

Hardhat Documentation

Hardhat Network is a local Ethereum network designed for development. It can easily deploy your contracts, run tests and debug Solidity code without dealing with live environments. It is made up of many parts that may be used to edit, compile, debug, and deploy your dApps and smart contracts, together forming a whole development environment.

When utilising Hardhat, the key element you deal with is Hardhat Runner. It is a versatile and extendable task runner that assists you in organising and automating the ongoing processes necessary for creating smart contracts and dApps.

The ideas of tasks and plugins are central to the design of Hardhat Runner. Every time you launch Hardhat from the command line, a task is launched. For instance, the built-in compile job is executed by `npx hardhat compile`. Because tasks may contact one another, complex workflows can be defined. Existing tasks may be replaced by users and plugins, allowing users to modify and expand processes.

In this documentation we'll guide you through:

1. Installing Hardhat and Creating a Hardhat project
2. Creating a Simple Smart Contract
3. Automated tests for your contract using Hardhat
4. Deploying on XDC Network using XDC Remix
5. Deploying the Smart Contract to Hardhat Network and Ethereum testnets

1. Installation

In your project, hardhats are implemented through a local installation. You may reproduce your environment in this manner and prevent further version disputes.

Before we begin, there are a few fundamental prerequisites. Install the following, if possible:

1. <https://nodejs.org/en/>
2. <https://git-scm.com/>

You need to navigate to an empty folder, launch `npm init`, and then follow the on-screen steps to establish a npm project in order to install it. Although you are free to use any package manager, such as yarn, we advise using npm 7 or later since it makes installing Hardhat plugins easier.

When your project is finished, you should run:

npm 7+ :

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

yarn:

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

Set up Hardhat

With an example contract, tests of the contract, and a script to deploy it, we will examine the fundamentals of constructing a Hardhat project.

Run `npx hardhat` in your project folder to build the sample project:

```
npm WARN config global '--global', '--local' are deprecated. Use '--location=global' instead.
Need to install the following packages:
  hardhat@2.11.1
Ok to proceed? (y) y
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 .d888888 888
888 888 888 888 888 Y88b 888 888 888 888 Y88b.
888 888 "Y888888 888 "Y88888 888 888 "Y888888 "Y888

Welcome to Hardhat v2.11.1

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

Create a JavaScript or TypeScript project and follow the steps below to compile, test, and deploy the sample contract. We recommend using TypeScript, but if you're unfamiliar, choose JavaScript.

```
E:\Projects\Hardhat>npx hardhat
Need to install the following packages:
  hardhat
Ok to proceed? (y) y
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 .d888888 888
888 888 888 888 888 Y88b 888 888 888 888 Y88b.
888 888 "Y888888 888 "Y88888 888 888 "Y888888 "Y888

Welcome to Hardhat v2.11.1

√ What do you want to do? · Create a JavaScript project
√ Hardhat project root: · E:\Projects\Hardhat
√ Do you want to add a .gitignore? (Y/n) · y

You need to install these dependencies to run the sample project:
  npm install --save-dev "hardhat@^2.11.1" "@nomicfoundation/hardhat-toolbox@^1.0.1"

Project created

See the README.md file for some example tasks you can run

Give Hardhat a star on Github if you're enjoying it!

https://github.com/NomicFoundation/hardhat
```

Running Tasks

The list of available tasks includes built-in tasks and tasks that come with installed plugins. `npx hardhat` is a starting point for finding tasks you can do.

While running this command, you might face these errors:

```
E:\Projects\Hardhat>npx hardhat
Error HH12: Trying to use a non-local installation of Hardhat, which is not supported.
Please install Hardhat locally using npm or Yarn, and try again.

For more info go to https://hardhat.org/HH12 or run Hardhat with --show-stack-traces
```

```
E:\Projects\Hardhat>npx hardhat
An unexpected error occurred:

Error: Cannot find module '@nomicfoundation/hardhat-toolbox'
Require stack:
- E:\Projects\Hardhat\hardhat.config.js
- E:\Projects\Hardhat\node_modules\hardhat\internal\core\config\config-loading.js
- E:\Projects\Hardhat\node_modules\hardhat\internal\cli\cli.js
  at Function.Module._resolveFilename (node:internal/modules/cjs/loader:933:15)
  at Function.Module._load (node:internal/modules/cjs/loader:778:27)
  at Module.require (node:internal/modules/cjs/loader:1005:19)
  at require (node:internal/modules/cjs/helpers:102:18)
  at Object.<anonymous> (E:\Projects\Hardhat\hardhat.config.js:1:1)
  at Module._compile (node:internal/modules/cjs/loader:1103:14)
  at Object.Module._extensions..js (node:internal/modules/cjs/loader:1157:10)
  at Module.load (node:internal/modules/cjs/loader:981:32)
  at Function.Module._load (node:internal/modules/cjs/loader:822:12)
  at Module.require (node:internal/modules/cjs/loader:1005:19) {
  code: 'MODULE_NOT_FOUND',
  requireStack: [
    'E:\\Projects\\Hardhat\\hardhat.config.js',
    'E:\\Projects\\Hardhat\\node_modules\\hardhat\\internal\\core\\config\\config-loading.js',
    'E:\\Projects\\Hardhat\\node_modules\\hardhat\\internal\\cli\\cli.js'
  ]
}
```

To solve this, first run the commands:

```
npm i hardhat
```

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

(For versions greater than 7) This will include ether.js library to interact with our contracts.

If you are using an older version of npm, you'll also need to install all the packages used by the toolbox.

```
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
```

Yarn –

```
yarn add --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
```

```
E:\Projects\Hardhat>npm install --save-dev @nomicfoundation/hardhat-toolbox
npm WARN deprecated mkdirp@5.0.1: This package is broken and no longer maintained. 'mkdirp' itself supports prom
ises now, please switch to that.
npm WARN deprecated request-promise-native@1.0.9: request-promise-native has been deprecated because it extends the now
deprecated request package, see https://github.com/request/request/issues/3142
npm WARN deprecated har-validator@5.1.5: this library is no longer supported
npm WARN deprecated debug@3.2.6: Debug versions >=3.2.0 <3.2.7 || >=4 <4.3.1 have a low-severity ReDos regression when u
sed in a Node.js environment. It is recommended you upgrade to 3.2.7 or 4.3.1. (https://github.com/visionmedia/debug/iss
ues/797)
npm WARN deprecated multicodec@1.0.4: This module has been superseded by the multiformats module
npm WARN deprecated uuid@2.0.1: Please upgrade to version 7 or higher. Older versions may use Math.random() in certain
circumstances, which is known to be problematic. See https://v8.dev/blog/math-random for details.
npm WARN deprecated uuid@2.0.1: Please upgrade to version 7 or higher. Older versions may use Math.random() in certain
circumstances, which is known to be problematic. See https://v8.dev/blog/math-random for details.
npm WARN deprecated uuid@3.4.0: Please upgrade to version 7 or higher. Older versions may use Math.random() in certain
circumstances, which is known to be problematic. See https://v8.dev/blog/math-random for details.
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/request/request/issues/3142
npm WARN deprecated multibase@0.6.1: This module has been superseded by the multiformats module
npm WARN deprecated multibase@0.7.0: This module has been superseded by the multiformats module
npm WARN deprecated uuid@3.3.2: Please upgrade to version 7 or higher. Older versions may use Math.random() in certain
circumstances, which is known to be problematic. See https://v8.dev/blog/math-random for details.
npm WARN deprecated multicodec@0.5.7: This module has been superseded by the multiformats module
npm WARN deprecated cids@0.7.5: This module has been superseded by the multiformats module

added 624 packages, changed 2 packages, and audited 925 packages in 33s

127 packages are looking for funding
  run `npm fund` for details

7 moderate severity vulnerabilities

To address issues that do not require attention, run:
  npm audit fix

Some issues need review, and may require choosing
a different dependency.

Run `npm audit` for details.
```

A complete video guide for the installation of Hardhat is given here:

<https://drive.google.com/drive/folders/1U4aV3DHuddnQy23XY3iwZd3ekUnNnrVU?usp=sharing>

Now run the `npx hardhat` command:

```

E:\Projects\Hardhat>npx hardhat
Hardhat version 2.11.1

Usage: hardhat [GLOBAL OPTIONS] <TASK> [TASK OPTIONS]

GLOBAL OPTIONS:

  --config           A Hardhat config file.
  --emoji            Use emoji in messages.
  --flamegraph       Generate a flamegraph of your Hardhat tasks
  --help            Shows this message, or a task's help if its name is provided
  --max-memory       The maximum amount of memory that Hardhat can use.
  --network          The network to connect to.
  --show-stack-traces Show stack traces.
  --tsconfig         A TypeScript config file.
  --typecheck        Enable TypeScript type-checking of your scripts/tests
  --verbose         Enables Hardhat verbose logging
  --version          Shows hardhat's version.

AVAILABLE TASKS:

  check            Check whatever you need
  clean            Clears the cache and deletes all artifacts
  compile          Compiles the entire project, building all artifacts
  console          Opens a hardhat console
  coverage         Generates a code coverage report for tests
  flatten          Flattens and prints contracts and their dependencies
  gas-reporter:merge
  help            Prints this message
  node            Starts a JSON-RPC server on top of Hardhat Network
  run             Runs a user-defined script after compiling the project
  test            Runs mocha tests
  typechain        Generate Typechain typings for compiled contracts
  verify          Verifies contract on Etherscan

To get help for a specific task run: npx hardhat help [task]

```

A list of all the available commands will be displayed in the terminal. These are the currently available tasks in your hardhat project.

You can create your own additional tasks, which will be shown in this list. Let's assume you want to create a task to print a message, say "Hello World", or a task to show your account balance.

Let's get an account's balance from the terminal through a task. We will be using Hardhat's config API, which is available in the global scope of `hardhat.config.js`.

```

require("@nomicfoundation/hardhat-toolbox");

/** @type import('hardhat/config').HardhatUserConfig */
module.exports = {
  solidity: "0.8.9",
};

task("accounts", "Prints the list of accounts", async (taskArgs, hre) => {
  const accounts = await hre.ethers.getSigners();

  for (const account of accounts) {
    console.log(account.address);
  }
});

task("balance", "Prints an account's balance")
  .setAction(async (taskArgs) => {});

```

Save this file and run the command `npx hardhat`:

You will be able to see the task in Hardhat:

```
AVAILABLE TASKS:

accounts      Prints the list of accounts
balance       Prints an account's balance
check         Check whatever you need
clean         Clears the cache and deletes all artifacts
compile       Compiles the entire project, building all artifacts
console       Opens a hardhat console
coverage      Generates a code coverage report for tests
deploy        Deploy the smart contracts
flatten       Flattens and prints contracts and their dependencies
gas-reporter:merge
help          Prints this message
node          Starts a JSON-RPC server on top of Hardhat Network
run           Runs a user-defined script after compiling the project
test          Runs mocha tests
typechain     Generate Typechain typings for compiled contracts
verify        Verifies contract on Etherscan

To get help for a specific task run: npx hardhat help [task]
```

For printing the account's balance, we need the user to give account's address. We can do this by adding parameter to the task.

```
task("balance", "Prints an account's balance")
  .addParam("account", "The account's address")
  .setAction(async (taskArgs) => {

  });
```

Run `npx hardhat help balance`

```
E:\Projects\Hardhat>npx hardhat help balance
Hardhat version 2.11.1

Usage: hardhat [GLOBAL OPTIONS] balance --account <STRING>

OPTIONS:

  --account    The account's address

balance: Prints an account's balance

For global options help run: hardhat help
```

To get the account's balance we will use ether.js plugin, which is included in the hardhat toolbox.

```
task("balance", "Prints an account's balance")
  .addParam("account", "The account's address")
  .setAction(async (taskArgs) => {
    const balance = await ethers.provider.getBalance(taskArgs.account);

    console.log(ethers.utils.formatEther(balance), "ETH");
  });
```

A new task with its full functionality is added to the list. Finally run:

```
npx hardhat balance --account 0x...F9
```

```
E:\Projects\Hardhat>npx hardhat balance --account 0x208b2D3926022D6555a406bD0121f7Ea675C90F9
0.0 ETH
```

Run `npx hardhat accounts`:

```
E:\Projects\Hardhat>npx hardhat accounts
0xf39Fd6e51aad88F6F4ce6aB8827279cFfFb92266
0x70997970C51812dc3A010C7d01b50e0d17dc79C8
0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC
0x90F79bf6EB2c4f870365E785982E1f101E93b906
0x15d34AAf54267DB7D7c367839AAf71A00a2C6A65
0x9965507D1a55bcC2695C58ba16FB37d819B0A4dc
0x976EA74026E726554dB657fA54763abd0C3a0aa9
0x14dC79964da2C08b23698B3D3cc7Ca32193d9955
0x23618e81E3f5cdF7f54C3d65f7FBc0aBf5B21E8f
0xa0Ee7A142d267C1f36714E4a8F75612F20a79720
0xBcd4042DE499D14e55001CcbB24a551F3b954096
0x71bE63f3384f5fb98995898A86B02Fb2426c5788
0xFAB80ac9d68B0B445fB7357272Ff202C5651694a
0x1CBd3b2770909D4e10f157cABC84C7264073C9Ec
0xdF3e18d64BC6A983f673Ab319CCaE4f1a57C7097
0xcd3B766CCDd6AE721141F452C550Ca635964ce71
0x2546BcD3c84621e976D8185a91A922aE77EEc30
0xbDA5747bFD65F08deb54cb465eB87D40e51B197E
0xD2FD4581271e230360230F9337D5c0430Bf44C0
0x8626f6940E2eb28930eFb4CeF49B2d1F2C9C1199
```

2. Creating a Simple Smart Contract

Here we are going to write a simple smart contract of NFT minting. We won't go in depth of the Solidity code of the contract.

Start by creating a new file called **NFT_Minter.sol** inside the directory called **contracts**.

```

//SPDX-License-Identifier: Unlicense
pragma solidity 0.8.9;

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

contract Artwork is ERC721 {

    uint256 public tokenCounter;
    mapping (uint256 => string) private _tokenURIs;

    constructor(
        string memory name,
        string memory symbol
    ) ERC721(name, symbol) {
        tokenCounter = 0;
    }

    function mint(string memory _tokenURI) public {
        _safeMint(msg.sender, tokenCounter);
        _setTokenURI(tokenCounter, _tokenURI);

        tokenCounter++;
    }

    function _setTokenURI(uint256 _tokenId, string memory _tokenURI) internal
virtual {
    require(
        _exists(_tokenId),
        "ERC721Metadata: URI set of nonexistent token"
    ); // Checks if the tokenId exists
    _tokenURIs[_tokenId] = _tokenURI;
}

    function tokenURI(uint256 _tokenId) public view virtual override
returns(string memory) {
    require(
        _exists(_tokenId),
        "ERC721Metadata: URI set of nonexistent token"
    );
    return _tokenURIs[_tokenId];
}

}

```

.sol is used for Solidity files. Matching the file name to the contract in contains is a common practice and recommended too.

The [ERC721](#) contract includes all standard extensions ([IERC721Metadata](#) and [IERC721Enumerable](#)). That's where the [setTokenURI](#) method comes from: we use it to store an item's metadata.

Compiling the Smart Contract

To compile the smart contract run `npx hardhat compile` in your terminal.

```
E:\Projects\Hardhat>npx hardhat compile
Compiled 1 Solidity file successfully
```

The contract is successfully compiled.

Note: You might face an error

```
E:\Projects\Hardhat>npx hardhat compile
Error HH606: The project cannot be compiled, see reasons below.

The Solidity version pragma statement in these files doesn't match any of the configured compilers in your config. Change the pragma or configure additional compiler versions in your hardhat config.

  * contracts/NFT_Minter.sol (^0.7.1)

To learn more, run the command again with --verbose
Read about compiler configuration at https://hardhat.org/config
```

This error arises due to mismatch in the compiler versions specified in hardhat config and that requested in solidity file pragma statement.

Updating the version in hardhat config file to the one specified in the contract can fix the issue. Customize the solidity compiler options in your `hardhat.config.js`.

```
module.exports = {
  solidity: "0.8.9",
};
```

3. Testing the Smart Contract

Testing the contract is of crucial importance as user's money is at stake. In our automated test, we are going to use `ethers.js` to interact with the contract and `Mocha` as the test runner. It comes built-in with Hardhat, and it's used as the default network. No setup is required to use it.

Create a new file **NFT.js** inside the **test** directory.

Let's start with code:

```

const { expect } = require('chai');
const { ethers } = require("hardhat")

describe("Artwork Smart Contract Tests", function() {

  let artwork;

  this.beforeEach(async function() {
    // This is executed before each test
    // Deploying the smart contract
    const Artwork = await ethers.getContractFactory("Artwork");
    artwork = await Artwork.deploy("Artwork Contract", "ART");
  })

  it("NFT is minted successfully", async function() {
    [account1] = await ethers.getSigners();

    expect(await artwork.balanceOf(account1.address)).to.equal(0);

    const tokenURI = "https://opensea-creatures-
api.herokuapp.com/api/creature/1"
    const tx = await artwork.connect(account1).mint(tokenURI);

    expect(await artwork.balanceOf(account1.address)).to.equal(1);
  })

  it("tokenURI is set successfully", async function() {
    [account1, account2] = await ethers.getSigners();

    const tokenURI_1 = "https://opensea-creatures-
api.herokuapp.com/api/creature/1"
    const tokenURI_2 = "https://opensea-creatures-
api.herokuapp.com/api/creature/2"

    const tx1 = await artwork.connect(account1).mint(tokenURI_1);
    const tx2 = await artwork.connect(account2).mint(tokenURI_2);

    expect(await artwork.tokenURI(0)).to.equal(tokenURI_1);
    expect(await artwork.tokenURI(1)).to.equal(tokenURI_2);

  })
})

```

Run `npx hardhat test` in your terminal. You should see the following output:

```
E:\Projects\Hardhat>npx hardhat test
```

```
Lock
Deployment
  ✓ Should set the right unlockTime (3466ms)
  ✓ Should set the right owner
  ✓ Should receive and store the funds to lock
  ✓ Should fail if the unlockTime is not in the future (58ms)
Withdrawals
Validations
  ✓ Should revert with the right error if called too soon
  ✓ Should revert with the right error if called from another account (47ms)
  ✓ Shouldn't fail if the unlockTime has arrived and the owner calls it (49ms)
Events
  ✓ Should emit an event on withdrawals
Transfers
  ✓ Should transfer the funds to the owner (44ms)

Artwork Smart Contract Tests
  ✓ NFT is minted successfully (55ms)
  ✓ tokenURI is set successfully (80ms)

11 passing (4s)
```

Your code has passed successfully.

Now let's understand the testing code –

- **describe**: This is a keyword used to give a name to the set of tests that are to be performed.
- **beforeEach**: The function inside this will be executed before every test case. In our case, deployment of the NFT_Minter contract will be done here because the contract must be deployed before each test is run.
- **it**: This function takes in a title for the test and a function which runs the test case.

Note: Hardhat has its own local testnet so there is no need to run `ganache-cli` separately for tests.

We must first obtain a reference to the developed smart contract using ethers.

`getContractFactory()` before we can deploy the smart contract. The smart contract may then be deployed by calling the `deploy()` function and passing the arguments. This is done in the section `beforeEach()`.

```
let artwork;

this.beforeEach(async function() {
  // This is executed before each test
  // Deploying the smart contract
  const Artwork = await ethers.getContractFactory("Artwork");
  artwork = await Artwork.deploy("Artwork Contract", "ART");
})
```

We first obtain one of the default accounts generated by HardHat in order to verify that the NFT is correctly minted. Then, using a random tokenURI, we invoke the mint function included within the smart contract. The account balance should be 0 before minting and 1 after minting, according to our checks. The NFTs are appropriately minted if the contract satisfies the test.

```
it("NFT is minted successfully", async function() {
    [account1] = await ethers.getSigners();

    expect(await artwork.balanceOf(account1.address)).to.equal(0);

    const tokenURI = "https://opensea-creatures-
api.herokuapp.com/api/creature/1"
    const tx = await artwork.connect(account1).mint(tokenURI);

    expect(await artwork.balanceOf(account1.address)).to.equal(1);
})
```

We choose two random tokenURIs and set them from various accounts in order to determine whether a tokenURI is set correctly. To check that the tokenURIs are set correctly, we next call the tokenURI() function to retrieve the tokenURI for each individual token and compare it to the given input.

```
it("tokenURI is set sucessfully", async function() {
    [account1, account2] = await ethers.getSigners();

    const tokenURI_1 = "https://opensea-creatures-
api.herokuapp.com/api/creature/1"
    const tokenURI_2 = "https://opensea-creatures-
api.herokuapp.com/api/creature/2"

    const tx1 = await artwork.connect(account1).mint(tokenURI_1);
    const tx2 = await artwork.connect(account2).mint(tokenURI_2);

    expect(await artwork.tokenURI(0)).to.equal(tokenURI_1);
    expect(await artwork.tokenURI(1)).to.equal(tokenURI_2);

})
```

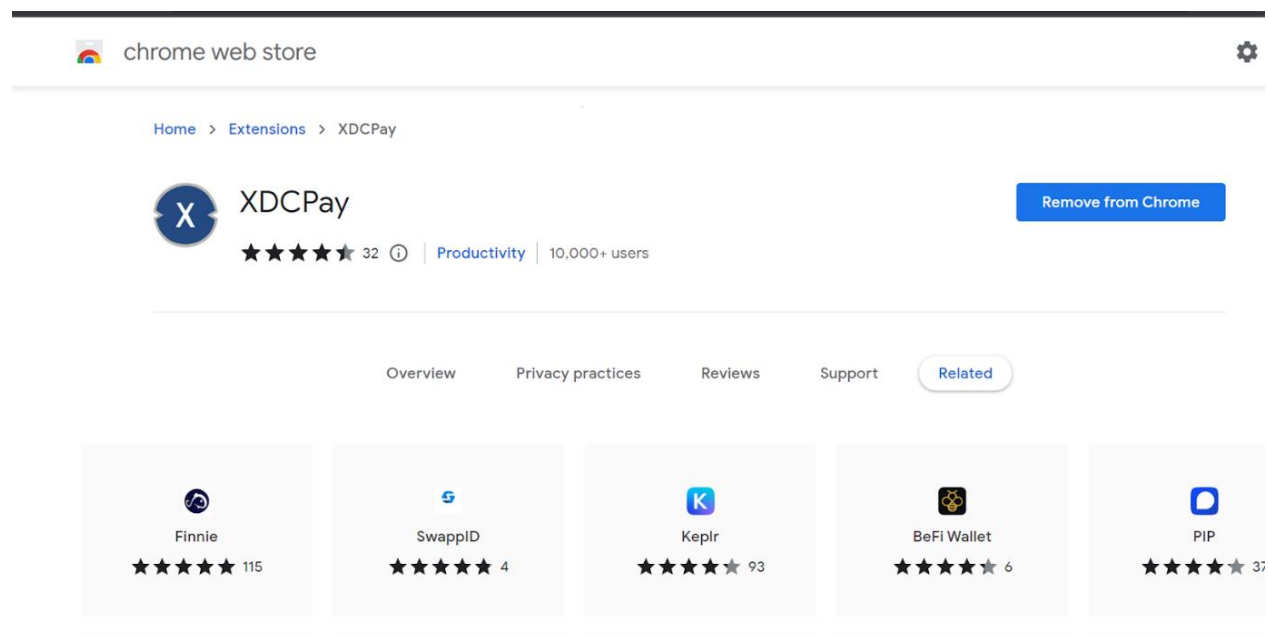
Note: Hardhat Network provides `console.log()` to print logging messages and contract variable while running your contracts and tests. The only requirement is to import `hardhat/console.sol` in your contract.

```
pragma solidity 0.8.9;

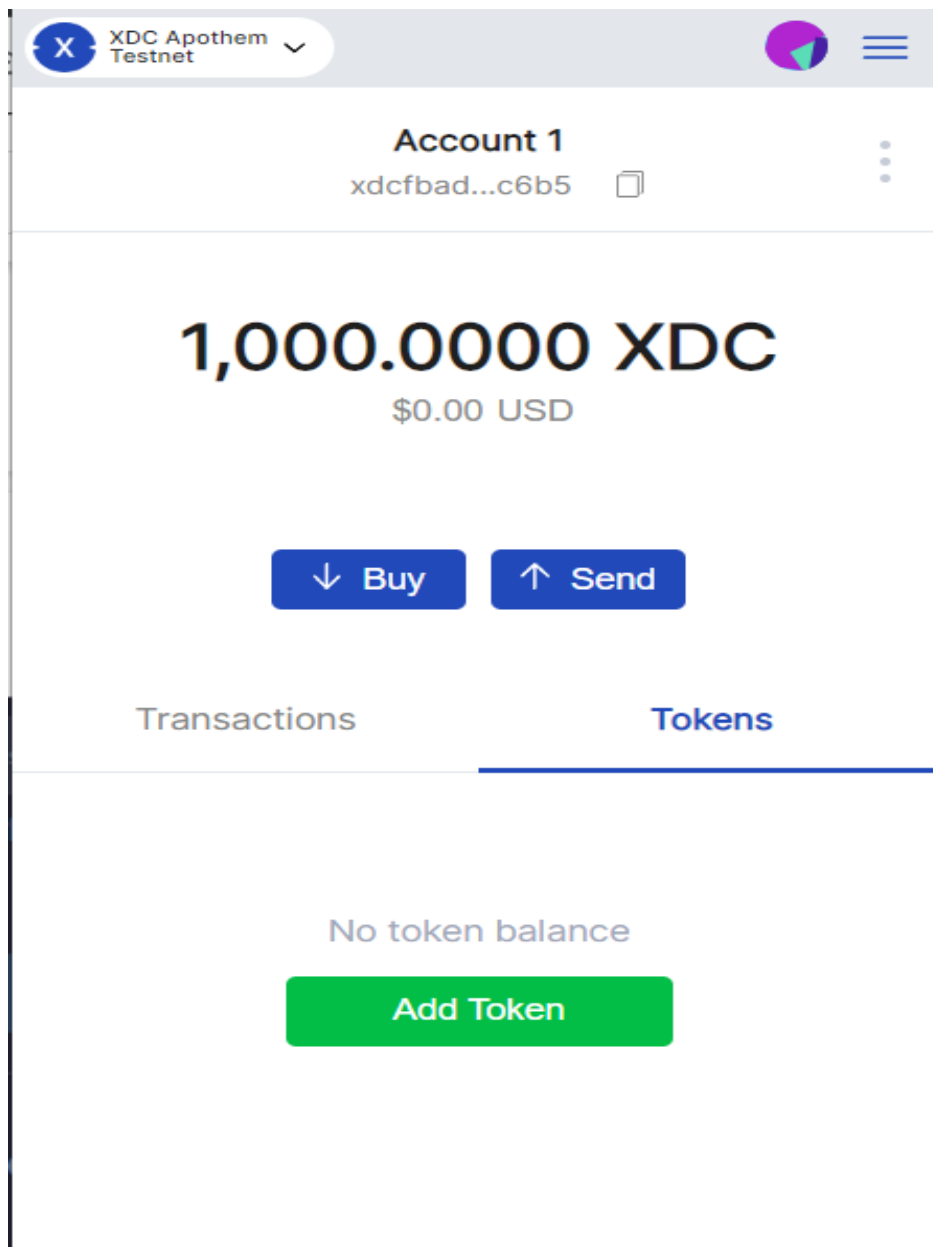
import "hardhat/console.sol";
contract Artwork is ERC721 {
  //...
}
```

4. Deploying on XDC Network using remix

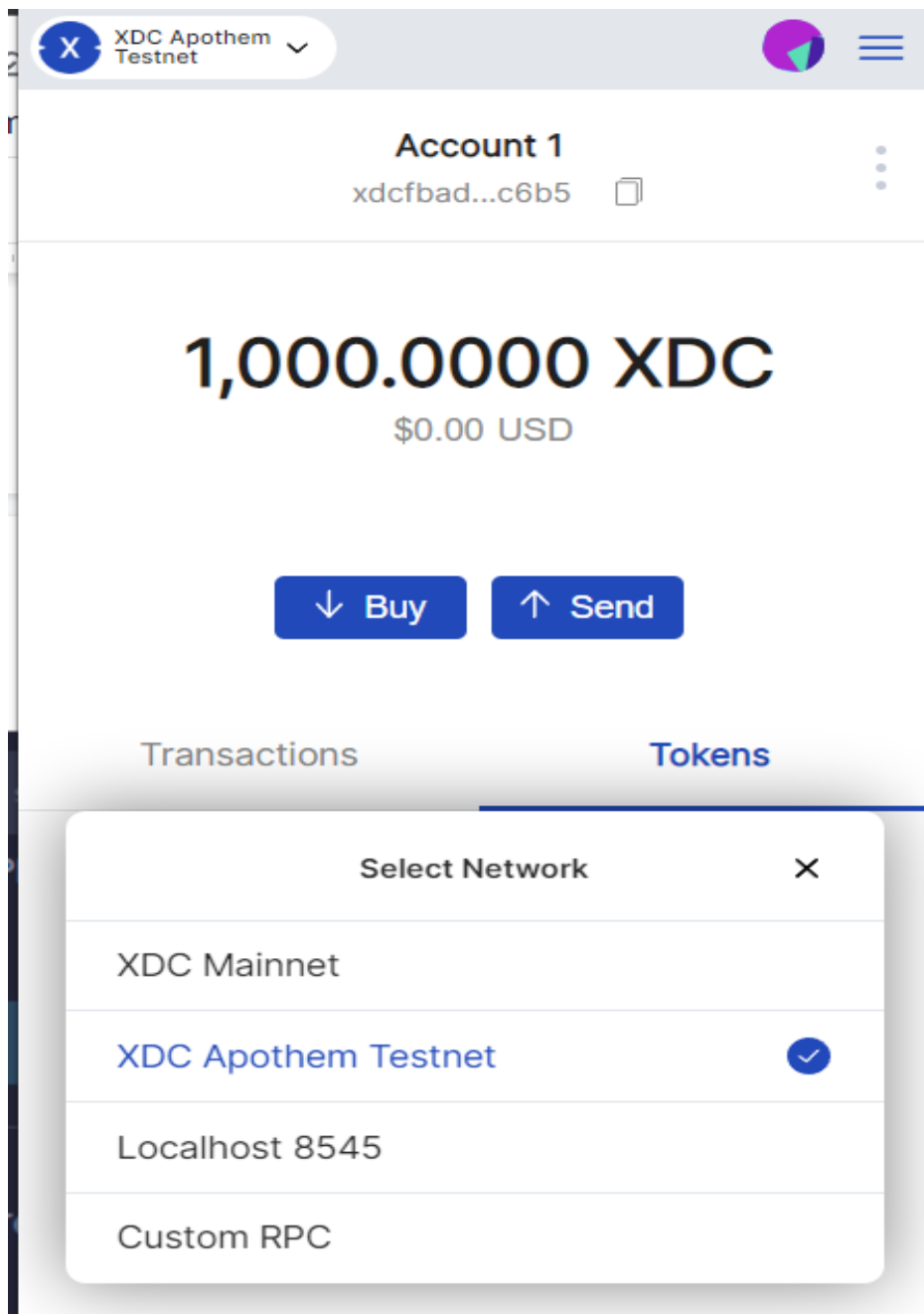
1. Search for XDCPay chrome extension(<https://chrome.google.com/webstore/detail/xdcpay/bocpokimicclpaiekenaeelehdjillofo/related>) and add it in the browser.



2. Create a wallet in XDC Pay. After creating and confirming you'll be given secret backup phrase so that you can restore your account even if you forget login credentials.



3. Select the XDC Apothem Testnet to test the smart contract before deploying it on mainnet.




4. Add test XDC to your wallet using Apothem Faucet (<https://faucet.apothem.network/>)

← → × 🏠 🔒 faucet.apothem.network

XDC APOTHEM NETWORK FAUCET

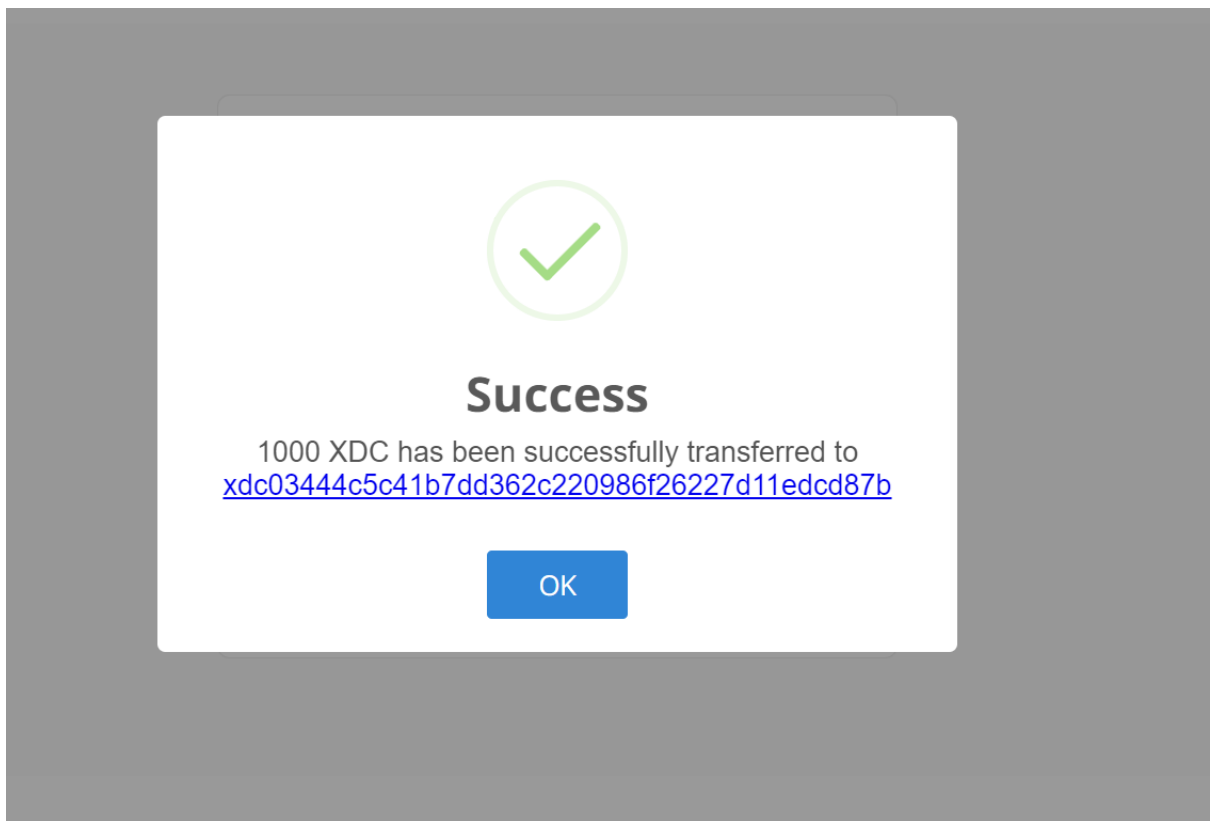
Enter your XDC address

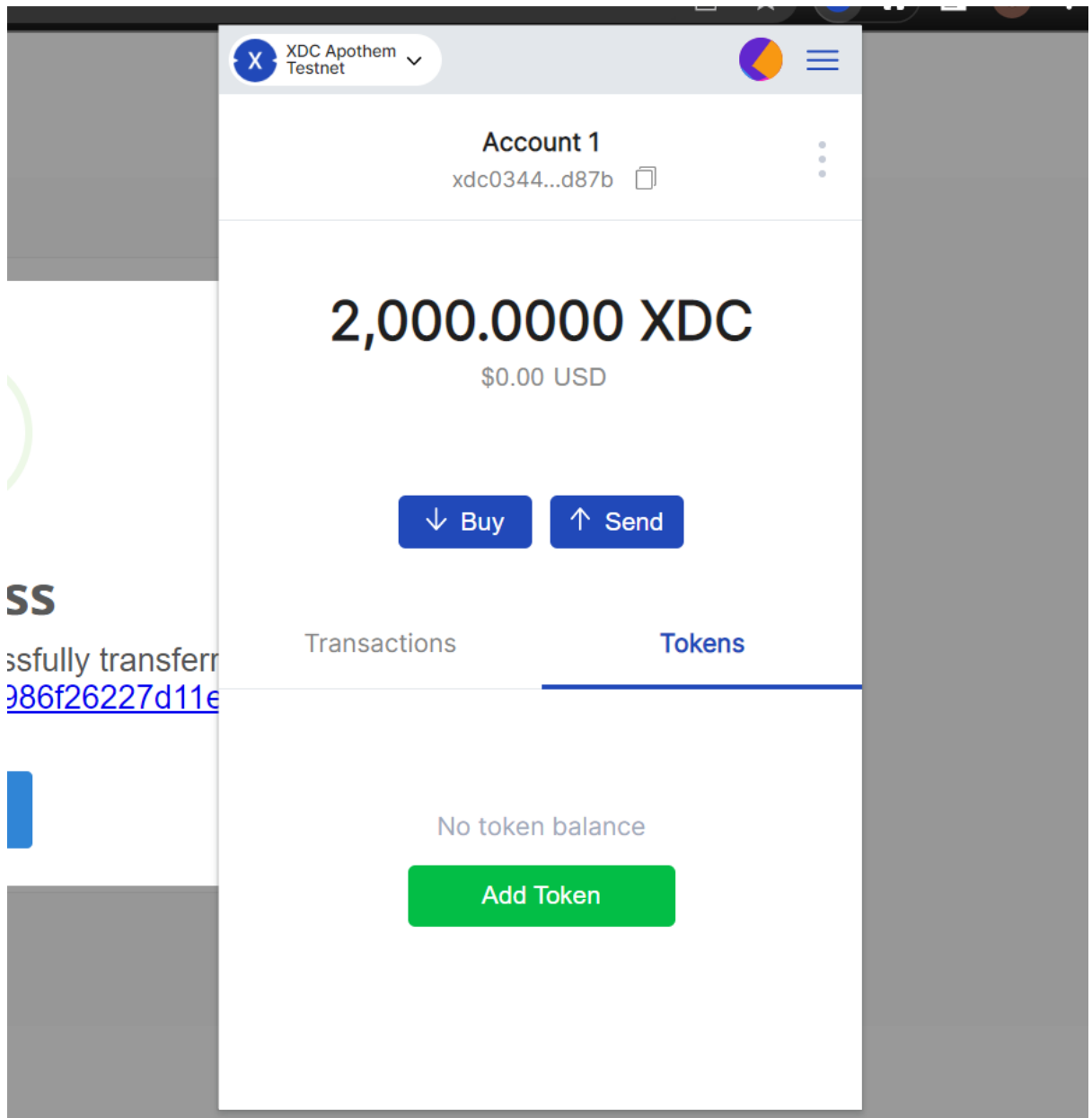
Please use <https://rpc.apothem.network> RPC to connect to the XDC Apothem testnet.

☐ I'm not a robot 
reCAPTCHA
Privacy - Terms

REQUEST 1000 XDC

After adding a pop-up window like this will appear on your screen





- Now open Remix IDE Editor(<https://remix.ethereum.org/#optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.9+commit.e5eed63a.js>). Create a .sol extension file for solidity. We will be writing a smart contract code to mint NFTs

```
//SPDX-License-Identifier: Unlicense
pragma solidity 0.8.9;

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

contract Artwork is ERC721 {

    uint256 public tokenCounter;
    mapping (uint256 => string) private _tokenURIs;
```

```

constructor(
    string memory name,
    string memory symbol
) ERC721(name, symbol) {
    tokenCounter = 0;
}

function mint(string memory _tokenURI) public {
    _safeMint(msg.sender, tokenCounter);
    _setTokenURI(tokenCounter, _tokenURI);

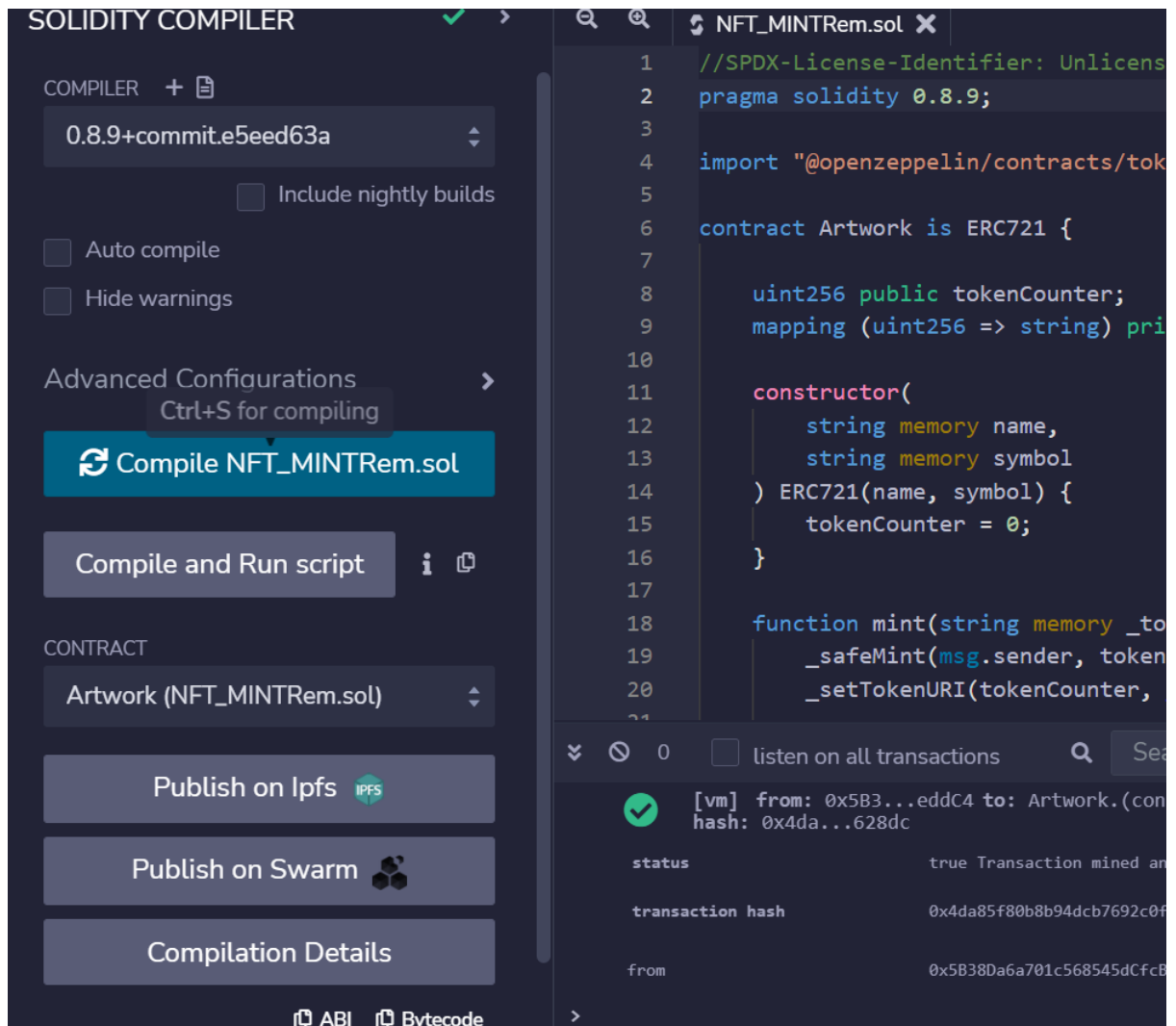
    tokenCounter++;
}

function _setTokenURI(uint256 _tokenId, string memory _tokenURI) internal
virtual {
    require(
        _exists(_tokenId),
        "ERC721Metadata: URI set of nonexistent token"
    ); // Checks if the tokenId exists
    _tokenURIs[_tokenId] = _tokenURI;
}

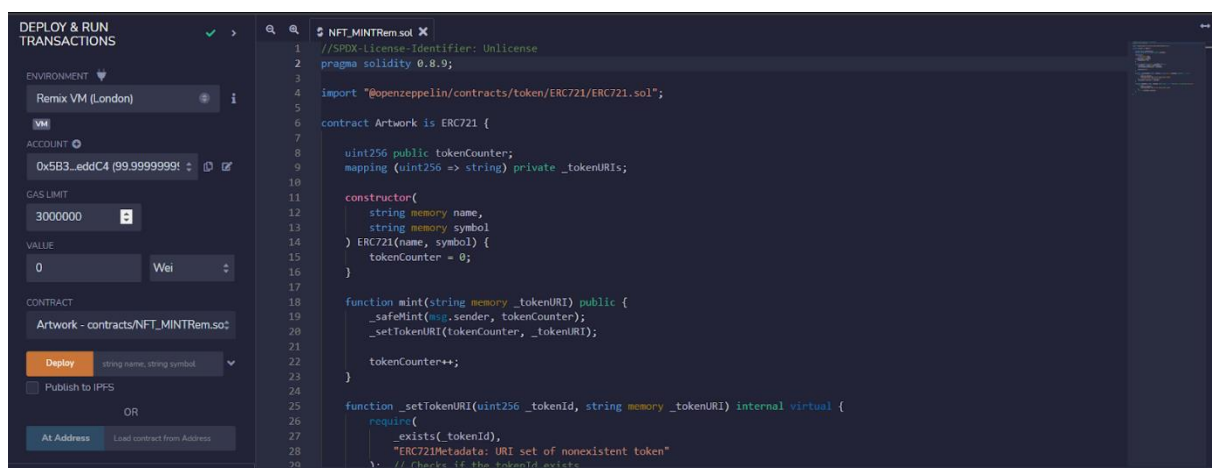
function tokenURI(uint256 _tokenId) public view virtual override returns(string
memory) {
    require(
        _exists(_tokenId),
        "ERC721Metadata: URI set of nonexistent token"
    );
    return _tokenURIs[_tokenId];
}
}

```

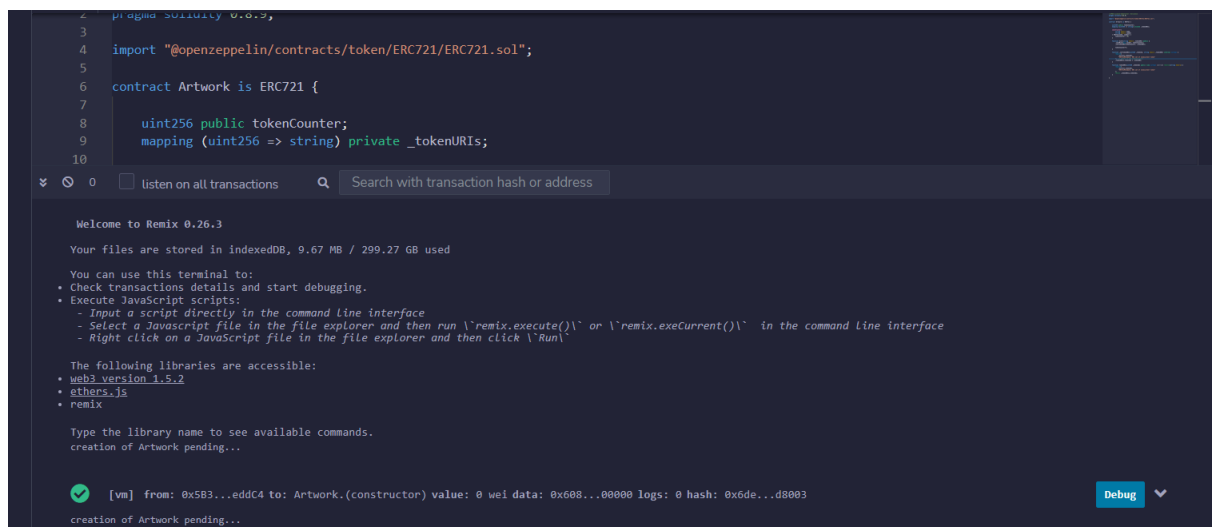
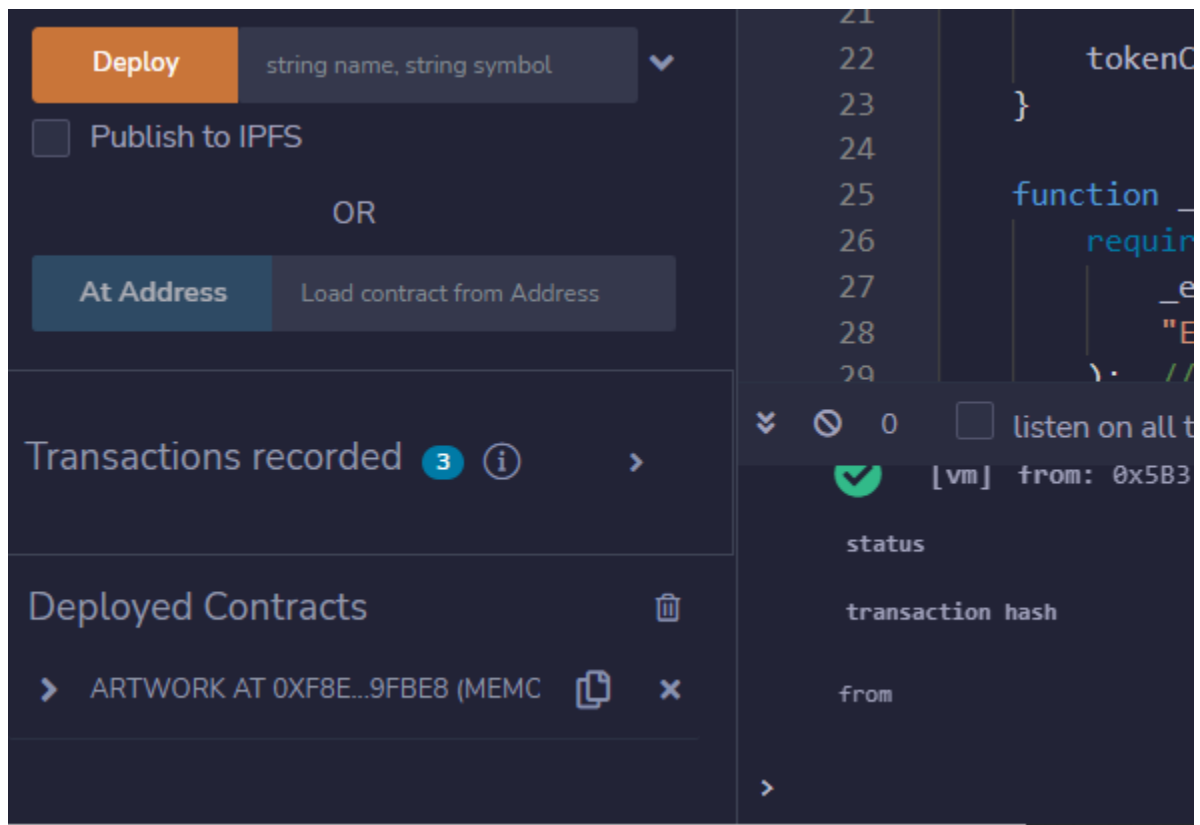
6. Now after this compile your code using Ctrl+S



After compiling, deploy your code at the XDC Apothem Testnet by clicking on the left navigation pane and there you will be able to see the deploy button



After deploying you will be able to view the deployed contract in the left nav pane and the details in the bottom.



After the transaction is complete open XDC Pay and go to the add token window and then add your token id and press add.

The screenshot shows a mobile application interface for the 'XDC Apothem Testnet'. At the top, there is a header bar with a blue circular icon containing a white 'X' on the left, the text 'XDC Apothem Testnet' in the center, and a small downward arrow on the right. On the far right of the header is a circular profile picture and a blue hamburger menu icon. Below the header, a modal dialog titled 'Add Token' is displayed. The dialog has a close button (an 'X' icon) in the top left corner. It contains three input fields: 'Token Address' with the placeholder text 'Token Contract Address', 'Token Symbol' with the placeholder text 'Like 'XDC'', and 'Decimals of Precision' with the value '0'. At the bottom of the dialog is a large green button labeled 'Add'. The background of the app is a light gray, and a portion of a purple sidebar is visible on the right edge.

XDC Apothem Testnet

Add Token

Token Address

Token Contract Address

Token Symbol

Like 'XDC'

Decimals of Precision

0

Add

A complete video guide to go through these steps is provided in the link given below:

<https://drive.google.com/drive/folders/1U4aV3DHuddnQy23XY3iwZd3ekUnNnrVU?usp=sharing>

5. Deploying the Smart Contract to remote networks

You might wish to deploy your dApp to a live network once you're ready to share it with others. In this way, instances that aren't operating locally on your computer can be accessed by other people.

There are distinct "testnet" networks that do not deal in real money, but the Ethereum "mainnet" network does. Several of Ethereum's testnets, such as Goerli and Sepolia, offer shared staging environments that effectively simulate the real world scenario without putting actual money at risk. You should launch your contracts on the Goerli testnet, as advised.

Here we are launching our contract on Matic Testnet.

Create a new file in the **scripts** directory called **deploynft.js**:

```
async function main() {
  const [deployer] = await ethers.getSigners();

  console.log("Deploying contracts with the account:", deployer.address);

  console.log("Account balance:", (await deployer.getBalance()).toString());

  const Artwork = await hre.ethers.getContractFactory("Artwork");
  const artwork = await Artwork.deploy("Artwork Contract", "ART");

  await artwork.deployed();

  hre.run("verify:verify", {
    address: artwork.address,
    constructorArguments: [
      "Artwork Contract",
      "ART"
    ]
  })

  console.log("NFT address:", artwork.address);
}

main()
  .then(() => process.exit(0))
  .catch((error) => {
    console.error(error);
    process.exit(1);
  });
```

Run the above code without the `--network` parameter. The code would execute against an embedded instance of Hardhat Network in our present setup. Even if the deployment gets lost in this case after Hardhat completes its execution, it's still helpful to validate that our deployment code functions as intended:

```
npx hardhat run scripts/deploynft.js
```

```
E:\Projects\Hardhat>npx hardhat run scripts/deploynft.js
Deploying contracts with the account: 0xf39Fd6e51aad88F6F4ce6aB8827279cfffB92266
Account balance: 10000000000000000000000
NFT address: 0x5FbDB2315678afecb367f032d93F642f64180aa3
```

When executing any job, you may use the `--network` argument to instruct Hardhat to connect to a particular Ethereum network, like in the following example:

```
module.exports = {
  solidity: "0.8.9",
  networks: {
    xdc: {
      url: "https://rpc.xinfin.network",
      accounts: [
        process.env.PRIVATE_KEY,
      ]
    },
  },
  etherscan: {
    apiKey: process.env.POLYGONSCAN_KEY,
  }
};
```

Since we are going to deploy our contract to Matic testnet, we created a new entry in the network section and added the RPC url to Matic testnet along with the private key of the account which will be used to deploy the smart contract.

Run the command `npx hardhat run scripts/deploynft.js --network matic` specifying the network parameter that you will provide.

Change the Metamask's network to Matic testnet before transacting.

```
E:\Projects\Hardhat>npx hardhat run scripts/deploynft.js --network matic
Deploying contracts with the account: 0x208b2D3926022D6555a406bD0121f7Ea675C90F9
Account balance: 166963958027187059
NFT address: 0x58e8BD7d19aCE9b4fD2fcAf1aDD066780703Cb92
```

Here you can see the deployed NFT contract address.