5 用户端的实现与测试

5.1 校园物流车小程序设计说明

5.1.1 产品概述

该系统主要辅助送货员完成用户签收快递，完成配送流程。该系统的目标是为校园内的快递配送提供更加便捷、高效的解决方案，通过智能化的设计和实现，提高快递配送的效率和用户体验。

5.1.2 功能概述

（1）用户登录和注册： 用户可以通过手机号码进行注册和登录。系统会验证手 机号码的合法性，并将用户信息存储在数据库中，用户可以在登录后修改个人信 息。

（2）个人信息二维码显示和识别： 用户可以在个人信息页面中查看自己的二维 码，并可以通过扫描二维码进行快递签收。系统会根据用户的个人信息生成二维 码，并在个人信息页面中显示。

用户可以通过扫描二维码进行快递签收。系统会根据二维码中的信息进行快 递签收操作

（3）快递下单和查询： 用户可以通过输入快递信息进行下单，系统会将快递信 息存储在数据库中，并生成订单号。用户可以在订单页面中查看订单状态和历史 记录。

（4）快递签收： 用户可以通过扫描快递上的二维码进行签收。 系统会根据二 维码中的信息进行快递签收操作，并更新订单状态。

5.1.3 实现逻辑

实现逻辑包括以下几个方面：

（1）用户登录和注册： 用户输入手机号码进行注册和登录，系统会验证手机号 码的合法性，并将用户信息存储在数据库中。

（2）个人信息二维码显示和识别： 系统会根据用户的个人信息生成二维码，并 在个人信息页面中显示。用户可以通过扫描二维码进行快递签收，系统会根据二 维码中的信息进行快递签收操作。

（3）快递下单和查询： 用户输入快递信息进行下单，系统会将快递信息存储在 数据库中，并生成订单号。用户可以在订单页面中查看订单状态和历史记录。

（4）快递签收： 用户可以通过扫描快递上的二维码进行签收，系统会根据二维 码中的信息进行快递签收操作，并更新订单状态。

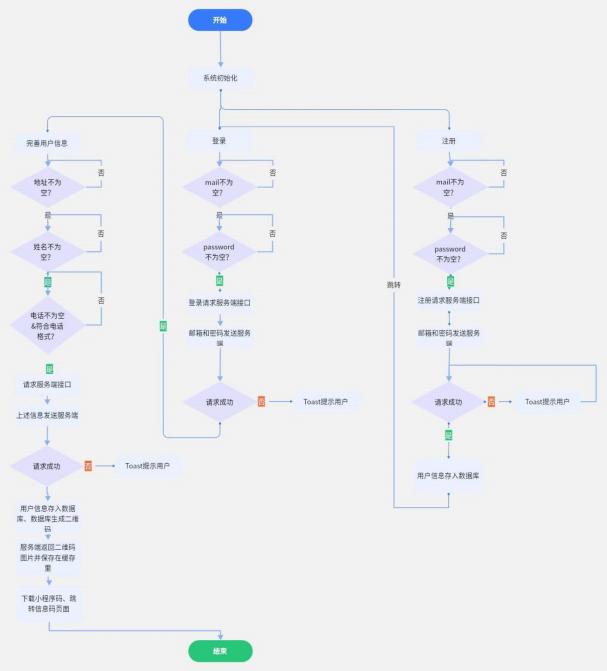


图 5-1 程序逻辑流程图

5.2 产品需求简述（全局说明）

5.2.1 总体功能框架

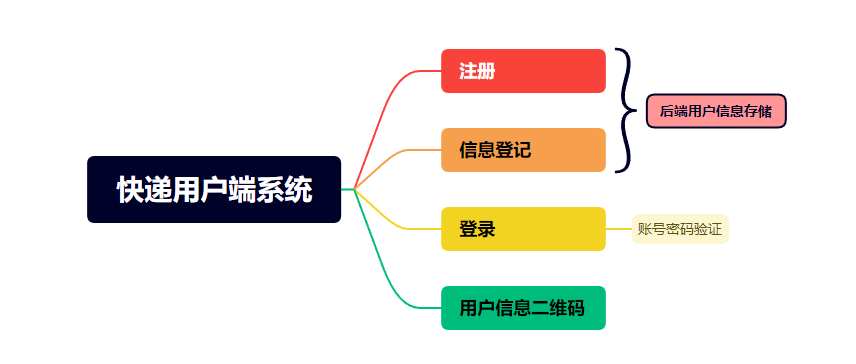


图 5-2 业务模型

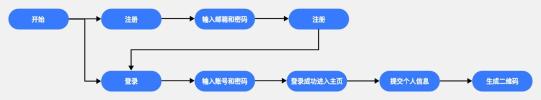


图 5-3 业务流程图

5.2.2 用户角色与权限控制

快递员和用户都是通过微信小程序端登录授权，确定身份。

表 5-1 用户与角色控制

|  |  |
| --- | --- |
| 用户角色 | 用户描述 |
| 普通用户 | 可以登录并使用该系统的为普通用户。它拥有常规功能权限， 包括在线寄件 、 取件出库 、 查件 ， 对用户个人资料等进行管理操作  等功能。 |
| 快递员 | 派送信息录取、可以登录后可以进行揽件和代派件的任务，对自己  资料进行管理 ， 查看自己任务量等等功能。 |
| 后端管理员 | 可以对用户信息和快递员信息进行管理 |

5.3详细实现步骤

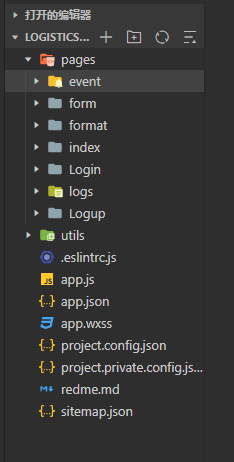
5.3.1 前端小程序的编写

1、微信开发者工具新建工程



2、新建pages

点击加号，新建pages



本小程序共需要新建五个pages,分别为

form:表格提交（信息注册界面）

index:初始界面

format：二维码界面

login：登录界面

logup: 注册界面

工程目录下还有一些初始文件：

（1）在utils中定义方法、工具等，主要使用common.js暴露接口

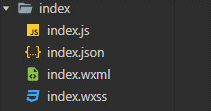
（2）pages中放我们的页面

（3）eslintrc.js文件在代码质量审查的时候的代码依赖

（4）app.js、app.json、app.wxss是我们微信小程序代码的重要部分。app.js是小程序逻辑，app.json是小程序配置，app.wxss是小程序公共样式表

3.pages界面编写

每个pages为一个界面，其下包含四个文件：



下面以初始界面为例：

我们创建文件会新建一个文件夹，右键新建Page，会自动创建四个文件，并把文件路径放入app.json中注册。

（1）wxml文件是页面的必须成分，作用是页面结构；

<!--index.wxml-->

<view class="container">

  <view class="userinfo">

    <block wx:if="{{canIUseOpenData}}">

      <view class="userinfo-avatar" bindtap="bindViewTap">

        <open-data type="userAvatarUrl"></open-data>

      </view>

      <open-data type="userNickName"></open-data>

    </block>

    <block wx:elif="{{!hasUserInfo}}">

      <button wx:if="{{canIUseGetUserProfile}}" bindtap="getUserProfile"> 获取头像昵称 </button>

      <button wx:elif="{{canIUse}}" open-type="getUserInfo" bindgetuserinfo="getUserInfo"> 获取头像昵称 </button>

      <view wx:else> 请使用1.4.4及以上版本基础库 </view>

    </block>

    <block wx:else>

      <image bindtap="bindViewTap" class="userinfo-avatar" src="{{userInfo.avatarUrl}}" mode="cover"></image>

      <text class="userinfo-nickname">{{userInfo.nickName}}</text>

    </block>

  </view>

  <view class="usermotto">

    <text class="user-motto">快递用户端系统</text>

  </view>

</view>

<!-- 注册页面 -->

<view>

    <view><button class="logup" type="primary" bindtap="jumpPage\_logup">log up</button></view>

</view>

<!-- 登录页面 -->

<view>

    <view><button class="login" type="primary" bindtap="jumpPage\_login">log in</button></view>

</view>

（2）js文件是页面的必须成分，作用是实现页面逻辑；

// index.js

// 获取应用实例

const app = getApp()

Page({

  data: {

    motto: 'LOGISTICS-CAR',

    userInfo: {},

    hasUserInfo: false,

    canIUse: wx.canIUse('button.open-type.getUserInfo'),

    canIUseGetUserProfile: false,

    canIUseOpenData: wx.canIUse('open-data.type.userAvatarUrl') && wx.canIUse('open-data.type.userNickName') // 如需尝试获取用户信息可改为false

  },

  // 事件处理函数

  bindViewTap() {

    wx.navigateTo({

      url: '../logs/logs'

    })

  },

  onLoad() {

    if (wx.getUserProfile) {

      this.setData({

        canIUseGetUserProfile: true

      })

    }

  },

  // 注册页面跳转

  jumpPage\_logup:function(){

    wx.navigateTo({

      url: '/pages/Logup/Logup',

    })

  },

  //登陆页面跳转

  jumpPage\_login:function(){

    wx.navigateTo({

      url: '/pages/Login/Login',

    })

  },

  getUserProfile(e) {

    // 推荐使用wx.getUserProfile获取用户信息，开发者每次通过该接口获取用户个人信息均需用户确认，开发者妥善保管用户快速填写的头像昵称，避免重复弹窗

    wx.getUserProfile({

      desc: '展示用户信息', // 声明获取用户个人信息后的用途，后续会展示在弹窗中，请谨慎填写

      success: (res) => {

        console.log(res)

        this.setData({

          userInfo: res.userInfo,

          hasUserInfo: true

        })

      }

    })

  },

  getUserInfo(e) {

    // 不推荐使用getUserInfo获取用户信息，预计自2021年4月13日起，getUserInfo将不再弹出弹窗，并直接返回匿名的用户个人信息

    console.log(e)

    this.setData({

      userInfo: e.detail.userInfo,

      hasUserInfo: false

    })

  }

})

（3）json文件不是页面的必须成分，作用是对页面进行配置；

{

  "usingComponents": {}

}

（4）wxss文件不是页面的必须成分，作用用来配置页面样式表；

/\*\*index.wxss\*\*/

.userinfo {

  display: flex;

  flex-direction: column;

  align-items: center;

  color: #aaa;

}

.userinfo-avatar {

  overflow: hidden;

  width: 128rpx;

  height: 128rpx;

  margin: 20rpx;

  border-radius: 50%;

}

.usermotto {

  margin-top: 130rpx;

}

/\* 注册页面 \*/

.logup{

  width: 15%;

  height: 90rpx;

  top: 130rpx;

}

/\* 登陆页面 \*/

.login{

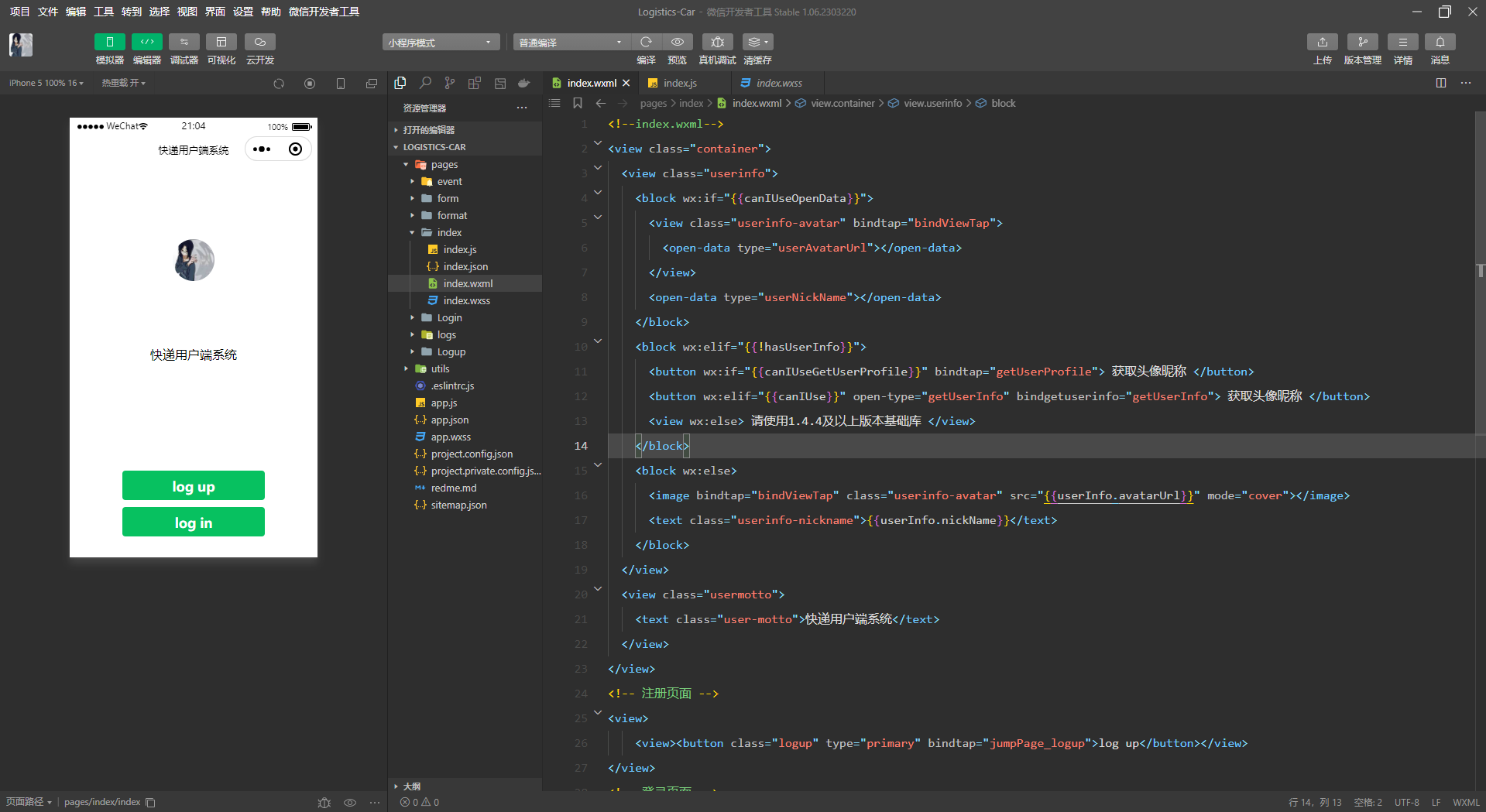
  width: 15%;

  height: 90rpx;

  top: 150rpx;

}

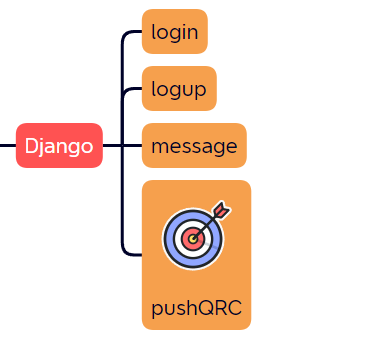
4．初始界面效果



初始界面最终效果如图左方所示。

5.3.2后端Django程序的实现

1.Django程序结构图：



2.Django的启动：

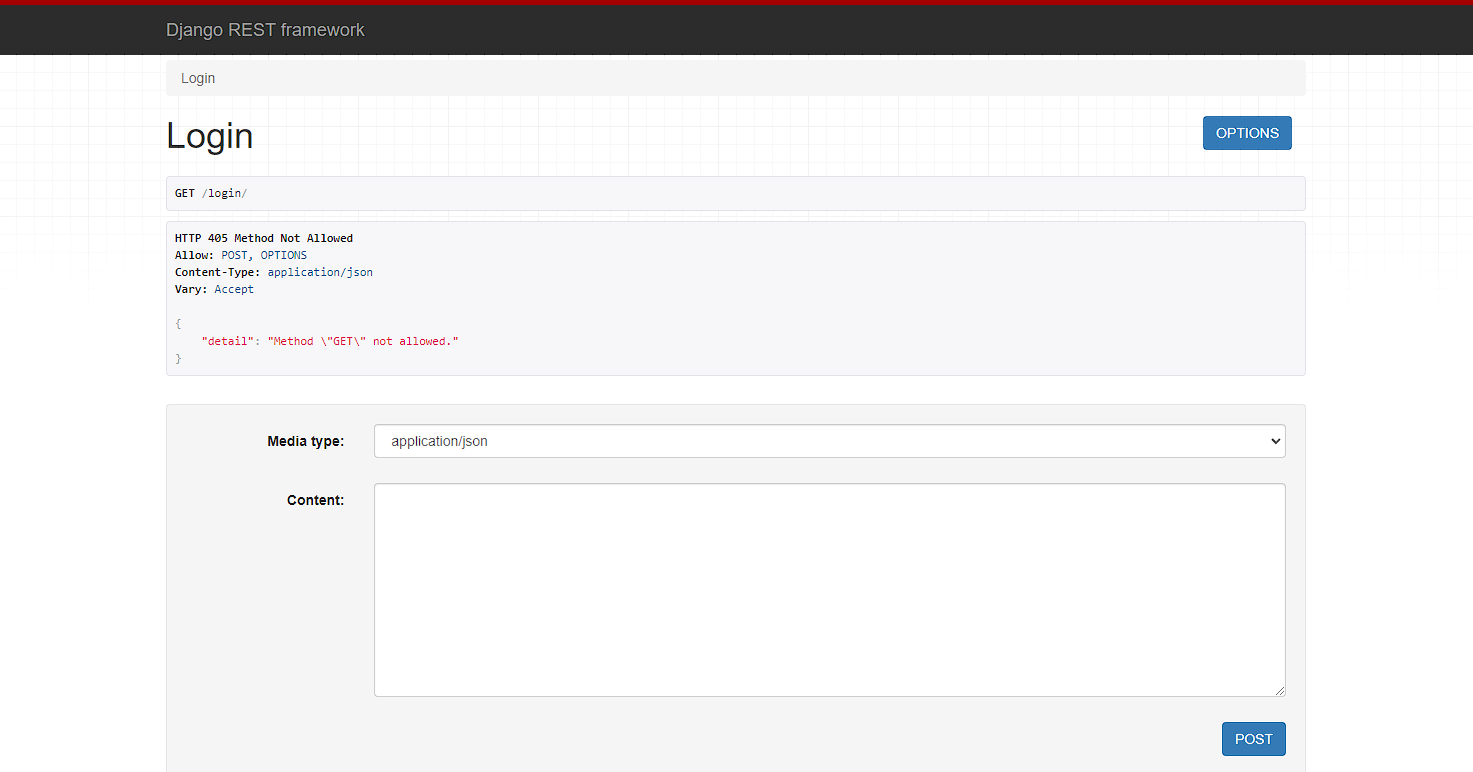
使用

python manage.py runserver

命令启动Django程序，会返回一个url。如下图所示



将链接复制到浏览器输入所对应的page页即：<http://127.0.0.1:8000/login/>，可看到如下界面：



3.Django新应用的添加

使用如下命令添加新应用：

python manage.py startapp 应用名

注意：使用命令行创建的应用，一定要在 settings.py-->INSTALLED\_APPS 中加入你的应用名字进行注册, 使用Pycharm创建的应用则不用。

4.APP文件目录



Views.py文件如下：

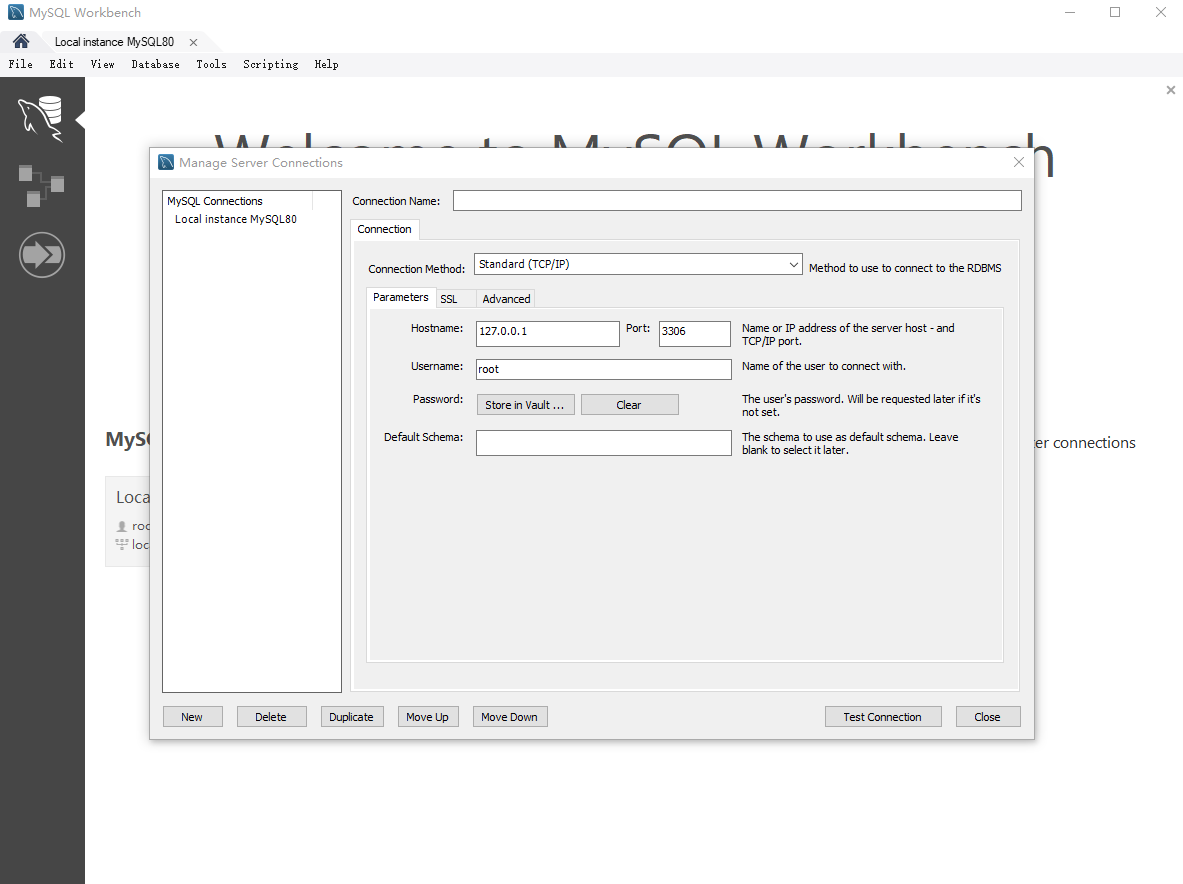
*from* django.shortcuts *import* render  
*from* django.http *import* JsonResponse  
*from* rest\_framework.views *import* APIView  
*from* rest\_framework.response *import* Response  
*from* Mysql *import* sql\_inset, sql\_read  
*from* rest\_framework *import* serializers  
*from* rest\_framework.exceptions *import* ValidationError  
  
*#login回传信息的获取  
class* LoginView(APIView):  
  
 *def* post(*self*, request, \*args, \*\*kwargs):  
 print(request.data)  
 mail\_password = request.data  
 print(mail\_password['password'])  
 url = sql\_read.sql\_get\_QRCimg(mail\_password['mail'], mail\_password['password'])  
 print(url)  
 *# return Response({"status": True})  
 return* Response(url)  
  
*#logup回传信息的获取  
class* LogupView(APIView):  
  
 *def* post(*self*, request, \*args, \*\*kwargs):  
 mail\_password = request.data  
 print(mail\_password['mail'])  
 sql\_inset.logup\_inset(mail\_password['mail'],mail\_password['password'])  
 *return* Response({"status": *True*})  
  
*class* MessageView(APIView):  
  
 *def* post(*self*, request, \*args, \*\*kwargs):  
 print(request.data)  
 name\_area\_phone = request.data  
 sql\_inset.logup\_message\_inset(name\_area\_phone['name'],name\_area\_phone['area'],name\_area\_phone['phone'])  
 *return* Response({"status": *True*})

其中class LoginView(APIView)、class LogupView(APIView)、class MessageView(APIView)分别对前端login回传消息，logup回传消息、message信息登记页面回传消息进行获取。消息回传结果如下图所示：



5.3.3Mysql数据库信息存储的实现

1.mysql数据库设置



设置好端口号、IP、用户、密码，并创建一个数据库。

2.建表

*import* pymysql  
*# 打开数据库连接  
try*:  
 db = pymysql.connect(host='localhost', user='root', passwd='2001WEIshuai10.@', port=3306, database='user')  
 print('连接成功！')  
*except*:  
 print('something wrong!')  
  
*# # 使用 cursor() 方法创建一个游标对象 cursor*cursor = db.cursor()  
*# 使用 execute() 方法执行 SQL，如果表存在则删除*cursor.execute("DROP TABLE IF EXISTS user\_USERS")  
  
*# 使用预处理语句创建表*sql = """CREATE TABLE user\_USERS (  
 USERNAME *CHAR*(30) ,  
 USERADDRESS *CHAR*(30),  
 USERPHONENUMBER BIGINT,  
 USERMAIL *CHAR*(30),  
 USERPASSWORD *CHAR*(30),  
 USERurl *CHAR*(50))"""  
  
cursor.execute(sql)  
print('建表成功！')

3.插入数据

*import* pymysql  
*import* qrcode  
*from* Mysql *import* QRC\_update  
*#注册邮箱密码登记  
  
def* logup\_inset(mail,password):  
 print(mail,password)  
 *global* mail\_password  
 mail\_password = {'mail':mail,'password':password}  
  
  
*#注册个人信息登记  
  
def* logup\_message\_inset(name,area,phone):  
 *try*:  
 db = pymysql.connect(host='localhost', user='root', passwd='2001WEIshuai10.@', port=3306, database='user')  
 print('连接成功！')  
 *except*:  
 print('something wrong!')  
  
 *#生成二维码并上传图床，保存url到表* data\_str = "%s\n\r %s\n\r %s" % \  
 (name, area, phone)  
 filename = str(name) + '\_QRC.png'  
 QRCimg = qrcode.make(data\_str)  
 QRCimg.save('QRC\_img/' + filename)  
 url = QRC\_update.SMMS().upload(filepath='QRC\_img/' + filename)  
 print(url)  
 *# # 使用 cursor() 方法创建一个游标对象 cursor* cursor = db.cursor()  
 sql = """INSERT INTO user\_users(USERNAME,  
 USERADDRESS, USERPHONENUMBER, USERMAIL, USERPASSWORD, USERurl)  
 VALUES ('%s','%s',%s,'%s', '%s','%s')""" % (name, area, phone,mail\_password['mail'],mail\_password['password'], url)  
 *try*:  
 *# 执行sql语句* cursor.execute(sql)  
 *# 提交到数据库执行* db.commit()  
 print('数据插入成功！')  
 *except*:  
 *# 如果发生错误则回滚* db.rollback()  
 print('数据插入错误！')  
  
 *# 关闭数据库连接* db.close()

将微信前端传回的邮箱、密码、用户信息、以及图床返回的链接存入数据库。

4.二维码的生成

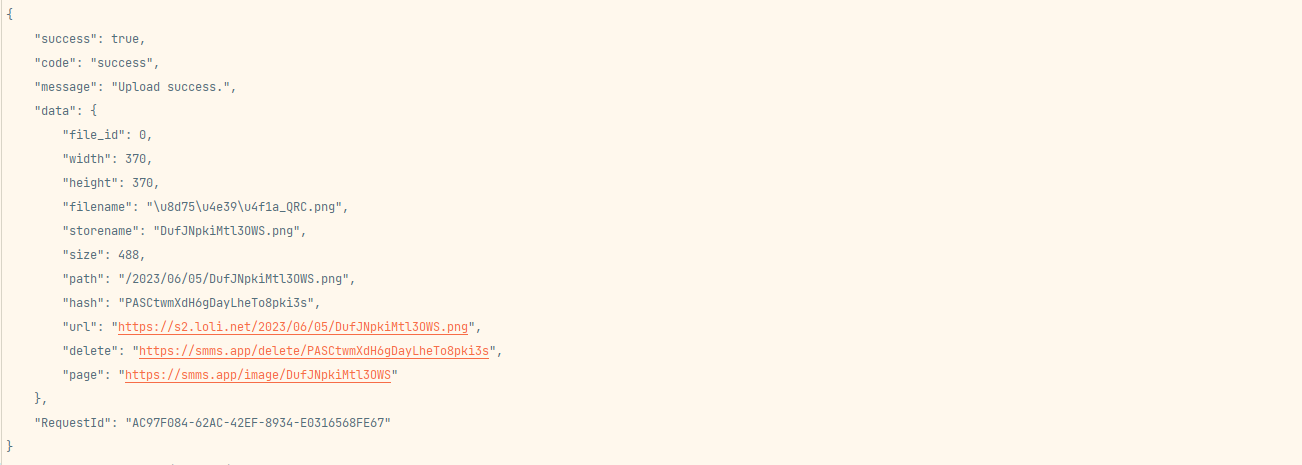
filename = str(name) + '\_QRC.png'  
QRCimg = qrcode.make(data\_str)  
QRCimg.save('QRC\_img/' + filename)

使用qrcode.make()函数将用户信息转换为二维码

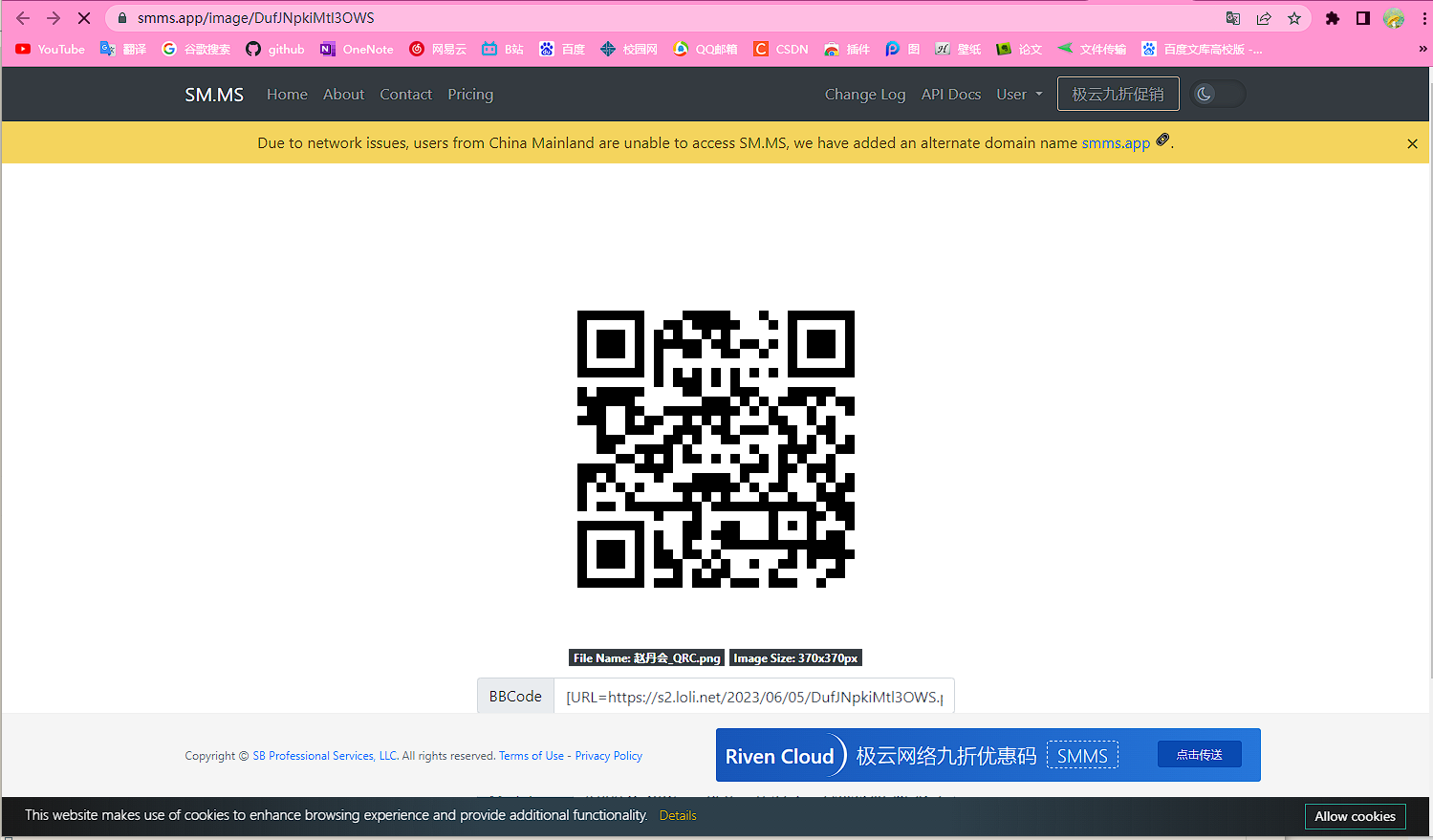
5、二维码上传图传

*import* time  
*import* requests  
*import* json  
*class* SMMS(object):  
 *def* \_\_init\_\_(*self*):  
 *self*.headers = {'Authorization': 'yi2YKlSlgFj1NEYVjzbXVEvYwlfvleRQ'}  
  
 *def* isDiskFull(*self*):  
 url = '<https://sm.ms/api/v2/profile>'  
 res = requests.post(url, headers=*self*.headers, timeout=5).json()  
 *if* res["disk\_usage\_raw"] > 0.9 \* res["disk\_limit\_raw"]:  
 *return True  
  
 def* upload(*self*, filepath):  
 files = {'smfile': open(filepath, 'rb')}  
 url = 'https://smms.app/api/v2/upload'  
 res = requests.post(url, files=files, headers=*self*.headers, timeout=5).json()  
 print(json.dumps(res, indent=4))  
 *return* res['images']  
  
 *def* getHistory(*self*):  
 url = '<https://sm.ms/api/v2/upload\_history>'  
 res = requests.get(url, headers=*self*.headers, timeout=5).json()  
 *return* res  
  
 *def* deleteHistory(*self*):  
 history = *self*.getHistory()  
 lastTime = time.time()  
 *for* item *in* history["data"]:  
 *if* item["created\_at"] < lastTime:  
 url = '<https://sm.ms/api/v2/delete/{}>'.format(item["hash"])  
 res = requests.get(url, headers=*self*.headers, timeout=5).json()  
 print("History deleted!")

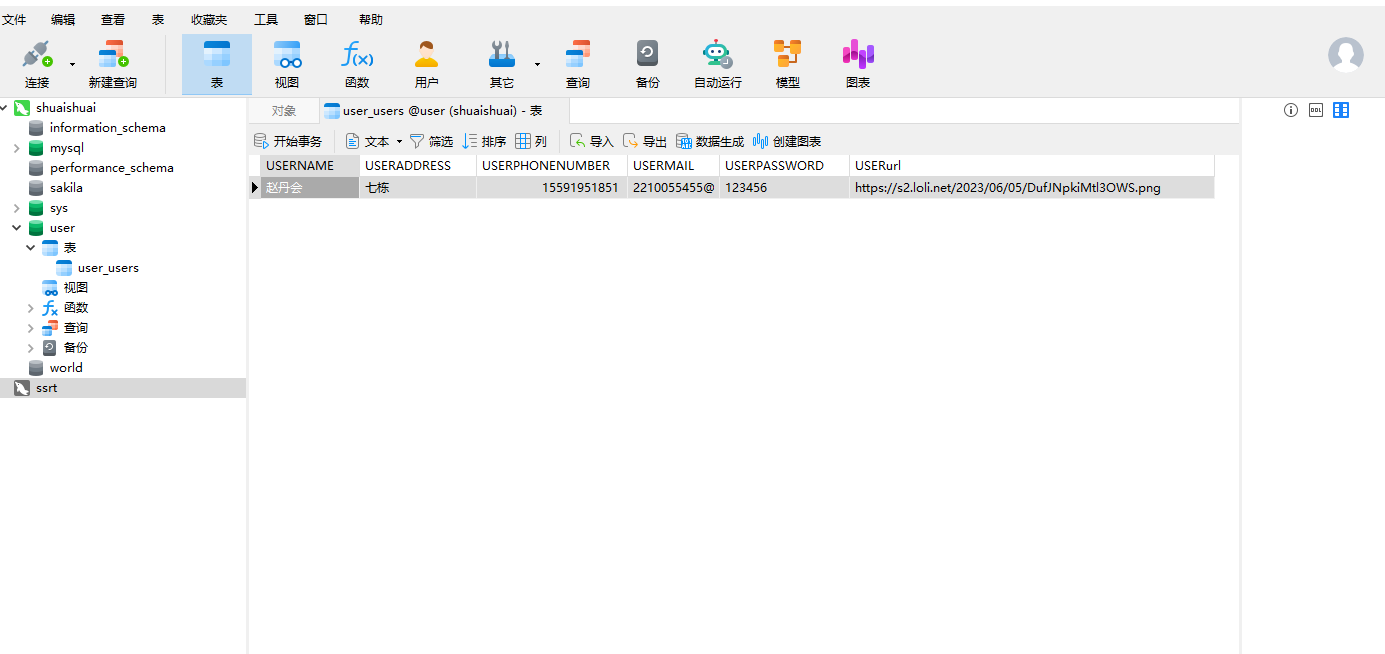
通过requests.post()函数发请求，将二维码上传图床并返回图片地址url，将其存入数据库。



将链接复制到浏览器可以看到上传的图片：



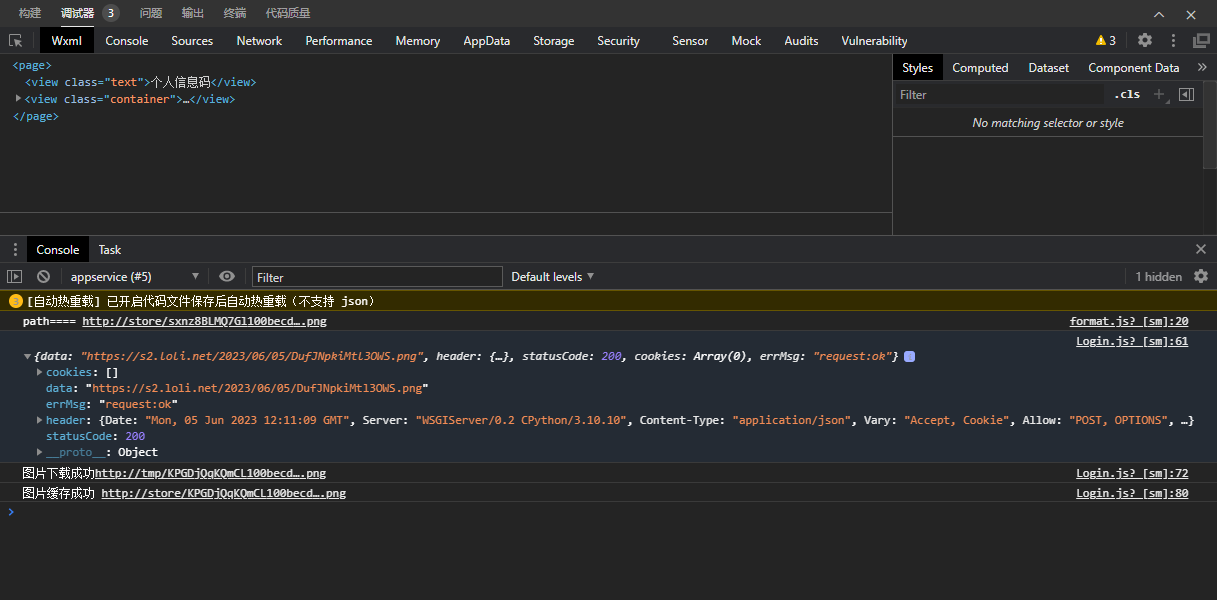
通过可视化工具Navicat可以看到存入数据库的信息：



6.读库

当前端登录操作时，程序会从数据库中找到对应的账号并比对密码，若密码正确则将图片链接传给前端小程序。

*import* pymysql  
*import* qrcode  
*from* Mysql *import* QRC\_update  
  
*def* sql\_get\_QRCimg(mail, password):  
  
 *global* USERPHONENUMBER, USERADDRESS, USERNAME, USERPASSWORD, db  
 print(mail, password)  
 *# 打开数据库连接  
 try*:  
 db = pymysql.connect(host='localhost', user='root', passwd='2001WEIshuai10.@', port=3306, database='user')  
 print('连接成功！')  
 *except*:  
 print('something wrong!')  
 *# 使用 cursor() 方法创建一个游标对象 cursor* cursor = db.cursor()  
 *# SQL 查询语句* sql = "SELECT *\** FROM user\_users \  
 WHERE USERMAIL = '%s' " % mail  
 *try*:  
 *# 执行SQL语句* cursor.execute(sql)  
 *# 获取所有记录列表* results = cursor.fetchall()  
 *for* row *in* results:  
 USERNAME = row[0]  
 USERADDRESS = row[1]  
 USERPHONENUMBER = row[2]  
 USERMAIL = row[3]  
 USERPASSWORD = row[4]  
 URSERurl = row[5]  
 *# 打印结果* print('数据查询成功！')  
 print("NAME=%s,ADDRESS=%s,PHONENUMBER=%s" % \  
 (USERNAME, USERADDRESS, USERPHONENUMBER))  
 *except*:  
 print("Error: unable to fetch data")  
 *#二登录信息正确则将url传回  
 if* password == USERPASSWORD:  
 url = URSERurl  
 *return* url  
 *else*:  
 print('密码错误')

传入小程序的图片链接会被小程序下载缓存然后展示。

5.4 产品功能详细说明及设计说明

5.4.1 系统登录与注册

未注册用户进入系统需要先进行注册再登录才可使用。

（1） 用户注册

图 5-4 注册界面

字段说明和展示说明

 log up： 注册

 log in;登录 交互逻辑说明

 注册顺序:点击注册（log up） →输入邮箱和密码→点击注册（log up） → 完成（跳转登录页）

 注册按钮： 跳转至注册页

5.4.2 二维生成

支持快递代取用户个人信息二维码的生成。

（1）快递二维码生成

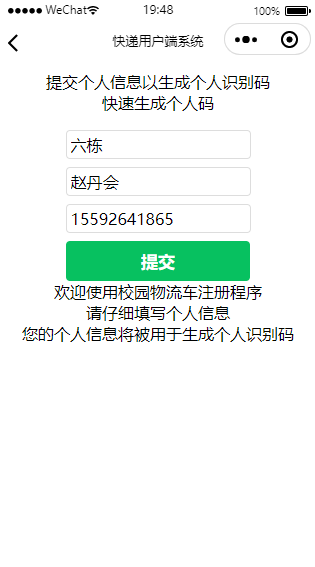
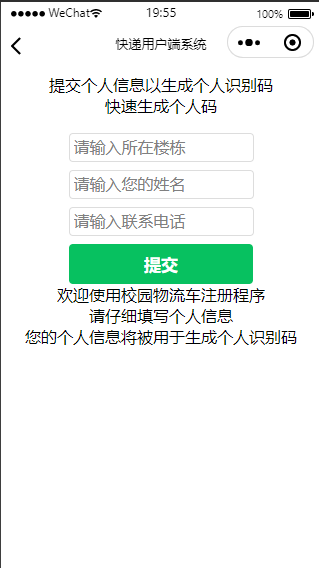


图 5-5 信息录入与二维码生成

字段说明和展示说明

 所在楼栋： 必填，长度限制 30 字节

 姓名:必填，长度限制 30 字节

 联系电话： 必填 交互逻辑说明

 操作顺序： 登录完成→提交个人信息（楼栋、姓名、联系电话） →点击提交 →生成二维码→完成

 提交按钮： 系统记录信息并返回变更个人用户信息

5.5 通信方式和功能架构

本系统采用 C/S（客户端/服务器） 架构，前后端分离的设计模式前端采用 微信小程序原生写法。后端采用python 技术进行开发。

用户和快递员端采用微信小程序展示，通过wx.reques 向后端服务器发送请 求和接受返回的数据。前后端通信数据格式采用 JSON 格式。数据库选择了体积 小、速度快的 MySql，用来存储用户、快递员和订单等信息。

本系统在登录授权信息方面采用了 OAuth 协议的授权方式。OAuth 协议可以 在第三方不需要接触用户名和用户密码的情况下，即可申请获得该用户资源的授权。OAuth 协议的特点是简单易懂，多框架支持;很安全，无需接触到用户隐私信息;同时也是开放的，任何服务商、开发者都可以使用。OAuth 协议里有四个 角色，分别是授权服务器、资源服务器、资源拥有者和客户端。其中资源服务器 和授权服务器可以是同一个服务器，也可以是不同的服务器，单独的授权服务器 可以为多个资源服务器发送 token。客户端首先请求用户的授权，用户同意后向 授权服务器申请令牌，授权服务器对客户端进行认证，确认无误后返回令牌，客 户端拿到令牌后即可向资源服务器申请资源。本系统里 token 的生成就是使用 JWT 方法。