**1. 实验环境**：

Chrome/Microsoft Edge等浏览器；

MetaMask 插件钱包；

Remix IDE： https://remix.ethereum.org/ ；

hardhat

IPFS

前端框架：React.js

后端框架：Node.js

1. **产品设计思路**

本课程项目是一个NFT交易市场，旨在提供一个平台供用户交易和管理NFT资产。用户可以创建账户、购买和出售NFT，并查看自己的NFT持有情况。整个项目分为三个主要组件：智能合约、后端和前端。

NFT创建和上传：用户可以上传数字艺术品、音乐、视频或其他数字资产，并将其转化为NFT。

NFT展示和交易：用户可以在市场上展示其拥有的NFT，并允许其他用户购买或出价竞拍。

交易记录和支付：平台记录NFT交易历史，并提供安全的支付系统，支持加密货币和传统货币支付。

用户账户和身份验证：用户可以创建个人账户、管理其NFT资产，并进行身份验证，以增加交易的可信度。

协同交互与实现：

合约与后端交互：后端通过Web3.js或其他以太坊库与智能合约进行交互，处理NFT的创建、拍卖和交易请求。

后端与前端交互：后端通过API向前端提供NFT数据、用户账户信息和交易记录，前端通过HTTP请求与后端进行通信。

前端与合约交互：前端通过Web3.js或其他以太坊库与智能合约进行交互，包括展示NFT、发起交易等。

1. **产品合约实现**

合约设计与开发：

设计：设计智能合约以实现NFT的创建、拍卖和交易功能。确定NFT的属性和元数据存储结构。

开发：使用Solidity编程语言开发智能合约，并使用Truffle或Hardhat等工具进行测试和部署。

使用Solidity编写NFT合约：

contract Market is IERC721Receiver {

    IERC20 public erc20;

    IERC721 public erc721;

    bytes4 internal constant MAGIC\_ON\_ERC721\_RECEIVED = 0x150b7a02;

    struct Order {

        address seller;

        uint256 tokenId;

        uint256 price;

    }

    mapping(uint256 => Order) public orderOfId; // token id to order

    Order[] public orders;

    mapping(uint256 => uint256) public idToOrderIndex;

    event Deal(address buyer, address seller, uint256 tokenId, uint256 price);

    event NewOrder(address seller, uint256 tokenId, uint256 price);

    event CancelOrder(address seller, uint256 tokenId);

    event ChangePrice(

        address seller,

        uint256 tokenId,

        uint256 previousPrice,

        uint256 price

    );

    constructor(IERC20 \_erc20, IERC721 \_erc721) {

        require(

            address(\_erc20) != address(0),

            "Market: IERC20 contract address must be non-null"

        );

        require(

            address(\_erc721) != address(0),

            "Market: IERC721 contract address must be non-null"

        );

        erc20 = \_erc20;

        erc721 = \_erc721;

    }

    function buy(uint256 \_tokenId) external {

        require(isListed(\_tokenId), "Market: Token ID is not listed");

        address seller = orderOfId[\_tokenId].seller;

        address buyer = msg.sender;

        uint256 price = orderOfId[\_tokenId].price;

        require(

            erc20.transferFrom(buyer, seller, price),

            "Market: ERC20 transfer not successful"

        );

        erc721.safeTransferFrom(address(this), buyer, \_tokenId);

        removeListing(\_tokenId);

        emit Deal(buyer, seller, \_tokenId, price);

    }

    function cancelOrder(uint256 \_tokenId) external {

        require(isListed(\_tokenId), "Market: Token ID is not listed");

        address seller = orderOfId[\_tokenId].seller;

        require(seller == msg.sender, "Market: Sender is not seller");

        erc721.safeTransferFrom(address(this), seller, \_tokenId);

        removeListing(\_tokenId);

        emit CancelOrder(seller, \_tokenId);

    }

    function changePrice(uint256 \_tokenId, uint256 \_price) external {

        require(isListed(\_tokenId), "Market: Token ID is not listed");

        address seller = orderOfId[\_tokenId].seller;

        require(seller == msg.sender, "Market: Sender is not seller");

        uint256 previousPrice = orderOfId[\_tokenId].price;

        orderOfId[\_tokenId].price = \_price;

        Order storage order = orders[idToOrderIndex[\_tokenId]];

        order.price = \_price;

        emit ChangePrice(seller, \_tokenId, previousPrice, \_price);

    }

    function getAllNFTs() public view returns (Order[] memory) {

        return orders;

    }

    function getMyNFTs() public view returns (Order[] memory) {

        Order[] memory myOrders = new Order[](orders.length);

        uint256 myOrdersCount = 0;

        for (uint256 i = 0; i < orders.length; i++) {

            if (orders[i].seller == msg.sender) {

                myOrders[myOrdersCount] = orders[i];

                myOrdersCount++;

            }

        }

        Order[] memory myOrdersTrimmed = new Order[](myOrdersCount);

        for (uint256 i = 0; i < myOrdersCount; i++) {

            myOrdersTrimmed[i] = myOrders[i];

        }

        return myOrdersTrimmed;

    }

    function isListed(uint256 \_tokenId) public view returns (bool) {

        return orderOfId[\_tokenId].seller != address(0);

    }

    function getOrderLength() public view returns (uint256) {

        return orders.length;

    }

    /\*\*

     \* @dev List a good using a ERC721 receiver hook

     \* @param \_operator the caller of this function

     \* @param \_seller the good seller

     \* @param \_tokenId the good id to list

     \* @param \_data contains the pricing data as the first 32 bytes

     \*/

    function onERC721Received(

        address \_operator,

        address \_seller,

        uint256 \_tokenId,

        bytes calldata \_data

    ) public override returns (bytes4) {

        require(\_operator == \_seller, "Market: Seller must be operator");

        uint256 \_price = toUint256(\_data, 0);

        placeOrder(\_seller, \_tokenId, \_price);

        return MAGIC\_ON\_ERC721\_RECEIVED;

    }

    function toUint256(

        bytes memory \_bytes,

        uint256 \_start

    ) public pure returns (uint256) {

        require(\_start + 32 >= \_start, "Market: toUint256\_overflow");

        require(\_bytes.length >= \_start + 32, "Market: toUint256\_outOfBounds");

        uint256 tempUint;

        assembly {

            tempUint := mload(add(add(\_bytes, 0x20), \_start))

        }

        return tempUint;

    }

    function placeOrder(

        address \_seller,

        uint256 \_tokenId,

        uint256 \_price

    ) internal {

        require(\_price > 0, "Market: Price must be greater than zero");

        orderOfId[\_tokenId].seller = \_seller;

        orderOfId[\_tokenId].price = \_price;

        orderOfId[\_tokenId].tokenId = \_tokenId;

        orders.push(orderOfId[\_tokenId]);

        idToOrderIndex[\_tokenId] = orders.length - 1;

        emit NewOrder(\_seller, \_tokenId, \_price);

    }

    function removeListing(uint256 \_tokenId) internal {

        delete orderOfId[\_tokenId];

        uint256 orderToRemoveIndex = idToOrderIndex[\_tokenId];

        uint256 lastOrderIndex = orders.length - 1;

        if (lastOrderIndex != orderToRemoveIndex) {

            Order memory lastOrder = orders[lastOrderIndex];

            orders[orderToRemoveIndex] = lastOrder;

            idToOrderIndex[lastOrder.tokenId] = orderToRemoveIndex;

        }

        orders.pop();

    }

}

NFT市场的智能合约，允许用户进行ERC20代币和ERC721代币之间的原子交换。Market合约接收了IERC721Receiver接口。同时，合约还引入了IERC20和IERC721接口，用于与ERC20和ERC721代币进行交互。

Order结构用于表示一个交易订单，包括卖家地址、代币ID和价格。

orderOfId是一个映射，将代币ID映射到订单。orders是一个订单数组，用于存储所有订单。idToOrderIndex是一个映射，将代币ID映射到订单在数组中的索引。

buy函数用于购买NFT，首先检查代币是否已经被列出，然后转移ERC20代币给卖家，并将NFT转移给买家，最后移除该代币的挂单。

cancelOrder函数用于取消挂单，首先检查代币是否已经被列出，然后将NFT转移给卖家，并移除该代币的挂单。

changePrice函数用于修改挂单的价格。

getAllNFTs和getMyNFTs函数用于获取所有NFT和用户自己的NFT。

onERC721Received函数是一个ERC721代币接收函数，当NFT被转移给合约时触发，用于将NFT挂单。

toUint256函数用于将字节数组转换为uint256类型。

placeOrder函数用于将NFT挂单。removeListing函数用于移除NFT的挂单。

实现了NFT市场的一些核心功能，包括挂单、购买、取消挂单和修改价格等。同时，还实现了ERC721代币的接收功能，以便用户可以将NFT转移到市场并进行挂单。

1. **产品后端实现**

后端设计与开发：

设计：设计后端服务，包括用户身份验证、NFT数据存储、交易记录等功能。

开发：使用Node.js与智能合约进行交互，并提供API供前端调用。

import express from 'express';

import cors from 'cors';

import bodyParser from 'body-parser';

import fileUpload from 'express-fileupload';

import { addFileToIPFS, addJSONToIPFS } from './ipfs-uploader.js';

import { mint } from './nft-minter.js';

import dotenv from 'dotenv';

dotenv.config("./.env");

const app = express()

app.set('view engine','ejs')

app.use(bodyParser.urlencoded({extended:true}))

app.use(fileUpload())

app.use(cors())

app.get('/', (req,res) => {

    res.render('home')

})

app.post('/upload', (req, res) => {

    if (!req.files || Object.keys(req.files).length === 0) {

        return res.status(400).json({ message: 'No files were uploaded.' });

    }

    const file = req.files.file;

    const fileName = file.name

    const filePath = 'files/' + fileName

    const title = req.body.title

    const description = req.body.description

    const address = req.body.address

    console.log(title, description, address)

    file.mv(filePath, async (err) => {

        if (err) {

            console.log('error: failed to download the file.')

            return res.status(500).send(err)

        }

        const fileResult = await addFileToIPFS(filePath)

        console.log('File added to IPFS:', fileResult.cid.toString());

        const metadata = {

            title,

            description,

            image: 'http://127.0.0.1:8080/ipfs/' + fileResult.cid.toString() + '/' + fileName

        }

        const jsonResult = await addJSONToIPFS(metadata)

        console.log('Metadata added to IPFS:', jsonResult.cid.toString());

        const userAddress = address || process.env.ADDRESS;

        await mint(userAddress, 'http://127.0.0.1:8080/ipfs/' + jsonResult.cid.toString())

        res.json({

            message: 'File uploaded successfully.',

            metadata

        });

    });

});

const HOST = process.env.HOST

const PORT = process.env.PORT

app.listen(PORT, HOST, () => {

    console.log(`Server is running on port ${PORT}`)

})

包含了一些常见的后端功能和中间件，以及用于处理HTTP请求的路由和处理程序，负责处理HTTP请求、文件上传、IPFS交互以及NFT代币铸造等功能。

1. **产品前端的实现**

前端设计与开发：

设计：设计用户界面，包括NFT展示页面、用户账户管理页面、交易页面等。

开发：使用React前端框架开发用户界面，使用Web3.js与智能合约进行交互，通过API调用后端服务。

function App() {

  const [walletAddress, setWallet] = useState("");

  useEffect(() => {

    addWalletListener();

  }, []);

  function addWalletListener() {

    if (window.ethereum) {

      window.ethereum.on("accountsChanged", (accounts) => {

        if (accounts.length > 0) {

          setWallet(accounts[0]);

        } else {

          setWallet("");

        }

      });

    }

  }

  const getWalletAddress = async () => {

    if (window.ethereum) {

      try {

        const accounts = await window.ethereum.request({ method: 'eth\_requestAccounts' });

        setWallet(accounts[0]); // Set the first account as the connected account

      } catch (error) {

        console.error('Error connecting to wallet:', error);

      }

    }

  };

  return (

    <div id="container">

      <Router>

        <Navbar onConnectWallet={getWalletAddress} address={walletAddress} />

        <Routes>

          <Route path="/create-nft" exact element={<UploadImage address={walletAddress}/>} />

          <Route path="/success" element={<UploadSuccess />} />

          <Route path="/" element={<NFTGrid />} />

          <Route path="/nft-detail/:tokenId" element={<NFTDetail />} />

        </Routes>

      </Router>

    </div>

  );

};

export default App;

负责渲染前端应用程序的UI，并处理与以太坊钱包交互的逻辑。它包含了钱包地址的监听、连接钱包的请求、以及页面路由的定义等功能。

web3.js与智能合约进行交互

import { ethers } from 'ethers';

import MyNFTABI from '../contracts/MyNFT.json';

async function main() {

  let provider = new ethers.BrowserProvider(window.ethereum)

  const contractAddress = "0x5FbDB2315678afecb367f032d93F642f64180aa3";

  let account = await provider.getSigner();

  const contract = new ethers.Contract(contractAddress, MyNFTABI, account);

  const result = await contract.totalSupply();

  await contract.safeMint('0x70997970C51812dc3A010C7d01b50e0d17dc79C8', 'https://ipfs.io/ipfs/QmZ4tj')

  console.log(result.toString());

}

export default main;

1. **产品开发心得体会**

在开发NFT交易市场时，需要充分考虑用户体验和安全性。用户认证和授权是非常重要的。另外，合约的设计和实现需要考虑到安全性和可扩展性，确保交易的正确性和高效性。前端界面应简洁明了，提供直观的操作和良好的反馈，以提升用户体验。

在实际开发中，还需要考虑其他方面的设计和实现，如错误处理、交易确认、数据存储和缓存等。此外，还可以考虑引入其他功能，如搜索、筛选和排序等，以提升用户的交易体验。

总之，NFT交易市场是一个复杂的项目，需要综合考虑产品设计、合约实现、后端开发和前端实现等方面的要求。通过合理的架构和良好的协同工作，可以打造出一个安全、高效且用户友好的NFT交易平台。