# **How to Make an NFT in 14 Lines of Code**

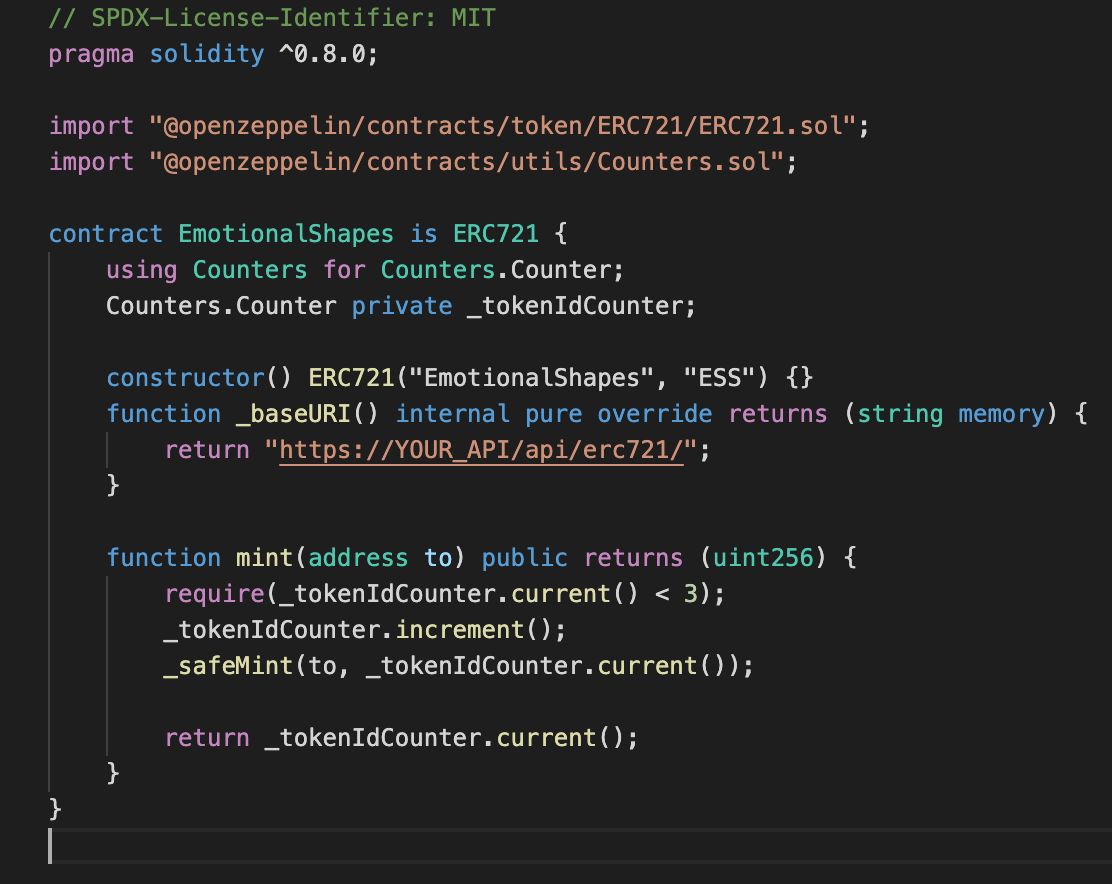
****

[**Nico**](https://www.freecodecamp.org/news/author/nicoinmeatworld/)

****

If you're a developer who's interested in Blockchain development, you should know something about NFTs, or Non-Fungible Tokens. So in this article, we'll learn about the engineering behind them so you can start building your own.

At the end of the project, you will have your own Ethereum wallet with a new NFT in it. This tutorial is beginner-friendly and does not require any prior knowledge of the Ethereum network or smart contracts.

The NFT contract has only 14 lines of code

## **What is an NFT?**

NFT stands for non-fungible token. [This quote from ethereum.org](https://ethereum.org/en/nft/) explains it well:

*NFTs are tokens that we can use to represent ownership of unique items. They let us tokenise things like art, collectibles, even real estate. They can only have one official owner at a time and they're secured by the Ethereum blockchain – no one can modify the record of ownership or copy/paste a new NFT into existence.*

## **What is an NFT standard or ERC-721?**

The ERC-721 is the most common NFT standard. If your Smart Contract implements certain standardized API methods, it can be called an ERC-721 Non-Fungible Token Contract.

These methods are specified in the [EIP-721](https://eips.ethereum.org/EIPS/eip-721). Open-sourced projects like OpenZeppelin have simplified the development process by implementing the most common ERC standards as a reusable library.

## **What is minting an NFT?**

By minting an NFT, you publish a unique token on a blockchain. This token is an instance of your Smart Contract.

Each token has a unique tokenURI, which contains metadata of your asset in a JSON file that conforms to certain schema. The metadata is where you store information about your NFT, such as name, image, description, and other attributes.

An example of the JSON file for the "ERC721 Metadata Schema" looks like this:

{

"attributes": [

{

"trait\_type": "Shape",

"value": "Circle"

},

{

"trait\_type": "Mood",

"value": "Sad"

}

],

"description": "A sad circle.",

"image": "https://i.imgur.com/Qkw9N0A.jpeg",

"name": "Sad Circle"

}

## **How do I store my NFT's metadata?**

There are three main ways to store an NFT's metadata.

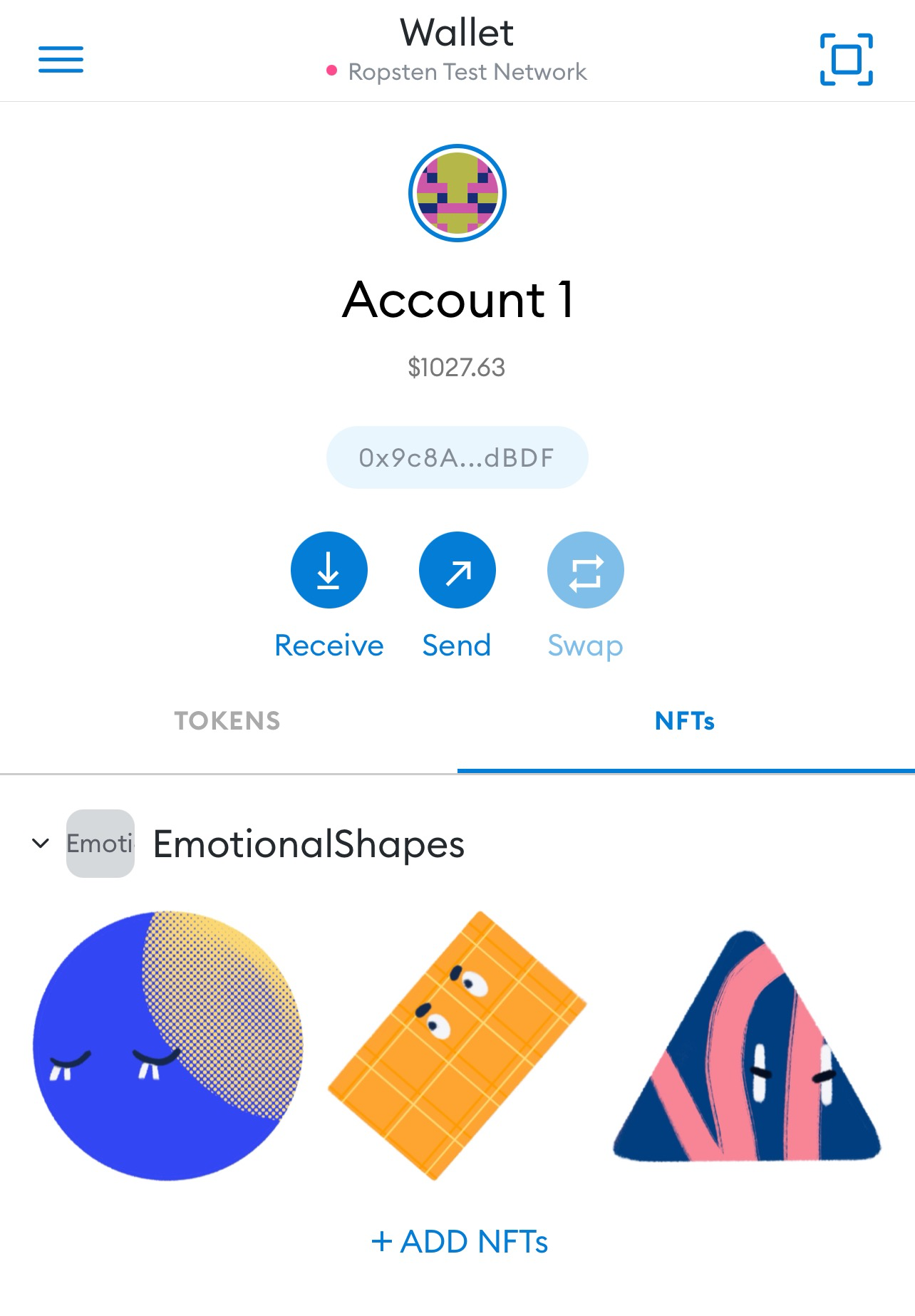
First, you can store the information on-chain. In other word, you can extend your ERC-721 and store the metadata on the blockchain, which can be costly.

The second method is to use [IPFS](https://docs.ipfs.io/concepts/what-is-ipfs/). And the third way is to simply have your API return the JSON file.

The first and second methods are usually preferred, since you cannot temper the underlying JSON file. For the scope of this project, we will opt for the third method.

For a good tutorial on using NFTs with IPFS, read [this article](https://docs.alchemy.com/alchemy/tutorials/how-to-create-an-nft/how-to-mint-a-nft#step-4-configure-the-metadata-for-your-nft-using-ipfs) by the Alchemy team.

## **What We'll Be Building**

****

In this tutorial, we'll be creating and minting our own NFT. It is beginner-friendly and does not require any prior knowledge of the Ethereum network or smart contracts. Still, having a good grasp on those concepts will help you understand what is going on behind the scenes.

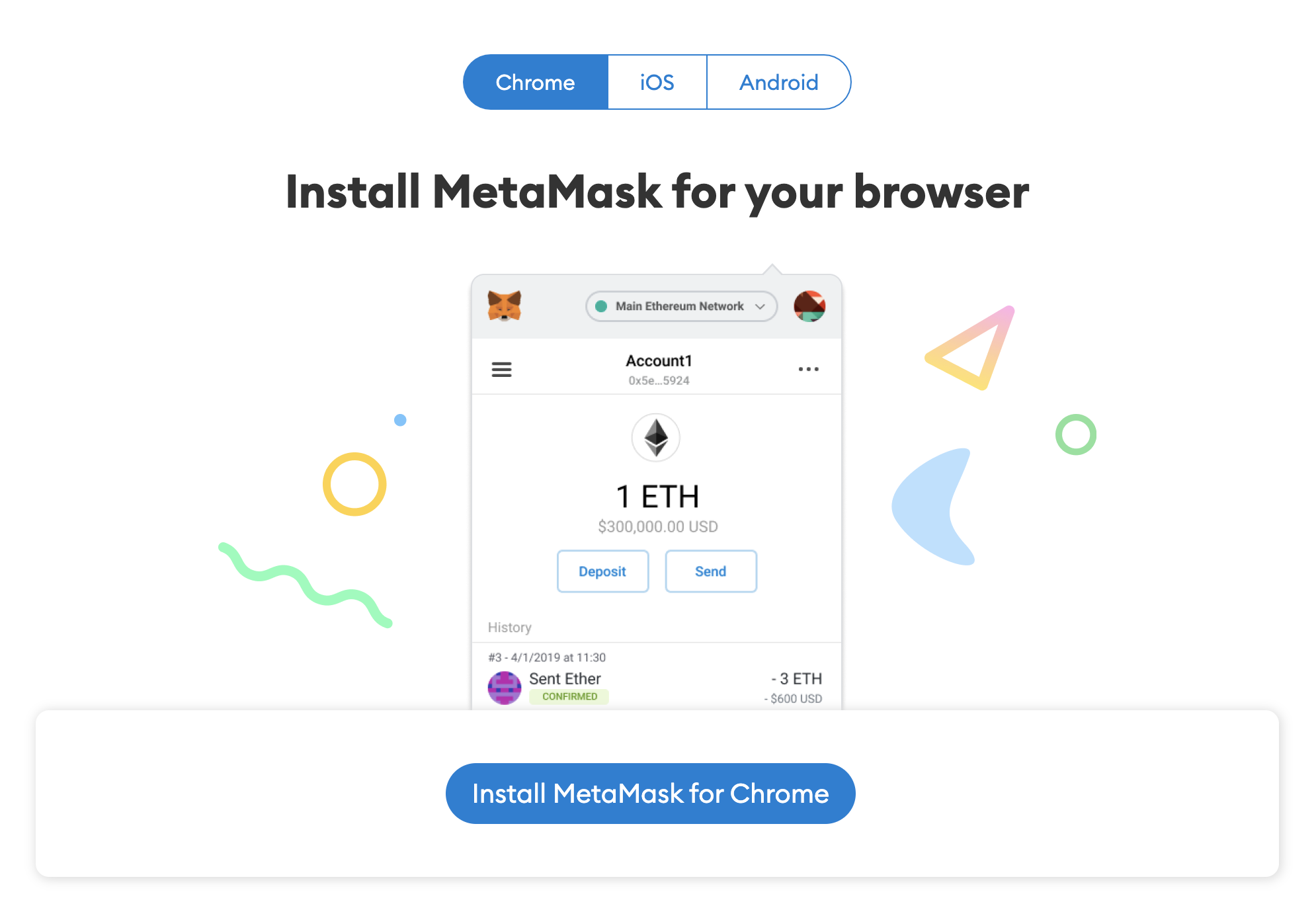
In an upcoming tutorial, we'll build a fully-functional React web app where you can display and sell your NFTs.

If you are just getting started with dApp development, begin by reading through [the key topics](https://ethereum.org/en/developers/docs/intro-to-ethereum/) and watch this [amazing course](https://www.youtube.com/watch?v=M576WGiDBdQ) by Patrick Collins.

*This project is intentionally written with easily understandable code and is not suitable for production usage.*

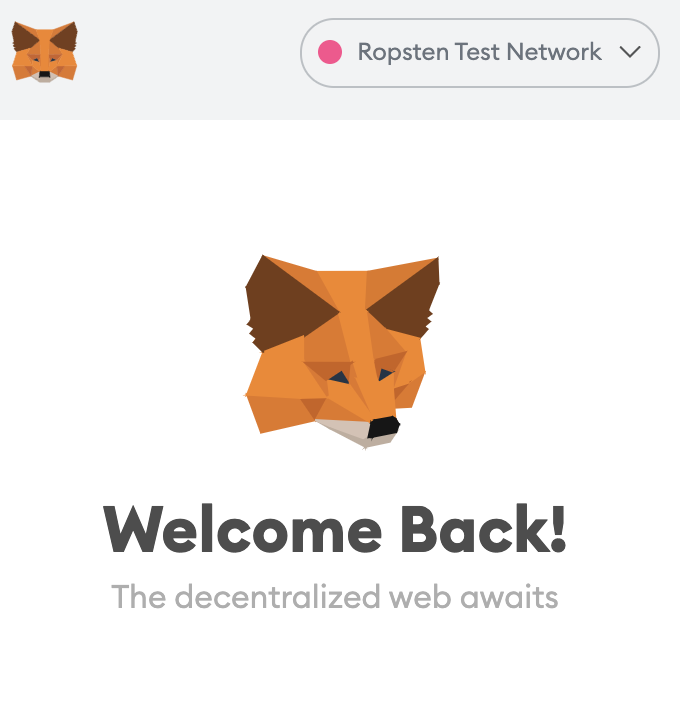
## **Prerequisites**

### **Metamask**

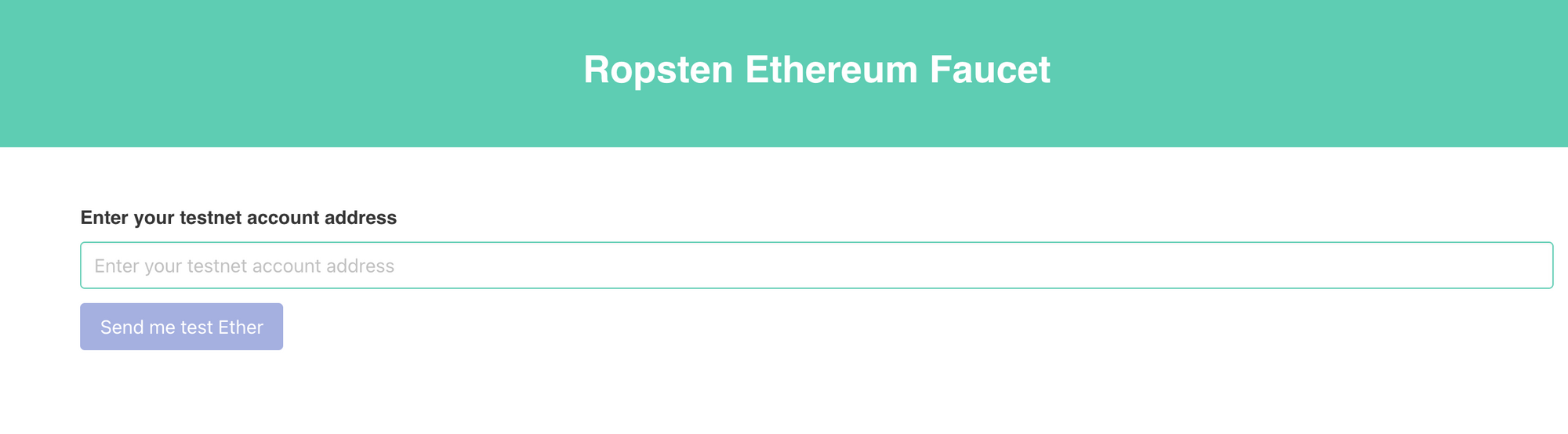
****

We need an Ethereum address to interact with our Smart Contract. We will be using [Metamask](https://metamask.io/) as our wallet. It is a free virtual wallet that manages your Ethereum addresses. We will need it to send and receive transactions (read more on that [here](https://ethereum.org/en/developers/docs/transactions/)). For example, minting an NFT is a transaction.

Download their Chrome extension and their mobile app. We will need both as the Chrome extension does not display your NFTs.



Make sure to change the network to "Ropsten Test Network" for development purposes. You will need some Eth to cover the fees of deploying and minting your NFT. Head to the [Ropsten Ethereum Faucet](https://faucet.ropsten.be/) and enter your address. You should soon see some test Eth in your Metamask account.

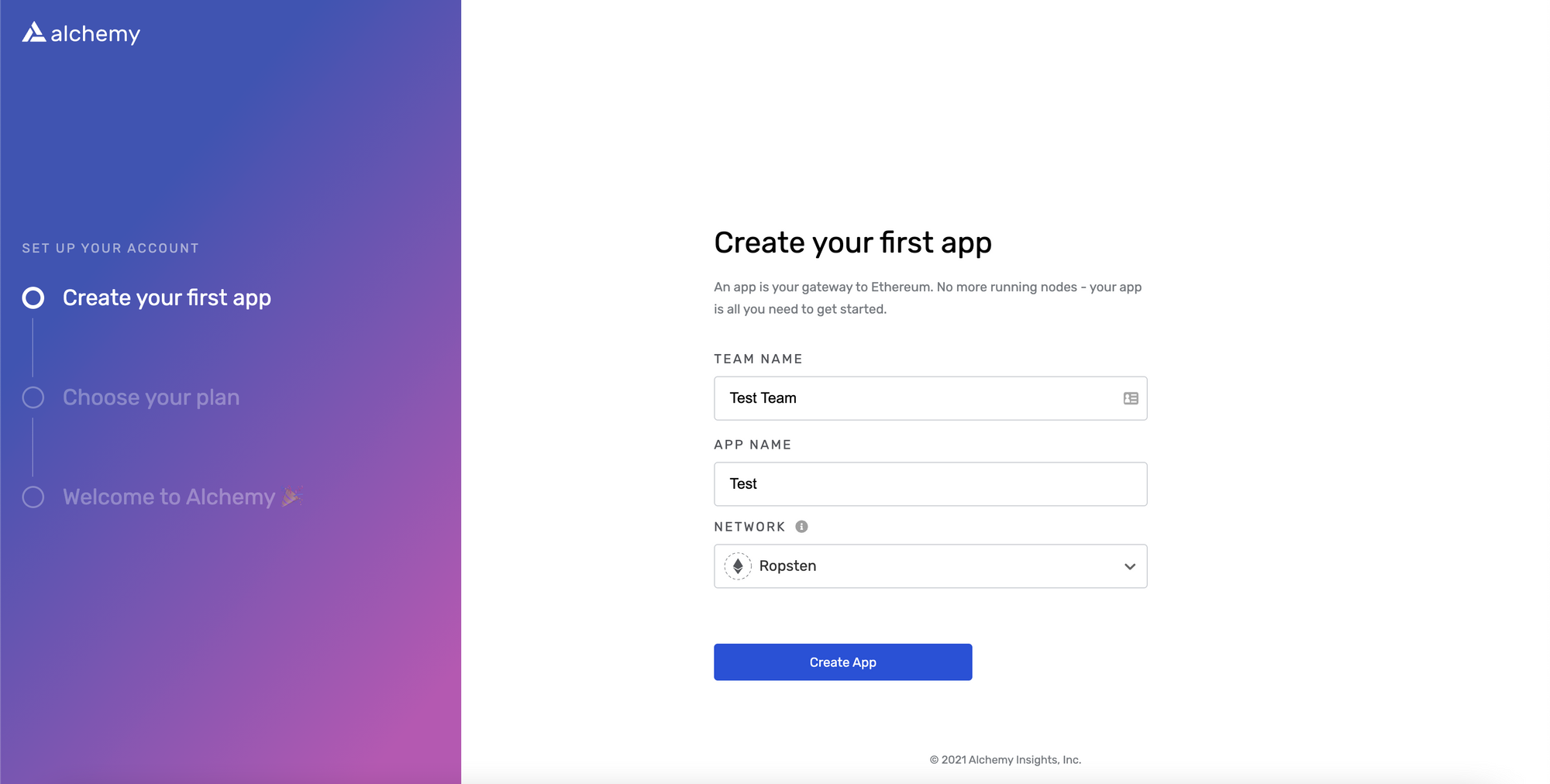


### **Alchemy**

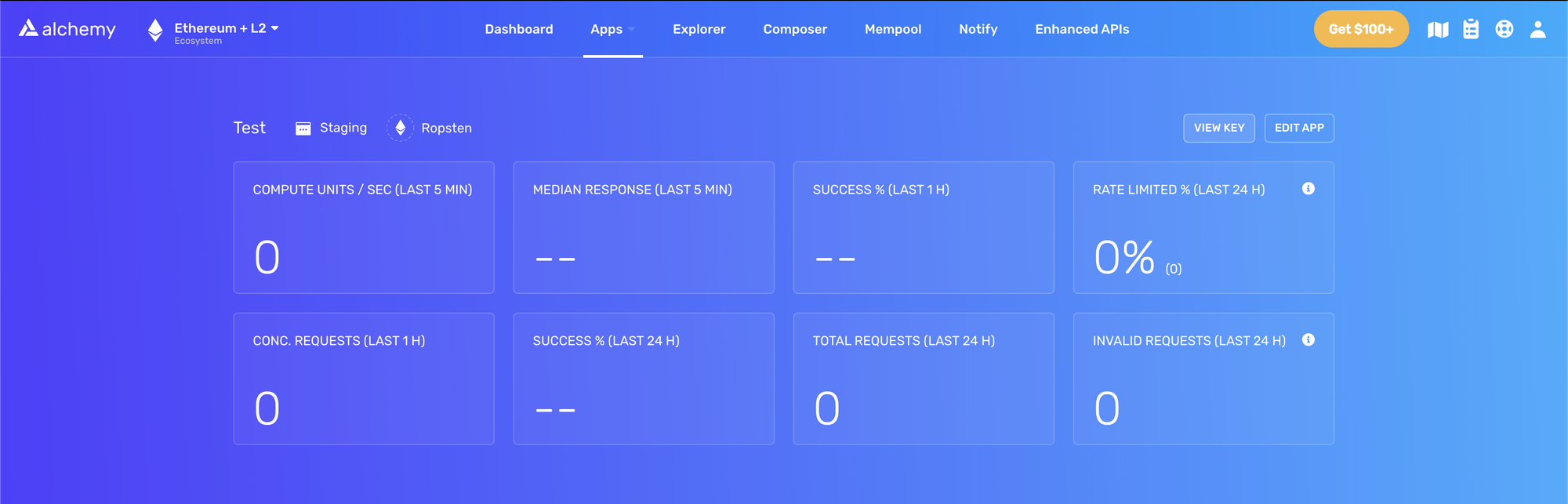
To interact with the Ethereum Network, you will need to be connected to an Ethereum Node.

Running your own Node and maintaining the infrastructure is a project on its own. Luckily, there are nodes-as-a-service providers which host the infrastructure for you. There are many choices like Infura, BlockDaemon, and Moralis. We will be using [Alchemy](https://www.alchemy.com/) as our node provider.

Head over to their website, create an account, choose Ethereum as your network and create your app. Choose Ropsten as your network.



On your dashboard, click "view details" on your app, then click "view key". Save your http key somewhere as we will need that later.



### **NodeJS/NPM**

We will be using NodeJS for the project. If you don't have it installed, [follow this simple tutorial](https://www.freecodecamp.org/news/how-to-install-node-in-your-machines-macos-linux-windows/) by freeCodeCamp.

## **Initialize the project**

In your terminal, run this command to make a new directory for your project:

mkdir nft-project

cd nft-project

Now, let's make another directory, ethereum/, inside nft-project/ and initialize it with [Hardhat](https://hardhat.org/getting-started/). Hardhat is a dev tool that makes it easy to deploy and test your Ethereum software.

mkdir ethereum

cd ethereum

npm init

Answer the questions however you want. Then, run those commands to make a Hardhat project:

npm install --save-dev hardhat

npx hardhat

You will see this prompt:

888 888 888 888 888

888 888 888 888 888

888 888 888 888 888

8888888888 8888b. 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 Y88b 888 888 888 888 888 Y88b.

888 888 "Y888888 888 "Y88888 888 888 "Y888888 "Y888

Welcome to Hardhat v2.0.8

? What do you want to do? …

Create a sample project

❯ Create an empty hardhat.config.js

Quit

Select create an empty hardhat.config.js. This will generate an empty hardhat.config.js file that we will later update.

For the web app, we will use [Next.js](https://nextjs.org/docs/getting-started) to initialize a fully-functional web app. Go back to the root directory nft-project/ and initialize a boilerplate Next.js app called web:

cd ..

mkdir web

cd web

npx create-next-app@latest

Your project now looks like this:

nft-project/

ethereum/

web/

Awesome! We are ready to dive into some real coding.

## **How to Define Our .env Variables**

Remember the Alchemy key we grabbed from our test project earlier? We will use that along with our Metamask account's public and private keys to interact with the blockchain.

Run the following commands, make a file called .env inside your ethereum/ directory, and install [dotenv](https://www.npmjs.com/package/dotenv). We will use them later.

cd ..

cd ethereum

touch .env

npm install dotenv --save

For your .env file, put the key you have exported from Alchemy and [follow those instructions](https://metamask.zendesk.com/hc/en-us/articles/360015289632-How-to-Export-an-Account-Private-Key) to grab your Metamask's private key.

Here's your .env file:

DEV\_API\_URL = YOUR\_ALCHEMY\_KEY

PRIVATE\_KEY = YOUR\_METAMASK\_PRIVATE\_KEY

PUBLIC\_KEY = YOUR\_METAMASK\_ADDRESS

## **The Smart Contract for NFTs**

Go to the ethereum/ folder and create two more directories: contracts and scripts. [A simple hardhat project](https://hardhat.org/guides/project-setup.html) contains those folders.

* contracts/ contains the source files of your contracts
* scripts/ contains the scripts to deploy and mint our NFTs

mkdir contracts

mkdir scripts

Then, install OpenZeppelin. [OpenZeppelin Contract](https://docs.openzeppelin.com/contracts/4.x/) is an open-sourced library with pre-tested reusable code to make smart contract development easier.

npm install @openzeppelin/contracts

Finally, we will be writing the Smart Contract for our NFT. Navigate to your contracts directory and create a file titled EmotionalShapes.sol. You can name your NFTs however you see fit.

The .sol extension refers to the Solidity language, which is what we will use to program our Smart Contract. We will only be writing 14 lines of code with Solidity, so no worries if you haven't seen it before.

Start with [this article](https://ethereum.org/en/developers/docs/smart-contracts/languages/) to learn more about Smart Contract languages. You can also directly jump to this Solidity [cheat sheet](https://reference.auditless.com/cheatsheet/) which contains the main syntax.

cd contracts

touch EmotionalShapes.sol

This is our Smart Contract:

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

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

import "@openzeppelin/contracts/utils/Counters.sol";

contract EmotionalShapes is ERC721 {

using Counters for Counters.Counter;

Counters.Counter private \_tokenIdCounter;

constructor() ERC721("EmotionalShapes", "ESS") {}

function \_baseURI() internal pure override returns (string memory) {

return "YOUR\_API\_URL/api/erc721/";

}

function mint(address to)

public returns (uint256)

{

require(\_tokenIdCounter.current() < 3);

\_tokenIdCounter.increment();

\_safeMint(to, \_tokenIdCounter.current());

return \_tokenIdCounter.current();

}

}

Let's go through the code and understand what is going on.

1. At the top of the file, we specified which OpenZeppelin module to import. We need the ERC721 contract as it is the 'base' of our Smart Contract. It has already implemented all the methods specified in [EIP-721](https://eips.ethereum.org/EIPS/eip-721) so we can safely use it.
2. A Counter is useful to generate incremental ids for our NFTs. We named the variable \_tokenIdCounter
3. In the constructor, we initialized our ERC721 with its name and its symbol. I chose EmotionalShapes and ESS.
4. We override the default \_baseURI function by returning our own. We will get to build that in a second. In summary, it is the URL that will be added as 'prefix' to all our tokenURIs. In the above example, the metadata of our NFTs will live in a JSON file at YOUR\_API\_URL/[api/erc721/](https://e110-99-121-58-31.ngrok.io/api/erc721/)1.
5. We implement the 'mint' function. It is the function that lets you publish an instance of this Smart Contract on the blockchain. I required the \_tokenIdCounter variable to be less than 3 as I will only create three instances of my NFT. You can remove that if you want to mint more.
6. Finally, inside the mint function, we increment the \_tokenIdCountervariable by 1, so our id will be 1, followed by 2, followed by 3. Then, we call the function provided by OpenZeppelin \_safeMint to publish the token.

Don't worry if you feel lost. You can attend a workshop led by volunteers from freeCodeCamp, where we invite devs of similar skill levels to build stuff together, including this NFT project.

The events are free and remote, so you can ask any questions directly. You can register [here](https://equia.io/). The seats are limited so you will be invited to the next available events.

## **How to Build the Metadata for our NFT**

As mentioned earlier, there are three main ways of storing your tokenURI. We will be building a simple API endpoint which resolve in our NFT's information as JSON.

Our Next.js project gives us a handy way to develop API routes. Go to the web/ folder, find the api/ folder within the pages/ folder, and make our dynamic [id].js route in a erc721/ folder (read more about routing [here](https://www.freecodecamp.org/news/p/18513919-9e93-4ab3-9f52-2448aafa8835/develop%20API%20routes)):

// web/pages/api/erc721/[id].js

const metadata = {

1: {

attributes: [

{

trait\_type: "Shape",

value: "Circle",

},

{

trait\_type: "Mood",

value: "Sad",

},

],

description: "A sad circle.",

image: "https://i.imgur.com/Qkw9N0A.jpeg",

name: "Sad Circle",

},

2: {

attributes: [

{

trait\_type: "Shape",

value: "Rectangle",

},

{

trait\_type: "Mood",

value: "Angry",

},

],

description: "An angry rectangle.",

image: "https://i.imgur.com/SMneO6k.jpeg",

name: "Angry Rectangle",

},

3: {

attributes: [

{

trait\_type: "Shape",

value: "Triangle",

},

{

trait\_type: "Mood",

value: "Bored",

},

],

description: "An bored triangle.",

image: "https://i.imgur.com/hMVRFoJ.jpeg",

name: "Bored Triangle",

},

};

export default function handler(req, res) {

res.status(200).json(metadata[req.query.id] || {});

}

For the sake of this project, I made the code as easily understandable as possible. This is definitely not suited for production (please don't use an Imgur url for your NFT). Make sure to define the metadata for all the NFTs that you intend to mint.

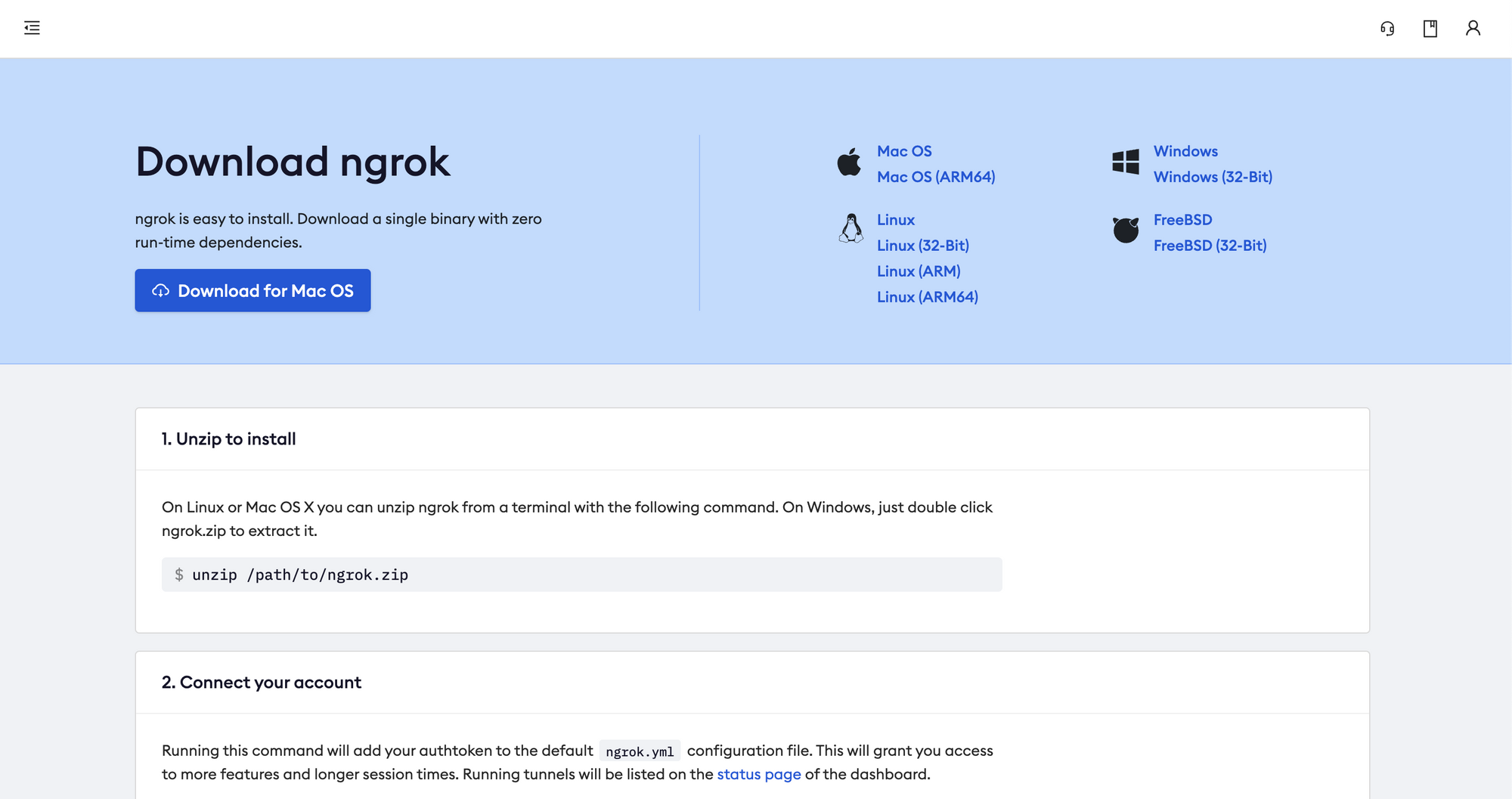
Now, go to the web directory, and start your Next.js app with this command:

npm run dev

Your app should be running on localhost:3000. To make sure our endpoint works, go to <http://localhost:3000/api/erc721/1> and it should resolve with a JSON object of your first NFT's metadata.

## **How to Expose the Metadata for our NFT**

Since your app is hosted locally, other apps cannot access it. Using a tool like [ngrok](https://ngrok.com/), we can expose our local host to a publicly accessible URL.



1. Go to [ngrok.com](https://ngrok.com/) and complete the registration process
2. Unzip the downloaded package
3. In your terminal, make sure you cd into the folder where you unzipped your ngrok package
4. Follow the instruction on your dashboard and run

./ngrok authtoken YOUR\_AUTH\_TOKEN

5. Then, run this command to create a tunnel to your web app hosted on localhost:3000

./ngrok http 3000

6. You are almost there! On your terminal, you should see something like this:

ngrok by @inconshreveable (Ctrl+C to quit)

Session Status online

Account YOUR\_ACCOUNT (Plan: Free)

Version 2.3.40

Region United States (us)

Web Interface http://127.0.0.1:4040

Forwarding http://YOUR\_NGROK\_ADDRESS -> http://localhost:3000

Forwarding https://YOUR\_NGROK\_ADDRESS -> http://localhost:3000

Go to YOUR\_NGROK\_ADDRESS/api/erc721/1 to make sure your endpoint works correctly.

## **How to Deploy our NFT**

Now that we have done all the ground work (oof), let's go back to our ethereum/ folder and get ready to deploy our NFT.

Change the \_baseURI function in your ethreum/contracts/YOUR\_NFT\_NAME.sol fileto return your ngrok address.

// ethereum/conrtacts/EmotionalShapes.sol

contract EmotionalShapes is ERC721 {

...

function \_baseURI() internal pure override returns (string memory) {

return "https://YOUR\_NGROK\_ADDRESS/api/erc721/";

}

...

}

To deploy our NFT, we will first need to [compile it using Hardhat](https://hardhat.org/guides/compile-contracts.html). To make the process easier, we will install [ethers.js](https://docs.ethers.io/v5/).

npm install @nomiclabs/hardhat-ethers --save-dev

Let's update our hardhat.config.js:

require("dotenv").config();

require("@nomiclabs/hardhat-ethers");

module.exports = {

solidity: "0.8.0",

defaultNetwork: "ropsten",

networks: {

hardhat: {},

ropsten: {

url: process.env.DEV\_API\_URL,

accounts: [`0x${process.env.PRIVATE\_KEY}`],

},

},

};

To learn more about the hardhat configuration file, take a look at their [documentation](https://hardhat.org/config/). We have configured the ropsten network with our Alchemy URL and provided it with the private key of your metamask account.

Finally, run:

npx hardhat compile

This lets hardhat generate two files per compiled contract. We should see a newly created artifacts/ folder that contains your compiled contracts in the contracts/ folder. To learn more about how that works, read [this tutorial](https://hardhat.org/guides/compile-contracts.html) by the Hardhat team.

Now, let's write a script to finally deploy our NFT to the test network. In your scripts/ folder, create a file called deploy.js.

// ethereum/scripts/deploy.js

async function main() {

const EmotionalShapes = await ethers.getContractFactory("EmotionalShapes");

const emotionalShapes = await EmotionalShapes.deploy();

console.log("EmotionalShapes deployed:", emotionalShapes.address);

}

main()

.then(() => process.exit(0))

.catch((error) => {

console.error(error);

process.exit(1);

});

This code is inspired by [the hardhat deployment tutorial](https://hardhat.org/guides/deploying.html).

A ContractFactory in ethers.js is an abstraction used to deploy new smart contracts, so EmotionalShapes here is a factory for instances of our token contract. Calling deploy() on a ContractFactory will start the deployment, and return a Promise that resolves to a Contract. This is the object that has a method for each of your smart contract functions.

### **How to view the NFT on the blockchain**

Run the deployment script:

node ./scripts/deploy.js

You should see in your terminal EmotionalShapes deployed: SOME\_ADDRESS. This is the address where your Smart Contract is deployed on the ropsten test network.

If you head over to https://ropsten.etherscan.io/address/SOME\_ADDRESS, you should see your freshly deployed NFT. Yes! You did it!

If you are stuck somewhere in the tutorial or feeling lost, again, you can [join our live workshops](https://equia.io/) where we will build this project together in a Zoom call.

## **How to Mint your NFT**

Now that you have deployed your NFT, it's time to mint it for yourself! Create a new file called mint.js in your scripts/ folder. We will be using ethers.js to help us.

Start by adding the ethers.js package:

npm install --save ethers

Then, populate the mint.js file:

require("dotenv").config();

const { ethers } = require("ethers");

const contract = require("../artifacts/contracts/EmotionalShapes.sol/EmotionalShapes.json");

const contractInterface = contract.abi;

// https://docs.ethers.io/v5/api/providers

const provider = ethers.getDefaultProvider("ropsten", {

alchemy: process.env.DEV\_API\_URL,

});

// https://docs.ethers.io/v5/api/signer/#Wallet

const wallet = new ethers.Wallet(process.env.PRIVATE\_KEY, provider);

//https://docs.ethers.io/v5/api/contract/contract

const emotionalShapes = new ethers.Contract(

YOUR\_NFT\_ADDRESS,

contractInterface,

wallet

);

const main = () => {

emotionalShapes

.mint(process.env.PUBLIC\_KEY)

.then((transaction) => console.log(transaction))

.catch((e) => console.log("something went wrong", e));

};

main();

I have left comments to where you can find more information about the different methods. We first grab the contract's interface (ABI). From ethereum.org:

An application binary interface, or ABI, is the standard way to interact with [contracts](https://ethereum.org/en/glossary/#contract-account) in the Ethereum ecosystem, both from outside the blockchain and for contract-to-contract interactions.

Your ABI defines how others interact with your contract. Then, we created our provider with Alchemy (remember about node-as-a-service). Finally, we initialize our wallet with our private key.

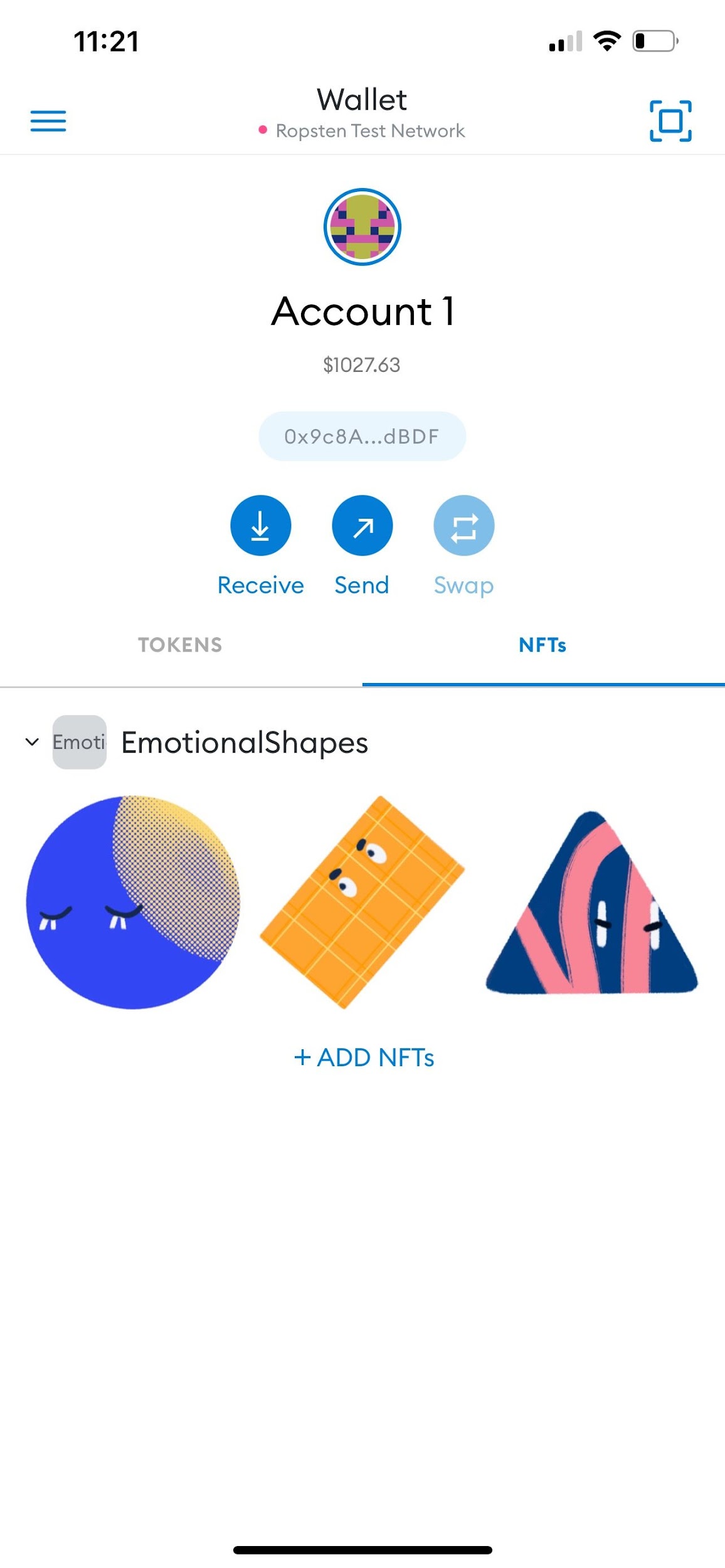
The main() function calls the mint method in the Smart Contract we had just deployed. The mint method takes only one parameter, to, which indicate the receiver of the token. Since we are minting for ourself, we put the public address of our Metamask account.

If everything goes well, you should see the transaction logged in your terminal. Grab the hash property and go to https://ropsten.etherscan.io/tx/YOUR\_HASH. You should see the minting transaction there!

## **How to View the NFT in your Metamask Wallet**

You need to start by downloading the mobile version of Metamask. Then, log into your account.

You should see an NFTs tab along with an add NFT button. Click on the button and enter the address of your Smart Contract along with the ids that you have minted. If you have followed the tutorial, you should start with an id of 1.

View NFTs in your Metamask wallet

## **Conclusion**

Congratulations! You have just minted your own NFT. In the next part of the project, we will be building the front end React app to interact with our contract. The end goal is to build a fully functional web app where you can sell your own NFTs.

Lastly, you can [join our live workshops](https://equia.io/) with volunteers from freeCodeCamp where we will build this project together with other developers.

The events are free for everyone across the world and invitations are sent first-come, first-serve. If you'd like to lead the workshops, DM me [on Twitter](https://twitter.com/aly4alyssa), we'd love to have you! We also organize other type of events like hiring fairs and social meetups.

Let me know what you want to build. NFTs are still in its infancy and novel ideas are more than welcome. Can't wait to see what crazy idea you have!