

School of Computing, Engineering and Mathematics (CEM)

Faculty of Engineering, Environment and Computing (EEC)

**5001CEM SOFTWARE ENGINEERING** | 2122

**PROJECT REPORT**

**NAME:** Pedro Azevedo

**SID: 10069934**

1. **CODE PURPOSE this has been changed from GitHub Version**

This code created a website that manages an online bookstore, it has a login page, add stock page, stock level, shopping cart, checkout and pay screen.

1. **CODE LOCATION this has been changed from GitHub Version**

<https://github.coventry.ac.uk/5001CEM-2122/bookStore---PA>

1. **CODE INSTALLATION**A screenshot of a computer

   Description automatically generated with medium confidence **this has been changed from GitHub Version**
2. Go to <https://github.coventry.ac.uk/5001CEM-2122/bookStore---PA>, download code.
3. Unzip Folder
4. Open code editor of choice, we will be using Visual Studio Code (steps may vary)

A screenshot of a computer

Description automatically generated with medium confidence

1. Open Folder
2. Text

   Description automatically generatedRun the following lines into the terminal:
   1. python -m venv venv
   2. venv\Scripts\activate
   3. pip install Flask
   4. Text

      Description automatically generatedpip install Flask-SQLAlchemy
   5. pip install flask-cors

Text

Description automatically generated

Text

Description automatically generated

1. Text

   Description automatically generatedRun code:
   1. set FLASK\_APP=app
   2. flask run
2. Click url link and you are on the website
3. **CODE EXPLANATION**
   1. **Login Requirement**
4. @app.before\_request #This will load first, as we require user to be logged in
5. def user\_login():
6. userID = session.get('userName') #this will get the user from the session userName,
7. if userID is None:               #if it is empty then we know no one is logged in
8. g.user = None                #g.user is a global variable
9. else:
10. g.user = 'set'
11. def login\_required(view):
12. @functools.wraps(view)
13. def wrapped\_view(\*\*kwargs): #if it is determined their no user, it will load the login url
14. if g.user is None: #this function will loop itself until their is a login
15. return redirect(url\_for('login'))
16. return view(\*\*kwargs)
17. return wrapped\_view

These two functions will require the user to login before anything else is done. User\_login will get the session value for userID if None is found then g.user is set to none, g.user is a global variable. Login\_required forces the user to redirect to the login page if g.user is equal to none. Otherwise, if g.user is not none, the user will be directed to the homepage. This function uses the functools import, allowing us to use the wrapped\_view function within the login\_required function.

* 1. **Login Process**

1. @app.route('/login', methods=['GET', 'POST']) #this is the login functions
2. def login():
3. if request.method == 'POST': #once it recieves information from the login submition it passes it to do\_login
4. return do\_login(request.form['userName'],request.form['password'])
5. else:
6. return show\_login\_page() #if not it loads the login page
7. def show\_login\_page():
8. return render\_template('login.html',page=url\_for('login')) #loads login page
9. def do\_login(userN,passW):
10. formatUserInfo = userN + "," + passW #format the given username and password from submit form
11. userName = "user"
12. admin = "admin"
13. session['userName'] = userName #creates the regular and admin user session
14. session['admin'] = admin
15. if 'userName' in session: #pops both session so we make sure it doesnt think the user is logged in yet
16. session.pop('userName',None)
17. if 'admin' in session:
18. session.pop('admin',None)
19. con = sqlite3.connect('mydatabase.db') #we connect to the database, and pull the username and password from user table
20. cur=con.cursor()
21. cur.execute("SELECT userName,password FROM user")
22. userInfo = cur.fetchall()
23. for user in userInfo:#loops through all the users pulled from table
24. user=str(user).replace("(","") #fetchall brings it as a tuple so we want to make sure its in the correct format when comparing the values
25. user=str(user).replace(")","")
26. user=str(user).replace("'","")
27. user=str(user).replace(" ","")
28. if user == formatUserInfo: #we then compare our values from the table, to values from submission form
29. if userN == 'admin': #if the username was admin we know this is an admin user and we activate both sessions
30. session['userName'] = userName
31. session['admin'] = admin
32. return redirect("http://127.0.0.1:5000/") #we then send it to the homepage
33. else:
34. session['userName'] = userName #is not admin so only user session
35. return redirect("http://127.0.0.1:5000/")
36. else:
37. return redirect(url\_for('login')) #loops if password or username was wrong
38. @app.route('/logout') #logs the user out
39. def logout():
40. session.clear() #this will clear all the current sessions
41. return redirect(url\_for('login'))

This section of code handles the login steps. It uses the GET POST methods as we must pull information from the html submission form.

In the login function, we check if a post request has come through, if it has we then send the information from the form to the do\_login function (e.g. request.form[‘userName’]). If no request has been made, then we load the show\_login\_page function. All this function does is simply render the login.html template and redirects the user to the login URL.

The do\_login function is where most of this happens. First it receives two variables from the form, a username and password. We combine these two variables so we can compare them later. We define a session for a regular user and a admin user, we then make sure these sessions are empty for now, by popping them. We then connect to the ‘mydatabase.db’ file, and we execute a query to pull the username and password from the user’s table. We assign them all to the userInfo variable We then loop through all the users in userInfo, as well as formatting the tuple user so we can compare the user variable to the combined user and password variable from earlier. If they match, we know this is a confirmed user. If the username is admin, we know it’s a admin user so we activate the admin session, as well as the user session. Otherwise, it’s a regular user. The user is now logged in, so they are redirected to the homepage.

The logout function simply clears all the session if the user hits logout and redirects them to the login page.

* 1. **Login HTML**

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <link href="{{ url\_for('static', filename='css/style.css') }}" rel="stylesheet" type="text/css" />
5. <meta http-equiv="content-type" content="text/html; charset=UTF-8">
6. <title></title>
7. </head>
8. <body style="background-color:powderblue;">
9. <h1 align="center"><br>
10. </h1>
11. <br>
12. <br>
13. <h1 style="text-align:center;">Login</h1>
14. <p style="text-align:center;">Insert Information<br>
15. </p>
16. <form action="{{page}}" method="post" enctype="multipart/form-data">
17. <label for="userName" style="text-align:center;">Username:</label><br>
18. <input id="userName" name="userName" value="username" type="text"><br>
19. <label for="password" style="text-align:center;">Password:</label><br>
20. <input id="password" name="password" value="password" type="text"><br>
21. <br> <input value="Submit" type="submit"> </form>
22. </div>
23. </body>
24. </html>

The login page loads in the shared css file, so it matches the rest of the pages. We create a submission for with the post method and a submit button. We give it the inputs username and password, both as text values. This will create two input boxes for the user to login with. This is the first page users should see before accessing main site.

* 1. **Homepage**

1. @app.route('/') #this is the homepage
2. @login\_required #user must be logged in to accesss
3. def book():
4. try:
5. con = sqlite3.connect('mydatabase.db')#connects to database and pulls all the information from books table
6. cur = con.cursor();
7. cur.execute("SELECT \* FROM books")
8. rows = cur.fetchall()
9. return render\_template('homepage.html', book=rows)#sends this info to table in homepage to load books
10. except Exception as e:#prints any expections
11. print(e)
12. finally:
13. cur.close() #closes the connection to the database
14. con.close()

This is the homepage function. It has @login\_required so when the user loads into it they must be logged in for it to work. It then connects to the database as usual, and this query pulls all the information from books and defines each book as ‘rows’. We then render the homepage html and pass through these rows as ‘book’ to the html page. We have an except statement to catch any exceptions. Finally, we close the table to not cause any conflict.

* 1. **Shopping Cart**

1. @app.route('/add', methods=['POST'])
2. @login\_required
3. def add\_product\_to\_cart():#this will add the books to the shopping cart
4. cursor = None
5. try:
6. \_quant = int(request.form['quant']) #gets the quantity and the the ISBN from the form
7. \_ISBN = request.form['ISBN']
9. if \_quant and \_ISBN and request.method == 'POST':
10. con = sqlite3.connect('mydatabase.db') #once these values are gotten, and the user has picked a book, we get all the info on that book
11. cur = con.cursor()
12. cur.execute("SELECT \* FROM books WHERE ISBN=?;", [\_ISBN]) #pulls book that match isbn
13. row = cur.fetchone()#only pulls the one row
14. itemArray = { row[4] : {'bookName' : row[0], 'ISBN' : row[3], 'quant' : \_quant, 'retPrice' : row[6], 'image' : row[8], 'total\_price': \_quant \* row[6]}}
15. print('itemArray is', itemArray) #this creates an item array based on the given info
17. all\_total\_price = 0 #defines the variables, total price and quantity for the cart session
18. all\_total\_quantity = 0
20. session.modified = True
22. if 'cart\_item' in session: #if the cart item session exists then this runs
23. print('in session')
24. if row[3] in session['cart\_item']: #if it has an isbn in the cart item session
25. for key, value in session['cart\_item'].items(): #loops through the cart items
26. if row[3] == key: #if the isbn matches the key
27. old\_quantity = session['cart\_item'][key]['quant'] #pulls the quantity from the cart session
28. total\_quantity = old\_quantity + \_quant #adds the quantity to the total quantity
29. session['cart\_item'][key]['quant'] = total\_quantity #updates total quantity in session
30. session['cart\_item'][key]['total\_price'] = total\_quantity \* row[6] #updates the price in session using the retail price
31. else:
32. session['cart\_item'] = array\_merge(session['cart\_item'], itemArray) #sends to the array merge
34. for key, value in session['cart\_item'].items():
35. individual\_quantity = int(session['cart\_item'][key]['quant']) #defines the quantity from the value inside session
36. individual\_price = float(session['cart\_item'][key]['total\_price'])#defiens the total price of a book from session
37. all\_total\_quantity = all\_total\_quantity + individual\_quantity #calculates total
38. all\_total\_price = all\_total\_price + individual\_price
39. else:
40. session['cart\_item'] = itemArray #if the session doesnt already exist this creates
41. all\_total\_quantity = all\_total\_quantity + \_quant
42. all\_total\_price = all\_total\_price + \_quant \* row[6]
44. session['all\_total\_quantity'] = all\_total\_quantity #creates the total quantity and price sessions
45. session['all\_total\_price'] = all\_total\_price
47. return redirect(url\_for('.book'))
48. else:
49. return 'Error while adding item to cart'
50. except Exception as e:
51. print(e)
52. finally:
53. cur.close()
54. con.close()
55. @app.route('/empty')#when the user hits the empty button all the cart sessions are popped and made empty
56. @login\_required
57. def empty\_cart():
58. session.pop('cart\_item')
59. session.pop('all\_total\_price',None)
60. session.pop('all\_total\_quantity',None)
61. return redirect(url\_for('.book'))
62. @app.route('/delete/<string:ISBN>') #deletes the item from the cart
63. @login\_required
64. def delete\_product(ISBN):
65. all\_total\_price = 0
66. all\_total\_quantity = 0
67. session.modified = True
69. for item in session['cart\_item'].items():
70. if item[0] == ISBN:
71. session['cart\_item'].pop(item[0], None)
72. if 'cart\_item' in session:
73. for key, value in session['cart\_item'].items():
74. individual\_quantity = int(session['cart\_item'][key]['quant'])
75. individual\_price = float(session['cart\_item'][key]['total\_price'])
76. all\_total\_quantity = all\_total\_quantity + individual\_quantity
77. all\_total\_price = all\_total\_price + individual\_price
78. break
80. if all\_total\_quantity == 0:
81. session.pop('cart\_item',None)
82. session.pop('all\_total\_price',None)
83. session.pop('all\_total\_quantity',None)
84. else:
85. session['all\_total\_quantity'] = all\_total\_quantity
86. session['all\_total\_price'] = all\_total\_price
87. return redirect(url\_for('.book'))
88. TAKEN FROM WEEK 5 5001CEM

def array\_merge( first\_array , second\_array ): #adds the lists together

1. if isinstance( first\_array , list ) and isinstance( second\_array , list ):
2. return first\_array + second\_array
3. elif isinstance( first\_array , dict ) and isinstance( second\_array , dict ):#if its a dictionary, turns into a list then merges
4. return dict( list( first\_array.items() ) + list( second\_array.items() ) )
5. elif isinstance( first\_array , set ) and isinstance( second\_array , set ):
6. return first\_array.union( second\_array )
7. return False

This section of code will manage the shopping cart.

First, we have the add\_product\_to\_cart function. This will handle adding the items to the cart session. First we pull the values inside the request.form for the books ISBN value and the quantity the user chose. The ISBN will help us define the book and the quantity will determine how many they wish to buy. If both requests come through, then we connect the database and pull the book from the table that matches the ISBN number. We then create an itemArray with the values: bookName, ISBN, quant, retPrice,image and total\_price. This will be used in the cart\_item session. If the cart\_item is not currently in session we then define it and make it equal to the itemArray. We also update the total quantity and price based on the books price and selected quantity.

If the cart is already in session and they try to add more, we check to see if the ISBN is already in the cart, if it is we then update the quantity of the book in the cart and update all the values, if not we simply add the item to the cart. We then recalculate the prices based on the total quantity of books and the books individual prices. We then create a session for the total price and quantity of all books.

The empty function simply empties all the cart values, but doesn’t touch the userID session so the user stays logged in.

The delete\_product function will attempt to delete a single record of a book inside the cart item. It will pull the ISBN number clicked in the same row as the remove button, it will then compare that to the ones in the cart\_item and delete that record.

* 1. **Homepage HTML**

1. <!DOCTYPE html>
2. <html>
3. <style>
4. /\* this is the css for all the header navigation and body \*/
5. ul {
6. list-style-type: none;
7. margin: 0;
8. padding: 0;
9. overflow: hidden;
10. background-color: #5A3727;
11. position: fixed;
12. }
13. li {
14. float: left;
15. }
16. li a {
17. display: block;
18. color: white;
19. text-align: center;
20. padding: 5px 5px;
21. font-size: 20px;
22. }
23. li a:hover {
24. background-color: #3d2b21;
25. }
26. </style>
27. <ul>
28. <!-- this is the navigation form, redirects to certain pages, requires user to have admin session to access add stock -->
29. <li><a class='active' href=http://127.0.0.1:5000> Home</a></li>
30. <li><a href=http://127.0.0.1:5000/logout>Log Out</a></li>
31. {% if 'admin' in session %}
32. <li><a href=http://127.0.0.1:5000/addBook>Add Stock</a></li>
33. {% endif %}
34. <li><a href=http://127.0.0.1:5000/showStock>Stock</a></li>
35. </ul>
36. <head>
37. <title>Shopping Cart</title>
38. <link href="{{ url\_for('static', filename='css/style.css') }}" rel="stylesheet" type="text/css" />
39. </head>
40. <body>
41. <div>
42. {% with messages = get\_flashed\_messages() %}
43. {% if messages %}
44. <ul class=flashes>
45. {% for message in messages %}
46. <li>{{ message }}</li>
47. {% endfor %}
48. </ul>
49. {% endif %}
50. {% endwith %}
51. </div>
52. <!-- We load the shopping cart, filling it with the values inside the session cart\_item -->
53. <div id="shopping-cart">
54. <div class="txt-heading">Shopping Cart</div>
55. {% if 'cart\_item' in session %}
56. <!-- empty cart button links to empty cart function -->
57. <a id="btnEmpty" href="{{ url\_for('.empty\_cart') }}">Empty Cart</a>
58. <table class="tbl-cart" cellpadding="10" cellspacing="1">
59. <tbody>
60. <tr>
61. <th style="text-align:left;">Name</th>
62. <th style="text-align:left;">Code</th>
63. <th style="text-align:right;" width="5%">Quantity</th>
64. <th style="text-align:right;" width="10%">Unit Price</th>
65. <th style="text-align:right;" width="10%">Price</th>
66. <th style="text-align:center;" width="5%">Remove</th>
67. </tr>
68. {% for key, val in session['cart\_item'].items() %}
69. {% set quant = session['cart\_item'][key]['quant'] %}
70. {% set retPrice = session['cart\_item'][key]['retPrice'] %}
71. {% set item\_price = session['cart\_item'][key]['total\_price'] %}
72. <!--load image based on image name in static folder  -->
73. <tr>
74. <td><img src="/static/images/book-images/{{ session['cart\_item'][key]['image'] }}" class="cart-item-image" width = "10" height = "20"/>{{ session['cart\_item'][key]['bookName'] }}</td>
75. <td>{{ session['cart\_item'][key]['ISBN'] }}</td>
76. <td style="text-align:right;">{{ quant }}</td>
77. <td  style="text-align:right;">&#8356; {{ retPrice }}</td>
78. <td  style="text-align:right;">&#8356; {{ item\_price }}</td>
79. <td style="text-align:center;">
80. <a href="{{ url\_for('.delete\_product', ISBN=session['cart\_item'][key]['ISBN']) }}" class="btnRemoveAction">
81. <img src="/static/images/icon-delete.png" alt="Remove Item"  width = "30" height = "40" />
82. </a>
83. </td>
85. </tr>
86. {% endfor %}
87. <tr>
88. <td colspan="2" align="right">Total:</td>
89. <td align="right">{{ session['all\_total\_quantity'] }}</td>
90. <td align="right" colspan="2"><strong>&#8356; {{ session['all\_total\_price'] }}</strong></td>
91. <td><a href ="http://127.0.0.1:5000/checkout"><button>
92. Checkout
93. </button>
94. </a></td>
95. </tr>
96. </tbody>
97. </table>
98. {% else: %}
99. <div class="no-records">Your Cart is Empty</div>
100. {% endif %}
101. </div>
102. <div id="product-grid">
103. <div class="txt-heading">Products</div>
104. {% for product in book %}
105. <!-- loads the books under the cart session -->
106. <div class="product-item">
107. <form method="post" action="/add">
108. <div class="product-image"><img src="/static/images/book-images/{{ product[8] }}" width = "100" height = "200"></div>
109. <div class="product-tile-footer">
110. <div class="product-title">{{ product[0] }}</div>
111. <div class="product-price">&#8356; {{ product[6] }}</div>
112. <div class="cart-action">
113. <input type="hidden" name="ISBN" value="{{ product[3] }}"/>
114. <input type="text" class="product-quantity" name="quant" value="1" size="2" />
115. <input type="submit" value="Add to Cart" class="btnAddAction" />
116. </div>
117. </div>
118. </form>
119. </div>
120. {% endfor %}
122. </div>
123. </body>
124. </html>

The style portion of the code loads the CSS for the navigation bar that sits on the top of the page. The navigation bar leads to 4 pages. The Home page, the Logout, the Add Stock page, and the Stock page. Only users who have the admin session active will be able to see the Add Stock page, otherwise you only see Home, Add Stock and Pay Now. The navigation CSS is used across all HTML files

The Shopping Cart Div pulls the information from the cart\_item in order to populate the fields. It uses the image name to find the local image file and load the book cover image. The shopping cart will display all the books in the cart in rows, showing the title of the book, how many you have selected, the price of that individual book and the total price of that quantity multiplied by price. It will then also display the total price combined off all books as well as a checkout button. This button will redirect the user to the checkout screen. The image for the remove button is stored locally like the book covers. The product div is like the previous forms, but simply has the ISBN and quantity as an input, except the ISBN Is hidden. It will display the book cover, the book title, and its price as well.

* 1. **Checkout HTML**

1. <!-- this is the navigation form, redirects to certain pages, requires user to have admin session to access add stock -->
2. <li><a class='active' href=http://127.0.0.1:5000> Home</a></li>
3. <li><a href=http://127.0.0.1:5000/logout>Log Out</a></li>
4. {% if 'admin' in session %}
5. <li><a href=http://127.0.0.1:5000/addBook> Add Stock</a></li>
6. {% endif %}
7. <li><a href=http://127.0.0.1:5000/showStock>Stock</a></li>
8. </ul>
9. <head>
10. <title>Shopping Cart</title>
11. <link href="{{ url\_for('static', filename='css/style.css') }}" rel="stylesheet" type="text/css" />
12. </head>
13. <body>
14. <div>
15. <!-- this will only display if quantity had to be changed due to a lack. -->
16. {% if 'quantityError' in session %}
17. <h1 style="text-align:center;">Out of Stock for certain books, quantity has been updated</h1>
18. {% endif %}
19. </div>
20. <!-- same base as the homepage.html -->
21. <div id="shopping-cart">
22. <div class="txt-heading">Checkout</div>
23. {% if 'cart\_item' in session %}
24. <a id="btnEmpty" href="{{ url\_for('.empty\_cart') }}">Empty Cart</a>
25. <table class="tbl-cart" cellpadding="10" cellspacing="1">
26. <tbody>
27. <tr>
28. <th style="text-align:left;">Name</th>
29. <th style="text-align:left;">Code</th>
30. <th style="text-align:right;" width="5%">Quantity</th>
31. <th style="text-align:right;" width="10%">Unit Price</th>
32. <th style="text-align:right;" width="10%">Price</th>
33. </tr>
34. {% for key, val in session['cart\_item'].items() %}
35. {% set quant = session['cart\_item'][key]['quant'] %}
36. {% set retPrice = session['cart\_item'][key]['retPrice'] %}
37. {% set item\_price = session['cart\_item'][key]['total\_price'] %}
38. <tr>
39. <td><img src="/static/images/book-images/{{ session['cart\_item'][key]['image'] }}" class="cart-item-image" width = "10" height = "20" />{{ session['cart\_item'][key]['bookName'] }}</td>
40. <td>{{ session['cart\_item'][key]['ISBN'] }}</td>
41. <td style="text-align:right;">{{ quant }}</td>
42. <td  style="text-align:right;">&#8356; {{ retPrice }}</td>
43. <td  style="text-align:right;">&#8356; {{ item\_price }}</td>
44. <td style="text-align:center;"></td>
46. </tr>
47. {% endfor %}
48. <tr>
49. <td colspan="2" align="right">Total:</td>
50. <td align="right">{{ session['all\_total\_quantity'] }}</td>
51. <td align="right" colspan="2"><strong>&#8356; {{ session['all\_total\_price'] }}</strong></td>
52. <td><a href ="http://127.0.0.1:5000/paynow"><button>
53. Pay Now
54. </button>
55. </a></td>
56. </tr>
57. </tbody>
58. </table>
59. {% else: %}
60. <div class="no-records">Your Cart is Empty</div>
61. {% endif %}
62. </div>
63. </body>
64. </html>

Like the previous HTML it pulls all the values from the session items and inserts them into the fields where they are appropriate. One they user has confirmed their prices are correct as well as their order they then click the paynow button which redirects them to the paynow page using a reference. This will display the same information seen in the shopping cart: Book’s Name, Quantity purchased, individual and total price of that book, total overall price and quantity.

* 1. **Checkout**

1. @app.route('/checkout') #this will checkout the cart
2. @login\_required
3. def checkout():
4. quantityError = "No books sir" #we create the quantity error session to use later
5. session['quantityError'] = quantityError
6. for key, value in session['cart\_item'].items(): #we loop through all the items in the session
7. quantity = int(session['cart\_item'][key]['quant'])#we pull the quantity and the book name
8. bookName = str(session['cart\_item'][key]['bookName'])
9. if 'quantityError' in session:
10. session.pop('quantityError', None) #we empty the quantError session
11. con = sqlite3.connect('mydatabase.db')
12. cur = con.cursor();
13. cur.execute("SELECT quant FROM books WHERE bookName=?;", [bookName]) # we pull the quant where it matches the book name
14. curQuant= cur.fetchone()
15. curQuant=str(curQuant).replace("(","")
16. curQuant=str(curQuant).replace(")","")
17. curQuant=str(curQuant).replace("'","")
18. curQuant=str(curQuant).replace(",","")
19. if int(curQuant) >= quantity: #if the quantity in the cart is less than the table this runs
20. newQuant = int(curQuant) - quantity #calculate new quant
21. con = sqlite3.connect('mydatabase.db')
22. cur = con.cursor();
23. update = "UPDATE books SET quant=" + str(newQuant) +" WHERE bookName ='" + bookName + "'" #update quant
24. cur.execute(update)
25. con.commit()
26. elif int(curQuant) < quantity: #if the cart quantity is more than the table
27. session['quantityError'] = quantityError # create session again
28. session['cart\_item'][key]['quant'] = int(curQuant)
29. newQuant = 0
30. con = sqlite3.connect('mydatabase.db')
31. cur = con.cursor();
32. update = "UPDATE books SET quant=" + str(newQuant) +" WHERE bookName ='" + bookName + "'" #quantity in table becomes 0
33. cur.execute(update)
34. con.commit()
35. quantValue = session['cart\_item'][key]['total\_price'] #we update the prices based on the actual quantity we have
36. session['cart\_item'][key]['total\_price'] = session['cart\_item'][key]['retPrice']\*int(curQuant)
37. session['all\_total\_price']  = session['all\_total\_price'] - quantValue + (session['cart\_item'][key]['retPrice']\*int(curQuant))
38. return render\_template('checkout.html')

This is the checkout function. The main purpose of this section is to check if the stock levels match the quantity selected. In order to do this, we create a session called quantityError. We loop through all the items in the cart\_item and pull the quantity and bookname, we pull the book from the table that matches this value, we only take the quantity as its all we require. If the quantity requested is greater than the stock available, we must update the total stock of that book to 0, as well as making the available stock in the checkout to match the old stock level of the book. We then recalculate the total prices based on the new quantity level and update the table as well. If the number of stocks requested is less than the available stock, then you simply must update the table.

* 1. **Pay now**

1. @app.route('/paynow', methods=['GET', 'POST'])
2. @login\_required
3. def paynow():
4. if request.method == 'POST': #this will load the pay now screen
5. return clear\_paynow(request.form['accountNum'],request.form['pinNum'])
6. else:
7. return show\_paynow()
8. def show\_paynow():
9. return render\_template('paynow.html',page=url\_for('paynow'))
10. def clear\_paynow(a,p): #empties all the sessions after paying
11. session.pop('cart\_item',None)
12. session.pop('all\_total\_price',None)
13. session.pop('all\_total\_quantity',None)
14. return redirect(url\_for('.book'))

Once the user has clicked checkout, they will be taken to a page where they input a 16-bit account number and a 4-bit pin number. Once paynow function receives that request it sends it to clear\_paynow function which wipes the cart and sends the user back to the homepage.

* 1. **Pay now HTML**

1. <!DOCTYPE html>
2. <html>
3. <style>
4. <!-- this is the navigation form, redirects to certain pages, requires user to have admin session to access add stock -->
5. <li><a class='active' href=http://127.0.0.1:5000> Home</a></li>
6. <li><a href=http://127.0.0.1:5000/logout>Log Out</a></li>
7. {% if 'admin' in session %}
8. <li><a href=http://127.0.0.1:5000/addBook>Add Stock</a></li>
9. {% endif %}
10. <li><a href=http://127.0.0.1:5000/showStock>Stock</a></li>
11. </ul>
12. <head>
13. <link href="{{ url\_for('static', filename='css/style.css') }}" rel="stylesheet" type="text/css" />
14. <meta http-equiv="content-type" content="text/html; charset=UTF-8">
15. <title></title>
16. </head>
17. <body style="background-color:powderblue;">
18. <h1 align="center"><br>
19. </h1>
20. <br>
21. <br>
22. <h1 style="text-align:center;">Pay Now</h1>
23. <p style="text-align:center;">Insert Information<br>
24. </p>
25. <!-- We create a form for the user to isnert the account and pin numbers -->
26. <form action="{{page}}" method="post" enctype="multipart/form-data">
27. <label for="accountNum" style="text-align:center;">16-Digit Account Number:</label><br>
28. <input id="accountNum" name="accountNum" value="" type="text" pattern='.{16}+' maxlength="16"><br>
29. <!-- they are both numbers, given a pattern limitation to require 16 digits.s -->
30. <label for="pinNum" style="text-align:center;">4-Digit Pin Number:</label><br>
31. <input id="pinNum" name="pinNum" value="" type="text" pattern='.{4}+' maxlength="4"><br>
32. <br> <input value="Submit" type="submit"> </form>
33. </div>
34. </body>
35. </html>

Uses a simple input form as before to accept user values for the 16-digit account number and the 4-digit pin number. We can use the maxlength to force the user to use a 16-digit number as well as a 4-digit pin.

* 1. **Add Stock**

1. @app.route("/addBook", methods=['GET', 'POST'])#This will add books and update the stock
2. @login\_required #requires user to be logged in
3. def addBook():
4. if request.method == 'POST':
5. ISBN = request.form['ISBN'] #gets the isbn given by user in form
6. quant = request.form['quant'] #gets the quantity given by user in form
7. con = sqlite3.connect('mydatabase.db')
8. cur = con.cursor();
9. cur.execute("SELECT ISBN FROM books") #this will be used to check if book already exists
10. isbnNum = cur.fetchall()
11. for book in isbnNum: #loops through all isbn numbers in book table
12. book=str(book).replace("(","")#formats it into useable format
13. book=str(book).replace(")","")
14. book=str(book).replace(",","")
15. book=str(book).replace("'","")
16. if book == ISBN: #comapres the isbn number to the one given in the form
17. cur.execute("SELECT quant FROM books WHERE ISBN = '" + ISBN + "'") #if it matches we then pull the quantity that matches that isbn
18. curQuant = cur.fetchall()
19. curQuant=str(curQuant[0]).replace("(","") #format it into a useable state
20. curQuant=curQuant.replace("(","")
21. curQuant=curQuant.replace(")","")
22. curQuant=curQuant.replace("'","")
23. curQuant=curQuant.replace(",","")
24. newQuant = int(curQuant) + int(quant) #we add old quantity to new quantity
25. cur.execute("UPDATE books SET quant = " + str(newQuant) + " WHERE ISBN = '" + ISBN + "'") #updates quantity in table
26. con.commit() #commits changes
27. return show\_add\_book()#goes back to stock levels
29. if 'image' not in request.files: #requires an image for the book table
30. return 'there is no image in form!'
31. image = request.files['image']#pulls image from form
32. path = os.path.join(app.config['UPLOAD\_FOLDER'], image.filename)#connects image to the UPLOAD\_FOLDER
33. imageName = image.filename #assign a variable with the same name as the image file
34. image.save(path)#saves the image to the given path
35. return add\_book(request.form['bookName'],request.form['author'],request.form['pubDate'],request.form['ISBN'], request.form['desc'],request.form['traVal'],request.form['retPrice'],request.form['quant'],imageName)
36. else:
37. return show\_add\_book()
38. def show\_add\_book():
39. return render\_template('addBook.html',page=url\_for('addBook'))#loads the add book page
40. def add\_book(bookName, author, pubDate, ISBN, desc, traVal, retPrice, quant,image):
41. con = sqlite3.connect('mydatabase.db')
42. cur = con.cursor(); #if the book does not exist this will insert the values from the form into the book tables
43. sql = "INSERT INTO books(bookName, author, pubDate, ISBN, desc, traVal, retPrice, quant,image) VALUES (?,?,?,?,?,?,?,?,?)"
44. #con.execute("INSERT INTO books(bookName, author, pubDate, ISBN, desc, traVal, retPrice, quant) VALUES (?,?,?,?,?,?,?,?)",[bookName, author, pubDate, ISBN, desc, traVal, retPrice, quant])
45. val = (bookName, author, pubDate, ISBN, desc, traVal, retPrice, quant,image)
46. cur.execute(sql,val) #executes the query with given variables
47. con.commit()#saves changes
48. con.close()
49. return show\_add\_book()

This function deals with adding books to the book table. It requests the information from the form in the html page. Once it gets these values it checks ISBN number. It pulls the ISBN values from every book in the table, it then goes book by book and compares the isbn value. If they match, instead of adding a new book it will update the quantity instead.

When adding a new book, the user must submit the image file as well. This is required, so we use an if statement to check. Using the predefined ‘UPLOAD\_FOLDER’, which is in the static folder, we then save the image to this path. In the table we save the images file name, so we can call on it when we need it.

Once it has checked for all this it passes the information to the add\_book function, which simply takes these variables and using the “INSERT INTO” sql query we add it to the table in the database.

1. **TESTING**

**5.1 Developer name / SID**

**Rui Pedro Azevedo**

**5.2 Project**

**Demo:**

Uploaded with report

**5.3 Test scope**

**5.3.1**

* Login Page (Regular user and admin)
* Stock Levels (new and update)
* Stock Page
* Add to Cart
* Cart Display
* Checkout
* Paying
* Logout

**5.3.2 Out of scope**

* Security
* Outlier Values

**5.4 Test regime this has been changed from GitHub Version**

4.1 Login(10 mins)

.1 User can login

Test 1: User is able to login as customer (Using “Username = customer1” AND Password = p455w0rd”)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

.2

Test 2: User is able to login as customer (Using “Username = customer2” AND “Password = p455w0rd”)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

.3 Admin can login

Test 3: Admin can login as admin ( Using “Username = admin” AND “Password =p455w0rd”)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

.4 Incorrect login

Test 4: Incorrect login information wont login

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

.5 Correct display

Test 5: All information displayed is correct

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

4.3 Homepage (20 mins)

.1 Database matches display

Test 1: inspect number of database records and cross reference to display. (Assuming user is using Visual Studio Code, simply click “Open Database” on the file end select the “books” table)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

.2 Images are correct

Test 1: images are correctly resized and fit bounding box

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

.3 Book price is correct

Test 1: page display correct price and format(£)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

.4 Book Names

Test 1: Book names are correctly displayed

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

.5 Page format (5 Minutes)

Test 1: All display is correct format and inside boudning box

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

.6 Navigation (10 Minutes)

Test 1: Home takes you to home page

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

Test 2: Stock takes you to stock page

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

Test 3: Logout logs user out

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

Test 4: Admin can access “Add Stock” page

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

Test 5: User cannot see “Add Stock”

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

.7 Add to Basket (5 Minutes)

Test 1: Add specific book to basket

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

Test 2: Quantity is correct when added to basket

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

4.4 Shopping basket (10 Minutes)

.1 Items can be added and removed

Test 1: Add item

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

Test 2: Remove item

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

Test 3: Empty Basket

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

.2 Quantity is correct (10 Minutes)

Test 1: Display correct quantity

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

Test 2: If requested quantity is greater than stock, corrects values

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

Test 3: Displays correct quantity for multiple books

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

.3 Pricing

Test 1: Displays correct price of book

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

Test 2: Displays correct total price

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

Test 3: Display correct price for multiple books

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

.4 Checkout

Test 1: Redirects to checkout page when clicked

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

Test 2: Updates stock

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

4.5 Checkout (10 Minutes)

.1 Display

Test 1: Correct books display

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

Test 2: Correct prices display

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

Test 3: Correct totals display

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

.2 Paynow

Test 1: Redirects to paynow screen

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

4.6 Paynow

.1 Display (5 Minutes)

Test 1: Insert 16-bit Account number

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

Test 2: Insert 4-bit Pin number

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

.2 Paying (5 Minutes)

Test 1: Sends back to homepage

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

Test 2: Empties cart

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

5.1 Add Stock (10 Minutes)

.1 Add new book to stock

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

.2 Add existing book to stock (Should simply update quantity)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

.3 Add book to stock with empty fields

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

.4 Add book to stock with incorrect fields

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Evidence |  | Comments |  |

**5.5 External Testing**

**TESTING**

Basic functional tests:

* Login
* Homepage display, links
* Adding/Removing from cart
* Checkout (stock error)
* Payment
* Stock Screen
* Adding Stock
* Log Out

Out of scope tests:

* Security
* SQL injection

Test time: 1 hour 35 minutes

**Login**

1. **Actual Login (10 mins)**

A) Login with the username=’customer1’ password=’p455w0rd’

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | User can login with this information |

B) Login with the username=’customer2’ password=’p455w0rd’

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | User can login with this information |

C) Login with the username=’admin’ password=’p455w0rd’

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Admin can login with this information. |

1. **Wrong Login (10 mins)**

Enter invalid data, try:

Correct username and a wrong password

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | It does not login if user has wrong password |

Incorrect username and correct password

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | It does not login if the username is wrong but the password is |

Incorrect username and incorrect password

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | If both inputs are wrong it does not login |

1. **SQL Injection (10 mins)**

Try logging in with User = admin Password = Test’ OR 1=1

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Does not allow user to login |

Try logging in with User = admin Password = Test’ OR 1=1#

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Does not allow user to login |

Try logging in with User = admin Password = Test’ OR 1=1—

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Does not allow user to login |

1. **Home page without Login (5 mins)**

Try accessing your URL without the ‘/login’ at the end without logging in to get to the homepage.

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Redirects you to login page |
|  |  |  |

**Homepage**

**Login as customer1**

1. **Display (5 mins)**

Login as customer1

Using sqlite3 in the command line, check that all books with quantity above 0 are displayed

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No | N/A | Does not include how to complete this test. Assumes tester is using Codio |

Check the navigation bar has been displayed appropriately

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | No issues with display |
|  |  |  |

1. **Log out (5 mins)**

Check the log out button redirects you to the login page

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Logout redirects you to correct page |

Try and access the home page using ‘/’ at the end of the URL now you have logged out.

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Does not allow you to access homepage |

**Login as admin**

1. **Display (5 mins)**

Login as admin

Using sqlite3 in the command line, check that all books with quantity above 0 are displayed

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Does not include how to complete this test. Assumes tester is using Codio |

Check the navigation bar has been displayed appropriately

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Correct navigation |

1. **Stock (5 mins)**

Click the stock button and see what it displays

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Stock shows the stock page |

**Cart**

**1. Adding Books (5 mins)**

Add one book

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Able to add sing book to cart |

Add multiple of the same book

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Able to add multiple books to cart |

**2. Deleting Books (5 mins)**

Delete one book

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Able to delete single book |

Delete multiple of the same book

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Able to delete multiple books |

**3. Emptying Cart (5 mins)**

‘Cancel’ your cart

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Empties the cart |

**Checkout (10 mins)**

Press checkout with one book in cart

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Able to checkout with current cart items |

Press checkout with 50 of one book in cart

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Able to checkout with multiple cart items |

Press ‘Pay Now’

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Pay Now Takes you to Pay now Screen |

**Payment (10 mins)**

Enter correct card credentials as displayed and submit

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Correct input accepted |

Enter incorrect card credentials as displayed and submit

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Incorrect input is still accepted and clears carts |

**Stock Screen(5 mins)**

While logged in as admin, click ‘Stock Levels’

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Admin is able to see Stock Levels |

Click any of the book links to the left

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Book link takes you to desired book |

Click ‘Add Stock’

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Add stock adds the book to the stocks |

**Adding Stock(15 mins)**

Add a book that doesn’t exist

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Book is correctly added to the database. All values are correctly added.  The description box could have been made bigger, as is it is very difficult to see full description. |

Add a book that already exists using the Isbnb number

|  |  |  |
| --- | --- | --- |
| Pass | Evidence | Comments: |
| Yes/No |  | Quantity was updated but did not add multiple copies. |

* 1. **External Testing Evaluation**

The quality of this testing is extremely high. All the functionality is tested, and tests all of the sections efficiently.

Due to the testing a few bugs were even discovered that could improve the quality of future iterations, such as the mistake with the “Pay Now” page as the user can input any value and it will confirm the purchase regardless.

The Main issue found with the test was during the display test for “customer1” and “admin”, it asks that the user use sql to check if the quantity is correct but does not tell the user how to do this. In the future make these sections clearer for the user.

1. **QUALITY ASSURANCE this has been changed from GitHub Version**

**Brief and Requirements**

The website was able to fulfil are required features. It has a login page, with 3 users and one being admin. It has a homepage that shows all the products available. It allows admins to access the Add Stock page in order to add stock to website, does not create duplicates if same book added twice but does update quantity. Users can view current stock as well as add items to the cart. Once added they can checkout product and insert payment information. They can also clear the cart when they want. The book database contains 10 books, each with: Book Name, Quantity, Retail Price, Trade Price, Book Cover, ISBN, Author Name and Release Date.

**Coding Standards**

Code is some-what reusable, some of the codebase was adapted from Weeks 1-5 on the Aula webpage (<https://coventry.aula.education/#/dashboard/d0f2228a-9eff-445e-953d-ebf4082fa218/journey/materials/20520568-9f24-4d32-a514-394b98fd0910>) . All the code is commented for user understandability., this is to ensure the user understand what each section of code is doing. So even if the code itself is not understood the purpose of that section is and can be updated or changed without interfering with other sections of code. Code was divided into appropriate functions with each function being appropriately named for user understanding. Variables used through out have appropriate names, to ensure readability, and if the user wishes to update the code, they know what each variables intended.

**Documentation Standards**

This documentation gives a decent explanation of how the main section of the code operates, as well as going through the installation process, Testing requirements and Code Purpose. It gives a decent documentation on all of these. Each section is laid out and structured.

**Time Delivery**

Code was completed and submitted within a given deadline.

.2 **External QA Evaluation 2122 PLACEHOLDER BOOKSHOP REPORT**

|  |
| --- |
| The placeholder QA statement is very brief. It has numerous spelling mistakes through out the statement. It is unfinished, in the coding standards it states the codebase was adapted from a repository and it simply says “<give source>”. This is a very poor attempt at the coding standards. It states the code is re-usable, but it is simply taken from a different code base, the comments were copied as well. Far more detail is needed in this section, but the user has very little done in all sections. The documentation standards shows that the user understands the documentation needs more work.  In the future the developer should attempt to ensure more of the brief and requirements are met. Only the shopping cart was completed, and even this was taken from somewhere else. In doing so the usability of the code will be improved. |

1. **DOCUMENTATION this has been changed from GitHub Version**
2. Section 1 – Code Purpose
3. Section 2 – Code Location
4. Section 3 – Installation Guide
5. Section 4 – Code Explanation
6. Section 5 – Testing
   1. Developer
   2. Demo
   3. Test Scope
   4. Tests
   5. External Test
7. Section 6 – Quality Assurance
   1. Internal QA
   2. External QA
8. Section 7 – Documentation List
   1. Internal Documentation
   2. External Documentation
9. Section 8 - References
   1. **External Documentation Matthew Clarke** **10274000**

|  |
| --- |
| **Code Purpose**  Accurate brief explanation of the code’s overall purpose. It contains all basic functions required of the website. It does not mention the website contains a login page, although it does refer to the users and admins. |
| **Code Location**  Gives the correct download link that directs the user to GitHub, from here they can download the code. The GitHub contains all the necessary files in order to run the program. It is appropriately titled, “Mattys-Bookshop” The Documentation is given its own folder, so the user knows where it is. |
| **Code Installation**  The installation guide gives an extremely thorough guide on how to properly set up the code inside of Codio. It ensures the user is correctly updated on all necessary extensions. It includes images with direct arrows to each section, so the user can follow. It makes the assumption the user is using Codio but the task was designed for Codio so this is not a fault of the developer. |
| **Code Explanation**  All the files that are used are listed beforehand, so the user knows what the explanation will describe.  The login functions are appropriately described, detail is given for each function for the user to understand it. We can even see the code is commented on most lines, so the user can follow that as well.  Each main function is described and explained in full detail.  The use of sub-headers for each function would have been useful to divide everything and make each section more visible.  Only the Admin Html file is described in total, but it is well detailed. The rest of the HTML files would not be too different from this. |
| **Testing**  The testing is very well done. Each section has at least one test for its main functionality.  Most sections contain a large array of tests, which test for outliers as well, like testing for incorrect login information as well as correct inputs.  Some of the later tests are less detailed, for example Pay Now only tests for the correct inputs. |
| **Quality Assurance**  The Quality Assurance contains all the requires sections. It talks about the brief and which were met. It talks about the functions the user implemented.  It talks about what the user could have improved upon inside the documentation. Such as going into more detail in the code explanation. |

**8 REFERENCES**

* <https://docs.python.org/3/>
* <https://flask.palletsprojects.com/en/2.0.x/>
* <https://werkzeug.palletsprojects.com/en/2.0.x/>
* <https://docs.microsoft.com/en-us/sql/?view=sql-server-ver15>
* <https://docs.python.org/3/library/functools.html>
* CSS – Week 5 AULA SHOPPING\_CART\_FLASK (<https://coventry.aula.education/#/dashboard/d0f2228a-9eff-445e-953d-ebf4082fa218/journey/materials/027b027e-0084-4de2-a0c2-02648c1eb585>)
* Array\_merge Function – Week 5 AULA SHOPPING\_CART\_FLASK (https://coventry.aula.education/#/dashboard/d0f2228a-9eff-445e-953d-ebf4082fa218/journey/materials/027b027e-0084-4de2-a0c2-02648c1eb585)