

School of Computing, Engineering and Mathematics (CEM)

Faculty of Engineering, Environment and Computing (EEC)

**5001CEM SOFTWARE ENGINEERING** | 2122

**PROJECT REPORT**

**NAME:** Ernest Zimus

**SID: 10135033**

1. **CODE PURPOSE**

The purpose of this code is a functioning book shop where you can register, login, add stock items, check stock items, see products and buy them.

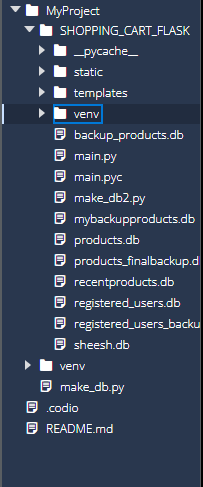
1. **CODE LOCATION**

Main project ocated: <https://github.coventry.ac.uk/5001CEM-2122/EZ-book-shop>

Checkout located in: <https://github.coventry.ac.uk/5001CEM-2122/flask-checkout>

1. **CODE INSTALLATION**

Download the zip file (thebookshop.zip), create a new python project in codio.co.uk, put all of the contents of the zip file to the codio project so it looks like this below. (IT MIGHT FLASH THAT THERE ARE SOME EXISTING FILES, JUST IGNORE IT)



After extracting the code into the project, you must open the terminal and write these commands in order for the store to work:

cd MyProject

pip install Flask

pip install Flask-SQLAlchemy

pip install flask-cors

sudo apt update

sudo apt-get install python3-venv

python3 -m venv venv

. venv/bin/activate

cd SHOPPING\_CART\_FLASK

export FLASK\_APP=main

flask run --host=0.0.0.0

that’s it for this part of commands, if you get prompted with a syntax error that looks like this Text

Description automatically generated

change the code line 165

from this: checksumstr = f"pid={pid:s}&sid={sid:s}&amount={all\_total\_price:.1f}&token={secret:s}"

to this checksumstr = "pid={pid:s}&sid={sid:s}&amount={all\_total\_price:.1f}&token={secret:s}"

if not, continue

ok as store front is fully functional, there is more to install, next to do is the checkout, download the zip file called from the other listed repository checkout.zip and then create another new python project and extract all of the files from the zip file into the project so it looks like that:

Text

Description automatically generated with low confidence please note that there will be no venv folder yet it will install itself after you input all of the required commands.

Okay so for the commands you must input are these:

sudo apt update

sudo apt-get install python3-venv

python3 -m venv venv

. venv/bin/activate

pip install Flask

pip install Flask-SQLAlchemy

pip install flask-cors

./venv/bin/pip3 install -e .

pip3 install PyJWT

pip install flask-wtf

Now to run it you need to use this command:

env FLASK\_APP=payments FLASK\_ENV=development ./venv/bin/flask run --host=0.0.0.0

Remember to use port 5000 in the url otherwise it will not run.

You have to get a new seller id, secret key, payment id that you will input into the main.py file of the store, use screenshots for reference.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text

Description automatically generated

Save the Seller ID and Secret Token somewhere for yourself and for future use and as well create a Payment ID name of your choice and save it too.

Graphical user interface, text, application, website

Description automatically generated

Put in the amount 5000 and create a request. Remember don’t close the connection in the terminal, keep it open during all time.

Graphical user interface, text, application, email

Description automatically generated

Keep this page open. You will need some html editing the marked url has to be replaced in of the forms. Open products.html in the previous codio project you have done AS WELL DO THAT IN adminproducts.html and go to the line 60 and replace it like that:

A picture containing text

Description automatically generated

Okay as url has been updated, we have still not done the part with seller id, payment id, secret key.

Open the file main.py and change the values according to the screenshot, keep in mind your values might be different.

Text

Description automatically generated

As you have done that press ctrl+s and if the store project is still running, press ctrl+c in the terminal so it stops running

And run the store again by writing in the terminal:

export FLASK\_APP=main

flask run --host=0.0.0.0

(keep in mind you must be in the directory called SHOPPING\_CART\_FLASK)

That is it everything should run now and the checkout has to work.

1. **CODE EXPLANATION**

**from** **flask** **import** Flask

**import** **sqlite3**

**from** **flask** **import** flash, session, render\_template, request, redirect, url\_for

**from** **werkzeug.security** **import** generate\_password\_hash, check\_password\_hash

**from** **markupsafe** **import** escape

**from** **flask** **import** abort

**from** **flask** **import** make\_response

**import** **sqlalchemy**

**from** **hashlib** **import** md5

*#all of the above are modules used for the code to work*

app = Flask(\_\_name\_\_)

app.secret\_key = "secret key" *#used for secure checkout*

sid = 'YxfIMEp1a2Vib3g=' *#used for secure checkout*

pid = 'payment1' *#used for secure checkout*

secret = 'QSDZ1zhECGazdOC9\_ygF3pUUOfIA' *#used for secure checkout*

@app.route('/register', methods=['GET', 'POST']) *#used to create url directories, methods methods used for the html forms to make a connection between them*

**def** register():

**if** request.method == 'POST':

**return** do\_the\_registration(request.form['uname'], request.form['pwd']) *#grabbing username and password from the register form*

**else**:

**return** show\_the\_registration\_form();

**def** do\_the\_registration(u,p):

con = sqlite3.connect('registered\_users.db') *#connects to registered\_users database*

**try**:

con.execute('CREATE TABLE users (name TEXT, pwd INT)') *#if table users does not exists creates a new one with name and pwd attributes*

**print** ('Table created successfully');

**except**:

**pass** *#if table users exists, skips the "try:" part*

con.close() *#closes connection of the connected database*

con = sqlite3.connect('registered\_users.db')

con.execute("INSERT INTO users values(?,?);", (u, p)) *#inserting username and password into the database whilst values are taken from the register() function*

con.commit()

con.close()

**return** show\_the\_login\_form()

**def** show\_the\_registration\_form():

**return** render\_template('register.html',page=url\_for('register')) *#page=url\_for is used to redirect to exact function*

@app.route('/', methods=['GET', 'POST']) *#route only is with a slash as this is the main page - login*

**def** login():

**if** request.method == 'POST':

**return** do\_the\_login(request.form['uname'], request.form['pwd'])

**else**:

**return** show\_the\_login\_form()

**def** show\_the\_login\_form():

**return** render\_template('login.html',page=url\_for('login'))

**def** do\_the\_login(u,p):

con = sqlite3.connect('registered\_users.db')

cur = con.cursor();

cur.execute("SELECT count(\*) FROM users WHERE name=? AND pwd=?;", (u, p)) *#fetching username and password and see if it matches*

x = cur.fetchone()

**if**(x[0])>0:

session['username'] = u

*#return redirect (url\_for('store'))*

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

**else**:

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

*#if x is not None:*

*#print('in login details')*

*#title = str(x[1])*

*#name = str(x[2])*

*#logged\_in(title,name)*

*#, title=title,name=name)*

@app.route('/stock') *#route to see book stock levels*

**def** stocklevel():

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

con.row\_factory = sqlite3.Row

cur = con.cursor()

cur.execute("SELECT \* from products")

rows = cur.fetchall()

con.close()

**return** render\_template("stock.html",rows = rows) *#prints out all of the books that are in stock*

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

**def** updatenewstock(): *#gets book attributes from html form and transfers them to the insertNewStock function*

**if** request.method == 'POST':

**return** insertNewStock(request.form['aiidd'], request.form['bname'], request.form['ccode'], request.form['iidir'], request.form['aprice'], request.form['ddscr'], request.form['ddate'], request.form['tprice'], request.form['quant'], request.form['aauth'])

**else**:

**return** showstockupd();

**def** showstockupd():

**return** render\_template('addstock.html') *#rendering the html form to input values*

**def** insertNewStock(a, n, c, i, p, d, w, t, q, z): *#grabs values that were put in by the user from the html form and then inserts it into the products sql database*

con = sqlite3.connect('products.db')

con.execute("INSERT INTO products values(?,?,?,?,?,?,?,?,?,?);", (a, n, c, i, p, d, w, t, q, z))

con.commit()

con.close()

**return** showstockupd()

This is the code for the login form.

<!DOCTYPE html>

**<html>**

**<head>**

**<meta** http-equiv="content-type" content="text/html; charset=UTF-8"**>**

**<title></title>**

**</head>**

**<body>**

**<h1** align="center"**><br>**

**</h1>**

**<div** align="center"**><img** moz-do-not-send="true"

src="https://images.pexels.com/photos/415071/pexels-photo-415071.jpeg?auto=compress&cs=tinysrgb&dpr=2&h=650&w=940"

alt="Book store Logo" width="250" height="200"**><br>**

**<br>**

**<br>**

**<h1>**The Book shop**</h1>**

**<p>**Restricted Access. Please log in below.**<br>**

**</p>**

*<!--This is a login form, alternative one exists for the login, primitive principle for it is used just stores inputted values and forwards them to the requested place for the data to be processed-->*

**<form** action="{{page}}" method="post"**>** **<label** for="uname"**>**Username:**</label><br>**

**<input** id="uname" name="uname" value="" type="text"**><br>**

**<label** for="pwd"**>**Password:**</label><br>**

**<input** id="pwd" name="pwd" value="" type="text"**><br>**

**<br>**

**<input** value="Submit" type="submit"**>** **</form>**

**<p><a** href="{{url\_for('register')}}"**><button>**Register instead**</button></a></p>**

**</div>**

**</body>**

**</html>**

This is the code for displaying the stock of the book shop.

<!doctype html>

**<html>**

**<body>**

**<p>**To inster new stock click the button**</p>**

**<p><a** href="{{url\_for('updatenewstock')}}"**><button>**Add new stock**</button></a></p>**

**<table** border =1**>**

*<!—That is a table that is built for displaying stock, mainly to explain lines like* {{row["id"]}}

*Are fetched from the stocklevel() function in the main.py file-->*

**<thead>**

**<td>**ID**</td>**

**<td>**Name**</td>**

**<td>**Author**</td>**

**<td>**ISBN-13**</td>**

**<td>**Price**</td>**

**<td>**Date**</td>**

**<td>**Description**</td>**

**<td>**Image**</td>**

**<td>**TradePrice**</td>**

**<td>**Quantity**</td>**

**</thead>**

{% for row in rows %}

**<tr>**

**<td>**{{row["id"]}}**</td>**

**<td>**{{row["name"]}}**</td>**

**<td>**{{row["author"]}}**</td>**

**<td>**{{row['code']}}**</td>**

**<td>**{{row["price"]}}**</td>**

**<td>**{{row['date']}}**</td>**

**<td>**{{row['description']}}**</td>**

**<td><img** src="/static/images/{{row['image']}}" width="80px" height="100px"**></td>**

**<td>**{{row["tradeprice"]}}**</td>**

**<td>**{{row["quantity"]}}**</td>**

**</tr>**

{% endfor %}

**</table>**

**<p>**To go back to the store click the button**</p>**

**<p><a** href="{{url\_for('products')}}"**><button>**Return to the store**</button></a></p>**

**</body>**

**</html>**

1. **TESTING**

TESTING REGIME

Book shop that has features such as simple register, login, store front, admin store front, viewing stock levels, adding new stock, checkout.

Using html, css, sql, flask, python. No really super advanced functionality in it. Security is basic nothing advanced or in other words barely exists.

**3** **Test scope**

* Basic tests of these components:
* Registration
* Login
* Shopfront
* Adding to and removing from cart
* Checking out

**Out of scope**

Removing stock items as that feature does not exist, adding duplicate isbn number books does not add up only just quantity. Checkout screens, only on codio console you will see checkout was successful, on the other side the page for you will not load up.

Bug: when user puts in items to cart and clicks pay will be redirected to the page which shows that the checksum value is wrong. I have been trying to solve that by checking the code, changing required values (creating a new merchant), nothing helped. I have original work where it works, taking that original work and putting it into a new codio project that just makes that feature stop working.

**4** **Test time**

Around 60 minutes

1. **Test regime**

**STUDENT ID: (Put yours here)**

1 Registration

.1 Test registration form

Test1: with new values, username coventry; password: university

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

Test2: open registration again and try duplicate values: user: admin; password: p455w0rd (to check if it has been created use a sql viewer of your choice and open registered\_users.db)

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

Test3: press **Login instead** button and see if it takes you to the login form:

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

2 Login

Test login form:

Test0: Test **register instead** button and then click **login instead** button, you will enter into register and login forms and make sure those buttons fully work:

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

Test 0.5: Try putting random login name and password, this should not let you into the store, will show you a screen with unsuccessful login, try to click the button that will take you back and see if it works:

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

Test1: Login as admin, user: admin; password: p455w0rd, click the login button, should redirect you to the screen where you click the button that the website asks to enter store front:

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

If successful continue to 4.4, if not you are doing something wrong, the code is correct.

4.4 Admin view :

Test1: Website shows that it’s the admin version and allows to click on **stock levels** button, after that shows the stock levels:

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

Test2: Stock table shows relevant information about the books, to check you can access the database products.db with your preferable sql viewer:

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

Test3: Button **Add new stock** works and redirects to the form where you input information about the book you want to add:

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

Test4: Input this data into the form and test if it adds it to the database, to check do this one with Test5 you will be taken to store front end and you will be able to see the new book that you have put in:

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

Test5: Check if you can return to store front page (displays products) using buttons:

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

Test 2: Login as basic user, username: either customer1 or customer2 ;  password: p455w04rd and see if it takes you to store products page, keep in mind basic user  does not see stock levels button:

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

3 Shopfront

.1 Images are correctly displayed

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

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

.2 Currencies are correctly displayed

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

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

.3 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 |  |

4 Shopping basket

.1 Items can be added and removed

Test 1: Add item

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

Test 2: Remove item

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

Test 3: empty cart

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

4 Checkout

.1 Payment is accepted

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

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

Test 2: Checkout successful, only terminal displays that, add books of your choice to cart, click pay, click accept payment and check terminal should be success in it:

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

.2 TESTS RUN ON EXTERNAL CODE

**TESTING REGIME**

**1 Developer name John Halloran**

**Tester SID 10135033**

**2 Project**

(material provided by collaborator)

**3**

**Test scope**

(material provided by collaborator)

3.1 out of scope

(material provided by collaborator)

**4 Test time**

(material provided by collaborator)

**5 Test regime**

(material provided by collaborator)

4.1 Registration (10 mins)

(material provided by collaborator)

4.2 Login

(material provided by collaborator)

4.3 Shopfront (20 mins)

.1 System deals with n-sized data

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

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

Graphical user interface, website

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

We can see that exact number of records in the database match the storefront, 8 products in the database, 8 in the storefront – pass.

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

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

Graphical user interface, text, application, email

Description automatically generatedAlternative database with expanded records is not provided. Populated the database manually with one line.

Graphical user interface, application

Description automatically generated

Had to resize page zoom to 25% to be sure that the product shows up on the website, well it does, but the code lacks automatic image resizing using css. As for the test – pass.

.2 Images are correctly displayed

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments | No issues with basic database as it does not ask to use alternative one for the test |

Graphical user interface, website

Description automatically generated

.3 Currencies are correctly displayed

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

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

Graphical user interface, website

Description automatically generated

.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 | No issues |

Graphical user interface, application, Excel

Description automatically generated

Datatype for price is DOUBLE which means it’s a number with 2 decimal points

Graphical user interface, text, application

Description automatically generated

Prices in database do not have any decimals as those are whole numbers, at the mean time on the website we can see .0 near prices so if we would put in .11 in any product price it would show up on the website, I have chosen to do that on the backpack:

A screenshot of a computer

Description automatically generated with medium confidence

Graphical user interface, application

Description automatically generated

We can see decimals on website and on database - pass

4.4 Shopping basket

.1 Items can be added and removed

Test 1: Add item

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

A screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with medium confidence

Test 2: Remove item

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

A screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with medium confidence

4.5 Checkout

.1 Payment is accepted

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pass | Yes / No | Fail | Yes / No | Comments | There is no pay button to do so. |

Graphical user interface, application, website

Description automatically generated

**Comments**

**Missing Empty cart button test, missing pay button therefore was not able to carry out the test for the payment system which is not even in this project. Image resizing does not exist in this project. Very abstract project only with features such as store front and only a shopping cart with an ability to add items/remove items, empty cart. Missing tests such as quantities in shopping carts, decimal numbers how that would display, changing an image of a pre made product. Overall this website needs a lot of improvements to be done. Including the test regime – must be more detailed.**

1. **QUALITY ASSURANCE**

.1 QUALITY ASSURANCE STATEMENT

Developed using HTML, CSS, Flask, Python, SQL. Contains not the most ideal features. Registration works, but there is probably a possibility of registering a duplicate user. Login works. Book shop greeting screen fully works. Admin version of book shop works, including ability enter stock levels page and also insert brand new books into it. The stock feature lacks removal of certain books or inputting duplicate books (when the isbn number matches the quantity does not update). Product list works, able to add any amount to cart (possible to add amount exceeding stock quantity), able to clear the cart, able to remove an item from the cart. Able to pay for the items. Checkout works successfully (that says in the console), but you will get redirected to the empty page which does not exist.

Bug: when user puts in items to cart and clicks pay will be redirected to the page which shows that the checksum value is wrong. I have been trying to solve that by checking the code, changing required values (creating a new merchant), nothing helped. I have original work where it works, taking that original work and putting it into a new codio project that just makes that feature stop working.

.2 EXTERNAL QA EVALUATION

Student ID 10248274

As if it is possible to login with 3 accounts of which 2 are customers and one of them is an admin that is well done, as

registration was not specified in the assignment brief and is not necessary it is okay, as specified fully working store front

with a shopping cart on one page where user can see books, add different quantities, see prices, interact with shopping

cart. The shopping cart should be hidden by the assignment brief, but it is not. It is possible to login with an admin account

and have an ability to enter the page to see the stock levels and add new stock, that is well done, but as said when

adding same book (matching isbn number) the duplicate line in the database creates instead of just the quantity updated. As

specified too the checkout is working but missing the final screens or so-called templates. What is written perfectly explains

what is for what, specifically features – description and what it lacks. Parts that are not working can be improved as it is

explained. Also the QA itself gives a view of what to expect from this product, and personally to say – a good one.

1. **DOCUMENTATION**

.1 DOCUMENTATION LIST

The user is able to install the project fully with the provided information above I have tested that myself. The user is able to see bits of the code explanation and understand what is for what. The quality assurance provided lets the person that installs and sees the project understand what to expect from it. The testing regime provided also lets the user understand what tests to do.

.2 EXTERNAL DOCUMENTATION INSPECTION

Student ID: 10107753

All of the documentation seems to be provided. As starting with the installation guide, it is lacking a tutorial on the

installation of checkout system, on the other side, the tutorial of how to install the book store itself is perfect and is step by

step with also images included. The testing regime is well done with many tests for the tester available but it lacks a bit of

organising which is not particularly necessary but it would save some time for the tester. As already seeing the bug report, I

can see that a lot of effort has been put in to this project. As about it, the bug report it does not lack anything and it is

perfect. As also project contains its architecture in flowcharts, it can be seen how well is everything planned before being

done in coding matters. The QA is long and explaining everything in detail, contains features – full explanation of them and

what the code is lacking. The documentation even contains the user manual, which is very nice to see as it really explains

how to use the features and it can be seen what they are capable of. The disclaimer is not that necessary, but it still shows

what improvements could be done on the code in the future. To sum up, everything is included what is needed and

everything is presented in a clear way how it should be.

**REFERENCES**

5001 CEM Lab tasks that can be accessed here:

https://coventry.aula.education/?#/dashboard/d0f2228a-9eff-445e-953d-ebf4082fa218/journey/materials/20520568-9f24-4d32-a514-394b98fd0910

5001CEM Lab tasks were used ranging from week 1 to week 7

<https://files.coventry.aula.education/d6d51e2422560aca9acd5b123a84853f5001cem_lab_set_up.pdf>

<https://files.coventry.aula.education/b36ab9f1913d9dae78306e5eae162ca45001cem_more_flask_v2.pdf>

<https://files.coventry.aula.education/e8894925b88a4babdd01557b7b0ed2635001cem_cookie_monsters.pdf>

<https://files.coventry.aula.education/6fdeb18edf6baecaa05c98aa73942af45001cem_2122_basic_database_setup.docx>

<https://files.coventry.aula.education/1897241ace77cac1dd778c026750c03a5001cem_2122_basic_register_and_login.docx>

<https://files.coventry.aula.education/eb0abff3df7ac0bae1287b0c185ad5a25001cem_2122_unit_monster_search_basic_solution.docx>

<https://files.coventry.aula.education/74ed05dea1ca87104b59cac0db51d7165001cem_2122_5_shopping_cart.docx>

<https://files.coventry.aula.education/4d5c7763258c8b95607c6d394df9373b5001cem_2122_6_flask_aalto_checkout.docx>

<https://files.coventry.aula.education/ee5329f13ead7f412f4e7456863a0e4f5001cem_2122_7_github_version_control_1.docx>

The main template was used the shopping-cart-flask activity as it contained full shopping cart and product list features, minor changes have been done to these to convert that into book shop. The other features register, login, stock, adding new stock, were developed according to the lab tasks as to only take the idea and think how I could implement that and what should I change and write new. Tried to use week 7 lab task to commit into git repository as soon as came into an issue, as not much time is left decided to cross this out, and upload files manually. Week 6 lab task was used for the checkout.

Minor clarifying references are located all across the various code lines, website of which was used for displaying the value across the html range sliders is here : <https://stackoverflow.com/questions/10004723/html5-input-type-range-show-range-value>

Testing regime template from here:

<https://files.coventry.aula.education/113142622c8395b0cc33af07f97ac8eb5001cem_2122_test_regime_template.docx>

Books and images taken from here:

<https://www.penguinrandomhouse.com/books/all-best-sellers>

Also special references

Gurvinder Student id: 10248274

Helped on common issues when running the project and reminding of that on every new codio project pip commands must be ran again as it is a virtual environment

Luke Student id: 10107753

Helped with flask explaining url\_for() function in detail, additional references are located in code.