

INGENIERÍA MECATRÓNICA



DI_CERO

DIEGO CERVANTES RODRÍGUEZ

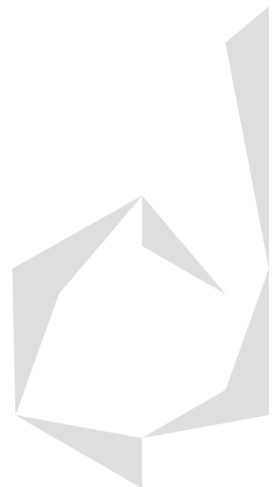
BLOCKCHAIN

ETHEREUM

Contratos Inteligentes con
Blockchain y Ethereum

Contenido

Blockchain	2
Criptografía:	2
Aplicaciones Web.....	4
Aplicaciones Web Normales	4
Aplicaciones Web con Blockchain.....	4
Contratos Inteligentes (Smart Contracts):	4
Ethereum:	4
Web 3js	12



Blockchain

Es un protocolo informático conformado por una red de computadoras para el almacenamiento de información segura, anónima, descentralizada y libre de falsificaciones.

El término significa cadena de bloques y se hace así:

1. **El primer ingrediente de la Blockchain son los bloques:** Son simples ficheros de texto que contienen la información que queremos guardar, estos forman una cadena porque cada bloque tiene información del bloque anterior, y esta a su vez del bloque anterior, así hasta el primer bloque de la cadena. A la información que cada bloque tiene sobre la anterior la llamamos hash y es una especie de número de serie que identifica a cada bloque.
2. **El segundo ingrediente de Blockchain es una red de ordenadores:** Todos los ordenadores de la red guardan una copia de la cadena de bloques, no hay una computadora central que contenga toda la información y los demás la buscan ahí, como en el caso de los servidores.

Por ejemplo, el Blockchain de bitcoin es la red de ordenadores, tablets, smartphones, etc. Que tienen el software de bitcoin instalado, esto es posible porque el software es de código abierto. ¡La cadena de ficheros (o bloques) de bitcoin contienen la lista de todas y cada una de las transacciones de bitcoin de la historia TODAS!! y por eso leyendo esa cadena de bloques sabemos cuántos bitcoins tiene cada usuario y de donde han salido.

Cada bloque de la cadena de bitcoin contiene entonces:

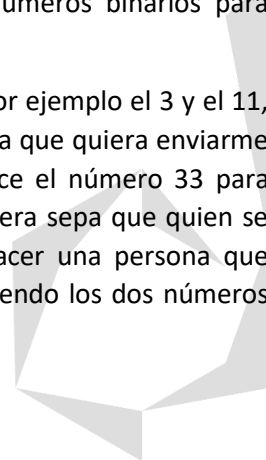
1. El hash de su bloque anterior (osea su número de serie, así están encadenados).
2. Una lista de transacciones de bitcoins.
3. Una información rara llamada prueba de trabajo.

Por lo tanto, en conclusión, Blockchain es una cadena de bloques (ficheros) con información relevante que está almacenada y replicada en una cadena de computadoras que forman una red, la información de los bloques está encriptada y sólo quien la creó la puede ver.

Criptografía:

Los números primos sirven para crear códigos y crear la criptografía de clave pública y clave privada, para ello primero tendría que convertir el mensaje que quiero encriptar a números binarios para transportarlos digitalmente.

- 1) **Public Key:** Voy a utilizar dos números primos para elaborar la clave, como por ejemplo el 3 y el 11, estos los multiplico y obtengo el número 33, ahora le digo a cualquier persona que quiera enviarme un mensaje secreto: "Quien quiera enviarme un mensaje secreto que utilice el número 33 para construir la clave, eso lo puedo decir abiertamente, me da igual que cualquiera sepa que quien se quiera comunicar conmigo usará el número 33". Ahora, lo que debería hacer una persona que quisiera descifrar el mensaje encriptado, es hacer el proceso inverso, obteniendo los dos números



que tuve que multiplicar para construir la clave. Esto cuando el número primo que conforma mi public key es muy grande se complica mucho.

- 2) **Private Key:** Esos dos números primos que multiplicados dan como resultado mi clave pública son mi clave privada, en este caso los números serían el 3 y el 11.

Para el 33 está fácil factorizar el número 33 y obtener los números primos que lo conforman, pero con **claves públicas** enormes es cuando se ve la fuerza de este método, ya que para obtener esa clave se tuvieron que haber multiplicado dos números de **clave privada** muy grandes y obtener esos dos números primos que al multiplicarlos nos proporciona la **clave pública** es extremadamente difícil. **Este método se llama RSA.**

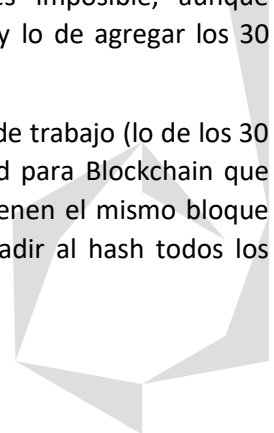
La criptografía es relevante en Blockchain porque eso hace que la información sea segura y anónima, pero ahora, ¿cómo se añade un bloque y cómo le hacemos para que todos los nodos de la red tengan una copia exacta de la cadena de bloque?

Esto se hace agregando un número (que siempre debe empezar con 30 ceros) llamado “la prueba de trabajo” al hash de los bloques, este proceso se hace en unos nodos (osea bloques) especiales de la red llamados mineros.

Cuando un nuevo bloque está listo para añadirse a la cadena, los mineros reciben un aviso, para que puedan añadir el bloque a la red, solo hace falta que alguien calcule la “prueba de trabajo” y la añada al bloque, el primero que lo consiga avisa que ya la consiguió, se comprueba por todos los mineros de la red y cuando la mayoría determina que todo está bien, el bloque se agrega a la red y el minero que lo logró se lleva un premio. Pero calcular la “prueba de trabajo” del bloque no es fácil, así que en cuanto salta el aviso de nuevo bloque, los mineros se ponen a trabajar. La “prueba de trabajo” tiene que ver con el hash de cada bloque (ese número de serie del que hablábamos); los hashes se hacen con una función matemática que recibe un fichero de texto y le asigna un número, de tal forma que el número creado depende totalmente del texto, si se le cambia un espacio, una letra, una coma o lo que sea al texto, el resultado ya no es el mismo. Y la forma de asignarlo es extremadamente complicada, de tal forma que, si nos dan el número final, sea imposible saber el texto del que viene. Eso es lo que hacen los mineros, generar estos números irrastreables.

Por lo tanto, en bitcoin funciona así:

1. Surge un bloque nuevo para meter a la cadena.
2. Los mineros buscan un número para añadirlo a la información del bloque y que aparezca en su hash, este número siempre deberá empezar con 30 ceros, esto se hace usando un algoritmo de generación de hash llamado SHA256, deshacer esta codificación es imposible, aunque conozcamos el funcionamiento del algoritmo, a esto se le llama minar y lo de agregar los 30 ceros se llama “dificultad de la prueba de trabajo”.
3. La existencia de algoritmos de hash, el añadido la dificultad de la prueba de trabajo (lo de los 30 ceros) y el hecho de tener un premio por minar son capas de seguridad para Blockchain que hacen que nadie pueda falsificar un bloque porque todos los mineros tienen el mismo bloque para añadir a la cadena y cuando alguien consigue el número para añadir al hash todos los



mineros la comprueban y la mayoría tiene que estar de acuerdo de que es bueno, sino no se añade.

Ethereum con sus contratos inteligentes (smartcontracts), validados por una blockchain lo ha aplicado también.

Aplicaciones Web

Aplicaciones Web Normales

El proceso de funcionamiento para las aplicaciones normales lo que se hace es que el navegador se conecta a un servidor, de esta extrae el documento HTML (que contiene archivos CSS para diseño estático y JavaScript para diseño dinámico) que es código de parte del cliente y posteriormente puede conectar con documentos PHP, Node.js, etc. que manejan el código Backend de la página (que crea, lee y/o actualiza los posibles datos que ingrese el usuario en el código cliente o que haya en la base de datos) para que luego los pueda meter a una base de datos, todo dentro de una misma computadora, que es el servidor.

Aplicaciones Web con Blockchain

Pero con un sitio Blockchain el proceso es muy diferente, ya que lo que pasa es que se corre y compila el código cliente en el navegador, luego en vez de conectarse directo a un servidor y comunicarse con un Backend o con una base de datos, lo que hará es comunicarse directamente con un bloque o nodo individual de la Blockchain para poder usar la cadena, esto porque recuerdo que cada bloque de la Blockchain tiene una copia de todo el código de la Blockchain. Todos los datos de nuestra aplicación web no estarán en una base de datos sino en la Blockchain misma.

Contratos Inteligentes (Smart Contracts):

En la Blockchain va a haber código, este va a estar contenido dentro de “contratos inteligentes” que simplemente son programas que corren en la Blockchain escritos en un lenguaje llamado **Solidity**, este código nos va a servir para cumplir las funciones que hacía la parte del Backend en el modelo normal de los sitios web. Cuando suba el código del Smart Contract a la Blockchain ya no lo podré cambiar, esto lo hace muy seguro, pero igual hace que no pueda actualizarlo.

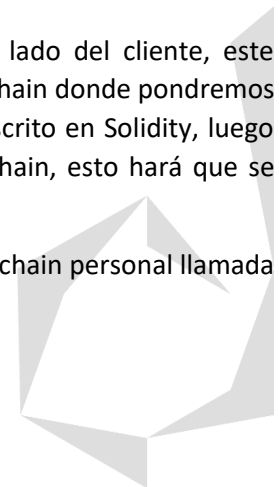
Ethereum:

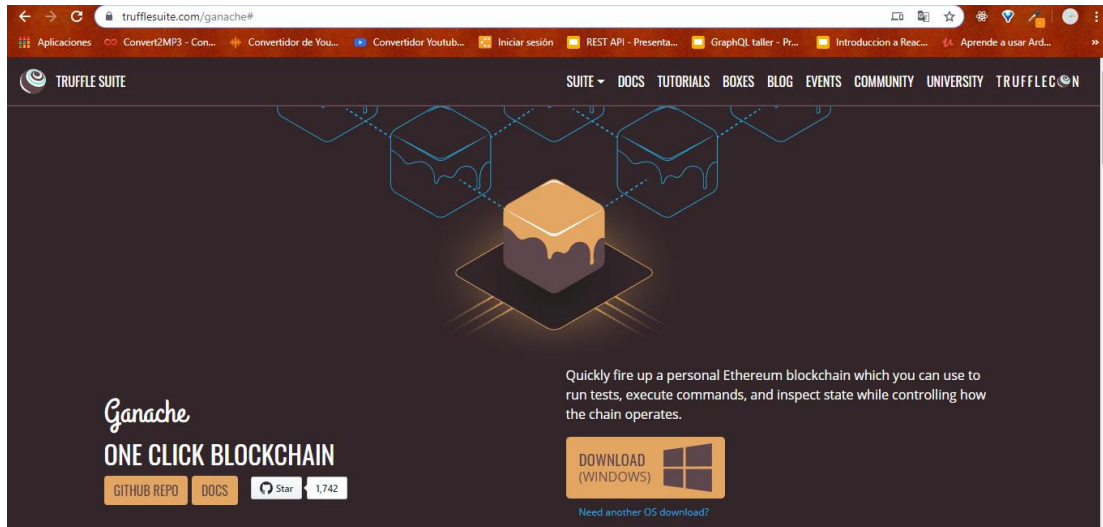
Los contratos inteligentes de Ethereum se realizan con su propio lenguaje.

Entonces lo que vamos a hacer es crear un archivo HTML/CSS/JavaScript del lado del cliente, este archivo se va a correr en el navegador para comunicarse directamente a la Blockchain donde pondremos nuestro Smart Contract, que servirá como el “Backend” de nuestra aplicación escrito en Solidity, luego este lo compilaremos y le daremos deploy (osea que lo subiremos) a la Blockchain, esto hará que se conecte a la red de Blockchain con nuestra cuenta personal de Ethereum.

Para ello necesitaremos instalar y usar Node.js También debemos bajar una Blockchain personal llamada Ganache, que podemos encontrar en el siguiente link:

<https://www.trufflesuite.com/ganache#>





Cuando la instalemos tendremos un Blockchain local corriendo (que es como un servidor local) para poder hacer pruebas. Esta funciona igual que una Blockchain real, pero corre en una sola computadora en vez de correr en una red de computadoras como las Blockchain reales, por eso solo es para hacer simulaciones y no hace las funciones de una Blockchain real.

Ya que la abramos podremos ver una interfaz de visualización de la Blockchain que sirve para ver los bloques de la Blockchain junto con sus respectivos Hash, los Ether (osea la moneda de Ethereum) que cada cuenta tiene, etc.

Luego deberemos instalar el framework Truffle para poder desarrollar los Smart Contracts de Ethereum con el lenguaje Solidity para posteriormente poder subirlo a la Blockchain (darle deploy).

Lo descargamos metiendo el siguiente comando dentro de la consola de GitBash:

`npm install -g truffle@5.0.2`

```
MINGW64:/c:/Users/diego/OneDrive/Documents/Devf/Cinta Roja/6.-Peticio...
identity: [Object],
domainName: 'sinwh92a0h.execute-api.us-east-1.amazonaws.com',
apiId: 'sinwh92a0h' },
body: '{"edad":18,"sexo":"M","calificacion_medica":"Jodido"}',
isBase64Encoded: false } }

diego@DIEGOBITCHES MINGW64 ~/OneDrive/Documents/Devf/Cinta Roja/6.-Peticiones
$ node hacerPost.js
statusCode: 502
{ message: 'Internal server error' }

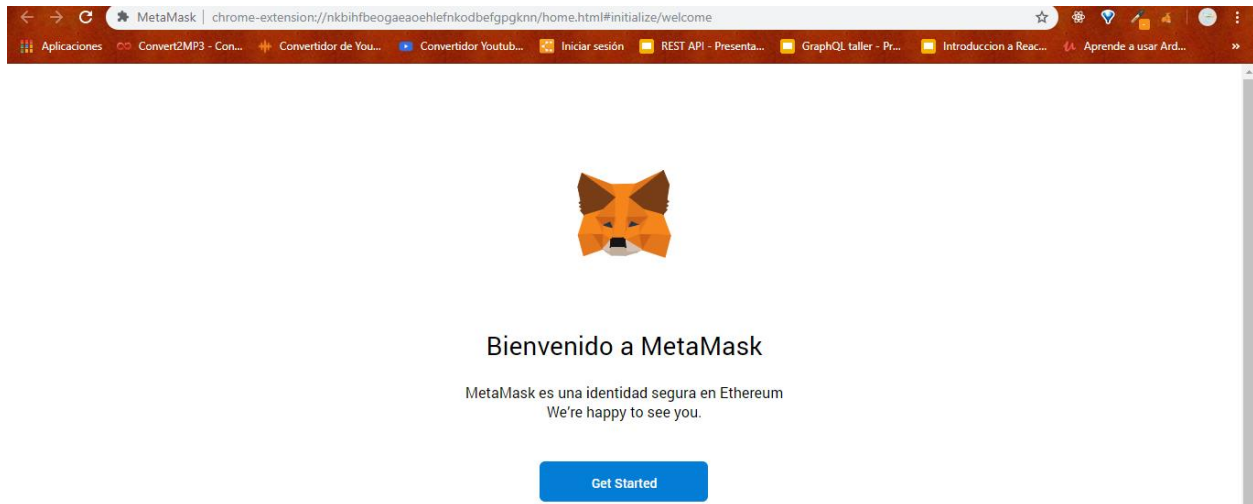
diego@DIEGOBITCHES MINGW64 ~/OneDrive/Documents/Devf/Cinta Roja/6.-Peticiones
$ node hacerPost.js
statusCode: 502
{ message: 'Internal server error' }

diego@DIEGOBITCHES MINGW64 ~/OneDrive/Documents/Devf/Cinta Roja/6.-Peticiones
$ node hacerPost.js
statusCode: 502
{ message: 'Internal server error' }

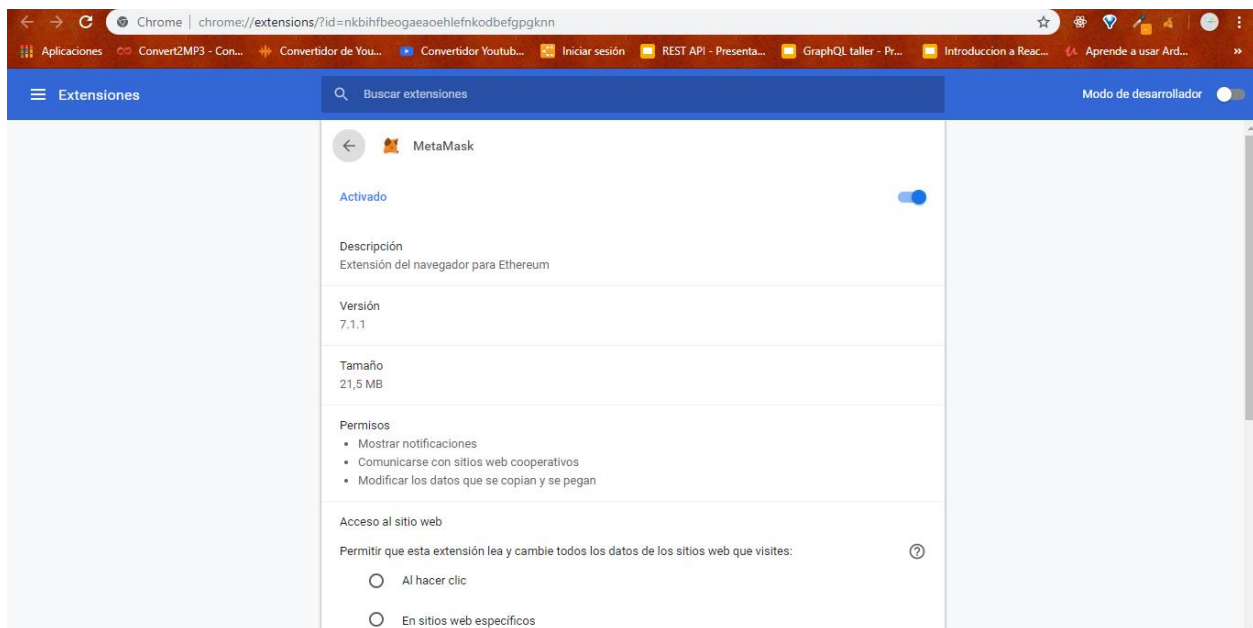
diego@DIEGOBITCHES MINGW64 ~/OneDrive/Documents/Devf/Cinta Roja/6.-Peticiones
$ npm install -g truffle@5.0.2
```

Finalmente tenemos que agregar una extensión llamada METAMASK a Google Chrome, esto porque como la Blockchain de Ethereum es una red de ordenadores, necesitamos una extensión especial para poder conectarnos a dicha red con nuestra cuenta personal e interactuar con el Smart Contract que vamos a crear, podemos descargar y activar la extensión dentro de este enlace:

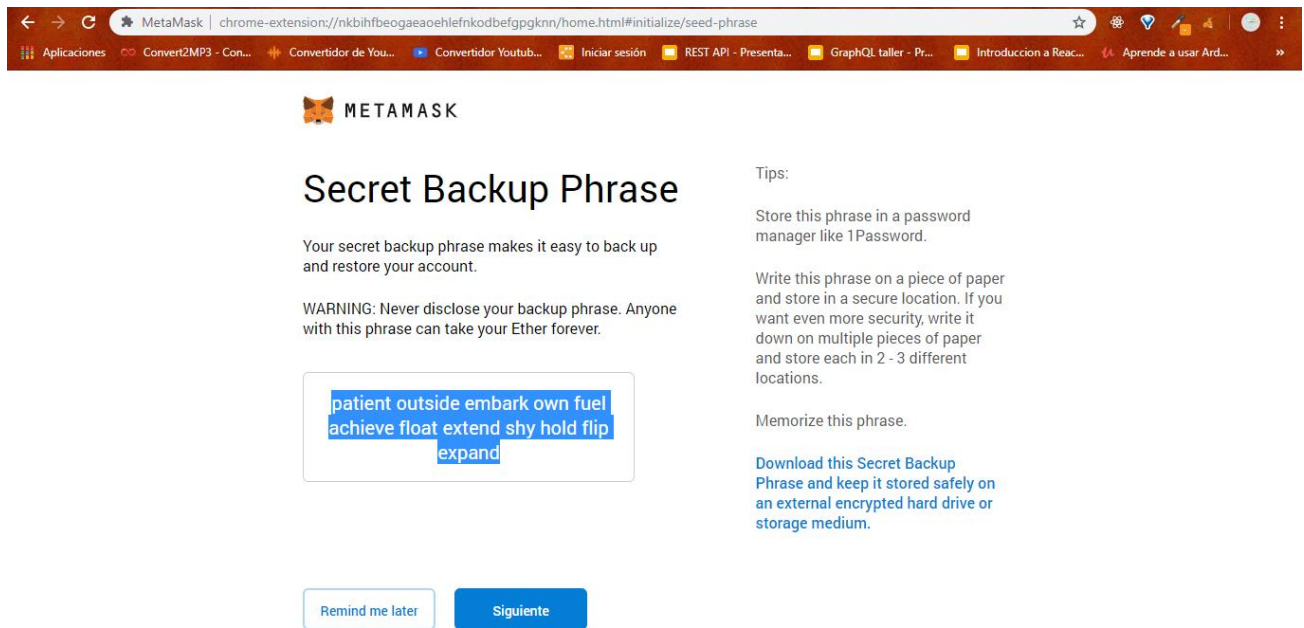
<https://metamask.io>



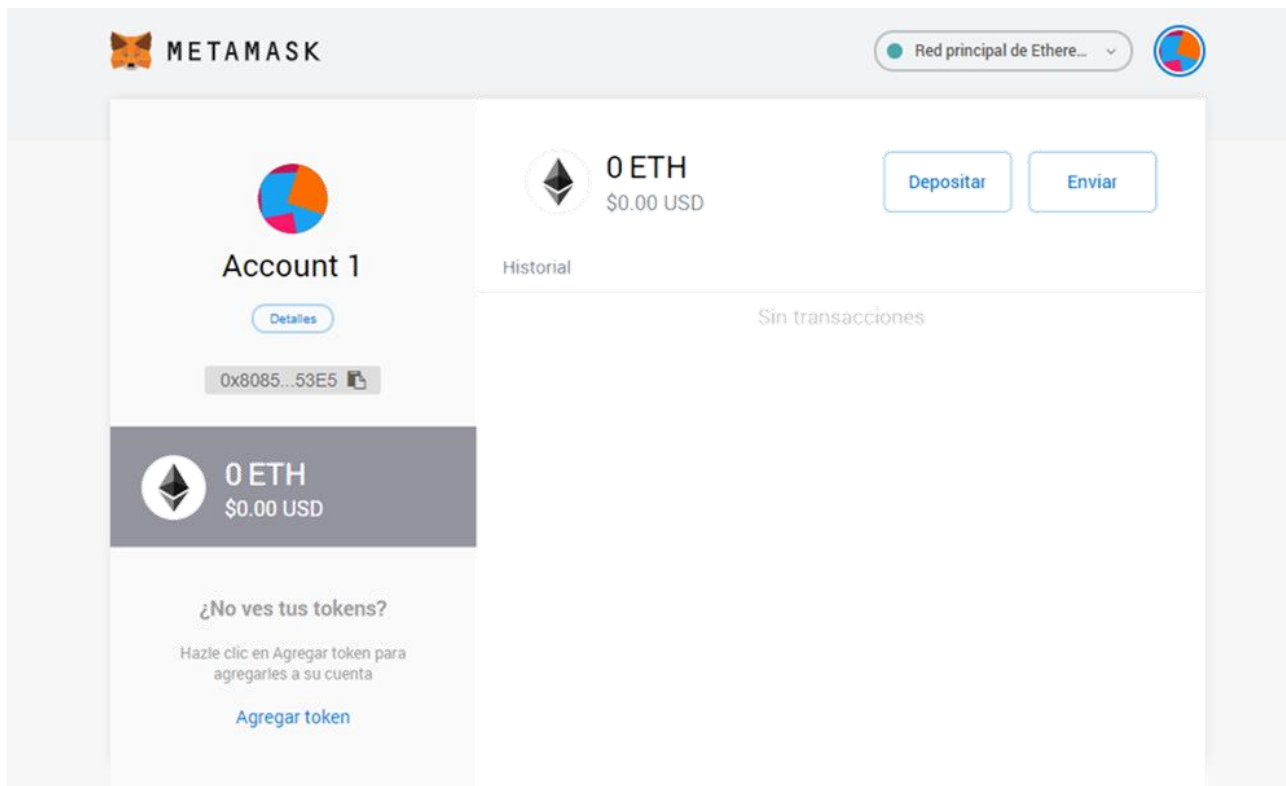
Debemos asegurarnos de activar la extensión de MetaMask para que pueda correr nuestro proyecto.



Cuando abro el ícono me dice que me registre, dentro de esta me dice que puedo tener una frase de respaldo por si se me olvida mi contraseña, si esta se me olvida o me la roban alguien más se podría quedar con mi dinero Ether. La contraseña es esta:

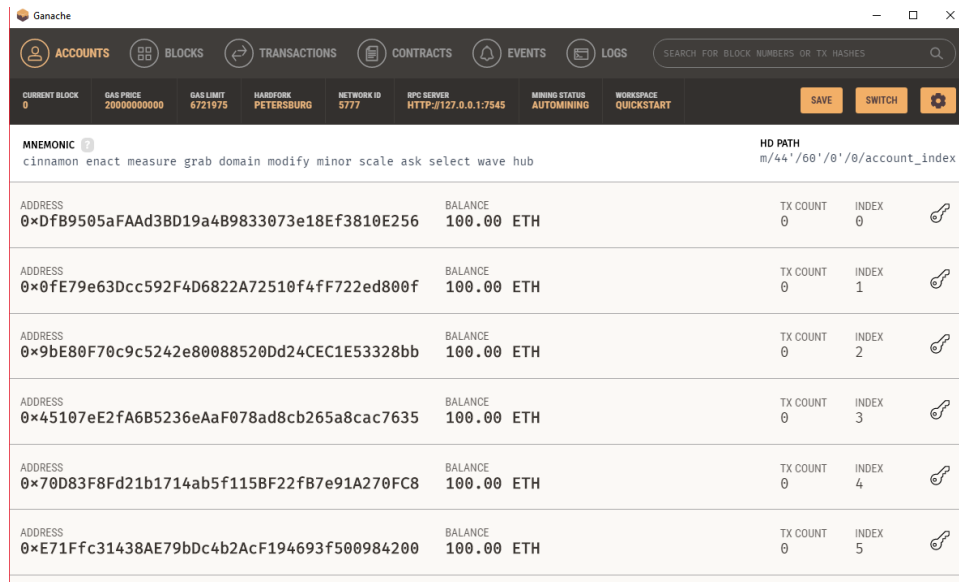


Con esto ya habré creado mi cartera en MetaMask.

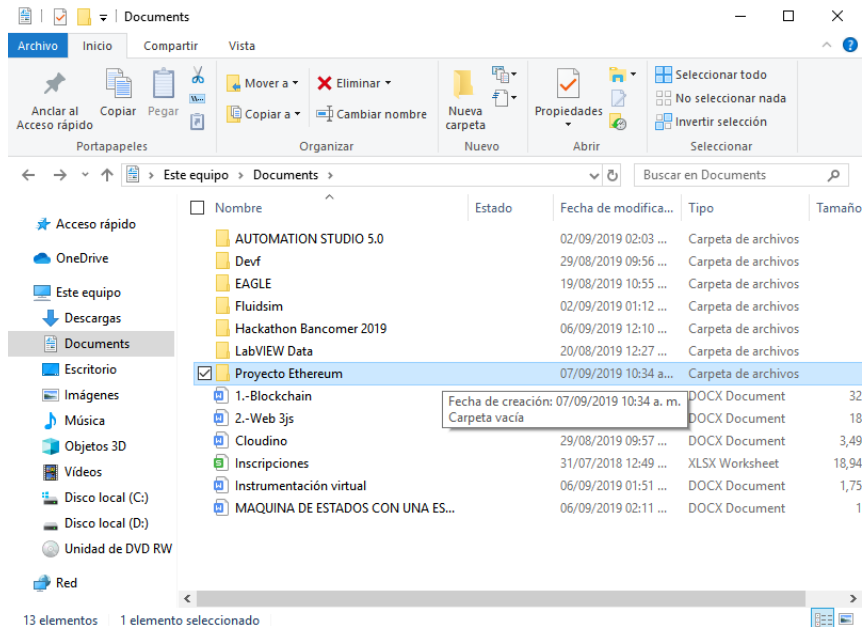


Ahora, cuando abramos la aplicación de Ganache veremos la interfaz donde se nos muestra las cuentas que existen dentro de la Blockchain y el dinero en Ether que cada cuenta tiene.

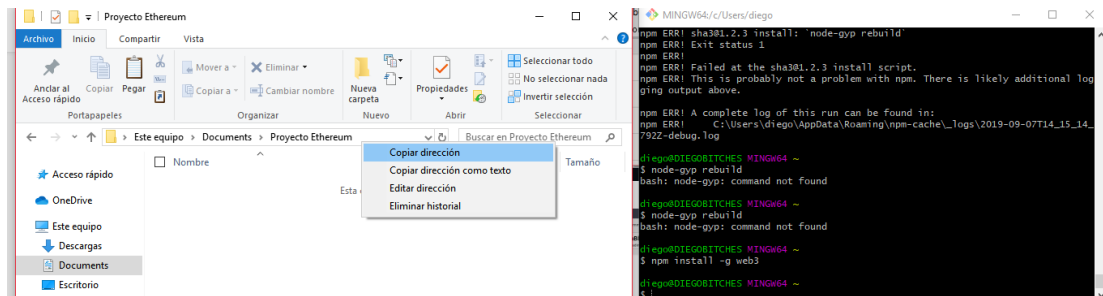
Para que podamos desarrollar nuestro proyecto necesitamos que Ganache esté corriendo.

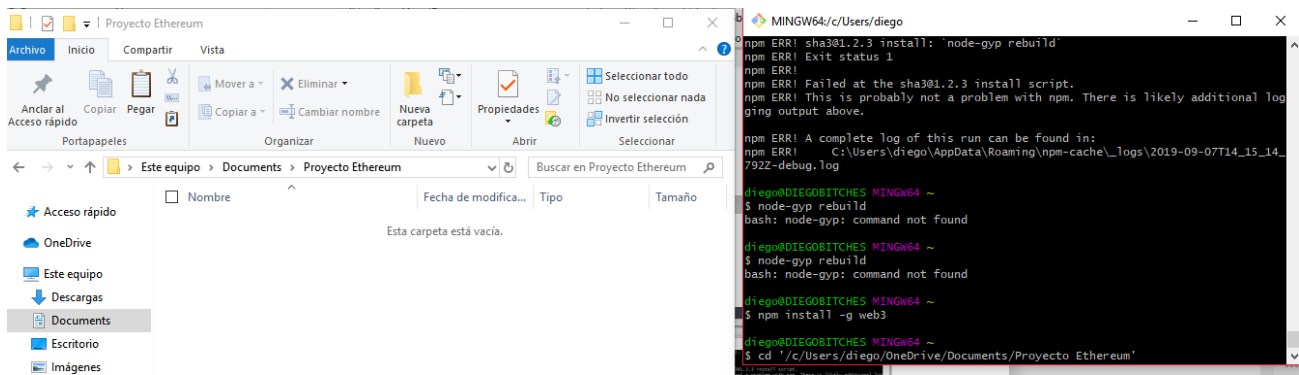


Ahora para crear un proyecto debo crear una nueva carpeta:



Ahora debo meterme a esa carpeta dentro de Git Bash.





Dentro de esta carpeta vamos a inicializar nuestro proyecto con Truffle, pero para ello debemos checar que tengamos la versión 5.0.2 de Truffle con el comando: `truffle version`

Ya que lo hagamos checado tenemos que iniciar nuestro proyecto truffle con el comando: `truffle init`

¡¡Todo esto debe ser en la carpeta de nuestro proyecto!!

```

MINGW64/c/Users/diego/OneDrive/Documents/Proyecto Ethereum
diego@DIEGOBITCHES MINGW64 ~/OneDrive/Documents/Proyecto Ethereum
$ truffle init

- Preparing to download
- Preparing to download
- Downloading
- Downloading
- Cleaning up temporary files
- Cleaning up temporary files
- Setting up box
- Setting up box

Unbox successful. Sweet!

Commands:
  Compile:      truffle compile
  Migrate:      truffle migrate
  Test contracts: truffle test

diego@DIEGOBITCHES MINGW64 ~/OneDrive/Documents/Proyecto Ethereum
$

```

Ahora vamos a crear un archivo package.json (aunque este se creará vacío) para ahí pueda incluir las dependencias de mi proyecto (donde dependencias se refiere a las librerías o frameworks que usaré para mi proyecto como por ejemplo Bootstrap para usar sus librerías de diseño si lo vamos a usar o también nodemon para que la consola se mantenga en ejecución o igual es necesario implementar las dependencias del framework de Truffle que es el que estamos usando y así demás cositas), esto lo haré con el comando: `touch package.json`

```

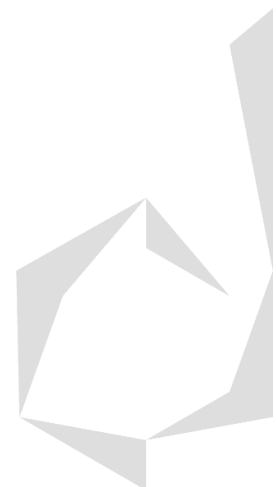
MINGW64/c/Users/diego/OneDrive/Documents/Proyecto Ethereum
- Preparing to download
- Preparing to download
- Downloading
- Downloading
- Cleaning up temporary files
- Cleaning up temporary files
- Setting up box
- Setting up box

Unbox successful. Sweet!

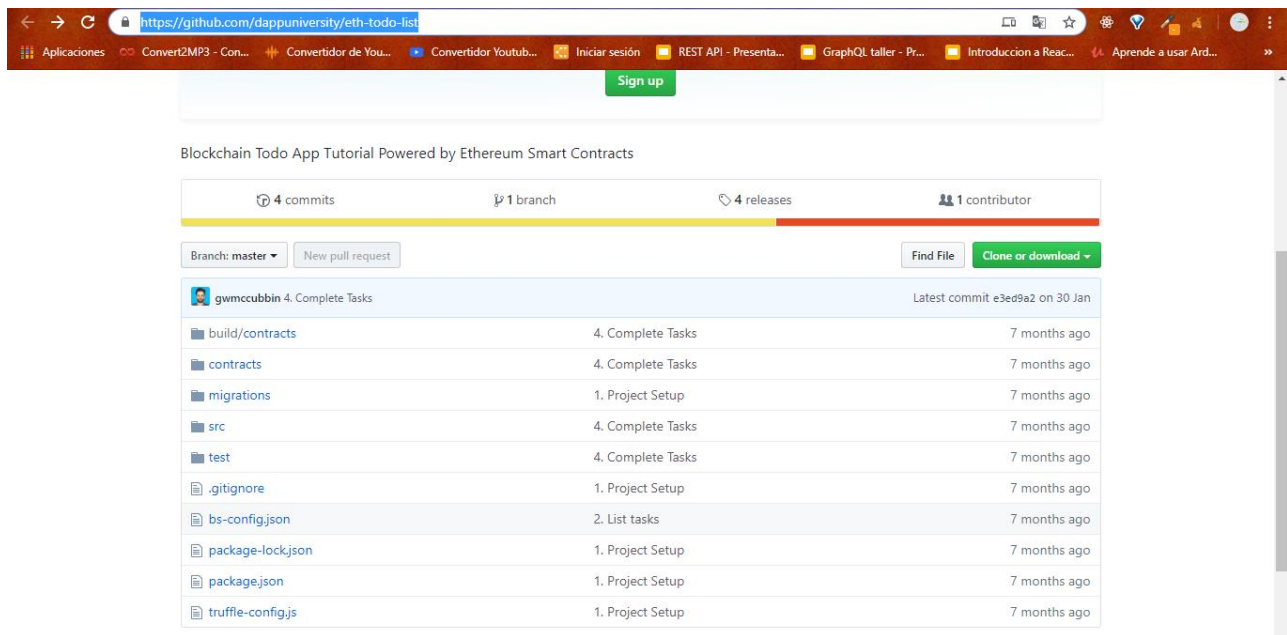
Commands:
  Compile:      truffle compile
  Migrate:      truffle migrate
  Test contracts: truffle test

diego@DIEGOBITCHES MINGW64 ~/OneDrive/Documents/Proyecto Ethereum
$ touch package.json
diego@DIEGOBITCHES MINGW64 ~/OneDrive/Documents/Proyecto Ethereum
$

```



Ahora voy a abrir mi editor de texto en la carpeta de mi proyecto, este ya tendrá mi archivo package.json, este va a estar vacío, el contenido que debe tener lo voy a copiar del siguiente link de github donde se divide por partes el código que haremos, ya hecho: <https://github.com/dappuniversity/eth-todo-list>



En este me meteré al archivo de package.json y usaré las siguientes dependencias:

A screenshot of a Visual Studio Code editor window. The title bar says 'package.json - Proyecto Ethereum - Visual Studio Code'. The left sidebar shows the 'EXPLORER' view with a tree structure of files: 'package.json' and 'truffle-config.js'. The main editor area shows the content of 'package.json'. The code is as follows:

```
1 {  
2   "name": "eth-todo-list",  
3   "version": "1.0.0",  
4   "description": "Blockchain Todo List Powered By Ethereum",  
5   "main": "truffle-config.js",  
6   "directories": {  
7     "test": "test"  
8   },  
9   "scripts": {  
10    "dev": "lite-server",  
11    "test": "echo \"Error: no test specified\" && exit 1"  
12  },  
13  "author": "gregory@dappuniversity.com",  
14  "license": "ISC",  
15  "devDependencies": {  
16    "bootstrap": "4.1.3",  
17    "chai": "^4.1.2",  
18    "chai-as-promised": "^7.1.1",  
19    "chai-bignumber": "^2.0.2",  
20    "lite-server": "^2.3.0",  
21    "nodemon": "^1.17.3",  
22    "truffle": "5.0.2",  
23    "truffle-contract": "3.0.6"  
24  }  
25 }
```

Ya que haya pegado esto en el archivo package.json de mi proyecto debo meter en mi consola Git Bash el comando: `npm install`

Y así se instalarán mis dependencias, osea los frameworks y librerías que haya metido en mi package.json

```
MINGW64:/c:/Users/diego/OneDrive/Documents/Proyecto Ethereum
- Preparing to download
✓ Preparing to download
- Downloading
✓ Downloading
- Cleaning up temporary files
✓ Cleaning up temporary files
- Setting up box
✓ Setting up box

Unbox successful. Sweet!

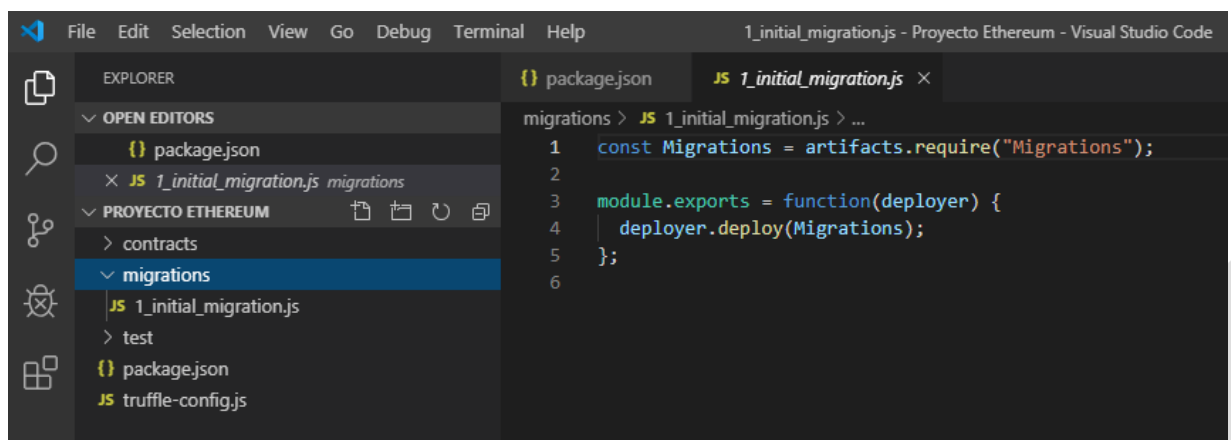
Commands:

  Compile:      truffle compile
  Migrate:      truffle migrate
  Test contracts: truffle test

diego@DIEGOBITCHES MINGW64 ~/OneDrive/Documents/Proyecto Ethereum
$ touch package.json

diego@DIEGOBITCHES MINGW64 ~/OneDrive/Documents/Proyecto Ethereum
$ npm install
```

Ahora debemos meternos a la carpeta llamada Migrations dentro de nuestro proyecto, en esta carpeta es donde se encontrarán todos los Smart Contracts de nuestro programa.



```
File Edit Selection View Go Debug Terminal Help 1_initial_migration.js - Proyecto Ethereum - Visual Studio Code

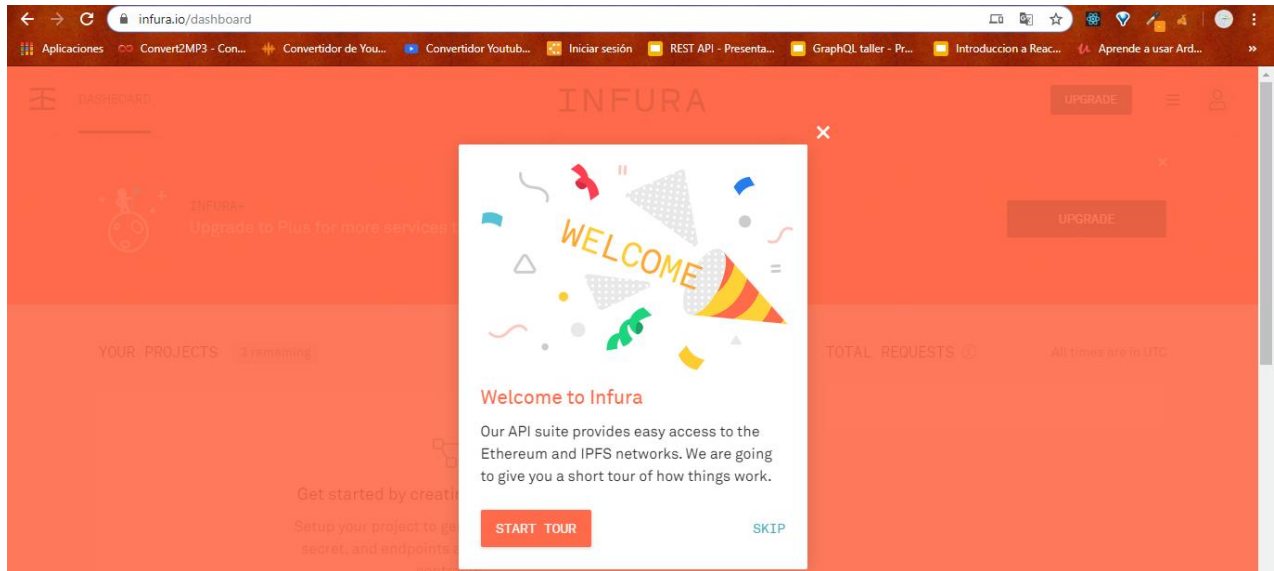
EXPLORER
├── OPEN EDITORS
│   ├── {} package.json
│   └── JS 1_initial_migration.js migrations
├── PROYECTO ETHEREUM
│   ├── > contracts
│   └── > migrations
│       ├── JS 1_initial_migration.js
│       ├── > test
│       ├── {} package.json
│       └── JS truffle-config.js
└── {} package.json

migrations > JS 1_initial_migration.js > ...
1  const Migrations = artifacts.require("Migrations");
2
3  module.exports = function(deployer) {
4    deployer.deploy(Migrations);
5  };
6
```

Web 3js

Me debo registrar en infura, de esta forma podré tener acceso a un bloque de Ethereum sobre JSON RPC de forma muy rápida, esto lo haré en el siguiente link donde simplemente me debo registrar.

<https://infura.io>



Además para ello me debo registrar en infura:

<https://infura.io/dashboard>

