

# 中山大学数据科学与计算机学院本科生实验报告

## (2019 年秋季学期)

课程名称： 区块链原理与技术

任课教师： 郑子彬

年级	2017 级	专业（方向）	软件工程
学号	17343162	姓名	郑琳
电话	15013018509	Email	248300727@qq.com
开始日期	2019/11/29	完成日期	2019/12/11

### 一、 项目背景

在最近的发展中，供应链金融逐渐发展成为了中小型企业融资的重要途径。然而，由于在供应链上不同主体之间信息的流通性较差，甚至是不透明的，整个交易的环节十分复杂，往往使得多方信息不对称，甚至形成信息孤岛，这对于交易过程造成了很大的困难，同时也提高了金融机构判断企业融资能力的难度。而区块链的出现，以下几个角度很好地解决了供应链金融现存的几个问题：信息不对称问题，信任传递，提高交易效率。

整个项目主要实现了一个简单的供应链金融平台，完成了传统供应链中应收账款的资产的溯源与流转。利用区块链的不可篡改性和可追溯性等优势，我们可以将整个供应链中的每一笔交易和应收账款单据上传到区块链中，同时再引入第三方可信的机构确认交易信息，确保交易和应收账款单据的真实性。在区块链网络中，整个金融平台支持企业的融资，应收账款的转让，应收账款的清算等等，支持让上游核心企业的信用可以传递到供应链的下游企业，减小了中小企业的融资难度。

## 二、 方案设计

### 1. 开发工具

- 1) 基于 FISCO-BCOS 平台进行在区块链上的部署.
- 2) Python SDK 进行应用程序与区块链的交互, 编程语言为 Python;
- 3) 使用 Python SDK 中的 Channel 和 JSON RPC 协议进行后端与链端通讯;
- 4) 使用 PyQt5 开发前端图形界面.
- 5) 使用 FISCO-BCOS 平台提供的 CRUD 接口进行表格存储.

### 2. 存储设计

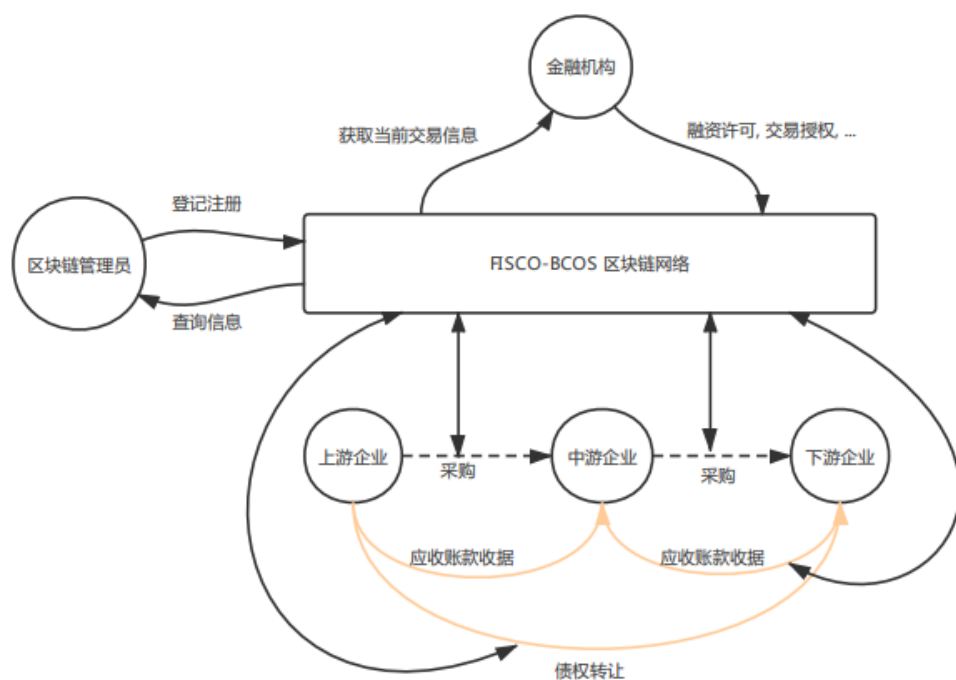
整个供应链的主体主要包含三类:

- 1) 企业. 所有企业之间债权的流动, 交易等组成了整个供应链; 每个企业都有一个自己的外部账户地址, 在链中的标识符是自己的公司名称. (实际上更合理的方式是将各自的地址作为唯一标识符, 但由于其地址太长, 在图形界面中难以完全显示, 这里使用名称作为标识符).
- 2) 金融机构. 金融机构 (包括银行等诸多机构, 在该实验报告中以银行代替) 在该供应链的主要作用是:
  - a) 对不同企业之间的交易信息进行验证, 确认和授权, 如果不满足规范或者企业的信用较差, 可以选择将交易请求打回.
  - b) 允许各个企业进行融资, 融资的金额和条件均要由银行决定.

- 3) 区块链管理员. 所有企业都要先在区块链中注册, 只有注册了的企业才能在区块链中完成资金, 债权的流转. 区块链的管理员拥有一个独有的外部账户地址.

在本次大作业中, 我使用 CRUD 接口对已注册的企业以及收据进行存储. CRUD 接口通过在 Solidity 合约中支持分布式存储预编译合约, 可以实现将 Solidity 合约中数据存储在 FISCO BCOS 平台 AMDB 的表结构中, 实现合约逻辑与数据的分离. 将所有收据和企业信息保存在区块链中的表格中, 通过发送交易进行插入, 修改, 删除, 查询, 也使得安全性更高.

### 3. 数据流图



- 1) 黑色实线箭头表示企业, 银行, 区块链管理员和区块链之间的互动. 企业之间指向区块链的箭头表示将交易信息和交易收据等上链; 而从区块链指向企业等表示各个企业将银行授权或打回的交易记录取回; 银行和区块链管理员也同样与区块链进行类似的通讯.

- 2) 黑色虚线箭头表示供应链中的采购过程, 在这一过程中, 采购中产生的应收账款收据会上传到区块链, 同时处于 “submitted” 状态, 需要等银行授权确认后才生效.
- 3) 黄色实线箭头则表示, 对于从企业 A 到企业 B 的一个应收账款收据, 企业 B 可以选择将该收据的有限额度转让给企业 C, 即这时企业 A 对企业 C 同样有一条应收账款收据. 通过这种方式, 供应链中的中下游企业同样握有上游信用力高的企业的应收账款收据, 它们同样具备了向金融机构融资的条件.

具体来说, 整个系统中, **企业进行的交易, 收据的确认, 债权的转让全都需要银行机构的确认后才能生效**; 银行收到收据或者债权转让请求后, 可以根据企业的信用力, 资质等判断是否授权或者打回.

#### 4. 核心功能介绍

- 1) 合约的编译, 部署以及调用 (整个过程实现在文件 init.py 中);
- 2) 首先开启 FISCO-BCOS 链节点. 这一步在之前的前期准备过程中就已经完成, 只需要启动链即可:

```
bash nodes/127.0.0.1/start_all.sh
```

```
fisco-bcos@fiscobcos-VirtualBox:~$ cd fisco/
fisco-bcos@fiscobcos-VirtualBox:~/fisco$ bash nodes/127.0.0.1/start_all.sh
try to start node0
try to start node1
try to start node2
try to start node3
try to start node4
node0 start successfully
node3 start successfully
node4 start successfully
node2 start successfully
node1 start successfully
fisco-bcos@fiscobcos-VirtualBox:~/fisco$
```

除此之外, 还需要安装 Python-SDK 所需的环境, 环境配置在文档中有着详细介绍, 这里就不再赘述. 文档中同样提供了两个 demo 文件, 为熟悉整个流程和 Python API 提供了很大帮助.

- 3) 接着需要对实现供应链编写好的合约 (合约的代码和功能同样会在下面的功能分析中逐步展开介绍) 进行编译和部署. FISCO-BCOS 平台实现了易用的 Python-SDK 接口调用, 由于已经安装好了 solc 编译器, 这里可以直接调用 API 完成合约的编译:

```
if os.path.isfile(client_config.solc_path) or os.path.isfile(client_config.solcjs_path):  
    Compiler.compile_file("contracts/SupplyCF.sol")
```

编译之后, 会生成 .bin 和 .abi 文件, 供后续调用.

接下来将合约部署到链上. 利用之前生成的 bin 文件, 通过实例化一个 BcosClient()客户端, 可以部署一个合约:

```
client = BcosClient()  
  
print(client.getinfo())  
  
with open("contracts/SupplyCF.bin", 'r') as load_f:  
    contract_bin = load_f.read()  
  
    load_f.close()  
  
result = client.deploy(contract_bin)
```

部署之后会返回一系列值, 其中包括了合约的地址:

```
new address : 0xf74c2c7131c2461db2e67362a8e7fca6dc0a66bf
```

其中 new address 表示新的合约地址.

利用该地址和生成的 abi 文件, 我们就可以调用合约中定义的函数:

```

    receipt =

    client.sendRawTransactionGetReceipt(to_address,contract_abi,"create_company_table")

    print("receipt:",receipt['output'])

```

其中, 调用了 sendRawTransactionGetReceipt 方法来发送一次交易, 调用合约的一个函数; 如果调用的函数需要参数, 可以传入一个列表; 调用之后会返回一个字典变量, 变量中的 'output' 字段就是调用的函数的返回值的十六进制码, 通过对它解析, 可以得到合约中的具体的返回值.

- a) 在大作业中, 我将企业信息和收据信息分别存储在了两个 FISCO-BCOS 内的 AMDB 表格中. 因此, 还需要始化表存储结构. 这一过程是调用了部署的智能合约的两个函数, 而这两个函数又调用了 FISCO-BCOS 中提供的 CRUD 接口, 以在 FISCO-BCOS 平台中的 AMDB 的表结构中存储:

- 后端:

```

receipt = client.sendRawTransactionGetReceipt(to_address,contract_abi,"create_company_table")
receipt = client.sendRawTransactionGetReceipt(to_address,contract_abi,"create_receipt_table")

```

- Solidity 代码:

```

function create_company_table() public returns(int){

    TableFactory tf = TableFactory(0x1001); //The fixed address is 0x1001 for TableFactory

    int count = tf.createTable("company_t", "dummy", "name, address, type");

    emit CreateResult(count);
}

```

```

        return count;
    }

//create table

function create_receipt_table() public returns(int){

    TableFactory tf = TableFactory(0x1001); //The fixed address is 0x1001 for TableFactory

    int count = tf.createTable("receipt_t", "dummy", "from, to, amount,status, due_date");

    emit CreateResult(count);

    return count;
}

```

通过上面的代码, 可以创建出" receipt\_t" 和" company\_t" 两个表格, 分别存储对应的信息. 两个表的结构分别是:

receipt					
dummy	from	to	amount	status	due date

其中, 除了 amount 为 int 类型外, 其余所有字段均为字符串

company			
dummy	name	address	type

所有字段均为字符串(address 为字符串类型的 16 进制码)

这里的 dummy 是未用字段, 充当两个表格的主键, 因为目前 FISCO BCOS 平台的 CRUD 接口还不甚完善, 只能完成相当有限的表格操作, 尤其是在查询时, 只能根据主键查询. 为了解决这个问题, 我将所有记录的主键都设置为同样的值, 这样, 在进行查

询时, 就可以将全部记录取出, 再根据其他条件进行筛选. 下面是两个建表函数执行的结果:

结果:

```
receipt: 0x0000000000000000000000000000000000000000000000000000000000000000
receipt: 0x0000000000000000000000000000000000000000000000000000000000000000
```

为 0 表示成功. 如果这时再创建一次表格, 会提示

```
receipt: 0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff3caf
receipt: 0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff3caf
```

即返回的值不是 0, 表示创建失败, 因为这两个表已存在.

4) 登陆界面, 供区块链管理员, 企业, 银行登录该系统完成区块链上链的相关操作.

登录主要完成的是对登陆者的验证操作:

- a) 如果账号密码分别是 admin, admin, 表示登入区块链管理员账号.
- b) 如果账号密码分别是 bank, 123456, 表示登入银行账号.
- c) 除此之外的情况, 需要核对账号与他对应的密钥(通过密码生成)的正确性, 如果正确则可以成功登入界面, 反之则会提示错误.

● 后端代码:

```
def validate(self):
    name = self.line_acc.text()
    password = self.line_pwd.text()
    if name == 'admin' and password == 'admin':
        manager_window.show()
        # self.close()
    elif name == "bank" and password == "123456":
        bank_window.show()
        bank_window.set_table_content()
        # self.close()
    else:
        keyfile = "{}/{ }.keystore".format(client_config.account_keyfile_
path, name)
        # the account doesn't exists
        if os.path.exists(keyfile) is False:
            QMessageBox.warning(self,
                                "error",
                                "account {} doesn't exists".format(name),
```



```

        QMessageBox.Yes)
    else:
        print("show account : {}, keyfile:{} ,password {} ".format(n
ame, keyfile, password))
        try:
            with open(keyfile, "r") as dump_f:
                keytext = json.load(dump_f)
                privkey = Account.decrypt(keytext, password)
                ac2 = Account.from_key(privkey)
                print("address:\t", ac2.address)
                print("privkey:\t", encode_hex(ac2.key))
                print("pubkey :\t", ac2.publickey)
                company_window.show()
                company_window.set_basic_info(name)
                # self.close()
        except Exception as e:
            QMessageBox.warning(self,
                                "error",
                                ("load account info for [{}] failed,"
                                " error info: {}!").format(name,
                                e),
                                QMessageBox.Yes)

```

##### 5) 区块链管理员 – 列出所有已注册的企业名称, 它可以通过查询表格

company\_t 列出所有已经注册的公司. 它的实现是利用 PyQt5 的信号槽机制,

通过前端的按钮的点击唤醒函数调用, 而该函数进一步调用智能合约中的

select\_registered 函数, 返回所有查询的结果.

##### ● Solidity 代码:

```

function select_registered() public returns (string[], string[],
string[]){
    TableFactory tf = TableFactory(0x1001);
    Table table = tf.openTable("company_t");
    Entries entries = table.select("active",
table.newCondition());
    string[] memory name_list = new
string[](uint256(entries.size()));

```

```

        string[] memory type_list = new
string[](uint256(entries.size()));
        string[] memory addr_list = new
string[](uint256(entries.size()));
        for(int i=0; i<entries.size(); ++i) {
            Entry entry = entries.get(i);
            name_list[uint256(i)] = entry.getString("name");
            type_list[uint256(i)] = entry.getString("type");
            addr_list[uint256(i)] = entry.getString("address");
        }
        return (name_list, type_list, addr_list);
    }
}

```

- 后端代码:

```

def set_table_content(self):
    global client, contract_abi, to_address
    receipt =
client.call(to_address,contract_abi,"select_registered")
    print("receipt:",receipt)
    # info_tuple =
(('zl', 'srh', 'cc',), ('srh', 'cc', 'rjj',), ('0xasdag', '0x135214', '0x123
1',))
    info_tuple = receipt
    info_rows = len(info_tuple[0])
    for i in range(info_rows):
        row = self.table.rowCount()
        self.table.setRowCount(row + 1)

self.table.setItem(row,0,QTableWidgetItem(info_tuple[0][i]))

self.table.setItem(row,1,QTableWidgetItem(info_tuple[1][i]))

self.table.setItem(row,2,QTableWidgetItem(info_tuple[2][i]))

```

6) 区块链管理员 – 企业注册; 依旧是通过信号槽机制, 使得注册时调用一个函数

on\_press\_register. 该函数的作用是:

a) 获取文本框的内容;

- b) 调用 FISCO-BCOS 中 Python-SDK 创建账号的方法, 创建一个新的账号, 生成对应的密钥, 并保存到 keystore 文件中进行账户管理.
- c) 之后, 调用智能合约中的函数 register, 将待注册的公司插入表中.

- Solidity 代码:

```
// register for companies
function register(string _name, string _address, string _type) public returns (int count){
    TableFactory tf = TableFactory(0x1001);
    Table table = tf.openTable("company_t");
    Entry entry_to_insert = table.newEntry();
    entry_to_insert.set("dummy", "active");
    entry_to_insert.set("name", _name);
    entry_to_insert.set("address", _address);
    entry_to_insert.set("type", _type);
    emit InsertResult(count);
    count = table.insert("active", entry_to_insert);
    assets[_name] = 1000;
    if(keccak256(_name) == keccak256("bank")){
        assets[_name] = 1000000;
    }
    return count;
}
```

可以看到在注册时, 管理员为每个企业注册了 1000 作为启动资金.

- 后端代码:

```
def on_press_register(self):
    name, password = self.edit_name.text(), self.edit_pwd.text()
    max_account_len = 240
    if len(name) > max_account_len:
        QMessageBox.warning(self, 'Error', 'The name should not exceed 240 characters!')
        sys.exit(1)
    print("starting : {} {}".format(name, password))
    ac = Account.create(password)
    print("new address :\t", ac.address)
    print("new privkey :\t", encode_hex(ac.key))
    print("new pubkey :\t", ac.publickey)
```

```

kf = Account.encrypt(ac.privateKey, password)
keyfile = "{}/{}.keystore".format(client_config.account_keyfile_path, nam
e)

print("save to file : {}".format(keyfile))
with open(keyfile, "w") as dump_f:
    json.dump(kf, dump_f)
    dump_f.close()
print(">>-----")
print(
    "INFO >> read {} again after new account,address & keys in file:".for
mat(keyfile))
with open(keyfile, "r") as dump_f:
    keytext = json.load(dump_f)
    privkey = Account.decrypt(keytext, password)
    ac2 = Account.from_key(privkey)
    print("address:\t", ac2.address)
    print("privkey:\t", encode_hex(ac2.key))
    print("pubkey :\t", ac2.publickey)
    print("\naccount store in file: {}".format(keyfile))
    print("\n**** please remember your password !!! ****")
    dump_f.close()

global client, contract_abi, to_address
args = [name, ac.address, 'Company']
# res = client.call(str('0x0559deb98ded0b3f2953490e77e4adf4773ac16d'), cont
ract_abi, "register", args)
# print("result :",res)
print(name)
receipt = client.sendRawTransactionGetReceipt(to_address,contract_abi,"reg
ister",args)
print("receipt:",receipt['output'])

QMessageBox.information(self,'Prompt','Register successfully!', QMessageBoxBo
x.Ok)

```

- 7) 企业 - 交易上链. 在交易时, 需要提交的收据信息包括采购方(from), 被采购方(to), 采购额(amount), 状态(status), 以及还款日(due date). 其中, 被采购

方, 采购额, 还款日期需要自己填写. 当点击按钮后, 后端会获取到对应文本框的内容, 接着将这些信息作为参数调用智能合约中的函数 `purchasing` 发送交易, 将交易上链. 之后, 该条交易的状态为 "submitted", 需要等待银行确认.

- Solidity 代码:

```
function purchasing(string _from, string _to, int amt, string d
d) public returns(int){
    if(is_registered(_from) == -1 || is_registered(_to) == -1){
        return -3;
    }
    int count = insert(_from, _to, amt, "submitted", dd);
    if(count == 1){
        return 1;
    }
    else{
        return -1;
    }
}
```

注意到, 对于两家进行交易的企业, 它们都需要经过注册, 否则没有办法进行交易.

- 后端代码:

```
def on_submit_purchase(self):
    _amt = self.line_pur_amt.text()
    _due = self.purchase_date.dateTime().toString("yyyy/MM/dd
hh:mm:ss")
    _to = self.line_pur_to.text()
    global client, contract_abi, to_address
    args = [self.company_name , _to, int(_amt),_due]
    info_tuple = client.sendRawTransactionGetReceipt(to_address, \
        contract_abi, "purchasing", args)
    print("receipt:",info_tuple['output'])
    res = hex_to_signed(info_tuple['output'])
    if res == -3:
        QMessageBox.warning(self, 'Error', 'All companies must be
```

```

registered first!', QMessageBox.Ok)
    elif res == 1:
        QMessageBox.information(self, 'Prompt', 'Purchasing request
submitted successfully.', QMessageBox.Ok)

```

- 8) 企业 - 债权转让. 债权可以转让的条件是: 之前的收据必须已经经过银行授权, 并且转让的额度必须小于原额度. 在该界面中, 会提供一个表格, 通过选中一条记录表示要对该记录进行操作; 之后, 输入转让给的企业名称以及转让额度, 最后点击按钮即可. 点击按钮后, 会调用智能合约中的 transfer 函数, 完成转让.

- Solidity 代码:

```

function transfer (string _from, string _to,string _to_to, int
prev_amt,int _amount, string dd) public returns(int) {
    if(is_registered(_from) == -1 || is_registered(_to) == -1){
        return -3;
    }
    update(_from, _to, prev_amt - _amount,"submitted", dd);
    insert(_from, _to_to, _amount,"submitted", dd);
    return 1;
}

```

- 后端代码:

```

def on_submit_transfer(self):
    global client, contract_abi, to_address
    if self.table_trans_lent.selectionModel().hasSelection():
        row = self.table_trans_lent.currentRow()
        _from = self.table_trans_lent.item(row, 0).text()
        _due = self.table_trans_lent.item(row, 4).text()
        _prev_amt = int(self.table_trans_lent.item(row, 2).text())
        self.line_trans_from.setText(_from)
        _to = self.line_trans_to.text()
        self.transfer_date.setDateTime(QDateTime.fromString(_due, 'yyyy/MM/dd
hh:mm:ss'))
        _amt = int(self.line_trans_amt.text())
        print(_from,_to,_due,_amt)

```

```

        if _amt > _prev_amt:
            QMessageBox.warning(self, 'Error', 'Your transfer amount is too
large!', QMessageBox.Ok)
        else:
            args = [_from, self.company_name, _to, _prev_amt, _amt, _due]
            if self.table_trans_lent.item(row, 3).text() == "authorized":
                info_tuple = client.sendRawTransactionGetReceipt(to_address,
contract_abi, "transfer", args)
                print("receipt:", info_tuple['output'])
                res = hex_to_signed(info_tuple['output'])
                if res == -3:
                    QMessageBox.warning(self, 'Error', 'All companies must be
registered first!', QMessageBox.Ok)
                elif res == 1:
                    QMessageBox.information(self, 'Prompt', 'Transfer
successfully.', QMessageBox.Ok)
                else:
                    QMessageBox.warning(self, 'Error', 'You can only transfer
[Authorized] receipts!', QMessageBox.Ok)
            else:
                QMessageBox.warning(self, 'Prompt', 'Please click to select a record!',
QMessageBox.Ok)

```

9) 企业 - 申请融资. 企业向银行申请融资时, 必须要满足的条件是他的总借款与

总贷款的差大于融资金额. 如果满足, 传入相应的参数, 调用智能合约的

borrow\_money 函数, 提交融资请求.

- Solidity 代码:

```

function borrow_money(string _from, string _to, int _amount, string dd) public
{
    insert(_from, _to, _amount, "submitted", dd);
}

```

- 后端代码:

```

def on_submit_borrowing(self):
    _amt = int(self.line_borr_amt.text())
    _due = self.borrowing_date.dateTime().toString("yyyy/MM/dd hh:mm:ss")
    if _amt > (self.total_lent - self.total_borrowed):
        QMessageBox.warning(self, 'Error', 'You don\'t have enough capacity
to finance.', QMessageBox.Ok)
    else:
        global client, contract_abi, to_address
        args = [self.company_name, "bank", _amt, _due]
        info_tuple = client.sendRawTransactionGetReceipt(to_address,
contract_abi, "borrow_money", args)
        QMessageBox.information(self, 'Prompt', 'Financing successfully.',
QMessageBox.Ok)

```

- 10) 企业 - 收据 (到期) 还款. 企业自己还款更符合实际情况; 因此, 对于每个属于该企业的发出的收据, 企业都必须在还款期内进行还款. 还款主要调用了“ due\_day” 的智能合约的函数

- Solidity 代码:

```

//the receipt on due day
function due_day(string _from, string _to, int _amount, string _expiration)
public returns(int){
    remove(_from, _to, _amount, _expiration);
    assets[_from] -= _amount;
    if(keccak256(_to) != keccak256("bank")){
        assets[_to] += _amount;
    }
}

```

由于 Solidity 中没有直接比较两个字符串的函数, 这里使用了 keccak256(.)函数计算字符串的 hash 值, 然后比较两者的 hash 值以判断是否相同.

- 后端代码:

```

def on_repayment(self):
    global client, contract_abi, to_address

```



```

        if self.table_repay.selectionModel().hasSelection():
            row = self.table_repay.currentRow()
            args = [self.table_repay.item(row, 0).text(),
self.table_repay.item(row, 1).text(), \
                    int(self.table_repay.item(row,
2).text()),self.table_repay.item(row, 4).text())]
            print(args)
            if self.table_repay.item(row, 3).text() == "authorized":
                info_tuple = client.sendRawTransactionGetReceipt(to_address,
contract_abi, "due_day", args)
                print("receipt:",info_tuple)
                QMessageBox.information(self,'Prompt','Repay.',
QMessageBox.Ok)
                self.table_repay.setRowCount(0)
                self.set_table_repay_content(self.company_name)
                #TODO Table refresh
            else:
                QMessageBox.warning(self,'Error','You can only repay
[Authorized] receipts!', QMessageBox.Ok)
            else:
                QMessageBox.warning(self,'Prompt','Please click to select a
record!', QMessageBox.Ok)

```

- 11) 银行 - 查看所有已提交的交易信息. 获取所有交易信息, 可以通过智能合约中 select 函数, 来查询表 receipt\_t 的数据. 要注意的是, 这里只列出所有已提交但未确认的交易记录.

- Solidity 代码:

```

//select records
function select(string company, int mode) public constant
returns(string[], string[], int[], string[], string[]){
    TableFactory tf = TableFactory(0x1001);
    Table table = tf.openTable("receipt_t");
    Condition condition = table.newCondition();
    if(mode == 1){
        condition.EQ("from", company);
    }
    else if(mode == 0){
        condition.EQ("to", company);
    }
}

```

```

else if(mode == 2){
    condition.EQ("status", "submitted");
}
Entries entries = table.select("active", condition);
return extract_info(entries);
}

```

- 后端代码:

```

def set_table_content(self):
    global client, contract_abi, to_address
    info_tuple = client.call(to_address, contract_abi, "select", ["", 2])
    print("receipt:", info_tuple)
    info_rows = len(info_tuple[0])
    for i in range(info_rows):
        row = self.table.rowCount()
        self.table.setRowCount(row + 1)
        self.table.setItem(row, 0, QTableWidgetItem(info_tuple[0][i]))
        self.table.setItem(row, 1, QTableWidgetItem(info_tuple[1][i]))
        self.table.setItem(row, 2, QTableWidgetItem(str(info_tuple[2][i])))
        self.table.setItem(row, 3, QTableWidgetItem(info_tuple[3][i]))
        self.table.setItem(row, 4, QTableWidgetItem(info_tuple[4][i]))

```

- 12) 银行 - 银行授权交易, 债权转让等交易请求. 对于选中的一条交易, 如果银行机构想要对他授权, 则调用智能合约中的 update 函数, 将其状态更改为" authorized" .

- Solidity 代码:

```

//update records
function update(string _from, string _to, int amt, string sta, string
dd) public returns(int) {
    TableFactory tf = TableFactory(0x1001);
    Table table = tf.openTable("receipt_t");
    Entry entry = table.newEntry();
    entry.set("dummy", "active");
    entry.set("from", _from);
    entry.set("to", _to);
    entry.set("amount", amt);
    entry.set("status", sta);
}

```

```

        entry.set("due_date", dd);
        Condition condition = table.newCondition();
        condition.EQ("from", _from);
        condition.EQ("to", _to);
        if(keccak256(sta) == keccak256("authorized") && keccak256(_to) ==
keccak256("bank")){
            assets[_from] += amt;
        }
        int count = table.update("active", entry, condition);
        emit UpdateResult(count);
        return count;
    }
}

```

- 后端代码:

```

def on_authorize(self):
    global client, contract_abi, to_address
    if self.table.selectionModel().hasSelection():
        row = self.table.currentRow()
        args = [self.table.item(row, 0).text(), self.table.item(row, 1).text(), \
            int(self.table.item(row, 2).text()), "authorized", self.table.item(row,
4).text()]
        print(args)
        info_tuple = client.sendRawTransactionGetReceipt(to_address, contract_abi,
"update", args)
        print("receipt:", info_tuple)
        QMessageBox.information(self, 'Prompt', 'Authorize successfully!',
QMessageBox.Ok)
        self.table.setRowCount(0)
        self.set_table_content()
    else:
        QMessageBox.warning(self, 'Prompt', 'Please click to select a record!',
QMessageBox.Ok)

```

13) 银行 - 拒绝交易记录, 将收据打回, 取消该次交易. 这时, 银行机构调用智能合约中的 remove 函数, 删除这条记录.

- Solidity 代码:

```

function remove(string _from, string _to, int amt, string dd) public
returns(int){

```

```

    TableFactory tf = TableFactory(0x1001);
    Table table = tf.openTable("receipt_t");
    Condition condition = table.newCondition();
    condition.EQ("from", _from);
    condition.EQ("to", _to);
    condition.EQ("amount", amt);
    condition.EQ("due_date", dd);

    int count = table.remove("active", condition);
    emit RemoveResult(count);

    //update the total money borrowed from others and money lent to others.
    return count;
}

```

- 后端代码:

```

def on_reject(self):
    global client, contract_abi, to_address
    if self.table.selectionModel().hasSelection():
        row = self.table.currentRow()
        args = [self.table.item(row, 0).text(), self.table.item(row,
1).text(), \
                int(self.table.item(row, 2).text()), self.table.item(row,
4).text()]]
        print(args)
        info_tuple = client.sendRawTransactionGetReceipt(to_address,
contract_abi, "remove", args)
        print("receipt:", info_tuple)
        QMessageBox.information(self, 'Prompt', 'Reject.', QMessageBox.Ok)
        self.table.setRowCount(0)
        self.set_table_content()
        #TODO Table refresh
    else:
        QMessageBox.warning(self, 'Prompt', 'Please click to select a
record!', QMessageBox.Ok)

```

上面的代码只是完成这些核心功能的调用函数代码，对于 CRUD 接口我还进

行了进一步的封装，以便更好地适配后端和前端代码，具体可见源文件。

## 5. 开发环境配置

- 1) 首先, 确认 FISCO-BCOS 链已经正常启动;
- 2) 按照文档安装好 FISCO-BCOS 的 Python-SDK 环境, 然后进入文件夹  
`python-sdk`;
- 3) 将 github 中的 `gui` 文件夹, `main.py` 以及 `init.py` 分别复制到该文件夹中,  
并将 `contracts` 文件夹中的 `SupplyCF.sol`, `Table.sol` 文件放在 `python-sdk/contracts/`中;
- 4) 执行 `init.py`, 进行智能合约的编译, 部署, 以及初始表格的创建;
- 5) 在 `python-sdk/bin/contract.ini` 中获取刚才部署的智能合约的地址, 填入 `main.py` 中的 `to_address` 处, 然后执行 `main.py`, 即可启动该应用程序.

## 三、 功能测试

整个功能测试部分包含三家公司, 分别是 `test1`, `test2`, `test3`. 进行登记之后:

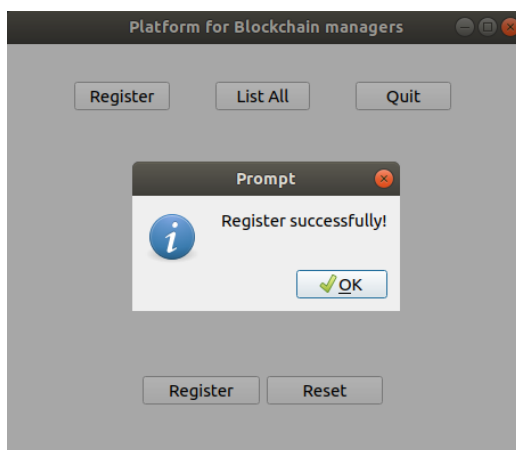
- `test1` 向 `test2` 采购 300 的物资;
- `test2` 向 `test3` 采购 400 的物资;
- 银行对上述的两笔交易进行确认;
- `test2` 将自己握有的 `test1` 的债权其中的 200 转让给 `test3`;

- test3 向银行申请融资, 金额为 200;
- 银行进行授权确认;
- test3 向银行申请融资, 金额为 300;
- 银行拒绝, 将提交的融资请求打回;
- test1, test2, test3 分别还款;
- 结束.

整个测试环节包括了上述所有的核心功能.

1. 对于区块链管理员, 其登录账号和密码均为 admin:

下面注册一个企业 test1, 以及对应的密码:



提示注册成功, 同时可以看到账号, 密码以及产生的密钥已存储在了 keystore

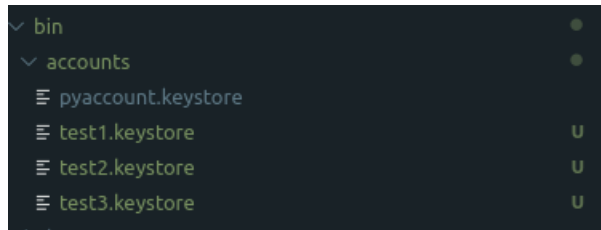
管理中:

```
>>-----
INFO >> read [bin/accounts/test1.keystore] again after new account,address & keys in file:
address:      0xb01dDC111cbBb9D59Ad8478c6362004815Fa7dce
privkey:      0x5745f472908b4116def4cb0d22df2df084cd29c5ac51004d8ec8a21b570774b9
pubkey :      0x4a1a397cf4e625d3f07da1dbf11f37d6f0444f2828a826b065b1276b7785975bd6019dd1af85e0f4c
891ff293afc7005591f6bed32a125b8ea7231bc59fe2d69

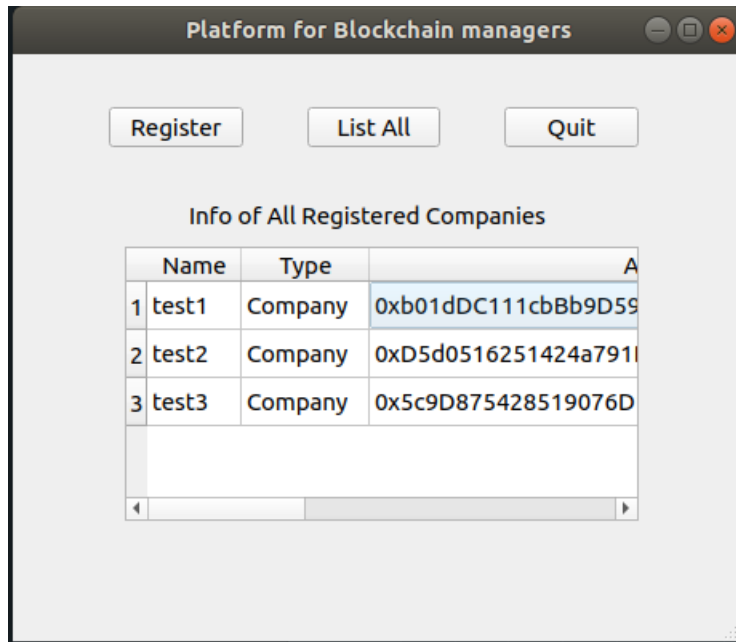
account store in file: [bin/accounts/test1.keystore]

**** please remember your password !!! ****
```

为方便后续的操作, 再注册两个账号 test2, test3. 可以看到 accounts 目录下多了三个密钥存储文件:



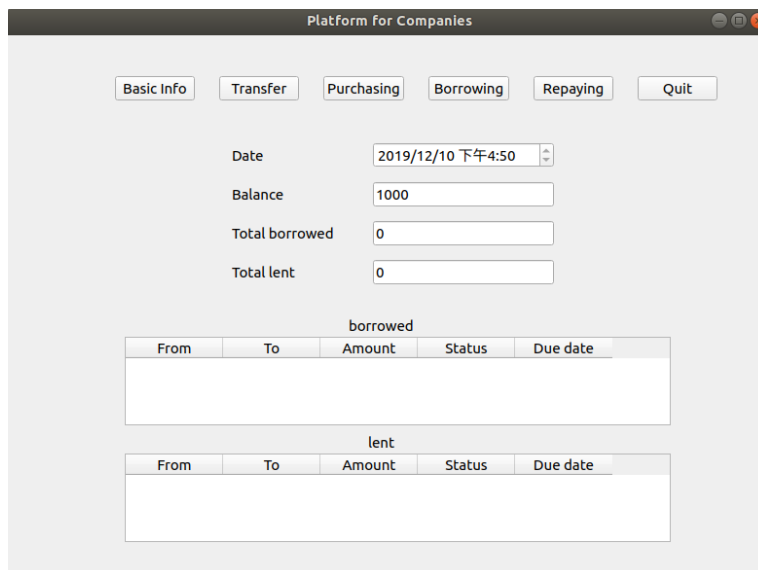
在图形界面中查询已注册的企业信息:



其中, 最后一列代表每个企业的公钥地址.

2. 在登陆界面中, 输入刚才注册的企业名称和密码, 即可进入公司界面, 这里

以 test1 为例:



如图是各个企业的图形界面, 包括五个面板:

- 基本信息, 包括当前日期, 余额, 总借款和总贷款, 下面的两个表格分别记录了所有借款记录和贷款记录. 对于每个企业, 其都有初始化的 1000 作为启动资金.

接下来对具体的交易作测试. 首先, 公司 test1 向 test2 采购 300:

The screenshot shows a window titled "Platform for Companies" with a menu bar containing "Basic Info", "Transfer", "Purchasing", "Borrowing", "Repaying", and "Quit". The "Purchasing Form" is active, displaying fields for "To" (test2), "Amount" (300), and "Due Date" (2020/12/10 下午9:04). There are "Submit" and "Reset" buttons at the bottom.

点击提交, 回到基本信息栏, 发现表格中有了新纪录:

The screenshot shows the "Basic Info" panel of the "Platform for Companies" interface. It displays the current date (2019/12/10 下午9:13), balance (1000), total borrowed (300), and total lent (0). Below these are two tables: "borrowed" and "lent". The "borrowed" table has one record showing a transaction from test1 to test2 for 300, with status "submitted" and due date "2020/12/1...". The "lent" table is currently empty.

From	To	Amount	Status	Due date
1 test1	test2	300	submitted	2020/12/1...

From	To	Amount	Status	Due date
------	----	--------	--------	----------

按照类似的方法, 企业 test2 向 test3 采购 400:



Platform for Companies

Basic Info Transfer Purchasing Borrowing Repaying Quit

Purchasing Form

To test3

Amount 400

Due Date 2020/12/10 下午9:13

Submit Reset

提交后, 在企业 test2 的基本信息中, 可以看到借贷情况:

Platform for Companies

Basic Info Transfer Purchasing Borrowing Repaying Quit

Date 2019/12/10 下午9:14

Balance 1000

Total borrowed 400

Total lent 300

borrowed

	From	To	Amount	Status	Due date
1	test2	test3	400	submitted	2020/12/1...

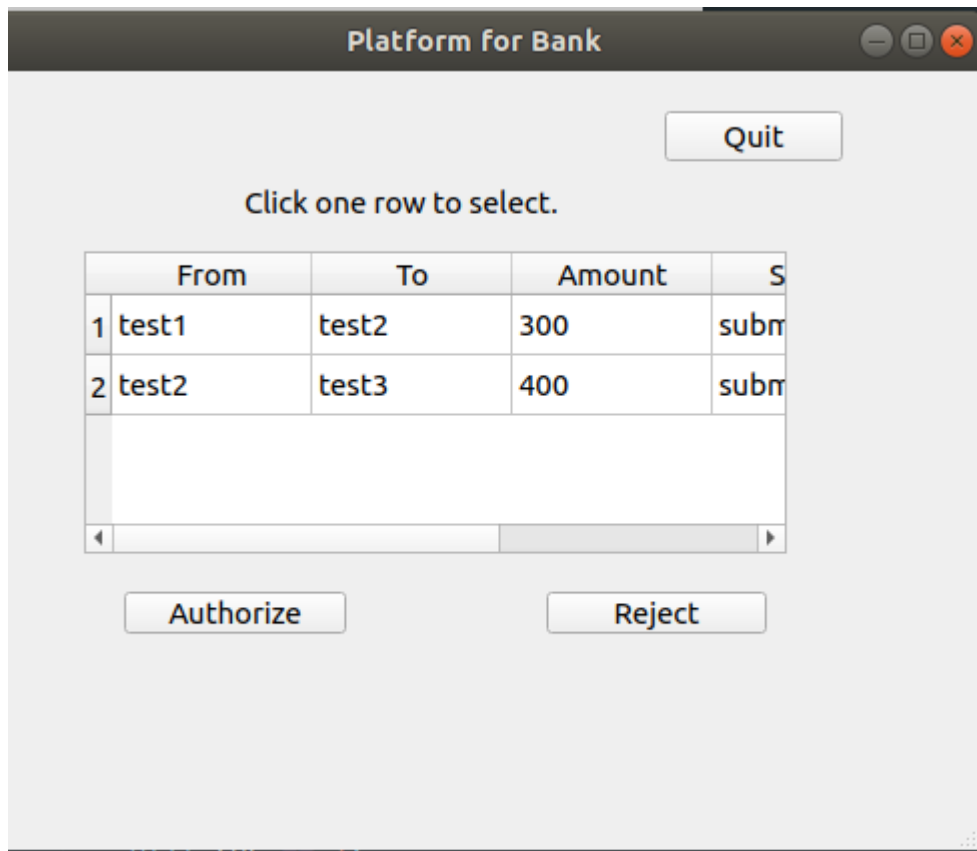
lent

	From	To	Amount	Status	Due date
1	test1	test2	300	submitted	2020/12/1...

可以看到总贷款, 总借款, 以及借贷记录都有更新.

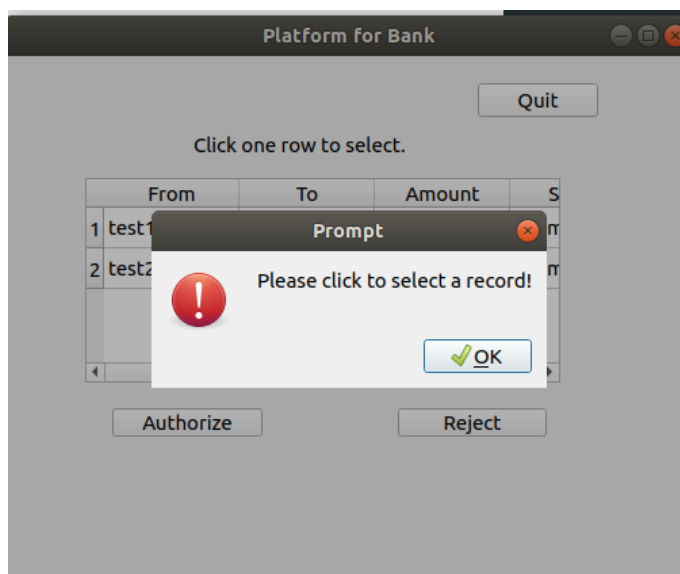
3. 接下来, 登录银行账号, 授权这两笔交易:

登录成功后, 界面如下所示:

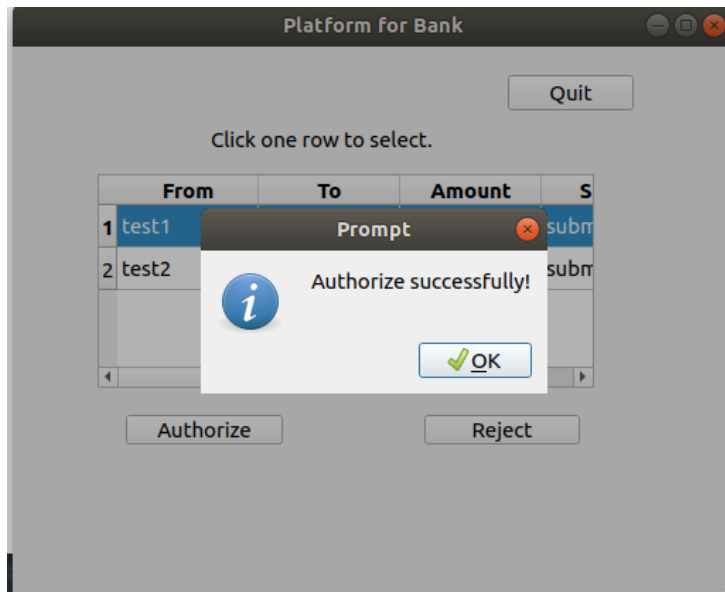


向右拖动滑动栏可以看到整个信息. 选中一行后, 即可进行授权或者打回操作.

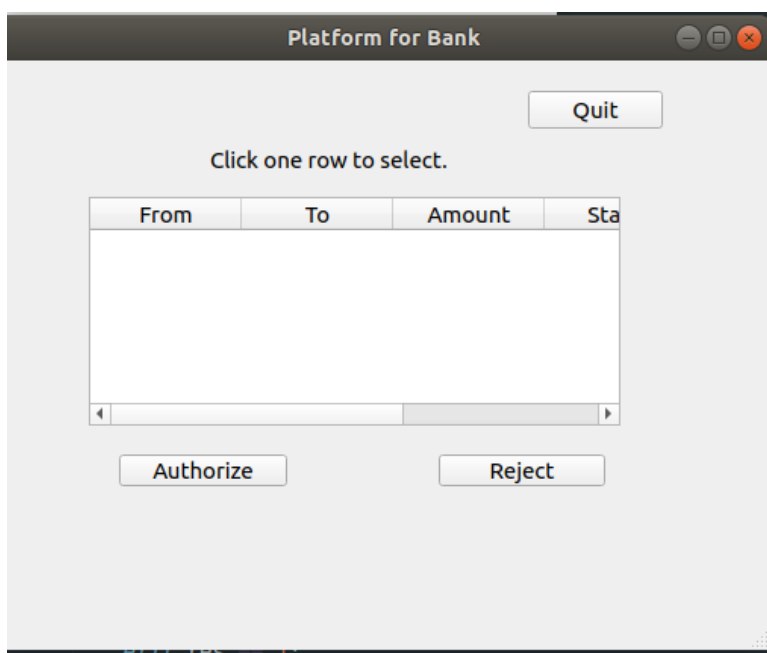
如果直接点击按钮, 会提示首先需要选中一行:



选中第一行, 进行授权:



接着对第二行数据也进行授权, 成功后可以看到, 此时银行已经没有记录需要处理:



4. 接着返回 test2 企业用户, 进行债权转让:

首先可以看到所有交易记录均已授权:

Platform for Companies

Basic Info Transfer Purchasing Borrowing Repaying Quit

Date 2019/12/10 下午9:21

Balance 1000

Total borrowed 400

Total lent 300

borrowed

	From	To	Amount	Status	Due date
1	test2	test3	400	authorized	2020/12/1...

lent

	From	To	Amount	Status	Due date
1	test1	test2	300	authorized	2020/12/1...

之后, 点击 Transfer 按钮, 转到转让页面:

Platform for Companies

Basic Info Transfer Purchasing Borrowing Repaying Quit

Lent List

	From	To	Amount	Status
1	test1	test2	300	authorized

From

To

Amount

Due Date 2019/12/10 下午9:22

Submit Reset

可以看到, 有一条记录可以进行转让. 选中该行记录, 填写相应字段, 将其中的 200 金额转让给 test3:

Platform for Companies

Basic Info Transfer Purchasing Borrowing Repaying Quit

Lent List

	From	To	Amount	Stat
1	test1	test2	300	authoriz

From: test1

To: test3

Amount: 200

Due Date: 2019/12/10 下午9:22

Submit Reset

点击提交, 这时转到 test1 企业用户, 其交易记录如下所示:

Platform for Companies

Basic Info Transfer Purchasing Borrowing Repaying Quit

Date: 2019/12/10 下午9:25

Balance: 1000

Total borrowed: 300

Total lent: 0

borrowed

	From	To	Amount	Status	Due date
1	test1	test2	100	submitted	2020/12/1...
2	test1	test3	200	submitted	2020/12/1...

lent

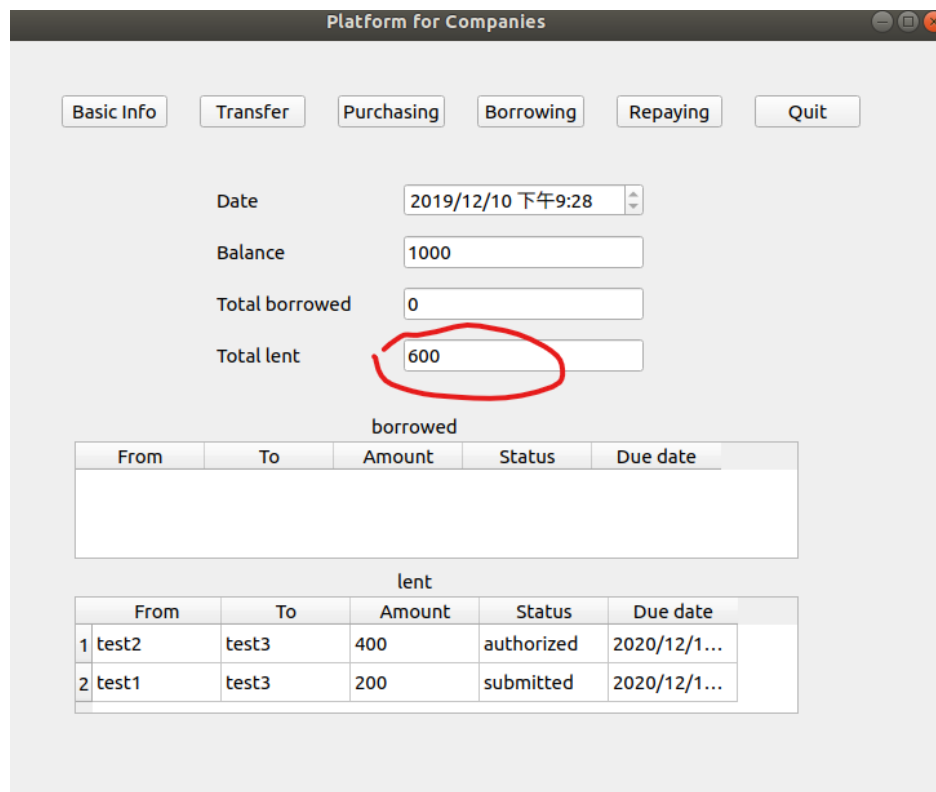
	From	To	Amount	Status	Due date
--	------	----	--------	--------	----------

Test1 拥有两条收据, 欠 test2 企业 100, 欠 test3 企业 200, 验证了转让的成功.

5. 接下来, 验证融资功能. 拥有了一大笔欠收款的 test3 企业, 尤其是拥有上游企

业 test1 的债权, 此时可以胸有成竹地向银行申请融资.

首先查看 test3 的基本信息:



Platform for Companies

Basic Info Transfer Purchasing Borrowing Repaying Quit

Date: 2019/12/10 下午9:28

Balance: 1000

Total borrowed: 0

Total lent: 600

borrowed

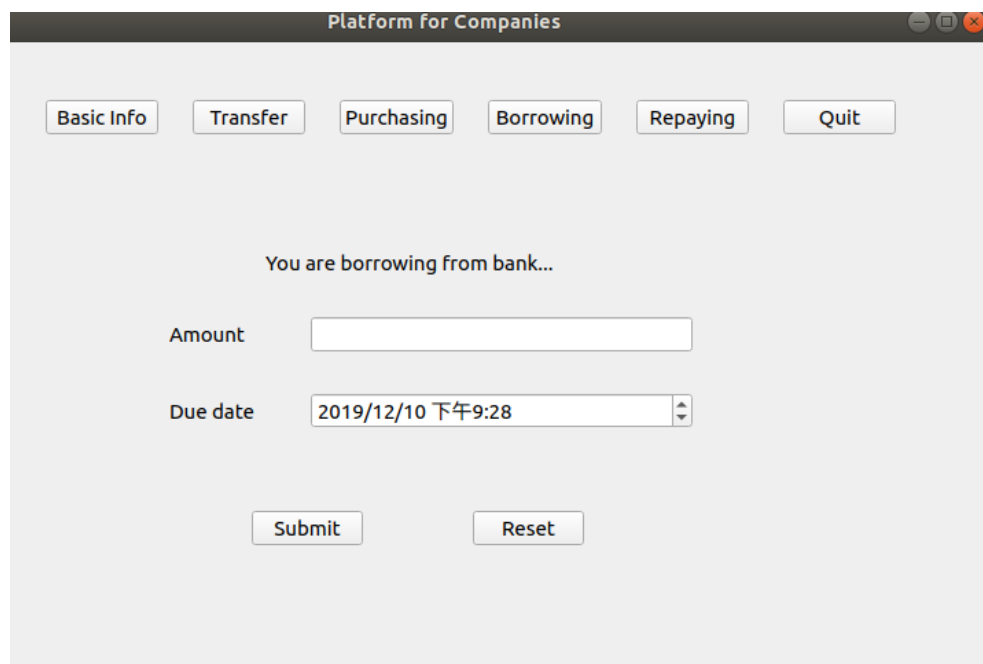
	From	To	Amount	Status	Due date
--	------	----	--------	--------	----------

lent

	From	To	Amount	Status	Due date
1	test2	test3	400	authorized	2020/12/1...
2	test1	test3	200	submitted	2020/12/1...

注意到它总贷款为 600, 而总借款为 0, 表示它最多可以向银行融资 600.

点开 Borrowing 页面:



Platform for Companies

Basic Info Transfer Purchasing Borrowing Repaying Quit

You are borrowing from bank...

Amount:

Due date: 2019/12/10 下午9:28

Submit Reset

首先申请 300 融资, 在明年的这个时候还款:

Platform for Companies

Info Transfer Purchasing Borrowing Repaying Quit

You are borrowing from bank...

Amount

Due date

Submit Reset

提交成功之后, 其基本信息就包含了向银行的借款:

Platform for Companies

Basic Info Transfer Purchasing Borrowing Repaying Quit

Date

Balance

Total borrowed

Total lent

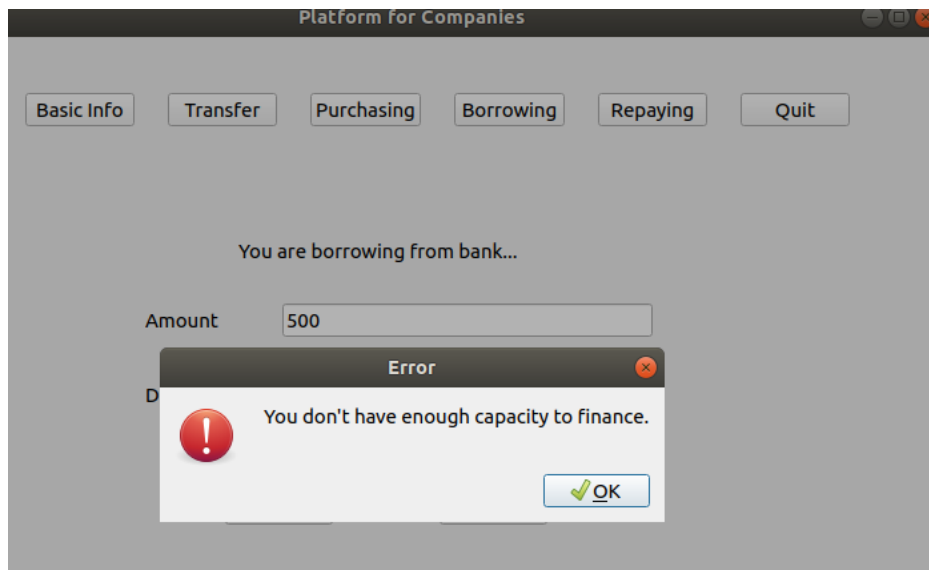
borrowed

	From	To	Amount	Status	Due date
1	test3	bank	300	submitted	2020/12/1...

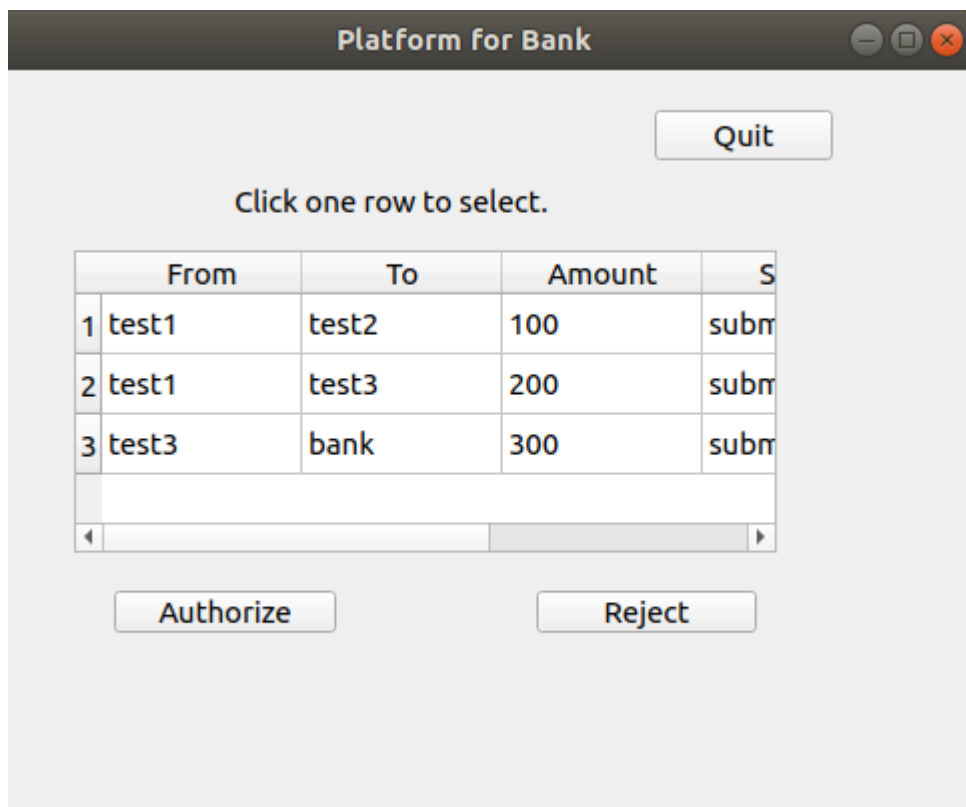
lent

	From	To	Amount	Status	Due date
1	test2	test3	400	authorized	2020/12/1...
2	test1	test3	200	submitted	2020/12/1...

如果这时再借 500, 会提示无法融资, 因为  $500 > 600 - 300$ , 不满足申请条件:



6. 此时，银行继续将所有交易记录授权确认：



授权完毕后，需要注意的是由于 test3 向银行申请融资并得到了同意，它的资产会增加，如下图所示：



Platform for Companies

Basic Info Transfer Purchasing Borrowing Repaying Quit

Date 2019/12/10 下午9:34

Balance 1300

Total borrowed 300

Total lent 600

borrowed

	From	To	Amount	Status	Due date
1	test3	bank	300	authorized	2020/12/1...

lent

	From	To	Amount	Status	Due date
1	test2	test3	400	authorized	2020/12/1...
2	test1	test3	200	authorized	2020/12/1...

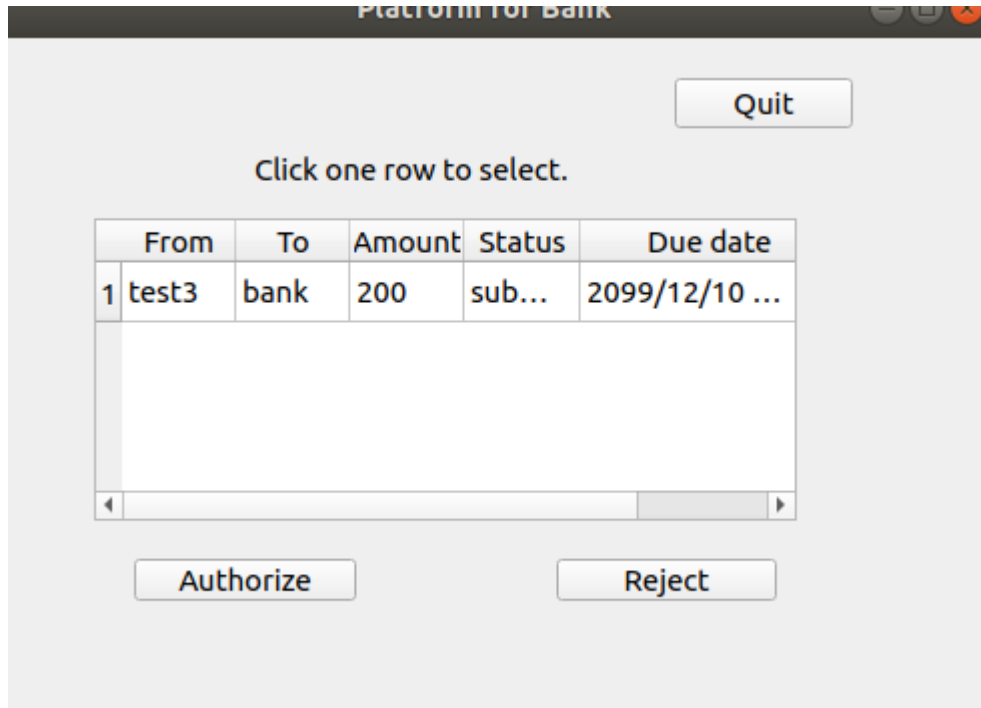
初始资产 1000 加上融资额 300, 就得到了现在的总金额.

7. 下面测试银行对交易记录的打回功能. 有一天, 企业 test3 的操作者喝高了, 向银行申请融资 200, 但还款年份是 2099 年! 虽然满足申请条件, 但银行显然不会满足他提出的无理要求.

当 test3 提交申请后, 它的借款记录如下所示:

borrowed					
	From	To	Amount	Status	Due date
1	test3	bank	300	authorized	2020/12/1...
2	test3	bank	200	submitted	2099/12/1...

银行用户收到了这条记录之后:



银行毫不留情将它驳回。选中该行，点击驳回按钮，将该交易取消。这样，test3 中这条荒谬的记录就被驳回了：

borrowed					
	From	To	Amount	Status	Due date
1	test3	bank	300	authorized	2020/12/1...

驳回功能验证完成。

#### 8. 最后，验证企业的还款功能。

以企业 test3 为例，打开还款界面，选中要还款的记录，进行还款：

Basic Info
Transfer
Purchasing
Borrowing
Repaying
Quit

Click a record to select and repay!

	From	To	Amount	Status	Due date
1	test3	bank	300	authorized	2020/12/1...

OK

选中该记录, 进行还款:

Click a record to select and repay!

	From	To	Amount	Status	Due date
1	test3	bank	300	authorized	2020/12/1...

Prompt

Repay.

OK

OK

还款前:

Balance

1300

Total borrowed

300

Total lent

600

borrowed

	From	To	Amount	Status	Due date
1	test3	bank	300	authorized	2020/12/1...

还款后:

Balance	<input type="text" value="1000"/>
Total borrowed	<input type="text" value="0"/>
Total lent	<input type="text" value="600"/>

borrowed				
From	To	Amount	Status	Due date

可以看到, 由于还款成功, test3 的借款记录已经清除, 而总资产则扣除了 300.

至此, 全部功能都测试完毕.

#### 四、 界面展示

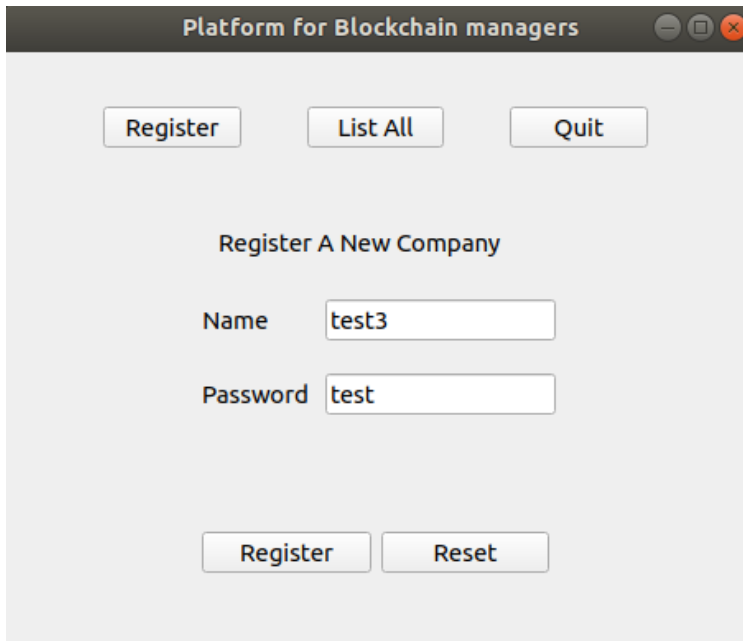
##### 1. 登录界面

##### 2. 区块链管理员界面

	Name	Type	Address
1	test1	Company	0x1faFE09...
2	test2	Company	0x5E4FFE8...
3	test3	Company	0xd26bfd1...

右图为登录后的界面, 可以点击上方的按钮进行企业注册, 查询企业信息等操作.

- 企业注册:



Platform for Blockchain managers

Register List All Quit

Register A New Company

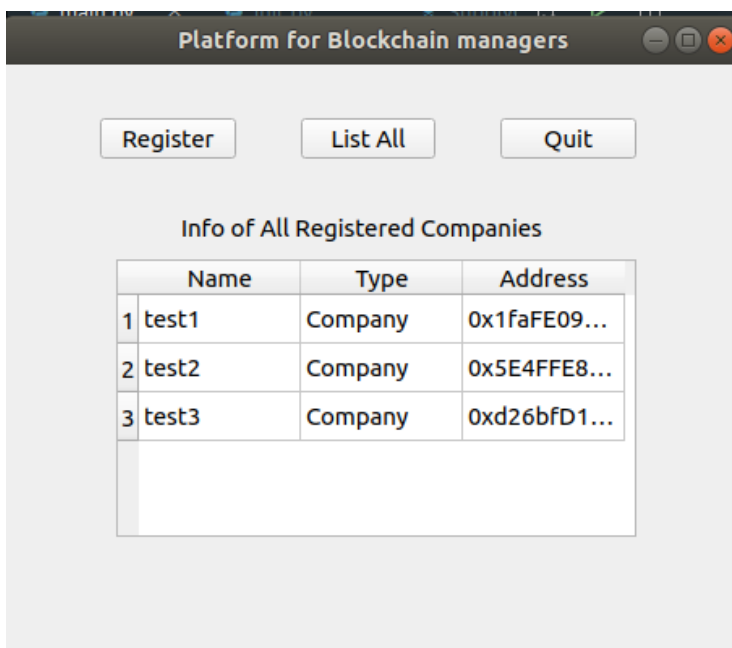
Name test3

Password test

Register Reset

如图, 需要填写企业的名称和密码, 之后会为该企业生成对应的密钥, 以供进入区块链系统.

- 查询界面



Platform for Blockchain managers

Register List All Quit

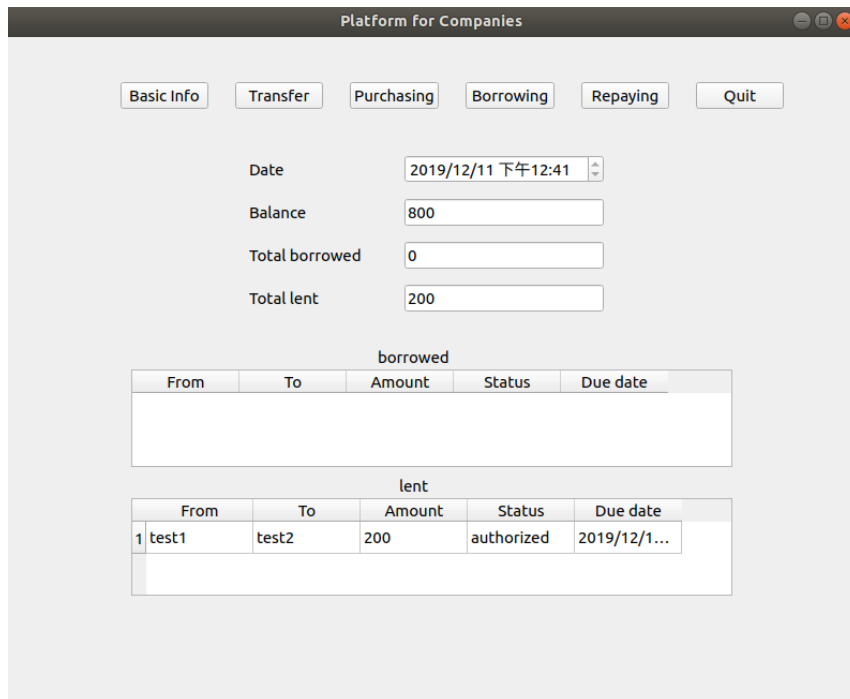
Info of All Registered Companies

	Name	Type	Address
1	test1	Company	0x1faFE09...
2	test2	Company	0x5E4FFE8...
3	test3	Company	0xd26bfD1...

可以查询当前所有注册的企业的名称, 类型以及区块链的账户地址.

### 3. 企业界面

- 在基本信息页面, 包含当前日期, 资产, 总借款, 总贷款, 借款记录和贷款记录;



Platform for Companies

Basic Info Transfer Purchasing Borrowing Repaying Quit

Date: 2019/12/11 下午12:41

Balance: 800

Total borrowed: 0

Total lent: 200

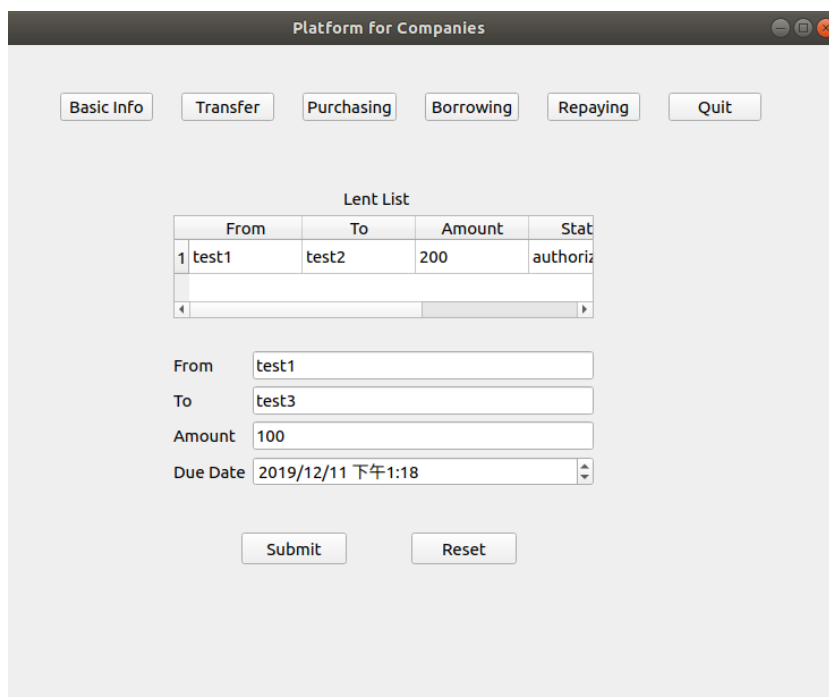
**borrowed**

From	To	Amount	Status	Due date
------	----	--------	--------	----------

**lent**

From	To	Amount	Status	Due date
1 test1	test2	200	authorized	2019/12/1...

- 在债权转让界面, 上方显示了借款记录, 通过选中一条记录, 可以对这条收据信息进行转让; 同时下方填写相应的信息即可.



Platform for Companies

Basic Info Transfer Purchasing Borrowing Repaying Quit

**Lent List**

	From	To	Amount	Stat
1	test1	test2	200	authoriz

From: test1

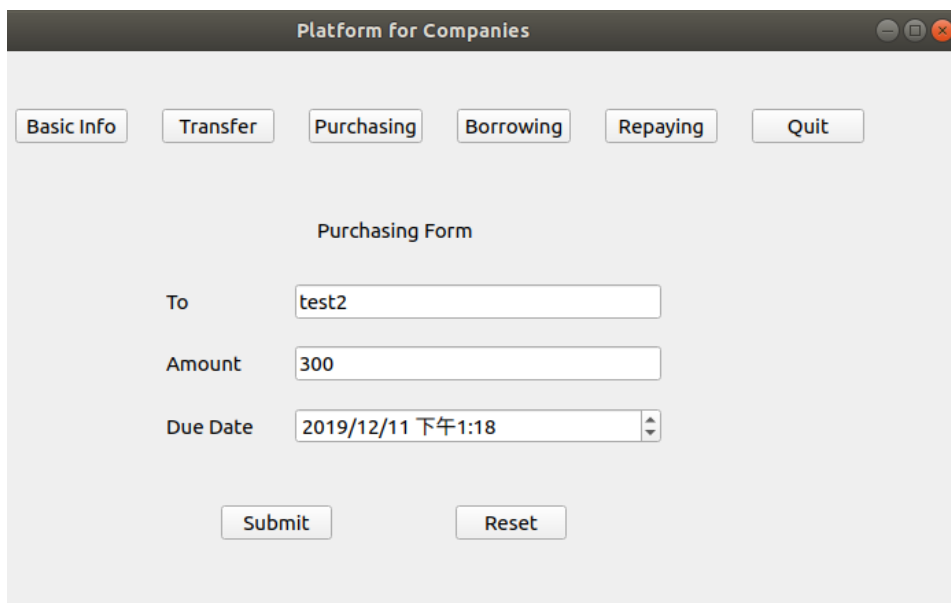
To: test3

Amount: 100

Due Date: 2019/12/11 下午1:18

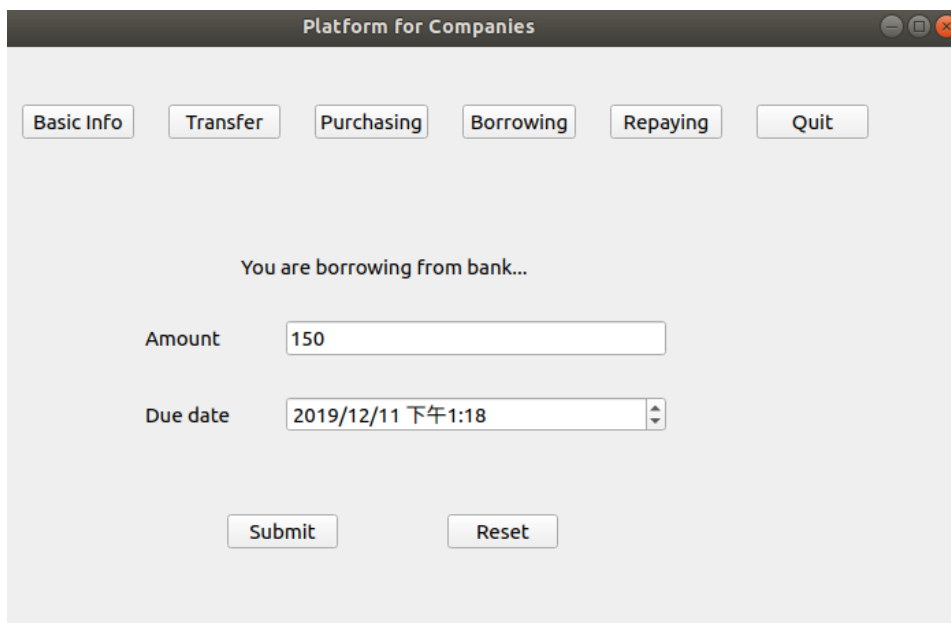
Submit Reset

- 在采购界面, 企业填入被采购方的名字和采购额, 设置相应的日期, 即可提交采购申请;



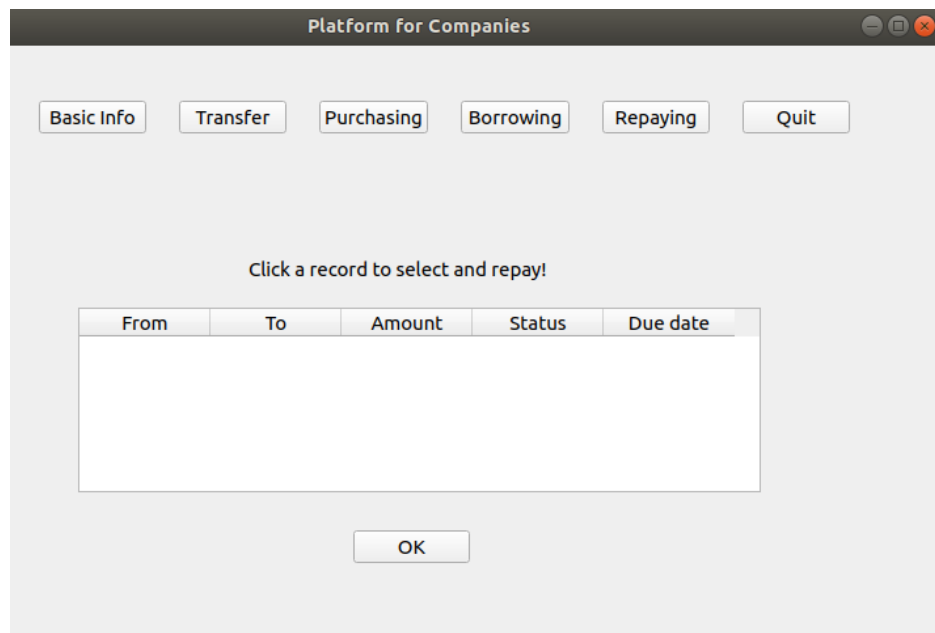
The screenshot shows a web application window titled "Platform for Companies". At the top, there is a navigation bar with six buttons: "Basic Info", "Transfer", "Purchasing", "Borrowing", "Repaying", and "Quit". The "Purchasing" button is highlighted. Below the navigation bar, the title "Purchasing Form" is centered. The form contains three input fields: "To" with the value "test2", "Amount" with the value "300", and "Due Date" with the value "2019/12/11 下午1:18". At the bottom of the form, there are two buttons: "Submit" and "Reset".

- 申请融资界面, 填入申请的融资额度以及还款日期, 即可提交融资申请;

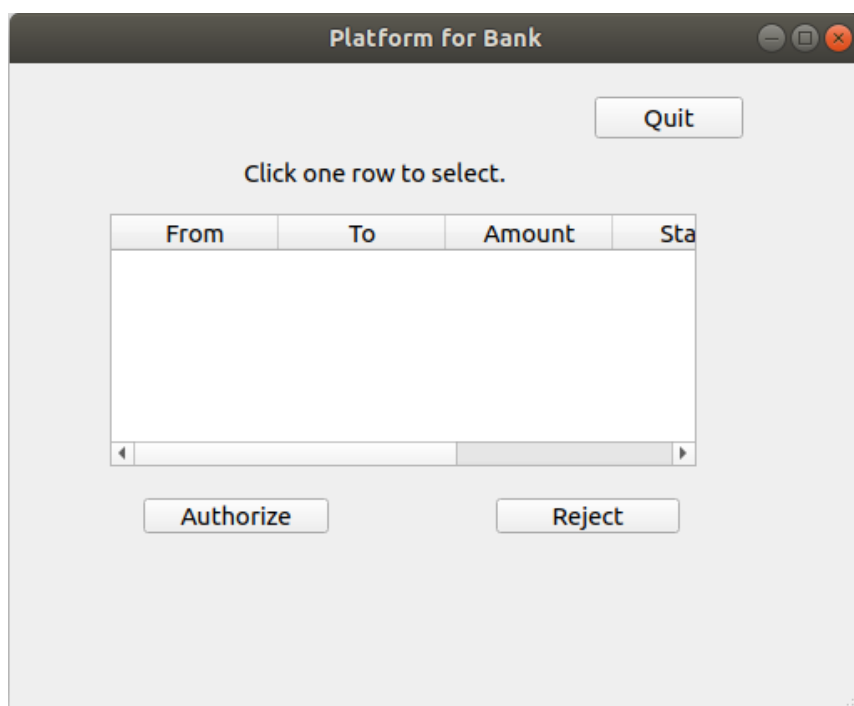


The screenshot shows the same "Platform for Companies" web application window, but now the "Borrowing" button in the navigation bar is highlighted. The title "You are borrowing from bank..." is centered. The form contains two input fields: "Amount" with the value "150" and "Due date" with the value "2019/12/11 下午1:18". At the bottom of the form, there are two buttons: "Submit" and "Reset".

- 还款界面, 在下方的表格中会出现自己需要还款的记录. 通过选中一行记录, 点击按钮, 即可进行还款.



#### 4. 银行界面



在该界面, 银行会收到所有提交但未处理的交易记录(即状态为" submitted" 的记录), 可以进行授权确认(Authorize)或者驳回取消(Reject).

#### 五、 加分项完成情况:



1. 设计了比较友好的图形界面, 此外在界面中利用与表格进行互动的方式, 简化了用户的操作难度;
2. 更完整的业务逻辑. 除了要求实现的基本功能之外, 我利用区块链技术分割了银行和企业这两类主体. 具体来说, 企业的所有交易记录都要提交上链; 之后, 银行通过登入区块链系统, 对所有提交上的交易记录进行审查, 判断进行授权确认抑或是驳回. 通过利用区块链的不可篡改性, 交易记录的可信度大大增加; 又由于企业和银行之间没有直接的交互, 而是通过区块链进行, 整个系统的效率也会大大提高.

## 六、 心得体会

1. 在完成本次课程设计的过程中, 我学习到了许多有关区块链的知识, 从底层的共识机制, 原理到上层的接口等基本系统设计方法都有了全新的认识, 让我受益匪浅.
2. 在实现链端代码的过程中, 我遇到了相当多的困难, 包括语法, 变量类型的使用等都碰到了数不清的 bug, 好在有各大提问网站的存在, 让我能够顺利 debug. 其中最难受的莫过于遇到 Stack too deep 的问题, 这时只能尽可能把合约封装得精简, 调用了多个函数, 去掉多余的局部变量, 为实现代码带来了不少的难处. 此外, 在配置连接 FISCO-BCOS 链的 Python-SDK 环境时也比较复杂, 看了好几天文档才大致搞明白如何在 python 中连接区块链, 进行交互并且获取数据.
3. 在实现前后端代码的过程中, 同样也有不小的麻烦. 主要原因是我对 PyQt5 的框架比较陌生, 写起来不太顺手, 尽管花了很长时间阅读文档, 对于有些地方也还是不太理解. 好在 PyQt5 有一个设计器, 是一个图形化的程序, 能够辅助构建图形界面, 为图形界面的开发提供了相当大的便利. 而在实现后端代码的过程中, 我遇到的主

要的问题是如何处理得当前后端, 后端与链端数据交互的问题. 虽然目前的整体框架不是最优的, 但好在能够顺利完成, 耦合性也不是特别强, 方便之后修改.

4. 整个大作业的过程不仅让我对区块链有了更全面的了解, 同样也让我对开发区块链的应用有了更直观的认识. 从供应链金融这一具体的应用目的出发, 利用区块链的不可篡改性及可追溯性等优势, 我们可以将整个供应链中的每一笔交易和应收账款单据上传到区块链中, 同时再引入第三方可信的机构确认交易信息, 确保交易和应收账款单据的真实性, 同时也解决了供应链中信息流通的问题.
5. FISCO-BCOS 为我使用区块链进行应用开发提供了一个绝佳的平台. 它具有很棒的文档, 这在配置环境时相当友好. 此外, 它提供的 Python-SDK 也很易用, 它自己已经实现了基本的与区块链进行通讯所需的函数, 接口, 解析器等, 我要做的基本只是调用. 虽然部分地方的文档还有些残缺, 部分功能还没有完全实现(例如 CRUD 接口, 只实现了基本的表操作), 但都可以通过其他方法避开这些困难.
6. 总的来说, TA, 老师们以及 FISCO-BCOS 平台提供给了我一个很棒的区块链学习资源! 十分感谢!

## 七、参考文献

- [1] 龙云安, 张健, 艾蓉. "基于区块链技术的供应链金融体系优化研究." *西南金融* 1 (2019): 9.
- [2] 许荻迪. "区块链技术在供应链金融中的应用研究." *西南金融* 2 (2019): 10.

[3] 张功臣,赵克强,侯武彬.基于区块链技术的供应链融资创新研究[J].信息技术与网络安全,2019,38(10):14-17.

[4] FISCO-BCOS 官方文档.