1. **Project Information**

Project Topic: NFT Trade Platform

Project type: coding application

Project description: in this project I will demonstrate how to use different tools to develop a DApp NFT trade platform, which users can list out the NFT sales for other users to purchase after they have connected with their wallet. I will explore using Hardhat to develop the market and NFT smart contracts, the Graph to create my own subgraph for querying, and Next.js for the frontend.

Project team (solo team): Ko Wang, SID: 1155160670

1. **Introduction**

In this project, an NFT marketplace is built for users to buy and sell NFTs. This project can be separated into three parts: smart contracts, querying and the front end.

In the application website, the user needs to connect with their wallets first before any action. Before connecting their wallet, the page will only notify the user to connect.

After connecting, in the index page, they will see the NFT listings. Here they can see all the NFTs on sale with their price listed. They can click on the NFT which will direct them to the individual NFT listing page.

In the individual listing page, users can buy the listed NFT with the price of the listed price plus the gas fee. If the user is the owner of the NFT, they can update the sale price which costs them gas fee as well.

1. **Development**

The following are the implementation code and brief introduction to them. Please note that even though the frontend occupies most of the source code, most of the time is spent into both Hardhat and the Graph on system configuration.

\*\*all the code is present in the Github link to this project

* 1. Smart Contract

We need to deploy contracts related to the marketplace and the NFTs that would be sold. For simplicity, the same NFT will be minted 4 times for the owner as 4 tokens for sale. The tool in this section includes Hardhat, a development environment for Ethereum to create, compiling and deploying smart contracts. To work with NFTs in the project smart contracts, OpenZeppelin is also used as a library for creating our NFTs (ERC721), so we only need to set the IPFS file of the NFT before deploying the contract.

2 contracts are deployed using Hardhat: NFTMarket.sol as the market contract and CourseNFT.sol as our hardcoded NFT contract (it can be changed to other names or IPFS files, this is just an example).

For the NFT market contract, it contains a mapping of NFT listings. It also contains functions that handle the NFT listing, including creating, update, remove, read, and purchase listings. It also has functions for checking, including if the NFT is listed or not, and the price is valid whether the user is the owner of the NFT.

NFTMarket.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC721/IERC721.sol";

contract NFTMarket {

    struct Listing {

        uint256 price;

        address seller;

    }

    mapping(address => mapping(uint256 => Listing)) public listings;

    modifier isNFTOwner(address nftAddress, uint256 tokenId) {

        require(

            IERC721(nftAddress).ownerOf(tokenId) == msg.sender,

            "You are not the owner of this NFT"

        );

        \_;

    }

    modifier validPrice(uint256 \_price) {

        require(\_price > 0, "The price must be greater than 0");

        \_;

    }

    modifier isNotListed(address nftAddress, uint256 tokenId) {

        require(

            listings[nftAddress][tokenId].price == 0,

            "This NFT is already listed"

        );

        \_;

    }

    modifier isListed(address nftAddress, uint256 tokenId) {

        require(listings[nftAddress][tokenId].price > 0, "This NFT is not listed");

        \_;

    }

    event ListingCreated(

        address nftAddress,

        uint256 tokenId,

        uint256 price,

        address seller

    );

    event ListingCancelled(address nftAddress, uint256 tokenId, address seller);

    event ListingUpdated(

        address nftAddress,

        uint256 tokenId,

        uint256 newPrice,

        address seller

    );

    event ListingPurchased(

        address nftAddress,

        uint256 tokenId,

        address seller,

        address buyer

    );

    function createListing(

        address nftAddress,

        uint256 tokenId,

        uint256 price

    )

        external

        isNotListed(nftAddress, tokenId)

        isNFTOwner(nftAddress, tokenId)

        validPrice(price)

    {

        IERC721 nftContract = IERC721(nftAddress);

        require(

            nftContract.isApprovedForAll(msg.sender, address(this)) ||

                nftContract.getApproved(tokenId) == address(this),

            "This NFT is not approved"

        );

        listings[nftAddress][tokenId] = Listing({

            price: price,

            seller: msg.sender

        });

        emit ListingCreated(nftAddress, tokenId, price, msg.sender);

    }

    function cancelListing(address nftAddress, uint256 tokenId)

        external

        isListed(nftAddress, tokenId)

        isNFTOwner(nftAddress, tokenId)

    {

        delete listings[nftAddress][tokenId];

        emit ListingCancelled(nftAddress, tokenId, msg.sender);

    }

    function updateListing(

        address nftAddress,

        uint256 tokenId,

        uint256 newPrice

    )

        external

        isListed(nftAddress, tokenId)

        isNFTOwner(nftAddress, tokenId)

        validPrice(newPrice)

    {

        listings[nftAddress][tokenId].price = newPrice;

        emit ListingUpdated(nftAddress, tokenId, newPrice, msg.sender);

    }

    function purchaseListing(address nftAddress, uint256 tokenId)

        external

        payable

        isListed(nftAddress, tokenId)

    {

        Listing memory listing = listings[nftAddress][tokenId];

        require(msg.value == listing.price, "The NFT price and the purchase price does not match");

            delete listings[nftAddress][tokenId];

        IERC721(nftAddress).safeTransferFrom(

            listing.seller,

            msg.sender,

            tokenId

        );

        (bool sent, ) = payable(listing.seller).call{value: msg.value}("");

        require(sent, "The NFT price cannot be sent to the buyer");

        emit ListingPurchased(nftAddress, tokenId, listing.seller, msg.sender);

    }

}

The other file is the NFT contract. The ERC721 must be deployed before entering the market. Just for simplicity, 5 NFTs of the same IPFS file will be created and minted to the user that deployed that contract. The only function in the contract is to get the token URI, which returns IPFS file which is uploaded there previously.

CourseNFT.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.20;

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

contract CourseNFT is ERC721 {

    constructor() ERC721("CourseNFT", "courseNFT") {

        // mint 5 NFTs to yourself

        for (uint i = 0; i < 5; i++) {

            \_mint(msg.sender, i);

        }

    }

    // Hardcoded token URI will return the same metadata

    // for each NFT

    function tokenURI(uint) public pure override returns (string memory) {

        return "ipfs://bafkreih4zukbxmfnl2g5bi7ji7sc5zth6yzryreu3bh4pa2na7hk3kfe7m";

    }

}

To interact with our smart contracts in the below sections, we also need to export the contract ABI to the Graph and the frontend as a JSON file.

NFTMarket.json

[

  {

    "anonymous": false,

    "inputs": [

      {

        "indexed": false,

        "internalType": "address",

        "name": "nftAddress",

        "type": "address"

      },

      {

        "indexed": false,

        "internalType": "uint256",

        "name": "tokenId",

        "type": "uint256"

      },

      {

        "indexed": false,

        "internalType": "address",

        "name": "seller",

        "type": "address"

      }

    ],

    "name": "ListingCancelled",

    "type": "event"

  },

  {

    "anonymous": false,

    "inputs": [

      {

        "indexed": false,

        "internalType": "address",

        "name": "nftAddress",

        "type": "address"

      },

      {

        "indexed": false,

        "internalType": "uint256",

        "name": "tokenId",

        "type": "uint256"

      },

      {

        "indexed": false,

        "internalType": "uint256",

        "name": "price",

        "type": "uint256"

      },

      {

        "indexed": false,

        "internalType": "address",

        "name": "seller",

        "type": "address"

      }

    ],

    "name": "ListingCreated",

    "type": "event"

  },

  {

    "anonymous": false,

    "inputs": [

      {

        "indexed": false,

        "internalType": "address",

        "name": "nftAddress",

        "type": "address"

      },

      {

        "indexed": false,

        "internalType": "uint256",

        "name": "tokenId",

        "type": "uint256"

      },

      {

        "indexed": false,

        "internalType": "address",

        "name": "seller",

        "type": "address"

      },

      {

        "indexed": false,

        "internalType": "address",

        "name": "buyer",

        "type": "address"

      }

    ],

    "name": "ListingPurchased",

    "type": "event"

  },

  {

    "anonymous": false,

    "inputs": [

      {

        "indexed": false,

        "internalType": "address",

        "name": "nftAddress",

        "type": "address"

      },

      {

        "indexed": false,

        "internalType": "uint256",

        "name": "tokenId",

        "type": "uint256"

      },

      {

        "indexed": false,

        "internalType": "uint256",

        "name": "newPrice",

        "type": "uint256"

      },

      {

        "indexed": false,

        "internalType": "address",

        "name": "seller",

        "type": "address"

      }

    ],

    "name": "ListingUpdated",

    "type": "event"

  },

  {

    "inputs": [

      { "internalType": "address", "name": "nftAddress", "type": "address" },

      { "internalType": "uint256", "name": "tokenId", "type": "uint256" }

    ],

    "name": "cancelListing",

    "outputs": [],

    "stateMutability": "nonpayable",

    "type": "function"

  },

  {

    "inputs": [

      { "internalType": "address", "name": "nftAddress", "type": "address" },

      { "internalType": "uint256", "name": "tokenId", "type": "uint256" },

      { "internalType": "uint256", "name": "price", "type": "uint256" }

    ],

    "name": "createListing",

    "outputs": [],

    "stateMutability": "nonpayable",

    "type": "function"

  },

  {

    "inputs": [

      { "internalType": "address", "name": "", "type": "address" },

      { "internalType": "uint256", "name": "", "type": "uint256" }

    ],

    "name": "listings",

    "outputs": [

      { "internalType": "uint256", "name": "price", "type": "uint256" },

      { "internalType": "address", "name": "seller", "type": "address" }

    ],

    "stateMutability": "view",

    "type": "function"

  },

  {

    "inputs": [

      { "internalType": "address", "name": "nftAddress", "type": "address" },

      { "internalType": "uint256", "name": "tokenId", "type": "uint256" }

    ],

    "name": "purchaseListing",

    "outputs": [],

    "stateMutability": "payable",

    "type": "function"

  },

  {

    "inputs": [

      { "internalType": "address", "name": "nftAddress", "type": "address" },

      { "internalType": "uint256", "name": "tokenId", "type": "uint256" },

      { "internalType": "uint256", "name": "newPrice", "type": "uint256" }

    ],

    "name": "updateListing",

    "outputs": [],

    "stateMutability": "nonpayable",

    "type": "function"

  }

]

We also have to configurate hardhat to enable Sepolia network

require("@nomicfoundation/hardhat-chai-matchers");

require("dotenv").config({ path: ".env" });

/\*\* @type import('hardhat/config').HardhatUserConfig \*/

module.exports = {

  solidity: "0.8.20",

  networks: {

    sepolia: {

      url: process.env.RPC\_URL,

      accounts: [process.env.PRIVATE\_KEY],

      chainId: 11155111,

    },

  },

};

In the next step, we have to deploy our contracts to Sepolia network.

const hre = require("hardhat");

async function main() {

  const CourseNftContract = await hre.ethers.deployContract("CourseNFT");

  await CourseNftContract.waitForDeployment();

  console.log("Sepolia NFT deployed to:", CourseNftContract.target);

  const NFTMarket = await hre.ethers.deployContract(

    "NFTMarket"

  );

   await NFTMarket.waitForDeployment()

  console.log("NFT Market deployed to:", NFTMarket.target);

}

main().catch((error) => {

  console.error(error);

  process.exitCode = 1;

});

* 1. The Graph

The Graph is a decentralized querying protocol, primarily used for querying and indexing blockchain data. Using this protocol, we can create our own subgraph which stores our NFT data like the buyers and sellers. It would be especially helpful if we needed to query a large amount of NFT data. We will store the 4 NFT details as a subgraph, which acts like a normal database, but decentralized. We need to use GraphQL to query the data. The Graph uses TypeScript (AssemblyScript in actual terms), which will be compiled into WebAssembly when a script runs.

In this (relatively simple) project, we only need to configurate 3 files: the schema file (schema.graphql), the subgraph metadata file (subgraph.yaml), and the function handler of the subgraph (src/nft-market.ts).

The schema of the NFT is:

Schema.graphql

type ListingEntity @entity {

  id: ID!

  nftAddress: Bytes!

  tokenId: BigInt!

  price: BigInt!

  seller: Bytes!

  buyer: Bytes

}

The configuration file is subgraph.yaml:

specVersion: 0.0.5

schema:

  file: ./schema.graphql

dataSources:

  - kind: ethereum

    name: NFTMarket

    network: sepolia

    source:

      address: "0x9FE577e44CC3e2DD4B73dc0D7C708511CD11406E"

      abi: NFTMarket

      startBlock: 4844962

    mapping:

      kind: ethereum/events

      apiVersion: 0.0.7

      language: wasm/assemblyscript

      entities:

        - ListingCancelled

        - ListingCreated

        - ListingPurchased

        - ListingUpdated

      abis:

        - name: NFTMarket

          file: ./abis/NFTMarket.json

      eventHandlers:

        - event: ListingCancelled(address,uint256,address)

          handler: handleListingCancelled

        - event: ListingCreated(address,uint256,uint256,address)

          handler: handleListingCreated

        - event: ListingPurchased(address,uint256,address,address)

          handler: handleListingPurchased

        - event: ListingUpdated(address,uint256,uint256,address)

          handler: handleListingUpdated

      file: ./src/nft-market.ts

The function handler of the NFT is nft-market.ts:

import { ListingCancelled, ListingCreated, ListingPurchased, ListingUpdated } from "../generated/NFTMarket/NFTMarket";

import { store } from "@graphprotocol/graph-ts";

import { ListingEntity } from "../generated/schema";

export function handleListingCreated(event: ListingCreated): void {

  const id =

    event.params.nftAddress.toHex() +

    "-" +

    event.params.tokenId.toString() +

    "-" +

    event.params.seller.toHex();

  let listing = new ListingEntity(id);

  listing.seller = event.params.seller;

  listing.nftAddress = event.params.nftAddress;

  listing.tokenId = event.params.tokenId;

  listing.price = event.params.price;

  listing.save();

}

export function handleListingCancelled(event: ListingCancelled): void {

  const id =

    event.params.nftAddress.toHex() +

    "-" +

    event.params.tokenId.toString() +

    "-" +

    event.params.seller.toHex();

  let listing = ListingEntity.load(id);

  if (listing) {

    store.remove("ListingEntity", id);

  }

}

export function handleListingPurchased(event: ListingPurchased): void {

  const id =

    event.params.nftAddress.toHex() +

    "-" +

    event.params.tokenId.toString() +

    "-" +

    event.params.seller.toHex();

  let listing = ListingEntity.load(id);

  if (listing) {

    listing.buyer = event.params.buyer;

    listing.save();

  }

}

export function handleListingUpdated(event: ListingUpdated): void {

  const id =

    event.params.nftAddress.toHex() +

    "-" +

    event.params.tokenId.toString() +

    "-" +

    event.params.seller.toHex();

  let listing = ListingEntity.load(id);

  if (listing) {

    listing.price = event.params.newPrice;

    listing.save();

  }

}

* 1. Frontend

Next.js is a frontend JavaScript framework, which is a meta framework of React with server-side rendering (SSR). We will import multiple libraries to support the queries and transactions. To connect the wallet with the browser, RainbowKit is used to provide the “Connect Wallet” component with its default widget display. Wagmi is needed for fetching public data on the blockchain and any interactions with smart contracts (except querying our own NFT market). We also need to connect to a public IPFS gateway to retrieve the NFT data, primarily the image, so we use the Pinata Cloud public gateway as the link.

\_document.js

import { Html, Head, Main, NextScript } from 'next/document'

export default function Document() {

  return (

    <Html lang="en">

      <Head>

      <link href='https://fonts.googleapis.com/css?family=Poppins' rel='stylesheet' />

      </Head>

      <body>

        <Main />

        <NextScript />

      </body>

    </Html>

  )

}

\_app.js

'use client'

import "@rainbow-me/rainbowkit/styles.css";

import "../styles/globals.css";

import { publicProvider } from "wagmi/providers/public";

import { createConfig, WagmiConfig, configureChains } from "wagmi";

import { createPublicClient, http } from "viem";

import { sepolia } from "@wagmi/core/chains";

import { getDefaultWallets } from "@rainbow-me/rainbowkit";

import { RainbowKitProvider } from "@rainbow-me/rainbowkit";

import React from "react";

const { chains } = configureChains(

  [sepolia],

  [publicProvider()]

);

const { connectors } = getDefaultWallets({

  appName: "Sepolia NFT Market",

  projectId: "80d507b460c69cbaaa0659957c484631",

  chains,

});

const config = createConfig({

  autoConnect: true,

  publicClient: createPublicClient({

    chain: sepolia,

    transport: http()

  }),

  connectors

});

export default function MyApp({ Component, pageProps }) {

  const [ready, setReady] = React.useState(false);

  React.useEffect(() => {

    if(!ready) {

      setReady(true);

    }

  }, [])

  if(!ready){

    return null;

  }

  else{

    return (

      <WagmiConfig config={config} suppressHydrationWarning>

        <RainbowKitProvider chains={chains}>

          <Component {...pageProps} />

        </RainbowKitProvider>

      </WagmiConfig>

    )

  }

}

Index.js

import { useEffect, useState } from "react";

import Navbar from "../components/Navbar";

import Listing from "../components/Listing";

import { createClient, fetchExchange} from "urql";

import { SUBGRAPH\_URL } from "../constants";

import { useAccount } from "wagmi";

import Head from "next/head";

export default function Home() {

    const [listings, setListings] = useState();

    const [loading, setLoading] = useState(false);

    const { isConnected } = useAccount();

    async function fetchListings() {

      setLoading(true);

      const listingsQuery = `

        query ListingsQuery {

          listingEntities {

            id

            nftAddress

            tokenId

            price

            seller

            buyer

          }

        }

      `;

      const urqlClient = createClient({

        url: SUBGRAPH\_URL,

        exchanges: [fetchExchange]

      });

      const response = await urqlClient.query(listingsQuery).toPromise();

      const listingEntities = response.data.listingEntities;

      const activeListings = listingEntities.filter((l) => l.buyer === null);

      setListings(activeListings);

      setLoading(false);

    }

    useEffect(() => {

      if (isConnected) {

        fetchListings();

      }

    }, [isConnected]);

    return (

      <>

        <Head>

          <title>CSCI2730 NFT Market</title>

        </Head>

        <Navbar />

        <div className="home\_container">

        {

          !isConnected

          ?(

            <div className="text-black dark:text-white font-semibold">Please connect your wallet first!</div>

          )

          :loading && isConnected

          ?(

          <div className="text-black dark:text-white font-semibold">Loading...</div>

          )

          :!loading && listings && listings.length === 0?

          (

          <div className="text-black dark:text-white font-semibold">No listings found!</div>

          )

          :!loading && listings &&

            listings.map((listing) => {

              return (

                <Listing

                  nftAddress={listing.nftAddress}

                  tokenId={listing.tokenId}

                  price={listing.price}

                  seller={listing.seller}

                  key={listing.id}

                />

              );

            }

          )

        }

        </div>

      </>

    );

}

Narbar.js

import Link from "next/link";

import { ConnectButton } from "@rainbow-me/rainbowkit";

export default function Navbar() {

  return (

    <div className="navbar" suppressHydrationWarning>

        <div className="left">

            <Link href="/" className="link">Home</Link>

            <Link href="/create" className="ml-8">Sell</Link>

        </div>

        <div className="right">

           <ConnectButton />

        </div>

    </div>

  );

}

Listing.js

'use client'

import { useEffect, useState } from "react";

import { useAccount, erc721ABI } from "wagmi";

import { readContract } from "@wagmi/core";

import Link from "next/link";

export default function Listing(props) {

  const [imageURI, setImageURI] = useState("");

  const [name, setName] = useState("");

  const [loading, setLoading] = useState(true);

  const { address } = useAccount();

  const isOwner = address.toLowerCase() === props.seller.toLowerCase();

  async function fetchNFTDetails() {

    try {

     let tokenURI = await readContract({

        address: props.nftAddress,

        abi: erc721ABI,

        functionName: "tokenURI",

        args: [0],

      });

      tokenURI = tokenURI.replace("ipfs://", "https://gateway.pinata.cloud/ipfs/");

      const metadata = await fetch(tokenURI);

      const metadataJSON = await metadata.json();

      let image = metadataJSON.imageUrl;

      image = image.replace("ipfs://", "https://gateway.pinata.cloud/ipfs/");

      setName(metadataJSON.name);

      setImageURI(image);

      setLoading(false);

    } catch (error) {

      console.error(error);

      setLoading(false);

    }

  }

  useEffect(() => {

    fetchNFTDetails();

  }, []);

  return (

    <div>

      {loading ? (

        <div className="card">

          <div className="text-white text-center my-auto h-full">Loading NFT...</div>

        </div>

      ) : (

        <div className="card">

          <Link href={`/${props.nftAddress}/${props.tokenId}`}>

            <img src={imageURI} alt="NFT Image Unavailable" key={Date.now()} />

            <div className="listing\_container">

              <span>

                <b>

                  {name} - #{props.tokenId}

                </b>

              </span>

              <span>Price: {props.price/1000000000000000000} Sepolia</span>

              <span>

                Seller: {isOwner ? "You" : props.seller.substring(0, 6) + "..."}

              </span>

            </div>

          </Link>

        </div>

      )}

    </div>

  );

}

Create.js

'use client'

import { isAddress, parseEther } from "ethers";

import Link from "next/link";

import { useState } from "react";

import { erc721ABI, useAccount } from "wagmi";

import { readContract, writeContract } from "@wagmi/core";

import MarketplaceABI from "../abis/NFTMarketplace.json";

import Navbar from "../components/Navbar";

import { MARKETPLACE\_ADDRESS } from "../constants";

import Head from "next/head";

import { useRouter } from 'next/router'

export default function Create() {

  const [nftAddress, setNftAddress] = useState("");

  const [tokenId, setTokenId] = useState("");

  const [price, setPrice] = useState("");

  const [loading, setLoading] = useState(false);

  const [showListingLink, setShowListingLink] = useState(false);

  const { address } = useAccount();

  const router = useRouter();

  async function handleCreateListing() {

    setLoading(true);

    try {

      const isValidAddress = isAddress(nftAddress);

      if (!isValidAddress) {

        throw new Error(`Invalid contract address`);

      }

      await requestApproval();

      await createListing();

      router.push(`/${nftAddress}/${tokenId}`);

    } catch (error) {

      console.error(error);

    }

    setLoading(false);

  }

  async function requestApproval() {

  const ownerOf = await readContract({

    address: nftAddress,

    abi: erc721ABI,

    functionName: "ownerOf",

    args: [tokenId]

  });

  const isApprovedForAll = await readContract({

    address: nftAddress,

    abi: erc721ABI,

    functionName: "isApprovedForAll",

    args: [address, MARKETPLACE\_ADDRESS]

  });

    if (ownerOf.toLowerCase() !== address.toLowerCase()) {

      throw new Error(`You do not own this NFT`);

    }

    if (!isApprovedForAll) {

    console.log("Requesting approval over NFTs...");

     await writeContract({

       account: address,

       address: nftAddress,

       abi: erc721ABI,

       functionName: "setApprovalForAll",

       args: [MARKETPLACE\_ADDRESS, true]

     });

    }

  }

  async function createListing() {

   await writeContract({

      account: address,

      address: MARKETPLACE\_ADDRESS,

      abi: MarketplaceABI,

      functionName: "createListing",

      args: [nftAddress, tokenId, parseEther(price)]

    });

  }

  return (

    <>

      <Head>

        <title>CSCI2730 NFT Market - Sell NFT</title>

      </Head>

      <Navbar />

      <div className="container">

        <input

          type="text"

          placeholder="NFT Address"

          value={nftAddress}

          onChange={(e) => setNftAddress(e.target.value)}

        />

        <input

          type="text"

          placeholder="Token ID"

          value={tokenId}

          onChange={(e) => setTokenId(e.target.value)}

        />

        <input

          type="text"

          placeholder="Price (Sepolia)"

          value={price}

          onChange={(e) => {

            if (e.target.value === "") {

              setPrice("0");

            } else {

              setPrice(e.target.value);

            }

          }}

        />

        <button onClick={handleCreateListing} disabled={loading}>

          {loading ? "Loading..." : "Create"}

        </button>

        {showListingLink && (

          <Link href={`/${nftAddress}/${tokenId}`}>

              <button>View Listing</button>

          </Link>

        )}

      </div>

    </>

  );

}

[nftcontract]/[tokenid].js

'use client'

import { formatEther, parseEther } from "ethers";

import { useRouter } from "next/router";

import { useEffect, useState } from "react";

import { createClient, fetchExchange } from "urql";

import { erc721ABI, useAccount } from "wagmi";

import MarketplaceABI from "../../abis/NFTMarketplace.json";

import Navbar from "../../components/Navbar";

import { MARKETPLACE\_ADDRESS, SUBGRAPH\_URL } from "../../constants";

import { readContract, writeContract, waitForTransaction } from "@wagmi/core";

import Head from "next/head";

export default function NFTDetails() {

  const router = useRouter();

  const nftAddress = router.query.nftContract;

  const tokenId = router.query.tokenId;

  const [listing, setListing] = useState();

  const [name, setName] = useState("");

  const [imageURI, setImageURI] = useState("");

  const [isOwner, setIsOwner] = useState(false);

  const [isActive, setIsActive] = useState(false);

  const [newPrice, setNewPrice] = useState("");

  const [loading, setLoading] = useState(true);

  const [updating, setUpdating] = useState(false);

  const [canceling, setCanceling] = useState(false);

  const [buying, setBuying] = useState(false);

  const {address} = useAccount()

  async function fetchListing() {

    const listingQuery = `

      query ListingsQuery {

  listingEntities(where: {nftAddress: "${nftAddress}", tokenId: "${tokenId}"}) {

    id

    nftAddress

    tokenId

    price

    seller

    buyer

  }

}

    `;

   const urqlClient = createClient({

     url: SUBGRAPH\_URL,

     exchanges: [fetchExchange],

   });

    const response = await urqlClient.query(listingQuery).toPromise();

    const listingEntities = response.data.listingEntities;

    if (listingEntities.length === 0) {

      window.alert("Listing does not exist or has been canceled");

      return router.push("/");

    }

    const listing = listingEntities[0];

    setIsActive(listing.buyer === null);

    setIsOwner(address.toLowerCase() === listing.seller.toLowerCase());

    setListing(listing);

  }

  async function fetchNFTDetails() {

    let tokenURI = await readContract({

      address: nftAddress,

      abi: erc721ABI,

      functionName: "tokenURI",

      args: [tokenId],

    });

    tokenURI = tokenURI.replace("ipfs://", "https://gateway.pinata.cloud/ipfs/");

    console.log(tokenURI);

    const metadata = await fetch(tokenURI, {method: 'GET', redirect: 'follow'});

    const metadataJSON = await metadata.json();

    let image = metadataJSON.imageUrl;

    image = image.replace("ipfs://", "https://gateway.pinata.cloud/ipfs/");

    setName(metadataJSON.name);

    setImageURI(image);

  }

  async function updateListing() {

    const { hash } = await writeContract({

      account: address,

      address: MARKETPLACE\_ADDRESS,

      abi: MarketplaceABI,

      functionName: "updateListing",

      args: [nftAddress, tokenId, parseEther(newPrice)],

    });

    setUpdating(true);

    await waitForTransaction({hash});

    await fetchListing();

    setUpdating(false);

  }

  async function cancelListing() {

   const { hash } = await writeContract({

      account: address,

      address: MARKETPLACE\_ADDRESS,

      abi: MarketplaceABI,

      functionName: "cancelListing",

      args: [nftAddress, tokenId]

    });

    setCanceling(true)

    await waitForTransaction({hash});

    window.alert("Listing canceled");

    await router.push("/");

    setCanceling(false);

  }

  async function buyListing() {

   setBuying(true)

     const { hash } = await writeContract({

       account: address,

       address: MARKETPLACE\_ADDRESS,

       abi: MarketplaceABI,

       functionName: "purchaseListing",

       args: [nftAddress, tokenId],

       value: listing.price,

     });

    await waitForTransaction({hash});

    await fetchListing();

    setBuying(false);

  }

   useEffect(() => {

     if (router.query.nftContract && router.query.tokenId && address) {

       Promise.all([fetchListing(), fetchNFTDetails()]).finally(() =>

         setLoading(false)

       );

     }

   }, [router, address]);

  return (

    <>

      <Head>

        <title>CSCI2730 NFT Market - NFT Details</title>

      </Head>

      <Navbar />

      <div>

        {loading ? (

          <span>Loading...</span>

        ) : (

          <div className="detail\_container">

            <div className="detail\_info">

              <img src={imageURI} />

              <span>

                <b>

                  {name} - #{tokenId}

                </b>

              </span>

              <span>Price: {formatEther(listing&&listing.price?listing.price:0)} Sepolia</span>

              <span>

                <a

                  href={`https://sepolia.etherscan.io/address/${listing&&listing.seller?listing.seller:null}`}

                  target="\_blank"

                >

                  Seller:{" "}

                  {isOwner ? "You" : listing && listing.seller ? listing.seller.substring(0, 6) + "...":null}

                </a>

              </span>

              <span>Status: {listing && listing.buyer === null ? "Active" : "Sold"}</span>

            </div>

            <div className="options">

              {!isActive && listing && listing.buyer && (

                <span>

                  Listing has been sold to{" "}

                  <a

                    href={`https://sepolia.etherscan.io/address/${listing&&listing.buyer?listing.buyer:null}`}

                    target="\_blank"

                  >

                    {listing.buyer}

                  </a>

                </span>

              )}

              {isOwner && isActive && (

                <>

                  <div className="update\_listing">

                    <input

                      type="text"

                      placeholder="New Price (Sepolia)"

                      value={newPrice}

                      onChange={(e) => {

                        if (e.target.value === "") {

                          setNewPrice("0");

                        } else {

                          setNewPrice(e.target.value);

                        }

                      }}

                    ></input>

                    <button disabled={updating} onClick={updateListing} className="update\_btn bg-orange-500 text-white">

                      Update sale price

                    </button>

                  </div>

                  <button

                    className="detail\_btn bg-red-600 text-white mt-4"

                    disabled={canceling}

                    onClick={cancelListing}

                  >

                    Remove from market

                  </button>

                </>

              )}

              {!isOwner && isActive && (

                <button

                  className="detail\_btn bg-green-600 mt-4"

                  disabled={buying}

                  onClick={buyListing}

                >

                  Purchase this NFT

                </button>

              )}

            </div>

          </div>

        )}

      </div>

    </>

  );

}

Css files will not be posted here, they are available on GitHub.

That is all there is to the development code.

1. **Features**
   1. Login to the marketplace by wallet.

Before performing any actions on this platform, users are required to connect their wallet to the website. By simply clicking the “Connect Wallet” button, users can connect to the website by various wallet options, including MetaMask, Coinbase, WalletConnect, etc. The underlying mechanisms and hooks are handled by RainbowKit and Wagmi (a JavaScript client to handle web3 and smart contract requests).

* 1. Users can view available NFTs in the market.

After connecting their wallet, users can view all available NFT listings on the index page, including the seller, image, and the price of the NFT.

* 1. Users can sell their NFTs if they own them.

Users can access the “Sell” page and input the address of the NFT they are selling, they also need to input an arbitrary ID for clarity (since I want to reuse some of the NFTs I deployed instead of deploying 10 different ERC721 contracts). Users must also set the price of the NFT, it must be larger than 0. After all these steps, users can click the “Sell” button to put their NFT for sale and add that token to the subgraph.

* 1. Sellers can update the price when the NFT is listed.

Before the NFT is sold, the seller can modify the price of the NFT, they need to enter the updated price of the token before confirming the change.

* 1. Buyers can purchase and own NFTs from the seller.

If the buyer does not own that NFT, they can purchase the token with the listed price. After confirming the transaction with their wallet, the token will be transferred from the seller to the owner, the buyer would pay the seller the listed price.

* 1. Sellers can remove the NFTs they listed on the market.

Before the NFT is purchased, the owner of the NFT can also remove the NFT from the market, the NFT will remain in the owner’s hand, but it is removed from the subgraph hence the display.

* 1. NFT attributes are stored on IPFS