

**School of Computing, Engineering and Mathematics (CEM)**

**Faculty of Engineering, Environment and Computing (EEC)**

**5001CEM SOFTWARE ENGINEERING | 2122**

**PROJECT REPORT**

**NAME: Muhammad Zahid Esmat**

**SID: 10323928**

**1.CODE PURPOSE**

This program implements a simple bookshop system on the web. It contains the functions of register, login, logout, add and delete products, and shopping cart.

**2.CODE LOCATION**

https://github.com/zahidm15/BookStore-Flask.git

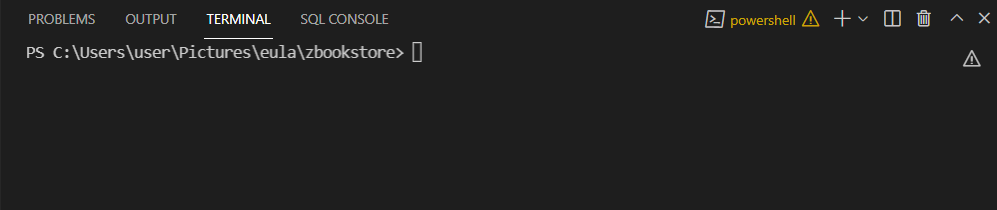
**3.CODE INSTALLATION**

.1 Download code from

<https://github.com/zahidm15/BookStore-Flask.git>.

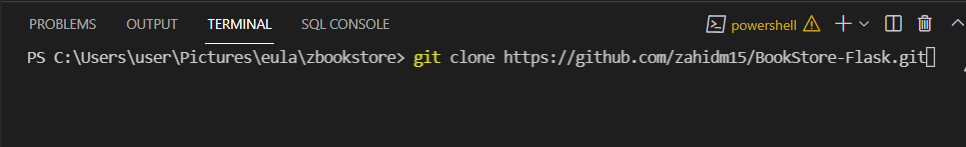
2 Open a new folder on Visual studio Code.

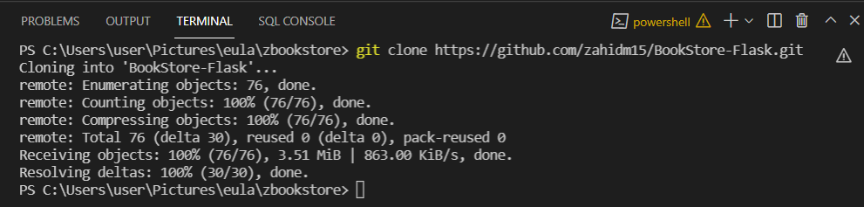
Then open the terminal.



Then follow the instruction below to install something:

Do **git clone** [**https://github.com/zahidm15/BookStore-Flask.gitgit**](https://github.com/zahidm15/BookStore-Flask.gitgit)

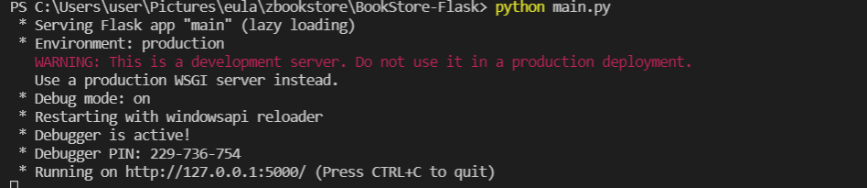




Then do **cd BookStore-Flask** to get into project folder

**python database.py** to install database.

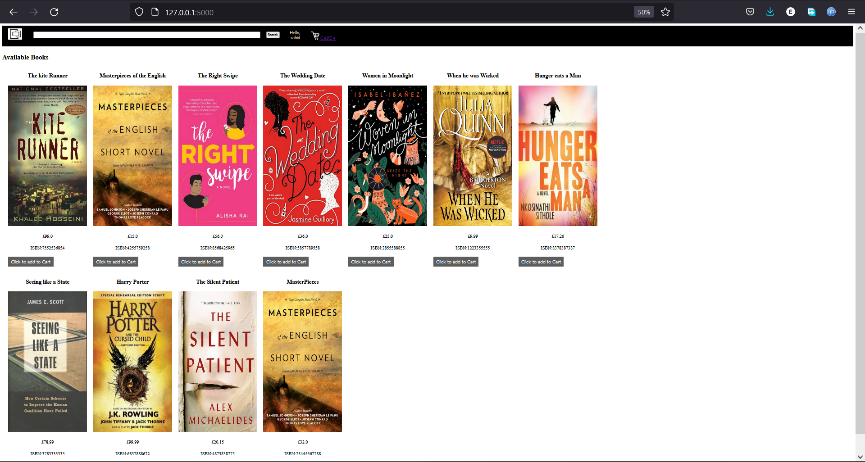
Do **python main.py** to run the main program.



Go to browser and paste the link

[**http://127.0.0.1:5000/**](http://127.0.0.1:5000/)





**4.CODE EXPLANATION**

When run the program, the first page is index page. It is consisting of navigation bar and products part. This code explanation will be divided into two parts, the navigation bar and products part.

**.1 Navigation bar:**

On all pages, I use the same format of the navigation bar. And because clicking on the link on the navigation bar will connect to other pages, so the back-end code will be mentioned in other parts.

* **front end:**

1. <title>Welcome To Our Online Book Store</title>
2. <link rel="stylesheet" href={{ url\_for('static', filename='css/home.css') }} />
3. <link rel="stylesheet" href={{ url\_for('static', filename='css/topStyle.css') }} />
4. </head>
5. <body>
6. <div id="title">
7. <a href="/">
8. <img id="logo" src= {{ url\_for('static', filename='images/logo.png') }} />
9. </a>
10. <form>
11. <input id="searchBox" type="text" name="searchQuery">
12. <input id="searchButton" type="submit" value="Search">
13. </form>
15. {% if not loggedIn %}
16. <div id="signInButton">
17. <a class="link" href="/loginForm">Sign In</a>
18. </div>
19. {% else %}
20. <div class="dropdown">
21. <button class="dropbtn">Hello, <br>{{firstName}}</button>
22. <div class="dropdown-content">
23. <a href="/account/orders">Your orders</a>
24. <a href="/account/profile">Your profile</a>
25. <hr>
26. <a href="/logout">Sign Out</a>
27. </div>
28. </div>
29. {% endif %}
30. <div id="cart">
31. <a class="link" href="/cart">
32. <img src={{url\_for('static', filename='images/shoppingCart.png')}} id="cartIcon" />
33. CART {{noOfItems}} <!-- cart and number of items added to cart -->
34. </a>
35. </div>
36. </div>

Explanation: This is a navigation bar, and all the links are shown on it. There is bookstore logo, sign up link, sign in link, cart link, search bar input text box, and search bar button. The format is made by css.

(1) **Registration link:**

When clicking on the sign up link it will return the sign up page. This page allows customer to register their account.

* **front end:**

1. <html>
2. <head>
3. <title>Registration</title>
4. <link rel="stylesheet" href={{ url\_for('static', filename='css/login.css') }} />
5. <script type="text/javascript" src="{{ url\_for('static', filename = 'js/validateForm.js') }}">
6. </script>
7. </head>
8. <body>
9. <form action="/register" method="POST" onsubmit="return validate()">
10. <legend>REGISTER::</legend>
11. <p>Email: <input type="email" name="email" placeholder="name@gmail.com"></p>
13. <P>Password: <input type="password" name="password" id="password" required></p>
15. <p>Confirm Password: <input type="password" name="cpassword" id="cpassword"></p>
17. <p>First Name: <input type="text" name="firstName"></p>
18. <p>Last Name: <input type="text" name="lastName"></p>
19. <p>Address Line 1: <input type="text" name="address1"></p>
20. <p>Address Line 2: <input type="text" name="address2"></p>
21. <p>Zipcode: <input type="text" name="zipcode"></p>
22. <p>City: <input type="text" name="city"></p>
23. <p>State: <input type="text" name="state"></p>
24. <p>Country: <input type="text" name="country"></p>
25. <p>Phone Number: <input type="text" name="phone"></p>
27. <p><input type="submit" value="Register"></p>
29. <button onclick="location.href='/loginForm'">Login here</button>
30. </form>
31. </body>
32. </html>

* **back end:**

1. @app.route("/register", methods = ['GET', 'POST'])
2. def register():
3. if request.method == 'POST':
4. #Parse form data
5. password = request.form['password']
6. email = request.form['email']
7. firstName = request.form['firstName']
8. lastName = request.form['lastName']
9. address1 = request.form['address1']
10. address2 = request.form['address2']
11. zipcode = request.form['zipcode']
12. city = request.form['city']
13. state = request.form['state']
14. country = request.form['country']
15. phone = request.form['phone']
17. with sqlite3.connect('database0.db') as con:
18. try:
19. cur = con.cursor()
20. cur.execute('INSERT INTO users (password, email, firstName, lastName, address1, address2, zipcode, city, state, country, phone) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)', (hashlib.md5(password.encode()).hexdigest(), email, firstName, lastName, address1, address2, zipcode, city, state, country, phone))
22. con.commit()
24. msg = "Registered Successfully"
25. except:
26. con.rollback()
27. msg = "Error occured"
28. con.close()
29. return render\_template("login.html", error=msg)
31. @app.route("/registerationForm")
32. def registrationForm():
33. return render\_template("register.html")
35. def allowed\_file(filename):
36. return '.' in filename and \
37. filename.rsplit('.', 1)[1] in ALLOWED\_EXTENSIONS
39. def parse(data):
40. ans = []
41. i = 0
42. while i < len(data):
43. curr = []
44. for j in range(7):
45. if i >= len(data):
46. break
47. curr.append(data[i])
48. i += 1
49. ans.append(curr)
50. return ans
52. if \_\_name\_\_ == '\_\_main\_\_':
53. app.run(debug=True)

Explanation: Got the data from the form after the customer registered. If the password is not matched with the password that was entered again, then return the registration page to enter again. If they are matched, then add the username and password into the database and return the login page.

(2) **Login link:**

When registration is successful then return login page or clicks on the login link to go to the login page. Customers were allowed to check their shopping cart after they logged in.

* **front end:**

1. <html>
2. <head>
3. <title> Book Store flask app </title>
4. <link rel="stylesheet" href={{ url\_for('static', filename='css/login.css') }} />
5. </head>
6. <body>
7. <p> {{error}} </p>
8. <form action="/login" method="POST">
9. <legend>LOGIN::</legend>
10. <p>Username: <input class="text" type="text" name="email" placeholder="name@gmail.com"></p>
11. <p>Password: <input type="password" name="password"></p>
12. <p><input class="button" type="submit"></p>
13. <button class="button" type="button" onclick="location.href='/registerationForm'">Register here</button>
14. </form>
16. </body>
17. </html>

* **back end:**

1. @app.route("/loginForm")
2. def loginForm():
3. if 'email' in session:
4. return redirect(url\_for('root'))
5. else:
6. return render\_template('login.html', error='')
8. @app.route("/login", methods = ['POST', 'GET'])
9. def login():
10. if request.method == 'POST':
11. email = request.form['email']
12. password = request.form['password']
13. if is\_valid(email, password):
14. session['email'] = email
15. return redirect(url\_for('root'))
16. else:
17. error = 'Invalid UserId / Password'
18. return render\_template('login.html', error=error)

Explanation: Get the username and password from the form after the customer entered, then use them as the search criteria to select from the database. If the username and password from the database are matched with that were entered by the customer, then return the index page. If not, then return the login page to enter again.

(3) **Logout link:**

When clicking on the logout link it will return the index page, so there is no front-end code of logout.

- **back end:**

1. @app.route("/logout")
2. def logout():
3. session.pop('email', None)
4. return redirect(url\_for('root'))
6. def is\_valid(email, password):
7. con = sqlite3.connect('database0.db')
8. cur = con.cursor()
9. cur.execute('SELECT email, password FROM users')
10. data = cur.fetchall()
11. for row in data:
12. if row[0] == email and row[1] == hashlib.md5(password.encode()).hexdigest():
13. return True
14. return False

(4) **Add product link:**

When clicking on the add product link, it will return a page that allows the administrator to add products. Book name, author, add date, code, image, stock, price, and detail will be allowed to add.

* **front end:**

1. <!DOCTYPE HTML>
2. <html>
3. <head>
4. <title>Admin</title>
5. </head>
6. <body>
7. <h2>Add items</h2>
8. <form action="/addItem" method="POST" enctype="multipart/form-data">
9. Name: <input type="text" name="name"><br>
10. Price: <input type="text" name="price"><br>
11. Description: <textarea name="description" rows=3 cols="40"></textarea><br>
12. Image: <input type="file" name="image"><br>
13. Stock: <input type="text" name="stock"><br>
14. Category: <select name="category">
15. {% for row in categories %}
16. <option value="{{row[0]}}">{{row[1]}}</option>
17. {% endfor %}
18. </select><br>
19. <input type="submit">
20. </form>
21. </body>
22. </html>

* **back end:**

1. @app.route("/add")
2. def admin():
3. with sqlite3.connect('database0.db') as conn:
4. cur = conn.cursor()
5. cur.execute("SELECT categoryId, name FROM categories")
6. categories = cur.fetchall()
7. conn.close()
8. return render\_template('add.html', categories=categories)
10. @app.route("/addItem", methods=["GET", "POST"])
11. def addItem():
12. if request.method == "POST":
13. name = request.form['name']
14. price = float(request.form['price'])
15. description = request.form['description']
16. stock = int(request.form['stock'])
17. categoryId = int(request.form['category'])
18. ISBN=int(request.form['ISBN'])
20. #Uploading image procedure
21. image = request.files['image']
22. if image and allowed\_file(image.filename):
23. filename = secure\_filename(image.filename)
24. image.save(os.path.join(app.config['UPLOAD\_FOLDER'], filename))
25. imagename = filename
26. with sqlite3.connect('database0.db') as conn:
27. try:
28. cur = conn.cursor()
29. cur.execute('''INSERT INTO products (name, price, description, image, stock, categoryId, ISBN) VALUES (?, ?, ?, ?, ?, ?, ?)''', (name, price, description, imagename, stock, categoryId, ISBN))
30. conn.commit()
31. msg="added successfully"
32. except:
33. msg="error occured"
34. conn.rollback()
35. conn.close()
36. print(msg)
37. return redirect(url\_for('root'))

Explanation: This method is used to add products by the administrator. If a product is already in the database, then gets its stock from the database and lets this stock plus the new stock that the administrator added to update the stock. If the product is not in the database, then gets its information and add it to the database.

(5) **Shopping cart link:**

When the user clicks on add to cart button or shopping cart button, it will return the shopping cart page and show the products which are in the shopping cart. This operation can only be done after the customer has logged in.

* **front end:**

1. <!DOCTYPE HTML>
2. <html>
3. <head>
4. <title>Your Cart</title>
5. <link rel="stylesheet" href={{url\_for('static', filename='css/cart.css')}} />
6. <link rel="stylesheet" href={{url\_for('static', filename='css/topStyle.css') }} />
7. </head>
8. <body>
9. <div id="title">
10. <a href="/">
11. <img id="logo" src= {{ url\_for('static', filename='images/logo.png') }} />
12. </a>
13. <form>
14. <input id="searchBox" type="text" name="searchQuery">
15. <input id="searchButton" type="submit" value="Search">
16. </form>
18. {% if not loggedIn %}
19. <div id="signInButton">
20. <a class="link" href="/loginForm">Sign In</a>
21. </div>
22. {% else %}
23. <div class="dropdown">
24. <button class="dropbtn">Hello, <br>{{firstName}}</button>
25. <div class="dropdown-content">
26. <a href="/account/orders">Your orders</a>
27. <a href="/account/profile">Your profile</a>
28. <hr>
29. <a href="/logout">Sign Out</a>
30. </div>
31. </div>
32. {% endif %}
33. <div id="cart">
34. <a class="link" href="/cart">
35. <img src={{url\_for('static', filename='images/shoppingCart.png')}} id="cartIcon" />
36. CART {{noOfItems}}
37. </a>
38. </div>
39. </div>
40. <div id="cartItems">
41. <h2>Shopping Cart</h2>
42. <div id="tableItems">
43. {% for row in products %}
44. <div>
45. <hr id="seperator">
46. <div id="itemImage">
47. <img src={{url\_for('static', filename='uploads/'+row[3])}} id="image"/>
48. </div>
49. <div id="itemName">
50. <span id="itemNameTag">{{row[1]}}</span><br>
51. In stock<br>
52. <a href="/removeFromCart?productId={{row[0]}}">Remove</a>
53. </div>
54. <div id="itemPrice">
55. ${{row[2]}}
56. </div>
57. </div>
58. {% endfor %}
59. <hr id="seperator">
60. <div id="total">
61. <span id="subtotal">Subtotal</span> : ${{totalPrice}}
62. </div>
63. </div>
64. </div>
65. <button onclick="location.href='/checkout'">Proceed to checkout</button>
66. </body>
67. </html>

* **back end:**

1. @app.route("/addToCart")
2. def addToCart():
3. if 'email' not in session:
4. return redirect(url\_for('loginForm'))
5. else:
6. productId = int(request.args.get('productId'))
7. with sqlite3.connect('database0.db') as conn:
8. cur = conn.cursor()
9. cur.execute("SELECT userId FROM users WHERE email = '" + session['email'] + "'")
10. userId = cur.fetchone()[0]
11. try:
12. cur.execute("INSERT INTO cart (userId, productId) VALUES (?, ?)", (userId, productId))
13. conn.commit()
14. msg = "Added successfully"
15. except:
16. conn.rollback()
17. msg = "Error occured"
18. conn.close()
19. return redirect(url\_for('root'))
21. @app.route("/cart")
22. def cart():
23. if 'email' not in session:
24. return redirect(url\_for('loginForm'))
25. loggedIn, firstName, noOfItems = getLoginDetails()
26. email = session['email']
27. with sqlite3.connect('database0.db') as conn:
28. cur = conn.cursor()
29. cur.execute("SELECT userId FROM users WHERE email = '" + email + "'")
30. userId = cur.fetchone()[0]
31. cur.execute("SELECT products.productId, products.name, products.price, products.image FROM products, cart WHERE products.productId = cart.productId AND cart.userId = " + str(userId))
32. product = cur.fetchall()
33. totalPrice = 0
34. for row in product:
35. totalPrice += row[2]
36. return render\_template("cart.html", product = product, totalPrice=totalPrice, loggedIn=loggedIn, firstName=firstName, noOfItems=noOfItems)

Explanation: This add\_to\_shopping\_cart method is used to add products to the shopping cart. First, the customer must be logged in the system uses the customer id to assign bought books. Whatever number of books the customer selects it will be referenced to his or her account in cart table in the database.

(6) **Remove to cart**

By clicking on remove to cart button the system will open a page where the customer will view all selected books in the cart and allow customer to remove products from cart.

* **Front end:**

1. <!DOCTYPE HTML>
2. <html>
3. <head>
4. <title>Remove</title>
5. <link rel="stylesheet" href={{url\_for('static', filename='css/remove.css')}}>
6. </link>
7. </head>
8. <body>
9. <table>
10. {% for i in range(6) %}
11. <tr>
12. {% for row in data %}
13. <td>
14. <a href="/removeItem?productId={{row[0]}}">
15. {% if i == 4 %}
16. <img src={{ url\_for('static', filename='uploads/' + row[i]) }} id="itemImage" />
17. {% else %}
18. {{row[i]}}
19. {% endif %}
20. </a>
21. </td>
22. {% endfor %}
23. </tr>
24. {% endfor %}
25. </table>
26. </body>
27. </html>

**-back end:**

1. @app.route("/remove")
2. def remove():
3. with sqlite3.connect('database0.db') as conn:
4. cur = conn.cursor()
5. cur.execute('SELECT productId, name, price, description, image, stock ISBN FROM products')
6. data = cur.fetchall()
7. conn.close()
8. return render\_template('remove.html', data=data)
10. @app.route("/removeItem")
11. def removeItem():
12. productId = request.args.get('productId')
13. with sqlite3.connect('database0.db') as conn:
14. try:
15. cur = conn.cursor()
16. cur.execute('DELETE FROM products WHERE productID = ' + productId)
17. conn.commit()
18. msg = "Deleted successsfully"
19. except:
20. conn.rollback()
21. msg = "Error occured"
22. conn.close()
23. print(msg)
24. return redirect(url\_for('root'))

Explanation: the function enables the customer to remove products from his shopping cart.

(7) **product description:**

This page displays more information about the book selected like book description and category.

* **front end:**

1. <!DOCTYPE HTML>
2. <html>
3. <head>
4. <title>Product Description</title>
5. <link rel="stylesheet" href={{url\_for('static', filename='css/productDescription.css')}} />
6. <link rel="stylesheet" href={{ url\_for('static', filename='css/topStyle.css')}} />
7. </head>
8. <body>
9. <div id="title">
10. <a href="/">
11. <img id="logo" src= {{ url\_for('static', filename='images/logo.png') }} />
12. </a>
13. <form>
14. <input id="searchBox" type="text" name="searchQuery">
15. <input id="searchButton" type="submit" value="Search">
16. </form>
18. {% if not loggedIn %}
19. <div id="signInButton">
20. <a class="link" href="/loginForm">Sign In</a>
21. </div>
22. {% else %}
23. <div class="dropdown">
24. <button class="dropbtn">Hello, <br>{{firstName}}</button>
25. <div class="dropdown-content">
26. <a href="/account/orders">Your orders</a>
27. <a href="/account/profile">Your profile</a>
28. <hr>
29. <a href="/logout">Sign Out</a>
30. </div>
31. </div>
32. {% endif %}
33. <div id="cart">
34. <a class="link" href="/cart">
35. <img src={{url\_for('static', filename='images/shoppingCart.png')}} id="cartIcon" />
36. CART {{noOfItems}}
37. </a>
38. </div>
39. </div>
40. <div id="display">
41. <div id="productName">
42. <h1>{{data[1]}}</h1>
43. </div>
44. <div>
45. <img src={{url\_for('static', filename='uploads/'+data[4]) }} id="productImage"/>
46. </div>
48. <div id="productDescription">
49. <h2>Details</h2>
50. <table id="descriptionTable">
51. <tr>
52. <td>Name</td>
53. <td>{{data[1]}}</td>
54. </tr>
55. <tr>
56. <td>Price</td>
57. <td>${{data[2]}}</td>
58. </tr>
59. <tr>
60. <td>Stock</td>
61. <td>{{data[5]}}</td>
62. </tr>
63. <tr>
64. <td>ISBN:</td>
65. <td>{{ data[6] }}</td>
66. </tr>
67. </table>
68. <h2>Description</h2>
69. <p>{{data[3]}}</p>
70. </div>
71. <div id="addToCart">
72. <a href="/addToCart?productId={{request.args.get('productId')}}">Add to Cart</a>
73. </div>
74. </div>
75. </body>
76. </html>

* back end:

1. @app.route("/productDescription")
2. def productDescription():
3. loggedIn, firstName, noOfItems = getLoginDetails()
4. productId = request.args.get('productId')
5. with sqlite3.connect('database0.db') as conn:
6. cur = conn.cursor()
7. cur.execute('SELECT productId, name, price, description, image, stock, ISBN FROM products WHERE productId = ' + productId)
8. productData = cur.fetchone()
9. conn.close()
10. return render\_template("productDescription.html", data=productData, loggedIn = loggedIn, firstName = firstName, noOfItems = noOfItems)

Explanation: the function displays more book details on a page that was not displayed on the home page.

(8) check out

This page displays the customer bill summary.

* front end:

1. <!DOCTYPE HTML>
2. <html>
4. <head>
5. <title>Admin</title>
6. </head>
8. <body>
9. <p>Your order has been successfully placed</p>
10. <div id="cartItems">
11. <h2>The items you have ordered</h2>
12. <div id="tableItems">
13. {% for row in products %}
14. <div>
15. <hr id="seperator">
16. <div id="itemName">
17. <span id="itemNameTag">{{row[1]}}</span><br>
18. </div>
19. <div id="itemPrice">
20. ${{row[2]}}
21. </div>
22. </div>
23. {% endfor %}
24. <hr id="seperator">
25. <div id="total">
26. <span id="subtotal">Your total order size is</span> : ${{totalPrice}}
27. <span if="deliverydate"></span>
28. </div>
29. <div><a href="/">Return to Home Page</div>
30. </div>
31. </div>
32. </body>
33. </html>

-back end:

1. @app.route("/checkout", methods=['GET','POST'])
2. def payment():
3. if 'email' not in session:
4. return redirect(url\_for('loginForm'))
5. loggedIn, firstName, noOfItems = getLoginDetails()
6. email = session['email']
8. with sqlite3.connect('database0.db') as conn:
9. cur = conn.cursor()
10. cur.execute("SELECT userId FROM users WHERE email = '" + email + "'")
11. userId = cur.fetchone()[0]
12. cur.execute("SELECT products.productId, products.name, products.price, products.image FROM products, cart WHERE products.productId = cart.productId AND cart.userId = " + str(userId))
13. product = cur.fetchall()
14. totalPrice = 0
15. for row in product:
16. totalPrice += row[2]
17. print(row)
18. cur.execute("INSERT INTO Orders (userId, productId) VALUES (?, ?)", (userId, row[0]))
19. cur.execute("DELETE FROM cart WHERE userId = " + str(userId))
20. conn.commit()

**5.TESTING**

**.1 TESTING REGIME**

**1.Developer name / SID**

**Muhammad Zahid Esmat**

**10323928**

­

**2.Project**

This project implements a simple book shop on the web. There is registration, login logout, add products, shopping cart, and payment module. Also, can show the tables in the database on web, but just the administrator can execute it.

**3.Test scope**

Registration

Login

Shopfront

Adding to and removing from cart

Checking out

**3.1 out of scope**

Enter a wrong number of the quantity. And return an alert.

When entered a wrong type of stock or price.

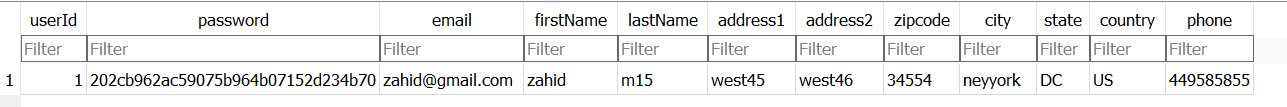
**4.Test time**

45 minutes

**5.Test regime**

**5.1 Registration**

Customers can register successfully on the registration module, but the problem is the program cannot recognize a username whether is registered or not, which means there might be two or more of the same usernames in the database.



**5.2Login**

Login module can run successfully. When entered a wrong username or password, it will return an alert.



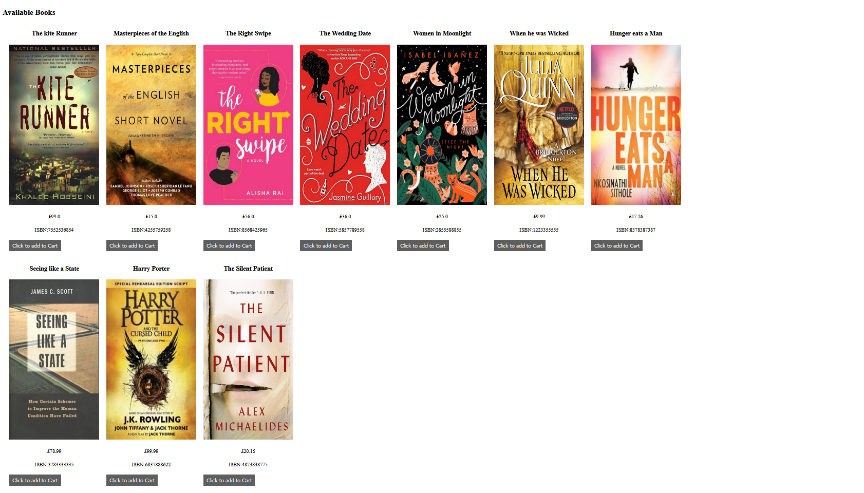
**5.3Shopfront**

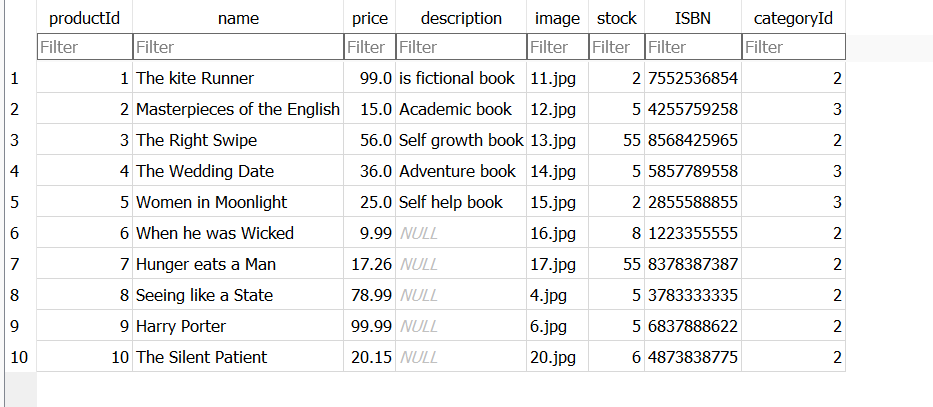
**5.3.1 System deals with n-sized data**

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

Pass Yes / No Fail Yes / No

Comments: The data in the database are matched with the items displayed on the web.

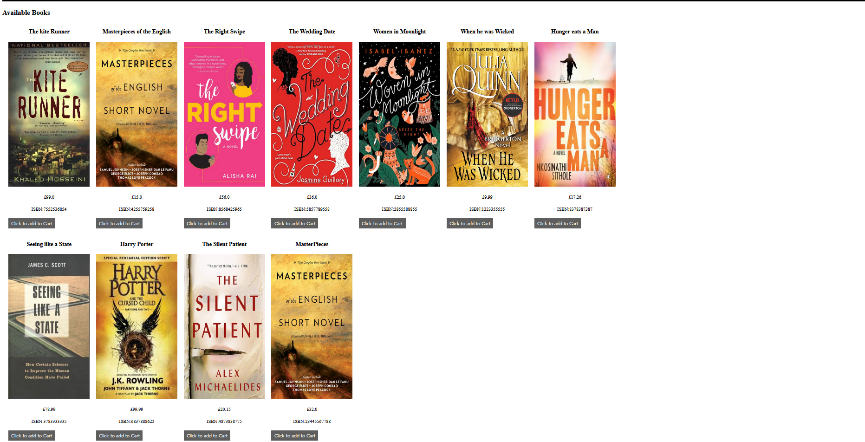


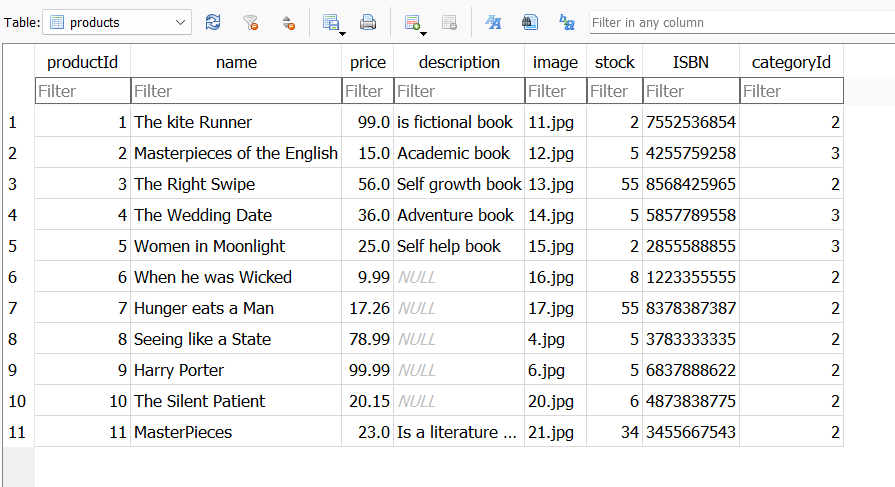


Test 2: use alternative database with expanded records and repeat Test 1

Pass Yes / No Fail Yes / No

Comments: Added one more item, the items displayed on the web is matched with the data in database.



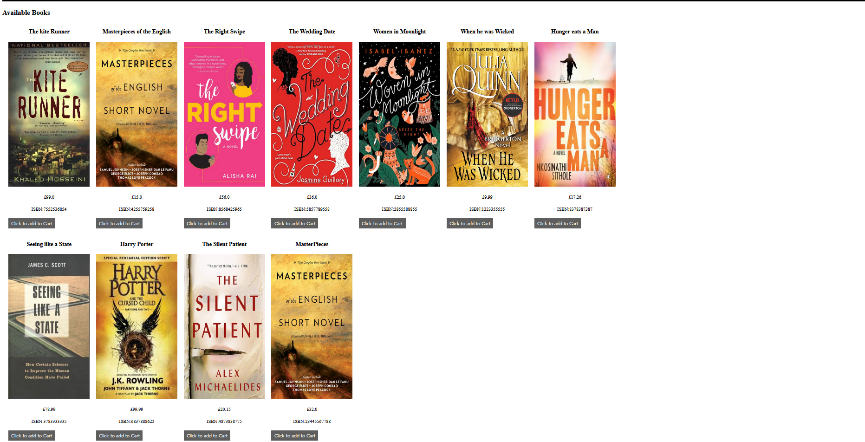


**5.3.2 Images are correctly displayed**

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

Pass Yes / No Fail Yes / No

Comments: The images are display in a correct way.

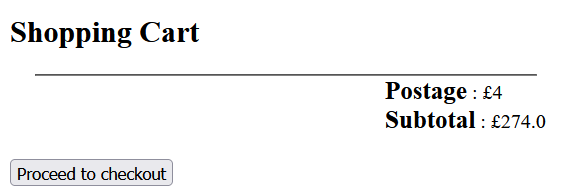


**5.3.3 Currencies are correctly displayed**

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

Pass Yes / No Fail Yes / No

Comments: The symbol for currency is £.

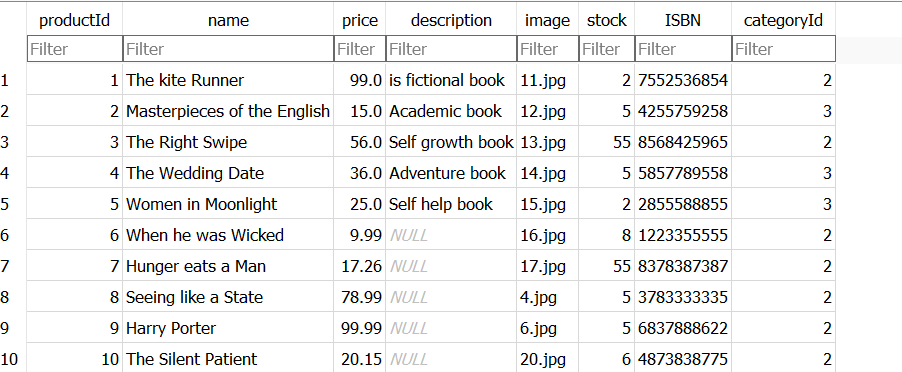


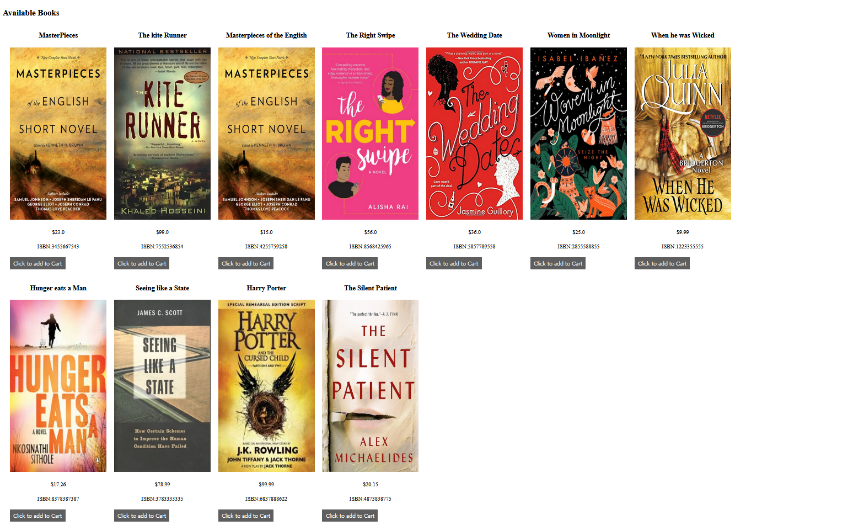
**5.3.4 Prices are correctly displayed**

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

Pass Yes / No Fail Yes / No

Comments: The data type of price is decimal, but there is just one decimal point.





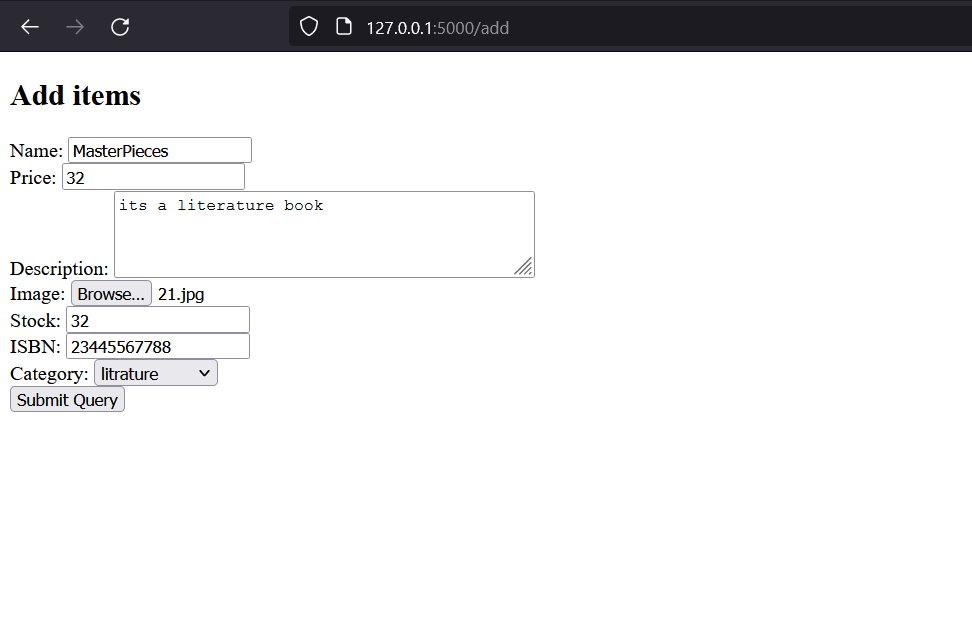
**5.4 hopping basket**

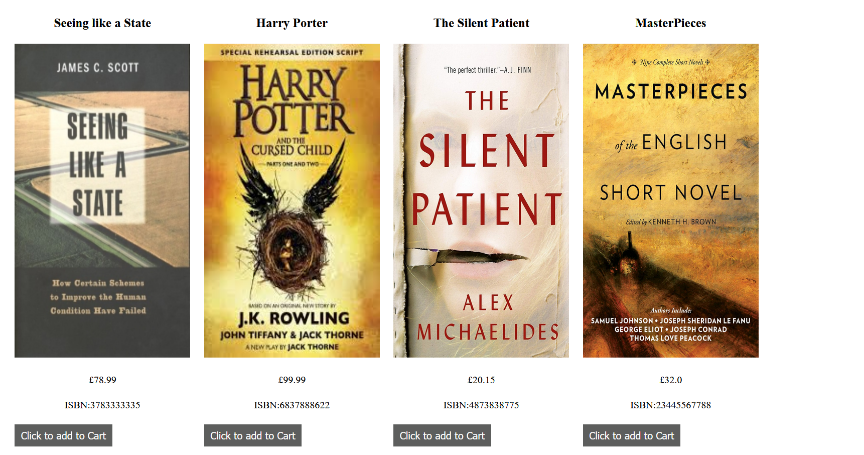
**5.4.1 Items can be added and removed**

Test 1: Add item

Pass Yes / No Fail Yes / No

Comments: Items can be added successfully, but the enter method of price and stock are not slider.





Test 2: Remove item

Pass Yes / No Fail Yes / No

Comments: Items can remove successfully. When the stock of an item is 0, the item will be deleted automatically.

**5.5 Checkout**

**5.5.1 Payment is accepted**

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

Pass Yes / No Fail Yes / No

**.2 TESTS RUN ON EXTERNAL CODE**

**6.QUALITY ASSURANCE**

.1 Quality assurance statement

1. The required functions are implemented, but the slider of entering price and stock is not. And the registration module allows the same username to register, which means there might be a lot of customers who have the same username and password. Also, the administrator and customer should log in to the different index pages, however this program just made one index page, which means the customer can also see the add product function and delete product function, but the two functions just allow the administrator to execute.

2. Some formats of the code are from the lab sheet on Aula, such as showing the tables in database on the web. And the CSS of the navigation bar on each page is referenced from bootstrap. These references are shown in section 8 of this report.

.2 External QA evaluation

**7.DOCUMENTATION**

**.1 DOCUMENTATION LIST**

1. Code purpose (Section 1 of this report)
2. Code location (Section 2 of this report)
3. How to get this code and run on the Codio. (Section 3 of this report)
4. The explanation of this code and some comments. (Section 4 of this report)
5. Text regime of this code. (Section 5 of this report)
6. Quality assurance statement. (Section 6 of this report)
7. Reference (Section 8 of this report)

EXTERNAL DOCUMENTATION INSPECTION

Tested Students ID:**11384841, 11384885,**

Codes:

After running his application, I saw that his page design is really attractive, and the website's layout and functionality are very consistent with client preferences. He designed a minimum and maximum price limit for the customer to choose from on the client's home page, for example. In addition, the design of his administrator interface is stunning.

The report parts:

The layout of his reports is very good, all the typography is good, and as stated in his personal QA, all the document design is very good.

QA:

His QA is very well written and does not use very verbal statements, but breaks them down into clear sections. This showed his good understanding and interpretation of his code, and clearly expressed the characteristics and mistakes of his work, so that I could clearly view and preview the code according

to his QA.

Tested Students ID:**11384885**

Codes:

After running his application, I saw that his page design is really intuitive, and the website's layout and functionality follows the principle of web design. The color choices are good and follows a certain theme. All the important functions are grouped together well.

The report parts:

The layout of his reports is good, the formatting and typography is correct.

QA:

His QA is clear and precise.

Tested Students ID:**11384841**

Codes:

After running and observing the web application I was impressed. The major functions executes as they were designed to perfectly and the general look of the website is attractive.

The report parts:

His reports format and content are well stated and correct.

QA:

His QA is sectioned well and easy to comprehend.

Tested Students ID:**10140288**

Codes:

After running his application, I observed everything was running okay except for the payment feature will was not functioning correctly. All the other pages and buttons were functioning correctly as designed to.

The report parts:

The layout and formatting of the report is up to standard, and the content to is clear and correct.

QA:

His QA is well stated.

**8.REFERENCES**

the html format of navigation bar

https://getbootstrap.com/docs/3.4/components/#nav

displays data from the database on the web

Alas, R., Kross, P., Liivat, A., & Saar, J. (2018). Success factors for teams in business game Dynama. *Problems and Perspectives in Management*, *16*(1), 309–319. <https://doi.org/10.21511/ppm.16(1).2018.30>

Sakaliuk, O. Yu. (2017). AUTOMATION OF CONTROL OF THE BUSINESS PROCESS OF PUBLISHING SCIENTIFIC JOURNALS. *Automation of Technological and Business Processes*, *8*(3). <https://doi.org/10.15673/atbp.v8i3.568>

Sandage, E. (2019). Make an Online Bookstore Available to Members. *The Membership Management Report*, *15*(6), 1–1. <https://doi.org/10.1002/mmr.31212>

SHavtikova, L. M. (2019). Using Python to schedule graphics. *SCIENTIFIC DEVELOPMENT TRENDS and EDUCATION*. https://doi.org/10.18411/lj-12-2019-30

‌