华东师范大学数据科学与工程学院实验报告

课程名称：区块链与分享型数据库　**年级：**2019 级　**上机实践成绩：**

指导老师：张召　**姓名：**郑逸潇　**学号：**10192100406

上机实践名称：基于以太坊的数字画作创作与拍卖平台

上机实践编号：Project　**上机实践日期：**2022-6-12

一、 实验目的

本次实验为期末项目，结合本学期所学知识以及个人想法实现一个基于以太坊/Fabric 的 DApp。本人实现的是一个基于以太坊的数字画作创作与拍卖平台。

二、 实验任务

（1） 实现数字画作创作（本次实验中即交易）智能合约。

（2） 实现数字画作拍卖智能合约。

（3） 根据合约完成前端页面。

三、 实验环境

Window11、VSCode、Ganache、Truffle、Metamask、Vue3、NaiveUI

涉及编程语言：HTML/JS(ES6)/CSS、Solidity

四、 实验过程

1、 实现数字画作创作（本实验中即交易）智能合约。

在原本的架构设想中，数字画作创作分为两个部分：预定约稿（发起交易）和提交稿件（交易完成），但是在实验过程中由于图片上传至以太坊平台图片转成 base64 编码后太大，会遇到 gas limit，导致图片无法上传，因此本功能简化为普通交易。

合约应当提供的接口有：上传约稿需求、订购约稿需求、获得所有约稿需求、获得未完成的越高需求。

Post（上传约稿需求）：

本接口接收稿件 id、作者、题材、价格、详情作为输入，将稿件内容存储在

智能合约中，这里的 post 接口也包含了更改稿件信息的功能，但是由于时间原因前端未实现。

```
function post(uint _id, string memory _author, string memory _title,
uint _price, string memory _info) public payable returns (uint) {
        // address payable _authorAddress =
address(uint160(msg.sender));
        address payable _authorAddress = payable(msg.sender);

        if (isExistMapping[_id]) {
            Comission memory comission = comissions[indexMapping[_id]];
            require(comission.authorAddress == _authorAddress);
            require(!isFinishedMapping[_id]);
            comission.author = _author;
            comission.title = _title;
            comission.price = _price;
            comission.info = _info;
            comissions[indexMapping[_id]] = comission;
        } else {
            comissions.push(Comission(
                _id, _authorAddress, _author, _title, _price , _info,
                false, address(0x0)
            ));
            indexMapping[_id] = posted_count;
            isExistMapping[_id] = true;
            isFinishedMapping[_id] = false;
            posted_count += 1;
        }

        return _id;
    }
```

Purchase（订购稿件需求）：

本接口接收稿件 id 作为参数，实现一个转账功能：将定稿者的稿费金额转入出稿者钱包中。

```
function purchase(uint _id) public payable {
        require(isExistMapping[_id]);
        require(!isFinishedMapping[_id]);
        address _purchaserAddress = msg.sender;
        Comission memory comission = comissions[indexMapping[_id]];
        comission.authorAddress.transfer((comission.price *
1000000000000));
```

```
        comission.finished = true;
        comission.purchaserAddress = _purchaserAddress;
        comissions[indexMapping[_id]] = comission;
        isFinishedMapping[_id] = true;
        finished_count += 1;
    }
```

get_all_comissions/get_selected_comissions（获取所有/选择性稿件需求）：

这两个接口为了获得所有稿件/已完成稿件/未完成稿件。

```
    function get_all_comissions() public view returns(Comission[]
    memory) {
        return comissions;
    }

    function get_selected_comissions(bool _finished) public view
returns(Comission[] memory) {
        uint idx = 0;
        if (_finished) {
            Comission[] memory selected_comissions = new
Comission[](finished_count);
            for (uint i = 0; i < posted_count; i++) {
                if (comissions[i].finished == true) {
                    selected_comissions[idx] = comissions[i];
                    idx += 1;
                }
            }
            return selected_comissions;
        } else {
            Comission[] memory selected_comissions = new
Comission[](posted_count - finished_count);
            for (uint i = 0; i < posted_count; i++) {
                if (comissions[i].finished == false) {
                    selected_comissions[idx] = comissions[i];
                    idx += 1;
                }
            }
            return selected_comissions;
        }
    }
```

2、 实现数字画作拍卖智能合约。

由于 Solidity 语言的限制，无法实现计时器功能，因此本拍卖智能合约会有一个“拍卖师”存在，相当于一个管理员。为了使得每次竞拍结束后会自动开启下一件物品拍卖(除非合约中无剩余拍卖物品)，每个竞品之间截止时间不得少于十分钟，当前竞标者不是您时会将上一次您竞标的金额回退，由于竞品拍卖结束需要由“拍卖师”发送指令，故可能会有延迟，合约提供了上传拍卖品、竞拍、pass 拍卖品、获得当前拍卖品、获得候选拍卖品、或者最后一个拍卖品等接口。

Post（上传拍卖品）：

本接口接收拍卖品 id，名字，信息和结束时间戳(会和当前队列最后一个做比较)作为输入，将拍卖品加入拍卖队列中。

```
    function post(uint _id, string memory _name, string memory _info,
uint _endTimestamp) public payable {
        require(!isExistMapping[_id]);
        if (posted_count > 0) {
            require(_endTimestamp > (lots[posted_count - 1].endTimestamp
+ INTERVAL));
        }
        string memory _token = '';
        lots.push(Lot(_id, auctioneer, _endTimestamp,
payable(msg.sender), _name, _token, _info, payable(address(0x0)), 0,
false));
        posted_count += 1;
    }
```

Bid（竞拍）：

本接口接收金额作为参数（事实上可以不用，直接调用 msg.value 也行），内部逻辑为判断价格是否合法，然后将上一个竞标者的钱退回，再将当前竞标者修改为当前拍卖品的竞标者。

```
    function bid(uint _price) public payable {
        require(!lots[current_idx].finished);
        require(_price >= (lots[current_idx].highestPrice +
LEASTMARKUP));
        // 将上一个竞标的钱退还给上一个竞标者
        lots[current_idx].bidder.transfer((lots[current_idx].highestPric
e * 1000000000000));
        // 更改新的竞标人和竞标价
        lots[current_idx].bidder = payable(msg.sender);
```

```
        lots[current_idx].highestPrice = _price;
    }
```

Pass（过拍卖品）：

由于这是一个有“拍卖师”的拍卖场，因此每个卖品都需要经过“拍卖师”的通过，才可以进行下一轮拍卖，本接口是将拍卖品当前竞标金额转给出拍卖品的用户。

```
    function pass() public payable {
        require(msg.sender == auctioneer);
        require(!lots[current_idx].finished);
        lots[current_idx].finished = true;
        // 此处可以加入拍卖行的手续费给 auctioneer
        lots[current_idx].seller.transfer((lots[current_idx].highestPric
e * 1000000000000));
        current_idx += 1;
    }
```

剩下三个接口和第一个合约的差不多，都是获取合约信息，public view，不再赘述。

3、 根据合约完成前端页面

本实验的难点在于前端与合约的相结合，之前的实验指导中并没有很详细的讲解如何结合前端与智能合约，因此这部分只能自己摸索。

问题一：Vue 框架与 truffle 框架如何交互问题：Vue 属于前后端分离式框架，而 truffle 官方和实验指导中都是一种不分离的结构，导致使用 Vue 的割裂感很强，在尝试了一些网上所谓的 Vue+truffle 框架后，总是存在一些问题。因此本项目基本是从源头构建了 dServer 和 dClient 结构，dServer 即 truffle 框架及其内部所有文件，dClient 即 Vue 框架及前端所有交互文件，之间的交互通过 @truffle/contract 完成（此库与实验指导中的 truffle-contract 并不相同，属于是较新的库，能更好的适配 Vue3 和一些 TS 语法），由于我们已经确定使用了 MetaMask，所以不需要使用 Web3.js 了，具体的使用方法如下：（这里将每个 vue 视图与每个合约对应部署）

```
// 初始化合约以及连接MetaMask
const contract = require('@truffle/contract')
const artifact = require('../assets/contracts/ArtComission.json')
const ArtComissionContract = contract(artifact);
ArtComissionContract.setProvider(window.ethereum)
```

问题二：使用了@truffle/contract 后调用问题：使用@truffle/contract 后，由于自身对于 js 一些语法不是很熟悉，导致调用失败，后面参考@truffle/contract 后，在实验指导的调用方法上做了一定的改变，使用了 await 和 async 来获得 account。实验指导中的获取方法被浏览器警告会在不久的将来被移除）

具体方法如下：（调用 bid 函数实例）

```
const purchase = () => {
    AuctionContract.deployed().then(async (instance) => {
        const accounts = await ethereum.request({ method: 'eth_requestAccounts' })
        const account = await accounts[0]
        return instance.bid(bid_price.value, { from: account, value: (bid_price.value * 1e12) })
    }).then((response) => {
        loadPostedData()
    }).catch((err) => {
        alert('error', err.message)
        console.log(err.message);
    });
}
```

由于这并不是前端课程，因此前端部分这里就不详细说明了，前端主要分为三个 Vue 视图：Index.vue/ArtComission.vue/Auction.vue。Index.vue 中就是 PPT 中的内容，介绍本项目；ArtComission.vue 中就是和 ArtComission.sol 的对接部分，各个组件采用了 NaiveUI；Auction.vue 中则是和 Auction.sol 的对接部分，这部分由于要实现前端自切换，倒计时，自动调用，实时对输入的判断（比如提交金额不能低于一些阈值，上传拍卖品的截止时间不能早于上一个，若是不符合就直接会把按钮 disabled 掉）等，实现较为复杂一点。

具体实现参考 dclient 中的文件。

4、实验运行方法

下载好项目后，1.首先启动 Ganache quick start；2.然后在 dserver 文件夹中使用 truffle migrate 对合约进行部署；3.然后将 dserver 中的 build/contract 文件夹中

的各个 json 文件复制到 dclient/src/assets/contract 中（重点！）；4.完成复制后启动 npm，在 dclient 中键入命令 npm run serve；5.最后打开浏览器，连接 MetaMask（这里随便点击产生什么交易就会触发将账户连接到 MetaMask，也可以手动导入），要注意不是公链，是我们 Ganache 创建的测试链，第一次需要自己手动设置，和实验指导的方法相同；6.都完成后可以尝试网页上的功能了！

五、 实验总结

本次实验融会贯通了以太坊和 Vue 的结合用法，实现了一个 DApp，在过程中也遇到了很多 bug，一一解决耗费了不少精力，整体实验代码量大部分还是集中在了前端，因为 solidity 语言特性导致很多事情无法在合约部分完成，总体来说实验前端代码 600+（实际没有那么多，挺多重复内容），合约部分 180 左右，由于时间关系，实现的接口也不是特别多。

实验还有很多不足：

本项目当前无法将图片格式文件上传至链上，仅做了 Demo。（网上看到两种方法，一个是文件转 base64 编码存储，可能会有 gas limit 问题，还有一个是 IPFS 方法，然后把文件哈希存入链中，由于时间问题没有细细了解）

本项目还有个重点 NFT.sol 未实现，内部应包含 NFT 生成、NFT 转移等操作。（当然 NFT.sol 也可以采用第三方 API 来实现）

网页布局还有部分 bug 未修改。

个人觉得这种项目如果有好点子事实上可以多人合作完成，可能完成度能高一点，在项目之初没考虑的那么仔细也是一个问题，本次实验如果是合作应该可以达到之前数据库大作业的效果：较为完整的网上书店系统 https://github.com/YeexiaoZheng/BookStore 。