2023年全国职业院校技能大赛

高职组

“区块链技术应用”

赛项赛卷（9卷）

任

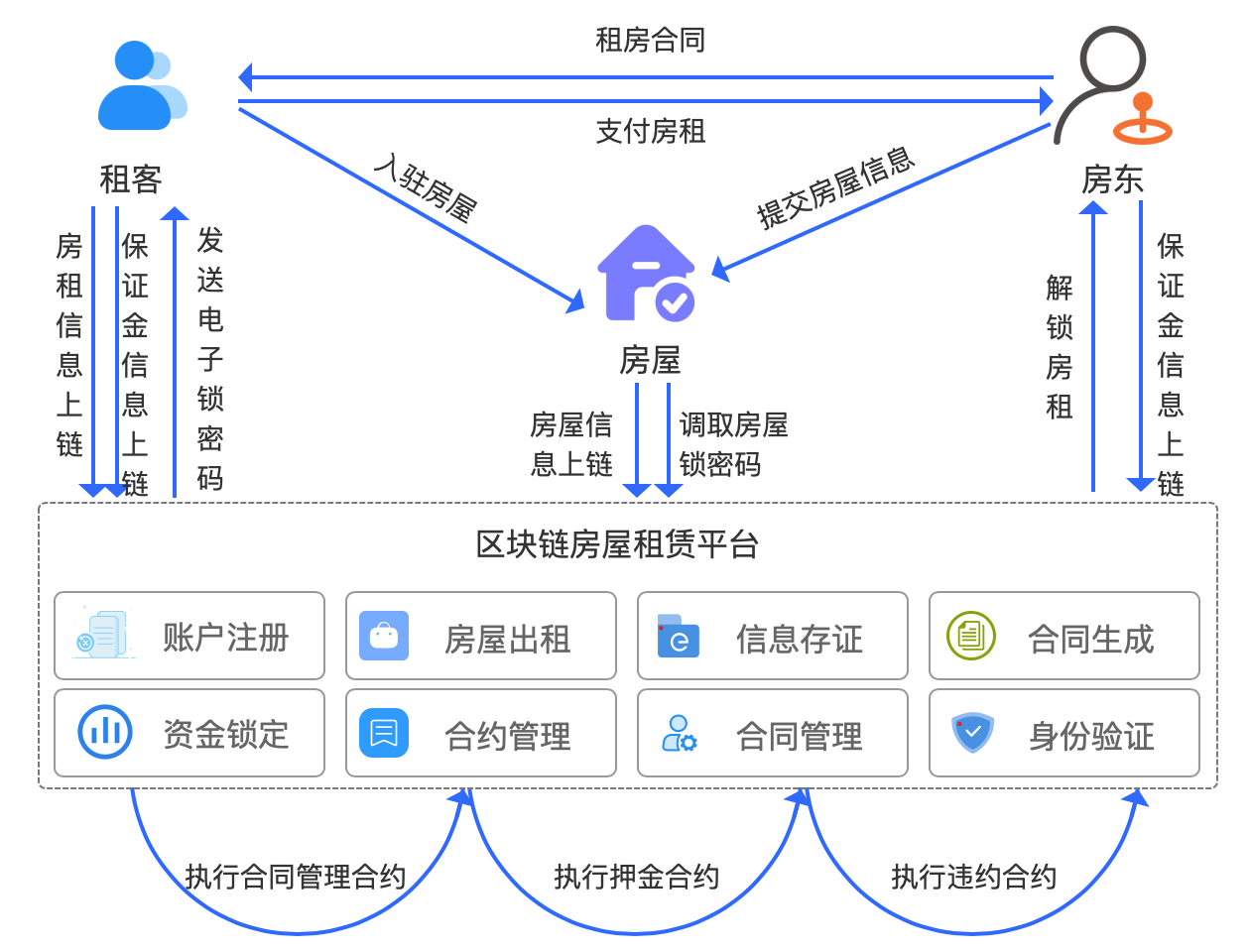
务

书

参赛队编号：

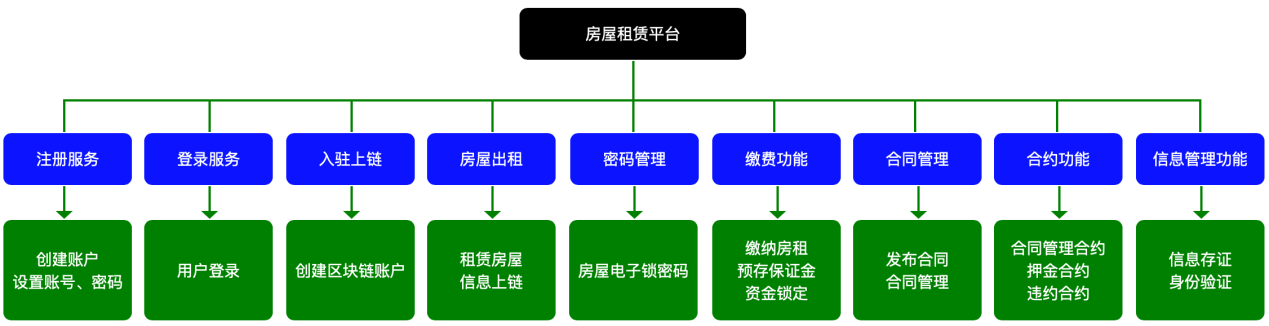
**背景描述**

随着异地务工人员的增多，房屋租赁成为一个广阔是市场；目前，现有技术中的房屋租赁是由房主发布租赁信息，租赁信息发布在房屋中介和/或租赁软件，租客获取租赁信息后，现场看房，并签订纸质的房屋租赁合同；房屋租赁费用通过中介或直接给房主；另外，后期的房租还需房主收取。



区块链房屋租赁平台业务流程图

现在利用区块链技术实现房屋租赁系统，将房东、房屋、租客加入到区块链网络中，将发布、租赁、合同、房租等信息存储在区块链的分布式网络中，永久有效，无法篡改。在房屋租赁场景中, 房东将房屋租出后，无法有效控制房屋的使用权，比如租客未履行租赁合约相应内容时，房东无法及时有效限制租客对房屋的使用，造成租金或房屋使用权的回收困难。现有一个基于区块链的房屋租赁平台S，房东L在平台中发布房源，S可以调用该房屋的电子锁E的密码。L在S中发放租房合同给租客T。在租房合同中规定，每个月的16号缴纳房租，房租是3000元，交付形式是押一付一。在T签署合同后的24小时之内，T需要在S中预存3000元押金、3000元房租，如果T未预存，交易失败。如果T预存了押金和房租，L需要在24小时内在S中预存3000元保证金，如果L未在规定时间内预存保证金，S将T的保证金与房租退回，如果L在规定时间内预存了保证金，双方达成交易。S自动将T的一个月房租转账给L，并将房屋E的密码发送给T。在每月16日，E会自动更换密码，如果T按时缴纳房租，S将房屋密码重新更新给T；当T到期未缴纳，S将T的保证金转账给L代替一个月的租金，S仍会将密码更新给T；如果保证金已被扣除后，T仍然到期未缴纳，平台进入等待补缴状态，T将收不到变更后的密码。如果未到租期L提前终止合同，S将L的保证金转给T，如果T还有保证金将保证金一并退还给T。如果未到租期T提前终止合同，如果还有保证金S将T的保证金转给L，T还需赔偿L一个月租金，如果T保证金已被扣除，T需赔偿L两个月租金，S将L的保证金退还L。当合同正常到期，S将L的保证金退还，T如果还有也退还。



区块链房屋租赁平台系统架构图

## 模块一：区块链产品方案设计及系统运维（35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\模块一提交结果.docx】中对应的任务序号下。

### 任务1-1：区块链产品需求分析与方案设计

房屋租赁平台中涉及到租客、房东、区块链租房平台、房屋等参与方，他们需要在区块链租房平台中完成账户注册、身份上链、出租房屋、生成合同等多种业务活动。通过对业务活动的功能分析，可以更好的服务系统的开发流程。基于房屋租赁平台系统架构，以区块链房屋租赁平台为背景，结合账户注册、登录服务、入驻上链、房屋出租、房租缴费等核心功能描述，撰写流程图/功能图、用例图等概要设计。本环节需要依据项目背景完成需求分析与方案设计，具体要求如下:

1. 依据给定区块链房屋租赁平台的业务流程图以及用例表，使用UML工具编制系统业务用例图，用例图中包含系统参与角色以及用例。

表1-1-1 用例表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **用例编号** | **用例标题** | **优先级** | **测试步骤** | **预期**  **结果** | **测试结果** |
| HK-TEST-01 | 房东在区块链房屋租赁平台出租房屋 | 高 | 在房屋租赁页面，点击出租功能按钮，跳转到维护房屋信息的页面 | 成功跳转 | 成功跳转 |
| HK-TEST-02 | 房东在区块链房屋租赁平台发放合同 | 高 | 在租房管理页面获取租客的租房列表信息，在租房列表中点击发放按钮，可以对该租客发放默认的房屋合同 | 发放成功 | 发放成功 |
| HK-TEST-03 | 租客在区块链房屋租赁平台签署合同 | 高 | 在租房管理页面获取租房信息列表，在列表中点击签署按钮，弹窗显示租房合同，将个人信息填充到合同中 | 收到合同  签署成功 | 收到合同签署成功 |
| HK-TEST-04 | 租客在区块链房屋租赁平台缴纳费用 | 高 | 在房租缴费页面，获取租房合同信息列表，在列表中点击缴费按钮，弹窗显示租金和保证金并显示缴费倒计时，在倒计时内缴费成功 | 合同记录  缴费成功 | 合同记录  缴费成功 |
| HK-TEST-05 | 房东在区块链房屋租赁平台缴纳费用 | 高 | 在租房缴费页面，获取租房合同信息列表，在列表中点击缴费按钮，弹窗显示保证金和缴费倒计时，在倒计时内缴费成功 | 合同记录  缴费成功 | 合同记录  缴费成功 |
| HK-TEST-06 | 区块链房屋租赁平台将房租转账给房东 | 高 | 租客、房东缴费成功后，平台自动将房租转账给房东。在房东账户管理界面，显示出收款记录。同时在租客的租赁记录里自动回显房屋的最新电子锁密码 | 收款成功  电子锁密码回显成功 | 收款成功 |
| HK-TEST-07 | 租客或者房东提前终止合同 | 高 | 租客或者房东在合同信息列表页，点击终止按钮，弹窗进行二次确认是否提前终止，确认则进入违约页面 | 终止成功 | 终止成功 |

2. 依据给定的背景信息、区块链房屋租赁业务流程图以及区块链房租租赁平台架构图以及给出的房屋租赁业务的核心流程，使用思维导图工具编制业务系统功能图；

表1-1-2 房屋租赁平台的核心流程

|  |  |
| --- | --- |
| 发布租房合同流程 | 房东起草租房合同协议，填写房屋信息、租期、房租等。使用房东的私钥对租房合同进行签名并广播到区块链中进行存证 |
| 签署合同流程 | 租客解密租房协议，验证租房协议的来源。通过租客节点在租房协议中填写个人信息和签名信息，生成租房合约 |
| 租房合约属性文本广播流程 | 租客对租房合约提炼合约属性文本，使用私钥对合约属性本文以及租房合约进行签名后，广播到区块链中 |
| 房东解密合约属性文本流程 | 房东使用自身私钥对加密后的租房合约进行解密，对解密后的租房合约提炼对照合约属性文本，并使用私钥对对照合约属性文本进行签名后广播到区块链中 |
| 缴纳房租流程 | 区块链房屋租赁平台节点实时房屋到期时间，如果房屋到期时，判断房租缴纳和电子锁密码的更新 |

3．按照基础层、合约层、接口层以及应用层来设计区块链系统的架构，画出系统架构图，其中在基础层需指明需要的节点、名称、协议、存储等信息；

4．结合案例背景将区块链房屋租赁平台核心功能进行划分，完成下方表格中各个主要模块要实现的功能；

|  |  |
| --- | --- |
| 房屋出租模块 | 请输入要实现的功能 |
| 合同发放模块 | 请输入要实现的功能 |
| 合同生效模块 | 请输入要实现的功能 |
| 费用缴纳模块 | 请输入要实现的功能 |
| 租房合约执行模块 | 请输入要实现的功能 |
| 违约判断功能模块 | 请输入要实现的功能 |

5．根据用例设计以及下方提供好的承诺集，使用Visio工具完成区块链应用系统业务流程图；

表1-1-3 承诺集

|  |  |
| --- | --- |
| 房屋出租承诺 | 判断房东是否在房屋租赁平台发布房屋 |
| 缴纳费用承诺 | 判断租客是否在房屋租赁平台缴纳房租和预存保证金；判断房东是否在房屋租赁平台预存保证金 |
| 房租到期缴纳承诺 | 判断租客是否按时将房租转账给房东，是则发送电子密码锁，否则解锁保证金 |
| 合同到期承诺 | 判断合同是否到期，是则进入判断保证金是否被转账流程，是则结束，否则退还 |

6．整合所有内容，模块调用接口编写并形成对应的需求用例文档至【区块链技术应用赛\重命名为工位号】下。

### 任务1-2：区块链系统部署与运维

通过给定区块链项目需求，进行区块链系统的的部署，包括系统部署、节点部署等。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的维护。

**子任务1-2-1：区块链网络环境搭建**

在本机部署区块链网络底层环境，我们需要准备编译运行的环境，并通过区块链底层源码脚本文件编译区块链网络。

（1）打开区块链源码文件，检查区块链网络基础运行环境，通过输入命令，检查Docker环境，并确定Docker版本；

（2）打开区块链源码文件，通过输入命令，make编译区块链网络，生成区块链网络编译执行文件，请把编译执行文件截图；

（3）成功编译区块链网络后，通过输入命令，启动区块链服务；

（4）启动区块链网络后，通过输入命令，检查区块链网络状态。

**子任务1-2-2：区块链网络服务运行**

区块链网络环境成功搭建后，启动区块链网络服务，区块链网络提供了很多的命令行基础功能，满足区块链网络服务的运行。

（1）进入区块链网络执行文件夹（output），通过命令行方式，创建区块链普通钱包账户（userTest），生产区块链钱包地址，私钥，公钥；

（2）进入区块链网络执行文件夹（output），通过命令行方式，获取区块链钱包余额（data/keys）；

（3）进入区块链网络执行文件夹（output），通过命令行方式，由data/keys向（userTest）转账10，并查看交易ID；

（4）进入区块链网络执行文件夹（output），通过命令行方式，使用上一步的交易ID，查询交易状态，交易源和目标账号、交易的金额等信息

**子任务1-2-3：部署区块链多节点服务**

搭建一个3个节点的区块链网络，需要通过命令行创建多节点网络部署环境，并监听各个节点的RPC监听端口，并查看p2p监听渡口配置，现有任务如下：

（1）进入区块链网络主目录下，通过命令行，创建多节点网络部署环境；

（2）进入多节点网络环境目录下，创建多节点网络部署环境后，通过命令行方式，查看P2P网络配置；

（3）节点加入网络需要通过配置种子节点的网络连接地址netURL,请通过命令行方式，获取node3对应的节点netURL。

**子任务1-2-4：搭建区块链网络监控系统**

镜像提供了Prometheus开源框架，用户可以通过Prometheus+Grafana构建自己应用的监控系统。请按如下步骤，完成对系统的搭建：

（1）进入区块链网络监控系统主目录，通过命令行方式，在节点 env.yaml 配置中打开 metricSwitch；

（2）配置prometheus 设置endpoint服务修改 prometheus.yml 文件，添加如下配置，每个 target 对应区块链节点的 server.yaml 中的 metricPort 端口；

（3）进入到对应目录下，通过命令行启动启动prometheus，打开区块链监控系统项目网址。

### 任务1-3：区块链系统测试

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．启动区块链浏览器系统，找到区块链最新区块列表接口，使用Postman工具对该接口进行HTTP请求测试，并将测试保存；

2．使用测试工具进行测试区块链网络，针对区块链网络吞吐量与时延进行性能指标分析；

3．通过SQL手工注入方法分析区块链系统漏洞，寻找环境注入点，判断注入类型并获取数据库中的信息。

## 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：智能合约设计

根据房屋租赁系统需求用例文档，设计合约接口，画出各需求用例的时序图。

### 任务2-2：智能合约开发

使用Solidity语言进行智能合约开发，根据需求用例文档在待补充源码中完成程序接口功能的编码，解决代码错误和警告，正确编译合约，功能调试正确，运行合约进行业务功能的验证，成功获取合约的ABI，将合约部署至区块链，获取部署的合约信息，将任务中编写代码、运行截图。

**子任务2-2-1：合同管理合约编码**

根据需求用例文档在待补充源码中完成合同管理合约的编码，解决代码错误和警告，正确编译合约，功能调试正确，运行合约中的房东签署合同、租金支付接口功能。

（1）编写房东签署合同接口，完成本合同位置只允许房东签署，通过合同中的信息生成租赁合同的链上哈希，触发协议签署合同的功能，其中合同中的信息包括房东链上账户、租客链上账户、租赁开始时间、租赁结束时间、月租金额、押金金额、交租时间；

（2）编写租金支付接口，完成只允许租客支付租金的规则，检查支付的租金金额是否正确，触发记录租金支付情况的功能。

待补充源码：

contract RentalAgreement {

address public landlord; // 房东的账户

address public tenant; // 租客的账户

uint public rentAmount = 3000; // 每月租金金额（以wei为单位）

uint public depositAmount = 3000; // 押金金额（以wei为单位）

uint public rentDueDate = 16; // 租金到期日

uint public leaseStartDate; // 租赁开始日期

uint public leaseEndDate; // 租赁结束日期

uint public leaseDuration; // 租赁期限（以秒为单位）

bytes32 public contractHash; // 租赁协议的哈希值

bytes32 public propertyHash; // 租赁物业信息的哈希值

event AgreementSigned(address indexed \_signer, bytes32 \_contractHash); // 租赁协议签署事件

event RentPaid(address indexed \_payer, uint \_amount); // 租金支付事件

event DepositPaid(address indexed \_payer, uint \_amount); // 押金支付事件

event LeaseTerminated(address indexed \_terminator); // 租赁终止事件

constructor(address \_landlord, address \_tenant, uint \_leaseDuration, bytes32 \_propertyHash) {

landlord = \_landlord;

tenant = \_tenant;

leaseDuration = \_leaseDuration; // 初始化租赁期限

propertyHash = \_propertyHash;

}

/\*\*\*\*\*\*\*\*\*\*\* 房东签署合同接口开发 \*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\* 房东签署合同接口开发 \*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\* 租金支付接口开发 \*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\* 租金支付接口开发 \*\*\*\*\*\*\*\*\*\*\*/

function payDeposit() public payable {

require(msg.sender == tenant, "Only the tenant can pay the deposit."); // 仅允许租客支付押金

require(msg.value == depositAmount, "Incorrect deposit amount."); // 检查支付的押金金额是否正确

emit DepositPaid(msg.sender, msg.value); // 触发押金支付事件

}

function terminateLease() public {

require(msg.sender == landlord, "Only the landlord can terminate the lease."); // 仅允许房东终止租赁

require(block.timestamp < leaseEndDate, "Lease has already ended."); // 检查租赁是否已经结束

leaseEndDate = block.timestamp; // 更新租赁结束日期为当前时间

emit LeaseTerminated(msg.sender); // 触发租赁终止事件

}

}

**子任务2-2-2：违约管理合约编码**

根据需求用例文档在待补充源码中完成违约管理合约的编码，解决代码错误和警告，正确编译合约，功能调试正确，运行合约中的房东终止合同、租客终止合同接口功能。

（1）编写房东终止合同接口，实现房东终止合同判断，如果租客已经终止合同则合同无效，如果合同有效，对合同终止状态进行标记，将剩余押金退还给租客的功能；

（2）编写租客终止合同接口，实现租客终止合同判断，如果房东已经终止合同则合同无效，如果合同有效，对合同终止状态进行标记，将剩余押金退还给房东的功能。

待补充源码：

contract RentalContract {

address payable public landlord; // 房东地址

address payable public tenant; // 租客地址

uint public rentAmount; // 租金

uint public securityDeposit; // 押金

uint public contractStartDate; // 合同开始日期

uint public contractEndDate; // 合同结束日期

uint public gracePeriod; // 宽限期

uint public terminationFee; // 终止合同费用

bool public contractActive; // 合同是否有效

bool public landlordTerminated; // 房东是否终止合同

bool public tenantTerminated; // 租客是否终止合同

bool public tenantDefaulted; // 租客是否违约

bool public securityDepositReturned; // 押金是否已退还

constructor(address payable \_landlord, address payable \_tenant, uint \_rentAmount, uint \_securityDeposit, uint \_contractStartDate, uint \_contractEndDate, uint \_gracePeriod, uint \_terminationFee) {

landlord = \_landlord;

tenant = \_tenant;

rentAmount = \_rentAmount;

securityDeposit = \_securityDeposit;

contractStartDate = \_contractStartDate;

contractEndDate = \_contractEndDate;

gracePeriod = \_gracePeriod;

terminationFee = \_terminationFee;

contractActive = true;

landlordTerminated = false;

tenantTerminated = false;

tenantDefaulted = false;

securityDepositReturned = false;

}

function payRent() public payable {

require(msg.sender == tenant, "只有租客可以支付租金。"); // 判断是否为租客

require(msg.value == rentAmount, "租金金额不正确。"); // 判断支付的租金是否正确

require(contractActive, "合同无效。"); // 判断合同是否有效

require(block.timestamp <= contractEndDate, "合同已过期。"); // 判断合同是否已过期

if (block.timestamp > contractStartDate + gracePeriod) { // 如果已经超过宽限期

tenantDefaulted = true; // 标记租客违约

if (securityDeposit >= rentAmount) { // 如果押金足够支付租金

securityDeposit -= rentAmount; // 押金减去租金

} else {

securityDeposit = 0; // 押金清零

}

if (block.timestamp > contractEndDate) { // 如果合同已经过期

contractActive = false; // 合同无效

if (securityDeposit > 0) { // 如果押金还有剩余

tenant.transfer(securityDeposit); // 将押金退还给租客

}

if (msg.value > rentAmount) { // 如果支付的租金超过了租金金额

payable(msg.sender).transfer(msg.value - rentAmount); // 将多余的租金退还给租客

}

} else {

landlord.transfer(rentAmount); // 将租金转账给房东

}

} else {

landlord.transfer(rentAmount); // 将租金转账给房东

}

}

function terminateContract() public {

require(contractActive, "合同无效。"); // 判断合同是否有效

if (msg.sender == landlord) { // 如果是房东

require(!landlordTerminated, "房东已终止合同。"); // 判断房东是否已经终止合同

/\*\*\*\*\*\*\*\*\*\*\* 编写房东终止合同接口开发 \*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\* 编写房东终止合同接口开发 \*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\* 编写租客终止合同接口开发 \*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\* 编写租客终止合同接口开发 \*\*\*\*\*\*\*\*\*\*\*/

}

function getSecurityDeposit() public {

require(!contractActive, "合同仍然有效。"); // 判断合同是否已经失效

require(!securityDepositReturned, "押金已经退还。"); // 判断押金是否已经退还

require(!tenantDefaulted, "租客已违约。"); // 判断租客是否违约

if (msg.sender == landlord) { // 如果是房东

securityDepositReturned = true; // 标记押金已经退还

landlord.transfer(securityDeposit); // 将押金退还给房东

} else if (msg.sender == tenant) { // 如果是租客

require(!landlordTerminated, "房东已终止合同。"); // 判断房东是否已经终止合同

require(!tenantTerminated, "租客已终止合同。"); // 判断租客是否已经终止合同

require(block.timestamp > contractEndDate, "合同尚未到期。"); // 判断合同是否已经到期

securityDepositReturned = true; // 标记押金已经退还

tenant.transfer(securityDeposit); // 将押金退还给租客

}

}

}

**子任务2-2-3：押金管理合约编码**

根据需求用例文档在待补充源码中完成押金管理合约的编码，解决代码错误和警告，正确编译合约，功能调试正确，运行合约中的租客缴纳押金情况查询、房东收取押金情况查询接口功能。

（1）编写租客缴纳押金情况查询接口，实现查询租客是狗已缴纳押金功能；

（2）编写房东收取押金情况查询接口，实现房东是否已收到押金的功能。

待补充源码：

contract Rent {

uint256 constant deposit = 3000; // 押金金额

uint256 constant rent = 3000; // 租金金额

uint256 constant timeLimit = 1 days; // 时间限制

struct Tenant {

bool deposited; // 是否已缴纳押金

bool rentPaid; // 是否已缴纳租金

uint256 depositTime; // 缴纳押金的时间

}

struct Landlord {

bool deposited; // 是否已收到押金

uint256 depositTime; // 收到押金的时间

}

mapping(address => Tenant) public tenants; // 租户信息

mapping(address => Landlord) public landlords; // 房东信息

function tenantDeposit() public payable {

require(!tenants[msg.sender].deposited && !tenants[msg.sender].rentPaid, "Tenant has already deposited or paid rent"); // 确保租户未缴纳押金或租金

require(msg.value == deposit + rent, "Tenant must deposit both deposit and rent"); // 确保租户缴纳的金额正确

tenants[msg.sender].deposited = true; // 标记租户已缴纳押金

tenants[msg.sender].depositTime = block.timestamp; // 记录缴纳押金的时间

}

function landlordDeposit() public payable {

require(tenants[msg.sender].deposited && tenants[msg.sender].rentPaid, "Tenant has not deposited or paid rent"); // 确保租户已缴纳押金和租金

require(!landlords[msg.sender].deposited, "Landlord has already deposited"); // 确保房东未收到押金

landlords[msg.sender].deposited = true; // 标记房东已收到押金

landlords[msg.sender].depositTime = block.timestamp; // 记录收到押金的时间

}

function checkTenantDeposit(address tenant) public view returns (bool) {

return tenants[tenant].deposited; // 返回租户是否已缴纳押金

}

/\*\*\*\*\*\*\*\*\*\*\* 租客缴纳押金情况查询接口 \*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\* 租客缴纳押金情况查询接口 \*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\* 收取押金情况查询接口 \*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\* 收取押金情况查询接口 \*\*\*\*\*\*\*\*\*\*\*/

function checkTenantDepositTime(address tenant) public view returns (uint256) {

return tenants[tenant].depositTime; // 返回租户缴纳押金的时间

}

function checkLandlordDepositTime(address landlord) public view returns (uint256) {

return landlords[landlord].depositTime; // 返回房东收到押金的时间

}

function checkTimeLimit() public view returns (bool) {

return block.timestamp <= tenants[msg.sender].depositTime + timeLimit; // 返回是否在时间限制内

}

function refund() public {

require(!landlords[msg.sender].deposited && block.timestamp > tenants[msg.sender].depositTime + timeLimit, "Landlord has already deposited or time limit has not been reached"); // 确保房东未收到押金且时间限制已到期

payable(msg.sender).transfer(deposit + rent); // 将押金和租金退还给租户

}

}

**子任务2-2-4：合约部署和调用**

（1）解决代码错误和警告，正确编译合约，成功获取三个合约的ABI;

（2）将三个合约形成一个文件部署至区块链，获取部署合约的地址信息。

### 任务2-3：智能合约测试

根据需求用例文档完成违约管理的功能测试以及性能测试，解决违约管理合约代码正确性的验证以及验证系统响应时间是否满足预期响应效果，编写功能测试用例，实现违约管理的业务测试，使用测试工具，对合同管理接口进行性能测试。

1.根据需求用例文档在以下测试用例表格中编写违约管理合约中的测用例，依据违约管理中租客违约、房东违约、合同违约的三种状态，分析在前置条件为签署合同、缴纳保证金、押金的情况下，租客提前解除合同、房东提前解除合同、合同失效等情况下，产生的保证金和押金的归属结果；

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 用例编号 | 用例标题 | 优先级 | 前置条件 | 测试步骤 | 测试数据 | 预期结果 |
| FW-RC0001 | 违约管理（租客违约） | 高 | 请填写 | 请填写 | 请填写 | 请填写 |
| 请填写 | 请填写 | 请填写 | 请填写 | 请填写 | 请填写 | 请填写 |
| 请填写 | 请填写 | 请填写 | 请填写 | 请填写 | 请填写 | 请填写 |

2.利用测试工具，模拟测合同管理接口发送150次请求，将所需HTTP请求、协议、服务器名称或IP以及路径填到工具Web服务器中，连同通请求一起发送的参数，运行成功后将汇总报告提交到指定位置。

## 模块三：区块链应用系统开发（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\模块三提交结果.docx】中对应的任务序号下。

### 任务3-1：区块链应用前端功能开发

完成区块链应用系统的构建、服务器端（后端）与Web端（前端）的接口的联调。要求如下：

1．使用VsCode工具，按照押金详情原型图的长度、宽度、行高、间距、文字样式、颜色等，完成押金详情页面的样式开发；

2．使用VsCode工具，完成Vue调用押金查询接口API，获取接口返回的租房人、房屋位置、楼号、押金金额、收款人、收款日期信息，填充至Vue页面中。



押金详情页面的样式开发补充源码：

|  |
| --- |
| .content{  width: 100%;  height: 100%;  background: #EEEEEE;  .contentModel{  position: absolute;  top: 50%;  left: 50%;  box-sizing: border-box;  width: 560px;  padding: 41px 40px 40px;  background: #fff;  border-radius: 8px;  box-shadow: 0 3px 12px 0 #c0c4cc;  transform: translate(-50%, -50%);  -ms-transform: translate(-50%, -50%);  -moz-transform: translate(-50%, -50%);  -webkit-transform: translate(-50%, -50%);  -o-transform:translate(-50%, -50%);  }  // 此处代码补全:按原型图样式，进行长度、宽度、行高、间距、文字样式、颜色等样式设置；  } |

押金详情页面的Html开发补充源码：

|  |
| --- |
| <template>  <div class="content">  <div class='contentModel'>  <!-- 此处代码补全:按照原型图格式，画出租房人、房屋位置、楼号、押金金额、收款人、收款日期信息 -->  </div>  </div>  </template> |

### 任务3-2：区块链应用后端功能开发

**子任务3-2-1：区块链网络环境启动**

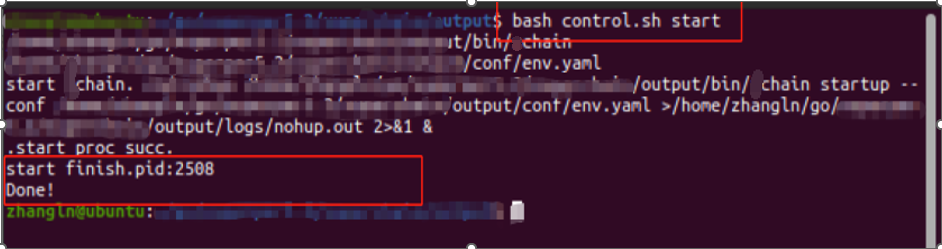
区块链应用系统开发需要区块链底层网络进行支撑，完成和链上数据进行交互，与节点建立链接，完成链上信息查询。要求如下：

（1）打开区块链网络文件目录，通过区块链启动脚本，输入bash control.sh start命令，启动区块链网络；

（2）使用IntelliJ IDEA工具，打开

（src/controller/BlockController.java）文件，在查询区块链信息接口中，使用Java-SDK获取区块链的最新高度和最新交易Hash，并将结果按十进制的整数和字符串类型返回。

启动区块链源码截图：



区块链的最新高度和最新交易Hash接口补充源码：

|  |
| --- |
| /\*\*获取最新区块高度和最新交易Hash\*/  @RequestMapping(value = "/getBlockInformation")  @ResponseBody  public Map<String,Object> getBlockInformation(FlightInformation flightInformation){  Map<String,Object> resMap = new HashMap<>();  try{    // 获取最新区块高度  long blockHeight = block.getLong("height");  System.out.println("最新区块高度：" + blockHeight);  // 获取最新区块交易Hash  String txHashes = block.getJSONArray("txHashes");  System.out.println("最新区块交易Hash：" + txHash);    resMap.put("code",200);  resMap.put("message","请求成功");  resMap.put("data",null);  } catch (ClientProtocolException e) {  log.error("操作失败—log：",e);  resMap.put("code",500);  resMap.put("message","请求失败");  } catch (IOException e) {  log.error("操作失败—log：",e);  resMap.put("code",500);  resMap.put("message","请求失败");  }catch (Exception e){  log.error("操作失败—log：",e);  resMap.put("code",500);  resMap.put("message","请求失败");  }  return resMap;  } |

**子任务3-2-2：实体类声明和数据库设计**

（1）根据“签署房屋租赁合同合约”中变量的字段，在Java项目中声明实体类（HouseLeasingContract），将声明代码结果截图保存；

包含字段：房东的账户公钥（landlord）、租客的账户公钥（tenant）、每月租金金额（rentAmount）、押金金额（depositAmount）、租金到期日（rentDueDate）、租赁开始日期（leaseStartDate）、租赁结束日期（leaseEndDate）、租赁期限（leaseDuration）、租赁协议的哈希值（contractHash）、租赁物业信息的哈希值（propertyHash）。

（2）请打开Mysql命令行，连接数据库，根据第1步声明的实体类，创建数据库表（house\_leasing\_contract）与表结构字段。

签署房屋租赁合同合约实体字段补充源码：

|  |
| --- |
| /\*\*  \* 签署房屋租赁合同实体  \*/  public class HouseLeasingContract {  private static final long serialVersionUID = 1L;  //在此处进行代码补全，声明签署房屋租赁合同实体字段，并添加Get和Set方法  } |

创建house\_leasing\_contract表补充源码：

|  |
| --- |
| CREATE TABLE `house\_leasing\_contract` (  `id` INT NOT NULL AUTO\_INCREMENT COMMENT '主键id',  //在此处进行建表sql语句补全或使用工具创建表字段  `create\_time` DATETIME NOT NULL COMMENT '创建时间',  `update\_time` DATETIME NOT NULL COMMENT '修改时间',  PRIMARY KEY (`id`) USING BTREE  )  COLLATE='utf8\_general\_ci'  ENGINE=InnoDB  ROW\_FORMAT=COMPACT  ; |

**子任务3-2-3：编写调用查询押金合约接口**

已将押金管理合约部署至区块链，通过Java-SDK完成调用押金管理合约接口，获取租房押金信息。要求如下：

（1）接受从Web端接收对应各种参数（包括钱包地址）；

（2）调用Java-SDK，运行调用智能合约API，接收租房押金信息传递给前端页面；

（3）租房押金信息查询成功后，将获取到的押金信息进行解析，并通过数据库依赖包（mysql-connector-java-bin.jar）存储到数据库中。

|  |
| --- |
| /\*\*获取押金详情接口\*/  @RequestMapping(value = "/getBlockCashPledgeInformation")  @ResponseBody  //此处代码补全:接收从Web端传输的参数（包含钱包地址）  public Map<String,Object> getBlockCashPledgeInformation(…){  Map<String,Object> resMap = new HashMap<>();  try{  //此处代码补全:调用Java-SDK，运行调用智能合约API，接收租房押金信息传递给前端页面  //此处代码补全：租房押金信息查询成功后，将获取到的押金信息进行解析，并通过数据库依赖包（mysql-connector-java-bin.jar）存储到数据库中  resMap.put("code",200);  resMap.put("message","请求成功");  resMap.put("data",depositInfo);  } catch (Exception e){  log.error("操作失败—log：",e);  resMap.put("code",500);  resMap.put("message","请求失败");  }  return resMap;  } |