数据库课程设计报告

PB20511898
徐泽军

1 前期设计

1.1 系统目标

本系统主要目标为开发一个学生宿舍管理系统。项目采用 B/S 架构，通过 Python 语言实现，借助 Django 实现 Web框架，后台 DBMS 采用 MySQL，开发工具为 JetBrains PyCharm 2024

1.2 需求说明

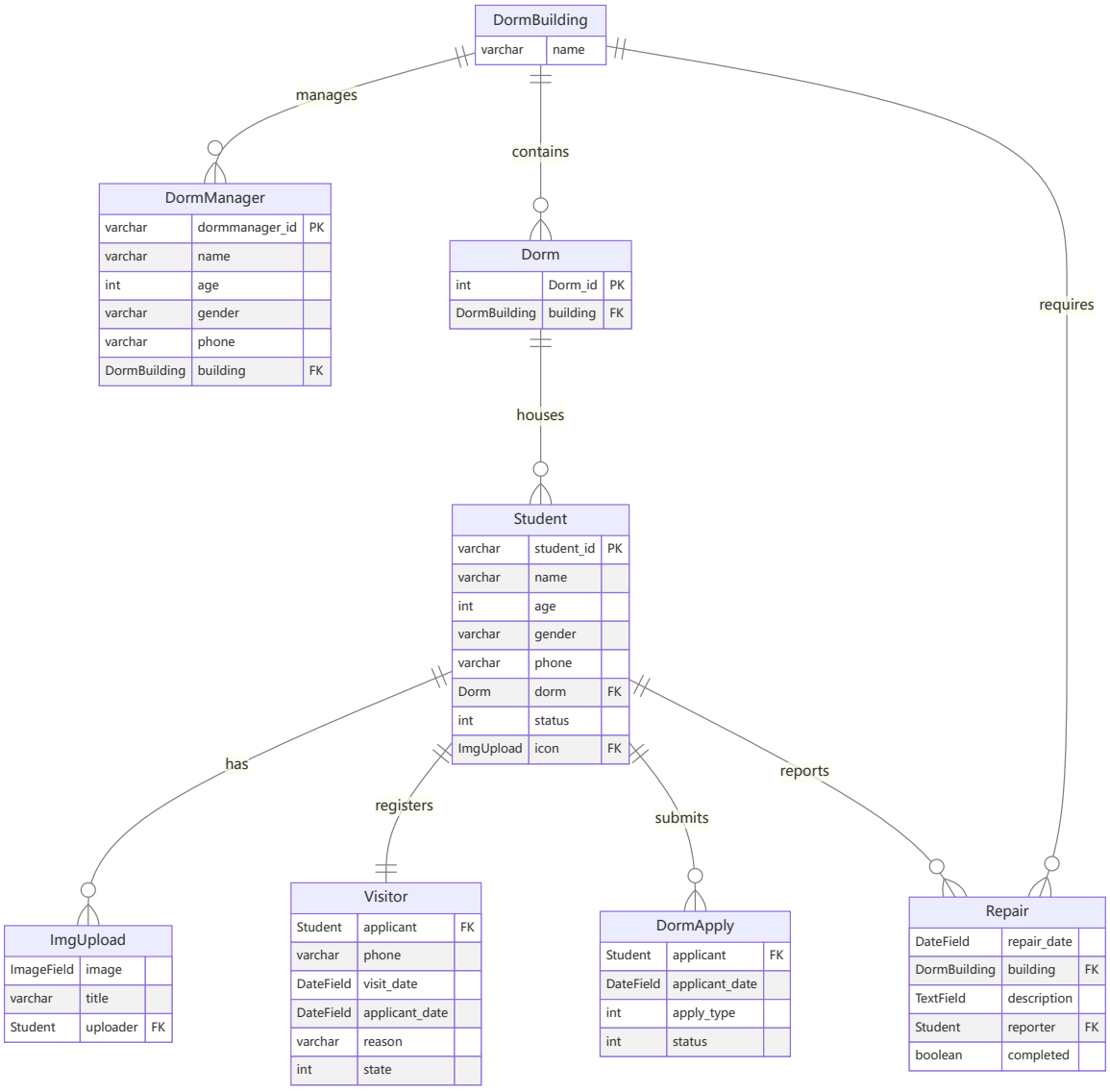
1.2.1 数据需求

- **学生**：学生通过学号`student_id`来唯一确定身份标识。数据库还需要存储学生姓名，年龄，性别，电话等信息。学生还需要属性`status`来确定当前住宿状态，包括未入宿，已入宿，申请入宿中，申请退宿中。已入宿和申请退宿的学生需要保存其目前所在宿舍。学生还需要保存其个人资料中的照片信息。
- **宿管**：宿管通过宿管号`dormmanager_id`来唯一确定身份标识。数据库中还需要存储宿管姓名，年龄，性别，电话。一个宿管只唯一管理一栋宿舍楼，对于已经管理某栋宿舍楼的宿管需要保存其管理楼宇的`id`。
- **宿舍**：宿舍通过`Dorm_id`来唯一确定，还需要标注其所属的楼宇`id`。
- **楼宇**：通过内置`id`唯一确定，还需要保存楼宇名称。
- **申请访客表**：通过内置`id`唯一确定，还需要保存申请人学号，来访人电话，来访日期，申请日期，申请理由。还需要属性`status`来保存申请的状态，包括未处理，已通过，未通过。
- **报修申请表**：通过内置`id`唯一确定，还需要保存报修日期，报修楼宇号，报修描述，报修人学号以及是否处理完成。
- **宿舍申请表**：通过内置`id`唯一确定，还需要保存申请人学号，申请日期，申请类型以及处理状态。

1.2.2 功能需求

- **学生注册**：学生可以登录网址注册账号，填写关键个人信息，设定密码。
- **用户登录**：设定三种用户身份，包括管理员、宿管、学生。不同用户登录对应不同的功能和界面。
- **学生功能**：包括编辑个人信息，提交入宿或退宿申请，提交访客申请，提交报修申请。
- **宿管功能**：包括查看管理楼宇内的所有学生信息，处理学生入宿退宿申请，处理学生报修申请，处理学生访客申请。
- **管理员功能**：包括删除学生账号，增加或删除宿管账号，增添楼宇和宿舍，绑定楼宇与宿管
- **使用体验**：包括退出登录，头像上传与选择，对表格内容进行排序

1.3 ER图



2 实现结构

2.0 创建相关数据表

通过 Django 的 `modeols` 模块进行建表，Django会将更新后的模块与之前的模块进行比较，根据差别自动执行建表、更改表、删除表等操作。代码如下：

```
from datetime import datetime

from django.db import models


# Create your models here.
class DormBuilding(models.Model):
    name = models.CharField(max_length=100)

class DormManager(models.Model):
    dormmanager_id = models.CharField(max_length=100, primary_key=True)
    name = models.CharField(max_length=100)
    age = models.IntegerField()
```

```
    gender = models.CharField(max_length=10)
    phone = models.CharField(max_length=15)
    building = models.ForeignKey(DormBuilding, on_delete=models.SET_NULL,
null=True)


class Student(models.Model):
    student_id = models.CharField(max_length=100, primary_key=True)
    name = models.CharField(max_length=100)
    age = models.IntegerField()
    gender = models.CharField(max_length=10)
    phone = models.CharField(max_length=15)
    dorm = models.ForeignKey('Dorm', on_delete=models.SET_NULL, null=True)
    status = models.IntegerField(default=0)  # 0表示未入住，1表示申请入宿，2表示已入住，
3表示申请退宿
    icon = models.ForeignKey('ImgUpload', on_delete=models.SET_NULL, null=True)

class ImgUpload(models.Model):
    image = models.ImageField(upload_to='img/')
    title = models.CharField(max_length=100,default=str(datetime.now))
    uploader = models.ForeignKey(Student, on_delete=models.CASCADE,null=True)

class Dorm(models.Model):
    Dorm_id = models.IntegerField(max_length=100, primary_key=True)
    building = models.ForeignKey(DormBuilding, on_delete=models.CASCADE)


class Visitor(models.Model):
    applicant = models.ForeignKey(Student, on_delete=models.CASCADE)
    phone = models.CharField(max_length=15)
    visit_date = models.DateField()
    applicant_date = models.DateField(null=True)
    reason = models.CharField(max_length=200, null=True)
    state = models.IntegerField(default=0)


class Repair(models.Model):
    repair_date = models.DateField()
    building = models.ForeignKey(DormBuilding, on_delete=models.SET_NULL,
null=True)
    description = models.TextField()
    reporter = models.ForeignKey(Student, on_delete=models.CASCADE)
    completed = models.BooleanField(default=False)


class DormApply(models.Model):
    applicant = models.ForeignKey(Student, on_delete=models.CASCADE)
    applicant_date = models.DateField(default=datetime.today().date())
    apply_type = models.IntegerField(default=0)  # 0为申请入住，1为申请退宿
    status = models.IntegerField(default=0)  # 0为未处理，1为允许，2为拒绝
```

2.1 用户登录注册以及多用户模块

2.1.1 概要

对于用户认证(User Authentication)我才用的是Django的用户认证API以及用户分组功能，通过API可以实现登录密码加密存储，避免用户密码明文存储在数据库中。

多用户模块通过分组功能实现，并在用户访问相关链接时通过网页返回信息判断是否是合法用户。

用户访问`localhost:8000/login`转到登录界面，访问`localhost:8000/register`转到注册界面。用户注册仅限学生注册，所以对学号格式有相关要求。

用户在网页中填写表单后点击提交按钮，服务器端接受到`POST`请求，读取表单内容，判断是否合法后执行后续步骤，包括向数据库中`Student`表插入新元组，以及通过Django的用户认证API创建新用户。

2.1.2 相关代码

通过Python脚本创建用户分组：

```
import os
import django

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'DataBase_Django.settings')
django.setup()

from django.contrib.auth.models import Group

#创建三个用户组
def create_groups():
    groups = ['DormManager', 'Student', 'SuperUser']
    for group_name in groups:
        group, created = Group.objects.get_or_create(name=group_name)
        if created:
            print(f"Group '{group_name}' created successfully.")
        else:
            print(f"Group '{group_name}' already exists.")


if __name__ == "__main__":
    create_groups()
```

服务器端获取用户`POST`请求后创建新用户：

```
def register_view(request):
    if request.method == 'POST':
        student_id = request.POST.get('student_id')
        name = request.POST.get('name')
        age = request.POST.get('age')
        gender = request.POST.get('gender')
        phone = request.POST.get('phone')
        password = request.POST.get('password')

        print(student_id, name, age, gender, phone, password)

        if not re.match(r'PB\d{8}', student_id):
```

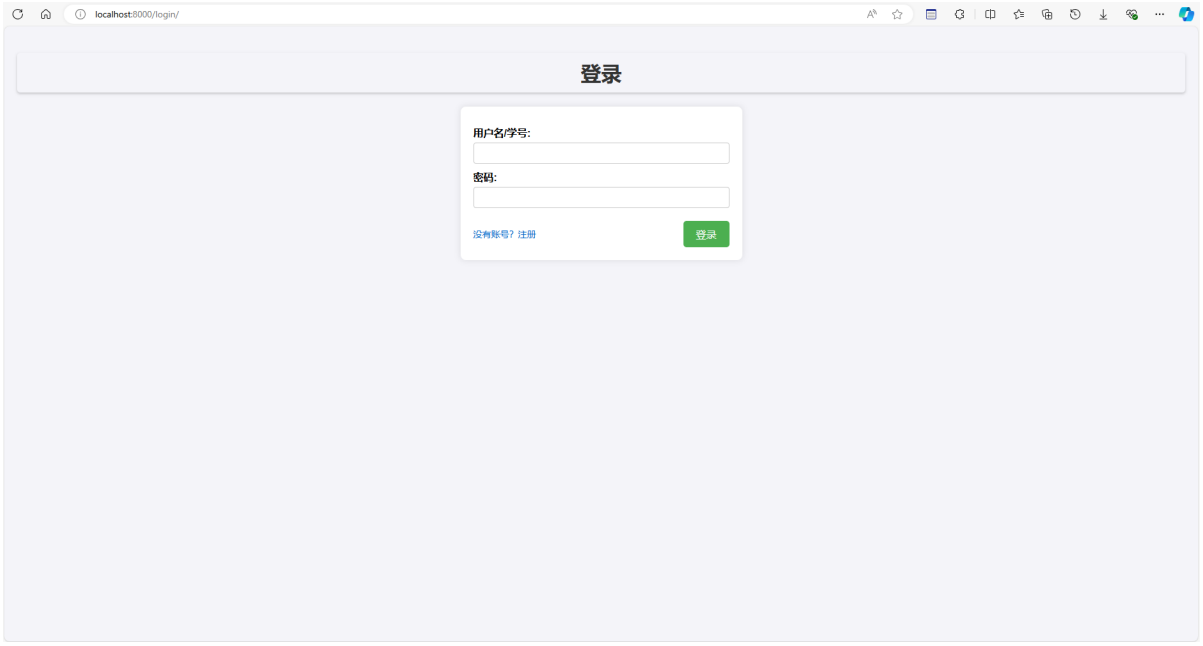
```
            messages.error(request, '学号格式不正确，必须以PB开头，后跟8位数字。')
            return render(request, 'mainpage/register.html')

        if Student.objects.filter(student_id=student_id).exists():
            messages.error(request, '该学号已经注册，请尝试登录或使用其他学号。')
            return render(request, 'mainpage/register.html')

        # 创建学生实例，添加到数据库，保存学生ID和密码到Django内置用户模型中，设置组别
        student = Student(student_id=student_id, name=name, age=age,
gender=gender, phone=phone)
        user = User.objects.create_user(username=student_id, password=password)
        user.groups.add(Group.objects.get(name='Student'))
        user.save()
        student.save()
        messages.success(request, '注册成功，请登录！')
        return redirect('login')
    else:
        return render(request, 'mainpage/register.html')
```

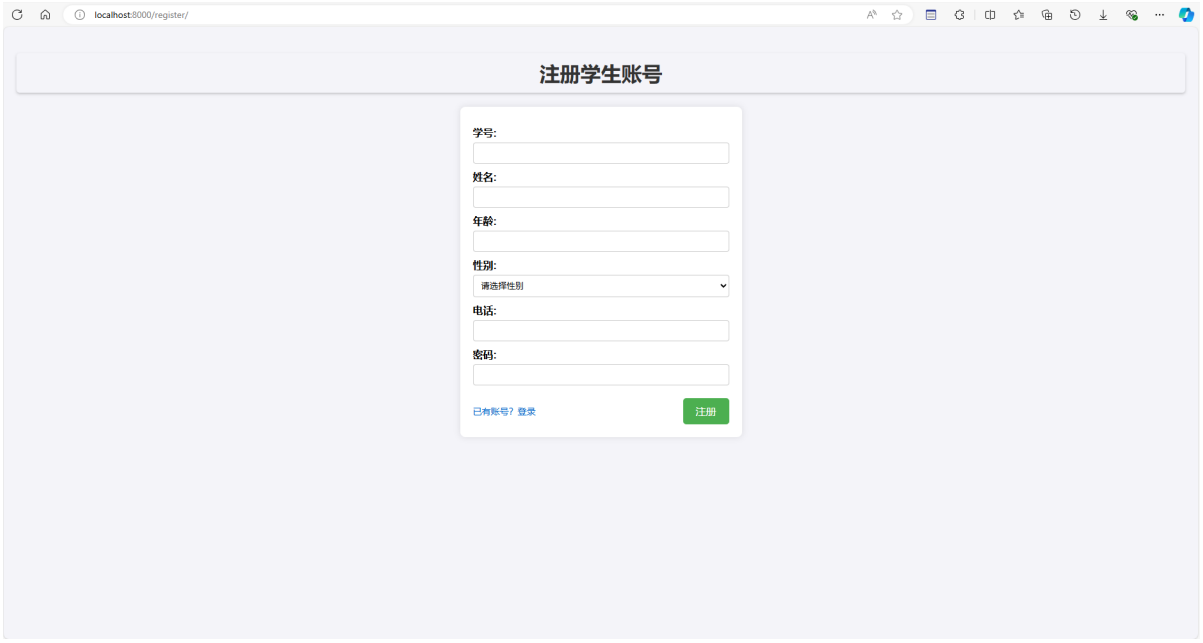
一个根据用户身份不同渲染不同页面的例子，不同身份用户访问`localhost:8000/dashboard`渲染到不同页面：

```
def dashboard(request):
    if request.user.is_authenticated:
        print(request.user)
        if request.user.groups.filter(name='Student').exists():  # 学生登录
            stu = Student.objects.get(pk=request.user.username)
            icon = stu.icon
            context = {'student': stu, 'icon': icon}
            return render(request, 'management/dashboard_stu.html', context)
        elif request.user.groups.filter(name='DormManager').exists(): # 宿管登录
            dm_id = request.user.username
            dm = DormManager.objects.get(pk=dm_id)
            try:
                build = dm.building
                dorm_count = Dorm.objects.filter(building=build.pk).count()
            except DormBuilding.DoesNotExist:
                dorm_count = 0

            try:
                build = dm.building
                stu_count =
Student.objects.filter(dorm__building=build.pk).count()
            except DormBuilding.DoesNotExist:
                stu_count = 0
            except Dorm.DoesNotExist:
                stu_count = 0
            context = {'dorm_count': dorm_count, 'dm': dm, 'stu_count':
stu_count}
            return render(request, 'management/dashboard_dm.html', context)
        elif request.user.groups.filter(name='SuperUser').exists(): # 管理员登录
            building_count = DormBuilding.objects.all().count()
            dorm_count = Dorm.objects.all().count()
            student_count = Student.objects.all().count()
            dormmanager_count = DormManager.objects.all().count()
```

```
            context = {'building_count': building_count, 'dorm_count':
dorm_count, 'student_count': student_count,
                       'dormmanager_count': dormmanager_count}
            return render(request, 'management/dashboard_su.html', context)
    else:
        messages.error(request, '用户未登录！')
        return redirect('login')
```

2.1.3 页面展示





2.2 学生模块

2.2.1 概要

学生账户登录后会展示相关个人信息。左侧展示相关功能跳转按钮，包括入宿退宿，访客申请，报修申请。右上角提供编辑个人资料以及退出登录按钮。

对于入宿退宿功能，当用户点击申请入宿/退宿时，浏览器会向服务器提交一条`POST`请求，服务器则向数据库中`DormApply`表插入一条新元组，同时通过触发器更改学生的状态。

对于访客申请功能，当用户填写表单并提交时，服务器接收到`POST`请求，对申请访问的时间进行合法性判断，合法申请则会向数据库中`Visitor`表插入一条新元组。

对于报修申请，当用户选择楼宇，填写相关描述并提交时，服务器接收到`POST`请求，接着向数据库中`Repair`表插入一条新元组。

对于编辑个人资料，当用户点击时跳转到编辑资料页面，在编辑资料页面还可以再跳转到上传头像页面。当用户从本地选择图片并上传时，服务器会将图片复制到相关目录下并向数据库中`ImgUpload`表插入一条新表项，属性包括图片ID，图片标题，图片地址以及上传者学号。

上述所有功能都基于身份验证的前提下。

2.2.2 相关代码

入宿退宿相关代码：

```
# 选取功能中与学生相关代码
        if request.user.groups.filter(name='Student').exists():
            if request.method == 'POST':
                applicant = Student.objects.get(pk=request.user.username)
                status = Student.objects.get(pk=request.user.username).status
                apply_type = 0
                if status == 1 or status == 3:
                    messages.error(request, '您有未处理的申请，不能再申请')
                    return redirect('inoutdorm')
                elif status == 0:
                    apply_type = 0
                elif status == 2:
                    apply_type = 1

                apply = DormApply(applicant=applicant, apply_type=apply_type)
                apply.save()
                return redirect('inoutdorm')
            else:
                applications =
DormApply.objects.filter(applicant=request.user.username).order_by('applicant_dat
e')
                applications_data = []
                for apply in applications:
                    applications_data.append({
                        'applicant': apply.applicant,
                        'applicant_date': str(apply.applicant_date),
                        'apply_type': apply.apply_type,
                        'status': apply.status,
                    })
                student_id = request.user.username
                student = Student.objects.get(pk=student_id)
                context = {'applications': applications_data, 'student': student}

                return render(request, 'management/inoutdorm_stu.html', context)
```

访客申请相关代码：

```
# 选取功能中与学生相关代码
        if request.user.groups.filter(name='Student').exists():
```

```
            if request.method == 'POST':
                phone = request.POST.get('phone')
                reason = request.POST.get('reason')
                applicant = Student.objects.get(pk=request.user.username)
                visit_date = datetime.strptime(request.POST.get('visit_date'),
'%Y-%m-%d').date()
                applicant_date = datetime.today().date()
                print(request.user.username, applicant_date)
                application_count =
Visitor.objects.filter(applicant=request.user.username,

applicant_date=applicant_date).count()
                visitors = Visitor(applicant=applicant, visit_date=visit_date,
phone=phone, reason=reason,
                                   applicant_date=applicant_date)

                print(application_count)
                if application_count >= 3:
                    messages.error(request, '今日已申请三次访问')
                    return redirect('visit')
                elif applicant_date > visit_date:
                    messages.error(request, '不可申请今天之前的访问')
                    return redirect('visit')
                else:
                    visitors.save()
                    return redirect('visit')
            else:
                visitors =
Visitor.objects.filter(applicant=request.user.username).order_by('applicant_date'
)
                visitors_data = []
                for visitor in visitors:
                    visitors_data.append({
                        'visitor_id': visitor.pk,
                        'visit_date': str(visitor.visit_date),
                        'visitor_phone': visitor.phone,
                        'visitor_reason': visitor.reason,
                        'visitor_applicant': visitor.applicant,
                        'applicant_date': str(visitor.applicant_date),
                        'state': visitor.state,
                        'reason': visitor.reason,
                    })
                context = {'visitors': visitors_data}

                return render(request, 'management/visitor_stu.html', context)
```

报修申请相关代码：

```
# 选取功能中与学生相关代码
        if request.user.groups.filter(name='Student').exists():
            if request.method == 'POST':
                repair_date = datetime.today().date()
                building =
DormBuilding.objects.get(name=request.POST.get('building'))
                description = request.POST.get('description')
```

```
                reporter = Student.objects.get(pk=request.user.username)
                repair_report = Repair(repair_date=repair_date,
building=building, description=description,
                                       reporter=reporter)
                repair_report.save()
                return redirect('repair')
            else:
                building = DormBuilding.objects.all().values('name')
                repairs = Repair.objects.filter(reporter=request.user.username)
                repairs_data = []
                for repair in repairs:
                    repairs_data.append({
                        'repair_id': repair.pk,
                        'repair_date': str(repair.repair_date),
                        'building': repair.building.name,
                        'description': repair.description,
                        'reporter': repair.reporter,
                        'completed': repair.completed,
                    })
                context = {'building': building, 'repair': repairs_data}

                print(context)
                return render(request, 'management/repair_stu.html', context)
```

编辑个人资料代码：

```
def edit_student(request):
    if request.user.is_authenticated:
        if request.user.groups.filter(name='Student').exists():
            if request.method == 'POST':
                stu = Student.objects.get(pk=request.user.username)
                name = request.POST.get('name')
                age = request.POST.get('age')
                gender = request.POST.get('gender')
                phone = request.POST.get('phone')
                password = request.POST.get('password')
                icon = request.POST.get('icon')
                stu.icon = ImgUpload.objects.get(pk=icon)
                stu.name = name
                stu.age = age
                stu.gender = gender
                stu.phone = phone
                user = request.user
                user.set_password(password)
                user.save()
                stu.save()
                messages.success(request, "更改信息成功")
                return redirect('dashboard')
            else:
                img = ImgUpload.objects.filter(uploader=request.user.username)
                context = {'img': img}
                return render(request, 'management/edit_student_stu.html',
context)
        else:
            messages.error(request, "用户身份错误")
```

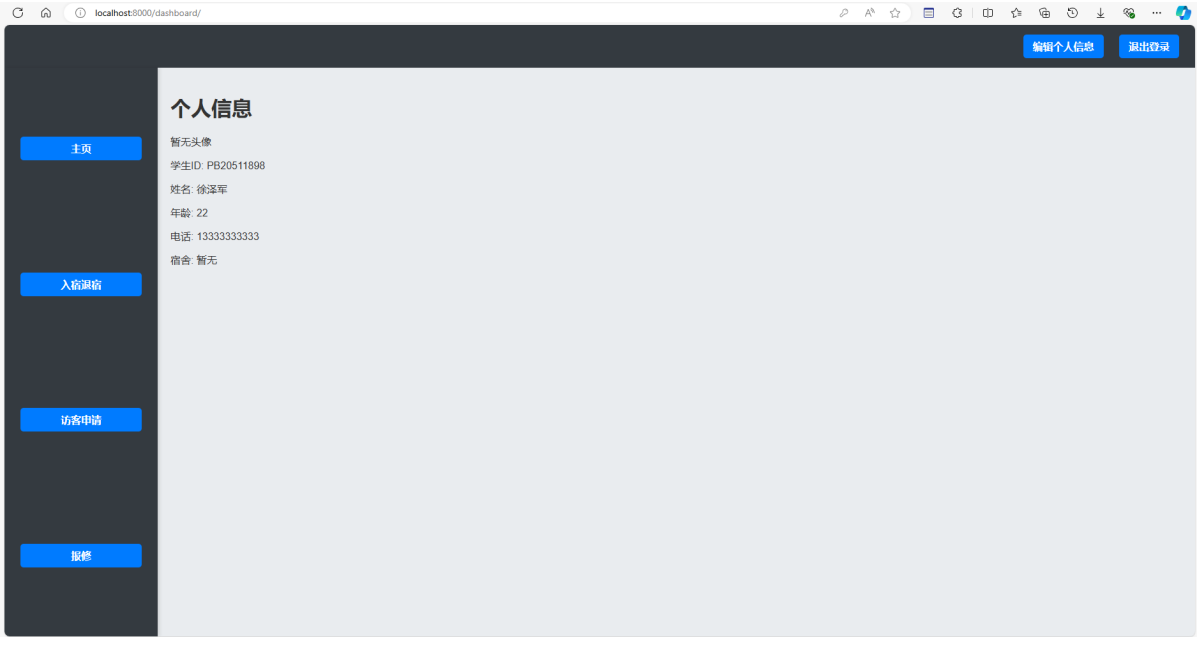
```
            return redirect('dashboard')
    else:
        messages.error(request, "用户未登录")
        return redirect('login')
```

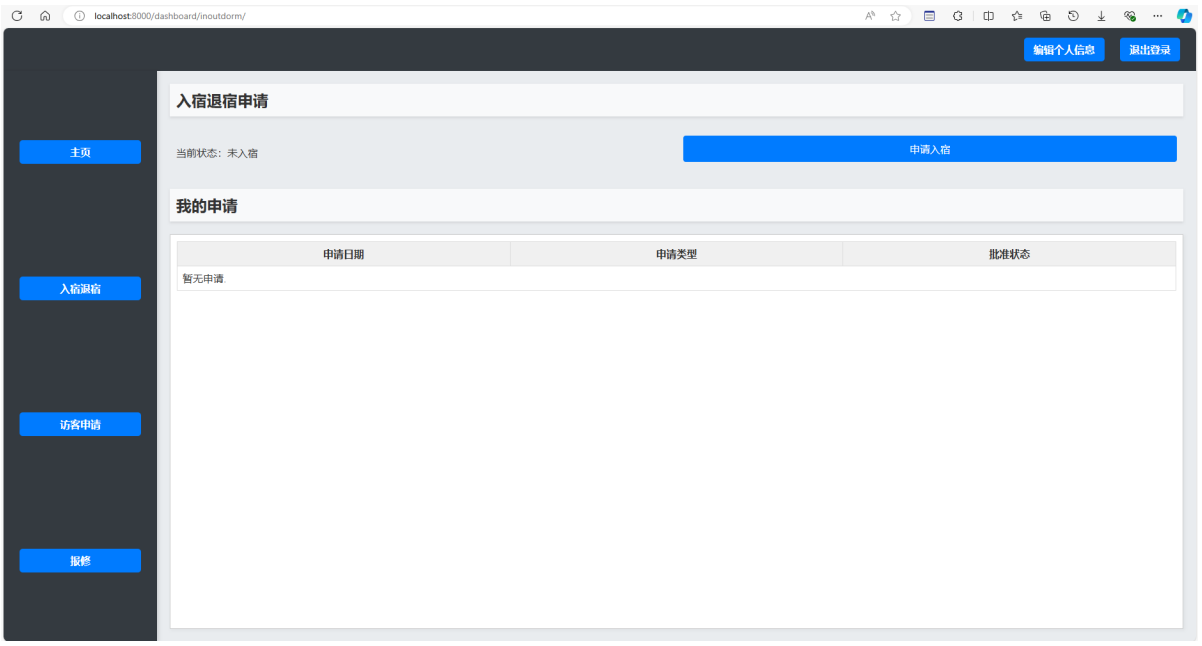
图片上传相关代码：

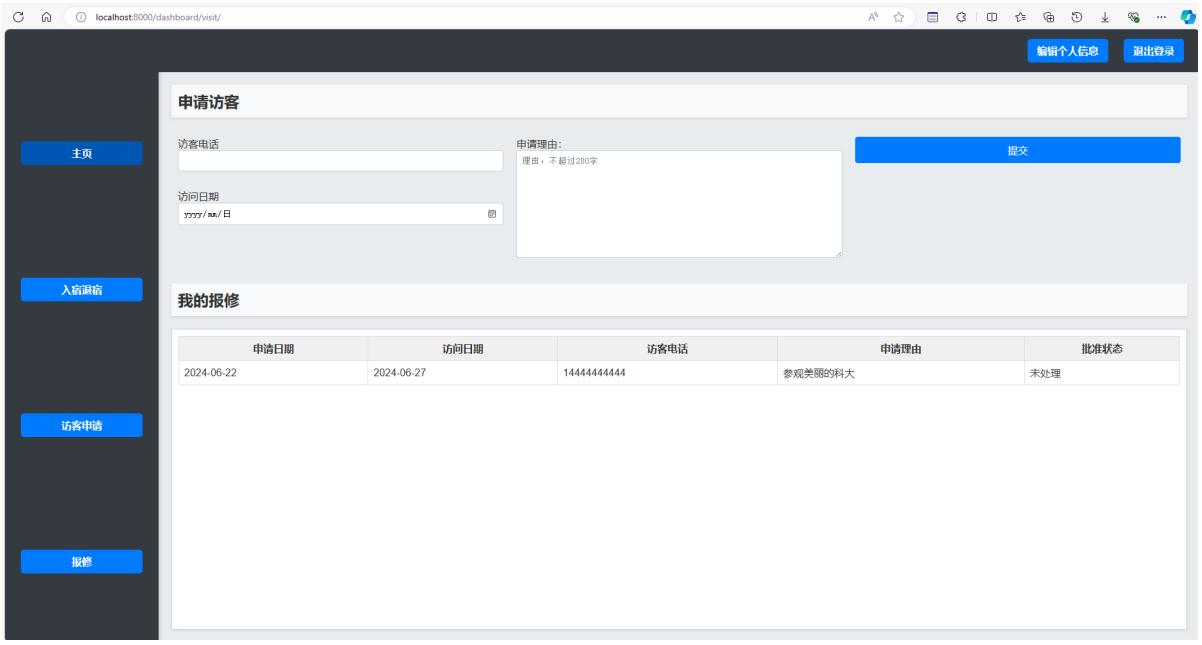
```
def image_upload(request):
    if request.user.is_authenticated:
        if request.user.groups.filter(name='Student').exists():
            if request.method == 'POST':
                form = ImageForm(request.POST, request.FILES)
                if form.is_valid():
                    new_img = form.save(commit=False)
                    new_img.uploader =
Student.objects.get(pk=request.user.username)
                    new_img.save()
                    messages.success(request, "上传图片成功")
                    return redirect('editprofile')
            else:
                form = ImageForm()
            return render(request, "management/image_upload.html", {'form':
form})
        else:
            messages.error(request, "用户身份错误")
            return redirect('dashboard')
    else:
        messages.error(request, "用户未登录")
        return redirect('login')
```

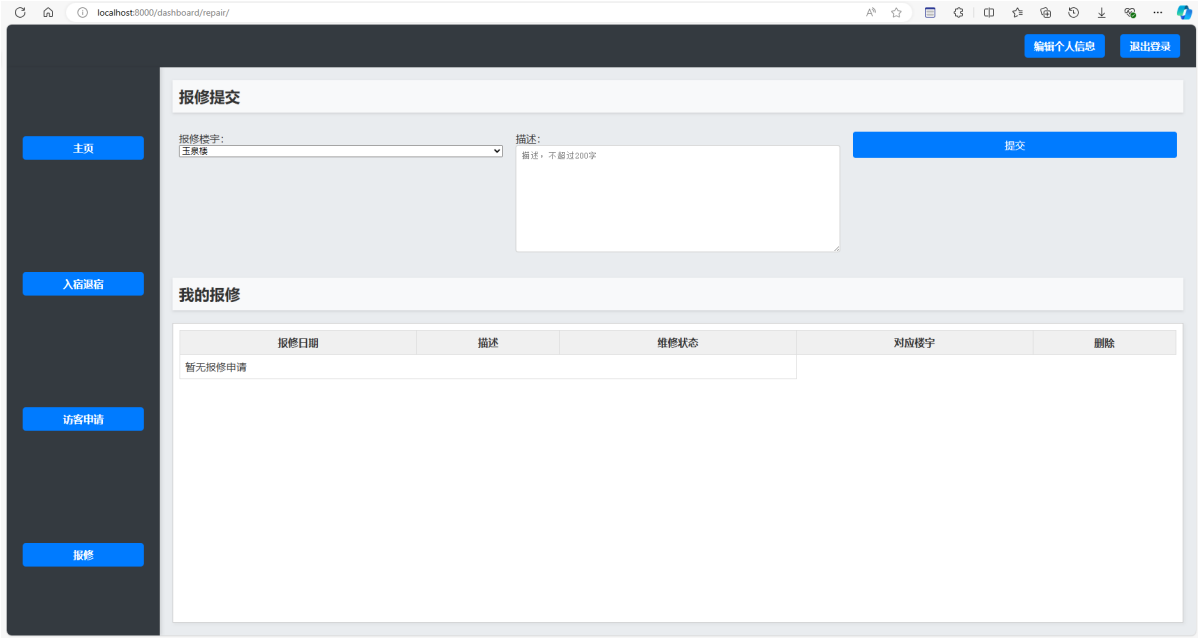
更多详细代码请访问公共仓库[ZeQueen-X/DataBase2024(github.com)]

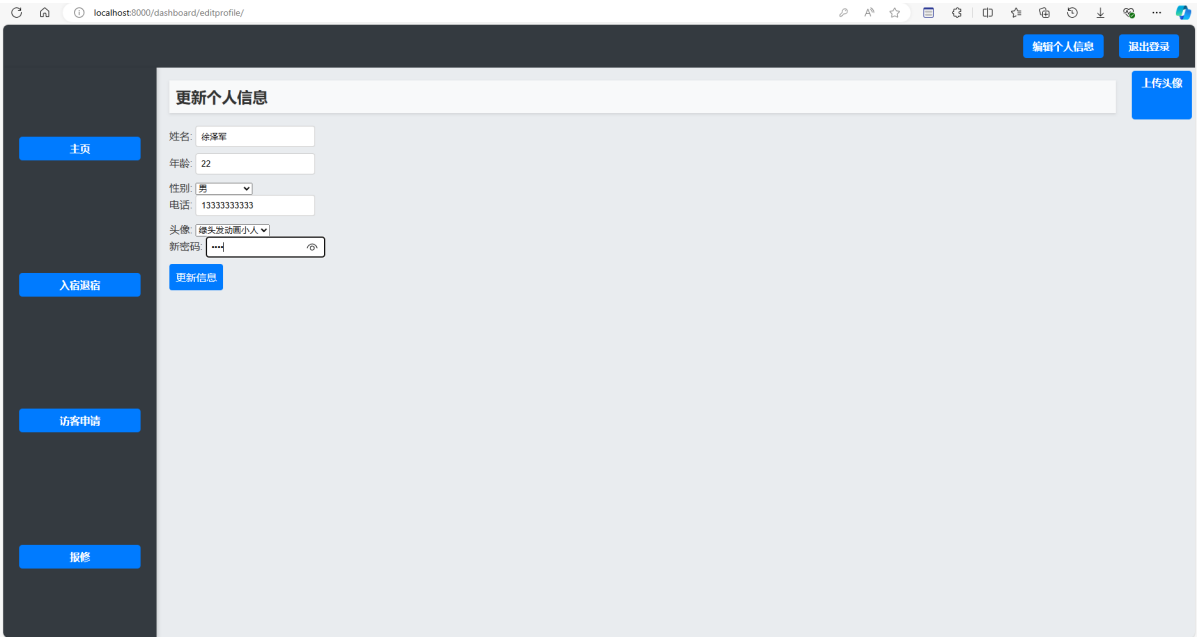
2.2.3 页面展示




localhost:8000/dashboard/inoutdorm/
编辑个人信息
退出登录
主页
入宿退宿
访客申请
报修
入宿退宿申请
当前状态：未入宿
申请入宿
我的申请
申请日期
申请类型
批准状态
暂无申请.


localhost:8000/dashboard/visit/
编辑个人信息
退出登录
主页
入宿退宿
访客申请
报修
申请访客
访客电话
申请理由：
理由，不超过200字
提交
访问日期
yyyy/mm/日
我的报修
申请日期
访问日期
访客电话
申请理由
批准状态
2024-06-22
2024-06-27
14444444444
参观美丽的科大
未处理


localhost:8000/dashboard/repair/
编辑个人信息
退出登录
主页
入宿退宿
访客申请
报修
报修提交
报修楼宇：
玉泉楼
描述：
描述，不超过200字
提交
我的报修
报修日期
描述
维修状态
对应楼宇
删除
暂无报修申请



2.3 宿管模块

2.3.1 概要

宿管登录后会展示个人信息，管理宿舍数量，管理学生数量。左侧导航栏提供功能按钮包括查看管理学生，处理入宿退宿申请，处理报修申请，处理访客申请。上方导航栏提供退出登录功能。

对于查看学生，将学生中属于该宿管管理楼宇中的宿舍的部分学生筛选出来展示，并提供按照学号、姓名、宿舍号等属性进行升序或降序排序。

对于入宿退宿，将`DormApply`表中`status`为0，即未处理的表项筛选出来展示，并提供同意和不同意处理手段，依据处理手段更新学生信息和请求信息，并依据触发器来实现非常规情况处理。

对于访客申请，查询所有访客申请，并通过页面中的标题实现多种排序方式，对未处理的申请提供批准和拒绝处理方式。在提交处理结果后更新申请状态。

对于报修申请，查询该宿管所管理楼宇的报修申请，并通过页面中的标题实现多种排序方式，当相关维修处理完成时宿管通过点击`已处理完成`按钮更新报修申请状态。

2.3.2 相关代码

查看学生部分相关代码：

```
        elif request.user.groups.filter(name='DormManager').exists():
            if request.method == 'POST':
                return redirect('student')
            else:
                sort_by = request.GET.get('sort', 'student_id')
                reverse = '-' if 'reverse' in request.GET else ''

                dm = DormManager.objects.get(pk=request.user.username)
                student_all_data = []

                if dm.building:
                    building = DormBuilding.objects.get(pk=dm.building.id)
                    dorm = Dorm.objects.filter(building=dm.building.id)
                    if (dorm.count() == 0):
                        stu_all = Student.objects.none()
                    else:
```

```
                        stu_all =
Student.objects.filter(dorm__building=building).order_by(reverse + sort_by)
                else:
                    stu_all = Student.objects.none()
                    dorm = Dorm.objects.none()

                for stu in stu_all:
                    student_all_data.append({
                        'stu_id': stu.student_id,
                        'stu_name': stu.name,
                        'stu_age': stu.age,
                        'stu_gender': stu.gender,
                        'stu_phone': stu.phone,
                        'stu_dorm': stu.dorm.Dorm_id,
                        'stu_status': stu.status,
                    })

                building = DormBuilding.objects.all()
                context = {'student_all_data': student_all_data,
                           'building': building,
                           'current_sort': reverse + sort_by,
                           'dorm': dorm}
                return render(request, 'management/student_dm.html', context)
```

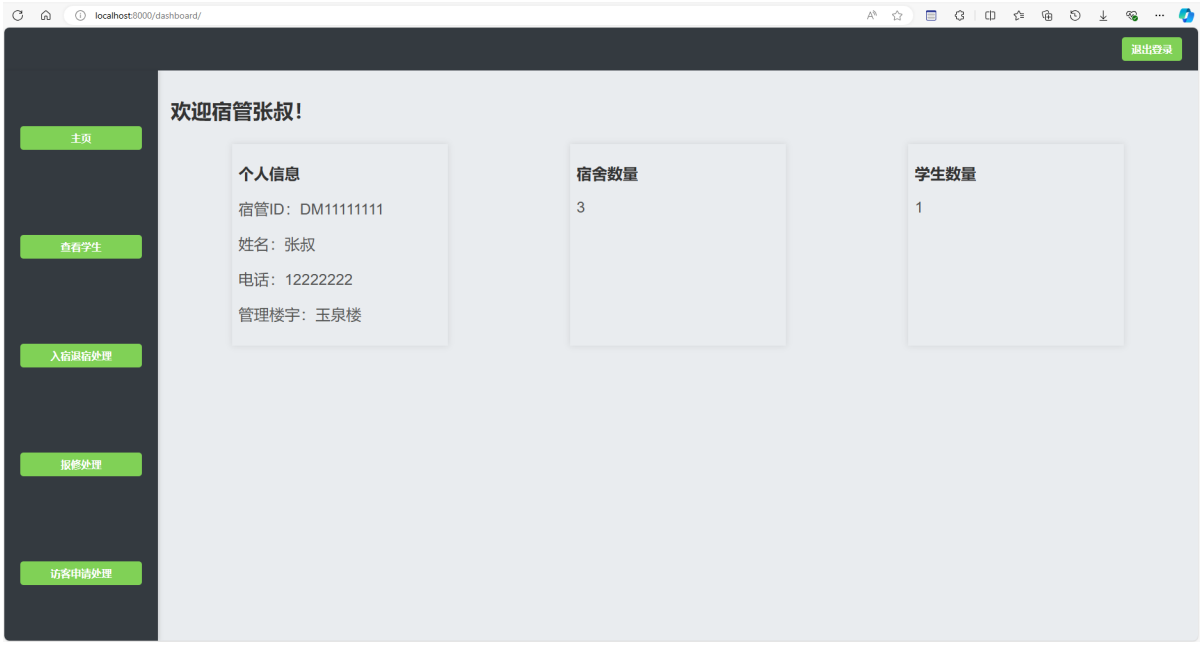
处理入宿退宿申请部分相关代码：

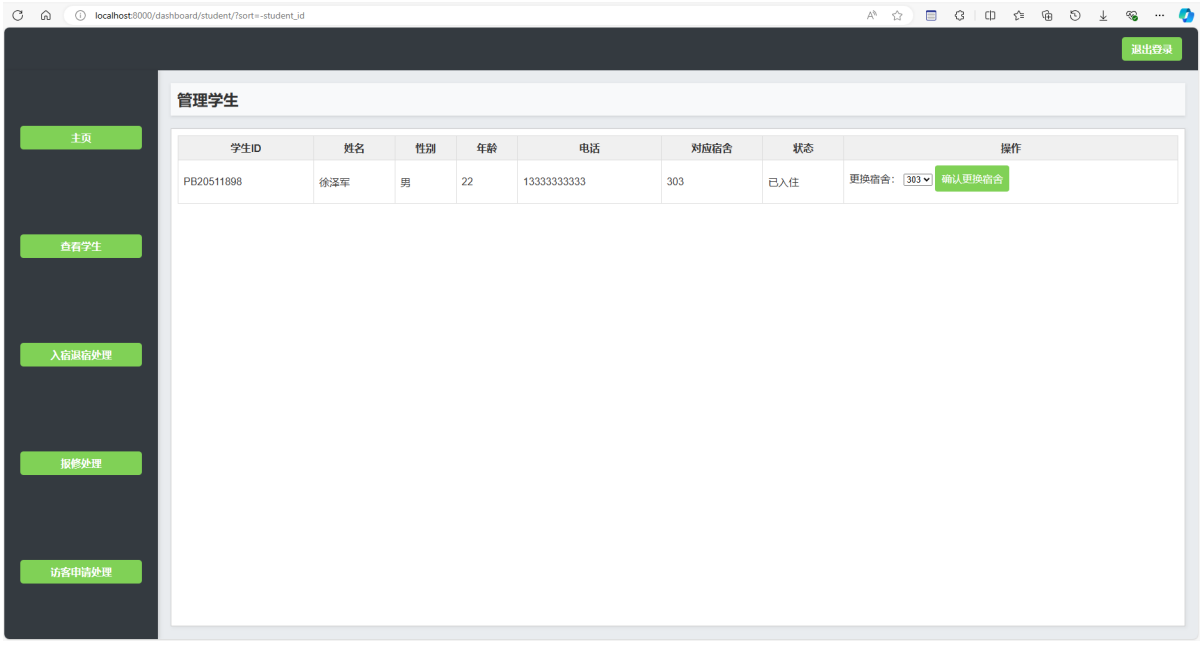
```
def apply_denied(request, apply_id):
    if request.user.is_authenticated:
        if request.user.groups.filter(name='DormManager').exists():
            apply = DormApply.objects.get(pk=apply_id)
            apply.status = 2
            apply.save()
            return redirect('inoutdorm')
        else:
            messages.error(request, "用户身份错误")
            return redirect('dashboard')
    else:
        messages.error(request, "用户未登录")
        return redirect('login')


def apply_in(request, applicant_id):
    if request.user.is_authenticated:
        if request.user.groups.filter(name='DormManager').exists():
            stu = Student.objects.get(pk=applicant_id)
            dorm = Dorm.objects.get(pk=request.POST.get('dorm'))
            stu.dorm = dorm
            apply = DormApply.objects.get(applicant=applicant_id, status=0)
            if apply:
                apply.status = 1
            else:
                messages.error(request, "不存在此类申请，请重试")
                return redirect('inoutdorm')
            apply.status = 1
            stu_dorm_count = Student.objects.filter(dorm=dorm).count()
```

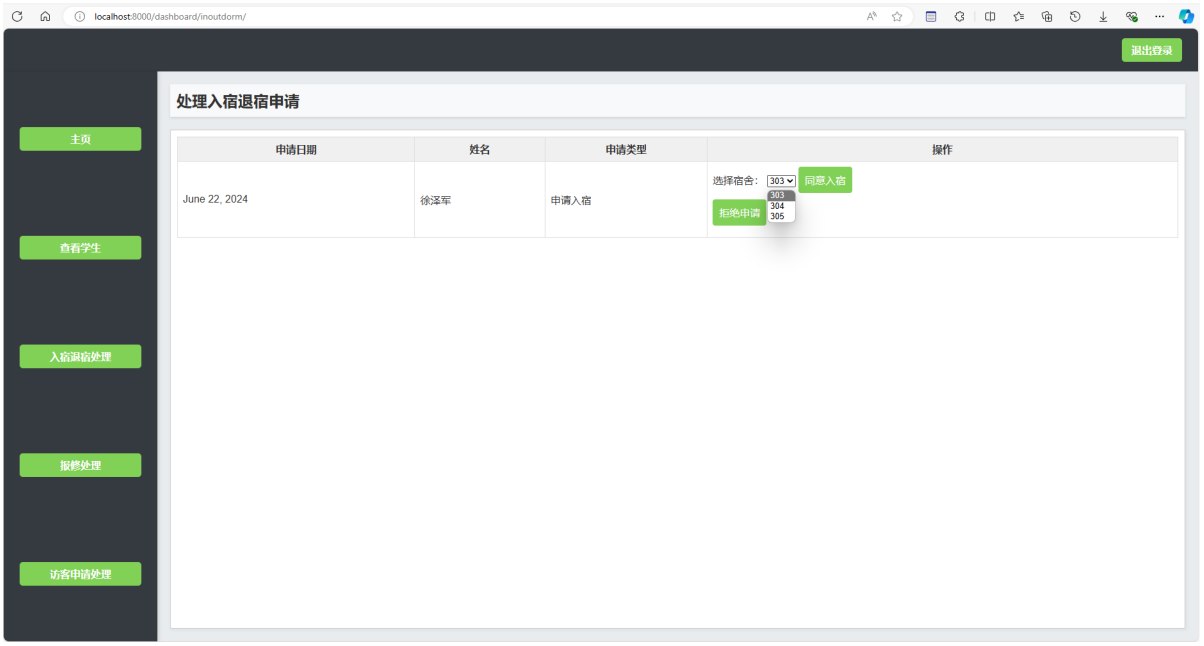
```
            if stu_dorm_count >= 4:
                messages.error(request, "目标宿舍已经有4名学生，不能继续添加")
                return redirect('inoutdorm')

            stu.save()
            apply.save()
            messages.success(request, "入宿成功")
            return redirect('inoutdorm')
        else:
            messages.error(request, "用户身份错误")
            return redirect('dashboard')
    else:
        messages.error(request, "用户未登录")
        return redirect('login')


def apply_out(request, applicant_id_out):
    if request.user.is_authenticated:
        if request.user.groups.filter(name='DormManager').exists():
            stu = Student.objects.get(pk=applicant_id_out)
            apply = DormApply.objects.get(applicant=applicant_id_out, status=0)
            if apply:
                apply.status = 1
            else:
                messages.error(request, "不存在此类申请，请重试")
                return redirect('inoutdorm')
            apply.save()
            messages.success(request, "退宿成功")
            return redirect('inoutdorm')
        else:
            messages.error(request, "用户身份错误")
            return redirect('dashboard')
    else:
        messages.error(request, "用户未登录")
        return redirect('login')
```

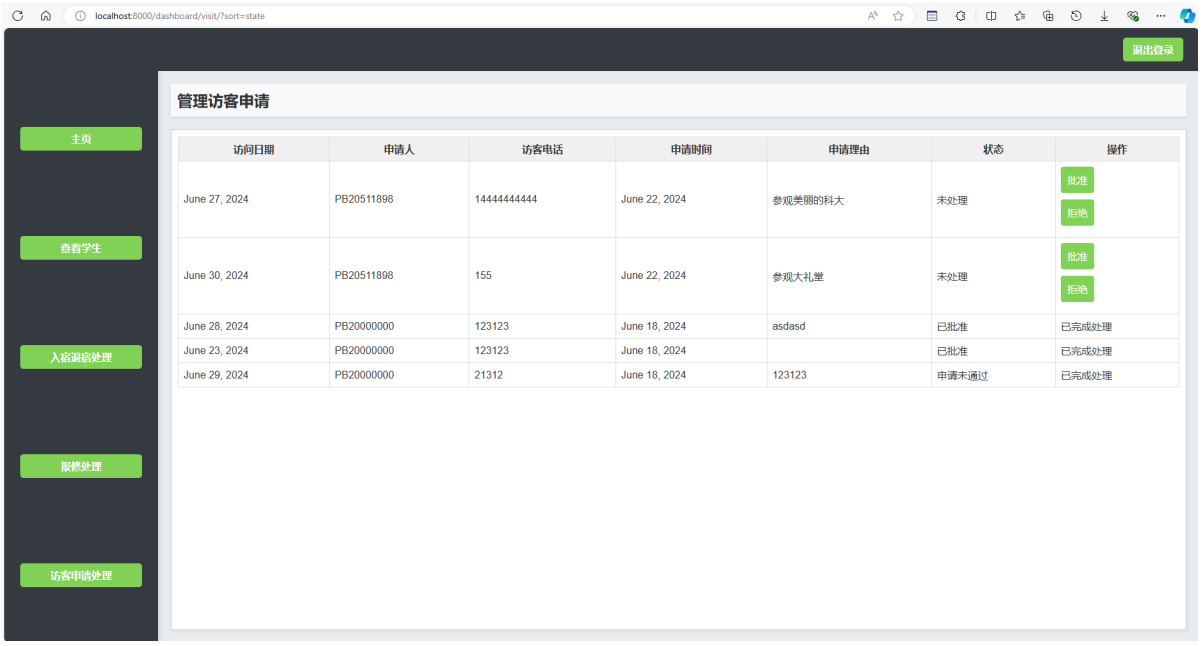
更多详细代码请访问公共仓库[ZeQueen-X/DataBase2024(github.com)]

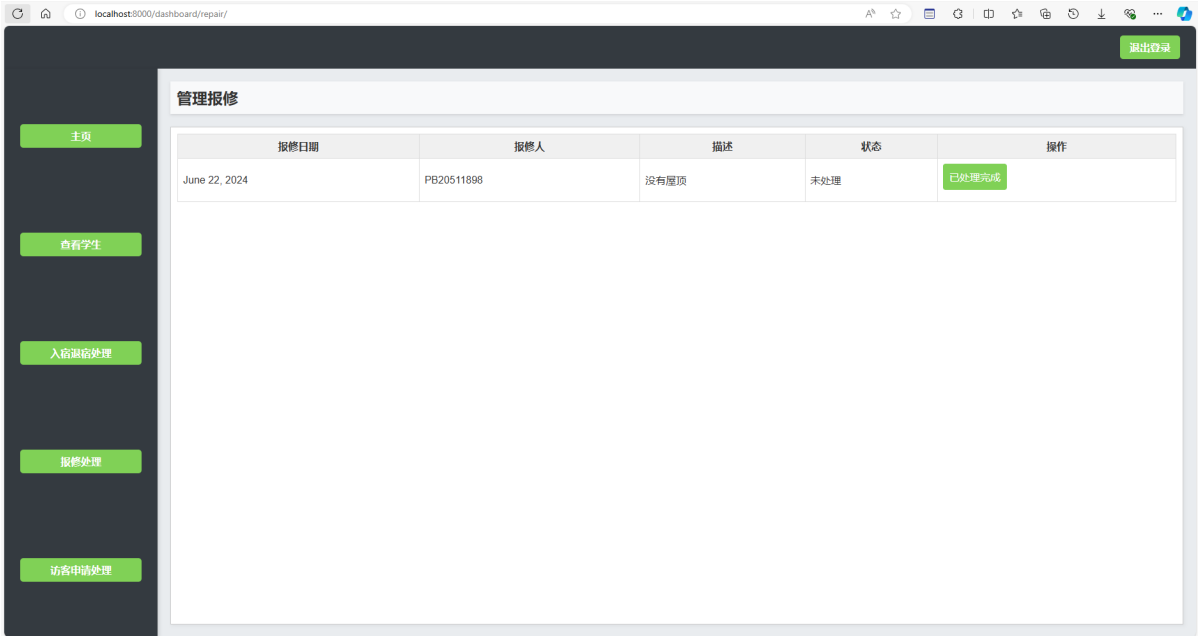
2.3.3 页面展示


localhost:8000/dashboard/
退出登录
主页
查看学生
入宿退宿处理
报修处理
访客申请处理
欢迎宿管张叔!
个人信息
宿管ID：DM11111111
姓名：张叔
电话：12222222
管理楼宇：玉泉楼
宿舍数量
3
学生数量
1


localhost:8000/dashboard/student/?sort=-student_id
退出登录
主页
查看学生
入宿退宿处理
报修处理
访客申请处理
管理学生
学生ID
姓名
性别
年龄
电话
对应宿舍
状态
操作
PB20511898
徐泽军
男
22
13333333333
303
已入住
更换宿舍：
303
确认更换宿舍


localhost:8000/dashboard/inoutdorm/
退出登录
主页
查看学生
入宿退宿处理
报修处理
访客申请处理
处理入宿退宿申请
申请日期
姓名
申请类型
操作
June 22, 2024
徐泽军
申请入宿
选择宿舍：
303
同意入宿
拒绝申请
303
304
305





2.4 管理员模块

2.4.1 概要

管理员需要通过Python脚本在服务端创建账号。管理员登录后展示相关数据面板，包括宿舍楼数量，宿舍数量，学生数量，宿管数量。左侧导航栏提供功能按钮包括宿管管理，楼宇管理和学生管理。上方导航栏提供退出登录功能。

对于宿管管理功能，分别提供新增宿管，编辑宿管和删除宿管功能。新增宿管时会创建相关的宿管账号，并将宿管信息保存到 `DormManager` 表中。

对于楼宇管理功能，可以分别新建楼宇和新建宿舍，提交表单后会在相应的数据库表中插入表项。同时提供管理宿舍楼和宿舍功能，左侧展示所有楼宇，右侧展示宿舍以及宿舍成员数量。点击左侧某个楼宇，则右侧宿舍更新为该楼宇所包含的宿舍。

对于学生管理功能，展示所有学生信息，并通过表头属性提供正向排序和反向排序功能。同时提供删除学生功能。

2.4.2 相关代码

宿管管理相关代码如下：

```
def dormmanager(request):
    if request.user.is_authenticated:
        if request.user.groups.filter(name='SuperUser').exists():
            if request.method == 'POST':
                dormmanager_id = request.POST.get('dormmanager_id')
                name = request.POST.get('name')
                age = request.POST.get('age')
                gender = request.POST.get('gender')
                phone = request.POST.get('phone')
                building_id = request.POST.get('building')

                # 如果不能获得楼宇数据则不填写宿管管理的楼宇
                try:
                    building = DormBuilding.objects.get(id=building_id)
                except DormBuilding.DoesNotExist:
                    building = None
                password = request.POST.get('password')

                if not re.match(r'DM\d{8}', dormmanager_id):
                    messages.error(request, '宿管ID格式不正确，必须以DM开头，后跟8位数
字。')
                    return redirect('dormmanager')

                if
DormManager.objects.filter(dormmanager_id=dormmanager_id).exists():
                    messages.error(request, '该宿管ID已存在，请添加新的宿管ID')
                    return redirect('dormmanager')

                with transaction.atomic():
                    try:
                        dm = DormManager(name=name, age=age, gender=gender,
phone=phone, dormmanager_id=dormmanager_id,
                                         building=building)
                        user = User.objects.create_user(username=dormmanager_id,
password=password)
                        user.groups.add(Group.objects.get(name='DormManager'))
                        user.save()
                        dm.save()
                        messages.success(request, "添加宿管成功！")
                    except Exception as e:
                        messages.success(request, e)
                        print(e)

                return redirect('dormmanager')
            else:
                # 默认按照ID排序，实现排序功能
                sort_by = request.GET.get('sort', 'dormmanager_id')
                reverse = '-' if 'reverse' in request.GET else ''

                dormmanagers_all = DormManager.objects.all().order_by(reverse +
sort_by)
                dormmanagers_all_data = []
```

```
                print(sort_by)
                for dm in dormmanagers_all:
                    building_name = dm.building.name if dm.building else "暂无管理
楼宇"
                    dormmanagers_all_data.append({
                        'dm_id': dm.dormmanager_id,
                        'dm_name': dm.name,
                        'dm_age': dm.age,
                        'dm_gender': dm.gender,
                        'dm_phone': dm.phone,
                        'dm_building_id': dm.building,
                        'dm_building': building_name,
                    })

                building = DormBuilding.objects.all()
                context = {'dormmanagers_all_data': dormmanagers_all_data,
                           'building': building,
                           'current_sort': reverse + sort_by}
                return render(request, 'management/dormmanager.html', context)
        else:
            messages.error(request, '您不是管理员用户，禁止访问')
            return redirect('dashboard')
    else:
        messages.error(request, '用户未登录！')
        return redirect('login')
```

楼宇和宿舍管理相关代码：

```
def dormbuilding(request):
    if request.user.is_authenticated:
        if request.user.groups.filter(name='SuperUser').exists():
            if request.method == 'POST':
                bd_name = request.POST.get('bd_name')

                if DormBuilding.objects.filter(name=bd_name).exists():
                    messages.error(request, '该楼宇已经存在，请不要重复添加')
                    return redirect('dormbuilding')

                with transaction.atomic():
                    try:
                        bd = DormBuilding(name=bd_name)
                        bd.save()
                        messages.success(request, "添加楼宇成功！")
                    except Exception as e:
                        messages.error(request, e)
                        print(e)
                return redirect('dormbuilding')
            else:
                bds = DormBuilding.objects.all()
                if bds.exists():
                    default_filter = bds.first().id
                else:
                    default_filter = None
                bds_data = []
                filter_by = request.GET.get('filter', default_filter)
```

```
                dorms = Dorm.objects.filter(building=filter_by)
                print(filter_by)
                dorms_data = []
                for dorm in dorms:
                    dorms_data.append({
                        'dorm_id': dorm.pk,
                        'dorm_bd': DormBuilding.objects.get(pk=filter_by).name,
                        'count':
Student.objects.filter(dorm=dorm.Dorm_id).count(),
                    })
                for bd in bds:
                    bds_data.append({
                        'bd_id': bd.pk,
                        'bd_name': bd.name,
                        'dm_count': Dorm.objects.filter(building=bd.id).count(),
                        # 'stu_count':Student.objects.filter(dorm = ).count(),
                    })
                context = {'bds_data': bds_data, 'dorms_data': dorms_data}
                return render(request, 'management/dormbuilding_su.html',
context)
        elif request.user.groups.filter(name='DormManager').exists():
            if request.method != 'Post':
                dormmanager = DormManager.objects.get(pk=request.user.username)
                bd = DormBuilding.objects.get(Dorm_id=dormmanager.building)
            else:
                messages.error(request, '您不是管理用户，禁止访问')
                return redirect('dashboard')
    else:
        messages.error(request, '用户未登录！')
        return redirect('login')

def dorm(request):
    if request.user.is_authenticated:
        if request.user.groups.filter(name='SuperUser').exists():
            if request.method == 'POST':
                dorm_id = request.POST.get('dorm_id')
                building =
DormBuilding.objects.get(name=request.POST.get('building'))
                dorm = Dorm(building=building, Dorm_id=dorm_id)

                if Dorm.objects.filter(Dorm_id=dorm_id).exists():
                    messages.error(request, '该宿舍已经存在，请不要重复添加')
                    return redirect('dormbuilding')
                with transaction.atomic():
                    try:
                        dorm.save()
                        messages.success(request, "添加宿舍成功！")
                    except Exception as e:
                        messages.error(request, e)
                        print(e)
                return redirect('dormbuilding')
            else:
                bds = DormBuilding.objects.all()
                if bds.exists():
                    default_filter = bds.first().id
                else:
```

```
                    default_filter = None
                bds_data = []
                filter_by = request.GET.get('filter', default_filter)
                dorms = Dorm.objects.filter(building=filter_by)
                dorms_data = []
                for dorm in dorms:
                    dorms_data.append({
                        'dorm_id': dorm.pk,
                        'dorm_bd': DormBuilding.objects.get(pk=filter_by).name,
                        'count':
Student.objects.filter(dorm=dorm.Dorm_id).count(),
                    })
                for bd in bds:
                    bds_data.append({
                        'bd_id': bd.pk,
                        'bd_name': bd.name,
                        'dm_count': Dorm.objects.filter(building=bd.id).count(),
                        # 'stu_count':Student.objects.filter(dorm = ).count(),
                    })
                context = {'bds_data': bds_data, 'dorms_data': dorms_data}
                return render(request, 'management/dormbuilding_su.html',
context)
        elif request.user.groups.filter(name='DormManager').exists():
            if request.method != 'Post':
                dormmanager = DormManager.objects.get(pk=request.user.username)
                bd = DormBuilding.objects.get(Dorm_id=dormmanager.building)
        else:
            messages.error(request, '您不是管理用户，禁止访问')
            return redirect('dashboard')
    else:
        messages.error(request, '用户未登录！')
        return redirect('login')
```

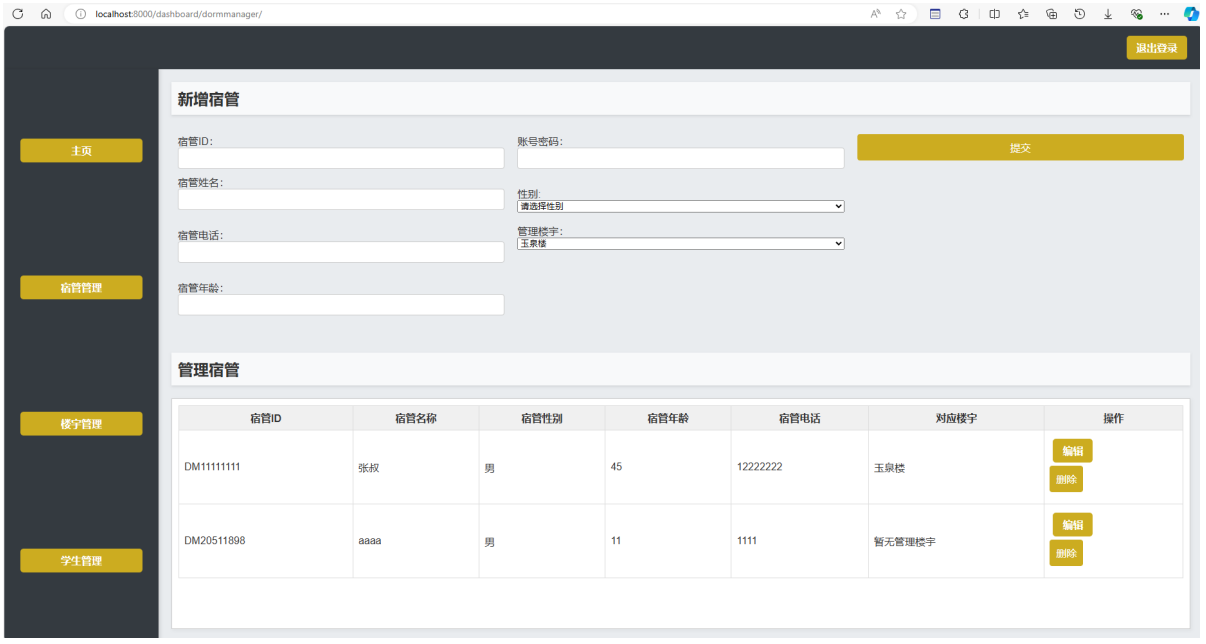
学生管理部分相关代码：

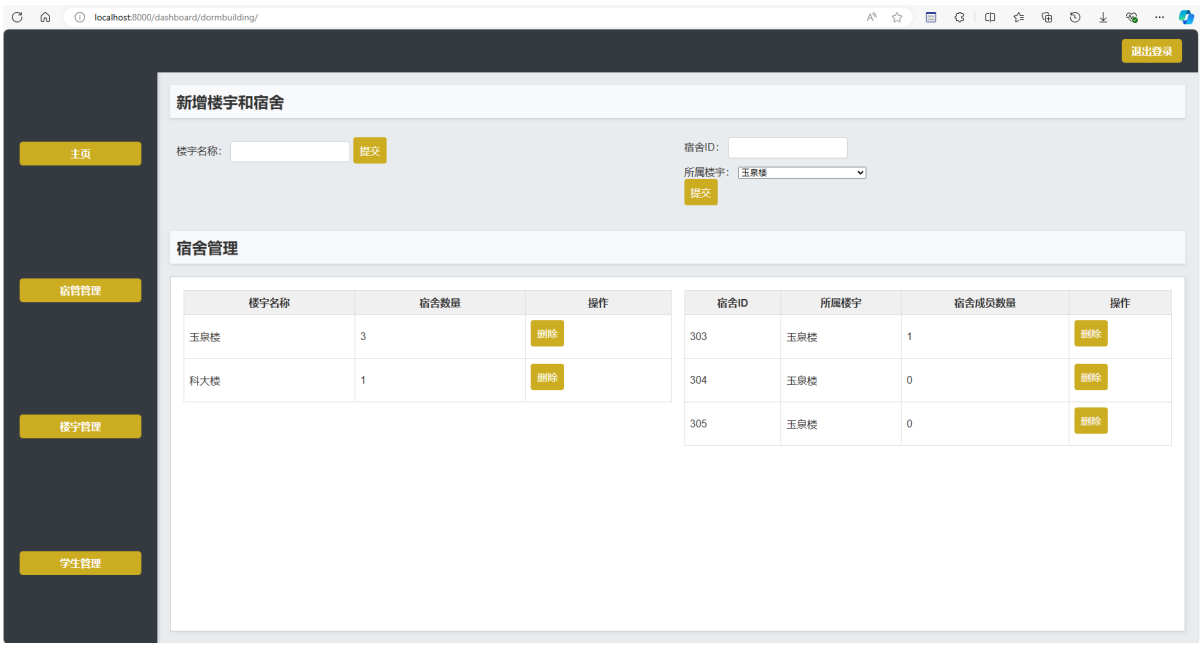
```
    if request.user.groups.filter(name='SuperUser').exists():
            if request.method == 'POST':
                return redirect('student')
            else:
                sort_by = request.GET.get('sort', 'student_id')
                reverse = '-' if 'reverse' in request.GET else ''

                student_all = Student.objects.all().order_by(reverse + sort_by)
                student_all_data = []
                print(sort_by)
                for stu in student_all:
                    student_all_data.append({
                        'stu_id': stu.student_id,
                        'stu_name': stu.name,
                        'stu_age': stu.age,
                        'stu_gender': stu.gender,
                        'stu_phone': stu.phone,
                        'stu_dorm': stu.dorm,
                        'stu_status': stu.status,
```

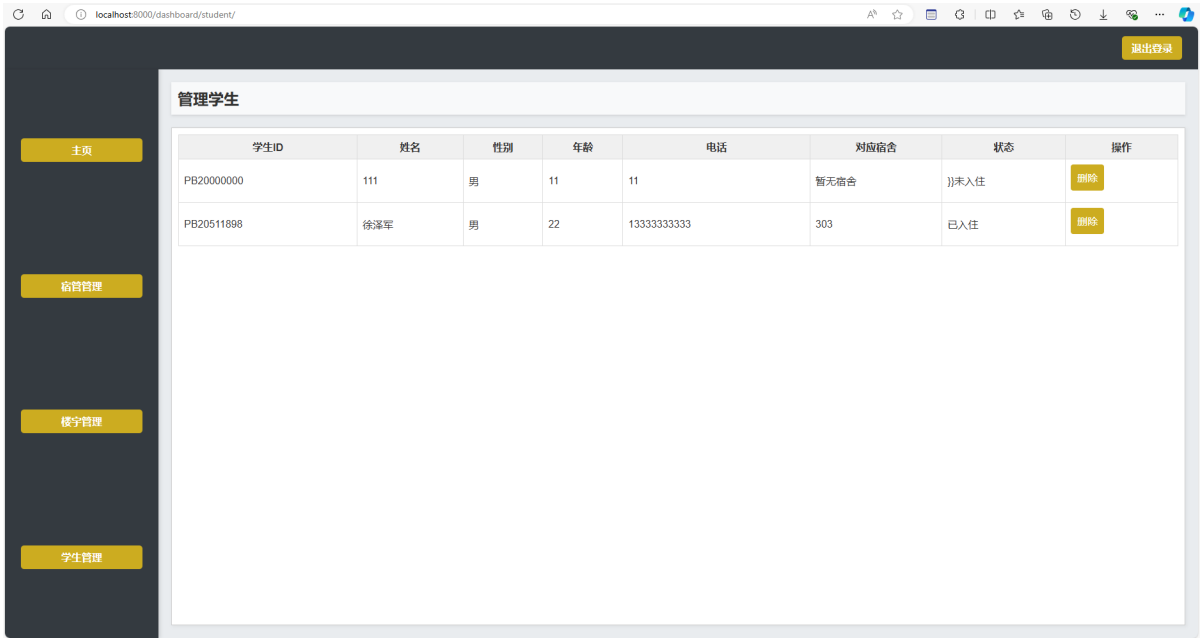
```
        })

    building = DormBuilding.objects.all()
    context = {'student_all_data': student_all_data,
               'building': building,
               'current_sort': reverse + sort_by}
    return render(request, 'management/student_su.html', context)
```

更多详细代码请访问公共仓库[ZeQueen-X/DataBase2024(github.com)]

2.4.3 页面展示







总结与讨论

通过本次课程设计，我收获颇丰。从需求设计，逐步设计模块功能，再到分类分模块实现，这次课程设计让我学到了很多Web开发设计、前后端开发的相关知识，更提高了我使用数据库的能力。

本课程设计是一个相对简单但也具有完整功能的宿舍管理系统，在前端的设计也较为美观。功能使用简洁易懂。由于涉及多用户，很多联动操作用图片展示起来较为繁琐，但所有功能在检查实验当天都展示给了负责检查的助教，得到了助教的认可。

当然，项目仍有不足之处：没有采用版本控制开发；项目开始前的需求分析不够详细导致开始时经常更改；没有设计脚本化调试；开发规范和设计规范还有待提高...等等。但是经过这次课程设计，我详细这些不足之处在未来的软件开发阶段肯定会有所改善！

总之，感谢助教和老师，这次课程我收获颇丰！