



**哈尔滨工业大学(威海)**  
Harbin Institute of Technology at Weihai

# 网络空间安全设计与实践 I

## 报告

设计题目：基于 B2C 型电子商务应用  
与安全平台系统的设计

班 级：2104301

学 号：2021211030

姓 名：包赛龙

验收日期：2023/12/29

哈尔滨工业大学（威海）计算机学院

二零二三年十二月

# 1 绪论

## 1.1 设计项目简介

选题 3、基于 B2C 型电子商务应用与安全平台系统的设计

具体要求：

(1)电子商务前台功能设计：包括用户管理、商品管理、订单管理、购物车设计、物流管理、支付管理。

(2)电子商务后台：包括用户管理、商品管理、订单管理。

(3)安全技术：系统要有保障电子商务系统安全的技术实现(比如加密、防攻击等技术)使用前端 vue+后端 flask+mysql 数据库进行开发。

使用方法：进入 code>frontend，打开 run\_frontend.bat。进入 code>backend>venv 打开 run\_backend.bat。进入 code>backend,运行 b2c.sql 创建数据库。然后创建使用账户 admin, 密码 200305 登录管理员账户。其他用户均可以注册使用。如果想要创建其他的管理员账户需要手动在数据库中添加。

## 1.2 设计目的

本设计旨在构建一个基于 B2C 型电子商务应用与安全平台系统，以满足用户需求并确保系统安全的高效系统。具体设计目的包括：

提供用户友好的前台功能：设计具有良好用户体验的电子商务前台功能，包括用户管理、商品管理、订单管理、购物车设计、物流管理和支付管理等。用户能够方便地浏览和购买商品，管理订单，使用购物车进行批量购买，并追踪物流状态，以及安全便捷地进行支付操作。

实现高效的后台管理：设计基于 B2C 型电子商务应用的后台管理功能，包括用户管理、商品管理和订单管理等。后台管理员能够高效地管理用户信息，管理商品的上架、下架和库存等，以及处理订单的确认、取消和退款等。后台管理功能的设计旨在提高系统的运营效率和管理便利性。

实施安全技术保障：系统要实现多种安全技术来保障电子商务系统的安全性。其中包括数据加密技术，确保用户数据在传输和存储过程中的隐私安全,sql 防注入技术，确保数据库不受注入攻击的影响。

## 1.3 设计内容

前端：

CartAdmin.vue 购物车设计

GoodsAdmin.vue 商品管理

LogisticsAdmin.vue 物流管理

OrdersAdmin.vue 订单管理

UsersAdmin.vue 用户管理

表 1：前端设计功能一览

前端页面	后端函数	数据库
usersAdmin(用户管理)	Check(确认是否为管理员) Getallusers(获取所有用户信息)	users
Login.vue(登陆管理)	Login,register(根据账户生成)	users

	密钥,用于加密) Getpublickey(获取公钥, 用于 传输密码加密)	
CartAdmin.vue(购物车管理)	Getcart(获取购物车信息) Updatecart(更新数据库购物车 信息) Deleteitem(允许用户删除购物 车内容)	cart
LogisticsAdmin.vue(物流管理)	Getlogistics(获取物流信息) Update(允许用户进行物流状 态更新)	logistics
ordersAdmin.vue(订单管理)	Getorders(获取订单信息)	orders
Payadmin(支付管理)	Getpayment(获取支付信息) Getorders(获取订单信息)	Orders Bankcard
GoodsAdmin(商品管理)	sell(上架) buy(添加至购物车) getgoods(传递给前端所有商 品的信息)	Cart Goods
无	rsa_key_to_pem(将 rsa 的密钥 转化为 pem 格式) pem_to_rsa_key(将 pem 转化 为 rsa) get_db(获取数据库的参数)	

## 1.4 应用范围

本系统适用于顾客, 商家在日常购买, 出售商品时的需求。允许商家发布商品和管理物流, 允许顾客购买商品和查看物流, 允许管理员查看订单内容, 对用户进行修改。并且对客户的登陆密码进行了 RSA 加密处理。实现了 mysql 的防注入以及加密通信的功能。

存储了用户信息内容到数据库, 便于之后的数据分析。

## 2. 系统总体设计功能图

本次系统设计使用了前端和后端结合进行开发。前端使用了 vue 进行开发, 通过 axios 库的 post 来构建 http 报文发送请求模拟与后端的通信, 或者传递数据。后端使用了 flask 进行开发, 通过 request 库来获取前端报文的信息, 从而实现与前端进行通信。

前端绑定于本地 8080 端口, 通过浏览器可以访问。我对页面进行了定向, 如图 1 所示, 访问 localhost:8080/cart 可以访问购物车界面, localhost:8080/login 为访问 localhost:8080 时的重定向界面, 即默认界面。

另外, orders 界面存在身份验证, 非管理员用户访问时将被重定向至其他界面, 即禁止访问。

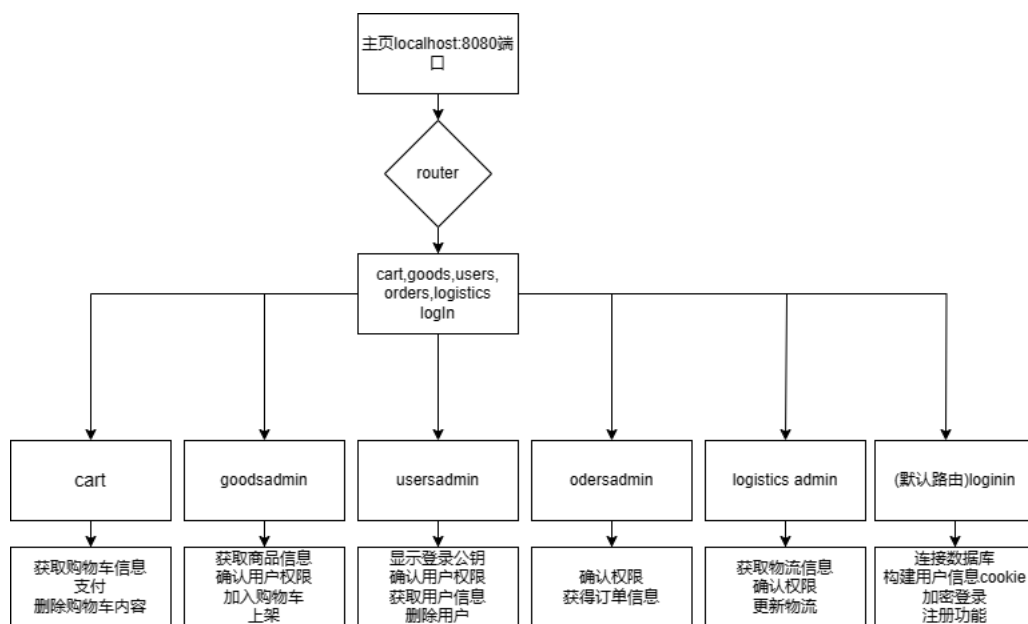


图 1 前端页面功能

后端绑定于本地 5000 端口。在实际应用中，这个地址应当放置在服务器中。在前端中，urlbase 变量为后端地址，通过这个地址加上路由名，就能够实现前后端通信。

如图 2 所示，后端功能分为三类：自用函数(主要为密钥编码,解码),数据库操作，前端通信。大致上来讲，后端主要完成数据的存储，处理功能。

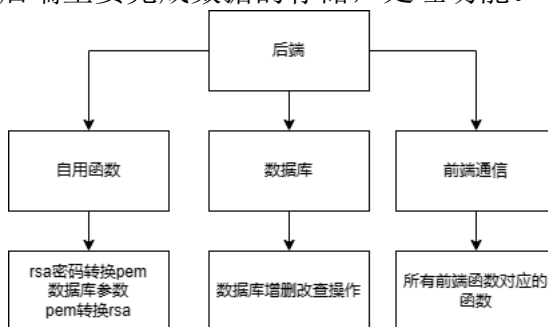


图 2 后端功能

数据库部分：除了 logistics，其他的数据库均具有主码 account。数据库具有防注入的功能，通过字符串过滤实现。

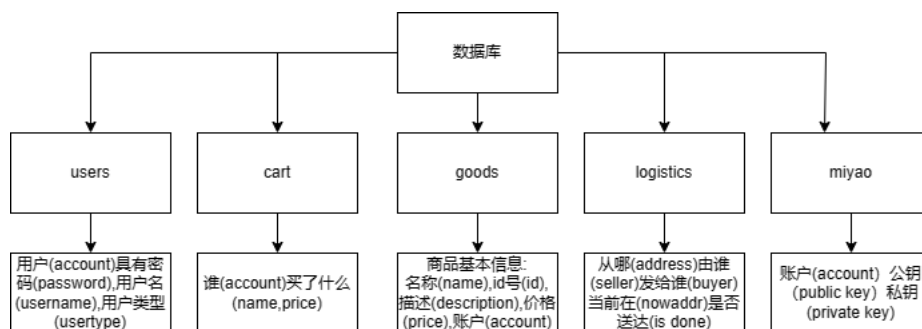


图 3.数据库功能

系统主要实现商家，管理员和顾客的基本功能，以及重要信息的加密处理。

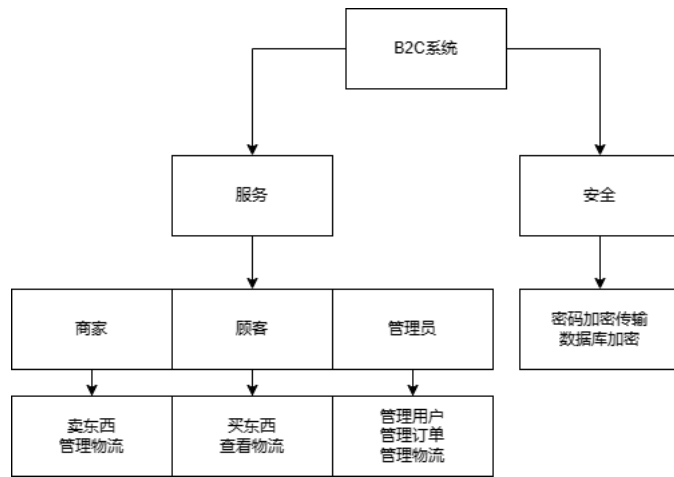


图 4.系统框架

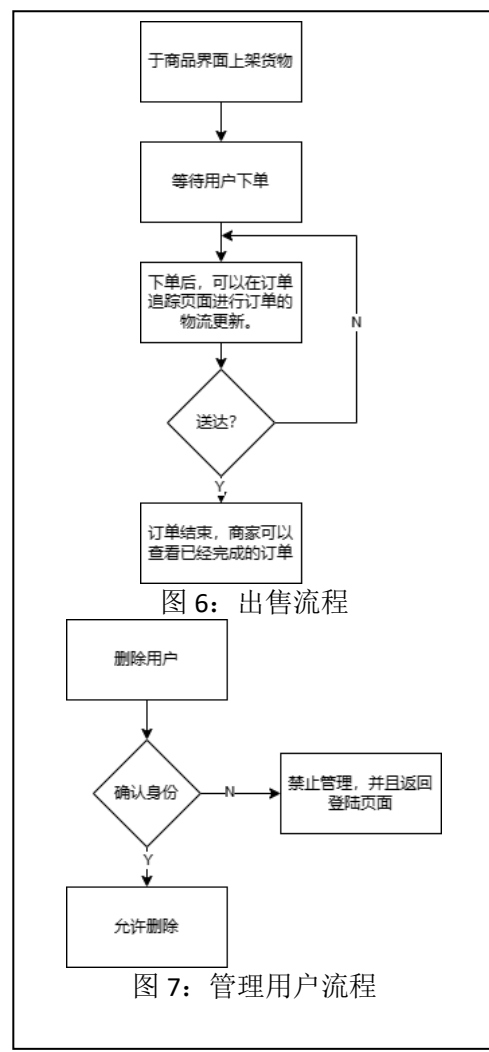
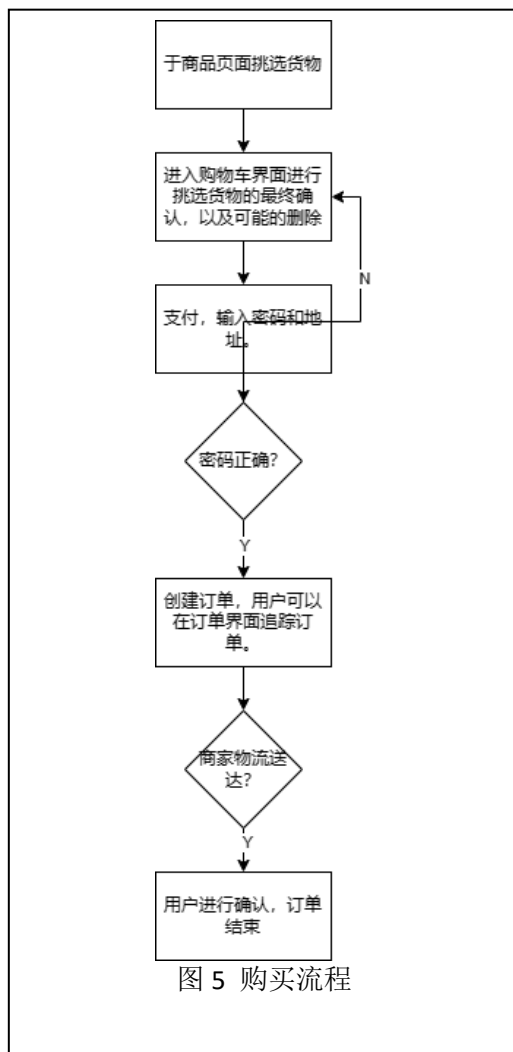
### 3. 设计思想、算法与功能流程图

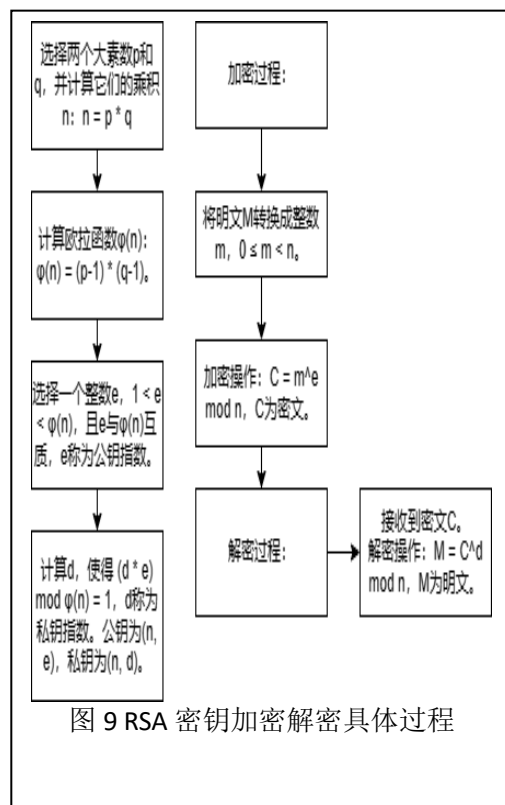
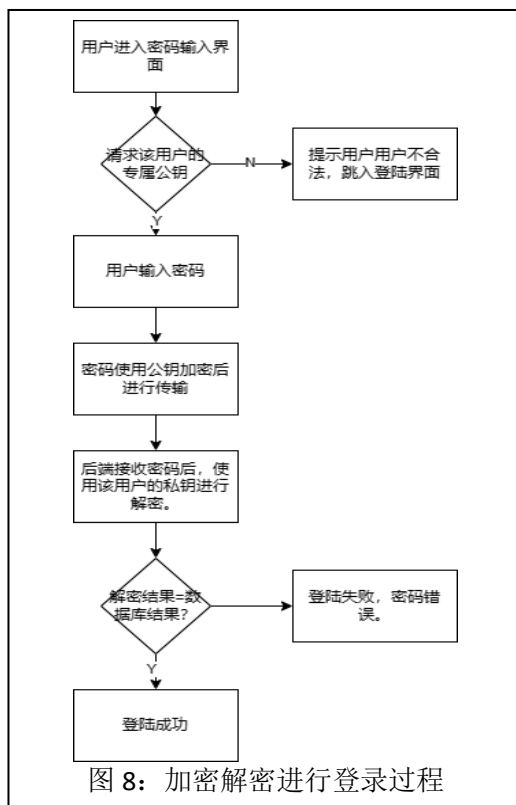
此次的设计为三种不同类型的用户提供，这三类用户都各自具有主动行为：购买，售出，管理用户。因此根据这三种行为设计流程：

首先为顾客的行为：购买，流程图图 5,商家的行为：出售，流程图如图 6，管理员的行为：管理用户，如图 7。

具体实现的功能 RSA 密钥加密密码具有在前后端进行流通的流程， 如图 8 所示。

RSA 密钥加密的具体流程，如图 9 所示





## 4. 系统功能实现

### 4.1 网页定向功能实现(路由)

给出主要功能部分的系统界面截图及核心代码来，并给出必要的描述说明。按功能模块来分节。

作用：通过 localhost:8080 加上路由符，可以访问构建生成的网站页面。

实现方式：使用 router.js 和 vue 实现了 vue 页面到网页页面的绑定。以下为部分代码以及解释：

```

import { createRouter, createWebHistory } from 'vue-router';
// 导入要显示的页面组件
import cart from './components/CartAdmin.vue';//这个是我的本地页面路径，使用相对路径
import goods from './components/GoodsAdmin.vue';
import users from './components/UsersAdmin.vue';
import payment from './components/PayAdmin.vue';
import orders from './components/OrdersAdmin.vue';
import logistics from './components/LogisticsAdmin.vue';
import login from './components/LogIn.vue';
const routes = [
  { path: '/', component: login },//默认为 login 页面
  { path: '/cart', component: cart },//把变量和路由绑定
  { path: '/goods', component: goods },
  { path: '/users', component: users },
  { path: '/payment', component: payment },
  { path: '/orders', component: orders },
  { path: '/logistics', component: logistics },

```

```

    { path: '/Login',component:login},
  ];
const router = createRouter({
  history: createWebHistory(),
  routes,
});
export default router;

```

实现后的效果如图 10,可以在浏览器中访问编写的网站

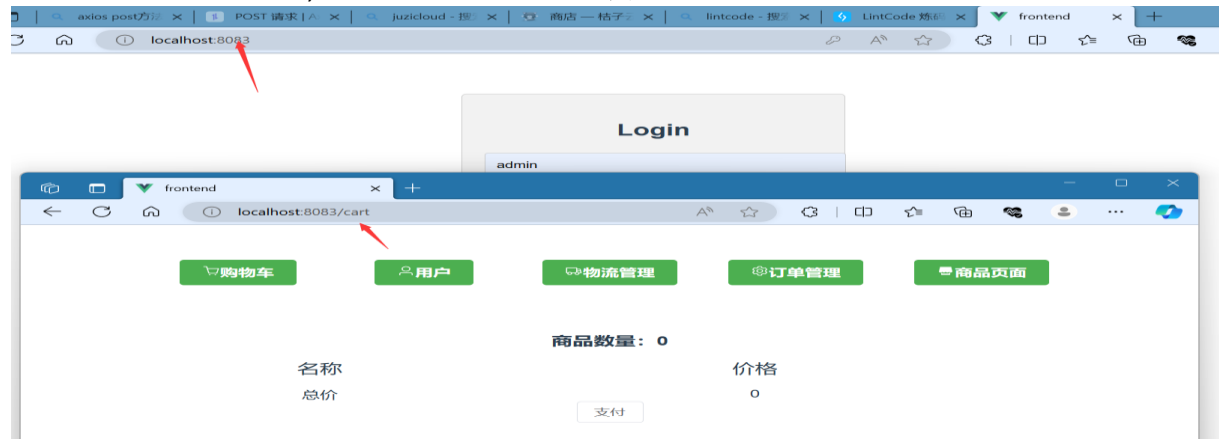


图 10

## 4.2 RSA 加密解密功能实现(前后端)

给出主要功能部分的系统界面截图及核心代码来，并给出必要的描述说明。按功能模块来分节。

首先，明确 RSA 密钥生成的时机：

当用户注册时，为该用户生成一个专属的 rsa 密钥对，存储至数据库并且与用户进行绑定。当用户发出登录请求时，由后端发给用户公钥用于加密。后端生成密钥对,并且插入数据库的代码如下，存储时，使用 pem 进行存储：

Import rsa

try:

#产生密钥和公钥

(publickey,privatekey)=rsa.newkeys(2048)

privatepem=privatekey.save\_pkcs1().decode()//进行解码后存储为 pem 格式

publicpem=publickey.save\_pkcs1().decode()

conn = get\_db()

with conn.cursor() as cursor:

cursor.execute('SELECT \* FROM users WHERE account = %s', (account))

existing\_user = cursor.fetchone()

if existing\_user:

return jsonify({'error': 'Account already exists'}), 400

# 保存用户信息到数据库

with conn.cursor() as cursor:

cursor.execute(

'INSERT INTO users (account, password, username, usertype) VALUES

(%s, %s, %s, %s)',

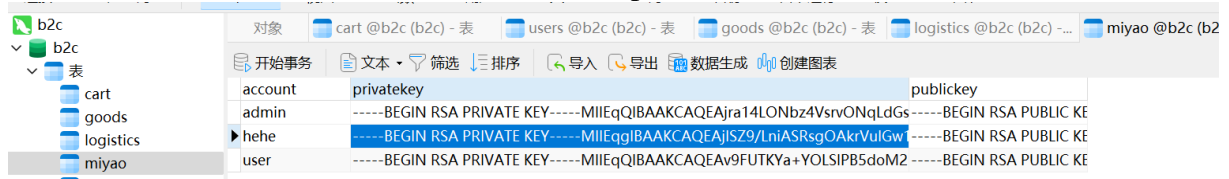
(account, password, username, usertype)

```

    )
    cursor.execute(
        'INSERT INTO miyao (account, privatekey,publickey) VALUES (%s, %s, %s)',
        (account, privatepem,publicpem)
    )
    conn.commit()
    return jsonify({'message': 'Registration successful'})
except pymysql.Error as e:
    print("Error",e)
    return jsonify({'error': 'Registration failed: {}'.format(str(e))}), 500

```

数据库中，用户的公钥私钥如图 11，使用了 pem 进行存储



account	privatekey	publickey
admin	-----BEGIN RSA PRIVATE KEY-----MIIQgQIBAAKCAQEjra14LONbz4VsrVONqLdGs-----BEGIN RSA PUBLIC KE	-----BEGIN RSA PUBLIC KE
hehe	-----BEGIN RSA PRIVATE KEY-----MIIQgQIBAAKCAQEjISZ9/LniASRsgOAkrVulGw-----BEGIN RSA PUBLIC KE	-----BEGIN RSA PUBLIC KE
user	-----BEGIN RSA PRIVATE KEY-----MIIQgQIBAAKCAQEA9FUTKYa+YOLSIPB5doM2-----BEGIN RSA PUBLIC KE	-----BEGIN RSA PUBLIC KE

图 11：用户的公钥私钥在数据库中的标识

其次，明确前后端在 RSA 加解密过程中扮演的角色：

前端：负责接收用户的输入，并且加密后传输给后端。

后端：负责接收前端传来的加密数据，使用该账户的私钥进行解密，并且和数据库的密码进行比较后判断是否允许登录。

以下为前端登录时的代码

```

async login(){
    var data2send = {account:this.account}
    axios.post(this.urlbase+"/getPublicKey",data2send)//获取公钥
    .then(response=>{
        this.publicPem=response.data.publicKey
        this.publicPem=this.publicPem.replace('-----BEGIN  RSA  PUBLIC
KEY-----','-----BEGIN PUBLIC KEY-----')
        this.publicPem=this.publicPem.replace('-----END  RSA  PUBLIC
KEY-----','-----END PUBLIC KEY-----')//转换头，便于下买你的 encrypt 操作
        console.log(this.publicPem)
        const encryptor = new JSEncrypt()
        encryptor.setPublicKey(this.publicPem)
        const encryptedPassword = encryptor.encrypt(this.password)
        password=encryptedPassword
        console.log(encryptedPassword)
        data2send = {
            account :this.account,
            password :this.password
        }
        axios.post(this.urlbase+"/login",data2send)
        .then(response=>{
            alert(response.data.message)
            this.$router.push({path:'/goods'})
        })
        .catch(error=>{
            alert(error.message)
        })
    })
}

```



```

    })
  })
  .catch(error=>{
    alert(error.message)
  })
},

```

图 12 为前端返回的公钥，以及密码 123456 加密后的结果。



```

-----BEGIN PUBLIC KEY-----
MIIBCgKCAQEAr14LONbz4VsrVONqLdGsXcFWbi2xK67SOcd+Lj3DnLYzmKd8Ob
J3uM49Nv6pNuXPDnLZ5aQ8NLFwvT7M1Pzyax8duMyIMTGZ3+OQf+cQ3eDTSFjfgX
4kjuw4GY6iRZ/+09Y9y1ld/Erw3yQDDuWTFjeFEaxkv+3gqM3HcWoMKLNLCgLT0
t1EwzTauHnZAV2PvI8938VupXnNOVlj1FxZhEDIj3K1l/IFzroUnhuNesuo5iGQe
vwrUlj6sFQ0oJ5CxUTS5b2KsjIvDAA+nVfXsIv7IB92VvspEMBqazp80tab+rCNw
P1IoYu3tv/PebkiH4BkpdTCsJLy0nUybdQIDAQAB
-----END PUBLIC KEY-----

OHUdaZ2YVv1UbGd9PueMPF5uMJiMQZs3XJ+TgvAOH/FkXFWnwF
ZtbG+LXR18Hbw6G/yIgQkdKFVXJ19/WpSHNhJpAs7I3bc56eut+B2cA/tAq87DJ
tFXJ9okvs18L6kUf1iqbVTuH+TsaoAci9OyHdxhWskcbf5Ho80r9UnYvsTtzPkQ
7xrdc18dQamA4WnnFZfiuVutJfAXzJGNoViQf4/kwECCApiPK0rcUq94anVPnkJ
syX1z0eucV/m+hXbm/HAlPYhqQ0Z5Wa0IMistk/FL8+XjZEP+hDwh6ff9VBt3p/F
GTw24+Qq6gN7PvJ7eOKrUuY0499eAGJL9gqA==

```

图 12

以下为后端代码，分为两个函数：getPublicKey 以及 login,前者用于发送给前端公钥，后者用于进行加密密码的解密以及验证。

```

def getPublicKey():
    account = request.json.get('account')
    try:
        conn = get_db()
        #查询
        with conn.cursor() as cursor:
            cursor.execute('SELECT publickey FROM miyao WHERE account = %s', (account,))
            public_key = cursor.fetchone()[0]
        if not public_key:
            return jsonify({'error': 'Invalid account or password'}), 401
        print(public_key)
        return jsonify({'publicKey': public_key})
    except pymysql.Error as e:
        print(e)
        return jsonify({'error': 'query failed'}), 500

def login():
    global globaluseraccount
    account = request.json.get('account')
    password = request.json.get('password')
    print(account,password)
    try:
        #判存在
        conn = get_db()
        with conn.cursor() as cursor:
            cursor.execute('SELECT * FROM users WHERE account = %s', (account,))
            existing_user = cursor.fetchone()

```

```

if not existing_user:
    return jsonify({'error': 'Invalid account or password'}), 401
#解密
with conn.cursor() as cursor:
    cursor.execute('SELECT privatekey FROM miyao WHERE account = %s', (account,))
    private_pem = cursor.fetchone()[0]
    rsa_key = rsa.PrivateKey.load_pkcs1(private_pem)

#解密密码
passwordbytes=password.encode('utf-8')
print(passwordbytes)
decrypted_password = request.json.get('password')
stored_password = existing_user[1]
if decrypted_password != stored_password:
    return jsonify({'error': 'Invalid account or password'}), 401
# 登录成功，可以进行进一步的操作
globaluseraccount=account
return jsonify({'message': 'Login successful'})
except pymysql.Error as e:
    print(e)
    return jsonify({'error': 'Login failed: {}'.format(str(e))}), 500

```

图 13 为后端发送的公钥，以及收到的账户，密码解密结果

```

127.0.0.1 - - [31/Dec/2023 21:45:54] "OPTIONS /getPublicKey HTTP/1.1" 200 -
-----BEGIN RSA PUBLIC KEY-----
MIIBBgKCAQEAgjra14LONbz4VsrVONqLdGsXcFWbi2xK67SOCd+Lj3DnLYzmKd8Ob
J3uM49Nv6pNuXPDnLZ5aQ8NLFwvT7M1Pzyax8duMy1MTGZ3+OQf+cQ3eDTSFjfGX
4kjwu4GY6iRZ/+09Y9y1ld/Erw3yQDDuWTFjeFEaxkv+3gqM3HcWoMKLNLVcgLT0
t1EwzTauHnZAV2PvI8938VupXnNOV1j1FxZhEDIj3K1l/IFzroUnhuNesuo5iGQe
vwrU1J6sFQ0oJ5CxUTS5b2KsjIvDAA+nVfXsIv7IB92VvspEMBqazp80tab+rCNw
P1IoYu3tv/PeBkiH4BkpdTCsJLy0nUybdQIDAQAB
-----END RSA PUBLIC KEY-----

127.0.0.1 - - [31/Dec/2023 21:45:54] "POST /getPublicKey HTTP/1.1" 200 -
127.0.0.1 - - [31/Dec/2023 21:45:54] "OPTIONS /login HTTP/1.1" 200 -
admin 200305
b'200305'
127.0.0.1 - - [31/Dec/2023 21:45:54] "POST /login HTTP/1.1" 200 -
admin
127.0.0.1 - - [31/Dec/2023 21:45:57] "POST /getGoods HTTP/1.1" 200 -
127.0.0.1 - - [31/Dec/2023 21:45:57] "POST /getUserAccount HTTP/1.1" 200 -

```

图 13

### 4.3 登陆界面实现

登陆功能在 4.2 中已经介绍过，这里着重介绍注册。

首先为登陆界面：使用 html，css 和 js 进行构建。如图 14，15 分别为注册界面和登陆界面



图 14



图 15

接下来为注册功能的详细介绍：提供了账户的防重，代码如下。访问数据库，搜寻这个账户是否已经存在。

```
with conn.cursor() as cursor:
    cursor.execute('SELECT * FROM users WHERE account = %s', (account))
    existing_user = cursor.fetchone()
    if existing_user:
        return jsonify({'error': 'Account already exists'}), 400
```

创造了密钥对，并且解码后存储到数据库，代码如下。`users` 数据库内容包括用户名，账户，密码，用户类型，`miyao` 数据库包括账户，公钥私钥。

```
(publickey,privatekey)=rsa.newkeys(2048)
privatepem=privatekey.save_pkcs1().decode()
publicpem=publickey.save_pkcs1().decode()
with conn.cursor() as cursor:
    cursor.execute(
        'INSERT INTO users (account, password, username, usertype) VALUES
        (%s, %s, %s, %s)',
        (account, password, username, usertype)
    )
    cursor.execute(
        'INSERT INTO miyao (account, privatekey,publickey) VALUES (%s, %s, %s)',
        (account, privatepem,publicpem)
    )
conn.commit()
return jsonify({'message': 'Registration successful'})
```

图 16 为账户数据库的部分截图。其中密码进行了加密。

开始事务	文本	筛选	排序	导入
account	password	username	usertype	
admin	95207ca5-8f0	admin	3	
hehe	80126b7e-9c7	DullBean	1	
user	328358cf-526	1111	2	

图 16

## 4.4 其余功能实现

### 4.4.1 商品界面

商品界面主要提供商品的浏览，添加至购物车，以及上架服务。由于涉及到用户权限问题，因此应当获取用户的 `type` 来判断是否允许上架。图 17 为商品界面预览，图 18 为上架功能预览。



图 17: 商品页面



图 18: 上架页面

### 4.4.2 购物车界面

在商品界面点击添加至购物车后，数据库中会存储该用户的购物车信息，每当打开购物车页面的时候，前端会向后端发送请求，后端会返回数据库内容，通信代码如下：

前端：

```
async getCartItems(){
  try{
    const response = await axios.post(this.urlbase+'/getcart')//发送 paost 请求
    this.cartCount = response.data.count
    this.cartItems = response.data.cart
    this.needToPay = response.data.totprice
  } catch(error){
    console.log (error)
    console.error(error)
  }
},
```

后端：

```
@app.route('/getcart',methods=['post'])
def getcart():
  global globaluseraccount#全局用户信息
  try:
    conn = get_db()
    #查询
    cart=[]
```

```

totprice=0
with conn.cursor() as cursor:
    cursor.execute('SELECT * FROM cart WHERE account=%s',globaluseraccount)
    rows =cursor.fetchall()
    for row in rows:
        cart.append({
            'name':row[1],
            'price':row[2]
        })
        totprice+=row[2]
    return jsonify({'cart': cart,'count':len(cart),'totprice':totprice})
except pymysql.Error as e:
    print(e)
    return jsonify({'error': 'query failed'}), 500

```

界面如图 19



图 19 购物车页面

点击我不想要，可以将从数据库中物品删除，同时重新发起数据库询问请求，代码如下：  
前端：

```

deleteitem(item){
    axios.post(this.urlbase+'/deleteitem',{'item':item})
    .then(response=>{
        alert('删除成功')
        console.log(response)
        location.reload()
    })
    .catch(error=>{
        alert(error)
    })
}

```

后端：

```
@app.route('/deleteitem',methods=['post'])
```

```
def deleteitem():
```

```
    item = request.json.get('item')#获取 item 信息 item 数据类型为字典
```

```
    name = item['name']
```

```
    global globaluseraccount
```

```
    try:
```

```
        conn = get_db()
```

```
        with conn.cursor() as cursor:
```

```
            cursor.execute('delete FROM cart WHERE name=%s AND
```

```

account=%s',(name,globaluseraccount))
    conn.commit()
    return jsonify({'message':'succeed'})
except pymysql.Error as e:
    print(e)
    return jsonify({'error': 'query failed'}), 500

```

删除成功后如图 20



图 20 删除效果

在挑选完货物之后，可以点击支付按钮进行支付，需要填写地址和密码。密码用于验证，在支付完后，创建订单。支付代码如下：

前端：

```

async pay(){
    axios.post(this.urlbase+'/pay',{'password':this.password,'address':this.address,'items':this.cartItems})
    .then(response=>{
        alert('支付成功！')
        console.log(response.message)
        location.reload()
    })
    .catch(error=>{
        alert(error)
    })
},

```

后端：主要流程为：获取前端信息，进行密码对比，清空购物车，遍历购物车的每一项内容创建订单并且插入数据库

```

def pay():
    global globaluseraccount
    address = request.json.get('address')
    password = request.json.get('password')
    items = request.json.get('items')
    try:
        #判存在
        conn = get_db()
        with conn.cursor() as cursor:
            cursor.execute('SELECT * FROM users WHERE account = %s', (globaluseraccount,))
            existing_user = cursor.fetchone()
            stored_password = existing_user[1]
            if password != stored_password:
                return jsonify({'error': 'Invalid account or password'}), 401
    
```

```

with conn.cursor() as cursor:
    cursor.execute('delete FROM cart WHERE account = %s',(globaluseraccount))
conn.commit()
#创建订单
print(items)
for item in items:
    name=item['name']
    with conn.cursor() as cursor:
        cursor.execute('SELECT account FROM goods where name=%s',(name))
        seller=cursor.fetchone()[0]
    with conn.cursor() as cursor:
        nowaddr="未知"
        done='0'
        cursor.execute(
            'INSERT INTO logistics (address,seller,nowaddr,buyer,name,done)
VALUES(%s,%s,%s,%s,%s,%s)',
            (address,seller,nowaddr,globaluseraccount,name,done)
        )
    conn.commit()
    return jsonify({'message': 'Login successful'})
except pymysql.Error as e:
    print(e)
    return jsonify({'error': 'Login failed: {}'.format(str(e))}), 500

```

支付界面图像如图 21



图 21: 支付内容图像

#### 4.4.3 物流管理

物流管理主要允许用户进行订单追踪和物流更新，主界面如图 22

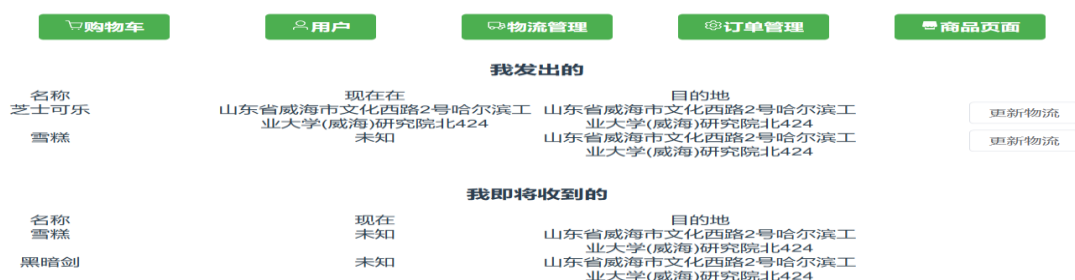


图 22: 物流界面

页面会在打开时向后端请求该用户作为 buyer 和 seller 的数据，其中作为 seller 的数据允许用户进行物流更新。由于顾客不能够进行上架，因此顾客账户必然没有发出的货物。

请求代码如下：

前端：

```
getallinfo(){
    axios.post(this.urlbase+'/getallinfo')
    .then(response=>{
        console.log(response)
        this.sended=response.data.send
        this.received=response.data.receive
    })
},
```

后端：具体流程为构建发送列表，构建接受列表。

```
def getallinfo():
    send=[]
    receive=[]
    global globaluseraccount
    try:
        #构建发送列表
        conn = get_db()
        with conn.cursor() as cursor:
            cursor.execute('SELECT * FROM logistics WHERE seller=%s',(globaluseraccount))
            rows =cursor.fetchall()
            for row in rows:
                send.append({
                    'address':row[0],
                    'name':row[4],
                    'done':row[5],
                    'nowaddr':row[2]
                })
            #构建接受列表
            cursor.execute('SELECT * FROM logistics WHERE buyer=%s',(globaluseraccount))
            rows =cursor.fetchall()
            for row in rows:
                receive.append({
                    'address':row[0],
                    'name':row[4],
                    'done':row[5],
                    'nowaddr':row[2]
                })
            return jsonify({'send': send,'receive':receive})
    except pymysql.Error as e:
        print(e)
        return jsonify({'error': 'query failed'}), 500
允许用户进行物流更新，如图 23
```





图 23，物流更新

在提交物流更新表单后，后端会对数据库进行 update。

4.4.4 订单管理和用户管理(管理员)

这两个页面仅有管理员可以进行有效操作。

首先是订单管理：管理员可以在订单管理页面中观察所有的订单以及完成情况。可以导出 excel 进行分析，界面如图 24。

<div>购物车 用户 物流管理 订单管理 商品页面</div>					
导出excel					
到货地址	卖家	目前地点	买家	是否完成?(1:0)	货物名称
山东省威海市文化西路2号哈尔滨工业大学(威海)研究院北424	admin	山东省威海市文化西路2号哈尔滨工业大学(威海)研究院北424	hehe	1	芝士可乐
山东省威海市文化西路2号哈尔滨工业大学(威海)研究院北424	xixi	山东省威海市文化西路2号哈尔滨工业大学(威海)研究院北424	hehe	1	黑暗剑
山东省威海市文化西路2号哈尔滨工业大学(威海)研究院北424	admin	未知	admin	0	雪糕
山东省威海市文化西路2号哈尔滨工业大学(威海)研究院北424	xixi	未知	admin	0	黑暗剑

图 24

管理员可以在用户界面对用户进行删除，如图 25



图 25

以下是非管理员用户对这两个网页进行访问时的页面。非管理员用户在访问用户页面的时候，仅仅可以查看自己的公钥和身份。访问订单管理页面时会被重定向。图 26 为点击管理用户时被拒绝的界面



图 26: 点击管理用户被拒绝的界面

图 27 为点击显示公钥时的界面

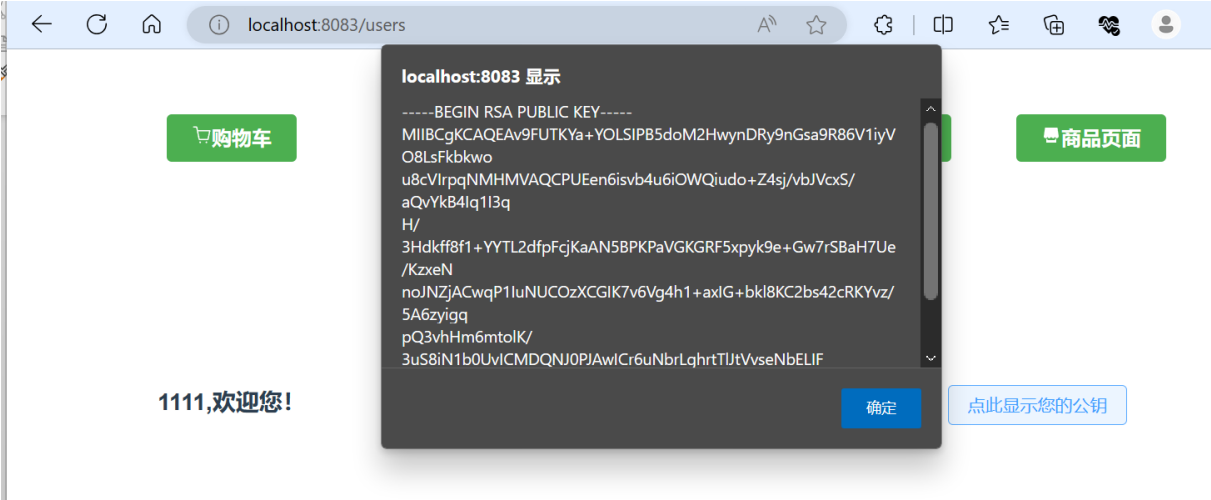


图 27: 公钥显示界面

图 28 为访问订单管理页面时，普通用户被拒绝的界面

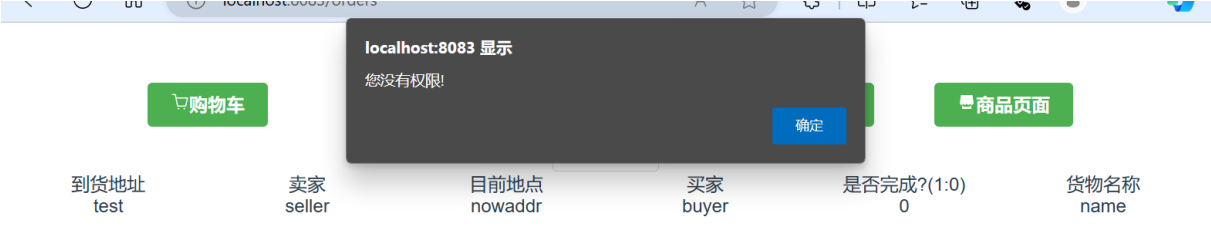


图 28: 普通用户点击订单页面时的界面

## 5. 总结

问题：前后端通信属于陌生通信，后端拒绝。

解决方案：使用 flask-cors 解决跨域问题，在 flask 添加如下代码：

`CORS(app, resources={r'/*': {'origins': '*'}}, supports_credentials=True)`

问题：数据库使用 insert 语句时不执行

解决方案：加入 commit 语句以执行数据库更改操作。

问题：RSA 密码进行直接存储后前端不识别。

解决方案：RSA 密码解码成 PEM 格式后存储至数据库再发送至前端，当前端传回时再编码为 key。即 KEY->PEM->存储+传输->PEM->KEY

预计实现但是出于时间原因未实现的功能：商品添加 ID 并且使用动态路由导向。