

Introducción a Ethereum blockchain en Español

Guia del proyecto

Sobre el curso de Ethereum

Esta práctica es para el curso de Introducción a Ethereum de Coding Bootcamps: <http://coding-bootcamps.com/es/> Para más información sobre este curso, visite el siguiente enlace: <https://learn.coding-bootcamps.com/p/introduccion-a-ethereum-blockchain-en-espanol>

Para más información sobre cursos de Blockchain en Español, visite el siguiente enlace: <https://learn.coding-bootcamps.com/p/coding-bootcamps-espanol>

CarAuction_DApp

Coding Bootcamps - Car Auction DApp

Car Auction DApp permitirá a los usuarios participar en una subasta de coches utilizando ether. Podremos administrar fondos de forma automática y segura, sin depender de un tercero de confianza.

Fases

1. Preparación del entorno de trabajo y creación del Smart Contract.
2. Desplegamos el Smart Contract en Remix y Truffle e interactuamos con él.
3. Testeando el Smart Contract.
4. Front end e interfaz.
5. Desplegando el Smart Contract en una testnet de Ethereum.

Descripción Auction DApp

El propietario de la subasta crea el Smart Contract y lo despliega en la Blockchain. Una vez acaba el tiempo definido de duración para la subasta, el mejor postor gana la subasta y los demás participantes retiran sus ofertas. El propietario de la subasta puede cancelar la subasta. Además, al final de la subasta puede retirar los fondos de la oferta ganadora.

Requisitos del entorno de trabajo

- Node.js instalado: <https://nodejs.org/es/download/>
- Ganache instalado: <https://www.trufflesuite.com/ganache>
- Truffle instalado: <https://www.trufflesuite.com/docs/truffle/getting-started/installation> `npm install -g truffle`
- Extensión de Solidity de Visual Studio
Code: <https://marketplace.visualstudio.com/items?itemName=JuanBlanco.solidity>
- Metamask instalado en navegador: <https://metamask.io/>
- Lite Server: <https://www.npmjs.com/package/lite-server>

Primeros pasos

1. Crear carpeta directorio de nuestro proyecto: CarAuction_DApp
2. Dentro de la carpeta de nuestro proyecto, ejecutamos: `truffle init` e indicamos Y (Yes). Se nos crea la estructura de carpetas: contracts, migrations, test y el archivo `truffle-config.js`
3. Ejecutamos `npm init`, se nos crea el archivo `package.json`.
4. Instalamos Lite Server: `npm install lite-server --save-dev`. En el archivo `package.json`, apartado `scripts`, añadimos: `"dev": "lite_server"`
5. En nuestra carpeta del proyecto creamos el archivo: `bs-config.json`, introducimos el siguiente código, para configurar Lite Server: `{ "server": { "baseDir": [".", "/src", "/build/contracts"] } }`
6. Instalar Truffle Contract: `npm install @truffle/contract`
7. Web3.js instalado: <https://github.com/ethereum/web3.js/> `npm install web3`
8. Instalar Truffle hdwallet provider: `npm install truffle-hdwallet-provider`

Creación del Smart Contract

1. Accedemos a la carpeta `contracts` y creamos el archivo `Auction.sol`. El código del Smart Contract se puede encontrar en: https://github.com/jordiguirao92/CarAuction_DApp/blob/master/contracts/Auction.sol
2. Compilamos el smart contract. En la carpeta de nuestro proyecto ejecutamos el comando: `truffle compile`. Se nos creará la carpeta `build`.

Desplegando e interactuando con el Smart Contract en Remix

1. Abrimos Remix y creamos un nuevo archivo de Solidity.
2. Copiamos el código de nuestro contrato.
3. Compilamos el contrato.
4. Desplegamos el contrato.
5. Interactuamos desde Remix con el contrato.

Desplegando el Smart Contract con Truffle

1. En la carpeta migrations, creamos el archivo: 2_CarAuction_migration.js. El código se encuentra en: https://github.com/jordiguirao92/CarAuction_DApp/blob/master/migrations/2_CarAuction_migration.js
2. Modificamos el archivo truffle-config.js. Donde configuraremos nuestra red local de Ethereum para el despliegue del contrato. El código está: https://github.com/jordiguirao92/CarAuction_DApp/blob/master/truffle-config.js
3. Iniciamos Ganache y comprobamos que los datos RPC SERVER sean los mismos que están en nuestro archivo truffle-config.js.
4. En la carpeta raíz de nuestro proyecto ejecutamos el comando: `truffle migrate --network development`, se iniciará el despliegue de nuestro smart contract.

Interactuamos con nuestro Smart Contract con Truffle Console

1. En la carpeta raíz de nuestro proyecto, ejecutamos `truffle console`, para acceder a la consola de truffle y poder interactuar con nuestro smart contract. Ejecutamos los siguientes comandos:

Dirección del contrato:

```
auction = await MyAuction.deployed()
auction.address
```

Consultando propietario del contrato:

```
ownerAddress = await auction.get_owner()
ownerAddress
```

Consultando cuando acaba la subasta:

```
auctionEnd = await auction.auction_end.call()  
auctionEnd  
auctionEnd.toString()
```

Realizando una apuesta:

```
auctionBid = await auction.bid({value:3000000000000000000})  
auctionBid
```

Consultando la apuesta más alta:

```
highestBid = await auction.highestBid.call()  
highetsBid  
highetsBid.toString()
```

Testing Car Auction DApp con Truffle

En la carpeta test, creamos el archivo CarAuction_test.js, donde escribiremos los diferentes test de nuestra DApp. El código de los test está

en: https://github.com/jordiguirao92/CarAuction_DApp/blob/master/test/CarAuction_test.js

Para comprobar nuestros test, ejecutamos: `truffle test` Podremos ver en la terminal el resultado de nuestros test.

Front End e Interfaz

Creamos la carpeta src dentro de la carpeta de nuestro proyecto. Dentro de la carpeta src, creamos la carpeta js y el archivo index.html.

index.html

En este archivo crearemos la interfaz gráfica y visual de nuestra DApp. El código está: https://github.com/jordiguirao92/CarAuction_DApp/blob/master/src/index.html

js

Contiene los archivos:

- app.js: En este archivo crearemos el código necesario para poder conectar con nuestro smart contract. El código está: https://github.com/jordiguirao92/CarAuction_DApp/blob/master/src/js/app.js

- truffle-contract.js: El código está: https://github.com/jordiguirao92/CarAuction_DApp/blob/master/src/js/truffle-contract.js
- web3.min.js: El código está: https://github.com/jordiguirao92/CarAuction_DApp/blob/master/src/js/web3.min.js

Desplegando Smart Contract en una testnet de Ethereum

1. Acceder a <https://infura.io/> y creamos un nuevo proyecto.
2. Actualizamos el archivo truffle-config.js. Añadimos la configuración de la red RINKEBY, indicamos las características de nuestra conexión a Infura y indicamos las palabras clave mnemonic de nuestra wallet.
3. Desplegamos el smart contract en la red de Rinkeby. `truffle migrate --network rinkeby`.

Si queremos resetear el contrato desplegado utilizar el comando: `truffle migrate --network rinkeby --reset`

Ejecutamos nuestra DApp

1. Tener nuestro Smart Contract desplegado. `truffle migrate`. Indicar que network queremos utilizar `truffle migrate --network nombredelared`
2. Inicializar nuestra DApp, ejecutando el comando: `npm run dev`.
3. Conectar Metamask a la red donde está desplegado nuestro Smart Contract.
4. Ya tenemos nuestra DApp preparada para interactuar con ella.