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	16 级	专业（方向）	嵌入式
学号	14331303	姓名	肖著成
电话	15622159875	Email	xiaozhch3@mail2.sysu.edu.cn
开始日期	2019/12/30	完成日期	2020/1/15

一、 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。

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，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。

实现功能：

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。

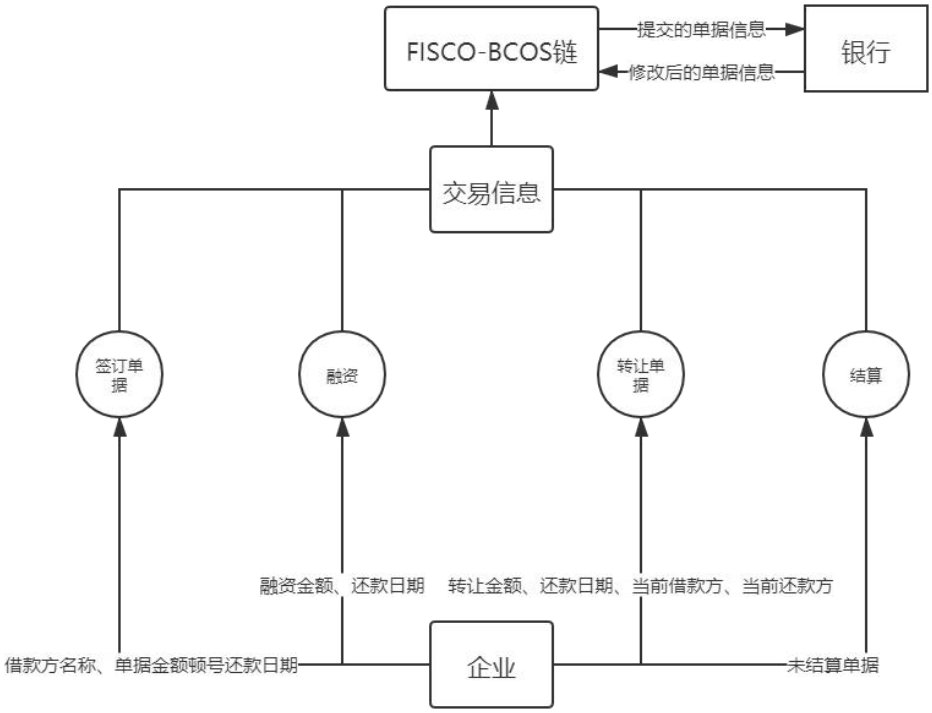
功能二：实现应收账款的转让上链，轮胎公司从轮毂公司购买一笔轮毂，便将于车企的应收账款单据部分转让给轮毂公司。轮毂公司可以利用这个新的单据去融资或者要求车企到期时归还钱款。
功能三：利用应收账款向银行融资上链，供应链上所有可以利用应收账款单据向银行申请融资。
功能四：应收账款支付结算上链，应收账款单据到期时核心企业向下游企业支付相应的欠款。

二、 方案设计

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。

数据流图



核心功能介绍

企业注册：为企业创建一个账户，生成对应秘钥

后端代码：

```
    def on_press_register(self):
        name, password = self.line_name.text(), self.line_pwd.text()
        max_account_len = 240
        if len(name) > max_account_len:
            QMessageBox.warning(self, 'Error', 'The name should be less than 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, 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']
        print(name)
        receipt =
client.sendRawTransactionGetReceipt(to_address,contract_abi,"register",args)
        print("receipt:",receipt['output'])


        QMessageBox.information(self,'Prompt','Successfully                registered!',
QMessageBox.Ok)
```

链端代码：

```
    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);
        return count;
    }
```

企业登录：已注册的企业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)
            #if the account doesn't exists
            if os.path.exists(keyfile) is False:
                QMessageBox.warning(self,
                        "error",
                        "Name {} doesn't exists. Please register first.".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)
```

签订单据：企业登录后，可与其他企业签订单据

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")
        _from = self.line_pur_from.text()
        global client, contract_abi, to_address
        args = [self.company_name , _from, int(_amt),_due]
        info_tuple = client.sendRawTransactionGetReceipt(to_address, contract_abi,
"purchase", args)
        print("receipt:",info_tuple['output'])
        res = hex_to_signed(info_tuple['output'])
        if res == -3:
            QMessageBox.warning(self,'Error','Companies must be registered first!',
QMessageBox.Ok)
        elif res == 1:

            QMessageBox.information(self,'Prompt','Successfully  submitted  purchasing
request.', QMessageBox.Ok)
```

链端代码：

```
function purchase(string _from, string _to, int amt, string dd) 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;
    }

}
```

融资：企业登录后，可向银行申请融资。融资金额不得超过该企业借款总金额与欠款总金额之差

后端代码：

```
    def on_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',"You don't have enough capacity to finance.
Your capacity is {}.".format(str(self.total_lent - self.total_borrowed)),
QMessageBox.Ok)
        else:
            global client, contract_abi, to_address
            args = [self.company_name,"bank", _amt,_due]
            info_tuple = client.sendRawTransactionGetReceipt(to_address, contract_abi,
"finance", args)
            QMessageBox.information(self,'Prompt','Successfully              financed.',
QMessageBox.Ok)
```

链端代码：

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

转让单据：企业登录后，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何一张被转让单据的金额

后端代码：

```
    def on_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()
            _from = self.table_trans_lent.item(row_lent, 1).text()
            _due = self.table_trans_lent.item(row_lent, 4).text()
            _from_prev_amt = int(self.table_trans_lent.item(row_lent, 2).text())
            _to_prev_amt = int(self.table_trans_bor.item(row_bor, 2).text())
            _to = self.table_trans_bor.item(row_bor, 0).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)
            args = [_from, self.company_name, _to, _from_prev_amt, _to_prev_amt,
_amt,_due]
            if self.table_trans_bor.item(row_bor, 3).text() == "authorized" and
self.table_trans_lent.item(row_lent, 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','Companies must be registered
first!', QMessageBox.Ok)
                elif res == -1:
                    QMessageBox.warning(self,'Error','Amount must be no more than the
amount of any selected records.', QMessageBox.Ok)
                elif res == 1:
                    QMessageBox.information(self,'Prompt','Successfully transferred.',
QMessageBox.Ok)
            else:
                QMessageBox.warning(self,'Error','Only [Authorized] receipts can be
transferred!', QMessageBox.Ok)
        else:

            QMessageBox.warning(self,'Prompt','Failed to transfer. Please click to
select records!', QMessageBox.Ok)
```

链端代码：

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

    }
```

结算：企业登录后，可对已有的欠款单据进行结算

后端代码：

```
def on_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":
                info_tuple = client.sendRawTransactionGetReceipt(to_address,
contract_abi, "repay", args)
                print("receipt:",info_tuple)
                QMessageBox.information(self,'Prompt','Successfully repayed.',
QMessageBox.Ok)
                self.table_repay.setRowCount(0)
                self.set_table_repay_content(self.company_name)
            else:
```

```
                QMessageBox.warning(self,'Error','Only [Authorized] receipts can be
repayed!', QMessageBox.Ok)
        else:
            QMessageBox.warning(self,'Prompt','Failed to repay. Please click to select
a record!', QMessageBox.Ok)
```

链端代码：

```
    function repay(string _from, string _to, int _amount, string _expiration) public
returns(int)
    {

        remove(_from, _to, _amount, _expiration);
```

银行确认单据：银行登录后（账号名密码为 bank），可批准或拒绝已提交的单据

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','Successfully authorized!',
QMessageBox.Ok)
            self.table.setRowCount(0)
            self.set_table_content()
        else:
            QMessageBox.warning(self,'Prompt','Failed to authorize. Please click to
select a record!', QMessageBox.Ok)

     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','Successfully rejected!',
QMessageBox.Ok)
            self.table.setRowCount(0)
            self.set_table_content()
        else:
```

```
            QMessageBox.warning(self,'Prompt','Failed to reject. Please click to select
a record!', QMessageBox.Ok)
```

链端代码:

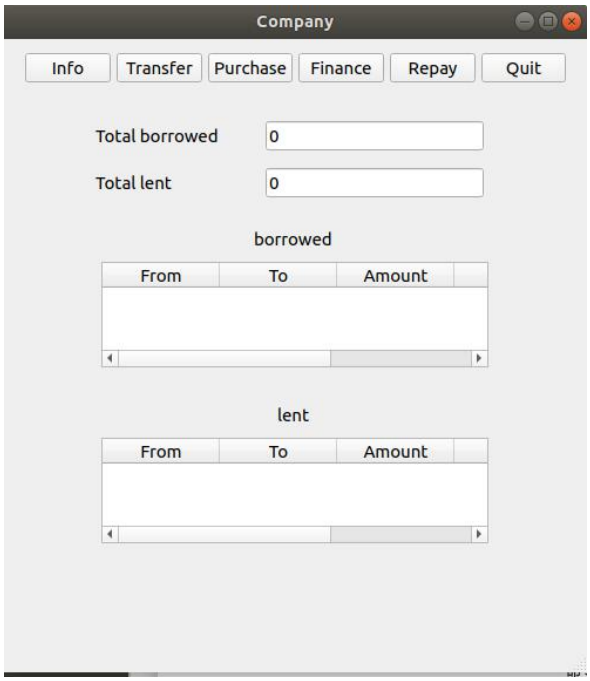
```
    function update(string _from, string _to, int amt, string sta, string dd) public
returns(int)
    {
        TableFactory tf = TableFactory(0x1001);
        Table table = tf.openTable("receipt_t");
        Entry entry = table.newEntry();
        entry.set("dummy", "active");
        entry.set("from", _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);
        int count = table.update("active", entry, condition);
        emit UpdateResult(count);
        return count;
    }

    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);

        return count;
    }
```

三、 功能测试&页面展示

企业注册：注册一个名称为 test，密码为 test 的账户

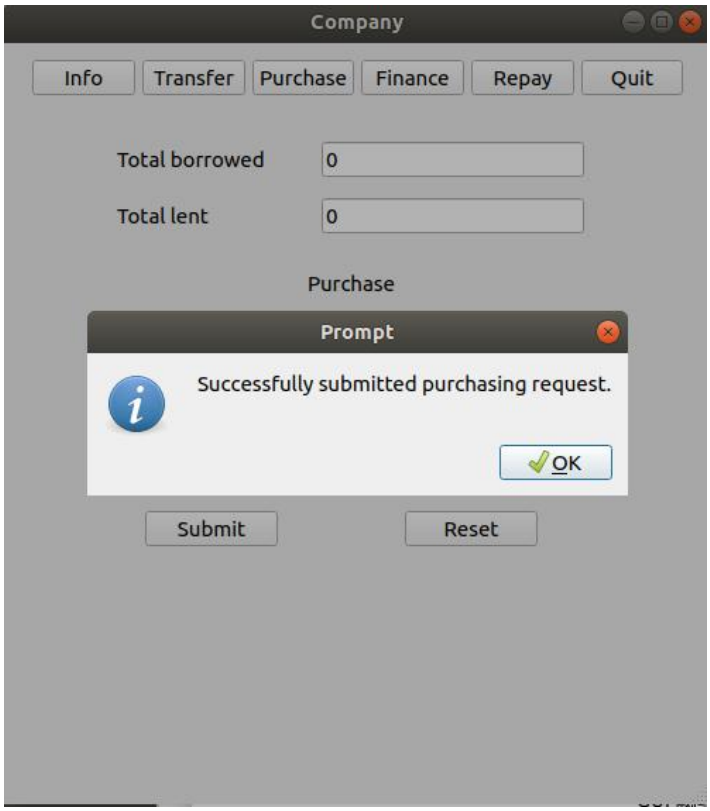

```
starting : test test
new address :     0xcBA7aD5c3657De3540f4026391440440D51b470d
new privkey :     0x3002cd29d8fe183cd7ddaf4faedb36bcfc50b3cef2dd2497c64c45a74dc8a
0e9
new pubkey :      0x0b266f6a2a1ec007551d1b6044cb048fc750e6aefc189d3b367a65b85e809
6c574c845eb91eae6add79ff2fd3360c9a60463495e502c241806f6d8416ff0a869
save to file : [bin/accounts/test.keystore]
INFO >> Read [bin/accounts/test.keystore] again after new account,address & keys
 in file:
address:          0xcBA7aD5c3657De3540f4026391440440D51b470d
privkey:          0x3002cd29d8fe183cd7ddaf4faedb36bcfc50b3cef2dd2497c64c45a74dc8a
0e9
pubkey :          0x0b266f6a2a1ec007551d1b6044cb048fc750e6aefc189d3b367a65b85e809
6c574c845eb91eae6add79ff2fd3360c9a60463495e502c241806f6d8416ff0a869

account store in file: [bin/accounts/test.keystore]
test
receipt: 0x0000000000000000000000000000000000000000000000000000000000000001
```

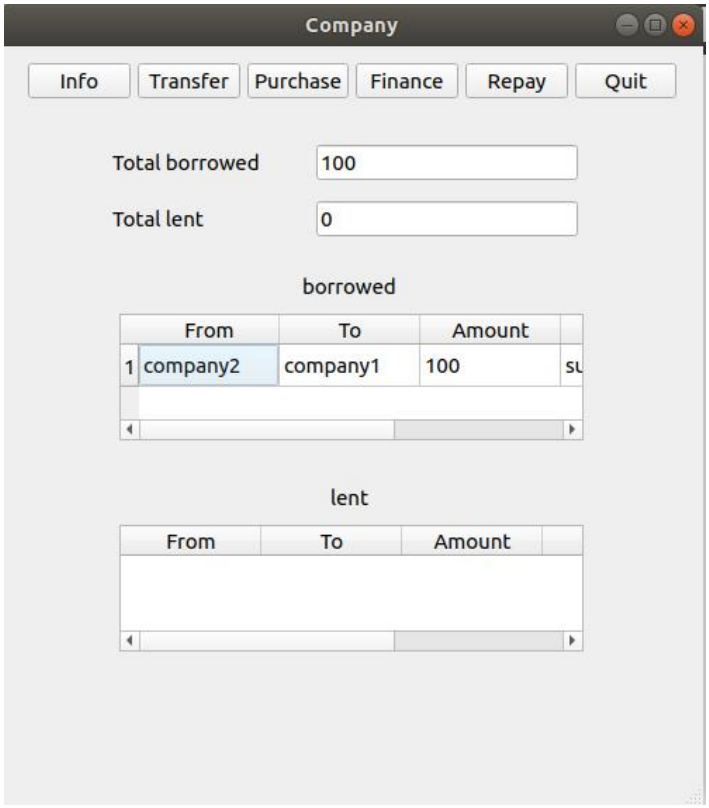
企业登录：登录 test 账户



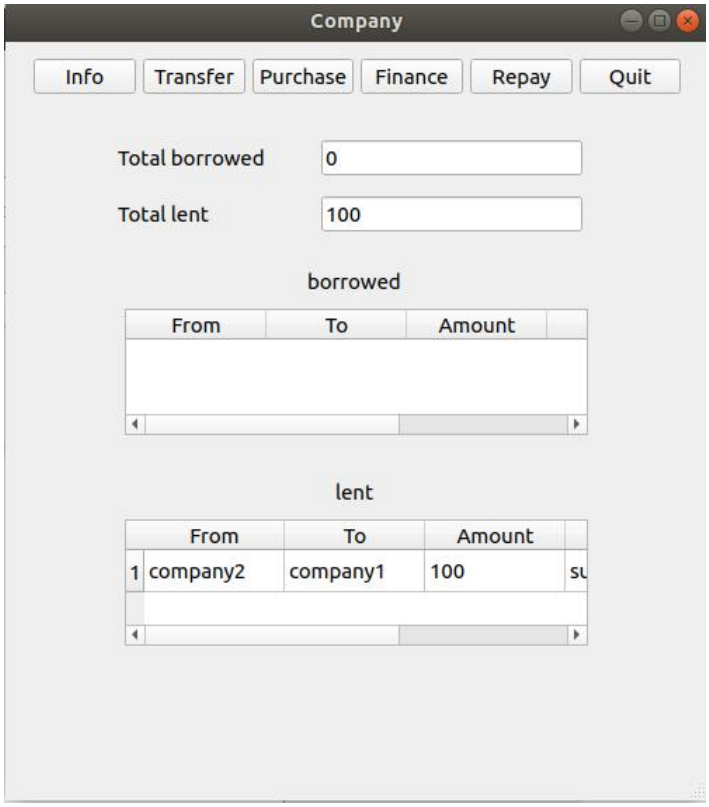

```
name : test, keyfile:bin/accounts/test.keystore ,password test
address:          0xcBA7aD5c3657De3540f4026391440440D51b470d
privkey:          0x3002cd29d8fe183cd7ddaf4faedb36bcfc50b3cef2dd2497c64c45a74dc8a
0e9
pubkey :          0x0b266f6a2a1ec007551d1b6044cb048fc750e6aefc189d3b367a65b85e809
6c574c845eb91eae6add79ff2fd3360c9a60463495e502c241806f6d8416ff0a869
receipt: ((), (), (), (), ())
receipt: ((), (), (), (), ())
receipt: ((), (), (), (), ())
receipt: ((), (), (), (), ())
receipt: ((), (), (), (), ())
```

签订单据：company1 向 company2 签订购买 100 元物品的单据



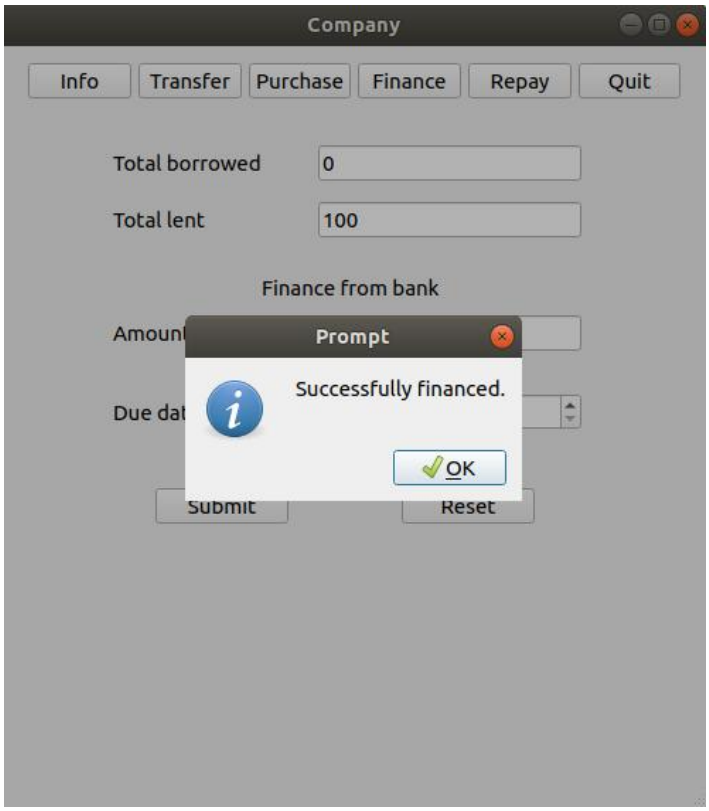


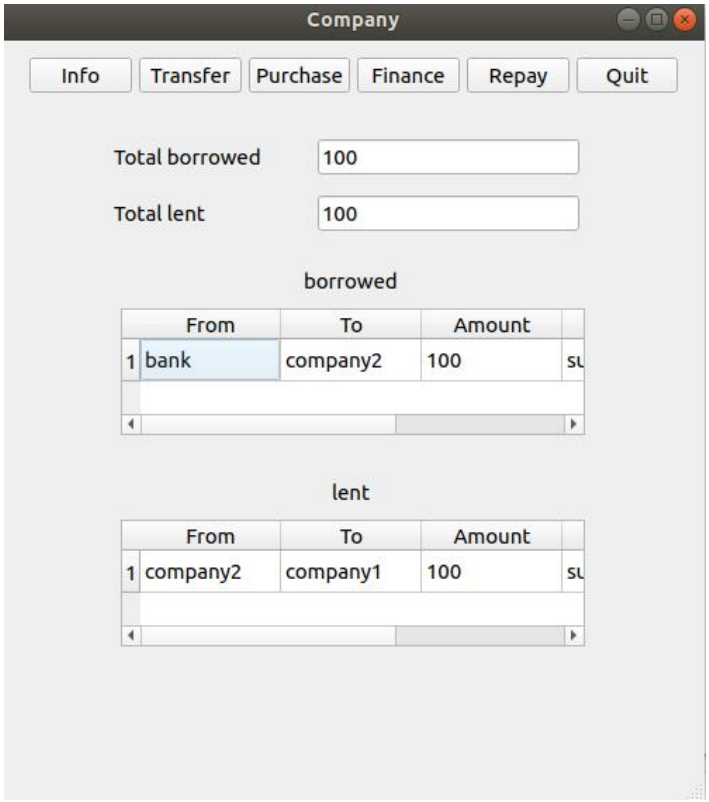
（Company1 信息页）



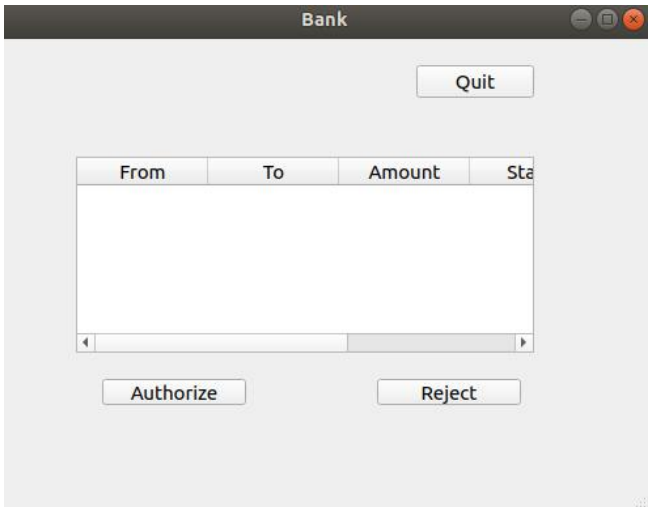
（Company2 信息页）

融资：company2 向银行融资 100 元

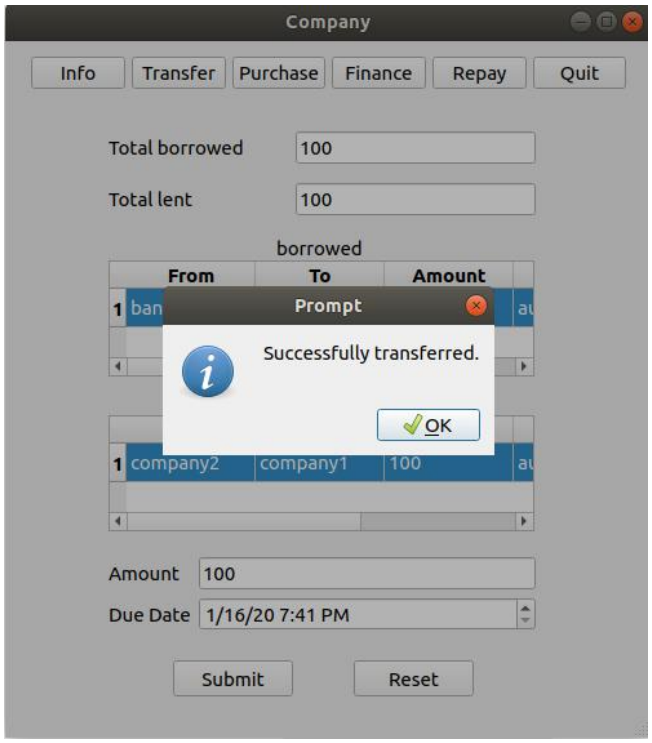


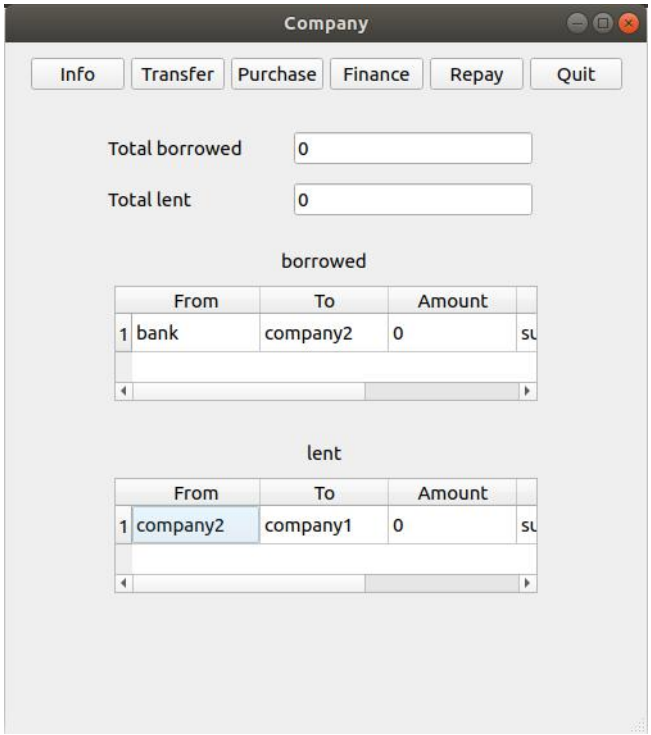


银行确认单据：批准订单

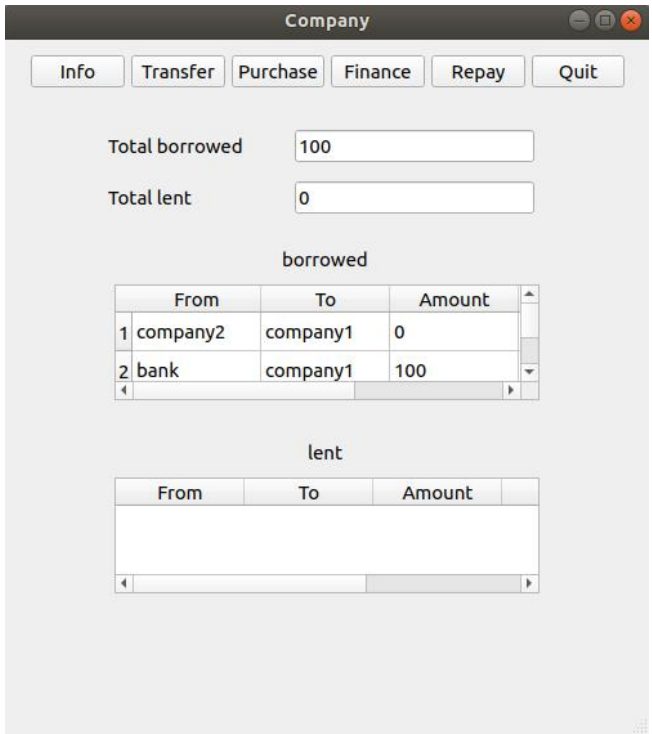


转让单据：company2 将融资欠款中的 100 元转让给 company1



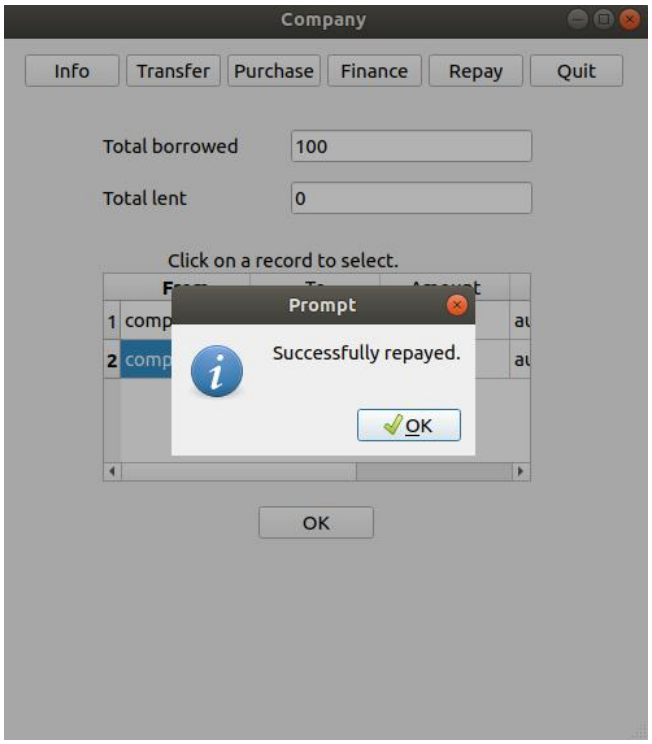


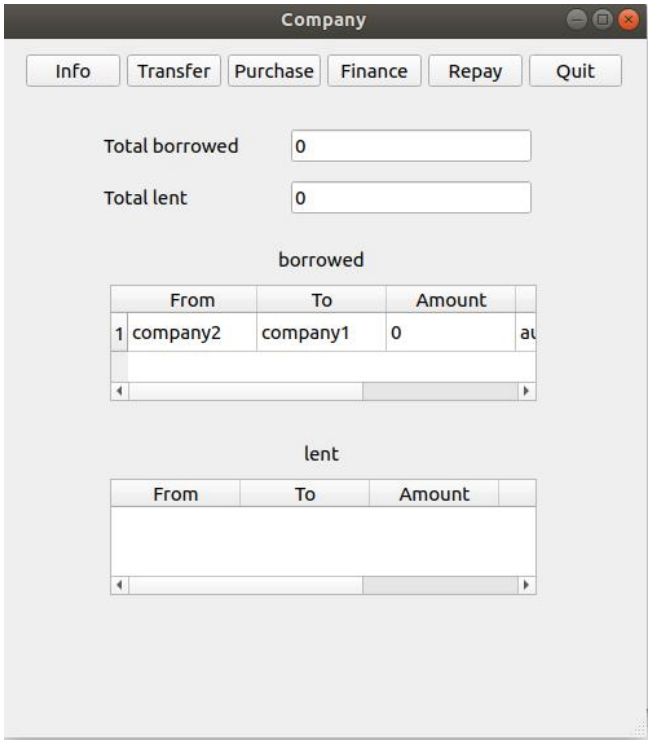
（Company2 信息页）



（Company1 信息页）

结算：company 结算与银行的单据





（Company1 信息页）