



**数据库系统实现报告**

**--电商交易系统**

学号：18373447 姓名：张文彬

学号：18373441 姓名：覃启浩

学号：姓名：

2020年12月

## 一、系统概述

### 1.1 系统设计背景

互联网应用的普及方便了大众的日常生活，网络购物就是随互联网发展浪潮而形成的一个具有极大体量的新市场。2010年中国网络购物市场交易规模接近5000亿，达4980.0亿元，占到社会消费品零售总额的3.2%；同时，网络购物用户规模达到1.48亿，在网民中的渗透率达30.8%。近年来，网络购物市场体量进一步扩大，今年年初爆发的全球新冠疫情，网络购物因其无人员接触的优势获得了进一步的发展。因此我们小组选择实现一个小型的电商交易系统，其简单、功能全面，容易部署的特性在校园、小区这样的小范围场景内具有较高的应用空间。

### 1.2 系统开发环境

(1) Django 3.1.2

(2) MySql 8.0

(3) pyecharts 1.9.0

### 1.3 系统设计目标

本项目为小型电商交易系统，能够帮助用户在小范围场景内实现商品的展示、交易。

## 二、系统功能结构

### 2.1. 系统功能概述

系统整体功能实现分为前台和后台，前后台模型独立但可交互，后台为商户应用模块，主要进行以商品为核心的相关信息管理，前台为用户应用模块，主要进行以交易为核心的相关信息交互管理。

2.2 后台功能概述

后台交互主要以商品为核心，其主要的功能结构如图一，后台功能主要可分为五个模块，分别为商户信息管理，商品管理，订单管理，前后台信息传递以及报表绘制。

商户信息管理主要针对商户的注册登录、信息修改、相关数据如优惠券信息的查看。

商品管理为后台应用的核心，商户可以定制并发布新商品，更改商品信息，调整商品的上下架参数来决定商品是否在前台显示，为商品设置优惠券，管理商品的类别。

订单管理主要用来显示商户对应的用户订单信息，商户在收到用户的下单请求后，可以选择为用户发货，催促用户评价等操作来推进订单流程。

前后台信息传递为商户联系用户的重要途径，当商户为用户订单发货，发布优惠信息时，会通过系统生成消息来对用户进行通知。

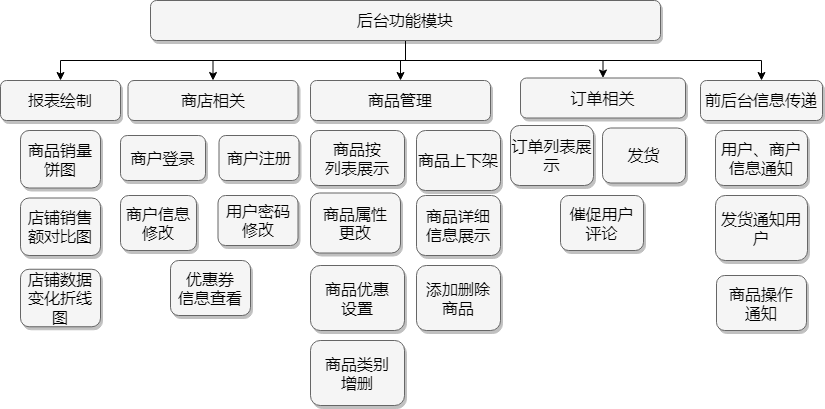


图 1 后台功能结构

### 2.3 前台功能概述

前台交互主要以交易为核心，用户通过搜索、浏览感兴趣的商品信息，将想要下单的商品添加到购物车，并进行下单操作，开启交易流程。前台功能主要有用户相关，商品展示，商户展示，订单操作，前后台信息传递这五个主要模块。如图二：

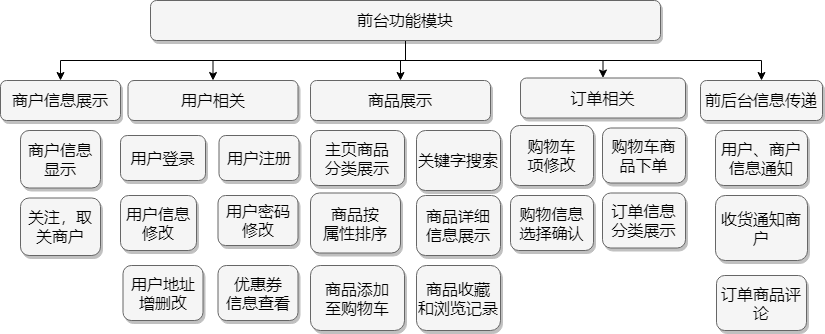


图 2 前台功能结构

用户相关为前台对应的用户模型的信息管理，同样具有注册、登录、信息更改等功能，除了基本的信息，用户还可以管理自己的地址信息，用于完善在下单时的地址选择。

商品展示为前台功能的核心，商户发布并上架的商品将会在前台显示，用户可以通过在搜索栏输入关键字搜索感兴趣的商品，也可以通过主页的分类栏获取某一类别的商品，主页同时也会推送一些商品供用户浏览。在用户发现合适的商品时，可以通过点击商品图片进入商品的详细界面，在商品详细界面，用户可以浏览商品的详细信息以及其他用户购买后的评价，选择是否将商品添加到购物车或是收藏该商品。用户在浏览商品时，系统会自动记录，并存储相关的浏览历史。

由于商品和商家的对应关系，用户也可以选择访问商品的对应的商店界面，来探索该店发布的其他商品，并通过关注店铺来获取该店铺第一时间的详细信息。

订单相关与后台的功能类似，用户在下单成功，商户发货后，根据自身情况选择是否已经收到货品，并在收货后选择对订单中的商品进行评价，推进订单流程。

前后台的信息传递是用户联系商户的方式，用户在完成一些动作时，例如商品的关注、商店的关注、商品的评论时，都会通知店铺。

## 三、后端实现细节

整个电子交易系统分为前端和后端两部分，后端管理数据库，并针对前端的 请求进行数据的发送，前端负责设计数据的呈现形式，并将用户和前端交互过程中必要的数据传输到后端。

### 3.1 基本表的定义

#### 后台数据表：



















#### 前台数据表：













### 3.2 系统安全性设计

整个电商交易系统分为前台和后台两部分，其中系统用户的权限共分三级，有权限从高到低依次为系统管理员、商户、用户，其中系统管理员能够管理整个系统的所有商户和用户，并对他们下属的信息进行编辑和删除。商户能够对系统中的商品类别、商品、评论以及优惠券进行管理。用户只能够访问商品数据，对自己的属性进行修改，而不能更改其他数据。

除了三级制的用户级别之外，在系统中的部分界面增加访问控制，避免越界访问。

### 3.3 触发器设计说明

触发器能够加强数据的完整性约束和业务规则，在数据库设计中，基于系统实现的完整性和稳定性，在如下场景使用触发器来实现部分功能。

#### 3.3.1 商品评分更新

商品在被用户评价时，用户会为该商品填写评分，评分作为商品的一个属性，可以在每次产生新评论时更新它。

1. CREATE TRIGGER  product\_score\_update
2. AFTER INSERT ON shop\_comment
3. FOR EACH ROW BEGIN
4. UPDATE shop\_product SET score =
5. (SELECT AVG(score) FROM shop\_comment
6. WHERE product\_id = new.product\_id GROUP BY product\_id)
7. WHERE id = new.product\_id;
8. END$$

#### 3.3.2 商品评分检查

商品评分因其特殊性，因此将值约束在0到10之内，对范围之外的评分，将修改评分值。

1. CREATE TRIGGER comment\_score\_check BEFORE UPDATE ON shop\_comment
2. FOR EACH ROW BEGIN
3. IF new.score < 0 THEN
4. SET new.score = 0;
5. END IF;
6. IF new.score > 10 THEN
7. SET new.score = 10;
8. END IF;
9. END$$

#### 3.3.3 订单状态检查

系统在运行过程中，订单模型的状态使用int类型整数标记，需要检查订单状态值保证订单处于合法状态。

1. CREATE TRIGGER order\_status\_check BEFORE UPDATE ON user\_order
2. FOR EACH ROW BEGIN
3. IF new.order\_status > 3 OR new.order\_status < 0 THEN
4. SIGNAL SQLSTATE 'TX000' SET MESSAGE\_TEXT = 'order status error';
5. END IF;
6. END$$

#### 3.3.4 商店销量更新

用户下单成功时将支付给商户订单总额，此时新生成的订单中的商品金额将折算到商店的销售额当中。

1. CREATE TRIGGER shop\_sales\_update AFTER INSERT ON user\_orderitem
2. FOR EACH ROW BEGIN
3. DECLARE new\_sales **double**;
4. DECLARE old\_sales **double**;
5. SET old\_sales = (SELECT total\_price FROM shop\_shop WHERE id = new.shop\_id);
6. SET new\_sales = old\_sales + new.total\_price;
7. UPDATE shop\_shop SET sales = new\_sales WHERE id = new.shop\_id;
8. END$$

### 3.4 存储过程简述

在Django框架中，数据表对应的实体按照属性对应到一个实体模型中，实体的插入、更改、删除都被包装成一个独立的方法，在方法中，Django框架会负责与数据库实现沟通，也即数据表中的每一个表项都可以用一个独立的python对象来表示，最终对表项的增删改就转化为了对python对象的操作。

### 3.5 实现技术论述

项目采用Django框架进行系统实现，Django框架将一部分数据库操作语言包装，使得开发者对数据库进行操作时，可以通过包装完毕的python语句进行操作，减少了一部分数据库操作流程。

#### 3.5.1 访问控制

系统为了防止有用户在未登录的情况下访问数据，对一些函数进行上锁，每当前端访问一个有访问控制的页面时，都会重定向到登录界面。为实现这一功能，系统会在用户登录时在网页端新建一个Cookie项，用以存储用户的登录信息，被访问控制约束的页面在访问时会执行loginValid函数，检查是否存在存储用户登录信息的Cookie，若不存在则重定向。

1. **def** loginValid(fun):
2. **def** inner(request,\*args,\*\*kwargs):
3. c\_user = request.COOKIES.get("user\_username")
4. **if** c\_user:
5. **return** fun(request, \*args, \*\*kwargs)
6. **else**:
7. **return** HttpResponseRedirect("/login/")
8. **return** inner

#### 3.5.2 信息传输

前端在请求访问一个具体的url时，会向后端发送请求并调用该url对应的一个函数，后端执行该函数，完成对数据库的查询，并将前端需要的数据发送出去，前端需要json格式的数据，而在Django框架下，访问数据库得到的结果为QuerySet类型，因此还需要将结果序列化并转化成json传递给前端。

如下面的例子：

1. **return** render(request,"user/product.html",{
2. 'favorited' : favorited,
3. 'shop\_followed' : shop\_followed,
4. 'on\_sale' : on\_sale,
5. 'product' : product,
6. 'comment\_list': mark\_safe(serializers.serialize('json',comment\_list)),
7. 'comment\_name\_list': mark\_safe(json.dumps(comment\_name\_list)),
8. 'comment\_image\_list' : mark\_safe(json.dumps(comment\_image\_list)),
10. })

#### 3.5.3 图表绘制

项目需要能够展示系统运行的这段时间中数据变化情况，需要通过统计图的形式显示给用户。通过pyecharts工具生成动态图表，并显示到前端。

如下面绘制店铺商品的销量饼图：

1. shop\_id = request.COOKIES.get('shop\_registered')
2. plist = Product.objects.filter(shopId\_id=shop\_id).order\_by('-sales')
3. name\_list = []
4. sale\_list = []
5. **if** len(plist) == 0:
6. **return** render(request,'shop/sales.html',locals())
7. **elif** len(plist) <= 10:
8. **for** p **in** plist:
9. name\_list.append(p.name)
10. sale\_list.append(p.sales)
11. **else**:
12. **for** i **in** range(0,9):
13. name\_list.append(plist.first().name)
14. sale\_list.append(plist.first().sales)
15. plist = plist.exclude(id=plist.first().id)
16. sale = 0
17. **for** i **in** plist:
18. sale += i.sales
19. name\_list.append('其他')
20. sale\_list.append(sale)
21. c = (
22. Pie()
23. .add("", [list(z) **for** z **in** zip(name\_list, sale\_list)])
24. .set\_global\_opts(
25. title\_opts=opts.TitleOpts(title="商品销量展示"),
26. legend\_opts=opts.LegendOpts(type\_="scroll",pos\_top="20%", pos\_left="80%", orient="vertical"),
28. toolbox\_opts=opts.ToolboxOpts(is\_show=True,pos\_top="top",pos\_left="right",),
29. )
30. .set\_series\_opts(label\_opts=opts.LabelOpts(formatter="{b}: {c}"))
31. .render("./templates/shop/sales.html")
32. )
33. **return** render(request,'shop/sales.html',locals())

#### 3.5.4 图片信息的存储

图片信息由于自身的存储方式的区别，不能直接存入数据库进行查询，因此，在系统实现过程中，对于图片的存储策略是，对于上传到系统的图片，将其存到项目内的静态文件区，并将其在静态存储区的路径存入数据库中，对图片的引用将通过访问数据库中的对应路径来实现。

#### 3.5.5 路由设置

前端通过访问对应路由，发送请求至后端，执行该路由对应的方法，并通过该方法获得对应的数据，以渲染前端页面，或获得json格式的返回结果。

项目路由共设三级，分别对应系统的三级用户——系统管理员、商户、用户，每一级路由下设分路由，分别对应每一级下的操作。

1. urlpatterns = [
2. path('admin/', admin.site.urls),
3. path('shop/', include("shop.urls")),
4. path('', include("user.urls")),
5. ]

## 四、前端实现细节

### 4.1 数据交互

前后端之间的数据交互通过Ajax传递，数据的格式是Json。部分页面直接使用Django框架进行模板渲染。

如下为数据交互过程中的一例JSON数据，是前端在渲染购物车页面时获取的购物车项信息。

1. {
2. "model":"user.cartitem",
3. "pk":56,
4. "fields":{
5. "cart\_user":9,
6. "product\_id":43,
7. "product\_name":"晨光中性笔笔芯",
8. "product\_price":18.5,
9. "product\_number":1,
10. "total\_price":18.5,
11. "shop\_id":8
12. }
13. }

向后端发送数据时，通过Ajax的POST方法或者使用表单来提交数据到后端进行处理。例如如下在前端进行的登录处理。

1. $.ajax({
2. type: "POST",
3. url: "/login/",
4. data: {
5. username: $("#username").val(),
6. password: $("#password").val(),
7. },
8. dataType: "json",
9. success: **function** (result) {
10. console.log(result);
11. **if** (result.data.status == "success") {
12. window.open("/index/", "\_self");
13. } **else** {
14. prompt.setAttribute("class", "alert alert-danger");
15. prompt.innerHTML = (result.data.data);
16. }
17. },
18. error: **function** (result) {
20. },
21. )

### 4.2 动态生成页面

前端使用了HTML中的DOM方法和Jquery来动态生成网页元素。在从后端获取到了数据之后，就可以把数据显示在网页上了。利用DOM中的方法可以创建元素并给元素添加属性，使用Jquery中的方法可以将元素添加到指定位置上。

### 4.3 实现分页和排序功能

实现分页和排序功能是一个难点，我们通过Javascript函数实现。在搜索商品的界面，我们实现了分页和排序功能，可以根据价格和销量对搜索到的商品进行排序。前端首先获取后端的数据，保存在本地变量中。当切换排序方式时，会根据不同的排序函数对数据进行排序。排完序之后，需要将排好序的数据分组，一页需要展示多少商品，一组就有几个商品数据。当切换页面时，获取当前的页面数，然后展示相应页面的商品。

1. //价格升序比较函数
2. **var** asc = **function** (x, y) {
3. **return** x.fields.price - y.fields.price;
4. }
5. //价格降序比较函数
6. **var** desc = **function** (x, y) {
7. **return** y.fields.price - x.fields.price;
8. }
9. //销量降序比较函数
10. **var** saleDesc = **function** (x, y) {
11. **return** y.fields.sales - x.fields.sales;
12. }
13. //上一页
14. **function** prevPage() {
15. **if** (nowPage > 1) {
16. nowPage--;
17. let nowActiveLi = $(".pagination").children(".active");
18. nowActiveLi.removeClass("active");
19. nowActiveLi.prev().addClass("active");
20. loadGoods(splitGoodList[nowPage - 1]);//展示nowPage页面的商品
21. }
22. }
24. //下一页
25. **function** nextPage();
27. //跳转到某一页
28. **function** thisPage(a);

### 4.4 组件使用

使用Bootstrap中的组件进行前端设计，用户体验较好。

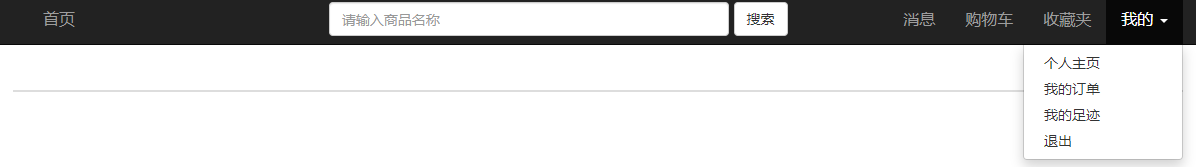


图 3 导航栏

模态框可以提供用户编辑数据或者提醒，独立于页面而存在，可以提供信息和交互。

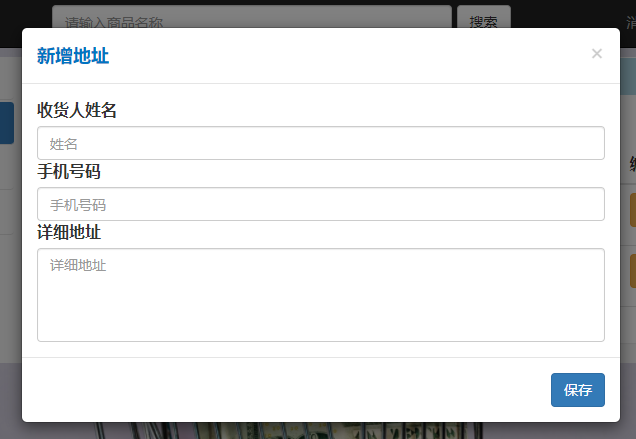


图 4 模态框

图片轮播可以作为广告位，让我们的页面看起来非常的精美。



图 5 图片轮播

## 五、系统运行实例

### 5.1 前台运行展示

图片展示如下

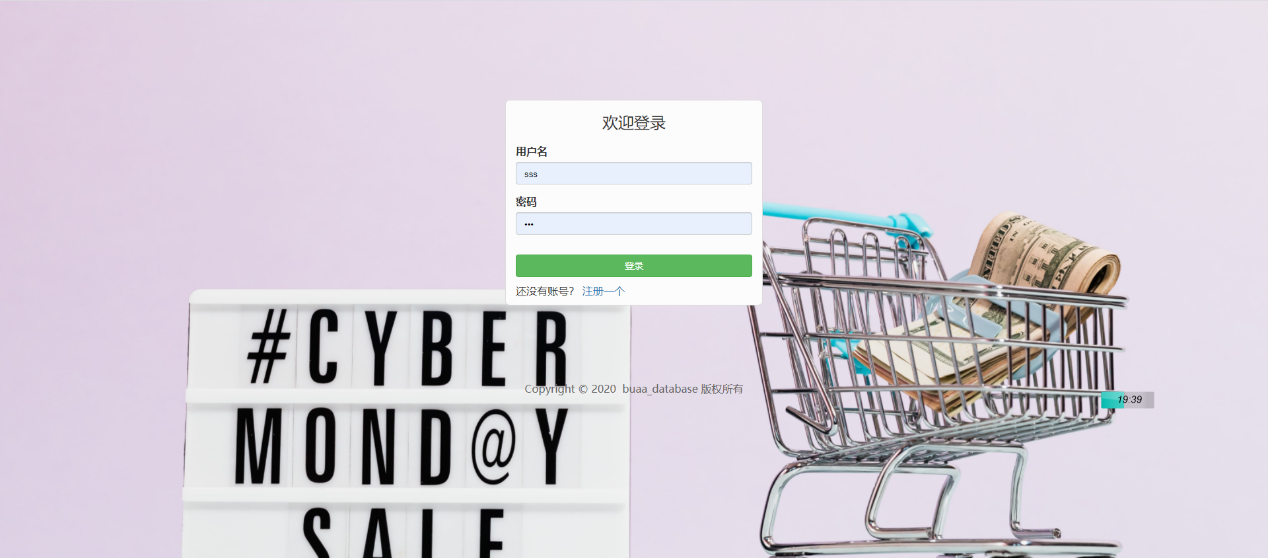


图 6 登录界面

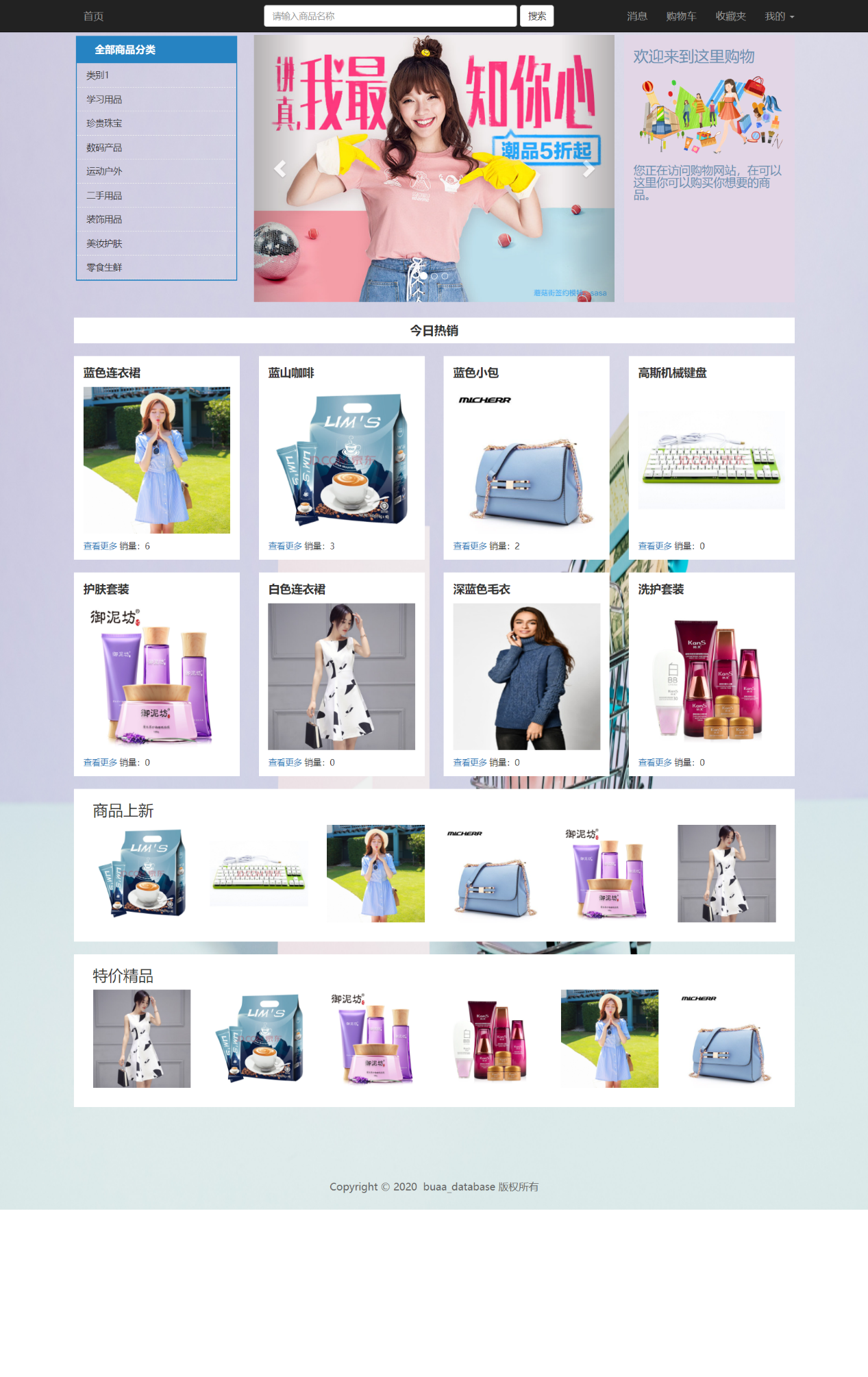


图 7 主页显示

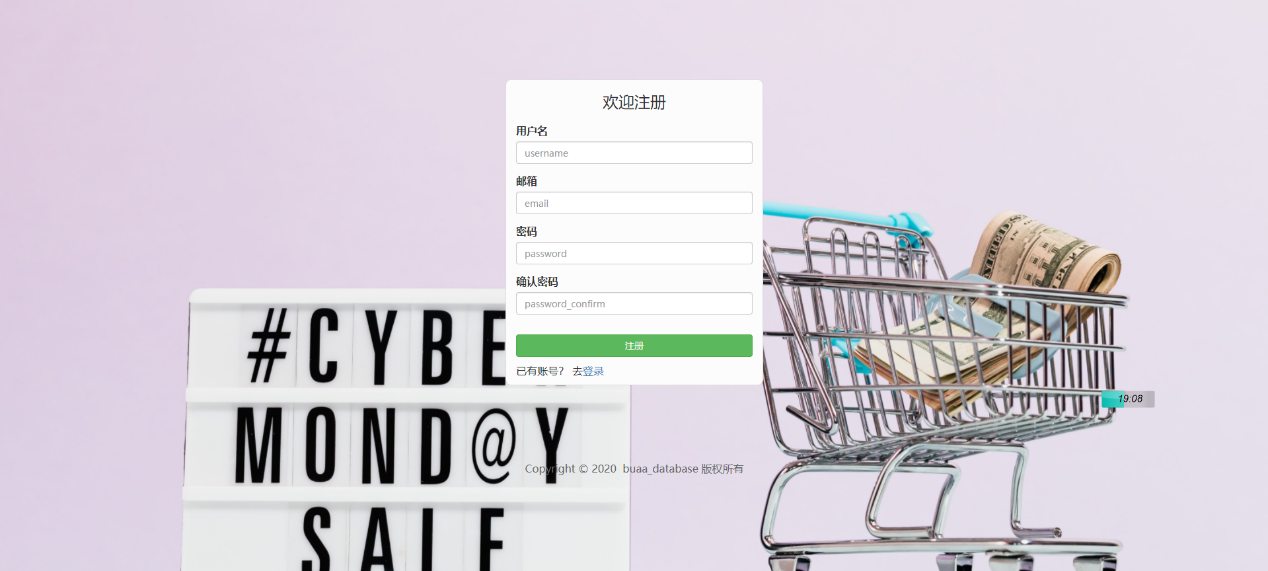


图 8 注册界面



图 9 商品主界面

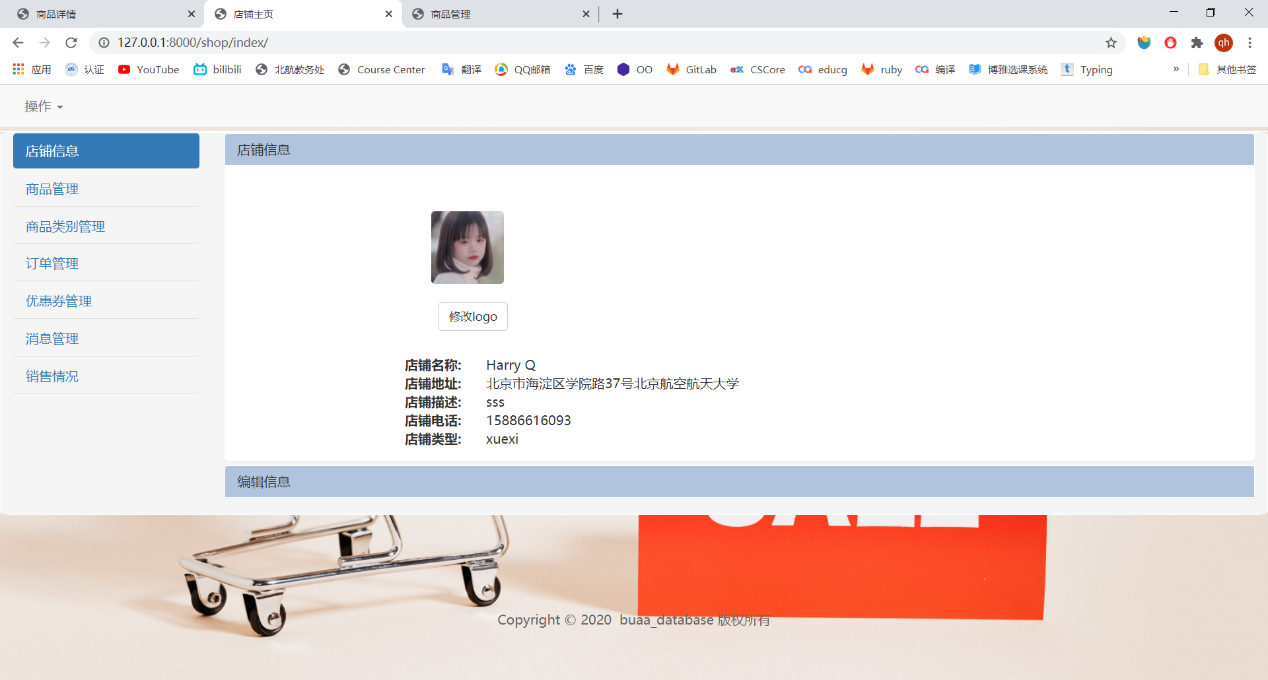


图 10 店铺主界面



图 11 用户评论



图 12 购物车界面



图 13 收藏夹

## 六、源程序简要说明

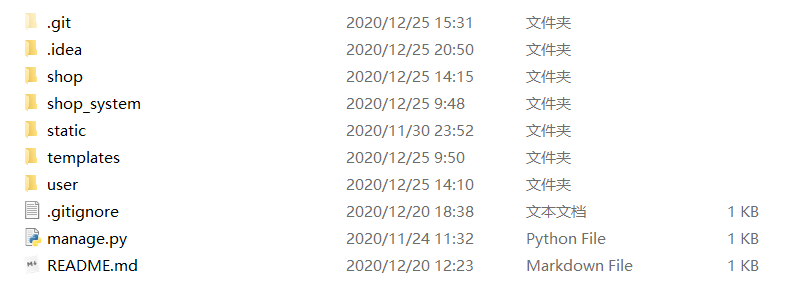


图 14 文件结构

如图14所显示的文件结构，manage.py文件为Django框架的启动文件，在部署好运行环境之后，在命令行下运行py manage.py runserver 命令，即可在本地浏览器上进行系统的操作。

其中shop\_system文件夹为项目的主框架，定义了系统的全局设置，路由设置。

user文件夹和shop文件夹分别对应系统中的前后台应用，包含了前后台的模型定义、路由定义、和方法定义。

static文件夹为静态资源文件夹，存放系统前端的样式、JS文件以及系统运行时需要的图片等。

templates文件夹存放前端html模板，系统通过渲染指定模板来实现系统运行效果。

在新环境中配置系统时，首先需要配置数据库连接，更改位于shop\_system/settings.py文件中的数据库连接信息，并执行py manage.py makemigrations 和py manage,py migrate命令进行数据库迁移，完成如上操作之后可通过运行py manage.py runserver 命令开始系统的使用。

系统前台和后台的url分别为 <http://localhost:8000/> 和<http://localhost:8000/shop/> ，在项目运行无误的情况下，通过在浏览器中访问如下地址即可进入系统。

## 七、感想与体会

18373447-张文彬

完成此次的数据库系统设计，我最大的感触就是体会到了团队合作的重要性，要在短时间内个人实现一个小型的电商交易系统并不容易，通过团队分工合作，下发任务，则可以减少个人任务量，并尽可能高质量的实现系统功能。

在本次数据库系统设计中，我主要负责项目设计、后端实现和前后端对接，任务量较大，为完成这些任务，我学习了Django框架相关知识、简单的JavaScript操作和数据库操作知识，虽然学习过程比较漫长，但通过系统的学习和调试，这些知识提高了我的工程能力和团队合作能力，对我有很大的帮助。在整个项目实现过程中，我也遇到了很多问题，这些问题的解决也提高了我解决问题的能力。

我和队友的分工比较明确，整个系统的开发流程是在讨论中不断迭代开发完成的，从构思，到设计，到开发，再到调试和优化，整个过程有序进行，对团队的提升很大。

数据库是信息在现实世界的一种载体，他的作用无疑是巨大的，通过本学期的数据库课程设计，学习并亲身设计和实现一个小型的数据库系统，这样的经历非常有意义，也能够帮助我们开发应用更广泛、使用价值更高的项目。

18373441-覃启浩

在这次数据库设计中，我负责的是前端的部分。

学习过程。为了完成这次实验，我需要学习很多新的东西，首先是html和css，我自学阅读玩了Head First HTML这本书，这大概花了断断续续一个月。为了实现页面逻辑，我需要学习JavaScript和Jquery，为了和后端交互，我还需要学习AJAX方法。为了让页面好看一点，我去学习框架和组件，在舍友的帮助下我了解到了Twitter开发的Bootstrap，这真的对我帮助很大。我学习了很多很多的新知识，这确实是一个快乐的过程。

分工明确很重要。由于对于前端和后端的不了解，我们开始都是各做各的，各学各的，也不去问对方做了什么工作，这样导致的一个后果是：对接的时候非常难受。前端不了解后端数据库，面对不断增加的需求显得不耐烦；后端把前端的工作一起做了，导致后端的工作量加大了。可以说，我们多花费了很多时间，而且各自都做得比较难受。这让我明白了一个道理，合作的时候一定要分工明确，一定要知道对方的工作，不能只顾自己的。

总结。说实话这个实验真的话费了我很多精力，我花了几乎数不清楚的时间来完成这实验，看着自己的系统变得日渐精美，还是有一点欣慰的。这应该是一次美好的体验。