

School of Computing, Engineering and Mathematics (CEM)

Faculty of Engineering, Environment and Computing (EEC)

**5001CEM SOFTWARE ENGINEERING** | 2122

**PROJECT REPORT**

**NAME:** Helena Torrinha

**SID:** 10613704

Table of contents

[1 - CODE PUROPOSE 4](#_Toc89013358)

[2 - CODE LOCATION 4](#_Toc89013359)

[3 - CODE INSTALATION 4](#_Toc89013360)

[4 - CODE EXPLANATION 7](#_Toc89013361)

[5 - TESTING 30](#_Toc89013362)

[6 - QUALITY ASSURANCE 34](#_Toc89013363)

[7 - DOCUMENTATION 35](#_Toc89013364)

[8 - REFERENCES 35](#_Toc89013365)

Table of Figures

[Figure 1: what the terminal should look like with the venv activated 4](#_Toc89013366)

[Figure 2: how the terminal looks like with the codes to run the project 5](#_Toc89013367)

[Figure 3: how to find the "Box Info" option 5](#_Toc89013368)

[Figure 4: where is the URL to see the project working 6](#_Toc89013369)

[Figure 5: Project diagram 8](#_Toc89013370)

[Figure 6: Log in page with the success message 9](#_Toc89013371)

[Figure 7: Log in page with an error message 9](#_Toc89013372)

[Figure 8: Register page with the error message when the user already exists 11](#_Toc89013373)

[Figure 9: Example of how the Shopping Cart looks like 14](#_Toc89013374)

[Figure 10: How the Add Stock page form looks like 17](#_Toc89013375)

[Figure 11: The Add Stock page form with the error message 18](#_Toc89013376)

[Figure 12: Homepage for customers with the error message 21](#_Toc89013377)

[Figure 13: Example of Checkout page with 1 book 24](#_Toc89013378)

[Figure 14: Example of Checkout page with more than 1 book 24](#_Toc89013379)

[Figure 15: Example of Checkout page where there is an error (there is just 1 "The Hunger Games" book left) 25](#_Toc89013380)

[Figure 16: Error message in the Homepage when the books in the cart of the customer are no longer available 26](#_Toc89013381)

## 1 - CODE PUROPOSE

This code aims to simulate a book shop where the customers can see the available books and buy them and administrators can change their quantity and add new ones// NB this is a comment for guidance. Delete all comments when you complete this template.

## 2 - CODE LOCATION

<https://github.coventry.ac.uk/5001CEM-2122/BookShop5001CEM>

## 3 - CODE INSTALATION

From the GitHub repository download the zip with the files and unzip it to the destination folder.

In the terminal do **sudo apt update**

Do cd into the directory where the project is and do **sudo apt-get install python3-venv**

Then do **python3 -m venv venv**

Then activate the venv virtual environment by doing **. venv/bin/activate**

You should get something like this but with a different box name:

**Nenhuma entrada de índice de ilustrações foi encontrada.**

Figure 1: what the terminal should look like with the venv activated

Then do **pip install Flask**

And do **pip install Flask-SQLALchemy**

And finally do **pip install flask-cors**

And the project is setup so now, every time you want to run the project you just need to **cd** into the directory where the project is, do **. venv/bin/activate** (if you just setup the project, you don’t need to do these two steps), export the file you want to run (in this case main.py) by doing **export FLASK\_APP=main.py** and then run it with **flask run --host=0.0.0.0**

Texto, Linha do tempo

Descrição gerada automaticamente

Figure 2: how the terminal looks like with the codes to run the project

To see the website running check your project box info and change the URL to port 5000.

Tela de computador

Descrição gerada automaticamente

Figure 3: how to find the "Box Info" option

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Figure 4: where is the URL to see the project working

So, in my case the URL will be **https://micro-corner-5000.codio-box.uk**.

## 4 - CODE EXPLANATION

Diagrama

Descrição gerada automaticamente

Figure 5: Project diagram



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

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

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

type = do\_the\_login(request.form['username'], request.form['password']) #check if the user exists and if so returns the type

**if** type: #if user exists

session['username'] = request.form['username'] #saves the username used in the login in a session

session['type'] = type #saves the type of user according to the account that did the login in a session

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

**else**:

**return** render\_template('login.html',page=url\_for('login'), error='Wrong username or password.') #the username or password are not correct so it displays an error message in the login page

**else**: #if it's method 'GET'

**if** 'success' **in** session: #If the user registered successfully and was redirected to the login page

success = session["success"] #there will be a session with a success message

session.pop("success") #deletes the session to avoid repeting the success message

**else**:

success=""

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

This function will be opened in method ‘GET’ when a user enters the /login route or when an account is created (the /register redirects to the /login route) and in this case, if the user registered successfully before, the login page will also show a success message.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Figure 6: Log in page with the success message

The login function will be opened in method ‘POST’ when the user submits his info (username and password). It will verify if the user exists in the database by calling the do\_the\_login() function from the databasemanager.py file. If the user exists, the function will return the type of that user or, if the user doesn’t exist it will return False and that value will be saved in the variable type. Then, if the user exists it will set the session['username']as the username the user wrote in the login form and the session['type'] as the value stored in type. After that, the user will be redirected to the /homepage route. If the user doesn’t exist, an error message ('Wrong username or password.') will be displayed in the login page.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Figure 7: Log in page with an error message

**def** **do\_the\_login**(u,p): #function to verify if a user is in the db

con = sqlite3.connect('bookshop.db') #connects with the db

cur = con.cursor()

cur.execute("SELECT count(\*), type FROM users WHERE username=? AND password=?;", (u, password\_encode(p))) #query to check if the account exists and that, in that case, returns the type of account

results = cur.fetchone() #gets the first row

**if**(int(results[**0**]))>**0**: #if the user exists

**return** results[**1**] #returns the type of the user

**else**: #the user doesn't exist

**return** **False**

As said before, the **do\_the\_login**() function verifies if a user is in the database and if the user exists it returns its type. In order to check this, we need to connect with the database (bookshop.db) and execute a query that verifies if the username and the password (that the user that is trying to log in submitted) exists. Since the passwords in the database are encoded, we need to encode the user’s password to compare them. That query will return 1 and the user type if the user exists, otherwise it will return 0. These values will be kept in an array (results).

If the value returned by the query is bigger that 0 (in our case it will be 1), the function will return the second element of results, that is the user type. Otherwise, the function will return False.

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

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

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

**if** add\_accounts(request.form['username'], request.form['password']): #checks if the username already exists and adds the account to the db if it doesn', using the add\_accounts function

session["success"] = "Account added." #message that will be displayed in the login page

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

**else**: #if the username already exists

**return** render\_template('register.html',page=url\_for('register'), error='Invalid account.') #shows an error message

**else**:

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

This function will be opened in method ‘GET’ when a user enters the /register route or when a user wants to create an account (the /login redirects to the /register page).

The register function will be opened in method ‘POST’ when the user submits his info (username and password). It will verify if the username already exists in the database by calling the add\_accounts() function from the databasemanager.py file. If the username exists, the function will return False. Otherwise, it will add the username and the password (that the user wrote in the register form) and the predefined type (customer) into de users table of the database and then return True. So, if the username doesn’t already exist, a success message ("Account added.") will be kept in the success session to be shown when the user is redirected to the login page (see Figure 6).

If the username exists, there will be shown an error message ('Invalid account.') on the Register page.

Interface gráfica do usuário, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

Figure 8: Register page with the error message when the user already exists

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

**@login\_required**

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

rows = display\_books\_homepage() #uses a function to get the books information

**if** 'error' **in** session: #if a function sends a session error to the homepage

error = session["error"]

session.pop("error") #deletes the error

**else**:

error=""

**return** render\_template('homepage.html', books = rows, type = session['type'], error=error, isShoppingCart=**True**)

The **homepage**() function is used for the /homepage route that requires the user to login to be able to access it (calls the **login\_required** function to check that). It calls the display\_books\_homepage() function from the databasemanager.py to get the books’ information from the database and saves them in an array called rows.

If some route redirects to the homepage and sends an error session, that error will be displayed and then the session that keeps it is deleted (to show the error only once).

**def** **display\_books\_homepage**(): #function that returns the books for the Homepage

**try**:

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

#code based on https://www.sqlitetutorial.net/sqlite-python/sqlite-python-select/

cur = con.cursor()

cur.execute("SELECT name, picture, isbn, retail\_price, quantity, description FROM books WHERE quantity>0") #gets the name, picture, isbn, retail\_price, quantity, description that have positive quantity

rows = cur.fetchall() #gets all rows

**return** rows

**except** **Exception**:

traceback.print\_exc()

As referred before, the **display\_books\_homepage**() function returns the books details from the database. To do that, we need to execute a query that gets the name, picture, isbn, retail\_price, quantity, description for the books that have positive quantity (we want to show only the books that have enough quantity to be bought) from the books table. We save what that query returns in an array called rows and then the function returns it.

<!-- Modal body -->

<div class="modal-body">

{% for row in session["cart\_books"] %} <!--go through all the cart books-->

<div class="d-flex align-items-center">

{% for line in books %} <!--go through all the books in the homepage-->

{% if row==line[2] %} <!--if the isbn’s are the same -->

<img class="me-1 mb-2" src="static/uploads/{{line[1]}}" alt="Book Picture" width="100" height="160">

<div>

<p>{{line[0]}}</p> <!--Name of the book-->

<p>{{session["cart\_books"][row][1]}}£</p> <!--Book's price-->

<p>Quantity: {{session["cart\_books"][row][0]}}</p> <!--Book's quantity in the cart-->

</div>

<a class="ms-auto" href="/delete\_book\_shopping\_cart/{{row}}" >

<img src="static/images/delete\_icon.png" alt="Delete icon" width="30" height="30">

</a>

{% endif %}

{% endfor %}

</div>

{% endfor %}

</div>

This html code belongs to the header.html file. It’s a part of the modal that contains the shopping cart. In order to make the cart work properly it needs to go through all the books in the cart books session, compare them with the one in the homepage and when the isbn’s are the same it will display the book’s picture, name, price and quantity in the cart. The shopping cart offers the option to remove a single book from the shopping cart through a button (this works because due to the **delete\_book\_shopping\_cart**()function).

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Figure 9: Example of how the Shopping Cart looks like

**def** **delete\_book\_stock\_level**(isbn, upload\_folder): #fuction to allow admins to delete books from the db

**try**:

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

cur = con.cursor()

cur.execute("SELECT picture FROM books WHERE isbn=?;", (isbn,))

picture = cur.fetchone()[**0**] #gets the picture's path

cur.execute("DELETE FROM books WHERE isbn=?",(isbn,)) #deletes the book

con.commit()

con.close()

remove\_picture(picture, upload\_folder) #deletes the picture

**except** **Exception**:

traceback.print\_exc()

This function is from the databasemanager.py and its called by the **delete\_book\_stock**() in the /stock\_level route and its aim is to delete a book from the database and also remove its picture from the folder where it is saved. So basically the function does a query to get the path for the picture corresponding to the book’s isbn we are deleting and saves it into the picture variable, then it does a query to delete the book in the database with the sent isbn. In the end, it calls the function remove\_picture() from the processing.py to delete the picture using the path to reach it.

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

**@login\_required**

**def** **add\_stock**():

**if** session["type"] == "customer": #only an admin is able to see this route so if a user tries to access it he will be redirected to the homepage

**return** redirect(url\_for('homepage'))

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

picture = upload\_file(request.files, app.config['UPLOAD\_FOLDER']) #stores the picture and retuns its name

data = dict(request.form) #stores the data from the form

#code inspired in https://stackoverflow.com/questions/40414526/how-to-read-multipart-form-data-in-flask

result = add\_books(data['isbn'], data['name'], data['author'], data['date'], data['description'], picture, data['quantityRange'], data['retailRange'], data['tradeRange']) #adds the book with the form's values

**if** result == **True**: #if all the fields are correct the user will go to the Homepage

**return** redirect(url\_for('homepage'))

**else**:

**return** render\_template('add\_stock.html',page=url\_for('add\_stock'), error=result) #if there is any issue in the result it will be displayed an error message in the Add Stock page

**else**:

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

The /add\_stock route is only available for administrators so the **add\_stock**() function will first verify that and automatically send the customers to the /homepage. When adding a book, the method ‘POST’ will be triggered. Firstly, it will call the upload\_file() function from the processing.py. That function will save the image in the upload folder and return the image’s name (to later save on the database). After that, the **add\_stock**() function will store the data that the admin submitted with the form as a dictionary in the data variable. It will then call the function add\_books() from the databasemanager.py (to add the new book into the database) and keep what that function returns in the result variable in order to then check if the book was correctly inserted. If the book’s information is incorrect, the Add Stock page will show an error message, otherwise, the user will be redirected to the Homepage.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente

Figure 10: How the Add Stock page form looks like

Uma imagem contendo Diagrama

Descrição gerada automaticamente

Figure 11: The Add Stock page form with the error message

**def** **upload\_file**(files, uploadfolder): #uploads the file and returns the file name

#code based on https://stackoverflow.com/questions/44926465/upload-image-in-flask

**if** 'picture' **not** **in** files: #if picture doesn't exist

**return** **False**

file = files['picture'] #gets the picture from the request files

filename = secure\_filename(file.filename)

path = os.path.join(uploadfolder, filename) #creates the picture path

file.save(path) #saves the picture

**return** filename

The **upload\_file**() function is called by the **add\_stock**() function. This function aims to save the book’s picture (from the book added in the Add Stock page) into the upload folder. If the picture wasn’t sent the function will return **False**. Otherwise, the function gets the picture from the request files and saves it in the file variable. The secure\_filename function will transform the image’s name into a secure name, so it is able to store the image into the folder securely. The filename will receive the secure image’s name. Then, to create the path to where the image is going to be saved we use the os.path.join() function. Finally, the image is saved in the path and the function returns the name of the file.

**def** **add\_books**(isbn, name, author, date, description, picture, quantity, retail\_price, trade\_price): #add books into the db

**try**:

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

cur = con.cursor()

cur.execute("SELECT count(\*) FROM books WHERE isbn=?;", (isbn,)) #returns the first row with that input

**if**(int(cur.fetchone()[**0**]))>**0**: #if the a book with that isbn already exists

**return** "The book already exists."

**else**: #if the a book with that isbn doesn't exist

cur.execute("INSERT INTO books (isbn, name, author, date, description, picture, quantity, retail\_price, trade\_price) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?);", (isbn, name, author, date, description, picture, quantity, retail\_price, trade\_price))

con.commit()

con.close()

**return** **True**

**except** **Exception**: #when there is an issue

traceback.print\_exc()

**return** "Invalid book."

The **add\_books**()function is also called by **add\_stock**() function in the main.py. It verifies if the book we are trying to add already exists in the database and if not, it adds that new book. To do that, we need to execute a query that compares the isbn of the new book with the isbn’s of the existing books and returns 1 if it gets a match, otherwise, it returns 0. So, if that value is bigger than 0, the function will return an error message that will be displayed on the Add Stock page. If the value is 0, that means the book is not in the database yet and so we need to do a query that will use the values the admin submitted to add the book into the books table of the database.

**@app**.route('/add\_to\_cart/<isbn>/<quantity>/<name>')

**@login\_required**

**def** **add\_to\_cart**(isbn, quantity, name): #function used to add items to the Shopping cart

**if** session["type"] == "admin": #page not available to admins

**return** redirect(url\_for('homepage'))

price = get\_price(isbn) #gets the book price from the db

**if** 'cart\_books' **in** session: #verifies if there is already books in the cart

**if** isbn **in** session["cart\_books"]: #verifies if the book we are trying to add already exists in the cart

**if** int(quantity)>session["cart\_books"][isbn][**0**]: #verifies if there are enough quantity of the book we want to add

session["cart\_books"][isbn][**0**] += **1** #adds 1 to the quantity of that book in the cart

session["total\_price"] += price #adds the price of the book to the total price

session["total\_quantity"] += **1** #adds 1 to the total\_quantity

**else**:

session["error"] = "Book not available." #error message when there are not enough books in stock

**else**: #if the book is not in the cart yet

session["cart\_books"][isbn] = [**1**, price, name] #sets the quantity of that book in the cart as 1

session["total\_price"] += price #sets this book price as sum of the total price with the book’s price

session["total\_quantity"] += **1** #sets the total quantity of books in the cart as the total quantity plus 1

**else**: #if there aren't books in the cart

session["cart\_books"] = { isbn: [**1**, price, name] } #sets the quantity of that book in the cart as 1

session["total\_price"] = price #sets this book price as the total price

session["total\_quantity"] = **1** #sets the total quantity of books in the cart as 1

**return** redirect(url\_for('homepage'))

The **add\_to\_cart**() function is used for the Homepage when the user wants to add a book into the shopping cart. It calls the get\_price() function from the databasemanager.py file to get the book’s price from the database through its isbn and saves that value in the variable price. The function first verifies if there is any book in the cart, then checks if the book we are trying to add is already in the cart and finally if there is enough stock quantity for the book we want to add.

If the book already exists in the cart and the quantity of the book we want to add is enough then we increase by 1 the quantity of that book in the cart and the total quantity. We also sum to the total price with the value in price.

If the book already exists in the cart but the quantity of the book we want to add is not enough then we display an error message.

Interface gráfica do usuário

Descrição gerada automaticamente com confiança média

Figure 12: Homepage for customers with the error message

If the book doesn’t exist in the cart yet we create that book in the cart session, set the total price as the sum of the total price with the price and increase by 1 the total quantity of books.

If the shopping cart doesn’t exist, we create the cart with the book we want to add, set the total price as the price of that book and the total quantity session as 1.

This function is similar to the **delete\_book\_shopping\_cart**() function but in this last one, we decrease the values according to the book we want to delete from the shopping cart.

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

**@login\_required**

**def** **checkout**(): #display the books in the shopping cart, checks if there are enough books and calculates the postage cost

**if** session["type"] == "admin" **or** 'cart\_books' **not** **in** session: #page not available for admins and users without the cart books session

**return** redirect(url\_for('homepage'))

**if** 'error' **in** session:

error = session["error"]

session.pop("error") #deletes the session error (to prevent the error from repeting)

**else**:

error=""

results = books\_available(session["cart\_books"]) #verifies if there are enough books in the db

**if** results == **True**: #if there are enough books

rows = display\_books\_checkout(session["cart\_books"]) #uses a function from the databasemanager.py file to get the books information

**if** session["total\_quantity"] == **1**: #if the cart only has 1 book

session["postage\_cost"] = **3** #the postage cost is 3 pounds

**else**: #if the shopping cart has more than 1 book

session["postage\_cost"] = **2** + session["total\_quantity"] #calculate the postage cost

session["total\_cost"] = session["total\_price"] + session["postage\_cost"] #sets the total cost as the sum of the cart total price and the postage cost

**return** render\_template('checkout.html', books = rows, type = session['type'], error=error)

**else** : #if there are not enough books

session["error"] = "The quantity you select for the following book(s) is no longer available: "

**for** book **in** results: #to show the books that are no longer in stock and set their quantity for the available quantity

session["total\_price"] = session["total\_price"] - ((session["cart\_books"][book[**0**]][**0**] - book[**2**]) \* session["cart\_books"][book[**0**]][**1**]) #removes the book's price

session["total\_quantity"] -= (session["cart\_books"][book[**0**]][**0**] - book[**2**]) #removes the book's quantity from the total quantity

session["cart\_books"][book[**0**]][**0**] = book[**2**] #removes the quantity for the book we are checking

**if** book[**2**]==**0**: #if the quantity is 0

**del** session["cart\_books"][book[**0**]] #deletes the session that keeps that book

session["error"] = session["error"] + book[**1**] + "; " #error message with the names of the missing books

**if** session["total\_quantity"] == **0**: #if there are no books in the cart

session.pop("total\_quantity") #deletes the session that keeps the total quantity of books in the cart

session.pop("cart\_books") #deletes the session that has each book quantity

session.pop("total\_price") #deletes the session that keeps the total price of the books in the cart

session["error"] = "Your books are no longer available."

**return** redirect(url\_for('checkout'))

The goal of the **checkout**() function is to show the books in the shopping cart, calculate the postage cost and verify if there are sufficient books in stock to fulfil the order and if not to adjust the quantity of books in the cart and the total price.

If an admin or a user without the cart book session created (for example, when the user doesn’t have any book in the cart) tries to access this page will be redirected to the Homepage.

When a route sends an error to the /checkout route, the **checkout**() function shows that error (that is kept in the error session).

To check if there are enough books it calls the books\_available() function that will return True if there are enough books in the database and otherwise it returns a variable (error) with the books that are not enough.

If there are enough books the function calls the display\_books\_checkout() (that is similar to the **display\_books\_homepage**() but it returns different values) from the databasemanager.py to get the books’ information. To get the postage cost the function checks if there is only 1 book in the cart (and in this case, the postage cost is 3) or if there are more than 1 book in the cart (and in this case the postage cost would be the sum of 2 with the total number of books) and creates the total cost session to store the sum of the total price for the books with the corresponding postage cost.

Interface gráfica do usuário, Aplicativo, Site

Descrição gerada automaticamente

Figure 13: Example of Checkout page with 1 book

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Figure 14: Example of Checkout page with more than 1 book

It will return the 'checkout.html' template that takes an error that will be “ “ if there are enough books or the error message mentioned before if there are missing books.

If there aren’t enough quantity for some or all of the books we are checking, the message for the error session is defined. Then, we change the total price, total quantity, and the quantity of that book in the cart accordingly to that book’s quantity in the database (in this case, we still show the books that didn’t suffer any change).

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Figure 15: Example of Checkout page where there is an error (there is just 1 "The Hunger Games" book left)

If the quantity of the book the user wants to buy is 0, than that book won’t be displayed in the checkout and the other books won’t have any change.

If none of the books the user had in the shopping cart is still available, the user will be sent to the Homepage and an error message will be shown.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Figure 16: Error message in the Homepage when the books in the cart of the customer are no longer available

All the books that had their quantity change will have their names in the error message (as we can see the Figure 15).

**@app**.route('/payment\_successful')

**@login\_required**

**def** **payment\_successful**():

**if** session["type"] == "admin": #page not available for admins

**return** redirect(url\_for('homepage'))

results = books\_available(session["cart\_books"]) #verifies if there are enough books in the db

**if** results == **True**: #if there are enough books

sell\_books(session["cart\_books"]) #calls a function to reduce the stock quantity of the books in the cart

session.pop("total\_quantity") #deletes the session that keeps the total quantity of books in the cart

session.pop("cart\_books") #deletes the session that of books in the cart

session.pop("total\_price") #deletes the session that keeps the total price of the books in the cart

**return** render\_template('payment\_successful.html')

**else**:

session["error"] = "The quantity you select for the following book(s) is no longer available, please add them again: "

**for** book **in** results: #to show the books that are no longer in stock and removing them from the shopping cart

session["total\_price"] -= (session["cart\_books"][book[**0**]][**1**] \* session["cart\_books"][book[**0**]][**0**]) #updates the total price

session["total\_quantity"] -= session["cart\_books"][book[**0**]][**0**] #updates the total quantity of books in the cart

**del** session["cart\_books"][book[**0**]] #deletes from the cart the books that are not available

session["error"] = session["error"] + book[**1**] + " " #error message with the names of the missing books

**if** session["total\_quantity"] == **0**: #if there are no books in the cart

session.pop("total\_quantity") #deletes the session that keeps the total quantity of books in the cart

session.pop("cart\_books") #deletes the session that of books in the cart

session.pop("total\_price") #deletes the session that keeps the total price of the books in the cart

**return** redirect(url\_for('homepage'))

The **payment\_successful**() function also calls the buy\_books() function from the databasemanager.py to check if the books the user wants to buy are in enough quantity in the database and saves what it returns in the variable results. If that variable returns **True**, that means that there are enough books for the user to buy. In this case, we call another function (sell\_books()) to remove the quantity of the books the user is buying from the database. In this situation we also need to clear the total quantity, cart books and total prices sessions so when the user returns to the homepage the shopping cart has no books added.

If the buy\_books() function returns error, the user is redirected to the Homepage and a personalised error message with the books not available will show. For each of these books, we will remove their price from the total price session and the quantity the user had in the shopping cart from the total quantity session and we will delete each book from the shopping cart. After that, if there is not any book in the cart (the total quantity is equal to 0) it will also clear the total quantity, cart books and total prices sessions.

**def** **books\_available**(books): #function that verifies all books to check if there are enough quantity in the db to buy

**try**:

error = []

isError = **False**

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

cur = con.cursor()

**for** isbn **in** books.keys():

cur.execute("SELECT count(\*), isbn, quantity FROM books WHERE isbn=? AND quantity<?;", (isbn, books.get(isbn)[**0**]))

book = cur.fetchone() #gets the first row

**if**(int(book[**0**]))>**0**: #if the book doens't have the necessary quantity of that book, the count is 1, so an error is raised

isError = **True**

error.append([isbn, books.get(isbn)[**2**], book[**2**]]) #adds the book info into the error(isbn, name, quantity)

**if** isError: #If there is an error

**return** error #returns an array of arrays with the books that didn't have enough quantity

**else**:

**return** **True**

**except** **Exception**:

traceback.print\_exc()

The **books\_available**()function is called by **checkout**()and **payment\_successful**()functions and verifies if the books the user is buying have enough quantity in the database.

To do that, we need to do a query (**for** each book) that will verify if the book has enough quantity (it will return 1 if it has and 0 if it doesn’t). For each error it gets, the name of the book and its isbn will be added to the error variable. Finally, it will verify if there was any error and accordingly to that return True (if there is no error) or error (if there are values inside the variable).

## 5 - TESTING

.1 TESTING REGIME

**1 Project**

BookShop consists of an online app for a company selling books that offer registration, login, homepage with a shopping cart and checkout with payment included for the customers. The app also has a version for the administrators to control the quantity of books in stock and to add new books. The checkout is available for every user. The app is done in Flask, SQL, html and css and (bootstrap) with no real payment and no advanced functionalities but that fit the requirements. Although the fact security is not a priority in this project, there are still some functions to reinforce that (namely the one to encode the password and the one that gets the book’s values directly from the database).

**2 Test scope**

Functional tests for each of the major system components:

* Register
* Login
* Homepage
* Stock Level
* Add Stock
* Shopping Cart
* Checkout
* Payment

**2.1 out of scope**

* Security
* UX

**3 Test regime**

The tests should take 60 minutes to be done.

3.1 Registration

Test 1: Use an existing username

Test 2: Submit without filling the username or password.

3.2 Login

Test 1: login with a username that is not in the database (that doesn’t exist yet)

Test 2: login with a wrong password

Test 3: login withot filling one of the fields.

Test 4: login with different kinds of users (admin and customer1)

3.3 Homepage - Customer

.1 Correct books displayed

Test 1: inspect the books with positive quantity in the database records and cross these with the displayed ones. These should match

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

Test 2: delete or set a different quantity of some book in the database and check if after that the changed book is not displayed

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

.2 Books are correctly displayed

Test 1: ensure that images are correctly resized and that the titles correspond with the images.

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

.3 Books can be added to the shopping cart

Test 1: add a book.

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

.4 Correct number of books and total price with the shopping cart icon

Test 1: add some books into the shopping cart. The number of books and total price should change according to that.

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

.5 Logout

Test 1: press the logout button on the top left corner

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

3.4 Shopping cart

.1 It is shown when pressing the shopping cart icon

Test 1: press the shopping cart icon

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

.2 Book are correctly shown (it includes having a thumbnail picture, the book’s name, the unit price and the quantity selected)

Test 1: ensure that the books information is correct

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

.3 Books can be added and removed through the corresponding buttons

Test 1: Add book

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

Test 2: Remove book

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

.4 The shopping cart can be deleted

Test 3: Delete the shopping cart

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

.5 Books’ prices and total price are correctly displayed

Test 1: Amounts should agree with the database

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

3.5 Checkout

.1 Check the if the books quantity in stock is enough

Test 1: decrease the quantity of some book in the cart so it’s lower then the number of that book in the cart. Try to checkout, the quantity of that book should change accordling to that

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

.2 Payment

Test 1: fill the payment form with acceptable details; payment successful screen will display

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

.3 The quantity of books bought is correctly updated in the database

Test 1: inspect in the database if the bought books quantity is correct after the payment

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

3.6 Homepage - Admin

.1 Correct books displayed

Test 1: inspect the books with positive quantity in the database records and cross these with the displayed ones. These should match

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

3.7 Stock level

.1 Correct books displayed

Test 1: book information displayed and database records should match

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

.2 Books quantity can be changed

Test 1: check if the wanted quantity form works

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

.3 Books can be deleted

Test 1: delete one of the books

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

3.8 Add Stock

.1 Add book form

Test 1: ensure it requires all the neccesary informations to add a new book into the database

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

.2 Adding a book that already exists in the database

Test 1: fill the form with a book from the stock level. A message error will be displayed

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

.2 TESTS RUN ON EXTERNAL CODE

## 6 - QUALITY ASSURANCE

.1 QUALITY ASSURANCE STATEMENT

Brief and requirements statement

This project is a simple bookshop website developed using Flask, SQL, html and css (with bootstrap). On this website the customers can see the available books, check their info and buy them. In order to do that, there is a fake payment option. For the admin, the website allows the user to see all the books in the database, delete them, change their quantity, and even add new books. I was able to implement all the necessary features and even add some extra ones like the “More info” to show the book’s description or the form to enable the admin to change the quantity of the books.

Coding Standards

The code is almost fully commented for the Python files and there are also some comments in the html files for a better understanding of the project. Code is divided into routes (the main pages of the website) and the functions needed for each route. The code is optimised since html and CSS files use inheritance to prevent the repetition of code.

Documentation standards

The documentation for the code is easily understandable since the it is well commented and all the needed instructions are available. The code explanation is divided into the principal functions (that correspond to the routes) and it explains in detail how the code works and how the functions are connected. The test regime offers the tester a guide in how to check the main requirements of the project for each page.

Team Details and support

Tabela

Descrição gerada automaticamente

Student 1: André Rodrigues

Student 2: Razvan-Andrey Enescu

Student 3: Helena Torrinha

Student 4: Pedro Rodrigues

To support my group, I made my work clear and easy to read, explained the most important parts of the code, wrote an adequate test regime for my project and delivered it before the soft deadline.

.2 EXTERNAL QA EVALUATION

// Your evaluation of the external’s QA statement

## 7 - DOCUMENTATION

.1 DOCUMENTATION LIST

Documentation included with this project is:

* Commented code at https://github.coventry.ac.uk/5001CEM-2122/serranotoh-BookShop
* Statement of code purpose (Section 1 of this report)
* Statement of code location (Section 2 of this report)
* Install instructions (Section 3 of this report and readme file on GitHub)
* Code explanation (Section 4 of this report)
* Testing Regime (Section 5 of this report)
* QA Statement (Section 6 of this report)

.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?

## 8 - 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.