Web3 Unleashed: Build a NFT Rental Marketplace Part 2¶

Written byEmily Lin

Last updated 11/10/2022

Overview¶

In episode 2 and 4, we created arentable NFT and wrote arental marketplace smart contract to interact with it! If you haven't gone through those two tutorials, they are prerequisites for completing this one.

In this guide, we'll create the frontend for the rental marketplace. Specifically, we will cover:

1. Writing a frontend for minting, listing, renting, and viewing NFTs in React
2. Using Infura's NFT API to query for all of a user's NFTs
3. Using an IPFS gateway to upload NFT metadata
4. Usingweb3.js
5. to call our smart contract functions

Watch the livestream onYouTube for a more in-depth walkthrough!

The completed code liveshere .

NOTE: It is assumed you have familiarity with React.js. This tutorial is meant to illustrate how to interact with your smart contracts with Web3js, use Infura's NFT API for querying NFTs, and uploading metadata through the IPFS gateway - NOT teach you the fundamentals of JavaScript or React development.

Download System Requirements¶

You'll need to install:

- Node.js
- , v14 or higher
- truffle
- ganache CLI

Create an Infura account and project¶

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

Once you're signed in, create a project! Let's call itrentable-nft-marketplace , and select Web3 API from the dropdown

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 itrental-marketplace . You will need to give your own dedicated gateway with its own unique name.

Register for a MetaMask wallet¶

To interact with your DApp in the browser, you'll need a MetaMask wallet. You can download it and create onehere .

Download VS Code¶

Feel free to use whatever IDE you want, but we highly recommend using VS Code! 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 .

Get Some Test Eth¶

In order to deploy to the public testnets, you'll need some test Eth to cover your gas fees has a great MultiFaucet that deposits funds across 8 different networks all at once.

Set Up Your Project¶

There's a lot of code to go over, so rather than build it piece by piece, go ahead and just clone the completed code here . Now, we'll go over the high level overview of the project structure, and the key components of how the frontend communicates with the smart contracts we had written before!

After cloning the project, you'll need to install the dependencies:

cd

unleashed_nft_rental_marketplace/truffle npm icd

../client npm i In order to utilize the NFT API, we'll need to createenv.json to store the necessary keys:

cp ./src/env.example.json ./src/env.json Then, add your Infura project id, secret, wallet private key, and IPFS subdomain toenv.json . The IPFS subdomain should be the UNIQUE SUBDOMAIN NAME you chose, not the full URL.

NOTE: SetinfuraSdk.active inenv.json to false if you are using a ganache instance to test. In this tutorial, we'll be doing everything on goerli.

client/src/contexts

¶

This folder includes all the files that provide the necessary information to interact with the blockchain.

client/src/contexts/EthContext/EthProvider.jsx

¶

EthContext contains the code that allows you to connect to the blockchain. Specifically,EthProvider.jsx ...

1. Retrieves the contract abis you will be interacting with
2. useEffect
3. (()
4. =>
5. {
6. const
7. tryInit
8. =
9. ()
10. =>
11. {
12. try
13. {
14. const
15. artifacts
16. =
17. {
18. RentableNft
19. :
20. require
21. (
22. "../../contracts/RentableNft.json"
23.),
24. Marketplace
25. :
26. require
27. (
28. "../../contracts/Marketplace.json"
29.)
30. };
31. init
32. (
33. artifacts
34.);
35. }
36. catch
37. (

```
err
)
{
console
.
error
(
err
);
}
};
tryInit
();
},
[
init
]);
Creates an instance ofweb3
to retrieve the current network and account information
const
web3
=
new
Web3
(
Web3
.
givenProvider
||
"ws://localhost:8545"
);
const
accounts
=
await
web3
.
eth
.
requestAccounts
();
const
networkID
=
await
web3
.
eth
.
net
.
getId
();
Gets the contract abstraction associated with the currently connected network
try
{
for
(
const
[
contractName
,
artifact
]
of
Object
.
entries
```

```
(
artifacts
))
{
const
address
=
artifact
.
networks
[
networkID
].
address
;
const
contract
=
new
web3
.
eth
.
Contract
(
artifact
.
abi
,
address
);
contracts
[
contractName
]
=
contract
;
}
}
catch
(
err
)
{
contracts
=
null
;
console
.
error
(
err
);
}
Updates contracts when the account or network is changed
useEffect
(()
=>
{
const
events
=
[
"chainChanged"
,
"accountsChanged"
```

```
];
const
handleChange
=
()
=>
{
init
(
state
.
artifacts
);
};
events
.
forEach
(
e
=>
window
.
ethereum
.
on
(
e
,
handleChange
));
return
()
=>
{
events
.
forEach
(
e
=>
window
.
ethereum
.
removeListener
(
e
,
handleChange
));
};
},
[
init
,
state
.
artifacts
]);
```

MetaMask injects a global API atwindow.ethereum
. This API allows websites to request users' Ethereum accounts, read data from blockchains the user is connected to, and suggest that the user sign messages and transactions.
window.ethereum.on
will detect if the network or account has changed and re-initializeweb3
,accounts
,networkID
, andcontracts
. As detailed in theMetaMask docs

241. , we should remove the event listener.

client/src/contexts/InfuraContext/index.js

¶

We are utilizing Infura's NFT API to get the user's owned NFTs. This file instantiates the necessary objects to do so:Auth andsdk . Stepping through the code:

1. AnAuth
2. object is created with your Infura project id, secret, and wallet private key that you added toenv.json
3. during project set up
4. const
5. auth
6. =
7. new
8. Auth
9. (
10. env
11. .
12. infuraSdk
13. .
14. authOptions
15.);
16. We check whether or not we are using ganache, set up the SDK so that we can call the NFT API, and get aweb3
17. andartifacts
18. instance fromEthContext
19. as we had covered just above.
20. const
21. {
22. active
23. }
24. =
25. env
26. .
27. infuraSdk
28. ;
29. const
30. [
31. sdk
32. ,
33. _setSdk
34.]
35. =
36. useState
37. (
38. new
39. SDK
40. (
41. auth
42.));
43. const
44. {
45. state
46. :
47. {
48. web3
49. ,
50. artifacts
51. ,
52. networkID
53. },
54. }
55. =
56. useEth
57. ();
58. Then, we want to get all the NFTs that are owned by the current user and fulfill the rentable criteria.
59. const
60. getOwnedRentableNfts

```
=
async
(
publicAddress
)
=>
{
const
{
assets
}
=
await
sdk
.
getNFTs
({
publicAddress
,
includeMetadata
:
true
,
});
const
filtered
=
[];
for
(
const
asset
of
assets
)
{
if
(
await
contracts
.
Marketplace
.
methods
.
isRentableNFT
(
asset
.
contract
).
call
())
{
filtered
.
push
(
asset
);
}
}
return
filtered
;
};
To break this down further,
const
```

{
assets
}
=
await
sdk
.
getNFTs
({
publicAddress
,
includeMetadata
:
true
,
});
sdk.getNFTs
retrieves all the NFTs and metadata ofpublicAddress
.
if
(
await
contracts
.
Marketplace
.
methods
.
isRentableNFT
(
asset
.
contract
).
call
())
{
filtered
.
push
(
asset
);
}
contracts.Marketplace.methods.isRentableNFT
is how we call theisRentableNFT
function on the smart contract. Note that we got thecontracts.Marketplace
abstract by using theEthProvider
throughuseEth()
like so:
const
{
state
:
{
contracts
},
}
=
useEth
();

client/src/contexts/MarketplaceContext/index.js

¶

This file aggregates and updates the data for displaying an NFT:

1. updateListings

is called whenever an NFT is listed, unlisted, or rented. It gets the image to be displayed by calling using thetokenURI function from the NFT smart contract

```
const nftContract = new web3.eth.Contract(artifacts.RentableNft.abi, nftContractAddress);
const tokenUri = await nftContract.methods.tokenURI(tokenId).call();
let tokenUriRes;
try {
tokenUriRes = await (await fetch(getIpfsGatewayUri(tokenUri))).json();
}
catch (err) {
console.error("Bad uri");
}
```

After resolving the image data,updateListings will transform all the variables into the correct type (i.e., useparseInt to convertpricePerDay

from astring
to anint
). The finalupdateListings
looks like this:
```
const
updateListings
=
useCallback
(
async
()
=>
{
if
(
marketplaceContract
)
{
const
res
=
await
marketplaceContract
.
methods
.
getAllListings
().
call
();
const
listingsExtendedTransformed
=
{};
const
listingsExtended
=
await
Promise
.
all
(
res
.
map
(
async
listing
=>
{
const
{
nftContract
:
nftContractAddress
,
pricePerDay
:
pricePerDayStr
,
startDateUNIX
:
startDateUnixStr
,
endDateUNIX
:
endDateUnixStr
,
```

```
expires
:
expiresStr
,
tokenId
,
owner
,
user
}
=
listing
;
const
nftContract
=
new
web3
.
eth
.
Contract
(
artifacts
.
RentableNft
.
abi
,
nftContractAddress
);
const
tokenUri
=
await
nftContract
.
methods
.
tokenURI
(
tokenId
).
call
();
let
tokenUriRes
;
try
{
tokenUriRes
=
await
(
await
fetch
(
getIpfsGatewayUri
(
tokenUri
))).
json
();
}
catch
(
err
)
```

```
{
console
.
error
(
"Bad uri"
);
}
// const noUser = parseInt(user) !== 0;
const
pricePerDay
=
parseInt
(
pricePerDayStr
);
const
startDateUnix
=
parseInt
(
startDateUnixStr
);
const
endDateUnix
=
parseInt
(
endDateUnixStr
);
const
duration
=
endDateUnix
-
startDateUnix
;
const
expires
=
parseInt
(
expiresStr
);
const
isOwner
=
owner
===
accounts
[
0
];
const
isUser
=
user
===
accounts
[
0
];
const
transformedData
=
{
pricePerDay
,
```

```
startDateUnix
,
endDateUnix
,
duration
,
expires
,
user
};
const
listingExtended
=
{
...
listing
,
...
transformedData
,
nftContractAddress
,
tokenUri
,
tokenUriRes
,
isOwner
,
isUser
};
[
...
Array
(
8
).
keys
(),
"nftContract"
,
"startDateUNIX"
,
"endDateUNIX"
,
].
forEach
(
i
=>
void
delete
listingExtended
[
i
]);
if
(
listingsExtendedTransformed
[
nftContractAddress
])
{
listingsExtendedTransformed
[
nftContractAddress
][
tokenId
]
```

```
=
transformedData
;
}
else
{
listingsExtendedTransformed
[
nftContractAddress
]
=
{
[
tokenId
]
:
transformedData
};
}
return
listingExtended
;
})
);
setListings
(
listingsExtended
);
setListingsTransformed
(
listingsExtendedTransformed
);
}
},
[
marketplaceContract
,
web3
?
.
eth
.
Contract
,
artifacts
?
.
RentableNft
.
abi
,
accounts
]
);
updateOwnedTokens
is called whenever a new NFT is minted to update your owned tokens.
It does this in two parts. First, it checksinfura.active
to see if we are using the Infura NFT SDK, which we would use if we were on a public testnet or mainnet. If so, we callgetOwnedRentableNfts
to retrieve all of user's NFTs and filter based on rentability.
if
(
infura
.
active
)
{
// Option 1 - Use infura nft sdk
```

```
if
(
accounts
&&
listingsTransformed
)
{
const
res
=
await
infura
.
getOwnedRentableNfts
(
accounts
[
0
]);
const
tokens
=
await
Promise
.
all
(
res
.
map
(
async
ele
=>
{
return
{
nftContractAddress
:
ele
.
contract
,
tokenId
:
ele
.
tokenId
,
tokenUriRes
:
ele
.
metadata
,
listingData
:
listingsTransformed
[
ele
.
contract
]
?
.[
ele
.
tokenId
```

]
}
})
)
setOwnedTokens
(
tokens
);
}
}
Otherwise, we're on ganache, our local test chain, and we will retrieve NFTs based on events emmitted. Specifically, we will trackTransfer
, which is an event emmitted when calling OpenZeppelin'smint
function on theirERC721
contract. Then, we will construct a token by creating a dictionary of the necessary values (nftContractAddress
,tokenId
,tokenUri
, etc).
if
(
rentableNftContract
&&
listingsTransformed
)
{
const
{
address
:
nftContractAddress
}
=
rentableNftContract
.
options
;
// This only checks rentableNftContract.
const
mintEvents
=
await
rentableNftContract
.
getPastEvents
(
"Transfer"
,
{
filter
:
{
from
:
"0x0000000000000000000000000000000000000000"
,
to
:
accounts
[
0
]
},
fromBlock
:
0
});
const
tokens

```
=
await
Promise
.
all
(
mintEvents
.
map
(
async
mintEvent
=>
{
const
{
tokenId
}
=
mintEvent
.
returnValues
;
const
tokenUri
=
await
rentableNftContract
.
methods
.
tokenURI
(
tokenId
).
call
();
let
tokenUriRes
;
try
{
tokenUriRes
=
await
(
await
fetch
(
getIpfsGatewayUri
(
tokenUri
))).
json
();
}
catch
(
err
)
{
console
.
error
(
"Bad uri"
);
}
```

```
return
{
nftContractAddress
,
tokenId
,
tokenUri
,
tokenUriRes
,
listingData
:
listingsTransformed
[
nftContractAddress
]
?
.[
tokenId
]
};
})
);
setOwnedTokens
(
tokens
);
}
```

The final code looks like this:

```
const
updateOwnedTokens
=
useCallback
(
async
()
=>
{
if
(
infura
.
active
)
{
// Option 1 - Use infura nft sdk
if
(
accounts
&&
listingsTransformed
)
{
const
res
=
await
infura
.
getOwnedRentableNfts
(
accounts
[
0
]);
const
tokens
=
```

```
await
Promise
.
all
(
res
.
map
(
async
ele
=>
{
return
{
nftContractAddress
:
ele
.
contract
,
tokenId
:
ele
.
tokenId
,
tokenUriRes
:
ele
.
metadata
,
listingData
:
listingsTransformed
[
ele
.
contract
]
?
.[
ele
.
tokenId
]
}
})
)
setOwnedTokens
(
tokens
);
}
}
else
{
// Option - 2 - Use contract events
// This is useful when using local network (ganache) or network otherwise unsupported
// docs.infura.io/infura/infura-custom-apis/nft-sdk/supported-networks
if
(
rentableNftContract
&&
listingsTransformed
)
{
```

```
const
{
address
:
nftContractAddress
}
=
rentableNftContract
.
options
;
// This only checks rentableNftContract.
const
mintEvents
=
await
rentableNftContract
.
getPastEvents
(
"Transfer"
,
{
filter
:
{
from
:
"0x0000000000000000000000000000000000000000"
,
to
:
accounts
[
0
]
},
fromBlock
:
0
});
const
tokens
=
await
Promise
.
all
(
mintEvents
.
map
(
async
mintEvent
=>
{
const
{
tokenId
}
=
mintEvent
.
returnValues
;
const
tokenUri
```

```
=
await
rentableNftContract
.
methods
.
tokenURI
(
tokenId
).
call
();
let
tokenUriRes
;
try
{
tokenUriRes
=
await
(
await
fetch
(
getIpfsGatewayUri
(
tokenUri
))).
json
();
}
catch
(
err
)
{
console
.
error
(
"Bad uri"
);
}
return
{
nftContractAddress
,
tokenId
,
tokenUri
,
tokenUriRes
,
listingData
:
listingsTransformed
[
nftContractAddress
]
?
.[
tokenId
]
};
})
);
setOwnedTokens
(
```

tokens
);
}
}
},
[
rentableNftContract
,
listingsTransformed
,
accounts
,
infura
.
getOwnedRentableNfts
]);
mint
does exactly what it sounds like it does: mint an NFT! In order to call a function causes a change, we use.send()
instead ofcall()
. In this case, the caller ofmint
is the active account, which is indicated byfrom: accounts[0]
.
const
mint
=
async
(
tokenUri
)
=>
{
const
tx
=
await
rentableNftContract
.
methods
.
mint
(
tokenUri
).
send
({
from
:
accounts
[
0
]
});
if
(
tx
.
status
)
await
updateOwnedTokens
();
};
list
lists the NFT and also collects the listing fee required to do so. As you can see, we addvalue: listingFee
, which will prompt the user to pay thelistingFee
when signing the transaction.
const
list

```
=
async
(
nftContractAddress
,
tokenId
,
price
,
duration
)
=>
{
// Time values are in seconds
const
buffer
=
30
;
const
start
=
Math
.
ceil
(
Date
.
now
()
/
1000
)
+
buffer
;
const
end
=
start
+
duration
;
const
listingFee
=
await
marketplaceContract
.
methods
.
getListingFee
().
call
();
const
tx
=
await
marketplaceContract
.
methods
.
listNFT
(
nftContractAddress
,
tokenId
```

,
price
,
start
,
end
).
send
({
from
:
accounts
[
0
],
value
:
listingFee
});
if
(
tx
.
status
)
await
updateListings
();
};
unlist
is called by the owner of the NFT to take down the listing. It calculates the refund to give if the NFT was currently being rented.
const
unlist
=
async
(
nftContractAddress
,
tokenId
)
=>
{
const
nftContract
=
new
web3
.
eth
.
Contract
(
artifacts
.
RentableNft
.
abi
,
nftContractAddress
);
const
expires
=
parseInt
(
await
nftContract

```
.
methods
.
userExpires
(
tokenId
).
call
());
const
{
pricePerDay
}
=
listingsTransformed
[
nftContractAddress
][
tokenId
];
const
refund
=
Math
.
ceil
((
expires
-
Date
.
now
()
/
1000
)
/
60
/
60
/
24
+
1
)
*
pricePerDay
;
const
options
=
{
from
:
accounts
[
0
],
value
:
Math
.
max
(
0
,
refund
)
```

```
};
const
tx
=
await
marketplaceContract
.
methods
.
unlistNFT
(
nftContractAddress
,
tokenId
).
send
(
options
);
if
(
tx
.
status
)
await
updateListings
();
};
rent
is our final action! This allows users to rent available NFTs.
const
rent
=
async
(
nftContractAddress
,
tokenId
,
duration
)
=>
{
const
{
pricePerDay
}
=
listingsTransformed
[
nftContractAddress
][
tokenId
];
const
now
=
Math
.
ceil
(
Date
.
now
()
/
1000
```

```
);
const
expires
=
now
+
duration
;
const
numDays
=
(
expires
-
now
)
/
60
/
60
/
24
+
1
;
const
fee
=
Math
.
ceil
(
numDays
*
pricePerDay
);
const
options
=
{
from
:
accounts
[
0
],
value
:
fee
};
const
tx
=
await
marketplaceContract
.
methods
.
rentNFT
(
nftContractAddress
,
tokenId
,
expires
).
send
(
```

```
options
);
if
(
tx
.
status
)
await
updateListings
();
};
```

client/src/contexts/TimeContext/index.js

¶

This file just helps manage all the math we have to do with regards to calculate rental times.

client/src/card

¶

This folder contains all the code for formatting and interacting with your owned, rented, and listed NFTs. It utilizes functions we defined in thecontext files we went over above.

client/layout

client/market client/owned client/rented ¶

These folders contain the code for setting up the layout of what each page will look like.

client/utils/index.js

¶

This file lays out some common utility functions - namely around formatting and time math. You'll note that this is where we do the string manipulation of our IPFS urls to fetch the data for displaying the NFTs.

```
export
const
IPFS_GATEWAY
=
https:// { env . infura . ipfs . subdomain } .infura-ipfs.io/ipfs/  ; export
function
isIpfsUri ( uri )
{
return
uri . match ( /^ipfs:\/\/ ); } export
function
getIpfsGatewayUri ( uri )
{
if
( typeof
uri
```

===

"string")

{

const

ipfsAddress

=

uri . replace (/^ipfs:\/\/i ,

"");

return

{ IPFS_GATEWAY }{ ipfsAddress } ;

} }

Future Extensions¶

After this, you've built a full stack dapp! This marketplace has a LOT of functionality, however, retrieving your smart contract abstractions and interacting with them is easy withWeb3.js . The bulk of the work is just your classic flavor of frontend web development. If you want to interact with a much simpler marketplace frontend that usesNext.js , check out our NFT Marketplace on Optimismhere .

Next episode we'll be covering Web3 Communication with Push Protocol - namely, how to add notifications to your dapp!

If you want to talk about this content, join ourDiscord ! If you need help coding, start a discussionhere . Lastly, don't forget to follow us onTwitter for the latest updates on all things Truffle.