**前端部分（陈彤）**

**1、前端开发github地址：https://github.com/qwersab/docs.git，切换到front分支就是**

**2、Docs更新的前端技术开发框架以及实现**

### 2.1开发框架之技术栈：

Vue.js 3：用于构建用户界面。

Vue Router：用于页面路由管理。

Pinia：用于状态管理。

Axios：用于封装 HTTP 请求，与后端进行数据交互。

Element UI：用于快速构建交互性较强的页面组件。

### 2.2实现内容

### 2.2.1前端框架搭建：

初始化 Vue 项目，安装 Vue Router、pinia、Axios、Element-plus 等依赖。

配置路由，设置页面组件（首页、商品详情页、我的商品页、收藏页、足迹页、订单页、发布/编辑商品页、个人中心页、登录注册页）。

### 2.2.2页面设计与实现：

### 使用 Element UI ，确保界面美观且交互性强。

### 实现主要页面，包括：

### 商品列表页：展示商品信息，支持搜索和筛选功能。

### 商品详情页：展示商品详细信息，提供购买按钮。

### 登录注册页：实现用户登录和注册功能。

### 个人中心：展示用户信息、可修改个人信息、密码。

### 发布商品页：提供表单用于上传商品信息和图片。

### 2.2.3与后端接口对接：

### 使用 Axios 封装 API 请求，与后端进行数据交互。

### 实现前端与后端的联调，包括商品发布编辑删除展示、订单管理、用户登录注册、用户管理、商品收藏、想要、等功能。

### 2.2.4前端功能测试：

### 测试页面的交互功能，确保用户操作流程顺畅。

### 检查数据的正确显示和交互逻辑。

1. **当前的开发进度**

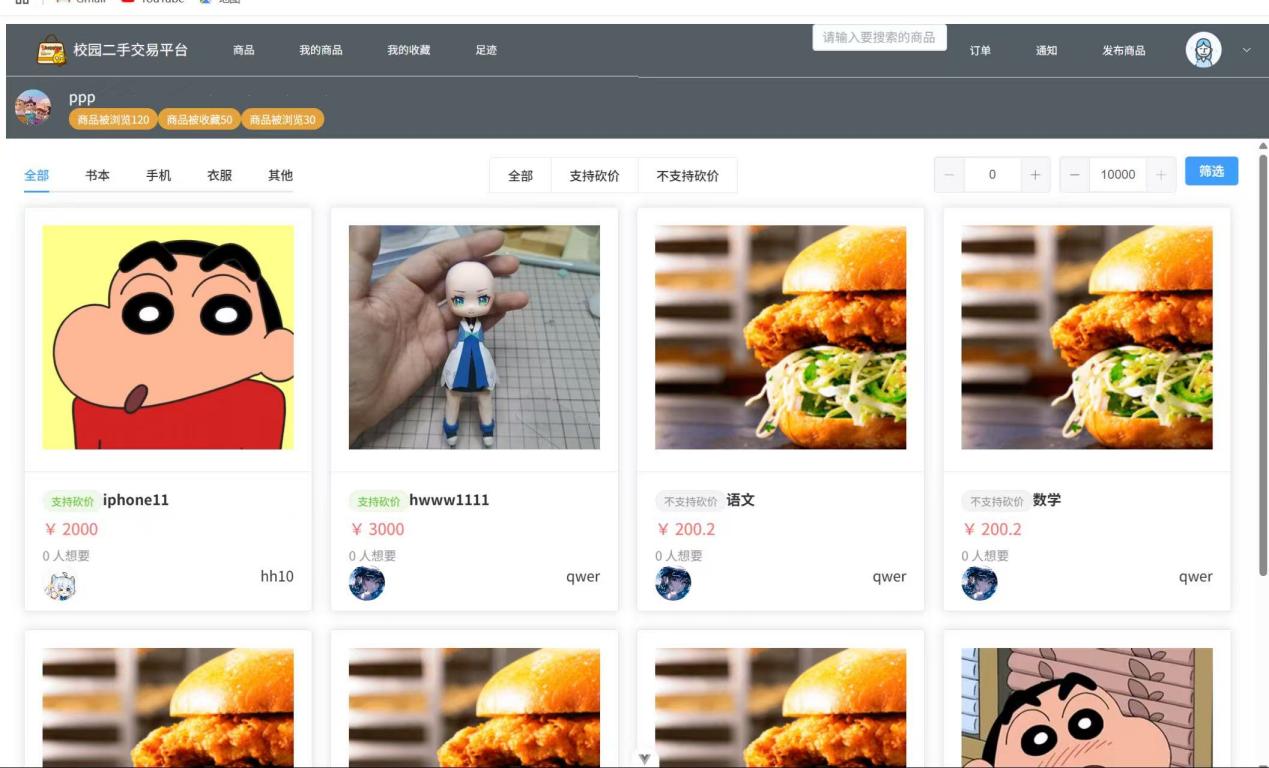
3.1 登录注册页面实现，以及个人中心修改个人信息、修改密码实现，接口联调

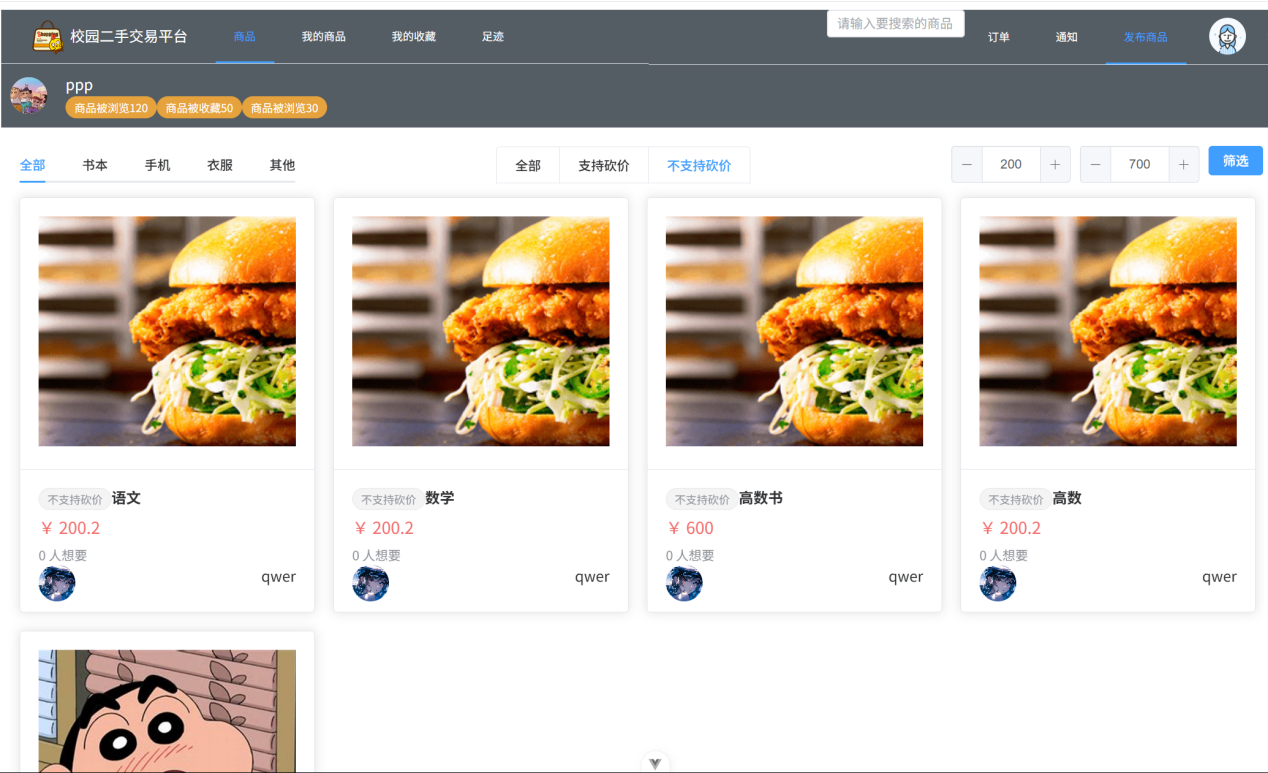
****

****

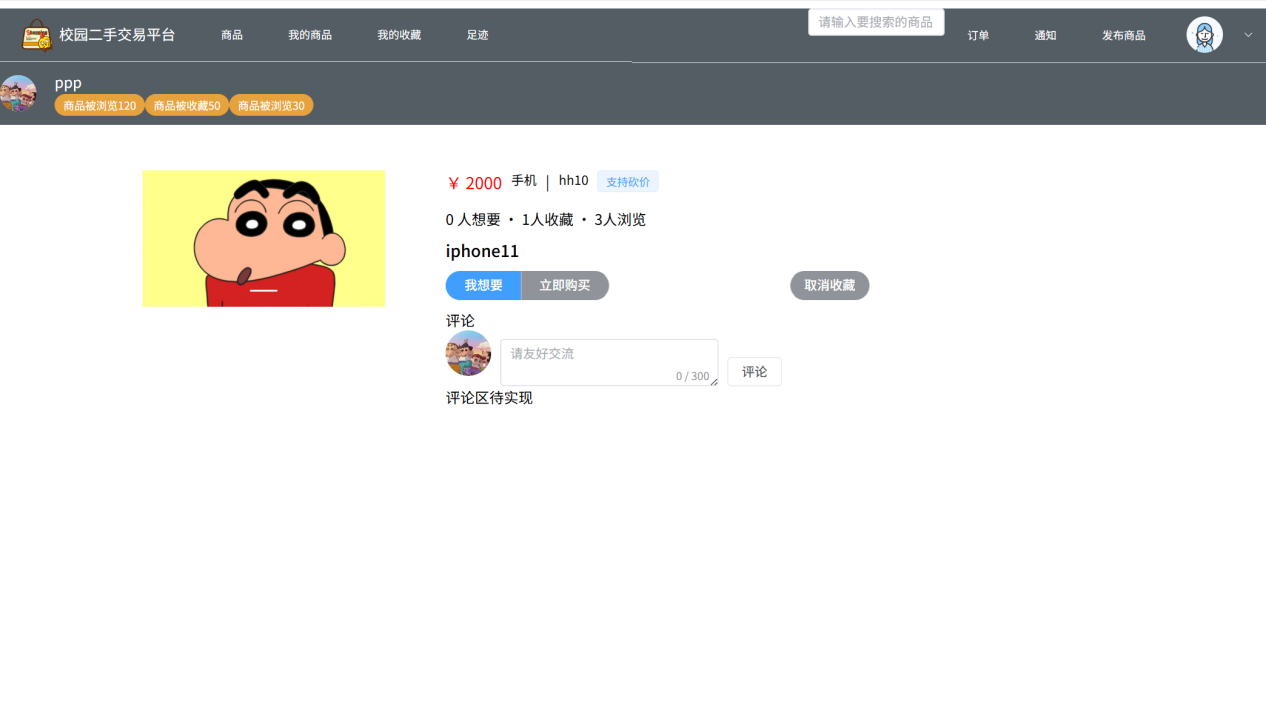


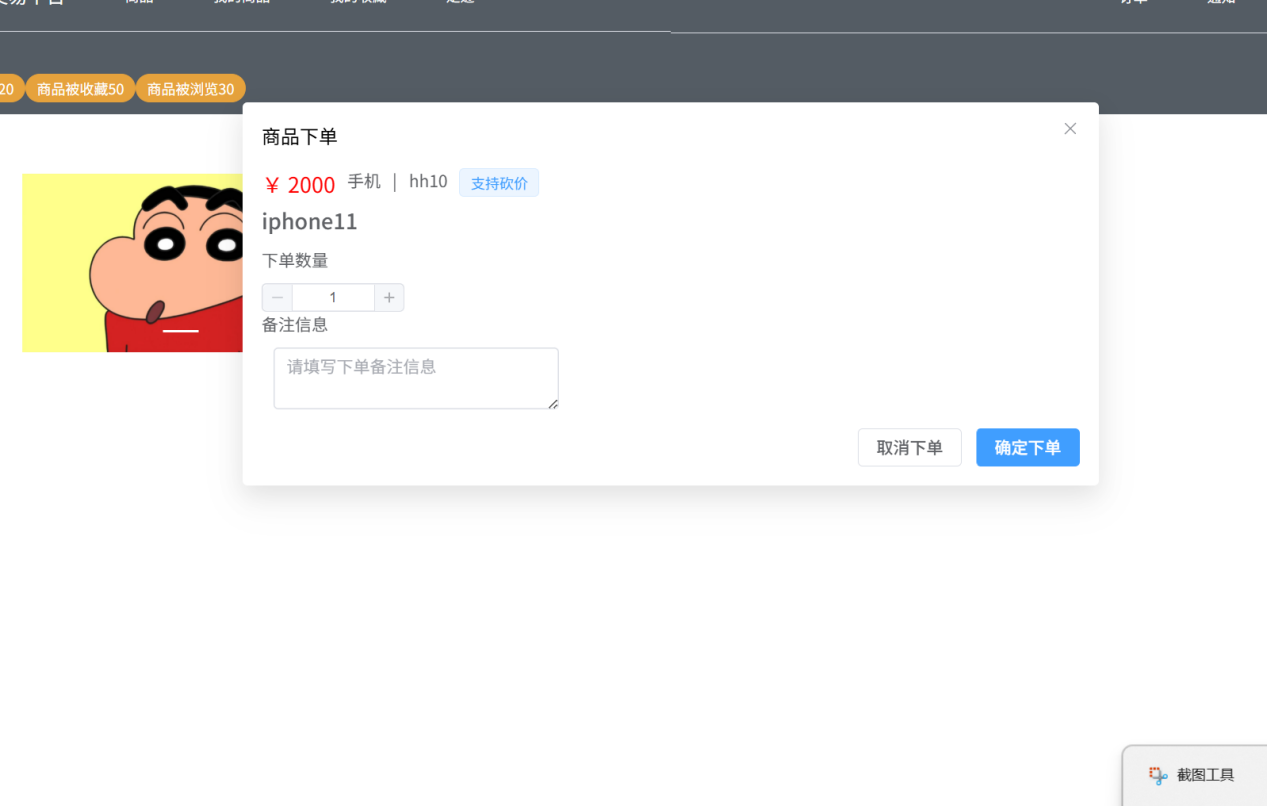
3.2商品展示默认界面以及分类筛选实现后的界面，接口联调



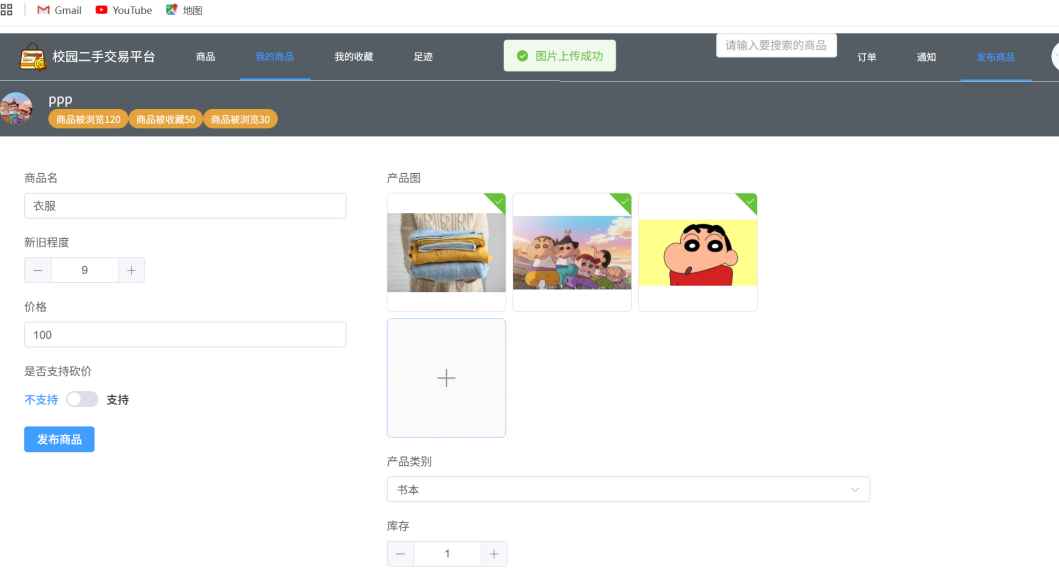


3.3 商品详情界面以及立即购买弹出下单弹窗，部分接口联调

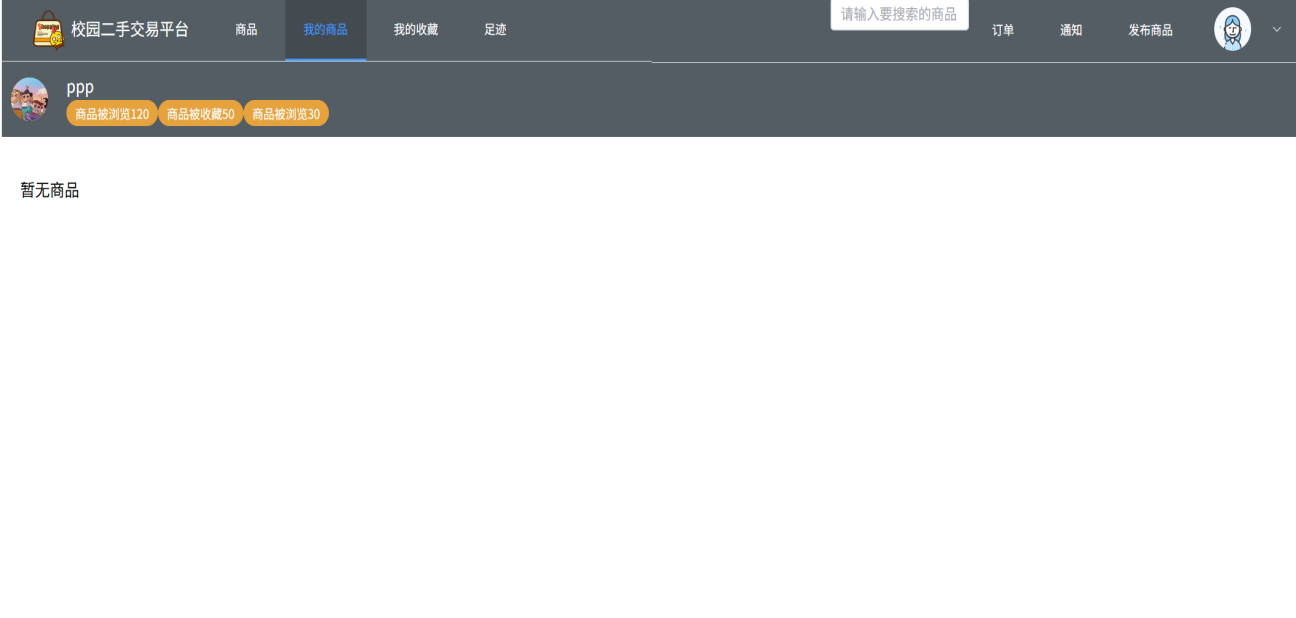


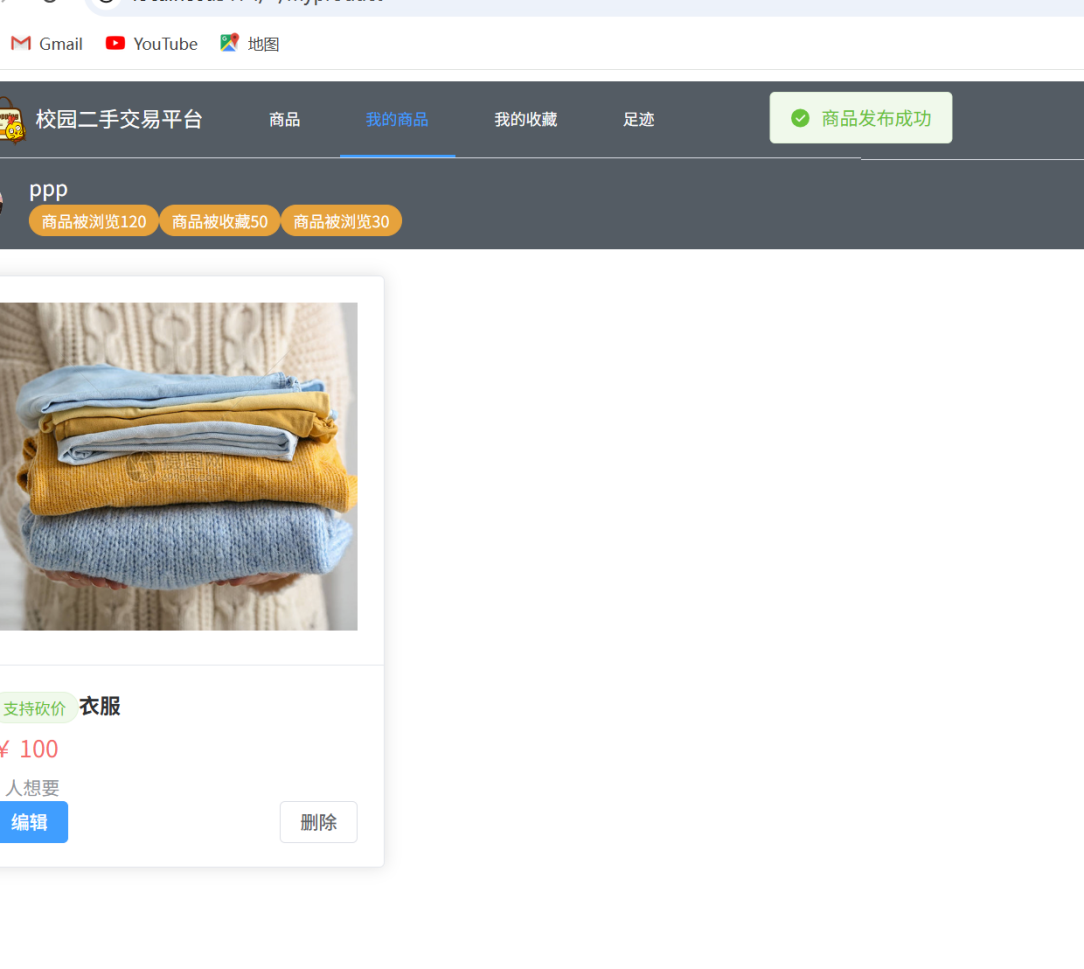


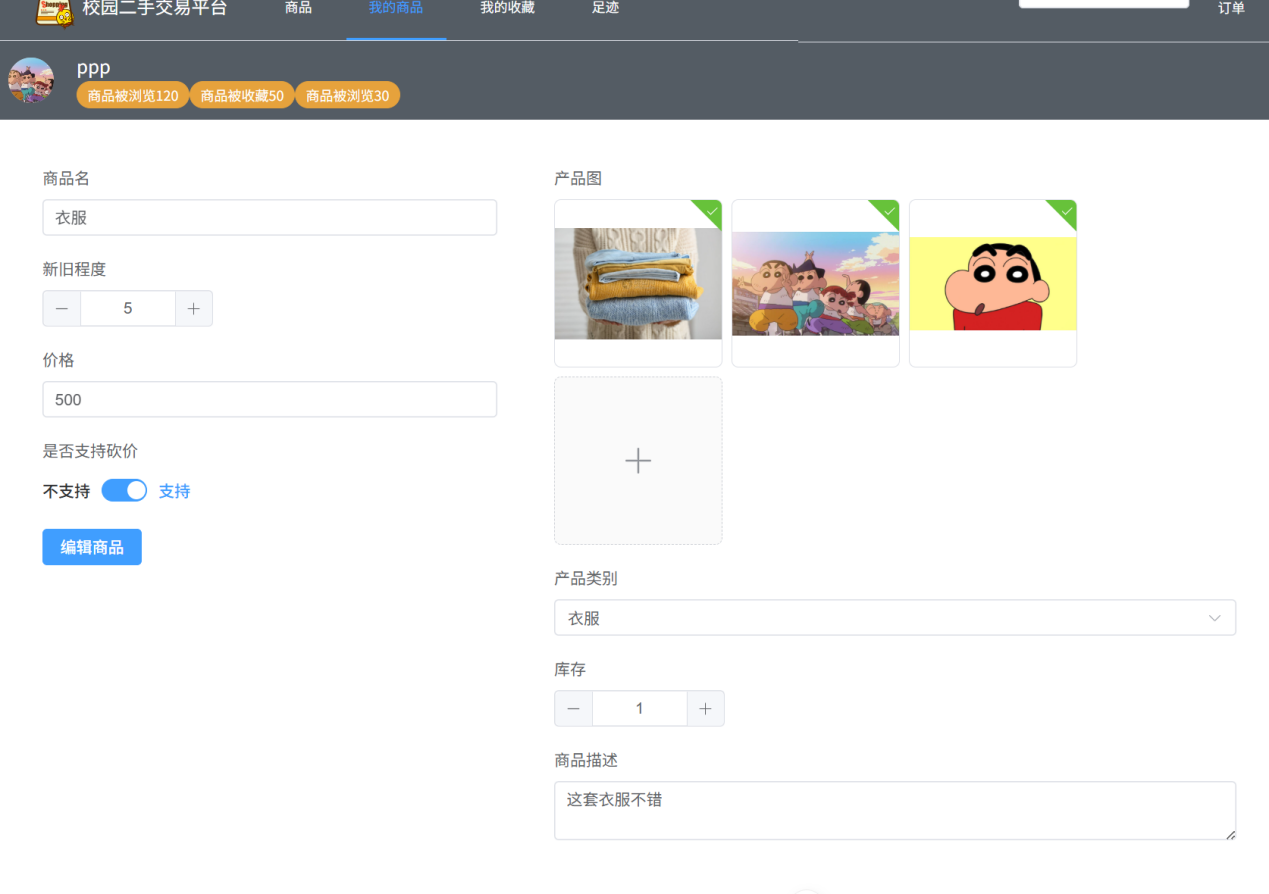
3.3 发布商品界面，接口联调

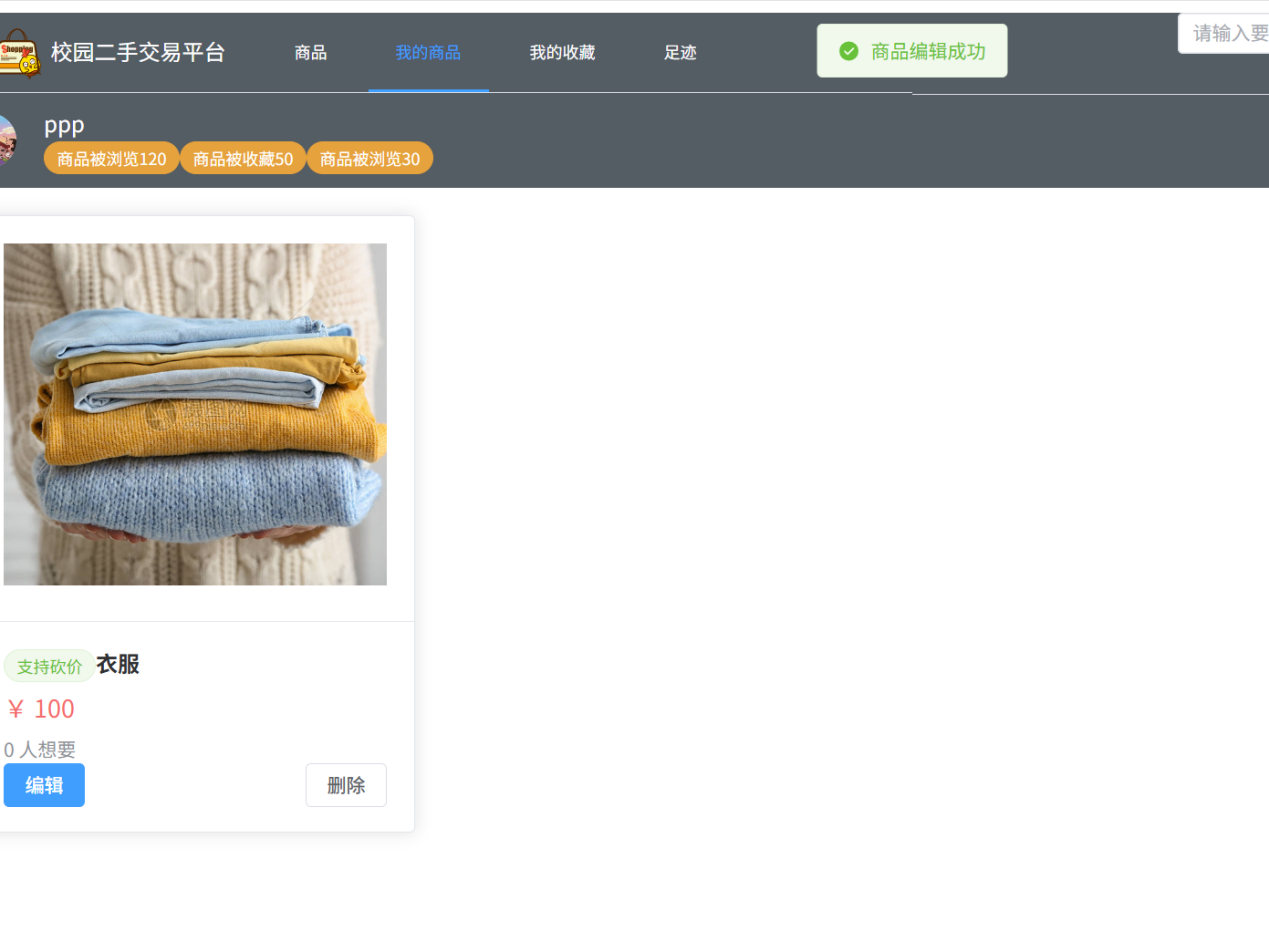


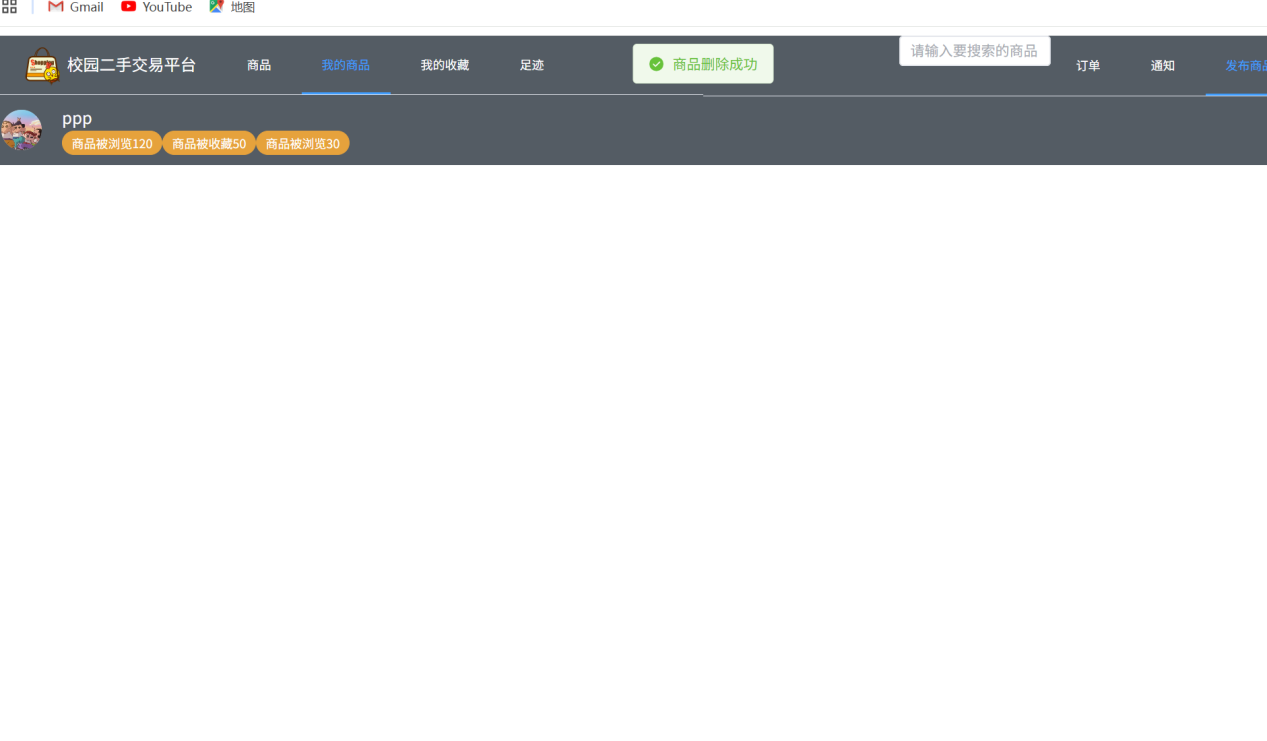
3.4 我的商品默认界面，以及发布商品之后的界面，以及我发布的商品可编辑删除，接口联调











**测试部分（张佳欣）**

1. **测试方案**

## 测试方案

### 11. 测试目标

验证“校园二手交易平台”是否满足以下要求：

功能完整性：所有功能模块（用户管理、商品管理、订单管理、评论与互动等）正常运行。

数据准确性：数据存储、查询、更新和删除操作正确无误。

用户体验：界面友好，操作流程顺畅，响应时间合理。

系统稳定性：在高并发和长时间运行下保持稳定。

安全性：用户数据和交易信息安全可靠。

### 1.2. 测试范围

#### 功能测试

用户模块：注册、登录、个人信息修改、密码重置、权限控制。

商品模块：商品发布、编辑、删除、搜索、筛选、分类管理。

订单模块：订单创建、查询、支付、状态更新。

评论与互动模块：评论、点赞、收藏、消息通知。

#### 1.3性能测试

页面加载时间。

数据库查询响应时间。

高并发下的系统稳定性。

#### 安全性测试

用户认证与授权。

数据加密与传输安全。

防止SQL注入、XSS攻击等。

#### 兼容性测试

浏览器兼容性（Chrome、Firefox、Safari等）。

### 1.4. 测试环境

#### 开发环境

前端：Vue.js 3、Vue Router、Vuex、Axios、Element UI。

后端：Django、Django REST Framework、SQLite。

开发工具：Visual Studio Code、PyCharm、Node.js、NPM。

#### 测试环境

服务器：Ubuntu 20.04，Nginx，Gunicorn。

数据库：SQLite（开发阶段）。

浏览器：Chrome、Firefox、Safari、Edge。

移动设备：Android、iOS。

### 5. 测试计划

#### 阶段 1：功能测试（第9-12周）

任务：完成所有功能模块的测试用例设计和执行。

责任人：张佳欣（测试负责人）。

输出：功能测试报告，列出所有通过和未通过的测试用例。

#### 阶段 2：兼容性测试（第10-13周）

任务：测试浏览器和移动端兼容性。

责任人：张佳欣（测试负责人）。

输出：兼容性测试报告，列出兼容性问题及修复建议。

#### 阶段 5：回归测试（第14周）

任务：修复问题后，重新测试所有功能模块。

责任人：全体开发人员。

输出：回归测试报告，确保所有问题已修复。

**2、自动测试**

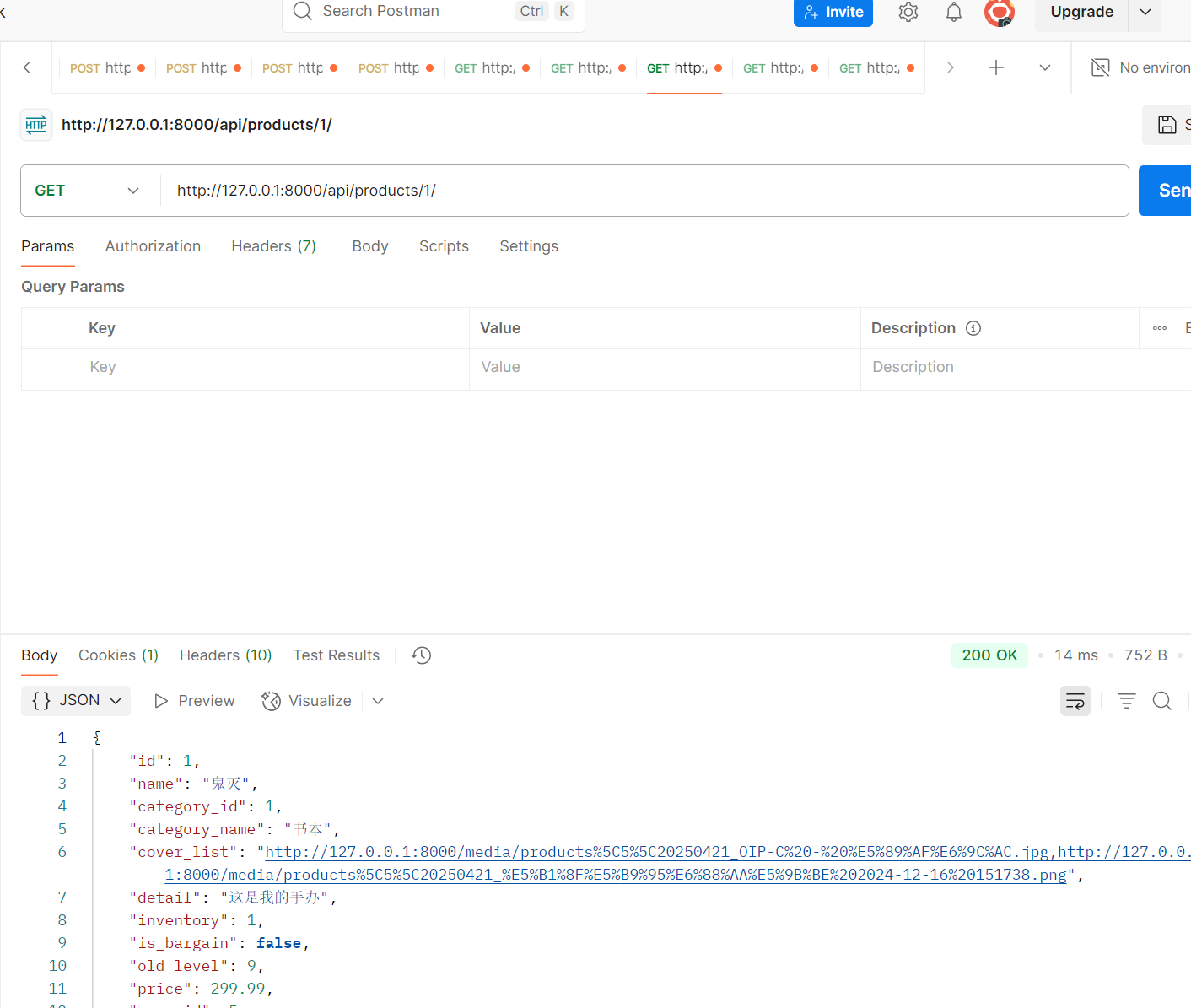
单元测试覆盖范围：用户注册与身份验证

数据创建和检索、权限验证

错误处理、边界情况



**3、手动测试**



**后端部分（徐鹏强）**

1. **后端开发github地址：https://github.com/qwersab/docs.git，切换到backend分支就是**

**2、Docs更新的后端技术开发框架以及实现**

### 开发框架

技术栈：

Django：后端框架，用于构建 RESTful API。

Django REST Framework（DRF）：用于简化 RESTful API 的开发。

SQLite：轻量级数据库，用于存储用户、商品、订单等数据。

**3、数据库设计文档**

1. 用户表（User）

| 字段名 | 类型 | 说明 | 是否为空 | 备注 |
| --- | --- | --- | --- | --- |
| id | Integer (PK) | 主键 | 否 | 自动增长 |
| username | Varchar(150) | 用户名 | 否 | Django 默认 |
| email | Email | 邮箱地址 | 是 | 唯一 |
| password | Varchar(128) | 密码 | 否 | 加密存储 |
| profile\_picture | Image | 头像 | 是 | 上传到 profiles/ |
| login\_at | DateTime | 最近登录时间 | 是 |  |
| created\_at | DateTime | 注册时间 | 是 | 自动创建 |

2. 商品表（Product）

| **字段名** | **类型** | **说明** | **是否为空** | **备注** |
| --- | --- | --- | --- | --- |
| id | Integer (PK) | 主键 | 否 | 自动增长 |
| name | Varchar(200) | 商品名称 | 是 |  |
| category\_id | Integer | 商品分类 ID | 是 | 使用常量或关联表 |
| cover\_list | Text | 封面图列表 | 是 | 存储 JSON 数组字符串 |
| detail | Text | 商品简介 | 是 |  |
| inventory | Integer | 库存数量 | 否 | 默认 1 |
| is\_bargain | Boolean | 是否支持砍价 | 否 | 默认 False |
| old\_level | Integer | 新旧程度 | 是 | 自定义 1～10 等级等 |
| price | Decimal(10,2) | 售价 | 是 |  |
| user\_id | FK -> User | 发布者 | 否 |  |
| create\_at | DateTime | 发布时间 | 是 | 自动创建 |

3. 商品分类表（ProductCategory）

| **字段名** | **类型** | **说明** | **是否为空** | **备注** |
| --- | --- | --- | --- | --- |
| id | Integer (PK) | 分类ID | 否 | 自动增长 |
| name | Varchar(100) | 分类名称 | 否 | 如：书籍、手机等 |

4. 订单表（Order）

| **字段名** | **类型** | **说明** | **是否为空** | **备注** |
| --- | --- | --- | --- | --- |
| id | Integer (PK) | 主键 | 否 |  |
| user\_id | FK -> User | 下单用户 | 否 |  |
| product\_id | FK -> Product | 购买商品 | 否 |  |
| quantity | Integer | 购买数量 | 否 | 默认 1 |
| status | Varchar(20) | 订单状态 | 否 | pending/paid/shipped |
| created\_at | DateTime | 下单时间 | 否 | 自动生成 |

5. 评论表（Comment）

| **字段名** | **类型** | **说明** | **是否为空** | **备注** |
| --- | --- | --- | --- | --- |
| id | Integer (PK) | 主键 | 否 |  |
| user\_id | FK -> User | 评论人 | 否 |  |
| product\_id | FK -> Product | 评论的商品 | 否 |  |
| content | Text | 评论内容 | 否 |  |
| rating | Integer | 评分（可选） | 是 | 可用于星级展示 |
| created\_at | DateTime | 评论时间 | 否 |  |

6. 互动行为表（Interaction）

| **字段名** | **类型** | **说明** | **是否为空** | **备注** |
| --- | --- | --- | --- | --- |
| id | Integer (PK) | 主键 | 否 |  |
| user\_id | FK -> User | 用户 | 否 |  |
| product\_id | FK -> Product | 商品 | 否 |  |
| behavior | Varchar(50) | 行为类型 | 否 | like / collect / view 等 |
| created\_at | DateTime | 发生时间 | 否 |  |

7. 消息表（Message）

| **字段名** | **类型** | **说明** | **是否为空** | **备注** |
| --- | --- | --- | --- | --- |
| id | Integer (PK) | 主键 | 否 |  |
| sender\_id | FK -> User | 发送者 | 否 |  |
| receiver\_id | FK -> User | 接收者 | 否 |  |
| content | Text | 消息内容 | 否 |  |
| is\_read | Boolean | 是否已读 | 否 | 默认 False |
| created\_at | DateTime | 发送时间 | 否 | 自动创建 |

8. 操作日志表（ActionLog）

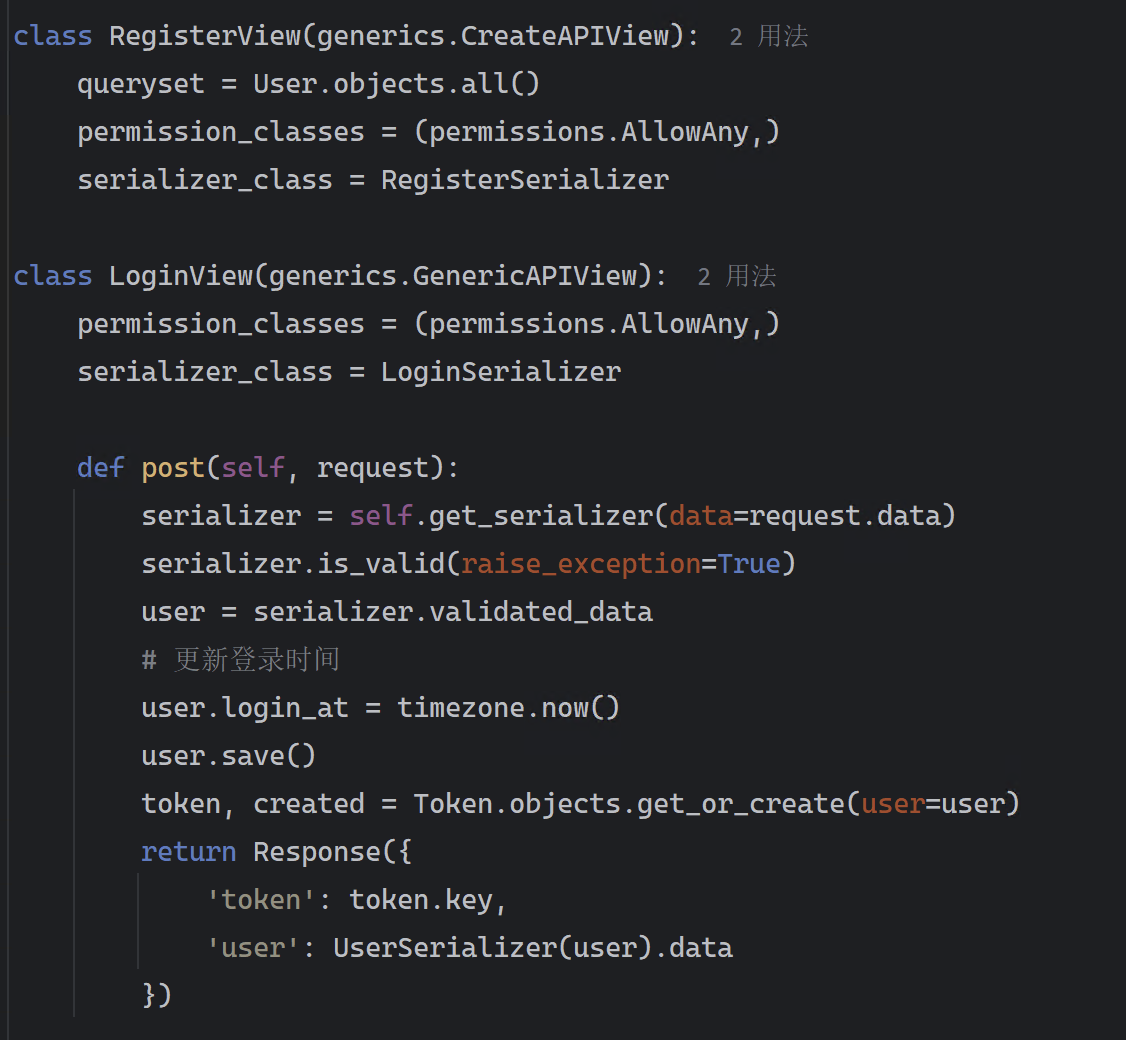
| **字段名** | **类型** | **说明** | **是否为空** | **备注** |
| --- | --- | --- | --- | --- |
| id | Integer (PK) | 主键 | 否 |  |
| user\_id | FK -> User | 操作用户 | 是 | 删除用户后置空 |
| action | Varchar(100) | 操作类型 | 否 | 如：登录、下单等 |
| detail | Text | 操作详情 | 是 | 可选 |
| created\_at | DateTime | 操作时间 | 否 | 自动创建 |

**4、当前开发进度**

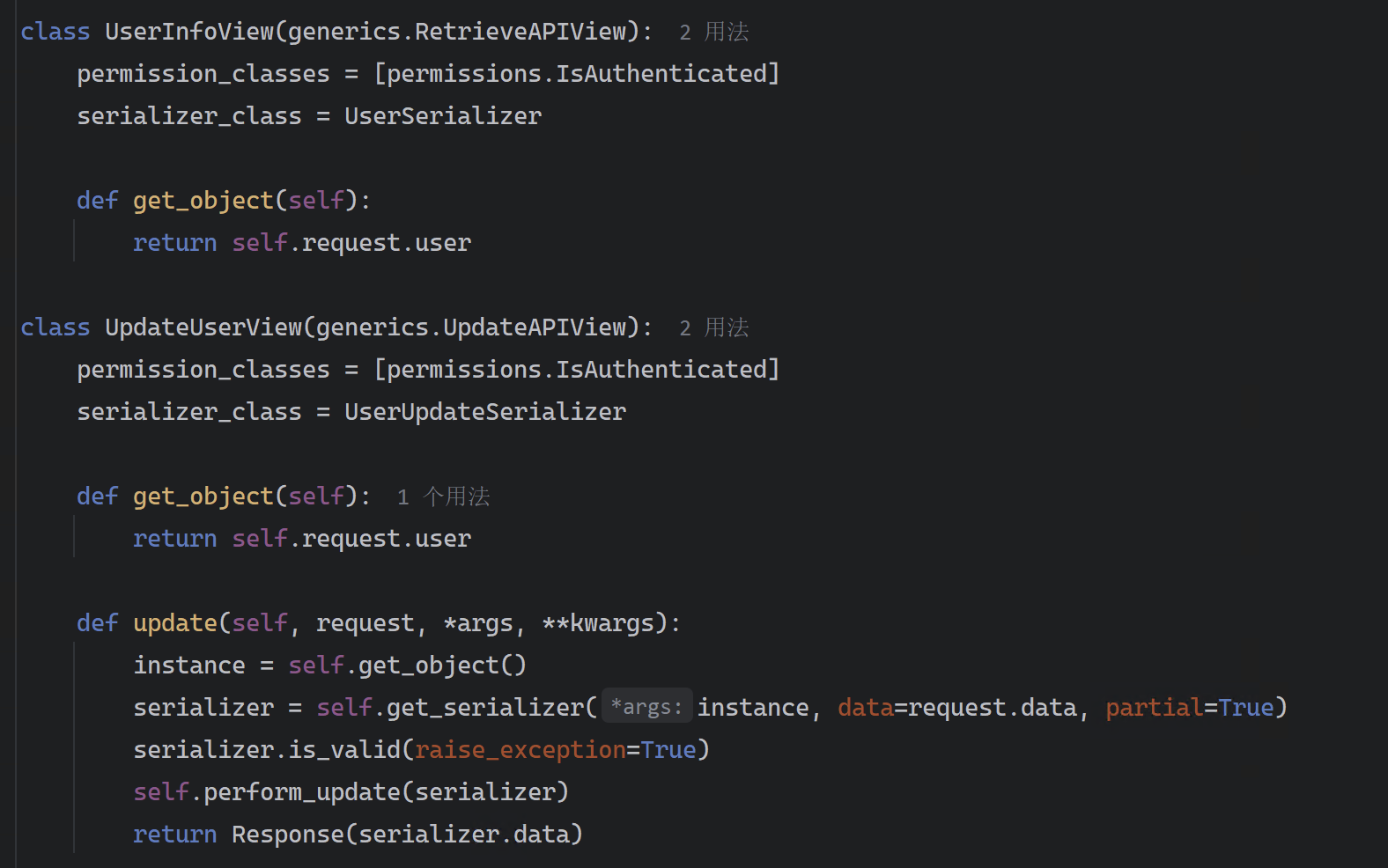
根据apifox接口文档实现了交易平台部分后端api,包括

用户管理系统：

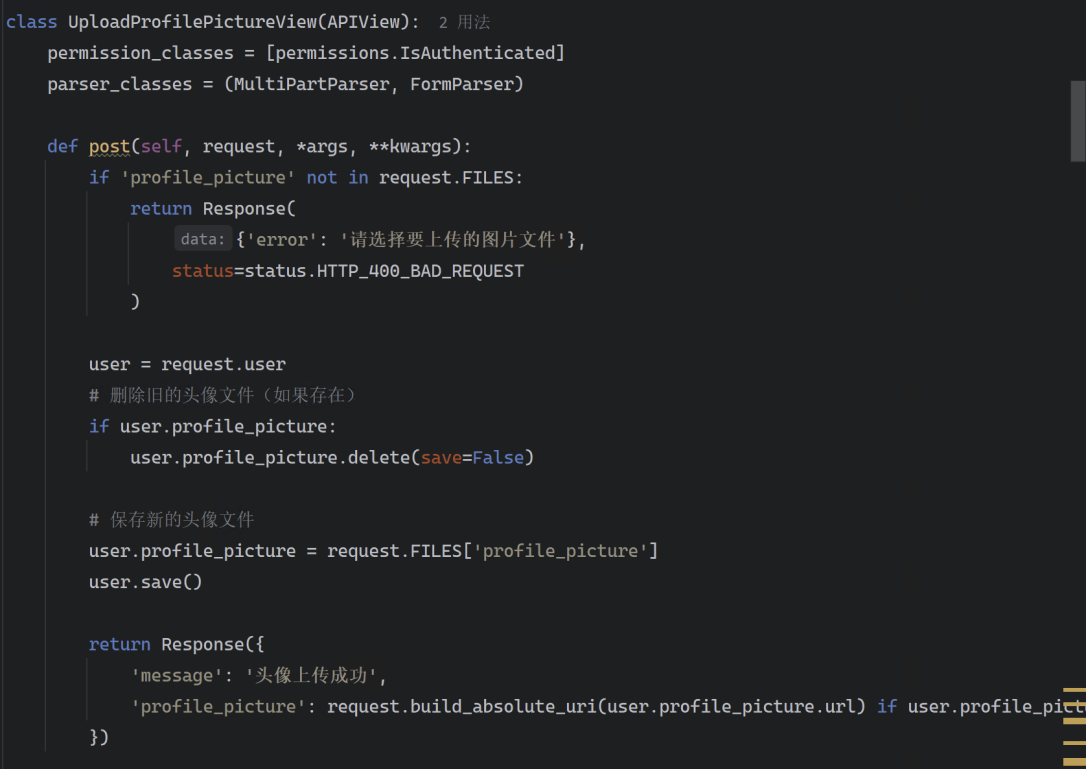
1.用户注册和登录



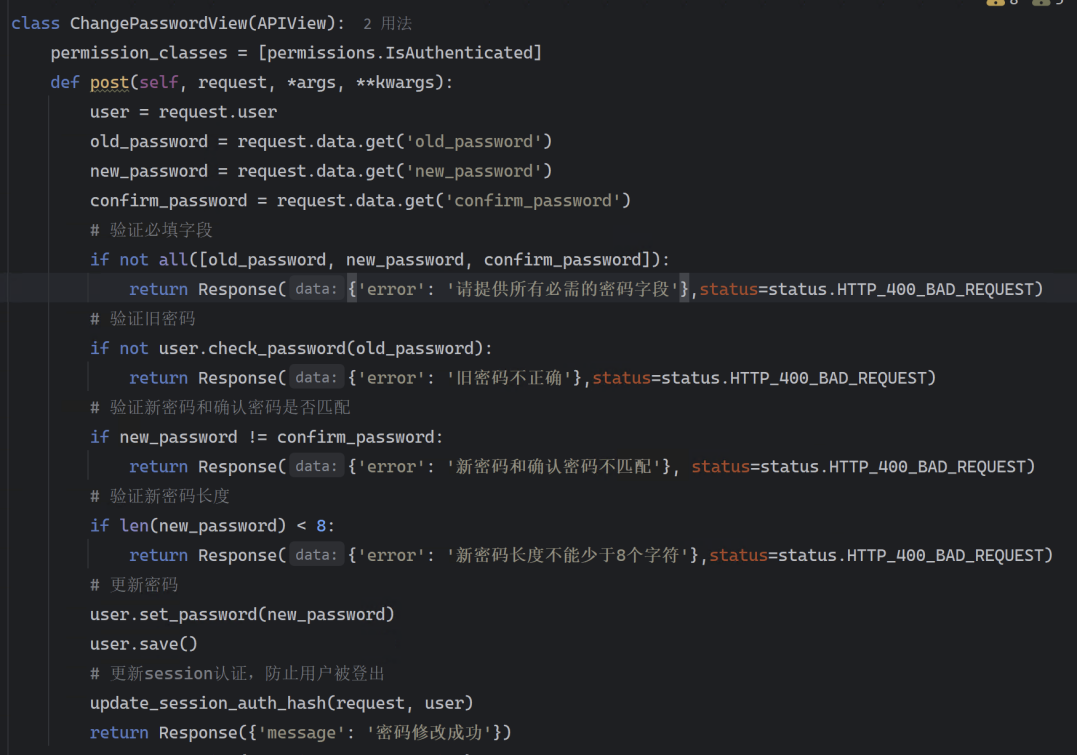
2.获取和更新用户信息



3.用户头像上传

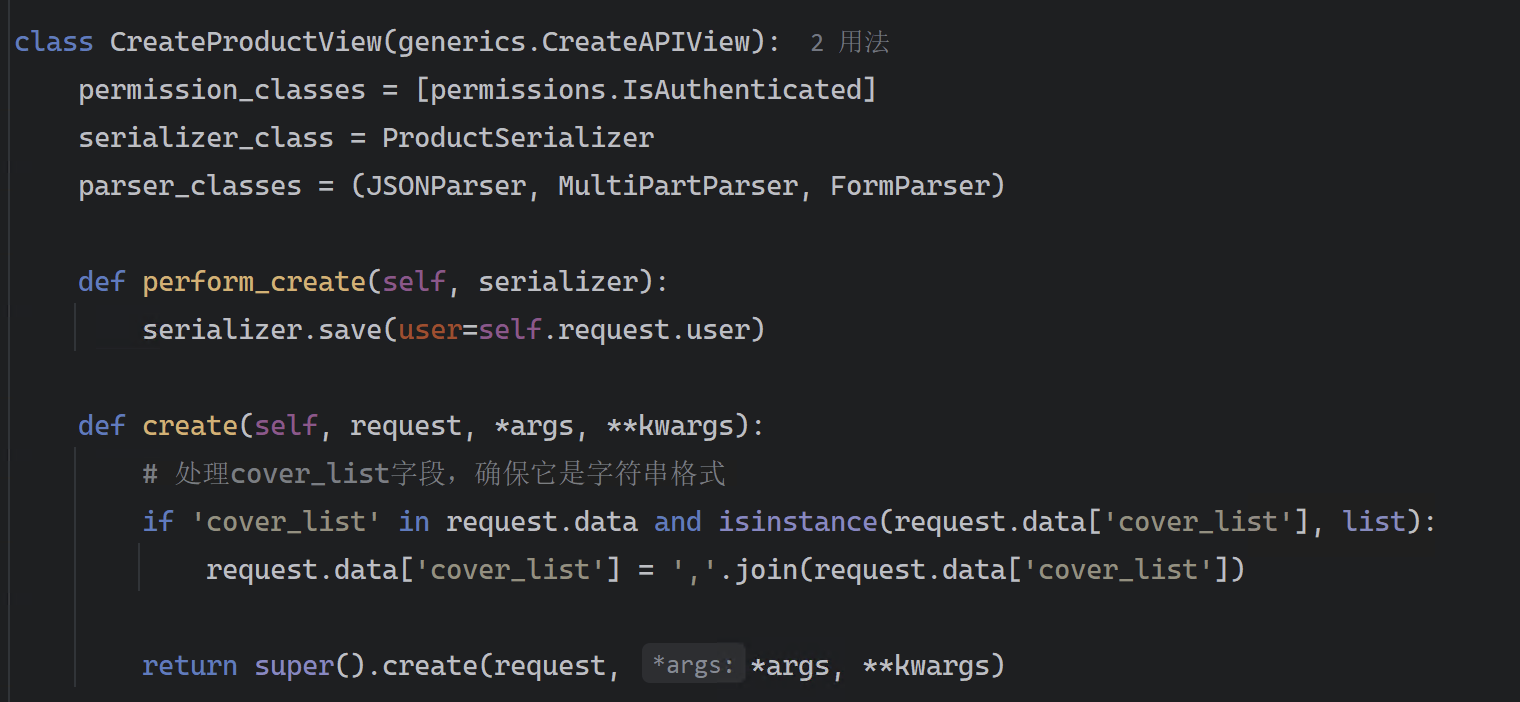


4.密码修改。



商品管理系统：

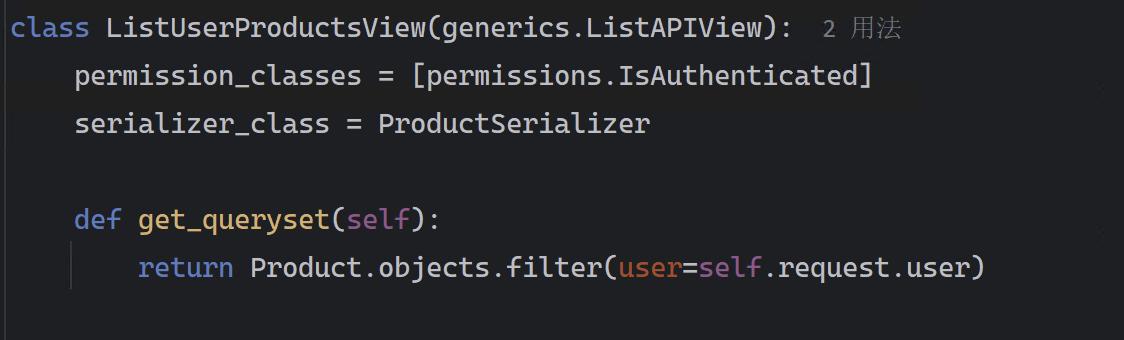
5.创建商品



6.商品图片上传

class UploadProductImageView(APIView):  
 *"""商品图片上传接口 - 支持多图上传"""* permission\_classes = [permissions.IsAuthenticated]  
 parser\_classes = (MultiPartParser, FormParser)  
  
 def post(self, request, \*args, \*\*kwargs):  
 if **'images[]'** not in request.FILES:  
 return Response(  
 {**'error'**: **'请选择要上传的图片文件'**},   
 status=status.HTTP\_400\_BAD\_REQUEST  
 )  
  
 # 获取所有上传的图片  
 images = request.FILES.getlist(**'images[]'**)  
   
 # 验证图片数量  
 if len(images) > 10:  
 return Response(  
 {**'error'**: **'一次最多只能上传10张图片'**},   
 status=status.HTTP\_400\_BAD\_REQUEST  
 )  
  
 # 允许的文件类型  
 allowed\_types = [**'image/jpeg'**, **'image/png'**, **'image/gif'**, **'image/webp'**]  
 # 存储上传成功的图片URL  
 uploaded\_urls = []  
 # 存储上传失败的图片信息  
 failed\_uploads = []  
  
 for image in images:  
 try:  
 # 验证文件类型  
 if image.content\_type not in allowed\_types:  
 failed\_uploads.append({  
 **'name'**: image.name,  
 **'error'**: **'不支持的图片格式。支持的格式：JPG, PNG, GIF, WEBP'** })  
 continue  
   
 # 验证文件大小（限制为5MB）  
 if image.size > 5 \* 1024 \* 1024:  
 failed\_uploads.append({  
 **'name'**: image.name,  
 **'error'**: **'图片大小不能超过5MB'** })  
 continue  
  
 # 生成文件保存路径  
 current\_date = datetime.now().strftime(**'%Y%m%d'**)  
 filename = **f"**{current\_date}**\_**{image.name}**"** relative\_path = os.path.join(**'products'**, str(request.user.id), filename)  
   
 # 确保目录存在  
 full\_path = os.path.join(settings.MEDIA\_ROOT, **'products'**, str(request.user.id))  
 os.makedirs(full\_path, exist\_ok=True)  
   
 # 保存文件  
 full\_file\_path = os.path.join(settings.MEDIA\_ROOT, relative\_path)  
 with open(full\_file\_path, **'wb+'**) as destination:  
 for chunk in image.chunks():  
 destination.write(chunk)  
   
 # 生成访问URL并添加到成功列表  
 image\_url = request.build\_absolute\_uri(settings.MEDIA\_URL + relative\_path)  
 uploaded\_urls.append(image\_url)  
  
 except Exception as e:  
 failed\_uploads.append({  
 **'name'**: image.name,  
 **'error'**: str(e)  
 })  
  
 # 构建响应数据  
 response\_data = {  
 **'message'**: **f'成功上传** {len(uploaded\_urls)} **张图片'** +   
 (**f'，**{len(failed\_uploads)} **张上传失败'** if failed\_uploads else **''**),  
 **'urls'**: uploaded\_urls,  
 }  
   
 # 如果有上传失败的图片，添加到响应中  
 if failed\_uploads:  
 response\_data[**'failed'**] = failed\_uploads  
  
 # 如果所有图片都上传失败，返回400状态码  
 if not uploaded\_urls and failed\_uploads:  
 return Response(response\_data, status=status.HTTP\_400\_BAD\_REQUEST)  
   
 return Response(response\_data)

7.获取用户发布的商品列表



8.获取所有商品列表（支持多种过滤和搜索）

class ListAllProductsView(generics.ListAPIView):  
 *"""获取所有用户发布的商品列表，支持搜索和过滤"""* permission\_classes = [permissions.AllowAny]  
 serializer\_class = ProductSerializer  
   
 def get\_queryset(self):  
 queryset = Product.objects.all().order\_by(**'-create\_at'**)  
   
 # 搜索商品名称  
 search = self.request.query\_params.get(**'search'**, None)  
 if search:  
 queryset = queryset.filter(name\_\_icontains=search)  
   
 # 按分类过滤  
 category = self.request.query\_params.get(**'category'**, None)  
 if category and category.isdigit():  
 queryset = queryset.filter(category\_id=int(category))  
   
 # 按价格范围过滤  
 min\_price = self.request.query\_params.get(**'min\_price'**, None)  
 max\_price = self.request.query\_params.get(**'max\_price'**, None)  
 if min\_price and min\_price.replace(**'.'**, **''**).isdigit():  
 queryset = queryset.filter(price\_\_gte=float(min\_price))  
 if max\_price and max\_price.replace(**'.'**, **''**).isdigit():  
 queryset = queryset.filter(price\_\_lte=float(max\_price))  
   
 # 按是否可议价过滤  
 is\_bargain = self.request.query\_params.get(**'is\_bargain'**, None)  
 if is\_bargain is not None:  
 is\_bargain = is\_bargain.lower() == **'true'** queryset = queryset.filter(is\_bargain=is\_bargain)  
   
 # 按新旧程度过滤  
 min\_old\_level = self.request.query\_params.get(**'min\_old\_level'**, None)  
 max\_old\_level = self.request.query\_params.get(**'max\_old\_level'**, None)  
 if min\_old\_level and min\_old\_level.isdigit():  
 queryset = queryset.filter(old\_level\_\_gte=int(min\_old\_level))  
 if max\_old\_level and max\_old\_level.isdigit():  
 queryset = queryset.filter(old\_level\_\_lte=int(max\_old\_level))  
   
 return queryset

9.获取商品分类

