

WebBookStore网上书店

简单的WebBookStore网上书店架构，基于CS架构和前后端分离，采用node.js作为脚本语言。依托express框架实现web服务端搭建，并通过Connection pooling机制维护与mysql数据库的连接。

简介

业务动机

疫情时代下，人们在实体商店下购物困难，书籍购买也不例外。但是快速发展的新时代下，无论是学习提升，还是陶冶心境，书籍终究不能离开我们的生活。因此，我们准备打造一个着重于订单管理和图书购买的网上书店，意图全面恢复线下实体书店业务功能。

项目介绍

功能介绍

- 提供基本的注册功能
- 提供基本的登录功能
- 提供浏览用户基础信息的功能
- 提供修改用户基础信息的功能
- 提供图书检索的功能，支持多条件模糊检索
- 提供订单下达的功能，支持购物车模式的订单形态
- 提供订单浏览的功能，对历史订单提供整体的视角
- 提供订单管理的功能，支持订单的加急、确认和取消

采用技术

HTML(HyperText Markup Language, 超文本标记语言). HTML语言运行在浏览器上，由浏览器进行解析，可以使用HTML建立Web站点，确定站点内容、框架，设置表单以及前端交互接口等。

CSS(Cascading Style Sheets, 层叠样式表) CSS用于为HTML添加样式，将style从HTML当中分离出来，便于统一网站风格，提高开发效率，也能实现良好的视觉呈现效果，优化用户互动体验。

JS(JavaScript) JavaScript是Web内置的脚本语言，原生JavaScript可以实现网站的基本交互功能，用以控制网站行为，结合诸如jQuery等第三方库，JavaScript能够实现非常丰富且强大的功能，特别是在Node.js框架下设置可以实现基于前后端分离的全栈式开发。

之所以选择Nodejs而不是php主要是考虑到要将网站做成前后端分离的架构，因为javascript发展历史相对悠久，且在互联网领域普及了相当长的一段时间，作为互联网世界的主要支柱，javascript在各种方面都比较传统和成熟，并拥有各种各样功能强大的库来优化我们的项目。像诸如回调函数，Promise机制等的特性，非常适合用于处理数据库的高并发、多独立的事务操作，因而成为我们的首选语言脚本。也就是说，我们舍弃了部分php的前后端交互的便利性，而更着眼于数据库操作的优化，我们认为这与本课的重点是息息相关的。

假设

用户假设

通过组内讨论，决定用户分成两类client和staff。这两类用户对网上书店的需求和目的是如下：

- client在网上书店想登记、登录、查询图书、催单、购买图书。对应地，在网上书店上设计了登记、登录、查询图书、催单、购买图书的功能。
- staff在网上书店想管理并查询顾客、图书和订单信息。对应地，在网上书店上设计了顾客管理功能、图书馆里功能，销售管理功能。

书本假设

书本的属性有图书号，书名，作者，出版社，摘要，价格，库存。其中图书号是作为主码，要能唯一识别一本书，因此图书号的重要性很大。价格和库存应该根据销售情况可变。

订单假设

订单的属性有订单号，订单项数，订单状态，账号，购买数量，图书号。订单装填有三类processing,failed和finished。processing表示订单正在进行中，failed表示因为某种原因订单被取消，finished表示订单已经完成。

系统描述

基础架构

本项目采用前后端分离的技术，因而系统架构自然也是围绕前后端来展开。首先，在前端由html报文和css层叠样式表组成用户交互界面。然后，利用javascript来在浏览器发起GET/POST的http请求，在这一过程中浏览器充当了客户端client的角色，我们的服务端server将始终监听8080端口，当接受到http请求后，根据其url将其引导至对应的路由并调用回调，之后进行数据库的连接并执行查询或者修改的事务。当事务完成后返回相应的相应体，供前端更新、渲染界面或是提示用户操作。

在这一架构中，“消息传递”是核心，也即http的GET/POST请求及其响应，包括报文格式的约定，报文字段的含义规定，在前后端分离的架构当中，这些约定和协议至关重要，可以说是本项目的生命和核心。在后面我们将看到，处理这些请求体的发送和解析相应体是前后端交互的主要任务，而后端则致力于读懂这些请求体并返回正确的相应体，我们的项目就是在一次次request和response中逐步搭建起来的。

项目亮点

- 1.采用javascript作为脚本语言，可被绝大多数浏览器编译执行，同时避免了apache或nginx等网络服务器的使用，因而项目具有较高的泛用性，轻量化，容易部署。
- 2.基于回调函数的异步非阻塞代码运行模式，在后端提高了数据库的查询、修改、更新速率，在前端提高了页面更新的速度。
- 3.利用Promis机制对数据库事务的执行进行了封装，能在更新表项发生故障时即时回滚，较好地维护了数据库事务的ACID性质，有利于维护数据库的正确性和完整性。

```
const exec = function(sqlArr) {
  return new Promise(function(resolve, reject) {
    var promiseArr = []
    db.getConnection(function(err, conn) {
      if(err) {
        console.log('connect failed')
        reject(err)
      }
      conn.beginTransaction(function(err) {
        if(err) {
          console.log('beginTransaction failed')
          reject(err)
        }
        // 将所有需要执行的sql封装为数组
        promiseArr = sqlArr.map(function({sql, value}) {
          return new Promise(function(resolve, reject) {
            conn.query(sql, value, function(e, results) {
              e ? reject(e) : resolve({ results, success: true })
            })
          })
        })
      })
    })
  })
}
```

```

        })
    })
    // Promise调用所有的sql，一旦出错则回滚，否则提交事务并释放连接
    Promise.all(promiseArr).then(function(res) {
        conn.commit(function(err) {
            if(err) {
                console.log('commit failed')
                reject(err)
            }
            conn.release()
            resolve(res)
        })
    }).catch(function(err) {
        conn.rollback(function() {
            console.log('rollback')
        })
        reject(err)
    })
})
})
})
}

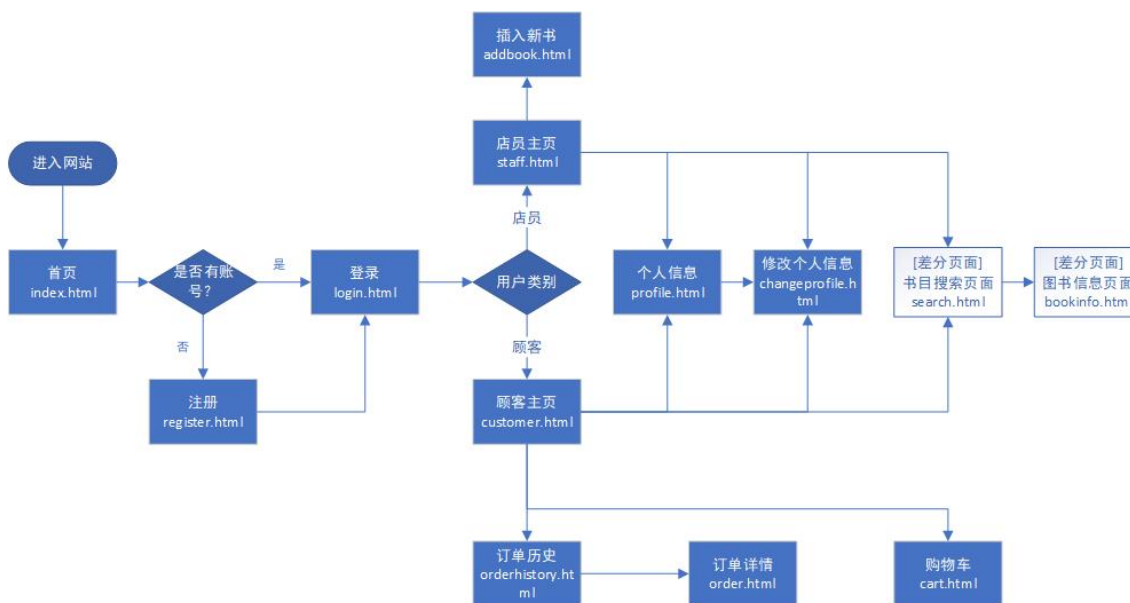
```

- 4.用数据池建立与数据库的连接，采用数据池的首要目标即在于实现同一时间的多个连接，是项目具有并发能力的关键之处，也因此我们与数据库的连接随时可以释放，提高了数据库的可操作性。
- 5.前后端分离架构，使得前端具有极高的自主性，前端不再依赖于后端，因而从理论上使前端更有可能按自己的想法丰富并完善网页设计任务。
- 6.良好的代码结构及模块化编程，提高了本项目的兼容性和易扩展性，团队之间分工协作，提升了代码质量和代码之间的耦合度。
- 7.前端建立储存机制sessionStorage，有效沟通前后端数据和消息体，维护了请求体数据的正确性，提升了前后端交流的效率。

前端部分

由于各浏览器对于网页元素的默认显示格式不同，故可能最终整体展示效果与测试环境不同。网页整体色调以灰色系为主，侧重于功能应用，因此网页的复杂性主要网页动态效果和前端存储管理上体现。为使得用户能够清晰地完成用户任务，网页制作时对所有的可选项、按钮、链接均使用了动态效果，并且利用JavaScript在部分页面实现了视角差分，即以不同身份进入页面时会展示不同的功能和信息陈列方式。前端将html、css、js文件分离，再由html调用指定的css和js文件，在保证网页结构的基础上尽量使代码简洁，易于维护、增添和修改。顾客与店员被同等视为用户而非用户与管理，两者的差分在于允许使用的功能操作以及视图有所差异。

网站架构



网站总计13个页面，根据页面在网站中的功能分为入口页面、导航页面和功能页面。

- 入口页面：首页index.html，注册页面register.html，登录页面login.html
- 导航页面：店员主页staff.html，顾客主页customer.html
- 功能页面：图书插入页面addbook.html，个人信息页面profile.html，修改个人信息页面changeprofile.html，图书检索页面search.html，图书信息页面bookinfo.html，订单历史页面orderhistory.html，订单详情页面order.html，购物车页面cart.html

入口页面

- 首页index.html：提供简易的网站入口，指向注册和登录页面
- 注册页面register.html：提供注册表单，允许不具有账号的新用户注册
- 登录页面login.html：用户使用账号和密码进行登录，登录成功后由后端验证其身份并让用户进入对应主页，同时在前端存储中预存身份信息。

导航页面

- 店员主页staff.html：身份为店员的用户能够进入的导航页，页面会有店员身份提示。直接链接的功能页面有图书插入页面，个人信息页面，修改个人信息页面，图书检索页面。
- 顾客主页customer.html：身份为顾客的用户能够进入的导航页。直接链接的功能页面有个人信息页面，修改个人信息页面，图书检索页面，订单历史页面，购物车页面。

功能页面

- 图书插入页面addbook.html：仅限店员访问。提供图书信息空白表单，提交后将表单内容发送给后端以插入书目。
- 个人信息页面profile.html：查看当前用户的信息，包括Id、用户名、用户类型、邮箱、电话、地址、用户描述，链接修改个人信息页面。
- 修改个人信息页面changeprofile.html：提供表单，允许修改当前用户信息，其中Id、用户名和用户类型不可修改。
- 图书检索页面search.html：允许以书名搜索图书，且可以使用其他图书属性精确检索。呈现检索结果后，图书标题链接图书信息界面。在图书框悬停时，顾客有add to order选项即将对应图书加入购物车（加入前端存储），点击后触发并将选项替换为drop order即移出购物车（移出前端存储），再次点击触发并返回add to order选项。店员有modify选项，链接图书信息页面。
- 图书信息页面bookinfo.html：展示特定图书信息。顾客有add to order选项。店员则有modify选项，点击后展开图书表单并提供三个表单按钮，功能分别为删除图书，修改图书信息，作为新书插入。
- 订单历史页面orderhistory.html：顾客访问。展示当前用户的历史订单，并提供订单详情页面入口。

- 订单详情页面order.html：展示选定的订单信息和细则。允许顾客取消订单或催办订单。
- 购物车页面cart.html：获取顾客挑选的图书（前端存储），允许顾客下单、移出图书、清空。

前端存储

为降低前后端数据流量，节省存储空间开销，同时有效沟通前后端数据和消息体，前端建立了储存机制sessionStorage。主要可能存储的数据有：

- 用户类别：用于区分用户主页，限制用户能够访问的页面，影响差分页面视图。
- 用户Id：访问订单历史页面时自动发送请求获取用户订单历史；访问个人信息页面时发送请求获取用户信息；提交订单时一并提交。
- 图书Id：用于访问图书信息页面时发送请求获取图书信息，在上一页面跳转前存储。
- 订单Id：类似于图书Id。
- 其他数据

用户主要任务流程

顾客-购买图书

进入网站->注册->登录->顾客主页->图书检索页面->搜索->选择图书加入订单|->顾客主页->购物车页面->修改->提交

顾客-订单操作

进入网站->登录->顾客主页->订单历史->订单详情->操作订单

顾客-个人信息设置

进入网站->注册->登录->顾客主页->个人信息页面->修改个人信息->填写表单->提交

店员-添加图书

进入网站->登录->店员主页->图书插入页面->填写表单->提交

店员-图书操作

进入网站->登录->店员主页->图书检索页面->搜索->选择图书->图书信息页面->修改->填写表单或其他操作->提交

后端部分

注册登录

注册

- 建立连接
- 查询用户是否已存在
- 插入用户数据
- 释放连接

登录

- 建立连接
- 查询用户是否不存在
- 比对用户密码
- 释放连接
- 返回用户标识码

用户信息

查询用户信息

- 建立连接
- 由accountid单表查询user表
- 释放连接
- 返回用户基本信息

修改用户信息

- 建立连接
- 截获请求体，编写update的sql修改语句
- 执行sql语句
- 释放连接

图书信息

图书信息查询

- 建立连接
- 由bookid单表查询book表
- 释放连接
- 返回图书信息

图书检索

- 建立连接
- 截获请求体数据
- 由截获数据编写检索式
- 执行检索任务
- 释放连接
- 返回图书检索结果

具有对不同检索方式具有较高的兼容性，针对未给出的检索关键词，由后台系统进行默认值补充，用以规范用户的检索条件。

订单

历史订单查询

- 建立数据库连接
- 由accountid查询orders表orderid，orderstatus
- 由orderid查询book，orderdetail的等值连接表bookname等属性
- 由book属性计算order总价
- 释放数据库连

由于mysql在node里面查询所采用的function回调函数方式是异步操作，直接循环查询会导致响应体整合滞后于响应体发送，致使响应体数据缺失；因而，采用promise封装+await等待使之强行同步，确保响应体数据的完整性。

订单加急

- 建立连接
- 查询订单现在的状态
- 根据合适的状态更新订单状态
- 释放连接

订单取消

- 建立连接
- 查询订单现在的状态
- 判断订单是否已经取消
- 更新订单状态
- 恢复图书库存
- 释放连接

订单确认

- 建立连接
- 查询订单现在的状态
- 判断订单是否已经取消
- 更新订单状态
- 释放连接

下单

- 建立连接
- 由accountid插入orders表order基础信息
- 由orderid插入orderdetail表对应图书的订单细节
- 由bookid, buynum更新book表库存
- 释放连接

前后端连接

发起GET/POST请求

使用jquery库，借助其ajax功能，设置请求的url，headers，body并在前端发起GET/POST的http请求至后端，成功后将后端发来的http响应体解析并根据相应体的内容如status，msg，data等属性内容来决定前端将要执行的任务。

连接缓存

在用户浏览网页时经常会遇到链接点击的需求，连接点击的实质是由浏览器作为客户端client发送http的GET请求，由服务器发送适合的html报文，这些报文固定存储在一个硬盘地址上。然而，在前后端分离的架构下，后端不负责渲染网页内容，它仅将对应的报文发送出去，这带来了一定的问题。

以图书为例，在搜索界面，我们期望用户点击图书名字后跳转之图书信息的页面，这个页面详细地介绍了图书的名称、作者、出版社、价格等信息。显然，我们不可能为每本图书都写一个独一无二的界面，相应的，我们希望他们具有相同的框架而只是内容不一样，所以服务器仅会发送这个框架的html，这个框架目前还没有任何内容！

当前端收到这个html报文，首先由浏览器进行渲染。渲染完成后，立刻向后端发送POST请求，期望获取有关这个图书的相关信息，因而请求体会需要图书id，而这个id从何而来呢？显然，我们需要在用户点击图书名字后缓存该图书的id（window.sessionStorage.setItem("bookid", 图书id)）。这种做法非常有效。

订单管理

利用浏览器的sessionStorage执行订单管理，通过addOrder()和dropOrder()等函数来维护订单内容并在cart购物车界面发起makeorder下单请求时将其作为请求体的数据发送给后端。order数据格式是前后端协定的。形如：

```
{
  accountid: 用户ID,
  booklist: [
    {bookname: 书名A, buynum: 购买数量},
    {bookname: 书名B, buynum: 购买数量},
    {bookname: 书名C, buynum: 购买数量},
  ]
}
```

显示数据

在前端更新页面内容时用到了jquery库，通过\$美元符号取DOM元素，并借助诸如attr，text，val等函数更改页面内容，结合复杂的数据结构可以实现前端页面内容的异步更新

数据库

数据库模式

- 图书（图书号，书名，作者，出版社，摘要，价格，库存）
- 订单（订单号，订单项数，订单状态，账号）
- 订单详情（订单详情号，订单号，图书号，购买数量，总价，折扣）

- 用户（账号，密码，用户名，描述，邮箱，电话，地址，权限）
- 用户限制（权限，限制）

完整性约束

- 用户名不能重复
- 用户有且仅有两类
- 订单状态有且仅有三类
- 单价乘以数量必须等于总价
- orders表里的订单号（orderid）应该对应于orderdetail表里的订单号
- orderdetail表里的图书号（bookid）应该对应于book表里的图书号
- orderdetail表里的数据量应该大于等于orders表里的数据量

用户权限

- client
 - 在orders表里能添加数据
 - 在book表里查询图书信息
 - 在orderdetail表没有进入权限
 - 在user表里能添加数据，也能修改自己添加的数据（不能重复）
- staff
 - 在orders表里能看出订单状态（orderstatus）
 - 在book表里能添加、删除、修改、查询图书信息
 - 在orderdetail表里查询订单信息
 - 在user表里能查询client信息

测试

测试数据

- 图书数据（有50本书籍的数据）
- 订单数据（有20条数据）
- 订单详情数据（有24条数据）
- 用户信息数据（有10条数据）
- 用户权限数据（有2条数据）

总共有106数据

测试方式

在后台补充本地mysql数据后，在浏览器运行网站，试验功能，检测能否正常访问数据库并展示数据，进行作业的debug

测试结果

基本能够满足我们的预期要求，但结果有待提高

用户手册

预备处理

- 数据库连接 public/lib/sql.js中的password需要修改当前主机的mysql的密码
- 在webbookstore文件夹下命令行运行npm start 来监听8080端口

- 浏览器访问本机8080端口，即可访问网上书店

首页

Web Book Store

[login](#)
[register](#)

- 进入首页（index.html）能看到两个按钮：login, register。按login按钮，跳转到login界面，按register按钮，跳转到register界面。

注册登陆

A screenshot of a web form titled "Register" on a dark gray background. The form contains two input fields: "UserName:" followed by a white rectangular input box, and "Password :" followed by another white rectangular input box. Below these fields is a button with the text "Submit" inside a black-bordered box.

- 这是register界面。添加信息之后，按submit就能登记成功。

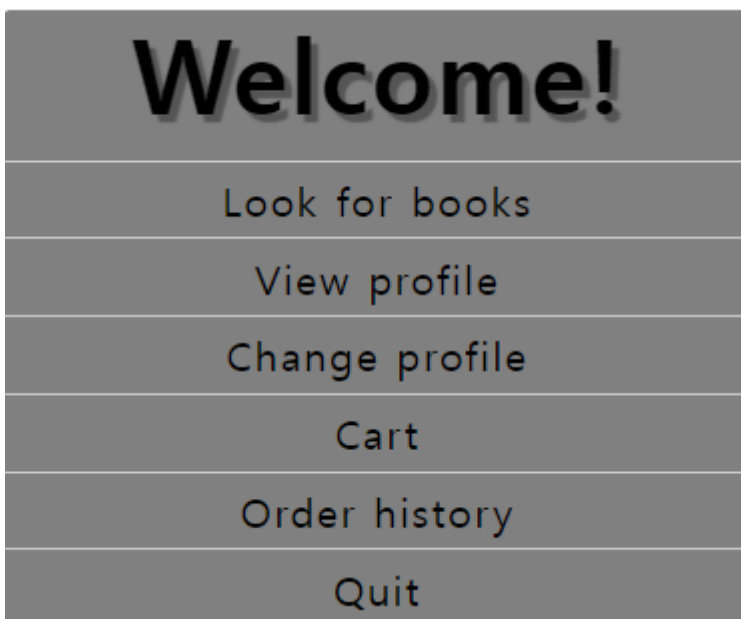
A login form on a dark gray background. At the top, the word "Login" is displayed in a large, white, serif font. Below it, the text "UserName:" is followed by a white rectangular input field. Underneath that, the text "Password :" is followed by another white rectangular input field. At the bottom center, there is a white rectangular button with the word "Submit" in a black, sans-serif font.

Login

UserName:

Password :

- 这是login界面。准确添加登记时使用的信息之后，按submit就能登录成功。

A vertical menu on a dark gray background. At the top, the word "Welcome!" is displayed in a large, white, serif font. Below it, there are seven white rectangular buttons, each containing text in a black, sans-serif font. The buttons are stacked vertically and separated by thin white lines.

Welcome!

Look for books

View profile

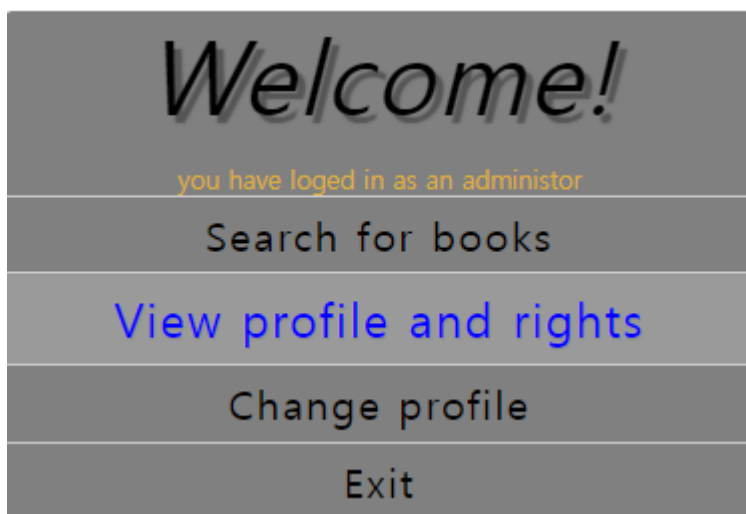
Change profile

Cart

Order history

Quit

- client界面上显示具有client权限的用户能选择的选项。按quit，返回到login界面。



- staff界面上显示具有staff权限的用户能选择的选项。按Exit，返回到login界面。

个人信息



- 在profile界面上，用户能看到自己的个人信息。

Profile

AccountId	Immutable
UserName	Immutable
Type	Immutable
E-mail	<input type="text"/>
Telephone	<input type="text"/>
Address	<input type="text"/>
Description	<input type="text"/>

- 在changeprofile界面上，用户能修改个人信息。

查询书本

Search

Bookname	<input type="button" value="Add to order"/> <input type="button" value="Modify"/>
Author	
Press	
Price	

- 在search界面上，能查询书本。简单地输入书名之后查询书本也可以，按extend search之后附加其他信息之后更准确地查询也可以。

- 下面黑框里面，会显示跟输入信息对应的查询结果。用户能看到书本的书名、作者、出版社和价格，为了查看更详细的结果，点击蓝色的书名就能跳转到bookinfo界面。把鼠标箭头放在黑框上面，会显示add to order,modify按钮。按add to order按钮，能把书放在购物车里。 具有staff权限的用户能按modify按钮，修改书本信息。

Book Detail

The book of someone

Author	Name
Press	Name
Price	Name
Stock	Name
Abstract	This test sample seems useless.

Modify

Cancel

- 在bookinfo界面上，除了书名、作者、出版社和价格以外，还能查询到库存信息和摘要。具有staff权限的用户能点击modify，修改书籍信息。

订单操作

Order

order id: 000000

customer id:000000

order status:submitted

Id	Book Title	Author	Price	Quantity	Total
xxx1	What	Silica	\$	0	=price*1
xxx2	Nice	Silica	\$\$	0	=price*2
xxx3	Never gonna...	Silica	\$\$\$	0	=price*3
xxx4	No god please no	Silica	\$\$\$\$	0	=price*4

All in Total: 10 books, \$214.00

Destination Address: 218

Back

Cancel

Urge

- 在order界面上，能查询到用户下单的书本信息。用户会能看到书本的按back按钮，就能跳转到orderhistory界面。
- 按urge按钮，用户能催单。

Cart

Id	Book Title	Author	Price	Quantity	Total
xxx1	What	Silica	\$	<input type="text"/>	=price*1
xxx2	Nice	Silica	\$\$	<input type="text"/>	=price*2
xxx3	Never gonna...	Silica	\$\$\$	<input type="text"/>	=price*3
xxx4	No god please no	Silica	\$\$\$\$	<input type="text"/>	=price*4

All in Total: 10 books, \$214.00

Destination Address: 218

Search

Abort

Confirm

- 在cart界面上实现了购物车的功能。点击书名，能跳转到bookinfo界面，能查看书本详细信息。 在界面右下角，显示用户加入购物车的书本的总量和总价。
- 在quantity方框里，用户能调整想要购买的数量。
- 按remove按钮，用户能删除加入到购物车的书本信息。
- 按confirm按钮，能确认并保存修改的信息。
- 按search按钮，用户能跳转到search界面，继续查询书本。

购买记录

History Orders

Order Id: xxxxx		Total Price: 0.00		Status: Processing		▼	
Id		Book Title		Author		Price	Quantity
xxx1		What		Silica		\$	0

- 在order history界面上，用户能查看订单的大概信息。把鼠标箭头放在order id,total price, processing上点击，就能进入到order界面，能查看更详细的信息。
- 点击书名，能跳转到bookinfo界面。

局限性

前后台系统

顾客前台系统和用户后台系统较为粘连，在网站设计上没有做到很好地分离，根据用户的身份类别给出的功能差异，因此结构稍显混乱

社区氛围交互缺少

由于书店系统着重于购物流程的管理，在图书宣传、氛围营造、评论设置等等的社区管理上较为缺乏

改进

程序清单

文件结构

```
.
├── index.js    (主函数/入口点)
├── index.html  (首页界面)
├── html       (诸html文件)
│   ├── login.html      (html示例)
│   └── register.html   (html示例)
├── public     (站点可调用资源)
│   ├── css          (css文件)
│   ├── img          (图片文件)
│   ├── js           (脚本文件)
│   └── lib           (函数库)
├── package.json
├── package-lock.json
└── README.md
```

前端清单

```
html
├─ addbook.html
├─ bookinfo.html
├─ cart.html
├─ changeprofile.html
├─ client.html
├─ login.html
├─ orderhistory.html
├─ order.html
├─ profile.html
├─ register.html
├─ search.html
└─ staff.html
```

```
public/css
├─ bookinfo.css
├─ index.css
├─ login.css
├─ order.css
├─ orderhistory.css
├─ profile.css
├─ search.css
├─ template.css
└─ usercenter.css
```

后端清单

```
public/lib
├─ base.js
├─ bookinfo.js
├─ cart.js
├─ exec.js
├─ order.js
├─ profile.js
├─ search.js
└─ sql.js
```

前后端交互清单

```
public/js
├─ bookinfo.js
├─ cart.js
├─ changeprofile.js
├─ hidden.js
├─ jquery-3.6.1.min.js
├─ jquery.form.min.js
├─ link.js
├─ login.js
├─ logout.js
└─ orderHistory.js
```



```
|— order.js
|— orderManage.js
|— profile.js
|— register.js
└— search.js
```

贡献

- 肖萧：前后端的连接以及沟通前后端的数据模式，编写对应报告
- 王泰宇：后端nodejs脚本，调用数据库，编写对应报告
- 郭明康：前端html和css网页编写，展示功能页面，编写对应报告
- 陈民后：填充数据库数据，完善数据库结构，编写对应报告