

School of Computing, Engineering and Mathematics (CEM)

Faculty of Engineering, Environment and Computing (EEC)

**5001CEM SOFTWARE ENGINEERING** | 2122

**PROJECT REPORT**

**NAME:** Maninder singh

**SID: 10340383**

1. **CODE PURPOSE**

Web application for an online company selling books

1. **CODE LOCATION**

<https://github.coventry.ac.uk/5001CEM-2122/Book_shop-maninder>

1. **CODE INSTALLATION**
2. Download code from <https://github.coventry.ac.uk/5001CEM-2122/Book_shop-maninder>
3. Unzip and set up a new python project in Codio, called book\_shop. Then follow the instructions below, adapting the filenames.

Create a new project directory (folder) e.g. bookshop by doing mkdir bookshop

Upload unzip files to this folder

cd into your directory and do **install python3-venv**

Then do **python3 -m venv venv**

Then activate the virtual environment: **. venv/bin/activate**

This is confirmed by the change to the prompt, which is now

**(venv) codio@emotion-theory:~/workspace/bookshop$ (your box will have a different name).**

Then do : **pip install Flask**

We need a database, so also do : **pip install Flask-SQLAlchemy**

Do **export FLASK\_APP=book\_shop**

Do **flask run –host=0.0.0.0** and select Box URL under project index if necessary

Change the port number from 3000 to 5000. Then you should see the application

1. **CODE EXPLANATION**

**Design and Features**

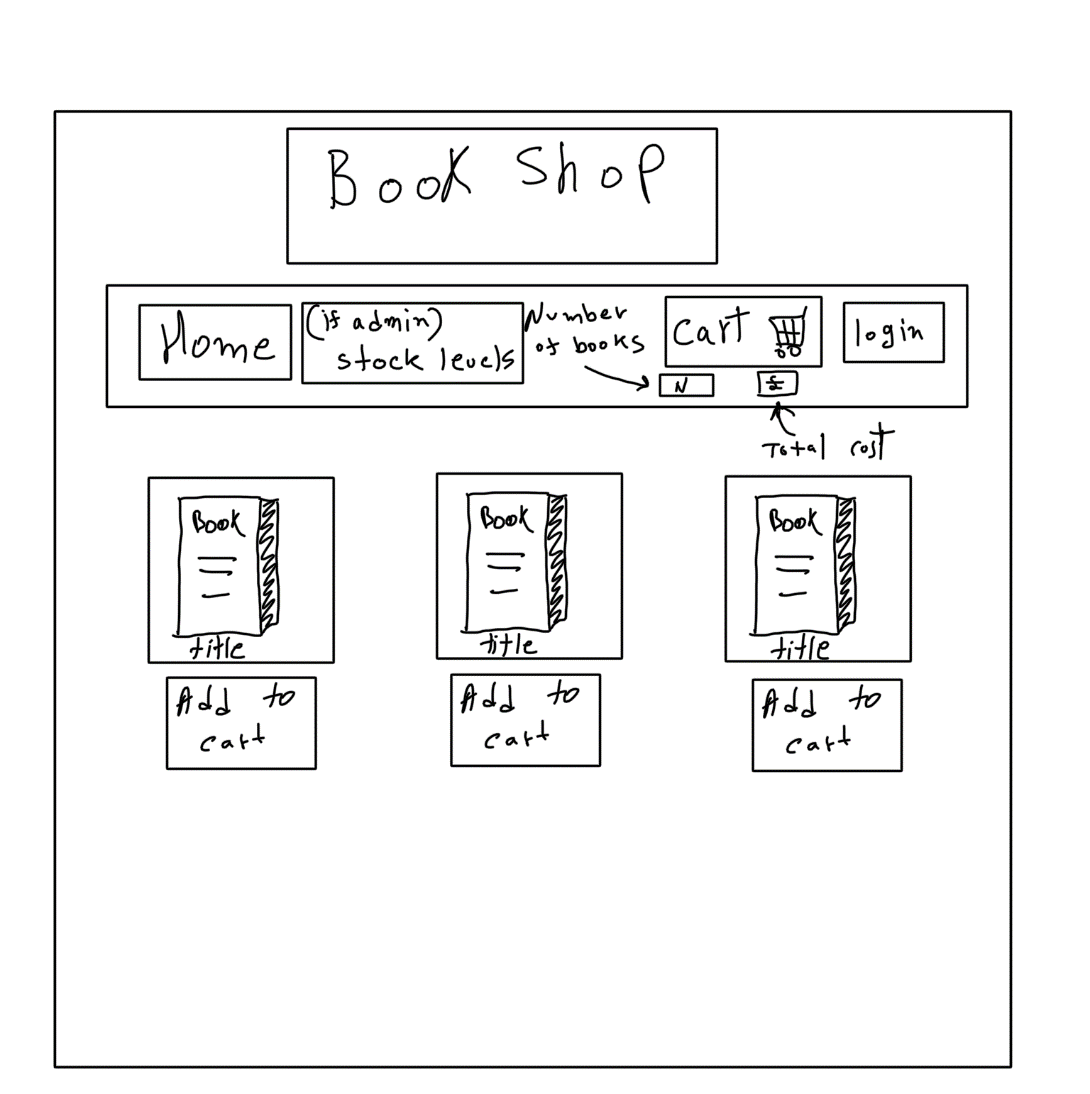
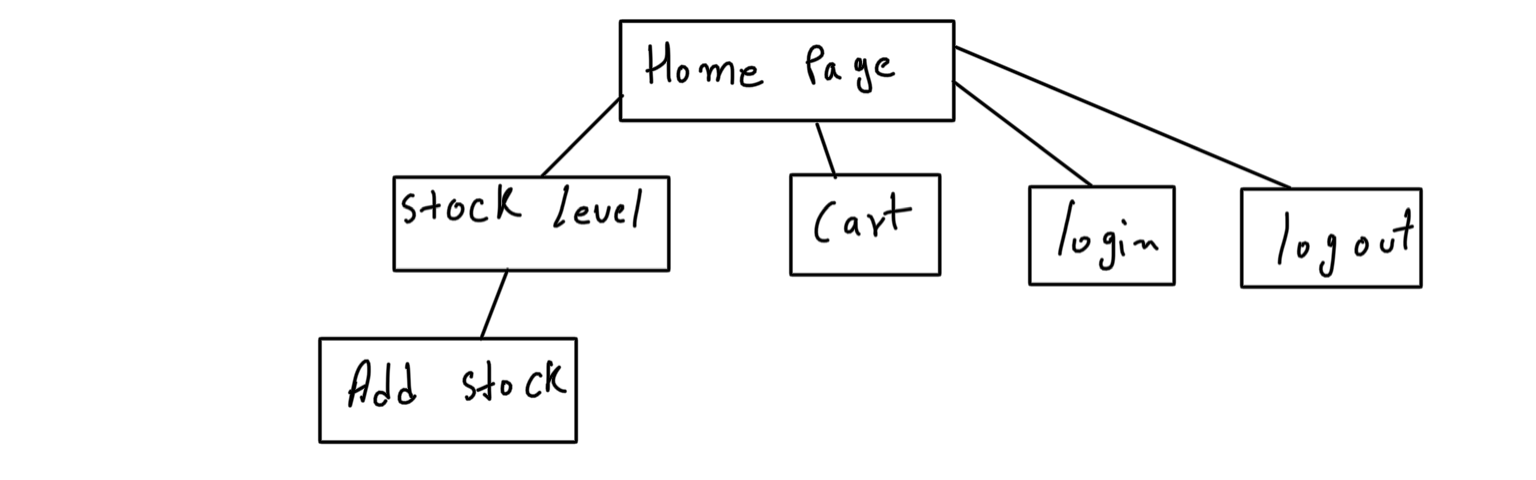
The Web application is an online book shop and tool for online company selling book. The design of the web application is made considering key features that are needed to build online book shop. The features are divided between admin and customer:

For admin key feature include: admin needs to be able to see stock level of products and to be able to add more books through add stock feature. These admin tools need to be restricted so that only admin has access to them.

For customer key features include: to be able login, have access to available books that user can buy and add them to their cart, once item has been added, there should be an information on number of books added to cart as well as total price of books. User also should be able to see book’s title, thumbnail image, and price at home page.

Flask which is a micro web framework written in Python, HTML5, and CSS will be used to develop the web application.

Simple rough design of web application is shown here:



Rough design of how pages will be linked together:

**Implementation**

**Home**

Base page is created in order to reduce duplicate code which will be inherited by all other pages.

*This code is inspired from-* [*https://www.youtube.com/watch?v=4nzI4RKwb5I&list=PLmQ3sIvZKC52f4hriHG1ck98gN2iMxcQE&index=9*](https://www.youtube.com/watch?v=4nzI4RKwb5I&list=PLmQ3sIvZKC52f4hriHG1ck98gN2iMxcQE&index=9)

{% extends "base.html" %}

Is used in every page to inherit the properties of *base.html* file

Home function is responsible for displaying books that are available in stock as well as displaying who has logged in, total number of books added in cart, and total price.

# home function displays books, user information, total cart items and total price

@app.route('/')

def home():

    conn = sqlite3.connect('book\_shop.sqlite3') # connect to the database

    cur = conn.cursor()# set cursor

    cur.execute(f"SELECT book\_name,book\_picture, book\_retail\_price,\

              book\_ISBN FROM stock ;")# SQLite Query to get  all the books

    book\_data = cur.fetchall() # fetch all the data

    conn.commit() # commit SQLite query

    conn.close() # close SQLite

    if "user" in session: # if user in session

        user = session["user"] #set a variable to session

        user\_info = "Logged in as: " # set variable to string

    else: # if user is not in session, set variables empty

        user\_info = ""

        user = ""

    if "items" in session: # if items in session

        items = session["items"] # set variable to item in session

        price = session["price"] # set variabel to price in sesion

    else: # if not , set them to empty

        items = ""

        price = ""

    return render\_template('home.html', books=book\_data,\

         user\_info=user\_info,user=user,

         total\_items=items, total=price ) # Render home.html and pass books data, user, items and price

*This is the end of code inspired from-* [*https://www.youtube.com/watch?v=4nzI4RKwb5I&list=PLmQ3sIvZKC52f4hriHG1ck98gN2iMxcQE&index=9*](https://www.youtube.com/watch?v=4nzI4RKwb5I&list=PLmQ3sIvZKC52f4hriHG1ck98gN2iMxcQE&index=9)

*@app.route('/')* It tells the system which URL to map to after this *home()* function is called which connects to database *conn = sqlite3.connect('book\_shop.sqlite3')*. *cur* variable is used to execute SQLite queries which gets information on all the books available in stock. *SELECT book\_name,book\_picture, book\_retail\_price, book\_ISBN FROM stock* SQLite query is used to information about *book\_name, book\_picture,* and *book\_retail\_price* which is retrieved from *stock* table from database. Then *book\_data* variable is used to store all the data that has been fetched from database using *cur.fetchall()* then *conn.commit()* and *conn.close()* are usedfor committing the changes and closing the database.

*“user”* is checked in the session if available then *user = session[“user”]* variable is used to store username, if *“user”* is not in session then variable is set to empty *user = "".* *user\_info* variable is used to store string which would only show if the user is in the session otherwise it won’t show.

After checking whether the user is in the session, system then checks if *“items”* is in the session if *‘items’* is in session then *item* variable is used to store data that is in ‘*item*’ session, and *price* variable is used to store data that is in *‘price’* session. if ‘*item*’ is not in session then *item* and *price* variable are set to empty *items = "", price=””. return* is then used to render *‘home.html’* and all the variables stored are assigned to the appropriate expressions so the data can be passed to the *html* file.

<div class="books">

    {% for book in books %}

        <ul>

            <img src="/static/{{ book[1] }}" alt="display this" width=190.5 height=292.25><br>

            {{ book[0] }}<br>

            Retail Price: £{{ book[2] }}

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

                <button id="item" name="item" type="submit" value={{book[3]}}> Add to cart </button>

            </form><br>

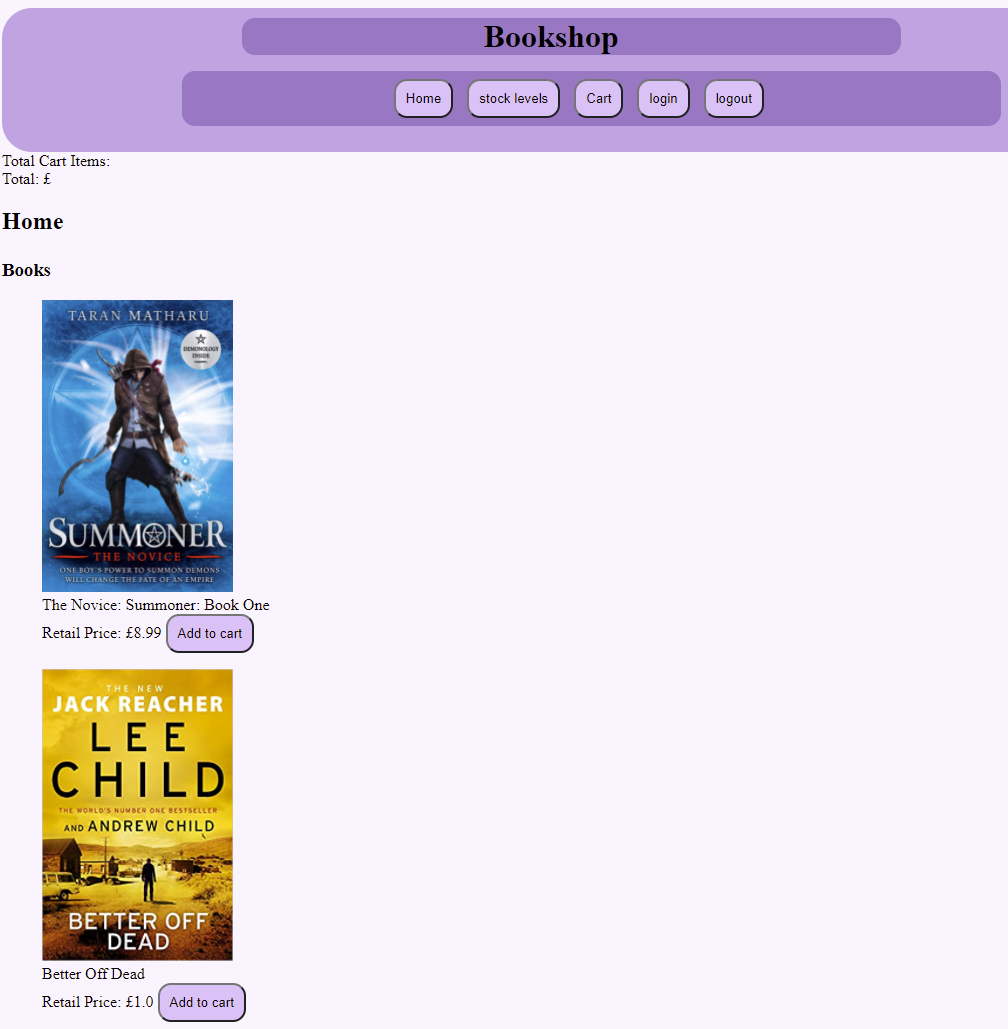
        </ul>

    {% endfor %}

</div>

In *home.html* file a for loop is created which will go through every book that has been fetched from database as well as getting necessary information needed to display book’s title, thumbnail, and price. *<img>* tag is to show thumbnail image of the book and *<form>* tag is used send information about which book the user wishes to add it to cart. <*button*> tag is used for interface so that user would only have to click to be able to add book in their cart.

An image of working home page is shown below:



**Login**

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

def login():#login function displays login form and checks if user is in session

    if request.method == 'POST': #if form is submitted

        return do\_login(request.form['uname'], request.form['pwd'])# send user info to do\_login

    else:

        if "user" in session: # check if user in session

            flash("Already logged in", "info")# message if user is already logged in

            return redirect(url\_for('home')) # rediret to home page

        return show\_login\_form()# if not show login form

def show\_login\_form(): # show login form

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

*‘GET’* and *‘POST’* methods are used to get data from user. After calling *login()* function *‘Post’* request is checked if user has given the system any data, if there is data given then *do\_login()* function is returned with *‘uname’* as user name and ‘*pwd*’ as password. If it is not ‘*POST’* request then system checks if ‘*user*’ is already in session if there is *user* in session then message is shown about user is has already logged in and user is *redirected* to home page otherwise if *user* is not in session *show\_login\_form()* function is returned where *‘login.html’* template is rendered whose data will be sent to the *login()* function by *login\_form* expression.

    <form action="{{login\_form}}" method="post">

        <label for="uname">Username:</label><br>

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

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

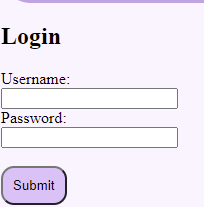
        <input id="pwd" name="pwd" value="" type="text"><br><br>

        <button value="Submit" type="submit">Submit</button>

    </form>

*login\_form* expression is used to determine where the form data will be sent. *<lable>* tags are used to show which kind of information is requested from user and *<input>* tags are used to get that information. *Submit* type button is used to send all the data for authentication.

Working *login* form is shown below



def do\_login(username, password):

    conn = sqlite3.connect('book\_shop.sqlite3')

    cur = conn.cursor()

    cur.execute(f"SELECT user\_password FROM user WHERE user\_name = '{username}';")

    password\_data = cur.fetchone()

    conn.commit()

    cur.execute(f"SELECT admin FROM user WHERE user\_name ='{username}';")

    admin\_data = cur.fetchone()

    conn.commit()

    conn.close()

    if password\_data is None or admin\_data is None: # if password data or admin\_data is None

        flash("username or password incorrect ", "info")#shows message: username or password incorect

        return redirect(url\_for('login')) # redirect to login function

    if password == str(password\_data[0]): # if given password is equal to password stored in database

        session['user'] = request.form['uname'] # set user in session

        if str(admin\_data[0]) =='1': # if admin data is True

            session['admin'] = str(admin\_data[0]) #set admin session

            flash("You have been logged in as Admin", "info") # message about successful login as admin

            return redirect(url\_for('home')) # redirect to home

        flash("You have been logged in", "info") # message about successful login as user

        return redirect(url\_for('home'))

    flash("username or password incorrect ", "info")#show message: username or password incorect

    return redirect(url\_for('login'))

*do\_login()* function is for authentication and *username* and *password* attributes are used to pass user information from here user’s password and admin data is fetched from the database. *SELECT user\_password FROM user WHERE user\_name = '{username}';")* query is used to get password and *WHERE* statement is used to specify what data the system is looking for, and that is assigned to *username* attribute. This data is stored in *password\_data* variable after this changes are committed with *conn.commmit().*

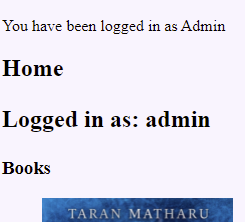
Next query is executed to get admin data which would determine whether user has administrative role or not this is done by *SELECT admin FROM user WHERE user\_name ='{username}* query where *{username}* variable is passed in query to find admin data stored in database and all the data that has been fetched from this query will be stored in *admin\_data* variable*,* after this changes are committed with *conn.commmit()* and database is closed with *conn.close().*

Then the fetched data is checked whether if its None or not, if the data is *None a* message is shown about username or password is incorrect and user is *redirected* to *login* function.

Working *login* form is shown below when authentication fails.



otherwise *password* attribute is compared against data fetched from data base and is stored in *session[‘user’]* then admin data is checked if it True(1) then the data is stored in *session[‘admin’],* user is then redirected to home page with massage *flash* stating that user has been logged in. Working *login* form is shown below when authentication is successful.



@app.route('/logout')

#logout function clears user and admin session

def logout():

    session.pop('user', None) # set user sesson to None

    session.pop('admin', None) # set admin sesson to NOne

    flash("You have been logged out","info") # message about user have been logged out

    return redirect(url\_for('home'))

*logout()* function removes session data and shows message to user then user is *redirected* to home page. *session.pop('user', None)* is used to remove any data that might be stored in *‘user’* session and is set to *None. session.pop('admin', None)* is used to remove any data that might be stored in *‘admin’* session and is set to *None.*

*This code is inspired from-* [*https://www.youtube.com/watch?v=DQRSvg54bhM*](https://www.youtube.com/watch?v=DQRSvg54bhM)

path= os.getcwd() # get path of current directory

filename = 'static' # set folder

file\_path = os.path.join(path,filename) # join the path to the folder

*This end of code is inspired from-* [*https://www.youtube.com/watch?v=DQRSvg54bhM*](https://www.youtube.com/watch?v=DQRSvg54bhM)

*os.getcwd()* which is set to variable *path* is used get the path of current directory and the path is joined to the *‘static’* folder where pictures from form will be stored. The path is joined by using *os.path.join(path,filename)* which is set to variable *file\_path* so that system can use this path later if needed.

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

@admin\_login\_required # restrict access to add stock

#add\_stock function saves book info from form

def add\_stock():

    if request.method == 'POST':

        book\_name = request.form['book\_name'] # get form information

        book\_author = request.form['book\_author']

        book\_date = request.form['book\_date']

        book\_ISBN = request.form['book\_ISBN']

        book\_description = request.form['book\_description']

        book\_picture = request.files['book\_picture']

        book\_trade\_price = request.form['book\_trade\_price']

        book\_retail\_price = request.form['book\_retail\_price']

        book\_quantity = request.form['book\_quantity']

        try:

            conn = sqlite3.connect('book\_shop.sqlite3')

            cur = conn.cursor()

            cur.execute(f"SELECT book\_quantity FROM stock WHERE book\_ISBN ='{book\_ISBN}' ;")

            quantity\_data = cur.fetchone()

            if quantity\_data is None: # checks if there is book in stock already

                if book\_picture.filename != '': # checks image is not submitted emply

                    book\_picture.save(os.path.join(file\_path, book\_picture.filename)) # saves image

                    cur.execute(f"INSERT INTO stock VALUES ('{book\_name}',\

                                                        '{book\_author}',\

                                                        '{book\_date}',\

                                                        '{book\_ISBN}',\

                                                        '{book\_description}',\

                                                        '{str(book\_picture.filename)}',\

                                                        {book\_trade\_price},\

                                                        {book\_retail\_price},\

                                                        {book\_quantity});") #add book in stock

                    conn.commit()

                    conn.close()

                    flash("Book has been added to stock", "info")  # message about book has been saved

                    return add\_book\_form()

            else:

                # updates book quantity if ISBN number is same

                new\_quantity\_data = quantity\_data[0] + int(book\_quantity) # increase quantity of book in stock

cur.execute(f"UPDATE stock SET book\_quantity = {new\_quantity\_data} \

                             WHERE book\_ISBN = '{book\_ISBN}' ;") # updates stock

                conn.commit()

                flash("Book has been added to stock", "info")  # message about book has been saved

                return add\_book\_form()

        # add message about book has been saved successfully

        except Exception as error:

            flash("Error", "info")# give Error

            print(error)

            return add\_book\_form()

    else:

        return add\_book\_form()

*@admin\_login\_required* is to restrict the user accessing *add\_stock()* unless they are admin. Information about where user is admin or not is checked from ‘*admin*’ data that is stored in session. ‘*POST*’ request is used to check if user has sent any data otherwise *add\_book\_form()* function is returned.

All the information about the book that has been received, is assigned to appropriate variables, the book\_ISBN number is fetched from the database by *SELECT book\_quantity FROM stock WHERE book\_ISBN ='{book\_ISBN}'* query, this data will be saved in *quantity\_data* variable and is check if the data that has been stored in variable *quantity\_data* is *None* otherwise book quantity is increased because of same book has already been stored in database, it is to reduce duplicates in database.

New quantity of the book is stored in *new\_quantity\_data* variable where it adds old quantity *quantity-data[0]* to new quantity *int(book\_quantity)* which is received from variable that stored number received from *request.form['book\_quantity'].* To update current quantity of book *cur.execute(f"UPDATE stock SET book\_quantity = {new\_quantity\_data} WHERE book\_ISBN = '{book\_ISBN}' ;")* query is executed *new\_quantity\_data* variable is passed and to where the new data needs to be added *book\_ISBN* number is used to locate book. After this changes are committed with *conn.commmit()*  and message is shown to the use that book has been added, then *add\_book\_form()* is returned if user would like to add another book.

If fetched data is *None* which would mean it is a new book and should be stored in database. *book\_picture.filename* is checked to make sure that user does not submit the form without book thumbnail image then *book\_picture.save(os.path.join(file\_path, book\_picture.filename))* is used to save book thumbnail image in *static* folder, *book\_picture.save* is to store file in given path, in this case *os.path.join(file\_path, book\_picture.filename))* is used to join path where we want the file to be stored. *cur.execute(f"INSERT INTO stock VALUES* query is executed to store all other information in database using *INSERT* statement, after this changes are committed with *conn.commmit()* and database is closed with *conn.close().* Message is shown that book has been added to stock and *add\_book\_form* function is returned is user wishes to add another book in stocks.

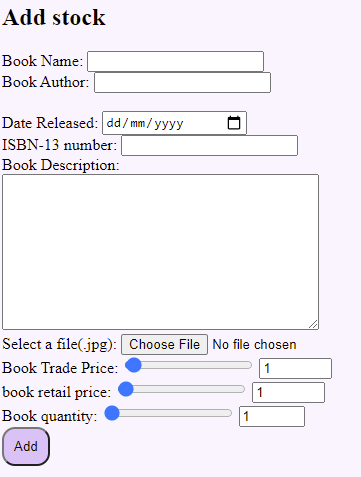
If any error is accrued during this process *except Exception as error* is used and message is shown to user that process has been failed then *add\_book\_form* function is returned if user want to try again.

def add\_book\_form():

    return render\_template('add\_stock.html', add\_book\_form=url\_for('add\_stock'))

*add\_book­\_form()* renders *add\_atock.html* file and any information sent to the form will be passed to the *‘add\_stock’* function.

Working *add\_stock\_form* is shown below



**Cart**

order\_list = []

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

def add\_to\_cart():

    if request.method == 'POST':

        conn = sqlite3.connect('book\_shop.sqlite3')

        cur = conn.cursor()

        item = request.form['item']

        int\_item = int(item)

        order\_list.append(int\_item) # add book in order list

        total\_price = 0

        for list in order\_list:

cur.execute(f"SELECT book\_retail\_price FROM stock where book\_ISBN = '{list}';") # get book price from database

            book\_data = cur.fetchone()

            book\_data = book\_data[0]

            total\_price = total\_price + book\_data # add total price of the books

        conn.commit()

        conn.close()

        session['items'] = len(order\_list) # get total numbers of books added

        session['price'] = round(total\_price, 2) #get total price

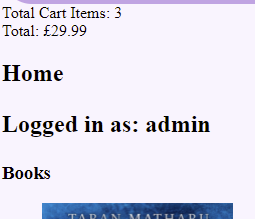
        return redirect(url\_for('home'))

*add\_to\_cart* function is called to add books in *order\_list* list, the *‘POST’* request is used to check whether user has requested to have book to be added in cart, then SQLite is used to connect to *'book\_shop.sqlite3'* database. Information from form *request.form['item']* is stored in *item* variable which is then converted to integer using *int(item). order\_list.append(int\_item)* is used to add book ISBN number in list.

*total\_price* variable is set to 0 so that system can add price to multiple books, after this SQLite query is run in order to get the price for all the books that have been added in *order\_list,* for that *cur.execute(f"SELECT book\_retail\_price FROM stock where book\_ISBN = '{list}';")* query is executed in for loop to get price for all the books in *order\_list.* Fetched data is stored in *book\_data* which is then converted to *book\_data[0]* to get price from list, then *book\_data* is added to the *total\_price.* After this changes are committed with *conn.commmit()* and database is closed with *conn.close().*

*len(order\_list)* is used in order to get the total numbers of books that are in *order\_list* which is then total number is stored in *session[‘item’]* and *round(total\_price, 2)* is then used to round *total\_price* to 2 decimal places to get total price of the books which is stored in *session[‘price’],* after this the user is redirected to hope page .

*add\_to\_cart* working is shown below.



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

#cart function saves books added in cart

def cart():

    cart\_list = {} #saves books in dictionary

    for item in order\_list:

        quantity= 0

        for book in order\_list:

            if book == item: # check for duplicate items in order list

                quantity += 1 # increase quantity if these is duplicate

cart\_list[item] = quantity # save book with quantity

    cart\_book = []

    for books in cart\_list.keys(): # to get book ISBN number

        cart\_book.append(books)

    book\_data = []

    for book in cart\_book: # get book info from database

        conn = sqlite3.connect('book\_shop.sqlite3')

        cur = conn.cursor()

        cur.execute(f"SELECT book\_name, book\_picture, book\_retail\_price FROM stock WHERE  book\_ISBN = {book} ;")

        book\_list = cur.fetchone()

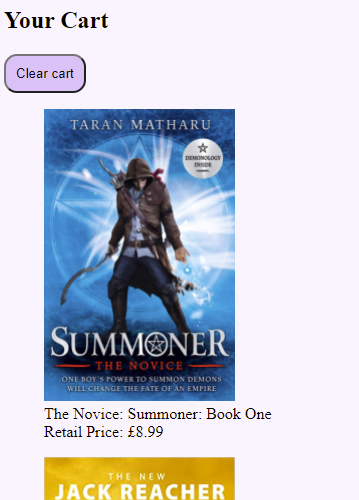
        book\_data.append(book\_list) #saves book information in book\_data

    return render\_template('cart.html', cart\_books = book\_data)

*cart()* function is called when user wishes to see which books has been added to their cart. *cart\_list* dictionary is created to store quantity of each product and to reduce duplicates. Two For loops are created in order to compare items to each other so that if there are same more than one items stored in *order\_list* their quantity is increased rather than having duplicates, items are stored in dictionary as *cart\_list[item] = quantity.*

After removing duplicates from cart another For loop is created in order to convert dictionary keys *cart\_list.keys()* to list *cart\_book.append(books)* this is done to pass book information in SQLite query. All the data etched from database is stored in *book\_data* list and *‘cart.html’* template is rendered to show books that have been added to cart.

*Cart()* function working is shown below.



@app.route('/empty\_cart', methods=['GET', 'POST'])

def empty\_cart():

    order\_list.clear() # empty cart

    session.pop('items', None) # clear session data

    session.pop('price', None) # clear session data

    flash("Your cart has been Cleared")

    return redirect(url\_for('home'))

*empty\_cart()* function is used to remove all the items that have been added to cart using *order\_list.clear(), session.pop('items', None)* and *session.pop('price', None)* is to remove any data stored in session, after this the user is redirected to *‘home’* page and message is shown to the user that their cart have been cleared.

1. **TESTING**

.1 TESTING REGIME

Test regime is provided on different file

.2 TESTS RUN ON EXTERNAL CODE

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

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

1. **QUALITY ASSURANCE**

.1 QUALITY ASSURANCE STATEMENT

**Brief and requirements delivered on**

The project is to create online tool for an online company selling books. The web application should let user order books and it should also have administrative tools which would allow admin to add stock.

The features that have been implemented are:

Stock levels and add stock which will be restricted to admin, no customer will be allowed to access these pages. In stock level page user will be able to see all the books that are in stock as well as book’s thumbnail picture, the book title, the ISBN-13 number and the quantity in stock. Add stock feature will let admin add books to stock

Home page will let user see all the books that are in stock and add to cart option next to every book which would add that book to cart and user will see total price and number of items in cart.

Cart feature would let user see all the book that have been added to cart with picture and price of each book.

Login feature would user login and if user has input wrong username or password the user will be shown message about it, this function will also check if the user is admin which would determine if user will be restricted from add stock feature.

Feature that has not been implemented:

Check out feature have not been implemented which would have let user taken to complete order screen that will display final order details.

**Coding standards**

The code is commented but layout of comments might not be clear for some viewers and code is modular whose elements can be reuse for future use. The code is made readable by using appropriate variable names.

**Documentation standards**

In documentation include rough diagrams on what the web application would look and detail explanation on critical components of web application.

**Team details and support**

Team details

Team members- [singhm50@uni.coventry.ac.uk](mailto:singhm50@uni.coventry.ac.uk), [aldulaimyf@uni.coventry.ac.uk](mailto:aldulaimyf@uni.coventry.ac.uk), [headl@uni.coventry.ac.uk](mailto:headl@uni.coventry.ac.uk)

Each team member will be doing all three roles of one other team member because there are only three of us.

**Timely delivery**

I was not very late on code delivery, but I was not able to deliver on time because of last minute problem found which affected critical component of web application.

.2 EXTERNAL QA EVALUATION

// Your evaluation of the external’s QA statement

1. **DOCUMENTATION**

// around 500 words

.1 DOCUMENTATION LIST

* Commented code at - <https://github.coventry.ac.uk/5001CEM-2122/Book_shop-maninder>
* Statement of code purpose
* Statement of code location
* Install instructions
* Code explanation
* Testing Regime
* QA Statement

.2 EXTERNAL DOCUMENTATION INSPECTION

// Does all documentation appear? Is it usable – can you install and run? Do the tests make sense and can you run them?

// Do you understand the code explanation? Is there anything missing? Are there improvements you’d suggest?

**REFERENCES**

// do not count towards word limit

// You must reference any sites consulted and code used in your work. Code re-use is fine and a common practice (one reason why documentation is so important). Lifting entire code blocks including complete applications without attribution is an academic conduct offence and this has consequences.

// I’ll be saying more about how to reference code.