Trevor White (20348221)
Sophia Pagazani (20333850)
Aniss Hamouda (20348807)
CISC 327
03/12/2024

Project Assignment 6

End-to-End System Test

This document provides an overview of how to run tests, descriptions of the three selected features, explanations of each test script and proof of success, and the task breakdown.

1. How to Run Tests

Before running tests, you must ensure the virtual environment is up-to-date and the browsers used by the `playwright` tool are installed on the system. To do this, activate the environment, then run the following commands:

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

As for the tests, they are organized by type under the `unit/`, `integration/`, and `end-to-end/` directories. The latter contains all the test scripts relevant to this assignment, which require an instance of the app to be concurrently running in the localhost. This is the default location, so, in a separate terminal, simply run from the `app/` directory:

```
python app.py
```

Then, you can run all the tests at once using `pytest` or target the subdirectories individually from the `app/` directory:

```
pytest tests/unit
pytest tests/integration
pytest tests/end-to-end
```

2. Selected Features

For end-to-end testing, we considered the three features that were implemented from the specifications. The requirements as they appear in assignment 1 are listed below. New test suites were written for them, using the `playwright` tool for Python to simulate a user opening the page in a web browser. Remark that requirement 3 covers several functionalities. However, the corresponding test excludes the feature of viewing detailed drug information. This is because searching was the only functional partition implemented.

1. Administrators shall be able to create a prescription order for a client. (3.1)
2. Administrators shall be able to view and update the status of a client's prescription. (3.2)
3. Administrators shall be able to search their inventory for detailed information about a particular drug, namely its identifiers, quantity in stock, dosage instructions, ingredients, and side effects. (3.3)

3. End-to-End Test Scripts

3.1 `test_user_create_prescription.py`

This test ensures that when a user tries to create a new prescription, they get the correct response. If the user enters no DIN, or a DIN that does not exist in the drug database, hitting the submit button on the prescription creation page should reload the same page again with a message saying, “Please enter valid data.” The file has 2 tests that each test one of those cases. The other test ensures that if the user has a valid submission, then after clicking the submit button, they are redirected to the client page and that the prescription they just created is present.

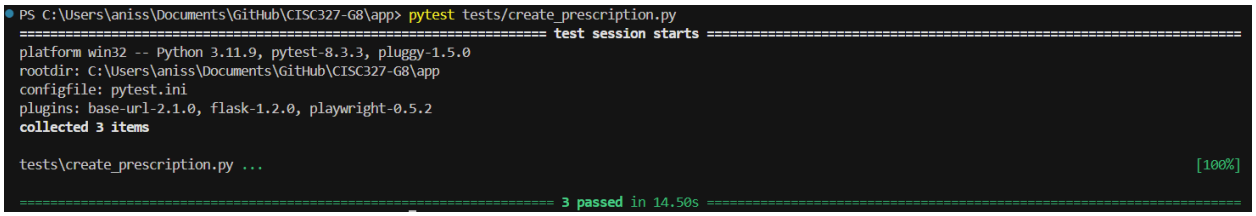

Figure 1: Passing output of `test_user_create_prescription.py`

3.2 `test_user_change_status.py`

This test ensures that when a user changes a prescription's status, the appropriate response is reflected to them. There are three types of test cases within that, with the first being changing an active prescription status to any other active status state (e.g., changing “Ready” to “Needs Filling”). The user should see the correct status appear, and the prescription should stay where it is. The other two cases are inverses of each other and entail switching from an active prescription to an inactive prescription and vice versa (e.g., changing “Ready” to “Non-Active” or changing “Non-Active” to “Needs Filling”). For these two cases, not only should the user see the status change, but the prescription should also move into the appropriate box (Active or History) for display. This test mimics test case `ChangePrescriptionStatus` from assignment 1 in requirement 2, and it intends to make sure that the user is accurately being informed on the status of the prescription.

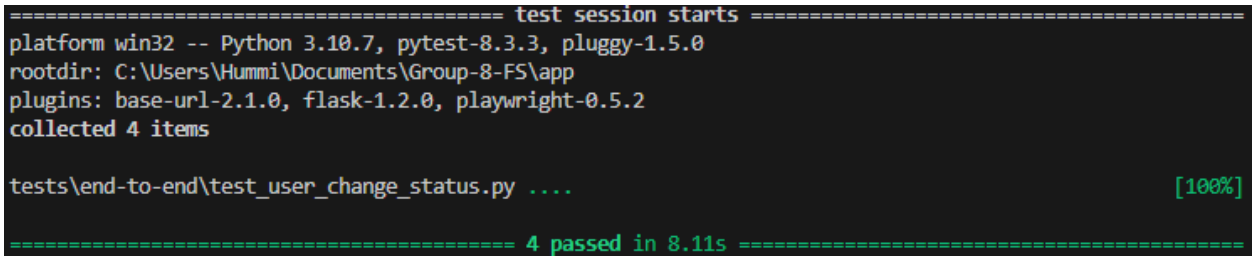

Figure 2: Passing output of `test_user_change_status.py`

3.3 `test_user_drug_client_search.py`

This test script targets the drug search functionality from requirement 3, as well as its reuse on the client search page. It defines four cases: search for a client by name, for a client by phone, for a drug by name, and for a drug by DIN. Each case loads the home page, navigates to the respective search pages, then enters a series of search queries into the on-screen search bar. These queries are designed to interface with the system as a user would, with some intended to succeed and others to fail. The cases that search by client or drug name have an additional query that expects multiple results to be found in the case where a real user may input a partial name. This is still considered a properly formatted query, so results must be returned accordingly, unlike an incomplete DIN or phone number.

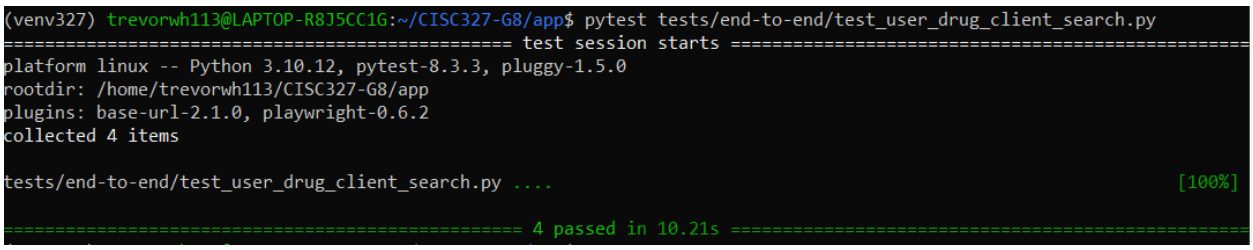

Figure 3: Passing output of `test_user_drug_client_search.py`

3.4 `test_user_navigation.py`

Lastly, this test does not pertain to any of the selected features. Rather, it serves to ensure that navigation between pages as a user, by clicking the provided buttons, functions as intended. This test was initially written for assignment 5 as an integration test under the name `verify_navigation.py`. That said, we realized it is better defined as an end-to-end test, which is why it appears here. Accordingly, it no longer uses `pytest-flask`, making it now platform-independent, and it is renamed `test_user_navigation.py`.

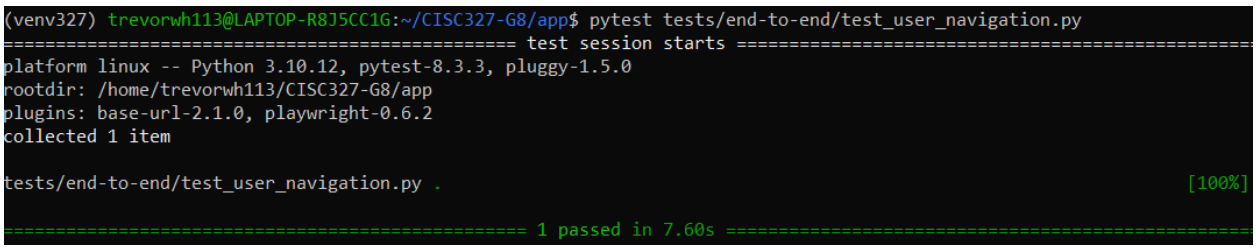

Figure 4: Passing output of `test_user_navigation.py`

4. Task Distribution

Trevor
Wrote end-to-end test for user navigation and the search functionality
Wrote associated explanation in the report document
Wrote all other sections of the report (How to Run Tests and Selected Features)

Sophia
Wrote end-to-end test for changing prescription status
Wrote associated explanation in the report document
Created readme file

Aniss
Wrote end-to-end test for creating prescriptions
Wrote associated explanation in the report document
Worked on task distribution file