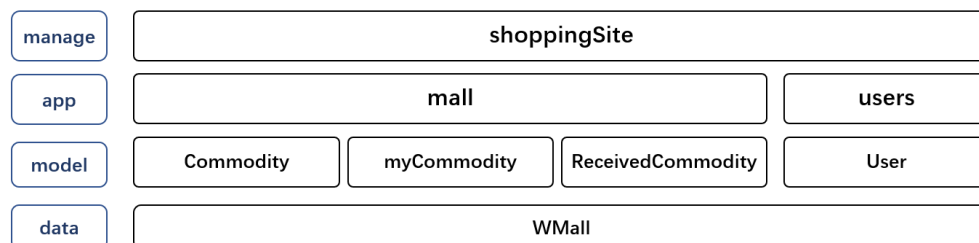


实现一个小型电子购物网站

一、设计思路

1) 系统架构:

架构图如下:



➤ 管理层（设置层）

主要是 manage.py、urls.py 以及 setting.py

- ✧ manage.py: 一个管理程序；可以接受命令并执行，如：运行服务器，数据库迁移等
- ✧ urls.py: 为不同 app 设置路由
- ✧ setting.py : 项目的设置程序，如：设置运行的 app、连接的数据库设置、邮件发送设置等。

➤ 应用层

- ✧ mall: 实现商城的功能：如购买、加入购物车等。
- ✧ users: 实现用户的登录、注册、注销功能。

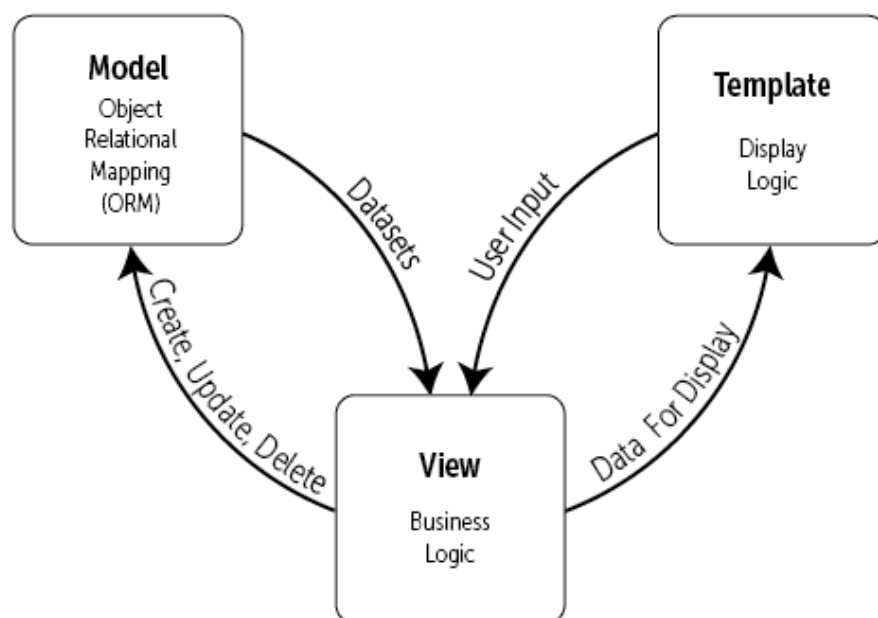
➤ 模型层

- ✧ mall:
 - Commodity: 商品
 - MyCommodity: 我的商品，即购物车的商品
 - ReceivedCommodity: 已买到的商品
- ✧ users: 使用 Django 自带的用户模型 User

➤ 数据层

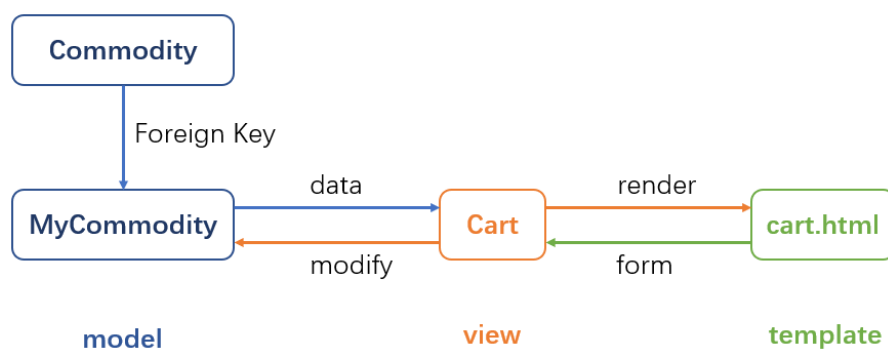
✧ 连接建立好的 postgres 数据库 Wmall

2) 设计框架：基于 MVC 的 MVT 模式（Django）



- **Model:** 与 MVC 中的 M 相同，负责对数据的处理
- **View:** 与 MVC 中的 C 类似，负责处理请求，进行逻辑处理（但是这个命名是 view 感觉很奇怪，Django 命名的一个遗憾）
- **Template:** 与 MVC 中的 V 类似，负责显示数据（html 页面）

以购物车页面为例，作图如下(便于理解)



二、代码实现

➤ 实现 mall app:

- 1) **实现游客限制访问:** 使用 Django 提供的装饰器@login_required 来限制游客对于购物车、订单记录页面的访问,
- 2) **普通用户权限限制:** 通过判断用户的权限级别从而实现区分管理员用户和普通用户, 以赋予他们不同的操作权限。
- 3) **使用隐藏的方法:** 对于游客, 直接隐藏加入购物车、购物车、订单管理等入口; 对于普通用户, 隐藏了商品管理、客户日志、销售日志的入口。隐藏了入口, 也就不存在游客、普通用户可以获取更高操作权限的可能, 也就不存在非法操作。
- 4) **model:**

a) Commodity:

构建代码如下: 这是商品模型, 共有六个属性

```
class Commodity(models.Model):
    # 商品图片; 名称; 描述; 价格; 销量; 上市日期
    photo = models.ImageField(upload_to='mall/images')
    name = models.CharField(max_length=50)
    description = models.TextField(default='物美价廉')
    price = models.FloatField(default=9)
    sales = models.PositiveIntegerField(default=0)
    date_added = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.name
```

b) MyCommodity

代码如下: 这是购物车的商品模型。其中商品种类使用了外键, 引用了模型 Commodity, 以便获取商品信息, 同时可以实现: 某种商品下架后, 购物车中的此商品也跟着被移除。拥有者 owner 也是用了外键,

引用模型 User，可根据此属性为不同用户归类所拥有的购物车商品。

同时加了一个计算总价的函数，便于计算多个相同商品时的总价。

```
class MyCommodity(models.Model):
    # 购物车中商品的 数量; 商品种类; 拥有者; 加入时间
    amount = models.PositiveIntegerField(default=1)
    goods = models.ForeignKey(Commodity, on_delete=models.CASCADE)
    owner = models.ForeignKey(User, on_delete=models.CASCADE)
    date_added = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.goods.name

    # 计算该商品的总价
    def total_cost(self):
        return self.amount * self.goods.price
```

c) ReceivedCommodity

代码如下：这里是已购买的商品模型。区别于购物车中的商品模型

MyCommodity，这里对于商品的种类不能使用外键，因为已经购买了的商品记录不能消失。若使用了外键，当此商品下架后，该购买记录也消失了，这是不可行的。所以此处用了，名称、价格、数量、照片来存储已购买的商品信息。

```

class ReceivedCommodity(models.Model):
    # 购买日期; 拥有者, 名字, 数量, 价格, 商品图片
    bought_date = models.DateTimeField(auto_now_add=True)
    # 每个用户有自己的订单查看
    owner = models.ForeignKey(User, on_delete=models.CASCADE)
    # 注意: 删除了 My Commodity 但是, 收到的不会消失, 所以不能用外键
    name = models.CharField(max_length=50)
    amount = models.PositiveIntegerField()
    price = models.FloatField()
    photo = models.ImageField(upload_to='mall/images')

    def __str__(self):
        return self.name

    # 计算总价
    def total_cost(self):
        return self.price * self.amount

```

5) view:

a) homepage: 浏览主页, 并实现加入购物车功能

代码如下: 所用的 model: Commodity、MyCommodity。展示商品逻辑较为简单, 获取所有商品信息传入前端即可。至于添加商品至购物车: 用户按下“加入购物车”发送‘POST’请求, 并传给后端一个商品的 id, 遍历购物车: 有此商品则数量加一, 无则新建一个购物车商品。

```

def homepage(request):
    if request.method == 'POST':
        # 点击了“加入购物车”后的操作
        gid = request.POST['gid'] # 获取所加商品的 id
        # 获取所有的购物车商品
        mys = MyCommodity.objects.filter(owner=request.user).order_by('date_added')
        flag = 0
        # 遍历购物车商品
        for m in mys:
            # 若存在，商品数量加一
            if m.goods.id == eval(gid):
                m.amount += 1
                m.save()
                flag += 1
        # 不存在，新建 MyCommodity 实体存储该信息
        if flag == 0:
            new_my = MyCommodity(amount=1,
                                   goods=Commodity.objects.get(id=gid),
                                   owner=request.user)
            new_my.save()
        # 获取所有商品，后渲染
        goods = Commodity.objects.all()
        context = {'goods': goods}
        return render(request, 'mall/home.html', context)

```

b) cart: 浏览购物车，并实现移除功能

代码如下：所用模型：MyCommodity。展示购物车商品逻辑：获取所有该用户的购物车商品信息，传给前端即可。移除购物车商品：根据得到的购物车商品 id，遍历购物车商品，数量为一则删去该实体，否则该购物车商品数量减一。

```

@login_required()
def cart(request):
    """ 显示用户的购物车 """
    if request.method == 'POST':
        # 实现购物车商品的移除（数量减一）
        remove = request.POST['remove']
        my_id = request.POST['my_id'] # 获取移除的购物车商品 id
        if eval(remove) == 1:
            temp = MyCommodity.objects.get(id=my_id)
            # 数量为一，直接移除；否则减一
            if temp.amount == 1:
                temp.delete()
            else:
                temp.amount -= 1
                temp.save()
        # 获取所有购物车商品
        my_cart = MyCommodity.objects.filter(owner=request.user).order_by('date_added')
        context = {'my_cart': my_cart}
        return render(request, 'mall/cart.html', context)

```

c) purchased: 购买商品的逻辑

代码较长，以下给出核心代码：购买单个商品为例

```

# 邮件内容清单
text = '您所购买的宝贝清单：\n'
email = request.POST['email']
wanted_id = request.POST.get('wanted_id')
if wanted_id:
    # 根据得到想要的商品信息
    wanted = MyCommodity.objects.get(id=wanted_id)
    text += f'{wanted.goods.name} '
    text += f'数量: {wanted.amount} '
    text += f'总价: ¥{wanted.total_cost()} \n'
    # 加入已购买的宝贝
    new_received = ReceivedCommodity(name=wanted.goods.name, amount=wanted.amount, price=wanted.goods.price,
                                     photo=wanted.goods.photo)
    new_received.owner = request.user
    new_received.save()
    # 给此商品加 销量
    com = Commodity.objects.get(id=wanted.goods.id)
    com.sales += wanted.amount
    com.save()
    # 从购物车移出此商品
    wanted.delete()

```

用户在购物车页面点击“购买”某商品时会传给后端该商品的 id，

待用户输入邮箱后即可购买。将商品的名称、数量、总价加入发送信息中。购买商品有三个要注意的操作：首先是要将该商品信息加入已购买的宝贝，其次是要修改此商品的销量，最后是要把购物车里的此商品实体删除。

购买所有商品的逻辑也是类似，在外围增加一个遍历即可。至于发送邮件，使用了 `django.core.mail` 的 `send_mail` 函数，使用了 QQ 邮箱的 SMTP 服务器（需要在 `manage` 层中的 `setting.py` 中设置，代码如下）

```
text += '欢迎下次使用，祝您生活愉快!!!' + '\n🍀🍀🍀🍀🍀🍀'
send_mail(
    subject='感谢使用WMall，您所购买的商品已送至门口，请及时取件',
    message=text,
    from_email='2396469068@qq.com', # 用于发送的邮箱
    recipient_list=[fr'{email}'], # 用户所输入的邮箱
    fail_silently=False
)
```

发送邮箱的代码

```
# Email setting
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_USE_TLS = False # 是否使用TLS安全传输协议(用于在两个通信应用程序之间提供保密性和数据完整性。)
EMAIL_USE_SSL = True # 是否使用SSL加密，qq企业邮箱要求使用
EMAIL_HOST = 'smtp.qq.com' # 发送邮件的邮箱的SMTP服务器，这里用了qq邮箱
EMAIL_PORT = 465 # 发件箱的SMTP服务器端口
```

`setting.py` 中的设置

d) received: 浏览已购买的订单

代码如下：根据用户筛选已购买的商品信息，传入给前端即可

```
@login_required()
def received(request):
    # 根据用户筛选已购买的商品
    all_received = ReceivedCommodity.objects.filter(owner=request.user).order_by('bought_date')
    context = {'all_received': all_received}
    return render(request, 'mall/received.html', context)
```

e) management: 浏览全部商品，并实现商品移除功能（管理员）

展示所有商品：获取所有商品信息，并传给前端

删除商品：得到 'POST' 请求，得到商品 id 筛选删除即可


```

        else: # 删除商品
            cd_id = request.POST.get('cd_id')
            if cd_id:
                temp = Commodity.objects.get(id=cd_id)
                temp.delete()

    # 得到所有商品
    commodities = Commodity.objects.all()
    context = {'commodities': commodities}
    return render(request, 'mall/management.html', context)

```

f) `add_commodity`: 增加商品，逻辑详见注释

```

@login_required()
def add_commodity(request):
    if request.method == 'POST':
        # 得到商品信息
        form = CommodityForm(request.POST, request.FILES)
        if form.is_valid():
            form.save()
            # 保存该商品并重定向到 商品管理页面
            return HttpResponseRedirect(reverse('mall:management'))
    else:
        # 不是 'POST' 请求则提供表单以便获取信息
        form = CommodityForm()
        context = {'form': form}
        return render(request, 'mall/add_commodity.html', context)

```

g) `modify_commodity`: 修改商品，逻辑详见注释

值得一提的是：这里的 view 多了一个参数，`cd_id` (`commodity_id`)，因为对于修改每个商品的页面，路由是这种形式的：以便于每种商品的修改时都有自己的路由，同时传送待修改的商品的 id。

```
path('modify_commodity/<int:cd_id>')
```

```

@login_required()
def modify_commodity(request, cd_id):
    # 根据 id 得到所需修改的商品
    commodity = Commodity.objects.get(id=cd_id)
    if request.method == 'POST':
        # 得到修改后的信息
        form = CommodityForm(instance=commodity, data=request.POST)
        if form.is_valid():
            form.save()
            # 保存该商品的新信息并重定向到 商品管理页面
            return HttpResponseRedirect(reverse('mall:management'))
    else:
        # 提供表单以供修改
        form = CommodityForm(instance=commodity)
        context = {'form': form, 'commodity': commodity}
        return render(request, 'mall/modify_commodity.html', context)

```

h) sales_statistics: 销售表单

这里考验的主要是对数据的处理能力,或者说是对于数据结构的灵活运用能力: 首先得到所有的 ReceivedCommodity 实体, 由于用户的不同, 所有同一种商品有许多的此实体, 要根据商品来归类成销售报表, 采用字典这个数据机构, 键值对是{str:list}, 用一个 list 来存储商品的多个信息。至于如何计算商品的总销量和总销售额, 详见代码。

```

@login_required()
def sales_statistics(request):
    all_received = ReceivedCommodity.objects.all()
    # 不同用户买的商品应该归类到一起
    all_sale = {} # { '商品名字', [商品价格, 商品数量, 商品总价] }
    # 单价 ; 销量 ; 总价
    for a in all_received:
        if a.name in all_sale:
            all_sale[f'{a.name}'][0] = a.price # 单价
            all_sale[f'{a.name}'][1] += a.amount # 销量
            all_sale[f'{a.name}'][2] += a.price * a.amount # 总价
        else:
            temp = {f'{a.name}': [a.price, a.amount, a.price * a.amount]}
            all_sale.update(temp)
    context = {'all_sale': all_sale}
    return render(request, 'mall/sales_statistics.html', context)

```

i) user_statistics: 客户日志（代码较长，此处展示核心代码）

难点与销售表单类似，故采用相似的字典结构：{str:list}，str 存放用户 id，list 存放用户的订单信息，至于用户的购物车记录实现方法也是类似。

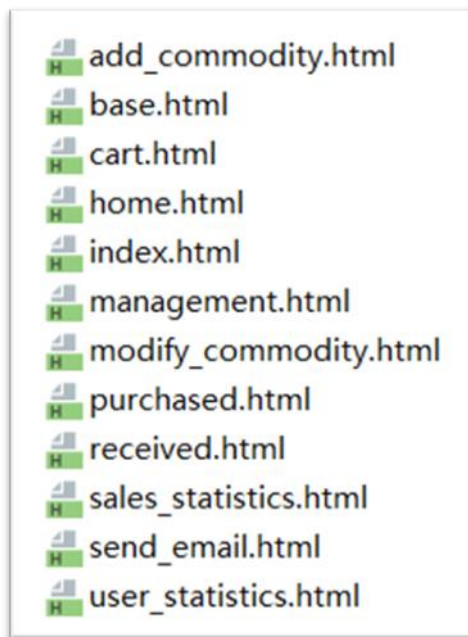
```

@login_required()
def user_statistics(request):
    # 订单记录
    received_c = ReceivedCommodity.objects.all().order_by('bought_date')
    indent = {}
    for r in received_c:
        if r.owner.email:
            continue # 忽略管理员用户
        elif r.owner in indent:
            # 已经有此用户，加一列购买记录
            indent[r.owner].append([r.owner.username, r.name, r.price, r.amount, r.total_cost(), r.bought_date])
        else:
            temp = {r.owner: [[r.owner.username, r.name, r.price, r.amount, r.total_cost(), r.bought_date]]}
            indent.update(temp)

```

6) template:

使用了 bootstrap 框架，减少了 css 与 js 的代码，减轻了前端开发的压力，文件列表如下：其中有些页面用于提示：如下单成功等，之前并未提及。而所有 html 继承了 base.html，实现导航条统一。以下抽出较为核心的代码进行说明



- 导航条 (base.html)：判断权限以提供对于入口
若是游客：仅提供登录、注册的选项

```
{% else %}  
  <li class="nav-item">  
    <a class="nav-link" href="{% url 'users:login' %}">登录</a>  
  </li>  
  <li class="nav-item">  
    <a class="nav-link" href="{% url 'users:register' %}">注册</a>  
  </li>
```

若是普通用户：提供：你好！用户，以及购物车、订单信息

```
{% if user.is_authenticated %}  
  <li class="nav-item">  
    <a class="nav-link" href="#">欢迎, {{ user.username }}! </a>  
  </li>  
  <li class="nav-item">  
    <a class="nav-link" href="{% url 'mall:cart' %}">购物车</a>  
  </li>  
  <li class="nav-item">  
    <a class="nav-link" href="{% url 'mall:received' %}">已买到的宝贝</a>  
  </li>
```

若是管理员，在普通用户的基础上进一步判断，提供一个下拉菜单：商品管理、销售记录、客户日志

```
{% if user.email %}
<li class="nav-item dropdown">
  <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button"
    data-toggle="dropdown" aria-expanded="false">
    管理员中心
  </a>
  <div class="dropdown-menu" aria-labelledby="navbarDropdown">
    <a class="dropdown-item" href="{% url 'mall:management' %}">商品管理</a>
    <a class="dropdown-item" href="{% url 'mall:sales_statistics' %}">销售统计</a>
    <a class="dropdown-item" href="{% url 'mall:user_statistics' %}">客户日志</a>
  </div>
</li>
```

其余 html 代码在此不展开解释，详见 GitHub 源码

➤ 实现 users app:

1) model:

使用 Django 内置的 User 模型，共有 12 个属性，主要使用了 username、password 这两个属性

2) view: 共三个：登录、注册、注销

a) 注册:

```
def register(request):
    # 如果不是 POST 请求，生成内置表单 UserCreationForm
    if request.method != 'POST':
        form = UserCreationForm()
    else:
        # 使用内置表单 UserCreationForm 得到注册信息
        form = UserCreationForm(data=request.POST)
        if form.is_valid():
            # 保存新用户信息，并授权登录
            new_user = form.save()
            authenticate_user = authenticate(username=new_user.username,
                                              password=request.POST['password1'])
            login(request, authenticate_user)
            # 登录后重定向至商品主页
            return HttpResponseRedirect(reverse('mall:home'))
    context = {'form': form}
    return render(request, 'users/register.html', context)
```

实现逻辑:

- i. 根据表单获取数据新建用户并授权登录，并重定向至商品主页面；

ii. 若不是 POST 请求，渲染 register.html，传输一个

UserCreationForm 让使用者可以输入注册相关信息。

b) 登录、注销：

使用了 Django 内置的 LoginView、LogoutView，调用实现

```
from django.contrib.auth.views import LoginView, LogoutView
```

3) **template:** 登录、注册、注销

登录、注册：（图片以登录为例），主要内容是表单 form，利用 bootstrap 快速生成的表单样式，简洁又美观

```
<form method="post" action="{% url 'users:login' %}">
  {% csrf_token %}
  {# bootstrap 格式的表单，快速生成 #}
  {% bootstrap_form form %}

  {% buttons %}
    <button name="login" class="btn btn-outline-primary">log in</button>
  {% endbuttons %}

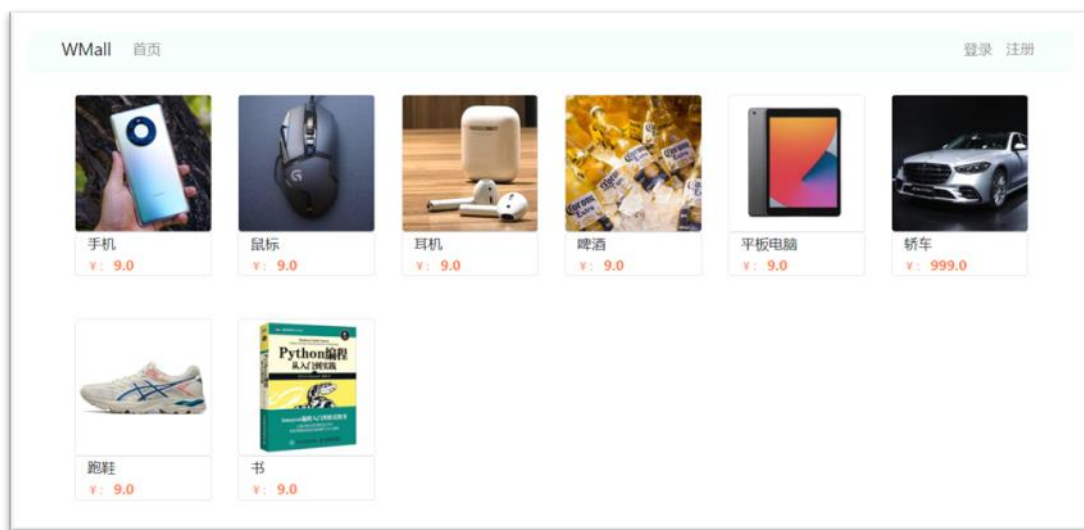
  <input type="hidden" name="next" value="{% url 'mall:home' %}" />
</form>
```

注销：使用 bootstrap 的巨幕，显示“成功注销”，无需提供表单

三、运行结果

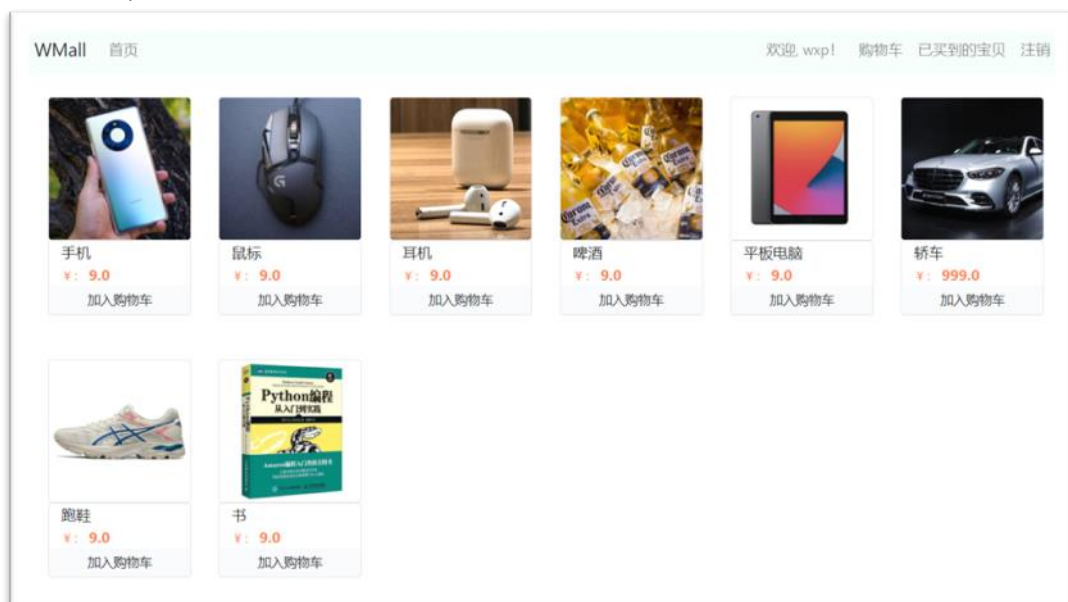
游客：

浏览商品主页：



普通用户:

1) 加入购物车




WMall [首页](#)

[欢迎 wxp!](#) [购物车](#) [已买到的宝贝](#) [注销](#)

wxp 的购物车

图片展示	商品名称	单价	数量	金额	操作
	鼠标	¥: 9	1	¥: 9	移除 购买
	耳机	¥: 9	1	¥: 9	移除 购买
	啤酒	¥: 9	4	¥: 36	移除 购买

2) 购买商品：购买一个鼠标，收到邮件确认收货

WMall 首页		欢迎 wxp! 购物车 已买到的宝贝 注销			
付款清单					
	鼠标	¥: 9 × 1 = 9			
共计 ¥:9.0					
邮箱:					
<input type="text" value="interlink_@outlook.com"/>					
输入邮箱以追踪商品物流信息					
<input type="button" value="付款"/>					

感谢使用WMall，您所购买的商品已送至门口，请及时取件	
2	2396469068@qq.com
周二 2021/12/14 11:02	
收件人: 你	
您所购买的宝贝清单:	
鼠标 数量: 1 总价: ¥9.0	
欢迎下次使用，祝您生活愉快!!!	
	
<input type="button" value="谢谢!"/>	<input type="button" value="了解了，谢谢!"/>
<input type="button" value="真可爱!"/>	

3) 查看购买订单

WMall [首页](#)

欢迎, wxp! [购物车](#) [已买到的宝贝](#) [注销](#)

wxp 已购买的宝贝


图片展示	商品名称	单价	数量	金额	购买日期
	鼠标	¥: 9	1	¥: 9	2021-12-14 03:02:39

管理员:

1) 管理商品

➤ 添加商品: ‘测试’商品

WMall 首页		欢迎, william! 购物车 已买到的宝贝 管理员中心 注销							
添加一个新的商品									
Photo									
<input type="button" value="选择文件"/> pink.jpg									
Name									
<input type="text" value="测试"/>									
Description									
<input type="text" value="物美价廉"/>									
Price									
<input type="text" value="9"/>									
Sales									
<input type="text" value="0"/>									
<input type="button" value="添加"/>									

	测试	¥: 9	2021-12-14 03:14:03	<input type="button" value="修改"/>	<input type="button" value="删除"/>
---	----	------	---------------------	-----------------------------------	-----------------------------------

➤ 修改商品: 商品 ‘测试’ 修改为 ‘测试之修改测试’

WMall 首页 欢迎, william! 购物车 已买到的宝贝 管理中心 注销

修改 测试 的相关属性

Photo

Currently: mall/images/pink.jpg

Change:

选择文件 未选择文件

Name

测试之修改测试

Description

物美价廉


Price

9.0

Sales

0

确认修改



测试之修改测试

¥: 9

2021-12-14 03:14:03

修改

删除

- 移除商品：移除成功
- 2) 查看销售报表

WMall 首页 欢迎, william! 购物车 已买到的宝贝 管理中心 注销

销售统计报表

商品名	单价	销售量	销售总额
跑鞋	9	4	36
手机	9	2	18
耳机	9	4	36
书	9	8	72
轿车	999	5	4995
啤酒	9	2	18
平板电脑	9	3	27
鼠标	9	3	27

3) 查看用户日志：购买记录、购物车记录

WMall	首页	欢迎, william!	购物车	已买到的宝贝	管理中心 ▾	注销
订单记录						
isudfv 的订单						
商品名	单价	数量	金额	购买日期		
跑鞋	9	1	9	2021-12-13 06:45:21		
手机	9	1	9	2021-12-13 06:45:21		
耳机	9	1	9	2021-12-13 06:45:21		
书	9	3	27	2021-12-13 06:45:21		

购物车记录				
wxp 的购物车				
商品名	单价	数量	金额	加入日期
耳机	9	1	9	2021-12-13 15:42:23
啤酒	9	4	36	2021-12-13 15:42:24
平板电脑	9	1	9	2021-12-13 15:42:25
轿车	999	1	999	2021-12-13 15:42:25
跑鞋	9	1	9	2021-12-13 15:42:26
书	9	1	9	2021-12-13 15:42:27
梦 的购物车				
商品名	单价	数量	金额	加入日期
耳机	9	1	9	2021-12-13 15:43:59

参考文献

- [1] the django book : <https://djangobook.com/mdj2-django-structure/>
- [2] django doc : <https://djangobook.com/mdj2-django-structure/>
- [3] bootstrap doc : <https://v4.bootcss.com/docs/getting-started/introduction/>
- [4] 《Python 编程：从入门到实践》【美】Eric Matthes