摘　要

本研究基于区块链平台实现了一种创新的商品拍卖系统，旨在提供更安全、透明和高效的拍卖环境。该系统具备以下主要功能：

商品展示与存储：利用区块链平台，实现了商品展示和文件存储功能。商品的图像和描述等大文本信息被上传至 IPFS（InterPlanetary File System），确保了数据的安全性和可靠性。

智能过滤与浏览：用户可以根据商品的类别、拍卖时间等条件，通过 Category 界面过滤功能快速浏览并找到感兴趣的商品。这一功能大大提高了用户的浏览效率和体验。

维克里密封拍卖：系统实现了维克里密封拍卖，保证了拍卖过程的公平性和安全性。维克里密封拍卖采用加密技术，确保竞拍者的出价信息在拍卖过程中不被泄露，从而防止了潜在的欺诈行为。

通过以上功能，本系统为商品拍卖提供了一种全新的解决方案，为用户和商家提供更加安全、便捷的交易环境。

本系统是基于以太坊上的智能合约，采用面向对象技术实现的，通过 css 的渲染，优化了 html 系统界面。

关键词：区块链，智能合约，以太坊，拍卖

目录

第一章 绪 论

1.1 区块链拍卖系统介绍

1.1.1 在线拍卖所存在的问题

传统拍卖环境下的风险防范措施不能保证在线拍卖业务的保密性、完整性和安全性。在线拍卖欺诈成为网络欺诈的最普遍形式，不仅影响拍卖交易双方的诚信度，而且直接影响在线拍卖商的利益。
此外，拍卖方需要交给第三方管理中心佣金以及其它费用，导致利润减少最后可能使得商品价格虚高，让消费者承担费用。

1.1.2 区块链技术为在线拍卖带来的优势

区块链是一种按照时间顺序将数据区块用类似链表的方式组成的数据结构，并以密码学方式保证不可篡改和不可伪造的分布式去中心化账本，能够安全存储简单的、有先后关系的、能在系统内进行验证的数据。区块链技术为交易双方提供无第三方的可信交易，通过参与节点保存的分布式账本，保证交易的不可篡改性、可追溯性和交易完整性；整个区块链网络的扁平拓扑模型确保了参与者的平等身份，保证交易双方的公平性；区块链还使用密码学相关技术，使得交易双方身份保持匿名状态，尤其是区块链的隐私保护技术研究的最新进展，已经可以实现较好的隐私保护；同样，匿名监管也应用到了当前的区块链技术中，保留用户匿名性的同时，所有交易都受到监管方监管。
综上所述，区块链技术可以最大程度的满足在线拍卖的相关需求。安全可靠的在线拍卖系统得以借助区块链技术实现。

1.2 开发框架介绍

本系统的智能合约是基于 truffle，部署到 Ganache 的本地测试以太坊

NodeJS 服务器

IPFS 去中心化存储文件

html 描述网页，css 设置网页样式

第二章 系统分析

2.1 功能分析

2.1.1 基本功能

1. 商品管理：查询商品、添加商品、查看商品
2. 拍卖、竞拍、交易结算
3. 浏览产品 ui 界面：主页，查询，添加商品，查看商品
4. 具有将商品图像和商品描述上传至 IPFS
5. 智能合约：维克里密封拍卖

2.1.2 拓展功能

1. 体验优化：用 css 美化网页

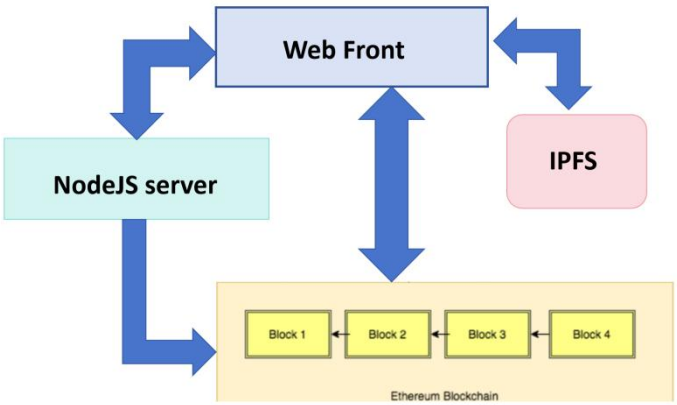
2.2 应用架构

1.Web 前端：web 前端由 HTML，CSS 和 JavaScript 组合而成（大量使用了 web3js）。用户将会通过这个前端应用于区块链，IPFS 和 NodeJS 服务器交互。

2. 区块链：这是应用的核心，所有的代码和交易都在链上。商店里的所有商品，用户的出价和托管合约都在链上。

3.NodeJS 服务器：这是后端服务器，前端通过它与区块链通信。我们会给前端暴露一些简单的 API 从数据库中查询和检索产品。

4. IPFS：当一个用户在商店里上架一个商品，前端会将产品文件和介绍上传到 IFPS，并将所上传文件的哈希存到链上。



第三章 详细设计及实现

3.1 智能合约

利用 Truffle 创建智能合约并部署到 Ganache，使用 MetaMask 支付

3.1.1 存储产品和元数据的数据结构

1.struct Product：当用户想要在商店列出一个商品时，他们必须要输入关于产品的所有细节。商品图片和商品描述存储 IPFS 链接（拓展中会介绍），渲染网页时，会从这些链接中获取这些细节。

```
struct Product{//商品数据结构
        uint id;//商品唯一编号
        string name;//商品名称
        string category;//商品类别
        string imageLink;//商品图片
        string descLink;//商品描述文本
        uint auctionStartTime;//开始拍卖时间
        uint auctionEndTime;//结束拍卖时间
        uint startPrice;//起拍价
        address highestBidder;//最高出价者
        uint highestBid;//最高价
        uint secondHighestBid;//次高价
        uint totalBids;//总竞拍人数
        ProductStatus status;//商品状态
        ProductCondition condition;//商品状况
    }
```

2.enum：将产品的条件和状态存储为枚举类型。所以 Open 的 ProductStatus 在链上存储为 0，已售出则为 1，如此类推。

```
enum ProductStatus{ Open, Sold, Unsold}//商品状态：可竞拍，已卖出，未卖出
    enum ProductCondition{ New, Used}//商品状况：全新，二手
```

3.productIndex：我们会给加入到商店的每个商品赋予一个 id。这就是一个计数器，每当一个商品加入到商店则加 1 。

stores：任何人都可以免费列出商店里的产品。通过 mapping 跟踪谁插入了商品。键位商家的账户地址，值为 productIndex 到 Product 结果的 mapping，相当于每个用户都有一个商店。

productIdInStore：这是一个 mapping，用于跟踪哪些商品在哪个商店。

```
unit public productIndex;//商品计数器
    mapping (address => mapping(uint => Product)) stores;//创建者关联他发布的所有商品
    mapping (uint => address) ProductInStore;//商品关联创建者
```

3.1.2 添加商品

1.新建一个叫做 addProductToStore 的函数，参数为构建 product 结构的所需内容（除了出价相关的变量）。
2.productIndex 计数加 1。
3.使用 require 来验证 auctionStartTime 小于 auctionEndTime。
4.初始化 Product 结构，并用传入函数的参数进行填充。
5.将初始化后的结构存储在 stores mapping。
6.同时在 productionIdInStore mapping 中记录是谁添加了商品。
7.创建一个叫做 getProduct 的函数，它将 productId 作为一个参数，在 stores mapping 中查询商品，返回商品细节

```
//添加商品
    function addProductToStore(
        string memory _name,
        string memory _category,
        string memory _imageLink,
        string memory _descLink,
        uint256 _auctionStartTime,
        uint256 _auctionEndTime,
        uint256 _startPrice,
        uint256 _productCondition
    ) public {
        require(_auctionStartTime < _auctionEndTime);
        productIndex += 1;//商品计数器加 1
        // 添加商品在以 msg.sender 为索引的商店里
        Product storage product = stores[msg.sender][productIndex];
        product.id = productIndex;
        product.name = _name;
        product.category = _category;
        product.imageLink = _imageLink;
        product.descLink = _descLink;
        product.auctionStartTime = _auctionStartTime;
        product.auctionEndTime = _auctionEndTime;
        product.startPrice = _startPrice;
        product.status = ProductStatus.Open;
        product.condition = ProductCondition(_productCondition);
        // 商品 ID 对应的商店
        productIdInStore[productIndex] = msg.sender;
        emit NewProduct(productIndex, _name, _category, _imageLink, _descLink,
_auctionStartTime, _auctionEndTime, _startPrice, _productCondition);
    }
```

用 Ganache 部署合约到区块链

```
   > gas used:           165475 (0x28663)
   > gas price:          3.375 gwei
   > value sent:         0 ETH
   > total cost:         0.000558478125 ETH

   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:      0.000558478125 ETH


2_deploy_contracts.js
=====================

   Deploying 'EcommerceStore'
   --------------------------
   > transaction hash:    0x960071627727052d7a19b7fc931c66b9b79d7c6cee5bc22c718e082f4d58d4fb
   > Blocks: 0            Seconds: 0
   > contract address:    0x34EdAbeB9ab0906cB2b61Bc2DdAD86121C4b0577
   > block number:        3
   > block timestamp:     1713957038
   > account:             0x33e0EcE594BCb02BF653E3f5E9bd4b65448663b4
   > balance:             99.996635446637796595
   > gas used:            836430 (0xcc34e)
   > gas price:           3.175945351 gwei
   > value sent:          0 ETH
   > total cost:          0.00265645596993693 ETH

   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:     0.00265645596993693 ETH

Summary
=======
> Total deployments:   2
> Final cost:          0.00321493409493693 ETH
```

部署成功！

3. 1. 3 维克里密封拍卖

投标者会在不知道其他人标价的情况下出价，且出价时必须同时支付 ETH，标价最高者得标，且只需支付第二高的标价，所有输掉竞价的人将会收回各自出价的 ETH（扣除一些手续费）。

1. 出价

如果 Alice 想要出价 10.50 美元，她将会对出价进行隐藏 keccak256($10.50, "secretstring")产生一个哈希，然后将这个字符串发送出去，同时发送价值 15 美元的 ETH。看到这笔交易的任何人就都知道了她发送了 15 美元，但是没有人知道她只出价 10.50 美元。

在这种情况下，Mary 并不知道 Alice 出价 10.50 美元。她只知道 Alice 发送了 15 美元。如果 Mary 想要出价 15 美元，她可以 keccak256($15, "marysecretstring") 进行哈希，并发送哈希后的字符串，同时发送 15 美元或者更多的 ETH。

注意，用户可能会发送一个小于实际出价的数额来迷惑其他人。比如：Alice 出价 keccak256($30, "secretstring")，但是实际只给合约转了 20 美元。在这种情况下，当她揭示出价时，合约会将这些钱归还回去，因为这是一个无效的出价。

接着 John 出价 12 美元。如果 Alice 想出更高价，尽管已经出价了一次，她仍然可以再次出价。

2. 揭示报价

一旦拍卖结束，所有的出价者都必须揭示各自报价。
为了揭示报价，出价者必须向合约发送他们出价的金额和 secret string（即向合约发送出价步骤中用于 keccak256($10.50, "secretstring")的两个值）。
合约通过将报价金额与 secret string 进行组合构造出哈希后的报价，然后与出价者之前所发送的哈希后的字符串进行匹配。如果匹配成功，则出价有效。否则出价无效，合约会返还相应的资金。

Alice 揭示报价
Alice 通过向合约发送 10.50 美元和 “secretstring” 来揭示她的报价。合约使用同一个算法来生成哈希。由于这是一个有效出价并且 Alice 发送了 15 美元，合约会记录它为有效出价，并将 15 - 10.5 = 4.5 美元返还给 Alice。

Mary 揭示报价
因为她出价 15 美元，她就是最高的出价者。合约会替换掉 Alice，Mary 成为最高的出价者，Alice 成为第二高的出价者。因为 Alice 没有赢得竞价，所以她会拿回自己所有的钱（刚才已经返回$4.5,所以现在返回$10.5）

John 揭示报价
John 仅出价 12 美元，但成为第二高报价。当揭示报价时，因为 John 输掉了竞价所以他会立刻收到返还的资金。

最终 Mary 赢得竞价，并支付 12 美元（第二高的报价）。

3. 代码设计

(1) 出价信息结构

```
struct Bid{
        address bidder;//出价人
        uint productId;//商品
        uint value;//出价人支付金额
        bool revealed;//是否揭示报价
    }
```

在 Bid 中不需要存储竞拍者实际出价的 hash，因为可以通过这个 hash 值对应 mapping 到这个竞拍者，故只需在 Product 结构中新增一行每个商品 mapping 不同的出价 hash 值从而 mapping 对应的出价人。

```
struct Product{//商品数据结构
        uint id;//商品唯一编号
        string name;//商品名称
        ......
    //用存储的出价 hash 值来 mapping 到出价者
```

```
        mapping (address => mapping(bytes32 => Bid)) bids;
    }
```

(2) 出价函数

bid 函数有两个参数，productId 和加密后的 bid 字符串
对 bid 的 hash 加密使用 keccak256（需要确保前端和后端使用同一个 hash 函数，不然可能会加密结果不一样）

```
//出价
    function bid(uint _productId, bytes32 _bid) public payable returns(bool){
        Product storage product = stores[ProductInStore[_productId]][_productId];
        require(now >= product.auctionStartTime);//当前出价时间不早于开始拍卖时间
        require(now <= product.auctionEndTime);//当前出价时间不晚于结束拍卖时间
        require(msg.value > product.startPrice);//出价要大于起拍价
        require(product.bids[msg.sender][_bid].bidder != msg.sender);//同一个人不能提交两次相同出价
        product.bids[msg.sender][_bid] = Bid(msg.sender, _productId, msg.value, false);
        product.totalBids += 1;//竞拍人数加 1
        return true;
    }
```

(3) 揭示出价函数

揭示出价就是要告诉合约你出价了多少。方式就是将的 secret string 和你打算出价的数量发送给合约。合约将同样的 kaccak256 算法应用于出价数量和 secret，并检查所生成的哈希是否在 bids mapping 里面。

```
require(now > product.auctionEndTime);//揭示时间大于结束拍卖时间
bytes32 sealedBid = keccak256(abi.encodePacked(_amount, _secret));//hash
```

当执行检查的时候，可能会出现以下场景：

```
Bid memory bidInfo = product.bids[msg.sender][sealedBid];//得到出价者信息
        require(bidInfo.bidder != address(0));//出价人存在
        require(bidInfo.revealed == false);//出价还未揭示
```

① 没有找到相关的哈希。这意味着用户尝试揭示不曾出价过的数量。在这种情况下，仅抛出一个异常
② 出价数量小于发送数量：比如用户出价 10 美元，但是只发送了 5 美元，这是无效的，所以我们只需要将这 5 美元返回给用户。
出价数量大于等于发送数量：这是一个有效出价，检查是否应该记录此次出价。
③ 首次揭示：如果这是第一个有效的出价揭示，我们会把它记录为最高出价，同时记录是谁出的价。将第二高的出价设置为商品的起始价格。
④ 更高的出价：如果用户揭示了出价，并且他们的出价高于现有所揭示的最高出价，将会记录该出价者，将其出价记录为最高出价，并设置第二高的出价为旧的出价数量。
⑤ 更低的出价：如果出价比最高出价要低，这是一个会失败的出价。但是我们也会检查它是否低于第二高的出价。如果是的话，只需要返还资金即可，因为他们已经输掉了竞价，否则将该出价设置为第二高的出价。

在所有情况中，我们会返还发送数量与实际出价的差额（先不考虑手续费），也就是，如果 Alice 出价 10 美元，但是发送了 15 美元，在揭示出价以后，将会返回给 Alice 5 美元。

```
if(bidInfo.value < amount){//出价小于起拍价
            refund = bidInfo.value;//返回提交金额（不是出价）
        }else{//第一次竞价
            if(address(product.highestBid) == address(0)){
                product.highestBidder = msg.sender;
                product.highestBid = amount;
                product.secondHighestBid = product.startPrice;//起拍价成为第二高价
                refund = bidInfo.value - amount;//退还金额
            }else{ //非第一次竞价
                    if(amount > product.highestBid){//出价大于最高价
                        product.secondHighestBid = product.highestBid;//最高价成为第二高
                        payable(product.highestBidder).transfer(product.highestBid);//退还第二高
                        product.highestBid = amount;
                        product.highestBidder = msg.sender;
                        refund = bidInfo.value - amount;

                    }else if(amount >product.secondHighestBid){//出价大于第二高价
                            product.secondHighestBid = amount;
                            refund = bidInfo.value;
                    }else{//出价无效
                        refund = bidInfo.value;//全部退还
                    }
                }
            }
        product.bids[msg.sender][sealedBid].revealed = true;//修改揭示出价状态为 true
```

(4) 辅助函数

在计算退回金额时，要将发送的 ETH-出价

```
uint  refund;//退款
uint amount = stringToUint(_amount);//转换_amount 类型方便计算
```

但_amount 类型为 string，为了便于计算需要写一个辅助函数来转换成 uint

```
function stringToUint(string memory s) private pure returns(uint){
        bytes memory b = bytes(s);
        uint result = 0;
        for(uint i = 0; i < b.length; i++){
            if(uint8(b[i]) >= 48 && uint8(b[i]) <= 57){//ASCII 码在数字范围内
                result = result * 10 + (uint8(b[i]) - 48);
            }
        }
        return result;
    }
```

(5) 其他功能函数

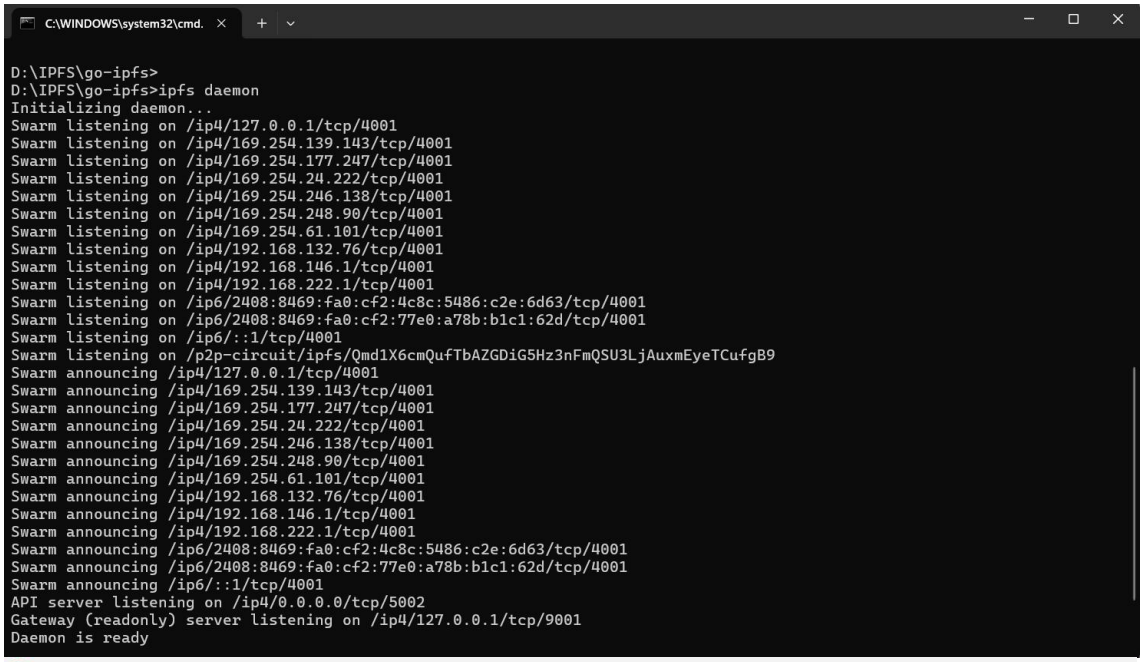
```
//查询最高出价人
    function highestBidderInfo(uint _productId) public view returns (address, uint, uint){
        Product memory product = stores[ProductInStore[_productId]][_productId];
        return (product.highestBidder, product.highestBid, product.secondHighestBid);
    }

    //查询竞价人数
    function totalBids(uint _productId) public view returns (uint){
        Product memory product = stores[ProductInStore[_productId]][_productId];
        return product.totalBids;
    }
```

3.2 IPFS

3.2.1 IPFS 简介

IPFS 是一个点对点的分布式文件系统，旨在用同一个文件系统连接所有的计算设备；是一个被设计用来创建一个永久的，去中心化方式存储和分享文件的协议。在 IPFS 的世界里，这些服务提供商将不再是中心化服务器，而是 P2P 网络里的计算机。与任何人都可以运行一个以太坊节点一样，任何人都可以运行一个 IPFS 节点，并加入网络来形成全球的文件系统。文件会在很多节点间复制，几乎不可能出现无法访问文件的情况，并且防审查。

3.2.2 IPFS 启动



3.2.3 IPFS 上传文件

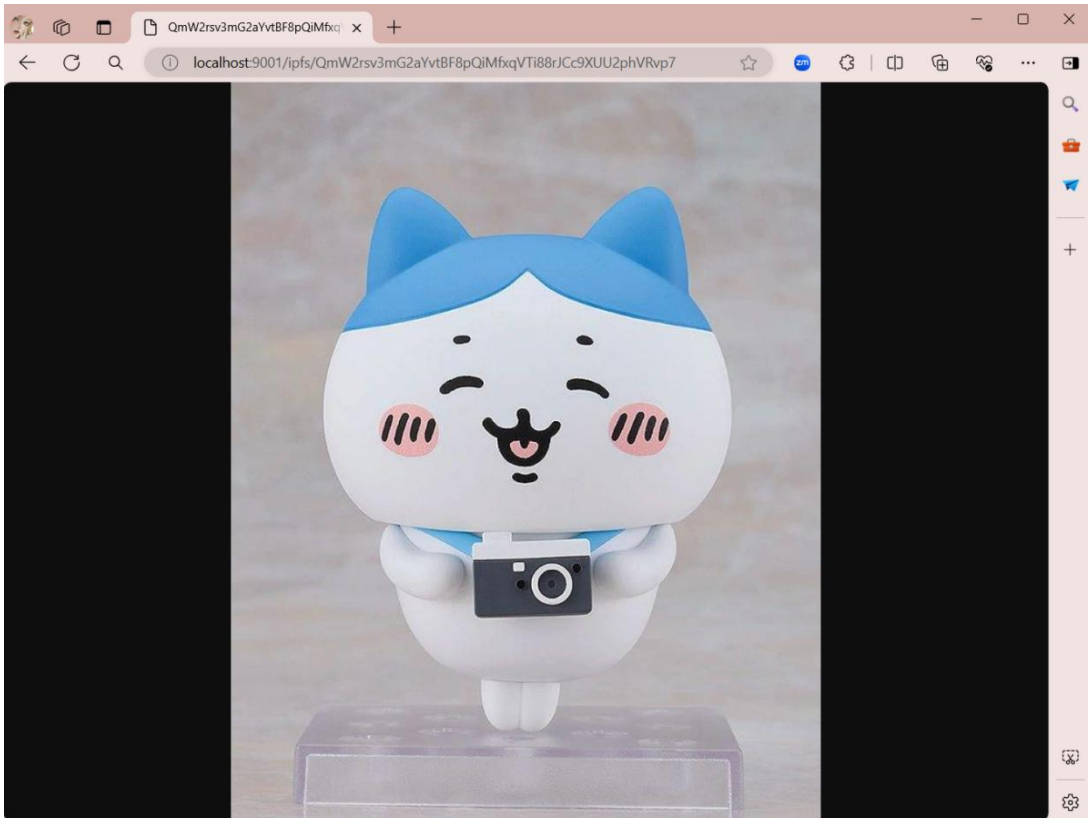
上传命令：ipfs add C:\Users\ASUS\Desktop\Recently\usagi.jpg
获得一串 hash：QmdKEDNMPgZzZ1Bd3q6pf2gnEpyqRkEsP9dHLJCkXTxcCF

```
D:\IPFS\go-ipfs>ipfs.exe add C:\Users\ASUS\Desktop\综合课程设计：区块链拍卖系统\商品
图片\jeans.jpg
added QmazhPWL7eECZv6wo3g1y7kSDezxnPt61NmsiR4WyoKJR5 jeans.jpg

D:\IPFS\go-ipfs>ipfs.exe add C:\Users\ASUS\Desktop\综合课程设计：区块链拍卖系统\商品
\chiikawa.txt
added QmUWUWo69j8umLaKmQ278rmUiDreKzTzoLGMwBaXhbaA1R chiikawa.txt

D:\IPFS\go-ipfs>ipfs.exe add C:\Users\ASUS\Desktop\综合课程设计：区块链拍卖系统\商品
\iPhone5.txt
added QmVPnc1vJESqsnCsEuL6GndT2VSRQ7oTxSNcwKcJeaxQVY iPhone5.txt

D:\IPFS\go-ipfs>ipfs.exe add C:\Users\ASUS\Desktop\综合课程设计：区块链拍卖系统\商品
\iPhone6.txt
added QmZU9sfWXPBqN8dEowhJjTiKRU7A2pkRAHH5GUFssj4MHe iPhone6.txt

D:\IPFS\go-ipfs>ipfs.exe add C:\Users\ASUS\Desktop\综合课程设计：区块链拍卖系统\商品
\iPhone12.txt
added QmX7VfaTwKRQEGknzDyMwhJJpRWCzd7bDNKokd7tsc8tRU iPhone12.txt

D:\IPFS\go-ipfs>ipfs.exe add C:\Users\ASUS\Desktop\综合课程设计：区块链拍卖系统\商品
\jeans.txt
added QmVSzr5fim6NRn8R4tE8VsXx3mahrzVqiNRof8NEY89YiX jeans.txt

D:\IPFS\go-ipfs>ipfs.exe add C:\Users\ASUS\Desktop\综合课程设计：区块链拍卖系统\商品
\Usagi.txt
added QmUvpAwRMcdGb7s356UHSrrhS8wdv9zDQVDnNH5vF99PwX Usagi.txt
```

将文件上传到 IPFS 成功后会自动获得一个 hash 值，只需要保存下这个哈希值就可以访问到文件了

访问：

http://localhost:9001/ipfs/QmW2rsv3mG2aYvtBF8pQiMfxqVTi88rJCc9XUU2phVRvp7



3.2.4 实现商品图片和描述上传

为了通过前端与 IPFS 进行交互，使用 ipfs-http-client 的 JavaScript 库。

```
"dependencies": {
    "ipfs-http-client": "^50.1.2",
  }
```

Line 5 - 9: 当用户点击 html 中的 file 字段并选择一个文件上传时，触发 change() 事件。将图片内容读取到一个缓冲区。

Line 11 - 23: 使用 JavaScript ipfs 库将图片上传到 IPF,使用 ipfs.add 函数将文件上传到 IPFS。将这个调用封装在一个 promise 中，以便于当我们调用 saveImageOnIpfs，可以等待上传完毕然后执行其他操作。

Line 24 - 36: 将产品介绍上传到 IPFS。

这里仅仅是定义了将资源上传到 IPFS 的函数。我们会在下一节调用这些函数。

```
const ipfsAPI = require('ipfs-api');
const ipfs = ipfsAPI({host: 'localhost', port: '5002', protocol: 'http'});

window.App = {
 start: function() {
  ...
  //上传图片和描述到 IPFS
  var reader;
  $('#product-image').change(event => {
    const file = event.target.files[0];
    reader = new window.FileReader();
    reader.readAsArrayBuffer(file);
  });
  ...
function saveImageOnIpfs(reader) {//上传图片
  return new Promise(function(resolve, reject) {
   const buffer = Buffer.from(reader.result);
   ipfs.add(buffer)
   .then((response) => {
    console.log(response)
    resolve(response[0].hash);
   }).catch((err) => {
    console.error(err)
    reject(err);
   });
  });
}

function saveTextBlobOnIpfs(blob) {//上传描述
  return new Promise(function(resolve, reject) {
   const descBuffer = Buffer.from(blob, 'utf-8');
   ipfs.add(descBuffer)
   .then((response) => {
    console.log(response)
    resolve(response[0].hash);
   }).catch((err) => {
    console.error(err)
    reject(err);
   });
  });
}
```

3.3 Web 产品

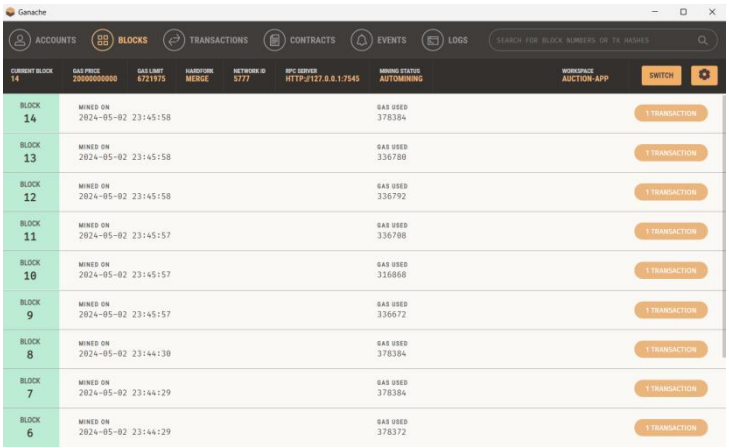
3.3.1 种子区块链

当开发本应用时，为了实现各种用户场景和测试，我们将会不断地将产品添加到区块链。与其通过 truffle 控制台一个一个地添加，创建一个有一些产品的脚本，任何时候我们需要更多的产品时，就运行该脚本。

执行命令： truffle exec seed.js

```
Eutil = require('ethereumjs-util');
EcommerceStore = artifacts.require("./EcommerceStore.sol");
module.exports = function() {
 current_time = Math.round(new Date() / 1000);//定义时间戳
 EcommerceStore.deployed().then(function(i) {i.addProductToStore('iphone 5', 'Cell
Phones & Accessories', 'QmW3bi4YHRhJWe3WTnJRkAx9Qpa5fkTcKENindiCzACLAN',
'QmVPnc1vJESqsnCsEuL6GndT2VSRQ7oTxSNcwKcJeaxQVY', current_time, current_time + 200,
web3.utils.toWei('2','ether'), 0).then(function(f) {console.log(f)})});
 EcommerceStore.deployed().then(function(i) {i.addProductToStore('iphone 6', 'Cell
Phones & Accessories', 'QmSxkP7vSVvMJWqQNjpuhQGTmLuEwnnC2xxbpNajWTCBZh',
'QmZU9sfWXPBqN8dEowhJjTiKRU7A2pkRAHH5GUFssj4MHe', current_time, current_time + 400,
web3.utils.toWei('3','ether'), 1).then(function(f) {console.log(f)})});
 EcommerceStore.deployed().then(function(i) {i.addProductToStore('iphone 12', 'Cell
Phones & Accessories', 'Qmf3aNDtmWwJfyLLX13WHa7wDjb46NWo1utMFAfjrjxexd',
'QmX7VfaTwKRQEGknzDyMwhJJpRWCzd7bDNKokd7tsc8tRU', current_time, current_time + 14,
web3.utils.toWei('1','ether'), 0).then(function(f) {console.log(f)})});
 EcommerceStore.deployed().then(function(i) {i.addProductToStore('Usagi doll', 'Toys &
Hobbies', 'Qmd7sYfBFjaLt2zMTAgAN7THMhTTuVQ8vcqV9KdwrtdD2A',
'QmUWUWo69j8umLaKmQ278rmUiDreKzTzoLGMwBaXhbaA1R', current_time, current_time + 86400,
web3.utils.toWei('4','ether'), 1).then(function(f) {console.log(f)})});
 EcommerceStore.deployed().then(function(i) {i.addProductToStore('chiikawa doll', 'Toys
& Hobbies', 'QmW2rsv3mG2aYvtBF8pQiMfxqVTi88rJCc9XUU2phVRvp7',
'QmUWUWo69j8umLaKmQ278rmUiDreKzTzoLGMwBaXhbaA1R', current_time, current_time + 86400,
web3.utils.toWei('2','ether'), 1).then(function(f) {console.log(f)})});
 EcommerceStore.deployed().then(function(i) {i.addProductToStore('Jeans', 'Clothing,
Shoes & Accessories', 'QmazhPWL7eECZv6wo3g1y7kSDezxnPt61NmsiR4WyoKJR5',
'QmVSzr5fim6NRn8R4tE8VsXx3mahrzVqiNRof8NEY89YiX', current_time, current_time + 86400,
web3.utils.toWei('5','ether'), 1).then(function(f) {console.log(f)})});
 EcommerceStore.deployed().then(function(i)
{i.productIndex.call().then(function(f){console.log(f)})});
}
```

可以从 Ganache 看见区块的增加

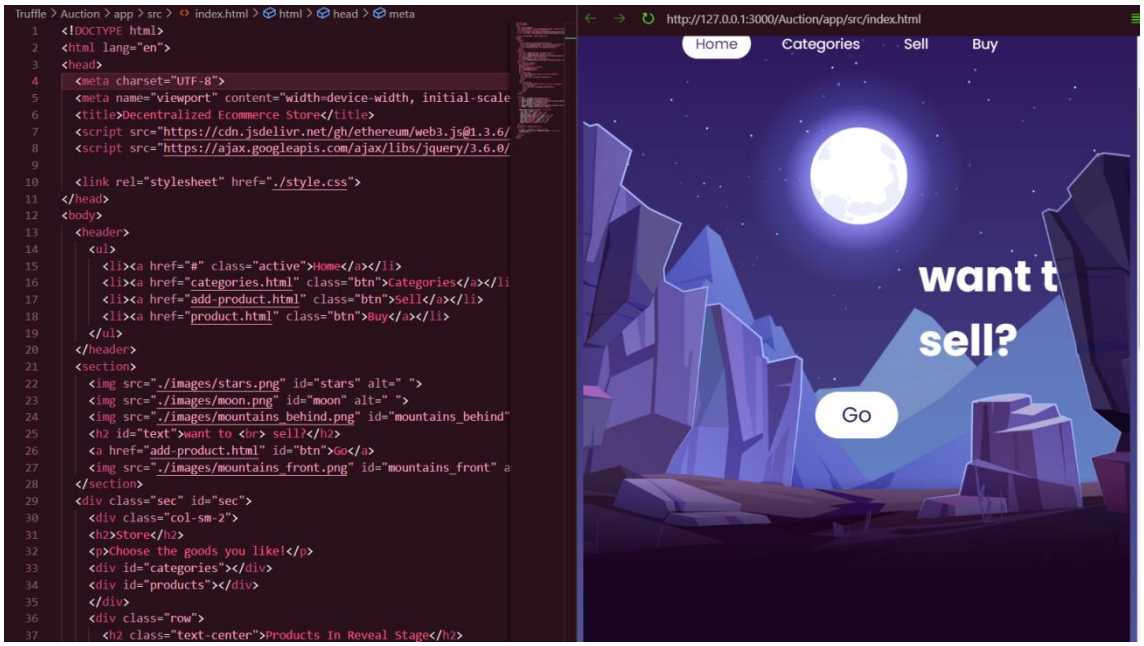


3.3.2 商品 HTML

1.首页

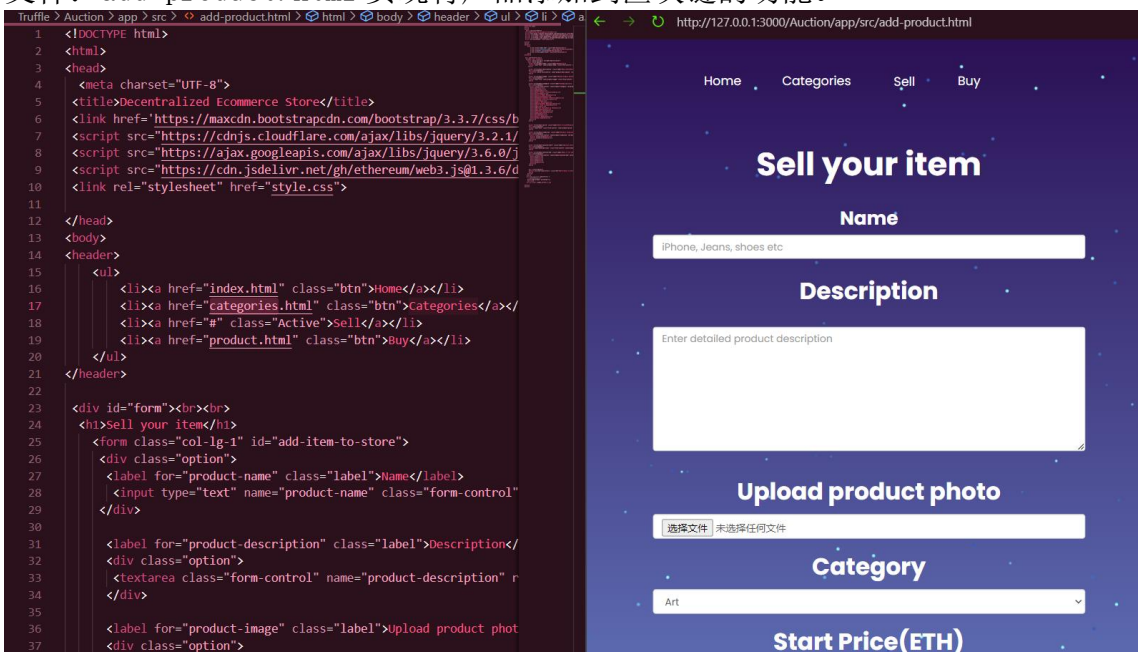
主要有两块内容，一个用来显示目前活跃并可出价的产品，一个用来显示拍卖已结束处于揭示出价阶段的产品，同时也支持通过各种目录过滤产品（手机，衣服，礼品卡片等等）

文件：index.html



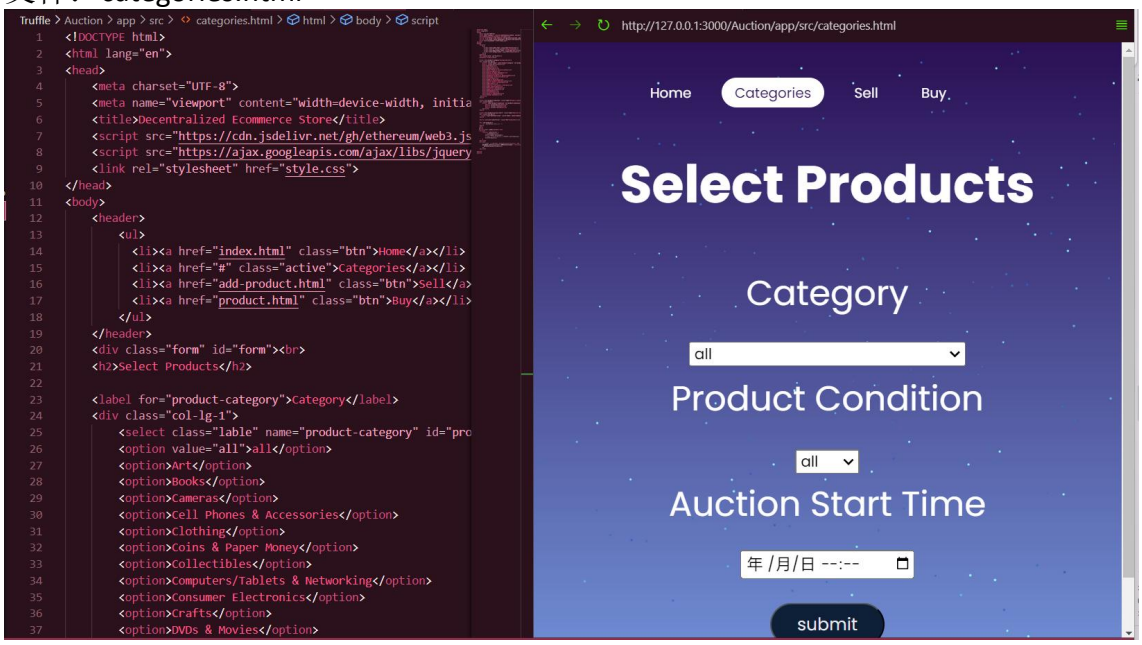
2.用户添加商品到区块链页面

文件：add-product.html 实现将产品添加到区块链的功能。



3. 用户可根据类别、商品状态和拍卖时间过滤筛选商品

文件：categories.html



功能实现：

```
function submitForm() {
      // 提交成功的提示框
      alert("submit success! ");
      // 调用 filterProducts() 函数来列出符合要求的商品
      filterProducts();
}
// 获取并显示所有商品
 async function getAndDisplayAllProducts() {
// 调用智能合约的方法获取所有商品
    try{
    const totalProducts = await instance.productIndex();
    const products = [];

    // 循环获取每个商品的详细信息
    for (let i = 1; i <= totalProducts; i++) {
        const product = await instance.getProduct(i);
        products.push({
        id: product[0].toNumber(),
        name: product[1],
        category: product[2],
        auctionTime: product[5].toNumber(),
        productCondition: product[6].toNumber(),
        });
    }
         // 显示所有商品列表
         displayProducts(products);
    }catch (error) {
          console.error("Error fetching products:", error);
     }
 }
```

```
// 根据用户选择的类别、商品状态和拍卖时间进行筛选
function filterProducts() {
   const selectedCategory = document.getElementById('product-category').value;
   const productCondition = document.getElementById('product-condition').value;
   const auctionTime = new
Date(document.getElementById('product-auction-start').value).getTime() / 1000;
// 调用智能合约的方法获取所有商品
   EcommerceStore.deployed().then(async function(instance) {
        const totalProducts = await instance.productIndex();
        const products = [];
        // 循环获取每个商品的详细信息
        for (let i = 1; i <= totalProducts; i++) {
                const product = await instance.getProduct(i);
                products.push({
                    id: product[0].toNumber(),
                    name: product[1],
                    category: product[2],
                    auctionTime: product[5].toNumber(),
                    productCondition: product[6].toNumber(),
                });
        }
// 根据选定的类别、商品状态和拍卖时间进行商品过滤
    const filteredProducts = products.filter(product => {
 return (selectedCategory === 'all' || product.category === selectedCategory) &&
(productCondition === 'all' || product.productCondition.toString() ===
productCondition) &&
(auctionTime === '' || product.auctionTime >= auctionTime);
            });
// 显示过滤后的商品列表
            displayProducts(filteredProducts);
        });
  }

// 显示商品列表
function displayProducts(products) {
  const productsDiv = document.getElementById('products');
  productsDiv.innerHTML = ''; // 清空原有内容

  products.forEach(product => {
      const productElement = document.createElement('div');
      productElement.innerHTML = `
          <p>Product ID: ${product.id}</p>
          <p>Product Name: ${product.name}</p>
          <p>Category: ${product.category}</p>
          <p>Auction Start Time: ${new Date(product.auctionTime *
1000).toLocaleString()}</p>
          <hr>
      `;
      productsDiv.appendChild(productElement);
  });
}
```

3.3.3 商品 JS

当用户点击“Add Product To Store”时会触发该事件，然后调用相关函数。
保存产品的步骤：

1. 调用函数将图片上传到 IPFS。
2. 一旦图片上传完毕，继续上传产品介绍。
3. 最后，将所上传资源的哈希截图，调用合约函数将所有细节保存到区块链。

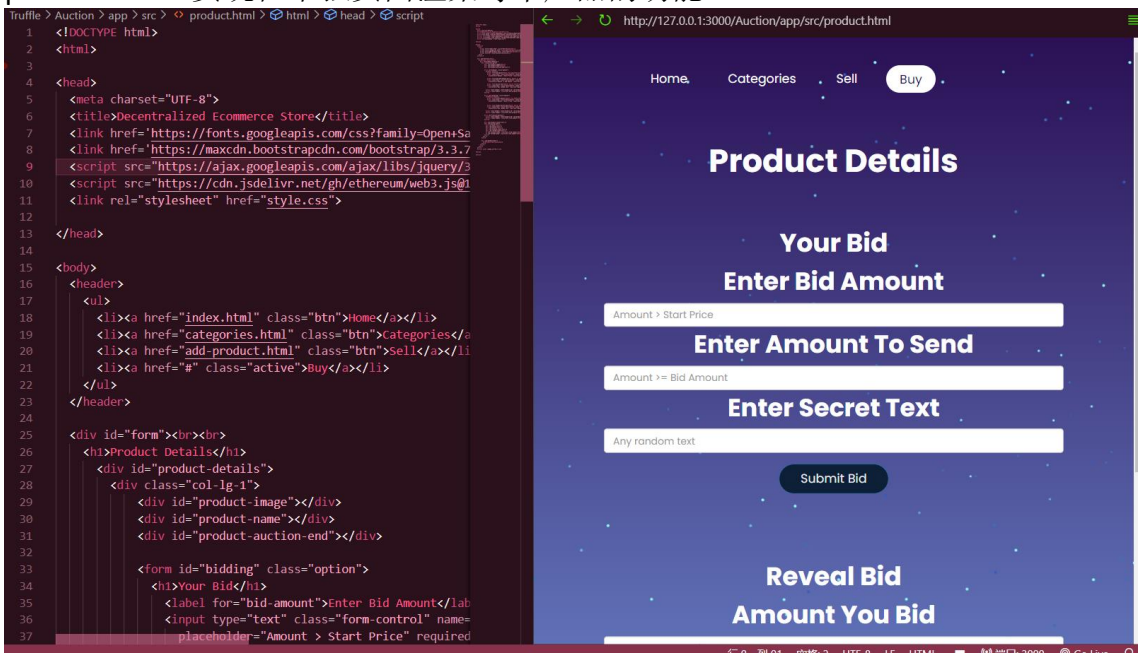
```
//保存商品
   $("add-item-to-store").submit(function(event) {
     const req = $("#add-item-to-store").serailize();
     console.log("req:", req);
     let params = JSON.parse('{"' + req.replace(/"/g, '\\"').replace(/&/g,
'","').replace(/=/g,'":"') + '"}');
     console.log("params:", params);
     let decodedParams = {};
     Object.keys(params).forEach(key => {
       decodedParams[key] = decodeURIComponent(decodeURI(params[key]));
     });
     saveProduct(reader, decodedParams);
     event.preventDefault();
   });

   if ($("#product-details").length > 0){
     let productId = new URLSearchParams(window.location.search).get("id");
     renderProductDetails(productId);
   }
    ...
//保存商品相关函数
function saveProduct(reader, decodedParams) {
   let imageId, descId;
   saveImageOnIpfs(reader).then(function(id) {
     imageId = id;
     saveTextBlobOnIpfs(decodedParams["product-description"]).then(function(i
d) {
       descId = id;
        saveProductToBlockchain(decodedParams, imageId, descId);
     })
  })
 }

 function saveProductToBlockchain(params, imageId, descId) {
   console.log(params);
   let auctionStartTime = Date.parse(params["product-auction-start"]) / 1000;
   let auctionEndTime = auctionStartTime +
parseInt(params["product-auction-end"]) * 24 * 60 * 60

   EcommerceStore.deployed().then(function(i) {
     i.addProductToStore(params["product-name"], params["product-category"],
imageId, descId, auctionStartTime,
    auctionEndTime, web3.utils.toWei(params["product-price"], 'ether'),
parseInt(params["product-condition"]), {from: getCurrentAccount(), gas:
440000}).then(function(f) {
    console.log(f);
    $("#msg").show();
    $("#msg").html("Your product was successfully added to your store!");
```

```
    })
  });
}
```

3.4 Web 拍卖

3.4.1 拍卖 HTML

product.html 实现在单独页面渲染每个产品的功能。



3.4.2 拍卖 JS

1.if($("#product-details").length > 0) 用于检查我们是否在产品细节的页面，如果在，调用 renderProductDetails 函数渲染产品细节。

```
//判断商品详情是否存在
 if($("#product-details").length > 0) {
   let productId = new URLSearchParams(window.location.search).get('id');
   renderProductDetails(productId);
  }
```

2.当访问 product.html 页面时，将一个请求参数 id=productId 包含在了 url 里面，可以用这个参数从区块链获取产品。

3.调用合约的 getProduct 获取产品细节，现在已经有了所存储的产品图片和描述信息的 IPFS 哈希。只需要用哈希即可渲染图片。但是对于描述信息，并不是使用一个指向描述信息的链接，而是使用 IPFS cat 命令来输出存储的描述文件的内容，然后显示在 HTML。

```
function renderProductDetails(productId) {
  EcommerceStore.deployed().then(function(i) {
    i.getProduct.call(productId).then(function(product) {
      let desc = '';
      ipfs.cat(product[4]).then(files => {
```

```
        desc = files.toString('utf8');
        $("#product-desc").append("<div>" + desc + "</div>");
      });
      $("#product-image").append("<img src='localhost:9001/ipfs/" + product[3]
+ "' width='250px' />");
      $("#product-name").html(product[1]);
      $("#product-id").val(product[0]);
      $("#product-price").html(displayPrice(product[7]));
      $("#product-auction-end").html(displayEndTime(product[6]));
      $("#bidding, #revealing, #finalize-auction, #escrow-info").hide();
      let currentTime = rrentTime();
     ...
}
```

4. 此外还定义了几个帮助函数，用于帮助显示更简洁。

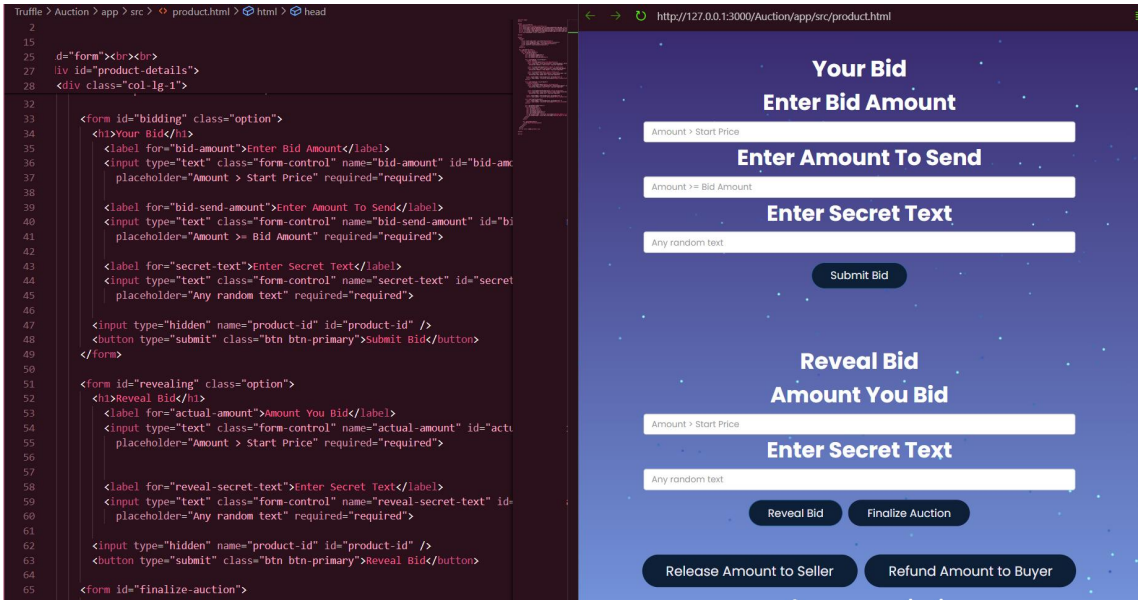
```
function displayPrice(amount)
function displayEndTime(endTime)
function getCurrentTime()
```

3. 4. 3 出价和揭示出价 HTML

除了显示产品细节，也要实现出价和揭示出价的功能创建了两个表单，一个用于出价，另一个用于揭示出价。

Bid Form: 出价单有三个参数，分别输入出价数量，secret 字符串和要发送的数量。

Reveal Form: 为了揭示出价，我们需要用户输入出价数量和 secret 字符串。



3. 4. 3 出价和揭示出价 JS

```
// 竞价
  $("#bidding").submit(function(event) {
    $("#msg").hide();
    let amount = $("#bid-amount").val();
    let sendAmount = $("#bid-send-amount").val();
```

```
    let secretText = $("#secret-text").val();
    let productId = $("#product-id").val();
    sealedBid = web3.utils.keccak256(productId + amount.toString() +secretText);

    EcommerceStore.methods.bid(parseInt(productId),
web3.utils.toWei(sendAmount.toString(), 'ether'), sealedBid)
      .send({from: getCurrentAccount(), gas: 440000})
      .then(function(res) {
        $("#msg").html("Your bid was successfully sent!");
        $("#msg").show();
        $("#bidding").hide(); // 隐藏 bid 表单
        $("#revealing").show(); // 显示 reveal 表单
      })
      .catch(err => {
        console.error(err);
      });
    event.preventDefault();
  });

  // 揭示报价
  $("#revealing").submit(function(event) {
    $("#msg").hide();
    let amount = $("#actual-amount").val();
    let secretText = $("#reveal-secret-text").val();
    let productId = $("#product-id").val();

    EcommerceStore.methods.revealBid(parseInt(productId), sealedBid)
      .send({from: getCurrentAccount(), gas: 440000})
      .then(function(res) {
        $("#msg").html("Your bid was successfully revealed!");
        $("#msg").show();
        $("#revealing").hide(); // 隐藏 reveal 表单
        $("#finalize-auction").show(); // 显示 finalize 表单
      })
      .catch(err => {
        console.error(err);
      });
    event.preventDefault();
  });
```

3.5 托管服务

一旦拍卖结束，所有出价方揭示各自出价，赢家诞生。接下来卖方必须将货物发送给买方，并且买方要付钱给卖方。这个时候可能出现几个问题：

1. 怎样保证卖方会如约交付货物？卖方可以轻松地拿着钱消失。
2. 如果卖方确实交付了货物，但是买方不承认收到货物怎么办？
3. 如果货物有损，应该返还给买方资金，但是卖方拒绝怎么办？

为了解决所有这些问题，我们创建一个 Multisig Escrow（多重签名托管）合约，里面存放了买方赢得拍卖的数量，买方，卖方和一个任意的第三方是参与者。托管的资金只能被释放给卖方，或是返回给买方，并且至少需要三个参与者中的两个同意才能执行。

3.5.1 托管合约

1. 创建一个叫做 Escrow 的合约，参数为买方，卖方，任意第三者和产品 id。买方，卖方和第三方实际上都是以太坊地址。

```
contract Escrow {
    uint public productId;
    address public seller;//卖家
    address public buyer;//买家
    address public arbiter;//仲裁者
    uint public amount;//竞拍金额
```

2. 必须跟踪所有的参与者，谁同意释放资金给卖家，谁同意返回资金给买家。所以，分别创建一个地址到布尔型的 mapping ，叫做 releaseAmount，另一个为 refundAmount 的 mapping。

```
    mapping(address => bool) releaseAmount;//释放资金（给卖家）
    mapping(address => bool) refundAmount;//退款资金（给买家）
```

3. 声明两个变量 releaseCount 和 refundCount。如果 releaseCount 达到 2，我们就释放资金。类似地，如果 refundCount 的值达到 2，我们将资金返回给买家。

```
 uint public releaseCount;//多少人同意
 uint public refundCount;
```

4. 创建一个构造函数，参数为所有的参与者和产品 id，并将它们赋值给我们已经声明的变量。将购买函数标记为 payable，当它初始化时，你可以向合约发送资金。托管合约创建后，赢家出价的资金就会发送给它。

```
constructor(uint _productId, address _seller, address _buyer, address _arbiter,
uint _amount) public payable{
      productId = _productId;
      seller = _seller;
      buyer = _buyer;
      arbiter = _arbiter;
      amount = msg.value;
      fundsDistribursed = false;
      emit CreateEscrow(_productId, _seller, _buyer, _arbiter, amount);
   }
```

5. 实现一个释放资金给卖方的函数。无论是谁调用该函数，它都应该更新 releaseAmount mapping，同时将 release count 加 1。如果 release count 计数达到 2，就用 transfer 方法将合约里托管的资金发送给卖方。

```
function releaseAmountToSeller(address caller) public {
       require(!fundsDisbursed);//判断是否已经释放资金
       require(caller == buyer || caller == seller || caller == arbiter);
       if ( !releaseAmount[caller] ) {//判断是否已经同意释放资金,防止多次同意
           releaseAmount[caller] = true;
           releaseCount += 1;
           emit UnlockAmount(productId, "release", caller);
       }
       if (releaseCount == 2) {//判断同意人数达到 2/3
           payable(seller).transfer(amount);//转给卖家
           fundsDisbursed = true;//修改状态
           emit DisburseAmount(productId, amount, seller);
```

```
        }
    }
```

6. 实现将资金撤回给买方的函数，它会更新 refundAmount mapping 并将 refundCount 加 1。如果 refundCount 达到 2，将资金转移给买方。

```
function refundAmountToBuyer(address caller) public {
        require(!fundsDisbursed);
        require(caller == buyer || caller == seller || caller == arbiter);
        if (!refundAmount[caller]) {
            refundAmount[caller] = true;
            refundCount += 1;
            emit UnlockAmount(productId, "refund", caller);
        }
        if (refundCount == 2) {
            payable(buyer).transfer(amount);
            fundsDisbursed = true;
            emit DisburseAmount(productId, amount, buyer);
        }
    }
```

7. 一旦合约释放完资金，应该禁用所有的函数调用。所以，声明一个叫做 fundsDisbursed 的字段，当资金释放给卖方或是返还给买方时，将其设置为 true。

```
  bool public fundsDisbursed;//是否释放资金
```

3. 5. 2 宣布赢家

一旦拍卖揭示阶段结束，会关闭拍卖并宣布赢家。这里是拍卖结束时的操作：

1. 拍卖由仲裁人结束。当结束时，创建有仲裁人，买方和卖方的托管合约，并将资金转移到合约。
2. 只向买方收取第二高的出价费用，所以需要将差额归还给赢家。

EcommerceStore.sol

```
//结束拍卖，宣布赢家
    function finalizeAuction(uint256 _productId) public {
        Product storage product = stores[productIdInStore[_productId]][ _productId];
        // 48 hours to reveal the bid
        require(block.timestamp > product.auctionEndTime);//结束拍卖时间大于当前时间
        require(product.status == ProductStatus.Open);//商品状态为可竞拍
        require(product.highestBidder != msg.sender);//当前出价人不是最高出价人
        require(productIdInStore[_productId] != msg.sender);//当前出价人不是商品创建者

        if (product.highestBidder == 0x0000000000000000000000000000000000000000) {//没有人出价
            product.status = ProductStatus.Unsold;
        } else {
            // Whoever finalizes the auction is the arbiter
            Escrow escrow = (new Escrow){value: product.secondHighestBid}(
                _productId,
                product.highestBidder,
                productIdInStore[_productId],
                msg.sender
            );
            productEscrow[_productId] = address(escrow);
            product.status = ProductStatus.Sold;
            // 赢家只需要付第二高竞价者的金额
```

```
            uint256 refund = product.highestBid - product.secondHighestBid;
            payable(product.highestBidder).transfer(refund);
        }
    }
```

向 index.js 添加一个结束拍卖的功能。并且检查产品状态，显示结束拍卖的按钮。

```
  $("#finalize-auction").submit(function(event) {
    $("#msg").hide();
    let productId = $("#product-id").val();

    EcommerceStore.methods.finalizeAuction(parseInt(productId))
      .send({from: getCurrentAccount(), gas: 440000})
      .then(function(res) {
        $("#msg").html("The auction was successfully finalized! Winner
declared!");
        $("#msg").show();
        location.reload(); // 刷新页面
      })
      .catch(err => {
        console.error(err);
        $("#msg").html("Your auction was not finalized. Please try again later.");
        $("#msg").show();
      });
    event.preventDefault(); // 提交页面
  });
```

更新拍卖结束后的操作,如果商品状态是已售出（"Sold"），那么就显示托管信息

```
function renderProductDetails(productId) {
    ...
    //判断商品状态
      //拍卖已结束
      if(parseInt(product[8]) == 1)//status 1 代表已卖出 sold
        EcommerceStore.deployed().then(function(i) {
          i.highestBidderInfo(productId).then(function(info) {
            $("#product-status").html("Auction has ended. Product sold to " + info[0] +
" for " + displayPrice(info[2]) +
            "The money is in the escrow. Two of the three participants (Buyer, Seller and
Arbiter) have to " +
            "either release the funds to seller or refund the money to the buyer");
          });
          i.escrowInfo(productId).then(function(info) {
            $("#seller").html('seller: ' + info[0]);
            $("#buyer").html('buyer: ' + info[1]);
            $("#arbiter").html('arbiter: ' + info[2]);
            if(info[3] == true){
              $("#release-count").html("Amount from the escrow has been released to
seller.");
            }else{
              $("#release-count").html(info[4] + "of 3 participants have agreed to
release the funds to seller.");
              $("#refund-count").html(info[5] + "of 3 participants have agreed to refund
the funds to buyer.");
            }
          });
        });
      else if(parseInt(product[8]) == 2)//status 2 代表未卖出 unsold
        $("#product-status").html("Product not sold");
        //拍卖未结束 status 0 代表拍卖 open
```

```
      if (currentTime < product[6]) {//竞拍结束前
        $("#bidding").show();
      } else if (currentTime - 600 < product[6]) {//竞拍结束 10min 内揭示报价
        $("#revealing").show();
      }else{//竞拍结束
        $("#finalize-auction").show();
      }
    });
  });
}
```

3. 5. 2 释放资金

EcommerceStore.sol

```
// 释放给卖家
    function releaseAmountToSeller(uint256 _productId) public {
        Escrow(productEscrow[_productId]).releaseAmountToSeller(msg.sender);
    }

    // 退回给买家
    function refundAmountToBuyer(uint256 _productId) public {
        Escrow(productEscrow[_productId]).refundAmountToBuyer(msg.sender);
    }
```

index.js

```
// 释放资金
  $("#release-funds").click(() => {
    let productId = new URLSearchParams(window.location.search).get("id");
    EcommerceStore.methods.releaseAmountToSeller(productId)
      .send({from: getCurrentAccount(), gas: 440000})
      .then(function(res) {
        location.reload();
      })
      .catch(err => {
        console.error(err);
      });
  });

  // 退款
  $("#refund-funds").click(() => {
    let productId = new URLSearchParams(window.location.search).get("id");
    EcommerceStore.methods.refundAmountToBuyer(productId)
      .send({from: getCurrentAccount(), gas: 440000})
      .then(function(res) {
        location.reload();
      })
      .catch(err => {
        console.error(err);
      });
});
```

至此，整个拍卖流程结束。

第四章 测试

4.1 应用环境的构建

4.1.1 需要的硬件环境

Windows 11

4.1.2 需要的软件环境

Truffle v5.11.5 (core: 5.11.5)
Ganache v7.9.1
Solidity v0.5.16 (solc-js)
Node v18.20.2
Web3.js v1.10.0

4.2 测试及界面显示

1. 启动 Ganache，编译部署合约到区块链

```
PS D:\Truffle\Auction> truffle migrate

Compiling your contracts...
===========================
> Compiling .\contracts\EcommerceStore.sol
> Compiling .\contracts\Escrow.sol
> Compiling .\contracts\Migrations.sol
> Artifacts written to D:\Truffle\Auction\build\contracts
> Compiled successfully using:
   - solc: 0.8.17+commit.8df45f5f.Emscripten.clang
Network up to date.
PS D:\Truffle\Auction> truffle console
```

2. 进入控制台，测试合约

获得时间戳

```
truffle(development)> current_time = Math.round(new Date() / 1000);
1713957449
```

查询产品（之前的测试已经添加）

```
truffle(development)> EcommerceStore.deployed().then(function(i) {i.getProduct(3).then(function(f) {console.log(f)})});
undefined
truffle(development)> Result {
  '0': BN {
    negative: 0,
    words: [ 3, <1 empty item> ],
    length: 1,
    red: null
  },
  '1': 'iphone 12',
  '2': 'Cell Phones & Accessories',
  '3': 'Qmf3aNDtmWwJfyLLX13WHa7wDjb46NWo1utMFAfjrjxexd',
  '4': 'QmX7VfaTwKRQEGknzDyMwhJJpRWCzd7bDNKokd7tsc8tRU',
  '5': BN {
    negative: 0,
    words: [ 38629148, 25, <1 empty item> ],
    length: 2,
    red: null
```

启动 IPFS

卖家添加产品

```
truffle(development)> EcommerceStore.deployed().then(function(i) {i.addProductToStore('iphone 5', 'Cell Phones & Accessories', 'image
link', 'desclink', current_time, current_time + 200, web3.utils.toWei('2','ether'), 0).then(function(f) {console.log(f)})});
undefined
truffle(development)> {
  tx: '0xbbbaff6034be1ed45514c839c94f889fb7f624e9891f0462c025b623002154ad',
  receipt: {
    transactionHash: '0xbbbaff6034be1ed45514c839c94f889fb7f624e9891f0462c025b623002154ad',
    transactionIndex: 0,
    blockNumber: 52,
    blockHash: '0x202c3a4baea356a5f9aca32e6fd534a9953dcea5cf985f1b69c7e4bc85228c00',
    from: '0x82051bcbb19afdc6a3121d4639925c71ba33ea64',
    to: '0x6c82947a8b9f2cc4540ab87242a320fd02487eec',
    cumulativeGasUsed: 287771,
    gasUsed: 287771,
    contractAddress: null,
    logs: [ [Object] ],
    logsBloom: '0x00000000000000000000000000000000000000000000000000000000000000400000000000000000000010000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000400000000000000000000000000000000000000000000000000000000000000000000000000000000000800000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000',
    status: true,
    effectiveGasPrice: 2502513924,
    type: '0x2',
    rawLogs: [ [Object] ]
```

查询刚刚添加的产品

```
truffle(development)> EcommerceStore.deployed().then(function(i) {i.getProduct(7).then(function(f) {console.log(f)})})
undefined
truffle(development)> Result {
  '0': BN {
    negative: 0,
    words: [ 7, <1 empty item> ],
    length: 1,
    red: null
  },
  '1': 'iphone 5',
  '2': 'Cell Phones & Accessories',
  '3': 'imagelink',
  '4': 'desclink',
  '5': BN {
```

用产品编号+出价+secret Text 做 hash 得到拍卖时要用到的 sealedBid（这里填真实

竞价为 3ETH）

```
truffle(development)> sealedBid = web3.utils.keccak256(7 + web3.utils.toWei('3','ether') + 'mysecret')
'0xc398a99dc28df5e43670a1fa522e9cb7efe4953b4a19f496cb3892c7ff959a9e'
```

买家查询余额

```
truffle(development)> web3.eth.getBalance("0xC3E988c974Bc57Af2C030F51E2B348acf5Df268a")
'100000000000000000000'
```

卖家查询余额

```
truffle(development)> web3.eth.getBalance("0x82051Bcbb19AfdC6A3121D4639925C71ba33EA64")
'47936865259227204950'
```

进行拍卖竞价（这里发送了 4ETH 用来迷惑对手）

```
truffle(development)> EcommerceStore.deployed().then(function(i) {i.bid(1, sealedBid, {from:"0xC3E988c974Bc57Af2C030F51E2B348acf5Df26
8a", value:(web3.utils.toWei('4','ether'))}).then(function(f) {console.log(f)})})
undefined
truffle(development)> {
  tx: '0x46801ba65c738786d3d1aec6165e54a456131788dd9709de62d53e7c71abb909',
  receipt: {
    transactionHash: '0x46801ba65c738786d3d1aec6165e54a456131788dd9709de62d53e7c71abb909',
    transactionIndex: 0,
    blockNumber: 59,
    blockHash: '0x500f5f2b829548d643e0db4512a91484f165bcd36b78dcccc9986a00859f5034',
    from: '0xc3e988c974bc57af2c030f51e2b348acf5df268a',
    to: '0x70255232d02c7a530e0b946629b5b6d66f496837',
    cumulativeGasUsed: 122967,
    gasUsed: 122967,
    contractAddress: null,
    logs: [],
    logsBloom: '0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000',
    status: true,
    effectiveGasPrice: 2501182825,
    type: '0x2',
    rawLogs: []
```

买家查询余额（发现已经给了发送的 4ETH 和一些 gas 费）

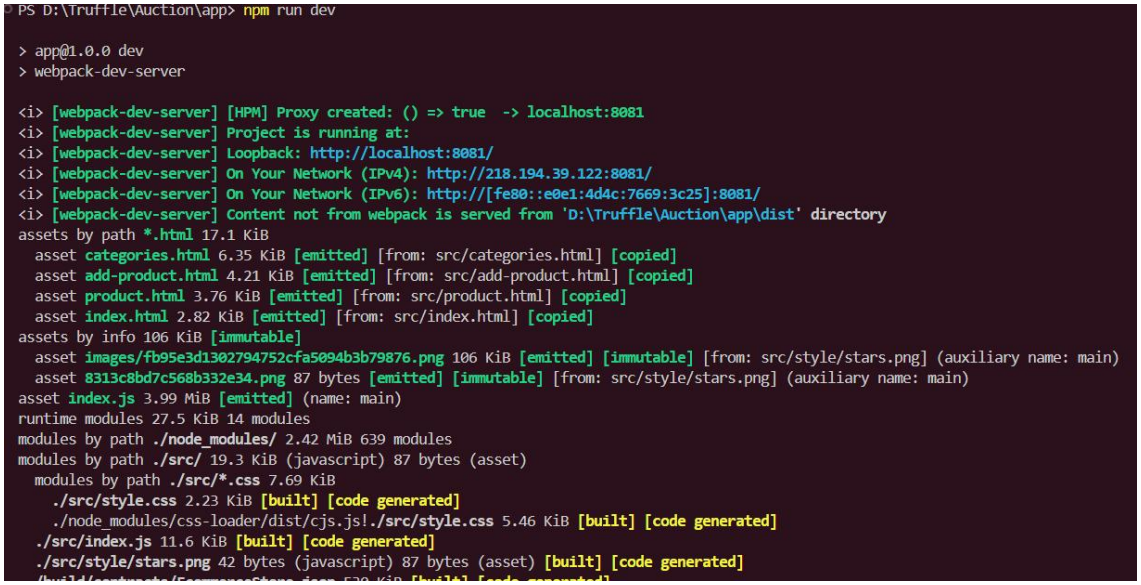
```
truffle(development)> web3.eth.getBalance("0xC3E988c974Bc57Af2C030F51E2B348acf5Df268a")
'95999692437051558225'
```

等待 600s 后拍卖结束，买家揭示报价->赢家获得产品，输家收回 ETH

ADDRESS	BALANCE	TX COUNT	INDEX
0×82051Bcbb19AfdC6A3121D4639925C71ba33EA64	47.91 ETH	71	0
0×C3E988c974Bc57Af2C030F51E2B348acf5Df268a	84.00 ETH	4	1
0×1f884f24ed2a7c8E10713C5cD4545AC848B7B4e9	85.00 ETH	4	2

3. 网页测试

进入 app 目录：npm run dev



成功启动后打开 http://localhost:8081

4.4 小结

在开发过程中，确保所有环境配置协调是一项技术含量较高的任务。我们的系统开发过程中遇到了诸多挑战，其中最显著的包括 npm install 安装的大部分包已经过期或废弃，以及网页调试的困难。这些问题增加了系统开发的复杂性，不仅需要我对技术进行深入的理解和应用，还需要我们在解决问题时保持耐心和创造性。尽管面临诸多困难，但通过不懈的努力，最终成功克服了这些挑战，并实现了一个具有创新性和实用性的商品拍卖系统，我也从中对区块链以及 web 前端有了更深刻的理解，技能得到极大提升。

参考文献

[1] 王李琰等. 一种基于区块链技术的在线拍卖方法及系统

[2] GitHub 项目：https://github.com/kpyaoqi/ebay-truffle-auction

[3] CSDN.智能拍卖：区块链上的经典智能合约

[4] CSDN.使用 MetaMask + Ganache 搭建本地私有网络并实现合约部署与互动

[5] IPFS Desktop 官方文档：https://docs.ipfs.tech/install/ipfs-desktop/