



Iniciación a la Blockchain

Welcome to bitcoin

¿Qué es
blockchain?

La parábola del Suricato Volador

Para que comprendas qué es blockchain de forma rápida imagina que vas caminando por la calle y, de repente, un suricato volador de proporciones pantagruélicas aterriza en una plaza llena de gente, se come los helados de todos los niños que hay en ella, suelta dos chillidos enormes y se va igual que ha venido.

Sin un segundo que perder, se coloca un detector de mentiras a las 1.000 personas que han sido testigos y se registra exactamente qué es lo que han visto.

Todos cuentan la misma historia con idénticos detalles.

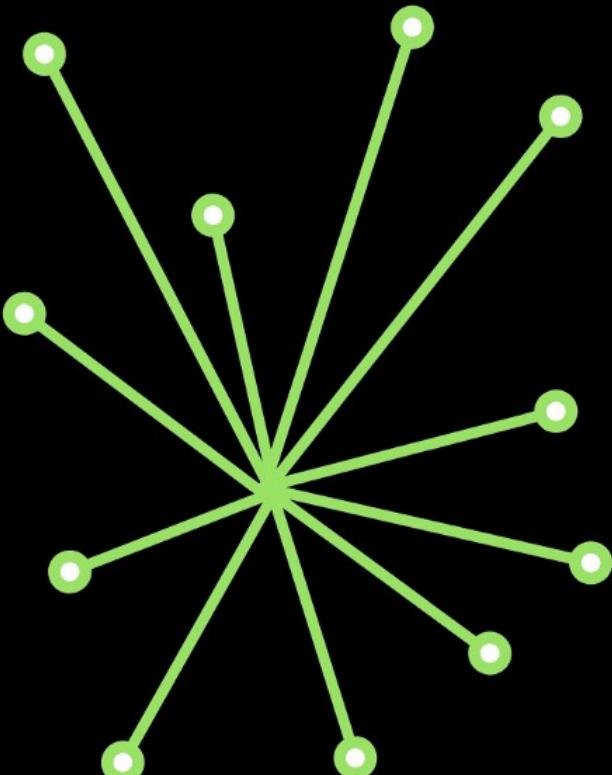
Explicando la parábola del Suricato Volador

En la analogía anterior, los incrédulos individuos que se ponen de acuerdo al explicar lo que han visto, vienen a ser nodos (más adelante hablaremos de ello) geográfica y computacionalmente aislados los unos de los otros.

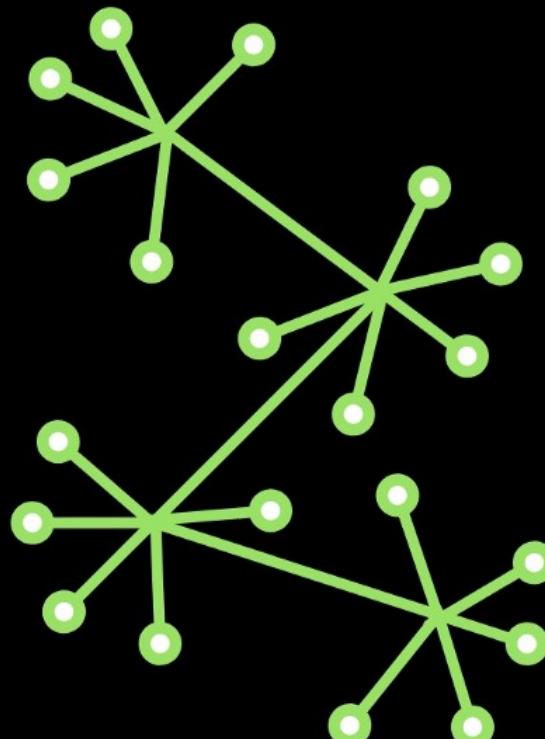
Al “detector de mentiras”, le enseñan una “prueba de trabajo” (proof of work en inglés), es decir, un proceso criptográfico que prueba que un ordenador/chip y no otro ha resuelto un problema de forma correcta.

Falsificar una entrada en la cadena de bloques equivaldría a conseguir que más de la mitad de la gente se pusiese de acuerdo en mentir acerca de los detalles del aterrizaje del suricato de la misma manera, todos al mismo tiempo y sin tener la posibilidad de coordinar esa mentira previamente.

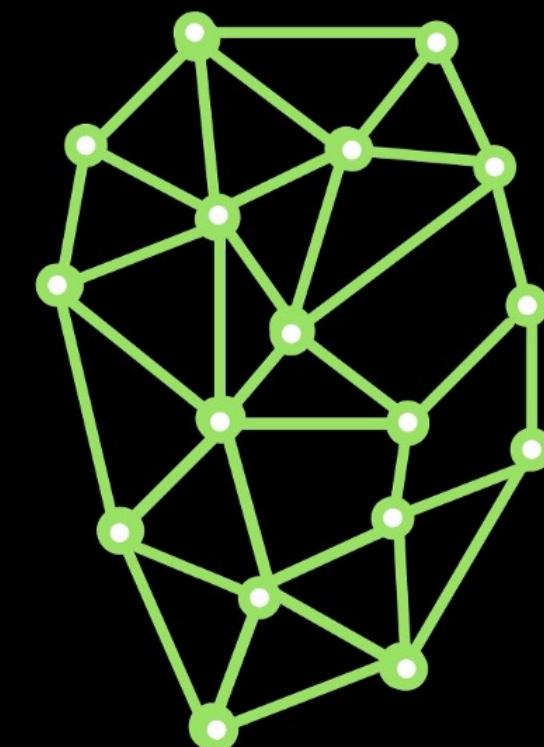
Lo que se plasma en el blockchain no puede desaparecer jamás. Blockchain es un registro inmutable y permanente. Se trata de una base de datos que solo permite escritura. No se puede modificar ni borrar nada de ello y siempre, bajo consenso.



Centralized



Decentralized



Distributed

Casos de uso de Blockchain

No solo de las criptomonedas vive blockchain

1. Almacenamiento en la nube distribuido
2. Patentes/Registro de Propiedad
3. Voto electrónico
4. Gobierno transparente

BASIC

¿Qué es Bitcoin **BITCOIN?**

El Bitcoin (BTC) es la primera criptomonedas del mundo, el génesis de esta tecnología y la criptomonedas más usada y fuerte en la actualidad.



Whitepaper Bitcoin

Un paper se entiende como un escrito científico, normalmente la exposición de una investigación o de un tema muy concreto.

Un whitepaper es un documento que sirve de guía para explicar un concepto determinado o la solución a un problema específico.

Satoshi Nakamoto publicó el whitepaper de bitcoin en un foro en el año 2008

Nacimiento Bitcoin

Pero no fue sino hasta el 9 de enero de 2009, cuando aparece la primera versión del software de Bitcoin:
la versión 0.1.0

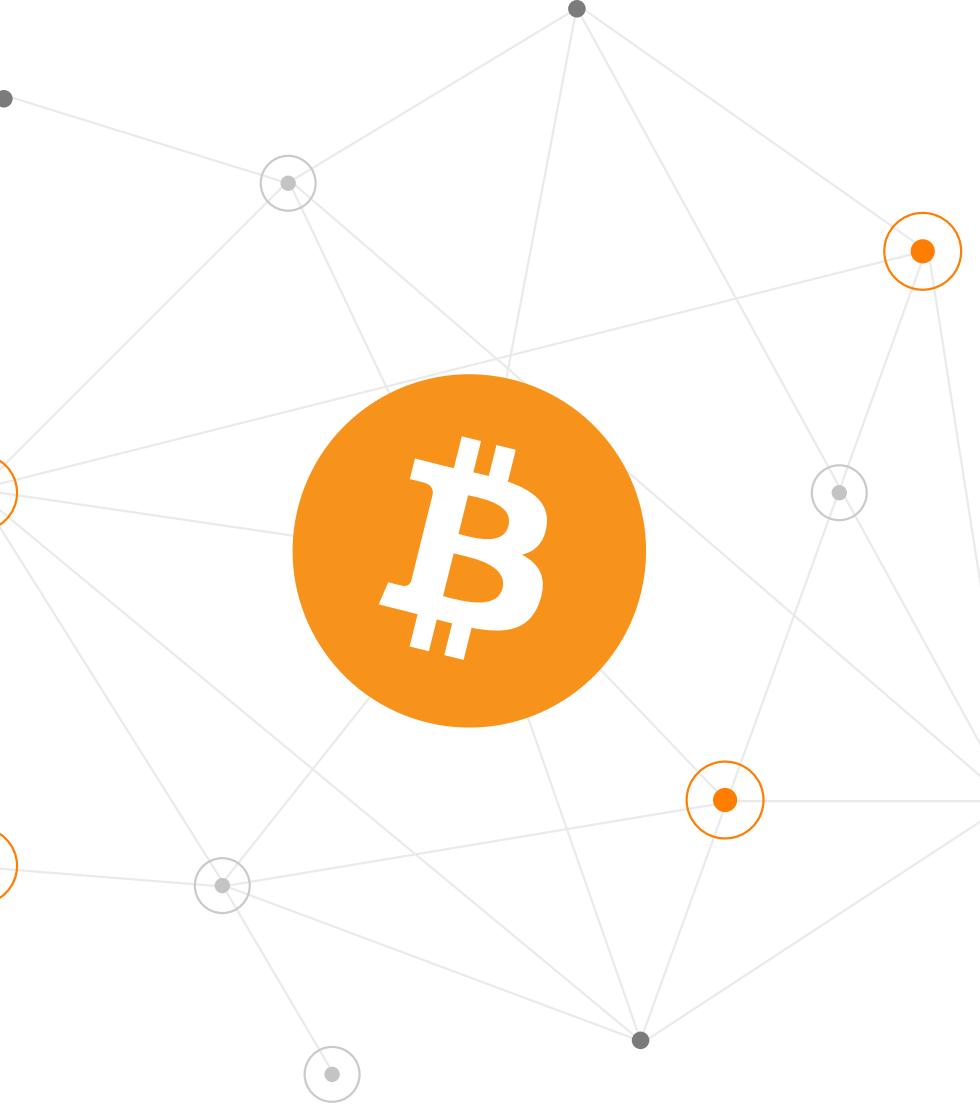
Actualmente está en la versión 0.23.x

Puedes visitar su repositorio en <https://github.com/bitcoin/bitcoin>

La propuesta inicial de Bitcoin

La creación de Bitcoin comienza con la idea plasmada en el proyecto DigiCash, un proyecto de dinero digital creado por David Chaum.

Satoshi Nakamoto uso este proyecto como la base para comenzar a crear Bitcoin, pero para ello debían corregirse algunos problemas dentro de DigiCash.



¿Por qué es tan importante Bitcoin para entender la tecnología blockchain?



La solución al problema del doble gasto.



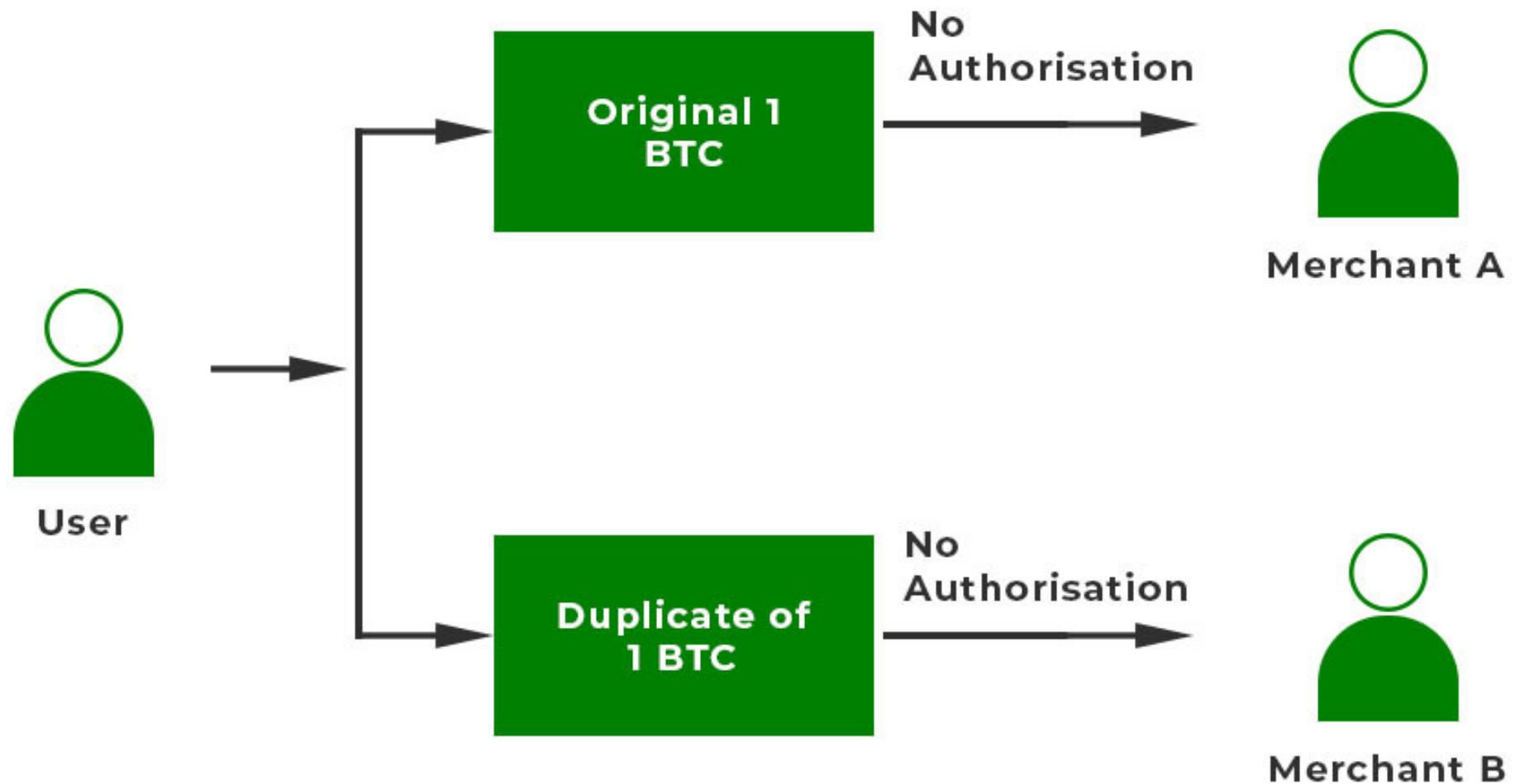
¿Qué es el doble gasto?

El doble gasto es un tipo de ataque que permitiría a un cibercriminal utilizar unas mismas monedas múltiples veces, un hecho que afecta a la tecnología de las criptomonedas debido a su descentralización y para el cual los desarrolladores han creado diversos mecanismos de protección.

¿Cómo funciona un ataque de doble gasto?

El doble gasto tiene lugar cuando un usuario desea utilizar unas mismas monedas múltiples veces.





Los ataques de doble gasto mas famosos

- Ataque 51
- Ataque de carrera (Race Attack)
- Ataque por fuerza bruta
 - Ataque Finney
 - Vector 76

Ataque 51

El famoso ataque del 51%, consiste en que una persona o una coalición logre hacerse con el 51% (más del 50%) del poder de hash de la red. Con lo que lograría controlarla. Así, por ejemplo, llevar a cabo un ataque de fuerza bruta sería 100% factible.

Race Attack

Cuando realizamos o recibimos una transacción en Bitcoin con 0 número de confirmaciones, la operación aún puede ser revertida. Este ataque ocurre cuando se realizan dos transacciones partiendo de los mismos fondos, es decir, se envía la misma cantidad de dinero a dos personas distintas. Pero sólo una de ellas recibirá los fondos, mientras que la otra no.

Por ejemplo el atacante puede enviar 2 transacciones, una para ti y otra para él, y en la que va para el poner mayor comisión para los mineros. Normalmente los mineros priorizarán la transacción con mayor comisión, invalidando la otra.

Ataque por fuerza bruta

Este ataque puede ser ejecutado aún cuando el receptor ha decidido esperar una cierta cantidad de confirmaciones. Consiste en que un atacante realiza una transacción a un usuario como pago de un producto. Y simultáneamente busca una variación de la blockchain donde incluir la transacción fraudulenta

Es importante señalar que la ejecución de este ataque es sumamente costosa, y su posibilidad de éxito es bastante baja si no se cuenta con suficiente velocidad y potencia de hash. Por lo que sólo es posible llevar a cabo el ataque en un plano teórico.



La minería al rescate



Minería

La minería es el método en el cual se resuelven problemas matemáticos de forma que la red premia con un incentivo a todos los que ofrecen sus pools para resolver estos problemas.

¿Qué es una **POOL DE MINERÍA** de criptomonedas?



Proof of Work

El protocolo de Prueba de Trabajo o Proof of Work, es el más conocido y antiguo protocolo de consenso que consiste en que las partes de una red realicen con éxito un trabajo computacionalmente costoso para acceder a los recursos de dicha red.



MINAR BITCOINS

En qué consiste
y cómo funciona



bit2me
ACADEMY

Bekijken op



Malentendidos típicos sobre Bitcoin

Malentendidos típicos sobre Bitcoin

Bitcoin es una red descentralizada

La cadena de bloques es una red P2P en la que todos los nodos son iguales entre sí dando como resultado un sistema distribuido resistente a ataques informáticos, fallos o falsificaciones.

De esta manera, aunque un nodo fallase se podría llegar a aquellos a los que estaba conectado por vías alternativas.

Esto no sería posible en un sistema descentralizado.

Malentendidos típicos sobre Bitcoin

Bitcoin es anonimo

El seudonimato y el anonimato son dos maneras de referirse a una persona. Es el secreto sobre el autor de un acto, dicho u obra. Se utilizan en las publicaciones de libros, artículos periodísticos, referencias policiales y otros medios de comunicación masivos.

Bitcoin se basa en los seudonimos

Malentendidos típicos sobre Bitcoin

Bitcoin sirve como moneda de prácticas ilícitas

Esta creencia proviene del hecho de que, en sus inicios, gracias a sus características seudónimas Bitcoin fue muy utilizado en los mercados de la Darknet

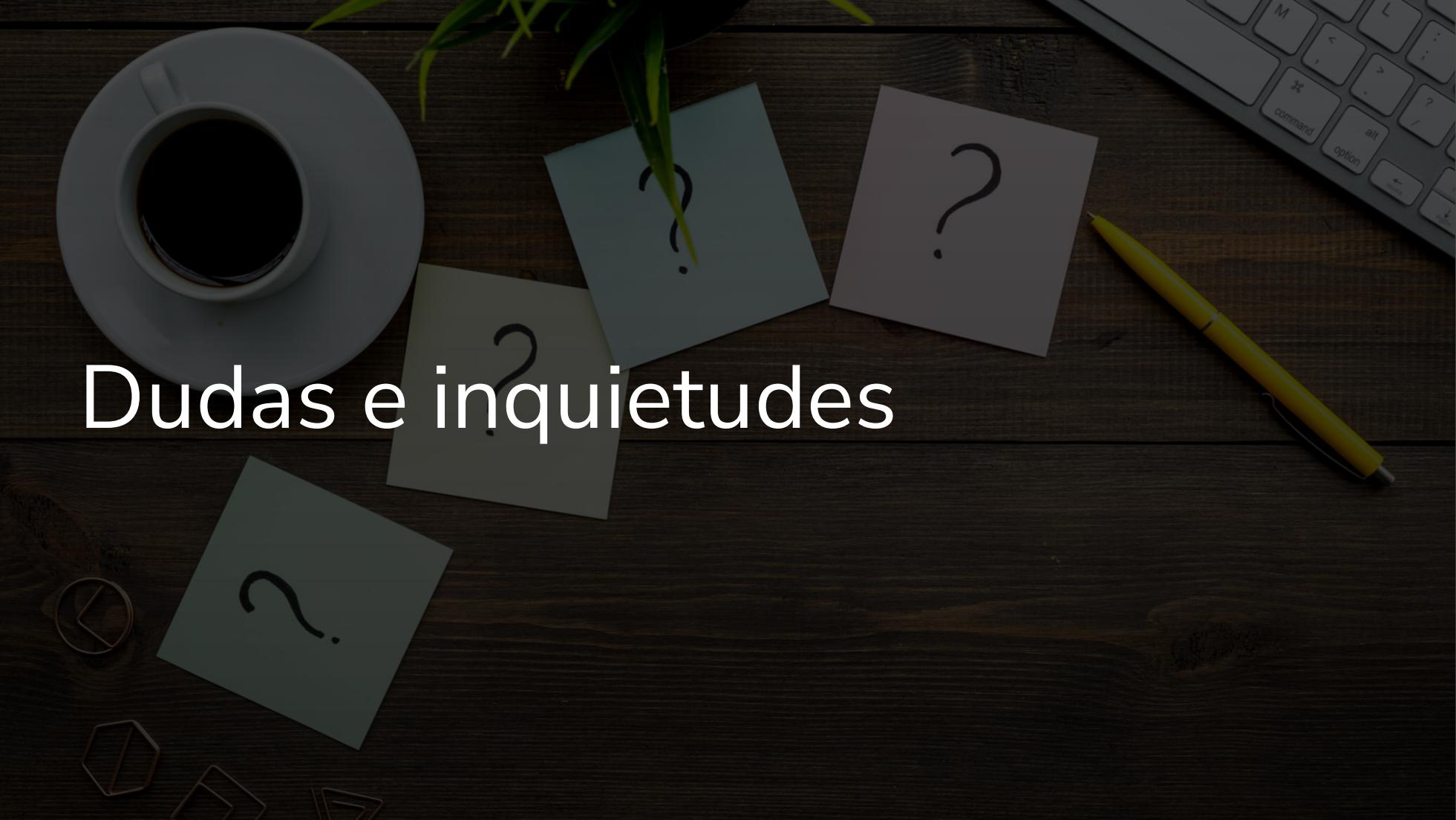
Pero al ser sedónimo puede llegar a facilitar la trazabilidad de donde va el dinero y aunque hay una guerra ética al respecto, puede ser una tecnología que facilite esta trazabilidad de forma transparente.



Bitcoin Pizza Day

El 22 de mayo de 2010 es muy importante para Bitcoin (BTC), ya que esta es la fecha en que Laszlo Hayneck realizó el primer pago con bitcoins por un bien o servicio. Hayneck compró dos pizzas en la popular cadena estadounidense Papa John's y pagó 10.000 BTC por ellas (entonces valían 0.003\$). Desde entonces, esta fecha se conmemora todos los años como el Bitcoin Pizza Day.

Dudas e inquietudes



Ethereum

Ethereum es uno de los proyectos más importantes del ecosistema blockchain. Fue el primero en permitir interactuar con la blockchain implementando fragmentos de código que pueden ejecutarse en la propia cadena de bloques, también llamados **Smart-contracts**. Actualmente se encuentra en la versión 2.0 que dentro de sus mayores features está el Proof of Staking.

Smart-Contracts

Los smart contracts se tratan de “scripts” escritos con lenguajes de programación. Esto quiere decir que los términos del contrato son puras sentencias y comandos en el código que lo forma.

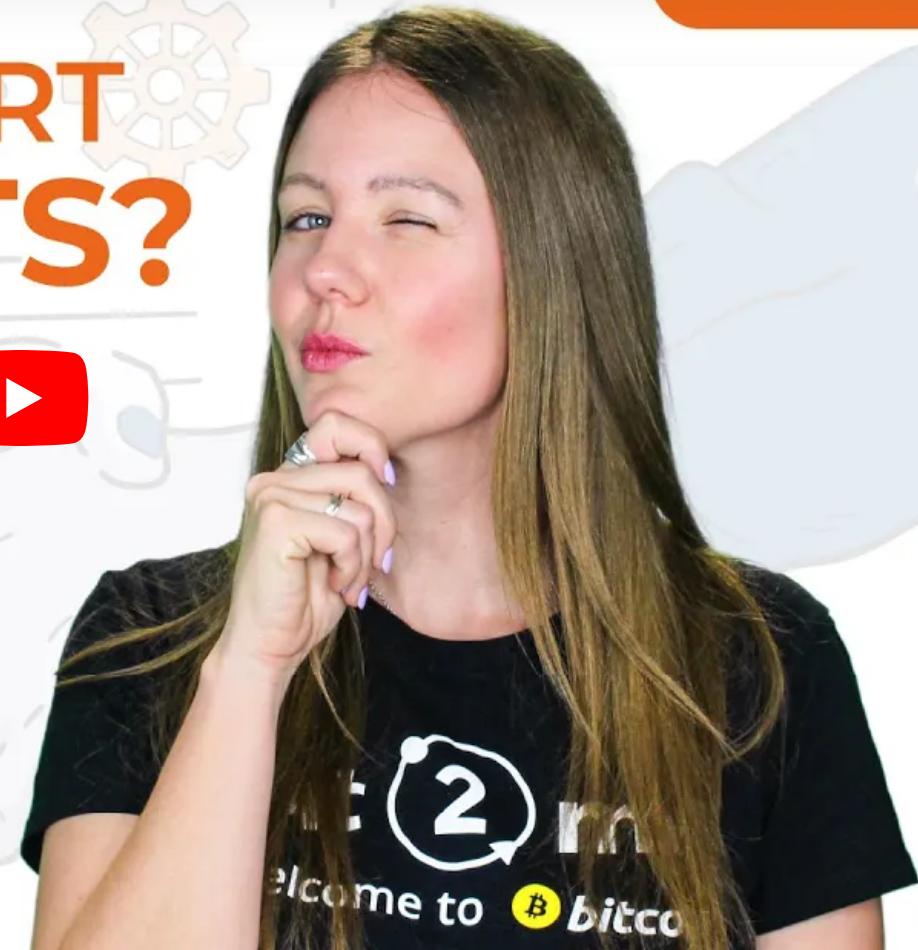
```
pragma solidity ^0.4.17;

contract SimpleStorage {
    uint storedData;

    function set(uint x) public {
        storedData = x;
    }

    function get() public view returns (uint) {
        return storedData;
    }
}
```

¿Qué son los SMART CONTRACTS?



Ethereum 2.0 The Merge

La ultima versión de Ethereum y como ha cambiado de PoW a PoS

Proof of Stake

Proof of Stake (PoS) o Prueba de Participación es un protocolo de consenso creado para reemplazar al conocido Proof of Work aportando una mejor seguridad y escalabilidad a las redes que lo implementen.



¿Qué es **STAKING?**



A white ceramic cup filled with dark coffee, topped with a layer of light-colored foam with small bubbles. The cup sits on a matching white saucer. Scattered coffee beans are visible around the base of the cup and saucer. The background is a warm, blurred yellow and brown.

Coffee Break (15-20 min)

Metamask



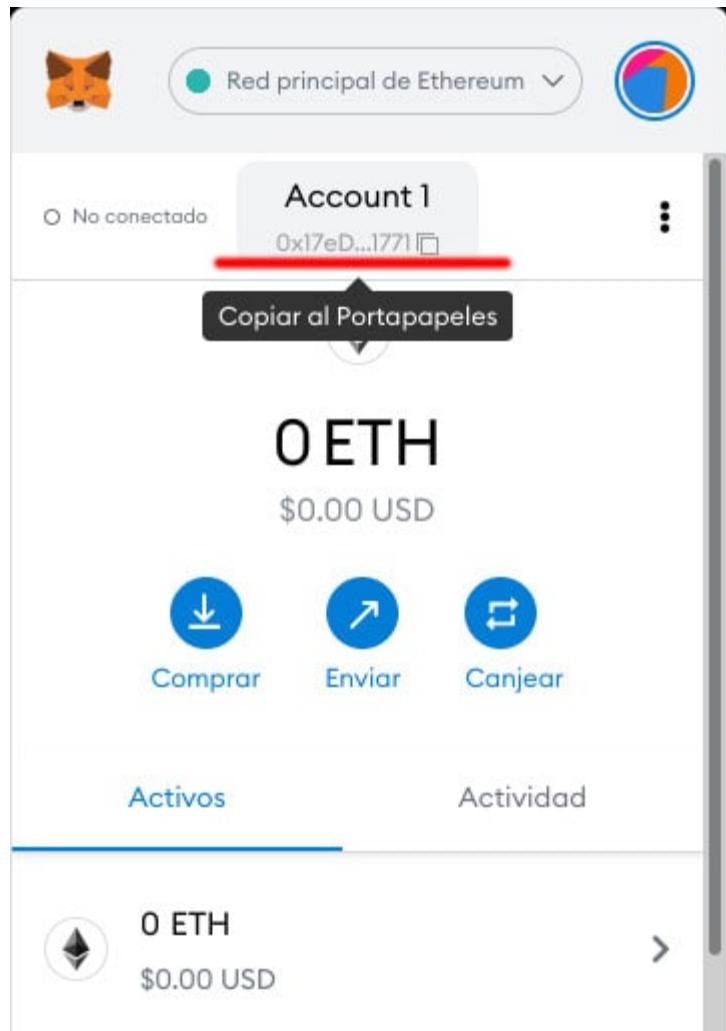
Instalando Metamask

<https://metamask.io/download.html>

Wallet que nos permitirá almacenar y administrar claves, transmitir transacciones, enviar y recibir criptomonedas y tokens basados en la plataforma Ethereum. Será nuestra Wallet

Vamos a seguir los siguientes pasos

- Crearnos una cuenta
- Conocer las diferentes redes de Ethereum



Redes Ethereum ¿Mainnet o Testnet?

Mainnet

Se denomina Mainnet a la red principal de la blockchain, cada plataforma tiene la suya propia. Sería el equivalente al entorno de producción o entorno real.

Testnet

Las plataformas Blockchain están pensadas para desarrolladores, es por ello que toda blockchain tiene su similar a la red principal pero en un entorno de pruebas, para poder crear todo lo que queramos sin gastar dinero real.

Redes Testnet Ethereum

En Ethereum existen actualmente 5 redes disponibles

Sepolia

Goerli

Ropsten (Deprecated)

Rinkeby(Deprecated)

Kovan(Deprecated)

¿Que es un faucet?

Un faucet es un servicio que queda al servicio de la comunidad y que permite pedirle monedas que recibiremos en nuestro monedero

- <https://goerlifaucet.com/>
- <https://goerli-faucet.mudit.blog/>
- <https://www.allthatnode.com/faucet/ethereum.dsrv>

Etherscan Explorer

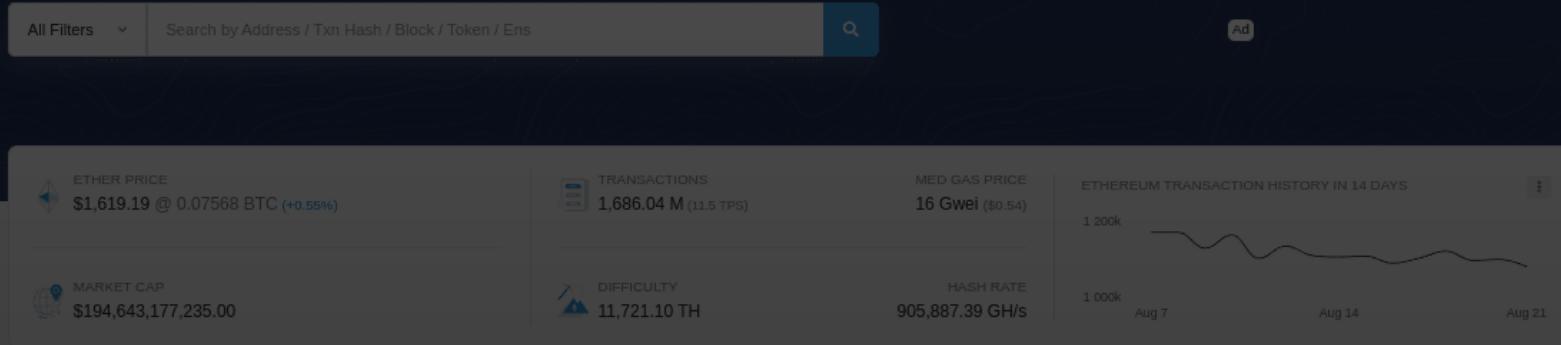
Un Explorer permite conocer todo lo que ocurre en una blockchain.

Uno de los mas conocidos de la red Ethereum es <https://etherscan.io>

Nosotros como trabajaremos con goerli, entraremos en <https://goerli.etherscan.io>

IMPORTANTE Nos registraremos en etherscan, más adelante explicaremos por qué

The Ethereum Blockchain Explorer



Conociendo Etherscan (15 min)

Latest Blocks			
Bk	15393363	Miner F2pool	2.267 Eth 68 txns in 23 secs
Bk	15393364	Miner F2pool	2.153 Eth 21 secs ago
Bk	15393365	Miner Poolin 4	2.081 Eth 215 txns in 17 secs
Bk	15393366	Miner GPUMINE Pool 1	2.119 Eth 333 txns in 17 secs
Bk	15393367	Miner Ethermine	2.098 Eth 324 txns in 28 secs
Bk	15393368	Miner Ethermine	2.035 Eth 303 txns in 23 secs
Bk	15393369	Miner Hiveon Pool	2.046 Eth 141 txns in 3 secs

[View all blocks](#)

Latest Transactions			
Tx	0x7a917b9e702a...	From 0x155072d97c80a83fa...	0.001 Eth
Tx	0x7a917b9e702a...	To 0x00000000000000000000000000000000...	0.001 Eth
Tx	0xc5141e0c95fc5...	From 0xfbbaaa1517b8a47dc1...	0.16 Eth
Tx	0xc5141e0c95fc5...	To 0x970436bb788809dc6f...	0.16 Eth
Tx	0x316705046747...	From 0xcc0a54835d9855b2ee...	0 Eth
Tx	0x316705046747...	To 0xe75ef1ec029c1c9db0...	0 Eth
Tx	0xffff286765da2...	From 0xf719071657308f7f139...	0 Eth
Tx	0xffff286765da2...	To 0x3f7724180aa6b9398...	0 Eth
Tx	0x0dddb786e415...	From 0xe41b294178e708495c...	0 Eth
Tx	0x0dddb786e415...	To 0x00000000000000000000...	0 Eth

[View all transactions](#)



Preparando el entorno

Crearemos la siguiente carpeta

```
`mkdir my-first-nft`
```

Entraremos en la carpeta recien creada

```
`cd my-first-nft`
```

Iniciamos el repositorio con npm

```
`npm i -y`
```

En cuanto a las dependencias del proyecto

```
`yarn add @openzeppelin/contracts`
```

```
`yarn add hardhat`
```

Inicializamos el proyecto `npx hardhat`

Redes Infura

my-first-nft ENDPOINTS SETTINGS SECURITY PROJECT SHARING

STATS

TXNS

IPFS

API KEY
150ca3c21c3640699555adbb0e35486

NETWORK ENDPOINTS ?
HTTPS WEBSOCKETS ▾ REFINE

GET STARTED WITH WEB3

Ethereum L1 Blockchain

Ethereum is the community-run technology powering the cryptocurrency ether (ETH) and thousands of decentralized applications.

GÖRLI <https://goerli.infura.io/v3/150ca3c21c3640699555adbb0e35486>

Polygon Ethereum Sidechain
select

Optimism Ethereum L2/Rollup
select

Arbitrum Ethereum L2/Rollup

Palm Ethereum NFT Sidechain

ERC721 y Presets de Open Zeppelin

Los Smart-contracts en solidity tienen un gran parecido con las clases en el paradigma OOP(Object Oriented Programming) Los contratos tienen datos persistentes en las variables del estado y funciones que modifican estas variables. Si llamasemos a una funcion de otro contrato, ejecutaría una funcion en EVM(Ethereum Virtual Machine)

Creando nuestro smart-contract

Iremos a la carpeta contracts y crearemos el fichero Tutorial.sol

Con el siguiente contenido

```
1 // SPDX-License-Identifier: Unlicense
2 pragma solidity 0.8.4;
3
4 import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
5
6 contract Tutorial is ERC721 {
7
8     uint256 public tokenCounter;
9
10    constructor(
11        string memory name,
12        string memory symbol
13    ) ERC721(name, symbol) {
14        tokenCounter = 0;
15    }
16
17 }
```

Que es tokenCounter Es una propiedad que hemos añadido a nuestro smart-contract y que cuando se construye, por defecto se pone a 0, de forma que cuando se inicie nuestro nft, tendremos 0 tokens emitidos

```
1 uint256 public tokenCounter;
```

- La herencia de OpenZeppelin ERC721 implementa en nuestra clase toda la funcionalidad necesaria para este estándar.
- El constructor con el que se instanciará nuestro Smart-contract, recibiendo dos parametros
- name y symbol son propiedades que tendrá nuestro NFT

Añadiendo la funcionalidad de Minting

A nuestro Tutorial.sol incluiremos la siguiente función

```
1  function mint(string memory _tokenURI) public {
2      _safeMint(msg.sender, tokenCounter);
3      _setTokenURI(tokenCounter, _tokenURI);
4
5      tokenCounter++;
6 }
```

`_safeMint` es otra función que existe en OpenZeppelin ERC721 y que nos implementa los siguientes parametros

- to: Dirección destino donde generaremos el nuevo token
- tokenId: el nuevo id del token minteado

`msg.sender` Es una palabra reservada que devuelve la dirección desde la cual estamos llamando al smart-contract.

`_setTokenURI()` A continuación explicaremos para qué necesitamos este método

Implementando setTokenURI()

Nuestro NFT, deberá tener un mapa de URIs donde nuestro smart-contract guardará el mapeo de los tokens disponibles

Incluiremos esta propiedad debajo del counter

```
1 mapping (uint256 => string) private _tokenURIs;
```

Y ahora implementaremos el siguiente método, que nos permitirá añadir en nuestro mapa de tokens una nueva URI

```
1 function _setTokenURI(uint256 _tokenId, string memory _tokenURI) internal virtual {
2     require(
3         _exists(_tokenId),
4         "ERC721Metadata: URI set of nonexistent token"
5     ); // Checks if the tokenId exists
6     _tokenURIs[_tokenId] = _tokenURI;
7 }
8 }
```

Explicación de los nuevos términos que han aparecido

- `internal` Esta declaración define que este método solo va a poder ser llamado por otros métodos del mismo smart-contract
- `virtual` Virtual permite que sea sobreescrita esta función por otra que herede de la clase.
- `require` Obliga que se cumpla la condición descrita como booleano, sería lo más parecido a un if
- `_exists()` Comprueba si ya existe un token con el id introducido, devolviendo un booleano

Creando tokenURI()

Esta función es la encargada de devolver nuestro token, es el nombre estándar que utilizan plataformas como OpenSea para obtener la información del NFT y sus propiedades

```
1  function tokenURI(uint256 _tokenId) public view virtual override returns(string memory) {
2      require(
3          _exists(_tokenId),
4          "ERC721Metadata: URI set of nonexistent token"
5      );
6      return _tokenURIs[_tokenId];
7  }
```

- `public` Determina que una función es pública y puede llamarse en cualquier momento
- `view` Cuando un método tiene la propiedad view, indica que no modifica el estado de la blockchain, por lo que no gasta ningún tipo de comisión (Gas) para realizar esta acción
- `override` Sobreescribe la funcionalidad que tuviera este método en la clase padre `ERC721` de OpenZeppelin. Por defecto este método realiza la siguiente acción: `baseURI + tokenId`
- `returns(string memory)` Indica que va a devolver un valor de tipo sstring, la propiedad memory indica que la información está guardada en el smart-contract

Esto sería nuestro fichero final

```
1  // SPDX-License-Identifier: Unlicense
2  pragma solidity 0.8.4;
3
4  import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
5
6  contract Tutorial is ERC721 {
7
8      uint256 public tokenCounter;
9      mapping (uint256 => string) private _tokenURIs;
10
11     constructor(
12         string memory name,
13         string memory symbol
14     ) ERC721(name, symbol) {
15         tokenCounter = 0;
16     }
17
18     function mint(string memory _tokenURI) public {
19         _safeMint(msg.sender, tokenCounter);
20         _setTokenURI(tokenCounter, _tokenURI);
21
22         tokenCounter++;
23     }
24 }
```

Compilando el Smart-contract

```
`npx hardhat compile`
```

Importante Revisad que el fichero `hardhat.config.js` contiene la versión `0.8.4` de solidity, que es la versión que estamos utilizando para nuestro SmartContract

Testeando el smart contract

En este tutorial vamos a realizar los siguientes tests

- El NFT ha sido minteado correctamente
- El tokenURI ha sido modificado correctamente

Chai

Para nuestros tests, utilizaremos chai, una assertion library que nos permitirá lanzar los tests en entorno nodejs

```
`yarn add chai`
```

Creando el test

Iremos a la carpeta test y eliminaremos cualquier fichero que haya dentro de esta carpeta.

Crearémos el siguiente script: `tutorial-test.js`

Con el siguiente contenido

```
1  const { expect } = require('chai');
2  const { ethers } = require("hardhat")
3
4  describe("Tutorial Smart Contract Tests", function() {
5
6      this.beforeEach(async function() {
7          // This is executed before each test
8      })
9
10     it("Deployed contract has correct name", async () => {
11         const deployedContract = await ethers.getContractFactory("SimpleStorage").deploy();
12         expect(deployedContract.name()).to.equal("SimpleStorage");
13     })
14
15     it("Initial value of storage is 123", async () => {
16         const deployedContract = await ethers.getContractFactory("SimpleStorage").deploy();
17         const initialValue = await deployedContract.retrieve();
18         expect(initialValue).to.equal("123");
19     })
20
21     it("After setting value, new value is stored", async () => {
22         const deployedContract = await ethers.getContractFactory("SimpleStorage").deploy();
23         const initialBalance = await deployedContract.retrieve();
24         expect(initialBalance).to.equal("123");
25
26         const txResponse = await deployedContract.store("123456789");
27         await txResponse.wait();
28
29         const finalBalance = await deployedContract.retrieve();
30         expect(finalBalance).to.equal("123456789");
31     })
32
33 })
```

- `describe` Set de tests que queremos definir
- `beforeEach` Función que se ejecutará antes de cada test
- `it` Define cada test que queremos desarrollar

Deploy del contrato y escribiendo los tests

Antes de cada test, haremos un deploy de nuestro contrato

```
1 let tutorialContract;
2
3 this.beforeEach(async function() {
4     // This is executed before each test
5     // Deploying the smart contract
6     const Tutorial = await ethers.getContractFactory("Tutorial");
7     tutorialContract = await Tutorial.deploy("Tutorial Contract", "TUT");
8 })
```

Probando la función mint

```
1  it("NFT is minted successfully", async function () {
2      [account1] = await ethers.getSigners();
3
4      const balanceBeforeMint = await tutorialContract.balanceOf(account1.address);
5      expect(balanceBeforeMint.toNumber()).to.equal(0);
6
7      const tokenURI =
8          "https://opensea-creatures-api.herokuapp.com/api/creature/1";
9      const tx = await tutorialContract.connect(account1).mint(tokenURI);
10
11     const balanceAfterMint = await tutorialContract.balanceOf(account1.address);
12     expect(balanceAfterMint.toNumber()).to.equal(1);
13 });


```

Probando la función tokenURI

```
1  it("tokenURI is set sucessfully", async function() {
2      [account1, account2] = await ethers.getSigners();
3
4      const tokenURI_1 = "https://opensea-creatures-api.herokuapp.com/api/creature/1"
5      const tokenURI_2 = "https://opensea-creatures-api.herokuapp.com/api/creature/2"
6
7      const tx1 = await tutorial.connect(account1).mint(tokenURI_1);
8      const tx2 = await tutorial.connect(account2).mint(tokenURI_2);
9
10     expect(await tutorial.tokenURI(0)).to.equal(tokenURI_1);
11     expect(await tutorial.tokenURI(1)).to.equal(tokenURI_2);
12
13 })
```

Como quedaría nuestro test

```
1  const { expect } = require('chai');
2  const { ethers } = require("hardhat")
3
4  describe("Tutorial Smart Contract Tests", function() {
5
6      let tutorialContract;
7
8      this.beforeEach(async function() {
9          // This is executed before each test
10         // Deploying the smart contract
11         const Tutorial = await ethers.getContractFactory("Tutorial");
12         tutorialContract = await Tutorial.deploy("Tutorial Contract", "ART");
13     })
14
15     it("NFT is minted successfully", async function () {
16         [account1] = await ethers.getSigners();
17
18         const balanceBeforeMint = await tutorialContract.balanceOf(account1.address);
19         expect(balanceBeforeMint.toNumber()).to.equal(0);
20
21         const tokenURI =
22             "https://opensea-creatures-api.herokuapp.com/api/creature/1";
```

Lanzando los tests

```
`npx hardhat test`
```

Deploy del Smart Contract

Dotenv

Nos permitirá añadir nuestras variables de entorno para conectar con el api de infura y añadir nuestro private key `yarn add dotenv`

@nomiclabs/hardhat-etherscan

Nos permitirá verificar nuestro smart contract desplegado en la red mediante etherscan `yarn add @nomiclabs/hardhat-etherscan`

@nomiclabs/hardhat-waffle

Framework que facilita los tests de smart contracts `yarn add @nomiclabs/hardhat-waffle`

Configuración dotenv

Creamos el fichero ` `.env` e incluimos la siguiente configuración

```
1 INFURA_KEY=Paste the API key here  
2 PRIVATE_KEY=Paste the private key here  
3 ETHERSCAN_KEY=Paste etherscan key
```

Ampliando la configuración de hardhat

Modificaremos el fichero `hardhat.config.js`

```
1  require("@nomiclabs/hardhat-waffle");
2  require("@nomiclabs/hardhat-etherscan")
3  require("dotenv").config();
4
5  task("accounts", "Prints the list of accounts", async (taskArgs, hre) => {
6    const accounts = await hre.ethers.getSigners();
7
8    for (const account of accounts) {
9      console.log(account.address);
10   }
11 });
12
13 task("deploy", "Deploy the smart contracts", async(taskArgs, hre) => {
14
15   const Tutorial = await hre.ethers.getContractFactory("Tutorial");
16   const tutorialContract = await Tutorial.deploy("Tutorial Contract", "ART");
```

Deploy del Smart Contract

Para poder subir nuestro smart contract, ejecutaremos el siguiente comando

```
`npx hardhat deploy --network goerli`
```

Posibles Errores

Fondos insuficientes, para ello, tendremos que conseguir más fondos en nuestro monedero de Goerli

```
1  Error: insufficient funds for intrinsic transaction cost
2  (error={"name":"ProviderError","code":-32000,
3  "_isProviderError:true}, method="sendTransaction",
4  transaction=undefined, code=INSUFFICIENT_FUNDS, version=providers/5.4.5)
5  ...
6  reason: 'insufficient funds for intrinsic transaction cost',
7  code: 'INSUFFICIENT_FUNDS',
8  error: ProviderError: insufficient funds for gas * price + value
9  ...
10 method: 'sendTransaction',
11 transaction: undefined
```

Volver al paso de los faucets

Contract 0x12Fc3C44b4092aD55cf0212fa3A84a1210fCED5f



Contract Overview

Balance:

10 MATIC

Mintiendo con nuestro Contrato

Entrando en goerli.etherscan.io poremos buscarlo mediante la dirección que se ha generado al hacer el deploy.

Transactions Contract ✓ Events

Una vez estemos allí, en el método mint, podremos introducir una URI para probarlo.

Code

Read Contract

Write Contract

Para probarlo, podemos introducir la siguiente URI:

● Connect to Web3

[Reset]

<https://gambakitties-metadata.herokuapp.com/metadata/1>

2. mint Esta URI ya contiene los metadatos necesarios para que el nft se genere correctamente.

_tokenURI (string)

_tokenURI (string)

Write

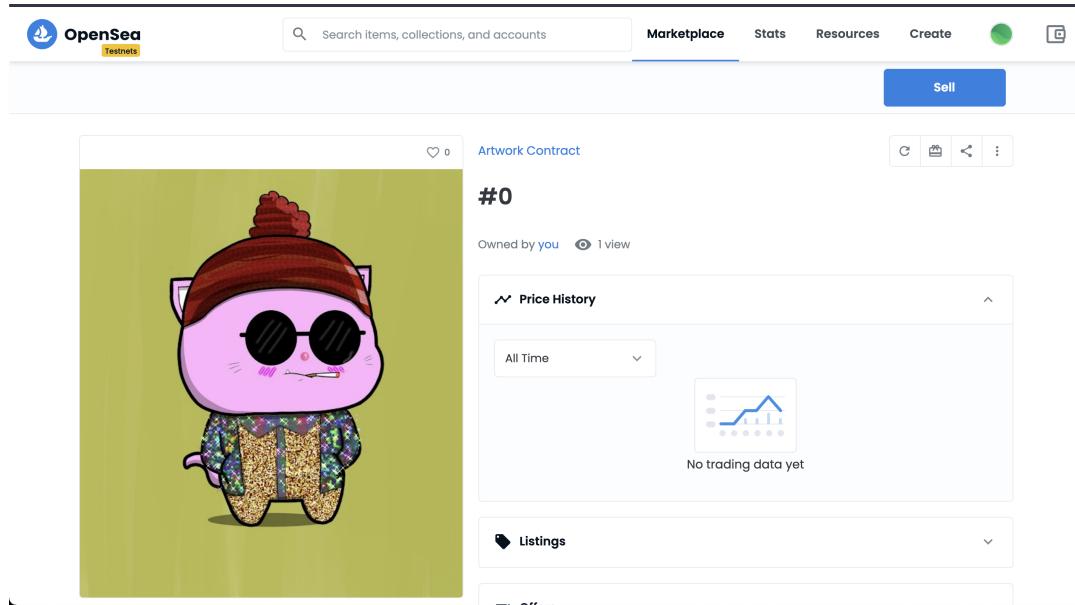
3. safeTransferFrom

Conectandonos con OpenSea Testnet

Iremos a OpenSea Testnet

Y conectaremos metamask

Una vez conectado, si entramos en nuestra colección, deberia aparecernos nuestro nft



3. Deployment and interaction (mint, burn, pause ...)

4. External users interaction (sell, transfer, bid ...)

Metadata

ERC721



RUFFLE

Deployment and
interaction



NFT



3



4



ERC-721 contiene un string metadata que es principalmente la base de los NFT. Hasta este punto no he profundizado mucho en esto, pero vamos a explicar que es el sistema de ficheros IPFS y que relación tiene con los NFT No entraremos en detalle, pero recomiendo una lectura de este articulo de OpenSea sobre metadata y descentralización <https://opensea.io/blog/announcements/decentralizing-nft-metadata-on-opensea/>



Asset



ERC721 Json SCHEMA



Metadata

Pinata

Pinata es un servicio en la nube que permite generar ficheros con IPFS y así poder utilizarlos para nuestros smart contracts

Esta es una imagen de ejemplo que subí

<https://gateway.pinata.cloud/ipfs/QmZM2ywL5z9UYM>



Como generar metadata en fichero json

Aquí hay un ejemplo de metadata.json

```
1  {
2      "description": "Friendly OpenSea Creature",
3      "image": "https://opensea-prod.appspot.com/puffs/3.png",
4      "name": "Dave Starbelly",
5      "attributes": [
6          { "trait_type": "Base", "value": "Starfish" },
7          { "trait_type": "Eyes", "value": "Big" },
8          { "trait_type": "Mouth", "value": "Surprised" },
9      ]
10 }
```

- `description` Breve descripción del nft
- `image` Url a la imagen que vamos a vincular al nft
- `name` Nombre del NFT
- `attributes` Propiedades únicas que tendrá nuestro NFT a la hora de mintearlo, recomiendo indagar más al respecto, ya que es extenso este apartado y no vamos a profundizar aquí

Subiendo el fichero a pinata

- Creamos una cuenta
- Subimos la carpeta donde tengamos el `nft.json` y `nft.png` por ejemplo
- Se generará un CID

Al indicar la url de la siguiente forma, podremos visitar nuestro IPFS

<https://ipfs.io/ipfs/{CID}/nft.png>

Mintiendo nuestro nuevo NFT

Si volvemos al smart-contract que ya teníamos desplegado, si vamos a la función de mint e indicamos el nuevo URI, podremos ver como estaremos generando un nuevo NFT con las propiedades que tenía `nft.json`

**Enhorabuena, ya has conseguido completar el
ciclo completo de un NFT**

Ahora solo te queda seguir aprendiendo y experimentando.

Espero que hayas disfrutado este tutorial.