浙江大学

本科实验报告

课程名称：	B/S 体系结构
姓　　名：	黄锦亮
学　　院：	计算机学院
系：	软件工程
专　　业：	软件工程
学　　号：	3150102289
指导教师：	胡晓军

2019 年　6　月　15　日

浙江大学实验报告

课程名称：B/S 体系结构 实验类型：设计

实验项目名称：旧书交易网站的实现

学生姓名：黄锦亮 专业：软件工程 学号：3150102289

同组学生姓名： 指导老师：胡晓军

实验地点： 实验日期：2019 年 6 月 5 日

目录

1、概述

1.1、实验目的

任选一种技术实现一个旧书交易的网站

1.2、实验要求

需要实现的基本功能如下：

1、 实现用户注册、登录功能，用户注册时需要填写必要的信息并验证，如用户名、密码要求在 6 字节以上，email 的格式验证，并保证用户名和 email 在系统中唯一。

2、 用户登录后可以发布要交易的书籍，需要编辑相关信息，包括书名、原价、出售价、类别和内容介绍等信息、外观照片等，可以通过 ISBN 和书名链接到外部系统（如 Amazon/京东/当当等网站）的详细介绍页面。

3、 根据用户发布的书籍聚合生成首页，可以分类检索。

4、 用户可以设置交易模式为寄送还是线下交易，生成订单时录入不同内容。

5、 集成一个消息系统，买家和卖家之间可以通信。

6、 提供求购模块，用户可以发布自己想要的书籍。

7、 界面样式需要适配 PC 和手机的浏览器。

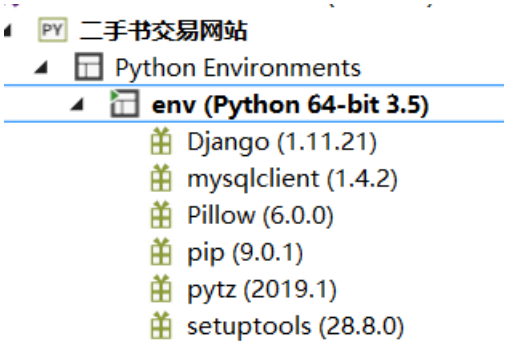
增强功能：

8、 实现一个 Android 或 iphone 客户端软件，功能同网站，额外支持定位功能，发布时记录位置，可以根据用户的位置匹配最近的待售书籍。消息和订单支持推送。

为了提交作业方便，如有数据库，建议使用 mysql 或 mangodb，提交作业时同时附带 SQL 脚本文件。

1.3、 系统配置与环境

Python 3.5
Django 1.11.21
Mysql
VS2015



1.4、数据库设计

```
class SellBook(models.Model):

    category = models.CharField(max_length = 10, default = "")#种类
    title = models.CharField(max_length = 40, default = "")#书名
    press = models.CharField(max_length = 30, default = "")#出版色
    author = models.CharField(max_length = 20, default = "")
    price = models.CharField(max_length = 20, default = "")#元价格
    newprice = models.CharField(max_length = 20, default = "")#二手价
    total = models.PositiveSmallIntegerField(default = 0)#库存
    stock = models.PositiveSmallIntegerField(default = 0)#现货量
    content  = models.CharField(max_length = 100, default = "")#简介
    image = models.ImageField(upload_to='avatar/')#图片
    seller = models.CharField(max_length = 40, default = "")
    isSelling = models.CharField(max_length = 40, default = "")#书是否卖完
    ISBN  = models.CharField(max_length = 40, default = "")#ISBN号

class customer(models.Model):
    user = models.OneToOneField(User, on_delete = models.CASCADE)

class Order(models.Model):
      seller = models.CharField(max_length = 40, default = "")#卖家
      buyer = models.CharField(max_length = 40, default = "")#买家
      buyNumber = models.CharField(max_length = 40, default = "")  #购买量
      style = models.CharField(max_length = 10, default = "")#交易方式
      address = models.CharField(max_length = 40, default = "")#送货地址
      iphone = models.CharField(max_length = 40, default = "")#买家联系电话
      book_id = models.CharField(max_length = 40, default = "")#购买的书的编号
#卡号用username

#密码用password
class wantBook(models.Model):
    title = models.CharField(max_length = 40, default = "")#书名
    press = models.CharField(max_length = 30, default = "")#出版色
    author = models.CharField(max_length = 20, default = "")
    total = models.PositiveSmallIntegerField(default = 0)#求购数量
    wanter = models.CharField(max_length = 40, default = "")
    image = models.ImageField(upload_to='ava/', default = "")#图片


class Message(models.Model):
   sender = models.CharField(max_length = 40, default = "")
   receiver = models.CharField(max_length = 40, default = "")
   msg = models.CharField(max_length = 100, default = "")
```

数据库由 models.py 文件进行定义

2、具体功能实现及页面设计

2.1、登陆

主要功能：进入网站主页面后，输入账号和密码来登陆进入用户界面。

具体设计：由于使用 Django 来完成这个项目，所以登陆模块使用了 django 原生的用户认证，在登陆时，只需要将网页上传来的账号与密码与数据库中的用户数据进行匹配，就能够完成判断是否存在该账号，然后选择登陆或是返回登陆界面。

页面设计：

函数实现：

```
def userlogin(request):

    if request.method == 'POST':

            username = request.POST['username']
            password = request.POST['password']
            #对比输入的用户名和密码和数据库中是否一致
            user = authenticate(username=username, password=password)

            if user is not None:
                if user.is_active:
                    login(request, user)
                    response = HttpResponseRedirect('/index/')
                    response.set_cookie('cookie_username', username)

                    return response
                else:
                    return render(request, 'login.html')
            else:

                 return HttpResponse("<script >alert('用户名密码错误');window.location.href='/login';</script>")



    else:

        return render(request, 'login.html')
```

流程：

1、 获取从网页传来的请求
2、 如果是 POST，则进入登陆模块
3、 判断用户名和密码是否与数据库中一致
4、 若一致则登陆，不然返回首页

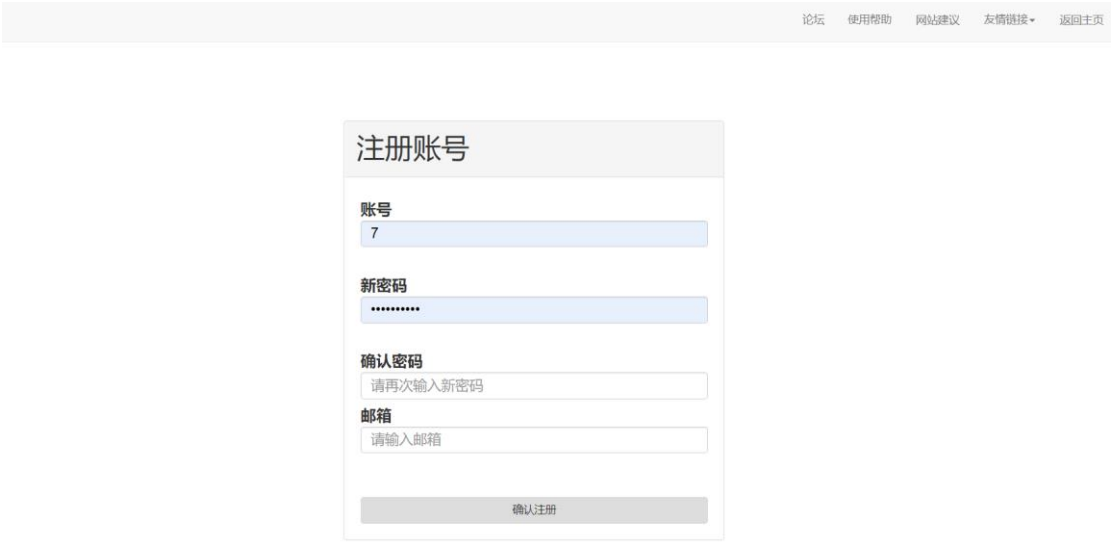
2.2、注册用户

主要功能：在网站主页面上，有注册账号的按钮来进入注册用户界面，用户名不能与已有用户名相同，且长度位于 6-18 位之间，只能包含数字字母下划线和@。用户邮箱号需符合命名规则，且唯一。

具体实现：注册用户的功能与登陆一样，也基于 Django 的登陆认证系统。首

先，接收到页面传来的注册信息后，网页上的 JS 会先行判断用户名是否符合长度位于 6-18 位之间，只能包含数字字母下划线和@这一要求，如果符合，在服务器上会再次判断用户名是否已经存在。如果存在，则返回信息该用户已经存在，如果不存在，则判断用户的邮箱是否已经存在。当两者同时满足了唯一的要求后，在邮箱进行正则表达式匹配，判断其是否符合命名标准。如果满足以上所有要求，就在用户数据库中添加账号，注册成功。

页面设计：



JS 实现：

```
<script type="text/javascript">
    function logout()
    {
        location.href="../../../../../logout/"
    }
     function check() {
        var password = document.getElementById("inputID").value
        var reg = /^[A-Za-z0-9_@]{6,18}$/;
        if (reg.test(password) == false) {
            alert('用户名密码输入不规范，长度在6-18之间，只能包含字符、数字、下划线和@');

        }

    }
   function check_check() {
        var password = document.getElementById("inputPassword").value
        var reg=/^[A-Za-z0-9_@]{6,18}$/;
        if (reg.test(password) == false) {
            alert('用户名密码输入不规范，长度在6-18之间，只能包含字符、数字、下划线和@');

        }

    }
</script>
```

流程：

1、 获取前端的输入字段

2、 进行正则匹配，看是否符合标准

函数实现：

```
def register(request):

    if request.method == 'GET':
        return render(request, 'register.html')

    if request.method == 'POST':
            new_username = request.POST['new_username']
            new_password = request.POST['new_password']
            check_password = request.POST['check_password']
            new_email = request.POST['new_email']
            if len(new_username)-6<0 :
                    return HttpResponse("<script >alert('用户名密码输入不规范，长度在6-18之间，只能包含字符、数字、下划线和@');window.loca
            if len(new_username)-18>0 :
                    return HttpResponse("<script >alert('用户名密码输入不规范，长度在6-18之间，只能包含字符、数字、下划线和@');window.loca
            check = re.match('^[a-z0-9A-Z]+[- | a-z0-9A-Z . _]+@([a-z0-9A-Z]+(-[a-z0-9A-Z]+)?\\.)+[a-z]{2,}$', new_email)
            if check:
                email = User.objects.filter(email = new_email)
                if len(email)>0:
                        return HttpResponse("<script >alert('邮箱已存在');window.location.href='/register';</script>")
                user_id = User.objects.filter(username = new_username)
                if len(user_id)>0:
                        return HttpResponse("<script >alert('账号已存在');window.location.href='/register';</script>")

                if check_password == new_password:
                    # 创建普通用户
                    User.objects.create_user(username=new_username, password=new_password, email = new_email)
                    return HttpResponse("<script >alert('注册成功');window.location.href='/register';</script>")
                else:
                    return HttpResponse("<script >alert('两次输入密码不同');window.location.href='/register';</script>")
            else:
                 return HttpResponse("<script >alert('邮箱格式不正确');window.location.href='/register';</script>")

    else:
            return render(request, 'register.html')
```

流程：

1、 判断网页传来的请求类别，如果是 GET，则跳转到注册界面，如果是 POST，则进入下一步

2、 获取用户名、密码、二次验证密码、邮箱

3、 对邮箱进行正则匹配，如果符合则进入下一步

4、 判断账号和邮箱是否在数据库中唯一

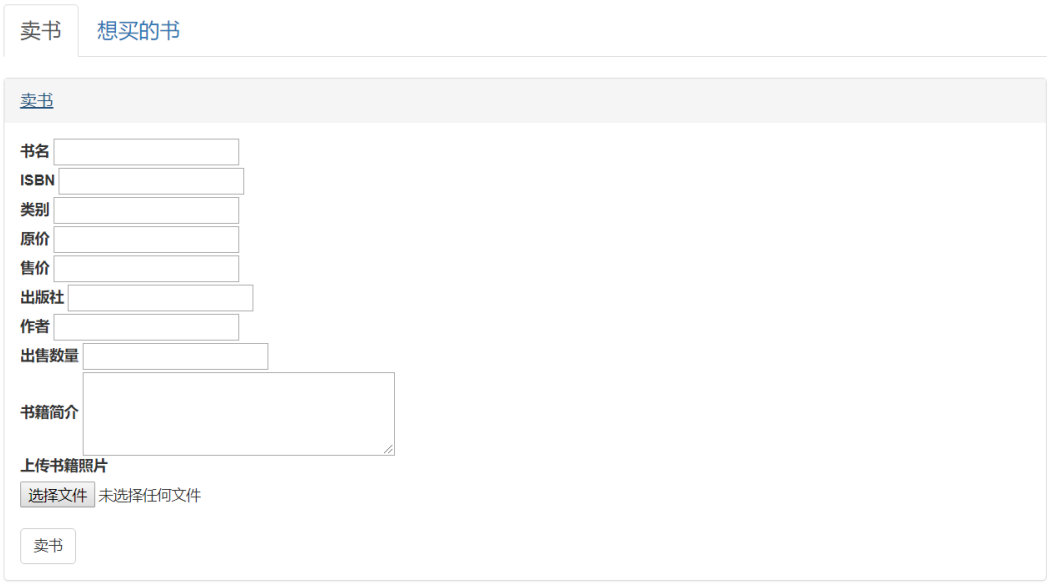
5、 创建用户

2.3、发布交易书籍

主要功能：用户可以在自己的个人主界面发布要交易的书籍，需要编辑相关信息，包括书名、原价、出售价、类别和内容介绍等信息、外观照片等，可以通过 ISBN 和书名链接到百度的详细介绍页面。

具体设计：在登陆旧书交易网站后，可以在个人界面发布自己想要出售的书籍。首先，在页面的表单上输入一系列有关自己发售的书籍的必要信息，包括书籍名称、ISBN、类别、原价格、售价、出版社、作者、出售数量、书籍简介，并且可以上传书籍照片。然后点击出售便可以发售旧书。当有输入信息位空时，页面会跳出提示要求输入完整信息。如果自己发售过该书，就会更新书籍的发售数目。

页面设计：



函数实现：

```
#卖书
def sellBook(request):
    username = request.COOKIES.get('cookie_username', '')
    ID = User.objects.filter(username = username)
    for i in ID:
        user_id = i.id
    if request.method == 'POST':
        bookname = request.POST['bname']
        category = request.POST['category']
        price = request.POST['price']
        newprice = request.POST['newprice']
        press = request.POST['press']
        author = request.POST['author']
        total = request.POST['total']
        content = request.POST['content']
        ISBN = request.POST['ISBN']
        image = request.FILES.get("logo", None)
        if bookname == "":
            return HttpResponse( "<script>alert('请填写书籍类别！');window.location.href='/user_homepage';</script>")
        if ISBN == "":
            return HttpResponse( "<script>alert('请填写ISBN号！');window.location.href='/user_homepage';</script>")
        if category == "":
            return HttpResponse( "<script>alert('请填写书籍类别！');window.location.href='/user_homepage';</script>")
        if bookname == "":
            return HttpResponse( "<script>alert('请填写书名！');window.location.href='/user_homepage';</script>")
        if press == "":
            return HttpResponse( "<script>alert('请填写出版社！');window.location.href='/user_homepage';</script>")
        if author == "":
            return HttpResponse( "<script>alert('请填写作者！');window.location.href='/user_homepage';</script>")
        if price == "":
            return HttpResponse( "<script>alert('请填写价格！');window.location.href='/user_homepage';</script>")
        if content == "":
```

```
    return HttpResponse( "<script>alert('请填写书籍简介！');window.location.href='/user_homepage';</script>")
if total == "":
    return HttpResponse( "<script>alert('请填写出售书籍数量！');window.location.href='/user_homepage';</script>")

ExistBname = SellBook.objects.filter(seller = user_id, title = bookname)#如果用户已经有该书籍出售
if ExistBname:

        Libtotal = SellBook.objects.filter(total = total)
        for T in Libtotal:
            booktotal = T.total

        stock  = SellBook.objects.filter(total = total)
        for S in stock:
            bookstock = S.stock
        if bookstock ==0 :
            SellBook.objects.filter(seller = user_id, title = bookname).update(isSelling = 1)
        booktotal = booktotal+ int(total)
        bookstock = bookstock + int(total)
        SellBook.objects.filter(seller = user_id, title = bookname).update(total = booktotal)
        SellBook.objects.filter(seller = user_id, title = bookname).update(stock = bookstock)
        return HttpResponse("<script>alert('发售成功！该书正在出售，对书本数量进行修改');window.location.href='/user_homepage';</script>")
else:
  SellBook.objects.create(ISBN = ISBN, category = category, title = bookname,press = press, author = author, price = price, newprice = ne
  return HttpResponse("<script>alert('发售成功！');window.location.href='/user_homepage';</script>")
```

流程：1、输入想卖的书籍的相关信息

2、判断所有需要的信息是否填写

3、判断该用户是否已经发售该书籍

4、如果有，则更新库存和总共数量

5、如果没有，则将该书进行发售

2.4、网站主页

主要功能：在网站主页上可以看到所有在售书籍，并且提供了分类检索功能，能够搜查自己想要的书籍是否在售。

具体设计：在旧书交易网站的主页面上，会显示所有已经发布的出售旧书信息。该功能由 html5 页面和功能函数共同实现。当进入网站主页面时，会向页面发送所有库存还不为 0 的书籍信息。该功能由 index 函数进行实现，具体代码可见源代码文件。

点击主要上的在售书籍查询，可以跳转到索引界面。

界面设计：

HTML 界面主要模块设计：

```
<div class="col-sm-8">
for book in selling_book %}
    <div class="col-sm-6">
     <div class="panel panel-default">
      <div class="panel-heading"><h2>在售书名: <a href="https://baike.baidu.com/item/{{ book.title }}">{{ book.title }}</a></h2></div>
        <div class="panel-body">
         <div>
              <img style="float:right" src="/media/{{book.image}}" height="240" width="180" alt=""/>
               <p style="font-size: 16px">卖家: {{ book.seller }} </p>
             <p style="font-size: 16px">类别: {{ book.category }} </p>
             <p style="font-size: 16px">出版社: {{ book.press }}</p>

             <p style="font-size: 16px">作者: {{ book.author }}</p>
             <p style="font-size: 16px">原价格: {{ book.price }}</p>
               <p style="font-size: 16px">现价: {{ book.newprice }}</p>
               <p style="font-size: 16px">内容简介: {{ book.content }}</p>
               <p style="font-size: 16px">ISBN: {{ book.ISBN }}</p>
               <p style="font-size: 16px">库存: {{ book.stock }}</p>
                 <form class="form-horizontal" action="../make_order/" method="post">
                     {% csrf_token %}
                     <button class="btn btn-dark" type="submit" name="type" value="{{book.book_id}}">下单</button>
                 </form>
         </div>
       </div>
     </div>
   </div>
endfor %}
/div>

div class="col-xs-6 col-sm-3"></div>
```

函数实现：

```
#主页
def index(request):
     username = request.COOKIES.get('cookie_username', '')
     selling_book = []
     SellingBook = SellBook.objects.filter( isSelling = 1)
     for i in SellingBook:
            category = i.category
            title = i.title
            press = i.press
            author = i.author
            price = i.price
            newprice = i.newprice
            stock = i.stock
            content = i.content
            image = i.image
            ISBN = i.ISBN
            seller_id = i.seller
            book_id = i.id
            username = User.objects.filter(id  = seller_id)
            for u in username:
                seller = u.username
            if stock>0:
                selling_book.append({'book_id':book_id,'seller':seller,'ISBN':ISBN, 'category':category ,'title':title,'press':press,'author':aut


     return render(request, 'index.html', {'username': username, 'selling_book':selling_book})
#下单
```

流程：

1、 当进入主页面时，从数据库中读取所有在售书籍

2、 获得这些书籍的信息

3、 判断书籍库存是否大于 0

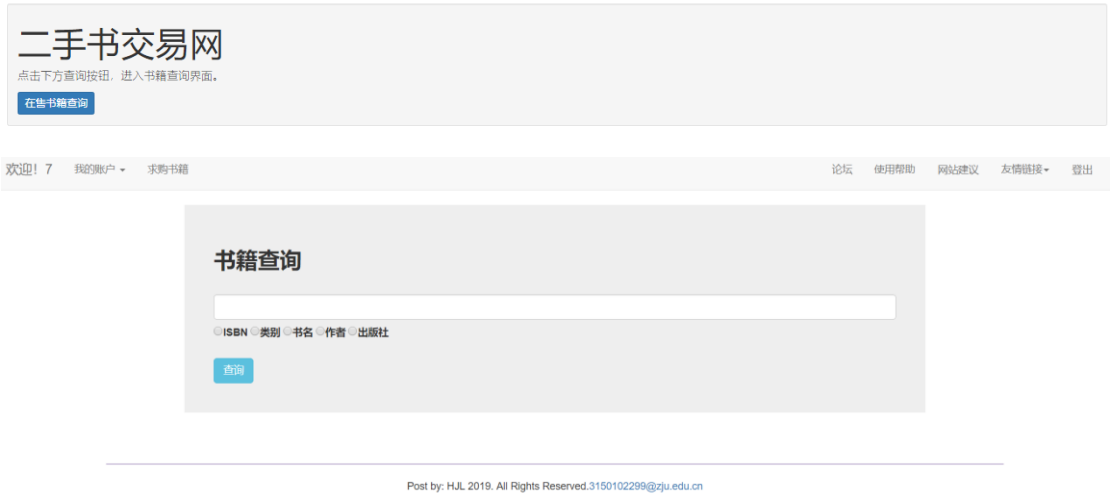
4、 通过 for 循环在页面上将信息进行展示

2.5、书籍索引

主要功能：输入相关的书籍信息，并且选择搜索信息的类别，就能够查找自己想要购买的书籍是否正在出售。

具体设计：当服务器接受到页面传来的信息后，会根据用户选择的类别，在出售书籍的数据库中进行匹配。如果存在该类别书籍，会将所有结果集成到一个搜索结果页面进行显示。可以在该界面下对搜索结果的书籍进行下单操作。

界面设计：



函数实现：

```
def query_result(request):
    username = request.COOKIES.get('cookie_username', '')
    if request.method =='POST':
        ret = []
        query_content = request.POST['content']

        query = request.POST['query']
        if(query == 'ISBN'):
            book = SellBook.objects.filter(ISBN = query_content);
            for b in book:
                if b.isSelling=='1' :
                    ISBN = b.ISBN
                    category = b.category
                    title = b.title
                    press = b.press
                    author = b.author
                    price = b.price
                    newprice = b.newprice
                    total = b.total
                    stock = b.stock
                    book_id = b.id
                    ret.append({'book_id':book_id,'ISBN':ISBN,'category':category,'title':title,'press':press,'author':author,'price':price,'newprice': newprice,'total':total,'s
            return render(request,'query_result.html',{'ret':ret})

        if(query == 'category'):
            category = SellBook.objects.filter(category = query_content);
```

流程：

1、 获取从前端传来的文本数据和类别数据

2、 判断查询的类别，跳入不同的 IF 语句

3、 在数据库中获取匹配的所有结果

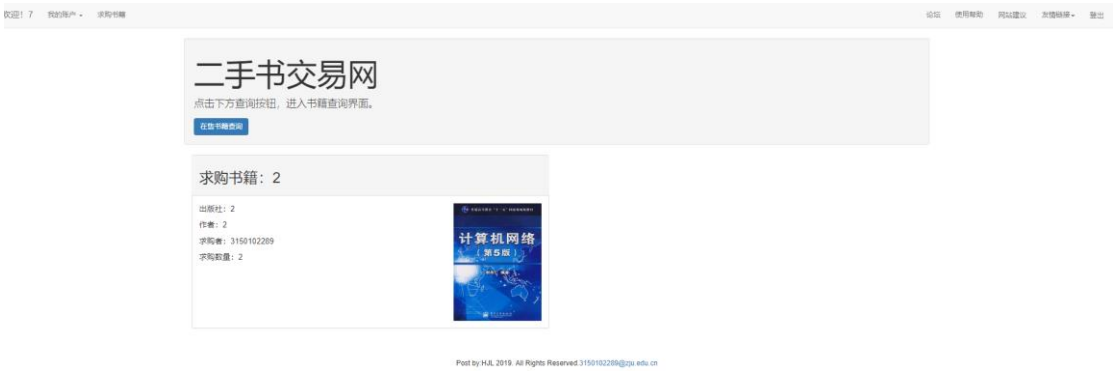
4、 将数据在查询结果页面上进行显示

2.6、求购书籍

主要功能：在自己的个人主页可以选择发布想要的书籍，并且提供基础信息。发布书籍后，在网站的求购版面，会显示用户所发布的书籍。

具体设计：原理同发布交易书籍一样，在输入了相关的求购书籍信息后，会将书籍信息存入数据库中。然后发布在求购书籍模块。点击用户界面的求购书籍模块就能够进入求购书籍界面，该界面上会在线显示所有用户已经发布的想要购买的书籍。该功能由 wanted_book 和 want_book 函数进行实现，具体代码可见源代码文件。

界面设计：



函数设计：

```
def want_book(request):
        username = request.COOKIES.get('cookie_username', '')
        ID = User.objects.filter(username = username)
        for i in ID:
            user_id = i.id
        if request.method =='POST':
            bookname = request.POST['bname']
            press = request.POST['press']
            total = request.POST['total']
            author = request.POST['author']
            image = request.FILES.get("logo", None)
            if bookname == "":
                return HttpResponse( "<script>alert('请填写书名！');window.location.href='/user_homepage';</script>")
            if press == "":
                return HttpResponse( "<script>alert('请填写出版社！');window.location.href='/user_homepage';</script>")
            if author == "":
                return HttpResponse( "<script>alert('请填写作者！');window.location.href='/user_homepage';</script>")
            if total == "":
                return HttpResponse( "<script>alert('请填写出售书籍数量！');window.location.href='/user_homepage';</script>")
            if image == "":
                return HttpResponse( "<script>alert('请填写出售书籍数量！');window.location.href='/user_homepage';</script>")
            ExistBook = SellBook.objects.filter(title = bookname, press = press, author = author)
            if ExistBook:
                return HttpResponse("<script>alert('该书已经发售，请前往购买！');window.location.href='/user_homepage';</script>")
            else:
                wantBook.objects.create(author = author,press = press, title = bookname, wanter = user_id,image = image,total = total)
                return HttpResponse("<script>alert('求购书籍已发布！');window.location.href='/user_homepage';</script>")
```

流程：

1、输入想买的书籍的相关信息，从前端读取信息

2、判断所有需要的信息是否填写

3、判断该书籍是否已经发售

4、如果有，则返回提醒“该书籍已经发售”

5、如果没有，则将对该求购书籍进行发布

```
def wanted_book(request):
        username = request.COOKIES.get('cookie_username', '')
        wanting_book = []
        wantbook = wantBook.objects.all()
        for w in wantbook:
          title = w.title
          image = w.image
          author = w.author
          press = w.press
          total = w.total
          wanter_id = w.wanter
          wanter =User.objects.filter(id = wanter_id)
          for t in wanter:
              buyer = t.username
          wanting_book.append({'wanter':buyer,'title':title, 'image':image,'author':author,'press':press,'total':total})
        return render(request, 'wanted_book.html', {'username': username, 'wanting_book':wanting_book})
#书籍索引
```

流程：

1、 当进入主页面时，从数据库中读取所有求购书籍

2、 获得这些书籍的信息

3、通过 for 循环在页面上将信息进行展示

2.7、个人主页

主要功能：显示自己已经发布出售的书籍和想要购买的书籍目录。可以在该页面选择出售书籍或者是发布自己想要购买的书籍。也能够前往书籍索引界面，搜索书籍。

具体设计：在进入个人主页时，服务器会向网页发送当前用户的所有在售书籍信息和求购书籍信息。网页上会在线显示书目。该功能由 user_homepage 函数进行实现，具体代码可见源代码文件。

界面设计：

函数实现：

```
#用户主页
def user_homepage(request):
    username = request.COOKIES.get('cookie_username', '')
    selling_book = []
    wanting_book = []
    send_messages = []
    user_id = User.objects.filter(username = username)
    for u in user_id:
        seller_id = u.id
    SellingBook = SellBook.objects.filter(seller = seller_id, isSelling = 1)
    for b in SellingBook:
        eachBookname = b.title
        book = SellBook.objects.filter(title = eachBookname,seller = seller_id)
        for i in book:
            category = i.category
            title = i.title
            press = i.press
            author = i.author
            price = i.price
            newprice = i.newprice
            stock = i.stock
            content = i.content
            image = i.image
            ISBN = i.ISBN
            if stock>0:
                selling_book.append({'ISBN':ISBN, 'category':category ,'title':title,'press':press,'author':author,'price':price,'newprice': newprice,'stock':stock,'content':content, 'image':image})
    wantbook = wantBook.objects.filter(wanter = seller_id)
    for w in wantbook:
            title = w.title
            image = w.image
            author = w.author
            press = w.press
            total = w.total
            wanting_book.append({'title':title, 'image':image,'author':author,'press':press,'total':total})
    message = Message.objects.filter(receiver = username)
    for m in message:
            sender = m.sender
            msg = m.msg
            if msg == "":
                continue
            send_messages.append({'sender':sender,'msg':msg})
    return render(request, 'user_homepage.html', {'username': username, 'selling_book':selling_book,'wanting_book':wanting_book,'send_messages':send_messages})
```

流程：

1、 当进入个人主页时，从数据库中获取该用户所有的在售书籍、想买的书以及买|卖家留言。

2、 通过 for 循环在个人主页上显示

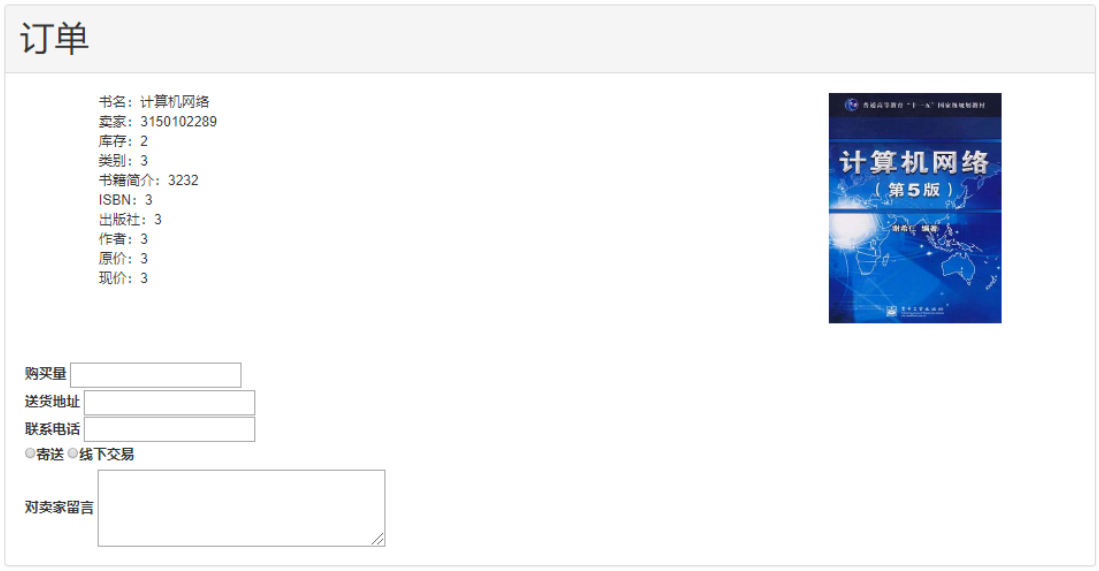
2.8、订单

主要功能：对于旧书交易网站主页上显示的在售数目和通过索引查找到的书籍，都可以进行下单操作。在订单界面会显示该书籍的所有信息，已经要求用户输入购买数量，送货地址和联系电话。可以选择寄送或者线下交易的方式进行交易。

在生成订单后，在个人订单界面，可以看到自己的所有订单，可以选择确认收货和取消订单。

具体设计：在点击书籍的下单按钮后，跳转到订单界面，同时服务器向网页发送该书籍的所有相关信息，并且显示在页面上。在用户输入了相关的购买信息并且选择了交易方式后，服务器会判断购买数量是否大于库存，并且购买者和发售者是否为同一人。如果购买数量大于库存，则会显示购买量大于库存，购入失败！的信息；如果用户自己购买自己的书，则会显示不能购买自己的书！；如果两者均满足要求，则下单成功。同时数据库中关于出售书籍的数目会对应减少。之后点击个人界面中的我的订单一项，就可以查看自己已经下的订单了。该功能由 win_order,lose_order,make_order,myorder 和 order 五个函数进行实现，具体代码可见源代码文件。

界面设计：


订单
书名：计算机网络
卖家：3150102289
库存：2
类别：3
书籍简介：3232
ISBN：3
出版社：3
作者：3
原价：3
现价：3
计算机网络
（第5版）
购买量
送货地址
联系电话
寄送 线下交易
对卖家留言
下单


已订书名：计算机网络
卖家：3150102289
类别：3
出版社：3
作者：3
原价格：3
现价：3
内容简介：3232
ISBN：3
交易方式：邮寄
送货地址：1
联系电话：1
确认收货
取消订单
计算机网络
（第5版）

函数实现：

```python
#生成订单界面
def make_order(request):
    username = request.COOKIES.get('cookie_username', '')
    islogin=request.user.is_authenticated()
    order = []
    if request.method == 'POST':
        b_ID = request.POST['type']
        bookname = SellBook.objects.filter(id = b_ID)
        for b in bookname:
            newtitle = b.title
        ordered_book  = SellBook.objects.filter(id = b_ID)
        for o in ordered_book:
            title = o.title
            category = o.category
            press = o.press
            price = o.price
            newprice = o.newprice
            image = o.image
            ISBN = o.ISBN
            author = o.author
            stock = o.stock
            content = o.content
            book_id = o.id
            seller_id = o.seller
            username = User.objects.filter(id  = seller_id)
            for u in username:
                seller = u.username
            order.append({'book_id':book_id,'ISBN':ISBN, 'category':category ,'title':title,'press':press,'author':author,'price':price,'newprice': newprice,'stock':stock,'content':c
    return render(request,'make_order.html',{'username':username, 'order':order})
```

流程：

1、 读取从按钮传来的书本 ID

2、 根据 ID 从数据库中读取相关信息

3、 将数据传回前端进行显示

```python
#下单
def order(request):
        username = request.COOKIES.get('cookie_username', '')
        islogin=request.user.is_authenticated()
        if request.method == 'POST':
            query = request.POST['query']
            buynumber = request.POST['number']
            address = request.POST['address']
            phone = request.POST['phone']
            book_id = request.POST['u']
            msg = request.POST['msg']
            if query=='mail':
                style = '邮寄'
            else:
                style = '线下交易'
            sellbook = SellBook.objects.filter(id = book_id)
            for s in sellbook:
                stock = s.stock
                seller_id = s.seller
            sellman = User.objects.filter(id = seller_id)
            for u in sellman:
                seller = u.username
            stock = stock-int(buynumber)
            if seller == username:
                return HttpResponse("<script>alert('不能购买自己的书！');window.location.href='/index';</script>")
            if stock>=0:
                Order.objects.create(book_id = book_id,seller = seller, buyer = username, buyNumber = buynumber, style = style, address = address,iphon
                SellBook.objects.filter(id = book_id).update(stock = stock)
                if msg=="":
                    pass
                else:
                    existMessage = Message.objects.filter(sender = username, receiver = seller)
                    if existMessage:
                        Message.objects.filter(sender = username,receiver = seller).update(msg = msg)
                    else:
                        Message.objects.create(sender = username, receiver = seller, msg = msg)
                        Message.objects.create(sender = seller,receiver = username, msg = "")
                return HttpResponse("<script>alert('下单成功！');window.location.href='/myorder';</script>")
            else:
                return HttpResponse("<script>alert('购买量大于库存，购入失败！');window.location.href='/index';</script>")
```

流程：

1、 读取从前端传回来的信息

2、 判断交易方式

3、 如果商品的卖家 ID 与自己一致，则返回警告“不能购买自己的书”

4、 当商品购买数量>=库存时，下单成功

5、 如果购买数量<库存，则返回警告“购买量大于库存，购入失败”。

```
#生成个人订单
def myorder(request):
        username = request.COOKIES.get('cookie_username', '')
        islogin=request.user.is_authenticated()
        my_order = []
        order = Order.objects.filter(buyer = username)
        for o in order:
            address = o.address
            iphone = o.iphone
            style = o.style
            book_id = o.book_id
            seller = o.seller
            order_id = o.id
            bookinfo = SellBook.objects.filter(id = book_id)
            for b in bookinfo:
                 title = b.title
                 category = b.category
                 price = b.price
                 newprice = b.newprice
                 press = b.press
                 author = b.author
                 content = b.content
                 ISBN = b.ISBN
                 image = b.image
                 my_order.append({'order_id':order_id,'address':address,'style':style,'iphone':iphone,'ISBN':ISBN, 'catego
        return render(request,'myorder.html',{'username':username, 'my_order':my_order})
```

流程：

1、 根据个人 ID，在数据库中读取自己的所有相关订单信息

2、 将信息传向网页

3、 通过 For 循环在我的订单界面进行显示

```
#确认收货
def win_order(request):
         username = request.COOKIES.get('cookie_username', '')
         islogin=request.user.is_authenticated()
         if request.method == 'POST':
                order_id  = request.POST['type']
                order = Order.objects.filter(id = order_id)
                for o in order:
                    sellers = o.seller
                Order.objects.filter(id = order_id).delete()
                existOrder = Order.objects.filter(seller = sellers, buyer = username)
                if existOrder:
                    pass
                else:
                    Message.objects.filter(sender = username, receiver = sellers).delete()
                    Message.objects.filter(sender = sellers, receiver = username).delete()
                return HttpResponse("<script>alert('确认收货! ');window.location.href='/myorder';</script>")
```

流程：

1、 在点击确认收货按钮后，前端向后端发送订单的 ID

2、 在 ORDER 数据库中删除该订单

3、 判断在买家和卖家之间是否还有订单存在

4、 如果订单存在，则不删除留言窗口

5、 如果没有订单，则取消两个人的所有留言。

```python
#取消订单
def lose_order(request):
        username = request.COOKIES.get('cookie_username', '')
        islogin=request.user.is_authenticated()
        if request.method =='POST':
            order_id  = request.POST['type']
            order = Order.objects.filter(id = order_id)
            for o in order:
               seller = o.seller
               buyNumber = o.buyNumber
               book_id = o.book_id
            book = SellBook.objects.filter(id = book_id)
            for b in book:
               stock = b.stock
            stock = stock + int(buyNumber)
            SellBook.objects.filter(id = book_id).update(stock = stock)
            Order.objects.filter(id = order_id).delete()
            existOrder = Order.objects.filter(seller = seller, buyer = username)
            if existOrder:
                  pass
            else:
                  Message.objects.filter(sender = username, receiver = seller).delete()
                  Message.objects.filter(sender = seller, receiver = username).delete()
            return HttpResponse("<script>alert('订单取消成功! ');window.location.href='/myorder';</script>")
```

流程：

1、 在点击取消订单按钮后，前端向后端发送订单的 ID

2、 更新订单取消后的库存书籍数量

3、 判断卖家买家之间是否还有订单存在

4、 如果订单存在，则不删除留言窗口

5、 如果订单不存在，则删除两人所有留言记录

2.9、卖家买家信息交互

主要功能：买家在进行下单时，可以向卖家发送信息。之后再卖家的个人主界面上，会显示买家发来的信息，并且可以回复。买家在收到卖家的回复后，在个人主界面上也会显示信息。

具体设计：买家卖家之间的信息交互，借用了贴吧的实现方式。在买家下订单时，可以向卖家发起对话，之后在个人主界面上，显示交流信息。该功能由 send_message 函数进行实现，具体代码可见源代码文件。

界面设计：

函数实现：

```
#发送信息
def send_message(request):
        username = request.COOKIES.get('cookie_username', '')
        islogin=request.user.is_authenticated()
        send_messages = []
        if request.method == 'POST':
               receiver = request.POST['type']
               msg = request.POST['msg']
               sender = username
               Message.objects.filter(sender = sender, receiver = receiver).update(msg = msg)
               return HttpResponse("<script>alert('回复成功！');window.location.href='/user_homepage';</script>")
```

流程：

1、获取从前端传来的信息

2、在 MESSAGE 数据库进行数据更新

三、系统功能测试

详情可见测试报告

四、实验感想

本次课程实验实现了一个旧书交易网站。在实验过程中，通过对于网站页面的设计，以及服务器端功能函数的实现，帮助我很好的理解了 B/S 体系结构，以及一个网站是如果搭建的。但是因为对于 html5 的不熟练，所以网站页面有些简陋，仅仅在清爽干净的程度上添加了功能，并没有很好的进行页面的美化。同时，对于 python 的 python 的掌握也还不到位，没有实现一个实时推送的对话平台，只能通过类似贴吧的方式进行留言交流。

总而言之，通过这门的学习，我了解了 B/S 体系结构，了解了前端后端的工作原理，锻炼了自己编写代码的能力。希望在以后的学习中能够更进一步。