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（7卷）

任

务

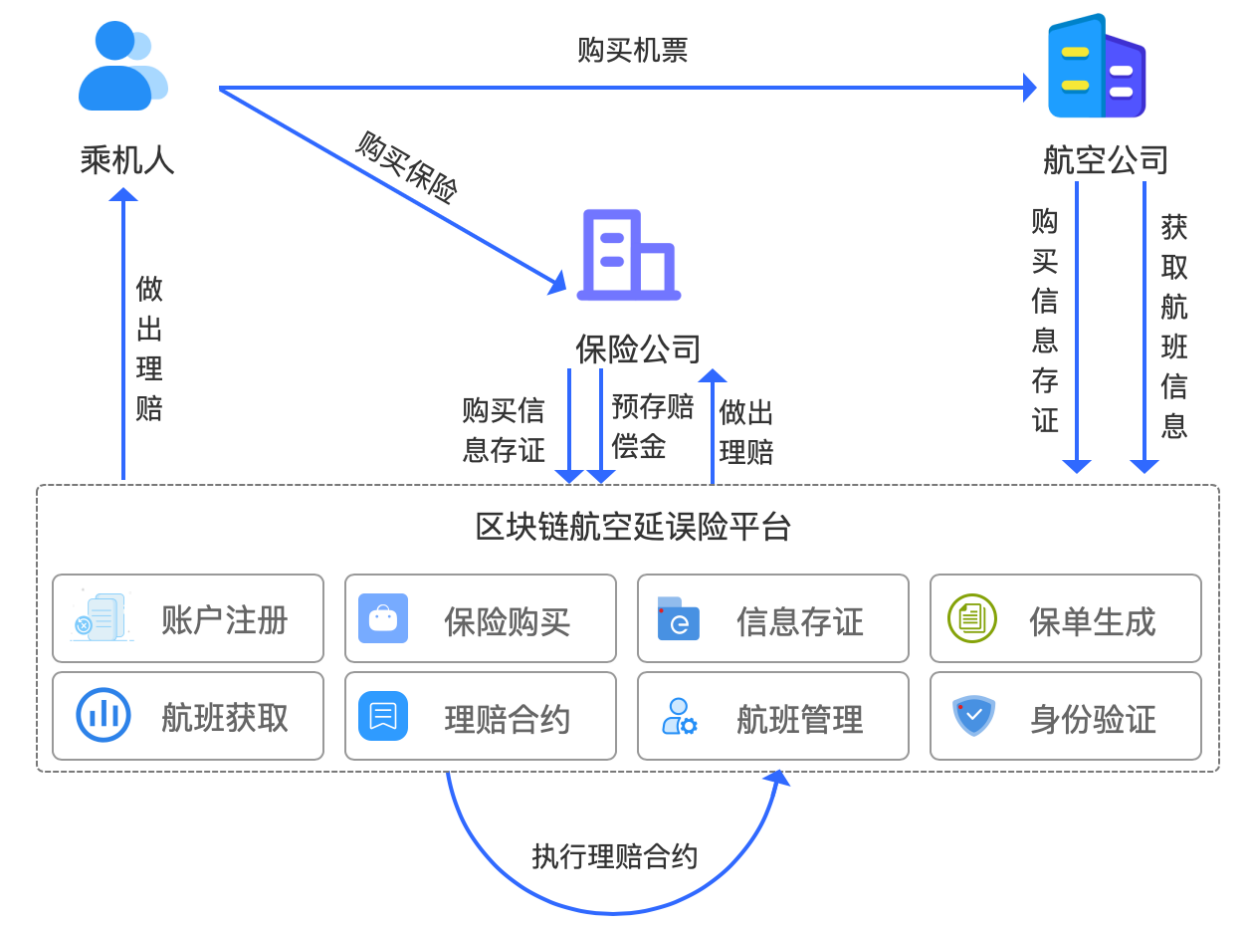
书

参赛队编号：

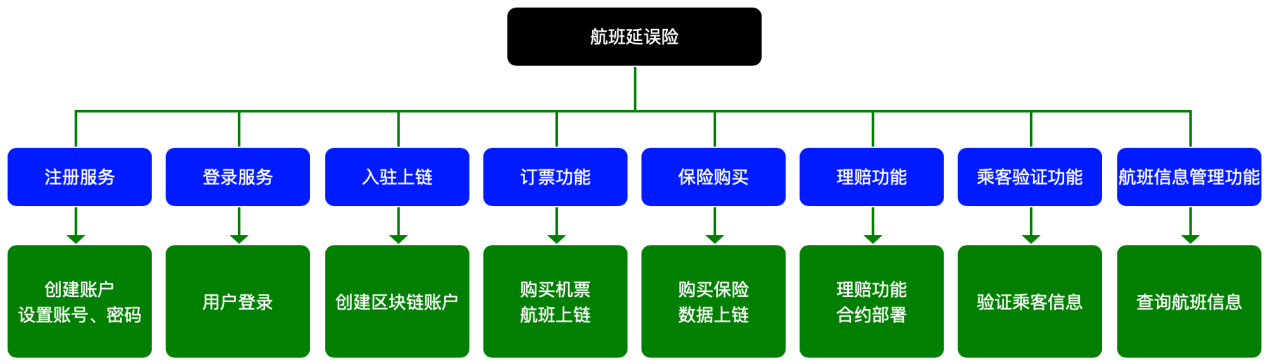
**背景描述**

XXX保险公司首次使用区块链技术为航空旅客提供自动航班延迟赔偿。

保险公司将区块链用于记录保险产品购买以及通过使用区块链上的智能合约来触发自动支付。该智能合约与记录飞行状态的飞行和空中交通数据库相连。当航班一旦延误超过2小时，赔偿机制将会自动执行，直接发送赔偿款到投保人的银行账户中。这种涵盖航班延误的保险，保险条款清晰，报销程序自动执行，客户不必亲自去保险公司索赔，也不需要出示任何文件，所有需要的信息已经在保险订阅期间记录。

区块链航班延误险系统业务流程图

现在利用区块链技术实现航班延误险系统，将乘机人、航空公司、保险公司加入到区块链网络中，将购买、航班、保险、保单等信息存储在区块链的分布式网络中，永久有效，无法篡改。在机票延误险场景中,乘机人购买机票之后在系统上预存10元保费；保险公司再在系统上预存相应的赔偿金1000元;如果保险公司没有按时预存赔偿金，系统就直接将保费退还给用户；如果保险公司预存了赔偿金，若航班没有延误，或者延误时间少于4h，系统就将乘机人预存的保费转账给保险公司，同时退还保险公司预存的赔偿金。如果航班延误超过4h，系统也会将保费转账给保险公司，但是会将保险公司预存的赔偿金赔偿给乘机人。



区块链航班延误险系统架构图

## 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

航班延误险中涉及到乘客、航空公司、区块链系统平台、保险公司等参与方，他们需要在区块链系统平台中完成账户注册、身份上链、机票购买、保险购买等多种业务活动。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。基于航班延误险系统架构，以区块链航班延误系统为背景，结合账户注册、登录服务、入驻上链、购买机票等核心功能描述，撰写流程图/功能图、用例图等概要设计。本任务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. 依据给定区块链航班延误险系统的业务流程图以及用例表，使用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 | 乘机人向航空公司购买机票 | 高 | 在航空公司页面搜索对应航班，点击购买是否能进入到下一个页面 | 成功跳转 | 成功跳转 |
| HK-TEST-02 | 乘机人在0.5h内向保险公司购买航班延误险并成功预存10元保费 | 高 | 在区块链航班延误险页面选择保险公司，正常进入保险公司页面，点击投保，在规定时间内缴纳10元保费，并提示缴费成功 | 购买成功 | 购买成功 |
| HK-TEST-03 | 保险公司在区块链系统中成功预存1000元赔偿金 | 高 | 保险公司保单界面展示乘机人购买保险的记录，点击预存，可成功预存1000元赔偿金 | 收到记录  预存成功 | 收到记录  预存成功 |
| HK-TEST-04 | 在区块链系统中保险公司为乘机人生成保单 | 高 | 保险公司在保单界面展示所有乘机人购买保险的记录，点击生成保单，可会回显出保单的文件 | 记录状态变更  生成成功 | 记录状态变更  生成成功 |
| HK-TEST-05 | 航空公司在区块链系统中上传航班的实时信息 | 高 | 航空公司在区块链系统实时航班页面拉取最新的航班动态信息，并展示到航班页面 | 实时展示 | 实时展示 |
| HK-TEST-06 | 航班延误判断 | 高 | 在区块链系统理赔界面，对最新的航班延误信息进行判断，超出4个小时进行理赔处理，可以在账户中自动收取赔偿金。未超出4个小时，进行无需理赔处理，在理赔界面可查看理赔状态 | 理赔成功执行 | 理赔成功执行 |

2. 依据给定的背景信息、区块链航班延误险业务流程图、区块链航班延误险系统架构图以及航班延误系统的核心流程，使用思维导图工具绘制业务系统功能图；

表1-1-2 航班延误系统的核心流程

|  |  |
| --- | --- |
| 发布理赔协议流程 | 保险公司起草航班延误险理赔协议，使用保险公司的私钥对理赔协议进行签名并广播到区块链中 |
| 生成航班延误险理赔合约流程 | 乘机人购买航空公司的机票后，通过乘机人节点在理赔协议中填写个人信息和航班信息，生成航班延误险理赔合约 |
| 理赔合约属性文本广播流程 | 乘机人对航班延误理赔合约提炼合约属性文本，使用私钥对合约属性本文以及理赔合约进行签名后，广播到区块链中 |
| 保险公司解密合约属性文本流程 | 保险公司使用自身私钥对加密后的理赔合约进行解密，对解密后的理赔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 |
| 航班延误赔付流程 | 链上节点实时监控航班时间，并在航班延误满足条件时，判断保险公司对乘机人的航班延误赔付 |

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航班延误险核心功能进行划分，完成下方表格中各个主要模块要实现的功能；

|  |  |
| --- | --- |
| 机票购买模块 | 请输入要实现的功能 |
| 购买保险模块 | 请输入要实现的功能 |
| 预存赔偿金模块 | 请输入要实现的功能 |
| 生成保单模块 | 请输入要实现的功能 |
| 航班信息上传模块 | 请输入要实现的功能 |
| 延误判断功能模块 | 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

|  |  |
| --- | --- |
| 购买机票承诺 | 判断乘机人是否在航空公司平台购买机票 |
| 购买保险承诺 | 判断乘机人是否在区块链航班延误险平台购买保险 |
| 生成保单承诺 | 判断保险公司是否在规定时间内预存赔偿金 |
| 理赔承诺 | 判断航班延误时间，是否满足理赔时间条件，满足则执行理赔，不满足则执行无需理赔 |

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，我们这些功能来满足我们日常的区块链基础运维工作。

（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监听端口配置，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**

（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，截图保存；

（2）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

根据航班延误险系统需求用例文档，设计合约接口，画出各需求用例的时序图。

### 任务2-2：使用Solidity语言进行智能合约开发（20分）

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。

**子任务2-2-1：航班延误保险购买合约编码（5分）**

根据需求用例文档在待补充源码中完成航班延误购买合约的编码，解决代码错误和警告，正确编译合约，功能调试正确，运行合约中的保险购买、退保功能。

（1）编写航班保险购买上链接口，完成只有购买机票的用户可以购买保险、必须缴纳正确的保费金额、购买保险的时间不能超过购买机票后的0.5小时的功能，符合条件则用户可以购买保险，将用户购买保险状态上链；（3分）

（2）编写退保接口，完成保险公司预存赔偿金后，用户无法退保，反之用户可退保的功能，将用户退保状态上链，并退还用户保费功能。（2分）

待补充源码：

contract FlightDelayInsurance {

address public platformS; // 平台S的地址

address public airlineV; // 航空公司V的地址

address public insuranceCompanyC; // 保险公司C的地址

uint public premium; // 保险费

uint public compensation; // 赔偿金额

uint public purchaseTime; // 购买保险的时间

uint public depositTime; // 存入赔偿金额的时间

bool public purchased; // 是否购买了保险

bool public deposited; // 是否存入了赔偿金额

mapping(address => bool) public insured; // 已退保的用户

mapping(address => bool) public policy; // 已生成保单的用户

mapping(address => bool) public purchasedTicket; // 已购买机票的用户

constructor(address \_platformS, address \_airlineV, address \_insuranceCompanyC, uint \_premium, uint \_compensation) {

platformS = \_platformS; // 初始化平台S的地址

airlineV = \_airlineV; // 初始化航空公司V的地址

insuranceCompanyC = \_insuranceCompanyC; // 初始化保险公司C的地址

premium = \_premium; // 初始化保险费

compensation = \_compensation; // 初始化赔偿金额

}

function purchaseTicket() public {

require(!purchasedTicket[msg.sender], "Ticket has already been purchased"); // 该用户已购买机票

purchasedTicket[msg.sender] = true; // 标记该用户已购买机票

purchaseTime = block.timestamp;

}

/\*\*\*\*\*\*\*\*\*\*\* 航班保险购买上链接口开发 \*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\* 航班保险购买上链接口开发 \*\*\*\*\*\*\*\*\*\*\*/

function depositCompensation() public payable {

require(msg.sender == insuranceCompanyC, "Only insurance company C can deposit compensation"); // 只有保险公司C可以存入赔偿金额

require(msg.value == compensation, "Compensation amount is incorrect"); // 赔偿金额不正确

require(block.timestamp < depositTime + 2 hours, "Deposit time has expired"); // 存入赔偿金额的时间已过期

deposited = true; // 标记已存入赔偿金额

}

/\*\*\*\*\*\*\*\*\*\*\* 退保接口开发 \*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\* 退保接口开发 \*\*\*\*\*\*\*\*\*\*\*/

function generatePolicy() public {

require(deposited, "Compensation has not been deposited"); // 赔偿金额未存入，无法生成保单

require(msg.sender == platformS, "Only platform S can generate policy"); // 只有平台S可以生成保单

require(!policy[msg.sender], "Policy has already been generated"); // 该用户已生成保单

policy[msg.sender] = true; // 标记该用户已生成保单

}

}

**子任务2-2-2：航班信息存证上链合约编码**

根据需求用例文档在待补充源码中完成航班延误存证上链合约的编码，解决代码错误和警告，正确编译合约，调试功能正确，然后运行合约中的获取航班信息、判断航班是否延误接口功能并截图保存，再将所编写代码截图保存。

（1）编写获取航班信息接口，实现依据航班号获得航班号对应的计划起飞时间、实际起飞时间、到达时间、是否延误状态的功能；

（2）编写判断航班是否延误接口，实现依据航班号获得航班号对应的航班是否延误，得到航班是否延误的结果功能。

待补充源码：

contract FlightManagement {

// 航班结构体

struct Flight {

string flightNumber; // 航班号

uint256 scheduledDepartureTime; // 计划起飞时间

uint256 actualDepartureTime; // 实际起飞时间

uint256 arrivalTime; // 到达时间

bool delayed; // 是否延误

}

// 航班号到航班信息的映射

mapping(string => Flight) flights;

// 添加航班

function addFlight(string memory \_flightNumber, uint256 \_scheduledDepartureTime, uint256 \_arrivalTime) public {

flights[\_flightNumber] = Flight(\_flightNumber, \_scheduledDepartureTime, 0, \_arrivalTime, false);

}

// 更新实际起飞时间

function updateActualDepartureTime(string memory \_flightNumber, uint256 \_actualDepartureTime) public {

flights[\_flightNumber].actualDepartureTime = \_actualDepartureTime;

}

/\*\*\*\*\*\*\*\*\*\*\* 获取航班信息接口开发 \*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\* 获取航班信息接口开发 \*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\* 编写判断航班是否延误接口开发 \*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\* 编写判断航班是否延误接口开发 \*\*\*\*\*\*\*\*\*\*\*/

}

**子任务2-2-3：航班延误险理赔合约编码**

根据需求用例文档在待补充源码中完成航班延误险理赔合约的编码，解决代码错误和警告，正确编译合约，完成功能调试，然后运行合约中的客户理赔、保险公司收取保费接口功能并截图保存，再将编写代码截图保存。

（1）编写客户理赔接口，实现如果航班延误超过4小时，将赔偿金赔偿给乘客的功能；

（2）编写保险公司收取保费接口，实现如果航班没有延误或者延误时间少于4小时，将保费转账给保险公司，并退还赔偿金的功能。

待补充源码：

contract Claims is FlightDelayInsurance{

// 定义航班信息结构体

struct Flight {

uint256 departureTime; // 航班出发时间

uint256 delayTime; // 航班延误时间

bool isDelayed; // 航班是否延误

bool isInsured; // 航班是否购买保险

}

// 定义保险公司地址

address public insuranceCompany;

// 定义航班信息映射

mapping(bytes32 => Flight) public flights;

// 定义购买保险事件

event BuyInsurance(bytes32 flightKey, address passenger, uint256 premium);

// 定义航班延误事件

event FlightDelay(bytes32 flightKey, uint256 delayTime);

// 定义理赔事件

event Claim(bytes32 flightKey, address passenger, uint256 amount);

// 定义保险公司预存的赔偿金

uint256 public compensationNew;

// 构造函数，初始化保险公司地址和赔偿金

constructor() FlightDelayInsurance(platformS,airlineV,insuranceCompanyC,premium,compensation){

insuranceCompany = msg.sender;

compensationNew = compensation;

}

// 更新航班信息函数

function updateFlight(bytes32 flightKey, uint256 departureTime, uint256 delayTime, bool isDelayed) payable public {

// 判断调用者是否为保险公司

require(msg.sender == insuranceCompany, "Only insurance company can update flight information");

// 更新航班信息

flights[flightKey].departureTime = departureTime;

flights[flightKey].isDelayed = isDelayed;

/\*\*\*\*\*\*\*\*\*\*\* 客户理赔接口 \*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\* 客户理赔接口 \*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\* 保险公司收取保费接口开发 \*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\* 保险公司收取保费接口开发 \*\*\*\*\*\*\*\*\*\*\*/

// 触发航班延误事件

emit FlightDelay(flightKey, delayTime);

}

}

**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链上合约地址中，获取部署合约的信息。

### 任务2-3：智能合约测试

根据需求用例文档完成航班延误险的功能测试以及性能测试，解决航班延误险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，实现理赔接口的业务测试，使用性能测试工具，对航班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航班延误险合约中的测试用例，依据航班延误险有延误和没有延误的两种状态，分析在前置条件为平台运行状态为真的情况下，用户购买机票、保险以及预存赔偿金后，不同航班延误时间大于、小于、等于理赔规定时间的情况下，产生的保费和赔偿金的归属结果。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 |
| HK-FM001 | 航班延误险（没有延误） | 高 | 请填写 | 请填写 | 请填写 | 请填写 |
| HK-FM002 | 航班延误险（有延误） | 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 |
| 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，模拟测试航班管理接口发送100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，并要包含请求参数，运行成功后提交汇总报告。

## 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

使用Vue调用航班管理接口，将获取的航班号、计划起飞时间、实际起飞时间、到达时间、是否延误等信息传递给前端模板，要求如下：

1. 使用VsCode工具，按照航班管理原型图的长度、宽度、行高、间距、文字样式、颜色等，完成航班信息管理页面的开发；



（2）完成Vue调用航班管理接口API，获取接口返回的航班号、计划起飞时间、实际起飞时间、到达时间、是否延误信息，填充至Vue页面中。

航班信息管理页面样式补充源码：

.content{

width: 100%;

height: 100%;

background: #EEEEEE;

.contentModel{

position: absolute;

top: 50%;

left: 50%;

box-sizing: border-box;

width: 500px;

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；

}

航班信息展示补充源码：

<template>

<div class="content">

<div class='contentModel'>

<h3 class='payment\_title'>航班信息</h3>

<ul class='payment\_info'>

<li>

<span>航班号：</span>

<div>{{获取接口中返回的flightNumber字段}}</div>

</li>

<li>

<span>计划起飞时间：</span>

<div>{{获取接口中返回的departureTime字段}}</div>

</li>

<li>

<span>实际起飞时间：</span>

<div>{{获取接口中返回的practicalDepartureTime字段}}</div>

</li>

<li>

<span>实际到达：</span>

<div>{{获取接口中返回的arrivalTime字段}}</div>

</li>

<li>

<span>是否延误：</span>

<div>{{获取接口中返回的delayState字段}}</div>

</li>

</ul>

</div>

</div>

</template>

### 任务3-2：区块链应用后端功能开发

在航班延误应用后端开发中，使用Java完成在区块链中实现航班信息和航班延误的管理，获取区块链交易信息、航班信息实体类设计、MySql数据库表结构创建、查询航班信息接口，与前端页面交互，形成完整的应用系统。

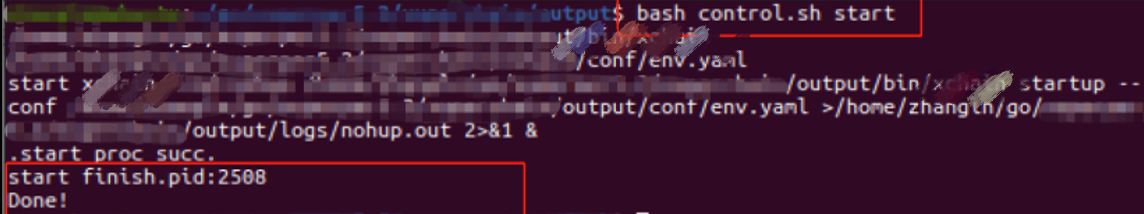
**任务3-2-1：启动区块链**

（1）使用Java语言编写后端代码进行交互，获取区块链的最新高度、最新交易Hash;

要求如下：

* 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，将输出结果截图保存;
* 在IntelliJ IDEA工具中，使用Java-SDK查询区块链的最新高度（数字）和最新交易Hash（字符串），并将结果返回。

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(){  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; } |

**任务3-2-2：实体类声明和数据库设计**

（1）根据“航班信息管理合约”中变量的字段，在Java项目中声明实体类（FlightInformation），将声明代码结果截图保存。

包含字段： 航班号（flightNumber）、计划起飞时间（departureTime）、实际起飞时间（practicalDepartureTime）、到达时间（arrivalTime）、是否延误（delayState）。

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

航班信息实体字段补充源码：

|  |
| --- |
| */\*\*  \* 航班信息实体类  \*/* **public class** FlightInformation {   **private static final long *serialVersionUID*** = 1L;   //在此处进行代码补全，声明航班信息实体字段，并添加Get和Set方法  } |

创建flight\_information表补充源码：

|  |
| --- |
| **CREATE** **TABLE** `flight\_information` (  `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**  ; |

**任务3-2-3：编写调用航班管理合约查询航班详情接口**

在航班延误应用中，使用Java语言调用航班管理智能合约，完成航班信息的添加和查询接口，通过Http请求与航班管理接口进行交互，并将获取航班号、计划起飞时间、实际起飞时间、到达时间、是否延误等信息传递给前端模板，要求如下：

（1）接收Web端传递的对应实体类参数（包括航班唯一标识）；

（2）调用Java-SDK，运行调用智能合约API，获取航班详情传递给前端页面；

（3）订单详情查询成功后，将获取到的订单信息进行解析，并通过数据库依赖包（mysql-connector-java-bin.jar）存储到数据库中。

调用查询航班智能合约接口源码截图：

|  |
| --- |
| */\*\*查询工程进度信息\*/* @RequestMapping(value = **"/getFlightInformation"**) @ResponseBody  //此处代码补全:接收从Web端传输的参数（包括航班唯一标识） **public** Map<String,Object> **getFlightInformation**(…){  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，查询工程进度信息，并返回状态  //此处代码补全：查询到工程进度信息后，将获取到的进度信息进行解析，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"**,result);  } **catch** (Exception e){  log.error(**"操作失败—log："**,e);  resMap.put(**"code"**,500);  resMap.put(**"message"**,**"请求失败"**);  }  **return** resMap; } |