02.1ethers区块链开发（react）

1.NFT

NFT，全称为Non-Fungible Token，指非同质化代币，是用于表示数字资产（包括jpg和视频剪辑形式）的唯一加密货币令牌，可以买卖。

2.Marketplace

我们所说的Marketplace所指是在线市场/交易平台，它的本质是**连接与撮合**，它将希望提供产品或服务的人(卖方/supplier)与希望购买产品或者服务的人(买方/buyer)联系起来，并进行匹配与撮合。

3.开发工具

node@v16.13.1

react@17.0.2

ethers@5.5.4

ipfs-http-client@56.0.1

web3modal@1.9.5

axios@0.25.0

4.部署工作

4.1MetaMask

参考web3.js初级调用智能合约：

https://note.youdao.com/s/uvB75k7

4.2创建react项目

```
1.下载安装脚手架
yarn global add create-react-app
2.使用脚手架创建项目
create-react-app 项目名
3.下载相应需求
yarn add --save ethers@5.5.4
yarn add web3.storage@3.5.4
yarn add web3modal@1.9.5
yarn add axios@0.25.0
```

ethers:

ethers.js库旨在为以太坊区块链及其生态系统提供一个小而完整的 JavaScript API 库 它最初是与 ethers.io 一起使用，现在已经扩展为更通用的库。

优点：ethers.js 对比使用 web3.js 代码量更少，接口也更简洁，推荐优先使用 ethers.js 。

包含功能：

- 将私钥保存在客户端，安全 可信赖
- 可支持导入和导出的 JSON钱包文件 （Geth，Parity和crowdsale）
- 导入和导出 BIP39 助记词 （12个词备份短语）和 HD（分层确定性）钱包（英语，意大利语，日语，韩语，简体中文，繁体中文; 更多即将推出）
- 从任何合同ABI创建JavaScript 元类对象，包括 ABIv2 和 可读的 ABI
- 支持通过 JSON-RPC ， INFURA ， Etherscan 或 MetaMask 连接到以太坊节点。
- ENS名称 是“一等公民”；它们可以在任何可以使用以太坊地址的地方使用
- 库 非常小 （压缩~88kb;未压缩284kb）
- 功能 完整 ，满足所有的以太坊相关需求
- 文档全面： 中文文档 及 documentation
- 大量维护和添加的 测试用例
- 完全支持 TypeScript 准备好，有定义文件和完整的TypeScript源文件
- 宽松的 MIT 协议许可 （包括所有依赖）; 完全开源可以随意使用

参考网站：https://learnblockchain.cn/docs/ethers.js/

web3.storage

当您的项目、网站或应用程序需要文件存储时，Web3.Storage 可以为您服务。借助 Web3.Storage，您可以获得去中心化存储技术的所有优势，以及您在现代开发工作流程中所期望的顺畅体验。使用 Web3.Storage 所需要的只是一个 API 令牌和您的数据。在后台，Web3.Storage 由可证明的Filecoin存储支持，并使您的用户可以通过公共IPFS网络访问数据——但是当涉及到构建您的下一个应用程序、服务或网站时，您需要知道的只是Web3.Storage 使基于去中心化技术的构建变得简单。最重要的是，Web3.Storage 是免费的。

参考网站：https://docs.web3.storage/ & https://www.npmjs.com/package/web3.storage

ipfs:

IPFS是星际文件系统，是一个旨在创建持久且分布式存储和共享文件的网络传输协议。它是一种内容可寻址的对等超媒体分发协议。在IPFS网络中的节点将构成一个分布式文件系统。它是一个开放源代码项目，自2014年开始由Protocol Labs在开源社区的帮助下发展。其最初由Juan Benet设计。

IPFS是一个互联网的底层协议，类似HTTP协议，上线时间是2015年的5月5号。它的目标是为了补充甚至是取代目前统治互联网的超文本传输协议（HTTP）。

IPFS是传输协议，不是区块链项目，没有使用任何区块链技术。但是具备区块链去中心化的精神。所以，IPFS没有Token、没有发币、不能挖矿；Filecoin才是Token，挖的是Filecoin。

IPFS目标是打造一个更加开放、快速、安全的互联网，利用分布式哈希表解决数据的传输和定位问题， 把点对点的单点传输改变成P2P（多点对多点）的传输，其中存储数据的结构是哈希链。
参考网站：https://www.zhihu.com/question/447408500/answer/1782830492

web3modal:

Web3Modal 从前叫做 Web3Connect，可帮助开发人员通过简单的配置，为其应用添加钱包支持。通过 Web3Modal 库，支持 Metamask、Dapper、WalletConnect 及其他 Web3 浏览器等，还可以轻松配置库以支持 Fortmatic 和 Portis等。
参考网站：https://github.com/Web3Modal/web3modal

5.项目代码展示

5.1CreateItem Page：

完成功能：

a. 用户能够上传和保存文件到IPFS；
b. 用户能够创建一个新的独特的 NFT，并同时设置作品的元数据

```
import { useState, useCallback } from "react";
import { ethers } from "ethers";
import { Web3Storage } from "web3.storage/dist/bundle.esm.min.js";
import Web3Modal from "web3modal";
import "../assets/styles/CreateItem.css";
import NFT from "../contracts/NFT_ABI.json"; // 引入ABI
const nftaddress = "0x90fe47327d2e2851fD4eFEd32bc64c4b14CB1D29"; // 定义合约地址

export default function CreateItem() {
  // 定义数据
  const [fileUrl, setFileUrl] = useState(null);
  const [formInput, updateFormInput] = useState({
    name: "",
    description: "",
  });

  // 让ref为uploaderd的input框变成上传文件夹
  const uploader = useCallback((node) => {
    if (node) {
      node.setAttribute("webkitdirectory", "");
      node.setAttribute("directory", "");
      node.setAttribute("multiple", "");
```

```
    }
  }, []);

  // 通过表单的change事件达成数据双向绑定
  async function onChange(e) {
      // 获取input上传的文件
      let file = e.target.files;
      const data = JSON.stringify({
        name: formInputFree.name,
        description: formInputFree.description,
      });
      //将文字转为上传的格式
      var blob = new Blob([data]);
      const newFile = new File([blob], "test.json");
      console.log(newFile);
      //将文字追加进数组，传入ipfs上传函数
      file = [...file, newFile];
      console.log(file);
      try {
        // 赋值
        setFileUrlFree(file);
      } catch (error) {
        console.log("Error uploading file: ", error);
      }
  }
  // 上传到IPFS并完成铸造
  async function createMarket() {
    // 1.发送到IPFS上
    // 定义上传的apiToken
    let apiToken =

"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJkaWQ6ZXRocjoweDE5N2VEYTQ5QjQyRmVjRjI2QzBhNWM4OThmNUYzNzVGNDU1Y2U2MWEiLCJpc3MiOiJ3ZWIzLXN0b3JhZ2UiLCJpYXQiOjE2NDU1ODEwMDQ4NjAsIm5hbWUiOiJkZXYyMjAyIn0.s9DZmDbB1VasuMmI50RzfFavxwIachm0XuELGz5RZY4";
    // 使用apiToken构造Web3Storage
    const client = new Web3Storage({ token: apiToken });
    // 将文件打包CAR中并发送到web3.storage
    const rootCid = await client.put(fileUrlFree, {
      name: "test1",
      maxRetries: 3,
      wrapWithDirectory: true,
```

```
    });
    try {
      alert("Upload to IPFs succeeded !!!");
      // 2.拼接从ipfs上拿到的url
      let url = `https://ipfs.io/ipfs/${rootCid}`;
      console.log(url);
      // 3.将url传入,并调用智能合约开始铸造NFT
      createSaleFree(url);
    } catch (error) {
      console.log("Error uploading file: ", error);
    }
  }
  // 铸造NFT
  async function createSale(url) {
    const web3Modal = new Web3Modal();
    const connection = await web3Modal.connect();
    const provider = new ethers.providers.Web3Provider(connection);
    const signer = provider.getSigner();
    // 4.接下来，创建项目
    let contract = new ethers.Contract(nftaddress, NFT.abi, signer);
    let transaction = await contract.mintOneToken(url);
    if (transaction) {
      await transaction.wait();
      alert("Project created successfully !!!");
      await window.location.replace("/creatorDashboard");
    }
  }
  // go back
  async function goBack() {
    await window.history.back(-1);
  }
  return (
    <div className="createItem">
      <div className="createItem-Head">
        <h2>Create Item</h2>
        <button onClick={goBack}>Go Back!</button>
      </div>
      <div className="createItem-from-box">
          <h3>Free casting</h3>
          <input
```

```
            style={{ textIndent: "0.6rem" }}
            placeholder="Asset Name"
            onChange={(e) =>
              updateFormInputFree({ ...formInputFree, name: e.target.value })
            }
          />
          <textarea
            style={{ textIndent: "0.6rem" }}
            placeholder="Asset Description"
            onChange={(e) =>
              updateFormInputFree({
                ...formInputFree,
                description: e.target.value,
              })
            }
          />
          {/* 上传文件夹 */}
          <input
            type="file"
            ref={uploader}
            onChange={(e) => onChange(e, "free")}
          />
          <button onClick={createFreeItem} className="createItem-from-btn">
            Create Free Digital Asset
          </button>
        </div>
    </div>
  );
}
```

5.2SellDigitalAsset Page：

完成功能：

a. 作为应用程序的主要入口；
b. 作为查询呈现待售数字资产的页面(MarketPlace上)；
c. 可以购买想要的作品

```
import { ethers } from "ethers";
import { useEffect, useState } from "react";
import axios from "axios";
import Web3Modal from "web3modal";
```

```
import "../assets/styles/SellDigitalAsset.css";
import Model from "./Model";// 引入3D模型
// 引入ABI
import NFT from "../contracts/NFT_ABI.json";
import Market from "../contracts/Mkt_ABI.json";
// 定义合约地址
const nftaddress = "0x90fe47327d2e2851fD4eFEd32bc64c4b14CB1D29";
const nftmarketaddress = "0xFC2Ea5A1F3Bed1B545A6be182BF52C20B5e45921";
// 定义需要链接的网络(此时为内网测试)
let rpcEndpoint = "http://192.168.11.120:8545";
// 判断网络并作出相应操作
if (process.env.NEXT_PUBLIC_WORKSPACE_URL) {
  rpcEndpoint = process.env.NEXT_PUBLIC_WORKSPACE_URL;
}
export default function Home() {
  // 定义数据
  const [nfts, setNfts] = useState([]);
  const [loadingState, setLoadingState] = useState("not-loaded");
  useEffect(() => {
    loadNFTs();
  }, []);
  async function loadNFTs() {
    // 连接网络并定义合约
    const provider = new ethers.providers.JsonRpcProvider(rpcEndpoint);
    const tokenContract = new ethers.Contract(nftaddress, NFT.abi, provider);
    const marketContract = new ethers.Contract(
      nftmarketaddress,
      Market.abi,
      provider
    );
    // 查找marketContract合约上的所有项目
    const data = await marketContract.fetchMarketItems();
    const items = await Promise.all(
      // 循环遍历所有拿到的数据
      data.map(async (i) => {
        const tokenUri = await tokenContract.tokenURI(i.tokenId);
        // 处理数据
        let mtl = `${tokenUri}/1.mtl`;
        let obj = `${tokenUri}/1.obj`;
        let txt = `${tokenUri}/Introduction.txt`;
```

```
        const meta = await axios.get(`${tokenUri}/test.json`);
        let price = ethers.utils.formatUnits(i.price.toString(), "ether");
        let item = {
          price,
          itemId: i.itemId.toNumber(),
          seller: i.seller,
          owner: i.owner,
          addr: tokenUri,
          mtl,
          obj,
          txt,
          name: meta.data.name,
          description: meta.data.description,
        };
        return item;
      })
    );
    setNfts(items);
    setLoadingState("loaded");
  }
  // buyBtn
  async function buyNft(nft) {
    const web3Modal = new Web3Modal();
    const connection = await web3Modal.connect();
    const provider = new ethers.providers.Web3Provider(connection);
    const signer = provider.getSigner();
    const contract = new ethers.Contract(nftmarketaddress, Market.abi, signer);
    const price = ethers.utils.parseUnits(nft.price.toString(), "ether");
    console.log(nft.itemId,nftaddress);
    const transaction = await contract.createMarketSale(
      nftaddress,
      nft.itemId,
      {
        value: price,
      }
    );
    await transaction.wait();
    // 重绘
    loadNFTs();
  }
```

```
  // go back
  async function goBack() {
    await window.history.back(-1);
  }
  if (loadingState === "loaded" && !nfts.length)
    return <h1>No items in marketplace</h1>;
  return (
    <div className="SellDigitalAsset">
      <div className="SellDigitalAsset-Box" style={{ maxWidth: "1600px" }}>
        <div className="SellDigitalAsset-Box-Head">
          <h2>Sell Digital Asset</h2>
          <button onClick={goBack}>Go Back!</button>
        </div>
        <div className="SellDigitalAsset-Box-content">
          {nfts.map((nft, i) => (
            <div key={i} className="SellDigitalAsset-Box-content-item">
               <div
                 className="model"
                 style={{ width: "16rem",height: "14rem",borderRadius: "0.5rem 0.5rem 0
0",}}>
                 <Model className="modelchild" data={nft} />
               </div>
               <div className="SellDigitalAsset-Box-content-item-info">
                 <p className="SellDigitalAsset-Box-content-item-info-name">
                   {nft.name}
                 </p>
                 <div style={{ height: "4rem", overflow: "hidden" }}>
                   <p style={{ paddingLeft: "1rem", fontSize: "14px" }} >
                     {nft.description}
                   </p>
                 </div>
               </div>
               <div className="SellDigitalAsset-Box-content-item-Btn">
                 <p style={{
                     textIndent: "1rem",
                     fontWeight: "bold",
                     fontSize: "18px", }} >
                    {nft.price} ETH
                 </p>
                 <button onClick={() => buyNft(nft)}>Buy</button>
```

```
            </div>
          </div>
        ))}
      </div>
    </div>
  </div>
  );
}


```

5.3MyDigitalAssets Page

完成功能:

a. 返回用户拥有的作品并展示a

```
import { ethers } from "ethers";
import { useEffect, useState } from "react";
import axios from "axios";
import Web3Modal from "web3modal";
import "../assets/styles/MyDigitalAssets.css";
import Model from "./Model.js";// 引入3D模型
// 引入ABI
import NFT from "../contracts/NFT_ABI.json";
import Market from "../contracts/Mkt_ABI.json";
// 定义合约地址
const nftaddress = "0x90fe47327d2e2851fD4eFEd32bc64c4b14CB1D29";
const nftmarketaddress = "0xFC2Ea5A1F3Bed1B545A6be182BF52C20B5e45921";
export default function MyAssets() {
  // 定义数据
  const [nfts, setNfts] = useState([]);
  const [loadingState, setLoadingState] = useState("not-loaded");
  useEffect(() => {
    loadNFTs();
  }, []);
  async function loadNFTs() {
    const web3Modal = new Web3Modal({
      network: "mainnet",
      cacheProvider: true,
    });
```

```
    const connection = await web3Modal.connect();
    const provider = new ethers.providers.Web3Provider(connection);
    const signer = provider.getSigner();
    const marketContract = new ethers.Contract(
      nftmarketaddress,
      Market.abi,
      signer
    );
    // 定义合约
    const tokenContract = new ethers.Contract(nftaddress, NFT.abi, provider);
    // 查找marketContract合约上的自己创建的项目
    const data = await marketContract.fetchMyNFTs();
    const items = await Promise.all(
      data.map(async (i) => {
        // 调用NFT合约获取tokenURI
        const tokenUri = await tokenContract.tokenURI(i.tokenId);
        // 处理数据
        let mtl = `${tokenUri}/1.mtl`;
        let obj = `${tokenUri}/1.obj`;
        const meta = await axios.get(`${tokenUri}/test.json`);
        let price = ethers.utils.formatUnits(i.price.toString(), "ether");
        let item = {
          price,
          tokenId: i.tokenId.toNumber(),
          seller: i.seller,
          owner: i.owner,
          addr: tokenUri,
          mtl,
          obj,
          name: meta.data.name,
          description: meta.data.description,
        };
        return item;
      })
    );
    setNfts(items);
    setLoadingState("loaded");
  }
  // 通过判断有无数据，选择显示
  if (loadingState === "loaded" && !nfts.length)
```

```
    return <h1 className="py-10 px-20 text-3xl">No assets owned</h1>;

  // go back
  async function goBack() {
    await window.history.back(-1);
  }

  return (
    <div className="MyDigitalAssets">
      <div className="MyDigitalAssets-" style={{ padding: "0.5rem" }}>
        <div className="MyDigitalAssets-Head">
          <h2>My Digital Assets</h2>
          <button onClick={goBack}>Go Back!</button>
        </div>
        <div className="MyDigitalAssets-Box">
          {nfts.map((nft, i) => (
            <div key={i} className="MyDigitalAssets-Box-item">
              <div
                className="model"
                style={{
                  width: "16rem",
                  height: "14rem",
                  borderRadius: "0.5rem 0.5rem 0 0",
                }}
              >
                <Model className="modelchild" data={nft} />
              </div>
              <div className="MyDigitalAssets-Box-item-info">
                <h3>{nft.name}</h3>
                <p>{nft.description}</p>
              </div>
              <div className="MyDigitalAssets-Box-item-price">
                <p className="text-2xl font-bold text-white">
                  Price - {nft.price} Eth
                </p>
              </div>
            </div>
          ))}
        </div>
      </div>
```

```
    </div>
  );
}
```

5.4CreatorDashboard Page

完成功能：

a. 展示该用户所创建的所有作品；
b. 可以把创建的作品设置价格并上架到Maketplace上面；
c. 展示该用户所出售的所有作品；
d. 展示该用户所有的作品(创建+出售)；
e. 展示该用户所上架的作品，并可以从Maketplace上面下架到自己仓库；

```
import { ethers } from "ethers";
import { useEffect, useState } from "react";
import axios from "axios";
import Web3Modal from "web3modal";
import Model from "./Model.js"; // 引入3D模型
import "../assets/styles/CreatorDashboard.css";

// 引入ABI
import NFT from "../contracts/NFT_ABI.json";
import Market from "../contracts/Mkt_ABI.json";
// 定义合约地址
const nftaddress = "0x90fe47327d2e2851fD4eFEd32bc64c4b14CB1D29";
const nftmarketaddress = "0xFC2Ea5A1F3Bed1B545A6be182BF52C20B5e45921";

export default function CreatorDashboard() {
  // 定义数据
  const [nfts, setNfts] = useState([]);
  const [sold, setSold] = useState([]);
  const [solds, setSolds] = useState([]);
  const [notSold, setNotSold] = useState([]);
  const [tokenUrii, setTokenUri] = useState(""); // 我创建的NFT(用作判断的时候)
  const [price, setPrice] = useState({
    price: "",
  });
  const [loadingState, setLoadingState] = useState("not-loaded");
  useEffect(() => {
    loadNFTs();
```

```
    loadMarkets();
  }, []);
  async function loadNFTs() {
    const web3Modal = new Web3Modal({
      network: "mainnet",
      cacheProvider: true,
    });
    const connection = await web3Modal.connect();
    const provider = new ethers.providers.Web3Provider(connection);
    const signer = provider.getSigner();
    const nftContract = new ethers.Contract(nftaddress, NFT.abi, signer);
    const tokenContract = new ethers.Contract(nftaddress, NFT.abi, provider);
    // 调用NFT合约查询自己创建的NFT
    const data = await nftContract.fetchMyNFTs();
    const items = await Promise.all(
      data.map(async (i) => {
        if (i[4] != "") {
          const tokenUri = await tokenContract.tokenURI(i.tokenId);
          // 赋值，为了给空盒子做判断
          setTokenUri(tokenUri);
          let mtl = `${tokenUri}/1.mtl`;
          let obj = `${tokenUri}/1.obj`;
          const meta = await axios.get(`${tokenUri}/test.json`);
          let item = {
            itemId: i.tokenId.toNumber(),
            tokenId: i.tokenId.toNumber(),
            seller: i.seller,
            owner: i.owner,
            sold: i.sold,
            addr: tokenUri,
            mtl,
            obj,
          };
          return item;
        } else {
          let item = {
            itemId: i.tokenId.toNumber(),
            tokenId: i.tokenId.toNumber(),
            seller: i.seller,
            owner: i.owner,
```

```
            sold: i.sold,
          };
          return item;
        }
      })
    );
    /* create a filtered array of created items */
    setNfts(items);
    setLoadingState("loaded");
  }
  // 查询自己所出售和未出售的
  async function loadMarkets() {
    const web3Modal = new Web3Modal({
      network: "mainnet",
      cacheProvider: true,
    });
    const connection = await web3Modal.connect();
    const provider = new ethers.providers.Web3Provider(connection);
    const signer = provider.getSigner();
    const marketContract = new ethers.Contract(
      nftmarketaddress,
      Market.abi,
      signer
    );
    const tokenContract = new ethers.Contract(nftaddress, NFT.abi, provider);
    const tokendata = await marketContract.fetchItemsCreated();
    const tokenitems = await Promise.all(
      tokendata.map(async (i) => {
        if (i == "") {
          alert("not-loaded");
        } else {
          const tokenUri = await tokenContract.tokenURI(i.tokenId);
          let mtl = `${tokenUri}/1.mtl`;
          let obj = `${tokenUri}/1.obj`;
          const meta = await axios.get(`${tokenUri}/test.json`);
          let price = ethers.utils.formatUnits(i.price.toString(), "ether");
          let item = {
            price,
            itemId: i.itemId.toNumber(),
            tokenId: i.tokenId.toNumber(),
```

```
            seller: i.seller,
            owner: i.owner,
            sold: i.sold,
            addr: tokenUri,
            mtl,
            obj,
          };
          return item;
        }
      })
    );
    /* create a filtered array of sold items and all items */
   // 筛选已出售的
    const soldItems = tokenitems.filter((i) => i.sold);
    console.log(soldItems);
    // 筛选未出售的(可下架)
    const notSoldItems = tokenitems.filter((i) => !i.sold);
    // sold(已出售的)
    setSold(soldItems);
    // notSold(未出售的)
    setNotSold(notSoldItems);
    // all(创建的+已出售的(下架等同于出售))
    setSolds(tokenitems);
    setLoadingState("loaded");
  }
   // 通过判断有无数据，选择显示
  if (
    loadingState === "loaded" &&
    !nfts.length &&
    loadingState === "loaded" &&
    !sold.length &&
    loadingState === "loaded" &&
    !solds.length
  )
    return <h1 className="py-10 px-20 text-3xl">No assets created</h1>;

  // go back
  async function goBack() {
    await window.history.back(-1);
  }
```

```
148
149   // put on the marketplace
150   var _price = price;
151   async function putOn(nft) {
152     const web3Modal = new Web3Modal();
153     const connection = await web3Modal.connect();
154     const provider = new ethers.providers.Web3Provider(connection);
155     const signer = provider.getSigner();
156     const price = ethers.utils.parseUnits(_price, "ether");
157     let contract = new ethers.Contract(nftmarketaddress, Market.abi, signer);
158     let listingPrice = await contract.getListingPrice();
159     listingPrice = listingPrice.toString();
160     let transaction = await contract.createMarketItem(
161       nftaddress,
162       nft.tokenId,
163       price,
164       {
165         value: listingPrice,
166       }
167     );
168     await transaction.wait();
169     await window.location.replace("/sellDigitalAsset");// 跳转到sellDigitalAsset页面
170   }
171   // 获取价格的change事件
172   async function change(e) {
173     console.log(e.target.value);
174     setPrice(e.target.value);
175   }
176   // 下架
177   async function offShelf(nft) {
178     console.log(nft);
179     const web3Modal = new Web3Modal({
180       network: "mainnet",
181       cacheProvider: true,
182     });
183     const connection = await web3Modal.connect();
184     const provider = new ethers.providers.Web3Provider(connection);
185     const signer = provider.getSigner();
186     const marketContract = new ethers.Contract(
187       nftmarketaddress,
```

```
      Market.abi,
      signer
    );
    const tokendatas = await marketContract.recallSellingItem(
      nftaddress,
      nft.itemId
    );
  }

  // 根据不同条件渲染不同组件
  function renderDiv() {
    const isTokenUrii = tokenUrii;
    if (isTokenUrii) {
      return (
        <div style={{ display: "flex", flexWrap: "wrap" }}>
          {nfts.map((nft, i) => (
            <div key={i} className="CreatorDashboard-items-Created">
              {/* <img
                style={{
                  width: "16rem",
                  height: "17rem",
                  borderRadius: "0.5rem 0.5rem 0 0",
                }}
                src={nft.image}
              /> */}
              <div
                className="model"
                style={{
                  width: "16rem",
                  height: "17rem",
                  borderRadius: "0.5rem 0.5rem 0 0",
                }}
              >
                <Model className="modelchild" data={nft} />
              </div>
              <div
                style={{
                  backgroundColor: "#000",
                  color: "#fff",
                  borderRadius: "0 0 0.5rem 0.5rem",
```

```
                flex: "1",
              }}
            >
              <p
                style={{ textAlign: "center", margin: "0.8rem 0 0.8rem 0" }}
                className="text-2xl font-bold text-white"
              >
                Price -
                <input style={{ width: "3rem" }} onChange={change}></input>
                 Eth
              </p>
              <p style={{ display: "flex", justifyContent: "center" }}>
                <button
                  style={{
                    cursor: "pointer",
                    border: "0",
                    backgroundColor: "#80c342",
                    color: "#fff",
                    height: "1.5rem",
                    borderRadius: "1.5rem",
                    width: "8rem",
                  }}
                  onClick={putOn.bind(this, nft)}
                >
                  Put on the shelf
                </button>
              </p>
            </div>
          </div>
        ))}
      </div>
    );
  } else {
    return <div></div>;
  }
}

return (
  <div>
    {/* 展示自己所创建的项目 */}
```

```
      <div className="p-4">
        <div className="CreatorDashboard-Head">
          <h2 className="text-2xl py-2">Items Created</h2>
          <button onClick={goBack}>Go Back!</button>
        </div>
        <div>{renderDiv()}</div>
      </div>
      {/* 展示已出售的项目 */}
      <div className="px-4">
        {Boolean(sold.length) && (
          <div className="CreatorDashboard">
            <h2 style={{ margin: "1rem" }}>Items sold</h2>
            <div style={{ display: "flex", flexWrap: "wrap" }}>
              {sold.map((nft, i) => (
                <div key={i} className="CreatorDashboard-items-sold">
                  <div
                    className="model"
                    style={{
                      width: "16rem",
                      height: "17rem",
                      borderRadius: "0.5rem 0.5rem 0 0",
                    }}
                  >
                    <Model className="modelchild" data={nft} />
                  </div>
                  <div style={{ flex: "1" }}>
                    <p style={{ textAlign: "center", paddingTop: "1rem" }}>
                      Price - {nft.price} Eth
                    </p>
                  </div>
                </div>
              ))}
            </div>
          </div>
        )}
      </div>
      {/* 展示未出售的项目 */}
      <div className="px-4">
        {Boolean(notSold.length) && (
          <div className="CreatorDashboard">
```

```
            <h2 style={{ margin: "1rem" }}>Items notSold</h2>
            <div style={{ display: "flex", flexWrap: "wrap" }}>
              {notSold.map((nft, i) => (
                <div key={i} className="CreatorDashboard-items-notSold">
                  <div
                    className="model"
                    style={{
                      width: "16rem",
                      height: "17rem",
                      borderRadius: "0.5rem 0.5rem 0 0",
                    }}
                  >
                    <Model className="modelchild" data={nft} />
                  </div>
                  <div style={{ flex: "1", textAlign: "center" }}>
                    <p style={{ textAlign: "center", paddingTop: "1rem" }}>
                      Price - {nft.price} Eth
                    </p>
                    <button
                      style={{
                        backgroundColor: "#f60",
                        color: "#fff",
                        width: "5rem",
                        height: "1.5rem",
                        borderRadius: "1rem",
                        marginTop: "0.2rem",
                      }}
                      onClick={offShelf.bind(this, nft)}
                    >
                      下架
                    </button>
                  </div>
                </div>
              ))}
            </div>
          </div>
        )}
      </div>
      {/* 展示所有的项目(创建的+已出售的(下架等同于出售)) */}
      <div className="px-4" style={{ marginBottom: "2rem" }}>
```

```
        {Boolean(solds.length) && (
          <div className="CreatorDashboard">
            <h2 style={{ margin: "1rem" }}>All Items</h2>
            <div style={{ display: "flex", flexWrap: "wrap" }}>
              {solds.map((nft, i) => (
                <div key={i} className="CreatorDashboard-items-all">
                  <div
                    className="model"
                    style={{
                      width: "16rem",
                      height: "17rem",
                      borderRadius: "0.5rem 0.5rem 0 0",
                    }}
                  >
                    <Model className="modelchild" data={nft} />
                  </div>
                  <div style={{ flex: "1" }}>
                    <p style={{ textAlign: "center", paddingTop: "1rem" }}>
                      Price - {nft.price} Eth
                    </p>
                  </div>
                </div>
              ))}
            </div>
          </div>
        )}
      </div>
    </div>
  );
}
```

5.5Pledge Page

完成功能：

a. 将指定NFT质押到HoldAssets合约中

```
import React from "react";
import { ethers } from "ethers";
import { useEffect, useState } from "react";
import axios from "axios";
import Web3Modal from "web3modal";
```

```
import Model from "./Model.js";
import "../assets/styles/Pledge.css";
// 引入ABI
import NFT from "../contracts/NFT_ABI.json";
import HoldAssets from "../contracts/HoldAssets_ABI.json";
// 定义合约地址
const nftaddress = "0x90fe47327d2e2851fD4eFEd32bc64c4b14CB1D29";
const holdAssetsaddress = "0xbD818FA996C19F64841390De77Ee2b43E0d78F0d";
export default function Pledge() {
  const [nfts, setNfts] = useState([]); // 查询到自己所创建的NFT
  const [pNfts, setPnfts] = useState([]); // 查询到自己所质押的NFT
  const [loadingState, setLoadingState] = useState("not-loaded");
  useEffect(() => {
    loadNFTs();
    pledgedNFTs();
  }, []);

  // 查询到自己所创建的NFT
  async function loadNFTs() {
    const web3Modal = new Web3Modal({
      network: "mainnet",
      cacheProvider: true,
    });
    const connection = await web3Modal.connect();
    const provider = new ethers.providers.Web3Provider(connection);
    const signer = provider.getSigner();


    const nftContract = new ethers.Contract(nftaddress, NFT.abi, signer);
    const tokenContract = new ethers.Contract(nftaddress, NFT.abi, provider);
    const data = await nftContract.fetchMyNFTs();
    console.log(data);
    const items = await Promise.all(
      data.map(async (i) => {
        const tokenUri = await tokenContract.tokenURI(i.tokenId);
        let mtl = `${tokenUri}/1.mtl`;
        let obj = `${tokenUri}/1.obj`;
        const meta = await axios.get(`${tokenUri}/test.json`);
        let item = {
          tokenId: i.tokenId.toNumber(),
```

```
          seller: i.seller,
          owner: i.owner,
          addr: tokenUri,
          mtl,
          obj,
          name: meta.data.name,
          description: meta.data.description,
        };
        console.log(item);
        return item;
      })
    );
    setNfts(items);
    setLoadingState("loaded");
  }

  if (loadingState === "loaded" && !nfts.length)
    return <h1 className="py-10 px-20 text-3xl">No assets owned</h1>;

  // go back
  async function goBack() {
    await window.history.back(-1);
  }

  // 质押
  async function pledge(nft) {
    console.log(nft);
    const web3Modal = new Web3Modal({
      network: "mainnet",
      cacheProvider: true,
    });
    const connection = await web3Modal.connect();
    const provider = new ethers.providers.Web3Provider(connection);
    const signer = provider.getSigner();
    const tokenContract = new ethers.Contract(nftaddress, NFT.abi, provider);
    const nftContract = new ethers.Contract(nftaddress, NFT.abi, signer);
    // 调起合约执行质押对应的NFT
    const data = await nftContract.pledgeToNFT(holdAssetsaddress,nft.tokenId);
    if (data) {
```

```
      // 质押的event事件
      await contract.tokenContract.on("Pledge", (author, event) => {
        console.log(author); // 质押的BigNumber
        console.log(event); // 当前事件的交易信息
        if (author) {
          // window.history.go(0); // 刷新页面
          // 将返回的BigNumber转化为数字
          let tokenid = author.toNumber();
          console.log(tokenid);
          alert("Pledge NFT succeeded!!!");
          window.location.replace("/releasePledge");
        }
      });
    } else {
      alert("Pledge NFT failed!!!");
    }
  }

  return (
    <div className="Pledge">
      <div className="Pledge-" style={{ padding: "0.5rem" }}>
        <div className="Pledge-Head">
          <h2>My Digital Assets</h2>
          <button onClick={goBack}>Go Back!</button>
        </div>
        <div className="Pledge-Box">
          {nfts.map((nft, i) => (
            <div key={i} className="Pledge-Box-item">
              <div
                className="model"
                style={{
                  width: "16rem",
                  height: "14rem",
                  borderRadius: "0.5rem 0.5rem 0 0",
                }}
              >
                <Model className="modelchild" data={nft} />
              </div>
              <div className="Pledge-Box-item-info">
                <h3>{nft.name}</h3>
```

```
                <p>{nft.description}</p>
              </div>
              <div className="Pledge-Box-item-price">
                <p className="text-2xl font-bold text-white">
                  Price - {nft.price} Eth
                </p>
                <p>
                  <button onClick={pledge.bind(this, nft)}>质押</button>
                </p>
              </div>
            </div>
          ))}
        </div>
      </div>
    </div>
  );
}


```

5.6ReleasePledge Page

完成功能：

a. 将指定NFT解除质押到原合约中

```
import React from "react";
import { useEffect, useState } from "react";
import { ethers } from "ethers";
import Web3Modal from "web3modal";
import "../assets/styles/ReleasePledge.css";
// 引入合约ABI
import HoldAssets from "../contracts/HoldAssets_ABI.json";
import NFT from "../contracts/NFT_ABI.json";
// 定义合约地址
const holdAssetsaddress = "0xbD818FA996C19F64841390De77Ee2b43E0d78F0d";
const nftaddress = "0x19FBDCcb3a1BC662638D8Aa9C5a139B3FB17eAE1";

export default function Releasepledge() {
  const [pNfts, setPnfts] = useState([]); // 查询到自己所质押的NFT
  const [loadingState, setLoadingState] = useState("not-loaded");
  useEffect(() => {
```

```
    pledgedNFTs();
  }, []);

  // 查询到自己所质押的NFT
  async function pledgedNFTs() {
    const web3Modal = new Web3Modal({
    network: "mainnet",
    cacheProvider: true,
    });
    const connection = await web3Modal.connect();
    const provider = new ethers.providers.Web3Provider(connection);
    const signer = provider.getSigner();
    const holdAssets = new ethers.Contract(holdAssetsaddress, HoldAssets.abi, signer),
    // 调起合约查询自己所质押的NFT
    const data = await holdAssets.fetchMyPledgeNFTs();
    console.log(data);
    setPnfts(data);
    setLoadingState("loaded");
  }

  if (loadingState === "loaded" && !pNfts.length)
    return <h1 className="py-10 px-20 text-3xl">No assets owned</h1>;

  // 解除质押
  async function turnbackpledge(nft) {
    const web3Modal = new Web3Modal({
    network: "mainnet",
    cacheProvider: true,
    });
    const connection = await web3Modal.connect();
    const provider = new ethers.providers.Web3Provider(connection);
    const signer = provider.getSigner();
    const holdAssets = new ethers.Contract(holdAssetsaddress, HoldAssets.abi, signer),
    const holdAssetsProvider = new ethers.Contract(holdAssetsaddress, HoldAssets.abi,
provider),
    // 调起合约执行解除质押对应的NFT
    const data = await holdAssets.turnbackNFT(
      nftaddress,
      nft
    );
```

```
    console.log(data);
    // await data.wait();
    if (data) {
      // 解除质押的event事件
      await holdAssetsProvider.on(
        "TurnbackNFT",
        (author, acc, event) => {
          console.log(author, author.toNumber());
          console.log("Event-tokenID:", author.toString()); // 解除质押的BigNumber
          console.log("Event-addr:", acc); // 当前账户地址
          console.log(event); // 当前事件的交易信息
          if (author) {
            alert("Successful release of pledge NFT!!!");
            window.location.replace("/pledge");
          }
        }
      );
    } else {
      alert("Pledge release NFT failed!!!");
    }
  }

  // go back
  async function goBack() {
    await window.history.back(-1);
  }

  return (
    <div className="ReleasePledge">
      <div className="ReleasePledge-Head">
        <h2>My NFT</h2>
        <button onClick={goBack}>Go Back!</button>
      </div>
      <div className="ReleasePledge-Box">
        {pNfts.map((nft, i) => (
          <div key={i} className="ReleasePledge-Box-items">
            <div className="ReleasePledge-Box-item-info">
              <h3>TokenId: {nft.toNumber()}</h3>
              <p>{nft.description}</p>
            </div>
```

```
            <div className="ReleasePledge-Box-item-price">
              <p className="text-2xl font-bold text-white">
                Price - {nft.price} Eth
              </p>
              <p>
                <button onClick={turnbackpledge.bind(this, nft)}>
                  取回质押
                </button>
              </p>
            </div>
          </div>
        ))}
      </div>
    </div>
  );
}


```

5.7Model Page

完成功能：

a. 3D模型

```
import React from "react";
import { useEffect, useState } from "react";
import * as THREE from "../utils/three.min.js";
import { OrbitControls } from "../utils/OrbitControls.js";
import { OBJLoader } from "../utils/OBJLoader.js";
import { MTLLoader } from "../utils/MTLLoader.js";


export default function Model(props) {
  var container; // 容器
  var renderer; // 渲染器
  var scene; // 场景
  var camera; // 相机
  var controller; // 视角控制器
  var light; // 光源
  var stats; // 性能检测器
```

```
17  useEffect(() => {
18    threeStart();
19  }, []);
20  // 初始化
21  async function threeStart() {
22    // canvas容器
23    container = document.getElementById("canvas");
24    container.id = "canvas" + props.data.tokenId;
25    // document.body.appendChild(container);
26    // 场景
27    initScene();
28    // 渲染器
29    initThree();
30    // 相机
31    initCamera();
32    // 插件
33    plugIn();
34    // 光
35    initLight();
36    // 画的内容
37    initObject();
38    // 游戏循环
39    render();
40  }
41
42  // 渲染器
43  function initThree() {
44    // 创建渲染器
45    renderer = new THREE.WebGLRenderer({
46      // 在 css 中设置背景色透明显示渐变色
47      alpha: true,
48      // 开启抗锯齿
49      antialias: true,
50    });
51    // 渲染背景颜色同雾化的颜色
52    renderer.setClearColor(scene.fog.color);
53    // 定义渲染器的尺寸；在这里它会填满整个屏幕
54    // console.log(container.clientWidth, container.clientHeight);
55    renderer.setSize(256, 272);
56    // renderer.setSize(window.innerWidth, window.innerHeight);
```

```
    // 打开渲染器的阴影地图
    renderer.shadowMap.enabled = true;
    // renderer.shadowMapSoft = true;
    renderer.shadowMap.type = THREE.PCFSoftShadowMap;
    // 在 HTML 创建的容器中添加渲染器的 DOM 元素
    container.appendChild(renderer.domElement);
    // 监听屏幕，缩放屏幕更新相机和渲染器的尺寸
    window.addEventListener("resize", handleWindowResize.bind(this), false);
  }


  // 窗口大小变动时调用
  function handleWindowResize() {
    // 更新渲染器的高度和宽度以及相机的纵横比
    // renderer.setSize(window.innerWidth, window.innerHeight);
    // camera.aspect = window.innerWidth / window.innerHeight;
    // camera.updateProjectionMatrix();
  }
  // 插件
  function plugIn() {
    // 性能检测器
    // stats = new Stats();
    // container.appendChild(stats.dom);
    //视角控制
    controller = new OrbitControls(camera, renderer.domElement);
    controller.target = new THREE.Vector3(0, 0, 0); //设置控制点
    controller.autoRotate = true;
    // 点击事件
    container.addEventListener("mousedown", (event) => {
      let mouse = new THREE.Vector2();
      let raycaster = new THREE.Raycaster();
      // 计算鼠标点击位置转换到3D场景后的位置
      mouse.x = (event.clientX / renderer.domElement.clientWidth) * 2 - 1;
      mouse.y = -(event.clientY / renderer.domElement.clientHeight) * 2 + 1;
      // 由当前相机（视线位置）像点击位置发射线
      raycaster.setFromCamera(mouse, camera);
      let intersects = raycaster.intersectObjects(scene.children, true);
      if (intersects.length > 0) {
        // 拿到射线第一个照射到的物体
        console.log(intersects[0].object);
```

```
97        }
98      });
99      // 辅助线
100     // this.scene.add(new THREE.GridHelper(50, 50, 0xffffff, 0x555555));
101   }
102   // 场景
103   function initScene() {
104     scene = new THREE.Scene();
105     // 在场景中添加雾的效果，参数分别代表‘雾的颜色’、‘开始雾化的视线距离’、刚好雾化至看不见的视线距离’
106     scene.fog = new THREE.Fog(0xeeeeee, 1, 600);
107   }
108   // 相机
109   function initCamera() {
110     camera = new THREE.PerspectiveCamera(
111       45,
112       container.clientWidth / container.clientHeight,
113       1,
114       10000
115     );
116     camera.position.z = 80;
117   }
118   // 初始化光
119   function initLight() {
120     // 户外光源
121     // 第一个参数是天空的颜色，第二个参数是地上的颜色，第三个参数是光源的强度
122     let hemisphereLight = new THREE.HemisphereLight(0xaaaaaa, 0x000000, 2);
123
124
125     // 环境光源
126     let ambientLight = new THREE.AmbientLight(0xffffff, 0.2);
127
128
129     // 方向光是从一个特定的方向的照射
130     // 类似太阳，即所有光源是平行的
131     // 第一个参数是关系颜色，第二个参数是光源强度
132     let shadowLight = new THREE.DirectionalLight(0xffffff, 0.9);
133
134
135     // 设置光源的位置方向
```

```
    shadowLight.position.set(50, 50, 50);


    // 开启光源投影
    shadowLight.castShadow = true;


    // 定义可见域的投射阴影
    shadowLight.shadow.camera.left = -400;
    shadowLight.shadow.camera.right = 400;
    shadowLight.shadow.camera.top = 400;
    shadowLight.shadow.camera.bottom = -400;
    shadowLight.shadow.camera.near = 1;
    shadowLight.shadow.camera.far = 1000;


    // 定义阴影的分辨率；虽然分辨率越高越好，但是需要付出更加昂贵的代价维持高性能的表现。
    shadowLight.shadow.mapSize.width = 2048;
    shadowLight.shadow.mapSize.height = 2048;


    // 为了使这些光源呈现效果，需要将它们添加到场景中
    scene.add(hemisphereLight);
    scene.add(shadowLight);
    scene.add(ambientLight);
  }
  // 画内容
  function initObject() {
    const mtlLoader = new MTLLoader();
    function mtlFun(mtlurl, objurl, x, y, z) {
      mtlLoader.load(mtlurl, (mtl) => {
        for (const key in mtl.materialsInfo) {
          let val = mtl.materialsInfo[key];
          let urls = val.map_kd.split("\\");
          val.map_kd = `${props.data.addr}/${urls[urls.length - 1]}`;
          // console.log(val.map_kd);
        }
        mtl.preload();
        const objLoader = new OBJLoader();
```

```
        objLoader.setMaterials(mtl);
        objLoader.load(objurl, (obj) => {
          obj.traverse(function (child) {
            if (child.material) {
              child.material.transparent = true; // 默认为true,可省略
              child.material.alphaTest = 0.7;
              child.material.depthWrite = true; // 默认为true,可省略
            }
          });
          obj.position.set(x, y, z);
          scene.add(obj);
        });
      });
    }
    mtlFun(props.data.mtl, props.data.obj, -15, -30, 0); // 中间
    // mtlFun(
    //
"https://ipfs.io/ipfs/bafybeihcwnwnycaopdhfel7sc3yvt4aq6czxe4a2d45fhqv4wj55neo4pq/3.mtl"
,
    //
"https://ipfs.io/ipfs/bafybeihcwnwnycaopdhfel7sc3yvt4aq6czxe4a2d45fhqv4wj55neo4pq/3.obj"
,
    //   -15,
    //   -30,
    //   0
    // ); // 中间
  }
  // 游戏循环
  function render() {
    renderer.clear();
    renderer.render(scene, camera);
    requestAnimationFrame(render);
    // stats.update();
    // TWEEN.update();
  }


  return (
    <div
      id="canvas"
      className="webgl"
```

```
      refs="webgl"
      style={{
        // display: "inline-block",
        width: "256px",
        height: "272px",
        backgroundColor: "#000",
        border: "0",
      }}
    ></div>
  );
}

```

技术支持：

十二生肖网络科技有限公司

技术部：总监：吕磊

前端：陆俊佑 杨加骏

3D模型：曾源