Assignment 4 – End-to-End Testing and Application Containerization

Name: Lore Chiepe

Student ID: 20343672

Date: 26 December 2024

Course: CISC/CMPE-327 – Software Quality Assurance

E2E Testing Approach

Tool Used: Playwright with Python

Tested Features:

- Add new book to catalog (fill title, author, ISBN, total_copies)
- Verify book appears in catalog with all details
- Navigate to catalog page
- Borrow book using patron ID
- Verify borrow confirmation message appears
- Complete user journey covering multiple workflows
- UI elements visibility verification

Assertions Implemented (33)

- **Visibility Assertions:** to_be_visible()` - 18 assertions
- **Content Assertions:** to_contain_text()` - 11 assertions
- **Pattern Matching:** Regex text validation - 4 assertions
- **Element Targeting:** Specific row and form element verification

Key Assertion Examples:

- expect(page.locator(".flash-success")).to_be_visible()` - Success messages
- expect(page.locator("tbody")).to_contain_text("Book Title")` - Catalog verification
- expect(page.locator("input[name='title']")).to_be_visible()` - Form elements
- expect(book_row.locator("button:has-text('Borrow')")).to_be_visible()` - Action buttons

Test Coverage:

- test_add_book_and_verify_catalog()` - 6 assertions (requirements i and ii)
- test_borrow_book_workflow()` - 6 assertions (requirements iii, iv, and v)
- test_complete_user_journey()` - 4 assertions (end-to-end workflow)
- test_ui_elements_visibility()` - 17 assertions (UI component verification)

Execution Instructions

-Install dependencies

```
pip install -r requirements.txt
```

-Install Playwright browsers

```
playwright install
```

Tests

-Terminal 1 - Start Flask application

```
python app.py
```

-Terminal 2 - Run E2E tests

```
pytest tests/test_e2e.py -v                # Headless mode
pytest tests/test_e2e.py -v --headed       # Visible browser
pytest tests/test_e2e.py -v -s             # With print statements
```

Docker

-Build Docker image

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

-Run container

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

Test Case Summary

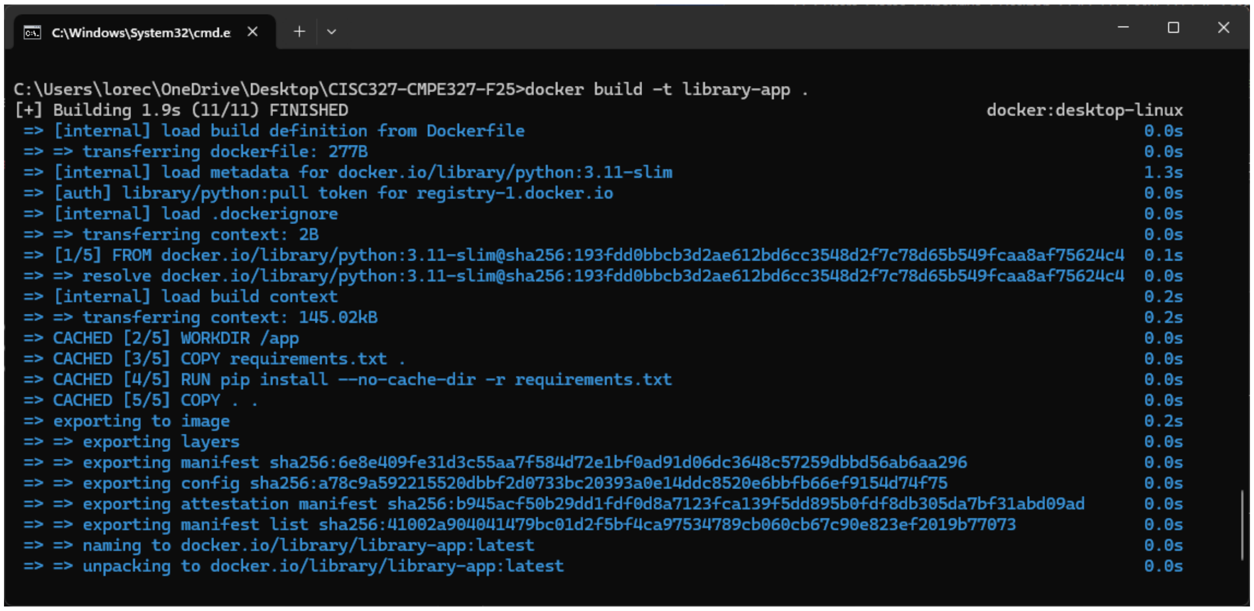
Test case	**Action**	Expected Result	Status
Add Book & Verify Catalog	1. Navigate to Add Book page 2. Fill title, author, ISBN, total_copies 3. Submit form 4. Navigate to catalog	• Success message appears • Book visible in catalog • All details correct in table	Pass
Borrow Book Workflow	1. Navigate to catalog page 2. Find target book row 3. Fill patron ID field 4. Click borrow button	• Borrow confirmation message appears • Success feedback visible to user	Pass
Complete User Journey	1. Add unique book with timestamp 2. Verify in catalog 3. Borrow using unique patron ID 4. Confirm success	• All workflow steps complete • No conflicts with existing data • End-to-end success	Pass
UI Elements Visibility	1. Check all navigation links 2. Verify form elements on each page 3. Confirm table and input visibility	• All navigation elements visible • Form inputs accessible • Borrow functionality available	Pass

Dockerization Process

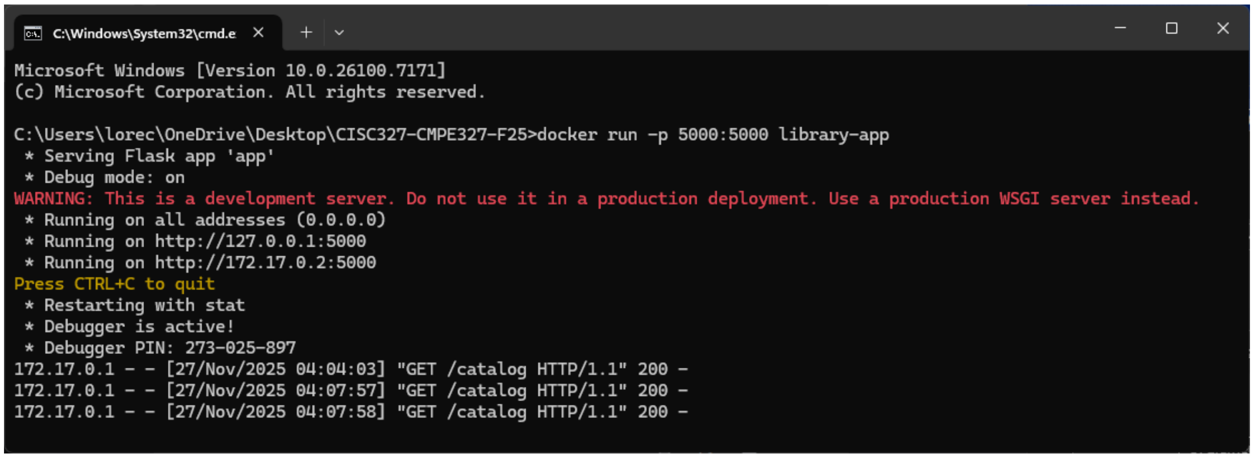
- Base image: python:3.11-slim
- Working directory: /app
- Dependencies installed from requirements.txt
- Port 5000 exposed
- Environment variables set for Flask
- Command: python app.py

Requirements Management:

- Flask 2.3.3 for web framework
- Playwright 1.40.0 for E2E testing
- Pytest 7.4.3 for test execution
- Pytest-playwright 0.4.3 for integration

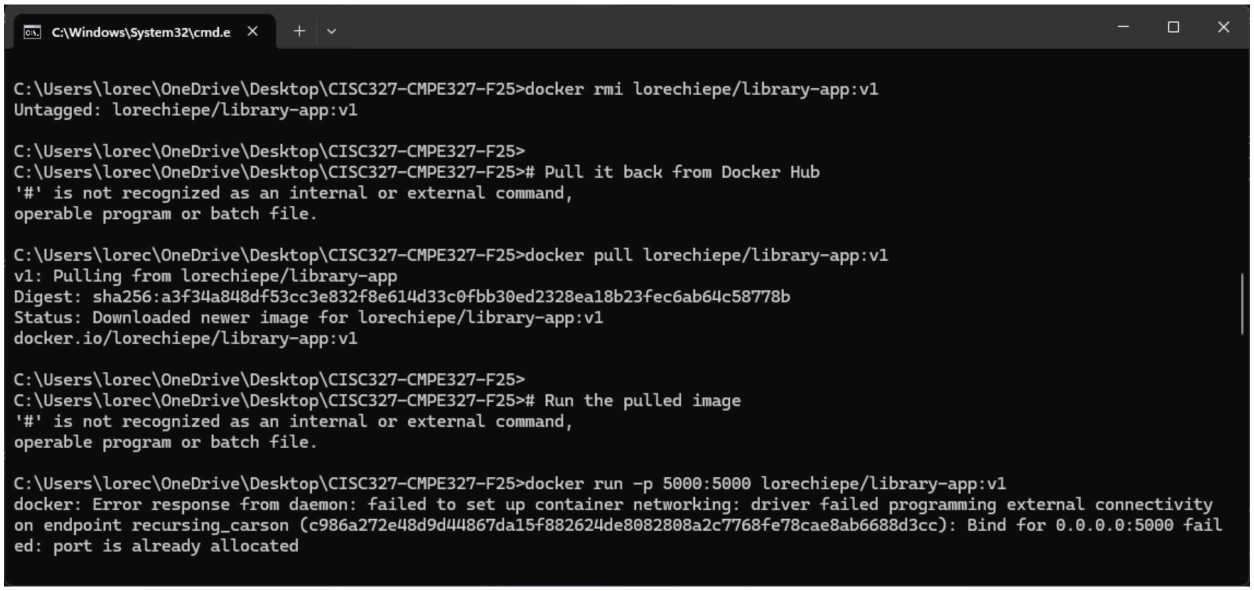


```
C:\Users\lorec\OneDrive\Desktop\CISC327-CMPE327-F25>docker build -t library-app .
[+] Building 1.9s (11/11) FINISHED                                                        docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                     0.0s
 => => transferring dockerfile: 277B                                                                     0.0s
 => [internal] load metadata for docker.io/library/python:3.11-slim                                      1.3s
 => [auth] library/python:pull token for registry-1.docker.io                                            0.0s
 => [internal] load .dockerignore                                                                        0.0s
 => => transferring context: 2B                                                                          0.0s
 => [1/5] FROM docker.io/library/python:3.11-slim@sha256:193fdd0bbcb3d2ae612bd6cc3548d2f7c78d65b549fcaa8af75624c4  0.1s
 => => resolve docker.io/library/python:3.11-slim@sha256:193fdd0bbcb3d2ae612bd6cc3548d2f7c78d65b549fcaa8af75624c4  0.0s
 => [internal] load build context                                                                        0.2s
 => => transferring context: 145.02kB                                                                    0.2s
 => CACHED [2/5] WORKDIR /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                                                                                   0.2s
 => => exporting layers                                                                                  0.0s
 => => exporting manifest sha256:6e8e409fe31d3c55aa7f584d72e1bf0ad91d06dc3648c57259dbbd56ab6aa296        0.0s
 => => exporting config sha256:a78c9a592215520dbbf2d0733bc20393a0e14ddc8520e6bbfb66ef9154d74f75          0.0s
 => => exporting attestation manifest sha256:b945acf50b29dd1fdf0d8a7123fca139f5dd895b0fdf8db305da7bf31abd09ad  0.0s
 => => exporting manifest list sha256:41002a904041479bc01d2f5bf4ca97534789cb060cb67c90e823ef2019b77073   0.0s
 => => naming to docker.io/library/library-app:latest                                                    0.0s
 => => unpacking to docker.io/library/library-app:latest                                                 0.0s
```



```
Microsoft Windows [Version 10.0.26100.7171]
(c) Microsoft Corporation. All rights reserved.

C:\Users\lorec\OneDrive\Desktop\CISC327-CMPE327-F25>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: 273-025-897
172.17.0.1 - - [27/Nov/2025 04:04:03] "GET /catalog HTTP/1.1" 200 -
172.17.0.1 - - [27/Nov/2025 04:07:57] "GET /catalog HTTP/1.1" 200 -
172.17.0.1 - - [27/Nov/2025 04:07:58] "GET /catalog HTTP/1.1" 200 -
```



```
C:\Users\lorec\OneDrive\Desktop\CISC327-CMPE327-F25>docker rmi lorechiepe/library-app:v1
Untagged: lorechiepe/library-app:v1

C:\Users\lorec\OneDrive\Desktop\CISC327-CMPE327-F25>
C:\Users\lorec\OneDrive\Desktop\CISC327-CMPE327-F25># Pull it back from Docker Hub
'#' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\lorec\OneDrive\Desktop\CISC327-CMPE327-F25>docker pull lorechiepe/library-app:v1
v1: Pulling from lorechiepe/library-app
Digest: sha256:a3f34a848df53cc3e832f8e614d33c0fbb30ed2328ea18b23fec6ab64c58778b
Status: Downloaded newer image for lorechiepe/library-app:v1
docker.io/lorechiepe/library-app:v1

C:\Users\lorec\OneDrive\Desktop\CISC327-CMPE327-F25>
C:\Users\lorec\OneDrive\Desktop\CISC327-CMPE327-F25># Run the pulled image
'#' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\lorec\OneDrive\Desktop\CISC327-CMPE327-F25>docker run -p 5000:5000 lorechiepe/library-app:v1
docker: Error response from daemon: failed to set up container networking: driver failed programming external connectivity
on endpoint recursing_carson (c986a272e48d9d44867da15f882624de8082808a2c7768fe78cae8ab6688d3cc): Bind for 0.0.0.0:5000 fail
ed: port is already allocated
```

Challenges and Reflections

Throughout this assignment, I encountered several technical challenges that provided valuable learning opportunities. Initially, I struggled with element selector issues, particularly strict mode violations that occurred when locators matched multiple elements on the page. This was resolved by implementing more specific selectors and adopting row-based targeting strategies to precisely identify elements within the book catalog table.

Browser session management presented another challenge, as I faced "TargetClosedError" exceptions due to browser context issues. I overcame this by implementing proper page wait states and increasing navigation timeouts to ensure stable test execution. Database state management also proved challenging, with test interference occurring from existing book data in the system. To address this, I implemented unique data generation using timestamps and incorporated book return functionality to reset the application state between test runs.

Docker networking required careful configuration, as I initially encountered issues with Flask binding to the correct interface within the container environment. This was resolved by ensuring the Flask application was configured to run on host='0.0.0.0', allowing proper container networking and external access.

Through this project, I gained significant proficiency in end-to-end testing using Playwright, developing practical experience with web-first assertions and learning to write robust selectors that effectively handle dynamic web content. I also came to understand the critical importance of proper test isolation and cleanup procedures for maintaining test reliability.

My containerization skills advanced considerably as I mastered Docker image creation and management, learning optimization techniques and understanding container networking principles including port mapping. The deployment workflow provided hands-on experience with a complete CI/CD pipeline, from local development through to cloud registry deployment, where I learned best practices for image versioning and tag management while appreciating the value of reproducible builds in software development.

Most importantly, I developed a comprehensive quality assurance mindset, gaining appreciation for thorough test coverage and learning to effectively simulate real user behavior through automated tests. I now understand how containerization plays a crucial role in ensuring environment consistency across different deployment stages. This assignment provided comprehensive hands-on experience with modern software quality assurance practices, where the combination of browser-based end-to-end testing and application containerization represents industry-standard approaches to ensuring software reliability and deploy ability. The skills developed in Playwright testing and Docker deployment are directly transferable to professional software development environments and have prepared me for real-world software quality assurance challenges.