**Foundry setup**

**Foundry Setup**

Now that we know what an NFT is, let's investigate how we can build our own. As always the code we're writing will be available in the [**GitHub Repo**](https://github.com/Cyfrin/foundry-nft-cu) associate with this section.

To start, let's initialize our workspace. Create a new directory in your course folder.

mkdir foundry-nft

We can then switch to this directory and open it in VSCode.

cd foundry-nft

code .

Finally, we can initialize our foundry project.

forge init

Once initialized, be sure to remove the example contracts src/Counter.sol, test/Counter.t.sol and script/Counter.s.sol. Finally, assure that your .gitignore contains .env and broadcast/

**NFT Contracts**

Now, as mentioned previously, NFTs are just another type of [**Token Standard**](https://eips.ethereum.org/EIPS/eip-721), similar to ERC20s. As such, we could simply write our contract and begin implementing all the necessary methods to comply with this standard. However, like ERC20s, we can also just import a library (like OpenZeppelin) which does all this heavy lifting for us.

Begin by creating src/BasicNft.sol and setting up our usual boilerplate.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.18;

contract BasicNft{}

We can install the OpenZeppelin Contracts library with:

forge install OpenZeppelin/openzeppelin-contracts --no-commit

To make things a little easier on ourselves, we can add this as a remapping to our foundry.toml. This remapping allows us to use some short-hand when importing from this directory.

[profile.default]

src = "src"

out = "out"

libs = ["lib"]

remappings = ["@openzeppelin/contracts=lib/openzeppelin-contracts/contracts"]

Now we can import and inherit the ERC721 contract into BasicNft.sol

// SPDX-License-Identifier: MIT

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";

pragma solidity ^0.8.18;

contract BasicNft is ERC721 {}

Your IDE will likely indicate an error until we've passed the necessary arguments to the ERC721 constructor. You can ctrl + left-click (cmd + left-click) on the imported ERC721.sol to navigate to this contract and confirm what the constructor requires.

constructor(string memory name\_, string memory symbol\_) {

\_name = name\_;

\_symbol = symbol\_;

}

Just like the ERC20, we need to give our token a name and a symbol, that makes sense. Feel free to choose your own, but I'm going to go with the name Doggie and the symbol DOG.

// SPDX-License-Identifier: MIT

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";

pragma solidity ^0.8.18;

contract BasicNft is ERC721 {

constructor() ERC721("Doggie", "DOG"){}

}

Great! While this contract may have the basic functionality of an NFT protocol, there's a lot to be done yet. Because each token is unique and possesses a unique tokenId, we absolutely need a token counter to track this in storage. We'll increment this each time a token is minted.

uint256 private s\_tokenCounter;

constructor() ERC721("Doggie", "DOG"){

s\_tokenCounter = 0;

}

Additionally, we want to add a function to mint NFTs. We will define it later:

function mintNft() public {}

**TokenURI**

It's hard to believe, but once upon a time the tokenUri was once considered an optional parameter, despite being integral to how NFTs are used and consumed today.

TokenURI stands for Token Uniform Resource Identifier. At its core it serves as an endpoint that returns the metadata for a given NFT.

**Example TokenURI Metadata Schema:**

{

"title": "Asset Metadata",

"type": "object",

"properties": {

"name": {

"type": "string",

"description": "Identifies the asset to which this NFT represents"

},

"description": {

"type": "string",

"description": "Describes the asset to which this NFT represents"

},

"image": {

"type": "string",

"description": "A URI pointing to a resource with mime type image/\* representing the asset to which this NFT represents. Consider making any images at a width between 320 and 1080 pixels and aspect ratio between 1.91:1 and 4:5 inclusive."

}

}

}

It's this metadata that defines what the properties of the NFT are, including what it looks like! In fact, if you go to [**OpenSea**](https://opensea.io/) and look at any NFT there, all the data and images you're being served come from calls to the tokenURI function.

What this means to us is - any time someone mints a Doggie NFT, we need to assign a TokenURI to the minted TokenID which contains all the important information about the Doggie. Let's consider what this function would look like.

❗ **NOTE** The OpenZeppelin implementation of ERC721, which we've imported, has it's own virtual tokenURI function which we'll be overriding.

By navigating to any NFT on OpenSea, you can find a link to the collection's contract on Etherscan. Click on Read Contract and find the tokenURI function (here's a link to [**Pudgy Penguins**](https://etherscan.io/address/0xbd3531da5cf5857e7cfaa92426877b022e612cf8#readContract) if you need it).

Entering any valid tokenId should return the TokenURI of that NFT!

By opening this URI in your browser, the details of that token's metadata should be made available:

Note the imageURI property. This is what defines what the NFT actually looks like, you can copy this into your browser as well to view the NFT's image.

Both the tokenUri and imageUri for this example are hosted on IPFS (Inter-planetary file system), a service offering decentralized storage that we'll go into in greater detail, in the next lesson.

So what's this tokenURI function going to look like for us? Well, our BasicNFT is going to also use IPFS, so similarly to our example above, we'll need to set up our function to return this string, pointing to the correct location in IPFS.

function tokenURI(uint256 tokenId) public view override returns (string memory) {}

Now, I've prepared some images you can choose from to use in your project, but feel free to use your own. Making these projects *yours* goes a long way towards committing these things to memory. You can find the images I've provided in the [**GitHub Repo**](https://github.com/Cyfrin/foundry-nft-f23/tree/main/images/dogNft) for this section.

Create a new folder in your workspace names img (image) and add the image of your choice to this directory.

In the next lesson, we'll learn how to host our NFT images in a decentralized manner by leveraging IPFS.

I'll see you there!