CISC/CMPE 327 – Assignment 4

Name: Aiden Ramezani
Student ID: 20259781
GitHub link: https://github.com/Aidenrmz/cisc327-library-management-a4-9781

1. End-to-End (E2E) Testing Approach

For browser-based end-to-end testing, **Playwright with Pytest** was used to automate real user interactions with the Flask Library Management System. Playwright was selected because it supports real browser automation, reliable UI selectors, and seamless integration with Pytest.

The E2E tests simulate realistic user behavior rather than isolated unit tests. The Flask application is automatically launched in the background using a Pytest fixture before the tests run and is cleanly terminated afterward. This ensures that all tests run against a real, live instance of the web application.

Tested User Flows

Two primary user workflows were automated:

1. **Adding a new book and verifying it appears in the catalog**
2. **Borrowing a book and verifying a successful confirmation message**

Each test includes clear UI-based assertions to confirm that:

- Books are successfully added,
- Newly added books appear in the catalog,
- Borrow operations complete correctly,
- Success messages are displayed to the user.

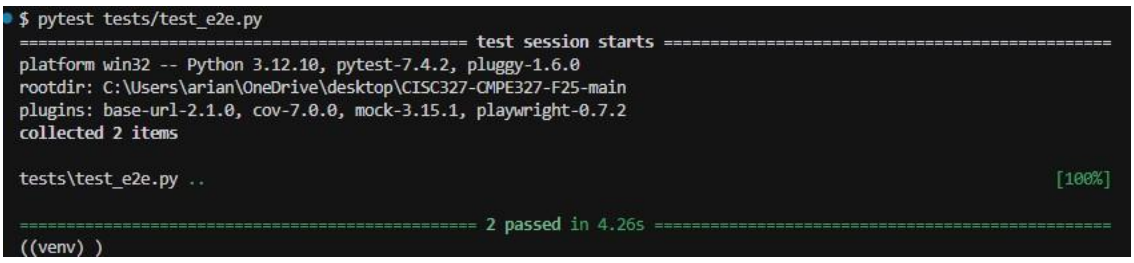

Figure 1 – Successful End-to-End Test Execution

2. Execution Instructions

2.1 Environment Setup

python -m venv venv

source venv/Scripts/activate # Windows Git Bash

pip install -r requirements.txt

playwright install

2.2 Run E2E Tests

pytest tests/test_e2e.py

2.3 Build Docker Image

docker build -t library-app .

2.4 Run Application in Docker

docker run -p 5000:5000 library-app

Access the application at:
http://localhost:5000

3. Test Case Summary

Test ID	Description	Actions	Expected Result
TC-E2E-01	Add new book and verify catalog	Navigate to Add Book → Enter book details → Submit → Navigate to Catalog	The new book appears in the catalog list
TC-E2E-02	Borrow a book and verify confirmation	Navigate to Borrow Page → Select book → Enter Patron ID → Submit	Confirmation message displayed indicating successful borrowing

4. Dockerization Process

The application was containerized using a **Dockerfile** built on top of a lightweight Python base image (python:3.11-slim). The Dockerfile performs the following steps:

1. Sets the working directory inside the container.
2. Copies and installs Python dependencies from requirements.txt.
3. Copies the Flask application source code into the container.
4. Sets required environment variables for Flask.
5. Exposes port **5000**.
6. Launches the application using:

flask run --host=0.0.0.0 --port=5000

The SQLite database is initialized automatically on application startup to ensure the container runs correctly without requiring any external database services or volume mounts.

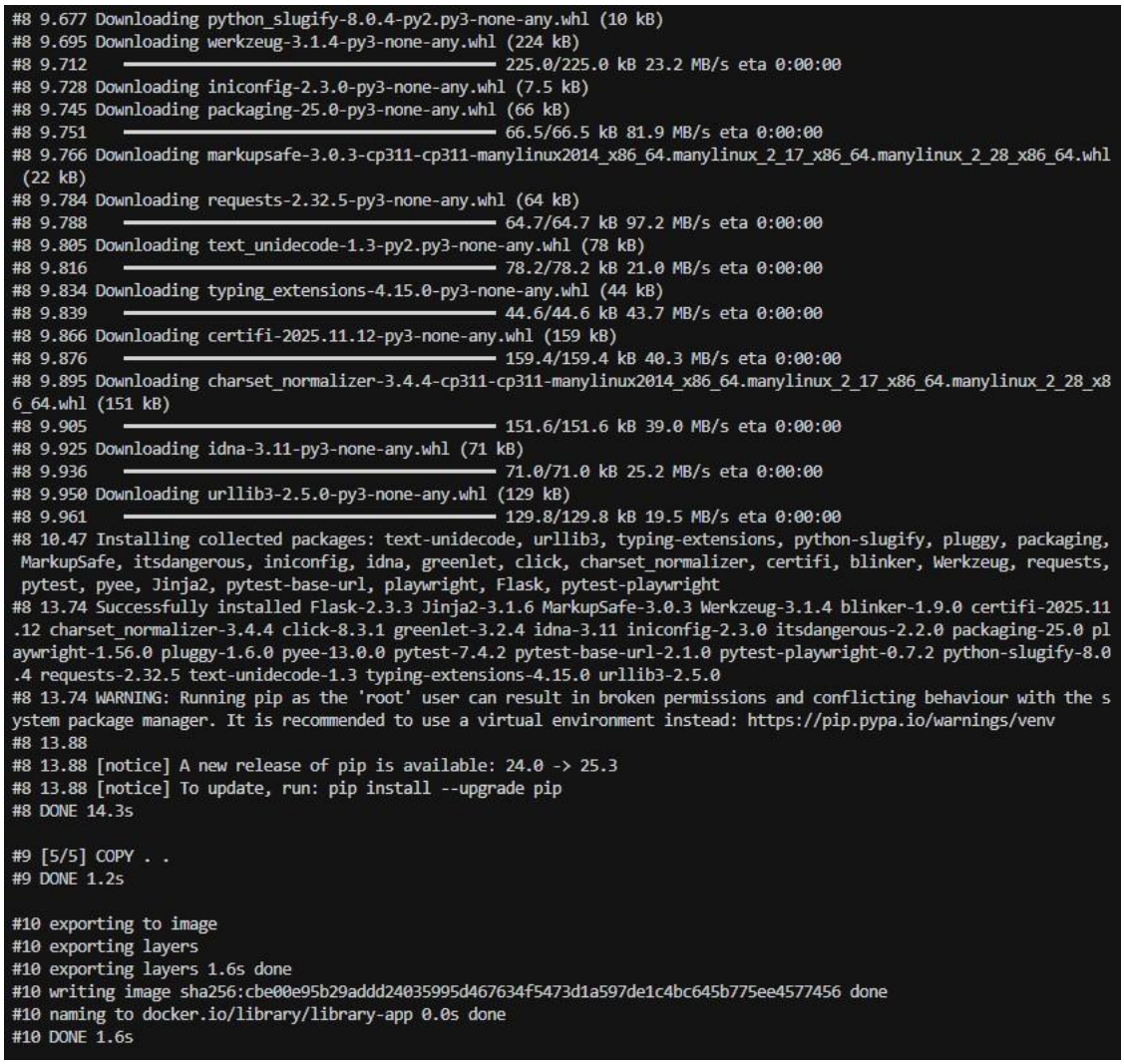

Figure 2 – Docker Build Output

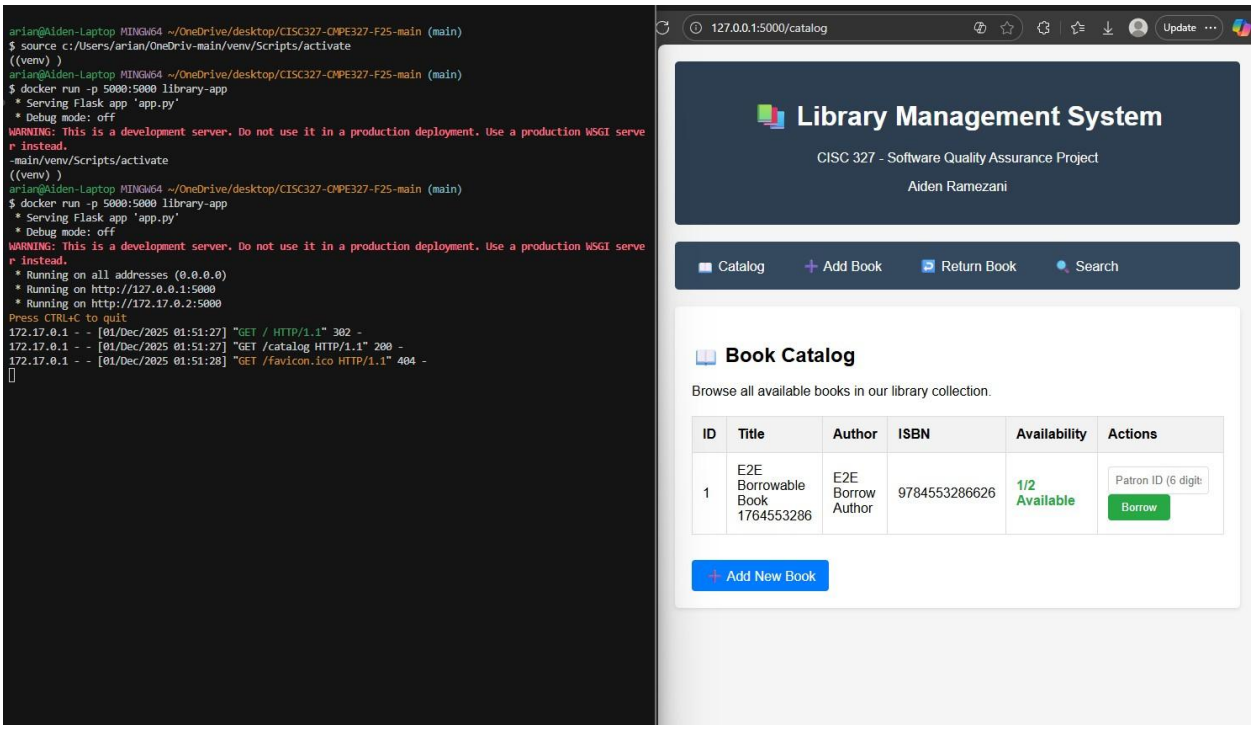

Figure 3 – Docker Run and Application in Browser

5. Docker Hub Deployment

The Docker image was successfully pushed to Docker Hub, removed locally, pulled from the registry, and executed again to verify cloud deployment functionality.

Docker Hub Command Sequence

docker build -t library-app .

docker tag library-app AIDENRMZ/library-app:v1

docker push AIDENRMZ/library-app:v1

docker rmi AIDENRMZ/library-app:v1

docker pull AIDENRMZ/library-app:v1

docker run -p 5000:5000 AIDENRMZ/library-app:v1

```
arian@Aiden-Laptop MINGW64 ~/OneDrive/desktop/CISC327-CMPE327-F25-main (main)
$ docker push aidenrmz/library-app:v1
The push refers to repository [docker.io/aidenrmz/library-app]
22e39856bc11: Pushed
1654ca450129: Pushed
34f125505920: Pushed
a38e001efb49: Pushed
720ee7aae3ad: Mounted from library/python
655ff69eb9c8: Mounted from library/python
5c988eaa8862: Mounted from library/python
70a290c5e58b: Mounted from library/python
v1: digest: sha256:9a8766a4fc598824e15a4c3cf1d51f3811ed0ee9bbbd7fff04205be0ec042711 size: 1996
((venv) )
```

Figure 4 – Successful Docker Push to Docker Hub

```
arian@Aiden-Laptop MINGW64 ~/OneDrive/desktop/CISC327-CMPE327-F25-main (main)
$ docker rmi aidenrmz/library-app:v1
Untagged: aidenrmz/library-app:v1
Untagged: aidenrmz/library-app@sha256:9a8766a4fc598824e15a4c3cf1d51f3811ed0ee9bbbd7fff04205be0ec042711
((venv) )
arian@Aiden-Laptop MINGW64 ~/OneDrive/desktop/CISC327-CMPE327-F25-main (main)
$ docker rmi aidenrmz/library-app:v1
Error response from daemon: No such image: aidenrmz/library-app:v1
((venv) )
```

Figure 5 – Docker Image Removed Locally (docker rmi)

```
arian@Aiden-Laptop MINGW64 ~/OneDrive/desktop/CISC327-CMPE327-F25-main (main)
$ docker pull aidenrmz/library-app:v1
v1: Pulling from aidenrmz/library-app
Digest: sha256:9a8766a4fc598824e15a4c3cf1d51f3811ed0ee9bbbd7fff04205be0ec042711
Status: Downloaded newer image for aidenrmz/library-app:v1
docker.io/aidenrmz/library-app:v1

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview aidenrmz/library-app:v1
((venv) )
```

Figure 6 – Docker Pull from Docker Hub

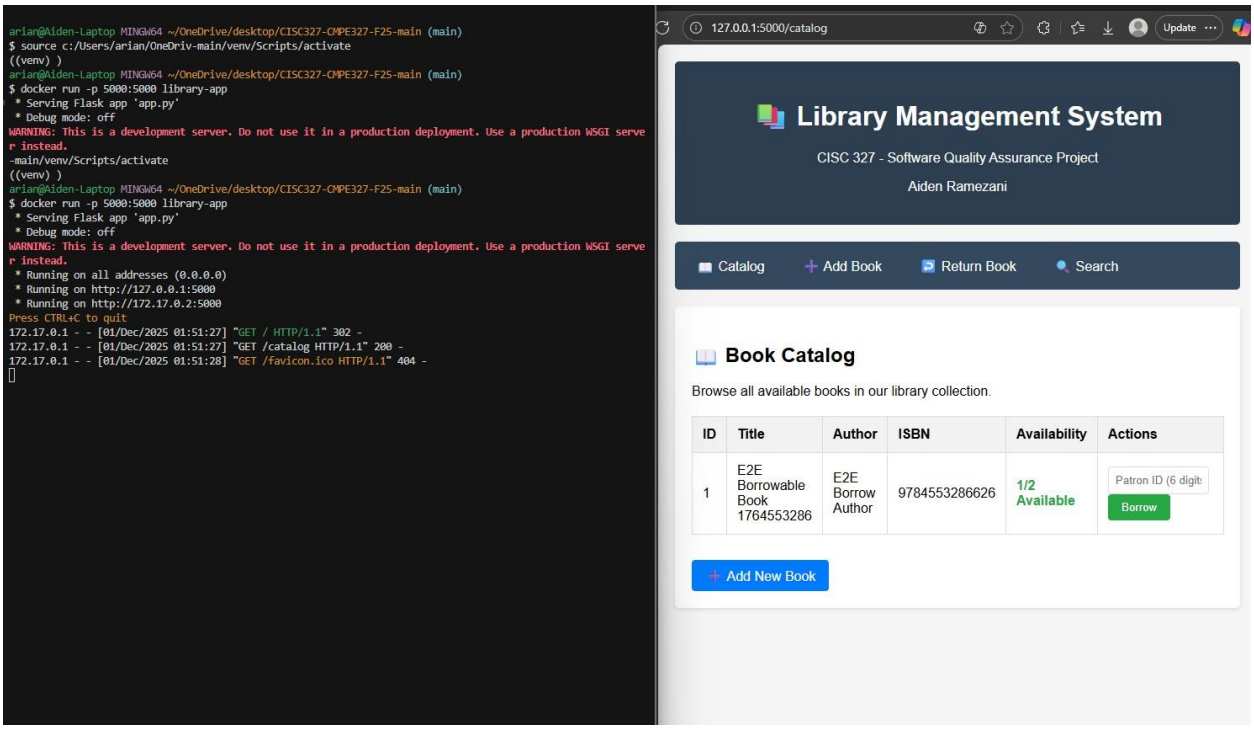

Figure 7 – Docker Run from Pulled Image

6. Challenges and Reflections

One of the main challenges encountered during this assignment was managing the automatic startup and shutdown of the Flask server for the E2E test environment. Unlike unit tests, browser-based tests require a fully running application, which required careful synchronization inside Pytest.

Another challenge involved ensuring that the SQLite database initialized automatically inside the Docker container without relying on persistent volumes or external authentication. This was resolved by initializing the database schema safely at application startup.

This assignment demonstrated the practical value of:

- End-to-end testing for verifying complete system behavior,
- Docker for consistent runtime environments,
- Cloud-based container deployment using Docker Hub.

Together, these tools improve software reliability, portability, and deployment reproducibility.