

School of Computing, Engineering and Mathematics (CEM)

Faculty of Engineering, Environment and Computing (EEC)

**5001CEM SOFTWARE ENGINEERING** | 2122

**PROJECT REPORT**

**NAME: YISONG LI**

**SID: 11384841**

1. **CODE PURPOSE**

The application is a book shopping website based on flask framework and SQlite3 database. The website page is written in HTML + CSS /JavaScript. Users can purchase and other operations, website managers can load goods

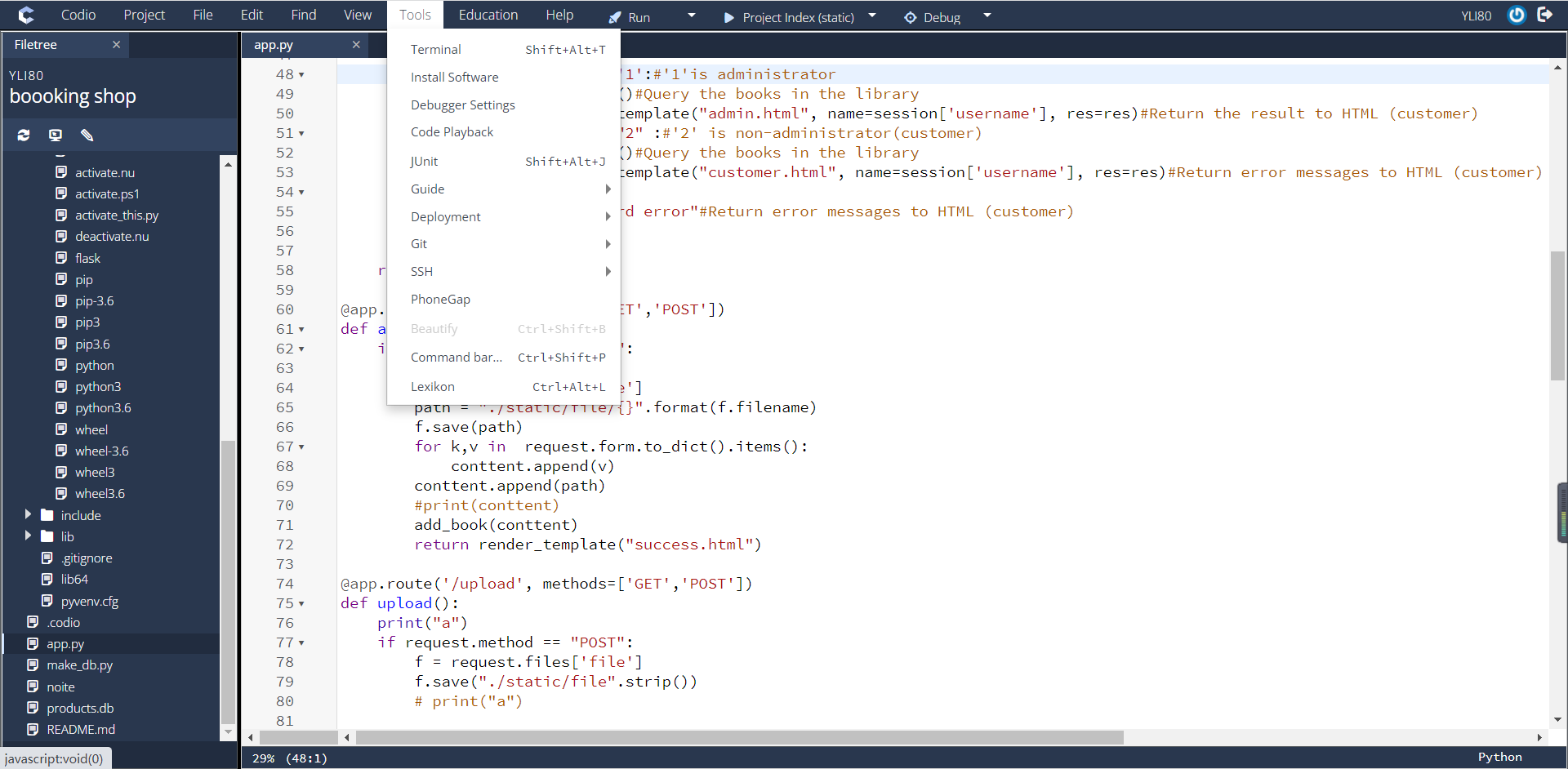
1. **CODE LOCATION**

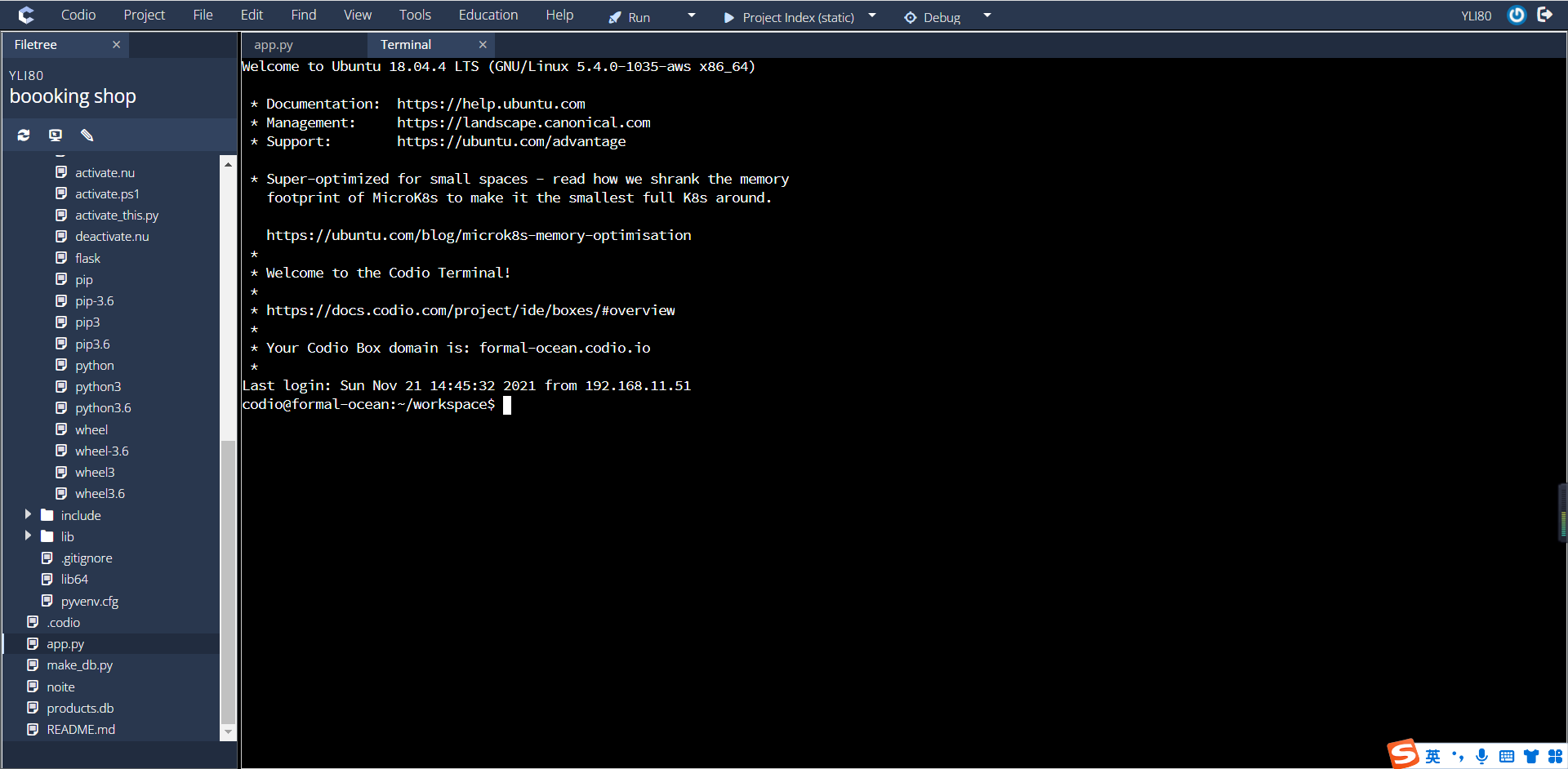
<https://github.coventry.ac.uk/5001CEM-2122/Yisong-Li-book-shop>

1. **CODE INSTALLATION**

First, download the code on GitHub and open it using Codio.

open the app.py, then go to Tools/terminal





Then, enter the following statements in terminal according to lab 1 set up on Aula

. venv/bin/activate

pip install Flask

pip install Flask-SQLAlchemy

pip install flask-cors

\*The purpose of doing this is to load flask and SQL onto Codio

Then, put those code into the terminal:

First,pip3 install virtualenv

Second,virtualenv venv

Then,source venv/bin/activate

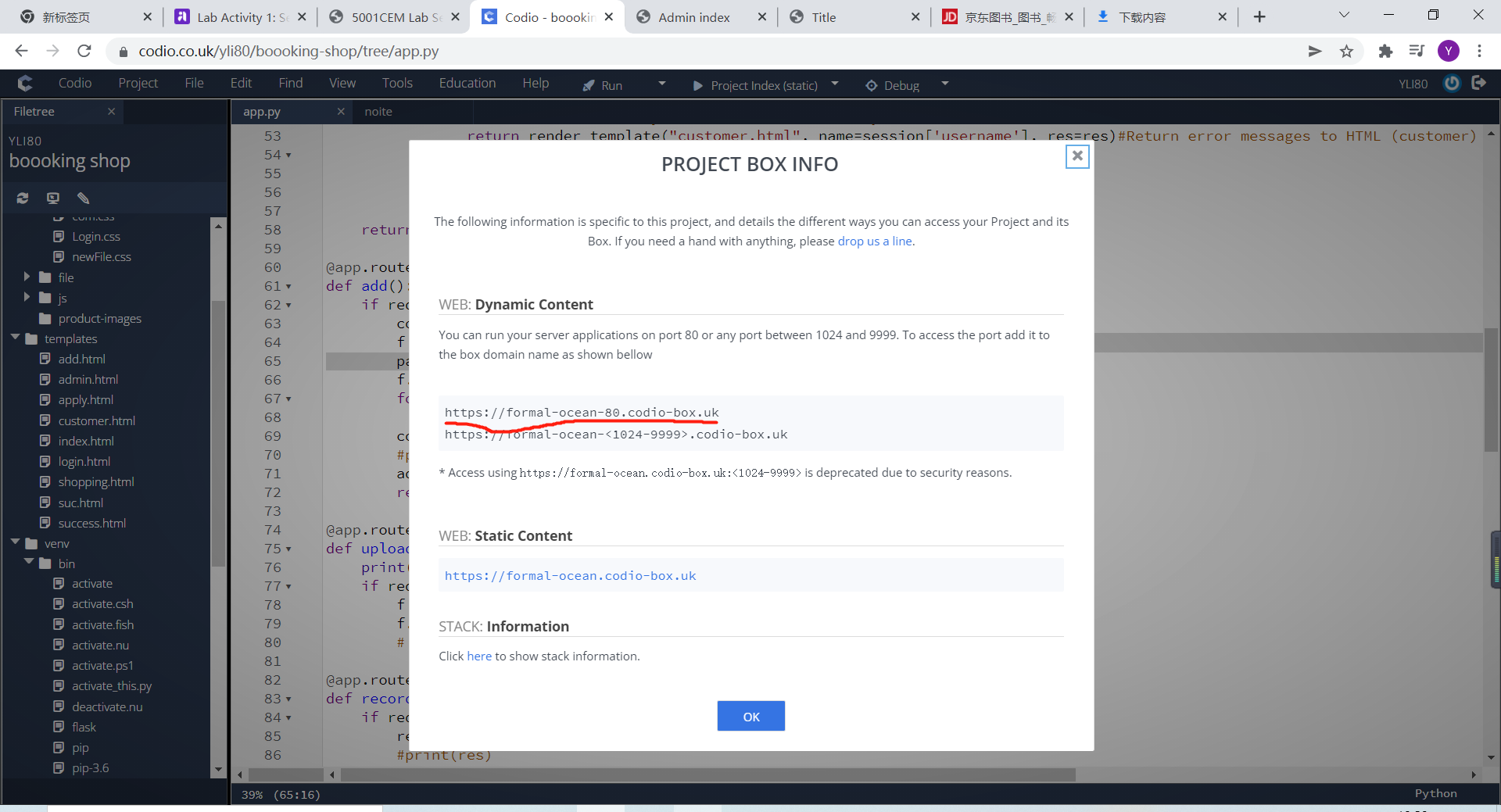
At the last,FLASK\_APP=app.py FLASK\_ENV=development flask run --host=0.0.0.0

\*‘Pip3 install virtualenv’ is activating a Virtual Environment

‘virtualenv venv’is a folder containing the Python executable and a copy of the PIP library will be created in the current directory

‘source venv/bin/activate’is activating a Virtual Environment

‘FLASK\_APP=app.py FLASK\_ENV=development flask run --host=0.0.0.0’is Run the app.py file]

After that, we can see there a link in the terminal(or open a new website), open Project/Box Info

Copy the red line link to the website(everyone will be a little different in the link, but attention that it must be the first line of link). after that, change 80 to 5000,then website can be open.

1. **CODE EXPLANATION**
2. The login module

1)The front-end code（login.html）`

<body>

<div id="login">

<h1>Login</h1>

<form method="POST" action="/login"> #Through the form method, the user clicks a button and sends a POST request to the login interface

<input type="text" required="required" placeholder="user" name="user"> #The user name of the input tag

<input type="password" required="required" placeholder="password" name="password">#The input tag is password

<button class="but" type="submit">login</button>

<p style="color: red">{{ result}}</p> #If the password is incorrect, a message is displayed

</form>

</div>

</body>

Explanation:This is the HTML for the login screen, which basically allows the setup box for the user to enter his username and password. The function of input is to display the user and password prompts in the box, so that the user can know the exact location to fill in his account information.

When creating the input username and password, set the input type to text and password, respectively, so that when the input type is text, the input data is displayed in clear text (visible). However, in the password, it is in the form of dark text (the input can only see’\*’), so that users can ensure certain security when entering information, so that the password is not easy to show to others.

1. The back-end code（app.py）

@app.route('/')

def login():

return render\_template("login.html")#Log in to the main page if requested

@app.route('/login', methods=['GET','POST'])

def index():

if request.method == "POST": #Determines whether this is a POST request?

session['username'] = request.form['user'] #Gets the user name entered by the user

session['password'] = request.form['password'] #Gets the password entered by the user

conn = sqlite3.connect('products.db') # connect with the database

c = conn.cursor()

sql ="select flag from users where username='{}' and password ='{}'".format(session['username'],session['password']) #ready for the database

c.execute(sql) #Executing database commands

res = c.fetchall() #Gets the query results of the database

c.close() #close the database

if res: #If the query results in the database, determine whether the returned administrator？

if res[0][0] == '1': # ‘1’ means administrator

res = all\_book() #Query information about all books in the database

return render\_template("admin.html", name=session['username'], res=res) #Return the result to the front (HTML) and display it

elif res[0][0] == "2": #Check whether you are a non-administrator (customer)

res = all\_book() #Query information about all books in the database

init\_record(session['username']) #Clean up the user's record table

return render\_template("customer.html", name=session['username'], res=res) #Return query results to HTML (customer)

else: #If the user's username and password are not stored in the database, return 'password error'

return "password error"

else:#Will return an error message to HTML (customer)

return render\_template("login.html", result="error")

else: #get request

conn = sqlite3.connect('products.db') #Connecting to a Database

c = conn.cursor() #Connect the cursor table

sql = """select flag from users where username='{}' and password ='{}'""".format(session['username'],session['password']) #The current and password are obtained through session and stored in SQL

c.execute(sql) #Execute SQL

res = c.fetchall() #Gets the results of a database query

c.close() #close the database

if res:

if res[0][0] == '1': #'1'is administrator

res = all\_book() #Query the books in the library

return render\_template("admin.html", name=session['username'], res=res)#Return the result to HTML (customer)

elif res[0][0] == "2" : #'2' is non-administrator(customer)

res = all\_book()#Query the books in the library

return render\_template("customer.html", name=session['username'], res=res) #Return error messages to HTML (customer)

else:

return "password error" #Return error messages to HTML (customer)

Explanation:This is the Python code in the background. The method is to use the user name and password imported from the front end, and then compare with the data stored in the back-end database. If the match is successful, the main interface of the store is directly jumped. However, if the match is not successful, an error message is displayed. In addition, when the login is successful, because when storing user information in make\_db file, use different flag (database language), directly determine the user predetermine whether the login person is an administrator or a customer, and return the product information to the front end (different HTML for administrator and customer).

Session statements are called 'session control', and the purpose is that the statements stored in the session are not lost when the user jumps from web page to web page, and remain for the duration of the user's session.It is used in the code 'session['username'] = request.form['user']'.

The format statement is a python function used to format a string. In the code, the value of flag is overwritten by the value entered by the original user, allowing the computer to determine the type of user logged in

Fetchall () is a python function used to query databases. It returns multiple arrays, that is, multiple rows of records. In the program I use fetchall to get the results of the database query data. However, using fetchone() only fetches one data, which is not what I want

The idea of identifying user categories is as follows: in the database, the flag of administrator and customer is 1,2 respectively. Then use the database statement to take out the flag of different users, Then, a two-dimensional array is used to let the computer identify the different flags of the members of different categories, so that different HTML can be returned to different members

If you want to see the elements in the array, you just type print (res) underneath the array.

1. The database code(make\_db.py)

def all\_book():

con = sqlite3.connect('products.db') #Connecting to a Database

sql = "select bookname,isbn,images,id,saleprice from books " #The SQL statement

res = con.execute(sql).fetchall() #Execute SQL statement

code\_list = list\_of\_groups(res,5) #Divide the data into 5 bits and group them into lists

con.close() #Close the database

return code\_list #Closing the database

def init\_record(user): #Initializes the shopping cart record

con = sqlite3.connect('products.db') #Linked database

sql = "delete from record where username='{}'".format(user) #SQL statement to delete goods

con.execute(sql) #Execute database statement

con.commit() #Submit data to the database

con.close() #close the database

Explanation:

This is the code to manage the database. The main function is to execute the database statement, respectively, the execution of the query has been input books and the initial user's shopping cart record. The idea can be divided into "connect to database/enter database specific statements/execute database statements/commit database/disconnect database/

When defining code\_list, all goods are arranged in a group of five books on the customer's shopping front page, so data should be divided into five groups for storage.

In the initialization user shopping cart records, the main purpose is, every time when the user login, the shopping cart list cleaning initialization (shopping cart), it is because a user increase goods and lead to shopping cart record record errors in the table, the user may be the wrong choice of goods and quantity, bring unnecessary trouble.

1. Administrator module

The front-end code(add.html)

<ul class="nav nav-tabs" style="background-color: #e55b00" > # Define the navigation bar color

<li class="active"><a href="#">Home</a></li> # home page button

<li><a href="/">Logout</a></li> #log out button

<img src="static/css/1.png" width="30px" height="30px" align="right" ><li> #picture of logo

</ul>

</div>

<div>

<p style="font-size:20px;color:red">User:{{name}}</p> #Displays the current logged-in person

<a href="/addadmin"><input type="button" class="my-btn-blue" value="Add Stock"></a> #Add buttons for goods

<!-- list -->

<table align="center" width="1000"> #Control the items displayed in each row by table

{% for j in res %}

<tr >

{% for i in j %}

<td width="250">

<p align="center"><img src={{i[2]}} height="100" width="100" /></p> #display the price of goods，’align’Used to align the contents of div elements

<h3 align="center">{{i[0]}}</h3> #dispaly the price of goods

<p align="center">{{i[1]}}</p> #dispaly the name of goods

</td>

{% endfor %}

</tr>

{% endfor %}

</table>

Adding a Product page(add.html)

<form action="/add" method="post" enctype="multipart/form-data"> #When the administrator submits the product information to add interface through POST request

<table>

<tr >

<td >

<p align="left">bookname<a style="color:red">\*</a></p> #display book name and set’\*’ in red

</td>

<td>

<input type="text" id="bookname" name = "bookname" align="right"/>#Enter the box for the title of the book

</td>

</tr>

<tr>

<td>

<p align="left">author<a style="color:red">\*</a></p>#display ‘author’ and set ‘\*’ in color red

</td>

<td>

<input type="text" align="right" name="author" id="author" />#Enter the author

</td>

</tr>

<tr>

<td>

<p>create\_time<a style="color:red">\*</a></p>#display ‘cerate\_time’ and set ‘\*’ in color red

</td>

<td>

<input type="date" name = "create\_time" value="2021-11-11"/>#use “data” type to Implement a calendar to display the time

</td>

</tr>

<tr>

<td>

<p>ISBN-13<a style="color:red">\*</a></p>#display ‘ISBN-13’ and set ‘\*’ in color red

</td>

<td>

<input type="text" name = "ISBN" id="isbn" /># put isbn-13 number

</td>

</tr>

<tr>

<td>

<p>describe</p>

</td>

<td>

<textarea rows="10" name="describe" cols="22"> #use tectarea rows to set an area to put the detail about the book

</textarea>

</td>

</tr>

<tr>

<td>

<p>saleprice(0-100)<a style="color:red">\*</a></p>#display ‘saleprice’ and set ‘\*’ in color red

</td>

<td align="left">

<input id="range2" type="range" min="0" max="100" name="saleprice" value="5"step="1" oninput="change2()" onchange="change2()"> #Sliding sales price

<span id="value2">0</span>

</td>

</tr>

<tr>

<td>

<p>retailprice (0-100)<a style="color:red">\*</a></p>#display ‘retailprice’ and set ‘\*’ in color red

</td>

<td align="left">

<input id="range" type="range" min="0" max="100" name="retailprice" value="5"step="1" oninput="change()" onchange="change()"> #Sliding retail prices

<span id="value">0</span>

</td>

</tr><tr>

<td>

<p>count(0-20)<a style="color:red">\*</a></p>#display ‘count’ and set ‘\*’ in color red

</td>

<td align="left">

<input id="range1" type="range" min="0" max="100" name="count" value="5"step="1" oninput="change1()" onchange="change1()">#Sliding number of book(how many books)

<span id="value1">0</span>

</td>

</tr>

<tr>

<td>

<p>picture<a style="color:red">\*</a></p> #display ‘picture’ and set ‘\*’ in color red

</td>

<td align="right">

<input type="file" name="file" id="file" data-max\_size="1000000" accept="image/\*" > #upload the picture

</td>

</tr>

</table>

<div >

<button class="my-btn-blue" style="width:100px;height:20px" >save</button> # button of save the detail that we just input

</div>

</form>

Explanation:

This is the front end of the administrator module code, the application of HTML technology to achieve the web page, the main realization of the administrator in the front end to manage the specific information of goods.

In HTML, I used a loop to walk through the books and have them displayed separately on the front page. {% for j in res %} is to extract a single book from all books in the database. {% for i in j %} is to extract product information from a single book. So I used a double loop for my purpose.

Similarly, when displaying specific information, I still use the way of array to display the information stored in the database at the front end

Using HTML methods, such as' text ', 'data', 'file' and other methods to achieve input data. And realize the loading of commodity information in a page, and annotate each box corresponding to the loading data.

When designing sliders (sliders are adjusting values). Here I used the HTML 'range' function, which allows the value to be changed by dragging the slider, and set the maximum and minimum values. The 'onInput' event is emitted when the user enters a value, whereas the onchange event is emitted when the element loses focus.

1. The back-end code

（app.py)

@app.route('/add', methods=['GET','POST']) #Add the interface of the product

def add(): #Define the add

if request.method == "POST": #Determines whether this is a POST request

conttent = [] #Define conttent as an empty list

f = request.files['file'] #Get photos submitted by the user

path = "./static/file/{}".format(f.filename) #The name of the construct file

f.save(path) #Save file picture

for k,v in request.form.to\_dict().items(): #Get the item information label submitted by the front end saved to the list

conttent.append(v) #Save to the list 'conttent'

conttent.append(path) #Save the path to the list 'conttent’

add\_book(conttent) #Insert the book information into the book's database

return render\_template("success.html") #Returns ‘success. Html’

@app.route('/upload', methods=['GET','POST']) #Interface for uploading files

def upload(): #Define the upload

if request.method == "POST": #Determines whether this is a POST request

f = request.files['file'] #Get file name

f.save("./static/file".strip()) #Get file name

@app.route('/addadmin', methods=['GET','POST']) #Adding a jump page

def addadmin():

return render\_template("add.html") #Return to the Add item page

Explanation:

This part is the backstage code of the administrator module, using Python to write methods. Includes the interface to add pictures, upload files and jump to add pages.

The main idea of add interface is that after the user submits the book information, it obtains the submitted picture first and saves the picture in the local path in the path. At the same time, other information of the goods submitted by the front end will be put into 'V' in a circular way. Finally, path and V will be added to the initial newly created empty list conttent. Finally, the Conttent table is inserted into the add\_book method (in make\_db.py), and when add\_book puts the basic book information into the database, the result is returned to the front end.

The main idea of the file upload interface is that after uploading information, the user obtains the file name and saves the image in the static or file directory.

The main function of the jump page is to return to 'add.html'.

1. The database code(make\_db.py)

def add\_book(content): #add the book detail into database

con = sqlite3.connect('products.db') #connect to database

sql = "insert into books values(null,'{}','{}','{}','{}','{}',{},{},{},'{}')".format(content[0],content[1],content[2],content[3],content[4],content[5],content[6],content[7],content[8]) #Insert the obtained data into the database

print(sql)

con.execute(sql) #Execution SQL

con.commit() #Submit SQL

con.close() # close the database

Explanation:This is the management database code, the main purpose is to insert the commodity information into the database, the main idea is to connect to the database, and execute SQL statements, insert the commodity information into the book table. Because there are nine data (book name/ author/ Create time/ ISBN-13/ ISBN-13/ describe/ salePrice/retailPrice/count/ picture path), So I created nine places to store the data. SQL is executed and committed, and the database is closed.

This method basically commits the data retrieved in the back-end code '/add' to the database.

3.The customer module

1. The front-end code

(customer.html)

<p >User:{{name}}</p> #Display the login name on the page

<a href="/car" > <button> shopping car</button></a> #Shopping cart button

<table align="center" width="1000"> #Display the book goods that have been stored

{% for j in res %}

<tr >

{% for i in j %}

<td width="250">

<p align="center"><img src={{i[2]}} height="100" width="100" /></p> #Show pictures of books and merchandise

<h3 align="center">{{i[0]}}</h3> #Show the price of book goods

<p align="center">${{i[4]}}</p> #The price of goods in books

<p align="center" ><input align="center" type="button" id="{{i[3]}}" value="Add" onmousedown= "down(id)" onclick="changeValue(this)"/></p> #The add button

</td>

{% endfor %}

</tr>

{% endfor %}

</table>

Explanation:

This is the front end code of the customer module, which makes the shopping cart homepage of the customer (Custome1/2), including the user name and specific information of the goods. Here I use a double loop, the database of books in the specific information displayed on the home page. In the operation of retrieving commodity information, I used the method of array to extract the information stored in different locations in the array (book information stored in the database) and display it in the place I set.

When designing the Add button, I still use the attribute of bottom in HTML to realize the function of Add.

1. The back-end code

(app.py)

@app.route('/record', methods=['GET','POST']) #Add an item to the shopping cart interface

def record(): #Define the ‘record’

if request.method == "POST": #Determines whether this is a POST request

res = int(request.data) #Get product information from the front end

add\_record(session['username'],res) #Inserts a record of a book into the database based on the item ID and user name

return ""

(make\_db.py)

def add\_record(user,id): # Add goods or delete goods records

con = sqlite3.connect('products.db') #Connecting to a Database

sql = "select \* from record where username='{}' and nums = {}".format(user,id) #Queries whether items already exist for the user

res = con.execute(sql).fetchall() #Get the database information and add it to the 'res' array

if res: #The record has been recorded and deleted

sql = "delete from record where username='{}' and nums = {}".format(user,id) #Delete a record from the database

con.execute(sql) #Execute database statement

con.commit() #Submit data to the database

else: #Query the user if the item is not available

sql = "insert into record values({},'{}',1)".format(id,user) #Execute the database statement that saves the record

con.execute(sql) #Execute database statement

con.commit() #Submit data to the database

Comment：This is the back-end code for the customer module, written in Python. It mainly realizes the interface of adding goods to shopping cart, and realizes the operation record of adding goods to shopping cart. For the record of the actions to the shopping cart adding items, first determine whether it is a POST request, then retrieve the items from the front end and connect to the database. Because in the shopping side of the page, the user can choose whether to need the goods. A user who has selected the book item, but does not need it deletes the record from the Record table. If the user does not obtain the item, the item record is added to the Record table.

1. Shopping cart module
2. The front-end code

(shopping.html)

#Display information about items in the shopping cart

<table width=100% height="20%">

<tr>

<td align="center" width=10%>

{{i[0]}} #Product ID Serial number

</td>

<td align="center" width=20%>

{{i[1][0]}} #Name of commodity

</td>

<td width=10%>

<img src="{{i[1][1]}}"> #Commodity images

</td>

<td align="center" width=20%>

{{i[1][2]}} #Commodity prices

</td>

<td width=20% align="center">

<a href="car"><input align="center" type="button" value="del" name="{{i[3]}}" onmousedown="del(name)" ></a>{{i[2]}}<a href="car"><input name="{{i[3]}}" align="center" type="button" value="add" onmousedown="add(name)"></a> #Increase or decrease the number of items button

</td>

<td width=30% align="center">

<a href="./car">

<input align="center" type="button" id="{{i[3]}}" onmousedown="down(id)" #Delete Goods button

value="del">

</a>

</td>

</tr>

</table>

Explanation：

This is the front end code of the shopping cart, use HTML to write website. which mainly contains various information about the goods and buttons to add/subtract goods. Because in the home page of the product, the customer only adds the product they want. So on the shopping cart page, select the quantity of individual items.

When displaying the commodity information, the commodity information in the database is called. Because the commodity information is already stored in the database at the time of addition, the direct call can be displayed directly.

2）The back-end code

（app.py)

@app.route('/car', methods=['GET','POST']) #Interface to get shopping cart information

def car():

list\_num = select\_id(session['username']) #Interface to get cart item information

res = select\_show(session['username'],list\_num) #Find information about a commodity based on the recorded commodity ID

if len(res) == 0: #If there is no record in the cart, 0 is displayed a=0

else:

a = res[-1][4] #If the cart records again, ’a‘ is the total price

return render\_template("shopping.html", res=res,a=a)

@app.route('/delcar', methods=['GET','POST']) #Reduce the interface of book goods

def delcar():

id =int(request.data) #Get the id that needs to be deleted for the item

del\_car(session['username'],id) #Operate the database to reduce the quantity of the book item by one

@app.route('/delcount', methods=['GET','POST']) #Add commodity quantity interface

def delcount():

id =int(request.data) #Gets the ID of the item you want to add

changecount("del",session['username'],id) #The number of book items operating on the database is increased by one

return "a"

@app.route('/clear', methods=['GET','POST']) #Empty the shopping cart interface #Empty shopping cart interface

def clear():

clearcar(session['username']) #Delete the shopping cart record by customer name #Delete the shopping cart interface by customer name

return "a"

Explanation:

This is the background code of the shopping cart module, written in Python. The main realization of goods in the shopping cart increase/decrease/delete.

First, the database looks to see if the user's shopping cart record is available, and if no data is available, the empty cart is displayed. If so, the product information is displayed. Furthermore, the function of each button on the shopping cart page is defined.

Add item button: When the user triggers the add button, it first gets the id of the item and adds one to the quantity of the item in the database. The decrease button reduces the quantity of goods by one.

Delete commodity button: the principle is the same as that of adding a button. When this button is triggered, it will delete the record in the shopping cart according to the customer name.

3)Database code(make\_db.py)

def del\_car(user,id):#Delete the cart record based on the item ID

con = sqlite3.connect('products.db') #Connect to database 'prodects.db'

sql = "delete from record where username = '{}' and id='{}'".format(user,id) #Execute the database statement to delete the record by ID

con.execute(sql) #Execute SQL statement

con.commit() #Submit data to the database

con.close() #Closing the database

def changecount(flag,user,id):

con = sqlite3.connect('products.db') #Connecting to a Database

sql = "select nums from record where username = '{}' and id='{}'".format(user, id) #sql statement:select username and id from database

res = con.execute(sql).fetchall() #Get the value from the database and insert it into the array 'res'

if flag == "add": #If it's increasing, it's increasing by one

num = int(res[0][0]) + 1

else: #If is subtract, subtract one

num = int(res[0][0]) - 1

if num < 0: #If the value is less than 0, it equals 0

num = 0

sql = "update record set nums = {} where username = '{}' and id='{}'".format(num,user, id) #Database statement to upload the resulting number/username/ID to the database

con.execute(sql) #Execute SQL statement

con.commit() #Submit data to the database

con.close() #Closing the database

def clearcar(user):

con = sqlite3.connect('products.db') #Connecting to a Database

sql = "delete from record where username = '{}'".format(user) #Database statement: delete username from record

con.execute(sql) #Execute SQL statement

con.commit() #Submit data to the database

con.close() #Closing the database

Explanation:This is the database code of the shopping cart module, written in Python language, mainly realizes the function of deleting the shopping cart record and adding or removing the button of the shopping cart according to the id of the goods. Delete shopping cart record process, the first is to connect to the database, according to the record table username and ID, the use of database language to delete goods. Because the data in different customer tables is different, it is important to delete records based on username.

The functionality of the buttons increase or decrease in the realization of the shopping cart goods process, first connect to the database, the amount of the goods according to the username and id(The product ID is automatically added to the database when the book information is added, which is different from the ISBN code.) to pick up the goods, and then use the if condition statement judge, if they are increased, the total number of them, if the time is little, is divided into the number greater than 1, the total number of goods minus one, if the number is 0, the total still is zero. Finally, the calculated amount is passed back to the database.

1. Payment module：
2. The front-end code

（apply.html)

<table align="center" width="500" border="1"> #Display information about settlement book products

<tr>

<td>

<div align="center">information:</div>

</td>

<td>

<div align="center">{{info}}</div> #A list of book items

</td>

</tr>

<tr>

<td>

<div align="center">price:</div> #total price

</td>

<td>

<div align="center">{{a}}</div>

</td>

</tr>

<tr>

<td>

<div align="center">fare:</div> #The postage

</td>

<td>

<div align="center">{{fare}}</div>

</td>

</tr>

</table>

<div>

<br/>

</div>

<div align="center">

<table align="center" width="500" > #Enter the bank card information

<tr>

<td>

<div align="center">first name:</div> #The first name

</td>

<td>

<input align="center">

</td>

</tr>

<tr>

<td>

<div align="center">last name:</div> #The last name

</td>

<td>

<input align="center">

</td>

</tr>

<tr>

<td>

<div align="center">car number:</div> #The car number

</td>

<td>

<input align="center">

</td>

</tr>

<tr>

<td>

<div align="center">vcc:</div> #card vcc(Security code)

</td>

<td>

<input align="center">

</td>

</tr>

<tr>

<td>

<div align="center">date:</div>

</td>

<td>

<input type="day" name = "create\_time" value="11/11"/> #Enter the date

</td>

</tr>

</table>

Expatiation:

This is the front end of the payment module code, using HTML code. Mainly realized in the customer payment interface, the user can use the bank card for payment, including filling in the card number/validity period/security/cardholder's name. And in the payment interface, users can check the name/unit price and postage of the product to determine the payment amount.

In the design of the page, borders are used where information is filled in, and examples are given at the expiration date so that customers can fill in the date correctly.

In addition, when designing a web page. For the beauty of the picture. All the information about the amount in the input card number information shown above, in the form of a small table, it is convenient to every information presented to the right customer, to let them to check their payment before the payment amount, or let them to consider whether the total amount of purchase in the range they can afford. And not showing the total price separately actually gives the customer the right to know her information.

1. The back-end code

(app.py)

@app.route('/apply', methods=['GET','POST']) #Payment interface

def apply(): #Define the 'apply' method

list\_num = select\_id(session['username']) #Find the id of all items you need to pay for

res = select\_show(session['username'], list\_num) #Find the ids of all items needed to pay for

info = "" #Define a new string and assign it to None

fare = 3 #The initial definition postage is 3$

all\_list = [ ] # Define an empty list of all\_list

for i in res: #Use the for loop to save the product information to the list for front-end display

tmp\_list = [] #Define an empty list

info = info + i[1][0]+"," #Concatenate all commodity names into strings

tmp\_list.append(i[1][0]) #Adds the prices saved in the database to the list

tmp\_list.append(i[2]) # Adds the address of the picture saved in the database to the list

all\_list.append(tmp\_list) #Save it in the tmp\_list you just defined

if len(res) == 0: #If there is no item, the price is not displayed

a = 0

else:

a = res[-1][4] #Displays the total price of the item

fare = (res[-1][5] -1 )\*1 +fare #Calculate the postage

session['list'] = all\_list #Store the book information in all\_list in session

return render\_template("apply.html",info=info,a=a,fare=fare) #Return to apply. html

(make\_db.py)

def select\_id(user): #Find all item ids in the shopping cart record table

con = sqlite3.connect('products.db') #Connecting to a Database

sql = "select DISTINCT(id),nums from record where username='{}'".format(user) #Returns a different NUMs from the record table, conditional on username.

res = con.execute(sql).fetchall() #Query data in the database and return results

list\_num = {} #Define an empty dictionary

for i in res: #The queried data IDS are stored in the dictionary in a circular manner

list\_num[i[0]]=i[1]

con.close() #Closing the database

return list\_num #Return list\_num

def select\_show(user,list\_num): #Find product information by ID

all\_m = 0 #Initialize variable 'all\_m'

book\_count = 0 #Initialize variable 'book\_count'

count=1 #Initialize variable

con = sqlite3.connect('products.db') #Connecting to a Database

list\_show = [] #Connecting to a Database

for k,v in list\_num.items(): #Use a for loop to save the data to list\_num

sql = "select bookname,images,saleprice from books where id='{}'".format(k) #Database statement to find the bookname/ images/ saleprice of an item by ID

res = con.execute(sql).fetchall() #Query data in the database and return results

res.insert(0,count) #Insert the number of books into the first space

count +=1 #Add the quantity by one

res.append(v) #Add the item to the list

all\_m = all\_m +int(v) \* int(res[1][2]) #Calculate the total price of the book

book\_count = book\_count+int(v) #Quantity of books in technician's shopping cart

res.append(k) #Add the result to the list 'k'

res.append(all\_m) #Add the result to the list 'all\_m'

res.append(book\_count) #Add the result to the list 'book\_count'

list\_show.append(res) #Add the result to the list 'res'

con.close() #Closing the database

return list\_show

Expatiation:

This is the back-end code of the shopping cart module, written in Python. It mainly realized the calculation of the total price and postage of the selected goods in the shopping cart Cheack out interface, and also realized the display of goods in the shopping cart, and display the quantity of different goods purchased. The main idea is to take out the information of the product through the product ID, and calculate the price that the end user needs to pay and the quantity of the product through the calculation of the code, and display it in the web page.

1. **TESTING**

1 TESTING REGIME

**1 Developer name / SID**

Yisong Li 11384841

**2 Project**

This is a simple book shopping website, including the user login interface, the administrator to add book information page and display loaded books interface, also includes the customer to buy books interface and shopping cart interface, also includes check out and payment interface.

**3 Test scope**

Registration

Login

Shopfront

Adding to and removing from cart

Checking out

3.1 out of scope

If the title or review has a “ ' ”sign, the submission will not be successful(in add the book detail)

**4 Test time**

Finish all the test cost about 43min.

**5 Test regime**

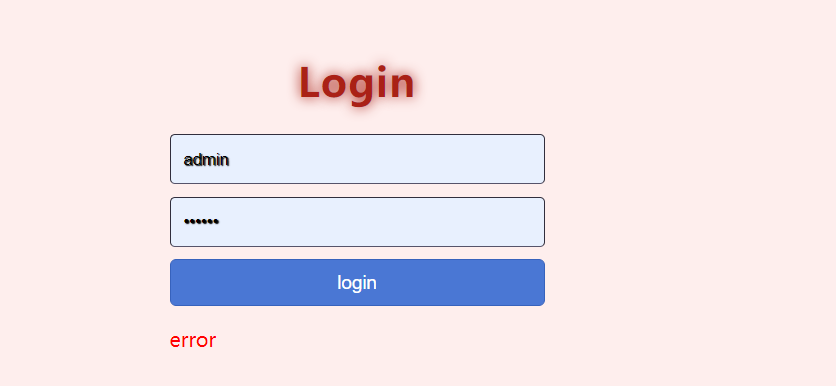
4.1 Registration (10 mins)

On the registration problem, I directly in the database with the admin/customer1 / customer2, no special registration screen, after the test, is successfully adding requirements needed for the account and password.(The code for adding accounts is in the ‘make\_db.py’ file).

1637927331(1)

4.2 Login

After the test, the result of the login, if the account password is not entered in advance, it prompts an error

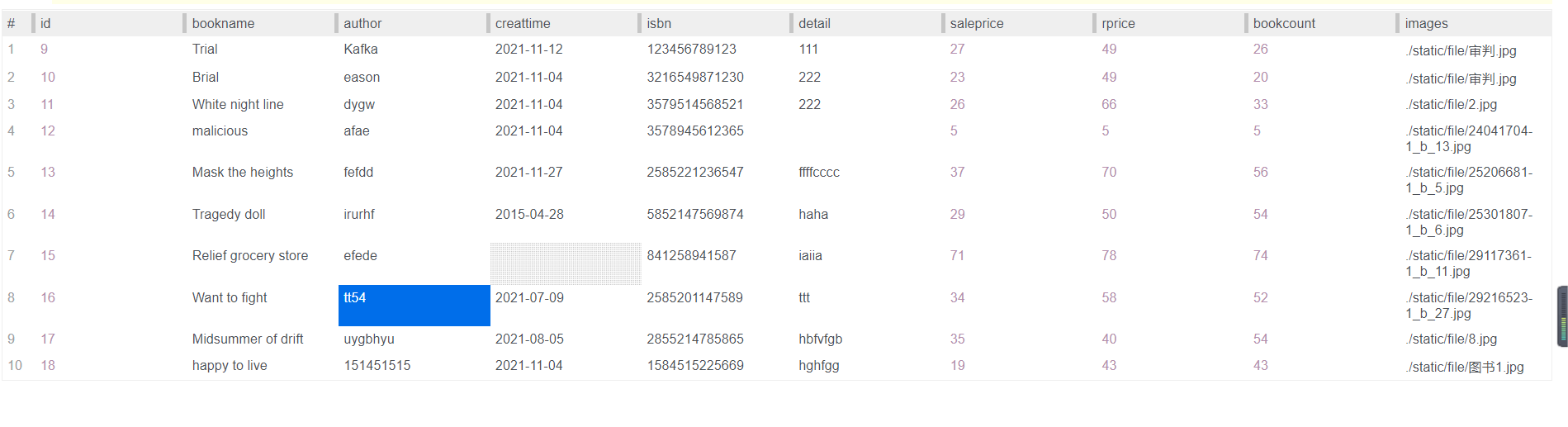


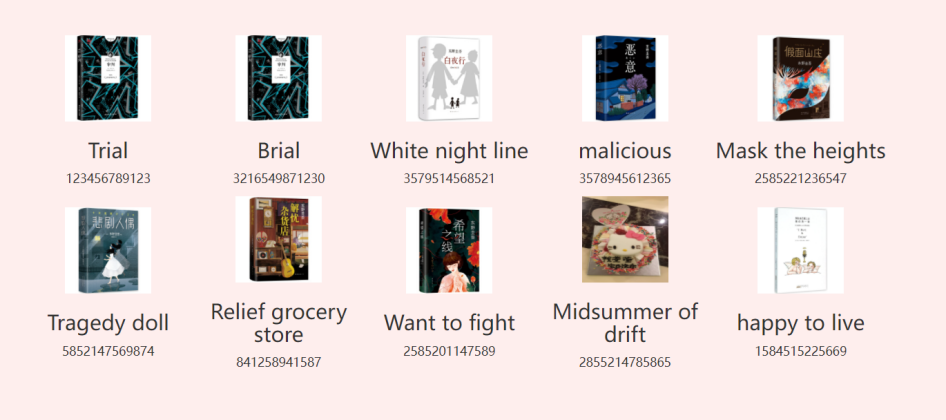
4.3 Shopfront (20 mins)

1 System deals with n-sized data

Test 1: inspect number of database records and cross reference to display. These should match

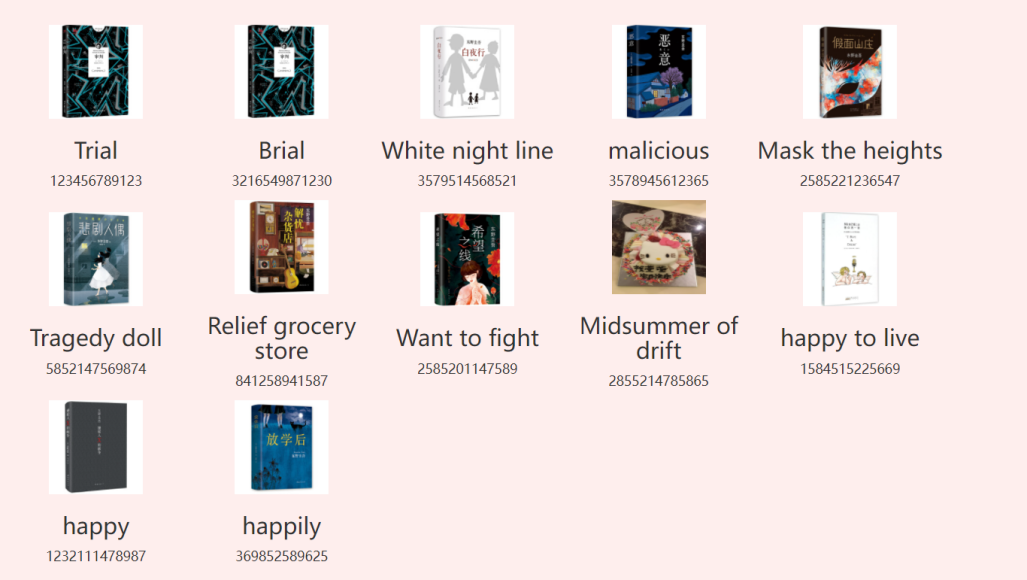
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments | After testing, the achievement is realized |





Test 2: use alternative database with expanded records and repeat Test 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments | After testing, the achievement is realized |

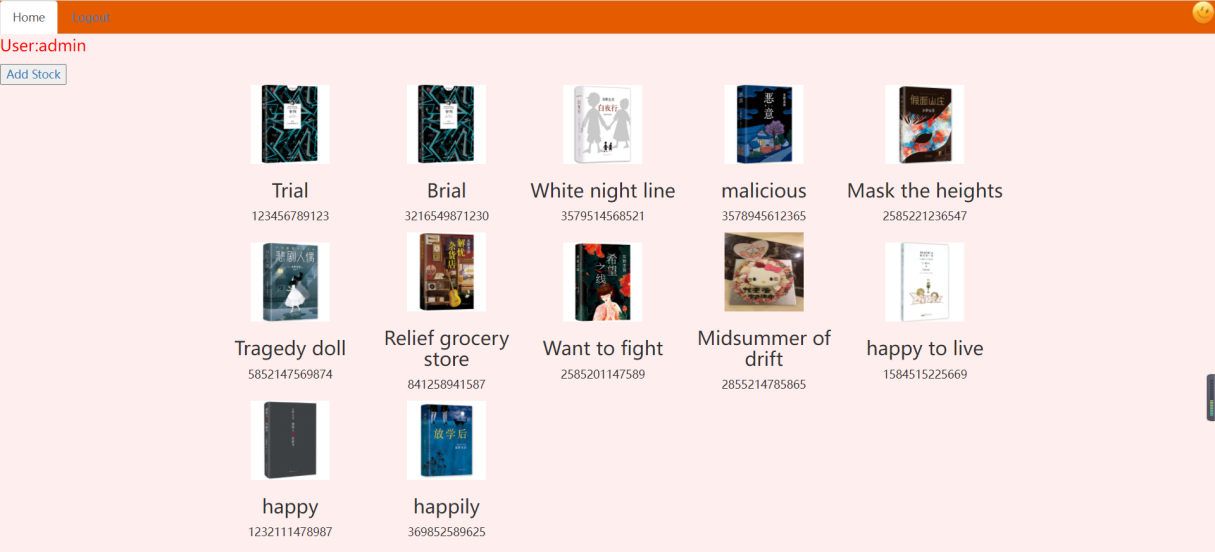




.2 Images are correctly displayed

Test 1: ensure that images are correctly resized and fit bounding box

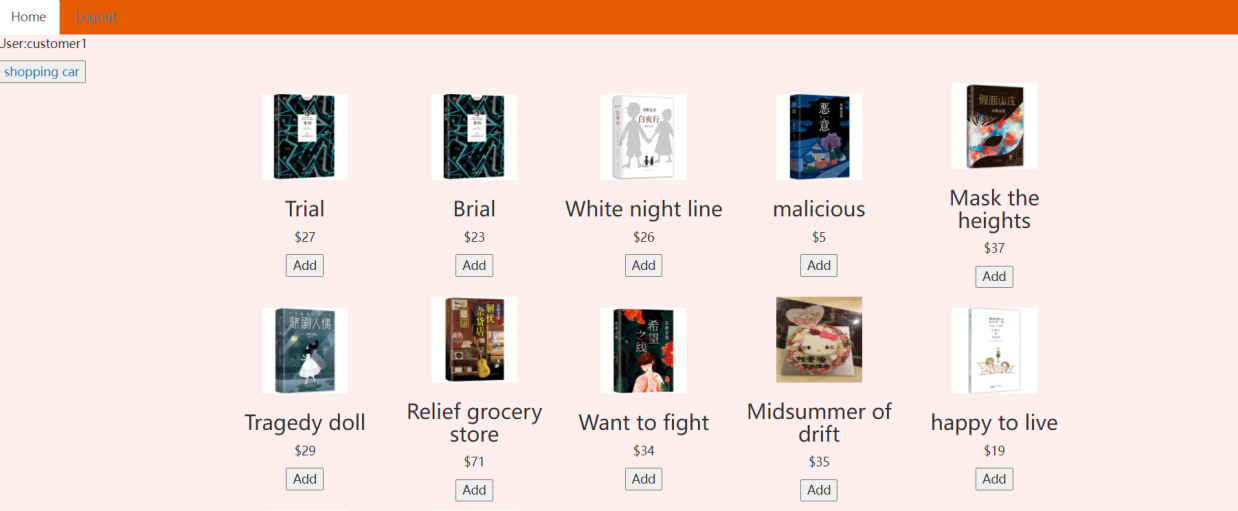
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments | After testing, the achievement is realized |



.3 Currencies are correctly displayed

Test 1: interface should display correct symbol for currency ($, £ etc.)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments | After testing, the achievement is realized |

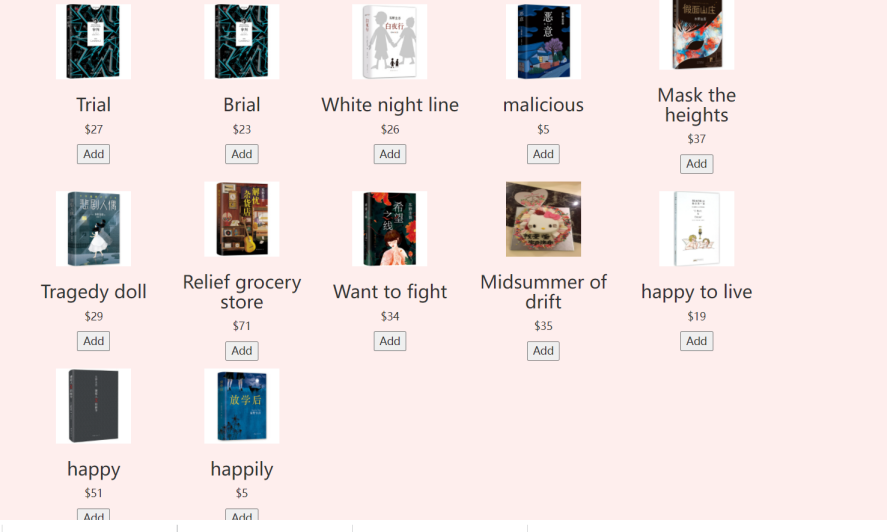


.4 Prices are correctly displayed

Test 1: Amounts should agree with database and be formatted in the same way (2 decimal points)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments | After testing, the achievement is realized |





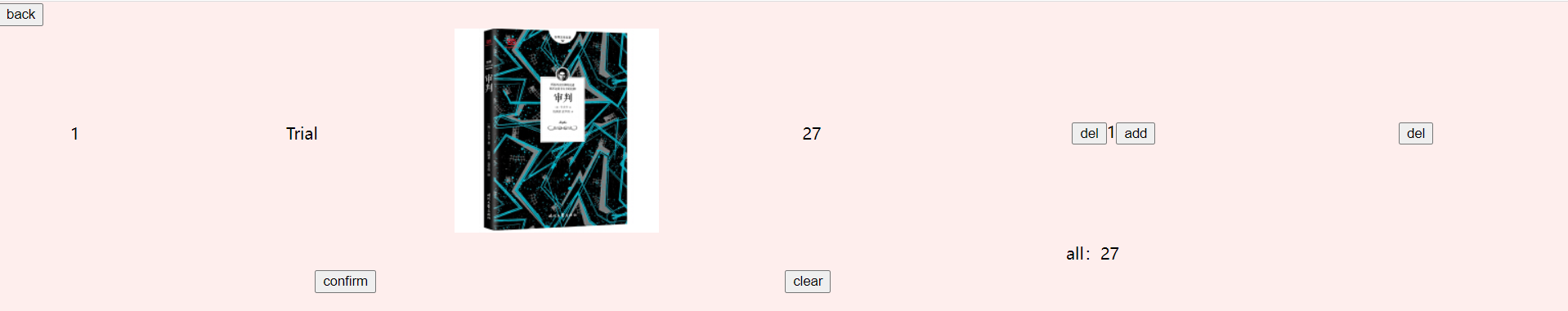
4.4 Shopping basket

.1 Items can be added and removed

Test 1: Add item

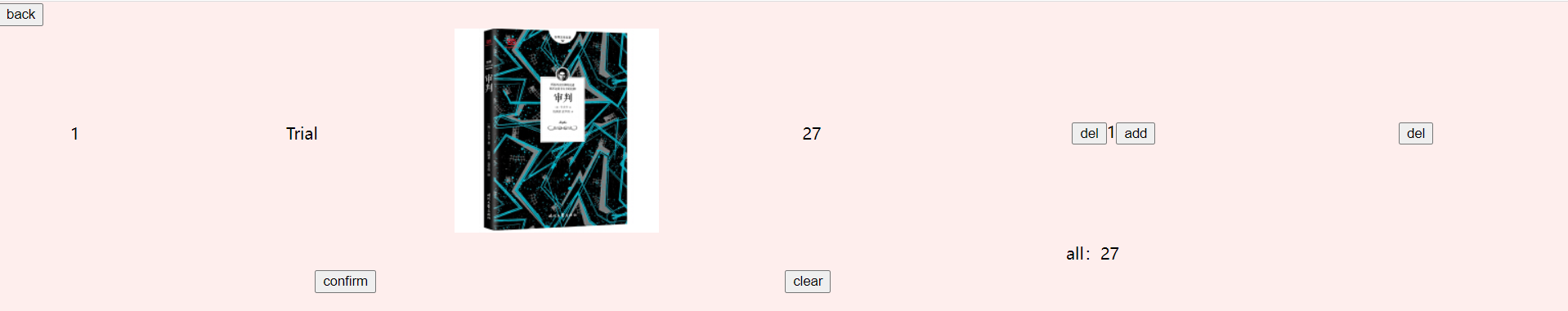
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments | After testing, the achievement is realized |





Test 2: Remove item

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments | After testing, the achievement is realized |

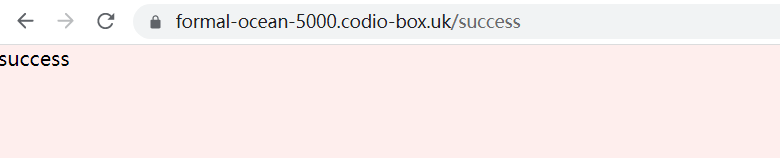
  
 

4.5 Checkout

.1 Payment is accepted

Test 1: Pay for items in basket; payment accept or cancel screen displays

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments | After testing, the achievement is realized |



.2 TESTS RUN ON EXTERNAL CODE

// Run the tests defined by your external (the other team member who has passed you their report)

// Show the output of the tests you’ve run and comment. Are there any tests which have failed? Are there tests which you’d expect to see, but which don’t appear? Are there further tests you’d suggest?

1. **QUALITY ASSURANCE**

.1 QUALITY ASSURANCE STATEMENT

* + **Brief and requirements delivered on**

This is a simple shopping cart web page, which can add product information as the administrator, select products as the customer, increase or decrease the number of products in the shopping cart, and pay with bank card. The shopping cart page I think is not well written. Because a good page is not designed, all the images are not displayed to scale, and the shopping cart will appear not very neatly. Exception: When adding product information, if there is a " ’ " symbol, it will report an error, for which I have not found a good solution.

* + **Coding standards**

The code is basically up to standard. Many of the methods are based on examples on the AULA, but not all are copy and paste. In part, I get ideas or implementation methods by checking website information or asking friends who study computer. The code is relatively simple and basically legible without comments.

* + **Documentation standards**

This is the first time for me to write such a complex document, so I don't think my writing is very formal. Many code notes and explanations are filled in according to my own ideas, which are more colloquial in language statements, so there is no special standard in structure. But I try to explain it in as much detail as possible so that people who have never seen my code can understand it

**Timely delivery**

YES

.2 EXTERNAL QA EVALUATION

// Your evaluation of the external’s QA statement

1. **DOCUMENTATION**

.1 DOCUMENTATION LIST

Section 1 of this report:Statement of code purpose

Section 2 of this report:Statement of code location

Section 3 of this report:Install instructions

Section 4 of this report:Code explanation

Section 5 of this report, also a standalone copy for use by tester: Testing Regime

Section 6 of this report:QA Statement

.2 EXTERNAL DOCUMENTATION INSPECTION

**REFERENCES**

Apply bootstrap to CDN library CSS and JavaScript

<http://www.staticfile.org/>

<https://getbootstrap.com>

Learn more about Flask, including some of the basics:

<https://blog.csdn.net/wangfan741/article/details/109781347>

About GET and POST requests:

<https://www.pianshen.com/article/3965995387/>

HTML <input>, The required attribute of the tag:

<https://www.w3school.com.cn/tags/att_input_required.asp>

Session usage and basics knowledge :

<https://baike.baidu.com/item/session/479100?fr=aladdin>

The difference between fetchone() and fetchall() functions when querying a database in Python:

<https://blog.csdn.net/qq_40625030/article/details/79722996>

The REQUIRED attribute of the INPUT tag in HTML5 prompts text modification:

<https://blog.csdn.net/kakaxiD/article/details/72303103>

How to use.to\_dict():

<https://www.runoob.com/python/att-dictionary-items.html>

Execute () of the Pymyql(python) module:

<https://blog.csdn.net/Kimidake/article/details/90755855>

Oninput event in the HTML template:

<https://www.runoob.com/jsref/event-oninput.html>

The label of the HTML< form >:

<https://www.w3school.com.cn/tags/tag_form.asp>

How do I traverse dictionaries in Python:

<https://realpython.com/iterate-through-dictionary-python/>