中山大学数据科学与计算机学院本科生实验报告

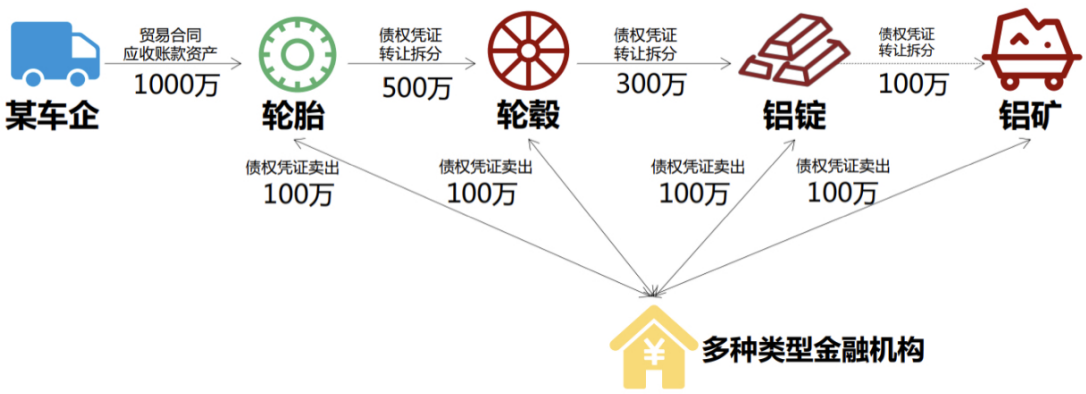
课程名称：区块链原理与技术　　任课教师：黄华威

学号	姓名
18340030	楚鸿飞
18340026	陈志昱
18340137	彭一铭

项目背景

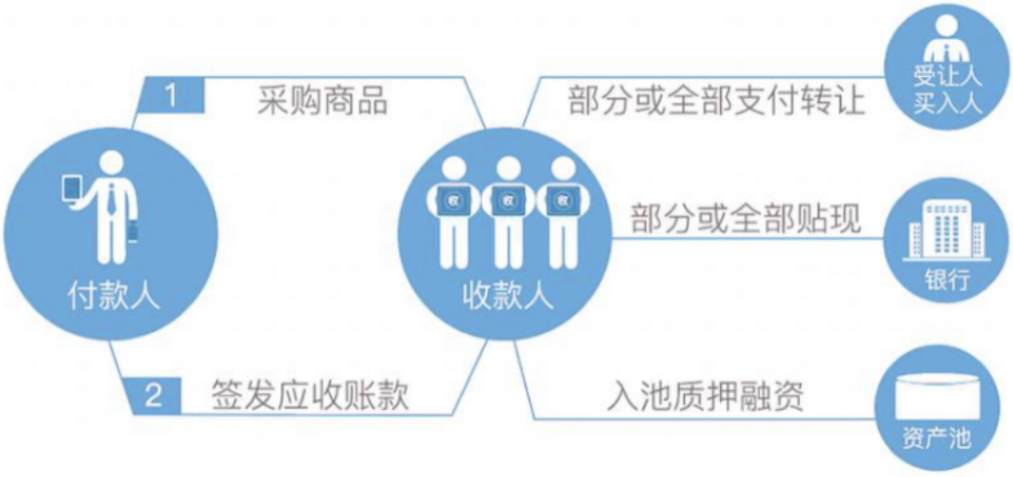
- 传统供应链金融

  某车企（宝马）因为其造车技术特别牛，消费者口碑好，所以其在同行业中占据绝对优势地位。因此，在金融机构（银行）对该车企的信用评级将很高，认为他有很大的风险承担的能力。在某次交易中，该车企从轮胎公司购买了一批轮胎，但由于资金暂时短缺向轮胎公司签订了1000万的应收账款单据，承诺1年后归还轮胎公司1000万。这个过程可以拉上金融机构例如银行来对这笔交易作见证，确认这笔交易的真实性。在接下里的几个月里，轮胎公司因为资金短缺需要融资，这个时候它可以凭借跟某车企签订的应收账款单据向金融结构借款，金融机构认可该车企（核心企业）的还款能力，因此愿意借款给轮胎公司。但是，这样的信任关系并不会往下游传递。在某个交易中，轮胎公司从轮毂公司购买了一批轮毂，但由于租金暂时短缺向轮胎公司签订了500万的应收账款单据，承诺1年后归还轮胎公司500万。当轮毂公司想利用这个应收账款单据向金融机构借款融资的时候，金融机构因为不认可轮胎公司的还款能力，需要对轮胎公司进行详细的信用分析以评估其还款能力同时验证应收账款单据的真实性，才能决定是否借款给轮毂公司。这个过程将增加很多经济成本，而这个问题主要是由于该车企的信用无法在整个供应链中传递以及交易信息不透明化所导致的。如图一所示，为该场景的介绍



- 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。



项目设计

- 存储设计

  采用 FISCO-BCOS 平台提供的 CRUD 接口实现企业信息和收据的存储。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。

- 功能设计

  - 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。
  - 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。
  - 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。
  - 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。

- 核心功能介绍

  - Sign Up：为用户创建一个账户，并生成相应的密钥

    后端代码：

```
def press_register(self):
        name, password, balance =
self.line_name.text(), self.line_pwd.text(),
self.line_balance.text()
        # balabce代表启动资金
        balance = int(balance)
```

```
        if len(name) > 256:
            QMessageBox.warning(self, 'Error', '名称
过长。')
            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, name)
        print("save to file : [{}]".format(keyfile))
        with open(keyfile, "w") as dump_f:
            json.dump(kf, dump_f)
            dump_f.close()
        print(
            "INFO >> Read [{}] again after new
account,address & keys in file:".format(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))
            dump_f.close()

        global client, contract_abi, to_address
        args = [name, ac.address, 'Company',
balance]
        # 调用智能合约中的register函数
        receipt =
client.sendRawTransactionGetReceipt(to_address,contr
act_abi,"register",args)
        print("receipt:",receipt['output'])
```

```
 QMessageBox.information(self,'Prompt','success。',
QMessageBox.Ok)
```

- Log In：已注册的用户输入名称和密码进行登录

  后端代码

```
def validate(self):
        name = self.line_name.text()
        password = self.line_pwd.text()
        if name == "bank" and password == "bank":
            bank_window.show()
            bank_window.set_table_content()
        else:
            keyfile = "
{}/{}.keystore".format(client_config.account_keyfile
_path, name)
            # 如果名字不存在
            if os.path.exists(keyfile) is False:
                QMessageBox.warning(self,
                        "error",
                        "名称 {} 不存在，请先注
册。".format(name),
                        QMessageBox.Yes)
            else:
                print("name : {}, keyfile:{}
,password {}  ".format(name, 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)
                except Exception as e:
                    QMessageBox.warning(self,
                            "error",
                            ("Failed to load account
info for [{}],"
                                                 " error
info: {}!").format(name, e),
                            QMessageBox.Yes)
```

- Purchase:用户登录后，可与其他用户签订单据

  后端代码

```
# 提交purchase设置
    def submit_purchase(self):
        global client, contract_abi, to_address
        args = [self.line_pur_from.text(),
self.company_name , int(self.line_pur_amt.text()),
self.purchase_date.dateTime().toString("yyyy/MM/dd
hh:mm:ss")]

 client.sendRawTransactionGetReceipt(to_address,
contract_abi, "purchase", args)

 QMessageBox.information(self,'Prompt','success。',
QMessageBox.Ok)
```

- Finance：用户登录后可向银行申请融资，融资金额不得超过该用户借款总金额与欠款总金额之差

  后端代码

```
# 提交融资设置
    def submit_finance(self):
        _amt = int(self.line_fin_amt.text())
        _due =
self.finance_date.dateTime().toString("yyyy/MM/dd
hh:mm:ss")
        if _amt > (self.total_lent -
self.total_borrowed):
            QMessageBox.warning(self,'Error',"failed
with {}.".format(str(self.total_lent -
self.total_borrowed)), QMessageBox.Ok)
        else:
            global client, contract_abi, to_address
            args = [self.company_name, _amt,_due]

 client.sendRawTransactionGetReceipt(to_address,
contract_abi, "finance", args)

 QMessageBox.information(self,'Prompt','success。',
QMessageBox.Ok)
```

- Transfer：用户登陆后，若同时存在借出单据和欠款单据，则可转让单据。转让金额不得超过任一被转让单据的金额

  后端代码

```
# 提交transfer设置内容
    def submit_transfer(self):
        global client, contract_abi, to_address
        if
self.table_trans_lent.selectionModel().hasSelection(
) and
self.table_trans_bor.selectionModel().hasSelection()
:
            row_lent =
self.table_trans_lent.currentRow()
            row_bor =
self.table_trans_bor.currentRow()
            args =
[self.table_trans_lent.item(row_lent, 1).text(),
self.company_name,
self.table_trans_bor.item(row_bor, 0).text(),
int(self.line_trans_amt.text())]
```

```
            print(args) # 在终端输出，便于调试
            if self.table_trans_bor.item(row_bor,
3).text() == "authorized" and
self.table_trans_lent.item(row_lent, 3).text() ==
"authorized":
                result =
client.sendRawTransactionGetReceipt(to_address,
contract_abi, "transfer", args)
                print("receipt:",result['output'])
                # 由于难以找到bug，所以这一功能暂时只能在链
端用命令实现

 QMessageBox.information(self,'Prompt','Fail',
QMessageBox.Ok)

            else:

 QMessageBox.warning(self,'Error','Please check',
QMessageBox.Ok)
        else:

 QMessageBox.warning(self,'Prompt','Please select',
QMessageBox.Ok)
```

- Repay: 用户登陆后，可对已有欠款单据进行结算

  后端代码

```
# 还款
    def repay(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":
                # 调用智能合约中的repay函数，并返回结果
```

```
                result =
client.sendRawTransactionGetReceipt(to_address,
contract_abi, "repay", args)
                print("receipt:",result)
                res =
hex_to_signed(result['output'])
                if res == 0:

 QMessageBox.information(self,'Prompt','success。',
QMessageBox.Ok)
                else:

 QMessageBox.warning(self,'Prompt','fail。',
QMessageBox.Ok)
                self.table_repay.setRowCount(0)

 self.set_table_repay_content(self.company_name)
            else:
                QMessageBox.warning(self,'Error','请
先认证相关交易。', QMessageBox.Ok)
        else:
            QMessageBox.warning(self,'Prompt','请先选
择相关记录。', QMessageBox.Ok)
```

- 银行确认单据：银行登录后（账号密码均为bank），可批准或拒绝已提交的单据

  后端代码

```
# 定义银行的认证功能
    def 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)
            # 调用合约函数 update ，将信息更新为
"authorized"
```

```
            result =
client.sendRawTransactionGetReceipt(to_address,
contract_abi, "update", args)
            print("receipt:",result)

 QMessageBox.information(self,'Prompt','Success',
QMessageBox.Ok)
            self.table.setRowCount(0)
            self.set_table_content()
        else: # 如果没有被选中的记录

 QMessageBox.warning(self,'Prompt','Please select',
QMessageBox.Ok)
```

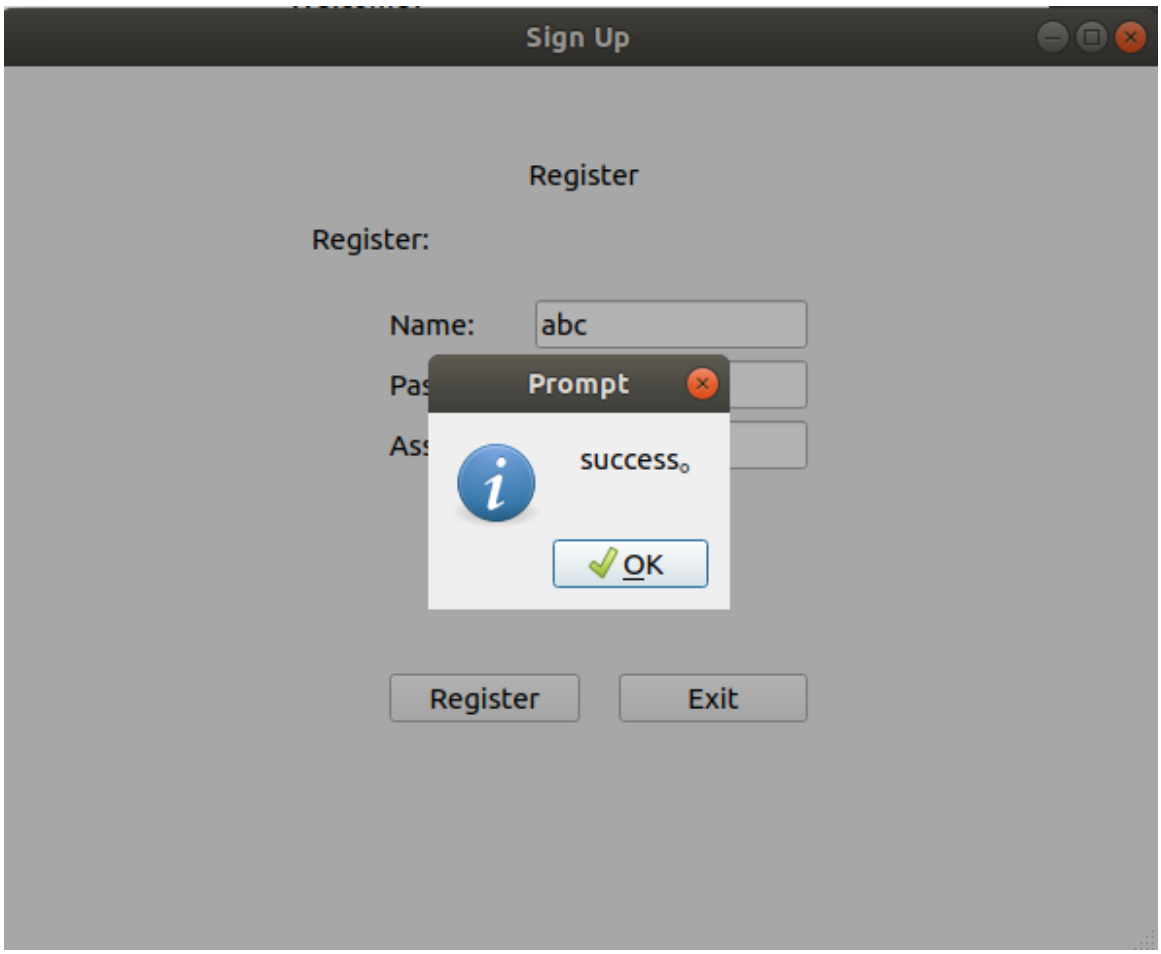
- 银行拒绝认证：银行登陆后，可以取消这条单据记录，表示银行拒绝认证

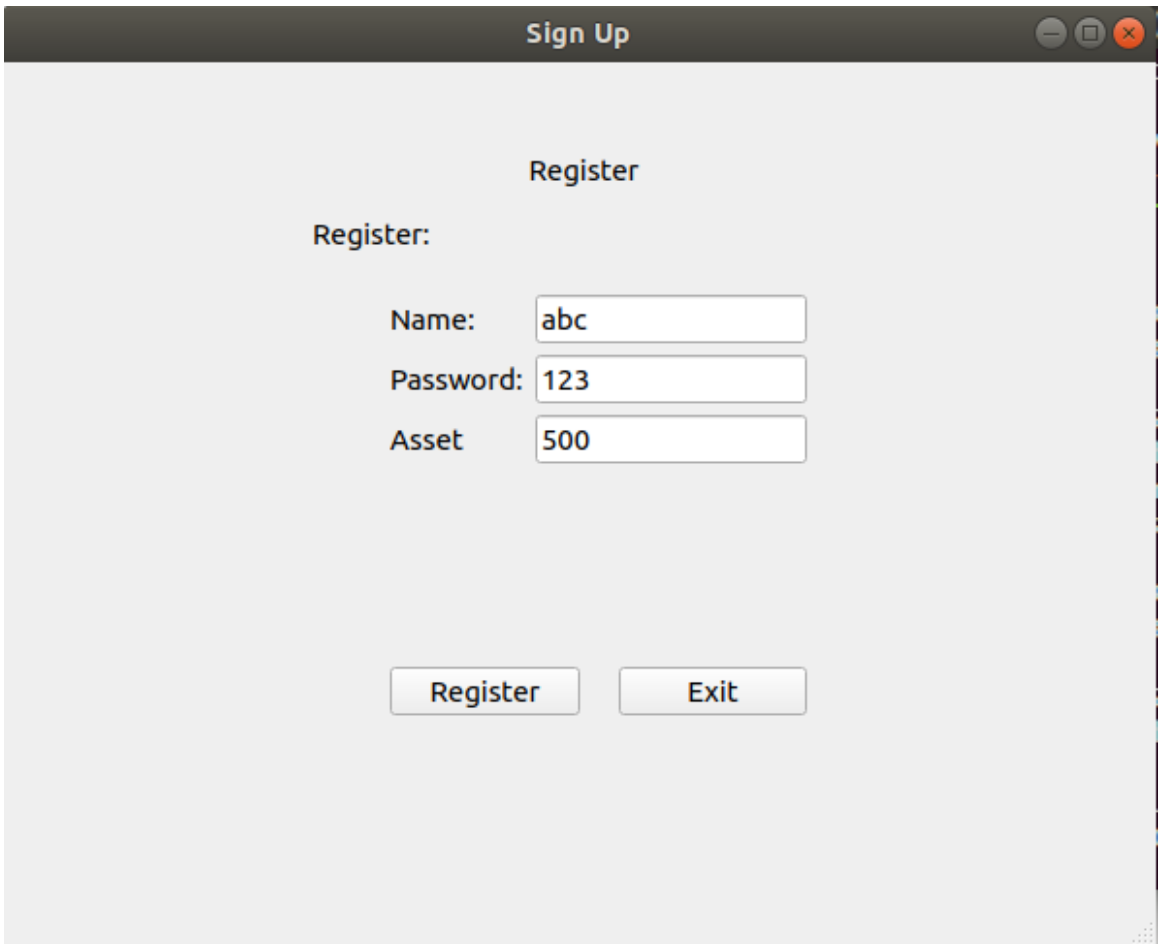
后端代码

```
# 取消这条记录，表示银行拒绝认证
    def 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)
            # 调用合约函数 remove ，移除这条记录
            result =
client.sendRawTransactionGetReceipt(to_address,
contract_abi, "remove", args)
            print("receipt:",result)

 QMessageBox.information(self,'Prompt','success',
QMessageBox.Ok)
            self.table.setRowCount(0)
            self.set_table_content()
        else:

 QMessageBox.warning(self,'Prompt','Please select',
QMessageBox.Ok)
```

功能测试

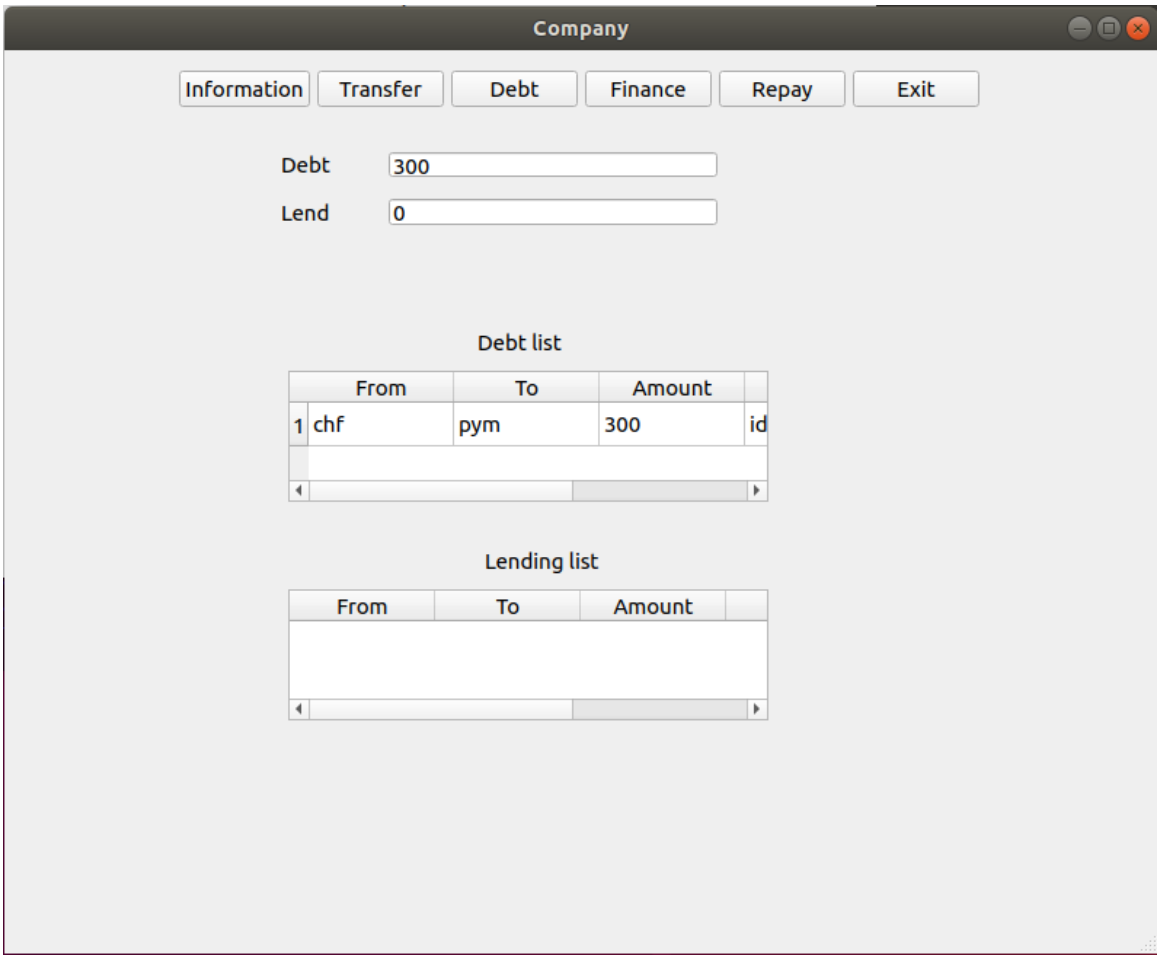
- 模拟创建账户


Sign Up
Register
Register:
Name:
abc
Prompt
success。
OK
Register
Exit

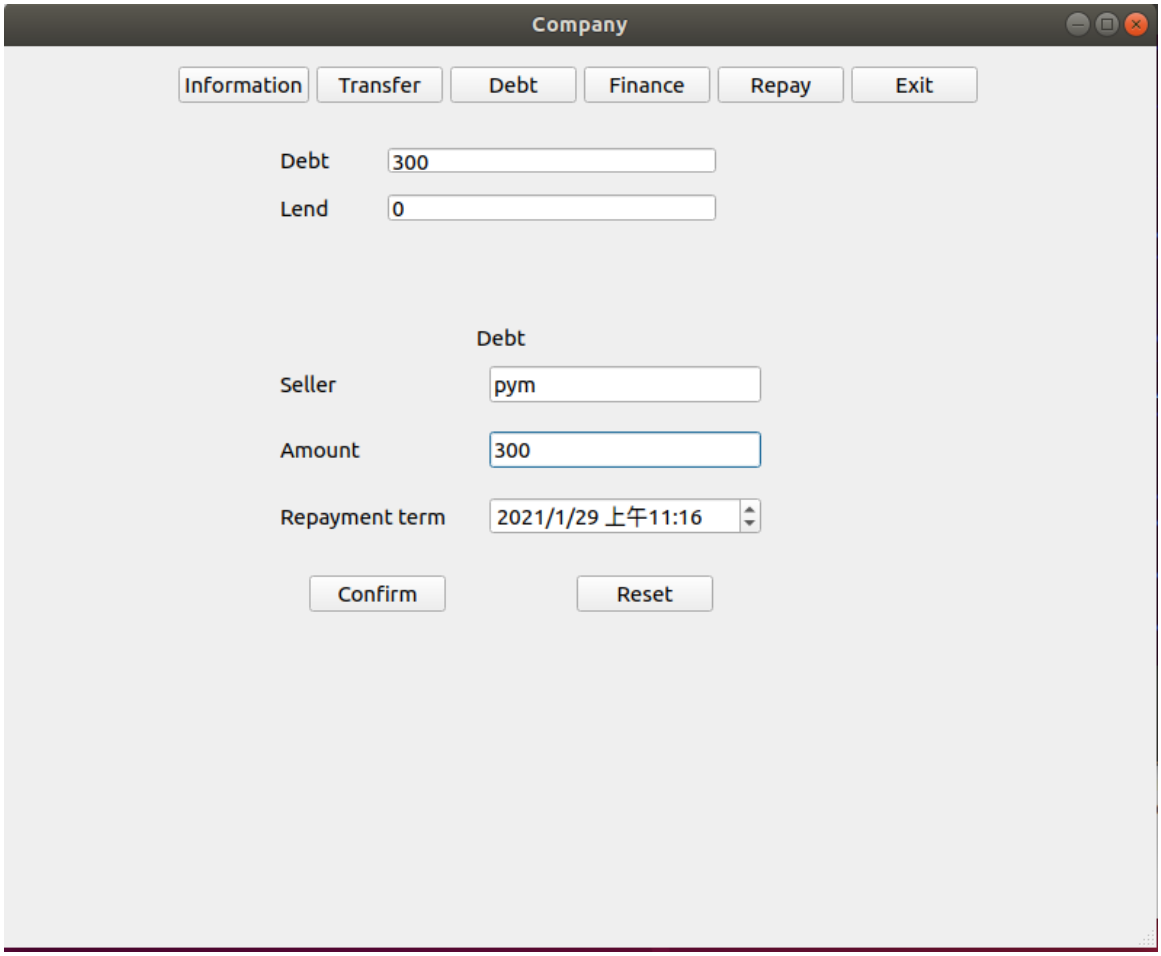

Sign Up
Register
Register:
Name:
abc
Password:
123
Asset
500
Register
Exit

```
[group:1]> call myAsset 0x6109e9e5e9595ef72638a17575fb61cdbd6ae41a register pymdd 500
transaction hash: 0x4ee557b823c860320fc62996917346a36369aba294e7de095ef71448a3ed88c7
---------------------------------------------------------------------------------------------
transaction status: 0x0
description: transaction executed successfully
---------------------------------------------------------------------------------------------
Output
Receipt message: Success
Return message: Success
Return value: [0]
---------------------------------------------------------------------------------------------
Event logs
Event: {"RegisterEvent":[[0,"pymdd",500]]}

[group:1]> call myAsset 0x6109e9e5e9595ef72638a17575fb61cdbd6ae41a register chfdd 500
transaction hash: 0xf3b81dd05d5e6b370791d8fe2262a92f89069c7660221d49b80fbd7f96a009aa
---------------------------------------------------------------------------------------------
transaction status: 0x0
description: transaction executed successfully
---------------------------------------------------------------------------------------------
Output
Receipt message: Success
Return message: Success
Return value: [0]
---------------------------------------------------------------------------------------------
Event logs
Event: {"RegisterEvent":[[0,"chfdd",500]]}

[group:1]> call myAsset 0x6109e9e5e9595ef72638a17575fb61cdbd6ae41a register czygg 500
transaction hash: 0x50d3ba751bf1f0f4343cfe529f6a8655c2f6a6dbea2821de792d2225a5539f0d
---------------------------------------------------------------------------------------------
transaction status: 0x0
description: transaction executed successfully
---------------------------------------------------------------------------------------------
Output
Receipt message: Success
Return message: Success
Return value: [0]
```

- 查看欠款信息

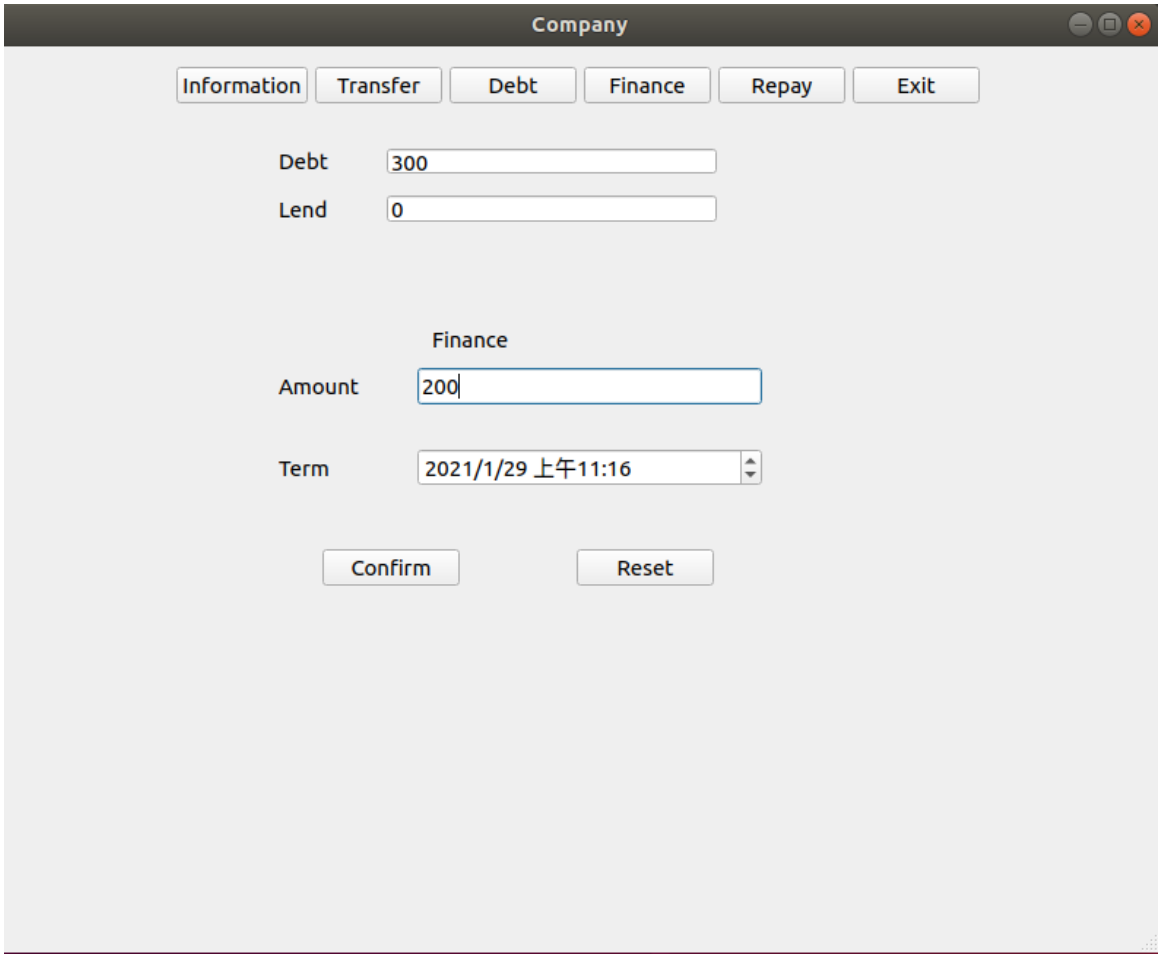


- 创建借条




```
[group:1]> call myAsset 0x62fc206e096ee3e92e621f56b19d6d0e6024b158 issueReceivables qiantiao14 chfdd pymdd 100
transaction hash: 0xaac9b431a7dffc81877f0b84e2e4fc401f86a181ffe6046e7982eb7bc72aebf9
-------------------------------------------------------------------------------------------
transaction status: 0x0
description: transaction executed successfully
-------------------------------------------------------------------------------------------
Output
Receipt message: Success
Return message: Success
Return value: [0]
-------------------------------------------------------------------------------------------
Event logs
Event: {"issueReceivablesEvent":[[0,"qiantiao14","chfdd","pymdd",100]]}
```

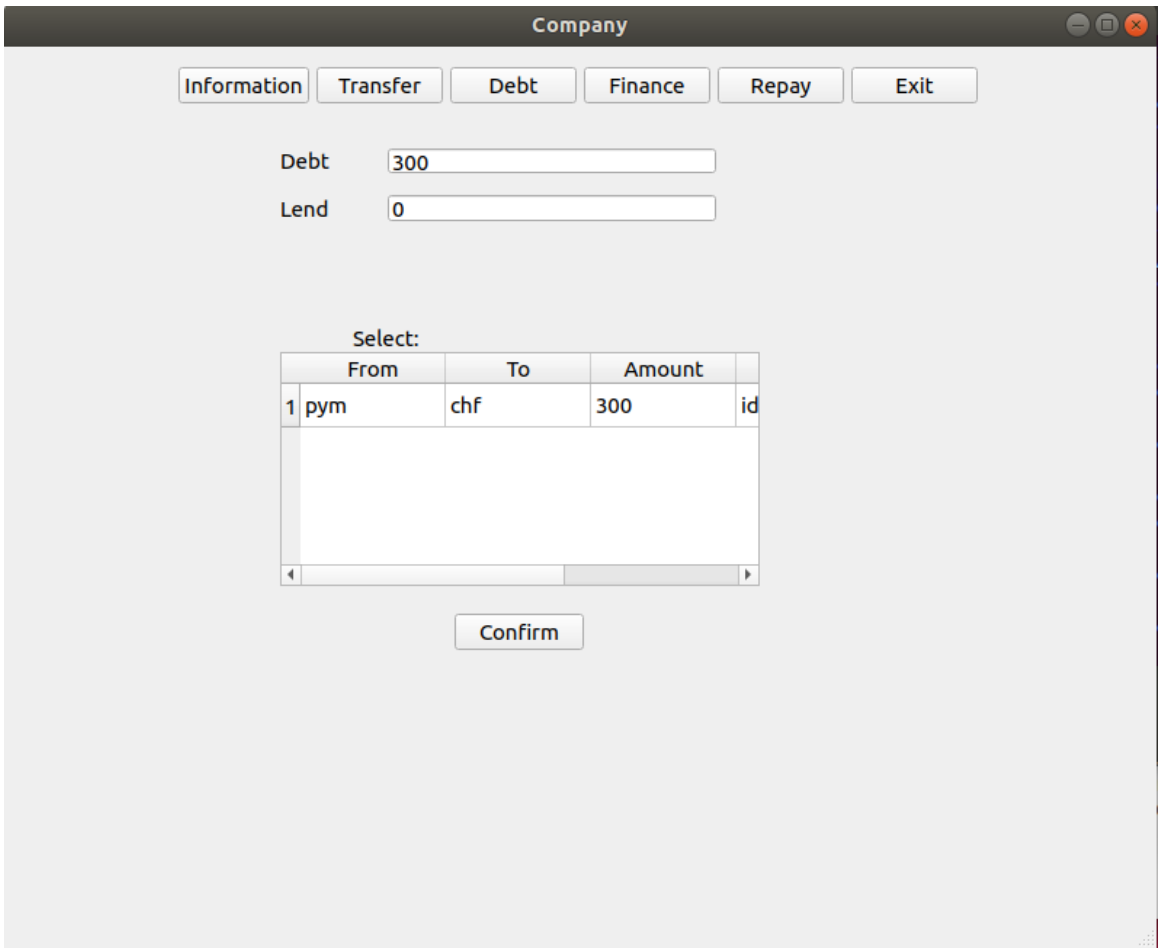
- 使用借条向银行融资




```
[group:1]> call myAsset 0x62fc206e096ee3e92e621f56b19d6d0e6024b158 receiveMoneyfromBank qiantiao14 qiantiao15 bigbank
transaction hash: 0xb79a063f21c1ff41d17edfde857f69040397b1c496d5464ed1da0b456df89c5a
--------------------------------------------------------------------------------------------
transaction status: 0x0
description: transaction executed successfully
--------------------------------------------------------------------------------------------
Output
Receipt message: Success
Return message: Success
Return value: [100]
--------------------------------------------------------------------------------------------
Event logs
Event: {"issueReceivablesEvent":[[0,"qiantiao15","chfdd","bigbank",100]],"transferReceivablesEvent":[[0,"qiantiao14",100],[0,"qiantiao14","qiantiao15","bigbank"]],"TransferEvent":[[0,"chfdd","pymdd",100],
[0,"pymdd","chfdd",100],[0,"bigbank","pymdd",100]]}
```

```
[group:1]> call myAsset 0x62fc206e096ee3e92e621f56b19d6d0e6024b158 select pymdd
---------------------------------------------------------------------------------------------
Return code: 0
description: transaction executed successfully
Return message: Success
---------------------------------------------------------------------------------------------
Return values:
[
    0,
    700
]
---------------------------------------------------------------------------------------------

[group:1]> call myAsset 0x62fc206e096ee3e92e621f56b19d6d0e6024b158 select bigbank
---------------------------------------------------------------------------------------------
Return code: 0
description: transaction executed successfully
Return message: Success
---------------------------------------------------------------------------------------------
Return values:
[
    0,
    99999899
]
---------------------------------------------------------------------------------------------
```

- 支付上链




```
[group:1]> call myAsset 0x6109e9e5e9595ef72638a17575fb61cdbd6ae41a payReceivables d 100
transaction hash: 0xd971eb4e99541cfa25b9848e9607b10b0acf422473b82f9c4af074fb04b3dd16
-------------------------------------------------------------------------------------------
transaction status: 0x0
description: transaction executed successfully
-------------------------------------------------------------------------------------------
Output
Receipt message: Success
Return message: Success
Return value: [0,["chfdd","pymdd"]]
-------------------------------------------------------------------------------------------
Event logs
Event: {"transferReceivablesEvent":[[0,"d",100]],"TransferEvent":[[0,"chfdd","pymdd",100]]}
```

心得体会

通过这次大作业，我们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。

这次大作业也使我们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。区块链采用去中心化分布式记账方式，系统中各个节点同时参与数据变动记录，每个节点都保留一份相同且完整的账本，单个节点被摧毁不会影响整个账本及记录的完整性，极大地提高了数据的安全性。公有区块链系统是开放的，由其中所有具有维护功能的节点共同维护，除了交易各方的私有信息被加密外，任何人都可以通过公开的接口查询

区块链数据和开发相关应用，因此整个系统信息高度透明。因此这些特性可以保护我们的交易，使得我们的交易更加透明。

当然区块链的应用远不止于此，虽然在当下其交易确认具有延迟性，但未来其前景是长远的，所以对于区块链的深入研究更加有必要。

加分项

- 友好的GUI界面
- 支持国密特性

附**github**仓库：https://github.com/chf-x/BlockChainFinalProjects