# 功能三的设计文档——数据库设计和部署报告

## 1. 功能三作用

功能三在功能一和功能二的基础上对数据库进行了设计和部署，具体实现了权限管理和用户业务对应的数据库操作。创建了数据库表，对用户表和药品表的字段进行了设计。使用Django ORM进行数据库的增删改查操作，定义了两个Django模型：User和Drug，实现了对用户和药品的创建和管理并写入数据库。通过配置文件的方式完成了数据库的连接和交互。

### 2.1 用户角色和权限

* 普通用户：能够注册账号、登录、登出以及访问主页。
* 管理员：除了普通用户的功能外，还能进行药品信息的管理，以及用户账号的管理。

### 2.2 功能需求

* 支持现代浏览器（如Chrome、Firefox、Safari）。
* 支持跨平台操作（Windows、macOS、Linux）。
* 使用SQLite数据库进行开发。

## 3. 系统设计

### 3.1 架构设计

* 前端：使用HTML、CSS和JavaScript构建用户界面。
* 后端：使用Django框架进行后端逻辑处理。
* 数据库：使用SQLite数据库存储用户信息和药品信息。

### 3.2 数据库设计

用户数据存储在`User`模型中，包含以下字段：

* email：用户邮箱，作为唯一标识。
* is\_active：用户账号是否激活。
* is\_admin：用户是否为管理员。

药品数据存储在’Drug’模型中，包含以下字段：

* drug\_name字段：药品名称。
* drug\_company字段：药品公司。
* drug\_quantity字段：药品数量。
* drug\_code字段：药品代码。

### 3.3 接口设计

* 注册接口：`POST /register`，接收邮箱和密码，创建新用户。
* 登录接口：`POST /login`，接收邮箱和密码，验证用户身份。
* 登出接口：`POST /logout`，清除用户会话，实现登出。

### 3.4 安全性设计

* 使用Django的`authenticate`和`login`方法进行用户认证。
* 密码加密存储，使用Django的密码哈希功能。
* 使用会话管理用户登录状态。

## 4. 功能实现

### 4.1 前端实现

* 使用HTML表单收集用户输入。
* 使用CSS进行样式设计，提供友好的用户界面。
* 使用JavaScript处理表单提交和用户交互。

### 4.2 后端实现

* 注册：`register`函数处理用户注册，使用`User.objects.create\_user`创建用户。
* 登录：`user\_login`函数处理用户登录，使用`authenticate`和`login`进行身份验证。
* 登出：`user\_logout`函数处理用户登出，清除会话信息。

### 4.3 数据库交互

使用Django ORM进行数据库操作，包括用户创建、验证和查询。

## 5. 测试计划

* 单元测试：对注册、登录、登出功能进行单元测试。
* 集成测试：测试前端和后端的集成，确保数据正确流转。
* 性能测试：测试系统在高并发情况下的表现。

## 6. 部署和维护

* 使用Docker容器化部署，确保环境一致性。
* 定期更新依赖库，修复安全漏洞。

## 7. 用户文档和帮助

* 提供用户手册，说明如何注册、登录和登出。
* 提供在线帮助和FAQ，解决用户常见问题。

## 8. 风险评估

* 安全风险：通过加密和安全认证机制降低风险。
* 性能风险：通过性能测试和优化确保系统稳定运行。

## 9. 结语

感谢老师百忙之中阅读本报告。我们小组使用Python和Django框架实现了一个具有用户验证登录和药物管理功能的Web应用程序。我们希望这个系统能够满足教学和实践的需求。

## 10. 附录

* 代码片段：包括权限分配、管理员功能设计的代码实现。
* 配置文件：Django项目的配置文件。
* 架构图：系统的架构图，展示各组件之间的关系。

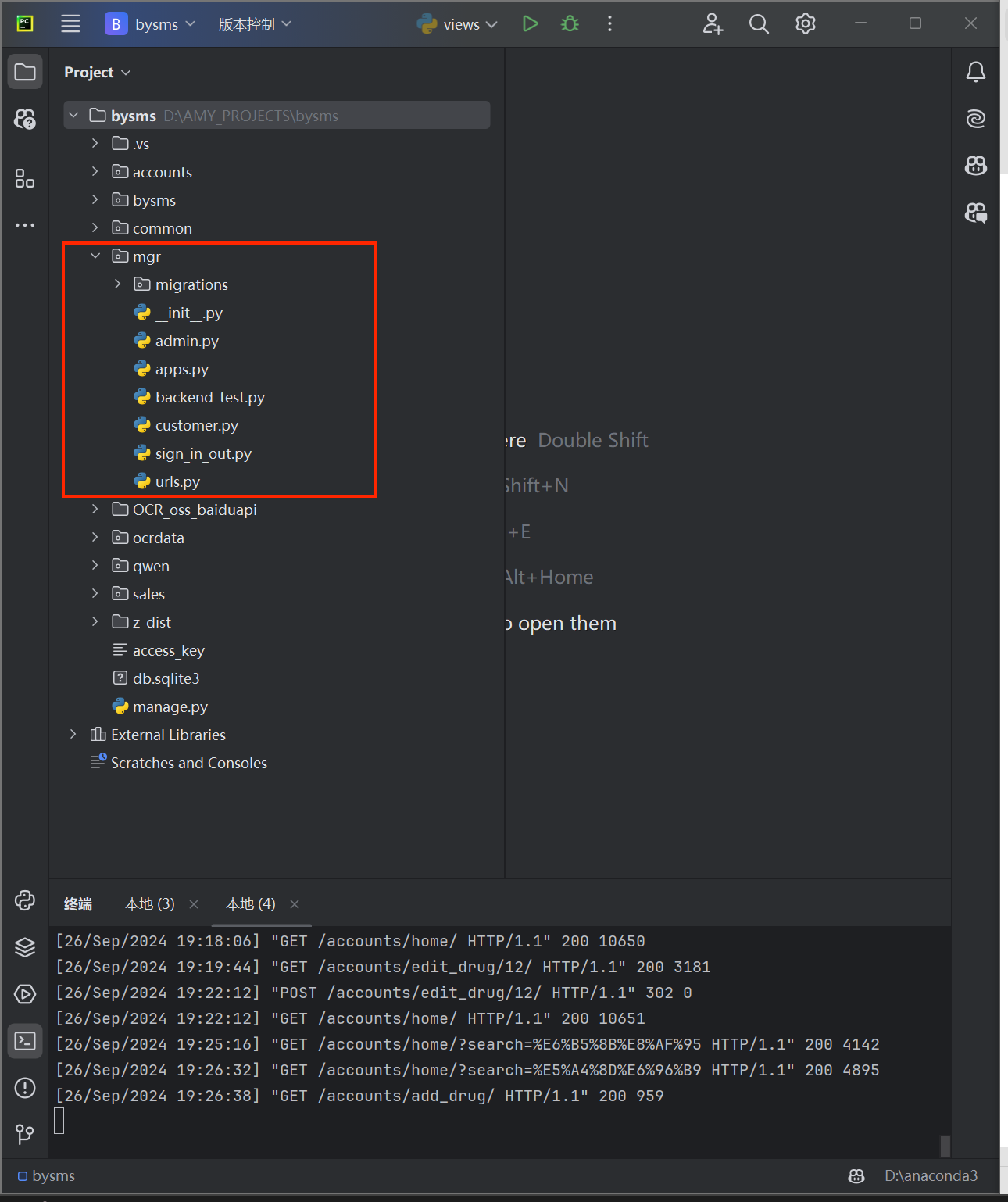
## 11. 参考文献

* Django官方文档
* HTML和CSS参考手册

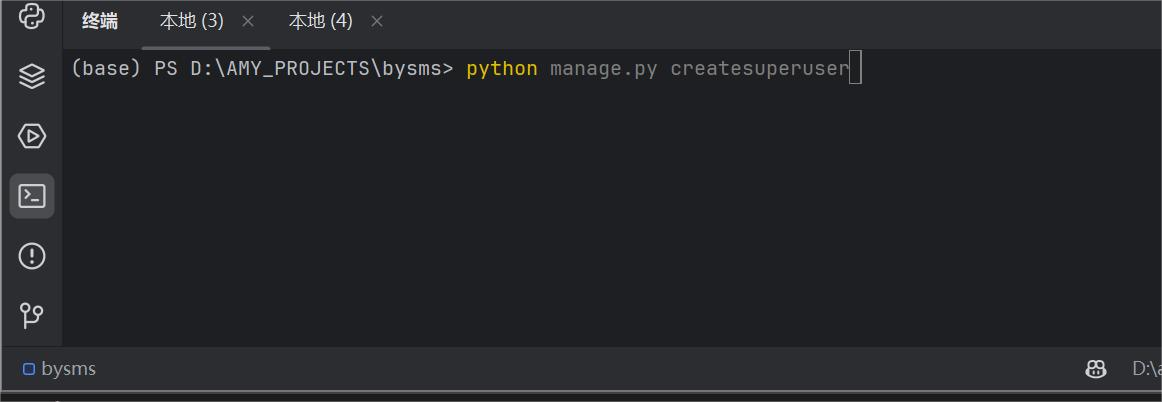
## 附录

### A.源码

#### 1.为超级管理员创建的应用



如果想要创建超级管理员，那么请在终端输入如下代码：  
python manage.py createsuperuser



访问超级管理员页面请在浏览器输入：localhost/admin/  
初始账号密码为：

账号：agacila  
密码：88888888（8个8）

#### 2.功能二：权限分配

|  |
| --- |
| class UserManager(BaseUserManager):      def create\_user(self, email, password=None):          if not email:              raise ValueError('Users must have an email address')          user = self.model(email=self.normalize\_email(email))          user.set\_password(password)          user.save(using=self.\_db)          return user      def create\_superuser(self, email, password=None):          user = self.create\_user(email, password)          user.is\_admin = True          user.save(using=self.\_db)          return user |

#### 3.功能二：管理员查询、修改、添加、删除功能

|  |
| --- |
| '''  这个程序是用来分发请求post/get/put  同时处理查询/修改/添加/删除数据库中的表的  '''  from django.http import JsonResponse  import json  from common.models import Customer  #定义一个分发函数。因为django不支持从url参数直接读取方法类型  def dispatcher(request):      # 将请求参数统一放入request 的 params 属性中，方便后续处理      print('到达dispatch处')      # GET请求 参数在url中，同过request 对象的 GET属性获取      if request.method == 'GET':#request.method是一个属性，它表示客户端发送请求所使用的 HTTP 方法          request.params = request.GET#request.GET包含了通过get方法提交的请求参数      # POST/PUT/DELETE 请求 参数 从 request 对象的 body 属性中获取      elif request.method in ['POST','PUT','DELETE']:          # 根据接口，POST/PUT/DELETE 请求的消息体都是 json格式          request.params = json.loads(request.body)      # 根据不同的action分派给不同的函数进行处理      action = request.params['action']#从名为request.params的对象中获取键为action的值      if action == 'list\_customer':          return listcustomers(request)      elif action == 'add\_customer':          return addcustomer(request)      elif action == 'modify\_customer':          return modifycustomer(request)      elif action == 'del\_customer':          return deletecustomer(request)      else:          return JsonResponse({'ret': 1, 'msg': '不支持该类型http请求'})  def listcustomers(request):#列出客户      # 返回一个 QuerySet 对象 ，包含所有的表记录      qs = Customer.objects.values()      # 将 QuerySet 对象 转化为 list 类型      # 否则不能 被 转化为 JSON 字符串      retlist = list(qs)      return JsonResponse({'ret': 0, 'retlist': retlist})  def addcustomer(request):#添加客户      info    = request.params['data']      # 从请求消息中 获取要添加客户的信息      # 并且插入到数据库中      # 返回值 就是对应插入记录的对象      record = Customer.objects.create(name=info['name'] ,                              phonenumber=info['phonenumber'] ,                              address=info['address'])      return JsonResponse({'ret': 0, 'id':record.id})  def modifycustomer(request):#修改客户      # 从请求消息中 获取修改客户的信息      # 找到该客户，并且进行修改操作      customerid = request.params['id']      newdata    = request.params['newdata']      try:          # 根据 id 从数据库中找到相应的客户记录          customer = Customer.objects.get(id=customerid)#get方法是查询数据的一种方法      except Customer.DoesNotExist:          return  {                  'ret': 1,                  'msg': f'id 为`{customerid}`的客户不存在'          }      if 'name' in  newdata:          customer.name = newdata['name']      if 'phonenumber' in  newdata:          customer.phonenumber = newdata['phonenumber']      if 'address' in  newdata:          customer.address = newdata['address']      # 注意，一定要执行save才能将修改信息保存到数据库      customer.save()      return JsonResponse({'ret': 0})  def deletecustomer(request):#删除客户      customerid = request.params['id']      try:          # 根据 id 从数据库中找到相应的客户记录          customer = Customer.objects.get(id=customerid)      except Customer.DoesNotExist:          return  {                  'ret': 1,                  'msg': f'id 为`{customerid}`的客户不存在'          }      # delete 方法就将该记录从数据库中删除了      customer.delete()      return JsonResponse({'ret': 0}) |

### 4.功能二：管理员登入登出处理

|  |
| --- |
| # -\*- coding: utf-8 -\*-  '''  管理员用户的登入登出处理  '''  from django.http import JsonResponse  from django.contrib.auth import authenticate, login, logout  # 登录处理  def signin( request):      # 从 HTTP POST 请求中获取用户名、密码参数      userName = request.POST.get('username')      passWord = request.POST.get('password')      # 使用 Django auth 库里面的 方法校验用户名、密码      # authenticate 方法会根据 Django 的配置，从默认的数据库中查找用户信息。具体来说，它会查找 django.contrib.auth 应用中定义的用户表，通常是 auth\_user 表      user = authenticate(username=userName, password=passWord)      # 如果能找到用户，并且密码正确      if user is not None:          if user.is\_active:              if user.is\_superuser:                  login(request, user)                  # 在session中存入用户类型                  request.session['usertype'] = 'mgr'                  return JsonResponse({'ret': 0})              else:                  return JsonResponse({'ret': 1, 'msg': '请使用管理员账户登录'})          else:              return JsonResponse({'ret': 0, 'msg': '用户已经被禁用'})      # 否则就是用户名、密码有误      else:          return JsonResponse({'ret': 1, 'msg': '用户名或者密码错误'})  # 登出处理  def signout( request):      # 使用登出方法      logout(request)#logout去处理request请求中包含的参数      return JsonResponse({'ret': 0}) |

### 5.功能三：用户和药品的Django模型

class User(AbstractBaseUser):

    email = models.EmailField(unique=True)

    is\_active = models.BooleanField(default=True)

    is\_admin = models.BooleanField(default=False)

    USERNAME\_FIELD = 'email'

    REQUIRED\_FIELDS = []

    objects = UserManager()

    def \_\_str\_\_(self):

        return self.email

    @property

    def is\_staff(self):

        return self.is\_admin

class Drug(models.Model):

    drug\_name = models.CharField(max\_length=255)

    drug\_company = models.CharField(max\_length=255)

    drug\_quantity = models.IntegerField()

    drug\_code = models.CharField(max\_length=255)

**User类：**

User类继承自AbstractBaseUser，表示自定义用户模型。

email字段：唯一的电子邮件地址。

is\_active字段：表示用户是否活跃（软删除）。

is\_admin字段：表示用户是否是管理员。

USERNAME\_FIELD：指定用于登录的字段，这里是email。

REQUIRED\_FIELDS：创建超级用户时需要的额外字段，这里为空。

objects：指定自定义的用户管理器UserManager。

\_\_str\_\_方法：返回用户的电子邮件地址。

is\_staff方法：返回用户是否是管理员。

**Drug类：**

Drug类表示药品信息。

drug\_name字段：药品名称。

drug\_company字段：药品公司。

drug\_quantity字段：药品数量。

drug\_code字段：药品代码。

### 6.功能三：药品的添加、修改、删除

*# 添加药品*

@login\_required

def add\_drug(request):

*#登录验证*

    if not request.session.get('is\_logged\_in'):

        return redirect('login')

    if request.method == 'POST':

        name = request.POST.get('name')

        company = request.POST.get('company')

        quantity = request.POST.get('quantity')

        code = request.POST.get('code')

        Drug.objects.create(drug\_name=name, drug\_company=company, drug\_quantity=quantity, drug\_code=code)

        messages.success(request, '药品添加成功！')

        return redirect('home')

    return render(request, 'accounts/add\_drug.html')

*# 编辑药品*

@login\_required

def edit\_drug(request, drug\_id):

*#登录验证*

    if not request.session.get('is\_logged\_in'):

        return redirect('login')

    drug = get\_object\_or\_404(Drug, pk=drug\_id)  *# 使用 pk 参数来获取主键*

    if request.method == 'POST':

*# 从表单获取数据*

        drug\_name = request.POST.get('drug\_name')

        drug\_company = request.POST.get('drug\_company')

        drug\_quantity = request.POST.get('drug\_quantity')

        drug\_code = request.POST.get('drug\_code')

*# 更新药品信息*

        drug.drug\_name = drug\_name

        drug.drug\_company = drug\_company

        drug.drug\_quantity = drug\_quantity

        drug.drug\_code = drug\_code

        drug.save()

*# 向用户显示成功消息*

        messages.success(request, '药品信息更新成功！')

*# 重定向到主页*

        return redirect('home')

    else:

*# 如果不是 POST 请求，则显示编辑表单*

        return render(request, 'accounts/edit\_drug.html', {'drug': drug})

*# 删除药品*

@login\_required

def delete\_drug(request, drug\_id):

*#登录验证*

    if not request.session.get('is\_logged\_in'):

        return redirect('login')

    Drug.objects.get(id=drug\_id).delete()

    messages.success(request, '药品删除成功！')

    return redirect('home')

这段代码定义了三个函数，用于添加、编辑和删除药品。每个视图函数都使用了Django的@login\_required装饰器，确保只有登录用户才能访问这些函数。

**添加药品：**

@login\_required：确保用户已登录。

检查用户是否在会话中已登录，如果没有则重定向到登录页面。

如果请求方法是POST，从表单中获取药品信息并创建新的药品记录，使用messages.success显示成功消息，并重定向到主页。

如果请求方法不是POST，则渲染添加药品的表单页面。

**修改药品：**

@login\_required：确保用户已登录。

检查用户是否在会话中已登录，如果没有则重定向到登录页面。

首先使用get\_object\_or\_404根据药品ID获取药品对象，如果不存在则返回404错误。

如果请求方法是POST，从表单中获取更新后的药品信息并保存，使用messages.success显示成功消息，并重定向到主页。

如果请求方法不是POST，则渲染编辑药品的表单页面，并传递药品对象。

**删除药品：**

@login\_required：确保用户已登录。

检查用户是否在会话中已登录，如果没有则重定向到登录页面。

根据药品ID获取药品对象并删除，使用messages.success显示成功消息，并重定向到主页。

### 7.功能三：数据库连接

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.sqlite3',

        'NAME': BASE\_DIR / 'db.sqlite3',*#base\_dir 根目录*

    }

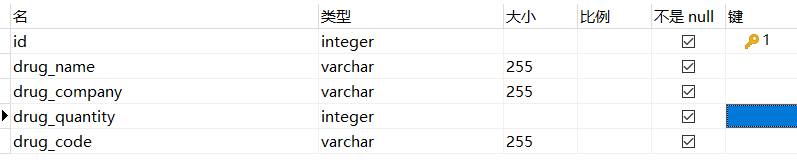
}

在Django项目的配置文件settings.py中配置数据库：

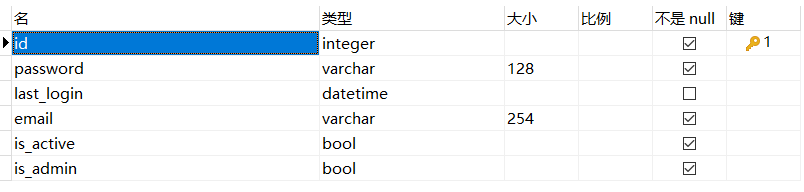
ENGINE：数据库引擎使用SQLite。

NAME：数据库文件的路径使用项目根目录下的db.sqlite3文件。

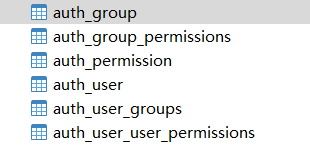
### 8.功能三：表设计



药品表，主键字段为自动生成的药品id。



用户表，主键字段为自动生成的用户id。



Django默认生成auth认证模块

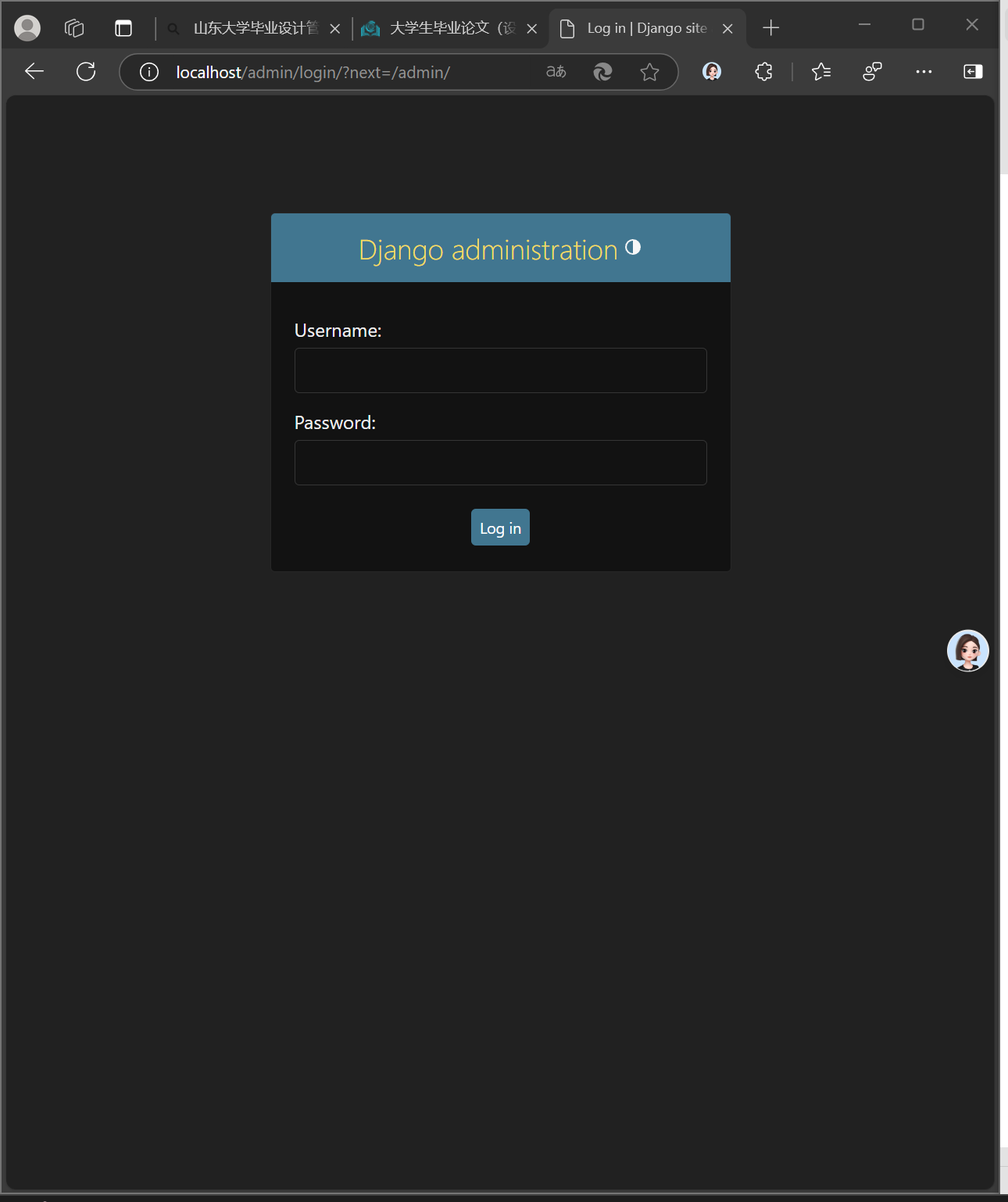
### 9.功能三：权限分配



后期可设置组织架构，将用户分配到不同的用户组中，根据需求对用户赋予各级别权限

### 运行截图

1. **登录界面**



1. **对已经注册创建的用户进行管理操作，比如删除用户和新增用户**

