How to Build a NFT Marketplace DApp on Ethereum or Optimism¶

Written byEmily Lin

Overview¶

Today, we're gonna build Bored Pets Marketplace, a simple NFT marketplace usingTruffle ,Infura ,MetaMask , andWeb3.js ! We'll start off with the basics and first show you how to deploy your ethereum contracts to Truffle's local blockchain, Ganache, and the Goerli testnet. Then, we'll show you how to covert those contracts over to Optimism and deploy on the Optimistic Goerli testnet. Note that this tutorial will not teach you Javascript and will only briefly go over some Solidity principles.

Our marketplace has a core set of functionality: 1. Minting and listing an NFT 2. Buying and selling an NFT 3. Viewing listed NFTs, NFTs you own, and NFTs you are selling

This tutorial was inspired by Nader Dabit'smarketplace tutorial . His tutorial usesethers.js and deploys to Polygon, both of which you can use Truffle with!

The completed code for this tutorial liveshere .

Prerequisites¶

System Requirements¶

We will start with Truffle'sOptimism box , scaffolding code to help get your Truffle project configured to start using Optimism asap! TheREADME explains the project structure in more detail.

At the very least, you'll need to install:

- Node.js
- , v12 or higher
- truffle
- ganache

If you want to run Optimism locally, you'll need thesepreqrequisites too.

Create an Infura account and project¶

To connect your DApp to Ethereum, Optimism, and other networks, you'll need an Infura account. Sign up for an accounthere .

Once you're signed in, create a project! Let's call itnft-marketplace . Since we're deploying to Optimism, go ahead and add the Optimistic Goerli endpoint. It'll ask you to sign up for a free trial.

Register for a MetaMask wallet¶

To interact with your DApp in the browser, you'll need a MetaMask wallet. Sign up for an accounthere .

Add Optimistic Goerli to your wallet¶

Let's add Optimistic Goerli to your list of available networks to your MetaMask wallet! To do so, open up the MetaMask extension, click on the network, and then click Add Network. Then, fill out the network properties (you can copy your Infura idf from your Infura project):

- Network Name: Optimistic Goerli
- New RPC URL: https://optimism-goerli.infura.io/v3/INFURA_PROJECT_ID
- Chain ID: 420
- Currency Symbol: ETH
- Block Explorer URL: https://blockscout.com/optimism/goerli/

Get Optimistic Goerli Eth¶

To use the Optimistic Goerli testnet, you'll need some test eth. To do so, you'll need to:

1. Get some Goerli Eth. You can use thisfaucet

2. , which will also give some Optimistic Goerli Eth
3. Get some Optimstic Goerli Eth by bridging through their app: https://app.optimism.io/bridge/deposit

VSCode¶

Feel free to use whatever IDE you want, but we highly recommend using VSCode! You can run through most of this tutorial using the Truffle extension to create, build, and deploy your smart contracts, all without using the CLI! You can read more about ithere .

Set Up Your Project¶

Unbox Optimism¶

To get started, we'll start off by unboxing Truffle's Optimism box. You can do this by callingtruffle unbox optimism [FOLDER_NAME] or through the VSCode extension command palette! Go ahead and install the requisite packages as well.

truffle unbox optimism nft-marketplacecd

nft-marketplace

Create your client folders¶

We'll also be usingNext.js andTailwind CSS for our client. So, to get that set up, we'll use Next'screate-next-app utility.

npx create-next-app client Then, we'll download the Tailwind dependencies and populate its config.

cd

client npm install -D tailwindcss postcss autoprefixer npx tailwindcss init -p Edittailwind.config.js

module . exports

=

{

content :

[

"./pages/*/.{js,ts,jsx,tsx}" ,

"./components/*/.{js,ts,jsx,tsx}" ,

],

theme :

{

extend :

{},

},

plugins :

[], } And replace the code fromstyles/global.css

@ tailwind

base ; @ tailwind

components ; @ tailwind

utilities ;

Edit the Truffle config files¶

Since we'll be referencing your compiled contracts in the client, let's changecontracts_build_directory in the Truffle config files:

- Intruffle-config.js
- , change the value to'./client/contracts/ethereum-contracts'
- .
- Intruffle-config.ovm.js
- , change the value to'./client/contracts/optimism-contracts'
- .

Your top level folder structure should look like this!

nft-marketplace ├── LICENSE ├── README.md ├── box-img-lg.png ├── box-img-sm.png ├── client ├── contracts ├── migrations ├── node_modules ├── package-lock.json ├── package.json ├── test ├── truffle-config.js └── truffle-config.ovm.js

Build the NFT Smart Contract¶

Since we are first showing you how to deploy on Ethereum, we'll edit theSimpleStorage.sol contract undercontracts/ethereum . Change the file name and contract name toBoredPetsNFT.sol .

We'll need to installOpenZeppelin . First, switch back into yournft-marketplace directory

cd

.. npm install @openzeppelin/contracts The smart contract looks like this:

// SPDX-License-Identifier: MIT pragma

solidity

^ 0.8.13 ; import

"@ openzeppelin / contracts / token / ERC721 / ERC721 . sol "; import " @openzeppelin / contracts / token / ERC721 / extensions / ERC721URIStorage . sol "; import " @openzeppelin / contracts / utils / Counters . sol "; contract BoredPetsNFT is ERC721URIStorage { using Counters for Counters.Counter; Counters.Counter private _tokenIds; address marketplaceContract; event NFTMinted(uint256); constructor(address _marketplaceContract) ERC721(" Bored

Pets

Yacht

Club ", " BPYC ") { marketplaceContract = _marketplaceContract; } function mint(string memory _tokenURI) public { _tokenIds.increment(); uint256 newTokenId = _tokenIds.current(); _safeMint(msg.sender, newTokenId); _setTokenURI(newTokenId, _tokenURI); setApprovalForAll(marketplaceContract, true); emit NFTMinted(newTokenId); } } Creating the NFT smart contract is short and sweet! Let's first take a look at the imports:

- @openzeppelin/contracts/token/ERC721/ERC721.sol

To be a valid NFT,BoredPetNFT implements theERC721 standard by inheriting the implementation ofERC721URIStorage.sol

- @openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol

This implementation ofERC721 is used so that we store the tokenURIs on chain instorage , which is what allows us to store the metadata we upload to IPFS off-chain.

- @openzeppelin/contracts/utils/Counters.sol

We use a counter to track the total number of NFTs and assign a unique token id to each NFT.

At the top we define a few variables:

- address marketplaceContract
- is the address of the Marketplace contract we'll be writing in the next section.
- event NFTMinted
- will be emitted every time a NFT is minted. When an event is emitted in solidity, the parameters are stored in the transaction's log. We will need the tokenId later when we build out the web app.

And finally, we only need to define themint function! It only has one parameter: -string memory _tokenURI points to the JSON metdata on IPFS that stores the NFT's metadata (i.e., image, name, description)

mint is relatively straightforward - it mints an NFT with an increasing, unique token Id. Of note, however, issetApprovalForAll

. This is important because our Marketplace contract will need approver access to transfer ownership of the NFT between various addresses.

Build the Marketplace Contract¶

Now, let's add a new contractMarketplace.sol undercontracts/ethereum , which will store all the marketplace functionality. This'll be what it ends up like:

// SPDX-License-Identifier: MIT pragma

solidity

^ 0.8.13 ; import

"@ openzeppelin / contracts / utils / Counters . sol "; import " @openzeppelin / contracts / token / ERC721 / ERC721 . sol "; import " @openzeppelin / contracts / security / ReentrancyGuard . sol "; contract Marketplace is ReentrancyGuard { using Counters for Counters.Counter; Counters.Counter private _nftsSold; Counters.Counter private _nftCount; uint256 public LISTING_FEE = 0.0001 ether; address payable private _marketOwner; mapping(uint256 => NFT) private _idToNFT; struct NFT { address nftContract; uint256 tokenId; address payable seller; address payable owner; uint256 price; bool listed; } event NFTListed(address nftContract, uint256 tokenId, address seller, address owner, uint256 price); event NFTSold(address nftContract, uint256 tokenId, address seller, address owner, uint256 price); constructor() { _marketOwner = payable(msg.sender); } // List the NFT on the marketplace function listNft(address _nftContract, uint256 _tokenId, uint256 _price) public payable nonReentrant { require(_price > 0, " Price

must

be

at

least

1

wei "); require(msg.value == LISTING_FEE, " Not

enough

ether

for

listing

fee "); IERC721(_nftContract).transferFrom(msg.sender, address(this), _tokenId); _marketOwner.transfer(LISTING_FEE); _nftCount.increment(); _idToNFT[_tokenId] = NFT(_nftContract, _tokenId, payable(msg.sender), payable(address(this)), _price, true); emit NFTListed(_nftContract, _tokenId, msg.sender, address(this), _price); } // Buy an NFT function buyNft(address _nftContract, uint256 _tokenId) public payable nonReentrant { NFT storage nft = _idToNFT[_tokenId]; require(msg.value >= nft.price, " Not

enough

ether

to

cover

asking

price "); address payable buyer = payable(msg.sender); payable(nft.seller).transfer(msg.value); IERC721(_nftContract).transferFrom(address(this), buyer, nft.tokenId); nft.owner = buyer; nft.listed = false; _nftsSold.increment(); emit NFTSold(_nftContract, nft.tokenId, nft.seller, buyer, msg.value); } // Resell an NFT purchased from the marketplace function resellNft(address _nftContract, uint256 _tokenId, uint256 _price) public payable nonReentrant { require(_price > 0, " Price

must

be

at

least

1

wei "); require(msg.value == LISTING_FEE, " Not

enough

ether

for

listing

fee "); IERC721(_nftContract).transferFrom(msg.sender, address(this), _tokenId); NFT storage nft = _idToNFT[_tokenId]; nft.seller = payable(msg.sender); nft.owner = payable(address(this)); nft.listed = true; nft.price = _price; _nftsSold.decrement(); emit NFTListed(_nftContract, _tokenId, msg.sender, address(this), _price); } function getListedNfts() public view returns (NFT[] memory) { uint256 nftCount = _nftCount.current(); uint256 unsoldNftsCount = nftCount - _nftsSold.current(); NFT[] memory nfts = new NFT; uint nftsIndex = 0; for (uint i = 0; i < nftCount; i++) { if (_idToNFT[i + 1].listed) { nfts[nftsIndex] = _idToNFT[i + 1]; nftsIndex++; } } return nfts; } function getMyNfts() public view returns (NFT[] memory) { uint nftCount = _nftCount.current(); uint myNftCount = 0; for (uint i = 0; i < nftCount; i++) { if (_idToNFT[i + 1].owner == msg.sender) { myNftCount++; } } NFT[] memory nfts = new NFT; uint nftsIndex = 0; for (uint i = 0; i < nftCount; i++) { if (_idToNFT[i + 1].owner == msg.sender) { nfts[nftsIndex] = _idToNFT[i + 1]; nftsIndex++; } } return nfts; } function getMyListedNfts() public view returns (NFT[] memory) { uint nftCount = _nftCount.current(); uint myListedNftCount = 0; for (uint i = 0; i < nftCount; i++) { if (_idToNFT[i + 1].seller == msg.sender && _idToNFT[i + 1].listed) { myListedNftCount++; } } NFT[] memory nfts = new NFT; uint nftsIndex = 0; for (uint i = 0; i < nftCount; i++) { if (_idToNFT[i + 1].seller == msg.sender && _idToNFT[i + 1].listed) { nfts[nftsIndex] = _idToNFT[i + 1]; nftsIndex++; } } return nfts; } } There's a lot to unpack here! You might notice that the Marketplace contract inheritsReentrancyGuard . We do this so that we can defend against reentrancy attacks. You can read more about themhere .

First, let's dive into the contract variables: -Counters.Counter private _nftsSold increments when a NFT is sold and decremented when a NFT is relisted. -Counters.Counter private _nftCount tracks how many NFTs have been listed. -uint256 public LISTING_FEE is taken from the seller and transferred to the marketplace contract owner whenever an NFT is sold. -address payable private _marketOwner stores the Marketplace contract owner, so that we know who to pay the listing fee to. -mapping(uint256 => NFT) private _idToNFT associates the unique tokenId to a the NFT struct. -struct NFT stores relevant information for an NFT listed in the marketplace. -event NFTListed is emitted every time a NFT is listed. -event NFTSold is emitted every time a NFT is sold.

Next, we'll go over the functions that will alter state: -listNft is called when a user first mints and lists their NFT. It transfers ownership from the user over to the Marketplace contract. -resellNft allows users to sell an NFT that they purchase on the marketplace. -buyNft is called when a user buys an NFT. The buyer becomes the new owner of the NFT, the token is transferred from the buyer to the seller, and the listing fee is given to the marketplace owner.

Finally, the query functions are relatively straightforward:

- getListedNfts
- retrieves the NFTs that are currently listed for sale.
- getMyNfts
- retrieves the NFTs the user has bought.
- getMyListedNfts
- retrieves the NFTs the user has listed for sale.

Deploy the Smart Contracts Locally¶

In order to deploy our smart contracts, we'll need to modifymigrations/1_deploy_contracts.js . BecauseBoredPetsNFT requires theMarketplace contract address, order matters here! Truffle allows you to deploy contracts in order usingPromise orawait /async . You can read more about ithere . (Note that previously, Truffle also required a separateMigrations.sol contract, but that is no longer the case!)

Your deployment file should look like this:

var

BoredPetsNFT

=

artifacts . require ("BoredPetsNFT"); var

Marketplace

=

artifacts . require ("Marketplace"); module . exports

=

async

function (deployer)

{

await

deployer . deploy (Marketplace);

const

marketplace

=

await

Marketplace . deployed ();

await

deployer . deploy (BoredPetsNFT ,

marketplace . address); } There are a variety of ways to get your local Ganache instance up: through the VS Code extension, Ganache CLI, and the Ganche graphical user interface. Each has its own advantages, and you can check out v7's coolest featureshere .

In this example, we'll be using the GUI. Open it up, create a workspace, and hit save!

This creates a running Ganache instance at HTTP://127.0.0.1:7545. You'll need to edit thedevelopment network in yourtruffle-config.js to match the port number. Now, just runtruffle migrate in the CLI from thenft-marketplace folder, which will default to thedevelopment network. This will compile and deploy your contracts. You should see output similar to this:

```
Compiling your contracts...===========================

    Compiling ./contracts/ethereum/BoredPetsNFT.sol.sol

    Compiling ./contracts/ethereum/Marketplace.sol

    Compiling @openzeppelin/contracts/security/ReentrancyGuard.sol

    Compiling @openzeppelin/contracts/token/ERC721/ERC721.sol

    Compiling @openzeppelin/contracts/token/ERC721/IERC721.sol

    Compiling @openzeppelin/contracts/token/ERC721/IERC721Receiver.sol

    Compiling @openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol

    Compiling @openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol

    Compiling @openzeppelin/contracts/utils/Address.sol

    Compiling @openzeppelin/contracts/utils/Context.sol

    Compiling @openzeppelin/contracts/utils/Counters.sol

    Compiling @openzeppelin/contracts/utils/Strings.sol

    Compiling @openzeppelin/contracts/utils/introspection/ERC165.sol

    Compiling @openzeppelin/contracts/utils/introspection/IERC165.sol

    Artifacts written to /Users/emilylin/truffle/nft-marketplace/client/contracts/ethereum-contracts

    Compiled successfully using: - solc: 0 .8.13+commit.abaa5c0e.Emscripten.clang
```

```
Starting migrations...======================

     Network name: 'development'

     Network id: 5777

     Block gas limit: 6721975

( 0x6691b7) 1_deploy_contracts.js=====================

Deploying 'Marketplace'
```

```
     transaction hash: 0xca3f5fcc301c700bdfd7bfb58c853e39085335ed0ef249bf57334ad856848383

     Blocks: 0

Seconds: 0

     contract address: 0x6F1A6D8a5414d1B1E67b69D69D5206498447aceE

     block number: 1

     block timestamp: 1652487728

     account: 0x24D126143330E7f76753e13FAE39a844cbcEe829

     balance: 99 .96632092

     gas used: 1683954

( 0x19b1f2)

     gas price: 20

gwei

     value sent: 0

ETH

     total cost: 0 .03367908 ETH Deploying 'BoredPetsNFT'
```

```
     transaction hash: 0x077ab3a709addd41229de58f3bb6ad913a5c4646cd5df39272a9db6ad3a04f05

     Blocks: 0

Seconds: 0

     contract address: 0x2935aBf19126137D47bCa1612Dc4900815A15E92

     block number: 2

     block timestamp: 1652487729

     account: 0x24D126143330E7f76753e13FAE39a844cbcEe829

     balance: 99 .91495866

     gas used: 2568113

( 0x272fb1)

     gas price: 20

gwei

     value sent: 0

ETH

     total cost: 0 .05136226 ETH
```

Saving artifacts

Total cost: 0 .08504134 ETHSummary =======

Total deployments: 2

Final cost: 0 .08504134 ETH You should be able to find your compiled contracts under./client/contracts/ethereum-contracts

Write a script¶

To perform common actions, we'll take advantage oftruffle exec to run scripts to automate common tasks. Let's write a script that will execute all of our different functions. First, create a new file under a newscripts folder calledrun.js .

var

BoredPetsNFT

=

artifacts . require ("BoredPetsNFT"); var

Marketplace

=

artifacts . require ("Marketplace"); async

function

logNftLists (marketplace)

{

let

listedNfts

=

await

marketplace . getListedNfts ()

const

accounts

=

await

web3 . currentProvider . request ({

method :

'eth_accounts' ,

params :

[],

});

const

accountAddress

=

accounts [0];

```
let
myNfts
=
await
marketplace . getMyNfts ({ from :
accountAddress })
let
myListedNfts
=
await
marketplace . getMyListedNfts ({ from :
accountAddress })
console . log (listedNfts: { listedNfts . length } )
console . log (myNfts: { myNfts . length } )
console . log (myListedNfts { myListedNfts . length } \n ) } const
main
=
async
( cb )
=>
{
try
{
const
boredPets
=
await
BoredPetsNFT . deployed ()
const
marketplace
=
await
Marketplace . deployed ()
console . log ( 'MINT AND LIST 3 NFTs' )
let
listingFee
=
await
marketplace . LISTING_FEE ()
listingFee
=
listingFee . toString ()
let
txn1
=
await
boredPets . mint ( "URI1" )
let
tokenId1
=
txn1 . logs [ 2 ]. args [ 0 ]. toNumber ()
await
marketplace . listNft ( boredPets . address ,
tokenId1 ,
1 ,
{ value :
listingFee })
console . log (Minted and listed { tokenId1 } )
let
txn2
=
await
boredPets . mint ( "URI1" )
let
tokenId2
=
txn2 . logs [ 2 ]. args [ 0 ]. toNumber ()
await
marketplace . listNft ( boredPets . address ,
tokenId2 ,
1 ,
{ value :
listingFee })
console . log (Minted and listed { tokenId2 } )
```

```
let
txn3
=
await
boredPets . mint ( "URI1" )
let
tokenId3
=
txn3 . logs [ 2 ]. args [ 0 ]. toNumber ()
await
marketplace . listNft ( boredPets . address ,
tokenId3 ,
1 ,
{ value :
listingFee })
console . log (Minted and listed { tokenId3 }  )
await
logNftLists ( marketplace )
console . log ( 'BUY 2 NFTs' )
await
marketplace . buyNft ( boredPets . address ,
tokenId1 ,
{ value :
1 })
await
marketplace . buyNft ( boredPets . address ,
tokenId2 ,
{ value :
1 })
await
logNftLists ( marketplace )
console . log ( 'RESELL 1 NFT' )
await
marketplace . resellNft ( boredPets . address ,
tokenId2 ,
1 ,
{ value :
```

listingFee })

await

logNftLists (marketplace)

}

catch (err)

{

console . log ('Doh! ' ,

err);

}

cb (); } module . exports

=

main ; In this script, you can use artifacts.require to gain access to the contract abstractions. Then, we interact with the contracts using the @truffle/contracts convenience library. You can use this functionality to write unit tests in Truffle in javasacript or typescript. Note that if you use typescript, you'll need to create atsconfig.json file and usetsc to compile down to javascript. You can read more abouttsc here .

Runtruffle exec scripts/run.js , and your output should look what's below:

MINT AND LIST 3

NFTs listedNfts: 3 myNfts: 0 myListedNfts 3 BUY 2

NFTs listedNfts: 1 myNfts: 2 myListedNfts 1 RESELL 1

NFT listedNfts: 2 myNfts: 1 myListedNfts 1 Success! If you want to deploy your contracts on a populated blockchain, you can use Ganache to fork mainnet with zero config .

Test your smart contracts¶

Let's test our Marketplace contract! To do so, call:

truffle create test

Marketplace Then, add this code:

require ("@openzeppelin/test-helpers/configure")({

provider :

web3 . currentProvider ,

singletons :

{

abstraction :

"truffle" ,

}, }); const

{

balance ,

ether ,

expectRevert ,

expectEvent

}

```
=
require ( '@openzeppelin/test-helpers' ); const
Marketplace
=
artifacts . require ( "Marketplace" ); const
BoredPetsNFT
=
artifacts . require ( "BoredPetsNFT" ); function
assertListing ( actual ,
expected )
{
assert . equal ( actual . nftContract ,
expected . nftContract ,
"NFT contract is not correct" );
assert . equal ( actual . tokenId ,
expected . tokenId ,
"TokenId is not correct" );
assert . equal ( actual . owner ,
expected . owner ,
"Owner is not correct" );
assert . equal ( actual . seller ,
expected . seller ,
"Seller is not correct" );
assert . equal ( actual . price ,
expected . price ,
"Price is not correct" );
assert . equal ( actual . listed ,
expected . listed ,
"Listed is not correct" ) } function
getListing ( listings ,
tokenId )
{
let
listing
=
{};
listings . every (( _listing )
=>
{
if
( _listing . tokenId
==
tokenId )
{
listing
=
_listing ;
return
false ;
}
else
{
return
true ;
}
});
return
listing } function
listingToString ( listing )
{
let
listingCopy
=
{... listing };
listingCopy . tokenId
=
listing . tokenId . toString ();
listingCopy . price
=
listing . price . toString ();
if
( listing . listed )
{
listingCopy . listed
```

```
=
listing . listed . toString ();
}
return
listingCopy ; } async
function
mintNft ( nftContract ,
tokenOwner )
{
return
( await
nftContract . mint ( "fakeURI" ,
{ from :
tokenOwner })). logs [ 0 ]. args . tokenId . toNumber () } contract ( "Marketplace" ,
function
( accounts )
{
const
MARKETPLACE_OWNER
=
accounts [ 0 ];
const
TOKEN_OWNER
=
accounts [ 1 ];
const
BUYER
=
accounts [ 2 ];
let
marketplace ;
let
boredPetsNFT ;
let
nftContract ;
let
listingFee ;
```

```
before ( 'should reuse variables' ,
async
()
=>
{
marketplace
=
await
Marketplace . deployed ();
boredPetsNFT
=
await
BoredPetsNFT . deployed ();
nftContract
=
boredPetsNFT . address ;
listingFee
=
( await
marketplace . LISTING_FEE ()). toString ();
console . log ( "marketplace %s" ,
marketplace . address )
console . log ( "token_owner %s" ,
TOKEN_OWNER )
console . log ( "buyer %s" ,
BUYER )
});
it ( "should validate before listing" ,
async
function
()
{
await
expectRevert (
marketplace . listNft ( nftContract ,
1 ,
ether ( ".005" ),
{ from :
TOKEN_OWNER }),
"Not enough ether for listing fee"
);
await
expectRevert (
marketplace . listNft ( nftContract ,
1 ,
0 ,
{ from :
TOKEN_OWNER ,
value :
listingFee }),
"Price must be at least 1 wei"
);
});
it ( "should list nft" ,
async
function
()
{
let
tokenID
=
await
mintNft ( boredPetsNFT ,
TOKEN_OWNER );
let
tracker
=
await
balance . tracker ( MARKETPLACE_OWNER );
await
tracker . get ();
let
txn
=
await
marketplace . listNft ( nftContract ,
tokenID ,
ether ( ".005" ),
{ from :
TOKEN_OWNER ,
value :
listingFee });
assert . equal ( await
tracker . delta (),
listingFee ,
"Listing fee not transferred" );
let
expectedListing
=
{
nftContract :
nftContract ,
tokenId :
tokenID ,
seller :
TOKEN_OWNER ,
owner :
marketplace . address ,
price :
ether ( ".005" ),
listed :
true
};
assertListing ( getListing ( await
marketplace . getListedNfts (),
tokenID ),
expectedListing );
assertListing ( getListing ( await
marketplace . getMyListedNfts ({ from :
TOKEN_OWNER }),
tokenID ),
expectedListing );
delete
expectedListing . listed ;
expectEvent ( txn ,
"NFTListed" ,
listingToString ( expectedListing ));
});
it ( "should validate before buying" ,
async
function
()
{
await
expectRevert (
marketplace . buyNft ( nftContract ,
1 ,
{ from :
BUYER }),
"Not enough ether to cover asking price"
);
});
it ( "should modify listings when nft is bought" ,
async
function
()
{
let
tokenID
=
await
mintNft ( boredPetsNFT ,
TOKEN_OWNER );
await
marketplace . listNft ( nftContract ,
tokenID ,
ether ( ".005" ),
{ from :
```

```
TOKEN_OWNER ,
value :
listingFee });
let
expectedListing
=
{
nftContract :
nftContract ,
tokenId :
tokenID ,
seller :
TOKEN_OWNER ,
owner :
marketplace . address ,
price :
ether ( ".005" ),
listed :
true
};
assertListing ( getListing ( await
marketplace . getListedNfts (),
tokenID ),
expectedListing );
let
tracker
=
await
balance . tracker ( TOKEN_OWNER );
let
txn
=
await
marketplace . buyNft ( nftContract ,
tokenID ,
{ from :
BUYER ,
value :
ether ( ".005" )});
expectedListing . owner
=
BUYER ;
expectedListing . listed
=
false ;
assert . equal (( await
tracker . delta ()). toString (),
ether ( ".005" ). toString (),
"Price not paid to seller" );
assertListing ( getListing ( await
marketplace . getMyNfts ({ from :
BUYER }),
tokenID ),
expectedListing );
delete
expectedListing . listed ;
expectEvent ( txn ,
"NFTSold" ,
listingToString ( expectedListing ));
});
it ( "should validate reselling" ,
async
function
()
{
await
expectRevert (
marketplace . resellNft ( nftContract ,
1 ,
0 ,
{ from :
BUYER ,
value :
listingFee }),
"Price must be at least 1 wei"
);
await
expectRevert (
marketplace . resellNft ( nftContract ,
1 ,
ether ( ".005" ),
{ from :
BUYER }),
"Not enough ether for listing fee"
);
});
it ( "should resell nft" ,
async
function
()
{
let
tokenID
=
await
mintNft ( boredPetsNFT ,
TOKEN_OWNER );
await
marketplace . listNft ( nftContract ,
tokenID ,
ether ( ".005" ),
{ from :
TOKEN_OWNER ,
value :
listingFee });
await
marketplace . buyNft ( nftContract ,
tokenID ,
{ from :
BUYER ,
value :
```

```
ether ( ".005" )});
let
expectedListing
=
{
nftContract :
nftContract ,
tokenId :
tokenID ,
seller :
TOKEN_OWNER ,
owner :
BUYER ,
price :
ether ( ".005" ),
listed :
false
};
assertListing ( getListing ( await
marketplace . getMyNfts ({ from :
BUYER }),
tokenID ),
expectedListing );
await
boredPetsNFT . approve ( marketplace . address ,
tokenID ,
{ from :
BUYER });
let
txn
=
await
marketplace . resellNft ( nftContract ,
tokenID ,
ether ( ".005" ),
{ from :
BUYER ,
value :
listingFee });
expectedListing . seller
=
BUYER ;
expectedListing . owner
=
marketplace . address ;
expectedListing . listed
=
true ;
assertListing ( getListing ( await
marketplace . getListedNfts (),
tokenID ),
expectedListing );
assertListing ( getListing ( await
marketplace . getMyListedNfts ({ from :
BUYER }),
tokenID ),
expectedListing );
delete
expectedListing . listed ;
expectEvent ( txn ,
"NFTListed" ,
listingToString ( expectedListing ));
}); });
```

Other Ways to Deploy¶

Deploy to Truffle Dashboards¶

In this tutorial, we'll take you through how to deploy to a testnet usingTruffle dashboards , which allows you to sign your transactions using MetaMask. This allows you to keep your private key safe, since you don't have to save it locally. In order to do so, you'll need to get some test eth from afaucet .

First, runtruffle dashboard in a separate terminal window. It should open truffle dashboards at http://127.0.0.1:24012/. Then, you can use your MetaMask wallet to connect to a network of your choice. In this case, we'll select the Goerli network.

Next, migrate your contract to dashboards usingtruffle migrate --network dashboard . If you head back to the dashboard tab, you'll see the request to sign using MetaMask. Hit accept, and voila! Head back to the terminal, and you should see the that your contracts were deployed.

After changing the account number, you can execute the script again to test:truffle exec scripts/run.js

Deploy Using a.env

File¶

If you don't want to use Dashboards, you can modify your Truffle config files to use environment variables you set up. Create a.env file with the following code:

INFURA_KEY

"" GANACHE_MNEMONIC = "" GOERLI_MNEMONIC = "" The.gitignore already ignores.env , but since you're populating your mnemonic/secret key here, PLEASE DO NOT COMMIT THIS ANYWHERE PUBLIC.

Then, just runtruffle migrate --network [NETWORK NAME] ornpm run migrate:ovm --network=[NETWORK NAME]

Deploy to Optimistic Goerli¶

Since Optimism is EVM equivalent withsmall exceptions , we can just copy our contracts from thecontracts/ethereum over tocontracts/optimism .

To get Eth for Optimistic Goerli, you can use thisfaucet . You'll need to sign up with a Github account older than 1 month and be following at least 5 repos: -trufflesuite -truffle-box -Infura -MetaMask -Optimism

Our box already includes the Optimistic Goerli configuration intruffle-confg.ovm.js . Runnpm run migrate:ovm --network=optimistic_goerli to deploy, andnpm run exec:ovm scripts/run.js --network=optimistic_goerli to test!

Since we are using a testnet, connectivity can get a bit flaky. You can try again by running themigrate command with--reset . Alternatively, you might see occasionally see something like this:

TypeError: Cannot read

properties of null (reading 'from') In which case, you can specify thefrom address in the deploy script as follows (ACCOUNT_ADDRESS is the address that will be signing the transaction):

module . exports

=

async

function (deployer)

{

await

deployer . deploy (Marketplace);

const

marketplace

=

await

Marketplace . deployed ();

await

deployer . deploy (BoredPetsNFT ,

marketplace . address ,

{ from :

"ACCOUNT_ADDRESS" }); }

Deploy to Optimism Locally¶

You can also deploy to a locally running instance of Optimism. You'll need to have veryspecific system requirements . Make sure ports 9545 and 8545 are free, and then run:

npm run installLocalOptimism npm run startLocalOptimism

Create an Infura IPFS project¶

You'll need Infura IPFS account and dedicated gateway to upload your NFT metadata. To create a IPFS project, select create IPFS project.

Then, you'll need to create a unique gateway name. In this project, we'll call itoptimism-demo . You will need to give your own dedicated gateway with its own unique name.

Create Your Front End¶

First, we need to install some packages to get our client up and running:

cd

client npm install axios npm install web3modal npm install web3 npm install ipfs-http-client Then, we need to create or edit 6 files that sit underclient/pages :

_app.js

¶

This file organizes the link routing

import

'../styles/globals.css' import

Link

from

'next/link' function

MyApp ({

Component ,

pageProps

})

{

return

(

< div

< nav

className

"border-b p-6"

< p

className

"text-4xl font-bold"

Bored

Pet

Marketplace < /p>

< div

className

"flex mt-4"

< Link

href

"/"

className

"mr-4 text-teal-400"

Home

< /Link>

< Link

href

"/create-and-list-nft"

className

"mr-6 text-teal-400"

Sell

a

new

NFT

< /Link>

< Link

href

"/my-nfts"

className

"mr-6 text-teal-400"

My

NFTs

< /Link>

< Link

href

"/my-listed-nfts"

className

"mr-6 text-teal-400"

My

Listed

NFTs

< /Link>

< /div>

< /nav>

< Component

{... pageProps }

/>

< /div>

) } export

default

MyApp

index.js

¶

Thie file is the Home tab, where a user can see and buy all of the listed NFTs.

import

Web3

from

'web3' ; import

Web3Modal

from

'web3modal' ; import

{

useEffect ,

useState

}

from

'react' ; import

axios

from

'axios' ; import

Marketplace

from

```
'../contracts/optimism-contracts/Marketplace.json' import
BoredPetsNFT
from
'../contracts/optimism-contracts/BoredPetsNFT.json' export
default
function
Home ()
{
const
[ nfts ,
setNfts ]
=
useState ([])
const
[ loadingState ,
setLoadingState ]
=
useState ( 'not-loaded' )
useEffect (()
=>
{
loadNFTs ()
},
[])
async
function
loadNFTs ()
{
const
web3Modal
=
new
Web3Modal ()
const
provider
=
await
```

```
web3Modal . connect ()
const
web3
=
new
Web3 ( provider )
const
networkId
=
await
web3 . eth . net . getId ()
// Get all listed NFTs
const
marketPlaceContract
=
new
web3 . eth . Contract ( Marketplace . abi ,
Marketplace . networks [ networkId ]. address )
const
listings
=
await
marketPlaceContract . methods . getListedNfts (). call ()
// Iterate over the listed NFTs and retrieve their metadata
const
nfts
=
await
Promise . all ( listings . map ( async
( i )
=>
{
try
{
const
boredPetsContract
=
new
web3 . eth . Contract ( BoredPetsNFT . abi ,
BoredPetsNFT . networks [ networkId ]. address )
const
tokenURI
=
await
boredPetsContract . methods . tokenURI ( i . tokenId ). call ()
const
meta
=
await
axios . get ( tokenURI )
const
nft
=
{
price :
i . price ,
tokenId :
i . tokenId ,
seller :
i . seller ,
owner :
i . buyer ,
image :
meta . data . image ,
name :
meta . data . name ,
description :
meta . data . description ,
}
return
nft
}
catch ( err )
{
```

```
console . log ( err )
return
null
}
}))
setNfts ( nfts . filter ( nft
=>
nft
!==
null ))
setLoadingState ( 'loaded' )
}
async
function
buyNft ( nft )
{
const
web3Modal
=
new
Web3Modal ()
const
provider
=
await
web3Modal . connect ()
const
web3
=
new
Web3 ( provider )
const
networkId
=
await
web3 . eth . net . getId ();
const
```

```
marketPlaceContract
=
new
web3 . eth . Contract ( Marketplace . abi ,
Marketplace . networks [ networkId ]. address );
const
accounts
=
await
web3 . eth . getAccounts ();
await
marketPlaceContract . methods . buyNft ( BoredPetsNFT . networks [ networkId ]. address ,
nft . tokenId ). send ({
from :
accounts [ 0 ],
value :
nft . price
});
loadNFTs ()
}
if
( loadingState
===
'loaded'
&&
! nfts . length )
{
return
( < h1
```

className

```
"px-20 py-10 text-3xl"
        No
pets
available !< /h1>)
}
else
{
```

return

(

< div

className

"flex justify-center"

< div

className

"px-4"

style

{

{

maxWidth :

'1600px'

}

}

< div

className

"grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-4 gap-4 pt-4"

{

nfts . map ((nft ,

i)

=>

(

< div

key

{ i }

className

"border shadow rounded-xl overflow-hidden"

< img

src

{ nft . image }

/>

< div

className

"p-4"

< p

style

{

{

height :

'64px'

}

}

className

"text-2xl font-semibold"

	{ nft . name } < /p>

< div

style

{

{

height :

'70px' ,

overflow :

'hidden'

}

}

< p

className

"text-gray-400"

	{ nft . description } < /p>

< /div>

< /div>

< div

className

"p-4 bg-black"

< p

className

"text-2xl font-bold text-white"

    { Web3 . utils . fromWei (nft . price ,

"ether")}

ETH < /p>

< button

className

"mt-4 w-full bg-teal-400 text-white font-bold py-2 px-12 rounded"

onClick

{()

=>

buyNft (nft)}

    Buy < /button>

< /div>

< /div>

))

}

< /div>

< /div>

< /div>

)

} }

create-and-list-nft.js

¶

This is the Sell tab, where a user can create and list an NFT. Make sure you replacewith the dedicated gateway name you create in your IPFS project on Infura. You'll also need to add in your IPFS API and Secret to create our IPFS client. To do so, create.env.local in yourclient folder. Then, populate it with these values:

NEXT_PUBLIC_IPFS_SECRET

NEXT_PUBLIC_IPFS_PROJECT_ID

Then, copy paste this code:

import

{

```
useState
}
from
'react' import
Web3
from
'web3' import
Web3Modal
from
'web3modal' import
{
useRouter
}
from
'next/router' import
{
create
as
ipfsHttpClient
}
from
'ipfs-http-client' import
Marketplace
from
'../contracts/optimism-contracts/Marketplace.json' import
BoredPetsNFT
from
'../contracts/optimism-contracts/BoredPetsNFT.json' const
projectId
=
process . env [ "NEXT_PUBLIC_IPFS_KEY" ]; const
projectSecret
=
process . env [ "NEXT_PUBLIC_IPFS_PROJECT_ID" ]; const
auth
=
'Basic '
```

```
+
Buffer . from ( projectId
+
':'
+
projectSecret ). toString ( 'base64' ); const
client
=
ipfsHttpClient ({
host :
'ipfs.infura.io' ,
port :
5001 ,
protocol :
'https' ,
headers :
{
authorization :
auth ,
}, }); export
default
function
CreateItem ()
{
const
[ fileUrl ,
setFileUrl ]
=
useState ( null )
const
[ formInput ,
updateFormInput ]
=
useState ({
price :
'' ,
name :
```

```
" ,
description :
"
})
const
router
=
useRouter ()
async
function
onChange ( e )
{
// upload image to IPFS
const
file
=
e . target . files [ 0 ]
try
{
const
added
=
await
client . add (
file ,
{
progress :
( prog )
=>
console . log (received: { prog } )
}
)
const
url
=
https://<DEDICATED_GATEWAY>.infura-ipfs.io/ipfs/ { added . path }
setFileUrl ( url )
```

```
}
catch
( error )
{
console . log ( 'Error uploading file: ' ,
error )
}
}
async
function
uploadToIPFS ()
{
const
{
name ,
description ,
price
}
=
formInput
if
( ! name
||
! description
||
! price
||
! fileUrl )
{
return
}
else
{
// first, upload metadata to IPFS
const
data
=
JSON . stringify ({
name ,
description ,
image :
fileUrl
})
try
{
const
added
=
await
client . add ( data )
console . log ( 'added: ' ,
added )
const
url
=
https://<DEDICATED_GATEWAY>.infura-ipfs.io/ipfs/ { added . path }
// after metadata is uploaded to IPFS, return the URL to use it in the transaction
return
url
}
catch
( error )
{
console . log ( 'Error uploading file: ' ,
error )
}
}
}
async
function
listNFTForSale ()
{
const
web3Modal
```

```
=
new
Web3Modal ()
const
provider
=
await
web3Modal . connect ()
const
web3
=
new
Web3 ( provider )
const
url
=
await
uploadToIPFS ()
const
networkId
=
await
web3 . eth . net . getId ()
// Mint the NFT
const
boredPetsContractAddress
=
BoredPetsNFT . networks [ networkId ]. address
const
boredPetsContract
=
new
web3 . eth . Contract ( BoredPetsNFT . abi ,
boredPetsContractAddress )
const
accounts
=
await
web3 . eth . getAccounts ()
const
marketPlaceContract
=
new
web3 . eth . Contract ( Marketplace . abi ,
Marketplace . networks [ networkId ]. address )
let
listingFee
=
await
marketPlaceContract . methods . LISTING_FEE (). call ()
listingFee
=
listingFee . toString ()
boredPetsContract . methods . mint ( url ). send ({
from :
accounts [ 0 ]
}). on ( 'receipt' ,
function
( receipt )
{
console . log ( 'minted' );
// List the NFT
const
tokenId
=
receipt . events . NFTMinted . returnValues [ 0 ];
marketPlaceContract . methods . listNft ( boredPetsContractAddress ,
tokenId ,
Web3 . utils . toWei ( formInput . price ,
"ether" ))
. send ({
from :
accounts [ 0 ],
value :
```

listingFee

}). on ('receipt' ,

function

()

{

console . log ('listed')

router . push ('/')

});

});

}

return

(

< div

className

"flex justify-center"

< div

className

"w-1/2 flex flex-col pb-12"

< input

placeholder

"Asset Name"

className

"mt-8 border rounded p-4"

onChange

{ e

=>

updateFormInput ({

... formInput ,

name :

e . target . value

})}

/>

< textarea

placeholder

"Asset Description"

className

"mt-2 border rounded p-4"

onChange

{ e

=>

updateFormInput ({

... formInput ,

description :

e . target . value

})}

/>

< input

placeholder

"Asset Price in Eth"

className

"mt-2 border rounded p-4"

onChange

{ e

=>

updateFormInput ({

... formInput ,

price :

e . target . value

})}

/>

< input

type

"file"

name

"Asset"

className

"my-4"

onChange

{ onChange }

/>

{

fileUrl

&&

(

< img

className

"rounded mt-4"

width

"350"

src

{ fileUrl }

/>

)

}

< button

onClick

{ listNFTForSale }

className

"font-bold mt-4 bg-teal-400 text-white rounded p-4 shadow-lg"

Mint

and

list

NFT

< /button>

< /div>

< /div>

) }

my-nfts.js

¶

This is the My NFTs tab, where the user can see the NFTs they own and choose to resell.

import

Web3

from

'web3' ; import

{

useEffect ,

useState

}

from

'react' import

axios

from

'axios' import

Web3Modal

from

'web3modal' import

{

useRouter

}

from

'next/router' import

Marketplace

from

'../contracts/optimism-contracts/Marketplace.json' ; import

BoredPetsNFT

from

'../contracts/optimism-contracts/BoredPetsNFT.json' ; export

default

function

MyAssets ()

{

const

[nfts ,

setNfts]

=

useState ([])

const

[loadingState ,

setLoadingState]

=

useState ('not-loaded')

const

router

=

useRouter ()

useEffect (()

=>

{

loadNFTs ()

},

[])

async

function

loadNFTs ()

{

const

web3Modal

=

new

Web3Modal ()

const

provider

=

await

web3Modal . connect ()

const

web3

=

new

Web3 (provider)

```
const
networkId
=
await
web3 . eth . net . getId ()
const
marketPlaceContract
=
new
web3 . eth . Contract ( Marketplace . abi ,
Marketplace . networks [ networkId ]. address )
const
boredPetsContractAddress
=
BoredPetsNFT . networks [ networkId ]. address
const
boredPetsContract
=
new
web3 . eth . Contract ( BoredPetsNFT . abi ,
boredPetsContractAddress )
const
accounts
=
await
web3 . eth . getAccounts ()
const
data
=
await
marketPlaceContract . methods . getMyNfts (). call ({ from :
accounts [ 0 ]})
const
nfts
=
await
Promise . all ( data . map ( async
i
=>
{
try
{
const
tokenURI
=
await
boredPetsContract . methods . tokenURI ( i . tokenId ). call ()
const
meta
=
await
axios . get ( tokenURI )
let
nft
=
{
price :
i . price ,
tokenId :
i . tokenId ,
seller :
i . seller ,
owner :
i . buyer ,
image :
meta . data . image ,
name :
meta . data . name ,
description :
meta . data . description ,
tokenURI :
tokenURI
}
return
```

nft

}

catch (err)

{

console . log (err)

return

null

}

}))

setNfts (nfts . filter (nft

=>

nft

!==

null))

setLoadingState ('loaded')

}

function

listNFT (nft)

{

router . push (/resell-nft?id= { nft . tokenId } &tokenURI= { nft . tokenURI })

}

if

(loadingState

===

'loaded'

&&

! nfts . length)

{

return

(< h1

className

"py-10 px-20 text-3xl"

    No

NFTs

owned < /h1>);

}

else

{

return

(

< div

className

"flex justify-center"

< div

className

"p-4"

< div

className

"grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-4 gap-4 pt-4"

{

nfts . map ((nft ,

i)

=>

(

< div

key

{ i }

className

"border shadow rounded-xl overflow-hidden"

< img

src

{ nft . image }

className

"rounded"

/>

< div

className

"p-4"

< p

style

{

{

height :

'64px'

}

}

className

"text-2xl font-semibold"

{ nft . name } < /p>

< div

style

{

{

height :

'70px' ,

overflow :

'hidden'

}

}

< p

className

"text-gray-400"

{ nft . description } < /p>

< /div>

< /div>

< div

className

"p-4 bg-black"

< p

className

"text-2xl font-bold text-white"

Price

-

{ Web3 . utils . fromWei (nft . price ,

"ether")}

Eth < /p>

< button

className

"mt-4 w-full bg-teal-400 text-white font-bold py-2 px-12 rounded"

onClick

{()

=>

listNFT (nft)}

List < /button>

< /div>

< /div>

))

}

< /div>

< /div>

< /div>

);

} }

resell-nft.js

¶

This is the page the user is directed to to resell their NFTs.

import

{

useEffect ,

useState

}

from

'react' import

Web3

from

```
'web3' import
{
useRouter
}
from
'next/router' import
axios
from
'axios' import
Web3Modal
from
'web3modal' import
Marketplace
from
'../contracts/optimism-contracts/Marketplace.json' import
BoredPetsNFT
from
'../contracts/optimism-contracts/BoredPetsNFT.json' export
default
function
ResellNFT ()
{
const
[ formInput ,
updateFormInput ]
=
useState ({
price :
'' ,
image :
''
})
const
router
=
useRouter ()
const
```

```
{
id ,
tokenURI
}
=
router . query
const
{
image ,
price
}
=
formInput
useEffect (()
=>
{
fetchNFT ()
},
[ id ])
async
function
fetchNFT ()
{
if
( ! tokenURI )
{
return
}
else
{
const
meta
=
await
axios . get ( tokenURI )
updateFormInput ( state
=>
({
... state ,
image :
meta . data . image
}))
}
}
async
function
listNFTForSale ()
{
if
( ! price )
{
return
}
else
{
const
web3Modal
=
new
Web3Modal ()
const
provider
=
await
web3Modal . connect ()
const
web3
=
new
Web3 ( provider )
const
networkId
=
await
```

```
web3 . eth . net . getId ()
const
marketPlaceContract
=
new
web3 . eth . Contract ( Marketplace . abi ,
Marketplace . networks [ networkId ]. address )
let
listingFee
=
await
marketPlaceContract . methods . LISTING_FEE (). call ()
listingFee
=
listingFee . toString ()
const
accounts
=
await
web3 . eth . getAccounts ()
marketPlaceContract . methods . resellNft ( BoredPetsNFT . networks [ networkId ]. address ,
id ,
Web3 . utils . toWei ( formInput . price ,
"ether" ))
. send ({
from :
accounts [ 0 ],
value :
listingFee
}). on ( 'receipt' ,
function
()
{
router . push ( '/' )
});
}
}
```

return

(

< div

className

"flex justify-center"

< div

className

"w-1/2 flex flex-col pb-12"

< input

placeholder

"Asset Price in Eth"

className

"mt-2 border rounded p-4"

onChange

{ e

=>

updateFormInput ({

... formInput ,

price :

e . target . value

})}

/>

{

image

&&

(

< img

className

"rounded mt-4"

width

"350"

src

{ image }

/>

)

}

< button

onClick

{ listNFTForSale }

className

"font-bold mt-4 bg-teal-400 text-white rounded p-4 shadow-lg"

List

NFT

< /button>

< /div>

< /div>

) }

my-listed-nfts.js

¶

This is the My Listed NFTs tab, where users can see what NFTs they have listed for sale.

import

Web3

from

'web3' ; import

{

useEffect ,

useState

}

from

'react' ; import

axios

from

'axios' ; import

Web3Modal

from

'web3modal' ; import

```
Marketplace
from
'../contracts/optimism-contracts/Marketplace.json' ; import
BoredPetsNFT
from
'../contracts/optimism-contracts/BoredPetsNFT.json' ; export
default
function
CreatorDashboard ()
{
const
[ nfts ,
setNfts ]
=
useState ([])
const
[ loadingState ,
setLoadingState ]
=
useState ( 'not-loaded' )
useEffect (()
=>
{
loadNFTs ()
},
[])
async
function
loadNFTs ()
{
const
web3Modal
=
new
Web3Modal ()
const
provider
```

```
=
await
web3Modal . connect ()
const
web3
=
new
Web3 ( provider )
const
networkId
=
await
web3 . eth . net . getId ()
// Get listed NFTs
const
marketPlaceContract
=
new
web3 . eth . Contract ( Marketplace . abi ,
Marketplace . networks [ networkId ]. address )
const
accounts
=
await
web3 . eth . getAccounts ()
const
listings
=
await
marketPlaceContract . methods . getMyListedNfts (). call ({ from :
accounts [ 0 ]})
// Iterate over my listed NFTs and retrieve their metadata
const
nfts
=
await
Promise . all ( listings . map ( async
i
=>
{
try
{
const
boredPetsContract
=
new
web3 . eth . Contract ( BoredPetsNFT . abi ,
BoredPetsNFT . networks [ networkId ]. address )
const
tokenURI
=
await
boredPetsContract . methods . tokenURI ( i . tokenId ). call ();
const
meta
=
await
axios . get ( tokenURI );
let
item
=
{
price :
i . price ,
tokenId :
i . tokenId ,
seller :
i . seller ,
owner :
i . owner ,
image :
meta . data . image ,
}
return
item
}
catch ( err )
{
console . log ( err )
return
null
}
}))
setNfts ( nfts . filter ( nft
=>
nft
!==
null ))
setLoadingState ( 'loaded' )
}
if
( loadingState
===
'loaded'
&&
! nfts . length )
{
return
( < h1
```

className

```
"py-10 px-20 text-3xl"
        No
NFTs
listed < /h1>)
}
else
{
return
(
< div
< div
```

className

"p-4"

< h2

className

"text-2xl py-2"

Items

Listed < /h2>

< div

className

"grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-4 gap-4 pt-4"

{

nfts . map ((nft ,

i)

=>

(

< div

key

{ i }

className

"border shadow rounded-xl overflow-hidden"

< img

src

{ nft . image }

className

"rounded"

/>

< div

className

"p-4 bg-black"

< p

className

"text-2xl font-bold text-white"

Price

-

{ Web3 . utils . fromWei (nft . price ,

"ether")}

Eth < /p>

< /div>

< /div>

))

}

< /div>

< /div>

< /div>

)

} }

Web3 Client Overview¶

Let's go over the Web3 concepts and utilities we've used to hook up our front end.

- web3Modal
- is a library we use to retrieve the user's network provider
- ipfs-http-client
- is a library we use to upload the NFTs metadata to IPFS
- web3
- is a library that allows us to use our smart contract abstractions

Depending on if you want to use your Optimism contracts or your Ethereum contracts, you'll change the contract import path:

import

Marketplace

from

'../contracts/optimism-contracts/Marketplace.json' import

BoredPetsNFT

from

'../contracts/optimism-contracts/BoredPetsNFT.json' When we call our contract methods withWeb3.js , we usecall when we don't alter the contract state, andsend when we do. You can read more about ithere .

Deploy Your Front End¶

To see the front end in action, just use the node scripts inpackage.json . Runnpm run dev from theclient folder, and your website should be brought up onhttp://localhost:3000/ !

Note that the front end will be using whatever network and account is set on your MetaMask extension.

Adding Ganache to MetaMask¶

If you would like to use Ganache as your network, you can add the network to your MetaMask wallet with the following properties:

- Network Name: Ganache
- New RPC URL: http://127.0.0.1:7545
- Chain ID: 1337
- Currency Symbol: ETH

Then, import a new account. If you are using the Ganache UI, you can grab the private key by clicking on the key icon to reveal the account keys.

Adding Optimistic Ethereum to MetaMask¶

If you would like to use your local instance of Optimism, you can add the network to your MetaMask wallet with the following properties:

- Network Name: Optimistic Ethereum
- New RPC URL: http://127.0.0.1:8545
- Chain ID: 17
- Currency Symbol: ETH

Then, import a new account. You can get the list of accounts and private keys here .

Additional Notes¶

And there you have it - a very basic marketplace! There are many ways you can build upon this project: - Allow users to bid on an NFT - Allow useres to delist their NFTs - Combining the Marketplace contract and NFT contract into one - Bridging the Optimism contracts (Truffle L2 bridge box coming soon!)

If you're interested in actually bridging between Ethereum and Optimism, check out our Optimism Bridge Box !

Join Github Discussions to join the Truffle community to discuss and ask questions!