

Web3 Unleashed: Build a NFT Rental Marketplace Part 2

Written by [Emily Lin](#)

Last updated 11/10/2022

Overview

In episode 2 and 4, we created [rentable NFT](#) and wrote a [rental 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. Using web3.js
5. to call our smart contract functions

Watch the livestream on [YouTube](#) for a more in-depth walkthrough!

The completed code lives [here](#).

NOTE: It is assumed you have familiarity with React.js. This tutorial is meant to illustrate how to interact with your smart contracts with Web3.js, 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 account [here](#).

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 one [here](#).

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 it [here](#).

Get Some Test Eth

In order to deploy to the public testnets, you'll need some test Eth to cover your gas fees. [FaucetHub](#) 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 i
```

```
../client npm i
```

 In order to utilize the NFT API, we'll need to create `env.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 to `env.json` . The IPFS subdomain should be the UNIQUE SUBDOMAIN NAME you chose, not the full URL.

NOTE: Set `infuraSdk.active` in `env.json` to `false` if you are using a ganache instance to test. In this tutorial, we'll be doing everything on goerli.

client/src/context



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

client/src/context/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.    (
```

```
38. err
39. )
40. {
41. console
42. .
43. error
44. (
45. err
46. );
47. }
48. };
49. tryInit
50. ();
51. },
52. [
53. init
54. ]);
55. Creates an instance of web3
56. to retrieve the current network and account information
57. const
58. web3
59. =
60. new
61. Web3
62. (
63. Web3
64. .
65. givenProvider
66. ||
67. "ws://localhost:8545"
68. );
69. const
70. accounts
71. =
72. await
73. web3
74. .
75. eth
76. .
77. requestAccounts
78. ();
79. const
80. networkID
81. =
82. await
83. web3
84. .
85. eth
86. .
87. net
88. .
89. getID
90. ();
91. Gets the contract abstraction associated with the currently connected network
92. try
93. {
94. for
95. (
96. const
97. [
98. contractName
99. ,
100. artifact
101. ]
102. of
103. Object
104. .
105. entries
```

```
106. (  
107. artifacts  
108. ))  
109. {  
110. const  
111. address  
112. =  
113. artifact  
114. .  
115. networks  
116. [  
117. networkID  
118. ].  
119. address  
120. ;  
121. const  
122. contract  
123. =  
124. new  
125. web3  
126. .  
127. eth  
128. .  
129. Contract  
130. (  
131. artifact  
132. .  
133. abi  
134. ,  
135. address  
136. );  
137. contracts  
138. [  
139. contractName  
140. ]  
141. =  
142. contract  
143. ;  
144. }  
145. }  
146. catch  
147. (  
148. err  
149. )  
150. {  
151. contracts  
152. =  
153. null  
154. ;  
155. console  
156. .  
157. error  
158. (  
159. err  
160. );  
161. }  
162. Updates contracts when the account or network is changed  
163. useEffect  
164. (()  
165. =>  
166. {  
167. const  
168. events  
169. =  
170. [  
171. "chainChanged"  
172. ,  
173. "accountsChanged"
```

```
174. ];
175. const
176. handleChange
177. =
178. ()
179. =>
180. {
181.   init
182.   (
183.     state
184.     .
185.     artifacts
186.   );
187. };
188. events
189. .
190. forEach
191. (
192.   e
193.   =>
194.     window
195.     .
196.     ethereum
197.     .
198.     on
199.     (
200.       e
201.       ,
202.       handleChange
203.     ));
204. return
205. ()
206. =>
207. {
208.   events
209.   .
210.   forEach
211.   (
212.     e
213.     =>
214.       window
215.       .
216.       ethereum
217.       .
218.       removeListener
219.       (
220.         e
221.         ,
222.         handleChange
223.       ));
224. };
225. },
226. [
227.   init
228.   ,
229.   state
230.   .
231.   artifacts
232. ];
233. MetaMask injects a global API at window.ethereum
234. . 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.
235. window.ethereum.on
236. will detect if the network or account has changed and re-initialize web3
237. , accounts
238. , networkID
239. , and contracts
240. . As detailed in the MetaMask 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 and sdk . Stepping through the code:

```
1. AnAuth
2. object is created with your Infura project id, secret, and wallet private key that you added to env.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 a web3
17. and artifacts
18. instance from EthContext
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
```

```
61. =
62. async
63. (
64. publicAddress
65. )
66. =>
67. {
68. const
69. {
70. assets
71. }
72. =
73. await
74. sdk
75. .
76. getNFTs
77. ({
78. publicAddress
79. ,
80. includeMetadata
81. :
82. true
83. ,
84. });
85. const
86. filtered
87. =
88. [];
89. for
90. (
91. const
92. asset
93. of
94. assets
95. )
96. {
97. if
98. (
99. await
100. contracts
101. .
102. Marketplace
103. .
104. methods
105. .
106. isRentableNFT
107. (
108. asset
109. .
110. contract
111. ).
112. call
113. ());
114. {
115. filtered
116. .
117. push
118. (
119. asset
120. );
121. }
122. }
123. return
124. filtered
125. ;
126. };
127. To break this down further,
128. const
```

```

129. {
130. assets
131. }
132. =
133. await
134. sdk
135. .
136. getNFTs
137. ({
138. publicAddress
139. ,
140. includeMetadata
141. :
142. true
143. ,
144. });
145. sdk.getNFTs
146. retrieves all the NFTs and metadata of publicAddress
147. .
148. if
149. (
150. await
151. contracts
152. .
153. Marketplace
154. .
155. methods
156. .
157. isRentableNFT
158. (
159. asset
160. .
161. contract
162. ).
163. call
164. ())
165. {
166. filtered
167. .
168. push
169. (
170. asset
171. );
172. }
173. contracts.Marketplace.methods.isRentableNFT
174. is how we call the isRentableNFT
175. function on the smart contract. Note that we got the contracts.Marketplace
176. abstract by using the EthProvider
177. through useEth()
178. like so:
179. const
180. {
181. state
182. :
183. {
184. contracts
185. },
186. }
187. =
188. useEth
189. ();

```

client/src/contexts/MarketplaceContext/index.js



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

1. updateListings


```

2. is called whenever an NFT is listed, unlisted, or rented. It gets the image to be displayed by calling using the tokenURI
3. function from the NFT smart contract
4. const
5. nftContract
6. =
7. new
8. web3
9. .
10. eth
11. .
12. Contract
13. (
14. artifacts
15. .
16. RentableNft
17. .
18. abi
19. ,
20. nftContractAddress
21. );
22. const
23. tokenUri
24. =
25. await
26. nftContract
27. .
28. methods
29. .
30. tokenURI
31. (
32. tokenId
33. ).
34. call
35. ();
36. let
37. tokenUriRes
38. ;
39. try
40. {
41. tokenUriRes
42. =
43. await
44. (
45. await
46. fetch
47. (
48. getIpfsGatewayUri
49. (
50. tokenUri
51. )))
52. json
53. ();
54. }
55. catch
56. (
57. err
58. )
59. {
60. console
61. .
62. error
63. (
64. "Bad uri"
65. );
66. }
67. After resolving the image data, updateListings
68. will transform all the variables into the correct type (i.e., use parseInt
69. to convert pricePerDay

```

```
70. from astring
71. to anint
72. ). The finalupdateListings
73. looks like this:
74. const
75. updateListings
76. =
77. useCallback
78. (
79. async
80. ()
81. =>
82. {
83. if
84. (
85. marketplaceContract
86. )
87. {
88. const
89. res
90. =
91. await
92. marketplaceContract
93. .
94. methods
95. .
96. getAllListings
97. ().
98. call
99. ();
100. const
101. listingsExtendedTransformed
102. =
103. {};
104. const
105. listingsExtended
106. =
107. await
108. Promise
109. .
110. all
111. (
112. res
113. .
114. map
115. (
116. async
117. listing
118. =>
119. {
120. const
121. {
122. nftContract
123. :
124. nftContractAddress
125. ,
126. pricePerDay
127. :
128. pricePerDayStr
129. ,
130. startDateUNIX
131. :
132. startDateUnixStr
133. ,
134. endDateUNIX
135. :
136. endDateUnixStr
137. ,
```

```
138. expires
139. :
140. expiresStr
141. ,
142. tokenId
143. ,
144. owner
145. ,
146. user
147. }
148. =
149. listing
150. ;
151. const
152. nftContract
153. =
154. new
155. web3
156. .
157. eth
158. .
159. Contract
160. (
161. artifacts
162. .
163. RentableNft
164. .
165. abi
166. ,
167. nftContractAddress
168. );
169. const
170. tokenUri
171. =
172. await
173. nftContract
174. .
175. methods
176. .
177. tokenURI
178. (
179. tokenId
180. ).
181. call
182. ();
183. let
184. tokenUriRes
185. ;
186. try
187. {
188. tokenUriRes
189. =
190. await
191. (
192. await
193. fetch
194. (
195. getIpfsGatewayUri
196. (
197. tokenUri
198. )))
199. json
200. ();
201. }
202. catch
203. (
204. err
205. )
```

```
206. {
207. console
208. .
209. error
210. (
211. "Bad uri"
212. );
213. }
214. // const noUser = parseInt(user) !== 0;
215. const
216. pricePerDay
217. =
218. parseInt
219. (
220. pricePerDayStr
221. );
222. const
223. startDateUnix
224. =
225. parseInt
226. (
227. startDateUnixStr
228. );
229. const
230. endDateUnix
231. =
232. parseInt
233. (
234. endDateUnixStr
235. );
236. const
237. duration
238. =
239. endDateUnix
240. -
241. startDateUnix
242. ;
243. const
244. expires
245. =
246. parseInt
247. (
248. expiresStr
249. );
250. const
251. isOwner
252. =
253. owner
254. ===
255. accounts
256. [
257. 0
258. ];
259. const
260. isUser
261. =
262. user
263. ===
264. accounts
265. [
266. 0
267. ];
268. const
269. transformedData
270. =
271. {
272. pricePerDay
273. ,
```

```
274. startDateUnix
275. ,
276. endDateUnix
277. ,
278. duration
279. ,
280. expires
281. ,
282. user
283. };
284. const
285. listingExtended
286. =
287. {
288. ...
289. listing
290. ,
291. ...
292. transformedData
293. ,
294. nftContractAddress
295. ,
296. tokenUri
297. ,
298. tokenUriRes
299. ,
300. isOwner
301. ,
302. isUser
303. };
304. [
305. ...
306. Array
307. (
308. 8
309. ).
310. keys
311. (),
312. "nftContract"
313. ,
314. "startDateUNIX"
315. ,
316. "endDateUNIX"
317. ,
318. ].
319. forEach
320. (
321. i
322. =>
323. void
324. delete
325. listingExtended
326. [
327. i
328. ]);
329. if
330. (
331. listingsExtendedTransformed
332. [
333. nftContractAddress
334. ])
335. {
336. listingsExtendedTransformed
337. [
338. nftContractAddress
339. ][
340. tokenId
341. ]
```

```

342. =
343. transformedData
344. ;
345. }
346. else
347. {
348. listingsExtendedTransformed
349. [
350. nftContractAddress
351. ]
352. =
353. {
354. [
355. tokenId
356. ]
357. :
358. transformedData
359. };
360. }
361. return
362. listingExtended
363. ;
364. })
365. );
366. setListings
367. (
368. listingsExtended
369. );
370. setListingsTransformed
371. (
372. listingsExtendedTransformed
373. );
374. }
375. },
376. [
377. marketplaceContract
378. ,
379. web3
380. ?
381. .
382. eth
383. .
384. Contract
385. ,
386. artifacts
387. ?
388. .
389. RentableNft
390. .
391. abi
392. ,
393. accounts
394. ]
395. );
396. updateOwnedTokens
397. is called whenever a new NFT is minted to update your owned tokens.
398. It does this in two parts. First, it checks infura.active
399. 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
    call getOwnedRentableNfts
400. to retrieve all of user's NFTs and filter based on rentability.
401. if
402. (
403. infura
404. .
405. active
406. )
407. {
408. // Option 1 - Use infura nft sdk

```

```
409. if
410. (
411. accounts
412. &&
413. listingsTransformed
414. )
415. {
416. const
417. res
418. =
419. await
420. infura
421. .
422. getOwnedRentableNfts
423. (
424. accounts
425. [
426. 0
427. ]);
428. const
429. tokens
430. =
431. await
432. Promise
433. .
434. all
435. (
436. res
437. .
438. map
439. (
440. async
441. ele
442. =>
443. {
444. return
445. {
446. nftContractAddress
447. :
448. ele
449. .
450. contract
451. ,
452. tokenId
453. :
454. ele
455. .
456. tokenId
457. ,
458. tokenUriRes
459. :
460. ele
461. .
462. metadata
463. ,
464. listingData
465. :
466. listingsTransformed
467. [
468. ele
469. .
470. contract
471. ]
472. ?
473. .[
474. ele
475. .
476. tokenId
```

```

477. ]
478. }
479. })
480. )
481. setOwnedTokens
482. (
483. tokens
484. );
485. }
486. }
487. Otherwise, we're on ganache, our local test chain, and we will retrieve NFTs based on events emitted. Specifically,
    we will trackTransfer
488. , which is an event emitted when calling OpenZeppelin's mint
489. function on their ERC721
490. contract. Then, we will construct a token by creating a dictionary of the necessary values (nftContractAddress
491. , tokenId
492. , tokenUri
493. , etc).
494. if
495. (
496. rentableNftContract
497. &&
498. listingsTransformed
499. )
500. {
501. const
502. {
503. address
504. :
505. nftContractAddress
506. }
507. =
508. rentableNftContract
509. .
510. options
511. ;
512. // This only checks rentableNftContract.
513. const
514. mintEvents
515. =
516. await
517. rentableNftContract
518. .
519. getPastEvents
520. (
521. "Transfer"
522. ,
523. {
524. filter
525. :
526. {
527. from
528. :
529. "0x0000000000000000000000000000000000000000000000000000000000000000"
530. ,
531. to
532. :
533. accounts
534. [
535. 0
536. ]
537. },
538. fromBlock
539. :
540. 0
541. });
542. const
543. tokens

```



```
544. =
545. await
546. Promise
547. .
548. all
549. (
550. mintEvents
551. .
552. map
553. (
554. async
555. mintEvent
556. =>
557. {
558. const
559. {
560. tokenId
561. }
562. =
563. mintEvent
564. .
565. returnValues
566. ;
567. const
568. tokenUri
569. =
570. await
571. rentableNftContract
572. .
573. methods
574. .
575. tokenURI
576. (
577. tokenId
578. ).
579. call
580. ();
581. let
582. tokenUriRes
583. ;
584. try
585. {
586. tokenUriRes
587. =
588. await
589. (
590. await
591. fetch
592. (
593. getIpfsGatewayUri
594. (
595. tokenUri
596. ))).
597. json
598. ();
599. }
600. catch
601. (
602. err
603. )
604. {
605. console
606. .
607. error
608. (
609. "Bad uri"
610. );
611. }
```

```
612. return
613. {
614. nftContractAddress
615. ,
616. tokenId
617. ,
618. tokenUri
619. ,
620. tokenUriRes
621. ,
622. listingData
623. :
624. listingsTransformed
625. [
626. nftContractAddress
627. ]
628. ?
629. .[
630. tokenId
631. ]
632. };
633. })
634. );
635. setOwnedTokens
636. (
637. tokens
638. );
639. }
640. The final code looks like this:
641. const
642. updateOwnedTokens
643. =
644. useCallback
645. (
646. async
647. ()
648. =>
649. {
650. if
651. (
652. infura
653. .
654. active
655. )
656. {
657. // Option 1 - Use infura nft sdk
658. if
659. (
660. accounts
661. &&
662. listingsTransformed
663. )
664. {
665. const
666. res
667. =
668. await
669. infura
670. .
671. getOwnedRentableNfts
672. (
673. accounts
674. [
675. 0
676. ]);
677. const
678. tokens
679. =
```

```
680. await
681. Promise
682. .
683. all
684. (
685. res
686. .
687. map
688. (
689. async
690. ele
691. =>
692. {
693. return
694. {
695. nftContractAddress
696. :
697. ele
698. .
699. contract
700. ,
701. tokenId
702. :
703. ele
704. .
705. tokenId
706. ,
707. tokenUriRes
708. :
709. ele
710. .
711. metadata
712. ,
713. listingData
714. :
715. listingsTransformed
716. [
717. ele
718. .
719. contract
720. ]
721. ?
722. .[
723. ele
724. .
725. tokenId
726. ]
727. }
728. })
729. )
730. setOwnedTokens
731. (
732. tokens
733. );
734. }
735. }
736. else
737. {
738. // Option - 2 - Use contract events
739. // This is useful when using local network (ganache) or network otherwise unsupported
740. // docs.infura.io/infura/infura-custom-apis/nft-sdk/supported-networks
741. if
742. (
743. rentableNftContract
744. &&
745. listingsTransformed
746. )
747. {
```

```
748. const
749. {
750. address
751. :
752. nftContractAddress
753. }
754. =
755. rentableNftContract
756. .
757. options
758. ;
759. // This only checks rentableNftContract.
760. const
761. mintEvents
762. =
763. await
764. rentableNftContract
765. .
766. getPastEvents
767. (
768. "Transfer"
769. ,
770. {
771. filter
772. :
773. {
774. from
775. :
776. "0x0000000000000000000000000000000000000000000000000000000000000000"
777. ,
778. to
779. :
780. accounts
781. [
782. 0
783. ]
784. },
785. fromBlock
786. :
787. 0
788. });
789. const
790. tokens
791. =
792. await
793. Promise
794. .
795. all
796. (
797. mintEvents
798. .
799. map
800. (
801. async
802. mintEvent
803. =>
804. {
805. const
806. {
807. tokenId
808. }
809. =
810. mintEvent
811. .
812. returnValues
813. ;
814. const
815. tokenUri
```

```
816. =
817. await
818. rentableNftContract
819. .
820. methods
821. .
822. tokenURI
823. (
824. tokenId
825. ).
826. call
827. ();
828. let
829. tokenUriRes
830. ;
831. try
832. {
833. tokenUriRes
834. =
835. await
836. (
837. await
838. fetch
839. (
840. getIpsGatewayUri
841. (
842. tokenUri
843. )))
844. json
845. ();
846. }
847. catch
848. (
849. err
850. )
851. {
852. console
853. .
854. error
855. (
856. "Bad uri"
857. );
858. }
859. return
860. {
861. nftContractAddress
862. ,
863. tokenId
864. ,
865. tokenUri
866. ,
867. tokenUriRes
868. ,
869. listingData
870. :
871. listingsTransformed
872. [
873. nftContractAddress
874. ]
875. ?
876. .[
877. tokenId
878. ]
879. };
880. })
881. );
882. setOwnedTokens
883. (
```

```

884. tokens
885. );
886. }
887. }
888. },
889. [
890. rentableNftContract
891. ,
892. listingsTransformed
893. ,
894. accounts
895. ,
896. infura
897. .
898. getOwnedRentableNfts
899. ]);
900. mint
901. does exactly what it sounds like it does: mint an NFT! In order to call a function causes a change, we use.send()
902. instead ofcall()
903. . In this case, the caller ofmint
904. is the active account, which is indicated byfrom: accounts[0]
905. .
906. const
907. mint
908. =
909. async
910. (
911. tokenUri
912. )
913. =>
914. {
915. const
916. tx
917. =
918. await
919. rentableNftContract
920. .
921. methods
922. .
923. mint
924. (
925. tokenUri
926. ).
927. send
928. ({
929. from
930. :
931. accounts
932. [
933. 0
934. ]
935. });
936. if
937. (
938. tx
939. .
940. status
941. )
942. await
943. updateOwnedTokens
944. ();
945. };
946. list
947. lists the NFT and also collects the listing fee required to do so. As you can see, we addvalue: listingFee
948. , which will prompt the user to pay thelistingFee
949. when signing the transaction.
950. const
951. list

```

```
952. =
953. async
954. (
955. nftContractAddress
956. ,
957. tokenId
958. ,
959. price
960. ,
961. duration
962. )
963. =>
964. {
965. // Time values are in seconds
966. const
967. buffer
968. =
969. 30
970. ;
971. const
972. start
973. =
974. Math
975. .
976. ceil
977. (
978. Date
979. .
980. now
981. ()
982. /
983. 1000
984. )
985. +
986. buffer
987. ;
988. const
989. end
990. =
991. start
992. +
993. duration
994. ;
995. const
996. listingFee
997. =
998. await
999. marketplaceContract
1000. .
1001. methods
1002. .
1003. getListingFee
1004. ().
1005. call
1006. ();
1007. const
1008. tx
1009. =
1010. await
1011. marketplaceContract
1012. .
1013. methods
1014. .
1015. listNFT
1016. (
1017. nftContractAddress
1018. ,
1019. tokenId
```

```
1020. ,
1021. price
1022. ,
1023. start
1024. ,
1025. end
1026. ).
1027. send
1028. ({
1029. from
1030. :
1031. accounts
1032. [
1033. 0
1034. ],
1035. value
1036. :
1037. listingFee
1038. });
1039. if
1040. (
1041. tx
1042. .
1043. status
1044. )
1045. await
1046. updateListings
1047. ();
1048. };
1049. unlist
1050. 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.
1051. const
1052. unlist
1053. =
1054. async
1055. (
1056. nftContractAddress
1057. ,
1058. tokenId
1059. )
1060. =>
1061. {
1062. const
1063. nftContract
1064. =
1065. new
1066. web3
1067. .
1068. eth
1069. .
1070. Contract
1071. (
1072. artifacts
1073. .
1074. RentableNft
1075. .
1076. abi
1077. ,
1078. nftContractAddress
1079. );
1080. const
1081. expires
1082. =
1083. parseInt
1084. (
1085. await
1086. nftContract
```



```
1087. .
1088. methods
1089. .
1090. userExpires
1091. (
1092. tokenId
1093. ).
1094. call
1095. ());
1096. const
1097. {
1098. pricePerDay
1099. }
1100. =
1101. listingsTransformed
1102. [
1103. nftContractAddress
1104. ][
1105. tokenId
1106. ];
1107. const
1108. refund
1109. =
1110. Math
1111. .
1112. ceil
1113. ((
1114. expires
1115. -
1116. Date
1117. .
1118. now
1119. ()
1120. /
1121. 1000
1122. )
1123. /
1124. 60
1125. /
1126. 60
1127. /
1128. 24
1129. +
1130. 1
1131. )
1132. *
1133. pricePerDay
1134. ;
1135. const
1136. options
1137. =
1138. {
1139. from
1140. :
1141. accounts
1142. [
1143. 0
1144. ],
1145. value
1146. :
1147. Math
1148. .
1149. max
1150. (
1151. 0
1152. ,
1153. refund
1154. )
```

```
1155. };
1156. const
1157. tx
1158. =
1159. await
1160. marketplaceContract
1161. .
1162. methods
1163. .
1164. unlistNFT
1165. (
1166. nftContractAddress
1167. ,
1168. tokenId
1169. ).
1170. send
1171. (
1172. options
1173. );
1174. if
1175. (
1176. tx
1177. .
1178. status
1179. )
1180. await
1181. updateListings
1182. ();
1183. };
1184. rent
1185. is our final action! This allows users to rent available NFTs.
1186. const
1187. rent
1188. =
1189. async
1190. (
1191. nftContractAddress
1192. ,
1193. tokenId
1194. ,
1195. duration
1196. )
1197. =>
1198. {
1199. const
1200. {
1201. pricePerDay
1202. }
1203. =
1204. listingsTransformed
1205. [
1206. nftContractAddress
1207. ][
1208. tokenId
1209. ];
1210. const
1211. now
1212. =
1213. Math
1214. .
1215. ceil
1216. (
1217. Date
1218. .
1219. now
1220. ()
1221. /
1222. 1000
```

```
1223. );
1224. const
1225. expires
1226. =
1227. now
1228. +
1229. duration
1230. ;
1231. const
1232. numDays
1233. =
1234. (
1235. expires
1236. -
1237. now
1238. )
1239. /
1240. 60
1241. /
1242. 60
1243. /
1244. 24
1245. +
1246. 1
1247. ;
1248. const
1249. fee
1250. =
1251. Math
1252. .
1253. ceil
1254. (
1255. numDays
1256. *
1257. pricePerDay
1258. );
1259. const
1260. options
1261. =
1262. {
1263. from
1264. :
1265. accounts
1266. [
1267. 0
1268. ],
1269. value
1270. :
1271. fee
1272. };
1273. const
1274. tx
1275. =
1276. await
1277. marketplaceContract
1278. .
1279. methods
1280. .
1281. rentNFT
1282. (
1283. nftContractAddress
1284. ,
1285. tokenId
1286. ,
1287. expires
1288. ).
1289. send
1290. (
```

```

1291. options
1292. );
1293. if
1294. (
1295. tx
1296. .
1297. status
1298. )
1299. await
1300. updateListings
1301. ();
1302. };

```

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 with Web3.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 uses Next.js , check out our NFT Marketplace on Optimism [here](#) .

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 our [Discord](#) ! If you need help coding, start a discussion [here](#) . Lastly, don't forget to follow us on [Twitter](#) for the latest updates on all things Truffle.