

# 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	17 级	专业（方向）	软件工程
学号	17343149	姓名	张浩轩
电话	19860204901	Email	<u>zhanghx59@mail2.s</u> <u>ysu.edu.cn</u>
开始日期	2019/10/17	完成日期	2019/12/12

目录

- 一. 项目背景 ..... 3
- 二. 方案设计 ..... 3
  - 1. 实现功能 ..... 3
  - 2. 流程图 ..... 4
  - 3. 链端 ..... 4
    - 3.1 具体实现 ..... 4
    - 3.2 核心功能 ..... 8
    - 3.3 附加功能 ..... 8
  - 4. 后端 ..... 9
    - 4.1 框架与语言选择 ..... 9
    - 4.2 具体实现 ..... 9
      - 4.2.1 部署 ..... 9
      - 4.2.2 查询 ..... 9
      - 4.2.3 Spring Controller 代码 ..... 10
  - 5. 前端 ..... 15
    - 5.1 登录界面 ..... 15
    - 5.2 用户界面 ..... 16
    - 5.3 操作界面 ..... 17
- 三. 功能测试 ..... 17
  - 1. 功能一 ..... 17
  - 2. 功能二 ..... 17
  - 3. 功能三(核心功能) ..... 18
  - 4. 功能四 ..... 19
- 四. 界面展示 ..... 20
  - 1. 登录界面 ..... 20
  - 2. 用户不存在界面 ..... 20
  - 3. 密码错误界面 ..... 21
  - 4. 用户界面 ..... 21
  - 5. 查询界面 ..... 22
  - 6. 签订应收账款界面 ..... 22
  - 7. 转移账款界面 ..... 23
  - 8. 小企业银行融资界面 ..... 23
  - 9. 到期还钱界面 ..... 24
- 五. 心得体会 ..... 25

## 一. 项目背景

**传统供应链金融领域中的信用评级方法不够完善, 中小企业的融资难度比较大, 如下例**

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。

## 二. 方案设计

完善传统供应链金融，将其与区块链的特性融合，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，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。

### 1. 实现功能

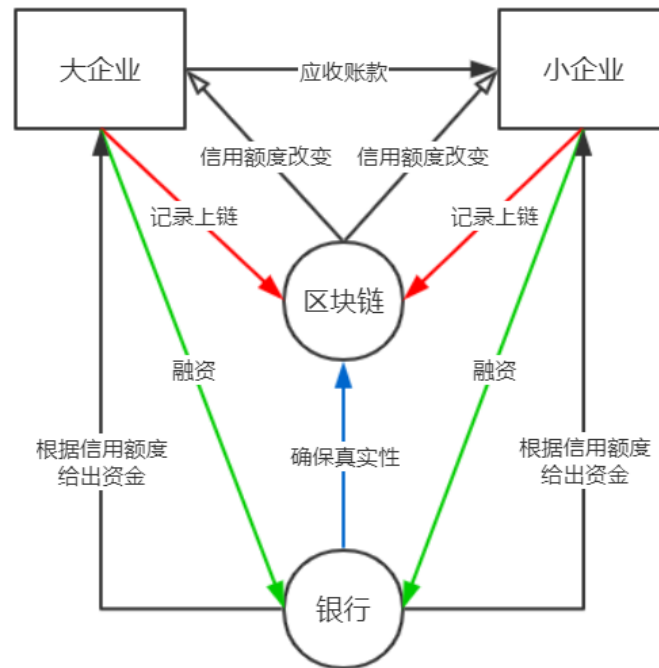
**功能一：**实现采购商品—签发应收账款 交易上链。例如车企从轮胎公司购买一批轮胎并签订应收账款单据。

**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。

## 2. 流程图



## 3. 链端

### 3.1 具体实现

**功能一：**实现采购商品—签发应收账款 交易上链。

在此处采用的并不是金额直接转移，而是签订应收账款的方式，在合约中实现 buy 函数

```
// @brief 签订应收账款
function buy(string name, int item_id, string item_name) public{
    TableFactory tf = TableFactory(0x1001);
    Table table = tf.openTable("t_test");

    // 条件为空表示不筛选 也可以根据需要使用条件筛选
    Condition condition = table.newCondition();

    Entries entries = table.select(name, condition);
    Entry entry;
    int count=0;
    if(entries.size() == 0){ // 没有则插入新条目
        insert(name, item_id, item_name);
    } else{ // 有就在此基础上增加钱的数量
        bytes32[] memory user_name_bytes_list = new bytes32[](uint256(entries
.size()));
        int[] memory item_id_list = new int[](uint256(entries.size()));
```

```

        bytes32[] memory item_name_bytes_list = new bytes32[](uint256(entries
.size()));

        bool is_insert = true;
        for(int i=0; i<entries.size(); ++i) {
            entry = entries.get(i);

            user_name_bytes_list[uint256(i)] = entry.getBytes32("name");
            item_id_list[uint256(i)] = entry.getInt("item_id");
            item_name_bytes_list[uint256(i)] = entry.getBytes32("item_name");
            if(compare(user_name_bytes_list[uint256(i)], name) && compare(item_name_bytes_list[uint256(i)], item_name)){
                is_insert = false;
                update(name, item_id + item_id_list[uint256(i)], item_name);
            }
        }
        if(is_insert){
            insert(name, item_id, item_name);
        }
    }
}

```

在此之前,所有的函数都依赖于 Table.sol, 具体内容在 Fisco 官网上有, 在此不再贴出代码.

## 功能二：实现应收账款的转让上链.

在合约中实现 buyWithLoan 函数, 例如轮胎厂要采购轮毂厂的产品, 但是自己目前没有足够的资金来支付, 目前车企与轮胎厂之间已经签订了一笔应收账款, 所以轮胎厂可以用在区块链上记录的应收账款来向轮毂厂采购商品, 从而解决资金不足的问题.

```

// @brief 应收账款转让上链
// owe_name 欠 item_name 钱, item_name 想购买 name 的东西, 即将部分债款转给 name
function buyWithLoan(int part_money, int item_id, string name, string item_name, string owe_name) public{
    TableFactory tf = TableFactory(0x1001);
    Table table = tf.openTable("t_test");

    Condition condition = table.newCondition();
    Entries entries = table.select(owe_name, condition); // 欠谁的钱
    Entry entry;

    // 减少 owe_name 与 name 之间的借款
    bytes32[] memory user_name_bytes_list = new bytes32[](uint256(entries.size()));
    int[] memory item_id_list = new int[](uint256(entries.size()));
    bytes32[] memory item_name_bytes_list = new bytes32[](uint256(entries.size()));
}

```

```

    for(int i=0; i<entries.size(); ++i) {
        entry = entries.get(i);

        user_name_bytes_list[uint256(i)] = entry.getBytes32("name");
        item_id_list[uint256(i)] = entry.getInt("item_id");
        item_name_bytes_list[uint256(i)] = entry.getBytes32("item_name");
        if(compare(user_name_bytes_list[uint256(i)], owe_name) && compare(item_name_bytes_list[uint256(i)], item_name)){
            update(owe_name, item_id_list[uint256(i)] - part_money, item_name);
            break;
        }
    }

    buy(owe_name, part_money, name);

    buy(item_name, item_id - part_money, name);
}

```

**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。

在合约中实现 loanWithBank 函数。例如轮毂厂想要利用车厂与自己签订的应收账款来向银行申请融资，此时银行会查询区块链上的交易内容，确认大企业与小企业之间确实存在应收账款，此时银行会给小企业一定的信用额度，在这里我设置信用额度为大企业与小企业之间应收账款的金额。

```

// @brief 小企业向银行申请融资
// 向银行借款，因为有大企业欠自己的钱
// 银行先查询是否确实有这笔账款，然后给定额度，判断是否通过申请
function loanWithBank(string name, string company_name, int money) public{
    TableFactory tf = TableFactory(0x1001);
    Table table = tf.openTable("t_test");

    Condition condition = table.newCondition();
    Entries entries = table.select(company_name, condition); // 查询大企业欠谁的钱

    Entry entry;

    // 向银行借款，额度为大企业借自己的钱的上限
    bytes32[] memory user_name_bytes_list = new bytes32[](uint256(entries.size()));
    int[] memory item_id_list = new int[](uint256(entries.size()));
    bytes32[] memory item_name_bytes_list = new bytes32[](uint256(entries.size()));

    for(int i=0; i<entries.size(); ++i) {

```

```

        entry = entries.get(i);

        user_name_bytes_list[uint256(i)] = entry.getBytes32("name");
        item_id_list[uint256(i)] = entry.getInt("item_id");
        item_name_bytes_list[uint256(i)] = entry.getBytes32("item_name");
        if(compare(user_name_bytes_list[uint256(i)], company_name) && compare
(item_name_bytes_list[uint256(i)], name)){
            if(money <= item_id_list[uint256(i)]){
                insert(name, money, "Bank");
            }
            break;
        }
    }
}

```

**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。

在合约中实现 payMoney 函数，企业之间只要有相应的应收账款都可以通过此函数来支付，此时区块链会记录下来，作为以后的查询凭证。

```

// @brief 应收账款支付
// 支付欠款，如果为 0 则销毁欠款账单
function payMoney(string name, int money, string boss_name) public{
    TableFactory tf = TableFactory(0x1001);
    Table table = tf.openTable("t_test");

    Condition condition = table.newCondition();
    Entries entries = table.select(name, condition); // 查询 name 欠谁的钱
    Entry entry;

    bytes32[] memory user_name_bytes_list = new bytes32[](uint256(entries.size()));
    int[] memory item_id_list = new int[](uint256(entries.size()));
    bytes32[] memory item_name_bytes_list = new bytes32[](uint256(entries.size()));

    for(int i=0; i<entries.size(); ++i) {
        entry = entries.get(i);

        user_name_bytes_list[uint256(i)] = entry.getBytes32("name");
        item_id_list[uint256(i)] = entry.getInt("item_id");
        item_name_bytes_list[uint256(i)] = entry.getBytes32("item_name");
        if(compare(user_name_bytes_list[uint256(i)], name) && compare(item_name_bytes_list[uint256(i)], boss_name)){
            if(money < item_id_list[uint256(i)]){
                update(name, item_id_list[uint256(i)] - money, boss_name);
            }else if(money == item_id_list[uint256(i)]){

```

```

        remove(name, boss_name);
    }
    break;
}
}
}
}

```

## 3.2 核心功能

链端核心功能其实为功能三，目的就是为了解决传统金融领域中小企业融资困难的问题，但是核心功能又要依赖于其他功能，所以只有其他功能完善才能保证核心功能不出错。在这里举个核心功能的例子，车企为大企业，轮胎厂，轮毂厂，钢铁厂分别为小企业，车企与轮胎厂之间存在一定金额的应收账款，轮胎厂与轮毂厂之间存在应收账款，而钢铁厂与轮毂厂之间存在应收账款，假设所有的应收账款金额都比较大，此时钢铁厂作为小厂，想要申请银行的高额融资，这在传统的金融供应链领域中基本上是不可能的事情，因为传统的金融供应链中银行会判定钢铁厂的信用额度不足以申请高额贷款，所以会导致钢铁厂无法融资。但是如果结合了区块链，所有的交易信息都记录在了区块链中，银行可以轻易的查询到每一次交易，并且可以确保每一次交易的真实性，所以银行有足够的资料来改变对小企业的信用额度评判，从而可以实现让小企业进行大额度的融资。

## 3.3 附加功能

链端通过加载 PEM 格式的私钥文件使用账户。首先在 applicationContext.xml 中配置 PEM 账户的私钥文件路径。

```

<bean id="pem" class="org.fisco.bcos.channel.client.PEMManager" init-method="load"
>
    <property name="pemFile"
value="classpath:0xcdc6e60801c0a2e6bb534322c32ae528b9dec8d2.pem" />
</bean>

```

然后在 client 中添加代码加载相应私钥和公钥

```

PEMManager pem = context.getBean(PEMManager.class);
ECKeypair pemKeyPair = pem.getECKeypair();
Credentials credentialsPEM = GenCredential.create(pemKeyPair.getPrivateKe
y().toString(16));

```

目的是防止其他用户恶意篡改数据



## 4. 后端

### 4.1 框架与语言选择

后端我使用的框架是 Spring, 对应语言为 Java, 因为 Fisco 上面的 Python SDK 与 Node.js SDK 并不是很完善, 所以选择了 Java 进行开发.

### 4.2 具体实现

首先将链端的合约通过 `sol2java.sh` 转换为 java, 根据 fisco 官网教程修改 `gradle` 配置文件 `build.gradle`, 使其能使用 web3 SDK 的同时可以启动 `spring` 框架. 这样一来后端和链端就已经通过相应的接口联通了, 其中调用 SDK 进行部署和查询的函数如下.

#### 4.2.1 部署

```
// @brief Deploy
// 部署合约
public void deployAssetAndRecordAddr() {

    try {
        // 部署自己的合约
        // load()加载合约, 需要知道合约地址
        String buy_address = "0xdbaf5d709a0d3545a519d09d467af5f914eba085";
        BuySomething buysth = BuySomething.load(buy_address, web3j, credentials, new StaticGasProvider(gasPrice, gasLimit));

        //BuySomething buysth = BuySomething.deploy(web3j, credentials, new StaticGasProvider(gasPrice, gasLimit)).send();

        System.out.println(" deploy BuySomething success, contract address is " + buysth.getContractAddress());

        System.out.println(" deploy Asset success, contract address is " + asset.getContractAddress());

        recordAssetAddr(asset.getContractAddress());
    } catch (Exception e) {
        // TODO Auto-generated catch block
        // e.printStackTrace();
        System.out.println(" deploy Asset contract failed, error message is " + e.getMessage());
    }
}
```

#### 4.2.2 查询

```

// @brief 功能一 --> 查询 Select
// select the data in table
public List<String> Select(String name) {
    List<String> str = new ArrayList<String>();
    try {
        String buy_address = "0xdbaf5d709a0d3545a519d09d467af5f914eba085";
        BuySomething buysth = BuySomething.load(buy_address, web3j, credentials, new StaticGasProvider(gasPrice, gasLimit));

        Tuple3<List<byte[]>, List<BigInteger>, List<byte[]>> result = buysth.select(name).send();

        System.out.println("<----- select ----->");
        int len = result.getValue1().size();
        for(int i = 0; i < len; ++i){
            String first_name = new String(result.getValue1().get(i));
            System.out.println(first_name);
            String tmp = new String(first_name);
            tmp += "$";

            System.out.println(result.getValue2().get(i));
            tmp += result.getValue2().get(i).toString();
            tmp += "$";

            String second_name = new String(result.getValue3().get(i));
            System.out.println(second_name);
            tmp += second_name;
            str.add(tmp);
        }
        System.out.println("<----- select ----->");

    }catch(Exception e){
        logger.error(" queryAssetAmount exception, error message is {}", e.getMessage());

        System.out.printf(" query asset account failed, error message is %s\n", e.getMessage());
    }
    return str;
}

```

### 4.2.3 Spring Controller 代码

接下来利用 Spring 实现与前端的交互, 通过响应前端的 Get 与 Post 请求来调用相应的处理函数, 因为 Spring 上手比较容易所以就不再细讲了, 这里只使用了框架中对 Http 请求处理的部分功能, 后端关键 Controller 文件 AccountController.java 的代码如下.

```
package org.fisco.bcos;
```

```

import org.fisco.bcos.asset.contract.Asset;

import java.math.BigInteger;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLConnection;

import org.fisco.bcos.asset.client.AssetClient;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.atomic.AtomicLong;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.fisco.bcos.web3j.tuples.generated.Tuple3;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import java.io.BufferedReader;
import java.io.IOException;

@Controller
public class AccountController {

    public List<String> loginUser;
    public AssetClient client;

    AccountController()throws Exception{
        System.out.println("<----- Constructor ----->");

        client = new AssetClient();
        client.initialize();
        client.deployAssetAndRecordAddr();

        loginUser = new ArrayList<String>();

        System.out.println("<----- Constructor End ----->");
    }

```

```

String removeSlash(String str){
    int len = str.length();
    int idx = 0;
    for(int i = 0; i < len; ++i, ++idx){
        if(str.charAt(i) == '/')
            break;
    }
    return str.substring(0, idx);
}

boolean findUser(String user){
    int len = loginUser.size();
    for(int i = 0; i < len; ++i){
        if(user.equals(loginUser.get(i))){
            return true;
        }
    }
    return false;
}

@RequestMapping(value = "/login")
public String indexPage() {
    return "index";
}

/**
 * @brief 用户页面，先查找 Map 中是否存在相应的用户正在登录状态
 *         如果存在则不需要密码，否则需要密码
 *
 * @param request
 * @param model
 * @return
 */
@RequestMapping(value = "/user")
public String login(HttpServletRequest request, Model model) {
    String username = request.getParameter("username");
    String pwd;
    username = removeSlash(username);
    if(findUser(username)){
        model.addAttribute("username", username);
        return "Account";
    }else if(!is_exist){
        return "usernoexist";
    }else{
        return "wrongpwd";
    }
}

// @brief Select index
@RequestMapping(value = "/user/select")

```

```

public String select(HttpServletRequest request, Model model) throws Exception{
    String username = request.getParameter("username");
    username = removeSlash(username);
    System.out.println("<----- Select ----->");

    System.out.println("<----- Select End----->");
    model.addAttribute("username", username);
    return "select";
}

@RequestMapping(value = "/user/select/result")
@ResponseBody
public Account selectResult(HttpServletRequest request) throws Exception{
    System.out.println("<----- Select Result ----->");
    String username = request.getParameter("username");
    System.out.println(username);
    username = removeSlash(username);

    Account res = new Account();
    List<String> str = new ArrayList<String>();
    str = client.Select(username);
    for(int i = 0; i < str.size(); ++i){
        String tmp = str.get(i);
        String name = new String();
        String count = new String();
        String oweName = new String();
        int choice = 0;
        int idx = 0;
        for(int j = 0; j < tmp.length(); ++j){
            if(tmp.charAt(j) == '$'){
                if(choice == 0){
                    name = tmp.substring(idx, j);
                    idx = j + 1;
                }else if(choice == 1){
                    count = tmp.substring(idx, j);
                    idx = j + 1;
                    oweName = tmp.substring(idx, tmp.length());
                    break;
                }
                ++choice;
            }
        }
        name = name.replaceAll("\u0000", "");
        oweName = oweName.replaceAll("\u0000", "");
        res.data.add(new UserData(name, count, oweName));
        System.out.println(str.get(i));
    }

    System.out.println("<----- Select Result End----->");
    return res;
}

```

```

}

// @brief Buy
@RequestMapping(value = "/user/buy")
public String buy(HttpServletRequest request, Model model) {
    String username = request.getParameter("username");
    username = removeSlash(username);
    model.addAttribute("username", username);
    model.addAttribute("method", "buy");
    return "buy";
}

// @brief BuyWithLoan
@RequestMapping(value = "/user/buyWithLoan")
public String buyWithLoan(HttpServletRequest request, Model model) {
    String username = request.getParameter("username");
    username = removeSlash(username);
    model.addAttribute("username", username);
    model.addAttribute("method", "buyWithLoan");
    return "buyWithLoan";
}

// @brief LoanWithBank
@RequestMapping(value = "/user/loanWithBank")
public String loanWithBank(HttpServletRequest request, Model model) {
    String username = request.getParameter("username");
    username = removeSlash(username);
    model.addAttribute("username", username);
    model.addAttribute("method", "loanWithBank");
    return "loanWithBank";
}

// @brief PayMoney
@RequestMapping(value = "/user/payMoney")
public String payMoney(HttpServletRequest request, Model model) {
    String username = request.getParameter("username");
    username = removeSlash(username);
    model.addAttribute("username", username);
    model.addAttribute("method", "payMoney");
    return "payMoney";
}

// @brief 处理成功
@RequestMapping(value = "/user/success")
public String success(HttpServletRequest request, Model model) {
    String username = request.getParameter("username");
    String count = request.getParameter("count");
    String company = request.getParameter("company");
    String oweCompany = request.getParameter("owecompany");
    String partMoney = request.getParameter("partmoney");

```

```

String bossCompany = request.getParameter("bosscompany");

String tradeMethod = request.getParameter("method");

username = removeSlash(username);
tradeMethod = removeSlash(tradeMethod);

model.addAttribute("username", username);
System.out.println("<----- username: " + username + " ----->");
System.out.println("<----- method: " + tradeMethod + " ----->");

if(tradeMethod.equals("buy")){
    client.Buy(username, new BigInteger(count), company);
}else if(tradeMethod.equals("buyWithLoan")){
    client.BuyWithLoan(username, new BigInteger(count), company, oweCompany,
new BigInteger(partMoney));
}else if(tradeMethod.equals("loanWithBank")){
    client.LoanWithBank(username, company, new BigInteger(count));
}else if(tradeMethod.equals("payMoney")){
    client.PayMoney(username, new BigInteger(count), bossCompany);
}else{
    System.out.println("<----- No Method ----->");
}

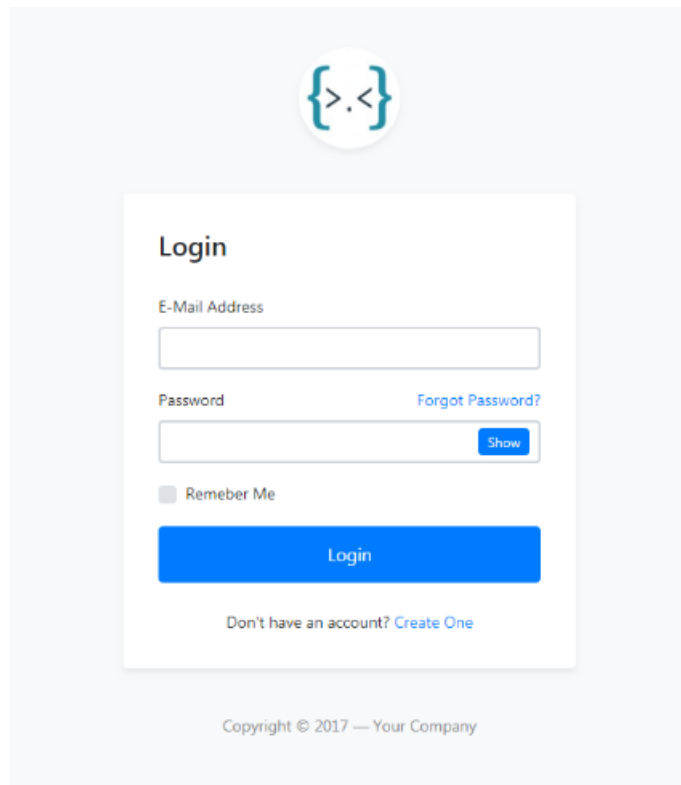
return "success";
}
}

```

## 5. 前端

### 5.1 登录界面

因为自己写出来的页面效果不好, 所以登录页面使用了 github 上现有的 html 文件, 具体网址为 <https://github.com/nauvalazhar/bootstrap-4-login-page>

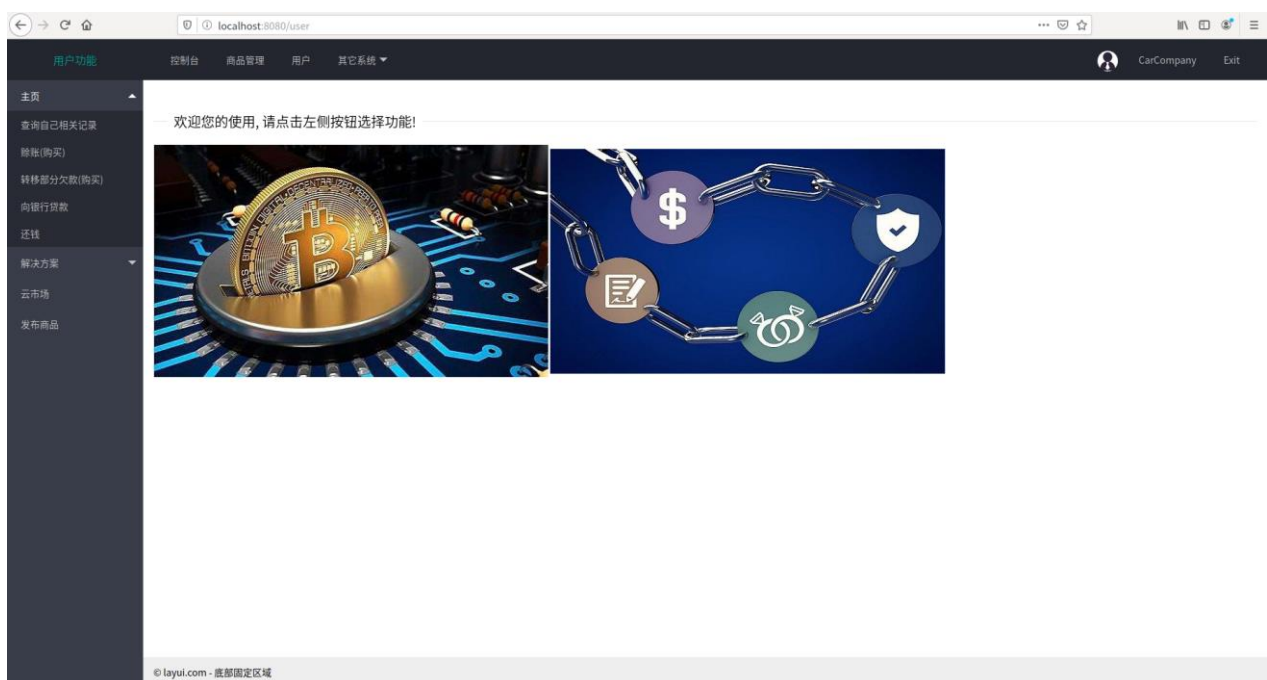


当然这只是一个 html 与 css 文件而已, 具体的登录以及用户和密码的管理都是自己在其基础上添加的.

## 5.2 用户界面

用户界面使用了 layui 的组件, 网址为 <https://www.layui.com/>

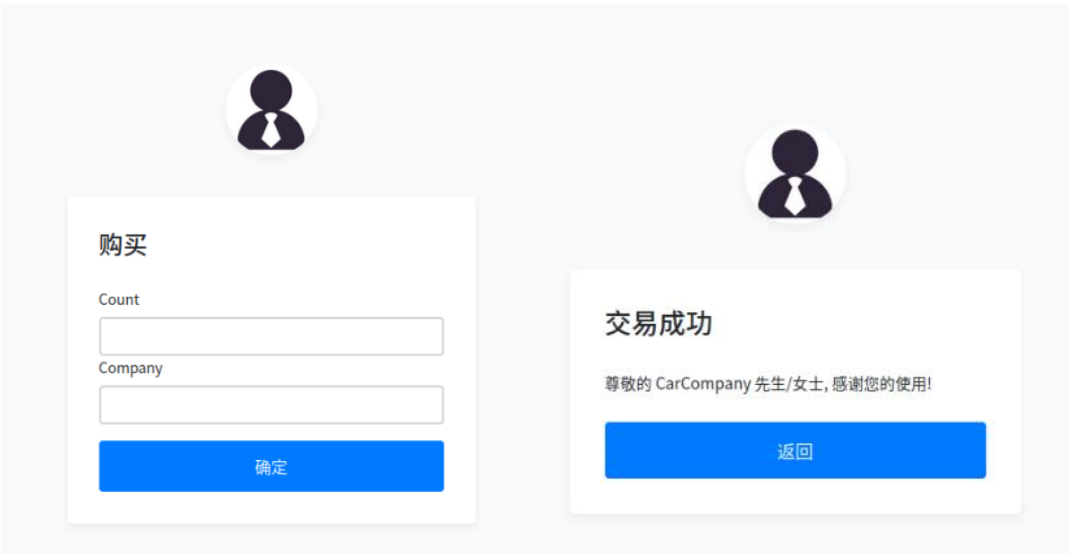
其实比较像 Webase...





### 5.3 操作界面

点击按钮后会出现相应的页面进行操作, 灵感来自于支付宝的支付操作, 先要填写支付金额并确定, 完成后转到一个操作成功的界面. 如下图所示



## 三. 功能测试

### 1. 功能一

车厂对轮胎厂签订 2000 的应收账款



### 2. 功能二

轮胎厂对轮胎厂签订 1000 的应收账款, 其中 500 由车厂承担.

### Buy With Loan

Count

Company

Owe Company

Part Money

确定

### TyreCompany

用户名	金额	欠款企业
TyreComp...	500	HubCompar

返回

### 交易成功

尊敬的 HubCompany 先生/女士, 感谢您的使用!

返回

### 3. 功能三(核心功能)

把所有账单清零, 让车企与轮胎厂之间签订 4000 的应收账款, 轮胎厂与轮毂厂之间签订 3500 的应收账款, 轮毂厂申请银行融资, 需要提供一个厂商的名字, 这里提供了轮胎厂, 因为轮胎厂也是小厂, 在传统金融供应链中融资一定会失败, 但是有了区块链后, 这里的信用额度转移制度将车企信用额度的一部分转移到了轮胎厂, 而轮胎厂将信用额度的一部分转移到了轮毂厂, 所以现在轮毂厂能够在银行中进行高额贷款, 只需要提供一个 Company 的名字与相应的贷款金额, 银行不仅会核实轮毂厂的信用额度, 而且会在区块链中核实所提供的 Company 是否确实与轮毂厂之间存在应收账款. 一经核实后就可以根据信用额度给相应的小企业进行融资.

### CarCompany

用户名	金额	欠款企业
CarCompany	4000	TyreComp...

返回

### TyreCompany

用户名	金额	欠款企业
TyreComp...	3500	HubCompar

返回

### Loan With Bank

Company

Count

确定



HubCompany

用户名	金额	欠款企业
HubCompany	3700	Bank

返回




交易成功

尊敬的 HubCompany 先生/女士, 感谢您的使用!

返回

## 4. 功能四


假设车企欠轮胎厂 1500, 欠轮毂厂 500, 现在让车企还清轮毂厂的钱. 对轮胎厂进行分期付款, 先支付 1000, 如下图所示.



CarCompany

用户名	金额	欠款企业
CarCompany	1500	TyreComp...
CarCompany	500	HubCompar

返回



Pay Money


Count

500

Boss Company

HubCompany


确定



CarCompany

用户名	金额	欠款企业
CarCompany	1500	TyreComp...

返回



Pay Money


Count

1000

Boss Company

TyreCompany

确定



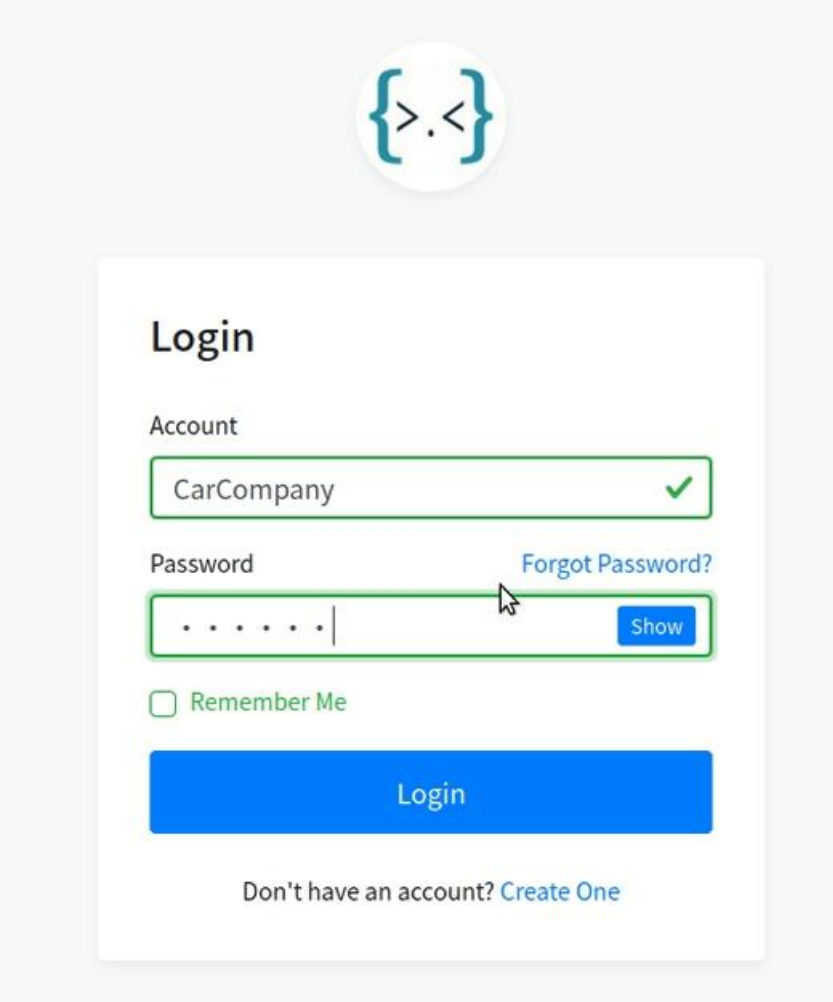
CarCompany

用户名	金额	欠款企业
CarCompany	500	TyreComp..

返回

## 四. 界面展示

### 1. 登录界面



The image shows a login form titled "Login" centered on a light gray background. Above the form is a circular logo containing a blue code symbol: `{>.<}`. The form itself is a white card with a green border. It contains the following elements: a label "Account" above a text input field with "CarCompany" and a green checkmark; a label "Password" above a password input field with six dots, a "Show" button, and a "Forgot Password?" link; a checkbox labeled "Remember Me"; a large blue "Login" button; and a link "Don't have an account? Create One" at the bottom.

**Login**

Account

CarCompany ✓

Password [Forgot Password?](#)

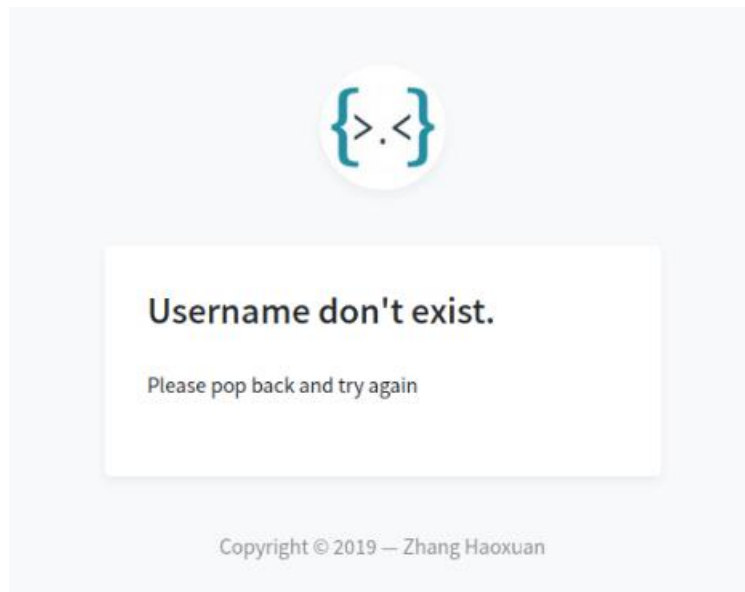
..... | Show

☐ Remember Me

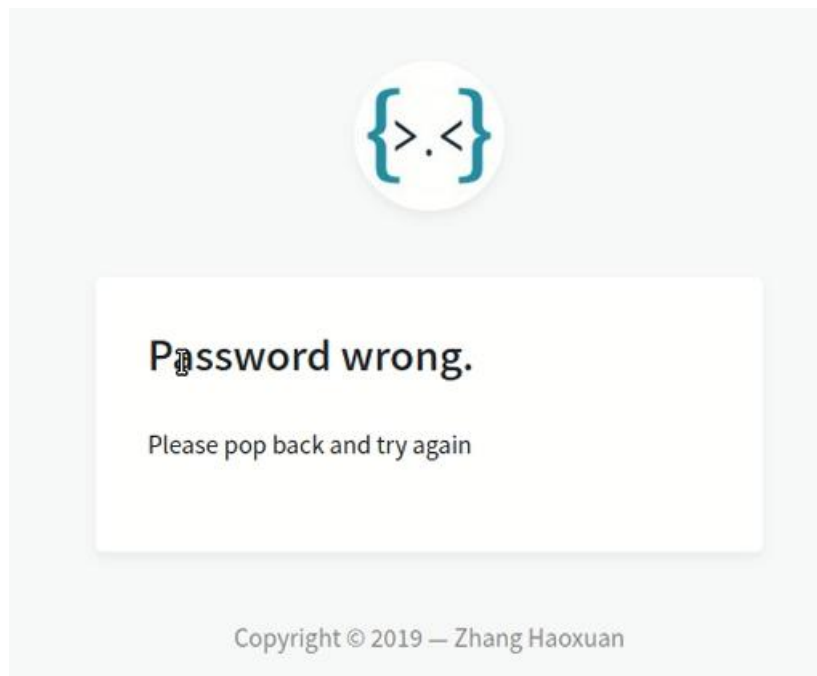
Login

Don't have an account? [Create One](#)

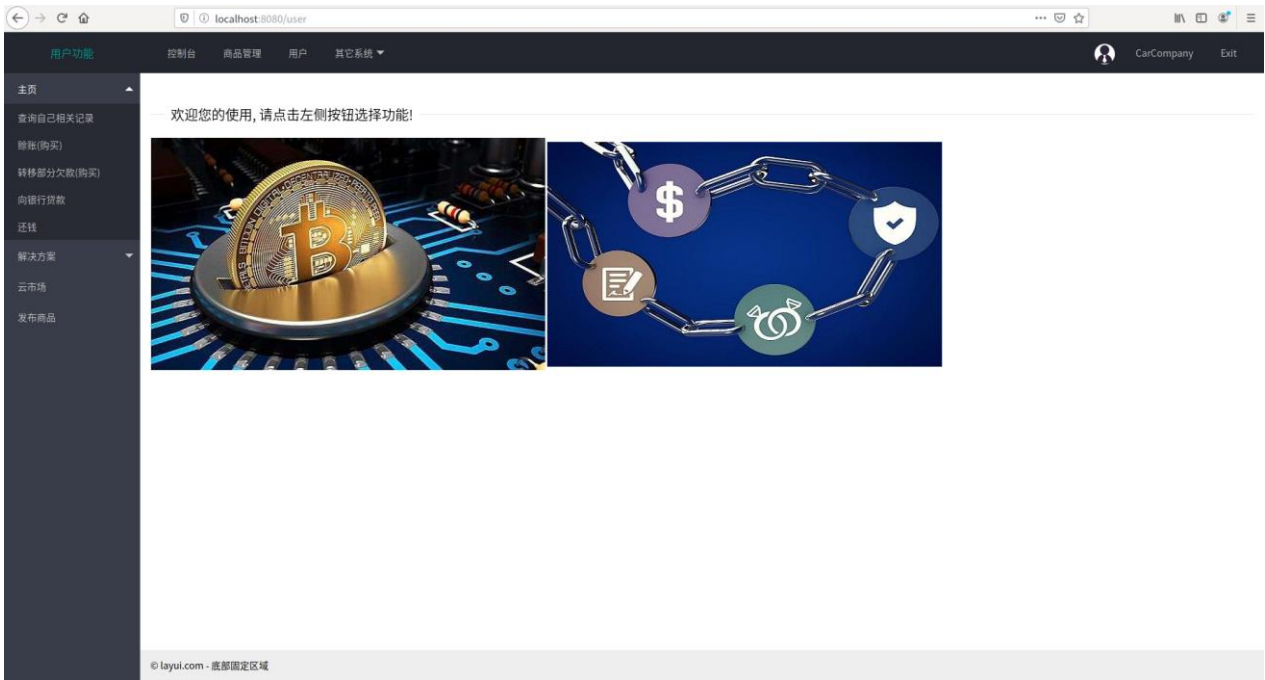
### 2. 用户不存在界面



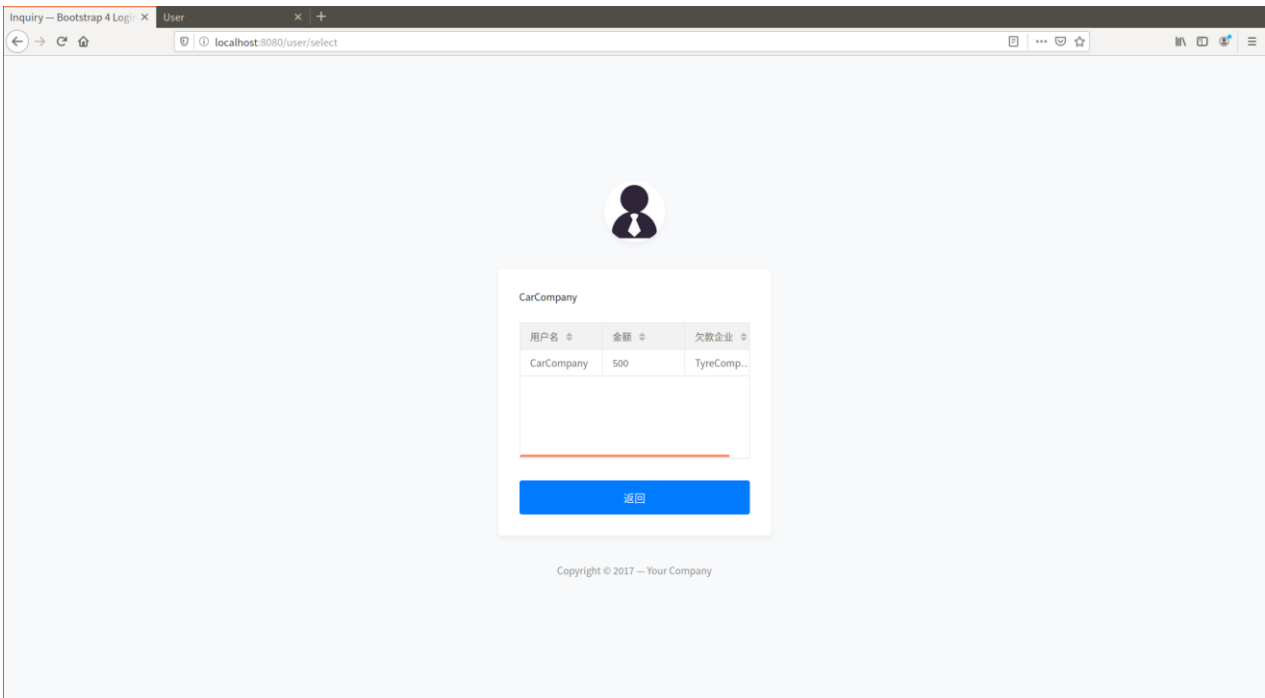
### 3. 密码错误界面



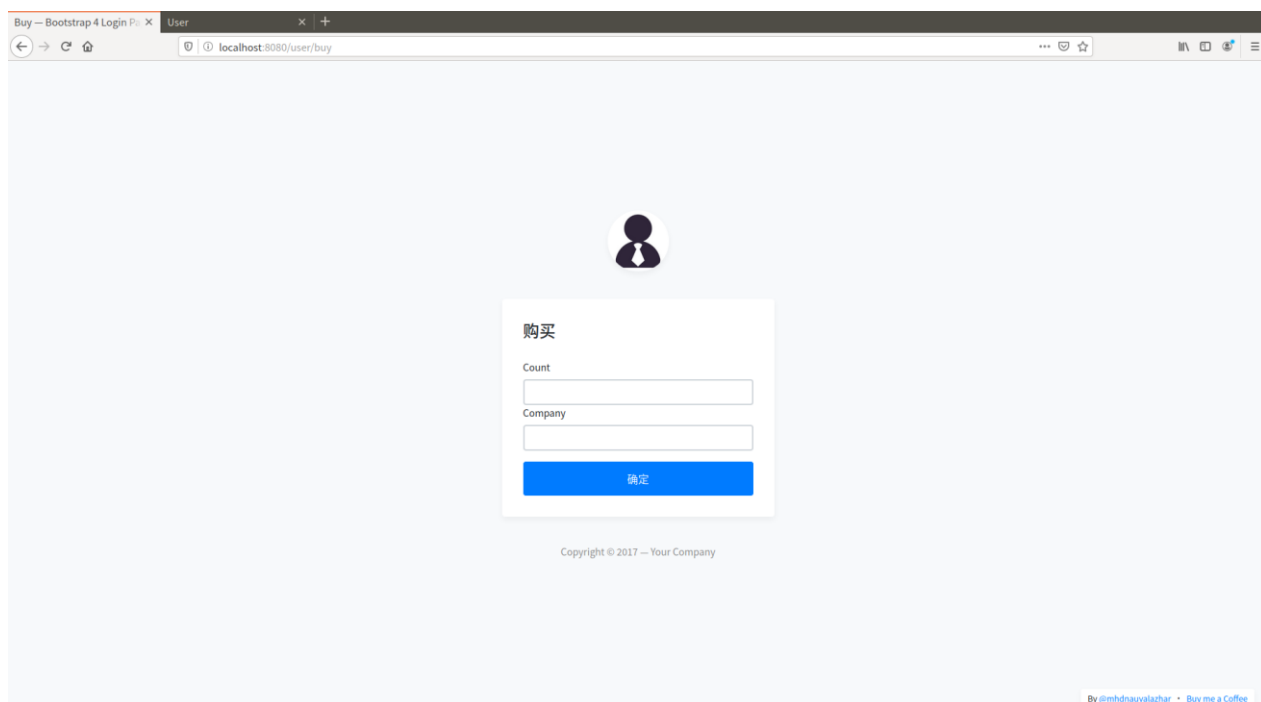
### 4. 用户界面



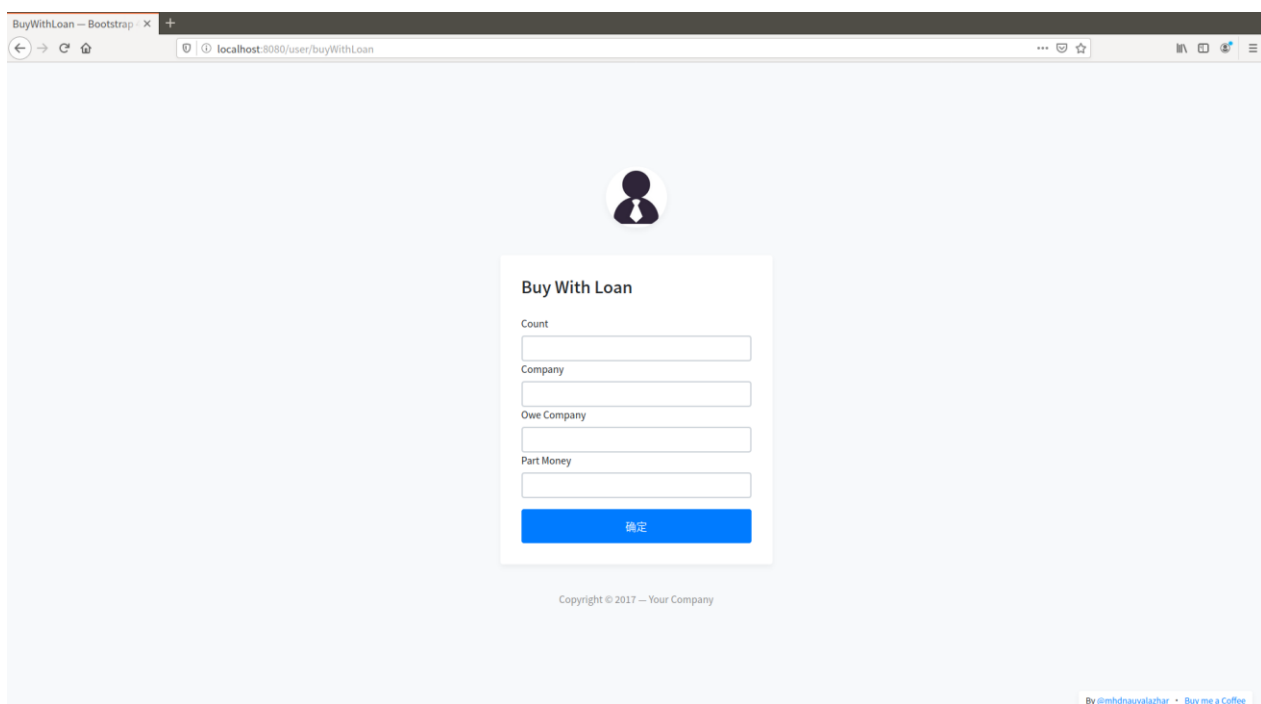
## 5. 查询界面



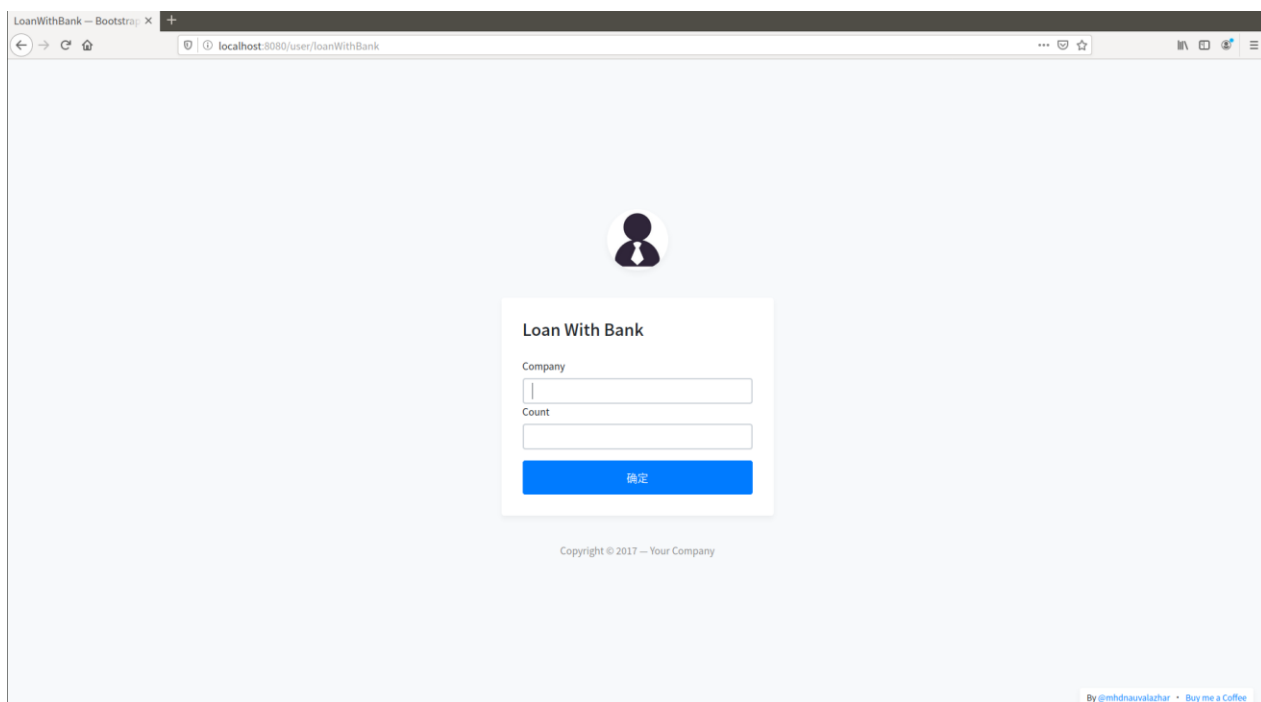
## 6. 签订应收账款界面



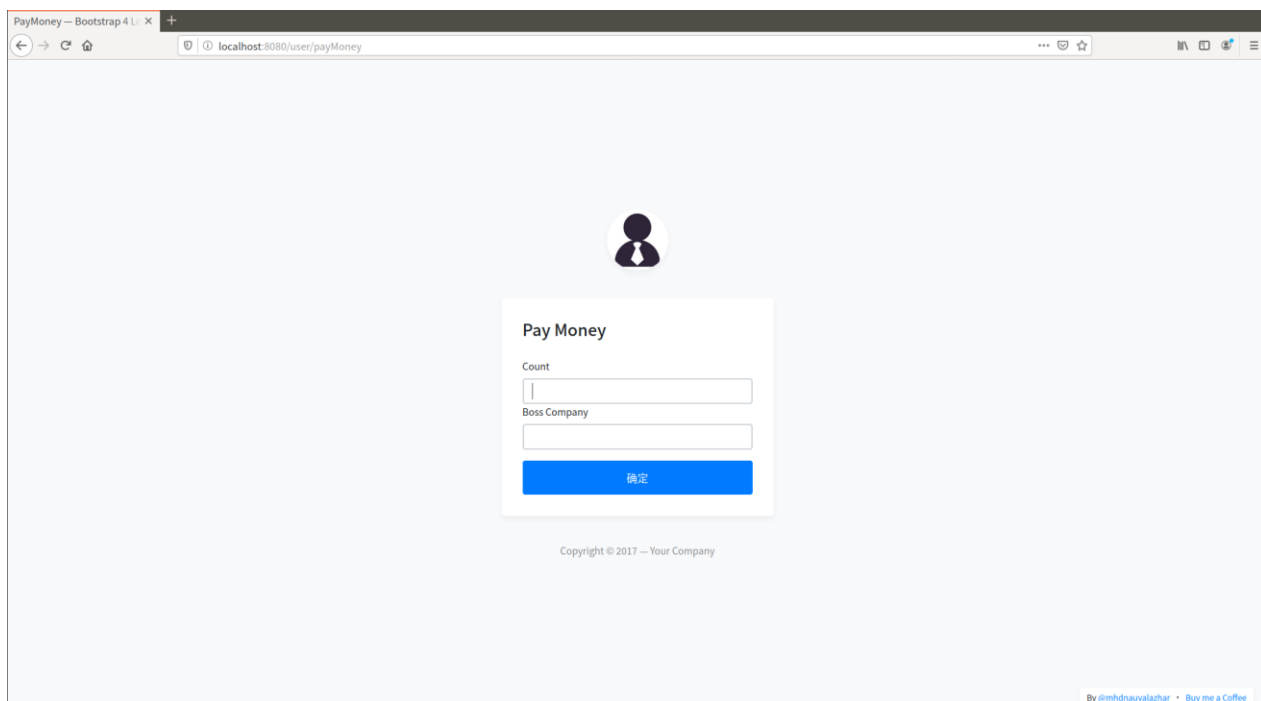
## 7. 转移账款界面



## 8. 小企业银行融资界面



## 9. 到期还钱界面





## 五. 心得体会

这次大作业的难度相比前面的准备阶段的难度简直不在一个层面上, 所以自己也用了大概 4 天的时间来完成这个作业, 其中我觉得最难的地方在于前端组件的调用和后端 Spring 框架实现, Spring 倒还好, 属于主流的框架, 官网的教程也很多, 甚至我自己在 Github 上对 Spring 某个教程提出的 Issue 也在一个晚上得到了 Spring 官方的回应, 并在第二天下午我再去查看的时候官方已经把教程完善了, 维护的速度非常快.

前端组件的调用是比较坑的, 我用的是 layui, 因为它的 html 与 css 文件直接拷过来就能快速上手, 但是其中表格部分也出了一些 bug, 大概就是发送的 POST 请求总得不到 Spring 后端的响应, 后来一查就没有发送请求, 但是表格框架在一个空白页面上是对的. 最后发现 layui 的表格框架竟然与 jquery 冲突了...这个 bug 也耗费了我一个下午的时间.

其实如果经常使用前端或者做相关的开发的话, 倒也没有多难, 但是距我上次使用前端开发已经快半年了, 所以生疏是必然的, 也就意味着要踩不少的坑, 事实上踩的坑还真不少, 所以这次作业对我来讲才这么耗时间, 前后大概用了 4 天的时间才完成.

个人收获相对来讲是非常大的, 首先回炉重造了前端部分, 使用了一直以来听得多但是没用过的 Spring 框架, 并结合 Fisco 完成了一个比较简单的区块链项目, 而且前端, 后端, 链端都是自己开发的, 所以个人经过此次开发后对前后端处理过程以及其中的细节有了更深层次的认识. 在这里要感谢老师和 TA 一个学期以来的辛苦栽培, 让区块链这个听起来很迷的名词在我的词典中有了更加深刻的解释.