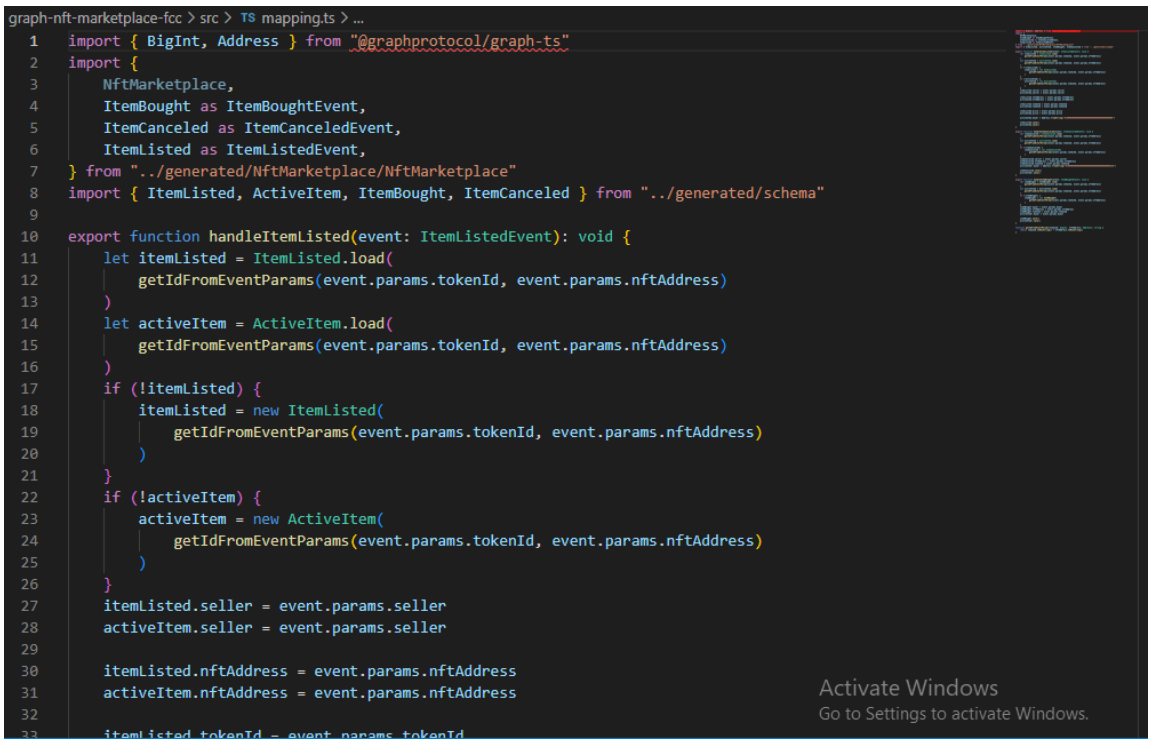
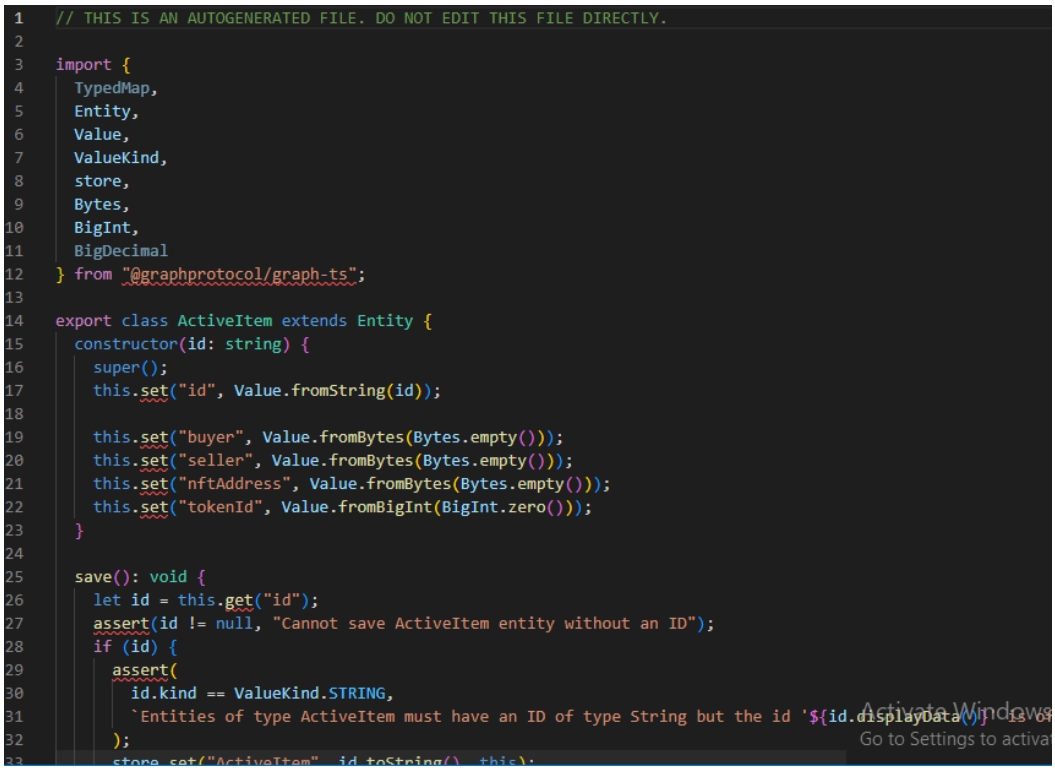
Graph-nft-marketplace-fcc

- src/mapping.ts




```ts
import { BigInt, Address } from "@graphprotocol/graph-ts"
import {
    NftMarketplace,
    ItemBought as ItemBoughtEvent,
    ItemCanceled as ItemCanceledEvent,
    ItemListed as ItemListedEvent,
} from "../generated/NftMarketplace/NftMarketplace"
import { ItemListed, ActiveItem, ItemBought, ItemCanceled } from "../generated/schema"

export function handleItemListed(event: ItemListedEvent): void {
    let itemListed = ItemListed.load(
        getIdFromEventParams(event.params.tokenId, event.params.nftAddress)
    )
    let activeItem = ActiveItem.load(
        getIdFromEventParams(event.params.tokenId, event.params.nftAddress)
    )
    if (!itemListed) {
        itemListed = new ItemListed(
            getIdFromEventParams(event.params.tokenId, event.params.nftAddress)
        )
    }
    if (!activeItem) {
        activeItem = new ActiveItem(
            getIdFromEventParams(event.params.tokenId, event.params.nftAddress)
        )
    }
    itemListed.seller = event.params.seller
    activeItem.seller = event.params.seller

    itemListed.nftAddress = event.params.nftAddress
    activeItem.nftAddress = event.params.nftAddress

    itemListed.tokenId = event.params.tokenId
```

- generated/schema.ts




```ts
// THIS IS AN AUTOGENERATED FILE. DO NOT EDIT THIS FILE DIRECTLY.

import {
  TypedMap,
  Entity,
  Value,
  ValueKind,
  store,
  Bytes,
  BigInt,
  BigDecimal
} from "@graphprotocol/graph-ts";

export class ActiveItem extends Entity {
  constructor(id: string) {
    super();
    this.set("id", Value.fromString(id));

    this.set("buyer", Value.fromBytes(Bytes.empty()));
    this.set("seller", Value.fromBytes(Bytes.empty()));
    this.set("nftAddress", Value.fromBytes(Bytes.empty()));
    this.set("tokenId", Value.fromBigInt(BigInt.zero()));
  }

  save(): void {
    let id = this.get("id");
    assert(id != null, "Cannot save ActiveItem entity without an ID");
    if (id) {
      assert(
        id.kind == ValueKind.STRING,
        `Entities of type ActiveItem must have an ID of type String but the id '${id.displayData()}` is o
      );
      store.set("ActiveItem", id.toString(), this);
```

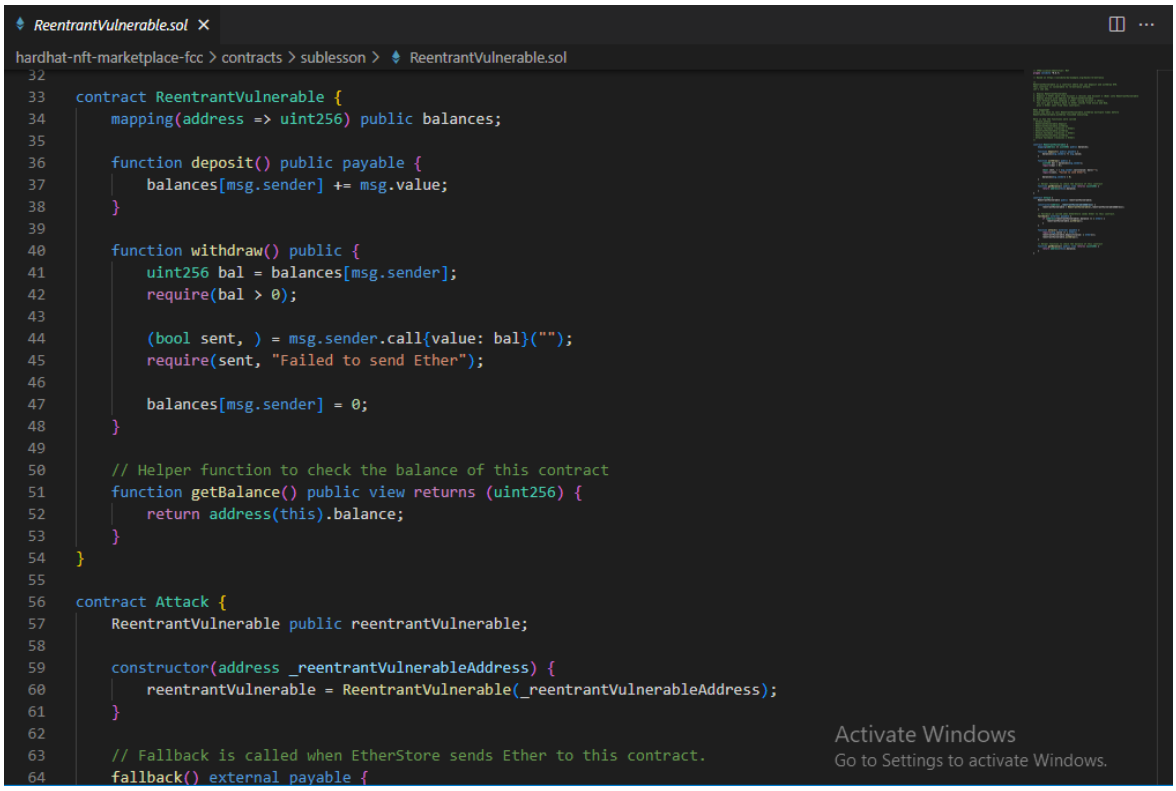
- NftMarketplace/NftMarketplace.ts

graph-nft-marketplace-fcc > generated > NftMarketplace > TS NftMarketplace.ts > ...

```ts
// THIS IS AN AUTOGENERATED FILE. DO NOT EDIT THIS FILE DIRECTLY.

import {
  ethereum,
  JSONValue,
  TypedMap,
  Entity,
  Bytes,
  Address,
  BigInt
} from "@graphprotocol/graph-ts";

export class ItemBought extends ethereum.Event {
  get params(): ItemBought__Params {
    return new ItemBought__Params(this);
  }
}

export class ItemBought__Params {
  _event: ItemBought;

  constructor(event: ItemBought) {
    this._event = event;
  }

  get buyer(): Address {
    return this._event.parameters[0].value.toAddress();
  }

  get nftAddress(): Address {
    return this._event.parameters[1].value.toAddress();
  }
```

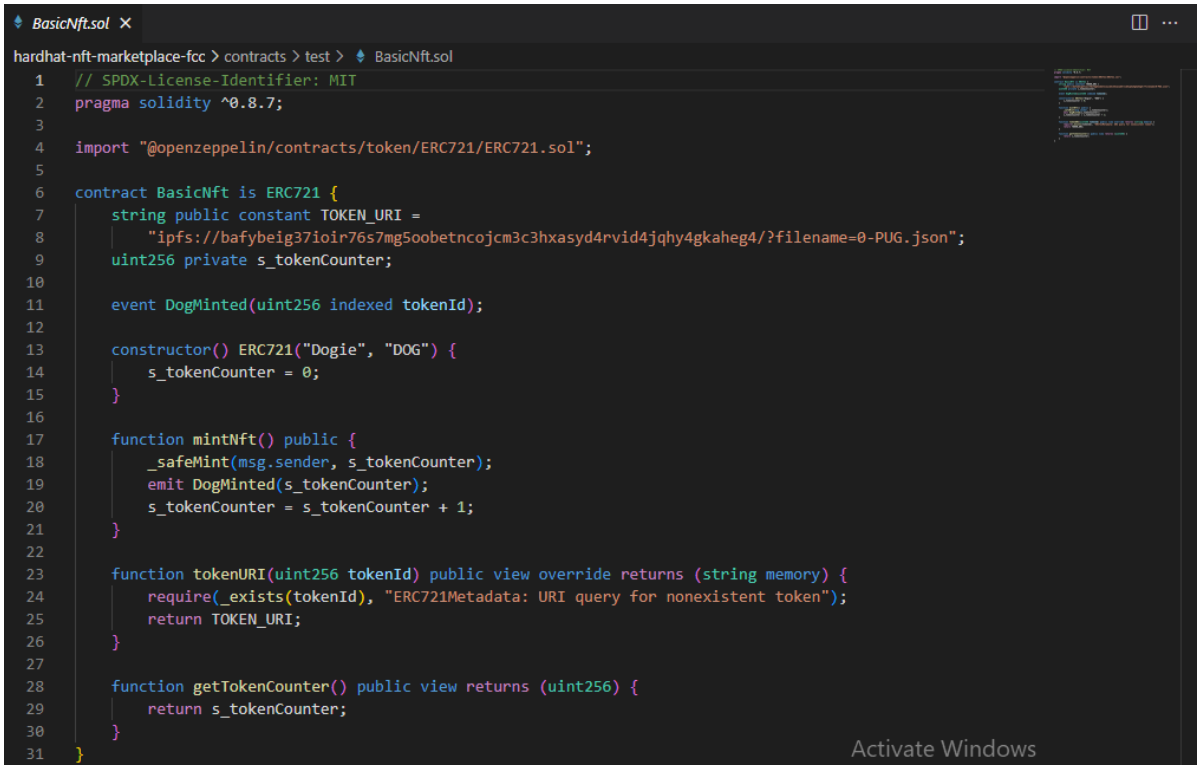
Hardhat-nft-marketplace-fcc

- contract/sublesson/ReentrantVulnerable.sol



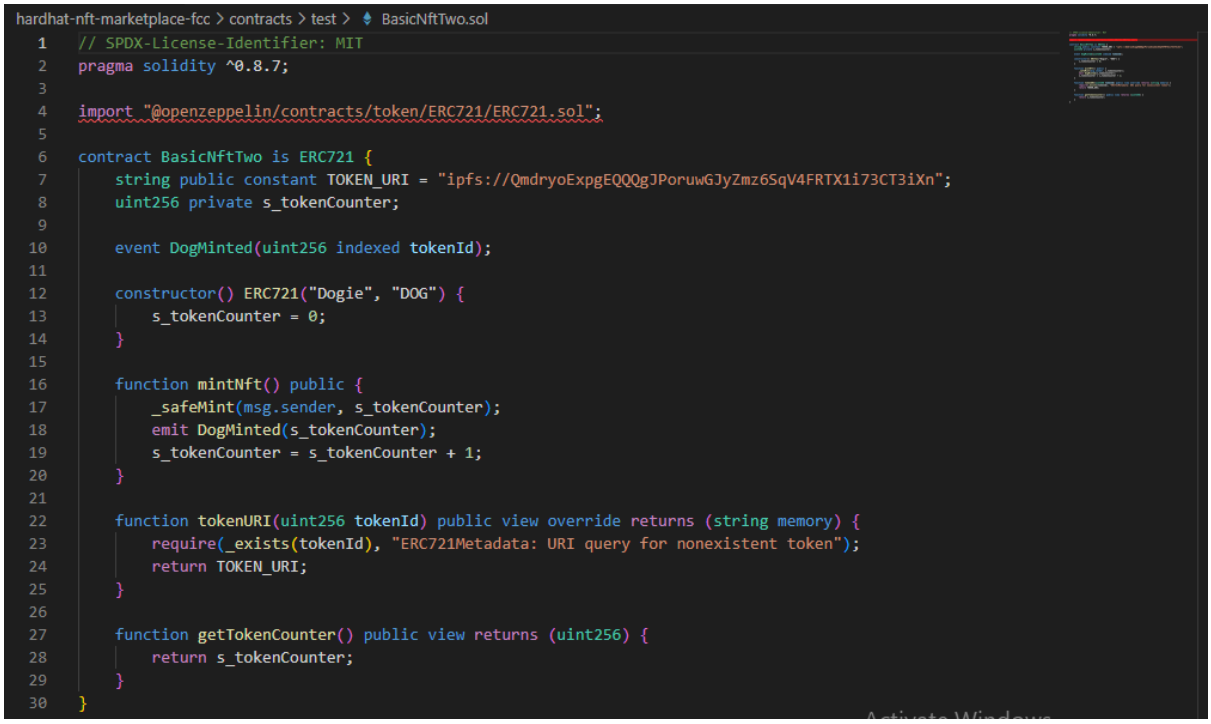
```
ReentrantVulnerable.sol
hardhat-nft-marketplace-fcc > contracts > sublesson > ReentrantVulnerable.sol
32
33  contract ReentrantVulnerable {
34      mapping(address => uint256) public balances;
35
36      function deposit() public payable {
37          balances[msg.sender] += msg.value;
38      }
39
40      function withdraw() public {
41          uint256 bal = balances[msg.sender];
42          require(bal > 0);
43
44          (bool sent, ) = msg.sender.call{value: bal}("");
45          require(sent, "Failed to send Ether");
46
47          balances[msg.sender] = 0;
48      }
49
50      // Helper function to check the balance of this contract
51      function getBalance() public view returns (uint256) {
52          return address(this).balance;
53      }
54  }
55
56  contract Attack {
57      ReentrantVulnerable public reentrantVulnerable;
58
59      constructor(address _reentrantVulnerableAddress) {
60          reentrantVulnerable = ReentrantVulnerable(_reentrantVulnerableAddress);
61      }
62
63      // Fallback is called when EtherStore sends Ether to this contract.
64      fallback() external payable {
```


- contract/test/BasicNft.sol



```
BasicNft.sol
hardhat-nft-marketplace-fcc > contracts > test > BasicNft.sol
1   // SPDX-License-Identifier: MIT
2   pragma solidity ^0.8.7;
3
4   import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
5
6   contract BasicNft is ERC721 {
7       string public constant TOKEN_URI =
8           "ipfs://bafybeig37ioir76s7mg5oobetncojcm3c3hxasyd4rvid4jqhy4gkaheg4/?filename=0-PUG.json";
9       uint256 private s_tokenCounter;
10
11      event DogMinted(uint256 indexed tokenId);
12
13      constructor() ERC721("Dogie", "DOG") {
14          s_tokenCounter = 0;
15      }
16
17      function mintNft() public {
18          _safeMint(msg.sender, s_tokenCounter);
19          emit DogMinted(s_tokenCounter);
20          s_tokenCounter = s_tokenCounter + 1;
21      }
22
23      function tokenURI(uint256 tokenId) public view override returns (string memory) {
24          require(_exists(tokenId), "ERC721Metadata: URI query for nonexistent token");
25          return TOKEN_URI;
26      }
27
28      function getTokenCounter() public view returns (uint256) {
29          return s_tokenCounter;
30      }
31  }
```

- contract/test/BasicNftTwo.sol




```sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

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

contract BasicNftTwo is ERC721 {
    string public constant TOKEN_URI = "ipfs://QmdryoExpgEQQQgJPoruwGJyZmz6SqV4FRTX1i73CT3iXn";
    uint256 private s_tokenCounter;

    event DogMinted(uint256 indexed tokenId);

    constructor() ERC721("Dogie", "DOG") {
        s_tokenCounter = 0;
    }

    function mintNft() public {
        _safeMint(msg.sender, s_tokenCounter);
        emit DogMinted(s_tokenCounter);
        s_tokenCounter = s_tokenCounter + 1;
    }

    function tokenURI(uint256 tokenId) public view override returns (string memory) {
        require(_exists(tokenId), "ERC721Metadata: URI query for nonexistent token");
        return TOKEN_URI;
    }

    function getTokenCounter() public view returns (uint256) {
        return s_tokenCounter;
    }
}
```

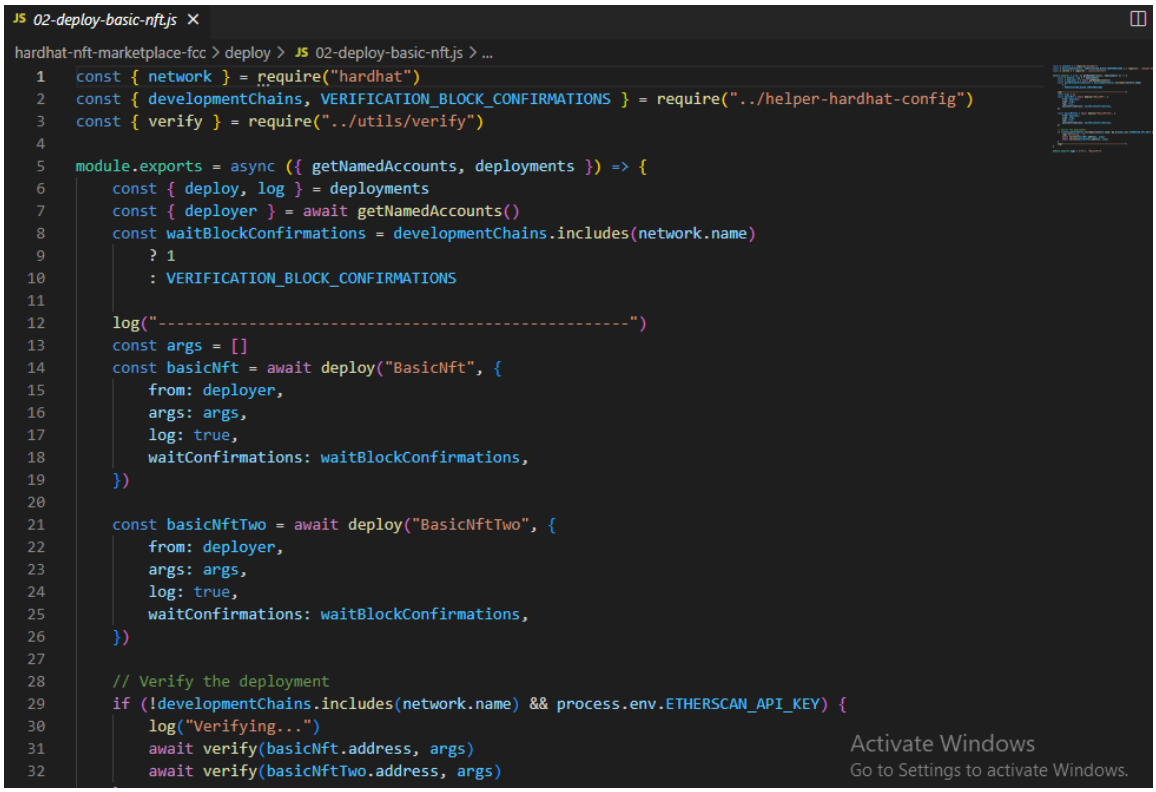
- contract/NftMarketplace.sol

```sol
error PriceNotMet(address nftAddress, uint256 tokenId, uint256 price);
error ItemNotForSale(address nftAddress, uint256 tokenId);
error NotListed(address nftAddress, uint256 tokenId);
error AlreadyListed(address nftAddress, uint256 tokenId);
error NoProceeds();
error NotOwner();
error NotApprovedForMarketplace();
error PriceMustBeAboveZero();

contract NftMarketplace is ReentrancyGuard {
    struct Listing {
        uint256 price;
        address seller;
    }

    event ItemListed(
        address indexed seller,
        address indexed nftAddress,
        uint256 indexed tokenId,
        uint256 price
    );

    event ItemCanceled(
        address indexed seller,
        address indexed nftAddress,
        uint256 indexed tokenId
    );

    event ItemBought(
        address indexed buyer,
        address indexed nftAddress,
```

- deploy/01-deploy-nft-marketplace.js

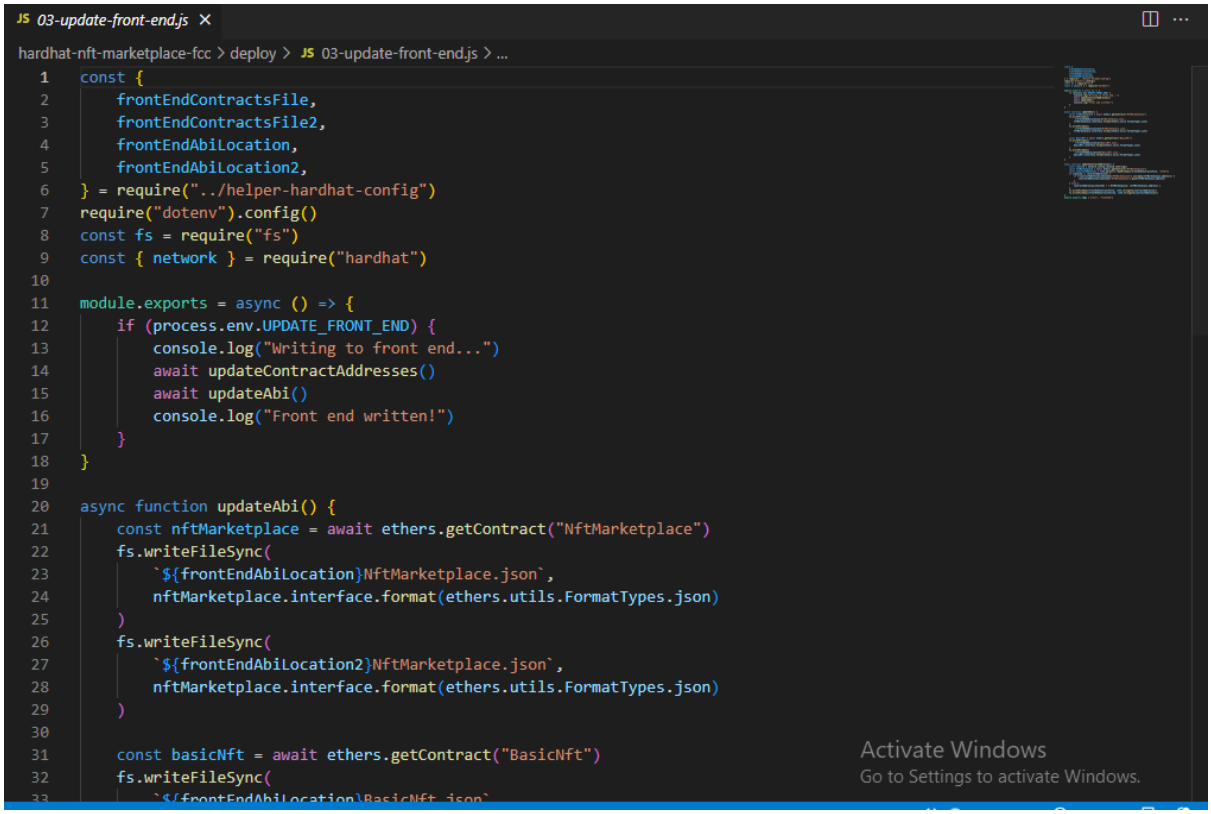
```
JS 01-deploy-nft-marketplace.js ×
hardhat-nft-marketplace-fcc > deploy > JS 01-deploy-nft-marketplace.js > ...
1   const { network } = require("hardhat")
2   const { developmentChains, VERIFICATION_BLOCK_CONFIRMATIONS } = require("../helper-hardhat-config")
3   const { verify } = require("../utils/verify")
4
5   module.exports = async ({ getNamedAccounts, deployments }) => {
6       const { deploy, log } = deployments
7       const { deployer } = await getNamedAccounts()
8       const waitBlockConfirmations = developmentChains.includes(network.name)
9           ? 1
10          : VERIFICATION_BLOCK_CONFIRMATIONS
11
12      log("----------------------------------------------------")
13      const arguments = []
14      const nftMarketplace = await deploy("NftMarketplace", {
15          from: deployer,
16          args: arguments,
17          log: true,
18          waitConfirmations: waitBlockConfirmations,
19      })
20
21      // Verify the deployment
22      if (!developmentChains.includes(network.name) && process.env.ETHERSCAN_API_KEY) {
23          log("Verifying...")
24          await verify(nftMarketplace.address, arguments)
25      }
26      log("----------------------------------------------------")
27  }
28
29  module.exports.tags = ["all", "nftmarketplace"]
```

- deploy/deploy-basic-nft.js



```
JS 02-deploy-basic-nft.js ×
hardhat-nft-marketplace-fcc > deploy > JS 02-deploy-basic-nft.js > ...
1   const { network } = require("hardhat")
2   const { developmentChains, VERIFICATION_BLOCK_CONFIRMATIONS } = require("../helper-hardhat-config")
3   const { verify } = require("../utils/verify")
4
5   module.exports = async ({ getNamedAccounts, deployments }) => {
6       const { deploy, log } = deployments
7       const { deployer } = await getNamedAccounts()
8       const waitBlockConfirmations = developmentChains.includes(network.name)
9           ? 1
10          : VERIFICATION_BLOCK_CONFIRMATIONS
11
12      log("----------------------------------------------------")
13      const args = []
14      const basicNft = await deploy("BasicNft", {
15          from: deployer,
16          args: args,
17          log: true,
18          waitConfirmations: waitBlockConfirmations,
19      })
20
21      const basicNftTwo = await deploy("BasicNftTwo", {
22          from: deployer,
23          args: args,
24          log: true,
25          waitConfirmations: waitBlockConfirmations,
26      })
27
28      // Verify the deployment
29      if (!developmentChains.includes(network.name) && process.env.ETHERSCAN_API_KEY) {
30          log("Verifying...")
31          await verify(basicNft.address, args)
32          await verify(basicNftTwo.address, args)
```

- deploy/update-front-end.js



```
const {
    frontEndContractsFile,
    frontEndContractsFile2,
    frontEndAbiLocation,
    frontEndAbiLocation2,
} = require("../helper-hardhat-config")
require("dotenv").config()
const fs = require("fs")
const { network } = require("hardhat")

module.exports = async () => {
    if (process.env.UPDATE_FRONT_END) {
        console.log("Writing to front end...")
        await updateContractAddresses()
        await updateAbi()
        console.log("Front end written!")
    }
}

async function updateAbi() {
    const nftMarketplace = await ethers.getContract("NftMarketplace")
    fs.writeFileSync(
        `${frontEndAbiLocation}NftMarketplace.json`,
        nftMarketplace.interface.format(ethers.utils.FormatTypes.json)
    )
    fs.writeFileSync(
        `${frontEndAbiLocation2}NftMarketplace.json`,
        nftMarketplace.interface.format(ethers.utils.FormatTypes.json)
    )

    const basicNft = await ethers.getContract("BasicNft")
    fs.writeFileSync(
```


- script/buy-item.js

```
const { ethers, network } = require("hardhat")
const { moveBlocks } = require("../utils/move-blocks")

const TOKEN_ID = 1

async function buyItem() {
    const nftMarketplace = await ethers.getContract("NftMarketplace")
    const basicNft = await ethers.getContract("BasicNft")
    const listing = await nftMarketplace.getListing(basicNft.address, TOKEN_ID)
    const price = listing.price.toString()
    const tx = await nftMarketplace.buyItem(basicNft.address, TOKEN_ID, { value: price })
    await tx.wait(1)
    console.log("NFT Bought!")
    if ((network.config.chainId = "31337")) {
        await moveBlocks(2, (sleepAmount = 1000))
    }
}

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

- scripts/cancel-item.js

cancel-item.js

hardhat-nft-marketplace-fcc > scripts > cancel-item.js > ...

```
const { ethers, network } = require("hardhat")
const { moveBlocks } = require("../utils/move-blocks")

const TOKEN_ID = 0

async function mintAndList() {
    const nftMarketplace = await ethers.getContract("NftMarketplace")
    const basicNft = await ethers.getContract("BasicNft")
    const tx = await nftMarketplace.cancelListing(basicNft.address, TOKEN_ID)
    await tx.wait(1)
    console.log("NFT Canceled!")
    if ((network.config.chainId = "31337")) {
        await moveBlocks(2, (sleepAmount = 1000))
    }
}

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

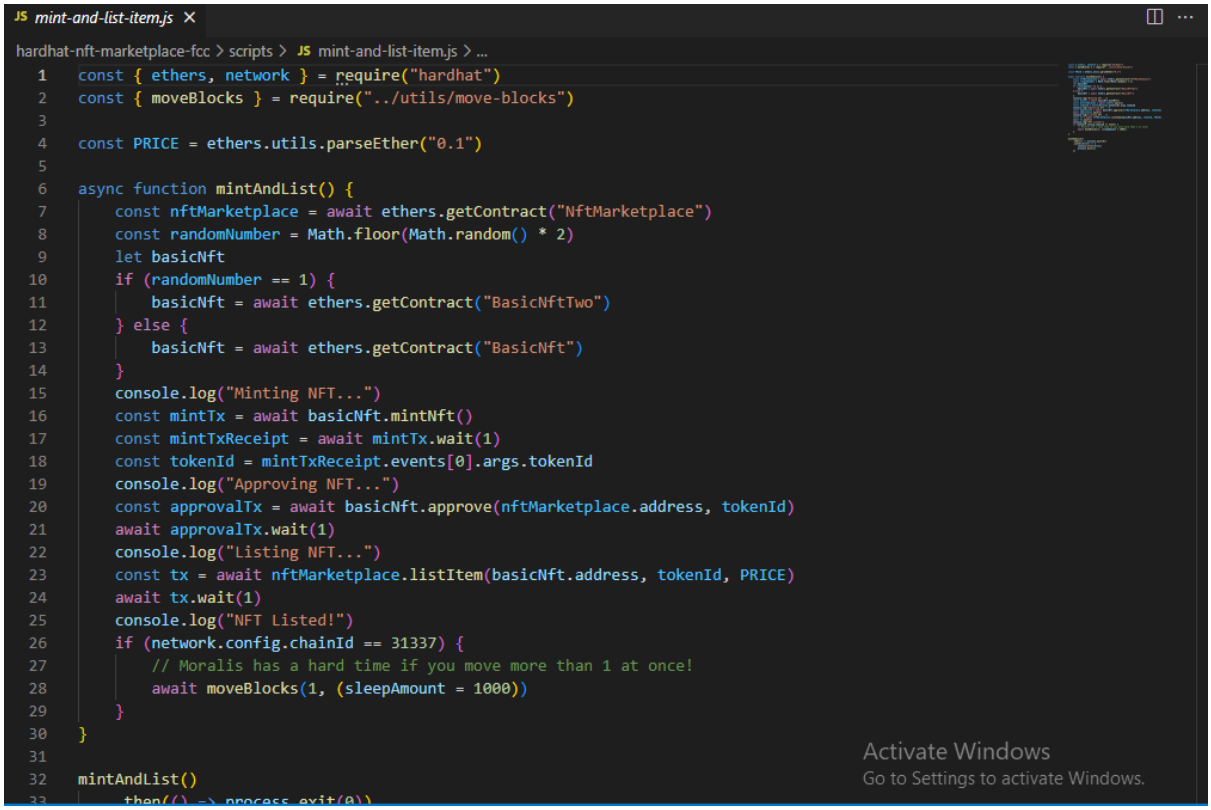
- scripts/mine.js

mine.js

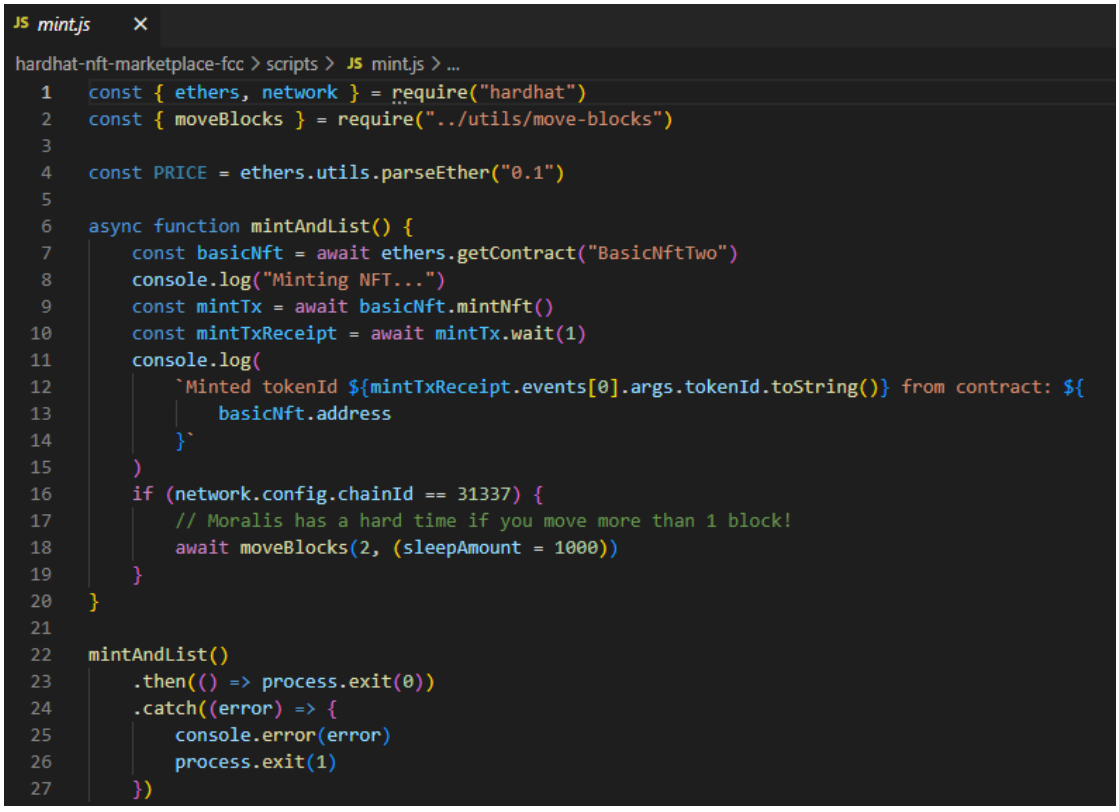
hardhat-nft-marketplace-fcc > scripts > mine.js > ...

```
const { moveBlocks } = require("../utils/move-blocks")

const BLOCKS = 5

async function mine() {
    await moveBlocks(BLOCKS)
}

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

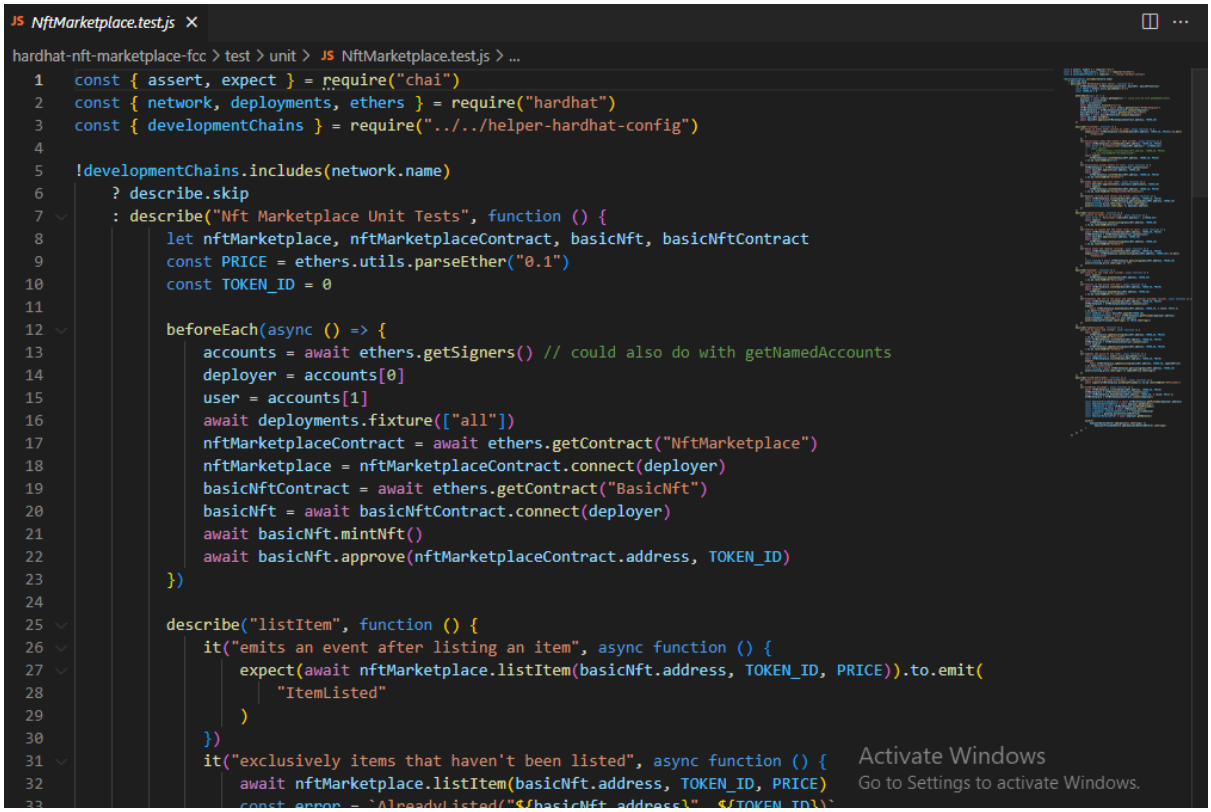
- scripts/mint-and-list-item.js



-scripts/mint.js

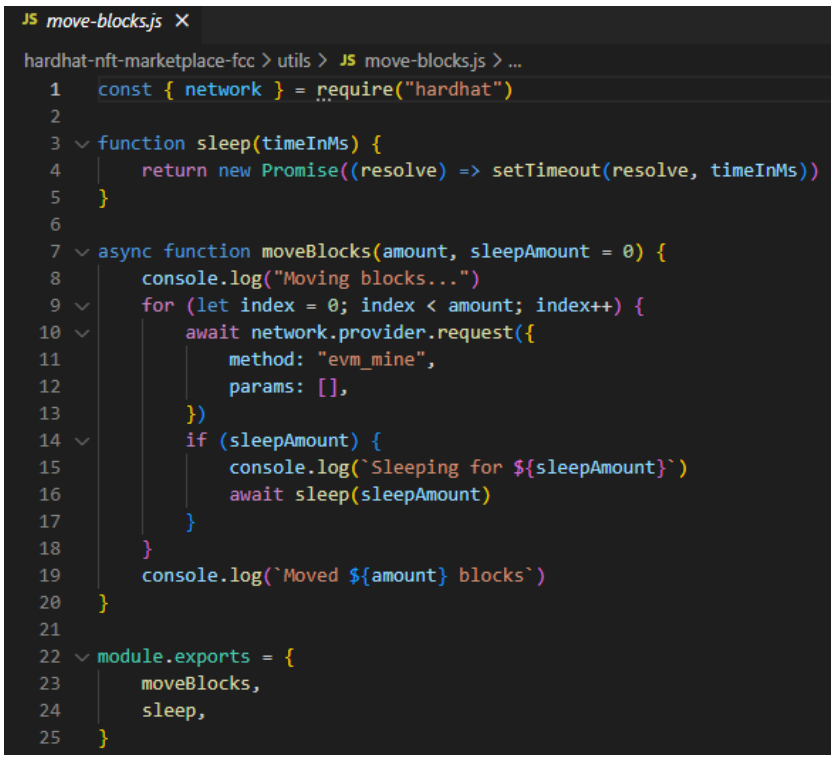


- test/unit/NftMarketplace.test.js



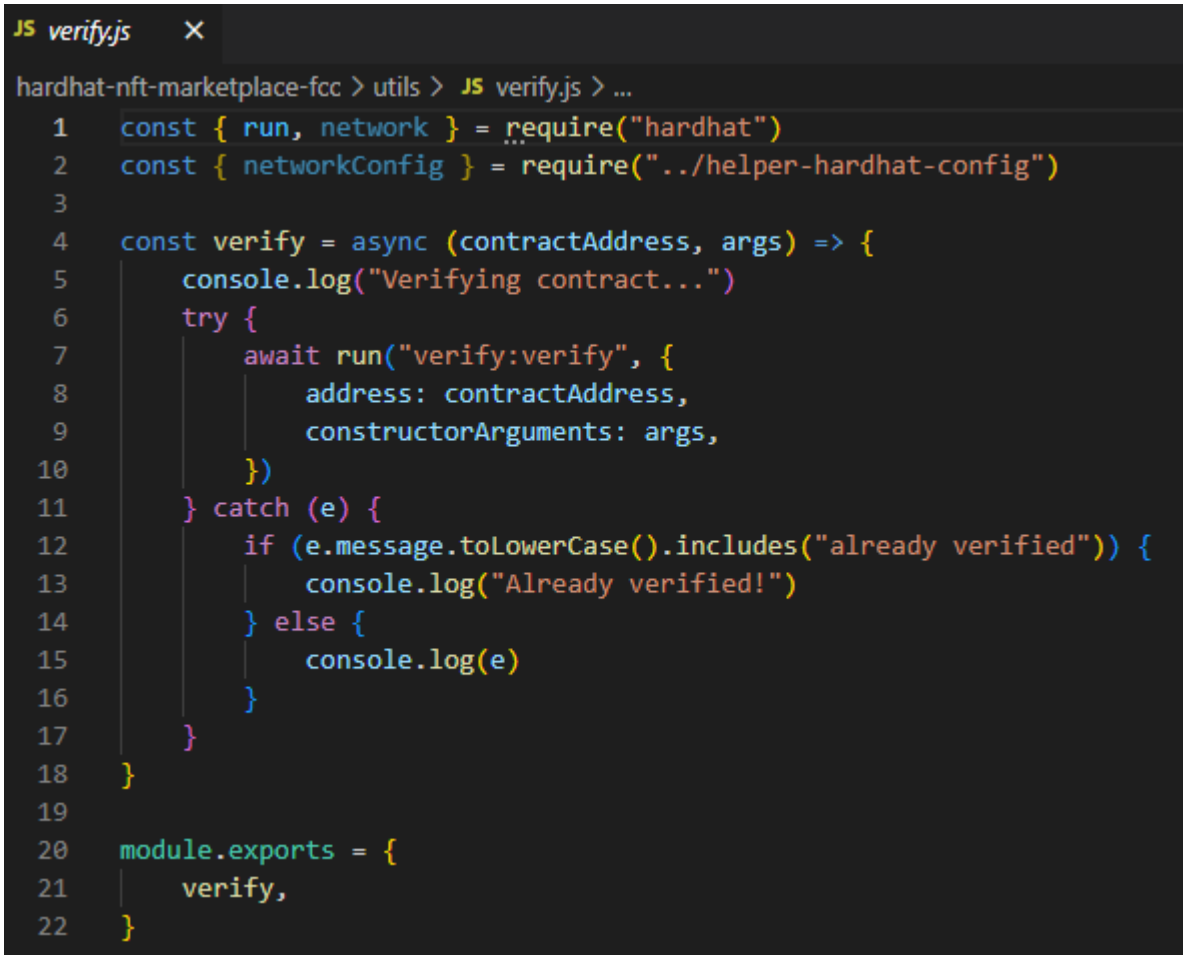
```
const { assert, expect } = require("chai")
const { network, deployments, ethers } = require("hardhat")
const { developmentChains } = require("../../helper-hardhat-config")

!developmentChains.includes(network.name)
    ? describe.skip
    : describe("Nft Marketplace Unit Tests", function () {
        let nftMarketplace, nftMarketplaceContract, basicNft, basicNftContract
        const PRICE = ethers.utils.parseEther("0.1")
        const TOKEN_ID = 0

        beforeEach(async () => {
            accounts = await ethers.getSigners() // could also do with getNamedAccounts
            deployer = accounts[0]
            user = accounts[1]
            await deployments.fixture(["all"])
            nftMarketplaceContract = await ethers.getContract("NftMarketplace")
            nftMarketplace = nftMarketplaceContract.connect(deployer)
            basicNftContract = await ethers.getContract("BasicNft")
            basicNft = await basicNftContract.connect(deployer)
            await basicNft.mintNft()
            await basicNft.approve(nftMarketplaceContract.address, TOKEN_ID)
        })

        describe("listItem", function () {
            it("emits an event after listing an item", async function () {
                expect(await nftMarketplace.listItem(basicNft.address, TOKEN_ID, PRICE)).to.emit(
                    "ItemListed"
                )
            })
            it("exclusively items that haven't been listed", async function () {
                await nftMarketplace.listItem(basicNft.address, TOKEN_ID, PRICE)
```


- utils/move-blocks.js



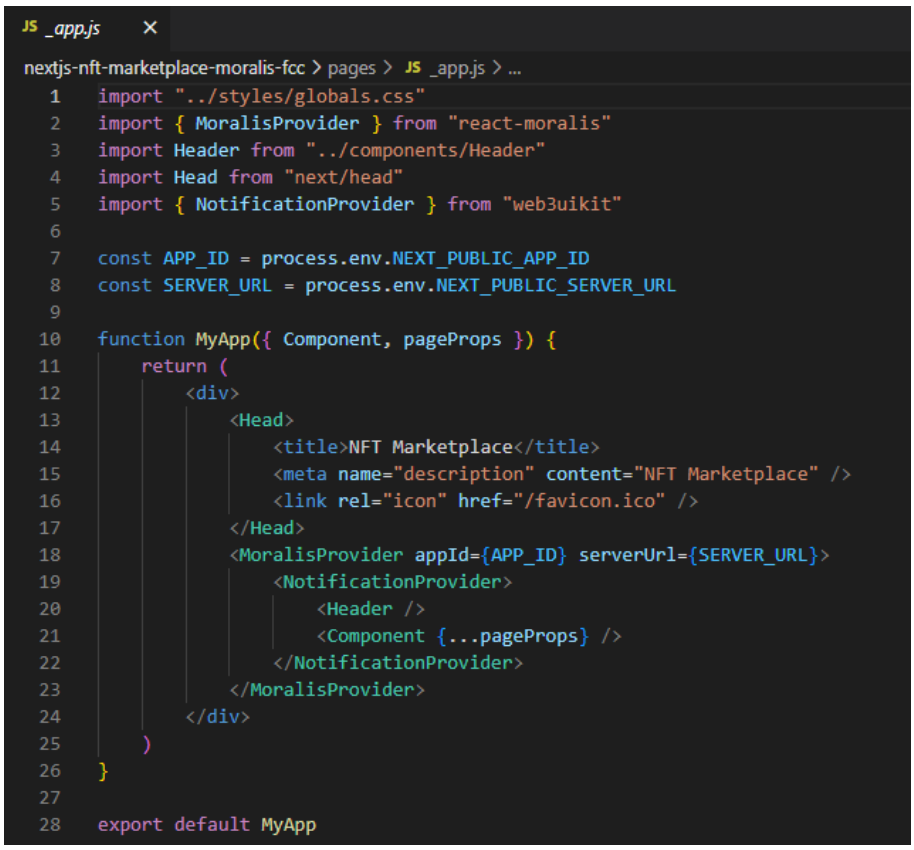
```
const { network } = require("hardhat")

function sleep(timeInMs) {
    return new Promise((resolve) => setTimeout(resolve, timeInMs))
}

async function moveBlocks(amount, sleepAmount = 0) {
    console.log("Moving blocks...")
    for (let index = 0; index < amount; index++) {
        await network.provider.request({
            method: "evm_mine",
            params: [],
        })
        if (sleepAmount) {
            console.log(`Sleeping for ${sleepAmount}`)
            await sleep(sleepAmount)
        }
    }
    console.log(`Moved ${amount} blocks`)
}

module.exports = {
    moveBlocks,
    sleep,
}
```

- utils/verify.js



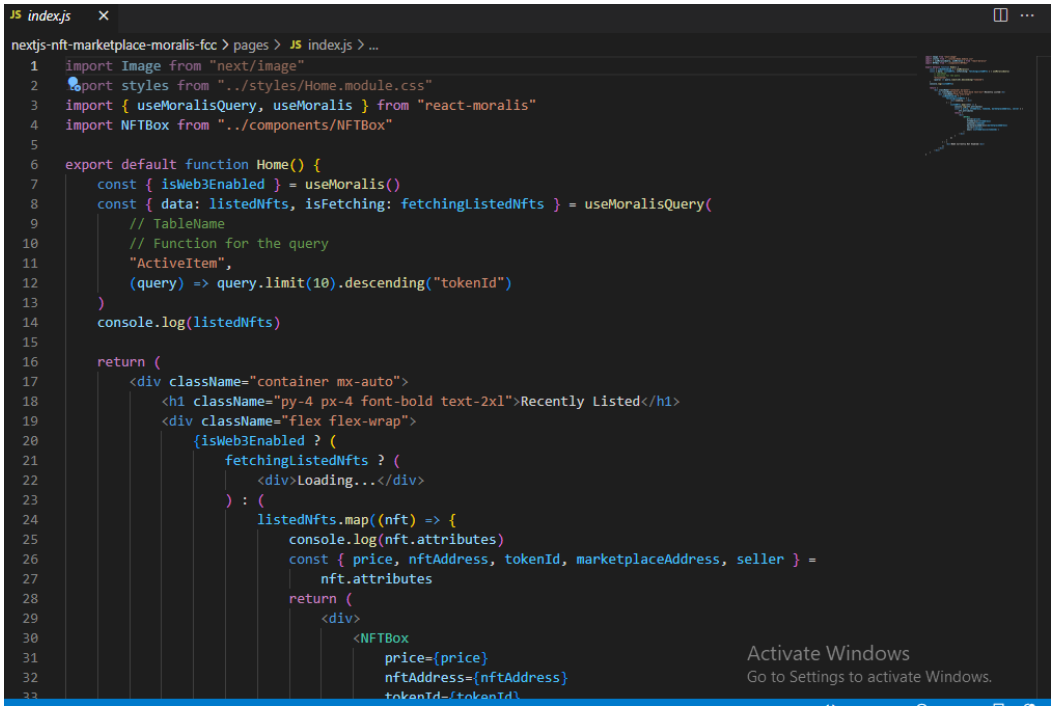
nextjs-nft-marketplace-moralis-fcc

- pages/_app.js



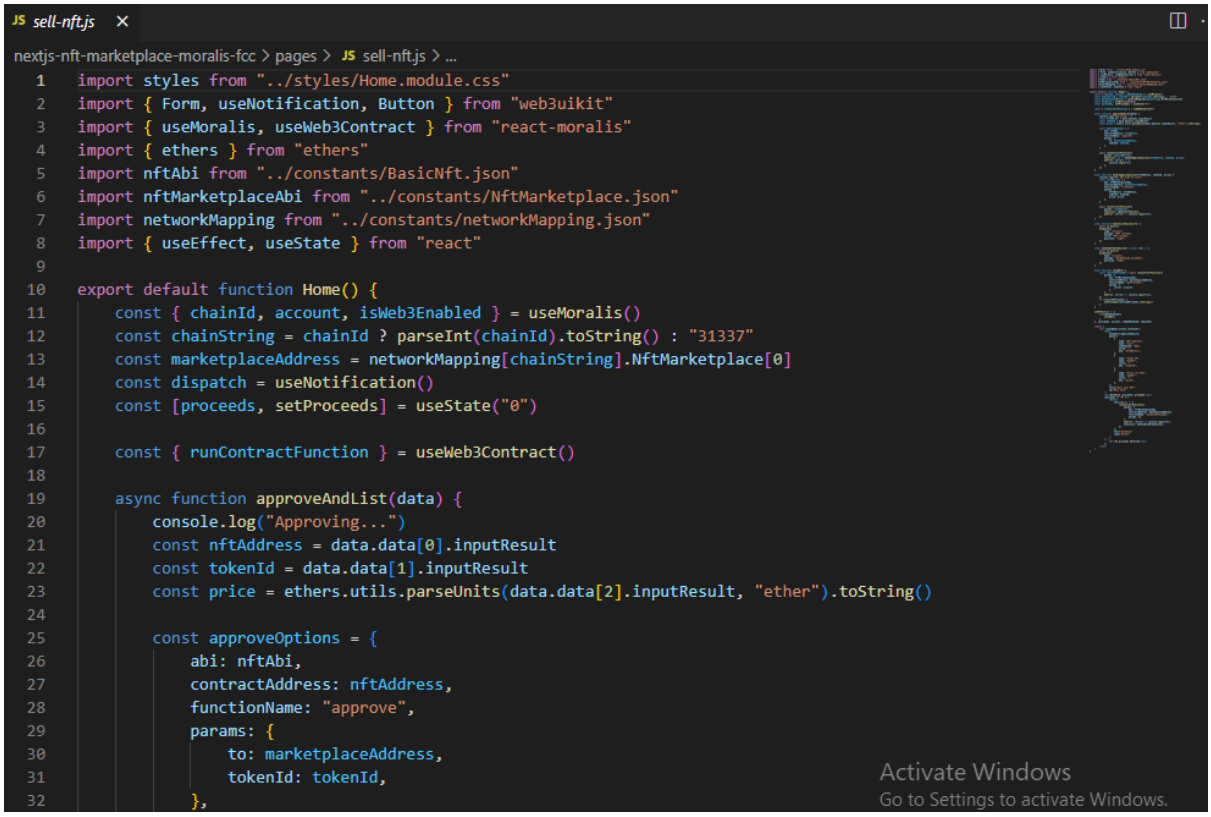
```
import "../styles/globals.css"
import { MoralisProvider } from "react-moralis"
import Header from "../components/Header"
import Head from "next/head"
import { NotificationProvider } from "web3uikit"

const APP_ID = process.env.NEXT_PUBLIC_APP_ID
const SERVER_URL = process.env.NEXT_PUBLIC_SERVER_URL

function MyApp({ Component, pageProps }) {
    return (
        <div>
            <Head>
                <title>NFT Marketplace</title>
                <meta name="description" content="NFT Marketplace" />
                <link rel="icon" href="/favicon.ico" />
            </Head>
            <MoralisProvider appId={APP_ID} serverUrl={SERVER_URL}>
                <NotificationProvider>
                    <Header />
                    <Component {...pageProps} />
                </NotificationProvider>
            </MoralisProvider>
        </div>
    )
}

export default MyApp
```

- pages/index.js



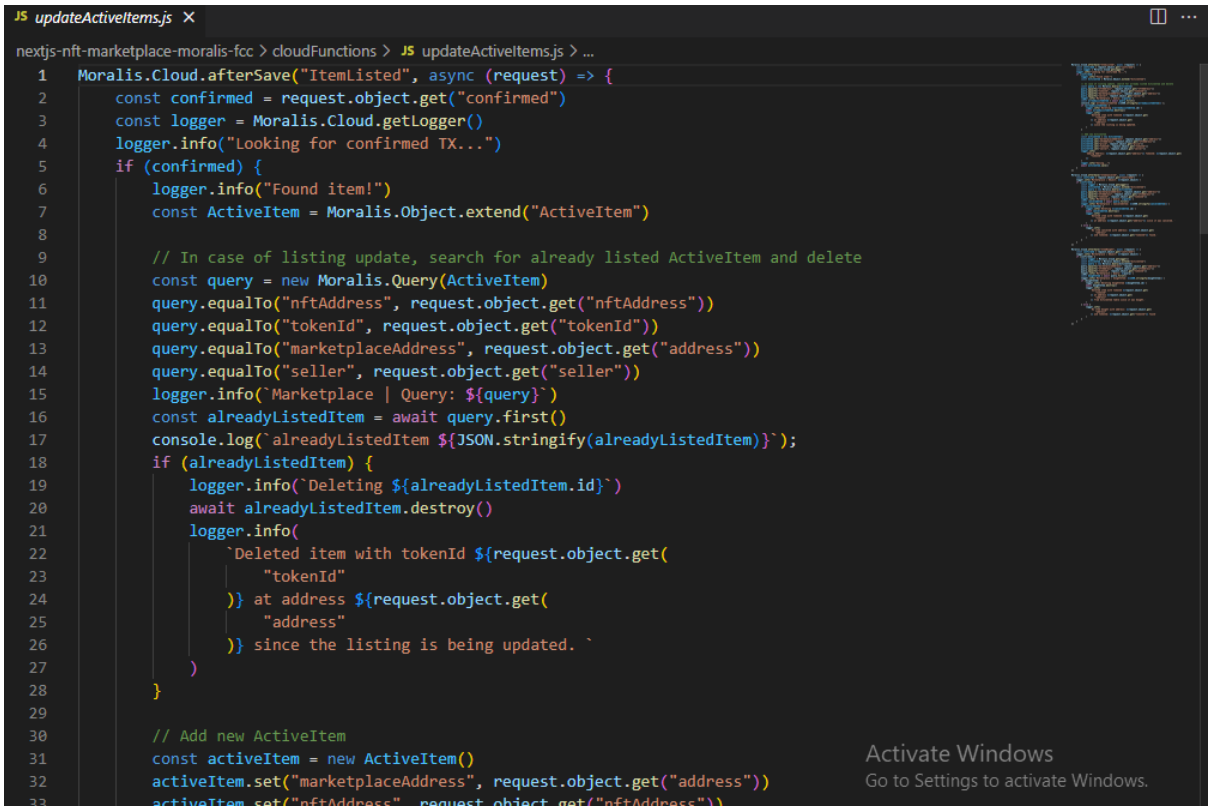
```
import Image from "next/image"
import styles from "../styles/Home.module.css"
import { useMoralisQuery, useMoralis } from "react-moralis"
import NFTBox from "../components/NFTBox"

export default function Home() {
    const { isWeb3Enabled } = useMoralis()
    const { data: listedNfts, isFetching: fetchingListedNfts } = useMoralisQuery(
        // TableName
        // Function for the query
        "ActiveItem",
        (query) => query.limit(10).descending("tokenId")
    )
    console.log(listedNfts)

    return (
        <div className="container mx-auto">
            <h1 className="py-4 px-4 font-bold text-2xl">Recently Listed</h1>
            <div className="flex flex-wrap">
                {isWeb3Enabled ? (
                    fetchingListedNfts ? (
                        <div>Loading...</div>
                    ) : (
                        listedNfts.map((nft) => {
                            console.log(nft.attributes)
                            const { price, nftAddress, tokenId, marketplaceAddress, seller } =
                                nft.attributes
                            return (
                                <div>
                                    <NFTBox
                                        price={price}
                                        nftAddress={nftAddress}
                                        tokenId={tokenId}
```

- pages/sell-nft.js



```
import styles from "../styles/Home.module.css"
import { Form, useNotification, Button } from "web3uikit"
import { useMoralis, useWeb3Contract } from "react-moralis"
import { ethers } from "ethers"
import nftAbi from "../constants/BasicNft.json"
import nftMarketplaceAbi from "../constants/NftMarketplace.json"
import networkMapping from "../constants/networkMapping.json"
import { useEffect, useState } from "react"

export default function Home() {
    const { chainId, account, isWeb3Enabled } = useMoralis()
    const chainString = chainId ? parseInt(chainId).toString() : "31337"
    const marketplaceAddress = networkMapping[chainString].NftMarketplace[0]
    const dispatch = useNotification()
    const [proceeds, setProceeds] = useState("0")

    const { runContractFunction } = useWeb3Contract()

    async function approveAndList(data) {
        console.log("Approving...")
        const nftAddress = data.data[0].inputResult
        const tokenId = data.data[1].inputResult
        const price = ethers.utils.parseUnits(data.data[2].inputResult, "ether").toString()

        const approveOptions = {
            abi: nftAbi,
            contractAddress: nftAddress,
            functionName: "approve",
            params: {
                to: marketplaceAddress,
                tokenId: tokenId,
            },
```


- cloudFunctions/updateActiveItems.js



```
Moralis.Cloud.afterSave("ItemListed", async (request) => {
    const confirmed = request.object.get("confirmed")
    const logger = Moralis.Cloud.getLogger()
    logger.info("Looking for confirmed TX...")
    if (confirmed) {
        logger.info("Found item!")
        const ActiveItem = Moralis.Object.extend("ActiveItem")

        // In case of listing update, search for already listed ActiveItem and delete
        const query = new Moralis.Query(ActiveItem)
        query.equalTo("nftAddress", request.object.get("nftAddress"))
        query.equalTo("tokenId", request.object.get("tokenId"))
        query.equalTo("marketplaceAddress", request.object.get("address"))
        query.equalTo("seller", request.object.get("seller"))
        logger.info(`Marketplace | Query: ${query}`)
        const alreadyListedItem = await query.first()
        console.log(`alreadyListedItem ${JSON.stringify(alreadyListedItem)}`);
        if (alreadyListedItem) {
            logger.info(`Deleting ${alreadyListedItem.id}`)
            await alreadyListedItem.destroy()
            logger.info(
                `Deleted item with tokenId ${request.object.get(
                    "tokenId"
                )} at address ${request.object.get(
                    "address"
                )} since the listing is being updated. `
            )
        }

        // Add new ActiveItem
        const activeItem = new ActiveItem()
        activeItem.set("marketplaceAddress", request.object.get("address"))
```

- components/Header.js

Header.js

nextjs-nft-marketplace-moralis-fcc > components > Header.js > ...

```jsx
import { ConnectButton } from "web3uikit"
import Link from "next/link"

export default function Header() {
    return (
        <nav className="p-5 border-b-2 flex flex-row justify-between items-center">
            <h1 className="py-4 px-4 font-bold text-3xl">NFT Marketplace</h1>
            <div className="flex flex-row items-center">
                <Link href="/">
                    <a className="mr-4 p-6">Home</a>
                </Link>
                <Link href="/sell-nft">
                    <a className="mr-4 p-6">Sell NFT</a>
                </Link>
                <ConnectButton moralisAuth={false} />
            </div>
        </nav>
    )
}
```

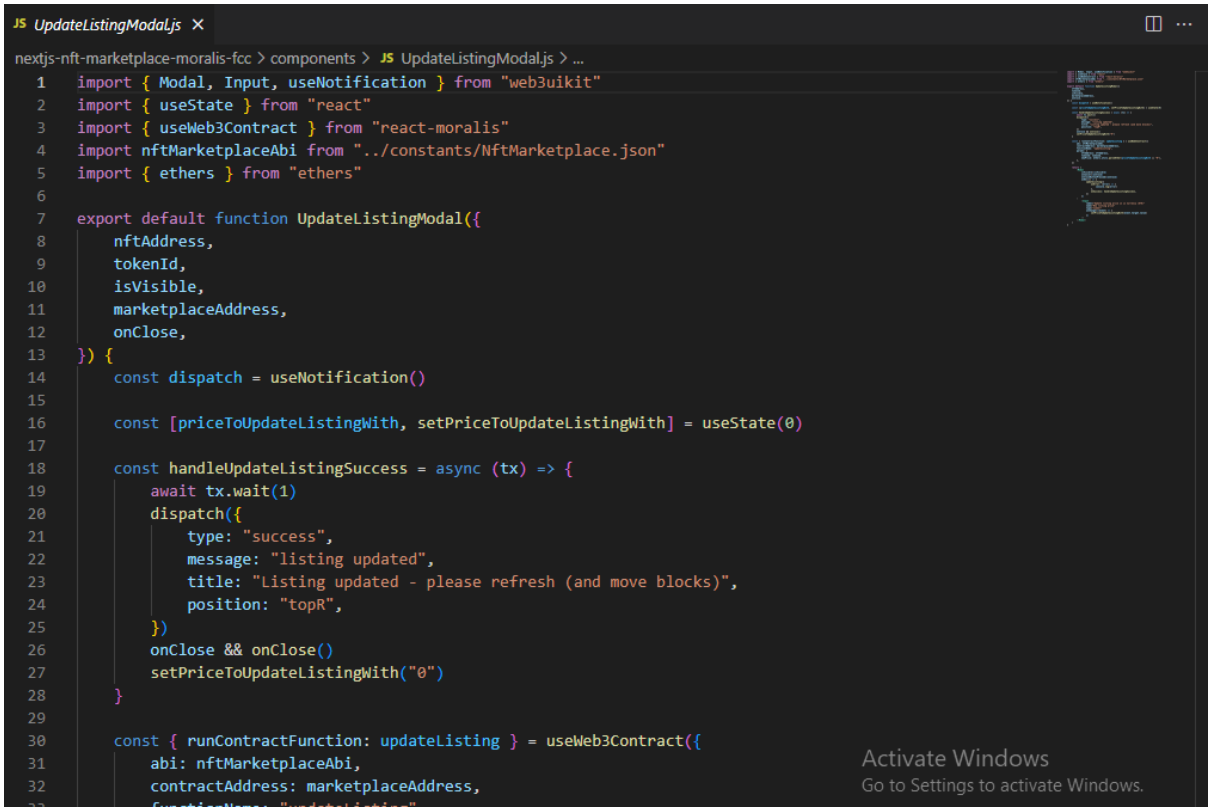
- component/NFTBox.js

NFTBox.js

nextjs-nft-marketplace-moralis-fcc > components > NFTBox.js > ...

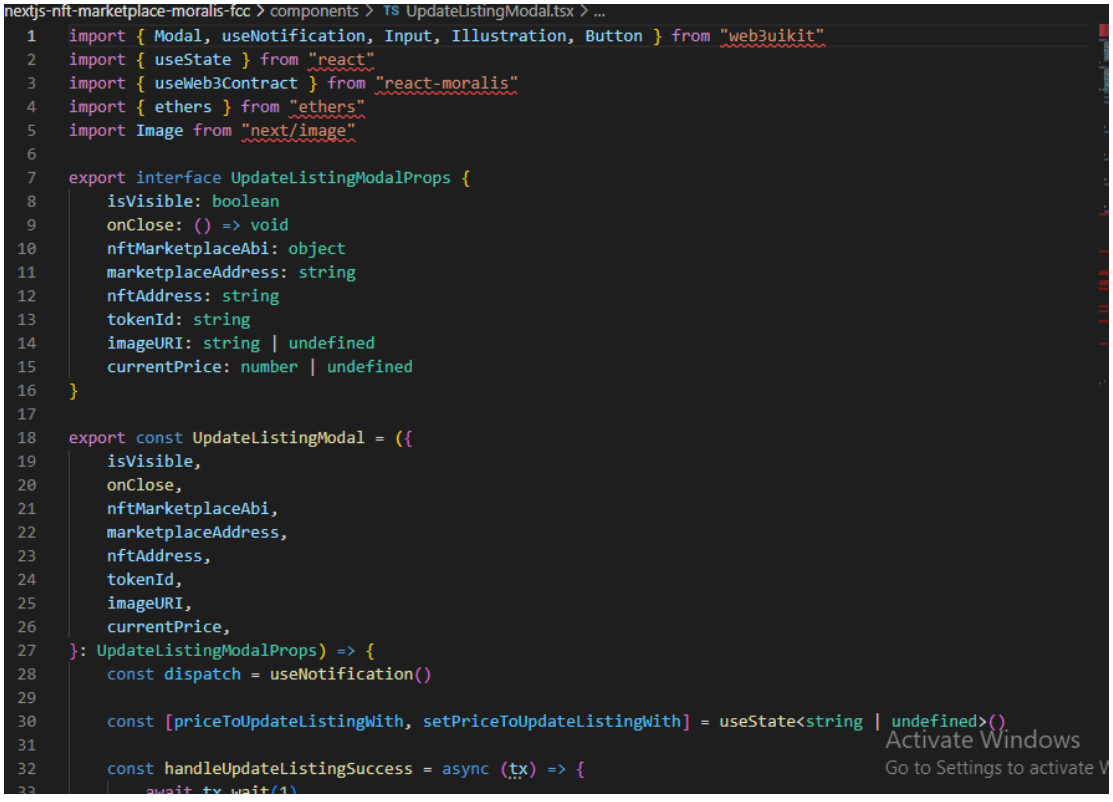
```jsx
import { useState, useEffect } from "react"
import { useWeb3Contract, useMoralis } from "react-moralis"
import nftMarketplaceAbi from "../constants/NftMarketplace.json"
import nftAbi from "../constants/BasicNft.json"
import Image from "next/image"
import { Card, useNotification } from "web3uikit"
import { ethers } from "ethers"
import UpdateListingModal from "./UpdateListingModal"

const truncateStr = (fullStr, strLen) => {
    if (fullStr.length <= strLen) return fullStr

    const separator = "..."
    const seperatorLength = separator.length
    const charsToShow = strLen - seperatorLength
    const frontChars = Math.ceil(charsToShow / 2)
    const backChars = Math.floor(charsToShow / 2)
    return (
        fullStr.substring(0, frontChars) +
        separator +
        fullStr.substring(fullStr.length - backChars)
    )
}

export default function NFTBox({ price, nftAddress, tokenId, marketplaceAddress, seller }) {
    const { isWeb3Enabled, account } = useMoralis()
    const [imageURI, setImageURI] = useState("")
    const [tokenName, setTokenName] = useState("")
    const [tokenDescription, setTokenDescription] = useState("")
    const [showModal, setShowModal] = useState(false)
    const hideModal = () => setShowModal(false)
    const dispatch = useNotification()
```

- components/UpdateListingModal.js



```
import { Modal, Input, useNotification } from "web3uikit"
import { useState } from "react"
import { useWeb3Contract } from "react-moralis"
import nftMarketplaceAbi from "../constants/NftMarketplace.json"
import { ethers } from "ethers"

export default function UpdateListingModal({
    nftAddress,
    tokenId,
    isVisible,
    marketplaceAddress,
    onClose,
}) {
    const dispatch = useNotification()

    const [priceToUpdateListingWith, setPriceToUpdateListingWith] = useState(0)

    const handleUpdateListingSuccess = async (tx) => {
        await tx.wait(1)
        dispatch({
            type: "success",
            message: "listing updated",
            title: "Listing updated - please refresh (and move blocks)",
            position: "topR",
        })
        onClose && onClose()
        setPriceToUpdateListingWith("0")
    }

    const { runContractFunction: updateListing } = useWeb3Contract({
        abi: nftMarketplaceAbi,
        contractAddress: marketplaceAddress,
```

-components/UpdateListingModal.tsx



```
import { Modal, useNotification, Input, Illustration, Button } from "web3uikit"
import { useState } from "react"
import { useWeb3Contract } from "react-moralis"
import { ethers } from "ethers"
import Image from "next/image"

export interface UpdateListingModalProps {
    isVisible: boolean
    onClose: () => void
    nftMarketplaceAbi: object
    marketplaceAddress: string
    nftAddress: string
    tokenId: string
    imageURI: string | undefined
    currentPrice: number | undefined
}

export const UpdateListingModal = ({
    isVisible,
    onClose,
    nftMarketplaceAbi,
    marketplaceAddress,
    nftAddress,
    tokenId,
    imageURI,
    currentPrice,
}: UpdateListingModalProps) => {
    const dispatch = useNotification()

    const [priceToUpdateListingWith, setPriceToUpdateListingWith] = useState<string | undefined>()

    const handleUpdateListingSuccess = async (tx) => {
```