《信息系统集成与管理》
实习报告二

学　　院：遥感信息工程学院

班　　级：22F12

学　　号：2022302131030

姓　　名：贾羽佳

实习地点：附三教学楼 203

指导老师：王华敏

2024 年 12 月 1 日

目录

一、实验目的

本实验旨在深化学生对 Web Service 理论知识的理解与应用能力，通过设计并实现一个具体的 Web 应用，包括前后端的编写，来构建 REST 风格的 Web API，服务于前端网页、以及各种 API 调用。通过实验过程，学生将熟悉 REST 架构风格的特点，掌握集合类资源的 Web API 设计方法，并学习如何以 JSON 格式进行数据表述；通过实际操作，学生将能够将理论知识与实践相结合，从而在实际开发中更好地应用 Web Service 技术。

二、实验环境

1. Pycharm Professional 2021.3.1
2. Python 解释器 3.10.11
3. Django 框架 5.1.3
4. mysql 数据库 8.0.36
5. MySQL Workbench 8.0 CE（mysql 数据库可视化操作界面）

三、项目介绍

1. 项目概述

项目 dace 是一个基于 python Django 框架构建的 Web 应用程序（结构组织如下图 1），专门用于采集和管理学生信息。项目采用清晰的目录结构，将核心应用逻辑封装在名为 dace01 的应用目录中，而项目级别的配置和入口点则位于主项目目录。

在主项目目录中，manage.py 作为 Django 管理命令的入口，允许开发者执行数据库迁移、启动开发服务器等任务。settings.py 文件包含了项目的关键配置，如数据库连接、中间件、静态文件设置等。urls.py 定义了项目级别的 URL 路由，将请求分发到相应的视图或应用。asgi.py 和 wsgi.py 分别提供了 ASGI 和 WSGI 服务器的配置，以支持异步和同步的 Web 请求处理。

dace01 应用目录包含了应用的核心组件。models.py 文件定义了 Student 模型，用于在数据库中存储学生的姓名、学号、邮箱、手机号码和爱好等信息。serializer.py 中的 StudentSerializer 负责将 Student 模型实例转换为 JSON 格式，以

便通过 API 传输，符合我们实习中要求的“collection+json”模式。views.py 文件包含了 StudentViewSet，是一个基于 Django REST framework 的视图集，提供了对学生信息的增删改查等操作。

位于 dace01 应用目录下的 urls.py 使用 Django REST framework 的路由器自动生成 API 路由，简化了 API 的创建过程。migrations 目录存储了数据库迁移文件，其中 0001_initial.py 是创建 Student 模型表的初始迁移。

此外，项目根目录中的 app.py 文件是实习中要求的客户端，通过命令行的控制，用于对后台数据库中数据的增删改查等工作；sql 文件夹下存放建立数据库的 sql 命令；而 dace.doc 文件则是本次实习的实习报告，其中包含项目的详细文档、使用指南和开发说明等。整个项目结构遵循 Django 的最佳实践，使得代码组织清晰，易于维护和扩展。

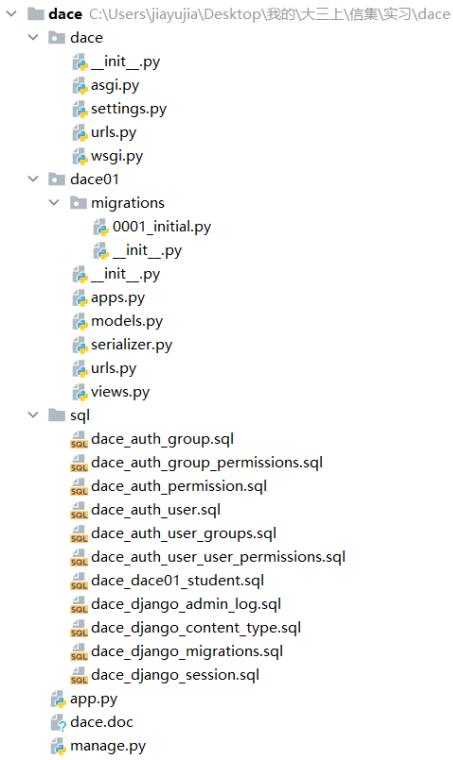

图 1 dace 的项目组织结构

2. 项目功能

这个项目包括网页端、客户端和服务端三个部分。网页端负责收集人员的基本资料，具体包括五项内容：姓名、学号、邮箱、手机号码和个人兴趣。在这些信息中，学号、邮箱和手机号码需要通过网页端进行格式校验，以确保它们符合

特定的规则；而姓名和个人兴趣则在网页前端验证限制字符长度，姓名的字符数不超过 8 个中文字符，个人兴趣的字符数不超过 32 个中文字符。

（1）网页端

①信息收集：收集学生信息提交给服务端

②信息展示：展示已有信息

③信息验证：验证所填信息是否合乎规范

④信息查询：查询想要的数据展示在展示界面

⑤信息更新：对信息进行数据更新

（2）客户端（app.py）

①数据展示：罗列出 mysql 数据库中已有信息

②增添数据：增加一条学生信息数据，不允许缺省值

③删除数据：删除一条指定 id 的数据

④更新数据：对指定 id 的数据的某些字段进行更新

⑤更新 id：更新数据库 id 为防止删除某条数据后 id 不从 1 开始连续自增

⑥帮助：展示 App 的帮助提示

⑦版本：展示与 InfoManager 相关的版本信息

（3）服务端

①API 路由管理：定义清晰的路由系统，使用 Django REST framework 的路由器自动生成 API 端点，用于处理学生信息的增删改查请求。

②数据模型定义：定义了 Student 模型与数据库中的表结构相对应，用于存储学生的姓名、学号、邮箱、手机号码和爱好等信息。

③序列化与反序列化：将模型实例转换为 JSON 格式和将 JSON 数据转换为模型实例以便在 API 中传输和接收数据。

④视图集与 CRUD 操作：提供一个完整的 CRUD 接口，允许前端通过 RESTful API 与后端进行交互，实现学生信息的管理。

⑤数据库迁移：通过迁移文件管理数据库的变更，同步数据库结构与模型定义。

四、部署方法

①安装 MySQL 8.0 及以上版本（如果已有可跳过此步骤，如果低于此版本需完全卸载再安装最新版本）。

②启动 MySQL 服务（这个步骤不可或缺，每次使用前都要确保 MySQL 的服务已经开启，否则服务器无法连接 mysql 数据库）。

③打开 MySQL Workbench 8.0 CE，以下面的信息登录 mysql 数据库如下图 2（密码默认为 123456，我设置的为 152935，这个根据实际情况填写）。连接后，新建数据库 dace，选项默认不需要改动。

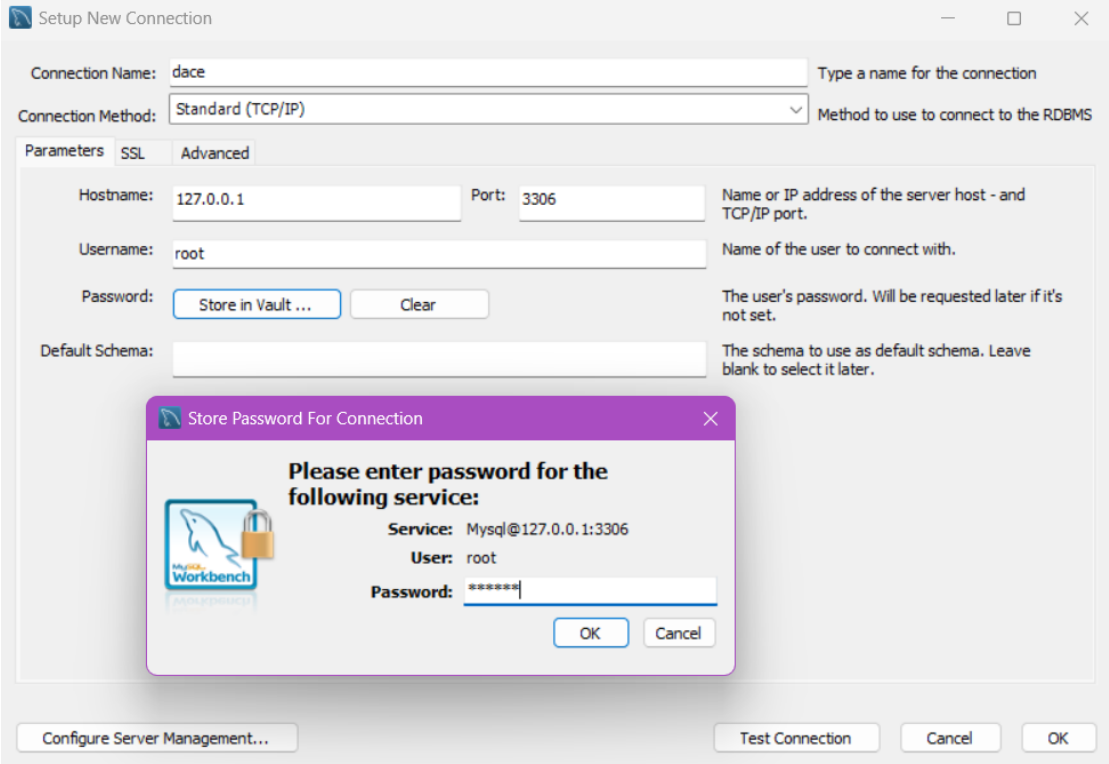

图 2 连接 mysql 数据库

④打开 cmd，输入 pip install Django==5.1.3，在 python 解释器中安装 Django。

⑤打开 pycharm，打开项目，配置解释器为 python 3.10.11，打开 dace 文件夹中的 settings.py，修改 DATABASES 的配置信息如下图 3。

```
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.mysql",  # 数据库引擎，不用改
        "NAME": "dace",  # 数据库名称
        "HOST": "127.0.0.1",  # mysql数据库的ip地址，如果是本机，则写127.0.0.1，127.0.0.1不行则填localhost
        "USER": "root",  # mysql数据库的登录用户名
        "PASSWORD": '152935',  # mysql数据库登录的密码，根据自己的设定修改
        "PORT": "3306",  # mysql数据库的端口号，默认是3306
    }
}
```

图 3 修改项目数据库属性配置

⑥在 pycharm 终端中，首先输入“python manage.py migrate”进行数据迁移，把项目的数据库的表、字段在电脑中迁移过来便于使用；接着输入“python manage.py runserver”，成功运行服务器，此时访问下方链接 http://127.0.0.1:8000/ 即可进入网页端（如下图 4）；如果提示“no module named xxx”就输入“pip install xxx”一下（这里面需要科学上网或者换国内镜像源）再重新“python manage.py runserver”直到服务器启动即可。

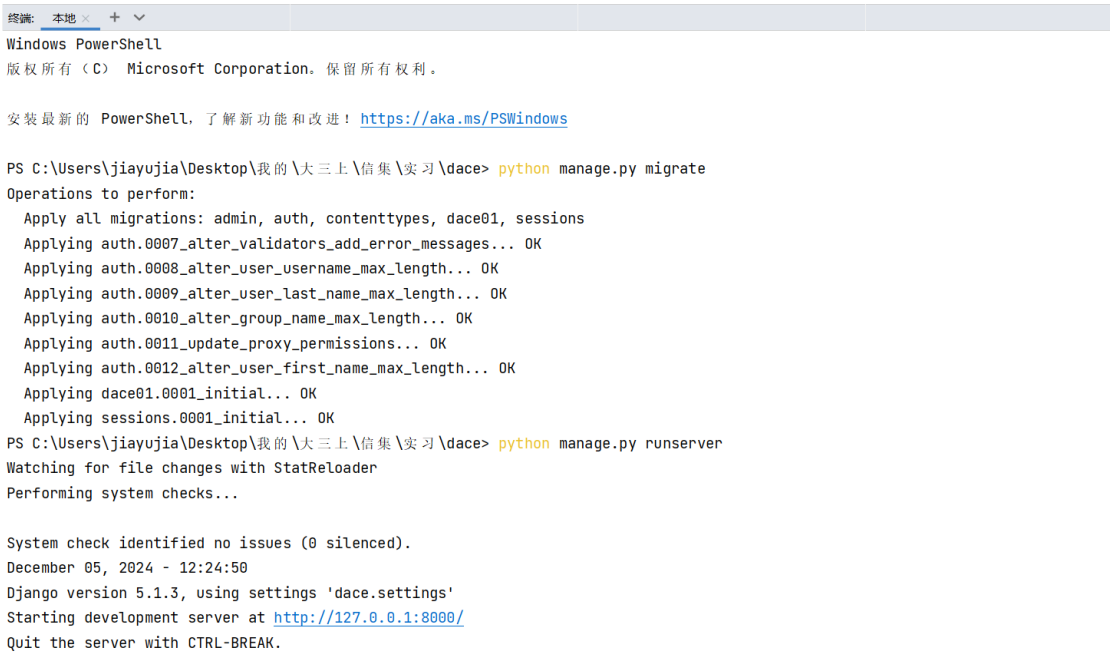

图 4 数据库迁移、服务器运行

⑦在 pycharm 的此环境中，配置所有依赖库后，运行 app.py，在命令行中对数据库进行相应的操作（详解操作方式见后文）

五、网页前端

1. 界面 UI

Django REST framework（DRF）是一个强大的工具包，能够帮助我们基于 Django 框架构建 Web API，能够利用框架内置函数完成前端框架的显示，节约了单独开发前端的时间。启动服务器后，在浏览器中访问 http://127.0.0.1:8000/，得到下面的界面（如下图 5），该页面展示了 DRF 的默认根视图，显示了一个简单的 JSON 响应，其中包含了一个链接到 Student 资源的 URL，指向了 Student 模

型的列表视图，允许客户端访问学生信息的集合；响应中还列出了网页允许的 HTTP 方法，如 GET、HEAD 和 OPTIONS。GET 方法用于请求资源，HEAD 方法用于获取资源的元数据而不返回资源本身，而 OPTIONS 方法用于描述目标资源的通信选项；响应头中的 Content-Type 指定了返回内容的媒体类型为 application/json，而 Vary: Accept 头部表明服务器会根据客户端请求的 Accept 头部来返回不同的内容类型。DRF 的根视图作为一个 API 的导航点，允许客户快速找到并理解 API 的结构，通过这种方式，Django REST framework 提供了一个清晰、直观且易于使用的 API 接口，使得前后端分离的开发模式更加高效。

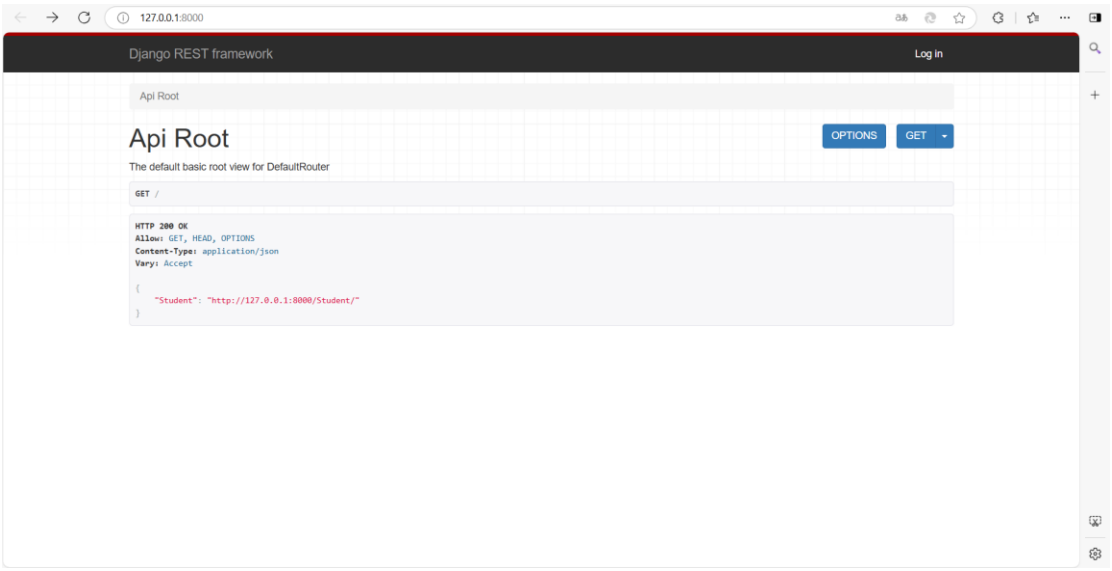

图 5 默认根视图

点击 http://127.0.0.1:8000/Student/，跳转学生信息采集界面（如下图 6），该界面展示了一个针对 Student 资源的 API 端点。这个端点允许使用 GET、POST、HEAD 和 OPTIONS 等 HTTP 方法，禁止 DELETE 和 UPDATE 方法从而保护学生提交信息的安全性和不可修改性(这两个 API 接口已设计但不对网页端暴露)，其中 GET 方法被用来检索学生信息，并返回了一个包含学生姓名、学号、邮箱、手机号码和爱好的 JSON 对象。POST 方法则用于提交新的学生信息，这可以通过在下方的 HTML 表单中填写相应的字段并通过 POST 请求发送到服务器来实现。页面初始化后在 Student 视图中也可以看到当前所有已提交的学生信息，便于方便我们理解 RESTFUL 风格的 API 设计和查看资源的 collection+json 组织方式。StudentViewSet 视图集处理与学生资源相关的所有请求，而路由器则负责将 URL 路径与视图集关联起来。这种设计模式使得 API 的开发和维护变得简洁且

高效，所有资源的变化都会在提交表单刷新后显示出来，给予用户反馈。

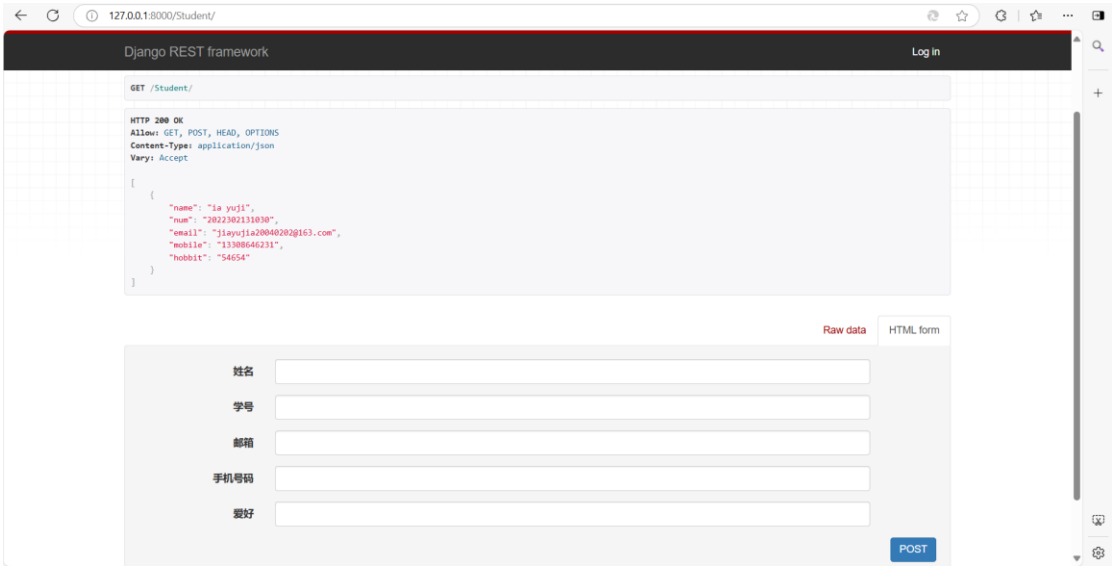

图 6 Student 视图及提交页面

在浏览器搜索框输入 http://127.0.0.1:8000/Student/x 跳转到表示 id 为 x 的学生界面（如下图 7），此页面中 Django REST framework 展示了一个针对单个学生实例和操作的详细页面。这个页面允许使用 GET、PUT、PATCH、DELETE、HEAD 和 OPTIONS 等 HTTP 方法，其中 GET 方法被用来检索特定学生的信息，并返回了一个包含学生姓名、学号、邮箱、手机号码和爱好的 JSON 对象，是随浏览器对服务器资源的请求自动触发的。PUT 方法通常用于更新整个资源，而 PATCH 方法用于部分更新资源。DELETE 方法用于删除该学生记录，HEAD 方法用于获取资源的元数据，OPTIONS 方法用于描述目标资源的通信选项。页面下方的 HTML 表单允许用户通过 PUT 方法更新学生信息。

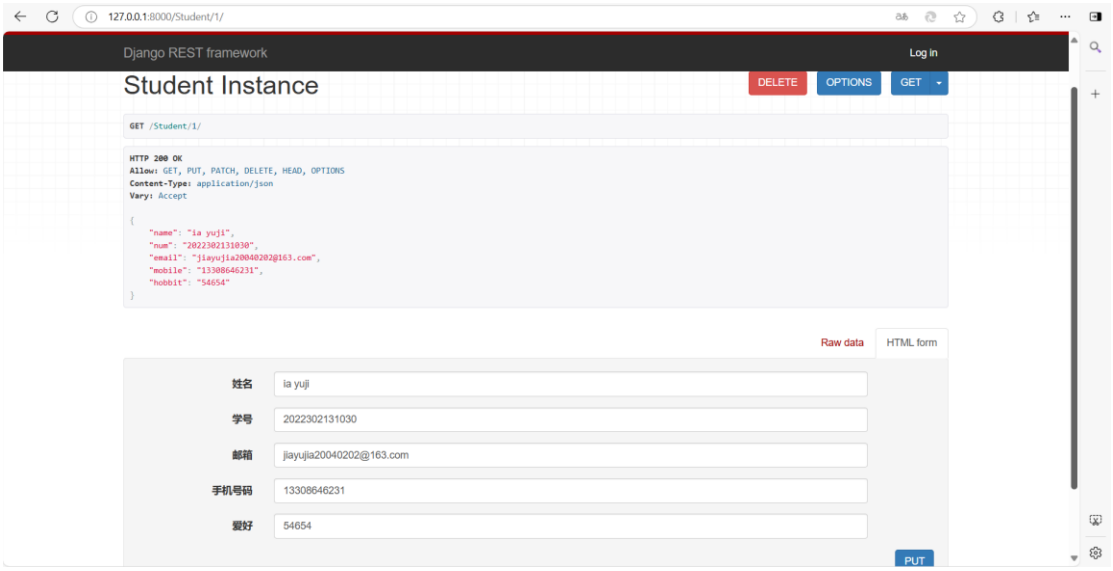

图 7 具体某学生的视图及信息修改界面

2. 功能介绍

在 Student 表单中输入对应的提交信息，点击 POST，前端页面会自行匹配用户输入的内容是否符合要求，符合要求（如下图 8）就以 json+collection 的集合方式提交到后端，后端放入数据库中对前端发送反馈，前端接受反馈自动刷新页面，显示出用户刚刚提交的信息（如下图 9）；如果用户的提交不符合要求，网页端自动做出正则匹配判断，在不符合提交规范的输入框下和上方的资源显示中作出反馈提示（如下图 10），匹配规则如下：

①姓名不超过 8 个字符且不能为空（中文和英文不限制）

②学号为 13 位数字且不能为空

③手机小于等于 11 位数字且不能为空（电话号码都是小于等于 11 位，包括移动和固定电话）

④邮箱认为不能为空，数据库中自动限制为 254 字符长度

⑤兴趣不超过 32 个字符且不能为空（中文和英文不限制）

修改后重新提交符合规范的信息又会显示如图 8 的页面信息，此时可填写新的信息。

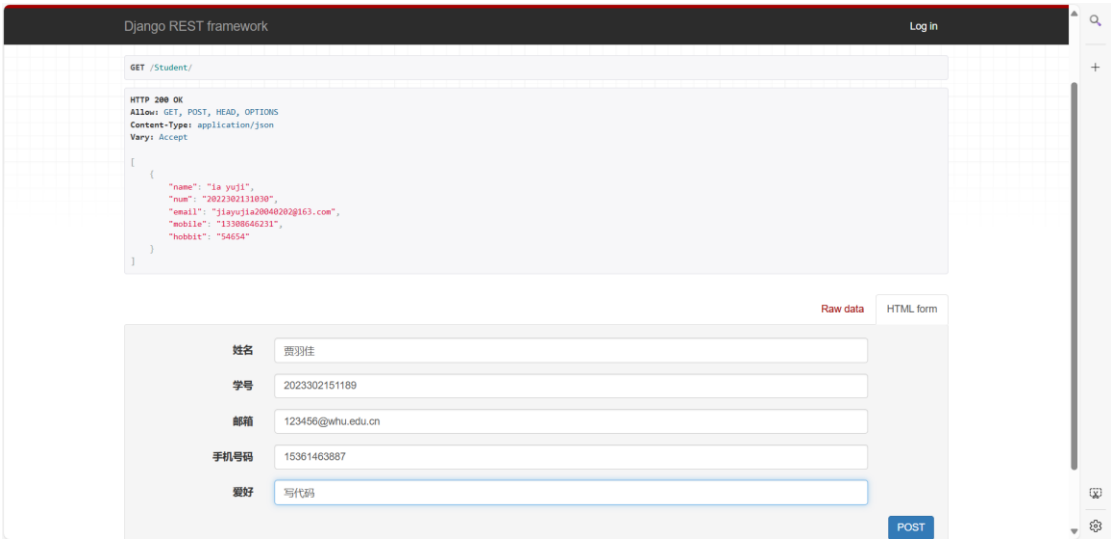

图 8 POST 方法提交符合要求的信息表单

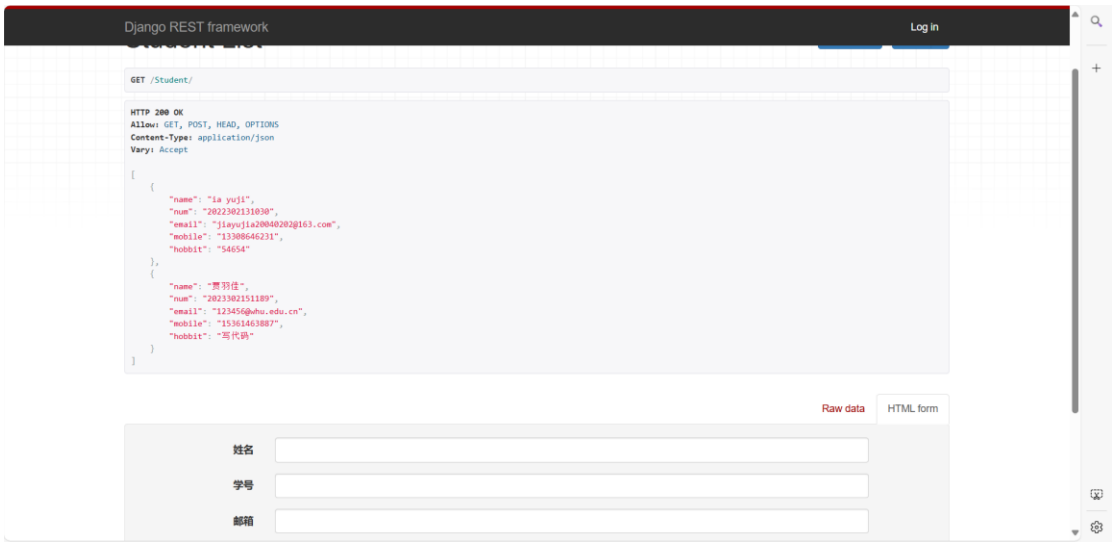

图 9 成功提交后更新资源显示

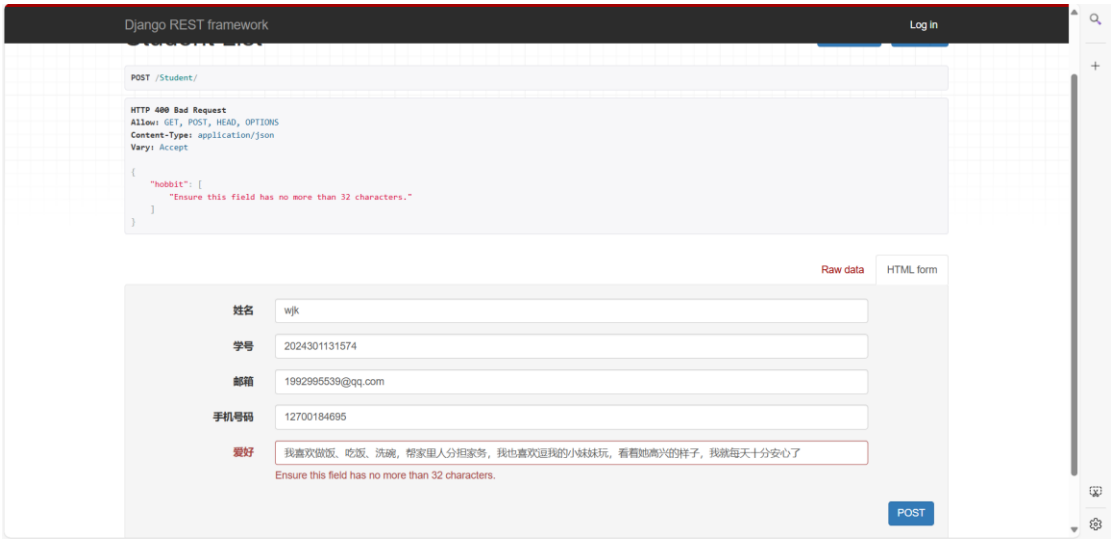

图 10 提交不符合要求的信息表单的反馈

在 Student/x/表单中可以查看某一 id 为 x 的具体学生信息，点击页面中 DELETE，浏览器上方弹出提示框（如下图 11），继续点击 DELETE 可以删除这条信息，重新刷新回到本界面，但此时表单上方的资源状态窗口将不再显示学生信息，表示学生信息已删除（如下图 12）；查看数据库，发现数据库也删除了 id 为 x 的数据，但由此带来一个问题，就是由于数据的删除，数据库 id 可能不再是从 1 开始连续自增的了。为了解决这个问题，在客户端设计采用命令行的方式可以对数据库中的数据进行 id 的连续自增重置，通过异步而不是同步方式解决问题，防止网页端在 id 改变后转移到后端原其他 id 对应的资源，造成资源的歧义性和分配逻辑不合理。

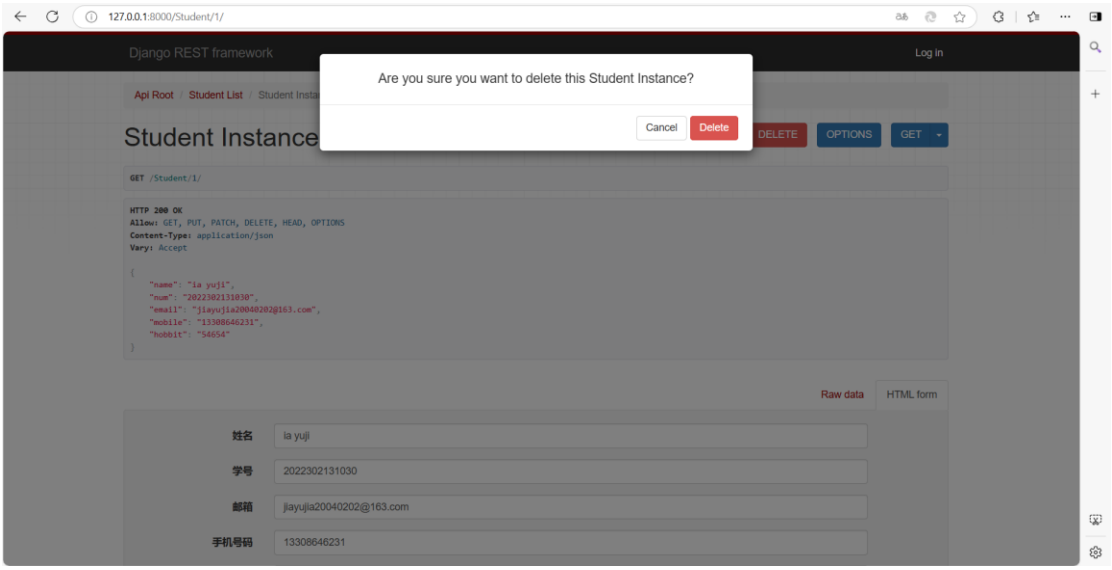

图 11 DELETE 方法删除某条数据

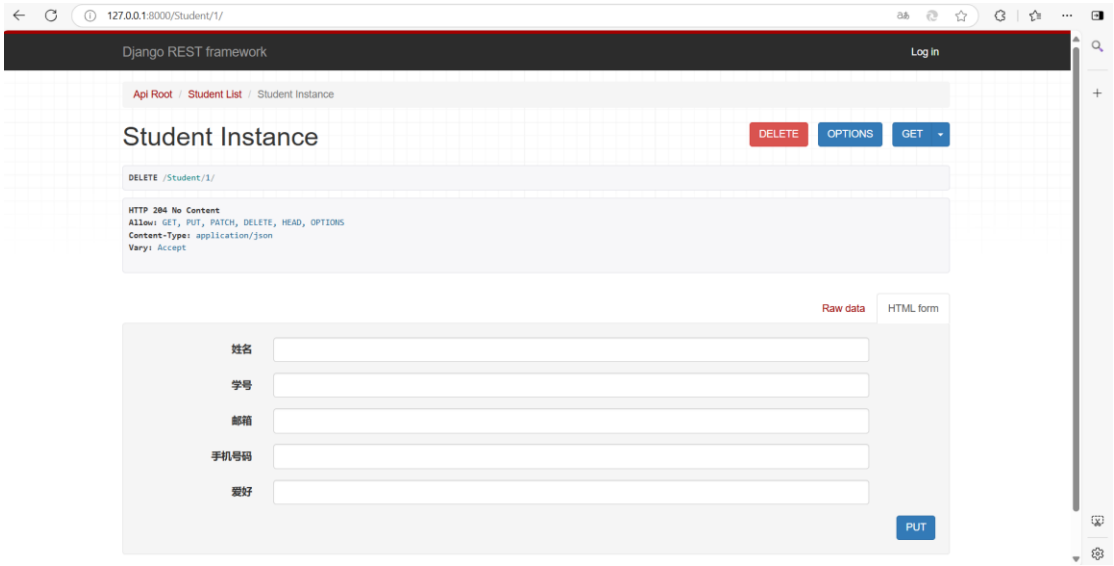

图 12 删除数据后对应资源也发生变化

在 Student/x/表单中也可以通过修改表单的信息并 PUT 来修改某条信息（如下图 13），我们以手机号为例进行修改，将学生手机号从 153 开头变更为 127 开头，修改表单后点击 PUT，如果修改的信息符合正则匹配要求，可以正常提交，页面刷新并显示已经修改完成的信息（如下图 14），检查 mysql 数据库发现对应 id 的手机号也成功变更的 127 开头；如果修改的信息不符合正则匹配要求，与 POST 方法一样在出问题的输入框下提示错误信息，修改后才能继续 POST 提交。

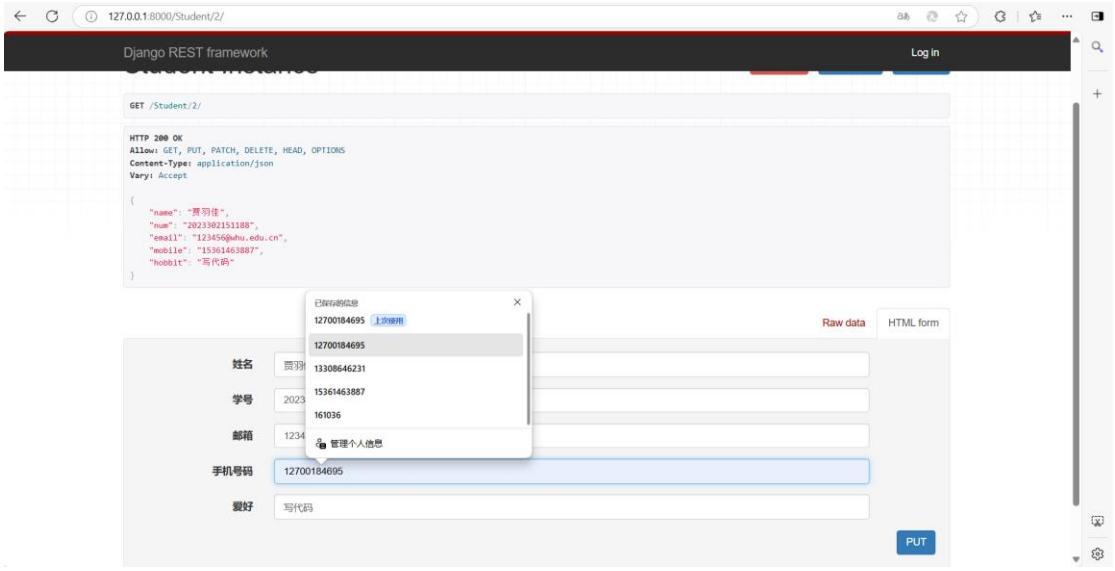

图 13 PUT 方法修改数据

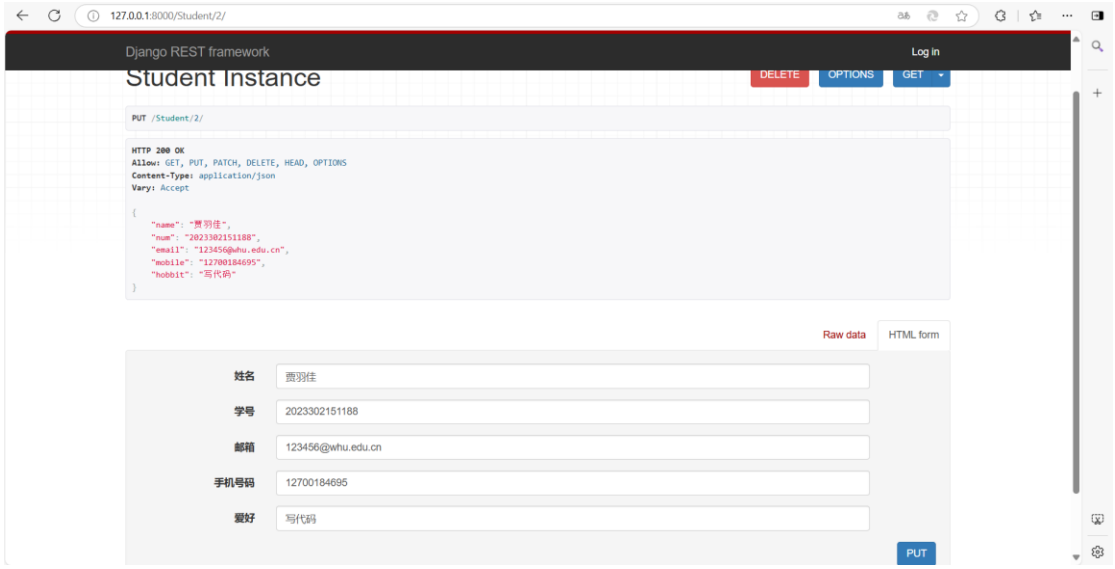

图 14 成功修改后显示新数据

六、客户端命令行工具

运行 app.py，会弹出一个持续监听的命令行窗口。该窗口监听用户的输入，将用户的输入看成字符串，以空格对字符串进行分割，得到不同的指令和内容，最后对指令进行识别，将指令附加的数据与数据库进行连接，从而完成数据的增删改查，其定位是为管理员提供服务器数据管理的工具。

一些不需要修改数据的客户端命令行功能如下图 15，包含显示帮助信息、显示版本信息、显示信息功能，其中-l 和—list、-h 和—help、-v 和—version 等价。代码的实现如图 16。

```
D:\Python\Python3.10.11\python.exe C:/Users/jiayujia/Desktop/我的/大三上/信集/实习/dace/app.py
App>App -h
用例: App>App [选项]
    可用的选项:
      -h, --help                显示帮助信息
      -v, --version             显示版本信息
      -a, --add                 增加信息
      -d, --delete              删除信息
      -u, --update              修改信息
      -l, --list                显示信息
      -r, --reset               更新id为从1开始连续自增
    注意: 命令行必须以App作为开头
App>App --list asc
当前数据库中一共有2条数据
id: 1, name: ia yuji, number: 2022302131030, email: jiayujia20040202@163.com, phone: 13308646231, hobby: 54654
id: 2, name: 贾羽佳, number: 2023302151189, email: 123456@whu.edu.cn, phone: 15361463887, hobby: 写代码
App>App -l
当前数据库中一共有2条数据
id: 2, name: 贾羽佳, number: 2023302151189, email: 123456@whu.edu.cn, phone: 15361463887, hobby: 写代码
id: 1, name: ia yuji, number: 2022302131030, email: jiayujia20040202@163.com, phone: 13308646231, hobby: 54654
App>App -v
App 1.0.0，为管理员对后台数据进行管理而设计，仅供内部使用，开发人员jyj
App>
```

图 15 -h、-l 与-v 的功能

```
def help():
    print('''用例: App>App [选项]
    可用的选项:
      -h, --help                显示帮助信息
      -v, --version             显示版本信息
      -a, --add                 增加信息
      -d, --delete              删除信息
      -u, --update              修改信息
      -l, --list                显示信息
      -r, --reset               更新id为从1开始连续自增
    注意: 命令行必须以App作为开头''')  # 帮助信息


# 版本信息-v --version
def version():
    print('App 1.0.0，为管理员对后台数据进行管理而设计，仅供内部使用，开发人员jyj')  # 版本号


# 显示信息-l --list
def list(flag):
    db = pymysql.connect(host='localhost', user='root', password='152935', db='dace', charset="utf8")
    cursor = db.cursor()
    if flag:  # 如果是默认显示，则降序排序
        cursor.execute("select * from dace01_student order by id desc")  # 降序
    else:
        cursor.execute("select * from dace01_student order by id asc")  # 升序
    result = cursor.fetchall()  # 获取所有数据
    print('当前数据库中一共有%s条数据' % len(result))
    for row in result:  # 遍历数据并输出
        print('id: %s, name: %s, number: %s, email: %s, phone: %s, hobby: %s' % (row[0], row[1], row[2], row[3], row[4],
                                                                                 row[5]))
```

图 16 -h、-l 与-v 的代码

添加数据使用功能（图 17）和代码（图 18）。

```
App>App -a -n 林则徐 -i 2023302131555 -e linzexu@whu.edu.cn -m 14523875642 -h 虎门销烟
已成功添加数据
App>
```

	id	name	num	email	mobile	hobbit
▶	2	贾羽佳	2023302151188	123456@whu.edu.cn	12700184695	写代码
	3	林则徐	2023302131555	linzexu@whu.edu.cn	14523875642	虎门销烟
*	NULL	NULL	NULL	NULL	NULL	NULL

图 17 添加数据命令和数据库数据添加情况

```python
# 增加信息-a --add
def add(name, num, email, mobile, hobbit):
    db = pymysql.connect(host='localhost', user='root', password='152935', db='dace', charset="utf8")
    try:
        with db.cursor() as cursor:
            cursor.execute("SELECT MAX(id) FROM dace01_student")
            result = cursor.fetchone()  # 获取最大id
            max_id = result[0]
            if max_id is None:
                new_id = 1  # id为1
            else:
                new_id = max_id + 1  # id自增
            sql = """
            INSERT INTO dace01_student (id, name, num, email, mobile, hobbit)
            VALUES (%s, %s, %s, %s, %s, %s)
            """
            cursor.execute(sql, (new_id, name, num, email, mobile, hobbit))  # 执行添加
            db.commit()
            if cursor.rowcount > 0:  # 判断是否添加成功
                print("已成功添加数据")
            else:
                print("添加数据失败")
    except Exception as e:
        db.rollback()  # 回滚
        print("发生错误: ", e)
    finally:
        db.close()  # 关闭连接
```

图 18 添加数据代码

修改指定 id 数据功能（图 19）和代码（图 20）。

```
App>App --update=3 -n 林霖 -i 2024151360248
已成功更新id为3的数据name字段
已成功更新id为3的数据学号字段
```

	id	name	num	email	mobile	hobbit
▶	2	贾羽佳	2023302151188	123456@whu.edu.cn	12700184695	写代码
	3	林霖	2024151360248	linzexu@whu.edu.cn	14523875642	虎门销烟

图 19 更改指定 id 数据

```python
def update(args):
    content_id = args[1].split('=')[1]  # 获取id
    content = args[2:]  # 获取修改指令和内容
    for element in content:
        if element == "-n":  # 如果修改的是name
            name = content[content.index("-n") + 1]  # 获取name内容
            db = pymysql.connect(host='localhost', user='root', password='152935', db='dace', charset="utf8")
            try:
                with db.cursor() as cursor:
                    sql = "update dace01_student set name = %s where id= %s "
                    cursor.execute(sql, (name, content_id,))  # 执行修改
                    db.commit()
                    if cursor.rowcount > 0:
                        print("已成功更新id为{}的数据name字段".format(content_id))
                    else:
                        print("未查询到此id")
            except Exception as e:
                db.rollback()  # 回滚
                print("发生错误: ", e)
            finally:
                db.close()
        elif element == "-i":  # 如果修改的是num
            num = content[content.index("-i") + 1]  # 获取num内容
            db = pymysql.connect(host='localhost', user='root', password='152935', db='dace', charset="utf8")
            try:
                with db.cursor() as cursor:
                    sql = "update dace01_student set num = %s where id= %s "
                    cursor.execute(sql, (num, content_id,))  # 执行修改
                    db.commit()
                    if cursor.rowcount > 0:  # 判断是否修改成功
                        print("已成功更新id为{}的数据学号字段".format(content_id))
                    else:
                        print("未查询到此id")
            except Exception as e:
                db.rollback()  # 回滚
                print("发生错误: ", e)
            finally:
                db.close()
```

图 20 更新数据（后续其他字段修改类似）

删除指定 id 数据功能（图 21）和代码（图 22）

```
App>App --delete=3
已成功删除id为3的数据
```

	id	name	num	email	mobile	hobbit
▶	2	贾羽佳	2023302151188	123456@whu.edu.cn	12700184695	写代码
*	NULL	NULL	NULL	NULL	NULL	NULL

图 21 删除指定 id 数据和数据库数据删除情况

```python
# 删除信息-d --delete
def delete(content):
    content_id = content.split('=')[1]  # 获取id
    db = pymysql.connect(host='localhost', user='root', password='152935', db='dace', charset="utf8")
    try:
        with db.cursor() as cursor:
            sql = "DELETE FROM dace01_student WHERE `id` = %s"
            cursor.execute(sql, (content_id,))  # 执行删除
            db.commit()
            if cursor.rowcount > 0:  # 判断是否删除成功
                print("已成功删除id为{}的数据".format(content_id))
            else:
                print("未查询到此id")
    except Exception as e:
        db.rollback()  # 回滚
        print("发生错误: ", e)
    finally:
        db.close()  # 关闭连接
```

图 22 删除指定 id 数据代码

重新将 id 更新为从 1 开始连续自增，防止删除数据后 id 不连续的问题，功能图 23，代码图 24。

```
App>App -r
已成功更新id排列
App>App -l
当前数据库中一共有1条数据
id: 贾羽佳, name: 2023302151188, number: 123456@whu.edu.cn, email: 12700184695, phone: 写代码, hobby: 1
```

	name	num	email	mobile	hobbit	id
▶	贾羽佳	2023302151188	123456@whu.edu.cn	12700184695	写代码	1
*	NULL	NULL	NULL	NULL	NULL	NULL

图 23 id 更新和数据库 id 变化情况

```python
def reset():
    db = pymysql.connect(host='localhost', user='root', password='152935', db='dace', charset="utf8")
    try:
        with db.cursor() as cursor:
            # 分开执行每个ALTER TABLE语句
            sql1 = 'ALTER TABLE dace01_student DROP id'
            cursor.execute(sql1)  # 执行删除id
            db.commit()

            sql2 = 'ALTER TABLE dace01_student ADD id int(10) not null'
            cursor.execute(sql2)  # 执行添加id
            db.commit()

            sql3 = 'ALTER TABLE dace01_student MODIFY column id int(10) not null auto_increment, ADD PRIMARY KEY (id)'
            cursor.execute(sql3)  # 执行重置id
            db.commit()

            if cursor.rowcount > 0:  # 判断是否修改成功
                print("已成功更新id排列")
            else:
                print("未查询到此id")
    except Exception as e:
        db.rollback()  # 回滚
        print("发生错误: ", e)
    finally:
        db.close()
```

图 24 id 更新代码

整体代码监听及逻辑控制语句如下图 25，它将上面实现的数据增删改查等功能结合在一起，通过不断监听用户输入作出相应的反馈，同时对错误输入进行提示以规避程序的错误输入与崩溃。至此完成了客户端命令行 App 的构建。

```python
def start():
    while True:
        args = input('App>').split(' ')  # 命令行参数
        args_len = len(args)  # 命令行参数长度
        if args_len > 0 and args[0] != 'App':  # 命令开头必须是App开头
            print('请以App作为命令行的开头')
        else:
            if args[1] == '--help' or args[1] == '-h':  # 帮助
                help()
            elif args[1] == '--version' or args[1] == '-v':  # 版本
                version()
            elif args[1] == '--list' or args[1] == '-l':  # 列表
                if args_len == 2:  # 默认降序
                    list(True)
                elif args_len == 3 and (args[2] == 'ascend' or args[2] == 'asc'):  # 升序
                    list(False)
                else:
                    print('输入错误，请重试')
            elif re.match(r'--delete=\d+', args[1]) or re.match(r'-d=\d+', args[1]):  # 删除
                delete(args[1])
            elif args[1] == '--add' or args[1] == '-a':  # 增加
                if args_len == 12 and args[2] == '-n' and args[4] == '-i' and args[6] == '-e' and args[8] == '-m' and \
                        args[10] == '-h':  # 参数正确
                    add(args[3], args[5], args[7], args[9], args[11])
                else:
                    print('输入错误，请检查是否缺项/多项/顺序有误')
            elif re.match(r'--update=\d+', args[1]) or re.match(r'-u=\d+', args[1]):  # 修改
                update(args)
            elif args[1] == '--reset' or args[1] == '-r':  # 重置id
                reset()
            else:
                print('输入错误，请重试')
```

图 25 监听执行及逻辑控制代码

七、服务端

1. 技术路线概述

采用技术框架 Django REST framework 提供了定义序列化器 Serializer 的方法，可以快速根据 Django ORM 或者其他库自动序列化或反序列化，节省了组织 collection+json 模式的过程。服务端的逻辑基于视图集（ViewSets）和路由器（Routers）。在我的项目中，StudentViewSet 是一个视图集，它封装了对 Student 模型进行 CRUD 操作的逻辑。路由器 DefaultRouter 自动为视图集生成标准的 RESTful 端点。当客户端访问根视图时，DRF 会提供一个链接到所有注册视图集的列表，使得网页端可以进一步与这些端点交互。

2. 项目新建与相关配置修改

首先新建项目，在终端输入“django-admin startproject dace”，随后切换到 dace 项目根目录下，输入“python manage.py startapp dace01”以新建 1 个 app。

在新建项目后，首先修改 settings.py，添加以下配置（如下图 26）。

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'dace01'
]

DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.mysql",  # 数据库引擎，不用改
        "NAME": "dace",  # 数据库名称
        "HOST": "127.0.0.1",  # mysql数据库的ip地址，如果是本机，则写127.0.0.1，127.0.0.1不行则填localhost
        "USER": "root",  # mysql数据库的登录用户名
        "PASSWORD": '152935',  # mysql数据库登录的密码，根据自己的设定修改
        "PORT": "3306",  # mysql数据库的端口号，默认是3306
    }
}

LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_L10N = True
USE_TZ = True
STATIC_URL = '/static/'
```

图 26 settings.py 相关配置修改

3. 准备 model

打开 dace/dace01/models.py，输入下面（图 27）的代码进行模型的建立，并

对输入的内容进行匹配限定长度。

```
# 定义了一个名为Student的模型
class Student(models.Model):
    name = models.CharField(max_length=8, null=False, verbose_name='姓名')
    num = models.CharField(max_length=13, null=False, verbose_name='学号')
    email = models.EmailField(null=False, verbose_name='邮箱')
    mobile = models.CharField(max_length=11, null=False, verbose_name='手机号码')
    hobbit = models.CharField(max_length=32, null=False,verbose_name='爱好')
```

图 27 建立 Student 模型

Student 模型定义好以后，输入“python manage.py makemigrations”和“python manage.py migrate”来更新 mysql 数据库。

4. DRF 编程实现 RESTful 接口

DRF 编程很关键的一步是定义 Serializer 类，用于将 model 数据序列化。其使用方式与 Django Form 表单非常相似。DRF 视图可采用函数式编程，或 Class Based View 视图类的方式编程，并且 DRF 内置了各种通用视图类来简化编程。

首先自定义 Serializer 类。新建 dace/dace01/serializer.py，输入以下代码（图 28）。

```
from rest_framework import serializers
from .models import Student


# 定义了一个名为StudentSerializer的类，继承自serializers.HyperlinkedModelSerializer。该类用于序列化和反序列化Student模型的数据
class StudentSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Student
        fields = ('name', 'num', 'email', 'mobile', 'hobbit')
```

图 28 定义 StudentSerializer 类

DRF 对视图编程，响应提供 json 字节串，不提供 DRF 特有的接口测试页面功能。打开 dace/dace01/views.py 文件，输入以下代码（图 29）建立视图；打开 dace/dace01/urls.py，输入以下代码（图 30）配置路由；最后打开项目的路由本置文件 dace/dace/urls.py，加入一条路由（图 31）。

```
from rest_framework import viewsets
from .serializer import StudentSerializer
from .models import Student


# 建立视图
class StudentViewSet(viewsets.ModelViewSet):
    queryset = Student.objects.all().order_by('num')  # 排序
    serializer_class = StudentSerializer  # 序列化器
```

图 29 建立视图

```python
from django.urls import include, path
from rest_framework import routers
from . import views

router = routers.DefaultRouter()  # 创建一个默认路由器
router.register(r'Student', views.StudentViewSet)  # 将StudentViewSet视图集注册到路由器，路径为/Student/

urlpatterns = [
    path('', include(router.urls)),  # 将路由器的路由添加到URL模式中
    path('api/', include('rest_framework.urls', namespace='rest_framework'))  # 添加RESTful API认证和授权
]
```

图 30 配置路由

```python
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('dace01.urls')),
]
```

图 31 全局路由配置

八、测试

1. API 接口测试

采用 curl 模拟浏览器方式，对 API 接口进行测试。

POST：（相应数据库也变化）

```
C:\Users\jiayujia>curl -X POST -H "Content-Type: application/json" --data-raw "{\"name\": \"Simon\", \"num\": \"2022302151987\", \"email\": \"jcsn@163.com\", \"mobile\": \"12345678912\", \"hobbit\": \"喝茶\"}" http://localhost:8000/Student/

{"name":"Simon","num":"2022302151987","email":"jcsn@163.com","mobile":"12345678912","hobbit":"喝茶"}
```

	name	num	email	mobile	hobbit	id
▶	贾羽佳	2023302151188	123456@whu.edu.cn	12700184695	写代码	1
	Simon	2022302151987	jcsn@163.com	12345678912	喝茶	2
	NULL	NULL	NULL	NULL	NULL	NULL

GET：

```
C:\Users\jiayujia>curl -X GET http://127.0.0.1:8000/Student/1/
{"name":"贾羽佳","num":"2023302151188","email":"123456@whu.edu.cn","mobile":"12700184695","hobbit":"写代码"}
```

PUT：（相应数据库也变化）

```
C:\Users\jiayujia>curl -X PUT -H "Content-Type: application/json" --data-raw "{\"name\": \"Simon\", \"num\": \"2022302151987\", \"email\": \"whurs@163.com\", \"mobile\": \"12345678912\", \"hobbit\": \"画饼\"}" http://localhost:8000/Student/2/
{"name":"Simon","num":"2022302151987","email":"whurs@163.com","mobile":"12345678912","hobbit":"画饼"}
```

	name	num	email	mobile	hobbit	id
▶	贾羽佳	2023302151188	123456@whu.edu.cn	12700184695	写代码	1
	Simon	2022302151987	whurs@163.com	12345678912	画饼	2

DELETE：（相应数据库也变化）

```
C:\Users\jiayujia>curl -X DELETE http://127.0.0.1:8000/Student/2/
```

	name	num	email	mobile	hobbit	id
▶	贾羽佳	2023302151188	123456@whu.edu.cn	12700184695	写代码	1
	NULL	NULL	NULL	NULL	NULL	NULL

2. 资源模版

本项目还定义了资源模板，也可以修改资源模板中的数据，然后通过 curl 上传数据，从而进行添加数据。资源模版如下图 32。

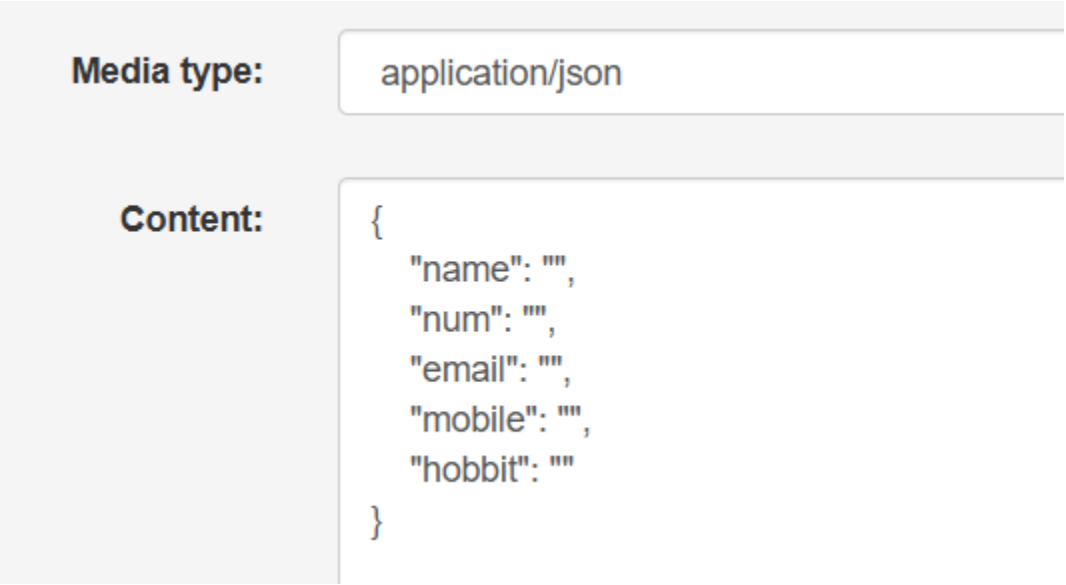

图 32 资源模板的定义

九、思考与总结

在开发这个基于 Django REST framework（DRF）的项目时，我面临了前所未有的挑战，但克服这些困难后的喜悦是难以言表的。

一开始，DRF 的复杂性让我感到有些不知所措。它的功能丰富而强大，但学习曲线相当陡峭。我花费了很长时间，钻研 csdn 上一些大神的博客，观看在线教程，解决一些奇奇怪怪的 bug 和因版本不同的报错，这极大地提高了我的耐心和 debug 的能力。

在项目中，序列化器的复杂性是一个巨大的挑战。我需要处理的不仅仅是简单的数据模型，还有那些错综复杂的关系。我深入研究了 DRF 的序列化机制，包括嵌套序列化器和自定义字段，这让我能够精确地控制数据的传输和展示。

当我克服了这些挑战，我感到一种难以抑制的喜悦。我不仅提升了自己的技

能，还构建出了一个健壮、可扩展和用户友好的 Web API。我为自己的努力感到骄傲，也为能够完成这样一个相对来说比较新颖项目而感到兴奋。这个经历不仅丰富了我的技术栈，也增强了我的信心，让我相信无论遇到多大的困难，只要坚持不懈，总有解决的办法。