Technical Project Presentation

Purpose: *This document aims to explain our NFT Project, its goals and implementation. It will explain the technical part. The public presentation for the sellable part of the project can be found on the document* Public Project Presentation

Table des matières

1 The Technical conception

Create NFT with hardhat

https://capbloc.notion.site/Create-NFT-with-hardhat-9a20a10237924d518597656cad8de7b0

To start our project, we first installed hardhat which is an Ethereum development environment for professionals and which will allow us to easily deploy our contracts.

ATTENTION: Everything must be placed in a specific folder

Before installing "hardhat" we have to install "yarn" and "npx", respectively a package manager and a tool to complete the package experience.

```
C:\Users\anais\Desktop\FG5\NFT>npm i --global yarn
```

Figure 1 - Install yarn

```
C:\Users\anais\Desktop\FG5\NFT>npm i --global npx
```

Figure 2 - Install npx

We created a new folder where we installed "hardhat".

```
yarn add --dev hardhat
```

Figure 3 - install "hardhat"

we can now launch the project with the commands :

```
npx hardhat

npm install --save-dev @nomicfoundation/hardhat-toolbox
@nomicfoundation/hardhat-network-helpers
```

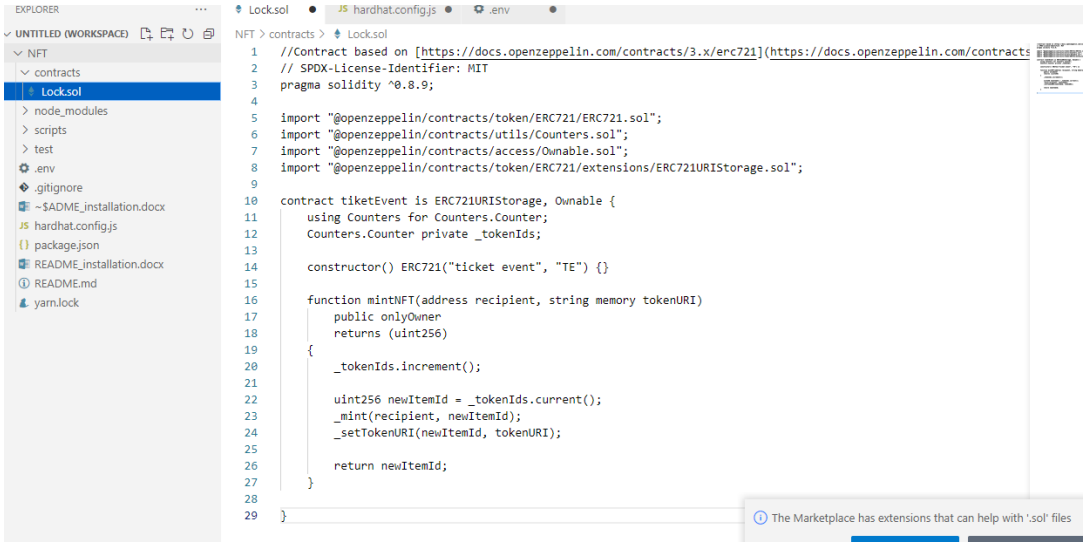
Figure 4 - Start project

We chose to create a javascript project

```
Done in 90.90s.

√ What do you want to do? · Create a JavaScript project
```

- After creating JavaScript project: we have changed the code of the file "Lock.sol", so that it corresponds to our project.



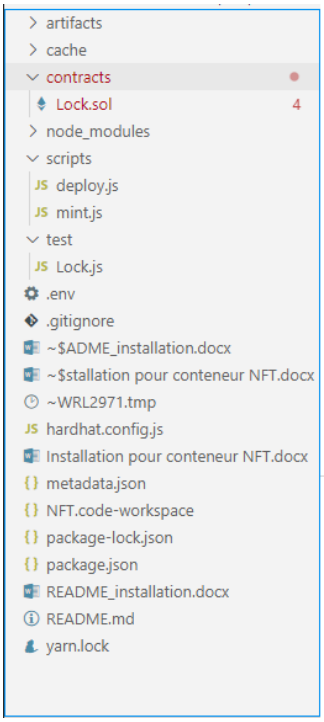


Figure 5 - New file Lock.sol / Smart Contract Code

We encountered some problems, here is how we were able to solve them:

In the Terminal :

```
yarn add dotenv

npx hardhat run scripts/deploy.ts --network Mumbai
```

- If it doesn't work: `>npm i dotenv`

`contract tiketEvent` : With **ticketEvent** as the project name

- metamask > import the private key in **hardhat.js**

`npx hardhat compile` : to check that everything is installed, we try to compile the code when finished

In case of any problem:

```
PS C:\Users\anais\Desktop\FG5\NFT> npx hardhat compile
Error HH801: Plugin @nomicfoundation/hardhat-toolbox requires the following dependencies to be installed: @ethersproject/providers, @nomicfou
ocha, @typechain/ethers-v5, @typechain/hardhat, chai, ethers, hardhat-gas-reporter, solidity-coverage, ts-node, typechain, typescript.
Please run: npm install --save-dev "@ethersproject/providers@^5.4.7" "@nomicfoundation/hardhat-network-helpers@^1.0.0" "@nomicfoundation/hard
hai-matchers@^1.0.0" "@nomiclabs/hardhat-ethers@^2.0.0" "@nomiclabs/hardhat-etherscan@^3.0.0" "@types/chai@^4.2.0" "@types/mocha@^9.1.0" "@ty
in/ethers-v5@^10.1.0" "@typechain/hardhat@^6.1.2" "chai@^4.2.0" "ethers@^5.4.7" "hardhat-gas-reporter@^1.0.8" "solidity-coverage@^0.8.1" "ts-
>=8.0.0" "typechain@^8.1.0" "typescript@>=4.5.0"
```

```
npm install --save-dev @nomicfoundation/hardhat-toolbox

@nomicfoundation/hardhat-network-helpers @nomicfoundation/hardhat-chai-matchers
@nomiclabs/hardhat-ethers @nomiclabs/hardhat-etherscan chai ethers

hardhat-gas-reporter solidity-coverage @typechain/hardhat typechain
@typechain/ethers-v5 @ethersproject/abi @ethersproject/providers
```

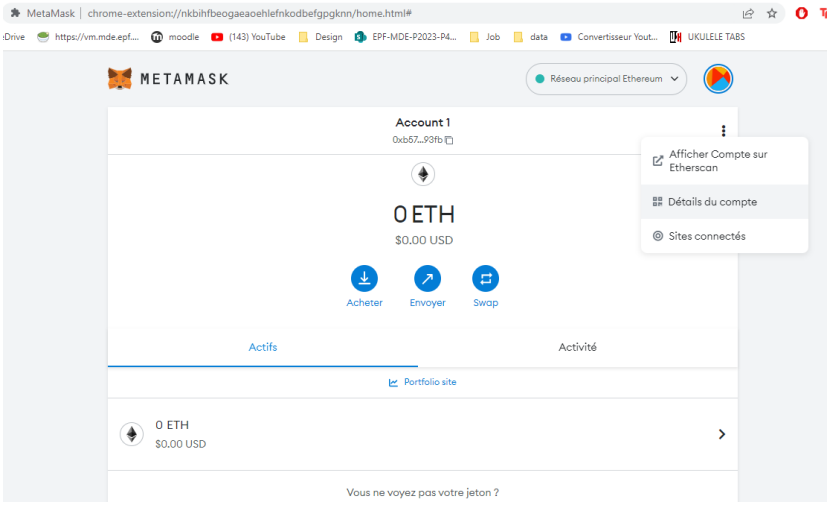
Download metamask chrome extension and setup Mumbai network

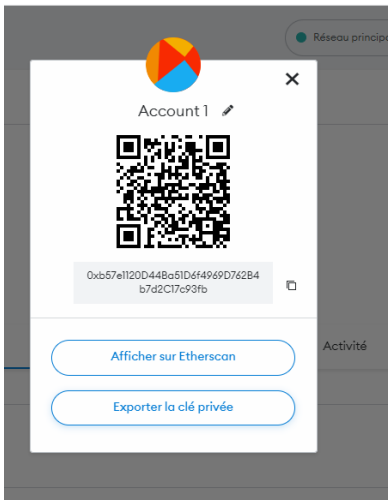
We had to install the "MetaMask" extension to our browser, this extension is equipped with a safe key, a secure connection, a token wallet and a token exchange system.

We will also need to add Mumbai to MetaMask, which allowed us to get Testnet tokens. These tokens have no value but allow us to test configurations and experiment with implementations.

we need to get our private key to add it to our code "**hardhat.js**"

- Generate the private key on our Wallet:





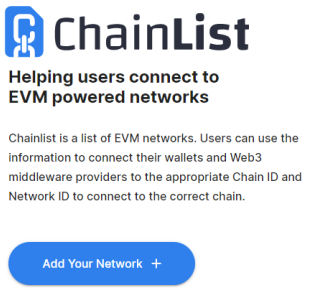
In “accounts”, we add our private key.

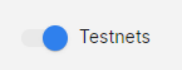
```
mumbai: {
  url: "https://polygon-mumbai.g.alchemy.com/v2/xPACxJYzW3ovyVZS_NMuOHO-BZaKcF2d",
  accounts:["4bf498dc523e5e07407ee36baa87611b997db7b867986d3fd3fa219a2227d5b6"],
```

Figure 6 - hardhat.config.js

We added the network Mumbai to MetaMask as follows :

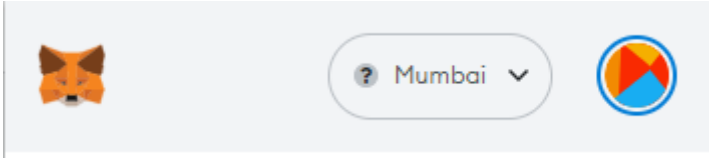
- In the websie : https://chainlist.org/



- We activate the testnets button 

and search network Mumbai to add it
- Add to metamask, approve

Mumbai is now added to our wallet :



And we can check that everything is well installed with :

```
npx hardhat compile
```

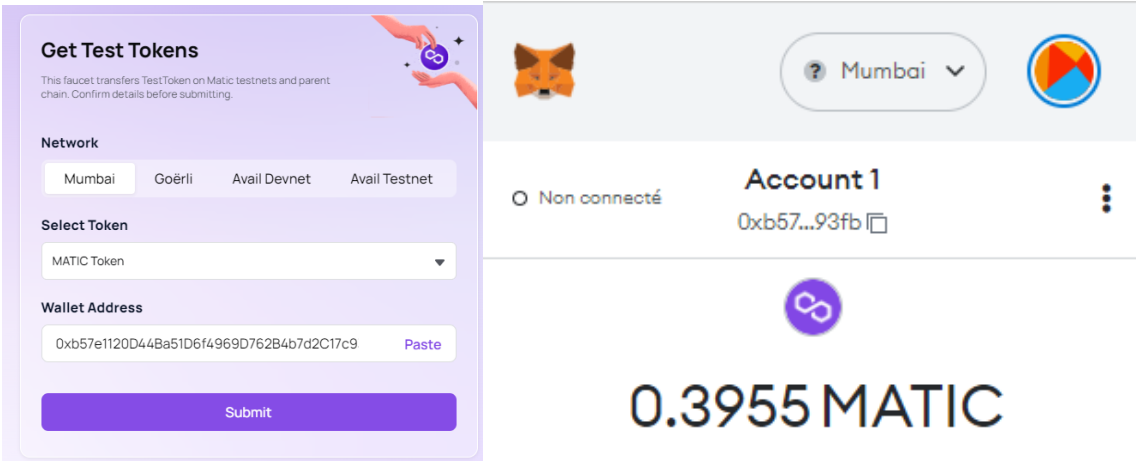
In case of problem:

```
npm install @openzeppelin/contracts
```

Ask Currency

A Polygon faucet is a developer tool to get testnet MATIC :

https://faucet.polygon.technology/



Create smart contract address

We deploy to Mumbai network with this command :

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

Which will allow us to recover the Smart Contract , here the Smart Contract is 0xD27FAe92b79159E349B571E6699b08e58B5Fee8c.

The smart contract can change when we deploy it.

Create NFT linked to contract address

```
npx hardhat run scripts/mint.js --network
```

Deployed to smart contract 0xD27FAe92b79159E349B571E6699b08e58B5Fee8c

See historic of transactions

Write at the end your smart contract address

With this link, we can see the historic of transaction.

https://mumbai.polygonscan.com/address/0x9d1868a7cC547C14384194549AD29e6f7a3a5399

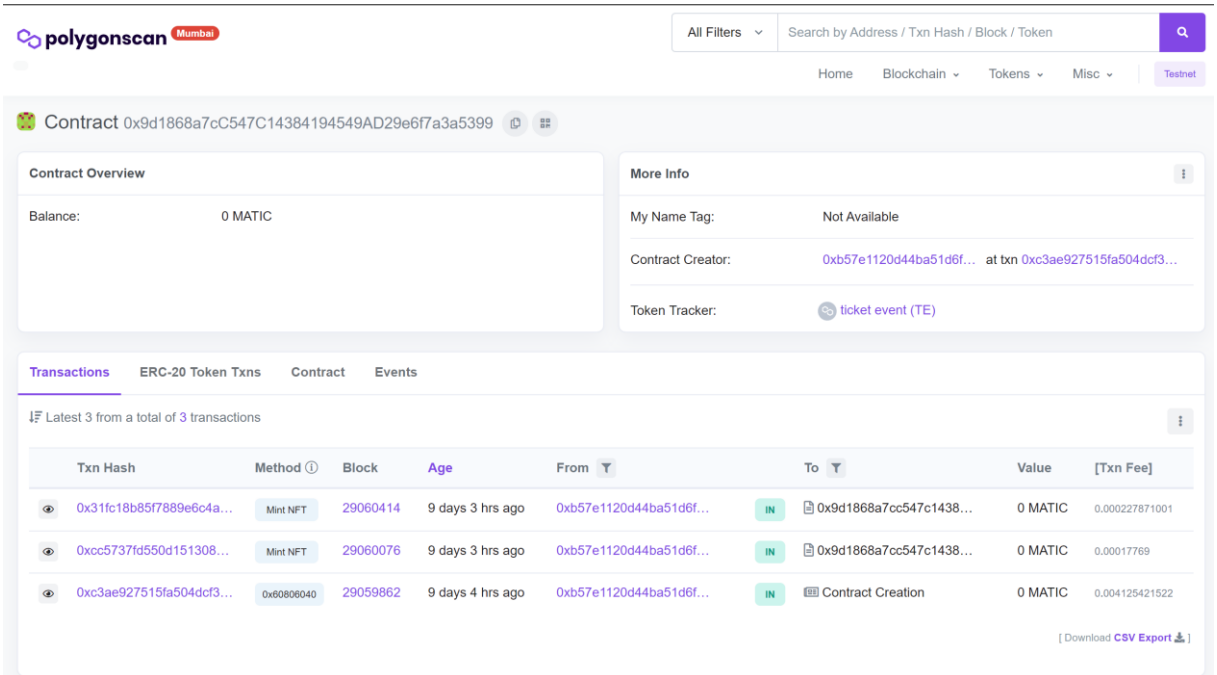


Figure 7 - Transaction we made

Upload our NFT with a Pinata account

Pinata allows to post NFT, upload them as a file.

https://app.pinata.cloud/

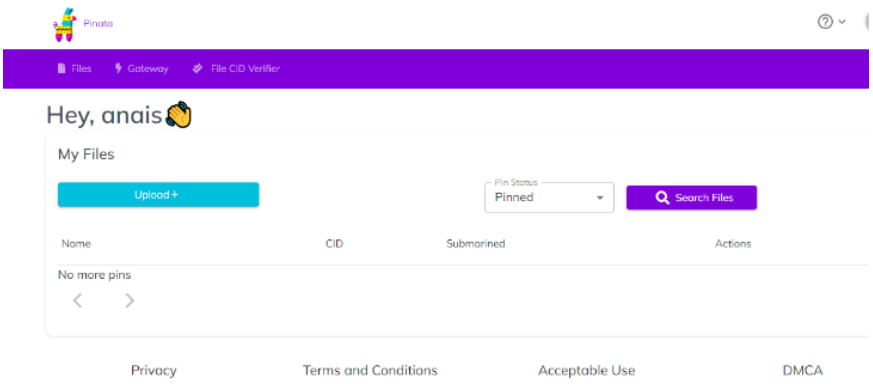
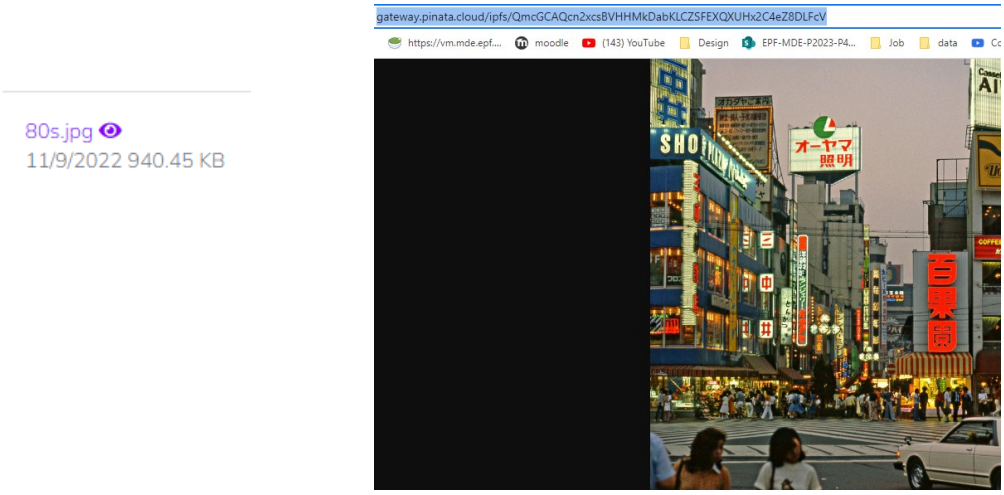


Figure 8 - Pinata

- For have the url of the NFT, we click on the eye



- We copy/paste the url of the NFT on metada.json file

```
{} metadata.json > ...
1  {
2    "attributes": [
3      {
4        "biome1": "Breed",
5        "biome2": "Maltipoo",
6        "NFT_type": "photography"
7      },
8      {
9        "biome1": "Eye color",
10       "biome2": "Mocha"
11     }
12   ],
13   "description": "The world's most adorable and sensitive pup.",
14   "image": "https://gateway.pinata.cloud/ipfs/QmcGCAQcn2xcsBVHHMkDabKLCZSFEXQXUHx2C4eZ8DLFcV",
15   "name": "Paul watson"
16 }
```

- We upload the metadata on pinata:

We now can upload metadata.json file on pinata

metadata.json 11/9/2022 429 B	QmRg5ZHDQSg7hM3t3CAzBQvQiNqi9SfzSeeMBKjdfpE4LK
80s.jpg 11/9/2022 940.45 KB	QmcGCAQcn2xcsBVHHMkDabKLCZSFEXQXUHx2C4eZ8DLFcV

- We copy the Cid of the metadata.json uploaded in pinata and we copy it in mint.js in

```
await tiketEvent.mintNFT()
```

```
scripts > JS mint.js > main
14   address smart contract
15  nst tiketEvent = await TiketEvent.attach("0x9d1868a7cC547C14384194549AD29e6f7a3a5399");
16
17
18  wait tiketEvent.mintNFT("Metamask public key", "Token uri = metadata URL"
19  ait tiketEvent.mintNFT("0xb57e1120D44Ba51D6f4969D762B4b7d2C17c93fb", "https://gateway.pinata.cloud/ipfs/QmRg5ZHDQSg7hM3t3
20
```

The metadata.json is now liked to the NFT url.

See our NFT

To be able to view our NFT, we must:

- Go to the site: https://testnets.opensea.io/fr
- Enter our smart contract address
- Put the number of times mintNFT was called : here 2 times

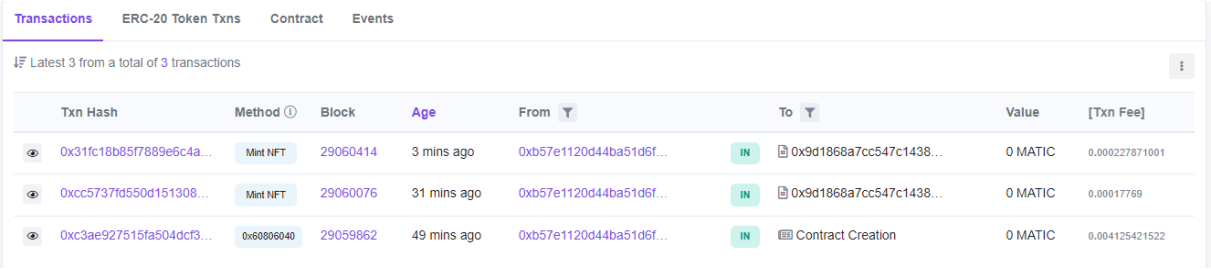
https://testnets.opensea.io/assets/network(mumbai)/*smart contract adress/ number times mintNFT*

https://testnets.opensea.io/assets/mumbai/0x9d1868a7cC547C14384194549AD29e6f7a3a5399/2

or
https://testnets.opensea.io/assets/mumbai/0xD27FAe92b79159E349B571E6699b08e58B5Fee8c/1

In https://mumbai.polygonscan.com/

- Enter the smart contract address
- We can see that mintNFT was used 2 times.



Transactions | ERC-20 Token Txns | Contract | Events

Latest 3 from a total of 3 transactions

Txn Hash	Method	Block	Age	From		To	Value	[Txn Fee]
0x31fc18b85f7889e6c4a...	Mint NFT	29060414	3 mins ago	0xb57e1120d44ba51d6f...	IN	0x9d1868a7cc547c1438...	0 MATIC	0.000227871001
0xcc5737fd550d151308...	Mint NFT	29060076	31 mins ago	0xb57e1120d44ba51d6f...	IN	0x9d1868a7cc547c1438...	0 MATIC	0.00017769
0xc3ae927515fa504dcf3...	0x60806040	29059862	49 mins ago	0xb57e1120d44ba51d6f...	IN	Contract Creation	0 MATIC	0.004125421522

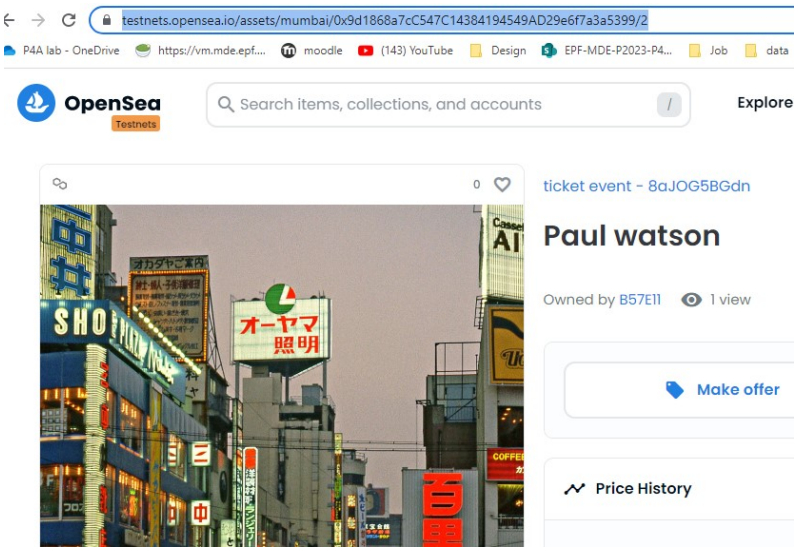


Figure 9 - View of our NFT

Create transactions

- **In Lock.sol, change public onlyOwner into external:**

```
contract tiketEvent is ERC721URIStorage, Ownable {
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;

    constructor() ERC721("ticket event", "TE") {}

    function mintNFT(address recipient, string memory tokenURI)
        external // modified
        returns (uint256)
    {
```

Transaction code

We write the following code outside of function mintNFT, the function payment does as follow:

```
    receive() external payable{
        uint256 QuantitySent = msg.value;
        (bool sent, bytes memory data) = address(0x5F8F81C73C34269aBF936333Fc2C9872aba18bCD).call{value: QuantitySent}("");
        require(sent, "Failed to send Ether");

        if (QuantitySent > 0.01 ether) {
            mintNFT(msg.sender , tokensURI[2]);
        } else if (QuantitySent > 0.005 ether) {
            mintNFT(msg.sender , tokensURI[1]);
        } else {
            mintNFT(msg.sender , tokensURI[0]);
        }

    }
```

When money is sent, an NFT is attributed.

To prevent the NFT from being resold, we modify the smart-contract behaviour and modify the "transfer" function :

```
function _beforeTokenTransfer(address from, address to, uint256) pure override internal {
    require(from == address(0) || to == address(0), "This a Soulbound token. It cannot be transferred. It can only be burned by the
}
```

- Detect that money was sent to smart contract

receive() external payable{} : defined to know the quantity and money sent
msg.value : quantity of money sent, all the data of the transaction. 1matic = 10^9 wei.

- We want to make different levels of NFT in terms of money sent.
- If the condition is respected
  - An NFT is then created
  - The recipient is msg.sender

To run : `npx hardhat run scripts/deploy.js --network`

Test

```
PS C:\Users\anais\Desktop\FG5\NFT> npx hardhat run scripts/deploy.js --network mumbai
Compiled 1 Solidity file successfully
 deployed to 0xCf63EfeF102C931D1510Ff6E422bB5cbc6B39C9c
```

The code is the contract, if someone wants to buy the NFT, they must send money to this contract number

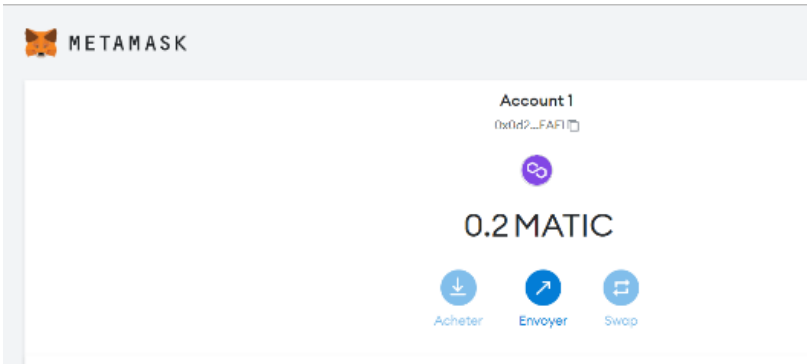
- We create as many different metadata.json files as there are created NFT


```
{
  "attributes": [
   {
    "biome1": "Breed",
    "biome2": "Maltipoo",
    "NFT_type": "photography"
   },
   {
    "biome1": "Eye color",
    "biome2": "Mocha"
   }
  ],
  "description": "The world's most adorable and sensitive pup.",
  "image":
"https://gateway.pinata.cloud/ipfs/QmcGCAQcn2xcsBVHHMkDabKLCZSFEXQXUHx2C4eZ8DLFcV",
  "name": "Paul watson"
 }
```

2 How to buy an NFT

Create a Metamask account

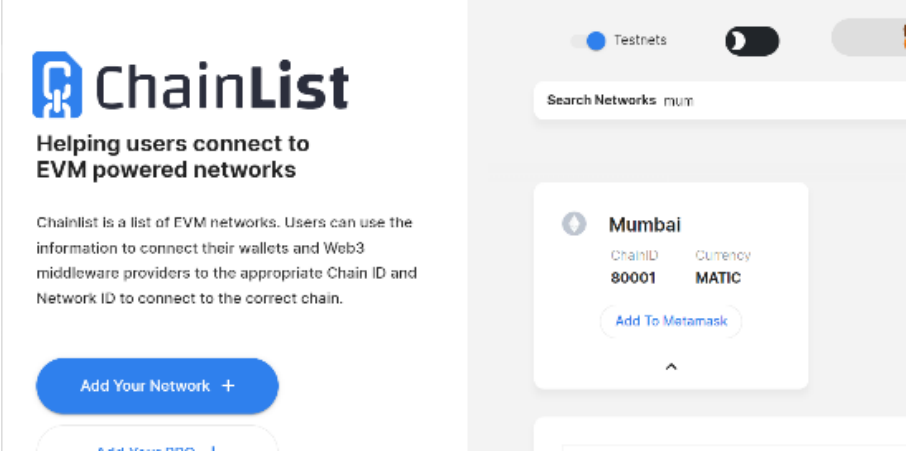
The crypto wallet for Defi, Web3 Dapps and NFTs | MetaMask



On Chainlist

Chainlist

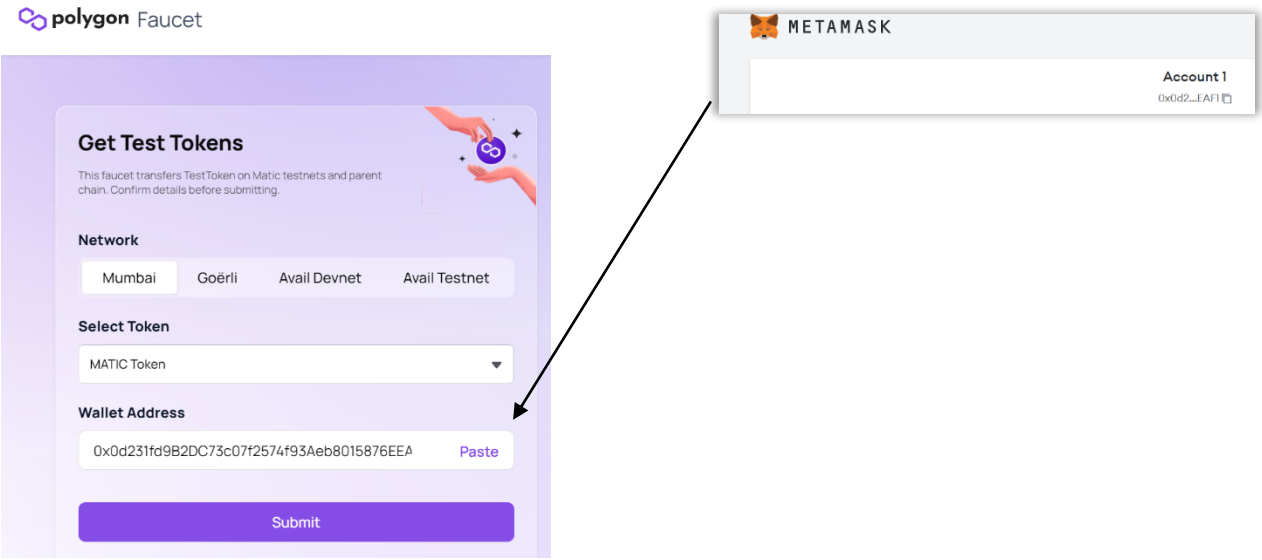
- Search for Mumbai
- Add to Metamask



On Polygon Faucet

Polygon Faucet

The wallet address can be found here :



In Metamask

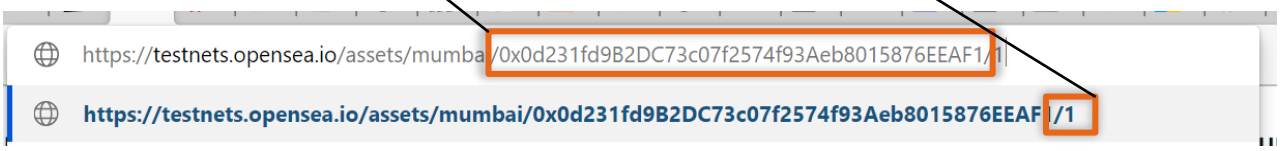
Paste the address and enter the number of MATIC you want.

In Opensea

To see our new minted NFT, we follow the link below:

https://testnets.opensea.io/assets/mumbai/*YOUR-SMART-CONTRACT-ADDRESS/YOUR-TOKEN-ID*

In our search bar, we replace the Mumbai's address by our wallet address (on metamask) and we add "/1"



On the profile, we click to connect to metamask