

Máster Universitario en Tecnologías del Sector Financiero
2019-2020

Práctica 3

“Aplicaciones sobre blockchain
mediante contratos inteligentes”

D. Álvaro Andrés Suárez Alfonso

Maestro

Dr. José María de Fuentes García-Romero de Tejada

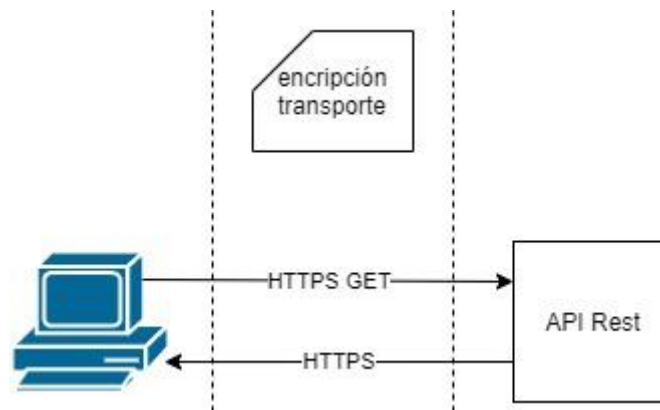
Puerta de Toledo, 2020

Introducción

En este documento se plasma el desarrollo de la práctica 3 para la materia de Blockchain y Tecnologías de Seguridad, en la que con el uso de una aplicación Web se ejecutan diferentes funciones de un contrato inteligente para interactuar con una blockchain local en ambiente de desarrollo.

Arquitectura de aplicación

La aplicación se basa en un API Rest desarrollada con NodeJS que esta securizada en la capa de transporte con el protocolo HTTPS como se muestra en el siguiente diagrama adhoc:



El API Rest consta de dos componentes: por un lado, el servidor que se encarga de exponer la aplicación en un servidor https y, por otro, la app que contiene los métodos que se consumen a través de una petición. La app consta de los siguientes métodos:

Método	Objetivo	Parámetros	Salida
getAddressess	Obtener las direcciones de la blockchain	N/A	Direcciones
getBalance	Obtener el balance de una dirección	from: Dirección a consultar	Balance disponible
depositFunds	Depositar balance en una dirección específica	from: Dirección en la que se va a depositar wei: Ether que se van a depositar en unidad de medida wei	Confirmación de deposito

transferFunds	Transferir balance de una dirección a otra	from: Dirección ejecutante de la transferencia receiver: Dirección recibiente wei: Ether transferidos en unidad wei	Confirmación de transferencia
withdrawFunds	Extraer balance de una dirección	from: Dirección en a la que se va a extraer balance wei: Ether que se van a extraer en unidad de medida wei	Confirmación de extracción

Diseño contrato inteligente

El proyecto truffle consta de un contrato inteligente que simula un cajero automático para gestionar los fondos de las cuentas de forma básica y, como vamos a ver ahora, algunos métodos expuestos en el API tienen su función de contrato asociada:

Función	Objetivo	Parámetros	Salida
deposit	Depositar balance en una dirección específica	N/A	N/A
transfer	Transferir balance de una dirección a otra	receiver: Dirección recibiente amount: Ether transferidos en unidad wei	N/A
withdraw	Extraer balance de una dirección	wei: Ether que se van a extraer en unidad de medida wei	N/A

Así mismo el contrato emite un evento asociado a cada función, de la siguiente forma:

Evento	Parametros
Deposit	sender: Dirección remitente amount: Ether transferidos en unidad wei
Transfer	sender: Dirección remitente amount: Ether transferidos en unidad wei
Withdrawal	receiver: Dirección extractora de fondos wei: Ether que se van a extraer en unidad de medida wei

Testing

Se hicieron distintas pruebas consumiendo los métodos del contrato para verificar su funcionalidad:

Se verificaron transacciones en 0 para el contrato:

NAME	ADDRESS	TX COUNT
ATM	0x09949abdf0fC5d5C24Ca782A2c8147CaadCAd8AF	0

Se verificaron 0 eventos para el contrato ATM:

Ganache

ACCOUNTS

BLOCKS

TRANSACTIONS

CONTRACTS

EVENTS

LOGS

SEARCH FOR BLOCK NUMBERS OR TX HASHES

CURRENT BLOCK
199

GAS PRICE
20000000000

GAS LIMIT
6721975

HARDFORK
MUIRGLACIER

NETWORK ID
5777

RPC SERVER
HTTP://127.0.0.1:7545

MINING STATUS
AUTOMINING

WORKSPACE
AXXA-CAN

SWITCH

EVENT NAME

Encoded Event

CONTRACT

0x398bc153c8160aedb079c2601e503db12058d87C

TX HASH

0x9e37f03149b6b2a24c94621bcc8d9262e6df1d480991a4d36cee306eb1d31a19

LOG INDEX

0

BLOCK TIME

2020-05-15 22:09:50

EVENT NAME

Encoded Event

CONTRACT

0x398bc153c8160aedb079c2601e503db12058d87C

TX HASH

0xfea2aae5afe7a05f3288ee5d3bdf0001ba419cd565008efbfbe193cd678c157e

LOG INDEX

0

BLOCK TIME

2020-05-15 21:03:49

EVENT NAME

Encoded Event

CONTRACT

0x398bc153c8160aedb079c2601e503db12058d87C

TX HASH

0xda382ae15f15313b1cf285e9e8841020b45f7b6a3b202f19e8c315002975651a

LOG INDEX

0

BLOCK TIME

2020-05-15 21:03:33

Verificación de denegación para canal http:

```
PS C:\Users\asan1> curl -v http://localhost:3000/getAddresses
VERBOSE: GET http://localhost:3000/getAddresses with 0-byte payload
curl : The underlying connection was closed: The connection was closed unexpectedly.
At line:1 char:1
+ curl -v http://localhost:3000/getAddresses
+ ~~~~~
+ CategoryInfo          : InvalidOperation: (System.Net.HttpWebRequest:HttpWebRequest) [Invoke-WebRequest], WebExc
eption
+ FullyQualifiedErrorId : WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWebRequestCommand
```

Se obtienen las direcciones de la red:

```
C:\Users\asan1>curl -k -s -o response.txt -w "%{http_code}" https://localhost:3000/getAddresses
201
C:\Users\asan1>
```

Y en el fichero response.txt se encuentran las siguientes direcciones:



Ahora obtenemos el balance de la primera dirección:

```
C:\Users\asan1>curl -k -s -o response.txt -w "%{http_code}" https://localhost:3000/getBalance/0xC5AA9c85CF955275267021b461e7eE4Aa70Bf3b0/2000
201
C:\Users\asan1>
```

```
{"balance":"2000000000000000000"}
```

Que corresponden a 0.02 ether.

Luego, hacemos un depósito de 2 eth:

```
C:\Users\asan1>curl -k -s -o response.txt -w "%{http_code}" https://localhost:3000/depositFunds/0xC5AA9c85CF955275267021b461e7eE4Aa70Bf3b0/2000
000000000000000000000000
201
C:\Users\asan1>
```

Y verificamos el balance nuevamente:

```
C:\Users\asan1>curl -k -s -o response.txt -w "%{http_code}" https://localhost:3000/getBalance/0xC5AA9c85CF955275267021b461e7eE4Aa70Bf3b0
201
C:\Users\asan1>
```

```
{"balance":"2020000000000000000"}
```

Ahora verificamos el balance de la segunda cuenta:

```
C:\Users\asan1>curl -k -s -o response.txt -w "%{http_code}" https://localhost:3000/getBalance/0x77aB069124b88Bc5f79A16126905cf1B5EEa6B37
201
C:\Users\asan1>
```

```
{"balance":"0"}
```

Y hacemos una transferencia a esta cuenta por 0.001 eth desde nuestra primera cuenta:

```
C:\Users\asan1>curl -k -s -o response.txt -w "%{http_code}" https://localhost:3000/transferFunds/0xC5AA9c85CF955275267021b461e7eE4Aa708F3b0/0x77aB069124b88Bc5f79A16126905cf185EEa6B37/1000000000000000
201
C:\Users\asan1>
```

Verificamos el balance de la primera cuenta:

```
{"balance": "20190000000000000000"}
```

Verificamos el balance de la segunda cuenta:

```
{"balance": "10000000000000000"}
```

Ahora vamos a extraer 0.0001 eth de la segunda cuenta:

```
C:\Users\asan1>curl -k -s -o response.txt -w "%{http_code}" https://localhost:3000/withdrawFunds/0x77aB069124b88Bc5f79A16126905cf185EEa6B37/1000000000000000
201
C:\Users\asan1>
```

Y verificamos su balance:

```
{"balance": "9000000000000000"}
```

Ahora vamos a intentar extraer 0.01 eth de la segunda cuenta:

Y nos retorna:

Error: Returned error: VM Exception while processing transaction: revert Insufficient funds

Volvemos a verificar el balance:

```
{"balance": "9000000000000000"}
```

En la interfaz del contrato se puede ver el log de transacciones y eventos:

TRANSACTIONS				
TX HASH 0*84f99dd54a769579b265193a36d8c8bd401e0e8b71f0d8a6c6d3373aa64ab81b				
FROM ADDRESS 0xC5AA9c85CF955275267021b461e7eE4Aa708F3b0	TO CONTRACT ADDRESS ATM	GAS USED 43505	VALUE 2000000000000000000	CONTRACT CALL
TX HASH 0*a92c4bf54bf6026055370fd9806dfc430fb48e7900ae9c61bd00dddb13bd4799				
FROM ADDRESS 0xC5AA9c85CF955275267021b461e7eE4Aa708F3b0	TO CONTRACT ADDRESS ATM	GAS USED 51325	VALUE 0	CONTRACT CALL
TX HASH 0*360f1e7df2e13ea7b40ecc3c780187b45aa47d1ae479b94c380125c7de93bc38				
FROM ADDRESS 0*77aB069124b88Bc5f79A16126905cf185EEa6B37	TO CONTRACT ADDRESS ATM	GAS USED 29674	VALUE 0	CONTRACT CALL
EVENTS				
EVENT NAME Deposit				
CONTRACT ATM	TX HASH 0*84f99dd54a769579b265193a36d8c8bd401e0e8b71f0d8a6c6d3373aa64ab81b	LOG INDEX 0	BLOCK TIME 2020-05-16 10:21:16	
EVENT NAME Transfer				
CONTRACT ATM	TX HASH 0*a92c4bf54bf6026055370fd9806dfc430fb48e7900ae9c61bd00dddb13bd4799	LOG INDEX 0	BLOCK TIME 2020-05-16 10:22:06	
EVENT NAME Withdrawal				

ATM

BALANCE
2.00 ETH

CREATION TX
0x2f258fb18b1Ee7B6bc5adE2cf471bd0773584f1Ff026D6dfac9E87E35F71f2db

STORAGE

```
▼ { 1 item
  | ► balances : {} mapping 0 items
  }
```

Igualmente se visualizan los bloques de estas transacciones, por ejemplo:

BLOCK 214

MINED ON
2020-05-16 10:22:30

FROM ADDRESS
0x77aB069124b88Bc5f79A16126905cf1B5EEa6B37

TO CONTRACT ADDRESS
ATM

La cual corresponde a un retiro (función `withdraw` del contrato):

[— BACK](#)

CONTRACT CALL

MINED IN BLOCK
214[illegible]

CONTRACT

```

INPUTS
10000000000000000

```

EVENTS

LOG INDEX	BLOCK TIME
0	2020-05-16 10:22:30

Instrucciones de despliegue de aplicación

Prerrequisitos:

- Tener como mínimo nodeJS v12.14 instalado, el cual se puede descargar de <https://nodejs.org/es/>
- Establecer la URL donde se encuentra la blockchain en el fichero app.js:

```
const url = 'HTTP://127.0.0.1:7545';
```

Descomprimir el fichero practica3 y dentro de la carpeta node_project ejecutar el siguiente comando:

```
npm install
```

Este comando instalará las dependencias necesarias para correr la aplicación, las cuales están referenciadas en package.json

Posterior a que acabe la instalación, correr el siguiente comando:

```
npm run start:server
```

Este comando correrá el fichero server.js el cual inicializa el API y la mantiene arriba para que pueda ser consumida desde el puerto 3000.

Ejemplos de consumo del API:

```
curl -k -s -o response.txt -w "%{http_code}" https://localhost:3000/getAddresses
```

```
curl -k -s -o response.txt -w "%{http_code}"  
https://localhost:3000/getBalance/0xC5AA9c85CF955275267021b461e7eE4Aa70Bf3b0
```

```
curl -k -s -o response.txt -w "%{http_code}"  
https://localhost:3000/depositFunds/0xC5AA9c85CF955275267021b461e7eE4Aa70Bf3b0/20000000000  
00000000
```

```
curl -k -s -o response.txt -w "%{http_code}"  
https://localhost:3000/getBalance/0x77aB069124b88Bc5f79A16126905cf1B5EEa6B37
```

```
curl -k -s -o response.txt -w "%{http_code}"  
https://localhost:3000/withdrawFunds/0x77aB069124b88Bc5f79A16126905cf1B5EEa6B37/1000000000  
0000000
```


Observaciones y dificultades

Se presentaron dificultades en el uso del contrato y la verificación del balance en Ganache, y se encuentra en las funciones transfer y deposit del contrato ATM las cuales se pueden verificar a la hora de pedir el balance desde el contrato, pero no se puede verificar desde la interfaz grafica de ganache. A continuación, la implementación del contrato:

```
pragma solidity 0.5.16;

contract ATM {

    mapping(address => uint) public balances;

    event Deposit(address sender, uint amount);
    event Withdrawal(address receiver, uint amount);
    event Transfer(address sender, address receiver, uint amount);



    function deposit() public payable {
        emit Deposit(msg.sender, msg.value);
        balances[msg.sender] += msg.value;
    }

    function withdraw(uint amount) public {
        require(balances[msg.sender] >= amount, "Insufficient funds");
        emit Withdrawal(msg.sender, amount);
        balances[msg.sender] -= amount;
    }

    function transfer(address receiver, uint amount) public {
        require(balances[msg.sender] >= amount, "Insufficient funds");
        emit Transfer(msg.sender, receiver, amount);
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
    }
}
```

Como se puede ver hay una variable publica que apunta al storage de balances por address.

Inicialmente tenemos las dos siguientes direcciones con sus balances:

ADDRESS	BALANCE	TX COUNT	INDEX	
0x5E65fD815CBB2525c10ba6642Cb39ca5E3e2814b	99.98 ETH	5	3	
ADDRESS	BALANCE	TX COUNT	INDEX	
0x498D57Af2481b788369b01D21Dfc50c93323FB6e	100.00 ETH	2	4	

Inicialmente produzco un withdraw de la segunda cuenta para verificar que se vea reflejado en Ganache:

Extraigo 0.1 eth:

```
curl -k -s -o response.txt -w "%{http_code}"
https://localhost:3000/withdrawFunds/0x498D57Af2481b788369b01D21Dfc50c93323FB6e/100000000
000000000
```

y en logs sale error:


- [10:02:29 PM] Transaction:
0x20a6cdfe956d4fed7bd849823b91038208087a33d6f52daf26e7c2ca046b4d8f
- [10:02:29 PM] Gas usage: 22518
- [10:02:29 PM] Block Number: 220
- [10:02:29 PM] Block Time: Tue May 19 2020 22:02:29 GMT+0200 (hora de verano de Europa central)
- [10:02:29 PM] Runtime Error: revert
- [10:02:29 PM] Revert reason: Insufficient funds


Al parecer el balance solo se puede hacer un withdraw si la dirección cuenta con balance dentro del contrato, ya que al hacer la verificación del balance para esta dirección usando el contrato obtengo lo siguiente:

```
{"balance":"0"}
```

De igual forma sucede con la función transfer, ya que si la dirección no tiene fondos en el contrato no puede hacer la transferencia a menos que le ingrese fondos con la función de deposit.

He intentado cambiar el contrato de forma que las funciones withdraw y transfer implementen de payable. Igualmente desde la api de Nodejs en el cual uso la abstracción de web3js para instanciar el smart contract, he llamado los métodos del contrato de dos formas que veo son las únicas posibles; method.NombreMetodo.call y method.NombreMetodo.send, obteniendo el mismo resultado. También he borrado la carpeta build de truffle y vuelto a compilar y desplegar el contrato.

ADDRESS	BALANCE	TX COUNT	INDEX	
0x498D57Af2481b788369b01D21Dfc50c93323FB6e	100.00 ETH	2	4	

ADDRESS	BALANCE	TX COUNT	INDEX	
0x498D57Af2481b788369b01D21Dfc50c93323FB6e	99.90 ETH	3	4	

```
{"balance":"10000000000000000000"}
```

```
curl -k -s -o response.txt -w "%{http_code}"  
https://localhost:3000/transferFunds/0x498D57Af2481b788369b01D21Dfc50c93323FB6e/0x77aB06912  
4b88Bc5f79A16126905cf1B5EEa6B37/1000000000000000000
```

- [10:34:39 PM] eth_sendTransaction
- [10:34:39 PM] Transaction:
0x4d42304bc6cf95263bdda06c3e1849d40115926ea05084eb0c457533343d7c2d
- [10:34:39 PM] Gas usage: 21325
- [10:34:39 PM] Block Number: 222
- [10:34:39 PM] Block Time: Tue May 19 2020 22:34:39 GMT+0200 (hora de verano de Europa central)
- [10:34:39 PM] eth_getBlockByNumber