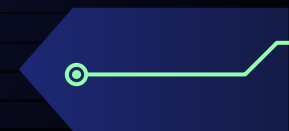


Creación de NFTs: ERC-721 y ERC-1155

Omar Delgado López






Introducción

- Los NFTs (Non Fungible Tokens) son activos digitales únicos que representan la propiedad verificable de un elemento, ya sea una obra de arte, coleccionable, videojuego o incluso bienes raíces virtuales.
 - Ésta singularidad se logra mediante la tecnología blockchain, que garantiza la autenticidad y propiedad de cada NFT.
 - Los estándares ERC-721 y ERC-1155, definidos por Ethereum Improvement Proposals (EIPs), se han convertido en los pilares fundamentales para la creación de NFTs.
 - Mientras que ERC-721 permite la creación de NFTs de forma individual asociando cada uno con un identificador único, ERC-1155 ofrece la flexibilidad de permitir la creación de múltiples tokens fungibles (intercambiables) y no fungibles a la vez.
- 



Tokens fungibles y no fungibles




- Fungible es una palabra equivalente a intercambiable, es decir, cualquier objeto que pueda ser intercambiado por otro objeto de su mismo tipo y valor como, por ejemplo, el dinero.
 - Criptomonedas como Bitcoin o Ethereum son ejemplos de tokens fungibles. Un Bitcoin es igual a otro en términos de valor y funcionalidad.
 - Por otra parte, ya se ha mencionado que un NFT representa un activo único y su autenticidad.
 - Objetos como obras de arte digitales, coleccionables en videojuegos o vienes raíces virtuales son ejemplos de NFTs. Cada uno tiene un valor y autenticidad únicos.
- 
- 
- 

Tokens fungibles

- **Intercambiables:** Un token puede ser intercambiado por otro de igual valor.
- **Divisibles:** Al ser intercambiables, pueden ser también divisibles. Un Bitcoin, por ejemplo, puede ser divisible en unidades más pequeñas llamadas Satoshis, equivaliendo un Bitcoin a 100 millones de Satoshis.
- **Uniformidad y valor constante:** Todos los tokens de un mismo tipo son iguales en valor y funcionalidad, lo cual facilita su intercambiabilidad.

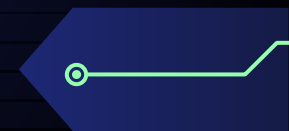


Tokens no fungibles

- **Únicos:** Cada uno tiene un identificador único que lo distingue de cualquier otro.
 - **Indivisibles:** Representan un elemento en su totalidad.
 - **Propiedad verificable:** Su propiedad está registrada en la blockchain, garantizando la autenticidad y singularidad.
- 
- 
- 



Estándar ERC-721

- Es un estándar de tokens no fungibles que establece ciertas reglas para la creación de NFTs de manera individual en la cadena de bloques Ethereum mediante smart contracts.
 - Permite la transferibilidad de un NFT de un propietario a otro, abriendo la puerta al comercio y venta de los activos digitales.
 - Permite asociar metadatos a cada token, incluyendo descripciones, imágenes u otros datos relevantes sobre el activo.
 - El estándar define una serie de funciones que deben implementar todos los contratos, incluyendo funciones para transferir tokens, comprobar el propietario, aprobar transferencias y, por supuesto, crear un NFT.
- 

Estándar ERC-1155



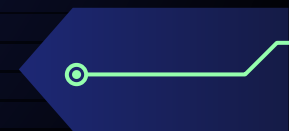
- Está basado en los estándares ERC-20 y ERC-721, definidos para la creación de tokens fungibles y no fungibles respectivamente, consiguiendo unificar la creación y transferencia de ámbos tipos de token mediante un único smart contract.
- Permite la creación y transferencia de lotes, es decir, de múltiples tokens tanto fungibles como no fungibles en una sola operación. Esto permite aumentar la eficiencia reduciendo los costes de operar con un sólo token a la vez.
- Atomicidad: o todas las transferencias de tokens en una transacción por lotes se completan con éxito o no lo hará ninguna.
- Permite la destrucción eficiente de un token ERC-1155.
- Igual que el ERC-721, también posee una serie de funciones predefinidas para crear, transferir, comprobar el propietario o balance, destruir, etc, tanto de tokens individuales como de lotes.

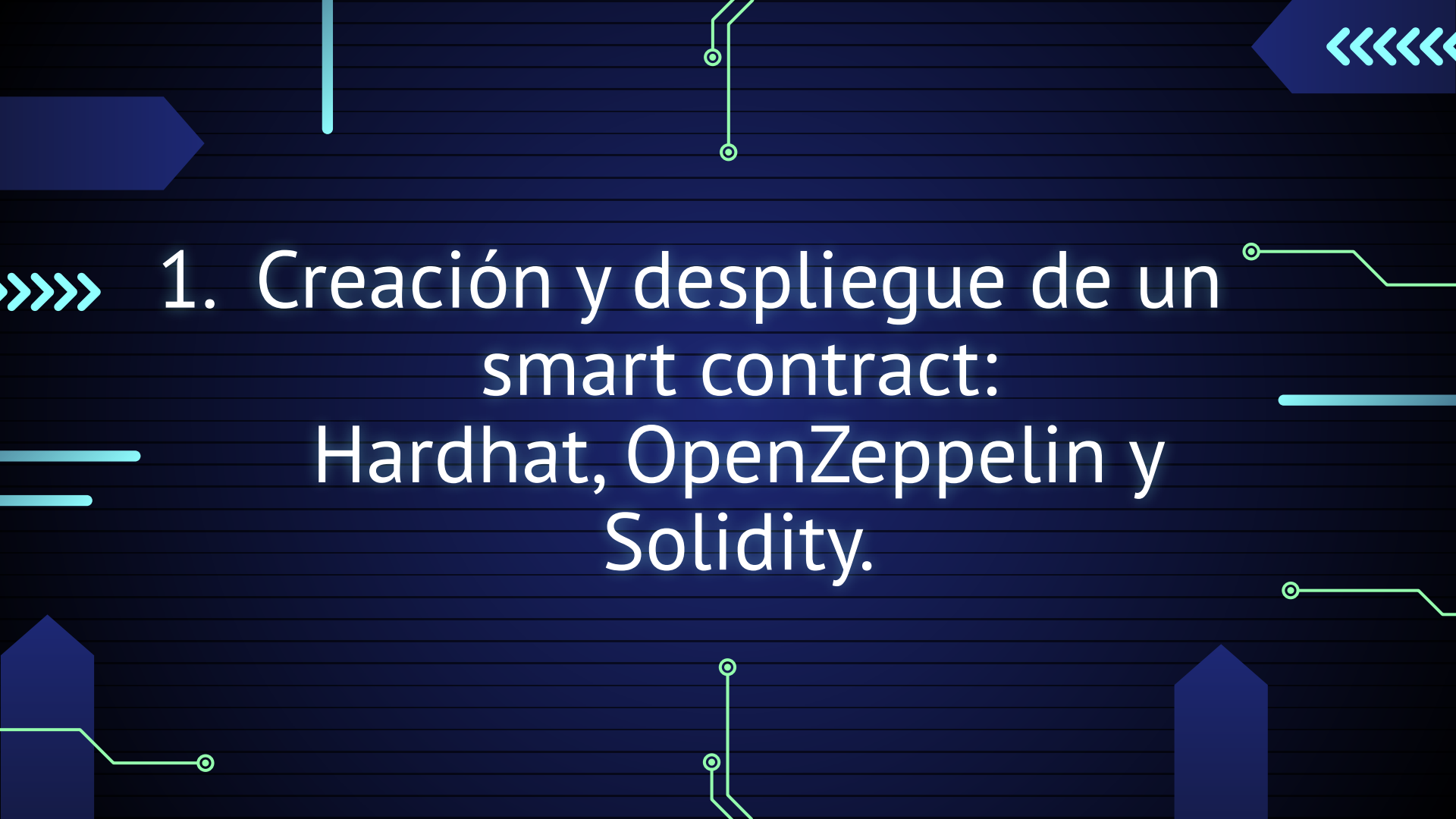


Proceso de creación



Proceso de creación

1. Creación y despliegue de un smart contract: Hardhat, OpenZeppelin y Solidity.
 2. Creación de los metadatos del token en IPFS a través de Pinnata.
 3. Llamada al smart contract y sus funciones: un sencillo frontend con Angular.
 4. Visualización de tokens: OpenSea.
- 
- 
- 



1. Creación y despliegue de un smart contract: Hardhat, OpenZeppelin y Solidity.

Smart contract ERC-721

- La forma habitual de desarrollar un smart contract es a través del lenguaje Solidity mediante el uso de librerías OpenZeppelin. El contenido de dichas librerías contiene aquellas funciones predefinidas en los estándares, lo cual es altamente útil para comprender el funcionamiento básico de la mayoría de variables y funciones que se pueden utilizar. Pueden ser accesibles a través de git:

[OpenZeppelin/ERC-721.sol](#)

[OpenZeppelin/ERC-1155](#)

```
pragma solidity ^0.7.3;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/utils/Counters.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
```

Smart contract ERC-721

- En este caso vemos que el contrato sigue el estándar ERC-721 y, además, es Ownable. Ésta propiedad indica que el contrato puede ser propiedad de un usuario, por defecto aquel que lo ha desplegado, y tiene sobre éste una serie de permisos administrativos como pausar su funcionamiento o la modificación de determinados parámetros.

```
contract mint_721 is ERC721, Ownable {  
  
    // Variable que nos permite incrementar el identificador único del NFT  
    using Counters for Counters.Counter;  
    Counters.Counter private _tokenIds;  
  
    // Variable que define al propietario del NFT  
    address private _owner;  
  
    constructor() ERC721("NST APP", "NFT") {}  
}
```

- La propiedad de un contrato es pública y puede ser verificada. A mayores, evita que el mismo pueda ser accedido o modificado sin autorización.

Smart contract ERC-721

```
constructor() ERC721("NST APP", "NFT") {}

/** Funcion definida por el usuario que recibe como parametros el URI a los metadatos y la direccion del propietario del NFT
    |   Devuelve el ID del nuevo token
    */
function mintNFT(address recipient, string memory tokenURI) public returns (uint256) {

    _tokenIds.increment();

    // Identificador unico del NFT
    uint256 newItemId = _tokenIds.current();

    /** Llamadas a dos funciones predefinidas en el estándar ERC-721
        |   _mint para mintear el NFT asociandolo a su propietario y un ID unico
        |   _setTokenURI para asociar el URI con los metadatos al identificador del NFT
        */
    _mint(recipient, newItemId);
    _setTokenURI(newItemId, tokenURI);

    return newItemId;
}
}
```

- En éste caso, el contrato ha sido simplificado al máximo permitiendo exclusivamente utilizar una única funcion para crear un NFT.

Smart contract ERC-1155

- Una de las principales diferencias en el estándar ERC-1155, es que en el constructor del contrato se añade el URI de un JSON que contendrá los metadatos de los distintos items que el contrato permite, limitando de esta manera el tipo de tokens que se puede crear con el contrato

```
constructor() public ERC1155("https://game.example/api/item/{id}.json") {  
    _mint(msg.sender, COPPER, 10**18, "");  
    _mint(msg.sender, CRYSTAL, 10**27, "");  
    _mint(msg.sender, ELDER_SWORD, 1, "");  
    _mint(msg.sender, KNIFE, 10**9, "");  
    _mint(msg.sender, WAND, 10**9, "");  
}
```

- Lo más habitual es, además usar la funcion predefinida en el estándar _mint para crear la cantidad de tokens de cada tipo dentro del propio constructor. Ésto limitará la cantidad de tokens que hay en la colección, permitiendo la creación de un token no fungible limitando a 1 su cantidad.

Smart contract ERC-1155

- Por otra parte, se puede limitar el tipo de token que cada usuario puede llegar a tener. En el ejemplo posterior se ha tomado la decisión de no crear una cantidad predefinida en el constructor, si no permitir que cada usuario pueda tener una cantidad máxima de cada token, para poder ver más comodamente la creación y manejo de éste tipo de tokens desde la aplicación.

Smart contract ERC-1155

```
pragma solidity ^0.7.3;

import "@openzeppelin/contracts/token/ERC1155/ERC1155.sol";

contract mint_1155 is ERC1155 {

    // Los mapeos permiten llevar la cuenta de determinados valores asociados a variables
    // En este caso, el balance o cantidad total de items asociados a un usuario,
    // el número máximo de cada item que se puede mintear en una sola operación
    // y el número total de elementos que se crean a la vez.

    mapping(uint256 => mapping(address => uint256)) private _itemBalances;

    mapping(uint256 => uint256) private _maxMintPerItem;
    mapping(uint256 => uint256) private _totalMinted;

    constructor() ERC1155("https://gateway.pinata.cloud/ipfs/QmcQKjv4Np3g2vtAaecjvh5eo1XLxTZ3ES7H3w9jNdFwcY/{id}.json") {

        // Se limita de cantidad de cada tipo de token que se puede crear a la vez
        _maxMintPerItem[1] = 1;
        _maxMintPerItem[2] = 10;
        _maxMintPerItem[3] = 100;
        _maxMintPerItem[4] = 1000;

    }
}
```


Smart contract ERC-1155

```
// Se crea un único token
function mintToken(uint256 itemId, uint256 amount) public {
    // Se comprueba que el usuario no sobrepasa la cantidad de tokens creados permitidos
    require(_itemBalances[itemId][msg.sender] + amount <= _maxMintPerItem[itemId], "Exceeds maximum mint limit");

    // Se comprueba que el usuario no excede la cantidad total de tokens permitidos
    require(_totalMinted[itemId] + amount <= _maxTotalMint(itemId), "Exceeds total mint limit");

    _mint(msg.sender, itemId, amount, "");

    // Se actualizan los parametros del usuario
    _itemBalances[itemId][msg.sender] += amount;
    _totalMinted[itemId] += amount;
}

// Se crea un lote de múltiples tipos de token
function mintBatch(uint256[] memory itemIds, uint256[] memory amounts) public {
    require(itemIds.length == amounts.length, "Invalid input length");

    for (uint256 i = 0; i < itemIds.length; i++) {
        mintToken(itemIds[i], amounts[i]);
    }
}
```

Smart contract ERC-1155

```
// Se comprueba la cantidad de un tipo de token que posee un usuario
function getBalances(address account, uint256[] memory itemIds) external view returns (uint256[] memory) {
    uint256[] memory balances = new uint256[](itemIds.length);

    for (uint256 i = 0; i < itemIds.length; i++) {
        balances[i] = balanceOf(account, itemIds[i]);
    }

    return balances;
}

// Se limita el numero maximo de tokens que se pueden mintear en una sola operacion
function _maxTotalMint(uint256 itemId) private pure returns (uint256) {
    if (itemId == 1) {
        return 1;
    } else if (itemId == 2) {
        return 100;
    } else if (itemId == 3) {
        return 1000;
    } else if (itemId == 4) {
        return 10000;
    }
    return 0;
}
```

1. Despliegue y creación del smart contract

- Se usará la herramienta Hardhat con el editor de código deseado, teniendo en cuenta que Remix es un editor especializado y recomendado para smart contracts y su uso y funcionamiento podría variar con el caso que aquí se muestra en el que se ha usado el editor de código habitual.
- A partir de este punto el proceso es similar para ambos smart contracts y la estructura inicial del directorio tras inicializar el proyecto con el comando “npm init” podrá ser similar a la siguiente:

```
✓ CONTRACTS
  ✓ ERC_721
    ✓ contracts
      ERC_721.sol
    > scripts
    {} package.json
```

1. Despliegue y creación del smart contract

- En primer lugar para poder interactuar con la blockchain Ethereum se recomienda usar Alchemy, una plataforma de desarrollo blockchain y API gratuita a través de la cual se podrá conseguir una API key pública necesaria para el proceso.
- Ethereum consta de varias testnet en las que los desarrolladores pueden conseguir una criptomoneda específica sin valor de forma gratuita para realizar pruebas. Ésta podrá ser seleccionada al crear la API en Alchemy y en el ejemplo se ha seleccionado Goerli.
- Se usará Metamask como monedero para éste ejemplo. Deberá tenerse en cuenta que tanto para el despliegue del contrato como para el minteo se requiere la criptomoneda de Goerli, que puede obtenerse gratuitamente a través de cualquier faucet disponible en [free Goerli faucet](#).

1. Despliegue y creación del smart contract

- Instalación de Hardhat y creación de un nuevo proyecto:

```
npm install --save-dev hardhat
```

```
npx hardhat
```

```

 888  888                888 888                888
 888  888                888 888                888
 888  888                888 888                888
88888888888 888b. 888d888 .d88888 88888b. 8888b. 888888
888  888      "88b 888P"  d88" 888 888 "88b      "88b 888
888  888 .d888888 888 888 888 888 888 .d888888 888
888  888 888 888 888 888 Y88b 888 888 888 888 Y88b.
888  888 "Y888888 888      "Y88888 888 888 "Y888888 "Y888

Welcome to Hardhat v2.18.2

? What do you want to do? ...
  Create a JavaScript project
  Create a TypeScript project
  Create a TypeScript project (with Viem)
> Create an empty hardhat.config.js
Quit
```


1. Despliegue y creación del smart contract

```
/** @type import('hardhat/config').HardhatUserConfig */
```



```
require("@nomiclabs/hardhat-ethers");  
.....  
require("@nomicfoundation/hardhat-verify");
```

```
module.exports = {  
  solidity: "0.7.3",  
  // Red en la que se desplegara el contrato  
  defaultNetwork: "goerli",  
  networks: {  
    hardhat: {},  
    goerli: {  
      // API Key de Alchemy  
      url: "https://eth-goerli.g.alchemy.com/v2/jIqdyZC5gfrzGvw_3qju9Af0sRMBs7IZ",  
      // Clave privada de metamask del propietario que creará el contrato  
      accounts: ["0x531bf8ad40390c3b38530fcf5997eda7bfca10beb2eb9d5ad2f434e15857d7ce"]  
    },  
  },  
  // Se requerira una API Key gratuita de Etherscan para poder validar el contrato con hardhat  
  etherscan: {  
    apiKey: "I6T9WSU7IY1MKK8ACIFA7JEXTJRD3CX1XZ"  
  }  
}
```

- hardhat.config.js

1. Despliegue y creación del smart contract

- Es importante tener en cuenta la versión de compilación del contrato y en consecuencia instalar las librerías de OpenZeppelin compatibles con dicha versión:

```
npm install @openzeppelin/contracts@3.1.0-solc-0.7
```

- Se debe instalar ethers.js, una librería que facilita la interacción de peticiones a la blockchain en el formato JSON-RPC para el despliegue de los contratos:

```
npm install --save-dev @nomiclabs/hardhat-ethers 'ethers@^5.0.0'
```

- Se creará un archivo deploy.js en la carpeta scripts para realizar el despliegue.

1. Despliegue y creación del smart contract

- scripts/deploy.js

```
async function main() {  
  const mint_721 = await ethers.getContractFactory("mint_721");  
  
  const mint721 = await mint_721.deploy();  
  console.log("Contract deployed to address:", mint721.address);  
}  
main()  
  .then(() => process.exit(0))  
  .catch(error => {  
    console.error(error);  
    process.exit(1);  
  });
```


1. Despliegue y creación del smart contract

- Compilación y despliegue del contrato:

```
npx hardhat compile
```

```
npx hardhat run scripts/deploy.js --network goerli
```

```
Successfully submitted source code for contract  
Successfully verified contract mint_721 on the block explorer.  
https://goerli.etherscan.io/address/0x1158774683249F57273049166DE9EF8BcE443C32#code
```

- Etherscan permite ver, buscando mediante la dirección, cualquier transacción, contrato o cuenta de propietario y las transacciones que realizan. Hay una version disponible para Goerli y otras testnets, a mayores de la principal: <https://goerli.etherscan.io/>

1. Despliegue y creación del smart contract

- Etherscan ^{***} permite ver también el código de los contratos una vez éstos sean verificados. Verificar un contrato es un proceso simple que permite validar la autenticidad y originalidad de un contrato, y puede realizarse con hardhat de manera sencilla:

1. Se obtiene una API key gratuita en etherscan a través del registro igual que con Alchemy
2. `npm install --save-dev @nomicfoundation/hardhat-verify`
3. Se añaden en `hardhat.config.deploy` las modificaciones pertinentes (ya vistas en el código mostrado)
4. Se introduce el comando de verificación indicando la red y la dirección del contrato

```
npx hardhat verify --network goerli 0x1158774683249F57273049166DE9EF8BcE443C32
```



2. Creación de los metadatos: IPFS y Pinnata

2. Creación de los metadatos: IPFS

- Para la creación del token es necesario previamente la creación de los metadatos y la imagen a la que se asocia, y que éstos estén almacenados apropiadamente. Lo más común es que estén almacenados en un sistema de archivos distribuido como IPFS, al que se accederá a través de la API Pinnata.
- En Pinnata, cualquiera puede crear una cuenta de forma gratuita y subir manualmente el contenido que desee, el cual será almacenado en IPFS. De forma externa, Pinnata aporta una API pública gratuita para que los desarrolladores puedan comunicarse con ella a través de una aplicación mediante peticiones en diversos formatos. La documentación disponible puede consultarse en [pinnata API docs](#)

2. Creación de los metadatos: IPFS

- Existe un estándar predeterminado para los metadatos en formato JSON, que permite almacenar el nombre, descripción, URI de la imagen y un conjunto indefinido de atributos formado por pares con nombre y valor.

```
{
  "name": "Telematic",
  "description": "Connected We Stand, Networked We Triumph.",
  "image": "https://gateway.pinata.cloud/ipfs/QmQWE1C3CsMZ7GCXgRq6aDD9XpvJ44s9KMxQUD5ncLhiq3",
  "attributes": [
    {
      "trait_type": "Hability",
      "value": "Programming"
    },
    {
      "trait_type": "Flag",
      "value": "Blue"
    },
    {
      "trait_type": "Tool",
      "value": "Computer"
    }
  ]
}
```

3. Creación de tokens a través de Angular

3. Creación de tokens a través de Angular



UVigo - NST

Welcome to NST minting APP!

MINT YOUR TOKENS WITH ERC-721 AND ERC-1155 STANDARDS

Log in



FOLLOW THE LINKS TO INSTALL METAMASK AND ACCESS GOERLI TESTNET



3. Creación de tokens a través de Angular

- Para usar Metamask con los smart contracts desplegados en Goerli se debe instalar y conectar a la testnet Goerli previamente. Si no aparece de forma automática, se podrá añadir a través del enlace disponible.
- Se han considerado una serie de comprobaciones al respecto para permitir continuar al usuario. Éstas han sido comprobadas en el navegador Chrome y su correcto funcionamiento en otros navegadores no está asegurado.

```
private checkIfMetamaskInstalled(): boolean {  
    if (typeof (window as any).ethereum !== 'undefined') {  
        this.ethereum = (window as any).ethereum;  
        return true;  
    } else {  
        return false;  
    }  
}
```

- Extensión de metamask instalada en el navegador

3. Creación de tokens a través de Angular

- Usuario logueado en Metamask

```
private async checkUserLogged(): Promise<boolean> {  
  try {  
    await this.ethereum.request({ method: 'eth_requestAccounts' });  
    return true;  
  } catch(error) {  
    return false;  
  }  
}
```

```
GOERLI_NETWORK_ID = 5;  
  
private async checkGoerliTestnet(): Promise<boolean> {  
  const network_id = await this.ethereum.request({method: 'net_version'})  
  
  if (network_id == this.GOERLI_NETWORK_ID) {  
    return true;  
  } else {  
    return false;  
  }  
}
```

- Usuario conectado a la testnet Goerli (las diferentes testnet así como la red principal están asociadas a diferentes IDs)

3. Creación de tokens a través de Angular



UVigo - NST

Welcome to NST minting APP!

MINT YOUR TOKENS WITH ERC-721 AND ERC-1155 STANDARDS

Mint 721

Mint 1155

Show NFTs

3. Creación de tokens: ERC-721



UVigo - NST

NFT DATA

NAME:

DESCRIPTION:

NFT METADATA ATTRIBUTES

+ Add attribute

NFT IMAGE

Seleccionar archivo Ninguno archivo selec.

3. Creación de tokens: ERC-721

- Para la subida a IPFS de la imagen y los metadatos se ha usado un formulario que posteriormente será enviado en formato JSON a través de llamadas a la API de Pinnata.
- Se requerirá el par de claves pública - privada de la API Pinnata del usuario.
- Axios ha sido la librería usada para realizar peticiones a la API, en este caso en formato HTTP. Dicha librería está diseñada para node.js pero puede usarse igualmente desde el frontend.

3. Creación de tokens: ERC-721

NFT DATA

NAME:

Test

DESCRIPTION:

Prueba de demostración

NFT METADATA ATTRIBUTES

+ Add attribute

ATTRIBUTE NAME:

Propietario

ATTRIBUTE NAME:

Yo mismo



NFT IMAGE

Seleccionar archivo satellite3.jpg

Upload image

YOU NEED TO UPLOAD AN IMAGE TO MINT A NFT!

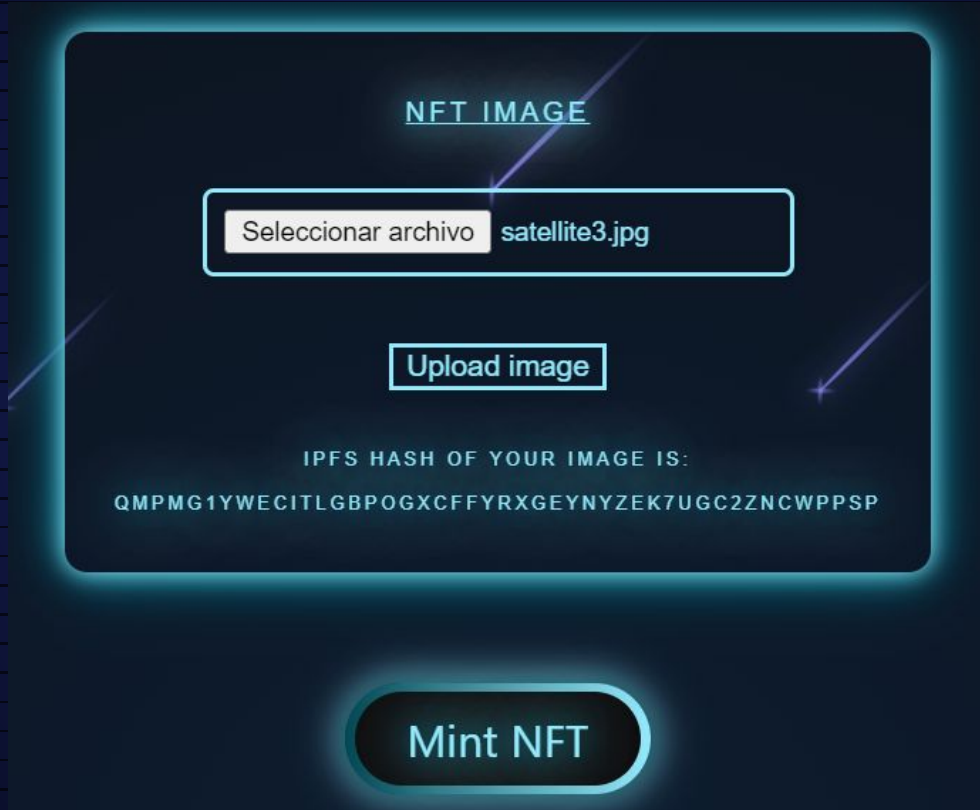
3. Creación de tokens: ERC-721

```
async uploadImage() {  
  
  const data = new FormData();  
  data.append('file', this.image_file);  
  
  await axios.post(this.PINATA_URL_PINFile, data, {  
    headers: {  
      'Content-Type': 'multipart/form-data',  
      'pinata_api_key': this.PINATA_API_PUBLIC_KEY,  
      'pinata_secret_api_key': this.PINATA_API_PRIVATE_KEY  
    })  
    .then((response: any) => {  
      this.image_hash = response.data.IpfsHash;  
      this.isUploaded = true;  
    })  
    .catch((error: any) => {  
      console.error('Error pinning file to IPFS:', error);  
      this.isUploaded = false;  
    })  
  })  
}
```

3. Creación de tokens: ERC-721

```
uploadMetadata() {  
  
  this.form_721.value.image = 'https://gateway.pinata.cloud/ipfs/' + this.image_hash;  
  console.log('Image hash: ', this.form_721.value.image);  
  
  const formdata = this.form_721.value  
  const jsondata = JSON.stringify(formdata)  
  
  return axios.post(this.PINATA_URL_PINJSON, jsondata, {  
    headers: {  
      'Content-Type': 'application/json',  
      'pinata_api_key': this.PINATA_API_PUBLIC_KEY,  
      'pinata_secret_api_key': this.PINATA_API_PRIVATE_KEY  
    }).then((response: any) => {  
      console.log(jsondata)  
      this.metadata = 'ipfs://' + response.data.IpfsHash;  
    })  
    .catch((error: any) => {  
      console.error('Error pinning JSON to IPFS:', error);  
    })  
  })  
}
```


3. Creación de tokens: ERC-721



The screenshot shows a web interface for creating an NFT. It features a central dark panel with a glowing cyan border. At the top, the text "NFT IMAGE" is displayed in cyan. Below it is a file selection box containing the text "Seleccionar archivo" and the filename "satellite3.jpg". Underneath the file box is a button labeled "Upload image". Below the button, the text "IPFS HASH OF YOUR IMAGE IS:" is shown, followed by a long alphanumeric hash: "QMPPMG1YWECITLGBPOGXCFYRXGEYNYZEK7UGC2ZNCWPPSP". At the bottom of the interface is a large, rounded button labeled "Mint NFT".

NFT IMAGE

Seleccionar archivo satellite3.jpg

Upload image

IPFS HASH OF YOUR IMAGE IS:
QMPPMG1YWECITLGBPOGXCFYRXGEYNYZEK7UGC2ZNCWPPSP

Mint NFT

3. Creación de tokens: ERC-721



UVigo - NST

Add attribute

ATTRIBUTE NAME: Propietario

ATTRIBUTE NAME: Yo mismo

NFT IMAGE

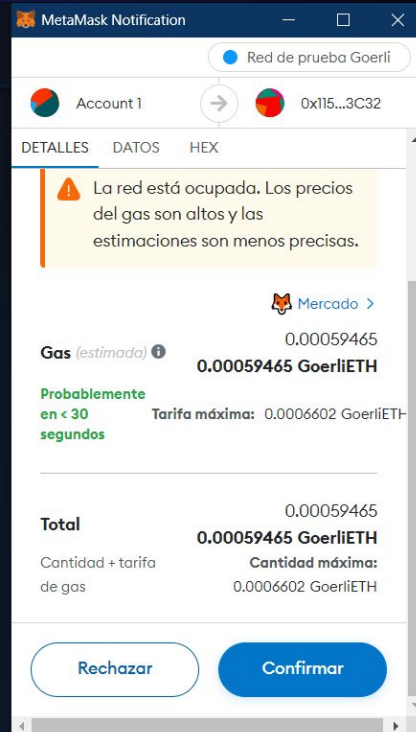
Seleccionar archivo satellite3.jpg

Upload image

IPFS HASH OF YOUR IMAGE IS:

QMPMG1YWECITLGBPOGXCFYRXGEYNYZEK7UGC2ZNCWPPSP

Mint NFT



3. Creación de tokens: ERC-721

- De nuevo, usaremos la librería ethers ya usada anteriormente para el despliegue para hacer llamadas a funciones concretas de un contrato.
- Se requiere usar la ABI (Interfaz Binaria de Aplicación) del contrato. Se genera automáticamente al compilar y define determinados tipos de datos y métodos relacionados con el contrato. Es el modo estándar de interactuar con contratos en Ethereum.
- La ABI que podrá ser accesible desde el directorio donde se han desplegado los contratos en `artifacts/contracts/nombre_del_contrato.json`.

3. Creación de tokens: ERC-721

```
async onSubmit() {  
  if (await this.loginCompleted()) {  
    await this.uploadMetadata();  
  
    const provider_var = new ethers.BrowserProvider((window as any).ethereum);  
  
    await provider_var.send("eth_requestAccounts", []);  
    const signer_var = await provider_var.getSigner();  
  
    const contract_mint_721 = new ethers.Contract(this.CONTRACT_ADDRESS, this.CONTRACT_ABI, signer_var);  
  
    const token_uri = this.metadata;  
  
    console.log('token: ', token_uri)  
  
    try {  
      | await contract_mint_721['mintNFT'](signer_var.getAddress(), token_uri);  
    } catch (err: any) {  
      | console.error(err.message);  
    }  
  
    this.ngZone.run(() => this.router.navigateByUrl('/add-NFT'))  
  } else {  
    | this.ngZone.run(() => this.router.navigateByUrl('/home'))  
  }  
}
```

3. Creación de tokens: ERC-1155

- En la creación de tokens con el estándar ERC-1155 se ha decidido mostrar desde el principio los tokens de cada tipo disponibles en el contrato.
- En éste caso, si el usuario no ha mintado alguno de los tipos, la imagen que se mostrará sera por defecto un interrogante, indicando al usuario que no dispone de tokens del tipo indicado.
- También se mostrará en la interfaz la cantidad máxima de tokens que el usuario tiene y puede tener en propiedad y la cantidad que puede mintear en una sola operación.

3. Creación de tokens: ERC-1155



UVigo - NST

ITEM 1

OWNED: 0/1



MAX AMOUNT PER MINT: 1

MINT:

ITEM 2

OWNED: 0/100

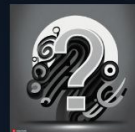


MAX AMOUNT PER MINT: 10

MINT:

ITEM 3

OWNED: 0/1000



MAX AMOUNT PER MINT: 100

MINT:

ITEM 4

OWNED: 0/10000



MAX AMOUNT PER MINT: 1000

MINT:

Mint

3. Creación de tokens: ERC-1155

- Para mostrar la cantidad de tokens de cada tipo que un usuario tiene en propiedad, se llamará a la función correspondiente del contrato:

3. Creación de tokens: ERC-1155

```
const provider_var = new ethers.BrowserProvider((window as any).ethereum);

await provider_var.send("eth_requestAccounts", []);
const signer_var = await provider_var.getSigner();

const contract_mint_1155 = new ethers.Contract(this.CONTRACT_ADDRESS, this.CONTRACT_ABI, signer_var);

try {
  const ammounts = await contract_mint_1155['getBalances'](signer_var.getAddress(), [1, 2, 3, 4])
  if (ammounts[0] > 0) {
    this.someItem1 = true;
    this.amountItem1 = ammounts[0]
  }
  if (ammounts[1] > 0) {
    this.someItem2 = true;
    this.amountItem2 = ammounts[1]
  }
  if (ammounts[2] > 0) {
    this.someItem3 = true;
    this.amountItem3 = ammounts[2]
  }
  if (ammounts[3] > 0) {
    this.someItem4 = true;
    this.amountItem4 = ammounts[3]
  }
} catch (err: any) {
  console.error(err.message);
}
```


3. Creación de tokens: ERC-1155

- A pesar de que el contrato impedirá que se superen los límites estipulados, la respuesta recibida sería únicamente de error, sin indicar cual de los límites estaría superando el usuario. Es por eso que se ha decidido implementar una detección en la propia interfaz para mostrar el aviso adecuado al usuario y evitar llamadas innecesarias.
- Se debe recordar que si se crea un lote, la atomicidad del estándar implica que o todas las transferencias dentro de un lote suceden con éxito, o ninguna lo hace.

3. Creación de tokens: ERC-1155

```
checkError(item: number) {  
  
  if (item == 1) {  
    if (this.amountItem1 == 1) {  
      this.err_msg = "You already get the maximum items of type 1.";  
    } else if (this.form_1155.value.type1 > 1) {  
      this.err_msg = "You can only mint 1 items of type 1 at a time.";  
    }  
  
    } else if (item == 2) {  
    if (this.amountItem2 == 100) {  
      this.err_msg = "You already get the maximum items of type 2.";  
    } else if (this.form_1155.value.type2 > 10) {  
      this.err_msg = "You can only mint 10 items of type 2 at a time.";  
    }  
  
    } else if (item == 3) {  
    if (this.amountItem3 == 1000) {  
      this.err_msg = "You already get the maximum items of type 3.";  
    } else if (this.form_1155.value.type3 > 100) {  
      this.err_msg = "You can only mint 100 items of type 3 at a time.";  
    }  
  
    } else if (item == 4) {  
    if (this.amountItem4 == 10000) {  
      this.err_msg = "You already get the maximum items of type 4.";  
    } else if (this.form_1155.value.type4 > 1000) {  
      this.err_msg = "You can only mint 1000 items of type 4 at a time.";  
    }  
  
  }  
}
```

3. Creación de tokens: ERC-1155



UVigo - NST

ITEM 1

OWNED: 0/1



MAX AMOUNT PER MINT: 1

MINT:

ITEM 2

OWNED: 0/100



MAX AMOUNT PER MINT: 10

MINT:

ITEM 3

OWNED: 0/1000



MAX AMOUNT PER MINT: 100

MINT:

ITEM 4

OWNED: 0/10000



MAX AMOUNT PER MINT: 1000

MINT:

YOU CAN ONLY MINT 100 ITEMS OF TYPE 3 AT A TIME.

Mint

3. Creación de tokens: ERC-1155

- Al crear los tokens, se llamará a la función del contrato para mintear tokens de un sólo tipo o para hacerlo de un lote entero según el caso.

```
async onSubmit() {  
  if (await this.loginCompleted()) {  
    const provider_var = new ethers.BrowserProvider((window as any).ethereum);  
  
    await provider_var.send("eth_requestAccounts", []);  
    const signer_var = await provider_var.getSigner();  
  
    const contract_mint_1155 = new ethers.Contract(this.CONTRACT_ADDRESS, this.CONTRACT_ABI, signer_var);
```

3. Creación de tokens: ERC-1155

```
var items: any[] = [];  
var ammount: any[] = [];  
  
if (this.form_1155.value.type1 != 0) {  
    items.push(1);  
    ammount.push(this.form_1155.value.type1);  
}  
if (this.form_1155.value.type2 != 0) {  
    items.push(2);  
    ammount.push(this.form_1155.value.type2);  
}  
if (this.form_1155.value.type3 != 0) {  
    items.push(3);  
    ammount.push(this.form_1155.value.type3);  
}  
if (this.form_1155.value.type4 != 0) {  
    items.push(4);  
    ammount.push(this.form_1155.value.type4);  
}
```

3. Creación de tokens: ERC-1155

```
if (items.length == 1) {  
  try {  
    await contract_mint_1155['mintToken'](items[0], ammount[0]);  
    this.err = false;  
  } catch (err: any) {  
    this.checkError(items[0])  
    this.err = true;  
  }  
} else if (items.length > 1) {  
  try {  
    await contract_mint_1155['mintBatch'](items, ammount);  
    this.err = false;  
  } catch (err: any) {  
    items.forEach(item => {  
      this.checkError(item)  
    })  
    this.err = true;  
  }  
}  
  
if (!this.err) {  
  this.ngZone.run(() => this.router.navigateByUrl('/add-NFT'))  
} else {  
  this.ngZone.run(() => this.router.navigateByUrl('/home'))  
}
```

3. Creación de tokens: ERC-1155

```
    if (items.length == 1) {
      try {
        await contract_mint_1155['mintToken'](items[0], ammount[0]);
        this.err = false;
      } catch (err: any) {
        this.checkError(items[0])
        this.err = true;
      }
    } else if (items.length > 1) {
      try {
        await contract_mint_1155['mintBatch'](items, ammount);
        this.err = false;
      } catch (err: any) {
        items.forEach(item => {
          this.checkError(item)
        })
        this.err = true;
      }
    }
  }
}

if (!this.err) {
  this.ngZone.run(() => this.router.navigateByUrl('/add-NFT'))
} else {
  this.ngZone.run(() => this.router.navigateByUrl('/home'))
}
```


3. Creación de tokens: ERC-1155



UVigo - NST

ITEM 1

OWNED: 0/1



MAX AMOUNT PER MINT: 1

MINT:

ITEM 2

OWNED: 0/100

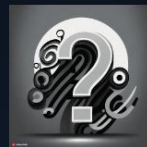


MAX AMOUNT PER MINT: 10

MINT:

ITEM 3

OWNED: 0/1000



MAX AMOUNT PER MINT: 100

MINT:

ITEM 4

OWNED: 0



MAX AMOUNT PER MINT: 1

MINT:

YOU CAN ONLY MINT 100 ITEMS OF TYPE 3 AT A TIME.

Mint

MetaMask Notification

Red de prueba Goerli

Account 1 → 0x7D6...2fB2

DETALLES DATOS HEX

⚠ La red está ocupada. Los precios del gas son altos y las estimaciones son menos precisas.

Mercado >

Gas (estimada) ⓘ 0.00060635

0.00060635 GoerliETH

Probablemente en < 30 segundos

Tarifa máxima: 0.00069029 GoerliETH

Total 0.00060635


0.00060635 GoerliETH

Cantidad + tarifa de gas

Cantidad máxima: 0.00069029 GoerliETH

Rechazar

Confirmar



4. Visualización de tokens: OpenSea API

4. Visualización de tokens

- Una forma común de visualizar un token es a través de OpenSea. Éste es un mercado en línea estadounidense de NFTs, que permite visualizar los tokens y sus metadatos y venderlos.
- Otra forma de visualizarlos es a través de Metamask, aunque para ello debemos introducir el ID del token y el Hash del contrato con el que se ha creado, importando cada token de forma individual.
- En éste ejemplo se ha usado la API de OpenSea, en la que se puede obtener una API Key para usar de manera gratuita.
- En el caso de los NFTs ERC-721, podemos verlos a través del botón Show NFTs del menú principal.
- En el caso de los tokens ERC-1155, al entrar en la sección para mintear con éste estándar, podremos ver un tipo de token si tenemos alguno en propiedad.

4. Visualización de tokens: ERC-721



- Haciendo click en la URL de los metadatos podremos acceder a los mismos

4. Visualización de tokens: ERC-721

```
const OPENSEA_API_URL = 'https://testnets-api.opensea.io/api/v2/chain/goerli/account/';

const accounts = await (window as any).ethereum.request({
  method: 'eth_requestAccounts',
});
const account = accounts[0];

console.log(account)

const options = {
  method: 'GET',
  url: OPENSEA_API_URL + account + '/nfts',
  headers: {accept: 'application/json'}
};
```

4. Visualización de tokens: ERC-721

```
axios.request(options).then( (response) => {  
  //this.nfts = response.data.nfts;  
  this.filteredNfts = response.data.nfts.filter((item: { contract: any; }) => item.contract === "0x4b01b68d50bc1370bd728de6c3cc9c8d5a29c64f");  
  this.nfts = this.filteredNfts.map((item: { metadata_url: any; }) => {  
    return {  
      ...item,  
      metadata_url: item.metadata_url.replace('ipfs://', 'https://gateway.pinata.cloud/ipfs/') // Modify the 'param' property here  
    };  
  });  
  console.log(this.nfts);  
}).catch(function (error) {  
  console.error(error);  
});
```

4. Visualización de tokens: ERC-1155



UVigo - NST

ITEM 1

OWNED: 1/1



MAX AMOUNT PER MINT: 1

MINT:

ITEM 2

OWNED: 8/100



MAX AMOUNT PER MINT: 10

MINT:

ITEM 3

OWNED: 10/1000



MAX AMOUNT PER MINT: 100

MINT:

ITEM 4

OWNED: 0/10000



MAX AMOUNT PER MINT: 1000

MINT:

Mint

4. Visualización de tokens: ERC-1155

- En este caso podrán consultarse los metadatos de cada tipo de token a través de Metamask, ya que permite directamente abrir un token en OpenSea, o a través del propio código del contrato en etherscan.

4. Visualización de tokens: ERC-1155

```
const options = {
  method: 'GET',
  url: this.OPENSEA_API_URL + signer_var.address + '/nfts',
  headers: {accept: 'application/json'}
};

axios.request(options).then( (response) => {

  this.filteredNfts = response.data.nfts.filter((item: { contract: any; }) => item.contract === "0x2e4f534bf1feb0e76b5380a8973d15f63319bd80");
  console.log(this.filteredNfts)

  this.filteredNfts.forEach(item => {
    if (item.identifier == 1) {
      this.imgItem1 = item.image_url;
    } else if (item.identifier == 2) {
      this.imgItem2 = item.image_url;
    } else if (item.identifier == 3) {
      this.imgItem3 = item.image_url;
    } else if (item.identifier == 4) {
      this.imgItem4 = item.image_url;
    }
  })

}).catch(function (error) {
  console.error(error);
});
}
```