基于区块链的数字藏品交易平台

1 问题描述

1.1 背景介绍

随着互联网时代的到来，数字技术不断成熟，现实世界中，稀有和独特的物品仅存在于固定的物理空间中，而在元宇宙世界中，区块链技术将构建一个去中心化的云算力市场，通过智能合约打造全新经济系统。以区块链技术为基础的非同质化代币（Non-Fungible Token，后简称 NFT）作为非同质化通证，能够映射虚拟物品，使之具备数字所有权和可验证性，也就是数字藏品。

数字藏品作为网络经济领域新兴事物应运而生，逐渐成为一个新型文化消费形态，受到越来越多消费者的关注和青睐。基于区块链技术开发的数字藏品，具有不可复制、不可篡改、永久存证的特性，将艺术品收藏推向了网络化、数字化、智能化的新赛道。2021 年，数字藏品大量涌现，此年也被称为数字藏品元年。当前，数字化的藏品受到了广泛的喜爱，并成功地建立了一个完整的产业链。

NFT 物理层面是基于区块链技术生成一串机器的代码，可将资产的相关权利内容、过往交易流转信息等记录在智能合约的标示信息中，并在对应的区块链上给该特定资产生成一个无法篡改的独特编码，且可通过交易流转实现特定资产的价值流转。数字藏品是一种将图片、音频、模型等数据基于区块链技术形成的非同质化加密数字凭证，能够记录数据的历史交易信息、交易方数字签名、资产元数据（或指向元数据的链接）以及赋予持有人或创作者的相关权益等内容，是一种数字资产记录的形式。

代币是同质化的，意味着每个代币的价值和特性是相同的。例如，1 个 USDT 代币的价值与另一个 USDT 代币相同。NFT 是非同质化的，每个 NFT 都有独特的属性和价值。例如，两个不同的数字艺术作品的 NFT 可能具有不同的价值和特性。所以，本平台上的数字藏品通过非同质化代币来保证唯一性。

数字藏品作为区块链技术的落地产物，具备了区块链技术的以下特性：

（1）唯一性：区块链技术为数字藏品提供了独一无二的标识编码，不可复制、不可篡改、不可分割。基于这一特性，数字藏品可以与实物一样进行确权，为知识产权保护提供新的思路。

（2）永久性：数字藏品作为数字代码，其数据储存在去中心化的区块链上，不易丢失、不可篡改、不会受到物理损害，为数字资产的永久性保存、收藏提供了一种选择。

（3）可交易性：基于一系列所有权、传输等标准，使得数字藏品与其他资产一样，可以进行交易或赠予。不同于国外的 NFT，由于国内弱交易化要求，数字藏品需要在特

定的环境下才可以进行二次交易。

（4）可追溯性：自创建起，数字藏品的铸造、流转等全周期信息都会在链上记录且公开透明，用户可通过代码标识对相关信息进行溯源、验证真伪，保证数据的真实性。

（5）可编程性：数字藏品具有智能合约可编程的特性，开发者可以通过编码赋予其更多的特性，这一特点为数字藏品的应用场景提供了丰富的可能性。如版税可以编程到通证中，使得创作者可以通过设置从而在作品每次销售中获得版税。

1.2 需求分析

该数字藏品交易平台旨在为买方、卖方提供一个便捷、安全的数字藏品交易环境，同时平台需要具备有效的管理和运营功能。

1.2.1 买方

数字藏品交易平台买方端的用例图如图 1 所示。

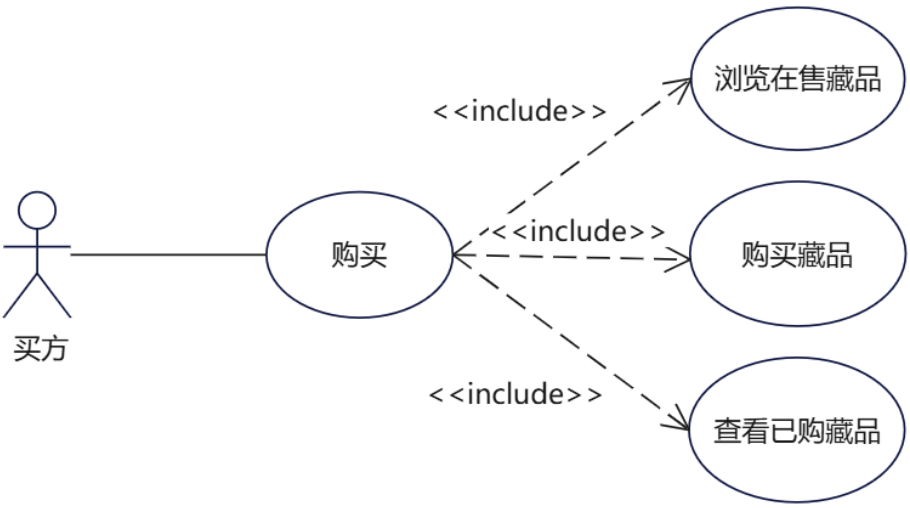

图 1 买方用例图

（1）浏览在售藏品：买方需要能够查看平台上所有可供购买的数字藏品信息，包括藏品名称、图片、描述、NFT 编码、价格、收藏者、以及上一位收藏者收藏的时间等信息。

（2）购买藏品：当买方决定购买某一藏品时，能够方便地发起购买操作，完成支付流程，并确保交易的安全性和可靠性。

（3）查看已购藏品：买方希望能够查看自己已经购买到的数字藏品清单，包括藏品的各类详细信息。

1.2.2 卖方

数字藏品交易平台卖方端的用例图如图 2 所示。

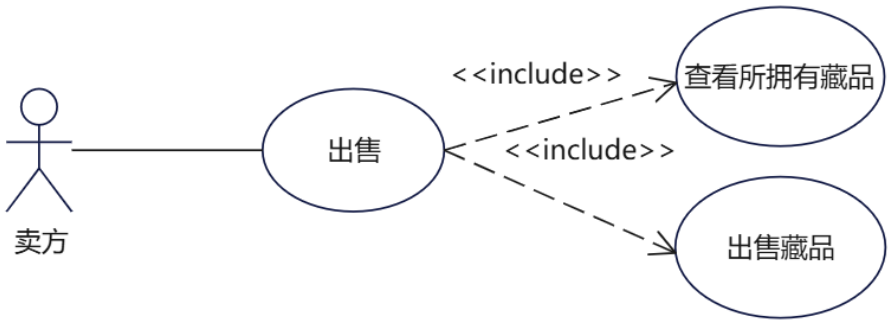

图 2 卖方用例图

（1）查看所拥有藏品：卖方应能够查看自己所拥有的全部数字藏品清单，包括藏品的一系列详细信息。

（2）出售藏品：卖方可以从自己所拥有的藏品中选择进行出售操作。

1.2.3 平台

数字藏品交易平台的用例图如图 3 所示。

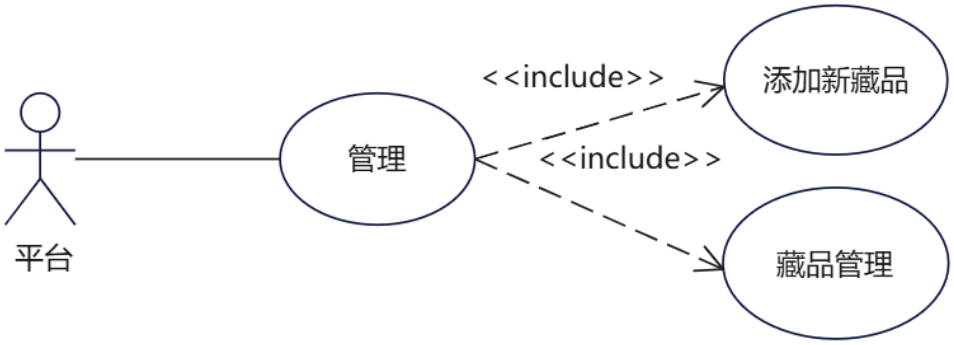

图 3 平台用例图

（1）添加新藏品：添加新的数字藏品，包括输入藏品的详细信息（如名称、价格、描述等），上传相关图片（如有需要）。

（2）藏品管理：负责对平台上的所有数字藏品进行管理，包括对藏品的交易状态、买卖方账户藏品状态等界面上的内容进行实时更新。当买卖方对藏品交易之后，平台应执行智能合约的以太币交易，并负责更新双方的账户状态和页面内容。

1.3 模块设计

数字藏品交易平台的功能模块设计如图 4 所示。该平台共有三大核心功模块：购买、出售和管理。购买模块面向买方，包括浏览在售藏品、购买藏品、查看已购藏品三个功能。出售模块面向卖方，包括查看所拥有藏品和出售藏品两个功能。管理模块面向平台，包括添加新藏品、藏品管理两个功能。

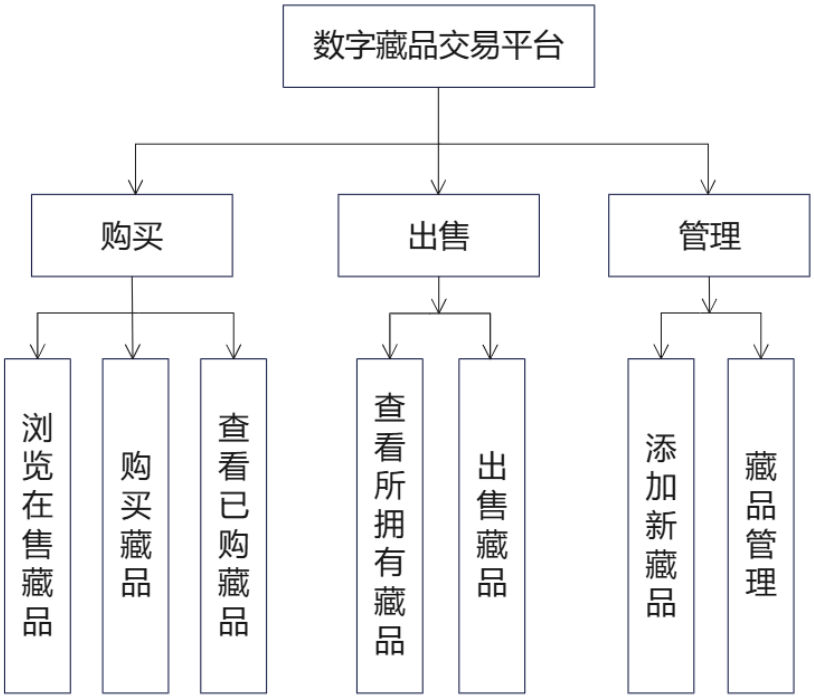

图 4 数字藏品交易平台功能模块设计

2 代码设计

代码已上传 GitHub 仓库：https://github.com/maochenxi/blockchain。

2.1 智能合约实现

2.1.1 初始化设置

（1）合约声明与导入：如图 5 所示，使用 pragma 指令指定 Solidity 编译器的版本，并引入 OpenZeppelin 的 ERC721 合约库，提供 NFT 的基本功能。

```
contracts > DigitalCollectibleContract.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;
pragma experimental ABIEncoderV2;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol"; // 引入 ERC721 合约
```

图 5 引入 NFT 相关的 OpenZeppelin 合约库

（2）数据结构定义：如图 6 所示，合约 DigitalCollectibleContract 继承了 ERC721，能够铸造代币，并定义了一个数据结构。数据结构中设置了每个数字藏品的属性，包括藏品名称、图片、价值、收藏时间戳、NFT 编码、藏品描述、收藏者和是否在售状态。其中，在售状态是指用户从平台购买藏品后，再次出售，挂在平台寄售。

```
contract DigitalCollectibleContract is ERC721 {
    struct DigitalCollectible {
        string name;
        string picture;
        uint value;
        uint timestamp;
        uint NFTCode;
        string description;
        address payable owner;
        bool forSale;
    }
```

图 6 定义数字藏品数据结构

（3）初始化映射及状态变量：如图 7 所示，nfts 映射代币 ID 到数字藏品信息，ownerTokens 映射账户地址到其拥有的代币 ID 列表，计数器 nftCount 记录当前数字藏品的数量。

```
    mapping(uint256 => DigitalCollectible) public nfts;
    mapping(address => uint256[]) public ownerTokens;
    uint256 public nftCount=8;
```

图 7 初始化映射及状态变量

（4）初始化合约：如图 8 所示，定义构造函数来初始化合约，设置 NFT 的名称和符号，并在合约部署时预先创建一些数字藏品，并将这些数字藏品存储在 nfts 这个映射中。每个藏品的在售状态均初始化为 false，因为首次出售都是平台出售，而不是用户寄售。

```
    constructor() ERC721("DigitalCollectible", "DCT") {

        nfts[1] = DigitalCollectible(
                "Statue of the Six Elders", // "六尊者像"
                "images/1.jpg",
                3,
                0,
                0,
                "The 'Statue of the Six Elders' is a painting by the Tang Dynasty artist Lu Lengjia, curre
                payable(address(0)),
                false
            );

        nfts[2] = DigitalCollectible(
                "Gold Medal of the Primordial Heavenly Sovereign",
```

图 8 初始化合约

（5）事件监听：如图 9 所示，进行事件声明。代码定义了三个事件 NFTPurchased、NFTSold、NFTAdded，用于记录 NFT 的购买、出售和添加操作，便于前端监听和更新

状态。

```
event NFTPurchased(uint nftId, address newOwner);
event NFTSold(uint nftId, address newOwner);
event NFTAdded(uint nftId, string name, address owner);
```

图 9 事件监听

2.1.2 购买藏品逻辑实现

购买藏品分为两种，NFTPurchase 函数是购买平台出售的藏品，也就是藏品第一次被购买；buySellNFT 函数是购买其他收藏者出售的藏品，使用不同的逻辑处理方式。

如图 10 所示，NFTPurchase 函数处理用户购买数字藏品的逻辑，允许用户购买平台出售的数字藏品，转移所有权并铸造 NFT。它首先检查所请求的 NFT 是否存在且不在卖方寄售状态，然后验证买方发送的以太币是否足够。接着，调用 transfer 函数，买方转移藏品价值的以太币给平台提供的地址。付款成功后，更新 NFT 的收藏者信息、NFT 编码和收藏时间戳，并铸造新的 NFT 给买方。最后，函数更新 ownerTokens 映射，将新铸造的 NFT ID 添加到购买者的代币列表中，并触发购买事件 NFTPurchased 以记录交易。

```
//第一次购买数字藏品
function NFTPurchase(uint nftId) public payable{
    require(nftId < nfts.length, "NFT does not exist");
    require(!nfts[nftId].forSale, "NFT already for sale");
    require(msg.value >= nfts[nftId].value * 1 ether, "Incorrect Ether amount provided.");

    nfts[nftId].owner.transfer(nfts[nftId].value);
    nfts[nftId].owner = payable(msg.sender);
    nfts[nftId].NFTCode = nftId;
    nfts[nftId].timestamp = block.timestamp;

    _mint(msg.sender, nftId); // 铸造 NFT

    ownerTokens[msg.sender].push(nftId); // 将新铸造的代币 ID 添加到所有者的数组中
    emit NFTPurchased(nftId, msg.sender);
}
```

图 10 第一次购买藏品函数

如图 11 所示，buySellNFT 的函数，用于购买他人出售的数字藏品。函数首先检查所请求的 NFT 是否存在、用户提供的以太币是否足够，以及该 NFT 是否在卖方寄售状态。接着，函数向 NFT 的当前拥有者支付以太币，并更新 NFT 的收藏者，同时记录 NFT 编码和收藏时间戳。最后，铸造新的 NFT 并将其添加到新收藏者的代币列表中，同时触发 NFTPurchased 事件，与前端进行交互。

```
//购买别人出售的数字藏品
function buySellNFT(uint nftId) public payable {
    require(nftId < nfts.length, "NFT does not exist");
    require(msg.value >= 0.1 ether, "Not enough Ether provided.");
    require(nfts[nftId].forSale, "NFT not for sale");

    nfts[nftId].owner.transfer(nfts[nftId].value);
    nfts[nftId].forSale = false;
    nfts[nftId].owner = payable(msg.sender);
    nfts[nftId].NFTCode = nftId;
    nfts[nftId].timestamp = block.timestamp;

    _mint(msg.sender, nftId); // 铸造 NFT

    ownerTokens[msg.sender].push(nftId); // 将新铸造的代币 ID 添加到所有者的数组中

    emit NFTPurchased(nftId, msg.sender);
}
```

图 11 购买他人出售的藏品

2.1.3 出售藏品逻辑实现

如图 12 所示，这段代码定义了一个智能合约中的两个函数：isTokenOwnedBy 和 sellNFTs_new。isTokenOwnedBy 函数用于检查某个地址是否拥有特定的 NFT 代币，通过遍历该地址的代币列表来实现。sellNFTs_new 函数则用于出售数字藏品，首先验证 NFT 是否存在、调用者是否为所有者，并确认其确实拥有该代币。若所有条件满足，则销毁该代币并将其标记为待售，从 nft 的映射中清除藏品被收藏的一些信息。同时从所有者的代币列表中移除该代币。

```
//查找某个代币是否是这个address的
function isTokenOwnedBy(address owner, uint nftId) internal view returns (bool) {
    uint256[] memory tokens = ownerTokens[owner];
    for (uint i = 0; i < tokens.length; i++) {
        if (tokens[i] == nftId) {
            return true; // 找到代币，返回 true
        }
    }
    return false; // 未找到代币，返回 false
}

//出售数字藏品
function sellNFTs_new(uint nftId) public {
    require(nftId < nfts.length, "NFT does not exist");
    require(nfts[nftId].owner == msg.sender, "Not the owner");
    require(isTokenOwnedBy(msg.sender, nftId), "Token not owned by this address"); // 添加检查
    require(ownerOf(nftId) == msg.sender, "Not the token owner"); // 再次确认代币的主人
    _burn(nftId); // 销毁代币
    nfts[nftId].forSale = true;
    nfts[nftId].NFTCode = 0;
    nfts[nftId].timestamp = 0;

    // 从所有者的代币列表中移除该代币
    uint256[] storage tokens = ownerTokens[msg.sender];
    for (uint i = 0; i < tokens.length; i++) {
        if (tokens[i] == nftId) {
            tokens[i] = tokens[tokens.length - 1]; // 用最后一个代币替换当前代币
            tokens.pop(); // 移除最后一个代币
            break;
        }
    }
    emit NFTSold(nftId, msg.sender);
}
```

图 12 出售所拥有的藏品

2.1.4 添加新藏品逻辑实现

如图 13 所示，addNFTs 函数，用于添加新的数字藏品。函数接收三个参数：藏品名称、价值和描述。每次调用该函数时，nftCount 会增加 1，生成一个新的 NFT ID，并将新的数字藏品信息存储在 nfts 映射中。最后，触发 NFTAdded 事件，通知外部前端系统有新的 NFT 被添加。

```
    function addNFTs(
        string memory name,
        uint value,
        string memory description
    ) public {
        nftCount = nftCount+1;
        uint newNFTId = nftCount;
        nfts[newNFTId]=DigitalCollectible({
                value: value,
                name: name,
                description: description,
                picture: "images/7.jpg",
                timestamp: 0,
                NFTCode: 0,
                owner: payable(address(0)),
                forSale: false
            });
        emit NFTAdded(newNFTId, name, msg.sender);
    }
```

图 13 添加新藏品

2.1.5 其他辅助函数逻辑实现

如图 14 所示，getOwnedTokens(address owner) 函数用于查询某个 owner 地址名下的所有代币 ID，并返回一个包含这些代币 ID 的数组。

getNFTsByOwner(uint nftId) 函数用于获取特定代币 ID（nftId）的详细信息，包括代币的名称、描述、价值、是否在出售、图片、NFT 编码和时间戳等信息。返回的结果是一个包含这些信息的元组。

```
    // 查询某个地址名下的所有代币 ID
    function getOwnedTokens(address owner) public view returns (uint256[] memory) {
        return ownerTokens[owner]; // 返回该地址拥有的所有代币 ID
    }

    function getNFTsByOwner(
        uint nftId
    )
        public
        view
        returns (
            string memory,
            string memory,
            uint,
            bool,
            // address,
            string memory,
            uint,
            uint
        )
    {
        DigitalCollectible memory nft = nfts[nftId];
        return (
            nft.name,
            nft.description,
            nft.value,
            nft.forSale,
            // ownerOf(nftId),
            nft.picture,
            nft.NFTCode,
            nft.timestamp
        );
    }
```

图 14 其他辅助函数

2.2 前端展示实现

如图 15 所示，这段 HTML 代码主要用于展示数字藏品。它包含了一个用于显示用户已购买的数字收藏品的区域 purchasedNFTsRow，以及一个用于展示可供购买的数字收藏品的区域 NFTsRow。在 NFTTemplate 部分，定义了一个 NFT 的模板，包括其图片、描述、编码、价值、收藏者信息、收藏时间戳和购买/出售按钮。这个模板在页面加载时是隐藏的，通过 app.js 中的 JavaScript 代码动态生成和显示，并控制不同操作按钮的显示和隐藏。

```
<div class="container">
  </div>
  <h3>My Digital Collectibles</h3>
  <div id="purchasedNFTsRow" class="row">
    <!-- PURCHASED NFTS LOAD HERE -->
  </div>
  <h3>Digital Collectible Shop</h3>
  <div id="NFTsRow" class="row">
    <!-- NFTS LOAD HERE -->
  </div>
</div>
Ctrl+L to chat, Ctrl+K to generate
<div id="NFTTemplate" style="display: none;">
  <div class="col-sm-6 col-md-4 col-lg-3">
    <div class="panel panel-default panel-NFT">
      <div class="panel-heading">
        <h3 class="panel-title">Scrappy</h3>
      </div>
      <div class="panel-body">
        <img alt="140x140" data-src="holder.js/140x140" class="img-rounded img-center" style="width: 100%;"
          src="https://animalso.com/wp-content/uploads/2017/01/Golden-Retriever_6.jpg" data-holder-rendered="true">
        <br /><br />
        <strong>藏品描述</strong>: <span class="NFT-description">Warren, MI</span><br /><br />
        <strong>藏品NFT编码</strong>: <span class="NFT-NFTCode">Warren, MI</span><br /><br />
        <strong>价值</strong>: <span class="NFT-value">3</span><br /><br />
        <strong>收藏者</strong>: <span class="NFT-Collector" style="word-wrap: break-word;">Warren, MI</span><br /><br />
        <strong>获得时间</strong>: <span class="NFT-timestamp">Golden Retriever</span><br /><br />
        <button class="btn btn-default btn-purchase" type="button" style="background-color: green; color: white;">Purchase</button>
        <button class="btn btn-default btn-sell" type="button" style="display: none; background-color: rgb(60, 107, 182); color: white;">Sell</button
      </div>
    </div>
  </div>
</div>
```

图 15 前端 HTML 代码

2.3 前端交互实现

app.js 文件作为应用的主要逻辑文件，负责与区块链交互、处理用户输入和更新用户界面。

2.3.1 初始化设置

(1) 初始化模块：如图 16 所示，初始化模块 init 的代码定义如何加载 NFT 数据并初始化 HTML 模板。首先初始化应用程序并加载 NFT 数据，从 collectibles.json 文件中获取 JSON 数据，遍历每个 NFT 的属性（如名称、图片、时间戳、价值、NFT 编码和描述），并将这些信息填充到 HTML 模板中，最后将生成的 NFT 模板添加到页面的 #NFTsRow 元素中。完成后，会调用 App.initWeb3() 来初始化 Web3 相关的功能。

```
init: async function () {
  // Load nfts.
  $.getJSON('../collectibles.json', function (data) {
    var NFTsRow = $('#NFTsRow');
    var NFTTemplate = $('#NFTTemplate');
    for (i = 0; i < data.length; i++) {
      NFTTemplate.find('.panel-title').text(data[i].name);
      NFTTemplate.find('img').attr('src', data[i].picture);
      NFTTemplate.find('.NFT-timestamp').text(data[i].timestamp);
      NFTTemplate.find('.NFT-value').text(data[i].value);
      NFTTemplate.find('.NFT-NFTCode').text(data[i].NFTCode);
      NFTTemplate.find('.NFT-description').text(data[i].description);
      NFTTemplate.find('.btn-purchase').attr('data-id', data[i].id);
      NFTTemplate.find('.btn-purchase').attr('data-value', data[i].value);
      NFTTemplate.find('.btn-sell').attr('data-id', data[i].id);
      NFTTemplate.find('.btn-sell').attr('data-value', data[i].value);
      // console.log("id",data[i].id);
      // console.log("value",data[i].value);
      NFTsRow.append(NFTTemplate.html());
    }
  });
  return await App.initWeb3();
},
```

图 16 初始化模块 init

(2) 初始化 Web3：如图 17 所示，这段代码定义了一个异步函数 initWeb3，用于初始化与以太坊区块链的连接。首先，它检查浏览器中是否存在 window.ethereum 对象，如果存在，则请求用户授权访问其以太坊账户。成功后，获取用户的账户并存储在 App.account 中。如果用户拒绝访问，则捕获错误并在控制台输出相应的错误信息。如果 window.ethereum 不存在，则检查 window.web3 ，并使用当前的 Web3 提供者。最后，如果都不满足，则创建一个新的 HTTP 提供者连接到本地的以太坊节点。该函数还调用 web3.eth.getAccounts 方法来获取账户信息，并在发生错误时记录错误信息。最终，函数返回 App.initContract()，用于初始化合约。

```
initWeb3: async function () {
  if (window.ethereum) {
    App.web3Provider = window.ethereum;
    try {
      // 请求账户访问
      await window.ethereum.enable();
      // 获取账户
      const accounts = await window.ethereum.request({ method: 'eth_requestAccounts' });
      App.account = accounts[0]; // 存储账户
    } catch (error) {
      // 用户拒绝账户访问...
      console.error("User denied account access");
    }
  }
  else if (window.web3) {
    App.web3Provider = window.web3.currentProvider;
  }
  else {
    App.web3Provider = new Web3.providers.HttpProvider('http://localhost:7545');
  }
  web3 = new Web3(App.web3Provider);
  web3.eth.getAccounts(function (error, accounts) {
    if (error) {
      console.error(error);
    }
    App.account = accounts[0];
  });
  return App.initContract();
},
```

图 17 初始化 Web3

（3）初始化合约：如图 18 所示，initContract 函数用于初始化以太坊智能合约并设置 NFT 相关的事件监听器。

首先，它通过 web3.eth.getBlockNumber 获取当前区块号，并在控制台打印结果。如果没有错误，则继续执行；否则，记录错误信息。接着，使用 jQuery 的 $.getJSON 方法加载名为 DigitalCollectibleContract.json 的合约文件，并将其解析为 NFTArtifact。然后，创建一个 Truffle 合约实例，并设置其提供者为 App.web3Provider。

（1）监听购买藏品的事件：调用 deployed() 函数部署合约后，使用 watch 方法监听 NFTPurchased 事件，捕获事件信息并在控制台打印。如果事件成功触发，则提取 NFT ID 和新拥有者的信息，并使用 Toastify 显示购买成功的提示；如果发生错误，则记录错误信息。

```
initContract: function () {
  var blocks = web3.eth.getBlockNumber(function (error, result) {
    if (!error) {
      console.log("block", result);
    } else {
      console.error(error);
    }
    return result;
  });
  $.getJSON('DigitalCollectibleContract.json', function (data) {
    var NFTArtifact = data;
    App.contracts.DigitalCollectibleContract = TruffleContract(NFTArtifact);
    App.contracts.DigitalCollectibleContract.setProvider(App.web3Provider);
    App.contracts.DigitalCollectibleContract.deployed().then(function (instance) {
      var NFTInstance = instance;
      NFTInstance.NFTPurchased({}, { fromBlock: 'latest' }).watch(function (error, event) {
        console.log(event)
        if (!error) {
          var nftId = event.args.nftId;
          var newOwner = event.args.newOwner;
          Toastify({
            text: 'NFT Purchased - NFT ID: ' + nftId + ', New Owner: ' + newOwner,
            duration: 3000,  // Toast 显示的时间（以毫秒为单位）
            gravity: 'bottom',  // Toast 的位置
            position: 'right',  // Toast 的对齐方式
            backgroundColor: 'white',  // Toast 的背景颜色
          }).showToast();
        } else {
          console.error('Error:', error);
        }
      });
    });
```

图 18 初始化合约

（2）监听出售藏品的事件：如图 19 所示，调用 deployed() 函数部署合约后，使用 watch 方法监听 NFTSold 事件，捕获事件信息并在控制台打印。如果事件成功触发，则提取 NFT ID 和新拥有者的信息，并使用 Toastify 显示出售成功的提示；如果发生错误，则记录错误信息。

```
    // 监听 NFTSold 事件
    App.contracts.DigitalCollectibleContract.deployed().then(function (instance) {
      var NFTInstance = instance;
      NFTInstance.NFTSold({}, {  fromBlock: 'latest' }).watch(function (error, event) {
        if (!error) {
          var nftId = event.args.nftId;
          var newOwner = event.args.newOwner;
          Toastify({
            text: 'NFT Sold - NFT ID: ' + nftId + ', New Owner: ' + newOwner,
            duration: 3000,
            gravity: 'bottom',
            position: 'right',
            backgroundColor: 'white',
          }).showToast();
        } else {
          console.error('Error:', error);
        }
      });
    });
```

图 19 监听出售藏品事件

（3）监听添加藏品的事件：如图 20 所示，调用 deployed() 函数部署合约后，使用 watch 方法监听 NFTAdded 事件，捕获事件信息并在控制台打印。

```
    // 监听 NFTAdded 事件
    App.contracts.DigitalCollectibleContract.deployed().then(function (instance) {
      var NFTInstance = instance;
      NFTInstance.NFTAdded({}, { fromBlock: 'latest' }).watch(function (error, event) {
        if (!error) {
          var nftId = event.args.nftId;
          var name = event.args.name;
          var owner = event.args.owner;
          Toastify({
            text: 'NFT Added - NFT ID: ' + nftId + ', Name: ' + name + ', Owner: ' + owner,
            duration: 3000,
            gravity: 'bottom',
            position: 'right',
            backgroundColor: 'white',
          }).showToast();
        } else {
          console.error('Error:', error);
        }
      });
    });
    return App.markPurchased();
  });

  return App.bindEvents();
},
```

图 20 监听添加藏品事件

（4）事件绑定：如图 21 所示，bindEvents 函数将点击事件绑定到特定的按钮上。函数使用 jQuery 的 $(document).on('click', ...) 方法，为类名为.btn-purchase、.btn-add 和.btn-sell 的按钮分别绑定点击事件处理函数 App.handlePurchase、App.handleAdd 和 App.handleSell。当用户点击这些按钮时，相应的处理函数将被调用，以执行与购买、添加或出售相关的操作。相应的处理函数代码将会在下文事件处理小节介绍。

```
  bindEvents: function () {
    $(document).on('click', '.btn-purchase', App.handlePurchase);
    $(document).on('click', '.btn-add', App.handleAdd);
    $(document).on('click', '.btn-sell', App.handleSell);
  },
```

图 21 事件绑定到按钮

2.3.2 NFT 管理

（1）markPurchased：如图 22 所示，markPurchased 函数用于展示平台中所有的藏品，并针对藏品的在售状态、收藏者、收藏时间、NFT 编码等信息，对藏品进行不同的前端展示。该函数首先调用合约实例 NFTInstance.getAllNfts 方法获取所有 NFT 的信息，并在循环中逐个处理每个 NFT。对于每个 NFT，提取其名称、描述、图像、NFT 编码和时间戳等信息，并使用模板填充这些数据。接着，格式化时间戳为可读的日期格式，并更新模板中的相应字段。如果 NFT 代码存在，更新 NFT 的拥有者信息，并根据 NFT 的状态（如出售或购买）设置按钮的文本和属性。最后，将填充好的模板添加到页面中以显示已购买的 NFT。若在过程中发生错误，则记录错误信息。

```
144     markPurchased: function () {
151       }).then(async function (count) {
152         for (var i = 1; i <= count.c[0]; i++) {
153           await NFTInstance.getAllNFTs.call(i).then(function (nft) {
154             const [name, description, value, sale, image, NFTCode, timestamp] = [nft[0], nft[1], nft[2], nft[3], nft[4], nft[5], nf
155             var NFTTemplate = $('#NFTTemplate');
156             NFTTemplate.find('.panel-title').text(name);
157             NFTTemplate.find('img').attr('src', image);
158             NFTTemplate.find('.NFT-description').text(description);
159             NFTTemplate.find('.NFT-value').text(value.c[0]);
160             NFTTemplate.find('.NFT-NFTCode').text(NFTCode);
161             var date = new Date(timestamp * 1000);
162             var formattedDate = date.getFullYear() + '.' +
163                                 String(date.getMonth() + 1).padStart(2, '0') + '.' +
164                                 String(date.getDate()).padStart(2, '0') + ' ' +
165                                 String(date.getHours()).padStart(2, '0') + ':' +
166                                 String(date.getMinutes()).padStart(2, '0') + ':' +
167                                 String(date.getSeconds()).padStart(2, '0');
168             NFTTemplate.find('.NFT-timestamp').text(formattedDate);
169             NFTTemplate.find('.NFT-Collector').text(App.account);
170             if (NFTCode == 0 || timestamp == 0) {
171               NFTTemplate.find('.NFT-NFTCode').text("None");
172               NFTTemplate.find('.NFT-Collector').text("None");
173               NFTTemplate.find('.NFT-timestamp').text("None");
174             }
175             NFTTemplate.find('.btn-purchase').attr('data-id', i)
176             if (sale) {
177               NFTTemplate.find('.btn-purchase').text("出售").attr("disabled", false);
178             }
179             else if (NFTCode != 0) {
180               NFTTemplate.find('.btn-purchase').text("已售出").attr("disabled", true);
181             }
182             else {
183               NFTTemplate.find('.btn-purchase').text("购买").attr("disabled", false);
184             }
185             NFTsRow.append(NFTTemplate.html());
186           });
187         }
188       }).catch(function (err) {
189         console.log(err.message);
190       });
191       App.displayOwnedNFTs();
```

图 22 展示平台中所有藏品

（2）displayOwnedNFTs：如图 23 所示，异步函数 displayOwnedNFTs 能够显示用户拥有的 NFT。首先，函数通过 web3.eth.getBalance 获取账户的以太坊余额，并将其显示在页面上。接着，调用智能合约实例中的 getOwnedTokens(account) 方法，获取该账户地址拥有的 NFT ID。对于每个 NFT，调用合约的 getNFTsByOwner 方法，传入 NFT ID，即可获取详细信息，包括名称、描述、图像、NFT 编码和时间戳等。然后，克隆一个 NFT 模板，并将这些信息填充到模板中。时间戳被格式化为可读的日期格式，并更新模板中的相应字段。最后，将填充好的 NFT 模板添加到页面中以显示该账户拥有的 NFT。

```
193     displayOwnedNFTs: async function () {
197       web3.eth.getBalance(account, function (error, balance) {
204           acc.append("your balance " + balanceEth);
205         } else {
206           console.error('获取余额时出错:', error);
207         }
208       });
209       App.contracts.DigitalCollectibleContract.deployed().then(async function (instance) {
210         NFTInstance = instance;
211         var tokensList = await NFTInstance.getOwnedTokens(account);
212         console.log("Owned Tokens List: ", tokensList);
213         return tokensList;
214       }).then(async function (tokensList) {
215         var NFTsRow = $('#purchasedNFTsRow');
216         var NFTT = $('#NFTTemplate');
217         NFTsRow.html('');
218         for (var i = 0; i < tokensList.length; i++) {
219           const tokenId = tokensList[i]
220           await NFTInstance.getNFTsByOwner(tokenId.c[0]).then(function (nft) {
221             const [name, description, value, sale, image, NFTCode, timestamp] = [nft[0], nft[1], nft[2], nft[3], nft[4], nft[5], nf
222             if (sale === false) {
223               var NFTTemplate = NFTT.clone();
224               NFTTemplate.find('.panel-title').text(name);
225               NFTTemplate.find('img').attr('src', image);
226               NFTTemplate.find('.NFT-description').text(description);
227               NFTTemplate.find('.NFT-value').text(value.c[0]);
228               NFTTemplate.find('.NFT-NFTCode').text(NFTCode);
229               NFTTemplate.find('.NFT-Collector').text(account);
230               var date = new Date(timestamp * 1000); // 假设时间戳是以秒为单位
231               var formattedDate = date.getFullYear() + '.' +
232                                   String(date.getMonth() + 1).padStart(2, '0') + '.' +
233                                   String(date.getDate()).padStart(2, '0') + ' ' +
234                                   String(date.getHours()).padStart(2, '0') + ':' +
235                                   String(date.getMinutes()).padStart(2, '0') + ':' +
236                                   String(date.getSeconds()).padStart(2, '0'); // 添加小时、分钟和秒
237               NFTTemplate.find('.NFT-timestamp').text(formattedDate);
238               NFTTemplate.find('.btn-purchase').hide();
239               NFTTemplate.find('.btn-sell').attr('data-id', tokenId.c[0]).text("出售").attr("disabled", false).show();
240               console.log("tokenId",tokenId.c[0]);
241
242               NFTsRow.append(NFTTemplate.html());
```

图 23 展示该账户拥有的所有藏品

2.3.3 事件处理

（1）购买 NFT：如图 24 所示，handlePurchase 函数，用于处理 NFT 的购买或出售操作。函数首先从点击的 HTML 组件的点击事件中提取出 NFT ID 和 NFT 价值。接着，使用 web3.eth.getAccounts 获取用户的以太坊账户，并请求用户授权访问其账户。然后，获取智能合约的实例，并根据事件目标的文本内容判断是购买还是出售 NFT。如果文本内容为“出售”，则调用合约中的 buySellNFT 方法，购买其他人在平台寄售的藏品；否则，调用合约中 NFTPurchase 方法，购买平台第一次出售的藏品。无论是调用哪个函数，都需要将从点击事件中获取到的 nftValue1 传进函数中，这样才能在智能合约中转账正确的金额。最后，处理结果并调用 markPurchased 函数更新前端界面信息，若发生错误则记录错误信息。

```
handlePurchase: function (event) {
  event.preventDefault();
  var nftId1 = parseInt($(this).data('id'));
  var nftValue1 = parseInt($(this).data('value'));
  var NFTInstance;
  web3.eth.getAccounts(async function (error, accounts) {
    if (error) {
      console.log(error);
    }
    var currentAccount = await window.ethereum.request({ method: 'eth_requestAccounts' });
    var account = currentAccount[0];
    App.contracts.DigitalCollectibleContract.deployed().then(function (instance) {
      NFTInstance = instance;
      if (event.target.textContent === "出售") {
        return NFTInstance.buySellNFT(nftId1, { from: account, value: web3.toWei(nftValue1.toString(), 'ether') });
      }
      else{
        return NFTInstance.NFTPurchase(nftId1, { from: account, value: web3.toWei(nftValue1.toString(), 'ether') });
      }
    }).then(function (result) {
      return App.markPurchased();
    }).catch(function (err) {
      console.log(err.message);
    });
  });
},
```

图 24 处理购买 NFT 事件

（2）添加 NFT：如图 25 所示，handleAdd 函数用于添加新的 NFT。首先，函数通过 prompt 提示输入 NFT 的名称、价值和描述，并将输入的字符串按空格分割成数组 a。接着，检查输入的长度是否为 3，以确保提供了所有必要的信息。如果信息不完整，则弹出警告提示输入完整信息。然后，获取用户的以太坊账户，并请求授权访问。接下来，部署智能合约获得实例，调用合约实例的 addNFTs 方法将新的 NFT 添加到合约中。成功后，调用 displayOwnedNFTs 函数更新界面中展示拥有 NFT 的区域。最后，页面会重新加载以反映最新的状态。

```
handleAdd: async function () {
  var nftName = prompt("Enter nft's (name,value,description):");
  var a = nftName.split(';');
  var currentAccount = await window.ethereum.request({ method: 'eth_requestAccounts' });
  var account = currentAccount[0];
  if (a.length != 3) {
    alert("Please enter all the details");
    return;
  }
  if (nftName) {
    App.contracts.DigitalCollectibleContract.deployed().then(async function (instance) {
      return await instance.addNFTs(a[0], parseInt(a[1]), a[2], { from: account });
    }).then(function (result) {
      App.displayOwnedNFTs();
      console.log("count ", instance.getCount());
    }).catch(function (err) {
      console.log(err.message);
    });
  }
  window.location.reload();
},
```

图 25 处理添加 NFT 事件

（3）出售 NFT：如图 26 所示，handleSell 函数用于处理 NFT 的出售操作。首先，函数从事件对象中提取 NFT 的 ID，并将其存储在 nftName 变量中。接着，获取用户的以太坊账户，并请求授权访问。然后，部署智能合约并获取实例，调用合约实例中的 sellNFTs_new 方法将指定的 NFT 进行出售。成功后，调用 displayOwnedNFTs 函数更新界面以反映最新的 NFT 状态。如果发生错误，则记录错误信息。最后，控制台输出”selled”，表示出售操作已被触发。

```
295    handleSell: async function (event) {
296      var nftName = $(event.target).data('id');
297      console.log("nftName ", nftName);
298      var currentAccount = await window.ethereum.request({ method: 'eth_requestAccounts' });
299      var account = currentAccount[0];
300      // if (nftName) {
301        App.contracts.DigitalCollectibleContract.deployed().then(async function (instance) {
302          return await instance.sellNFTs_new(parseInt(nftName), { from: account });
303        }).then(function (result) {
304          console.log("result ", result);
305          App.displayOwnedNFTs();
306        }).catch(function (err) {
307          console.log(err.message);
308        });
309      // }
310      console.log("selled")
311    }
312  };
```

图 26 处理出售 NFT 事件

3 功能展示

如图 27 所示，利用 Web3.js 实现合约与前端的结合，启动代码后，进入数字藏品交易平台的主界面。

数字藏品交易平台由 My Digital Collectibles 和 Digital Collectible Shop 两大部分组成。My Digital Collectibles 罗列当前账户拥有的数字藏品。Digital Collectible Shop 展示平台在售的所有数字藏品。每一个数字藏品的信息包括：藏品名、藏品图片、藏品描述、藏品 NFT 的编码、藏品价格、收藏者和获得时间。当藏品未售出的时候，藏品的 NFT 编码、收藏者和获得时间都为 None。此外，在页面最上方，还有添加数字藏品和出售数字藏品的按钮，以及当前操作账户的地址和余额，与 Ganache 和 MetaMask 中保持一致。

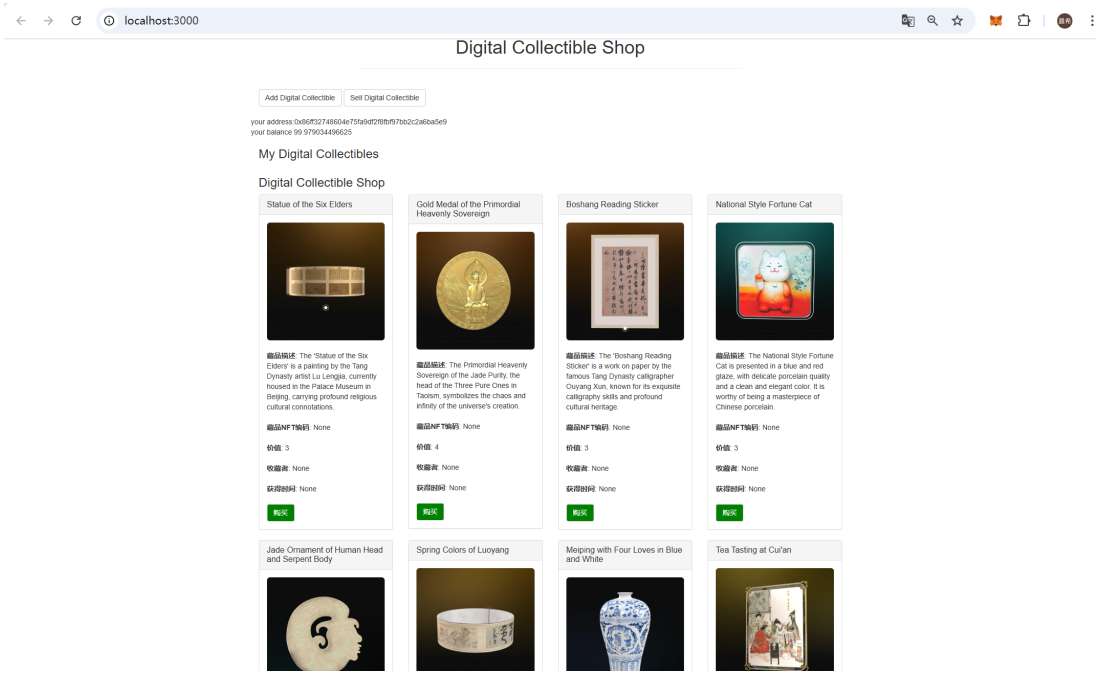

图 27 数字藏品交易平台主界面

3.1 购买数字藏品

如图 28 所示，可以选择一个藏品进行购买。以第一个藏品为例，点击藏品详情下方的绿色的购买按钮，进行购买。此时，藏品的 NFT 编码、收藏者和获得时间都为 None，购买按钮也是亮的，可以购买。

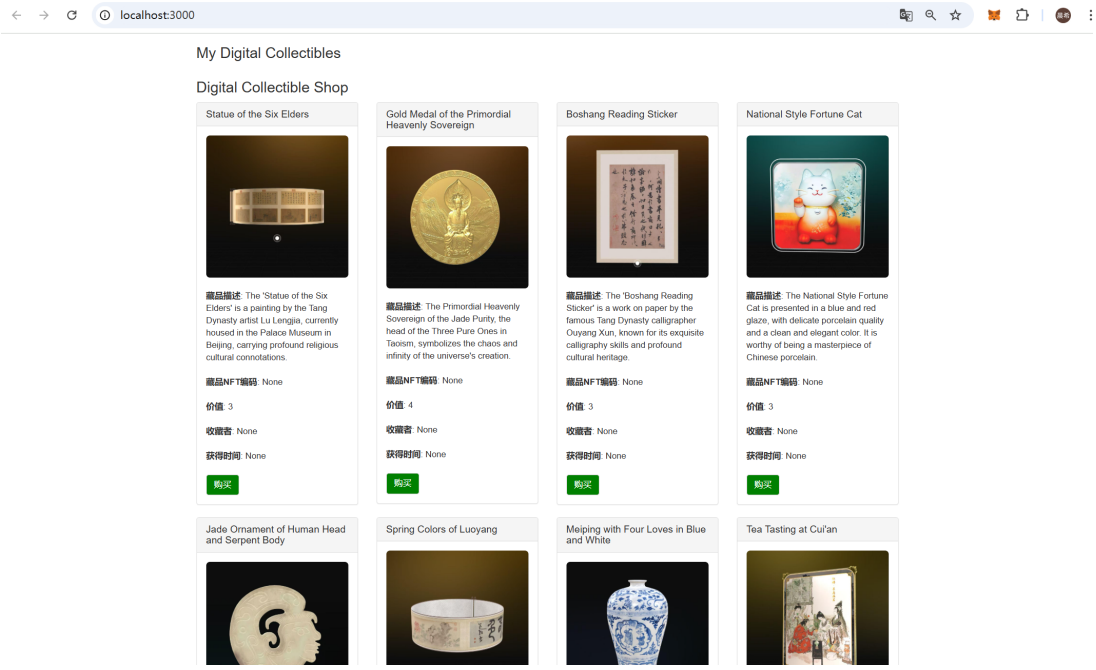

图 28 选择第一个藏品，点击购买按钮

如图 29 所示，MetaMask 弹出交易请求，请求支付藏品价格数量的以太币，并且需要支付一定数量的 Network fee。

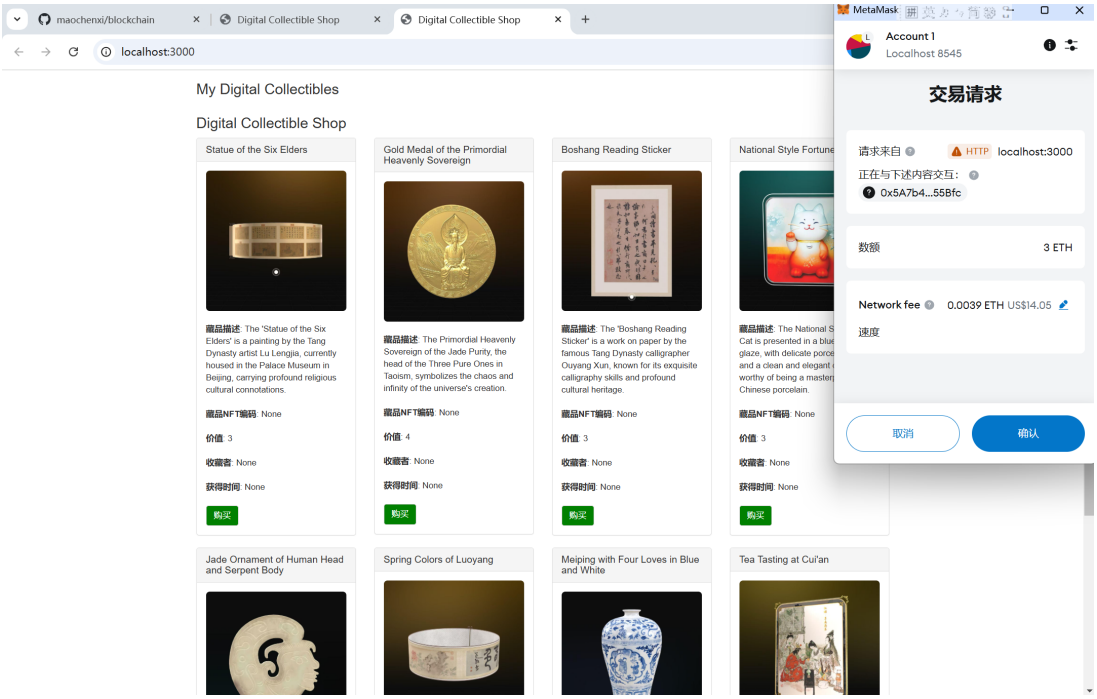

图 29 以太币交易请求

如图 30 所示，确认交易后，页面左上角弹出消息弹窗，提示 NFT 已经被购买，并从 Ganache 和 MetaMask 中更新当前余额，在账户地址下方显示。查看 My Digital Collectibles，当前账户购买到的藏品已经被添加，并且 NFT 编码、收藏者和获得时间也被更新为最新的实际购买信息，而不是原来的 None。并且，每个藏品的按钮变为蓝色的出售按钮。

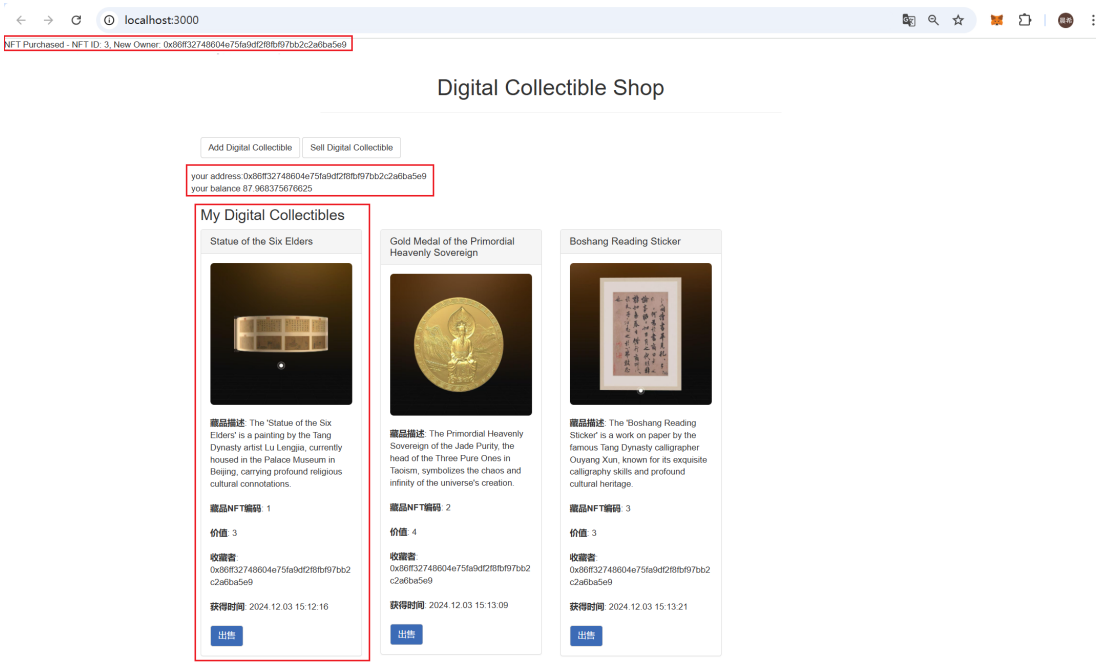

图 30 更新我的藏品界面及相关收藏信息

如图 31 所示，查看 MetaMask 中该账户的收藏品一栏，发现确实有三个数字藏品。

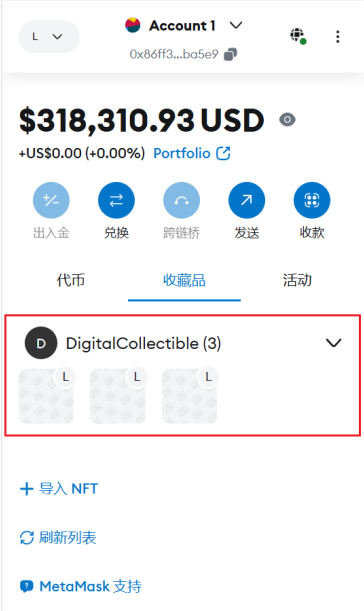

图 31 MetaMask 账户的收藏品

如图 32 所示，查看 Ganache 中此次交易，SENDER ADDRESS 为当前登录的账户地址，交易数量为 3 个以太币，执行成功。

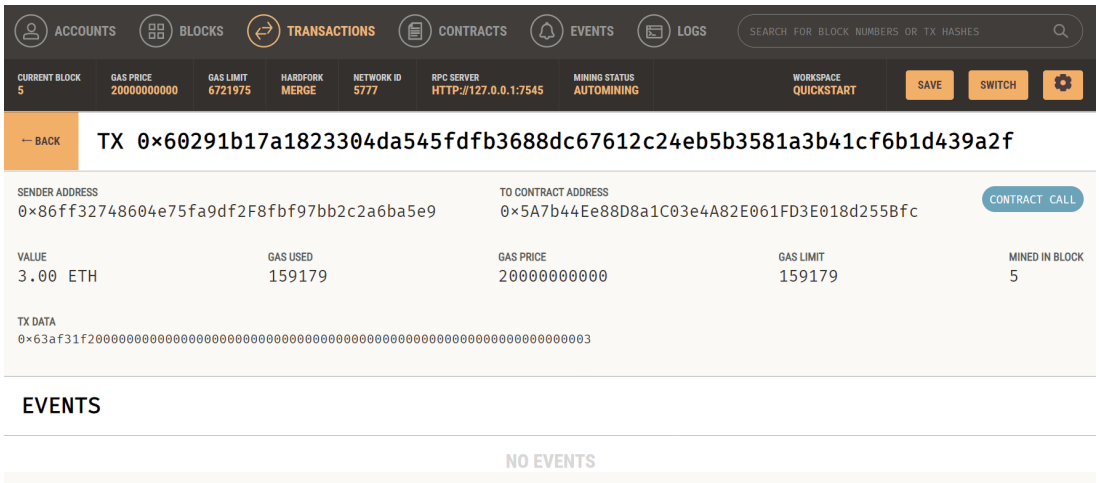

图 32 Ganache 中的交易信息

如图 33 所示，藏品售出后，在 Digital Collectible Shop 中，也会实时更新 NFT 编码、收藏者、获得时间等信息，并且购买按钮变成已售出，不可点击。

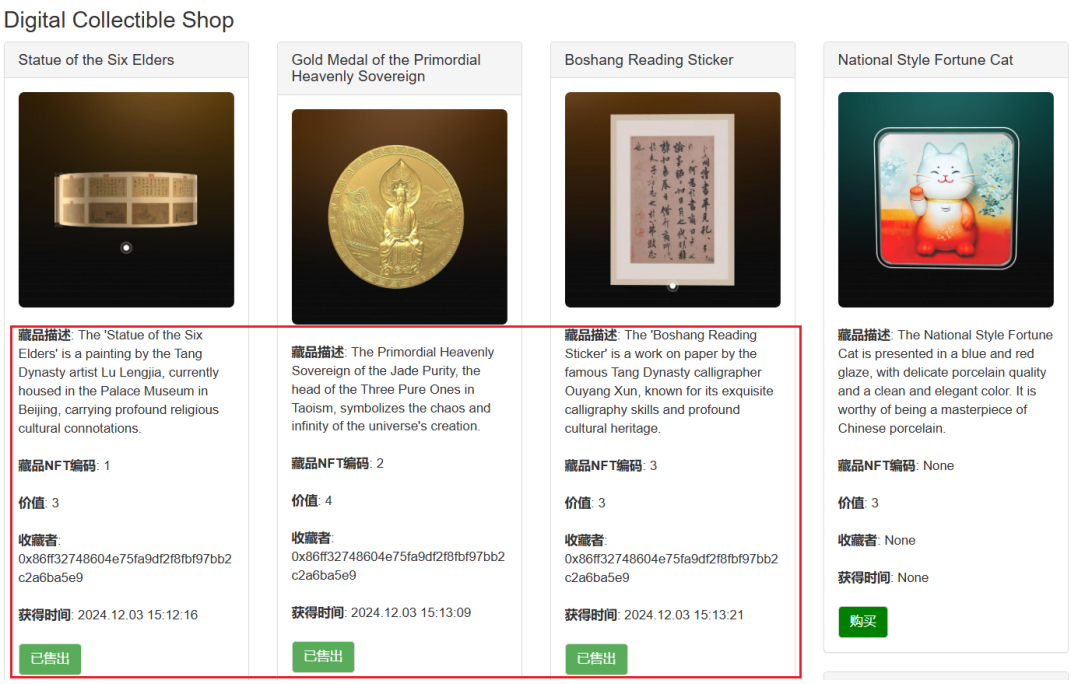

图 33 售出藏品更新信息和已售出按钮

3.2 出售数字藏品

如图 34 所示，在 My Digital Collectibles 中，每个拥有的藏品都有一个蓝色的出售按钮。点击按钮，账户 1 可以选择出售自己拥有的其中一个藏品，将藏品放回 Shop 中供其他账户购买。点击后，MetaMask 弹出交易请求，由于是出售藏品，所以账户 1 只需要支付 NetWork fee，并不需要支付藏品价格的以太币。

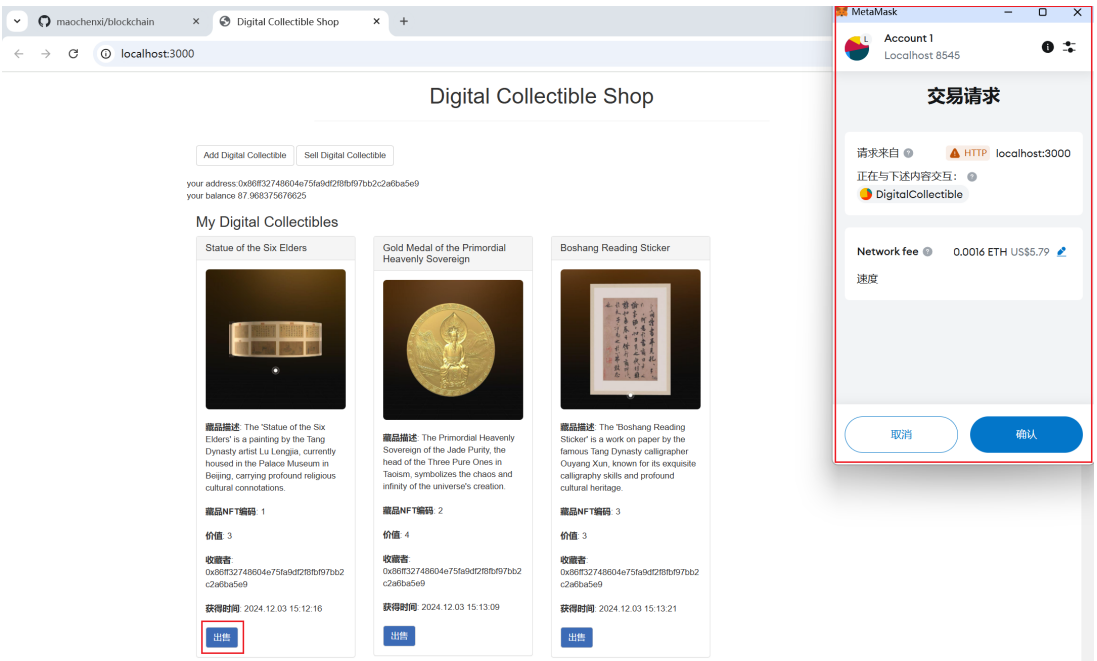

图 34 藏品收藏者出售藏品

如图 35 所示，在 MetaMask 中切换帐户，进入账户 2 的界面。

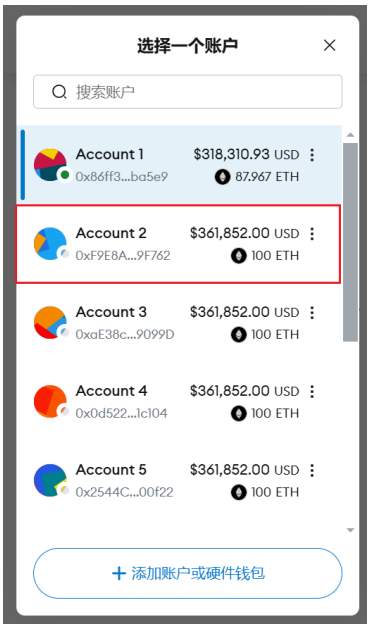

图 35 切换账户

如图 36 所示，左上角的账户地址和余额都已经更新为账户 2 的账户信息，余额为初始的 100 个以太币。在账户 2 的视角中，藏品 2 和藏品 3 都已售出，存在有效的 NFT 编码，收藏者是账户 1 的地址 0x86ff，获得时间就是账户 1 购买的时间。藏品 1 由于已经被账户 1 挂售，所以目前 NFT 编码、收藏者和获得时间都又变为了 None，按钮变为可点击的出售状态。

账户 2 点击出售按钮后，MetaMask 弹出交易请求，交易数额为藏品 1 的价值——3 个以太币。

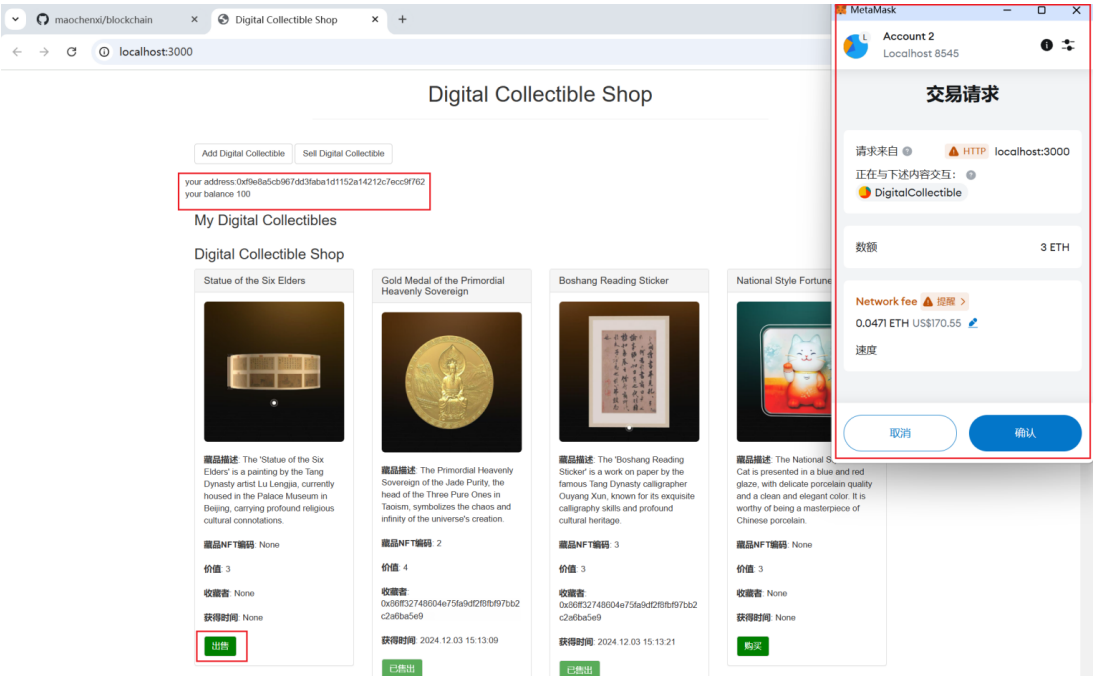

图 36 账户 2 购买账户 1 出售的藏品（图片有问题，要处理）

如图 37 所示，交易完成后，可以看到账户 2 的账户余额被扣除了 3 个以太币的价格和 NetWork fee，藏品 1 更新入账户的 My Digital Collectibles，NFT 编码为 1，收藏者为账户 2 的地址，获得时间为账户 2 购买操作的时间。并且 MetaMask 中账户 2 名下也出现了一个收藏品。

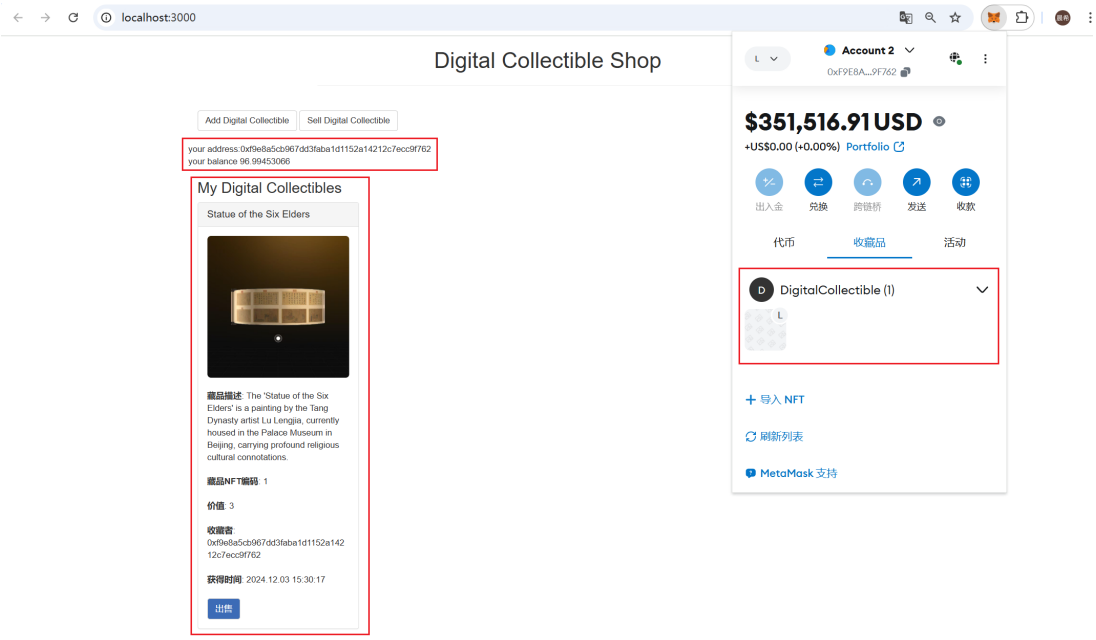

图 37 账户 2 购买藏品成功

3.3 添加新数字藏品

如图 38 所示，点击 Add Digital Collectible 按钮，在输入框内输入藏品名、价格、描述，可以在交易平台上上架一个新的藏品。

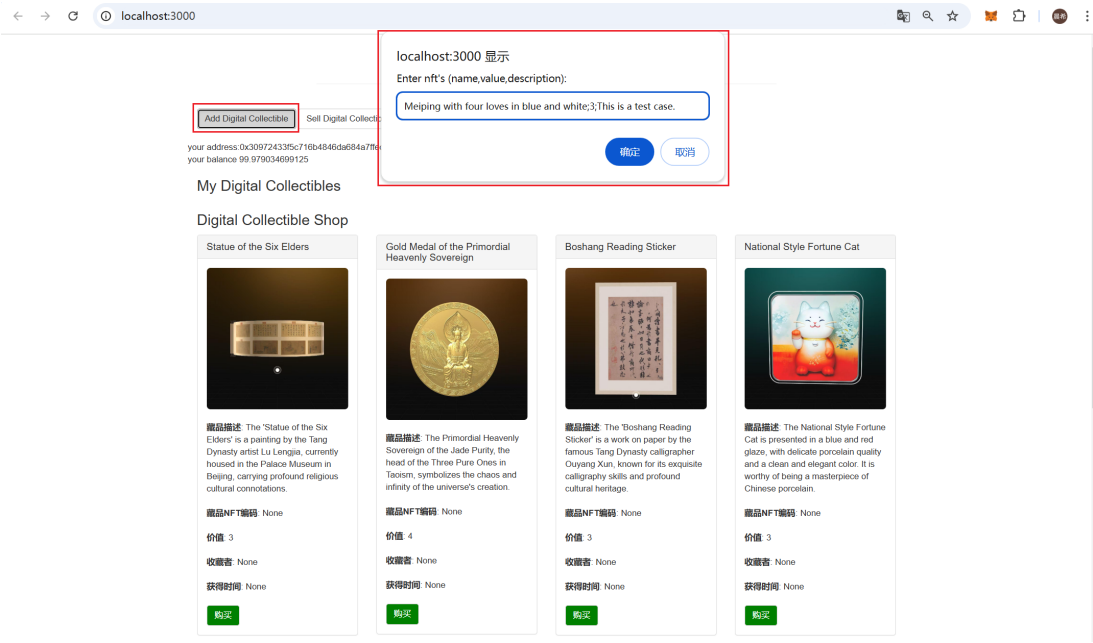

图 38 上架新藏品

如图 39 所示，点击确认之后，MetaMask 弹出交易请求，确认请求即添加新数字藏品成功。

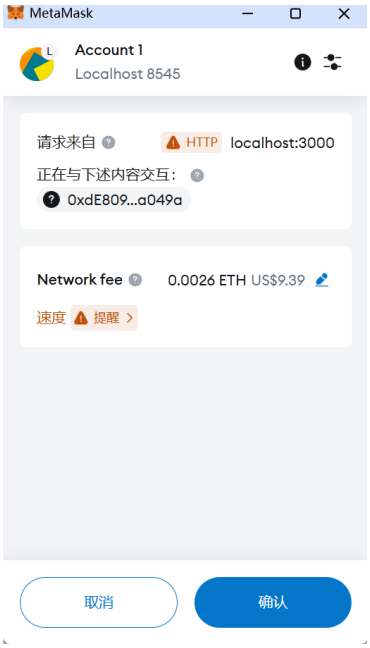

图 39 确认添加新藏品的交易

如图 39 所示，新的藏品已经更新在 Digital Collectible Shop 中，供买家购买。

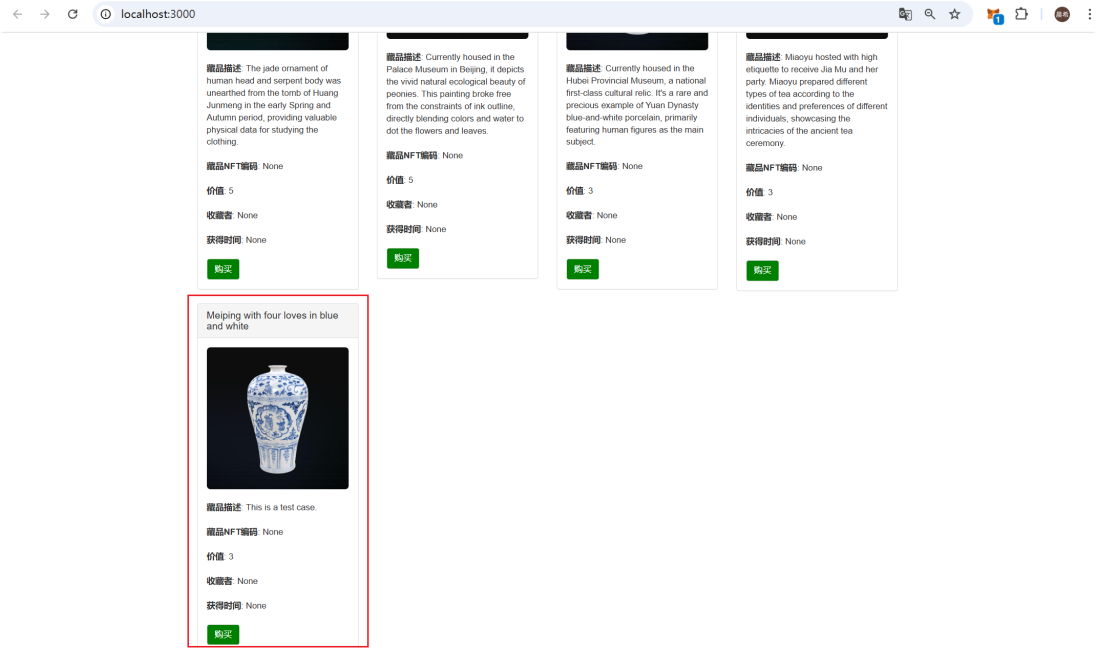
localhost:3000
藏品描述: The jade ornament of human head and serpent body was unearthed from the tomb of Huang Junmeng in the early Spring and Autumn period, providing valuable physical data for studying the clothing.
藏品NFT编码: None
价值: 5
收藏者: None
获得时间: None
购买
藏品描述: Currently housed in the Palace Museum in Beijing, it depicts the vivid natural ecological beauty of peonies. This painting broke free from the constraints of ink outline, directly blending colors and water to dot the flowers and leaves.
藏品NFT编码: None
价值: 5
收藏者: None
获得时间: None
购买
藏品描述: Currently housed in the Hubei Provincial Museum, a national first-class cultural relic. It's a rare and precious example of Yuan Dynasty blue-and-white porcelain, primarily featuring human figures as the main subject.
藏品NFT编码: None
价值: 3
收藏者: None
获得时间: None
购买
藏品描述: Miaoyu hosted with high etiquette to receive Jia Mu and her party. Miaoyu prepared different types of tea according to the identities and preferences of different individuals, showcasing the intricacies of the ancient tea ceremony.
藏品NFT编码: None
价值: 3
收藏者: None
获得时间: None
购买
Meiping with four loves in blue and white
藏品描述: This is a test case.
藏品NFT编码: None
价值: 3
收藏者: None
获得时间: None
购买

图 40 上架新藏品成功