayan paket olmaktadır.

```
const port = ""
```

Tablo 110

Fonksiyonun içerisinde girdiğimizde birkaç değişken tanımlamamız gerekmektedir. Bu değişkenlerden ilki Tablo 110'da ifade edildiği üzere "port" değeri olmaktadır. Port değeri kontrolün gerçekleşeceği bağlantı portunu ifade etmektedir. Bu işlemin için boş bırakılabilir veya 4001 değeri girilebilir.

```
const token = {"x-api-key": process.env.API}
```

Tablo 111

Tanımlanması gereken bir sonraki değişken ise Purestake sitesinden elde ettiğimiz API anahtarıdır. Tablo 111'de görüldüğü üzere bu anahtarları kullanarak sorgularımızı gerçekleştireceğiz. Token değeri içerisinde anahtarımızı kaydederken obje halinde kaydetmemiz gerekmektedir. Bu objenin içerisinde anahtar olarak "x-api-key" string değeri girilmelidir. Bu sayede sonradan çağrıcağımız fonksiyon tam olarak hangi değerin anahtar olduğunu anlar. Tanımlamanın diğer tarafında API anahtarımız gelmektedir. API anahtarımızı ".env" dosyası içerisinde kaydettik ancak henüz herhangi bir şekilde dosyaya erişmemiştik. Erişimi gerçekleştirmek için "process.env." metodlarını çağırırız. Son noktadan sonra ".env" dosyasından hangi veriyi elde etmek istiyorsak o verinin kaydederken kullandığımız anahtarını girmemiz gerekmektedir. Biz dosya içerisinde API değerine erişmek istediğimiz için "process.env.API" komutu ile bu işlemi gerçekleştirebiliriz.

```
const Server = "https://testnet-algorand.api.purestake.io/ps2"
```

Tablo 112

Tanımladığımız değişkenleri birleştirmeden önce son bir parametre tanımlaması yapmamız daha gerekmektedir. Bu tanımlama sorguyu gerçekleştireceğimiz testnetin adresi olmaktadır. Şekil 112'de ifade edildiği üzere bu adresi API anahtarını aldığımız Purestake sitesi üzerinden Algorand Testnet başlığı altında görüntülemek mümkündür.

```
let Kullanıcı = new Algosdk.Algodv2(token, Server, port)
```

Tablo 113

Daha öncesinde tanımladığımız parametreler Kullanıcı değişkeni üzerinde birleştirmektedir. İlk olarak “let” ile bir değişken tanımlamaktayız. Tablo 113’te gösterildiği üzere bu değişkene kodumuz içerisinde anlaşılırlığın artması amacıyla Kullanıcı adını verdik ancak yabancı kaynaklarda “client” olarak geçtiğini de görebilirsiniz. Devamında tanımın diğer tarafında new anahtar kelimesini görüyoruz. Bu anahtar kelime değişkenin yeni boş bir obje oluşturup içerisinde parametreler almasını sağlamaktadır. SDK içerisinde “algosdk.Algodv2()” metodu ile obje oluşturma koduna erişmektedir. Bu metot içerisinde önceden oluşturduğumuz port, token ve Server değişkenlerini almaktadır. Parametrelerin sıralamasının ifade edildiği düzende olması gerekmektedir. Aksi halde istenilen şekilde çalışmaması söz konusudur.

```
var irem_mnemonic = "ramp later certain business possible memory cupboard armed
truly learn spring you vague vintage nut series curious lonely emerge width patch
turtle wool able rice";
var iremHesabı = Algosdk.mnemonicToSecretKey(irem_mnemonic);
```

Tablo 114

Kullanıcının tanımlanmasından sonra ortadan kaldırılacak olan oluşturulucu hesabın sonradan kullanılmak üzere bir değişkene kaydedilmesi gerekmektedir. Bu işlemi gerçekleştirmek için kaydettiğimiz mnemonic anahtar kullanılmaktadır. Mnemonic anahtarlar insanlar tarafından rahatça okunabilece de makineler tarafından anlaşılır olmamaktadır. Bu nedenle mnemonic anahtarın Algosdk içerisinde bulunan mnemonicToSecretKey() anahtarı kullanılarak Tablo 114’de gösterildiği gibi hesabın kendisine dönüştürülmesi gerekmektedir. Bu sayede “iremHesabı” olarak kaydedilen değişken bir hesap objesi haline gelecektir. İçerisinde hesap adresini, hesabın Secret Key değerini ve mnemonic anahtarını barındıracaktır.

```
(async () => {
})
```

Tablo 115

Token ortadan kaldırma işlemi bir transfer olarak yapılacaktır ve zincire bağlanma gerekmektedir. Bu işlem zaman almaktadır. Bu nedenle JavaScript yapısının sağladığı nimetlerden biri olan asenkron yapısı kullanılacaktır. Asenkron fonksiyon Tablo 115’de ifade edildiği üzere “async” anahtar kelimesi ile tanımlanmaktadır. Herhangi bir çağrıci değişkene sahip olmamasından dolayı çalışma satırının gelmesi halinde otomatik

olarak işlevle geçecektir. İçerisine herhangi bir parametre gerektirmemektedir.

```
let assetID = 112384980;
let params = await kullanıcı.getTransactionParams().do();
let addr = iremHesabı.addr;
let note = undefined;
```

Tablo 116

Asenkron fonksiyon içerisine girildiğinde bazı parametrelerin tanımlanması gerektiği Tablo 116'da ifade edilmektedir. Bu parametrelere teker teker bakılacak olunursa:

- **AssetID:** Ortadan kaldırılmak istenilen token veya varlığın zincir üzerindeki tanımlayıcı numarasını ifade etmektedir. Bu değer token oluşurken zincir tarafından otomatik olarak atanmıştır ve oluşturulmuş tarafından değiştirilemez.
- **Params:** Kullanıcı ile türetilen işlem parametrelerini oluşturan parametredir. “getTransactionParams()” metodu ile çalıştırılır ve fonksiyon olmasından dolayı “.do()” eklentisi ile çalıştırılır. İstenilen isim ile kaydedilebilir.
- **Addr:** Ortadan kaldırılacak hesabın adresini ifade etmektedir. Daha önce mnemonic anahtar ile oluşturduğumuz hesap objesi içerisinde bulunan “addr” anahtar kelimesine erişmektedir.
- **Note:** Transferin gerçekleşirken içermesi istenilen not değişkenidir. Undefined yani boş olarak bırakılabilir. İçerisine girilecek değer transfer örneğinde de ifade edildiği üzere “TextEncoder” ile şifrelenmelidir. Bu şifreleme sayesinde zincirde okunabilen bir hal alacaktır. İstenilen isim ile değişken üzerinde kaydedilebilir.

```
let txn = Algosdk.makeAssetDestroyTxnWithSuggestedParams(addr, note, assetID, params);
let rawSignedTxn = txn.signTxn(iremHesabı.sk);
```

Tablo 117

Gerekli parametrelerin tanımlanmasından sonra bu parametrelerin bir metotta birleştirilmesi gerekmektedir. Bu işlemi gerçekleştirmek için Algosdk içerisinde yer alan “makeAssetDestroyTxnWithSuggestedParams()” metodu kullanılmaktadır. Bu metot Tablo 117'de ifade edildiği üzere içerisinde tanımladığımız değişkenleri verilen sıralama ile almaktadır. Aksi takdirde istenilen şekilde çalışmayacaktır. Txn olarak kaydedilen bu değişken aslında bir transfer olmaktadır ve transfer işlemlerine sokulmalıdır. Bu ne-

denle transfer oluşturan hesabın Secret Key değeri ile imzalanır ve değişkende kaydedilir. İmzalanmış bu transfer zincire gönderilmek üzere hazır hale gelmiştir.

```
let tx = (await kullanıcı.sendRawTransaction(rawSignedTxn).do());
console.log("Transfer : " + tx.txId);
```

Tablo 118

İmzalanan transfer sonrasında asenkron await yapısı ile kullanıcı tarafından “sendRawTransaction()” metodu kullanılarak Tablo 118’de ifade edildiği üzere zincire gönderilir. Bu metot bir obje içerisinde bulunduğu içim “.do()” anahtar kelimesine ihtiyaç duyulmaktadır. Transfer gönderildikten sonra transfer numarası bir değişken üzerinde kaydedilir ve bu numara sonrasında “console.log()” yapısı ile terminale yazdırılır.

```
O.catch(e => {
  console.log(e);
});
```

Tablo 119

Asenkron fonksiyonunun tam çıkış parantezine gelindiğinde Tablo 119’da verildiği üzere bir hata yakalama fonksiyonu eklenir. Bu sayede fonksiyonda oluşabilecek herhangi hatalar ekrana yazdırılır. Bu durumu sağlamak içim “().catch” Arrow Fonksiyonu ile yakaladığımız hatalar “console.log()” fonksiyonu ile terminale yazdırılır.

```

const Algosdk = require('algosdk');
require("dotenv").config()

const server="https://testnet-algorand.api.purestake.io/ps2",
const port="";
const token={“x-api-key”: process.env.API};
let kullanici = new Algosdk.Algodv2(token, server, port);

var irem_mnemonic = “ramp later certain business possible memory cupboard armed truly learn
spring you vague vintage nut series curious lonely emerge width patch turtle wool able rice”;
var iremHesabi = Algosdk.mnemonicToSecretKey(irem_mnemonic);

(async () => {
let assetID = 112384980;
let params = await kullanici.getTransactionParams().do();
let addr = iremHesabi.addr;
let note = undefined;

let txn = Algosdk.makeAssetDestroyTxnWithSuggestedParams(addr, note, assetID, params);
let rawSignedTxn = txn.signTxn(iremHesabi.sk);

let tx = (await kullanici.sendRawTransaction(rawSignedTxn).do());
console.log(“Transfer : “ + tx.txId);

})().catch(e => {
console.log(e);
});

```

Tablo 120 irem_token_destroy.js Kodu Tamamı

Tablo 120’de Token’ı ortadan kaldırmak için gerekli olan kodun tamamı yer almaktadır. Bu kodun işlevine teker teker bakılacak olunursa:

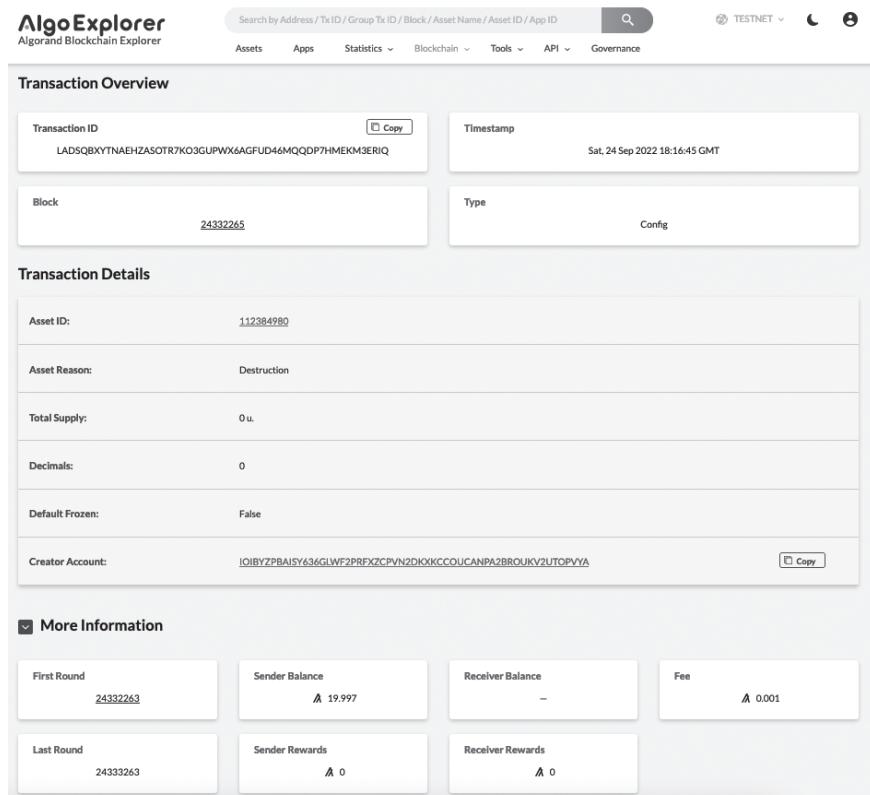
- İlk olarak gerekli paketler tanımlanmaktadır.
- Zincir üzerinde yapılacak sorguyu tanımlayan parametreler değişkenlere aktarılmalıdır.
- Zincir parametreleri bir obje oluşturularak içerisinde farklı özellikler beraberinde eklenir.
- Oluşturan ve ortadan kaldırılacak hesabın mnemonic anahtarı kaydedilerek bu anahtar üzerinden türetilen hesap bir obje içerisinde kaydedilir.
- Asenkron fonksiyon tanımlanır.
- Bu asenkron fonksiyonun içerisinde hangi Token’ın ortadan kaldırılacağı ve transfer parametreleri gibi birçok değişken tanımlanır.
- Tanımlanan değişkenler Algosdk üzerinde bulunan “makeAssetDestroyTxnWithSuggestedParams()” metodu ile bir ortadan kaldırma transferinde birlleştirilir.
- Transfer oluşturan hesabın Secret Key değeri tarafından imzalanır.
- İmzalanan transfer kullanıcı tarafından zincirle gönderilir.
- Transfer sonucu terminale yazdırılır.

- Olası hatalar “.catch()” fonksiyonu tarafından yakalanarak terminale yazdırılır.

Transfer : LADSQBXYTNAEHZASOTR7KO3GUPWX6AGFUD46MQQDP7HMEKM3ERIQ

Tablo 121

Dosya tamamlandıktan sonra bulunduğu klasörde açılan terminal ile “node irem_token_destroy.js” komutu ile çalıştırılacak olunursa Tablo 121’da ifade edilen çıktıyı verecektir. Bu şekilde bir transfer numarası alınmış olunursa işlem başarı ile gerçekleşmiş olacaktır. Bu transfer numarasına Goalseeker ve Algoexplorer gibi sitelerden bakmak mümkündür.



Transaction Overview

Transaction ID	LADSQBXYTNAEHZASOTR7KO3GUPWX6AGFUD46MQQDP7HMEKM3ERIQ <input type="button" value="Copy"/>	Timestamp	Sat, 24 Sep 2022 18:16:45 GMT
Block	24332265	Type	Config

Transaction Details

Asset ID:	112384980
Asset Reason:	Destruction
Total Supply:	0 u.
Decimals:	0
Default Frozen:	False
Creator Account:	JQIBYZPBAISY636GLWF2PRFXZCPVN2DOKKCCOUCANIP2BROUJKV2UTOPVYA <input type="button" value="Copy"/>

More Information

First Round 24332263	Sender Balance A 19.997	Receiver Balance -	Fee A 0.001
Last Round 24333263	Sender Rewards A 0	Receiver Rewards A 0	

Şekil 33

Şekil 33’te gerçekleşen bu ortadan kaldırma işleminin AlgoExplorer’dan göz atılması ifade edilmiştir. “Asset Reason” anahtar kelimesinin karşısına bakılacak olunursa bu işlemin “Destruction” yani imha etme işlemi olduğu ifade edilmektedir.

The image consists of three side-by-side screenshots of the Algosigner web application interface, all set to the TestNet environment.

- Left Screenshot:** Shows the "ALGOSIGNER" interface with two accounts: "Beren" and "Irem". "Beren" has 0 ASAs and 9.995 Algos. "Irem" has 0 ASAs and 19.997 Algos. A "Send" button is visible between the accounts.
- Middle Screenshot:** Shows the "Transactions" section. It lists three asset transfers:
 - ASA FROM: BDG7MLDNCHDJH4XV7FS3Y... (59 MINS AGO)
 - ASA FROM: BDG7MLDNCHDJH4XV7FS3Y... (1 HOUR AGO)
 - ASA TO: BDG7MLDNCHDJH4XV7FS3Y... (2 HOURS AGO)
 The last transfer is to "JS Coin". A "DRAFT" button is visible at the bottom of this section.
- Right Screenshot:** Shows the "Opt-in to an asset" section. It has a search bar containing "JSCOIN". A "Verified" checkbox is checked. A "No results!" message is displayed below the search bar.

Small text at the bottom of each screenshot indicates copyright: "© 2022 PureStake".

Şekil 34

Yapılan imha işlemine Algosigner üzerinden bakmak mümkündür. Şekil 34'te ifade edildiği üzere her iki hesapta da herhangi bir Asset bulunmuyor olmaktadır. Ayrıca "Opt-in to an asset" sekmesinde öncesinde var olan Token'in artık bulunmuyor olduğu görülmektedir.

Bu şekilde bir ASA Token'ı oluşturulabilir, sorgulanabilir, transfer edilebilir ve yok edilebilir.

İKİNCİ BÖLÜM

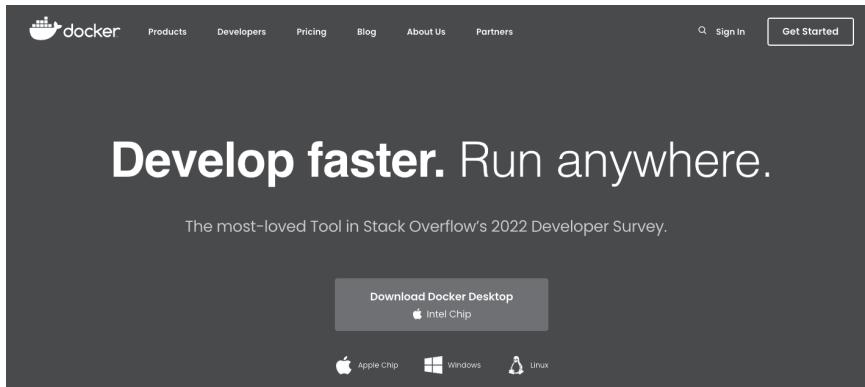
PYTEAL İLE PYTHON ÜZERİNDE AKILLI KONTRAT GELİŞTİRME

Pyteal, Algorand Akıllı Kontratları yazmak için geliştirilen TEAL dilini Python üzerinde yazmamızı sağlayan bir kütüphanedir. TEAL diline daha yakından bakacak olursak Algorand Akıllı Kontratlarını yazmamızı sağlayan Stack tabanlı yeni bir kodlama dilidir. Açılmış “Transaction Execution Approval Language (TEAL)” yani Türkçeye çevirisi “İşlem Yürütme Onay Dili” şeklindedir. Bu dil non-Turing Complete bir dildir yani Tamamlanmamış bir Turing dilidir. Bunun anlamı içerisinde branch forwards'a yani dallanmaya izin verirken döngülere yani recursive logic'e izin vermez. Bu nedenle güvenliği ve performansı maksimum düzeyde tutar. Yinede, Teal temelinde bir birleştirici dil. Pyteal ile geliştiriciler sadece Python kullanarak tam işlevli bir akıllı kontrat oluşturabilir, bu kodları kolayca Teal diline çevirebilir.

Docker Kurulumu

Docker, uygulamalarınızı hızla derlemenizi, test etmenizi ve dağıtmayı sağlayan bir yazılım platformudur. Docker, yazılımları kitaplıklar, sistem araçları, kod ve çalışma zamanı dahil olmak üzere yazılımların çalışması için gerekli her şeyi içeren konteyner adlı standartlaşırılmış birimler halinde paketler. Docker'ı kullanarak her ortama hızla uygulama dağıtip uygulamaları ölçeklendirebilir ve kodunuzun çalışacağından emin olabilirsiniz.

Projemizde de Sandbox'ın çalışması için sürekli olarak Docker'ın arka planda açık olması gerekmektedir. Bu nedenle Docker kurulumu gerçekleştirilmeden akıllı kontratlarımızı test edemeyiz. Docker'ı <https://www.docker.com/> adresinden işletim sistemine uygun olarak indirip kurabiliriz.

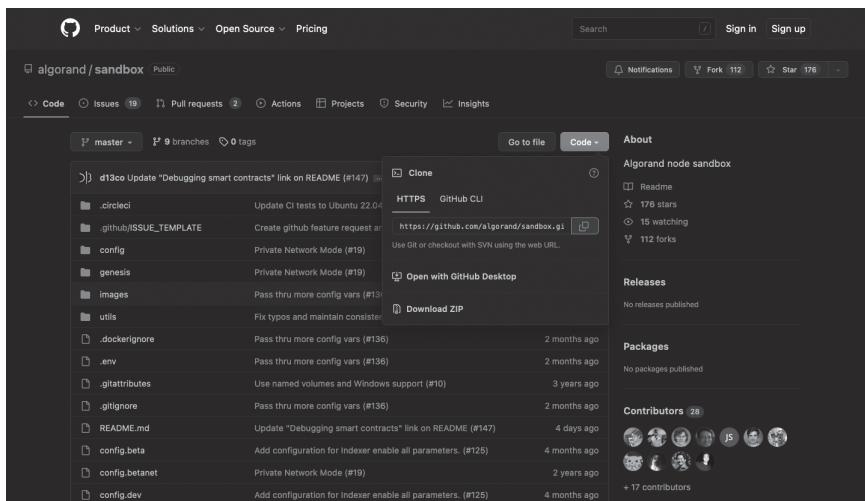


Şekil 35

Şekil 35'te Docker sitesi ifade edilmiştir. Bu site içerisinde de görüldüğü üzere Docker MacOS, Windows ve Linux sistemlerde desteklenmektedir.

Sandbox Kurulumu

Sandbox Algorand akıllı kontratlarının oluşturulmasını ve geliştirilmesini hızlandıran bir ortam olmaktadır. Sandbox Algorand ağının lokal olarak çalışmasını sağlayan Docker konteynerleri bütünüdür. Üzerinde işlemler yapabileceğimiz local bir Algorand ağı oluşturmaktadır. Çalışması için Docker aracının yüklü ve arka planda çalışıyor olması gerekmektedir. Algorand Sandbox ortamının kurulması için Algorand'in Github sayfası üzerinden Sandbox klasörünün klonlanması gerekmektedir. Bu işlem için <https://github.com/algorand/sandbox> sitesi ziyaret edilmektedir.



Şekil 36

Sandbox klasörüne giriş yapıldıktan sonra Şekil 36'da ifade edildiği üzere yeşil renkli “Code” butonuna tıklanmalı ve oradan da “HTTPS” başlığı altında bulunan URL kopyalanmalıdır. Kopyalama işlemi URL'nin yanında bulunan iki içe geçmiş kare sembolüne tıklanarak otomatik olarak gerçekleştirilebilir. Bağlantı kopyalandıktan sonra proje içerisine eklenmeye hazır bir hal almaktadır. Kurulum için ilk olarak VS Code editörü açılır. Sonrasında tüm Pyteal kodlarını barındıracak bir klasör oluşturulması beklenir. Hali hazırda bulunan klasörlere de eklenebilse de kullanım kolaylığı ve düzen açısından yeni bir klasörün oluşturulması tavsiye edilmektedir.

```
git clone https://github.com/algorand/sandbox.git
```

Tablo 122

Oluşturulan klasöre cd komutları ile terminalden erişilmesi sonrasında Tablo 122'de ifade edildiği üzere “git clone” komutu ile Github projesi klasörümüze eklenecektir. Klonlama işlemi sonrasında proje klasörü içerisinde Sandbox adlı bir klasör oluşacaktır. Bu klasör Sandbox ortamında erişmek istediğimiz tüm özellikleri barındırmaktadır.

Sandbox ortamı kurulumundan sonra projelerimizin yer alacağı bir iskelet yapısı gerekmektedir. Bu yapı için yine Algorand Github sayfası içerisinde bulunan <https://github.com/algorand-devrel/pyteal-course> linkinden erişilen klasörü klonlamamız gerekmektedir.

```
git clone https://github.com/algorand-devrel/pyteal-course.git
```

Tablo 123

Sandbox dosyasını içerisinde oluşturduğumuz klasöre terminalden ulaşarak Tablo 123'de ifade edilen “git clone” kodunu çağırabiliriz. Bu şekilde örnek bir iskelet projemize eklenmiş olacaktır. Oluşacak “pyteal-course” adlı bu klasör VS Code veya dosya yöneticisi üzerinden kullanım kolaylığı amacıyla “project” olarak değiştirilecektir.

Project dosyası içerisinde bazı düzenlemeler yapılması gerekmektedir. Düzenlemeler için ilk olarak “cd” komutu ile proje klasörüne terminalden girilmesi gerekmektedir. Klasör içerisinde bir Python Sanal Makinesi kurulması gerekmektedir. Bu şekilde kurduğumuz paketler ve kütüphaneler sadece bu sanal makine üzerinde var olacaktır.

```
python -m venv venv
```

Tablo 124

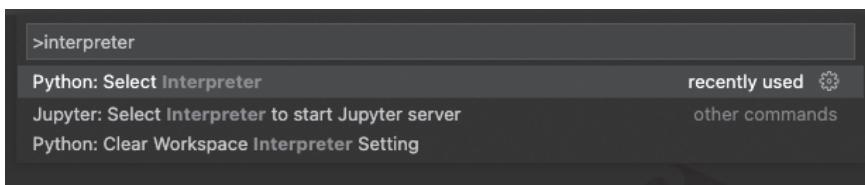
Sanal ortamı kurmak için Tablo 124’te ifade edildiği üzere “python -m venv” komutu girilmelidir. Bu komuttan sonra tekrarlanan “venv” ibaresi sanal makinenin adını ifade etmektedir ve herhangi bir değer alabilir. Ancak VS Code ve diğer editörlerin daha kolay bir şekilde sanal makineyi tespit etmesi amacıyla “venv” adı tercih edilmiştir.

```
source ./venv/Scripts/activate #Windows işletim sistemine sahip makinalar için
source ./venv/bin/activate #Mac ve Linux işletim sistemine sahip makinalar için
```

Tablo 125

Oluşturulan sanal makineyi terminalden erişebilecek şekilde aktif etmemiz gerekmektedir. Bu işlemi gerçekleştirmek için Tablo 125’te ifade edildiği üzere işletim sistemine özgü aktif etme kodunu terminal üzerinden çalıştırmak gerekmektedir. Bu sayede sanal makine aktif hale gelecektir.

Terminal üzerinde bir değişiklik yapsak da bu değişikliği aynı zamanda VS Code için de yapmamız gerekmektedir. Bu işlem için MacOS üzerinde “cmd + shift + p” ile Windows üzerinde “crtl + shift + P” ile eriştiğimiz komut paletine “interpreter” anahtar kelimelerini yazmamız gerekmektedir.



Şekil 37

Şekil 37’de ifade edildiği üzere karşımıza çıkan seçenekler arasından “Python: Select Interpreter” seçeneğini seçmemiz gerekmektedir. Bu kısımda karşımıza öncesinde oluşturduğumuz veya “defult” olarak oluşan sanal makineler gelmektedir. Ancak bu makineler arasında seçmek yerine direkt olarak dosyayı seçirmemiz gerekmektedir. Bu işlem için en üst kısımda bulunan “Enter interpreter path” seçeneği seçildikten sonra karşımıza çıkan “Find” tuşuna basarak dosya gezgini içerisinde proje klasörünün içerisinde bulunan “venv” klasörünü bulmamız gerekir. Bu klasör içerisinde bulunan bin alt klasörü içerisindeki “Python” adlı dosyayı seçeriz ve “Select Interpreter” butonu ile işlemi tamamlarız. Bu şekilde sanal makine VS Code içerisinde çalışmaya hazır hal olacaktır.

```
pip install -r ./requirements.txt
```

Tablo 126

Aktif edilen sanal makine içerisinde bazı paketler eklenmelidir. Bu paketler requirements.txt klasörü içerisinde yer almaktadır. Tablo 126'da ifade edildiği üzere “pip install” komutu ile indirilmekte ve kurulmaktadır. Kurulum bittikten sonra paketler başarılı bir şekilde projemize eklenmiş olur. Bu dosyaların içerisinde Python Algorand SDK ve Pyteal kütüphaneleri de bulunmaktadır.

Son olarak kurulumu tamamlamak için Sandbox klasörü ile Project klasörünü bağlamamız gerekmektedir. Bu işlemi gerçekleştirmek için Sandbox klasörü içerisinde bulunan “docker-compose.yml” dosyası içerisinde bazı değişiklikler yapmamız gerekmektedir. Dosya içerisinde girildikten sonra “services” ve sonrasında “algod” anahtarları altına geldiğimizde eklemeye yapacak yeri belirlemiştir.

```
ports:  
...  
volumes:  
- type: bind,  
  source: ../project  
  target: /data
```

Tablo 127

Tablo 127'de ifade edildiği üzere “ports” anahtarı hizasında “ports” içerisinde değişiklik yapılmadan yeni bir “volumes” anahtarı eklenir. Bu anahtar içeriğine type, source ve data parametreleri eklenmektedir. Bu anahtarlar teker teker bakılacak olunursa:

- **Type:** İşlem türünü ifade eder. Bağlama yapılacağı için bind olarak ifade edilmektedir.
- **Source:** Bağlamanın kaynağının konumu ifade etmektedir. Bir üst klasördeki Project klasörü kaynak olduğu için ../project olarak ifade edilmiştir.
- **Target:** Bağlamanın hedefini ifade etmektedir. Hedefte bulunan / data klasörü ile Sandbox konteyniri tarafından erişilebilir hal alır.

Kurulum sonrası Sandbox çağrılmaya hazır hal alır. Bu işlem için ilk olarak Sandbox klasörüne terminal üzerinden erişilmesi gerekmektedir. Klasör değişimi sağlandıktan sonra test kodları çalıştırılabilir.

```
./sandbox up
```

Tablo 128

İlk olarak çalıştırılacak test komutu “sandbox up” olmaktadır. Bu sayede Docker konteynerleri kurulur ve çalıştırılır. Bu komutun çalışabilmesi için arka planda Docker’ın bilgisayarda çalışıyor olması gerekmektedir. Tablo 128’de ifade edilen bu kod ilk çalışmasında uzun bir süre alabilemektedir. Kodu ilk defa çalıştırduğumızda bizlere birçok bilgi ve örnek olarak çalıştırabilecek komutları çıktı olarak vermektedir.

```
./sandbox goal account list
```

Tablo 129

Tablo 129’da ifade edildiği üzere bu komutlardan biri de örneklerimiz içerisinde çokça kullanacak olduğumuz local adres listesini ekrana yazdırın kod parçasıdır. Bu kodu çalıştırabiliyor olmamız doğru bir şekilde kurulum yaptığımızı ifade etmektedir.

Pyteal Araçlarının Kullanılması

Hadi ilk Pyteal kodumuzu yazarak başlayalım. Bu işlem için “counter” klasörü içerisinde girilir ve içerisindeki tüm dosyalar silinerek “step_01.py” adında bir dosya oluşturulur. Her Algorand Akıllı kontratı temel olarak 2 parça Teal kodu içermektedir. Bunlar onayla anlamına gelen “approvel” ve temizle anlamına gelen “clear” dosyaları olmaktadır. Bu dosyalar Python üzerinde fonksiyon olarak gösterilir ancak çalıştırıldıklarında “.teal” uzantılı dosyalar olmaktadır.

```
def approval():
    pass
```

Tablo 130

Kod dosyasının içerisinde girildiğinde ilk olarak “approval” fonksiyonu ile başlanması gerekmektedir. Bu fonksiyonu Tablo 130’da ifade edildiği üzere Python içerisinde bulunan def metodu ile tanımlamaktayız. Fonksiyon içerisinde parametre olarak herhangi bir değer almamaktadır. Fonksiyonun içerisinde girildiğinde “pass” anahtar kelimesi ile karşılaşılmaktadır. Bu komut fonksiyonun içerisinde girildiğinde direkt olarak çıkışmasını sağlamaktadır.

```
def approval():
    pass
def clear():
    pass
```

Tablo 131

Aynı işlemi Tablo 131'de ifade edildiği üzere “clear” için de yapmamız gerekmektedir. Bu sayede akıllı kontratı oluşturan tüm basamaklar tamamlanmış olunur. Clear fonksiyonu da aynı “approval” gibi içine parametre almayaarak direkt olarak fonksiyondan çıkış sağlar.

```
import importlib
import sys

from pyteal_helpers import program

if __name__ == "__main__":
    mod = sys.argv[1]

    try:
        approval_out = sys.argv[2]
    except IndexError:
        approval_out = None

    try:
        clear_out = sys.argv[3]
    except IndexError:
        clear_out = None

    contract = importlib.import_module(mod)

    if approval_out is None:
        print(program.application(contract.approval()))
    else:
        with open(approval_out, "w") as h:
            h.write(program.application(contract.approval()))

    if clear_out is not None:
        with open(clear_out, "w") as h:
            h.write(program.application(contract.clear()))
```

Tablo 132 compile.py

Oluşturduğumuz dosyanın Teal koduna “compile” edilmesi için bir “compiler” yani derleyici yapısı gerekmektedir. Bu işlemi project klasörü içerisinde bulunan compile.py dosyası halletmektedir. Bu dosya içerisinde yer alan Tablo 132'de belirletilen koda baktığımızda kontrat içerisinde bulunan “approval” ve “clear” fonksiyonlarını çağrılığını gözlemebiliriz.

Fonksiyonu çağırırken “pyteal_helpers” kütüphanesi içerisinde çalışan program metodu kullanılmaktadır.

```
#!/usr/bin/env bash

mkdir -p ./build/
# clean
rm -f ./build/*.teal

set -e # die on error

python ./compile.py "$1" ./build/approval.teal ./build/clear.teal
```

Tablo 133 build.sh

Derlenen bu kodlar build.sh dosyası içerisinde bulunan ve Tablo 133’te ifade edilen terminal komutları sayesinde oluşturulan “./build” klasörü içersine “approval.teal” ve “clear.teal” dosyaları olarak kaydedilir.

Artık akıllı kontrat içersine küçük bir Pyteal komutu yazmaya başlayabiliriz. Bunun için yazılabilen en basit Pyteal kodu olan gelen tüm işlemleri onaylayan bir kontrat yazılabilir.

```
from pyteal import *
```

Tablo 134

İlk olarak Python dosyası içersine tüm Pyteal fonksiyonlarını Tablo 134’de ifade edildiği üzere ekleyerek başlanılabilir. Yıldız (*) yapısı ile çağırılan kütüphane içerisindeki bütün fonksiyonlar başlarına Pyteal ibaresi eklenmeden çağrılabilecek hale gelmektedir.

```
from pyteal import *

def approval():
    return Approve()
def clear():
    return Approve()
```

Tablo 135

Gelen tüm işlemlerin onaylanması için oluşturduğumuz “approval” ve “clear” fonksiyonları içerisinde “Approve()” fonksiyonunun “return” edilmesi gerekmektedir. Bu sayede fonksiyona gelen bilgi ne olursa olsun fonksiyon bu bilgiye bakmadan işlemi onaylayacaktır. Bu şekilde Tablo 135’ ifade edildiği üzere fonksiyon tamamlanmış hale gelmektedir.

```
./build.sh contracts.counter.step_01
```

Tablo 136

Tamamlanan fonksiyonu project klasörü içerisinde bulunan terminalde Tablo 136'da belirtildiği şekilde çalıştırabiliriz. Bazı işletim sistemlerinde bu kod erişim izni hatası verebilmektedir. Bu nedenle baş kısmına “sudo” komutunun eklenmesi dosyayı yönetici olarak açmamızı sağlar ve izin hatalarını ortadan kaldırır.

Derlenen kod project klasörü içerisinde yeni bir “build” klasörü oluşturmaktadır. Bu klasör içerisinde iki adet Teal dosyası bulunmaktadır. Bu dosyalar “approval.teal” ve “clear.teal” dosyaları olmaktadır.

```
#pragma version 5
int 1
return
```

Tablo 137 approval.teal

```
#pragma version 5
int 1
return
```

Tablo 138 clear.teal

Her iki fonksiyon için de aynı kod yazılmasından dolayı her iki teal dosyası da Tablo 137'de ve Tablo 138'de ifade edildiği üzere aynı kodu içermektedir. Bu kodları satır satır anlatmamız gereklidir:

- İlk olarak Teal dilinin versiyonu belirtilmektedir. Kodumuz içerisinde yazım aşamasında en güncel sürüm olan 5 sürümü kullanılmaktadır.
- Alt satıra inildiğinde integer bir 1 değerinin yer aldığı görülmektedir. Algorand akıllı kontratları bir transferin geçerli veya geçerli olmadığına karar verirken programın çıkış yaparken ki ifade ettiği bit değerine bakarlar. Eğer bu bit değeri 1 ise transfer geçerlidir ancak bu değer 1 değerinin dışında herhangi bir değer ise transfer geçersiz olmaktadır.
- Son olarak akıllı kontratı kapatmak için “return” komutu kullanılmaktadır. Bu sayede akıllı kontrat tamamlanmış olacaktır.

Yazılan bu kontrat önüne çıkan tüm transferleri onaylamaktadır.

Akıllı Kontratın Başlatılması

Akıllı kontratlarımıza ile ilgiliş şeyle yapmaya bu adımda başlayabiliriz. Akıllı kontratlar yapısı gereği bir hayat döngüsünden geçmektedir. Oluşturulurlar, güncellenirler, katılınlar (opt-in) ayrınlırlar (opt-out) ve silinirler. Her akıllı kontratta çok yaygın olarak kullanılan işlemlerin koşullu olarak hangi tür transfere denk geldiğini ifade eden yapılar bulunmaktadır.

Bu işlemlerin hangi tür olduğunu tespit etmek için “pyteal-helpers” dosyası içerisinde program.py dosyası bünyesinde “event” fonksiyonu bulunmaktadır. Bu fonksiyonun kontrolünü gerçekleştirdiği birkaç değer bulunmaktadır.

```
def event(
    init: Expr = Reject(),
    delete: Expr = Reject(),
    update: Expr = Reject(),
    opt_in: Expr = Reject(),
    close_out: Expr = Reject(),
    no_op: Expr = Reject(),
) -> Expr:
    return Cond(
        [Txn.application_id() == Int(0), init],
        [Txn.on_completion() == OnComplete.DeleteApplication, delete],
        [Txn.on_completion() == OnComplete.UpdateApplication, update],
        [Txn.on_completion() == OnComplete.OptIn, opt_in],
        [Txn.on_completion() == OnComplete.CloseOut, close_out],
        [Txn.on_completion() == OnComplete.NoOp, no_op],
    )
```

Tablo 139 program.py Kod Parçası

Event fonksiyonu içerisinde bakıldığından hayat döngüsünden bulunan değerlere denk gelen fonksiyonların var olduğu göz önüne gelmektedir. Bu işlemlere Tablo 139'da ifade edildiği üzere bakılacak olunursa:

- **Txn.application_id()**: Eğer 0 değerine sahipse yeni bir uygulama numarası atanmamıştır. Bu durumda zincir kontratın kurulum aşamasında olduğunu anlamaktadır.
- **Txn.on_completion()**: İşlemlerin tamamlanması halinde hangi transfer metoduna karşılık geldiğini ifade etmektedir. Bu işlemler silme (delete), güncelleme (update), ekleme (opt-in), çıkış yapma (close_out) ve özel işlem yapma (no_op) işlemleri olmaktadır. No_op transferi üzerinde kendi hareketlerini belirlediğimiz fonksiyonlar yazılabilir.

Transferleri yazdığımız akıllı kontratlar içerisinde kullanabilmek için ilk olarak “pyteal_helpers” paketini kodumuza eklememiz gerekmektedir.

```
from pyteal_helpers import program
```

Tablo 140

Tablo 140'ta ifade edildiği üzere Pyteal kütüphanesini eklediğimiz satırın altına bu eklemeyi yapabiliriz. İçerisinde birçok metod bulunan “pyteal_helpers” kütüphanesi içerisinde sadece program fonksiyonunu kullanmamız yeteri olmaktadır.

```
from pyteal import *
from pyteal_helpers import program
def approval():
    return program.event(
        init=,
    )
def clear():
    return Approve()
```

Tablo 141

Akıllı kontratın içerisinde girildiğinde önüne gelen tüm transferleri onaylaması yerine Tablo 141'de ifade edildiği üzere program fonksiyonunun içerisinde bunulduğu “event” metoduna parametre olarak “init” anahtarını eklemektediriz.

Eklediğimiz “init” parametresi içerisinde “Seq()” fonksiyonu eklememiz gerekmektedir. Bu fonksiyon birden fazla işlemi birbirine bağlamayı sağlamaktadır. Bir sayıci devresi oluşturduğumuz için sayıci değerini almaktadır. Sayıcı değerini oluşturmak için “approve” fonksiyonu içerisinde bir sayıci değeri tutmamız gerekmektedir. Bu değer normal Python veri tiple-rine kıyasla farklı kaydedilecektir.

```
global_sayici = Bytes("sayici")
```

Tablo 142

Akıllı kontrat yazım aşamasında değişken tanımlarken birkaç özellikle dikkat edinilmesi gerekmektedir. Algorand akıllı kontratlarında veriler global ve local olarak iki farklı şekilde depolabildikleri için nasıl bir değişken olduğu kolaylık açısından tanım içerisinde belirtilebilir. Tanımlayacağımız değişken global türde sahip olacağı için “global_sayici” isimlendirmesini kullanmayı tercih ettiğim. Akıllı kontratta verilerin kaydedilmesi için Byte formatında olması gerekmektedir. Bu nedenle Tablo 142'de belirtildiği üzere tanımlanırken “Bbytes” fonksiyonu içerisinde tanımlanması gerekmektedir. Ancak dikkat edilmesi gereken bu fonksiyona parametre olarak girilen

“sayıcı” değerinin Byte değerinin kendisi olmaması tanımlayıcısı yani bir bakıma değişken ismi olmalıdır.

Değişkeni Algorand Akıllı Kontratına uygun bir şekilde kaydetmemizden sonra “init” içerisinde uygun şekilde eklenebilir.

```
from pyteal import *
from pyteal_helpers import program

def approval():
    global_sayici = Bytes("sayici")
    return program.event(
        init= Seq(
            App.globalPut(global_sayici, Int(0))
        )
    )
def clear():
    return Approve()
```

Tablo 143

Tablo 143’da ifade edildiği üzere Seq() fonksiyonu içerisinde ilk işlem eklenebilir. Bu işlem App.globalPut() fonksiyonu ile gerçekleştirilmektedir. Bu fonksiyon içerisinde iki adet parametre almaktadır. Parametrelere teker teker bakılacak olunursa:

- **Değişken:** İçerisine eklenenek Byte türünde tanımlanan değişkeni ifade etmektedir.
- **Değer:** Değişkenin içerisinde eklenenek değeri formatıyla beraber ifade etmektedir. Python üzerinde kullanılan ve küçük harf ile başlayan “int()” fonksiyonu yerine büyük harfle kullanılan ve Pyteal içerisinde gelen “Int()” fonksiyonunun kullanıldığı görülmektedir.

Genellikle bazı durumlarda bir tanımlayıcı olarak kontratı oluşturan veya dağıtan adresin bilgilerinin kontrat üzerinde yer alması gerekmektedir. Bu işlem için daha önce de yapıldığı üzere bir Byte türüne ait tanımlayıcı oluşturulmaktadır.

```
global_tanimlayici = Bytes("tanimlayici")
```

Tablo 144

Tanımlayıcıyı Tablo 144’te ifade edildiği üzere global olarak tanımlanmaktadır. Ancak önceki tanımlama ile kendi içerisinde farklılık gösterecektir. Bu değişiklik Byte türlerinden olmaktadır. Bir önceki kaydettiğimiz değer integer değeri olmaktadır ancak “global_tanimlayici” olarak kayde-

dilen değer bir “Byteslice” değeri olmaktadır. Byteslices baytlardan oluşan bir dizi olmaktadır ve string olarak düşünülebilir.

```
from pyteal import *
from pyteal_helpers import program

def approval():
    global_sayici = Bytes("sayici")
    global_tanimlayici = Bytes("tanimlayici")

    return program.event(
        init= Seq(
            App.globalPut(global_sayici, Int(0)),
            App.globalPut(global_tanimlayici, Txn.sender())
        )
    )
def clear():
    return Approve()
```

Tablo 145

Tanımlanan bu tanımlayıcının da aynı şekilde kod içerisinde eklenmesi gerekmektedir. Bu işlemi Tablo 145’de ifade edildiği üzere Seq() fonksiyonundan yararlanarak yapmaktadır. Önceki eklenen değerden sonra virgül koyularak bir sonraki satır geçiş yapılır ve bu satır içerisinde aynı şekilde “App.globalPut()” çağrılrır. Bu fonksiyon ilk olarak kaydedilecek değişken olan “global_tanimlayici” değerini almaktadır. Aldığı bir diğer parametre ise kontrata eklenecek bu değişkenin değeridir. Gönderici adresin eklenmesini istememizden dolayı buraya “Txn.sender()” fonksiyonu eklenebilir. Bu fonksiyon 64 karakterli gönderici adresini kontrat içerisinde “global_tanimlayici” içerisinde ekleyecektir.

```

from pyteal import *
from pyteal_helpers import program

def approval():
    global_sayici = Bytes("sayici")
    global_tanimlayici = Bytes("tanimlayici")

    return program.event(
        init= Seq(
            App.globalPut(global_sayici, Int(0)),
            App.globalPut(global_tanimlayici, Txn.sender()),
            Approve()
        )
    )
def clear():
    return Approve()

```

Tablo 146

Son olarak “Seq()” fonksiyonunu “Approve()” komutu ile tamamlamaktayız. Bu şekilde “initialization” işlemi tamamlanmış olmaktadır. Kodumuzun bir kısmı Tablo 146’da ifade edildiği üzere hazır olduğu için “compile” edip test edebiliriz.

```
./build contracts.counter.step_01
```

Tablo 147

Derleme işlemini başlatmak için tablo 147’de ifade edildiği üzere “build” komutunu uygun dosya yolu ile çalıştırabiliriz.

```
#pragma version 5
txn ApplicationID
int 0
==
bnz main_l12
txn OnCompletion
int DeleteApplication
==
bnz main_l11
txn OnCompletion
int UpdateApplication
==
bnz main_l10
txn OnCompletion
int OptIn
==
bnz main_l9
txn OnCompletion
int CloseOut
==
bnz main_l8
txn OnCompletion
int NoOp
==
bnz main_l7
err
main_l7:
int 0
return
main_l8:
int 0
return
main_l9:
int 0
return
main_l10:
int 0
return
main_l11:
int 0
return
main_l12:
byte "sayici"
int 0
app_global_put
byte "tanimlayici"
txn Sender
app_global_put
int 1
return
```

Tablo 148 approval.teal

Derlenip Teal formatına getirilen kod incelendiğinde Teal mimarisinin nasıloluştuğu gözler önüne sürülmektedir. Tablo 148'da ifade edilen “approval.teal” dosyasının neler yaptığına bakılacak olunursa:

- Başlangıçta Teal dilinin versiyonu ifade edilmektedir.
- Sonrasında ApplicationID tanımlayıcı tanımlanıp hemen altında bu tanımlayıcıya int 0 değeri girilmiştir. Bu durum yazılan kontratın şu anlık 0 uygulama numarasına sahip olduğunu ifade etmektedir.
- Alt kısmına göz gezdirildiğinde Teal yapısı içerisinde tanımlanan fonksiyonlar yer almaktadır. Bu fonksiyonlar “main_lX” isimleri ile oluşmaktadır. Öncesinde bahsedilen “Oncompletion” değişkeni içe-risine “DeleteApplication”, “OptIn” ve “No-op” gibi integer değerlerini atamaktadır. Bu değerler kontrat içerisinde 1 değerine sahip olduklarıda denk gelen işlem gerçekleştirilecektir.
- Fonksiyon tanımlanması sonrasında bu fonksiyonlara verilecek gir- dilerin tanımlanması gerekmektedir. Herhangi bir silme veya gün- celleme işlemi yapılmayacağı için bu fonksiyonların içerisinde “int 0” değeri girilmektedir.
- Main_l12 adı ile tanımlanan fonksiyon içerisinde akıllı kontrat üz-erinde yaptığı işlevler yer almaktadır. Sırasıyla gidilecek olunur- sa ilk olarak “byte” komutu ile değişken tanımlanır ve alt satırında bu tanımlayıcı içerisinde int 0 değeri atanmaktadır. Atanma sonrası “app_global_put” ile zincire bu değer global olarak eklenmektedir. Aynı işlevler “tanimlayıcı” için de yapılmakta olunup değerine “sa- yıcı” tanımlayıcısına farklı olarak “int” değeri atanmamakta olunup “txn Sender” fonksiyonu ile adresi tanımlamaktayız.
- İşlemin onaylaması için çıkış yapılırken “int 1” değeri girilir ve “re- turn” ile dosya tamamlanır.

Şu ana kadar yazdığımız fonksiyon sadece “initialization” işlevini des-teklemektedir. Ancak yazdığımız bu kontrat testlere sokulabilir. Testi sağ-lamak için ilk olarak Sandbox klasörü içerisinde “cd” komutu ile girilmesi gerekmektedir.

```
./sandbox enter algod
```

Tablo 149

Sandbox klasörü içerisinde terminalden Tablo 149'da ifade edildiği üzere komut çalıştırılmalıdır. Bu komut Docker konteynerleri içerisinde gi-rlmesini sağlamaktadır.

```
ls /data
```

Tablo 150

Giriş sağlandıktan sonra öncesinde tanımladığımız bağlantıyı test edebiliriz. Bu işlem için Tablo 150'da ifade edilen komutu girebiliriz. Eğer karşımıza project içerisinde oluşturduğumuz dosyalar çıkıyorsa bağlama işlemi başarılı bir şekilde tamamlanmıştır.

```
goal account list
```

Tablo 151

Yayınlamadan önce Tablo 151'da ifade edilen komut ile sahip olunan local adreslere baktık. Bu adreslerde birisini sonrasında kullanmak için kaydetmemiz gerekmektedir.

```
goal app create --creator
EHIGDIXP6QUWMUQRTTI6W36TKZ3AWUVQEJMZ4BPTE3T32DWWALPVHSE3QY
--approval-prog /data/build/approval.teal --clear-prog /data/build/clear.teal --global-byteslices 1
--global-ints 1 --local-byteslices 0 --local-ints 0
```

Tablo 152

Kontrat Tablo 152'da ifade edilen kod ile terminalden çalıştırılabilirichtetir. Bu kod birkaç kısımdan oluşmaktadır. Bu kısımları teker teker incelememiz gerekirse:

- **goal app create:** Kontratın oluşacağını ifade etmektedir.
- **Creator:** Kontratın oluşturucu hesabını ifade etmektedir. Bu adresi local olarak “goal account list” komutu ile elde edebiliriz.
- **Approval-prog:** “approval.teal” dosyasının konumunu ifade etmektedir.
- **Clear-prog:** “clear.teal” dosyasının bulunduğu konumu ifade etmektedir.
- **Global-byteslices:** Akıllı kontrat içerisinde bulunan global “byteslices” türüne sahip değişkenin sayısını ifade etmektedir. Kontratımız için bu değer “global_tanımlayıcı” olmaktadır.
- **Global-ints:** Akıllı kontrat içerisinde bulunan global “int” türüne sahip değişkenlerin sayısını ifade etmektedir. Kontratımız için bu değer “global_sayıci” olmaktadır.
- **Local-byteslices:** Akıllı kontrat içerisinde bulunan “local byteslices” türüne sahip değişkenlerin sayısını ifade etmektedir.

- **Local-ints:** Akıllı kontrat içerisinde bulunan local “ints” türüne sahip değişkenlerin sayısını ifade etmektedir.

```
goal app info --app-id X
```

Tablo 153

Çalıştırılan kod bir “app-id” değeri vermektedir. Tablo 153’da ifade edildiği üzere bu değeri komutu ile X yerine koyarak çalıştığımızda birçok bilgi çıkmaktadır.

```
Application ID: 1
Application account:
WCS6TVPJRB SARHLN2326LRU5BYVJZUKI2VJ53CAWKYYHDE455ZGKANWMGM
Creator: EHIGDIXP6QUWMUQRTI6W36TKZ3AWUVQEJMZ4BPTE3T32DWWALPVHSE3QY
Approval hash:
WPBQ6TZHTB5FZBL4PVRYMJ5SXTML3AF6FBKFQA7WYMSID5MA5ZEMGXBPPEE
Clear hash: BJATCHES5YJZJ7JITYMVLSSIQAVAWBQRVGPQUDT5AZ2QSLDSXWWM46THOY
Max global byteslices: 1
Max global integers: 1
Max local byteslices: 0
Max local integers: 0
```

Tablo 154

Tablo 154’de bir çıktı örneği ifade edilmiştir. Bu değerler “app” olarak adlandırılan kontratın bilgilerini ifade etmektedir.

```
goal app read --global --app-id X
```

Tablo 155

Aynı zamanda oluşturduğumuz akıllı kontratların depolaması bulunmaktadır. Bu depolamaya Tablo 155’da ifade edildiği şekilde bakılacak olunursa bir json çıktısının oluşturduğu gözlemlenebilir.

```
{
  "sayici": {
    "tt": 2
  },
  "tanimlayici": {
    "tb": "\ufffd\uufffd\uufffd\uufffd\uufffd\uufffd\u0011\uufffd\uufffd\uufffd\u000bR\uufffdVv\u000bR\uufffd\u0005\uufffd&\ufffd\u000e\uufffd\u0002\uufffd",
    "tt": 1
  }
}
```

Tablo 156

Tablo 156'da akıllı kontratın depoladığı JSON verisi ifade edilmektedir. Bu dosya içerisinde tanımladığımız iki tanımlayıcı da yer almaktadır. Tanımlayıcıların içerisinde kaydedilen bu değerler bazı anahtarlar ile ifade edilmiştir. Bu anahtarlar arasından “tb” anahtarları “Transection Bytes” anlamına gelerek tanımlayıcının içerisinde tuttuğu veriyi ifade etmektedir. Başta tanımlaman “sayıcı” değişkeninin değerinin 0 olmasından mütevelli herhangi bir “tb” değerinin oluşturulmasına gerek yoktur. Ancak 0 dışındaki herhangi bir değer için “tb” değeri oluşturulacaktır. İlkinci anahtar kelime “tt” olmaktadır. Bu anahtar “Transection Type” anlamına gelerek tutulan tanımlayıcıların tipini etiketlemektedir. Bu değerin “tt=1” olması kaydedilen değerin byteslices türüne ait olduğunu gösterirken değerin “tt=2” olması kaydedilen değerin “int” türüne sahip olduğunu ifade etmektedir.

```
goal app read --global --app-id X --guess-format
```

Tablo 157

Oluşturulan çıktı ham durumda olmaktadır ve bazı “tanımlayıcı” gibi “byteslices” değerler okunamıyor durumda olmaktadır. Bu durumun çözümlesi için Tablo 157'de ifade edilen şekilde okuma fonksiyonu sonuna “--guess-format” ibaresi eklenebilir.

```
{
  "sayici": {
    "tt": 2
  },
  "tanımlayıcı": {
    "tb": "EHIGDIXP6QUWMUQRTTI6W36TKZ3AWUVQEJMZ4BPTE3T32DWWALPVHSE3QY",
    "tt": 1
  }
}
```

Tablo 158

Tablo 158'de okuma fonksiyonu sonrası “--guess-format” anahtar kelimesi eklenilmesi sonrası oluşan çıktı ifade edilmektedir. Görüldüğü üzere artık “tanımlayıcı” değişkenin “tb” anahtar kelimesine denk gelen değeri 64 karakterden meydana gelen ve kontratı derlerken kullandığımız adres olmaktadır.

Artık “no-op” transfer türü içerisinde kendi istediğimiz özelliklerini ekleyebiliriz. Bu sayede kendi operasyonlarımızı tanımlayabiliriz. Eklenecek bu işlemler “program.event()” fonksiyonu içerisinde init parametresinden sonra yer alacaktır.

```
op_arttir = Bytes("art")
op_azalt = Bytes("az")
```

Tablo 159

Daha öncesinde yazdığımız “step_01.py” dosyası içerisinde kod yazımıma devam ederiz. İlk olarak “approval” fonksiyonu içerisinde yeni değişkenleri Tablo 159’da ifade edildiği üzere tanımlayarak başlayabiliriz. Bu değişkenlerin global olmamasından dolayı tanımlarken global ibaresini eklemeye gerek bulunmamaktadır.

Kitabın önceki bölümlerinde yaptığımız üzere sıralı işlemleri transfer içerisinde yapmak istediğimiz zaman “Seq()” anahtar kelimesini kullanmaktaydık. Ancak transfer içerisinde şartlı koşul olarak “switch case” gibi çalışan bir “Cond()” fonksiyonu kullanabiliriz. Kullanılacak olunan bu “Cond()” fonksiyonu yapısı gereği argümanların yer aldığı array liste yapılarından bir araya gelmektedir. Arrayların ilk elemanı çalışma şartı olacaktır, ikinci eleman ise çalışacak işlem olmaktadır.

Array listesinde bulunan bu şartlar tekil olmaktadır. Eğer bir şart sağlanır ve denk gelen işlemi çalıştırılırsa diğer şartların ne olduğuna bakılmaksızın çalıştırılamaz. Eğer hiçbir şart sağlanmamışsa transfer işlemi gerçekleşmemiş olur ve hata verir.

```
[Txn.application_args[0]= op_arttir, arttir]
```

Tablo 160

Bakılacak ilk şart “Txn.application” listesinde yer alacak ilk elementin arttir fonksiyonuna denk gelip gelmemesidir. Gelmesi halinde Tablo 160’da ifade edildiği üzere sonrasında tanımlayacağımız “arttir” adına sahip arttirma fonksiyonu çalışacaktır.

```
no_op= Cond(
  [Txn.application_args[0]== op_arttir, arttir],
  [Txn.application_args[0]== op_azalt, azalt],
)
```

Tablo 161

Tablo 161’da ifade edildiği üzere aynı yapı azaltma için de gerçekleştiriliyor. Oluşan arraylar ilk olarak belirtildiği gibi şartı almaktadır. Bu şart transferin içerisinde bulunan ilk argümanının hemen üst satırında tanımladığımız “op_azalt” ve “op_arttir” tanımlayıcılarının byte değerlerine denk gelip gelmemesine bakmaktadır. Sonrasında alınan ikinci parametre şar-

tin sağlanması halinde gerçekleştirilecek fonksiyonu ifade etmektedir. Bu fonksiyonları henüz tanımlamamış olsak da birazdan tanımlayacağız.

No_op transfer türü içerisinde gerekli şartların “Cond()” metodu ile eklenmesinden sonra artırma ve azaltma fonksiyonları şartlı durumda tanımladığımız isimler ile oluşturulabilir. Bu fonksiyonlar birden fazla işlem yapacağı için kendi içlerinde “Seq()” metodu ile tanımlanacaktır.

```
arttir = Seq(
    App.globalPut(global_sayici, App.globalGet(global_sayici) + Int(1)),
    Approve()
)
```

Tablo 162

Artırma fonksiyonu her defasında global olarak tanımlanan global_sayici değerini alarak bir artıracaktır. Bu işlemi gerçekleştirmek için Tablo 162’de ifade edildiği üzere bir fonksiyon yazılması gerekmektedir. Bu fonksiyonun işlevlerine bakılacak olunursa ilk olarak “App.globalPut()” fonksiyonu ile kontrat üzerine bir global değer konulmaktadır. Bu fonksiyon içerisinde iki adet değer almaktadır. İlk değer akıllı kontrat üzerine konulacak değişkenin adı olmaktadır. Bu değer Byte türüne sahip olmaktadır ve tanımlar içerisinde “global_sayici” olarak kaydedilmiştir. Sonrasında alınacak ikinci değer kontrat içerisinde aktarılacak bu tanımlayıcının değeri olmaktadır. Artırma fonksiyonu tanımladığımız için önceki değeri alıp bir artırması gerekmektedir. Bu işlemi gerçekleştirmek için “App.globalGet()” komutu ile kontrat üzerinden tanımlayıcıyı almaktadır. Sonrasında üzerine 1 değerine sahip integer eklemektedir. Bu integer tanımlaması yapılrken Python üzerinde bulunan “int()” fonksiyonu yerine Pyteal’ın sağladığı “Int()” fonksiyonunun kullanıldığı dikkat edilmesi gereken bir durumdur. Artırılmış tanımlayıcıya aktarılan aktarılan değerin başarı ile sonuçlanması sonrasında transferin onaylanması gerekmektedir. Bu nedenle son olarak “Approve()” fonksiyonu eklenmektedir.

```
arttir = Seq(
    App.globalPut(global_sayici, App.globalGet(global_sayici) + Int(1)),
    Approve()
)
azalt = Seq(
    App.globalPut(global_sayici, App.globalGet(global_sayici) - Int(1)),
    Approve()
)
```

Tablo 163

Arttırma fonksiyonunun tanımlaması sonrasında aynı işlem azaltma için de gerçekleştirilecektir. Ancak bu sefer farklılık olarak Tablo 163'te ifade edildiği üzere kontrat üzerinden alınan “global_sayici” değeri bir artırılmayacak olunup bir azaltılacaktır. Sonrasında kod hazır hale gelecektir.

```
from pyteal import *
from pyteal_helpers import program

def approval():
    global_sayici = Bytes("sayici")
    global_tanimlayici = Bytes("tanimlayici")

    op_arttir = Bytes("art")
    op_azalt = Bytes("az")

    arttir = Seq(
        App.globalPut(global_sayici, App.globalGet(global_sayici) + Int(1)),
        Approve()
    )
    azalt = Seq(
        App.globalPut(global_sayici, App.globalGet(global_sayici) - Int(1)),
        Approve()
    )
    return program.event(
        init= Seq(
            App.globalPut(global_sayici, Int(0)),
            App.globalPut(global_tanimlayici, Txn.sender()),
            Approve()
        ),
        no_op=Cond(
            [Txn.application_args[0] == op_arttir, arttir],
            [Txn.application_args[0] == op_azalt, azalt],
        )
    )
def clear():
    return Approve()
```

Tablo 164

Tablo 164'te step_01.py dosyasını tamamı ifade edilmiştir. Sırasıyla yapılan işlemlere bakılacak olunursa:

- Gerekli kütüphaneler dosya içeresine eklenir
- Approval ve clear adında iki fonksiyon tanımlanır.
- Approval fonksiyonu içeresine global olarak “sayici” ve “tanimlayici” değişkenleri eklenir.
- Fonksiyonların terminal üzerinden çağırılmasını sağlayacak byte “arttir” ve “azalt” operatörü tanımlanır.

- Arttırma ve azaltma fonksiyonu yazılır.
- Kontratın başlangıçta sahip olacağı değerler tanımlanır.
- No_op transfer türü içerisinde şartlı olarak artırma ve azaltma şartları girilir.
- Clear fonksiyonuna gelen tüm transferler onaylanır.

```
./build.sh contracts.counter.step_01
```

Tablo 165

Kontrat yazılmış ve derlenmeye hazır bir hale sahip olmuştur. Derleme işlemini gerçekleştirmek için Tablo 165'te ifade edildiği üzere “./build.sh” dosyası içerisinde kodun bulunduğu adresin girilmesi gerekmektedir. Bu sadece kod derlenir ve “.teal” uzantılı dosyalar oluşur. Komutu çalıştırmak için Docker konteynerlerinden çıkış yapılip project klasörüne giriş yapılması gerekmektedir.

```
#pragma version 5
txn ApplicationID
int 0
==
bnz main_l16
txn OnCompletion
int DeleteApplication
==
bnz main_l15
txn OnCompletion
int UpdateApplication
==
bnz main_l14
txn OnCompletion
int OptIn
==
bnz main_l13
txn OnCompletion
int CloseOut
==
bnz main_l12
txn OnCompletion
int NoOp
==
bnz main_l7
err
main_l7:
txna ApplicationArgs 0
byte "art"
==
bnz main_l11
txna ApplicationArgs 0
byte "az"
==
```

```

bnz main_l10
err
main_l10:
byte "sayici"
byte "sayici"
app_global_get
int 1
-
app_global_put
int 1
return
main_l11:
byte "sayici"
byte "sayici"
app_global_get
int 1
+
app_global_put
int 1
return
main_l12:
int 0
return
main_l13:
int 0
return
main_l14:
int 0
return
main_l15:
int 0
return
main_l16:
byte "sayici"
int 0
app_global_put
byte "tanimlayici"
txm Sender
app_global_put
int 1
return

```

Tablo 166 approval.teal

Tablo 166'da "approval.teal" dosyasında "step_01.py" dosyası ile oluşturduğumuz akıllı kontratın derlenmiş hali yer almaktadır. Bu dosya çokunlukla bir önceki dosyaya benzerlikler göstermektedir. Ancak farklılıklar da bulunmaktadır. Bu farklılıklar mainl10 ve mainl11 üzerinde kendilerini göstermektedir.

```
./sandbox enter algod
```

Tablo 167

Derlenen teal kodu test edilmeye hazır bir hal almaktadır. Test işlemini gerçekleştirmek için Tablo 167'de ifade edildiği üzere algod olarak adlandırılan Docker konteynerlarına Sandbox ile giriş sağlamamız gerekmektedir. Bu komutu çalıştmak için Sandbox klasörü içerisinde girilmesi gerekmektedir.

```
goal account list
```

Tablo 168

Docker konteynerleri içerisinde girildikten sonra bu işlemin başarılı olup olmadığına Tablo 168'da ifade edilen şekilde bakılabilir. Bu liste içerisinde local adreslerin herhangi biri sonradan kullanılmak için kaydedilmektedir.

```
goal app create --creator
EHIGDIXP6QUWMUQRITI6W36TKZ3AWUVQEJMZ4BPTE3T32DWWALPVHSE3QY
--approval-prog /data/build/approval.teal --clear-prog /data/build/clear.teal --global-byteslices 1
--global-ints 1 --local-byteslices 0 --local-ints 0
```

Tablo 169

Lokalde çalıştırılmaya hazır olan kodu Tablo 169'da ifade edilen komutu çalıştırabiliriz. Sonrasında karşımıza derlenmiş bir kod çıkmaktadır. Pyteal bu kontrata kendi içerisinde bir numara atanmıştır ve bu numarayı kullanarak kontrata erişebiliriz. Çalıştırma kodunun içeriğine bakılması gerekmektedir. Bu kısımları teker teker incelememiz gerekirse:

- **Goal app create:** Kontratın oluşacağını ifade etmektedir.
- **Creator:** Kontratın oluşturucu hesabını ifade etmektedir. Bu adresi local olarak goal account list komutu ile elde edebiliriz.
- **Approval-prog:** “approval.teal” dosyasının konumunu ifade etmektedir.
- **Clear-prog:** “clear.teal” dosyasının bulunduğu konumu ifade etmektedir.
- **Global-byteslices:** Akıllı kontrat içerisinde bulunan global “byteslices” türüne sahip değişkenin sayısını ifade etmektedir. Kontratımız için bu değer “global_tanimlayici” olmaktadır.
- **Global-ints:** Akıllı kontrat içerisinde bulunan global “int” türüne sahip değişkenlerin sayısını ifade etmektedir. Kontratımız için bu değer “global_sayici” olmaktadır.
- **Local-byteslices:** Akıllı kontrat içerisinde bulunan “local byteslices” türüne sahip değişkenlerin sayısını ifade etmektedir.

- **Local-ints:** Akıllı kontrat içerisinde bulunan “local ints” türüne sahip değişkenlerin sayısını ifade etmektedir.

```
goal app call --app-id X --app-arg "str:art" --from
EHIGDIXP6QUWMUQRTT16W36TKZ3AWUVQEJMZ4BPTE3T32DWWALPVHSE3QY
```

Tablo 170

Kontratın derlenmesinden ve uygun uygulama kodunun atanmasından sonra fonksiyonlar denenebilir. Deneme işlemini gerçekleştirmek için Tablo 170'da ifade edilen terminal komutunu çalıştırılmamız gerekmektedir. Bu komuta parça parça bakılacak olunursa:

- **App-id:** Bir üst kısımda bulunan kodu çalıştırılmamız halinde atacak uygulama numarasını ifade etmektedir. X yerine bu numara yazılmalıdır.
- **--app-arg:** Kontrat içerisinde gönderilecek argümanları ifade etmektedir. Bu değerin ilk olarak türünün belirlenmesi gerekmektedir. Bizim projemiz için bu string türü olmaktadır. Sonrasında bu değişkenin bağlanacağı Byte değeri girilmelidir. “op_arttir = Bytes(“art”)” tanımlaması içerisinde bu değeri “art” olarak belirlediğimiz için komut içerisinde de “art” kelimesini kullanmaktadır.
- **--from:** Kontrat içerisinde argümanı gönderecek kullanıcının adresini ifade etmektedir. Local olarak tanımladığımız adreslerin biri kullanılabilir.

```
goal app read --global --app-id 12 --guess-format
```

Tablo 171

Sonrasında sonucu görüntülemek için Tablo 171'da ifade edilen komutu kullanabiliriz. Bu işlem kontrat depolaması görüntülenebilir.

```
{
  "sayici": {
    "tt": 2,
    "ui": 1
  },
  "tanimlayici": {
    "tb": "EHIGDIXP6QUWMUQRTT16W36TKZ3AWUVQEJMZ4BPTE3T32DWWALPVHSE3QY",
    "tt": 1
  }
}
```

Tablo 172

Tablo 172'de ifade edildiği üzere öncesinde var olmayan “ui” değeri oluşmuştur. Bu değer aslında kontrat oluşturulurken 0 olarak atanmıştır. Ancak artırma fonksiyonu gereği bu sıfır değeri üzerine bir eklenerek “ui”: 1 olmaktadır.

```
{
  "sayici": {
    "tt": 2,
    "ui": 2
  },
  "tanimlayici": {
    "tb": "EHIGDIXP6QUWMUQRTTI6W36TKZ3AWUVQEJMZ4BPTE3T32DWWALPVHSE3QY",
    "tt": 1
  }
}
```

Tablo 173

Fonksiyon yeniden çağrıldığında “ui” değerinin bir arttığı görülebilir. Bu şekilde kontrat içerisinde yer alan değerler Tablo 173'de ifade edildiği gibi değiştirilebilir.

```
goal app call --app-id 12 --app-arg "str:az" --from
EHIGDIXP6QUWMUQRTTI6W36TKZ3AWUVQEJMZ4BPTE3T32DWWALPVHSE3QY
```

Tablo 174

Aynı şekilde artırma işlemi öncesinde tanımlanan azaltma işlemi için de uygulanabilir. Yapılması gereken tek değişiklik Tablo 174'de gösterildiği üzere “str:art” değeri yerine “str:az” değerinin kullanılması gerekmektedir. Sonrasında değişiklik kontrat depolamasına yansımaktadır.

```
{
  "sayici": {
    "tt": 2,
    "ui": 1
  },
  "tanimlayici": {
    "tb": "EHIGDIXP6QUWMUQRTTI6W36TKZ3AWUVQEJMZ4BPTE3T32DWWALPVHSE3QY",
    "tt": 1
  }
}
```

Tablo 175

Tablo 175'da ifade edildiği üzere azaltma komutunu çağrılmamız halinde “ui”:2 olan değer bir azalarak “ui”: 1 değerine azalacaktır. Bu şekilde yazdığımız sayma kodunun başarılı bir şekilde çalıştığı görüntülenebilir.

Bu basamaklar ile Pyteal ile Algorand akıllı kontratların nasıl kurulduğunu, nasıl test edildiğini, hangi veri tiplerine sahip oldukları ve kullanım alanlarını görüntülemiştir olmaktadır.

KAYNAKÇA

“Where Do I Start? - Algorand Developer Portal.” *Where Do I Start? - Algorand Developer Portal*, developer.algorand.org/docs/get-started/basics/where_to_start. Son Erişim Tarihi: 2 Ekim 2022

“Docker Nedir? Avantajları Nelerdir? | İşNet Blog.” *Docker Nedir? Avantajları Nelerdir? | İşNet Blog*, www.isnet.net.tr/BlogIcerik/docker-nedir-ve-avantajlari-nelerdir-isnet-blog. Son Erişim Tarihi: 2 Ekim 2022

“Download Python.” *Python.org*, 6 Eylül. 2022, www.python.org/downloads.

“Home - Docker.” *Docker*, 10 Mayıs 2022, www.docker.com.

algorand. “GitHub - Algorand/Pyteal: Algorand Smart Contracts in Python.” *GitHub*, 20 Eylül. 2022, github.com/algorand/pyteal.

algorand. “GitHub - Algorand/Py-algorand-sdk: Algorand Python SDK.” *GitHub*, 19 Eylül. 2022, github.com/algorand/py-algorand-sdk.

Node.js. “Node.js.” *Node.js*, nodejs.org/en. Son Erişim Tarihi: 2 Ekim 2022

“Visual Studio Code - Code Editing. Redefined.” *Visual Studio Code - Code Editing. Redefined*, code.visualstudio.com. Son Erişim Tarihi: 2 Ekim 2022

“Algorand (ALGO) Blockchain Explorer.” *Algorand (ALGO) Blockchain Explorer*, algoexplorer.io. Son Erişim Tarihi: 2 Ekim 2022

“GoalSeeker by PureStake | Algorand Block Explorer.” *GoalSeeker by PureStake | Algorand Block Explorer*, goalseeker.purestake.io/algorand/testnet. Son Erişim Tarihi: 2 Ekim 2022

“AlgoSigner - Chrome Web Store.” *AlgoSigner - Chrome Web Store*, chrome.google.com/webstore/detail/algosigner/kmmolakhbgdlpkjkcjkebenjheonagdm. Son Erişim Tarihi: 2 Ekim 2022

“Algorand Dispenser.” *Algorand Dispenser*, bank.testnet.algorand.network. Son Erişim Tarihi: 2 Ekim 2022

“Javascript İle Algorand Eğitimi | Udemy.” *Udemy*, www.udemy.com/course/javascript-algorand. Son Erişim Tarihi: 2 Ekim 2022