《数据库系统原理》大作业

系统实现报告

题目名称： 北航十三公寓二手市场交易平台

学号及姓名： 21373061方沐阳

21373119 杨茜木

21182632苏伊

2023年 12 月 1 日

## 一．系统结构设计

**（包括体系结构和功能结构）**

* 1. **环境**

Python 3.9.9

Django 4.2.6

Vue 3.3.4

Vuex 4.1.0

Npm 8.19.3

Node 18.13.0

Vue-router 4.2.5

Webfontloader 1.0.0

Sass 1.32

Webpack-cli 5.1.4

Axios 1.6.2

* 1. **运行步骤**

前端：

npm install

npm install?@highlightjs/vue-plugin?@moefe/vue-aplayer?aplayer?axios?docx-preview?dplayer?element-plus?highlight.js?js-md5?sass?sass-loader?spark-md5?vue-clipboard3?vue-cookies?vue-pdf-embed?vue-router?vue3-pdfjs?xlsx?--save

npm run dev

后端：

python manage.py makemigrations # 生成迁移文件

python manage.py migrate # 迁移数据库，创建新表

python manage.py runserver

* 1. **体系结构**

├─backend

│ ├─.idea

│ │ ├─dataSources

│ │ │ └─5f159e85-41e2-44f6-8ef6-c6c50cd79c82

│ │ │ └─storage\_v2

│ │ │ └─\_src\_

│ │ │ └─schema

│ │ └─inspectionProfiles

│ ├─app

│ │ ├─migrations

│ │ │ └─\_\_pycache\_\_

│ │ ├─static

│ │ ├─views

│ │ │ └─\_\_pycache\_\_

│ │ └─\_\_pycache\_\_

│ ├─backend

│ │ └─\_\_pycache\_\_

│ └─media

│ ├─avator

│ └─pic

└─frontend

├─.vscode

├─public

└─src

├─api

├─assets

│ ├─icon

│ └─icon-image

├─components

├─plugins

├─router

├─store

├─utils

└─views

├─admin

├─finished

├─main

└─share

前端：使用Vue3作为基本框架进行web开发，采用响应式布局，适配不同分辨率的屏幕，具有美观性；

采用了vue单文件组件格式，使用template进行布局，JavaScript为网页处理动态事件，css美化页面；

同时基于Vue Router实现了前端路由跳转；

项目采用前后端分离的方式，前端通过api用axios从后端获取数据，减少了IO开销,降低前后端的耦合度；

项目构建上，采用vue-cli搭建项目，最后使用webpack打包。

在src/api目录下提供自定义的api接口供前后端数据的交互

在src/assets目录下放置静态文件

在src/components目录下提供各自定义组件

在src/router目录下定义路由

在src/views目录下提供各个页面代码

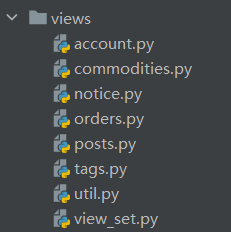
在src/utils目录下存放vue项目的全局方法

后端：使用Django作为框架，使用DjangoRestFrameWork为前端提供api接口。

在app/models.py中定义了数据库的各个表单。

在backend/urls.py中将路由转发到app/urls.py

在app/urls.py中定义了路由

****

在views文件夹中定义了多个python文件，分别负责管理：

- 账户相关的功能

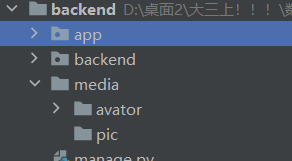
- 商品相关的功能

- 公告相关的功能

- 订单相关的功能

- 发帖相关的功能

- 标签相关的功能



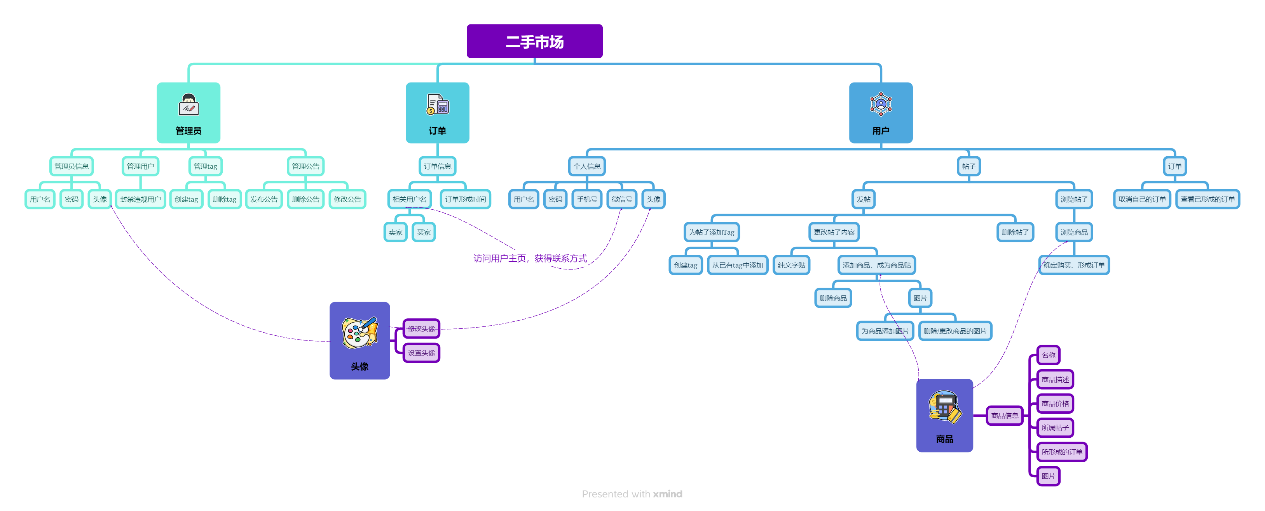
在backend/settings.py中配置了：

MEDIA\_URL = '/media/'  
MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media').replace('\\', '/')

使得用户上传的图片保存在media文件夹。

* 1. **功能结构**

功能结构的简图如下：

****

1. 登录注册

注册：输入符合要求的各项信息，成为一名用户。

登录：确认自己的身份，登入系统。（如果被管理员封禁将不能再登入）

1. 用户
   1. 管理个人信息
      1. 更改自己的联系方式
      2. 更改自己的头像
   2. 帖子
      1. 发帖

为帖子添加描述，添加商品。更改帖子内容。

* + 1. 为帖子添加/删除tag

2.2.3浏览贴子

* 1. 商品

为自己的帖子增加商品、删除商品，管理商品信息，添加和删除商品的图片

* 1. 订单
     1. 选择想要的商品，形成订单
     2. 改变主意，删除订单
     3. 通过订单查看买卖另一方的信息，获得联系方式

1. 管理员
   1. 更改个人信息
   2. 封禁用户
   3. 管理tag
   4. 管理公告

**二、数据库基本表的定义**

（没有显式定义primary key的模型，primary key为Django自动生成的id）

下列模型中关于Foreign Key的约束除了Commodity的order属性的on\_delete设置为models.SET\_NULL(表示当这个商品所属的订单被取消之后，商品的order属性设置为NULL)之外，其它都设置为models.CASCADE（举例：当商品被删除之后，属于这个商品的图片也一并被删除）。

| **Attribute** | **Primary Key** | **Constraint** |
| --- | --- | --- |
| username | True | Max length = NAME\_LEN |
| password | False | Max length = PASSWD\_LEN |
| phone | False | Max length = PHONE\_LEN, Can be null |
| wechat | False | Max length = WECHAT\_LEN, Unique to each user |
| isBaned | False | Boolean |

### User

| **Attribute** | **Primary Key** | **Constraint** |
| --- | --- | --- |
| username | True | Max length = NAME\_LEN |
| password | False | Max length = PASSWD\_LEN |
| phone | False | Max length = PHONE\_LEN, Can be null |
| wechat | False | Max length = WECHAT\_LEN, Unique to each user |
| isBaned | False | Boolean |

### Administrator

| **Attribute** | **Primary Key** | **Constraint** |
| --- | --- | --- |
| username | True | Max length = NAME\_LEN |
| password | False | Max length = PASSWD\_LEN |

### Post

| **Attribute** | **Primary Key** | **Constraint** |
| --- | --- | --- |
| time | False | Auto-generated timestamp |
| title | False | Max length = TITLE\_LEN |
| content | False | Text field |
| user | False | Foreign Key to User |

### Notice

| **Attribute** | **Primary Key** | **Constraint** |
| --- | --- | --- |
| time | False | Auto-generated timestamp, cannot be modified |
| title | False | Max length = TITLE\_LEN |
| content | False | Text field |
| admin | False | Foreign Key to Administrator |

### Tag

| **Attribute** | **Primary Key** | **Constraint** |
| --- | --- | --- |
| name | False | Max length = NAME\_LEN, Unique |
| num | False | Integer |

### Order

| **Attribute** | **Primary Key** | **Constraint** |
| --- | --- | --- |
| time | False | Auto-generated timestamp, cannot be modified |
| saler | False | Foreign Key to User, Related name = 'saler\_order' |
| buyer | False | Foreign Key to User, Related name = 'buyer\_order' |

### Commodity

| **Attribute** | **Primary Key** | **Constraint** |
| --- | --- | --- |
| name | False | Max length = TITLE\_LEN |
| description | False | Text field |
| price | False | Integer |
| post | False | Foreign Key to Post |
| order | False | Foreign Key to Order, Can be null, Set to null if Order is deleted |

### Photo

| **Attribute** | **Primary Key** | **Constraint** |
| --- | --- | --- |
| file | False | Image field, Upload to 'pic/' |
| commodity | False | Foreign Key to Commodity, Can be null |

### Avator

| **Attribute** | **Primary Key** | **Constraint** |
| --- | --- | --- |
| file | False | Image field, Upload to 'avator/' |
| user | False | Foreign Key to User, Can be null |

### Tag\_Post

| **Attribute** | **Primary Key** | **Constraint** |
| --- | --- | --- |
| tag | False | Foreign Key to Tag |
| post | False | Foreign Key to Post |

**三、系统重要功能实现方法**

**1.路由转发**

backend/urls

urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('api/', include('app.urls')), # add  
]

app/urls

urlpatterns = [  
 # path('demo/', User.Demo.as\_view()),  
 path('', include(router.urls)),  
 # 关于账号的功能  
 path('register/', account.Register.as\_view()),  
 path('login/', account.Login.as\_view()),  
 path('setAvator', account.SetAvator.as\_view()),  
 path('getAvator', account.GetAvator.as\_view()),  
 path('myPosts', account.MyPosts.as\_view()),  
 path('myOrdersAsSaler', account.MyOrdersAsSaler.as\_view()),  
 path('myOrdersAsBuyer', account.MyOrdersAsBuyer.as\_view()),  
 path('getPassword', account.GetPassword.as\_view()),  
 path('changePassword', account.ChangePassword.as\_view()),  
 path('getUserInfo', account.GetUserInfo.as\_view()),  
 path('banAUser', account.BanAUser.as\_view()),  
 path('checkBan', account.CheckBan.as\_view()),  
 path('allUsers', account.AllUsers.as\_view()),  
 # 关于帖子的功能  
 path('createPost', posts.CreatePost.as\_view()),  
 path('deletePost', posts.DeletePost.as\_view()),  
 path('getPostCommodities', posts.GetPostCommodities.as\_view()),  
 path('getPost', posts.GetPost.as\_view()),  
 path('getAllPosts', posts.GetAllPosts.as\_view()),  
 path('getPostTags', posts.GetPostTags.as\_view()),  
 path('addTagToPost', posts.AddTagToPost.as\_view()),  
 path('deleteTagToPost', posts.DeleteTagToPost.as\_view()),  
 path('deletePost', posts.DeletePost.as\_view()),  
 path('changePost', posts.ChangePost.as\_view()),  
 path('getUser', posts.GetUser.as\_view()),  
 # 关于商品的功能  
 path('createCommodity', commodities.CreateCommodity.as\_view()),  
 path('getCommodityPictures', commodities.GetCommodityPictures.as\_view()),  
 path('addCommodityPicture', commodities.AddCommodityPicture.as\_view()),  
 path('checkIfOrdered', commodities.CheckIfOrdered.as\_view()),  
 path('belongToPost', commodities.BelongToPost.as\_view()),  
 path('getCommodityDetail', commodities.GetCommodityDetail.as\_view()),  
 # 关于订单的功能  
 path('createOrder', orders.CreateOrder.as\_view()),  
 path('cancelOrder', orders.CancelOrder.as\_view()),  
 path('getOrder', orders.GetOrder.as\_view()),  
 # 关于标签的功能  
 path('createTag', tags.CreateTag.as\_view()),  
 path('allTags', tags.AllTags.as\_view()),  
 path('deleteTag', tags.DeleteTag.as\_view()),  
 path('getTagDetail', tags.GetTagDetail.as\_view()),  
 # 关于公告的功能  
 path('createNotice', notice.CreateNotice.as\_view()),  
 path('allNotice', notice.AllNotice.as\_view()),  
 path('deleteNotice', notice.DeleteNotice.as\_view()),  
 ] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

**2.图片上传和展示**

在backend/settings.py中配置了：

MEDIA\_URL = '/media/'  
MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media').replace('\\', '/')

在models中设置了：

class Photo(models.Model):  
 file = models.ImageField(upload\_to='pic/')  
 commodity = models.ForeignKey(Commodity, on\_delete=models.CASCADE, null=True)  
  
  
class Avator(models.Model):  
 file = models.ImageField(upload\_to='avator/')  
 user = models.ForeignKey(User, on\_delete=models.CASCADE, null=True)

当用户上传照片的时候，会将文件保存在media文件夹里，同时将文件的位置保存在数据库中。

当前端请求获取图片的时候，以获取用户的头像getAvator功能为例：

class GetAvator(APIView):  
 def post(self, req: Request):  
 username = req.data['username']  
 path = ''  
 try:  
 user = User.objects.get(username=username)  
 pic = Avator.objects.get(user=user)  
 path = pic.file.path  
 value = 0  
 except Exception as e:  
 print(e)  
 value = 1  
 return Response({  
 'value': value,  
 'path': path,  
 'base64': changePicPath(path)  
 })

在Avator模型中查找到所需图片的路径，将其转化为base64格式的数据返回给前端。

将图片转化为base64格式的功能实现：

def changePicPath(path):  
 with open(path, "rb") as image\_file:  
 # 读取文件  
 image\_data = image\_file.read()  
 # 对数据进行Base64编码  
 base64\_encoded\_data = base64.b64encode(image\_data)  
 # 将Base64编码的数据转换为字符串  
 base64\_message = base64\_encoded\_data.decode('utf-8')  
 return base64\_message

**3.用户注册及登录**

def check\_wechat(wechat):  
 value = 0  
 try:  
 User.objects.get(wechat=wechat)  
 except User.DoesNotExist:  
 value = 1 # 具有相同wechat的用户不存在 check通过  
 return value == 1  
  
  
def check\_username(username):  
 value = 0  
 try:  
 User.objects.get(username=username)  
 except User.DoesNotExist:  
 value = 1  
 try:  
 Administrator.objects.get(username=username)  
 except Administrator.DoesNotExist:  
 value = 2  
 # value == 2 代表 库中不存在此username 即check通过  
 return value == 2  
  
  
class Register(APIView):  
 # 普通用户  
 def post(self, request):  
 data = request.data  
 # print("register data:")  
 # print(request.data)  
 name = str(data.get('name'))  
 password = str(data.get('password'))  
 phone = str(data.get('phone'))  
 wechat = str(data.get('wechat'))  
 if not check\_username(name):  
 return Response({"value": 3}) # 此用户名已被注册  
 if not check\_wechat(wechat):  
 return Response({"value": 2}) # 此微信号已注册账号  
 try:  
 u = User.objects.create(  
 username=name,  
 password=password,  
 phone=phone,  
 wechat=wechat,  
 )  
 u.save()  
 except Exception as e:  
 print(e)  
 return Response({"value": 1}) # 数据未成功录入数据库  
 return Response({"value": 0})  
  
  
class Login(APIView):  
 def post(self, request):  
 data = request.data  
 print("login data:")  
 print(data) # debug  
 name = data.get('name')  
 password = data.get('password')  
 value = 0  
 type = -1  
 debug = True  
 if (debug):  
 all\_user = User.objects.all()  
 for user in all\_user:  
 print(user.username)  
 try:  
 item = User.objects.get(username=name)  
 print("user")  
 print(item)  
 type = 0 # 普通用户  
 except User.DoesNotExist:  
 try:  
 item = Administrator.objects.get(username=name)  
 type = 1 # 管理员  
 except Administrator.DoesNotExist:  
 value = 1 # 此用户名不存在  
 # 用户存在  
 if value == 0:  
 if password != item.password:  
 value = 2 # 密码错误  
 else:  
 # 成功登录  
 request.session['name'] = name  
 request.session['type'] = type  
 return Response({  
 'value': value,  
 'type': type,  
 })

在用户模型和管理员模型中，username都是主键，不允许重复。除此之外，也不允许同一个微信号的用户重复注册。提供了check\_username的功能来确保在注册的时候，用户提供的用户名不与现有库中任一用户或管理员重名。

**4.其它增删改查**

以对帖子的管理为例：

增:

class CreatePost(APIView):  
 def post(self, req: Request):  
 data = req.data  
 username = data['username']  
 title = data['title']  
 content = data['content']  
 post\_id = -1  
 value = -1  
 try:  
 user = User.objects.get(username=username)  
 post = Post.objects.create(  
 title=title,  
 content=content,  
 # user = user,  
 user=user,  
 )  
 post\_id = post.id  
 post.save()  
 value = 0  
 except Exception as e:  
 print(e)  
 value = 1  
 return Response({'value': value, 'post\_id': post\_id})

删：

class DeletePost(APIView):  
 def post(self, req: Request):  
 post\_id = req.data['post\_id']  
 value = 1  
 try:  
 Post.objects.get(id=post\_id).delete()  
 value = 0  
 except Exception as e:  
 print(e)  
 return Response(value)

改：

class ChangePost(APIView):  
 def post(self, req: Request):  
 data = req.data  
 post\_id = data['post\_id']  
 title = data['title']  
 content = data['content']  
 value = -1  
 try:  
 post = Post.objects.get(id=post\_id)  
 if title != '':  
 post.title = title  
 if content != '':  
 post.content = content  
 post.save()  
 value = 0  
 except Exception as e:  
 print(e)  
 value = -1  
 return Response({  
 'value': value  
 z

查：

获取所有帖子的信息：

class GetAllPosts(APIView):  
 def post(self, req: Request):  
 return\_data = []  
 allPost = Post.objects.all()  
 for post in allPost:  
 return\_data.append({  
 'post\_id': post.id,  
 'title': post.title,  
 'content': post.content,  
 'username': post.user.username,  
 'date': post.time  
 })  
 return Response({  
 "allposts": return\_data  
 })

根据id对帖子进行查询：

class GetPost(APIView):  
 def post(self, req: Request):  
 post\_id = req.data['post\_id']  
 post = Post.objects.get(id=post\_id) # 根据主键查找  
 return Response({  
 'title': post.title,  
 'content': post.content,  
 'username': post.user.username,  
 'date': post.time  
 })

其它：

为帖子增加tag：

class AddTagToPost(APIView):  
 def post(self, req: Request):  
 data = req.data  
 post\_id = data['post\_id']  
 tag\_id = data['tag\_id']  
 value = -1  
 find = True  
 try:  
 item = Tag\_Post.objects.get(post\_\_id=post\_id, tag\_\_id=tag\_id)  
 except Tag\_Post.DoesNotExist:  
 find = False  
 if(not find):  
 try:  
 tag = Tag.objects.get(id=tag\_id)  
 Tag\_Post.objects.create(  
 post=Post.objects.get(id=post\_id),  
 tag=tag,  
 )  
 tag.num = tag.num + 1  
 tag.save()  
 value = 0  
 except Exception as e:  
 print(e)  
 value = 1  
 else:  
 value = 2  
 return Response({  
 'value': value  
 })

通过查询来确保不可以为同一帖子重复增加同一标签。

删除标签：

class DeleteTagToPost(APIView):  
 def post(self, req: Request):  
 data = req.data  
 post\_id = data['post\_id']  
 tag\_id = data['tag\_id']  
 value = -1  
 try:  
 tag = Tag.objects.get(id=tag\_id)  
 Tag\_Post.objects.filter(post\_\_id=post\_id, tag\_\_id=tag\_id).delete()  
 tag.num = tag.num - 1  
 tag.save()  
 value = 0  
 except Exception as e:  
 print(e)  
 value = 1  
 return Response({  
 'value': value  
 })

**5.创建订单**

class CreateOrder(APIView):  
 def post(self, req: Request):  
 data = req.data  
 buyer\_name = data['username']  
 commodity\_id = data['commodity\_id']  
 value = 1  
 id = -1  
 try:  
 # 在order表中增添记录  
 post\_id = Commodity.objects.get(id=commodity\_id).post.id # 查询商品所属的帖子  
 saler = Post.objects.get(id=post\_id).user # 查询帖子所属的用户  
 buyer = User.objects.get(username=buyer\_name)  
 order = Order.objects.create(  
 saler=saler,  
 buyer=buyer,  
 )  
 # 在commodity表中将commodity和order关联起来  
 c0 = Commodity.objects.get(id=commodity\_id)  
 c0.order = order  
 c0.save() # 保存  
 value = 0  
 id = order.id  
 except Exception as e:  
 print(e)  
 return Response({  
 'value': value,  
 'order\_id': id,  
 })

1. **系统实现结果**

### 4.1 登录注册页面

#### 4.1.1 登录

用户进入网站首先看到登录注册页面，输入用户名和密码可以进入网站。



#### 4.1.2 注册

在登录页面点击“没有账号”自动跳转到注册，后端验证用户名是否已存在、微信号是否重复。点击注册跳会转到登录页面，此时用户可以用新注册的账号进行登录。



### 4.2 导航页面

该项目采用左侧双重导航栏，可以在左侧点击跳转到想去的页面。



### 4.3 修改个人信息

在头像位置可以修改头像或密码，也可以登出该网站。



#### 4.3.1 修改头像

点击“选择”按钮可以选择本地图片并上传新头像。



#### 4.3.2 修改密码

在弹框中输入正确的旧密码和想要修改的新密码，点击确认提交表单。



### 4.4 首页

#### 4.4.1 帖子广场页面

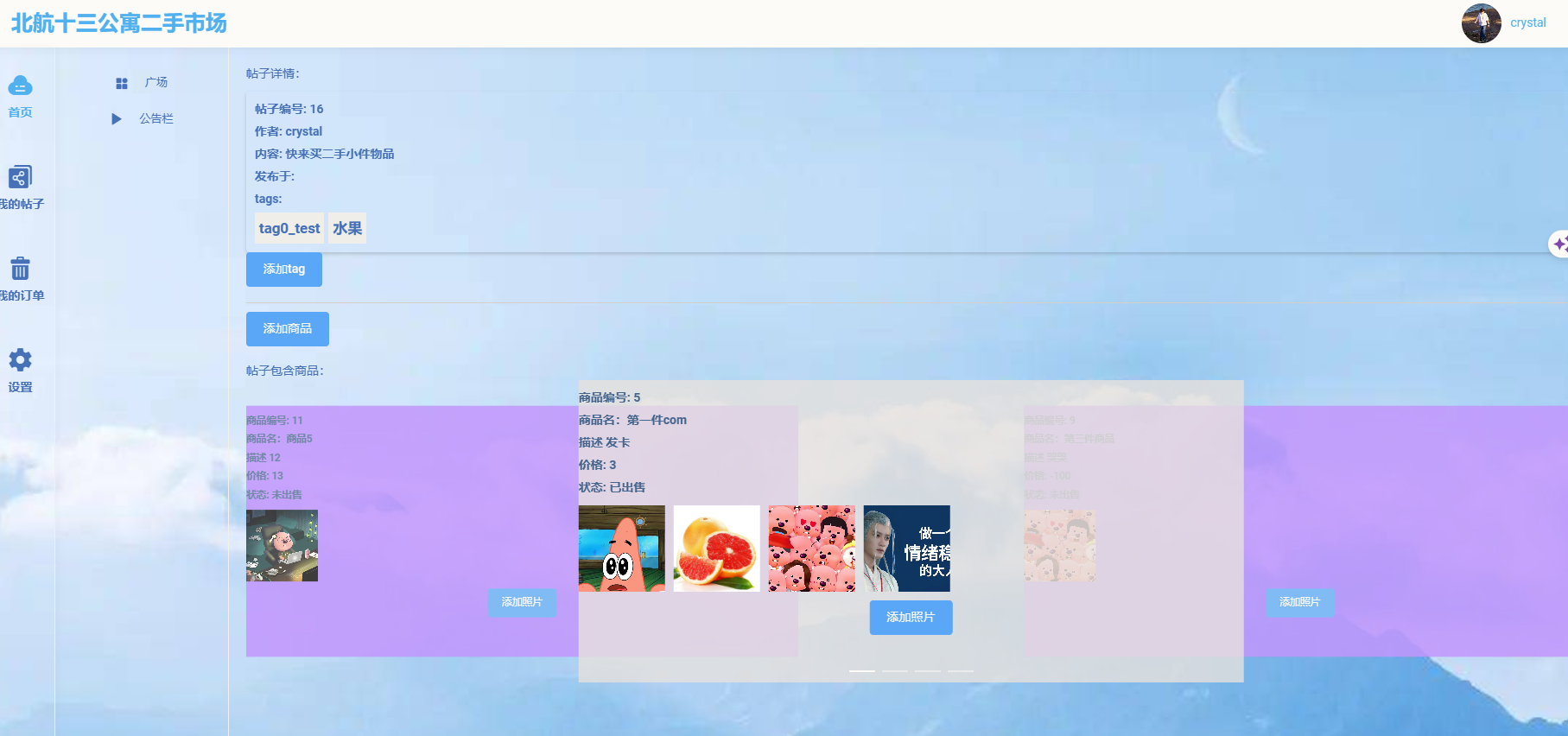


普通用户可以点击“发布帖子”来发布帖子：

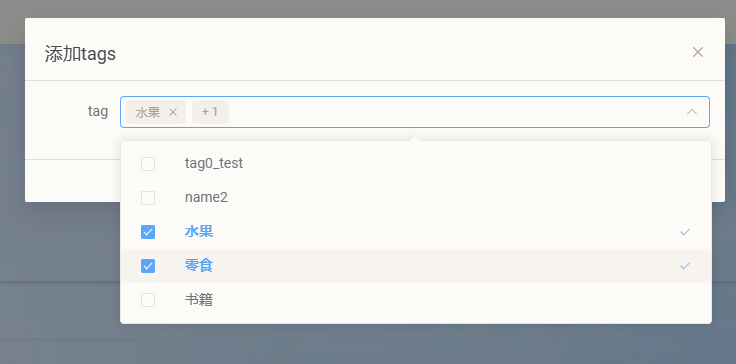


也可以点击其中一个帖子查看帖子详情：

详情包括帖子编号、作者、内容、帖子包含的tag及帖子包含的商品。



发布该帖子的用户可以点击tag选择删除，或者点击“添加tag”选择想添加的tag来添加。



可以左右滑动来查看商品、添加商品图片，该帖子所属用户也可以点击添加商品给当前帖子添加商品。

#### 4.4.2 公告栏

公告栏展示了所有管理员发布的公共，管理员用户还可以点击“发布公告”来创建新公告或删除对应公告。



### 4.5 我的帖子页面

我的帖子页面展示所有该用户发布的帖子，用户可以删除自己帖子或修改自己的帖子，也可以点击帖子标题跳转到帖子详情页。



### 4.6 我的订单页面

该页面展示所有已交易的订单，包括用户作为买家的订单和用户作为卖家的订单，用户可以点击商品名进入商品详情页查看，也可以选择取消订单。



### 4.7 设置页面

#### 4.7.1 用户管理

管理员可以查看用户列表，并选择是否封禁用户。



#### 4.7.2 tag管理

管理员可以查看tag列表并选择删除或创建tag。



**五、总结**

这份作业的创意来源于北航十三公寓的许多二手群。在这些二手群里，每天都活跃着许多与我们三位组员住在同一公寓的姐妹。这些二手群为我们的生活提供了很大的便利，有人以低廉的价格售出多余的衣服，就有人以划算的价格获得了二手“新”衣服；有人售出从餐厅打包回来的多余食物，就有人心情烦闷待在宿舍、刷着手机突然眼睛一亮，发现可以便宜方便地吃到自己喜欢的美食。在这些群里，我们进行奶茶拼单，我们出售十元两三片的达美乐披萨，我们借给彼此需要的物品。在疫情的时候，我在这些群里借到过药、买到过新冠病毒的检测试剂，我借出过自己的体温计。我在二手群里借用打火机给朋友过生日，不到五分钟就有三四个人加我的微信。除此之外，我们在群里询问信息，也总是能很快地得到应答。

以上的经历是本次作业中“信息贴”和“商品贴”的来源。在微信群里不容易及时标记物品是否已经售出，在网页上却可以用形成订单来标记物品的交易状态。

在二手微信群里可能会有一些外来的打广告人员，在web中，管理员可以对违规用户进行封禁，维护友善互助的交易氛围。

在这份作业的完成过程中，我们三名队友主要通过微信交流，发现问题及时通知彼此，一起完善需求、创意和实现，不仅完成自己的工作，还帮助彼此搜集资料、实现功能。