# **NFT Token Development – The Ultimate Guide**

NFTs, or non-fungible tokens, have generated a lot of buzz in the past year. They connected the crypto, blockchain, art, and gaming industries into one expansive ecosystem. Furthermore, with NFTs surging in popularity since early 2021, the market has continuously shown strong interest in NFTs. Because of NFT’s ongoing success, programmers have been keen to learn NFT token development. Moreover, if you aspire to become an NFT developer, now is an excellent time to start. As such, this guide will advance your programming journey and educate you regarding NFT token development.

There’s no doubt that anyone interested in being part of the blockchain and crypto space should learn about NFTs. Whether you’re into programming or looking to spread your wings in the art world, understanding how to [create your own NFT](https://moralis.io/how-to-create-your-own-nft-in-5-steps/?utm_source=blog&utm_medium=post&utm_campaign=NFT%2520Token%2520Development%2520%25E2%2580%2593%2520The%2520Ultimate%2520Guide) will bring a huge advantage to your career.

You can learn simple NFT token development by using powerful [Web3](https://moralis.io/the-ultimate-guide-to-web3-what-is-web3/?utm_source=blog&utm_medium=post&utm_campaign=Want%2520the%2520Latest%2520in%2520%253Cspan%253EBlockchain%2520Development%253F%253C%252Fspan%253E) building tools such as Moralis to speed up your NFT creation. Moralis is the ultimate Web3 operating system, offering everything to host, manage and scale NFT dApps (decentralized applications). With [Moralis Speedy Nodes](https://moralis.io/speedy-nodes/?utm_source=blog&utm_medium=post&utm_campaign=NFT%2520Token%2520Development%2520%25E2%2580%2593%2520The%2520Ultimate%2520Guide), you can take advantage of fast, reliable, and world-class nodes without needing to set up and sync your own nodes, such as an [Ethereum node](https://moralis.io/how-to-connect-to-ethereum-nodes/?utm_source=blog&utm_medium=post&utm_campaign=NFT%2520Token%2520Development%2520%25E2%2580%2593%2520The%2520Ultimate%2520Guide). In addition, because it’s cross-chain, you can also set up Binance Smart Chain (BSC), Polygon, [Avalanche](https://moralis.io/moralis-announces-avalanche-support/?utm_source=blog&utm_medium=post&utm_campaign=Want%2520the%2520Latest%2520in%2520%253Cspan%253EBlockchain%2520Development%253F%253C%252Fspan%253E), or [Arbitrum nodes](https://moralis.io/announcing-moralis-speedy-nodes-support-of-arbitrum-nodes/?utm_source=blog&utm_medium=post&utm_campaign=NFT%2520Token%2520Development%2520%25E2%2580%2593%2520The%2520Ultimate%2520Guide) with Moralis. In fact, Moralis acts as an [Infura for Avalanche](https://moralis.io/infura-for-avalanche-exploring-infura-avalanche-alternatives/?utm_source=blog&utm_medium=post&utm_campaign=NFT%2520Token%2520Development%2520%25E2%2580%2593%2520The%2520Ultimate%2520Guide) and [Infura for BSC](https://moralis.io/infura-bsc-alternatives-the-best-alternative-to-infura-for-bsc/?utm_source=blog&utm_medium=post&utm_campaign=NFT%2520Token%2520Development%2520%25E2%2580%2593%2520The%2520Ultimate%2520Guide).

As you go further into your NFT dApp or platform development, you should also check out useful tools such as Moralis’ [ultimate NFT API](https://moralis.io/ultimate-nft-api-exploring-moralis-nft-api/?utm_source=blog&utm_medium=post&utm_campaign=NFT%2520Token%2520Development%2520%25E2%2580%2593%2520The%2520Ultimate%2520Guide), which gives you access to NFT data, history, ownership, and much more. This means you can create an NFT dApp, platform, or marketplace even without deep EVM ([Ethereum Virtual Machine](https://moralis.io/evm-explained-what-is-ethereum-virtual-machine/?utm_source=blog&utm_medium=post&utm_campaign=NFT%2520Token%2520Development%2520%25E2%2580%2593%2520The%2520Ultimate%2520Guide)) knowledge.

## **Learn NFT Token Development – Informative Video**

By following this guide, you’ll learn everything needed to perform NFT token development in a few simple steps. However, you can also go to [Moralis’ Youtube channel](https://www.youtube.com/channel/UCgWS9Q3P5AxCWyQLT2kQhBw) to understand the process in a concise video tutorial:

## **Quick NFT Token Development – Build a Minting dApp**

So, [what are NFTs](https://moralis.io/non-fungible-tokens-explained-what-are-nfts/?utm_source=blog&utm_medium=post&utm_campaign=NFT%2520Token%2520Development%2520%25E2%2580%2593%2520The%2520Ultimate%2520Guide), and what makes them so exciting? The acronym NFT stands for “Non-Fungible Token”, and it means that this type of token, unlike ERC-20 tokens or others used as a medium of exchange, cannot be interchanged as an equivalent of another token.

NFTs are tokens that represent unique values, goods, or objects. In effect, they are perfect for representing digital works of art, creative projects, music, or anything that has unique properties. The two popular Ethereum standards used in NFT token development are ERC-721 and ERC-1155.

### **NFT Minting dApp**

One of the best shortcuts to creating your own NFTs is to build a minting dApp that does the minting work for you. With an NFT dApp, you can mint as many NFTs as you like.

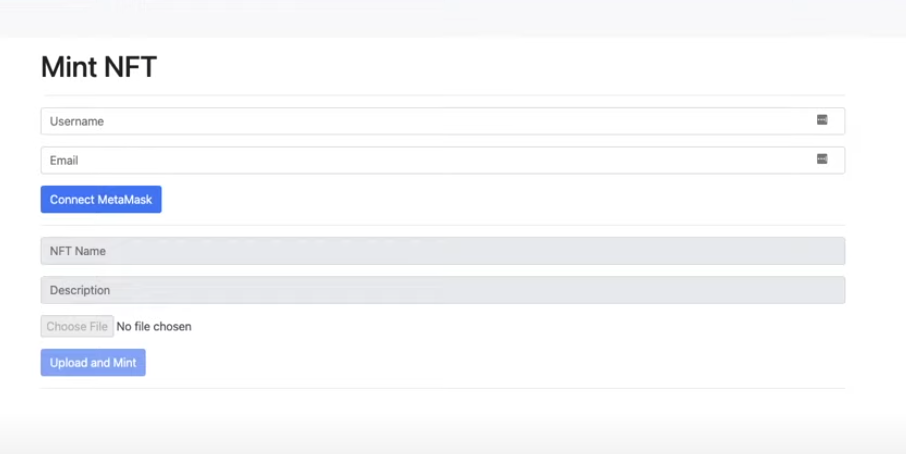
The good news is that building an NFT dApp doesn’t require heavy programming skills. With Moralis, you can reduce the process to five easy steps:

1. Initialize Moralis and find a smart contract that suits your project.
2. Create an HTML index file.
3. Make a simple login function for your dApp.
4. Create an upload function so you can upload your NFTs.
5. Build a mint function that will let you create NFTs.

## **NFT Token Development – dApp Features**

You’re aiming to create a minting dApp, and your dApp should have all the basic features required to mint an NFT independently.

The finished product should look like this:



As you can see, the dApp has fields to enter your username and email, as well as a link to connect you to MetaMask. Why is this needed? It’s because the app needs a function to [authenticate with MetaMask](https://moralis.io/how-to-authenticate-with-metamask/?utm_source=blog&utm_medium=post&utm_campaign=NFT%2520Token%2520Development%2520%25E2%2580%2593%2520The%2520Ultimate%2520Guide) so users can verify themselves on the blockchain. Take note that the storage of user data and wallet addresses is decentralized. MetaMask is crucial to accessing this data and interacting with Ethereum’s network. Moreover, remember that minting is a type of transaction. Therefore, you need to be connected to an active Ethereum address to perform it.

## **Get Started on NFT Token Development**

### **Sign Up with Moralis**

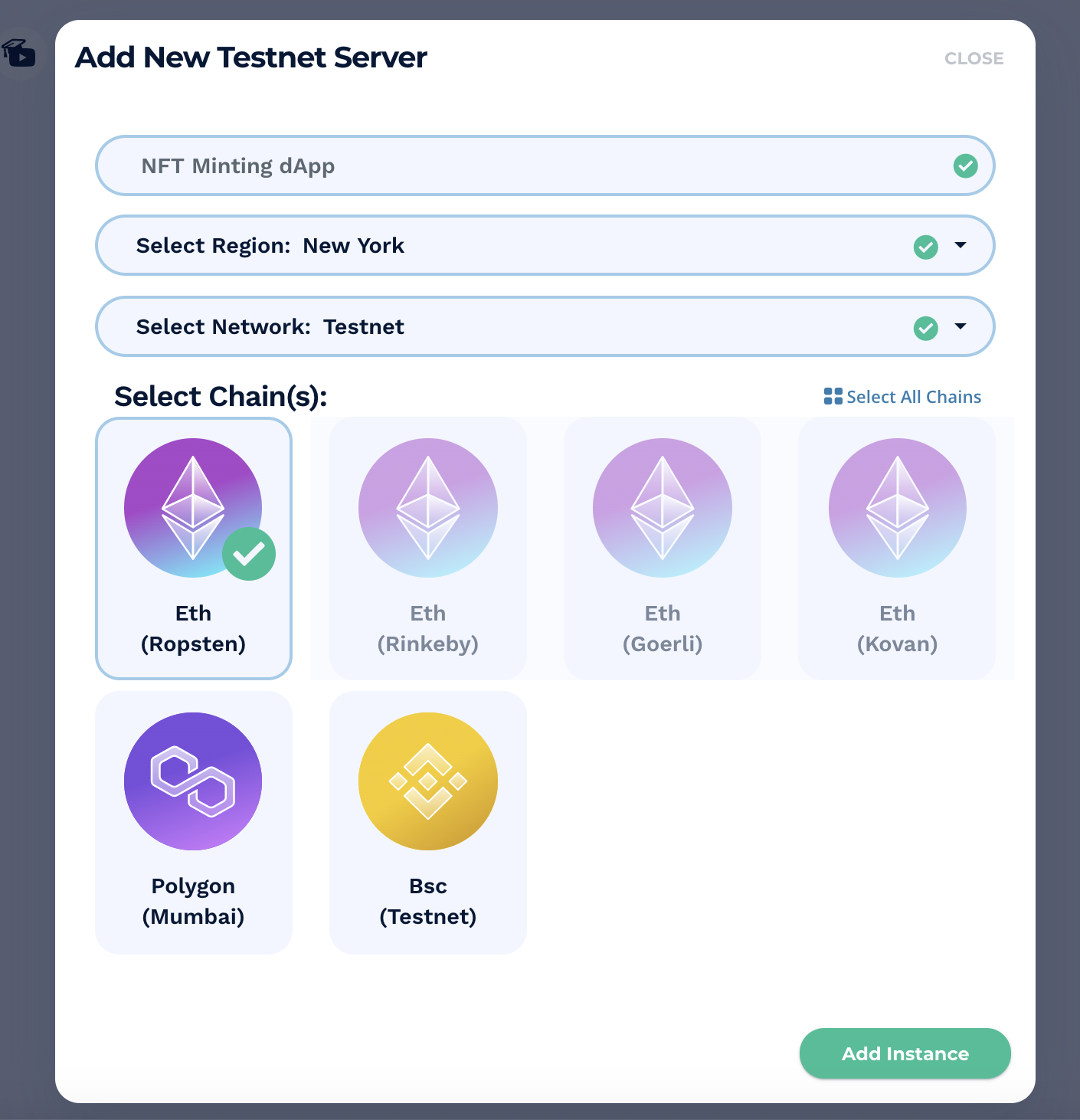
If you’re not registered yet, you need to [sign up for a new Moralis account](https://admin.moralis.io/register?utm_source=blog&utm_medium=post&utm_campaign=NFT%2520Token%2520Development%2520%25E2%2580%2593%2520The%2520Ultimate%2520Guide). Don’t worry; it’s simple, free, and will only take a minute.

#### **Create a New Server**

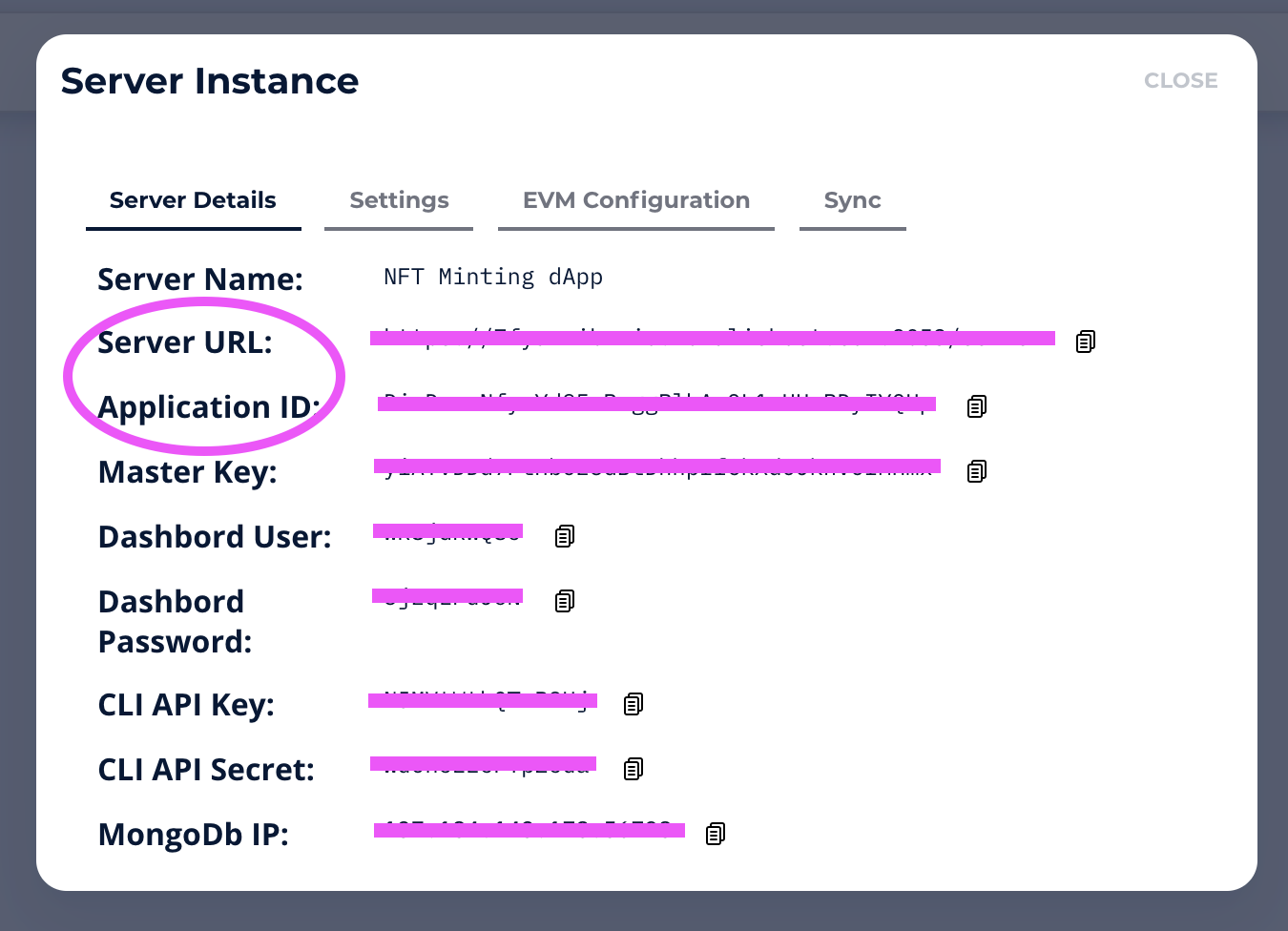
After signing up, you’ll find the “+ Create a New Server” button on the upper right side of your interface. Click the button and select the testnet as your server option.



After you’ve selected the “Testnet Server” option from the drop-down menu, a pop-up will appear where you need to fill in some details. Select a name for your project, and in our case, we’re naming it “NFT Minting dApp”. Select your preferred region and network. Under “Select Chain(s)”, make sure to use “Eth (Ropsten)” since we’re using the Ropsten testnet for this project. Then click the “Add Instance” button on the bottom right to complete this step.



After clicking the “Add Instance” button, your new server should appear pretty quickly on your screen. Click the “View Details” button on the server you just created, and take note of the details as you’ll need these to proceed to your next step.



Take note of the “Application ID” and the “Server URL” information because you’ll be copying and pasting them into the first lines of your code later.

### **NFT Token Development – Minting dApp Prerequisites**

This phase requires at least Python 3.0; however, if you don’t have it installed yet, you can look up several resources online that will give you installation guides that suit your platform.

In addition, we also assume that you are proficient in HTML, JavaScript (JS), Solidity, and Python in order to continue. If you need a refresher on [JavaScript](https://moralis.io/javascript-explained-what-is-javascript/?utm_source=blog&utm_medium=post&utm_campaign=NFT%2520Token%2520Development%2520%25E2%2580%2593%2520The%2520Ultimate%2520Guide) or need to understand “[what is Solidity](https://moralis.io/solidity-explained-what-is-solidity/?utm_source=blog&utm_medium=post&utm_campaign=NFT%2520Token%2520Development%2520%25E2%2580%2593%2520The%2520Ultimate%2520Guide)?”, look up Ivan on Tech Academy for excellent courses on topics such as [JavaScript programming for blockchain developers](https://academy.ivanontech.com/courses/javascript-programming-for-blockchain-developers) and [Ethereum smart contract programming 101](https://academy.ivanontech.com/courses/ethereum-smart-contract-programming-101) to bring you up to speed on blockchain development.

Another thing you need to remember is that we are using a smart contract that is already prepared beforehand. Developers have the option to write their smart contracts or simply copy a pre-built contract that’s confirmed to be a reliable and tested one from GitHub. In this case, we’re trying to make things as simple as possible. Therefore, the best option is to use a smart contract that’s already been written and reviewed.

### **Building the NFT Minting dApp**

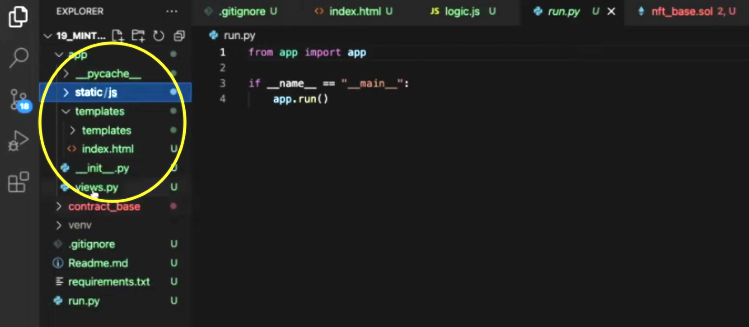
The dApp is a simple flask app, and in a flask app, you have a “run.py” as shown here:

from app import app

if \_\_name\_\_ == "\_\_main\_\_":

app.run()

This “run.py” will trigger everything in the directory, located on the left side of the screen:



## **NFT Token Development in Five Steps**

Now, let’s break down the necessary code and see how it enables all the steps to create your own NFT and proceed with NFT token development.

1. Initialize Moralis

To initialize Moralis, you only need to input a few lines of code, plus the details (application ID and server URL) from your newly created Moralis server.

Moralis.initialize(""); // Application ID from moralis.io

Moralis.serverURL = ""; // Server URL from moralis.io

After initializing Moralis, you need to get the smart contract. Hence, you need to input:

const nft\_contract\_address = "" // NFT minting contract, use this one "Batteries Included", code of this contract is in the GitHub repository under contract\_base for your reference.

/\*

Available deployed contracts

Ethereum Rinkeby 0x0Fb6EF3505b9c52Ed39595433a21aF9B5FCc4431

Polygon Mumbai 0x351bbee7C6E9268A1BF741B098448477E08A0a53

BSC Testnet 0x88624DD1c725C6A95E223170fa99ddB22E1C6DDD

\*/

After looking up the full details of the smart contract in the [GitHub repository](https://github.com/DanielMoralisSamples/19_MINTNFT), the next step is to proceed with the HTML index file.

1. Create an HTML Index File

We need an “index.html” file with all the elements necessary for the dApp. This means that it contains all the interaction buttons used for the following:

* MetaMask login.
* Fields for NFT minter: NFT name, description, and file element.

#### **Get the HTML Index File**

You can fin the complete HTML index file in the [HTML section](https://github.com/DanielMoralisSamples/19_MINTNFT/blob/master/app/templates/index.html) of the GitHub repository.

This excerpt points out some important interactions in the code:

<div class="input-group mb-3">

<input type="file" disabled = "true" id="file">

</div>

</div>

<div>

<button disabled = "true" class="btn btn-primary" id="upload" onclick="upload();">Upload and Mint</button>

</div>

<hr>

<div class="input-group mb-3" id="resultSpace">

The file element is an important field. Using this file element, you can upload any file and mint it using the “Upload and Mint” button that triggers the dApp’s logic. As soon as the logic runs, the dApp displays the results of the transaction that triggered the NFT minting in a designated result space.

1. Create the Login Function

Now, go back to the JavaScript file. After initializing Moralis and initiating the smart contract, you can now proceed by creating a function that’s triggers when clicking the “Connect MetaMask” button. This is the code that enables the function:

async function login(){

document.getElementById('submit').setAttribute("disabled", null);

document.getElementById('username').setAttribute("disabled", null);

document.getElementById('useremail').setAttribute("disabled", null);

Moralis.Web3.authenticate().then(function (user) {

user.set("name",document.getElementById('username').value);

user.set("email",document.getElementById('useremail').value);

user.save();

document.getElementById("upload").removeAttribute("disabled");

document.getElementById("file").removeAttribute("disabled");

document.getElementById("name").removeAttribute("disabled");

document.getElementById("description").removeAttribute("disabled");

})

}

Before clicking the button created here, users have already entered the information in the fields requiring their username and email. The code in between simply deactivates the elements so they can’t change after calling the function. Users will not be able to change their username or email data and won’t be able to press the button again once the process has started.

After disabling the buttons and fields, the code triggers an authentication function. As previously mentioned, we’re using MetaMask to connect to the blockchain and verify users. Moralis enables MetaMask authentication by default. After user authentication, the function fetches the user data to add to a new user profile on Moralis’ database. Afterward, the code removes the “disabled” attribute to resume user interaction and the creation of a new NFT.

1. Add the Upload Function

As a next step, you need to create an upload function. When you click the “Upload and Mint” button, this is the function that we calle. In the first part, the function collects the “.getElementByID” of the file component in the HTML file. Furthermore, the component captures files in an array. This is the reason you need to point to the element zero “[0]”, which signifies the first element in the array. We only want to get the first file that is captured by that interaction. The next line of code creates a new Moralis file object. This object is needed to execute the rest of the function.

async function upload(){

const fileInput = document.getElementById("file");

const data = fileInput.files[0];

const imageFile = new Moralis.File(data.name, data);

document.getElementById('upload').setAttribute("disabled", null);

document.getElementById('file').setAttribute("disabled", null);

document.getElementById('name').setAttribute("disabled", null);

document.getElementById('description').setAttribute("disabled", null);

await imageFile.saveIPFS();

const imageURI = imageFile.ipfs();

const metadata = {

"name":document.getElementById("name").value,

"description":document.getElementById("description").value,

"image":imageURI

}

const metadataFile = new Moralis.File("metadata.json", {base64 : btoa(JSON.stringify(metadata))});

await metadataFile.saveIPFS();

const metadataURI = metadataFile.ipfs();

const txt = await mintToken(metadataURI).then(notify)

}

Now it becomes easy to save that object either to the Moralis server or to IPFS.

#### **Moralis IPFS Storage**

In this guide, we are choosing to store our data in a decentralized way. Therefore, we are choosing IPFS (InterPlanetary File System). To save the file, use the following command (included in the code above):

await imageFile.saveIPFS();

With just one line of code, Moralis allows you to upload and save a file to IPFS. The alternative – by doing the backend work yourself without middleware like Moralis – is unnecessarily laborious and time-consuming.

When saving an image or file to IPFS, Moralis provides us with two things: (1) the hash of the file, which is how IPFS determines that the file has no duplicate in the network, (2) the file’s URI (the equivalent of a URL), provided by someone in the network with an IPFS gateway.

#### **Fetching the URI**

This command allows Moralis to fetch the URI:

const imageURI = imageFile.ipfs();

Now that we have the file’s URI, we can create a metadata object that contains everything stored that concerns the NFT, such as the name, description, and URI.

Const metadata = {

“name”: document.getElementById(“name”).value,

“description”: document.getElementById (“description”).value,

“Image”: imageURI

}

We’re getting the “ElementByID” here, which is the name of the NFT, plus the description, which describes the NFT, and the image an image URI that we got from Moralis in the previous step.

Moreover, with the metadata ready, you can now store it as a JSON object or file in IPFS:

const metadataFile = new Moralis.File("metadata.json", {base64 : btoa(JSON.stringify(metadata))});

await metadataFile.saveIPFS();

const metadataURI = metadataFile.ipfs();

const txt = await mintToken(metadataURI).then(notify)

}

With this JSON JavaScript object created, the file will be serialized so that it can be stored in IPFS. After the JSON file is stored in IPFS, you can then fetch the metadata URI by calling the “mintToken” function.

1. Create the Mint Function

To complete your NFT minting dApp, you need to create the minting function. This function creates a new object that contains an encoded function call, which is needed for executing the transaction in the smart contract.

After establishing the function, the code will add it to the “transactionParameters” object. The code will add it alongside the smart contract address we refer to in the initial step. As you can see below, the last section of the function will send the transaction to the blockchain, which will also return a transaction hash. In addition, the user receives the confirmed transaction ID.

async function mintToken(\_uri){

const encodedFunction = web3.eth.abi.encodeFunctionCall({

name: "mintToken",

type: "function",

inputs: [{

type: 'string',

name: 'tokenURI'

}]

}, [\_uri]);

const transactionParameters = {

to: nft\_contract\_address,

from: ethereum.selectedAddress,

data: encodedFunction

};

const txt = await ethereum.request({

method: 'eth\_sendTransaction',

params: [transactionParameters]

});

return txt

}

async function notify(\_txt){

document.getElementById("resultSpace").innerHTML =

`<input disabled = "true" id="result" type="text" class="form-control" placeholder="Description" aria-label="URL" aria-describedby="basic-addon1" value="Your NFT was minted in transaction ${\_txt}">`;

}

## **NFT Token Development – Summary**

There you have it, an NFT minting dApp created in a few easy steps with Moralis. One of the most important features of Moralis that gives you an edge in NFT development is its capability of supporting both centralized and decentralized options when it comes to storing your NFT metadata. Moralis does not dictate how you build your dApps. Instead, it puts the power of flexible, agile, and powerful dApp development in your hands!

In addition, [Moralis](https://moralis.io/speedy-nodes/?utm_source=blog&utm_medium=post&utm_campaign=NFT%2520Token%2520Development%2520%25E2%2580%2593%2520The%2520Ultimate%2520Guide) lets you create a server for your NFT project in minutes, and [Moralis Speedy Nodes](https://moralis.io/speedy-nodes/?utm_source=blog&utm_medium=post&utm_campaign=NFT%2520Token%2520Development%2520%25E2%2580%2593%2520The%2520Ultimate%2520Guide) saves you the tiresome heavy lifting of running and setting up your own node from scratch! Furthermore, with a single line of code, Moralis supports every step of your dApp development, from enabling seamless user authentication to connecting to [IPFS](https://moralis.io/what-is-ipfs-interplanetary-file-system/?utm_source=blog&utm_medium=post&utm_campaign=NFT%2520Token%2520Development%2520%25E2%2580%2593%2520The%2520Ultimate%2520Guide). It truly is the ultimate Web3 development platform to help you build and scale your dApps with less cost and complexity.