2024年“中银杯”甘肃省职业院校技能大赛

高职学生组电子与信息大类

区块链技术应用赛项竞赛样题（3）

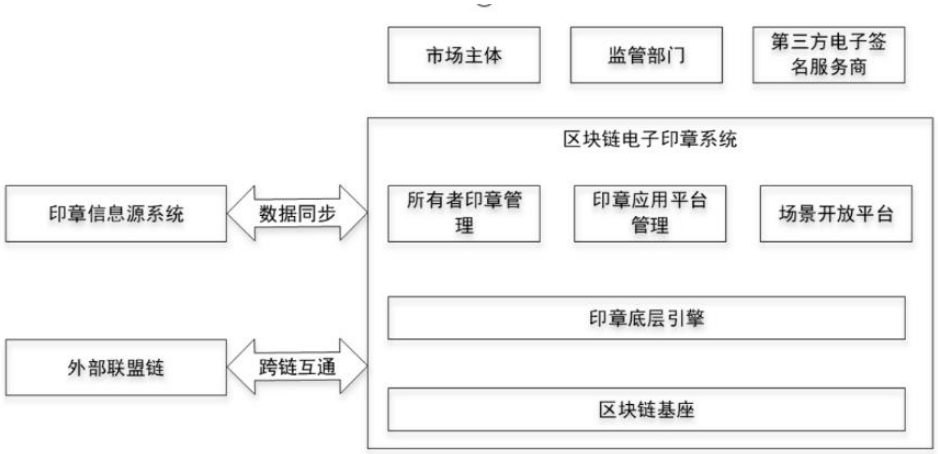
背景描述

电子签章可实现与纸质文件盖章操作相似的可视效果，以保障数据来源的真实性、数据完整性以及签名人行为的不可否认性。

传统的电子签章系统是基于中心化的，也就是数据是集中存储在中心数据库中，这就导致传统电子签章使用记录存在被篡改、伪造的风险。

而区块链电子签章系统在传统电子签章系统的基础上，可以借助于区块链技术，用不可篡改、不可抵赖的方式记录电子签章的整个流转过程，如领取、使用、查询等，从而解决传统电子签章系统存在的潜在风险。

某市政务部分拟开发一款区块链电子签章系统，该区块链电子签章系统包含为印章使用者、监管、平台运营方的管理功能，场景开放平台，印章底层引擎，和区块链基座。场景开放平台面向各类政务系统与第三方电子签名服务商提供电子签章相关的开放能力。场景开放平台对上提供相关的基础能力，对下连接区块链基座。



模块一：区块链产品方案设计及系统运维（35 分）

任务 1-1：区块链系统部署与运维（25 分）

围绕电子签章区块链平台部署与运维需求，进行项目相关系统、节点以及管理工具的部署工作。通过通过监控工具完成对网络、节点服务的监控。最终利用业务需求规范，完成系统日志、网络参数、节点服务等系统结构维护。

1. 登陆 Linux 服务器，安装并部署下图所示的单机、四机构、三群组、八节点的星形组网拓扑区块链系统；

2. 登陆 Linux 服务器，安装并部署区块链系统控制台，检查部署控制台是否正常运行；

3. 登录 Linux 服务器，部署区块链管理前置平台；

4. 登陆 Linux 服务器，使用终端生成新的节点，并且将该节点加入对应群组然后启动节点。

子任务 1-2-1： 登陆 Linux 服务器，安装并部署下图所示的单机、四机构、三群组、八节点的星形组网拓扑区块链系统，具体工作内容如下：

- 搭建部署多群组联盟链并启动所有节点；
- 通过命令验证区块链节点进程运行状况；
- 通过命令验证区块链连接状态和共识状态日志输出。

子任务 1-2-2：登陆 Linux 服务器，安装并部署区块链系统控制台，检查部署控制台是否正常运行，具体工作内容如下：

- 解压控制台安装包并拷贝配置文件到当前目录；
- 配置控制台证书；
- 启动控制台；
- 验证控制台是否正常运行。

子任务 1-2-3：登录 Linux 服务器，部署区块链管理前置平台，具体内容如下

- 解压平台操作对应安装包；
- 配置秘钥文件并检查前置平台启动情况；
- 查看日志以及查看中间件进程；

- 检查进程端口以及访问服务。

子任务 1-2-4：登陆 Linux 服务器，使用终端生成新的节点，并且将该节点加入对应群组然后启动节点，具体内容如下：

- 生成新节点，修改新节点配置；
- 启动新节点，并查看节点的 nodeid；
- 将新节点作为观察节点加入群组 1 当中，并检查是否加入成功。

任务 1-2：区块链系统测试（10 分）

设计对区块链系统的测试流程；结合实际业务需求，调用部署的智能合约中进行系统测试、性能测试等；根据业务需求，分析并且修复给定智能合约中的安全漏洞。利用模拟业务和测试工具来完成对区块链系统服务数据的测试。

1. 基于给定脚本完成区块链管理平台部署以及结果验证，最后将执行结果截图保存。

- 实现区块链管理平台部署；
- 实现管理平台启动情况验证；
- 验证身份认证进程启动情况验证和浏览器验证。

2. 智能合约安全漏洞测试。

有如下智能合约：

```
pragma solidity ^0.8.3;

contract EtherGame {
    uint public targetAmount = 7 ether;
    address public winner;

    function deposit() public payable {
        require(msg.value == 1 ether, "You can only send 1 Ether");

        uint balance = address(this).balance;
        require(balance <= targetAmount, "Game is over");

        if (balance == targetAmount) {
            winner = msg.sender;
        }
    }
```

```
    function claimReward() public {
        require(msg.sender == winner, "Not winner");

        (bool sent, ) = msg.sender.call{value: address(this).balance}("");
        require(sent, "Failed to send Ether");
    }
}

contract Attack {
    EtherGame etherGame;

    constructor(EtherGame _etherGame) {
        etherGame = EtherGame(_etherGame);
    }

    function attack() public payable {
        address payable addr = payable(address(etherGame));
        selfdestruct(addr);
    }
}
```

- 分析智能合约中存在问题，并说明危害；
- 创建新的智能合约，修复其中问题；
- 说明修复内容并测试其智能合约。

模块二：智能合约开发与测试（30 分）

选手完成本模块的任务后，将任务中设计结果、运行代码、运行结果等截图粘贴至客户端桌面【区块链技术应用赛\重命名为工位号\模块二提交结果.docx】中对应的任务序号下。

任务 2-1：智能合约开发（20 分）

使用 Solidity 语言完成智能合约开发、部署和调用，要求如下：

1.个人签章信息接口编码

（1）编写个人签章智能合约的实体接口，完成实体通用数据的初始化，实现签章和用户实体信息上链的功能；

```
struct SealInfo {
        //①姓名
        //②身份证
        //③印章数据
        //④创建印章时间戳
}
```

（2）编写签章信息上链的接口，实现 Seal 合约的构造函数；

```
//@dev 构造函数
// @param _name 姓名
// @param _cardId 身份证号码
// @param _data 个人印章数据
// @param _datetime 账户注册时间
    constructor(string _name,string _cardId,string _data,string _datetime) public {
        //①用户姓名
        //②用户身份 ID
        //③用户数据
        //④时间戳
      isUsed = true;
```

}

（3）基于给定的智能合约代码以及注释，完成 ElectronicSeal 合约判断多人签章文件编号是否存在的函数。

//@dev 判断多人签名文件编号是否存在

//@param _code 签章文件编号

//@return bool true:存在 false：不存在

function isExistMultiCode(string _code) view internal returns(bool) {

//①遍历所有多人签章文件编号并与传入签章文件编号参数进行比较

//②返回不存在

}

2.电子印章接口编码（8 分）

（1）基于给定的智能合约代码以及注释，完成 ElectronicSeal 合约获取多人签章信息函数（3 分）

//@dev 获取多人签章信息

//@param _code 签章文件编号

//@return (签章文件摘要、多人签章发起人账户地址、签章文件编号、签章人数、签章账户地址列表)

function getMultiSingInfo(string _code) public view onlyPromoterOrSinger returns(string,address,string,uint,address[] memory) {

//①获取多人签章合约对象实例

//②返回多人签章信息

}

（2）基于给定的智能合约代码以及注释，完成 ElectronicSeal 合约多人签章函数（5 分）

//@dev 多人签章

//@param _address 签章人账户地址

```
        //@param _hash 签章文件摘要
        //@param _datatime 签章时间戳
        function doMultiSign(address _address,string _hash,string
_datetime) public onlyPromoter {
            //①判断传入的签章人账户地址不能为空
            //②获取多人签章合约对象实例
            //③获取当前等待签章账户地址
            //④判断合约调用者是否为等待签章人账户地址
            //⑤进行签章
        }
```

3. 合约编译、部署和调用

（1）解决代码错误和警告，正确编译并部署合约，成功获取部署的合约地址和 abi;

（2）调用区块链电子签章智能合约的接口，完整验证业务流程。

任务 2-2：智能合约测试（10 分）

根据已完成的智能合约，针对开发功能开展相关合约测试工作，具体工作内容如下：

1. 生成测试文件，在 Remix 中的合约测试目录 tests 中生成 ElectionSeal_Test.sol 的测试文件

编译并部署合约，配置项目的 ElectionSeal_Test.sol 测试脚本文件。

2. 注释测试文件中自动生成的测试函数 checkSuccess()、checkSuccess2()、checkFailure()、checkSenderAndValue()相关代码,运行测试按钮

基于 remix 的测试项目，将 checkSuccess()、checkSuccess2()、checkFailure()、checkSenderAndValue()相关代码注释掉并进行相关测试;

3. 实例化

在测试文件中定义测试 ElectronicSeal 合约变量并在 beforeAll()函数中进行实例化。

4. 创建测试函数测试印章账户数量

基于remix 的测试项目,补全位于test文件夹中的ElectionSeal_Test.sol合约测试文件，添加测试用例，补全测试印章账户数量函数。

5. 创建测试函数测试印章账户签章

基于remix 的测试项目，补全位于test文件夹中ElectionSeal_Test.sol合约测试文件，添加测试用例，创建测试函数测试印章账户签章。

模块三：区块链应用系统开发（30 分）

任务 3-1：区块链电子签章前端功能开发（10 分）

1.请基于前端系统的开发模板，在注册页面 register.html 文件中添加对应的代码逻辑，实现对前端系统的注册功能，并测试功能完整性，示例页面如下;

题目的具体要求如下：

(1) 注册身份证号码的数据实现双向绑定

(2) 注册按钮的事件绑定

(3) 将用户输入的注册信息向后端系统发送注册请求

register.html

代码片段 1:

```
<div class="input-group" style="width:80%">
    <span class="input-group-addon glyphicon glyphicon-list-alt"
id="basic-addon1"></span>
    <input type="text" class="form-control1" placeholder="身份证号码
(必填)" autocomplete="off"选手填写部分>
</div><br />
```

代码片段 2:

```
<div class="modal-footer" style="margin-left: 150px;">
```

```
                <div class="form-group" style="width:50%">

                    <!-- TODO: 请补充注册按钮的事件绑定代码-->
                    <button type="button" class="btn btn-success
form-control"选手填写部分>确定</button>
                </div>

            </div>
```

代码片段 3:

```
methods: {
                submit() {
                    if (this.user.username == '') {
                        alert("用户名不能为空")
                        return
                    }
                    if (this.user.password == '') {
                        alert("密码不能为空")
                        return
                    }
                    if (this.user.chainAccount == '') {
                        alert("区块链账户地址不能为空")
                        return
                    }
                    if (this.user.name == '') {
                        alert("姓名不能为空")
                        return
                    }
                    if (this.user.cardId == '') {
                        alert("身份证号码不能为空")
                        return
```

```
            }
            if (this.user.phone == '') {
                alert("电话号码不能为空")
                return
            }
            选手填写部分
        }
    }
```

2.请基于前端系统的开发模板，在登录页面 login.html 文件中添加对应的代码逻辑，实现对前端系统的登录功能，并测试功能完整性,示例页面如下:

本题目的具体要求如下:

(1) 登录按钮的事件绑定

(2) 用户输入的登录用户名和密码通过双向绑定存放在 user 中

(3) 登录成功页面跳转到 main.html 页面

(4) 本地保存登录状态信息(token)

login.html

代码片段 1:

```
<div class="modal-footer" style="margin-left: 100px;">
            <div class="form-group" style="width:75%">
                <button type="button" class="btn btn-primary
form-control"选手填写部分>确定</button>
            </div>
        </div>
```

代码片段 2:

```
//创建 Vue 实例
    new Vue({
        el: '#app', //el 用于指定当前 Vue 实例为哪个容器服务，值通常为 css 选择器字符串
        data: { //data 中用于存储数据，数据供 el 所指定的容器去使用，值我们暂时先写成一个对象。
            user: {

                选手填写部分
            },
            tip: false,
            msg: ""
        },
        methods: {
            submit() {
                if (this.user.username == '') {
                    this.msg = "用户名不能为空"
                    this.tip = true
                    setTimeout(() => {
                        this.tip = false
                    }, 1000)
                    return
                }
                if (this.user.password == '') {
                    this.msg = "密码不能为空"
                    this.tip = true
                    setTimeout(() => {
                        this.tip = false
                    }, 1000)
```

```
                    return
                }

                axios.post("http://localhost/user/login",
this.user).then(response => {
                    if (response.data.resultCode == 200) {
                        this.msg = "登录成功"
                        this.tip = true
                        setTimeout(() => {

                            选手填写部分

                        }, 1000)

                        选手填写部分
                    } else {
                        this.msg = "登录失败"
                        this.tip = true
                        setTimeout(() => {
                            this.tip = false
                        }, 1000)
                    }
                })
            },
        }
})
```

3.请基于前端系统的开发模板，在个人印章页面 mysea1.html 文件中添加对应的页面代码逻辑，实现对前端系统的个人印章功能，并测试功能完整性，示例页面如下:

具体要求如下:

(1) 注销按钮的事件绑定

(2) 注销时清除本地保存的登录状态信息(token)

(3) 发送请求到后端系统携带本地保存的登录状态信息(token)

(4) 接收后端系统发送过来的印章图片信息

myseal.html

代码片段 1:

```
<div class="container-fluid">
            <div class="navbar-header">
                <a class="navbar-brand" href="#"><img alt="Brand"
src="../images/logo.png" style="height: 50px;"></a>
            </div>
            <img alt="Brand" src="../images/main.png" style="height:
40px; margin-top: 20px;">
            <ul class="nav navbar-nav" style="float: right;">
                <li><a href="../index.html"><button type="button"
class="btn btn-success"选手填写部分>注销</button></a></li>
            </ul>
        </div>
```

代码片段 2:

```
methods:{
            logout() {
                选手填写部分
            }
```

```
        },
```

代码片段 3:

```
// 请求发前拦截，header 中添加 token
            axios.interceptors.request.use(config => {
                config.headers.Authorization =选手填写部分
                return config
            })
```

代码片段 4:

```
axios.get("http://localhost/user/seal").then(response => {
                if (response.data.resultCode == 200) {
                    this.seal =选手填写部分
                }
            })
```

4.请基于前端系统的开发模板，请在文件签章页面 signature.html 文件中添加对应的页面代码逻辑，实现对前端系统的文件签章业务功能，并测试功能完整性，示例页面如下:

具体要求如下:

(1) 创建 canvas 画布实例

(2) 向后端系统发送签章请求 url

(3) 向后端系统获取个人印章请求 url

(4) 接收后端系统发送过来的印章图片信息

signature.html

代码片段 1:

```
signature() {
                if (this.status == 0) {
                    this.msg = "请上传文档"
                    this.tip = true
                    setTimeout(() => {
                        this.tip = false
                    }, 1000)
                    return
                }

                this.status = 2

                var canvas;
                canvas =选手填写部分

                canvas.width = $("#poster").width()
                canvas.height = $("#poster").height()
                var context = canvas.getContext("2d");
                context.rect(0, 0, canvas.width, canvas.height);
                context.fillStyle = "#fff";
                context.fill();
```

代码片段 2:

```
uploadSeal() {
                if (this.status != 2) {
                    this.msg = "文档未上传或未签章"
                    this.tip = true
                    setTimeout(() => {
                        this.tip = false
                    }, 1000)
```

```
            return
        }

        var posterImg = document.getElementById('poster');
        var imgBase64 = posterImg.getAttribute("src");

        axios.post("选手填写部分",
            {
                "filename": this.filename,
                "imgBase64": imgBase64,
            }
        ).then(response => {
            if (response.data.resultCode == 200) {
                this.msg = "签章文档上链成功"
                this.tip = true
                setTimeout(() => {
                    this.tip = false
                    window.location.href = "main.html";
                }, 1000)
            } else {
                this.msg = "签章文档上链失败"
                this.tip = true
                setTimeout(() => {
                    this.tip = false
                }, 1000)
            }
        })
    }
```

代码片段 3:

```
mounted() {
```

```
        // 请求发前拦截，header 中添加 token
        axios.interceptors.request.use(request => {
            request.headers.Authorization =
localStorage.getItem("Authorization")
            return request
        })

        //响应拦截器(认证授权)
        axios.interceptors.response.use(response => {
            if (response.data.resultCode == 401) {
                //跳转页面
                window.location.href = "login.html";
            }
            return response
        })

        //TODO:【5-1-4-3】请补充向服务器发送获取个人印章请求代码
        axios.get("选手填写部分").then(response => {
            if (response.data.resultCode == 200) {
                this.seal =选手填写部分


            }
        })
    }
```

5.请基于前端系统的开发模板，在文件验章页面 verity.html 文件中添加对应的页面代码逻辑，实现对前端系统的文件验章业务功能，并测试功能完整性，示例页面如下:

具体要求如下:

(1) 将验章结果数据显示在页面中

(2) 向后端系统发送验章请求并携带验章文件数据

verity.html

代码片段 1:

```
<div class="modal-dialog" role="document">
                        <div class="modal-content">
                            <div class="modal-header">
                                <button type="button" class="close"
data-dismiss="modal" aria-label="Close"><span

aria-hidden="true">&times;</span></button>
                                <h4 class="modal-title"
id="myModalLabel">签章信息</h4>
                            </div>
                            <div class="modal-body">
                                <p style="margin-left: 50px;">
                                    <label>姓名: </label>选手填写部
分

                                </p>
                                <p style="margin-left: 50px;">
                                    <label>身份证号码: </label>选手
填写部分
```

```
                                </p>
                                <p style="margin-left: 50px;">
                                    <label>区块链账户地址: </label>
选手填写部分
                                </p>
                                <p style="margin-left: 50px;">
                                    <label>文件名称: </label>选手填
写部分
                                </p>
                                <p style="margin-left: 50px;">
                                    <label>文件编码: </label>选手填
写部分
                                </p>
                                <p style="margin-left: 50px;">
                                    <label>签章时间: </label>选手填
写部分
                                </p>
                            </div>
                            <div class="modal-footer" style="width:
300px;">
                                <button type="button" class="btn
btn-success" data-dismiss="modal" @click="finish">确定</button>
                            </div>
                        </div>
                    </div>
```

代码片段 2:

```
async verify(event) { //同步发送 axios 请求(获取数据后弹出模态框)
                if (this.status == 0) {
                    this.msg = "请上传文档"
                    this.tip = true
```

```
                    event.stopPropagation() //阻止事件冒泡(防止弹出
模态框)
                    setTimeout(() => {
                        this.tip = false
                    }, 1000)
                    return
                }

                await axios.post("选手填写部分",
{"imgBase64":this.contract}).then(response => {
                    if (response.data.resultCode == 200) {
                        this.verifyInfo = response.data.data
                    }
                })
            },
```

任务 3-2：区块链应用系统后端开发（20 分）

1.请基于后端系统的开发模板，在 userService.java 文件中补充对应的代码逻辑，完成后端代码逻辑，实现对后端系统的注册功能，并测试功能完整性，题目具体要求如下:

(1) 根据用户注册信息中的用户名和账户地址从数据库中查询账户信息是否存在

(2) 将用户密码进行 md5 加密操作

(3) 将电子印章字节数组转为字符串

(4) 将用户注册信息写入数据库汇总

userService.java

代码片段 1:

```
public boolean register(RegisterBean registerBean) {
```

```
        //校验用户名或账户地址是否重复
        QueryWrapper<UserEntity> queryWrapper = new QueryWrapper<>();
        queryWrapper.eq("username", registerBean.getUsername())
                .or().eq("chain_account",registerBean.getChainAccou
nt());

        UserEntity userOne =选手填写部分

        if (userOne != null) {
            log.info("注册失败: 用户名或账户地址重复");
            return false;
        }

        //拷贝对象
        UserEntity userEntity = new UserEntity();
        BeanUtils.copyProperties(registerBean, userEntity);

        userEntity.setPassword(选手填写部分);//md5 加密密码

        //生成印章 (Base64)
        byte[] buffer = Seal.getSealImg(registerBean.getUsername(),
registerBean.getName());
        Encoder encode = Base64.getEncoder();
        String sealImg =选手填写部分

        userEntity.setSealImg(sealImg);

        //用户注册信息上链
        List funcParams = new ArrayList<>();
        funcParams.add(registerBean.getChainAccount());
```

```
        funcParams.add(registerBean.getName());
        funcParams.add(registerBean.getCardId());
        funcParams.add(String.valueOf(System.currentTimeMillis()));
        JSONObject res = HttpUtils.writeContract("addSealAccount",
funcParams);
        if (res == null) {
            log.info("注册失败: 用户注册信息上链失败");
            return false;
        }
        int result =选手填写部分

        return result > 0 ? true : false;
    }
```

2.请基于后端系统的开发模板，补充代码完成对应的后端代码逻辑，实现后端系统的登录功能，并测试功能完整性，具体要求如下:

(1) 登录成功向前端系统返回登录状态信息（token）

(2) 添加登录的用户名和密码两个数据库查询条件

(3) 根据登录的用户名和密码到数据库中进行查询操作

UserController.java

代码片段 1:

```
public AjaxResult<String> login(@RequestBody LoginBean loginBean) {
        log.info(loginBean.toString());
        if (userService.isLogin(loginBean)) {
            //创建
            String token =
TokenUtil.sign(loginBean.getUsername(),String.valueOf(System.curren
tTimeMillis()));
            AjaxResult<String> result = new AjaxResult<>(200, "login
```

```
ok");
            result.setData(选手填写部分);

            return result;
        } else {
            return new AjaxResult<String>(403,"login fail");
        }
    }
```

UserService.java

代码片段 1:

```
public boolean isLogin(LoginBean loginBean) {
        QueryWrapper<UserEntity> queryWrapper = new QueryWrapper<>();

loginBean.setPassword(SecureUtil.md5(loginBean.getPassword()));//md
5 加密密码

        //TODO:请补充 queryWrapper 添加用户名和密码两个数据库查询条
件代码
        queryWrapper.eq(选手填写部分)
                .eq(选手填写部分);

        //TODO:请补充数据库查询登录用户信息代码
        UserEntity userEntity =选手填写部分

        return userEntity != null ? true : false;
    }
```

3.请基于后端系统的开发模板，补充代码完成对应的后端代码逻辑，实现对后端系统的账户信息查看功能，并测试功能完整性，具体要求如下:

(1) 从前端系统发送的请求头中获取登录状态信息（token）

(2) 根据登录状态信息（token）获取登录用户名

(3) 根据用户名查询数据库

(4) 将查询结果数据封装到 RegisterBean 中

UserController.java

代码片段 1:

```
@ApiOperation(value = "用户信息",notes = "获取用户详细信息")
    @GetMapping("/user/info")
    public AjaxResult<RegisterBean> getUserInfo(HttpServletRequest
request) {
        String token =选手填写部分
        String username =选手填写部分
        RegisterBean registerBean = userService.getUser(username);
        AjaxResult<RegisterBean> result = new AjaxResult<>(200,
"ok");
        result.setData(registerBean);
        return result;
    }
```

UserService.java

代码片段 1:

```
public RegisterBean getUser(String username) {
        QueryWrapper<UserEntity> queryWrapper = new QueryWrapper<>();
        queryWrapper.eq("username", username);

        UserEntity userEntity =选手填写部分


        RegisterBean registerBean = new RegisterBean();
```

```
        选手填写部分
        return registerBean;
    }
```

4.请基于后端系统的开发模板，补充代码完成对应的后端代码逻辑，实现对后端系统的文件签章功能，并测试功能完整性，具体要求如下:

(1) 验证登录状态信息（token）

(2) 获取签章文件的UUID编码

(3) 获取验章文件摘要（hash）

(4) 将验章文件信息写入智能合约中

TokenInterceptor.java

代码片段1:

```
public boolean preHandle(HttpServletRequest request,
HttpServletResponse response, Object handler) throws Exception {

        if ("OPTIONS".equals(request.getMethod())) {
            response.setStatus(HttpServletResponse.SC_OK);
            return true;
        }

        response.setCharacterEncoding("utf-8");

        String token = request.getHeader("Authorization");
        log.info("token:"+token);
        if (token != null) {
            boolean result =选手填写部分

            if (result) {
                log.info("通过拦截器");
                return true;
```

```
            }
        }
        response.setCharacterEncoding("UTF-8");
        response.setContentType("application/json; charset=utf-8");
        PrintWriter out = null;
        try {
            JSONObject json = new JSONObject();
            json.put("message", "认证失败，未通过拦截器");
            json.put("resultCode", "401");
            response.getWriter().append(json.toJSONString());
            log.info("认证失败，未通过拦截器");
        } catch (Exception e) {
            e.printStackTrace();
            response.sendError(500);
            return false;
        }
        return false;
    }
```

SealService.java

代码片段 1:

```
public boolean signature(ContractBean contractBean, String
chainAccount) {
        //获取签名文档 uuid 编码

        String code =选手填写部分

        SealEntity sealEntity = new SealEntity();
        BeanUtils.copyProperties(contractBean, sealEntity);
        String[] str = contractBean.getFilename().split("\\.");
        sealEntity.setFilename(str[0]);
```

```
        sealEntity.setType(str[1]);
        sealEntity.setCode(code);

        //签章记录上链
        try {
            //获取印章文档Hash值
            MessageDigest messageDigest =选手填写部分

            byte[] bytes =
messageDigest.digest(contractBean.getImgBase64().getBytes("UTF-8"))
;
            String hash = Hex.toHexString(bytes);
            log.info("hash:" + hash);

            List funcParams = new ArrayList<>();
            funcParams.add(chainAccount);
            funcParams.add(hash);
            funcParams.add(code);

funcParams.add(String.valueOf(System.currentTimeMillis()));

            JSONObject res =选手填写部分

            if (res == null) {
                log.info("印章记录上链失败");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
        int result = sealMapper.insert(sealEntity);

        return result > 0 ? true : false;
    }
```

5.请基于后端系统的开发模板，补充代码完成对应的后端代码逻辑，实现对后端系统的文件验章功能，并测试功能完整性，具体要求如下:

(1) 调用验章方法进行验章

(2) 读取合约获取区块链账户信息

(3) 读取合约查询区块链验章结果

SealController.java

代码片段 1:

```
public AjaxResult<VerifyBean> verify(@RequestBody ImgBase64Bean
imgBase64Bean, HttpServletRequest request) {
        String imgBase64 = imgBase64Bean.getImgBase64();
        String token = request.getHeader("Authorization");
        String username = TokenUtil.getLoginName(token);
        RegisterBean registerBean = userService.getUser(username);

        VerifyBean verifyBean =选手填写部分

        if (verifyBean == null){
            return new AjaxResult<VerifyBean>(403,"fail");
        }
        AjaxResult<VerifyBean> result = new AjaxResult<>(200,"ok");
        result.setData(verifyBean);
        return result;
    }
```

SealService.java

代码片段 1:

```
public VerifyBean verify(String chainAccount, String imgBase64) {
        VerifyBean verifyBean = new VerifyBean();
        verifyBean.setChainAccount(chainAccount);

        //获取账户信息
        List params = new ArrayList();
        params.add(chainAccount);

        JSONArray accountInfo =选手填写部分
        if (accountInfo == null) {
            log.info("获取区块链账户信息失败");
            return null;
        }

        List<String> list =
JSONArray.parseArray(accountInfo.toJSONString(), String.class);
        String name = list.get(0);
        String cardId = list.get(1);
        verifyBean.setName(name);
        verifyBean.setCardId(cardId);

        try {
            //获取印章文档Hash值
            MessageDigest messageDigest =
MessageDigest.getInstance("SHA-256");
            byte[] bytes =
messageDigest.digest(imgBase64.getBytes("UTF-8"));
            String hash = Hex.toHexString(bytes);
            log.info("hash:"+hash);
```

```
//获取印章信息
List funcParams = new ArrayList();
funcParams.add(chainAccount);
funcParams.add(hash);

JSONArray sealInfo =选手填写部分
if (sealInfo == null) {
    log.info("获取区块链印章信息失败");
    return null;
}
```