

School of Computing, Engineering and Mathematics (CEM)

Faculty of Engineering, Environment and Computing (EEC)

**5001CEM SOFTWARE ENGINEERING** | 2122

**PROJECT REPORT**

**NAME:** Harinderveer Singh

**SID:** 10267530

1. **CODE PURPOSE**

The code is for a web application developed using python programming language and flask framework. The application is for a bookshop where user can buy books, they can also register and login to their accounts.

1. **CODE LOCATION**

The code is posted on the university GitHub: https://github.coventry.ac.uk/5001CEM-2122/harinder\_bookshop

1. **CODE INSTALLATION**

The code needs to be download from the github link above. Unzip the downloaded file.

On codio create new project select python language and give a name and create the project.

Go to file select upload and choose the unzipped folder.

Use “cd” command to go into the directory

Go to tools and open terminal a do the following commands one by one in the same order:

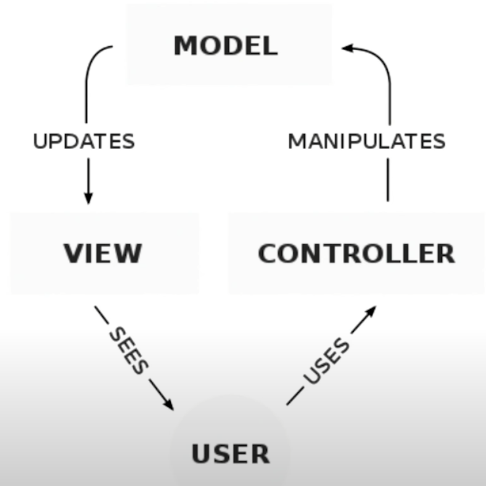
1. sudo apt update
2. sudo apt-get install python3-venv
3. python3 -m venv venv
4. . venv/bin/activate
5. pip install Flask
6. export FLASK\_APP=app
7. flask run --host=0.0.0.0

In the navbar click on the small arrow next to “Project Index(static)” and click on Box URL and then change 3000 to 5000

1. **CODE EXPLANATION**

The code was developed to make a basic bookshop web application. The main application code is inside the app.py, the templates folder has all the page layout files and static folder has the styles.css which is used by the page layouts and there is a folder called book-images which contains all the images of the books. It was designed to allow the users to look at the books available in the store with the option to create an account, login and purchase the books. It was also designed to have the shopkeeper as the admin, where they had more functionality, like having the options to see and change the stock levels of the books and a form to add any new books in the future.

The web app is developed using Python programming language, SQL is used for database and HTML and CSS for the webpage and styling. The web app framework is Flask, which uses the model-view-controller pattern.



The user when first goes to the web app ask for the

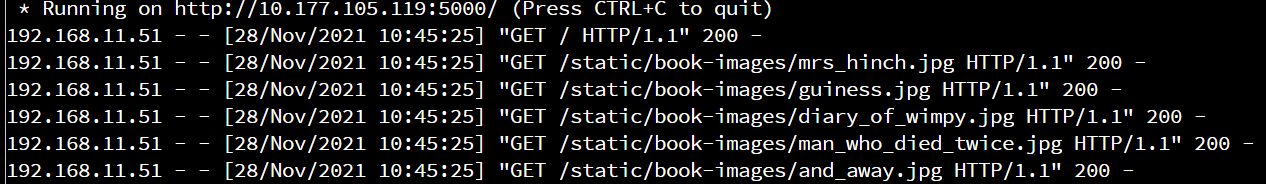
The controller is our python code, it is the brain of our web app, as it controls which data needs to be taken from the database and then displayed to the user.

The view is our HTML and CSS code to produce a user interface, this is the information that the user can see on their screen.

The model is the data and how we manipulate it, in this case I am using the SQLite database.

An example of this model in our app:

When the user goes to our web app with URL they send a GET request to the server.



The server then goes through our code and finds the code to run. In this case the URL ends with “/”. So it executes the following code.

**@app**.route('/')

**def** **homepage**():

**try**:

con = sqlite3.connect('bookshop.db') #Connect to the database

cur = con.cursor(); #Create cursor object

cur.execute("SELECT \* FROM books WHERE quantity!= 0 ") #Select all data from the row if the value in column quantity is not 0#

rows = cur.fetchall() #Store data as variable rows

**return** render\_template('index.html', books=rows) #Render the page and pass the data to template

**except** **Exception** **as** e: #If there is exception error print it

print(e)

**finally**:

cur.close() #Close the cursor object

con.close() #Close the connection with database

This is the code for the homepage. So the controller manipulates the model because it creates a connection with database and executes the query and gets all the data from the database.

Then the model updates the view by rendering the template named “index.html” and sending the data from the database as a variable “books”.

The code for index.html.

{% **extends** "base.html" %}

{% **block** tabtitle%}

Homepage

{% **endblock** %}

{% **block** header%}

Bookshop

{% **endblock** %}

{% **block** body%}

<a href="/stock\_level"><button>Stock level</button></a>

<section class="py-5">

<div class="container px-4 px-lg-5 mt-5">

<div class="row gx-4 gx-lg-5 row-cols-2 row-cols-md-3 row-cols-xl-4 justify-content-center">

{% **for** row **in** books %}

<form action='/add\_to\_cart' method='post'>

<div class="col mb-5">

<div class="card h-100">

<img class="card-img-top" src="/static/book-images/{{ row[**4**] }}"alt="" height="400px", width="350px" />

<div class="card-body p-4">

<div class="text-center">

<h5 class="fw-bolder">{{ row[**0**] }}</h5>

</div>

</div>

<div class="card-footer p-4 pt-0 border-top-0 bg-transparent">

<div class="text-center"><a class="btn btn-outline-dark mt-auto" >Add to cart</a></div>

</div>

</div>

</div>

</form>

{% **endfor** %}

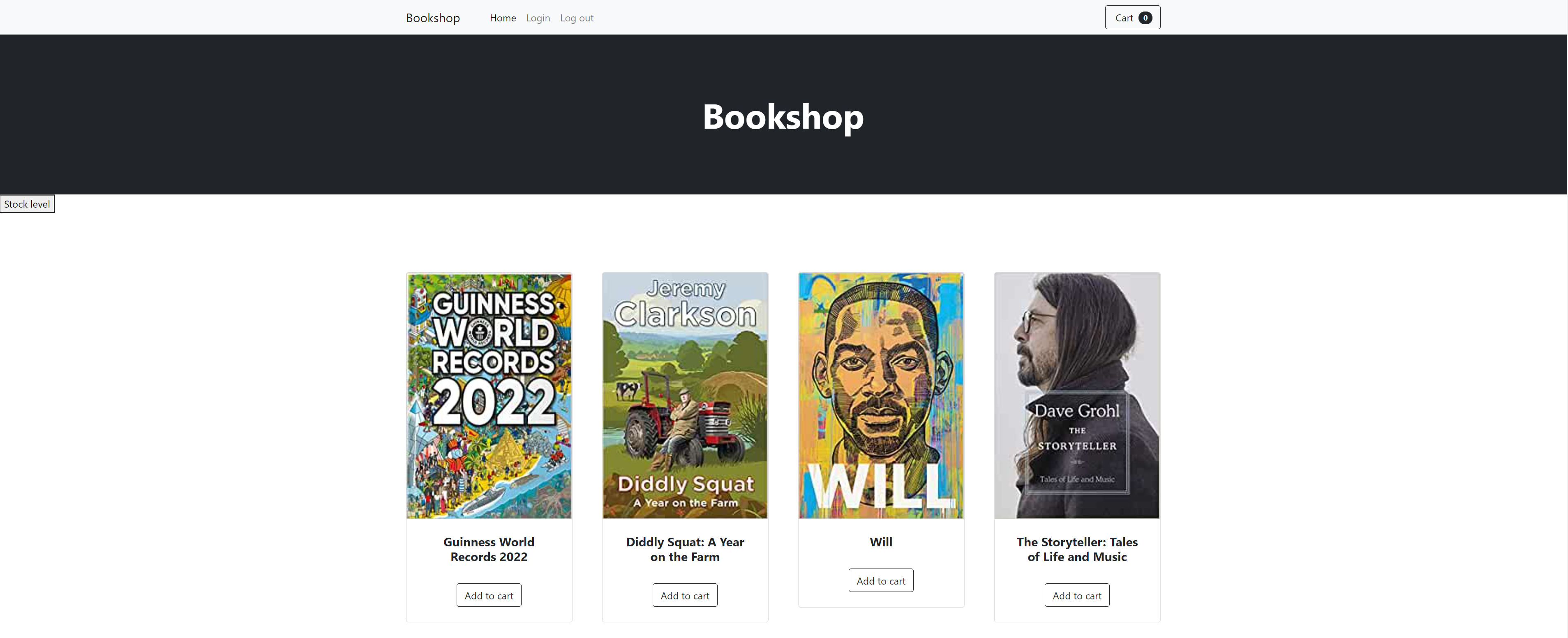
</div>

</div>

</section>

{% **endblock** %}

The code uses a bootstrap template to style the page, but there is a “for loop” that goes through the variable books and displays the information that we get from the database.



This is the process of interaction between the user and our web app. So if the user click on login it would go through the same process of going to the controller, manipulating the data and viewing it to the user.

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

**def** **login**():

**if** request.method == "POST": #If the request method is POST

u=request.form['name'] #Store value from the form as u

p=request.form['pass'] #Store value from the form as p

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

cur = con.cursor(); #Create cursor object

cur.execute("SELECT username FROM customer\_login WHERE username = ? AND Password=?;", (u, p)) #Select the username row from table cusutomer\_login if the value in column username is same as varaible u and value p in column password

x = cur.fetchone() #The data from query is stored in varaible x

**if** x **is** **not** **None**: #If the value of x is not None meaning there is this username in the database

name= request.form.get("name") #Get the name from the for and store it inside varaible name

session["name"] = name #Set session name to name

**if** name == 'admin': #If the name is admin

**global** user\_is\_admin

user\_is\_admin = **True**; #Set global variable to True

**return** redirect(url\_for('homepage')) #Redirect to homepage

**else**:

msg="Username not registered. Please sign up." #Set msg to message inside string

**return** render\_template('login.html',msg=msg) #Render the login page and pass the variable msg

**else**:

**return** render\_template('login.html') #Render the login page

The function for **login** allows users who are already registered to login. The function has 3 nested if statements.

The outer if statement checks if the form sent data with post method. If that is not True, then it renders the login page. But if it is True then the function will get data from the form and store the username and password inside variables (u and p) and then it performs the check to see if the login details are correct and there is a user with these login details. It connects to the database and perform a query which select the username from the customer\_login table only if the details from the form are matching details inside the database and it will store the result in variable x. Then it goes to the second if statement.

The second if statement checks if variable x does not have the value None. If is it none it means the details did not match any of the records inside our database and that the details are wrong, so it renders the login page again but shows message to user that there is no username with these details. If the variable x is not None so it has found matching details. It goes to the inner most if statement.

The inner most if statement is used to check if the user is admin. If it is then it sets the global variable “user\_is\_admin” to True and redirects the user to the homepage.

**@app**.route('/logout')

**def** **logout**():

session["name"]=**None** #Set session name to None

**global** user\_is\_admin

**if** user\_is\_admin == **True**: #If the global variable is True

user\_is\_admin= **False**; #Set it to False

**return** redirect('/') #Redirect the user to homepage

This function **logout** sets the session name to “None” and it check if the global variable “user\_is\_admin” is True. If it is True, then it sets that to False and redirects the user to homepage.

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

**def** **register**():

**if** request.method == "POST": #If the request method is POST

user=request.form['user'] #Store value from the form as user

passw=request.form['passw'] #Store value from the form as passw

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

cur = con.cursor(); #Create cursor object

cur.execute("SELECT \* FROM customer\_login WHERE username=?",(user,)) #Select all rows from table cusutomer\_login where the value in column username is same as varaible user

data= cur.fetchone() #The data from query is stored in variable data

**if** data **is** **not** **None**: #If the value of data is not None meaning there is this username in the database

msg="Username already used" #Set msg to string username already used

cur.close() #Close the cursor

con.close() #Close the connection

**return** render\_template('register.html', msg=msg) #Render the template for register and pass the varaible msg

**else**: #Else if data is none meaning there is no username as variable user

cur.execute("INSERT INTO customer\_login(username,password) VALUES (?,?);",(user,passw)) #Insert the data in variable user and passw into column username and password

con.commit() #Commit the change to database

cur.close() #Close cursor object

con.close() #Close connection

msg="User registration successful.Please use log in." #Set variable msg to successful registration

**return** render\_template('register.html', msg=msg) #Render the template for register.html and pass the variable msg

**else**:

**return** render\_template ('register.html') #Render the template for register.html

This is the **register** function it allows the user to register with their details.

The register function has two if statements, same as login the first if statement checks if form sent data with method POST. If it is not True it renders the template register.html. If the form did send data with method POST, then it stores the two value inside two variables (user and passw). To check if the username is already used by someone else, we connect to the database select all the rows from column username where variable user from the form is same as username stored inside the database. The result from this query is stored inside variable data.

The second if checks the variable data, if the value of this variable is not “None” that means the username is already used by someone else, so it closes the connection to database, render the register template again and passes the variable msg which tells the user that username is already used.

But if the value of data is “None”, so the username entered is not used before. It inserts the username and password into the database in the correct columns and then commits the query. It renders the register template and passes the message that the user registration was successful. After this the user can log in.

**@app**.route('/stock\_level', methods=["GET","POST"])

**def** **stock\_level**():

**global** user\_is\_admin

**if** user\_is\_admin == **False**: #If variable is false

**return** render\_template('unauthorized.html') #Render the unauthorized template

**else**:

con = sqlite3.connect('bookshop.db') #Connect to the database

cur = con.cursor(); #Create cursor object

cur.execute("SELECT book\_name,isbn,front\_cover,quantity FROM books ") #Select the data from column book\_name,isbn,front\_cover, quantity from table books

rows = cur.fetchall() #Store data from database in variable rows

**return** render\_template('stock\_level.html',rows=rows) #Render the stock\_level template and pass rows variable

app.config['IMAGE\_UPLOADS'] = "/home/codio/workspace/bookshop/static/book-images" #Set image\_upload in app configuration to the absolute path of book-images folder

The **stock level** should only be accessible by the admin as it shows all the books in the store with their isbn and quantities and ability to change stock. The function first checks if the global varaible user\_is\_admin is False, if it false it render the template unathorized.html which tells the user that it only of admin use.

If the variable is True then the function moves to else statement and connects to the database and it selects the book name, the front cover, ISBN and quantity from the database and it renders the template stock\_level.html and passes the data from the database as variable to the template.

**@app**.route('/add\_book', methods=['GET', 'POST'])

**def** **add\_book**():

**global** user\_is\_admin

**if** user\_is\_admin == **False**:

**return** render\_template('unauthorized.html')

**elif** request.method == "POST": #If request method is POST

book=request.form['bookname'] #Get input from the form and store inside a variable book

author=request.form['author'] #Get input from the form and store inside a variable author

isbn=request.form['isbn'] #Get input from the form and store inside a variable isbn

desc=request.form['desc'] #Get input from the form and store inside a variable desc

tp=request.form['tp'] #Get input from the form and store inside a variable tp

rp=request.form['rp'] #Get input from the form and store inside a variable rp

quantity=request.form['quantity'] #Get input from the form and store inside a variable quantity

date=request.form['date'] #Get input from the form and store inside a variable date

image=request.files['pic'] #Get input from the form and store inside a variable pic

image.save(os.path.join(app.config['IMAGE\_UPLOADS'],image.filename)) #Save the image in the book-images folder

image\_name=image.filename #Set variable to name of the image from the form

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

cur = con.cursor(); #Create cursor object

cur.execute("INSERT INTO books(book\_name,author\_name,isbn,book\_description,front\_cover,trade\_price,retail\_price,publication\_date,quantity) VALUES (?,?,?,?,?,?,?,?,?);",(book,author,isbn,desc,image\_name,tp,rp,date,quantity)) #Insert the data in variables into the correct column in database

con.commit() #Commit changes to database

**return** redirect(url\_for('stock\_level'))

**else**:

**return** render\_template('/add\_book\_form.html')

The **add\_book** function allows the admin to add a new to the store. First it checks if the user is admin, if it is not then it renders the authorized.html page. If the user is admin then it checks if the form has sent POST request, if it is not True then it renders the add\_book\_form.html. If it is True, then it request all the values from the form and stores them inside a variables. Then connects to the database and does an INSERT query to add all the values from the variables to all the correct columns inside the database. The front cover is stored differently, the name of the image is store inside the database, but the actual image is stored inside the folder static/book-images.

1. **TESTING**

.1 TESTING REGIME

4.1 Registration (5 mins)

.1 Register new account

Test 1: Register a new account with new details

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments |  |

.2 Usernames already in use should not be allowed to be used again

Test 2: Register new user but with username: customer1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments |  |

4.2 Login (10 mins)

.1 Login with existing user details

Test 1: Login with existing customer details

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments |  |

Test 2: Login with correct username but wrong password

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments |  |

Test 3: Login with wrong username but correct password

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments |  |

Test 4: Login with the new account created on test 4.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments |  |

4.3 Shopfront (30 mins)

.1 System deals with n-sized data

Test 1: The number of database records should match with the number of books display on homepage

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments |  |

Test 2: Delete one of the records from database and repeat Test 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments |  |

.2 If book quantity is 0 it should not be shown on the homepage, but should be shown in stock level

Test 1: Set quantity to 0 on one of the records and check homepage.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments |  |

Test 2: Set quantity to 0 on one of the records and check stock level.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments |  |

.3 Only the admin should be able to open stock level page and add book page

Test 1: Login as admin open stock level page

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments |  |

Test 2: Login as admin open add book page

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments |  |

Test 3: Login as customer open stock level page

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments |  |

Test 4: Login as customer open add book page

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments |  |

4.4 Adding books (15 mins)

.1 Books can be added only if you fill all the details in the form

Test 1: Add a new book – fill all details

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments |  |

Test 2: Add a new book – without filling book description and quantity

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments |  |

Test 3: Add a new book – fill all details (set retail price to two decimal points)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments |  |

.2 TESTS RUN ON EXTERNAL CODE

1. **QUALITY ASSURANCE**

.1 QUALITY ASSURANCE STATEMENT

The first two features from the brief are implemented and the third feature is partly implemented as the shopping icon does not show the total cost and the number of books. The feature 4 and 5 are not implemented. The feature two the web app shows the stock level but it does not have the option to increase or decrease the stock level. This main function of the web app is as a bookshop that allows user to buy books. So without feature 4 and 5 the project is not fully developed and is missing the most important feature.

Coding standards - The code is very basic and made up of small function that are reusable. The code is commented.

Team details - Harinderveer Singh, Ibrahim Issa and Hassam Ahsan

Timely Delivery - The code and report were not submitted on time, they were late by approximately 2 hours.

.2 EXTERNAL QA EVALUATION

1. **DOCUMENTATION**

.1 DOCUMENTATION LIST

Documentation that is included in this report:

1. Code on GitHub
2. Code purpose
3. Code location
4. Installation instructions
5. Code explanation
6. Testing Regime
7. QA Statement

.2 EXTERNAL DOCUMENTATION INSPECTION

**REFERENCES**

All the material from the labs

Flask application - <https://www.youtube.com/watch?v=x_c8pTW8ZUc&ab_channel=CS50>

Form data/HTTP requests - <https://www.youtube.com/watch?v=9MHYHgh4jYc&list=PLzMcBGfZo4-n4vJJybUVV3Un_NFS5EOgX&index=4&ab_channel=TechWithTim>

Login - <https://stackoverflow.com/questions/16469366/flask-login-not-sure-how-to-make-it-work-using-sqlite3>

Upload image - <https://www.youtube.com/watch?v=6WruncSoCdI&list=PLF2JzgCW6-YY_TZCmBrbOpgx5pSNBD0_L&index=15&ab_channel=JulianNash>

Upload image - <https://blog.miguelgrinberg.com/post/handling-file-uploads-with-flask>

SQL query -- <https://www.youtube.com/watch?v=HX-ChCQfJEo&ab_channel=Pythonist>

SQLITE queries- <https://docs.python.org/3/library/sqlite3.html>

CSS style - <https://www.youtube.com/watch?v=fGtB9VRHQb0&list=RDCMUCBlr2jG1onljL-gUy9bbhJw&index=3&ab_channel=CodingLab>

Bootstrap template - <https://startbootstrap.com/template/shop-homepage>

Slider and input - <https://stackoverflow.com/questions/13475016/how-to-update-html5-range-on-change-of-a-text-input>