# Introducción

Vamos a modificar el contrato anterior, añadiremos funcionalidades relativas a **transferencias**.

Las transferencias se realizan por la blockchain, pueden utilizar BNB o tokens construidos sobre la plataforma.

No todas las criptomonedas funcionan sobre la misma red de blockchain, por ejemplo, no podríamos usar ETH ni BTC sobre la red Binance Smart Chain.

## Funciones que reciben cryptomoneda

Lo primero que vamos a hacer es que haga falta entregar **0.05 BNB** para comprar un tiket.

**msg.value** permite saber la cantidad token enviado en una transacción (no estamos hablando del gas si no de moneda realmente enviada), da la cantidad en **Wei. Wei** es básicamente una fracción de la **criptomoneda de la red**, evita el uso de decimales.

1000000000000000000 (18 ceros) **wei** es = a **1 ETH**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

<https://coinguides.org/ethereum-unit-converter-gwei-ether/>

Se ha diseñado así porque ETH admite hasta 18 decimales, en nuestro caso la moneda nativa de la red es **BNB**, que solo admite 8 decimales, así que solo podremos usar las primeras 8 cifras, el resto deberían ser 0.

Por lo tanto **0.05 BNB/Ether** deberían ser = 50000000000000000 (se puede usar el conversor anteriormente citado) También hay utilidades dentro de solidity, utilizando las palabras clave **ether (equivaldría a un BNB)**

Vamos a crear la variable **balanceWei** para contabilizar cuanta moneda hay en el contrato. La variable será de tipo **uint** (entero sin signo), es muy importante porque luego vamos a sumarle los valores recibidos que vienen en ese mismo tipo de dato.

|  |
| --- |
| contract MyContract {      address[16] public tikets;      uint public balanceWei = 0;        function buyTiket(uint tiketIndex) public returns (bool) { |

Lo siguiente va a ser modificar la función **buyTiket()** hay que hacer que la función sea **payable,** significa que va a poder recibir fondos de cryptomoneda, la cantidad recibida vienen en el parámetro **msg.value**.

Lo primero que hago es sumar la cantidad de cryptmoneda que ha llegado **(msg.value)** al **balanceWei**.

El usuario solo gastará la cantidad de criptomoneda enviada en el **msg.value** si el método acaba de ejecutarse si sale por un **require** NO gastará nada

|  |
| --- |
| function buyTiket(uint tiketIndex) payable public returns (bool) {          // Comprobación          require(tiketIndex >= 0 && tiketIndex <= 15);          require(msg.value == 0.05 ether, "Insuficient amount of BNB");          balanceWei += msg.value;          bool sucess = true; |

El usuario solo gastará la cantidad de criptomoneda enviada en el **msg.value** si el método acaba de ejecutarse si sale por un **require** NO gastará nada

Una vez recibido el BNB va a quedar almacenado en el contrato inteligente. Los contratos tienen una dirección igual que las wallets. Vamos a implementar un método para poder transferir toda la cryptomoneda que hay en el contrato.

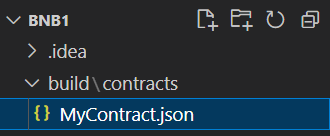
Lo primero va a ser anotar la dirección del **admin**. Será la única persona que podrá ejecutar la función de retirada de fondos. Declaramos, una variable **admin** de tipo **address payable**. El tipo de dato **address payable** es una dirección de una wallet a la que se podrán enviar transferencias.

Creamos la función **withdrawBalance()**, en primer lugar comprueba que el usuario que la llamo **msg.sender** es el **admin,** si es correcto va a llamar al método **transfer(wei)** que hay en las variables de tipo **address payable.** Va a transferir toda la cryptomoneda y poner a 0 la variable **balanceWei.**

|  |
| --- |
| contract MyContract {      address[16] public tikets;      uint public balanceWei = 0;      address payable public admin  = payable(0x50d3b84a2b1787b7704DD50c67733b6b5B293629);  function withdrawBalance () public {  require(msg.sender == admin, "No eres el admin");    admin.transfer(balanceWei);  balanceWei = 0;  } |

## Redesplegar el contrato

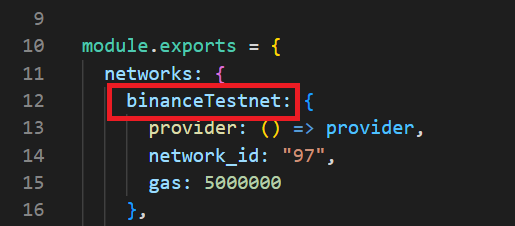
En primer paso **eliminamos** el contenido de la carpeta **build/contracts** generados en despliegues anteriores. Recordamos que hay se guardaban los **json** para conectarnos a los contratos.

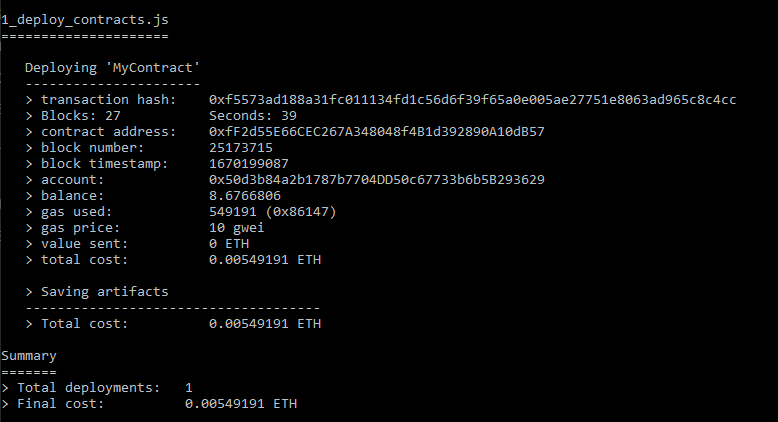


Redesplegamos el contrato, abrimos un **cmd** en el directorio del proyecto y ejecutamos

truffle migrate --reset --network binanceTestnet

El nombre “binanceTestnet” tiene que coincidir con el nombre del fichero **truffle-config.js**



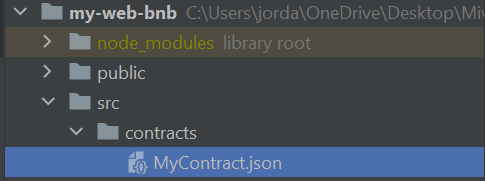


La parte final de la consola indicará que los contratos se han **desplegado correctamente**.

## Aplicación Web

Copiamos el fichero .json generado en el **build/contracts** tras desplegar los contratos.

Pegamos los **ficheros.json** en nuestra aplicación web **/my-web-bnb/** en el directorio /**src/contracts**, estos ficheros contienen los datos para que la librería **Ether** se conecte de forma sencilla al contrato inteligente.



Accedemos a **index.js**.

Vamos a utilizar nuevas funcionalidades de la librería ETHERS

<https://docs.ethers.io/v5/getting-started/>

Modificamos la función **clickBuyTicket,** tiene que enviar 0.05 BNB cuando se ejecuta. Solo hay que añadir un parámetro más a la llamada de **buyTicket(id)** en el contrato inteligente.

Se añade un objeto con el atributo **value**, y los **0.05 BNB** convertidos a wei, usando las utilidades de la librería **ethers**.

|  |
| --- |
| let clickBuyTiket = async (i) => {  const tx = await myContract.current.buyTiket(i, {  value: ethers.utils.parseEther("0.05")  });  await tx.wait();   const tiketsUpdated = await myContract.current.getTikets();  setTikets(tiketsUpdated); } |

Al ejecutarse esta función se va a realizar una transferencia de cryptomoneda, como todas las transferencias tienen asociado un gasto de **gas**, por defecto usa el gas que sea necesario, pero podríamos limitarlo añadiendo parámetros opcionales, como **gasLimit** y **gasPrice**.

|  |
| --- |
| let clickBuyTiket = async (i) => {  const tx = await myContract.current.buyTiket(i, {  value: ethers.utils.parseEther("0.05"),  gasLimit: 6721975,  gasPrice: 20000000000,  });  await tx.wait();   const tiketsUpdated = await myContract.current.getTikets();  setTikets(tiketsUpdated); } |

Ahora vamos a crear una función **clickWithdrawBalance()** para retirar el balance almacenado en el contrato inteligente, esta función va a ser llamada desde un botón en la vista. Su contenido será muy simple, solamente accede al contrato y ejecuta el **withdrawBalance();**

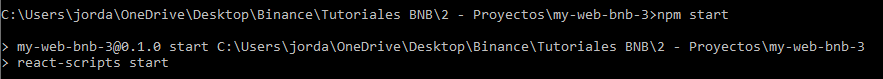
|  |
| --- |
| let withdrawBalance = async () => {  const tx = await myContract.current.withdrawBalance(); }  return (  <div>  <h1>Tikets store</h1>  <ul>  { tikets.map( (address, i) =>  <li>Tiket { i } comprado por { address }  { address == ethers.constants.AddressZero &&  <a href="#" onClick={()=>clickBuyTiket(i)}> buy</a> }  </li>  )}  </ul>  </div> ) |

Modificamos la vista JSX del componente para añadir un nuevo botón que ejecute **clickWithdrawBalance**

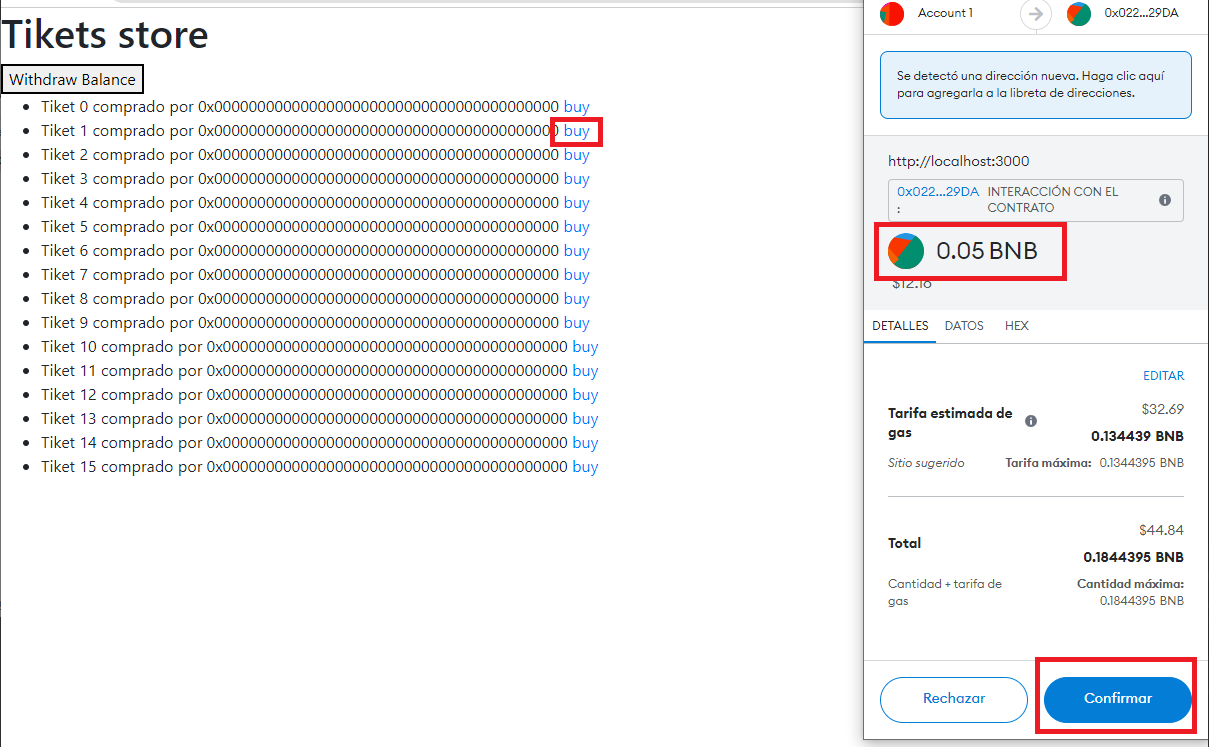
|  |
| --- |
| return (  <div>  <h1>Tikets store</h1>  <button onClick={() => clickWithdrawBalance()}>Withdraw Balance</button>  <ul>  { tikets.map( (address, i) =>  <li>Tiket { i } comprado por { address }  { address == ethers.constants.AddressZero &&  <a href="#" onClick={()=>clickBuyTiket(i)}> buy</a> }  </li>  )}  </ul>  </div> ) |

Ejecutamos la aplicación **React** abriendo el cmd en el directorio del proyecto y ejecutando el comando

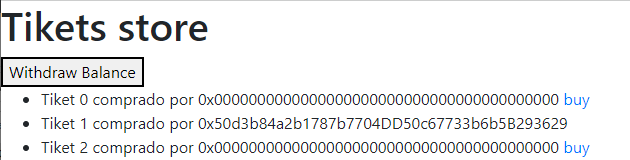
npm start



Abrimos la web en el <http://localhost:3000> en un navegador que tenga instalada la wallet de Metamask, al intentar adoptar un perro ahora nos solicitará 0.05BNB

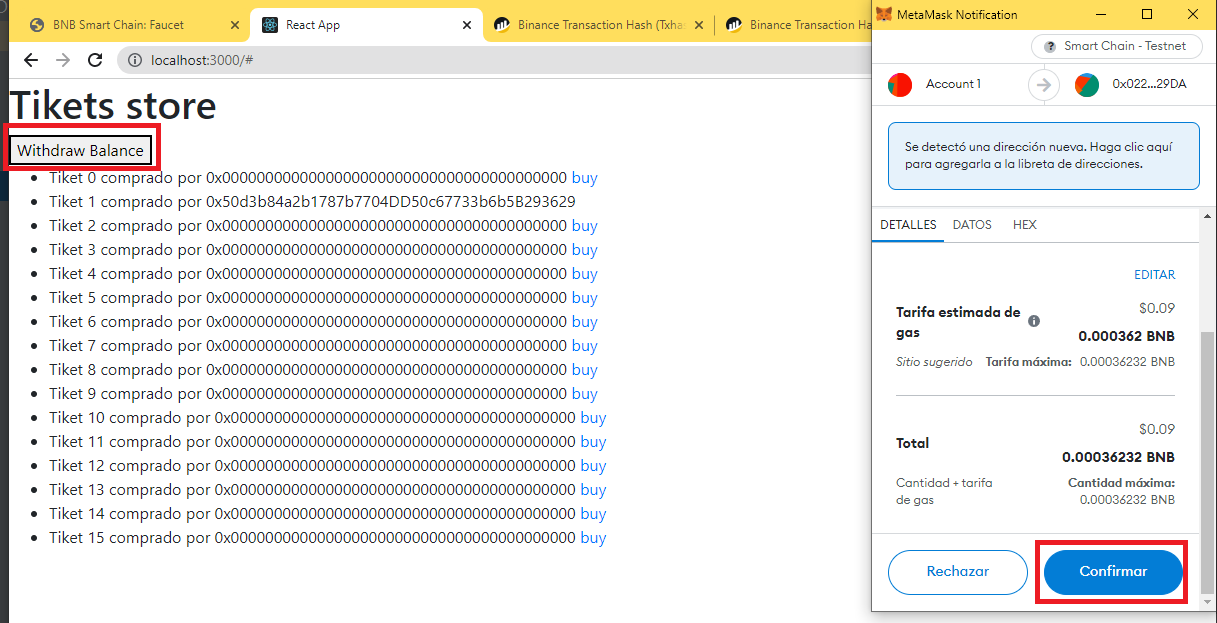


Compramos un ticket, después nos fijamos en nuestro balance BNB.





Después pulsamos en **Withdraw balance,** tras aceptar la transacción y pasar unos segundos deberíamos recibir la transferencia de todos los BNB que había en el contrato, un Ticket 0.01



Logotipo, nombre de la empresa

Descripción generada automáticamente

Tras obtener el balance hemos conseguido 16.6697 - 16.6199 = **0.0498 BNB   
¿Dónde ha ido la diferencia respecto a las 0,5 enviados?**

* Ejecución de la función **getContractBalance**
* Gas de envió desde el contrato a nuestra cartera, aunque el contrato tuviera 0,05 tiene que pagar gas para enviarlo