

数据库期中实验报告

李培基 20307140044^{*}, 公超 20307140043[†]

2023 年 5 月 7 日

目录

1	摘要	2
2	开发环境	2
3	需求分析与设计	2
3.1	功能需求	2
3.2	E-R 图设计	3
4	具体实现	3
4.1	用户管理	3
4.2	书籍管理	8
4.3	订单管理	10
4.4	财务管理	12
4.5	新增功能与特性	14
5	总结	15

^{*}负责书籍管理, 订单管理, 账单信息

[†]负责用户管理, 账单查询, 报告撰写

1 摘要

此项目为图书销售管理系统的设计与实现，实现了一套图书管理系统对用户、图书的进货、销售、财务等方面进行统一管理。项目使用 Django 进行开发，前端界面使用 Bootstrap + jQuery。

2 开发环境

本系统的开发环境为 Django 框架和 MySQL 数据库。Django 是一个开源的 Web 框架，基于 Python 开发，遵循 MTV 模式，即将 web 应用分为模型 (Model)，模板 (Template)，视图 (View) 三层。它可以快速构建高质量、高性能的 Web 应用程序。Django 拥有丰富的内置功能，如：模型层、视图层、模板层、身份认证系统、管理界面等。MySQL 是一种开源的关系型数据库管理系统，使用 SQL 作为查询语言。它是 Web 应用程序最常使用的数据库之一。

在本系统中，前端使用了 Bootstrap3 + jQuery，后端选择 Django 4.2.1 作为 Web 框架，MySQL Client 2.1.1 作为数据库。Django 通过 ORM 工具 (Object-Relational Mapping, 对象关系映射) 将 MySQL 的表映射为 Python 对象，从而实现对数据库的操作。在 Django 项目的 settings.py 文件中配置 MySQL 的数据库连接信息，包括 host、port、database、username、password 等。然后安装 MySQLclient 以连接 MySQL 数据库。Django 通过配置和 migrate 命令将 MySQL 数据库映射为模型，然后可以在视图和模板中简单的进行数据库操作，这大大方便了 Web 应用的开发。

3 需求分析与设计

3.1 功能需求

根据项目文档的要求，系统模块结构分为四部分，主要如下：

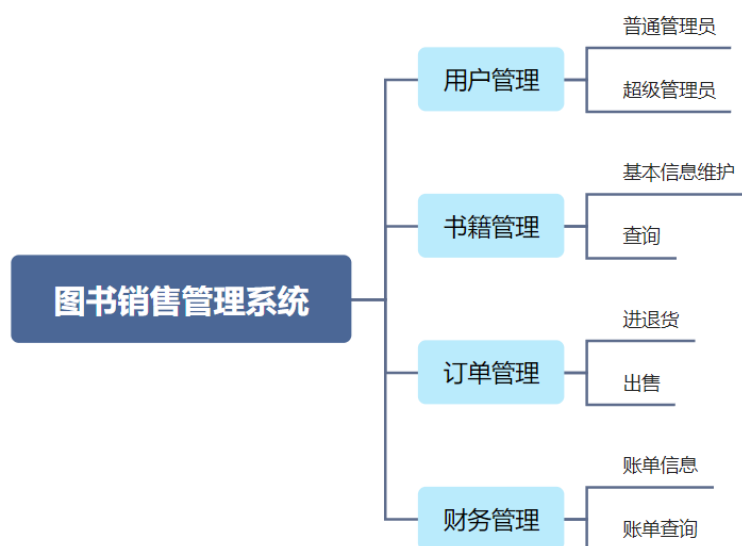


图 1: 系统主要功能

其中每部分的具体功能在项目文档中已有详细介绍。

3.2 E-R 图设计

下面是本系统的 E-R 图，主键已用下划线标出：

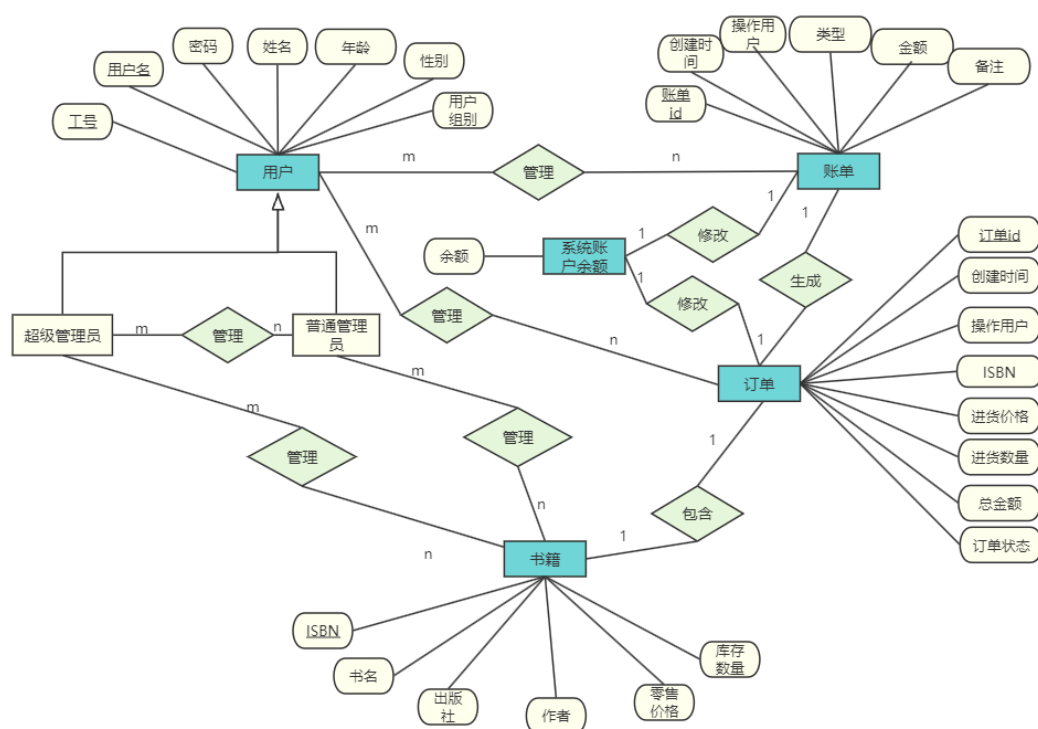


图 2: 系统 E-R 图

可以看到，本系统的实体主要有用户、账单、订单、书籍和系统账户余额，其中账单是对财务的记录，订单是对进货信息的记录。用户分为超级管理员用户和普通管理员用户，超级管理员可以对普通管理员的信息进行创建、管理，两者都可以对账单、进货、书籍信息进行管理。订单信息中包含每一条订单完成后会生成相应的账单记录，每次售书也会生成账单记录。账单信息会影响到系统的账户余额，例如启动资金、售书利润等，订单由于需要进货付款，也会影响系统账户余额。订单中包含了书籍的 ISBN 信息，订单和账单中均有指向用户的外键。

容易看出，本系统的实体均较为简单，不需要分解且不存在冗余信息。

4 具体实现

4.1 用户管理

- 1) 在 models.py 中定义了 Admin 模型，包括用户名、密码、工号、姓名、年龄、性别、用户组属性，其中用户组属性用来区分超级管理员和普通管理员。在系统初始化时，先在数据库中添加超级管理员的信息。

```
class Admin(models.Model):
    username = models.CharField(verbose_name='用户名', max_length=32, unique=True)
    password = models.CharField(verbose_name='密码', max_length=64)
    employee_id = models.CharField(verbose_name='工号', max_length=11, unique=True)
```

```

name = models.CharField(verbose_name=" 姓名", max_length=32)
age = models.IntegerField(verbose_name=" 年龄")
# 在 django 中做的约束 建立映射关系
gender_choices = (
    (1, " 男"),
    (2, " 女"),
)
gender = models.SmallIntegerField(verbose_name=" 性别", choices=gender_choices)
group_choices = (
    (1, " 超级管理员"),
    (2, " 普通管理员"),
)
group = models.SmallIntegerField(verbose_name=' 用户组', choices=group_choices)

```

- 2) 在 `utils\form.py` 中, 定义了关于用户管理的表单数据。AdminModelForm 以 Admin 为模型, 在新建用户时可以用来排除已经存在的用户名、工号, 检查输入的密码与确认密码的内容是否一致, 同时将密码经过 md5 加密后返回。这里的 md5 加密在 `utils\encrypt.py` 中实现:

```

def md5(data_string):
    obj = hashlib.md5(settings.SECRET_KEY.encode('utf-8'))
    obj.update(data_string.encode('utf-8'))
    return obj.hexdigest()

```

AdminEditModelForm 在编辑用户信息时使用, 检查用户名、工号是否重复, AdminResetModelForm 在重置密码时确保新密码与原密码不相同、两次输入的密码一致。

- 3) 在 `views\admin.py` 中定义了超级管理员进行用户管理的函数, 包括展示分页的全部用户信息, 添加新的用户, 删除用户, 编辑用户信息, 重置用户密码。其中分页功能是我们补充实现的。最终的界面如下:



图 3: 超级管理员用户管理界面

普通管理员不能访问这一页面, 首先在前端界面中设置了

```

{% if request.session.info.group == 1 %}
    <li><a href="/admin/list/"> 用户管理 </a></li>
{% endif %}

```

这使得普通管理员的界面没有访问此页面的链接。但是同时为防止通过输入网址 +GET 请求访问, 中间件中还设置了:

```
class UserManagerMiddleware(MiddlewareMixin):
    # 只有超级管理员才可以访问全部用户信息
    def process_request(self, request):
        if '/admin/' in request.path_info:
            info_dict = request.session.get("info")
            group = info_dict['group']
            if group == 1:
                return
            else:
                return render(request, 'error.html', {'error_msg': ' 您不是超级管理员用户，'})
```

在 views\user.py 中，设置了编写自己个人信息的入口，通过 filter 函数得到自身 id 对应的用户信息，并利用 UserEditModelForm 表单修改信息。此功能虽然通过 GET 请求完成，但仍然只允许当前用户访问自己的信息。同样在中间件中有设置：

```
class UserEditMiddleware(MiddlewareMixin):
    # 个人信息编辑：此入口只允许修改自己的信息，不允许访问别人
    def process_request(self, request):
        info_dict = request.session.get("info")
        if '/user/' in request.path_info and '/edit/' in request.path_info and request.path_info.endswith('<int>'):
            # 如果访问/user/<int>/edit/而且访问的 nid 不是自己的 id，不被允许
            # 后来想想自己写的不好，这样的话就不如直接写在 user_edit 函数中
            return render(request, 'error.html', {'error_msg': ' 您无权从此处修改他人信息，'})
```

实现的界面如下：

图 4: 修改个人信息界面

- 在 views\account.py 中编写了登录与退出函数。下面的代码给出了实现细节，主要是利用 LoginForm 表单的信息查找是否存在符合用户名和密码要求的数据，如果符合则跳转至功能区页面并保存 7 天 cookie，否则仍需登录。因此，如果没有 cookie 信息，则访问系统时会跳

转到登录界面，否则会直接跳转到图书信息界面。这里的 LoginForm 存有用户名和 md5 加密后的密码。

```
def login(request):
    """ 登录 """
    if request.method == "GET":
        form = LoginForm()
        return render(request, 'login.html', {'form': form})

    # POST 提交
    form = LoginForm(data=request.POST)
    if form.is_valid():
        # 表单有效，需要后端自行检查用户名密码是否正确（是否存在于数据库）

        admin_object = models.Admin.objects.filter(**form.cleaned_data).first()
        if not admin_object:
            form.add_error("password", " 用户名或密码错误")
            # form.add_error("username", " 用户名或密码错误")
            return render(request, 'login.html', {'form': form})
        # 用户名和密码正确
        # 写到用户浏览器的 cookie 中
        request.session["info"] = {'id': admin_object.id, 'username':
                                     admin_object.username, 'group': admin_object.group}
        # session 可以保存 7 天
        request.session.set_expiry(60 * 60 * 24 * 7)
        return redirect("/book/list/")

    # 若输入表单无效，form 携带错误信息返回登录页面
    return render(request, 'login.html', {'form': form})

def logout(request):
    request.session.clear() # 清除 session 中的信息
    return redirect('/login/') # 退出到登录页面
```

登录界面如下：



The image shows a user login interface with a white background and a light blue border. At the top, the title '用户登录' (User Login) is centered in a large, bold, black font. Below the title, there are two input fields. The first field is labeled '用户名' (Username) in a bold black font, and it contains the text 'gc'. The second field is labeled '密码' (Password) in a bold black font, and it contains three dots '...' indicating a password field. Below these fields is a blue button with the text '登录' (Login) in white.

图 5: 登录界面

- 5) 所有功能只有登录后才能使用，在 `middleware\auth.py` 中实现。定义 `AuthMiddleware`，如果 `session` 中已有用户信息，说明不需要重复登录，否则需要回到登录页面：

```
class AuthMiddleware(MiddlewareMixin):
    # 登录才能进入页面
    def process_request(self, request): # 在视图函数执行之前调用
        # 0. 排除那些不需要登录就能访问的页面
        # request.path_info 获取当前用户请求的 URL /login/
        if request.path_info == "/login/":
            return

        # 1. 根据请求体 (request) 中的 cookie,
        # 读取当前访问的用户在 session 中的 info 信息,
        # 如果能读到，说明已登陆过，就可以继续向后走。
        info_dict = request.session.get("info")
        if info_dict:
            return

        # 2. 没有登录过，重新回到登录页面
        return redirect('/login/')
```

这里还定义了类似的 `UserManageMiddleware`，对于用户管理的部分，根据用户组别判断是否可以访问全部的用户信息，保证只有超级管理员可以查看所有用户的信息，普通管理员只能查看自己的信息。`UserEditMiddleware` 对应修改自己的个人信息界面，通过用户 ID 保证只能修改自己的用户信息。

4.2 书籍管理

- 1) 在 models.py 中定义了 BookInfo 模型，包括 ISBN，书名，出版社，作者，零售价格，库存数量属性。也定义了 SystemBalance 模型，只有一个属性：系统账号余额。
- 2) 在 utils\form.py 中定义了多种表单，包括 BookInfoModelForm，存有书籍的基本信息，通过正则表达式判断 ISBN 格式是否正确，并通过 filter 函数确保 ISBN 信息不重复：

```
class BookInfoModelForm(BootStrapModelForm):
    isbn = forms.CharField(label="ISBN", max_length=13,
                           validators=[RegexValidator(
                               r'^(?:ISBN(?:-10)?:? )?(?=[0-9X]{10}$|(?=(?:[0-9]+[- ]){3})'
                               'ISBN 格式错误'), ]])

class Meta:
    model = models.BookInfo
    fields = ['isbn', 'name', 'press', 'author', 'retail_price', 'amount']
    # 可以自己指定 也可以 __all__

# 校验方法 2: 钩子函数 去重 不允许添加重复的 isbn 号书目
def clean_isbn(self):
    txt_isbn = self.cleaned_data['isbn']
    exist = models.BookInfo.objects.filter(isbn=txt_isbn).exists()
    if not exist:
        return txt_isbn
    else:
        raise ValidationError(' 该 ISBN 对应书目信息已存在')
```

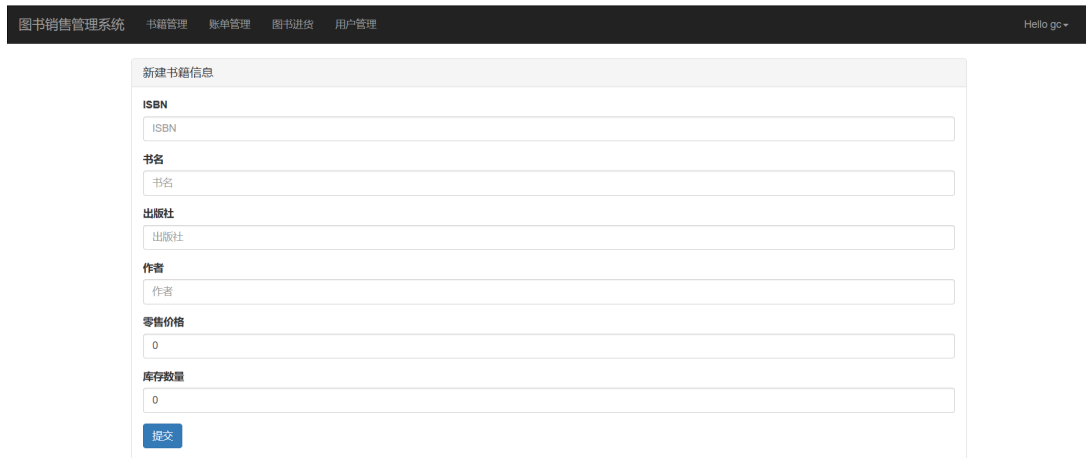
定义 BookEditInfoModelForm，通过 disabled=True 设置不允许修改 ISBN 和库存数量，保证库存数量只通过进出货修改。定义 BookSaleModelForm，包括 ISBN，作者，零售价，出售数量属性，并保证库存数量不少于出售数量。

- 3) 在 views\book.py 中定义了相关的视图函数。book_list 函数从前端获取 query 变量，如果获取到了则利用 queryset = models.BookInfo.objects.filter(Q(isbn__contains=search_data) | Q(name__contains=search_data) | Q(author__contains=search_data)) .order_by('name') 获取符合搜索条件的书本，否则展示全部的书籍。这是书籍展示的画面：



图 6: 书籍管理

`book_add` 函数用从前端获取的数据实例化一个 `BookInfoModelForm`，在确认书籍信息正确后，新建一个书籍信息条目。



The screenshot shows a web application titled '图书销售管理系统' (Library Sales Management System). The top navigation bar includes links for '书籍管理' (Book Management), '账单管理' (Bill Management), '图书进货' (Book Purchase), and '用户管理' (User Management). A user greeting 'Hello gc' is visible on the right. The main form, titled '新建书籍信息' (Add New Book Information), contains several input fields: 'ISBN' (with placeholder 'ISBN'), '书名' (with placeholder '书名'), '出版社' (with placeholder '出版社'), '作者' (with placeholder '作者'), '零售价格' (with placeholder '0'), and '库存数量' (with placeholder '0'). A blue '提交' (Submit) button is at the bottom.

图 7: 新建书籍信息

`book_edit` 函数类似，利用从前端获取的信息，对指定 `id` 对应的书籍进行修改。



The screenshot shows a web application titled '编辑图书信息' (Edit Book Information). The form contains several input fields with pre-filled values: 'ISBN' (0-670-82162-4), '书名' (nndi), '出版社' (66出版社), '作者' (gc), '零售价格' (88.00), and '库存数量' (3). At the bottom, there are two buttons: a blue '提交' (Submit) button and an orange '放弃修改' (Abandon Edit) button.

图 8: 编辑书籍信息

`book_delete` 函数较为简单，直接利用 `models.BookInfo.objects.filter(id=nid).delete()` 删除 `nid` 对应的书籍。`book_sale` 函数主要分为三部分，更新库存数量、创建对应的账单、更新系统账户的余额，其代码和界面如下：

```

def book_sale(request, nid):
    title = '图书出售'
    row_object = models.BookInfo.objects.filter(id=nid).first()
    info_dict = request.session.get('info')
    username = info_dict['username']
    if request.method == 'GET':
        form = BookSaleModelForm(instance=row_object)
        # 使用instance属性, 这种方式可以实现填写默认值, 相当于把value属性全部设置。还有一个功能是能找到更新的位置
        return render(request, 'change.html', {'form': form, 'title': title})
    else:
        remain_amount = row_object.amount
        form = BookSaleModelForm(data=request.POST, instance=row_object)
        if form.is_valid():
            with transaction.atomic():
                # 更新库存数量
                # remain_amount = row_object.amount 放在这里就会和sale_amount相同
                sale_amount = form.cleaned_data['sale_amount']
                models.BookInfo.objects.filter(id=nid).update(amount=remain_amount - sale_amount)

                # 创建账单
                amount_type = 1 # 1表示收入
                amount = row_object.retail_price * sale_amount # 此处amount是本次销售额
                models.Bill.objects.create(type=amount_type, amount=amount,
                                           description='图书出售:《{name}》*{num}'.format(name=form.cleaned_data['name'],
                                                                                       num=sale_amount),
                                           username=models.Admin.objects.filter(username=username).first())

                # 更新余额
                balance_object = models.SystemBalance.objects.first()
                if not balance_object:
                    models.SystemBalance.objects.create(balance=0)
                    balance_object = models.SystemBalance.objects.first()
                origin_balance = balance_object.balance
                models.SystemBalance.objects.all().update(balance=origin_balance + amount)

            return redirect('/book/list/')
        else:
            return render(request, 'change.html', {'form': form, 'title': title})

```

图 9: book_sale 函数

图 10: 图书出售界面


4.3 订单管理

- 1) 在 models.py 中定义了订单 Order 模型, 包括创建时间、操作用户 (外键)、ISBN、进货价格、进货数量、总金额、订单状态属性。在 utils/forms.py 中定义了 OrderModelForm 表单, 以 Order 为模型, 存有 ISBN、进货价格、进货数量属性, 并通过 clean isbn 函数确保对于需要进货的书籍, 如果库存中曾经有这本书的信息的话, 则直接将这本书的 ID 列入进货清单, 否则需要先添加书籍信息:

```
class OrderModelForm(BootstrapModelForm):  
    class Meta:  
        model = models.Order  
        fields = ['isbn', 'purchase_price', 'purchase_amount']  
  
    def clean_isbn(self):  
        txt_isbn = self.cleaned_data['isbn']  
        if not models.BookInfo.objects.filter(isbn=txt_isbn).exists():  
            raise ValidationError(' 该书籍信息在数据库中不存在，请先添加书籍信息')  
        return txt_isbn
```

2) 在 views\order.py 中实现了查看订单、新建订单、取消未付款订单、删除已付款或已取消的订单、订单支付。

order\list 函数取出所有的 Order 实体后分页展示出来：



创建时间	操作用户	ISBN	进货数量	进货价格 ¥	总金额 ¥	订单状态	操作
2023-05-08 06:56:21	gc	0-670-82162-6	5	50.00	250.00	未支付	支付订单 取消订单
2023-05-05 13:37:57	gc	0-670-82162-6	80	5.00	400.00	已支付	删除订单
2023-05-08 06:56:21	gc	0-670-82162-4	5	70.00	350.00	已取消	删除订单

图 11: 订单管理界面

我们发现订单的时间有误，经检查发现是当 USE\TZ = True 时，系统采用默认的 UTC 时间；只有当 USE\TZ = False 时，系统才采用我们设置的 TIME\ZONE(时区)。

上图中订单状态通过 state 属性控制，其值为 1，2，3 时分别对应未支付、已支付、已取消，state 为 1 的订单可以通过设置 state 为 3 来取消，state 为 2 或 3 的订单可以删除。订单支付通过 order\pay 函数实现，主要是创建一个对应的支出账单，减少系统账户余额，将此订单状态设为已支付，增加对应书籍的数目。代码如下：

```

def order_pay(request, nid):
    if models.Order.objects.filter(id=nid).exclude(state=1).exists():
        # 说明操作错误, 只允许支付“未支付”的订单
        return redirect('/order/list/')
    row_object = models.Order.objects.filter(id=nid).first() # 获取当前订单对象
    info_dict = request.session.get('info')
    username = info_dict['username'] # 获取当前用户名
    txt_isbn = row_object.isbn # 获取当前isbn
    book_name = models.BookInfo.objects.filter(isbn=txt_isbn).first().name # 找到该isbn对应的书名
    models.Bill.objects.create(type=2, amount=-row_object.total,
                               description='图书进货付款: 《{name}》 * {num}'.format(name=book_name,
                               num=row_object.purchase_amount),
                               username=models.Admin.objects.filter(username=username).first())
    balance_object = models.SystemBalance.objects.first()
    if not balance_object:
        models.SystemBalance.objects.create(balance=0)
        balance_object = models.SystemBalance.objects.first()
    origin_balance = balance_object.balance
    models.SystemBalance.objects.all().update(balance=origin_balance - row_object.total)
    models.Order.objects.filter(id=nid).update(state=2)
    updated_amount = row_object.purchase_amount + models.BookInfo.objects.filter(isbn=txt_isbn).first().amount
    models.BookInfo.objects.filter(isbn=txt_isbn).update(amount=updated_amount)
    return redirect('/order/list/')

```

图 12: 订单支付函数

新建进货订单通过 order\add 函数实现, 用前端获取的信息实例化一个前面提到的 OrderModelForm, 并将外键 username 指向操作的用户。项目要求没有库存信息的书需要先添加书籍信息, 这通过 OrderModelForm 中的 clean\isbn 函数实现, 此函数在前面有介绍, 如果在数据库中找不到要进货的书籍信息, 会报出错误“该书籍信息在数据库中不存在, 请先添加书籍信息”。以下是相应界面:

图 13: 新建进货订单界面

4.4 财务管理

- 1) 在 models.py 中定义了 Bill 模型, 包括时间、操作用户 (外键)、类型 (收入/支出)、金额、备注属性。在 utils\form.py 中定义了 BillModelForm 表单, 以 Bill 为模型, 包括类型、金额、备注属性, 还有用于保证收入金额为正数、支出金额为负数的函数 clean\amount。
- 2) 在 views\bill.py 中定义了相关的视图函数。bill\list 获取全部的 Bill 实例并展示, bill\add 用来新建除了图书出售、书本进货以外的额外账单, 例如启动资金等, 该函数利用 BillModelForm 表单, 创建一个新的账单, 外键关联到操作用户, 完成后更新系统余额。bill\delete 将某 id 对应的账单删除。下面是对应的界面:

图书销售管理系统

书籍管理账单管理图书进货用户管理

Hello gc

新建额外账单

查看指定时间段账单

账单列表					
时间	用户	类型	金额 ¥	备注	操作
2023-05-08 06:56:21	gc	支出	-250.00	图书进货付款:《篮球》*5	删除
2023-05-05 13:37:57	gc	支出	-400.00	图书进货付款:《表弟》*80	删除
2023-05-05 12:59:38	gc	收入	200.00	卖书	删除
2023-05-05 12:59:38	gc	支出	-700.00	进货	删除
2023-05-05 06:12:44	gc	收入	8888.00	启动资金	删除

账户余额: 7018.00

首页

«

1

»

尾页

页码

跳转

图 14: 账单管理界面

图书销售管理系统						Hello gc ~							
书籍管理		账单管理		图书进货		用户管理							
新建账单													
类型													
<input type="text"/>													
金额													
<input type="text"/>													
备注													
<input type="text"/>													
提交		放弃修改											

图 15: 新建额外账单界面

还定义了 `financial\history` 函数查看某段时间的收入/支出记录，从前端获取起始日期、结束日期、查看账单的类型（收入、支出、总收益），用 `filter` 函数过滤得到指定日期范围与类型的账单信息并展示：

```
def financial_history(request):
    """
    period: 时间段，如 '2020-01'(至今)，'2020-05-01~2020-05-31'
    income_expense: 收入/支出, 1 表示收入, 2 表示支出, 0 表示总收益
    """
    start_date = request.GET.get('start_date')
    end_date = request.GET.get('end_date')
    income_expense = request.GET.get('income_expense')
    if not (start_date and end_date and income_expense):
        return render(request, 'financial_history.html')
    start_date = datetime.datetime.strptime(start_date, '%Y-%m-%d')
    end_date = datetime.datetime.strptime(end_date, '%Y-%m-%d')
    start_date = datetime.datetime.combine(start_date, datetime.time.min)
    end_date = datetime.datetime.combine(end_date, datetime.time.max)
    queryset=models.Bill.objects.filter(timestamp__range=(start_date,end_date))
    total_amount=0
    type_display=' 总收益'
    if int(income_expense)==1:
```

```
queryset=queryset.filter(type=1)
type_display=' 收入'
if int(income_expense)==2:
    queryset=queryset.filter(type=2)
    type_display=' 支出'
for bill in queryset:
    total_amount+=bill.amount
context={
    'bills':queryset,
    'total_amount':total_amount,
    'period':start_date.strftime('%Y-%m-%d')+ '~'+end_date.strftime('%Y-%m-%d'),
    'income_expense':type_display
}
return render(request, 'financial_history.html', context)
```

下面是对应的界面：

图书销售管理系统 书籍管理 账单管理 图书进货 用户管理 Hello gc

财务记录

起始日期: 结束日期:

收入/支出: ☒ 收入 ☐ 支出

2023-05-01~2023-05-07 支出

总额¥-1100.00

记账时间	操作用户	收入/支出	金额(¥)	备注
2023年5月5日 12:59	gc	支出	-700.00	进货
2023年5月5日 13:37	gc	支出	-400.00	图书进货付款:《表弟》*80

图 16: 查看财务记录界面

4.5 新增功能与特性

在实验文档要求的基础上新增了对用户信息、书籍、账单、订单的删除功能，这些在前面已有介绍。在基本功能之上，还有如下特点：

1. 在用户层面对表单输入进行了严格检查：如：ISBN 号的正确格式，建立或编辑用户时检查对主键属性查重，支出收入的正负要求，修改密码不允许重复，等等。一旦在表单中输入错误会直接给用户显示错误信息。在这个层面的限制会减少数据库中错误的出现。
2. 在订单的设计中，使用了更实际的三种状态：未支付、已支付、已取消，同时增加了删除订单的操作，只可以取消“未支付”的订单，只可以删除“已支付”的订单。这样的设计使得订单页面更符合用户实际需求。

此外，我们还在表的展示中新增了分页功能的插件，其效果在前面的展示页面中可以看到。在 `utils\pagination` 中实现，代码较长不便于展示，这里对其实现简要说明。

分页功能的核心是根据每页显示条数和总条数计算出总页数，然后在页面中显示页码链接，允许用户点击翻页。Pagination 组件采用以下技术实现：

1. 初始化参数：获取每页显示条数 `page\size`、URL 中分页参数 `page\param`、显示页码数量 `plus` 等初始化参数。这些参数用于后续计算。

2. 获取请求页码: 从请求参数中获取页码 `page`, 如果不存在默认为 1。对 `page` 进行校验, 不能小于 1。
3. 计算分页相关参数:
 - `start`: 开始索引 = $(page - 1) * page_size$
 - `end`: 结束索引 = $page * page_size$
 - `page\queryset`: 当前页查询集 = `queryset[start:end]`
 - `total\count`: 总条数 = `queryset.count()`
 - `total\page\count`: 总页数 = $total_count // page_size + 1$
4. 生成分页 HTML。计算显示的页码范围, 生成首页、上一页、页码、下一页、尾页链接, 根据当前页高亮相应页码, 添加页码跳转框。返回分页 HTML 至模板, 在模板中显示。

通过上述技术, `Pagination` 组件实现了一个具备首页、尾页、上下翻页、页码链接等完整分页功能的组件。用户可以通过点击页码自由翻页, 并可以通过页码输入框跳转到指定页码。

5 总结

通过此次实验, 实现了一个较为完整的图书销售管理系统, 涵盖了用户管理、书籍管理、订单管理和财务管理等主要业务功能。考虑到功能比较简单, 没有使用更加复杂的前后端分离架构, 而是采用 Django 框架实现前后端均在同一工程中的模式。在开发过程中, 最大的难点是需要比较短的时间内掌握 Django 框架, 熟悉其视图、模型、模板、表单、管理后台等概念与开发方式。Django 作为一个高效的 Web 框架, 其内部封装了大量便捷功能, 但作为初学者却不太容易在短期完全掌握, 这也是我此次开发最大的收获所在。通过学习和开发, 我对 Django MVT 开发模式、ORM、后台管理界面、中间件等有了比较深入的理解。但是, 由于时间原因, 此系统的功能仍然比较简陋, 许多地方有待优化和完善, 例如系统的扩展性较差, 如果要添加新的业务, 可能需要较大程度地修改代码。综上, 此次开发获得了一定成果, 但系统的功能和可扩展性仍需提高。我们会在今后的工作中不断学习和优化此系统, 来进一步熟悉和运用 Django 框架。

本次实验, 由于采取了 Django 框架, 并没有实现严格的前后端分离。我们更多的根据此系统的功能进行了分工: 公超同学主要负责了数据库设计, 财务统计, 财务报表, 报告撰写; 李培基同学主要负责了用户管理, 账单管理, 订单管理, 前端开发; 两人共同完成了书籍管理, 登录控制等其他功能。代码已经同步到 <https://github.com/FlyingDutchman26/Bookstore-Management-System>。

最后, 感谢卢瞰老师的教授, 他的课程讲解让我们如沐春风, 使我们对数据库的设计理论有了更深刻的认识。也感谢两位助教为我们耐心的解答了项目设计中的各种问题。我们会一起继续努力。

参考文献

- [1] Django Software Foundation. *Django Documentation*. <https://docs.djangoproject.com/zh-hans/4.1/>. Django Software Foundation, 2023.
- [2] Bootstrap Contributors. *Bootstrap Documentation*. <https://getbootstrap.com/docs/5.3/getting-started/introduction/>. 2023.