Section 1 - A4 - Matt Dobaj - 20350312 - TA Group 1 - Nov. 28, 2025

Github repo: https://github.com/dobaj/cisc327-library-management-a2-0312

Section 2 - E2E Testing Approach

I used Playwright as it seemed a lot easier to use compared to Selenium. I made use primarily of its `get_by_role` and `get_by_label` functions to find the elements on the page via their recommended approach, which is said to test accessibility better.

The main features I tested were first adding and borrowing a book, and second checking a books borrowed status and returning it.

The add/borrow test first asserted that the add book page was reached from the home page. After adding the book, it asserted that the add book confirmation was shown and that the added book's details were correctly displayed in the catalog. After borrowing the book, it asserted that the successfully borrowed confirmation was shown.

The status/return test required the use of one of my previously made helper functions to borrow a book before the test began. It asserted that the borrowed book was present in the currently borrowed table and that the return success confirmation is shown to the user after returning the book.

Section 3 - Execution Instructions

The tests require the server to be running, which can be done either with: `python app.py` or by running the docker container.

To run the container, you can use the command:

```
docker run -p 5000:5000 21mtd4/library-app:v1
```

or if you built it yourself:

```
docker run -p 5000:5000 library-app
```

You can run all tests using:

```
pytest
```

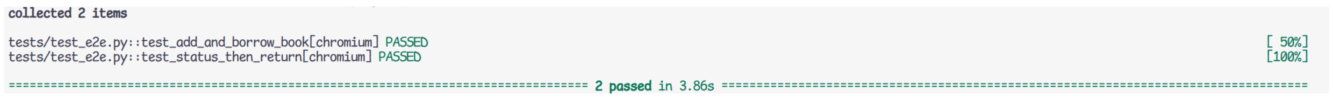
or you can run just the end to end tests using:

```
pytest .\tests\test_e2e.py
```

Section 4 - Test Case Summary

Test Function Name	Expected Results
test_add_and_borrow_book	PASS
test_status_then_return	PASS

The results should look like this:

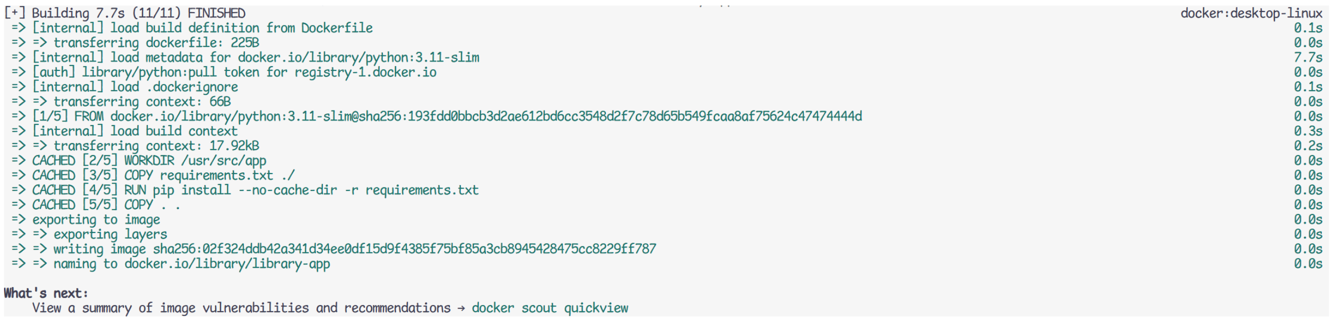


```
collected 2 items

tests/test_e2e.py::test_add_and_borrow_book[chromium] PASSED                                                  [ 50%]
tests/test_e2e.py::test_status_then_return[chromium] PASSED                                                   [100%]

=============================================== 2 passed in 3.86s ===============================================
```

Section 5 - Dockerization Process

After adding an example Dockerfile I found on the internet I could build the project using the following command:

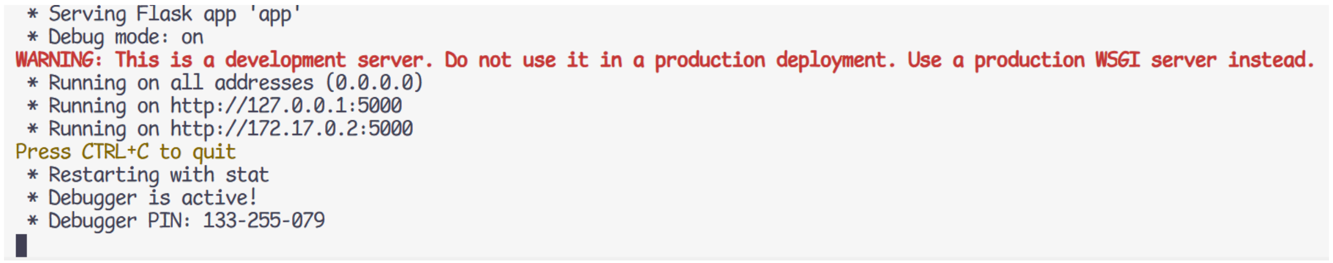
```
docker build -t library-app .
```



```
[+] Building 7.7s (11/11) FINISHED                                                              docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                            0.1s
 => => transferring dockerfile: 225B                                                                            0.0s
 => [internal] load metadata for docker.io/library/python:3.11-slim                                             7.7s
 => [auth] library/python:pull token for registry-1.docker.io                                                   0.0s
 => [internal] load .dockerignore                                                                               0.1s
 => => transferring context: 66B                                                                                0.0s
 => [1/5] FROM docker.io/library/python:3.11-slim@sha256:193fdd0bbcb3d2ae612bd6cc3548d2f7c78d65b549fcaa8af75624c47474444d  0.0s
 => [internal] load build context                                                                               0.3s
 => => transferring context: 17.92kB                                                                            0.2s
 => CACHED [2/5] WORKDIR /usr/src/app                                                                           0.0s
 => CACHED [3/5] COPY requirements.txt ./                                                                       0.0s
 => CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt                                             0.0s
 => CACHED [5/5] COPY . .                                                                                       0.0s
 => exporting to image                                                                                          0.0s
 => => exporting layers                                                                                         0.0s
 => => writing image sha256:02f324ddb42a341d34ee0df15d9f4385f75bf85a3cb8945428475cc8229ff787                    0.0s
 => => naming to docker.io/library/library-app                                                                  0.0s

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
```

I was then able to run the project after some slight modifications to the Dockerfile using the following command:

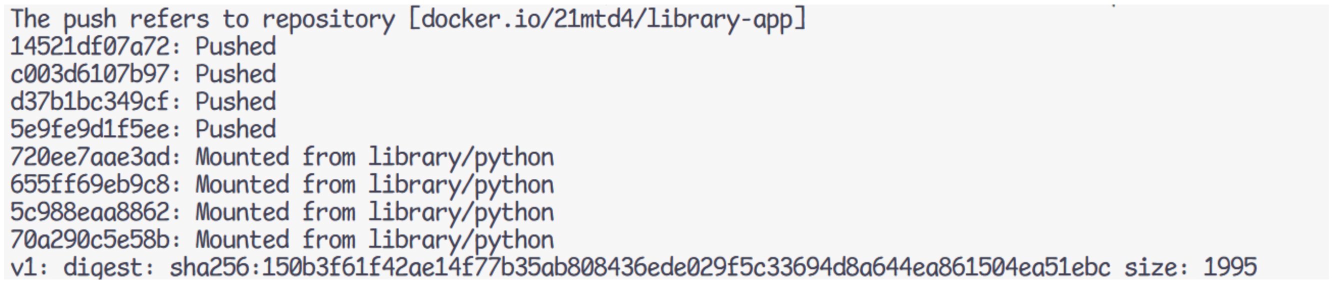
`docker run -p 5000:5000 library-app`



```
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 133-255-079
```

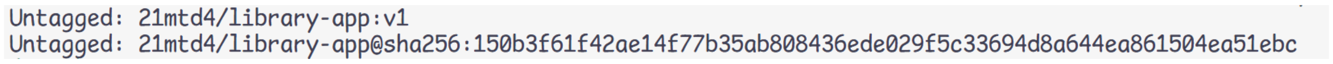
Section 6 - Docker Hub Deployment

Docker Push:



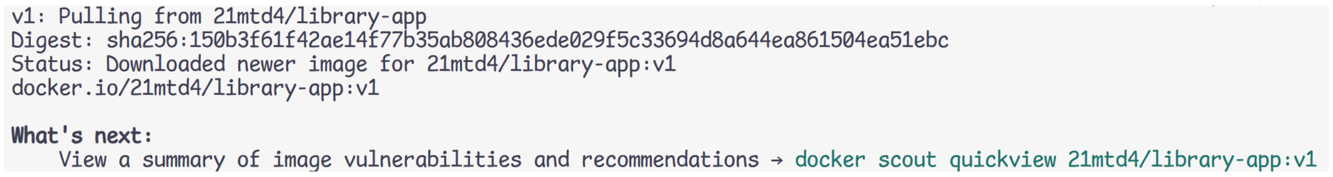
```
The push refers to repository [docker.io/21mtd4/library-app]
14521df07a72: Pushed
c003d6107b97: Pushed
d37b1bc349cf: Pushed
5e9fe9d1f5ee: Pushed
720ee7aae3ad: Mounted from library/python
655ff69eb9c8: Mounted from library/python
5c988eaa8862: Mounted from library/python
70a290c5e58b: Mounted from library/python
v1: digest: sha256:150b3f61f42ae14f77b35ab808436ede029f5c33694d8a644ea861504ea51ebc size: 1995
```

Docker Remove:



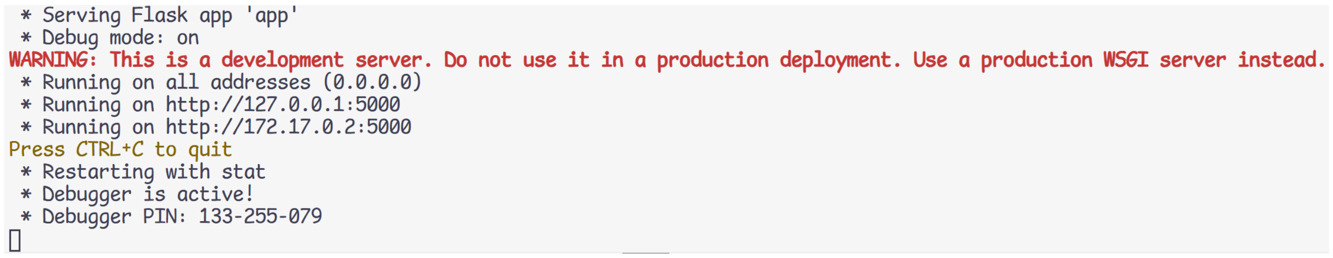
```
Untagged: 21mtd4/library-app:v1
Untagged: 21mtd4/library-app@sha256:150b3f61f42ae14f77b35ab808436ede029f5c33694d8a644ea861504ea51ebc
```

Docker Pull:



```
v1: Pulling from 21mtd4/library-app
Digest: sha256:150b3f61f42ae14f77b35ab808436ede029f5c33694d8a644ea861504ea51ebc
Status: Downloaded newer image for 21mtd4/library-app:v1
docker.io/21mtd4/library-app:v1

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview 21mtd4/library-app:v1
```

Docker Run:



```
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 133-255-079
```

Section 7 - Challenges and Reflections

I encountered some difficulties when trying to see where the testing was not producing expected results. In my add book test, for example, I couldn't see why the book wasn't being added. After a lot of trial and error, I discovered online that I could launch a browser manually in the foreground so that I could see what the tests were doing. This helped me see what element was failing but it didn't show me why the test was failing. I eventually realized that I was trying to click the link element of the "Add Book" button instead of the button itself. After fixing this, the test started performing normally. In hindsight I would probably start my testing process with the visible browser process sooner so that I could visually walk through the tests rather than guessing and checking.