实验报告

姓名	学号
赖韵恬	18340084
李芷阳	18340101

一、项目背景

1.1 任务要求

1.1.1 实现功能

> 1. 实现采购商品—签发应收账款交易上链。例如车企从轮胎公司购买一批轮胎并签订应收账款单据。
> 2. 实现应收账款的转让上链，轮胎公司从轮毂公司购买一笔轮毂，便将于车企的应收账款单据部分转让给轮毂公司。轮毂公司可以利用这个新的单据去融资或者要求车企到期时归还钱款。
> 3. 利用应收账款向银行融资上链，供应链上所有可以利用应收账款单据向银行申请融资。
> 4. 应收账款支付结算上链，应收账款单据到期时核心企业向下游企业支付相应的欠款。

1.1.2 加分项

> a. 功能：除了必要功能之外，实现更多的功能，限定：供应链金融领域相关
>
> b. 底层：改进区块链平台底层或自行开发区块链（共识算法等）
>
> c. 合约：实现链上数据隐私保护（同态加密、属性加密等）
>
> d. 前端：友好高效的用户界面
>
> e. 其他有挑战性的创新

1.2 项目背景说明

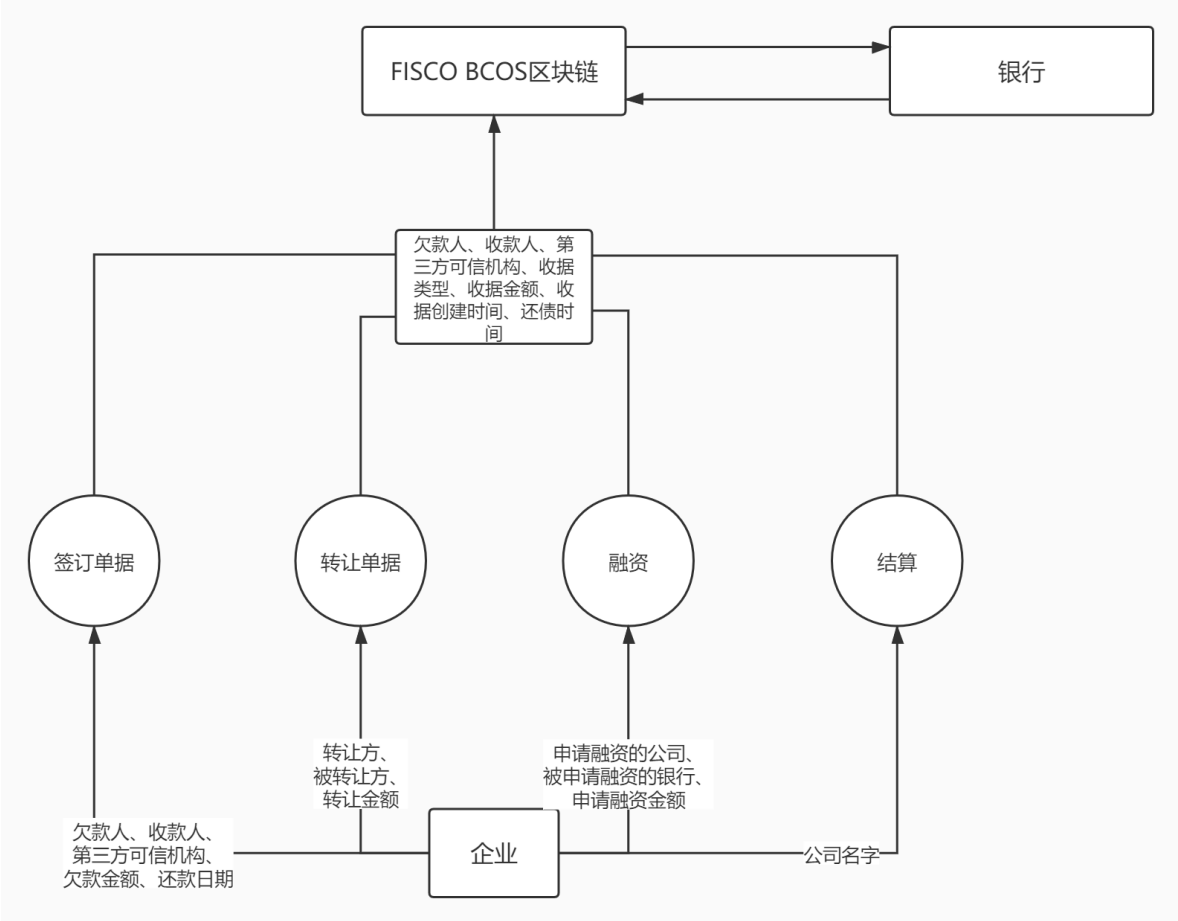
基于已有的开源区块链系统FISCO-BCOS（https://github.com/FISCO-BCOS/FISCO-BCOS），以联盟链为主，开发基于区块链或区块链智能合约的供应链金融平台，实现**供应链收账款资产的溯源、流转。**

区块链+供应链金融：

将供应链上的每一笔交易和应收账款单据上链，同时引入第三方可信机构来确认这些信息的交易，例如银行，物流公司等，确保交易和单据的真实性。同时，支持应收账款的转让，融资，清算等，让核心企业的信用可以传递到供应链的下游企业，减小中小企业的融资难度。

二、方案设计

数据流图



三、功能实现

3.1 基本结构：

根据Solidity语言给出的官方文档中的例子，进行语法学习，然后结合本次项目的要求，进行合约的实现：

由于我们要实现四种基本功能，基于这四种功能，先声明相对应的结构体：`Bill`和`Organization`

链端代码：

```
struct Bill {//收据
        string giver;//欠款人
        string receiver;//收款人
        string middle;//第三方可信机构
        uint Type;//收据类型：1表示普通收据；2表示转让收据；3表示融资收据；0表示已偿还收据
        int money;//收据金额
        uint createdate;//收据创建时间
        uint repaydate;//还债时间
    }
    struct Company {//公司
        string name;//名字
        int money;//总金额
        bool isused;//判断是否存在对应的公司
    }
    struct Bank {//银行
        string name;//名字
        int money;//总金额
        bool isused;//判断是否存在对应的银行
```

```
    }
```

单单只是上面的两个结构体无法完成合约的实现，所以在此基础上声明几个辅助变量，用于计算公司的数量、银行的数量、收据的数量（包括已偿还的收据和未偿还的收据）等。

链端代码：

```
    int lenCom;//公司总数
    int lenBank;//银行总数
    int lenBill;//收据总数
    mapping(string => Company) Companys;//公司名-> 公司信息
    mapping(string => Bank) Banks;//银行名-> 银行信息
    mapping(int => Bill) Bills;//收据id-> 收据信息
    constructor() public {
        lenCpom = 0;
        lenBank = 0;
        lenBill = 0;
    }
```

此外，由于Solidity语言中没有字符串的运算符重载，所以增加辅助函数 `isEqual` 用来实现对判断字符串是否相等。

链端代码：

```
function isEqual(string a, string b) public returns(bool) {
        if (bytes(a).length != bytes(b).length) {
            return false;//如果两个字符串的长度不同，那么字符串肯定不同
        }
        for (uint i=0; i< bytes(a).length; i++) {
            //对字符串进行一对一比较，一旦有一个位置字符串不同，就返回false
            if (bytes(a)[i] != bytes(b)[i]) {
                return false;
            }
        }
        //否则字符串相同
        return true;
    }
```

3.2 功能一：

实现采购商品—签发应收账款交易上链。例如车企从轮胎公司购买一批轮胎并签订应收账款单据。

具体实现步骤：

- 输入：欠款人giver，收款人receiver，第三方可信机构middle，欠款money，还款限定时间（以月为单位）
- 过程：调用链端CreateBill将普通收据实例化，填入相应的收据
- 输出：创建收据结果

链端代码：

```
    function CreateBill(string giver, string receiver, string middle, int money,
unit repaytime) public returns(int) {
        //异常判断：收据双方要是存在的公司不且能是同一个公司
        if (Companys[giver].isused && Companys[reveiver].isused &&
!isEqual(giver, receiver)) {
            //实例化收据：ID从0开始；收据类型为普通收据；还款时间以月为单位
            Bills[lenBill++] = Bill(giver, receiver, middle, 1, money, now, now
+ 60*60*24*30*repaytime);
            return lenBill-1;
        }
        return -1;
    }
```

后端代码

```
#main.py
def
createbill(contract_address,contract_abi,giver,receiver,middle,money,repaytime):

    money=int(money)
    repaytime=int(repaytime)
    receipt=client.sendRawTransactionGetReceipt(contract_address,contract_abi,
                "CreateBill",[giver,receiver,middle,money,repaytime])
    txhash=receipt['transactionHash']
    txresponse=client.getTransactionByHash(txhash)
    inputresult=data_parser.parse_transaction_input(txresponse['input'])

 outputresult=data_parser.parse_receipt_output(inputresult['name'],receipt['outp
ut'])
    return outputresult
#run.py
@app.route('/createBill',methods=['GET','POST'])
def createB():
    return render_template('createB.html')

@app.route('/createBill/result',methods=['GET','POST'])
def createB_re():
    giver=request.form['giver']
    receiver=request.form['receiver']
    middle=request.form['middle']
    money=request.form['money']
    repaytime=request.form['repaytime']
    receipt=createbill(contract_address,contract_abi,giver,receiver,
                       middle,money,repaytime)
    bid=receipt[0]
    if bid==-1:
        result="failed"
    else:
        result="succeed"
    return
render_template('createB_result.html',bid=bid,giver=giver,receiver=receiver,

middle=middle,money=money,repaytime=repaytime,result=result)
```

前端代码

（后面功能前端代码部分基本相同）

```
#createB.html(部分)
<h1>CreateBill</h1>
<form action='/createBill/result' method=post enctype=multipart/form-data>
    Giver <input type=text name=giver><br><br>
    Receiver <input type=text name=receiver><br><br>
    Middle <input type=text name=middle><br><br>
    Money <input type=text name=money><br><br>
    Repaytime <input type=text name=repaytime><br><br>
    <input type=submit value=Submit style="height:40px; width:240px;
display:inline-block; text-align:center;font-family: ' Arial Narrow', Arial,
sans-serif;">
</form>

#createB_result.html(部分)
<h2>Bill id: {{bid}}</h2>
<h2>Giver: {{giver}}</h2>
<h2>Receiver: {{receiver}}</h2>
<h2>Middle: {{middle}}</h2>
<h2>Money: {{money}}</h2>
<h2>Repaytime: {{repaytime}}</h2>
<h2>Create Bill {{result}} </h2>
<form action='/' method=post enctype=multipart/form-data>
    <input type=submit value=return style="height:40px; width:240px;
display:inline-block; text-align:center;font-family: ' Arial Narrow', Arial,
sans-serif;">
</form>
```

3.3 功能二：

实现应收账款的转让上链，轮胎公司从轮毂公司购买一笔轮毂，便将于车企的应收账款单据部分转让给轮毂公司。轮毂公司可以利用这个新的单据去融资或者要求车企到期时归还钱款。

可以分两步实现功能二：首先计算可以被转让的最大金额，然后再进行转让的过程

3.3.1 计算可转让的最大金额

具体实现步骤：

- 输入：转让方giver，被转让方receiver
- 输出：可以转让的最大金额

链端代码：

```
//计算能够转让收据的最大金额数
    function CheckTransfer(string giver, string receiver) public returns(int) {
        int sum = 0;
        if (Companys[giver].isused && Companys[reveiver].isused &&
!isEqual(giver, receiver)) {
            for (int i=0; i<lenBill; i++) {
                //要将giver的应收账单转让receiver
                //判断：首先收据的类型只能是普通收据或者是转让收据，其次收据的收款人必须是
giver，收据的欠款人不能是receiver
                if ((Bills[i].type == 1 || Bills[i].type == 2) &&
isEqual(Bills[i].receiver, giver) && !isEqual(Bills[i].giver, receiver) {
                    sum += Bills[i].money;
```

```
                }
            }
            return sum;
        }
        return -1;
    }
```

3.3.2 正式转让

具体实现步骤：

- 输入：转让方giver，被转让方receiver，转让金额money
- 过程：调用链端Transfer转让收据。
- 输出：转让结果

链端代码：

```
    //转让过程
    function Transfer(string giver, string receiver, int money) public
returns(int) {
        int sum = money;
        //如果最大可转让金额大于要转让的金额
        if (CheckTransfer(giver, receiver) >= money) {
            //遍历所有收据
            for (int i=0; i<lenBill; i++) {
                //判断条件：首先，收据要是普通收据或者是转让收据；其次，收据的收款人必须是
giver，收据的欠款人不能是receiver
                if ((Bills[i].Type == 1 || Bills[i].Type == 2) &&
isEqual(Bills[i].receiver, giver) && !isEqual(Bills[i].giver, receiver)){
                    //如果收据的金额大于剩下未转让的金额
                    if(Bills[i].money > sum){
                        Bills[i].money -= sum;
                        //创建一个新的转让收据转让剩下为转让的金额，结束遍历
                        Bills[lenBill++] = Bill(Bills[i].giver,
receiver,Bills[i].middle, 2, sum, Bills[i].createdate, Bills[i].repaydate);
                        sum = 0;
                    }
                    //如果最大可转让金额小于要转让的金额
                    else{
                        //直接将收据改为转让收据，收款人改成receiver
                        Bills[i].receiver = receiver;
                        Bills[i].Type = 2;
                        sum -= Bills[i].money;
                    }
                    if(sum == 0)
                        break;
                }
            }
            return 0;
        }
        return -1;
    }
```

后端代码

```
#main.py
def transfer(contract_address,contract_abi,giver,receiver,money):
```

```
    money=int(money)
    receipt=client.sendRawTransactionGetReceipt(contract_address,contract_abi,
                "Transfer",[giver,receiver,money])
    print("receipt:",receipt['output'])
    txhash=receipt['transactionHash']
    txresponse=client.getTransactionByHash(txhash)
    inputresult=data_parser.parse_transaction_input(txresponse['input'])

 outputresult=data_parser.parse_receipt_output(inputresult['name'],receipt['outp
ut'])
    return outputresult
#run.py
@app.route('/transfer',methods=['GET','POST'])
def billtransfer():
    return render_template('transfer.html')

@app.route('/transfer/result',methods=['GET','POST'])
def billtransfer_re():
    giver=request.form['giver']
    receiver=request.form['receiver']
    money=request.form['money']
    receipt=transfer(contract_address,contract_abi,giver,receiver,money)
    result=receipt[0]
    if result==0:
        result="succeed"
    else:
        result="failed"
    return render_template('transfer_result.html',giver=giver,receiver=receiver,
                           money=money,result=result)
```

3.4 功能三：

利用应收账款向银行融资上链，供应链上所有可以利用应收账款单据向银行申请融资。

3.4.1 公司可以申请融资的最大金额

- 输入：申请融资的公司giver，被申请融资的银行receiver
- 输出：公司可以申请的融资数

链端代码：

```
    //可以申请融资的最大金额
    function CheckFinancing(string giver, string receiver) public returns(int) {
        int sum = 0;
        //首先公司和银行要存在
        if(Companys[giver].isused && Banks[receiver].isused){
            //遍历所有收据
            for(int i=0;i<lenBill;i++){
                //如果收据是普通收据或者是转让收据，并且满足收据是收款人是giver
                if ((Bills[i].Type == 1 || Bills[i].Type == 2) &&
isEqual(Bills[i].receiver,giver)){
                    //可以申请融资的金额数增加
                    sum += Bills[i].money;
                }
            }
            return sum;
        }
```

```
        return -1;
    }
```

3.4.2 公司申请融资过程

- 输入：申请融资的公司giver，被申请融资的银行receiver
- 过程：调用链端Financing融资
- 输出：融资结果

链端代码：

```
    //公司申请融资过程
    function Financing(string giver, string receiver, int money) public
returns(int) {
        int sum = money;
        //可以申请的最大融资数大于申请的融资数
        if(CheckFinancing(giver,receiver) >= money) {
            //遍历所有收据
            for(int i=0;i<lenBill;i++){
                //如果收据是普通收据或者是转让收据，且收据的收款人是giver
                if ((Bills[i].Type == 1 || Bills[i].Type == 2) &&
isEqual(Bills[i].receiver,giver)){
                    //如果收据的金额大于剩下的未申请融资数
                    if(Bills[i].money > sum){
                        Bills[i].money -= sum;
                        //创建一个新的融资收据融资剩下为融资的金额，结束遍历
                        Bills[lenBill++] = Bill(Bills[i].giver, receiver,
Bills[i].middle, 3, sum, Bills[i].createdate, Bills[i].repaydate);
                        sum = 0;
                    }
                    else{
                        Bills[i].receiver=receiver;
                        Bills[i].Type = 3;
                        sum -= Bills[i].money;
                    }
                    if(sum == 0)
                        break;
                }
            }
            Companys[giver].money += money;
            Banks[receiver].money -= money;
            return 0;
        }
        return -1;
    }
```

后端代码

```
#main.py
def financing(contract_address,contract_abi,giver,receiver,money):
    money=int(money)
    receipt=client.sendRawTransactionGetReceipt(contract_address,contract_abi,
                        "Financing",[giver,receiver,money])
    print("receipt:",receipt['output'])
    txhash=receipt['transactionHash']
    txresponse=client.getTransactionByHash(txhash)
```

```
    inputresult=data_parser.parse_transaction_input(txresponse['input'])

 outputresult=data_parser.parse_receipt_output(inputresult['name'],receipt['outp
ut'])
    return outputresult
#run.py
@app.route('/financing',methods=['GET','POST'])
def billfinancing():
    return render_template('financing.html')

@app.route('/financing/result',methods=['GET','POST'])
def billfinancing_re():
    giver=request.form['giver']
    receiver=request.form['receiver']
    money=request.form['money']
    receipt=financing(contract_address,contract_abi,giver,receiver,money)
    result=receipt[0]
    if result==0:
        result="succeed"
    else:
        result="failed"
    return
render_template('financing_result.html',giver=giver,receiver=receiver,
                           money=money,result=result)
```

3.5 功能四：

应收账款支付结算上链，应收账款单据到期时核心企业向下游企业支付相应的欠款。

3.5.1 统计公司应还收据的所有金额

- 输入：公司名称
- 输出：所有欠款

链端代码：

```
    function CheckRepay(string giver) public returns(int) {
        int sum = 0;
        //公司存在
        if (Companys[giver].isused) {
            //遍历所有收据
            for (int i=0; i<lenBill; i++) {
                //如果收据的欠款人是giver，且收据不是已偿还收据，且现在已经到了偿还改收据的
时间
                if (isEqual(Bills[i].giver, giver) && Bills[i].Type != 0 && now
>= Bills[i].repaydate) {
                    //增加金额
                    sum += Bills[i].money;
                }
            }
            return sum;
        }
        return -1;
    }
```

3.5.2 偿还所有收据

- 输入：公司名称giver
- 过程：调用链端Repay偿还
- 输出：偿还结果

链端代码：

```
    function Repay(string giver) public returns(int) {
        int sum = 0;
        int m=CheckRepay(giver);
        //如果存在需要偿还的收据
        if (m != -1 && m <= Companys[giver].money) {
            遍历所有收据
            for (int i=0; i<lenBill; i++) {
                //如果收据的欠款人是giver，且收据不是已偿还收据，且现在已经到了偿还改收据的
时间
                if (isEqual(Bills[i].giver, giver) && Bills[i].Type != 0 && now
>= Bills[i].repaydate) {
                    //偿还
                    Bills[i].Type = 0;
                    sum += Bills[i].money;
                    Companys[Bills[i].receiver].money += Bills[i].money;
                    Banks[Bills[i].receiver].money += Bills[i].money;
                }
            }
            Companys[giver].money -= sum;
            return sum;
        }
        return -1;
    }
```

后端代码

```
#main.py
def repay(contract_address,contract_abi,giver):
    receipt=client.sendRawTransactionGetReceipt(contract_address,contract_abi,
               "Repay",[giver])
    print("receipt:",receipt['output'])
    txhash=receipt['transactionHash']
    txresponse=client.getTransactionByHash(txhash)
    inputresult=data_parser.parse_transaction_input(txresponse['input'])

 outputresult=data_parser.parse_receipt_output(inputresult['name'],receipt['outp
ut'])
    return outputresult
#run.py
@app.route('/repay',methods=['GET','POST'])
def billrepay():
    return render_template('repay.html')

@app.route('/repay/result',methods=['GET','POST'])
def billrepay_re():
    giver=request.form['giver']
    receipt=repay(contract_address,contract_abi,giver)
    result=receipt[0]
    if result==0:
        result="succeed"
    else:
```

```
        result="failed"
    return render_template('repay_result.html',giver=giver,result=result)
```

3.6 额外功能：

3.6.1 注册公司

输入公司的名称和账户金额，调用链端RegisterCompany保存并输出注册结果。

链端代码：

```
    function RegisterCompany(string memory name, int money) public{
        Companys[name] = Company(name, money, true);
        lenCom++;
    }
```

后端代码

```
#main.py
def registercompany(contract_address,contract_abi,name,money):
    money=int(money)
    receipt=client.sendRawTransactionGetReceipt(contract_address,contract_abi,
                        "RegisterCompany",[name,money])
    print("receipt:",receipt['output'])
    return receipt
#run.py
@app.route('/registerCompany',methods=['GET','POST'])
def registerc():
    return render_template('registercompany.html')

@app.route('/registerCompany/result',methods=['GET','POST'])
def registerc_re():
    name=request.form['cname']
    money=request.form['money']
    receipt=registercompany(contract_address,contract_abi,name,money)
    res=hex_to_signed(receipt['output'])
    print(res)
    if res==0:
        result="succeed"
    else:
        result="failed"
    return render_template('registercompany_result.html',cname=name,
                           money=money,result=result)
```

3.6.2 注册银行

输入银行的名称和账户金额，调用链端RegisterBank保存并输出注册结果。

链端代码：

```
    function RegisterBank(string memory name, int money) public{
        Banks[name] = Bank(name, money, true);
        lenBank++;
    }
```

后端代码

```
#main.py
def registerbank(contract_address,contract_abi,name,money):
    money=int(money)

 receipt=client.sendRawTransactionGetReceipt(contract_address,contract_abi,"Regi
sterBank",[name,money])
    print("receipt:",receipt['output'])
    return receipt
#run.py
    #该部分与注册公司基本一致
```

3.6.3 输出公司

通过输入公司的名字，调用链端GetCompany获得对应公司的金额并输出。

链端代码：

```
    function GetCompany(string memory name) public view returns(string memory,
uint) {
        return (Companys[name].name, Companys[name].money);
    }
```

后端代码

```
#main.py
def getcompany(contract_address,contract_abi,name):
    receipt=client.sendRawTransactionGetReceipt(contract_address,contract_abi,
                                                "GetCompany",[name])
    txhash=receipt['transactionHash']
    txresponse=client.getTransactionByHash(txhash)
    inputresult=data_parser.parse_transaction_input(txresponse['input'])

 outputresult=data_parser.parse_receipt_output(inputresult['name'],receipt['outp
ut'])
    return outputresult
#run.py
@app.route('/getCompany',methods=['GET','POST'])
def getc():
    return render_template('getcompany.html')

@app.route('/getCompany/result',methods=['GET','POST'])
def getc_re():
    name=request.form['cname']
    receipt=getcompany(contract_address,contract_abi,name)
    money=receipt[1]
    return render_template('getcompany_result.html',cname=name,money=money)
```

3.6.4 输出银行

通过输入银行的名字，调用链端GetBank获得对应公司的金额并输出。

链端代码：

```
function GetBank(string memory name) public view returns(string memory, uint) {
        return (Banks[name].name, Banks[name].money);
    }
```

后端代码

```
#main.py
def getbank(contract_address,contract_abi,name):
    receipt=client.sendRawTransactionGetReceipt(contract_address,contract_abi,
                    "GetBank",[name])
    txhash=receipt['transactionHash']
    txresponse=client.getTransactionByHash(txhash)
    inputresult=data_parser.parse_transaction_input(txresponse['input'])

 outputresult=data_parser.parse_receipt_output(inputresult['name'],receipt['outp
ut'])
    return outputresult
#run.py
    #该部分与输出公司基本一致
```

3.6.5 输出收据

通过输入收据的ID，调用链端GetBill获得对应收据的具体信息并输出。

链端代码：

```
    function GetBill(int id) public view returns(string memory, string
memory,string memory, uint, uint, uint, uint) {
        return (Bills[id].giver, Bills[id].receiver,Bills[id].middle,
Bills[id].Type, Bills[id].money, Bills[id].createdate, Bills[id].repaydate);
    }
```

后端代码

```
#main.py
def getbill(contract_address,contract_abi,name):
    name=int(name)
    receipt=client.sendRawTransactionGetReceipt(contract_address,contract_abi,
                        "GetBill",[name])
    txhash=receipt['transactionHash']
    txresponse=client.getTransactionByHash(txhash)
    inputresult=data_parser.parse_transaction_input(txresponse['input'])

 outputresult=data_parser.parse_receipt_output(inputresult['name'],receipt['outp
ut'])
    return outputresult
#run.py
@app.route('/getBill',methods=['GET','POST'])
def getbi():
    return render_template('getBi.html')

@app.route('/getBill/result',methods=['GET','POST'])
def getbi_re():
    bid=request.form['bid']
    receipt=getbill(contract_address,contract_abi,bid)
```

```
    giver=receipt[0]
    receiver=receipt[1]
    middle=receipt[2]
    Type=receipt[3]
    money=receipt[4]
    createdate=receipt[5]
    repaydate=receipt[6]
    return render_template('getBi_result.html',bid=bid,giver=giver,

receiver=receiver,middle=middle,Type=Type,money=money,
                            createdate=createdate,repaydate=repaydate)
```

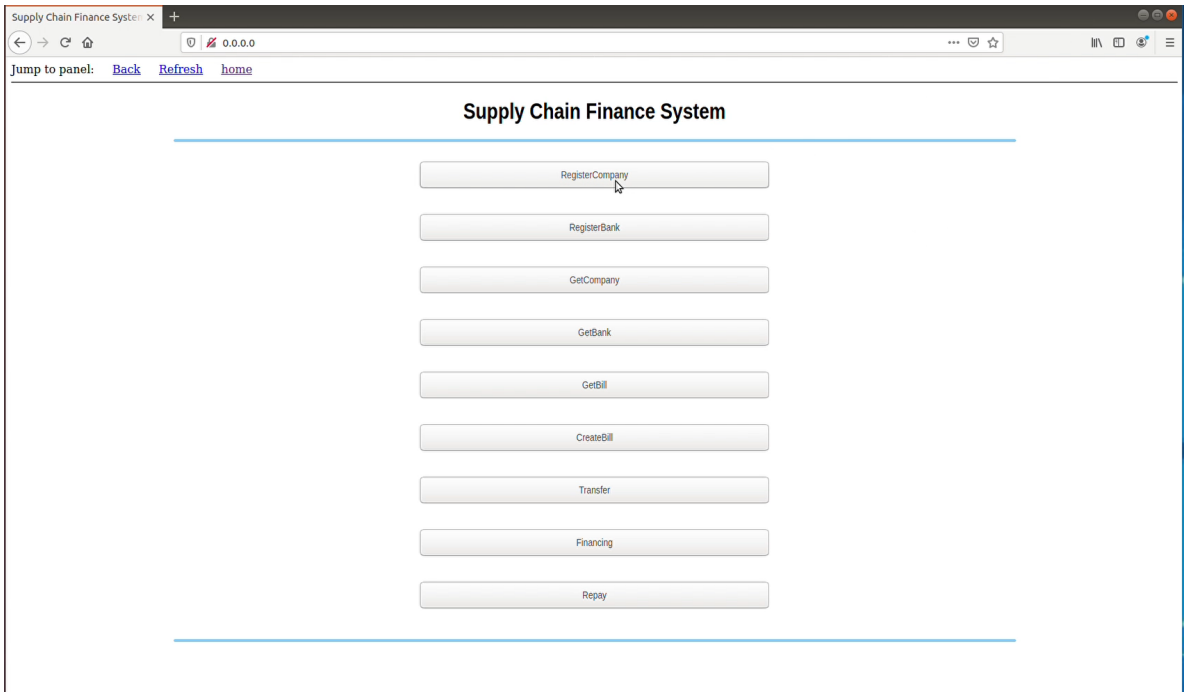
四、功能实现

4.1 启动

1. 启动节点
   `bash nodes/127.0.0.1/start_all.sh`
2. 部署Python SDK
3. 编译并部署合约
   `python3 console.py deploy qukuailian save`
4. 将run.py中的`contract_address`赋值为上一步部署的智能合约的地址
5. `python3 run.py`

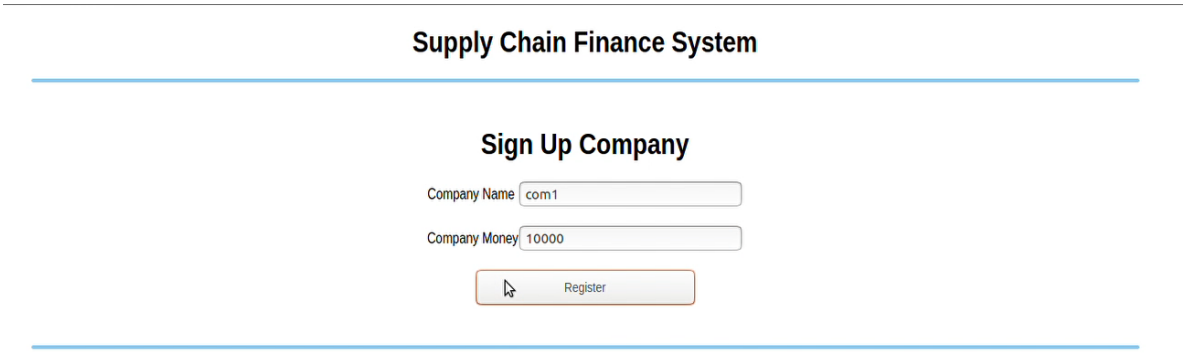
4.5 功能展示

首先，进入一个选择界面，可以看到，一共有九个可以选择的功能按钮，功能分别为：注册公司、注册银行、

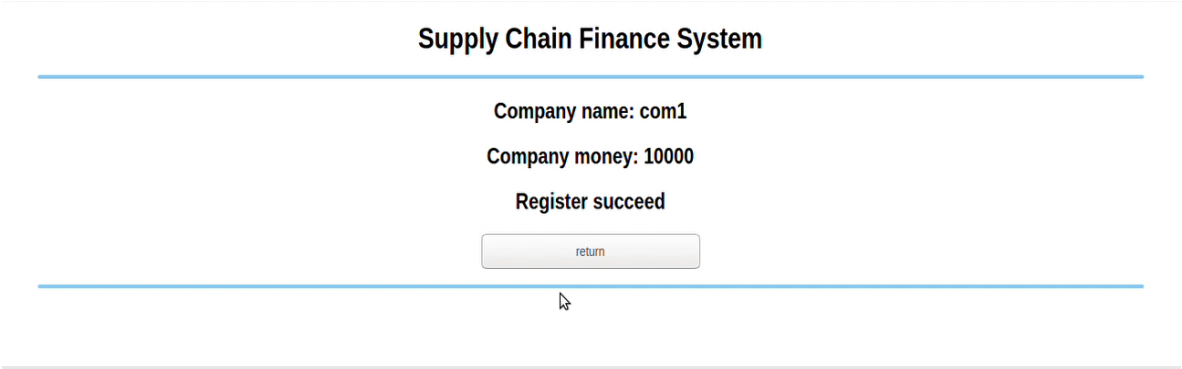


接下来，按照上面的功能按钮的顺序，依次展示功能：

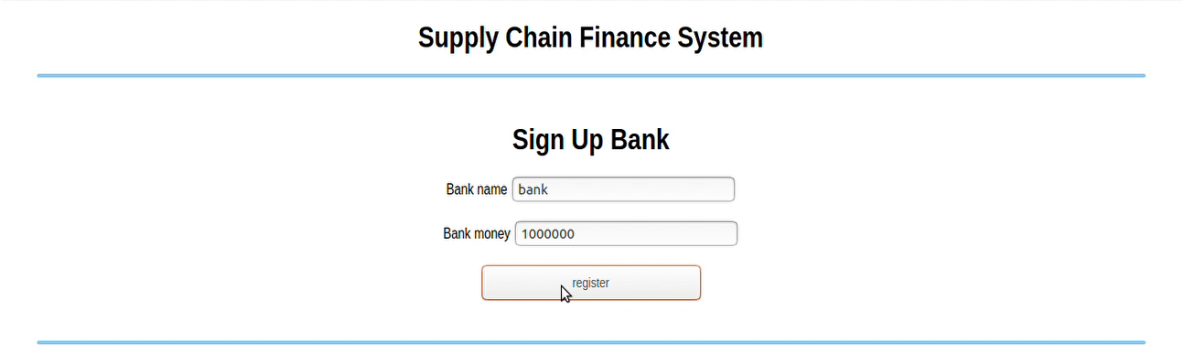
注册公司com1



注册成功！显示注册公司的基本信息：



注册银行bank

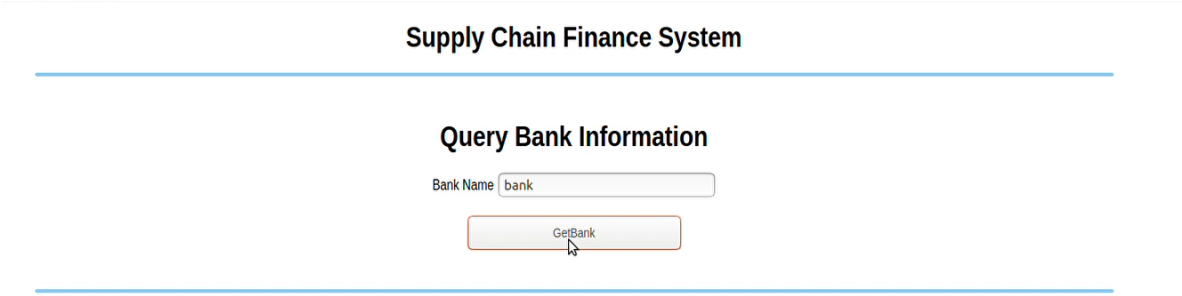


注册成功！显示银行信息：

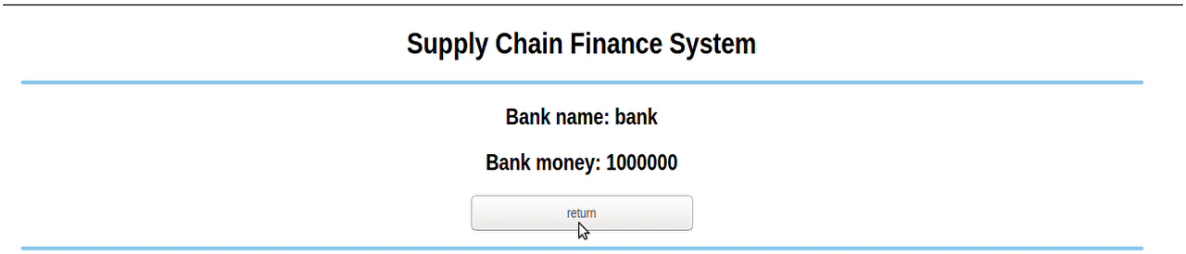
展示公司信息

输入想要查看的公司名称，显示对应的公司的名字和资金。

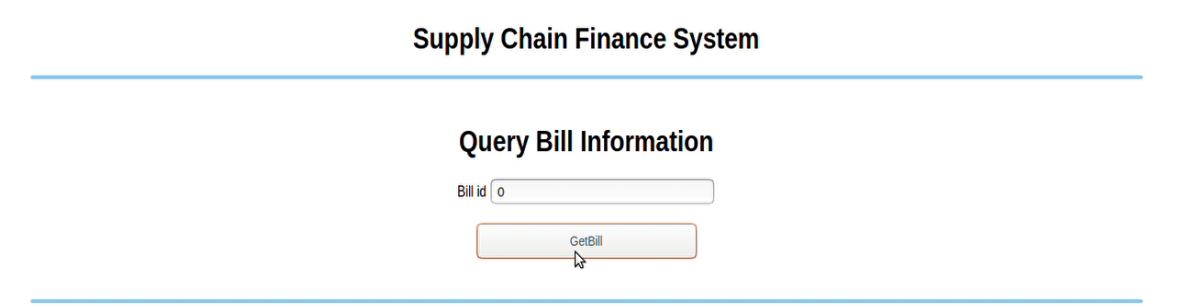
展示银行信息



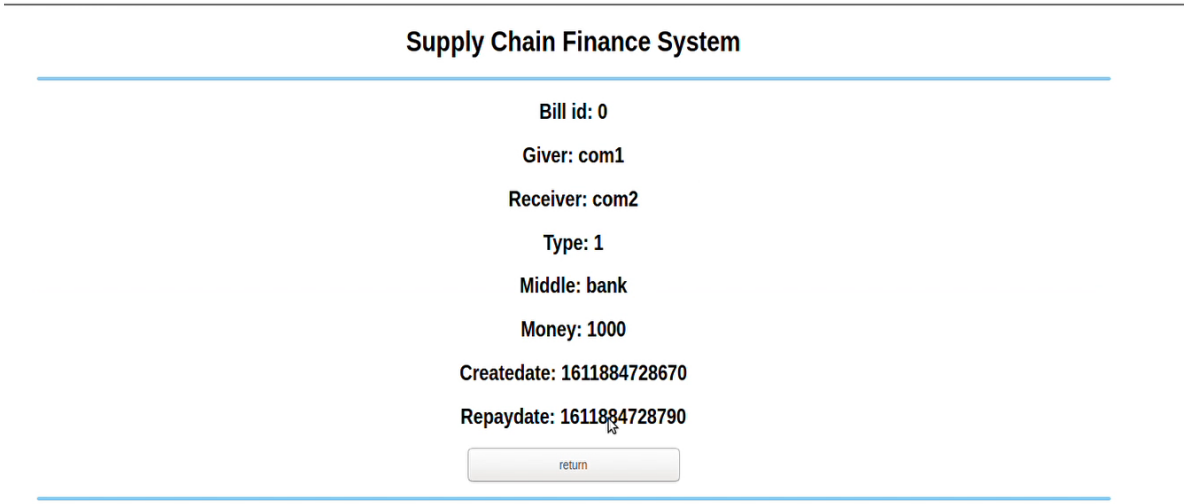
输入想要查看的银行名称，显示对应的银行的名字和资金。



展示收据信息

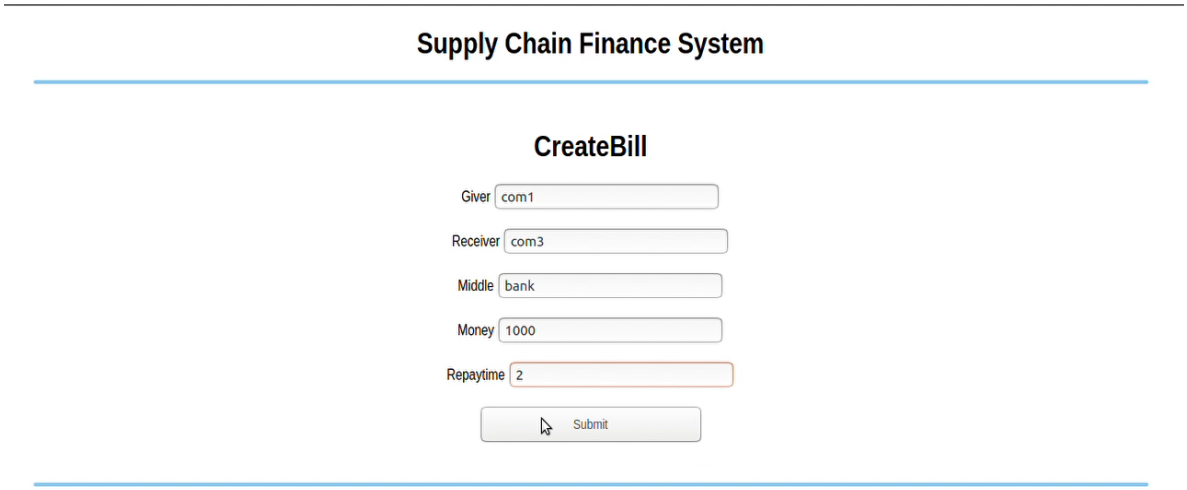


输入想要查看的收据id，显示对应的收据具体信息。

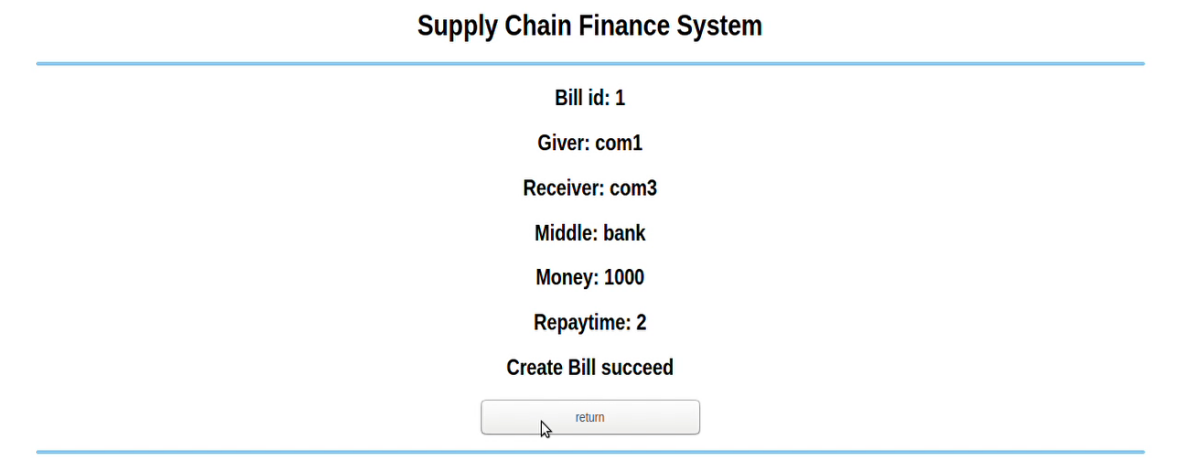


此时，我们具有以下信息：公司com1的资金为10000，公司com2的资金为10000，公司com3的资金为10000，id为0的收据的欠款人com1欠com2金额1000、最长还债时间为2个月。

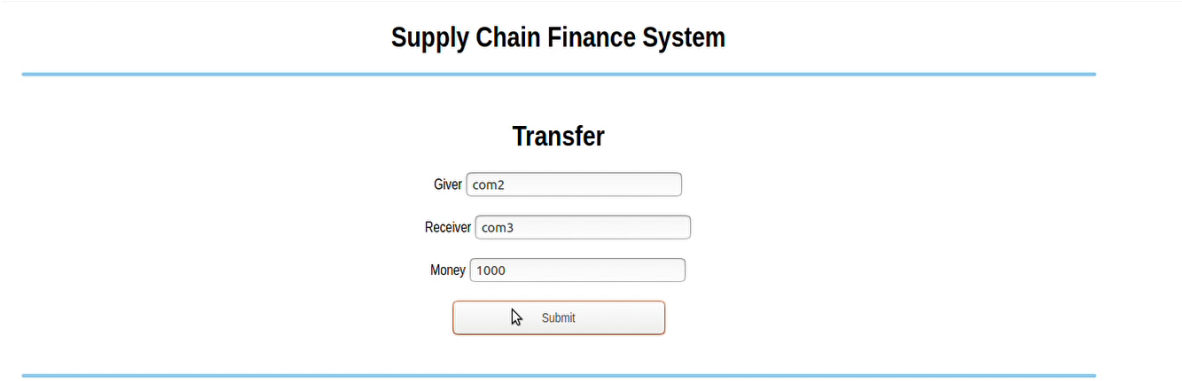
创建收据



输入想要创建的收据的具体信息：com1欠com3金额1000、最长还债时间为2个月。创建收据成功：

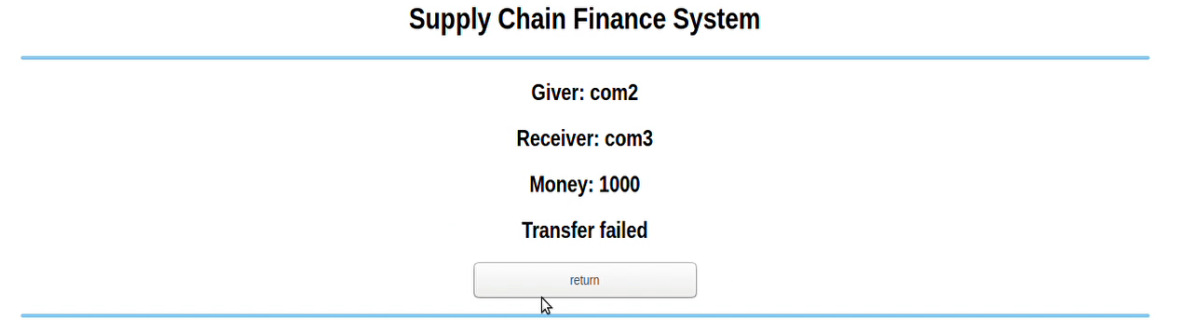


转让

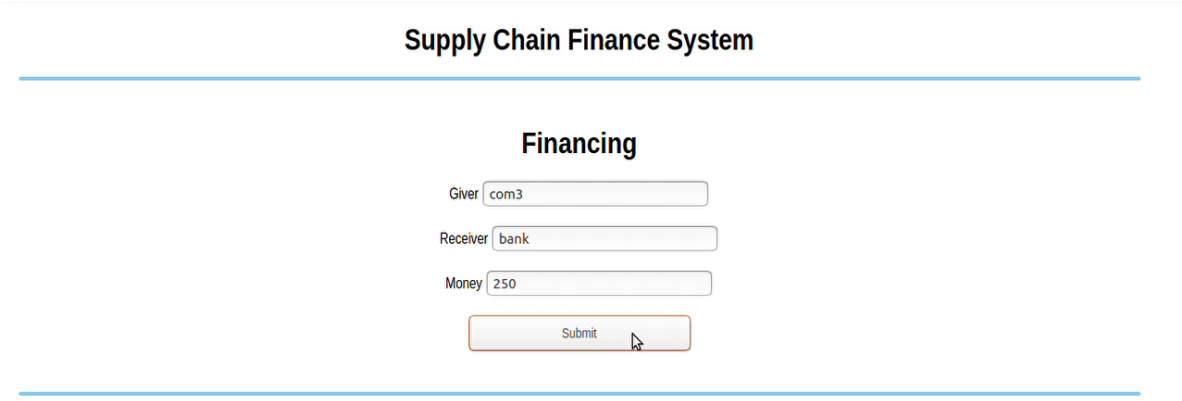


输入转让方com2，被转让方com3，转让金额为1000，此时，由于转让方com2的资金足够，转让成功：

再次进行转让操作时，由于转让方com2的资金不足，转让失败：



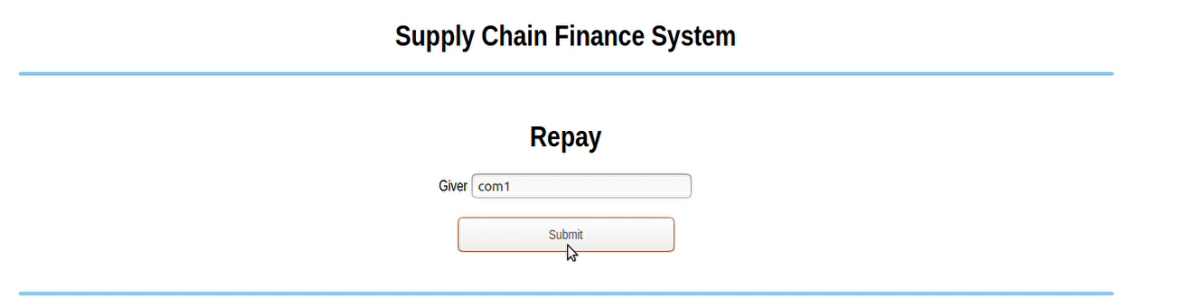
融资



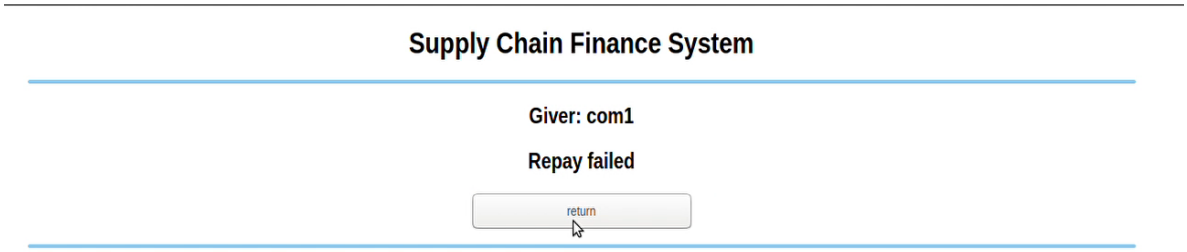
输入申请融资的公司com3和被申请融资的银行bank以及申请的融资金额250，由于此时银行bank有足够的资金，申请融资成功：

此时，由于公司申请的融资金额超过了银行拥有的资金数，申请融资失败：

还款



输入想要还款的公司com1，由于此时未到还款时间，还款失败：



五、心得体会

完成本次大作业后，我对区块链的原理有了更深的理解，对前端、后端和链端的开发工具的使用和开发过程也更加熟悉，对其各自的功能和它们之间的数据交互也有了更直观的认识。

一开始我们对如何处理后端以及其与链端的通信毫无头绪，后来在网上查阅资料后发现了Python-sdk的教程，意识到可以用python-sdk的client模块来进行通信，但在安装环境的过程中遇到了许多问题，例如Python版本不符等等，但最终都通过查询了解解决了。

这次大作业也使我更清晰地认识到了区块链去中心化、可追溯、防篡改的特性在生活中的应用场景的丰富。这次实现的供应链管理的模式同样可以推广到其他领域，比如学历的认证与管理、食品原料的交易与溯源等。联想如今区块链应用的落地情况，可以看出其还有更大的发展空间和发展潜力。

六、加分项

- 添加了额外的功能：获取公司资料、获取银行资料等；
- 设计较为友好的用户界面