Graphical User InterfaceTesting Assignment

Sumanth Reddy Velagala

School of Computing and Augmented Intelligence, Arizona State University

April 25, 2025

Task 1 - Develop two verisons of an application with a GUI:

This Grocery Tracker application allows users to input grocery items along with quantity, department, and date of purchase. It stores the data locally in the browser using localStorage. Users can view a summary of all added items in a scrollable list. The summary can be filtered by purchase date or department (Fruits and Vegetables). It also includes options to clear filters or remove all items. It is developed using HTML.

Version - 1

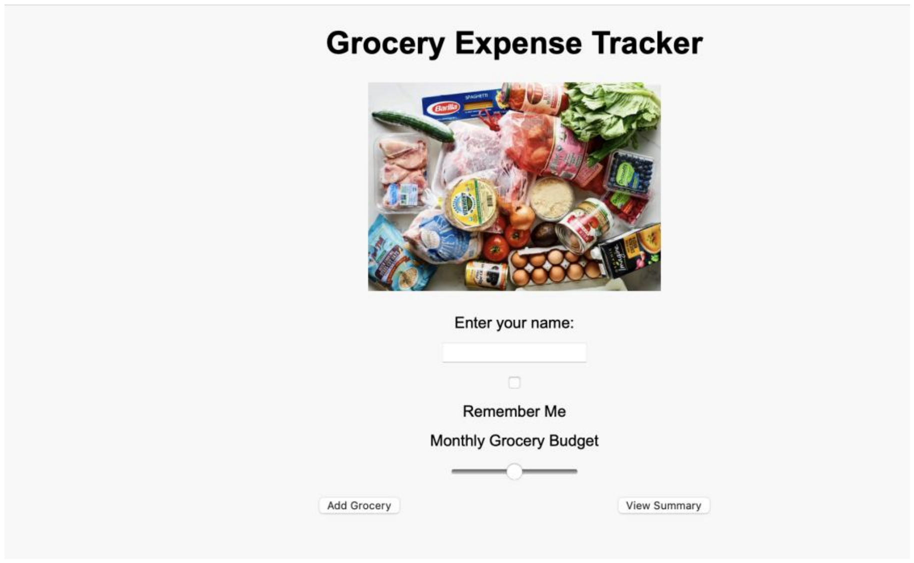


Figure 1 : Snippet of index page

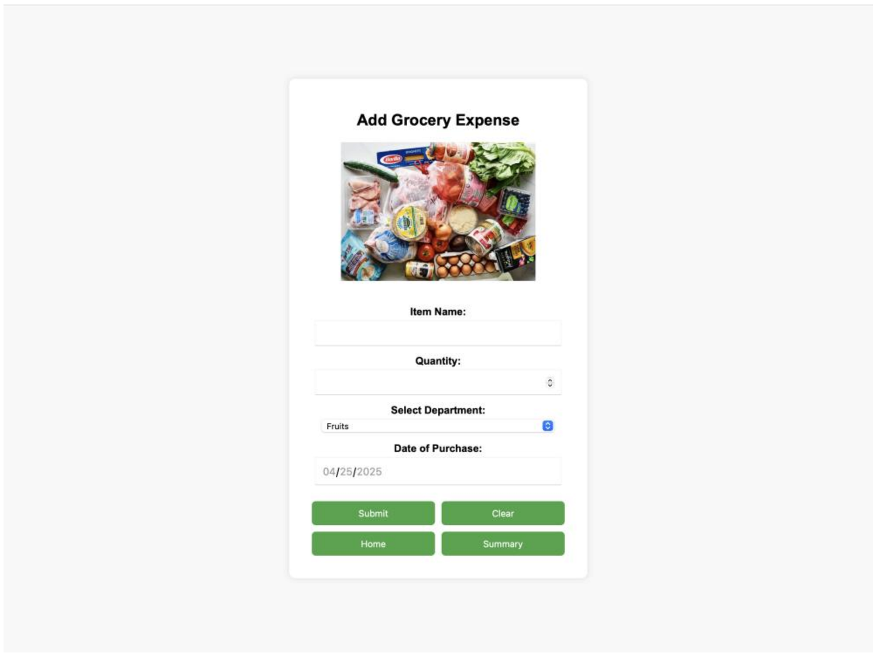
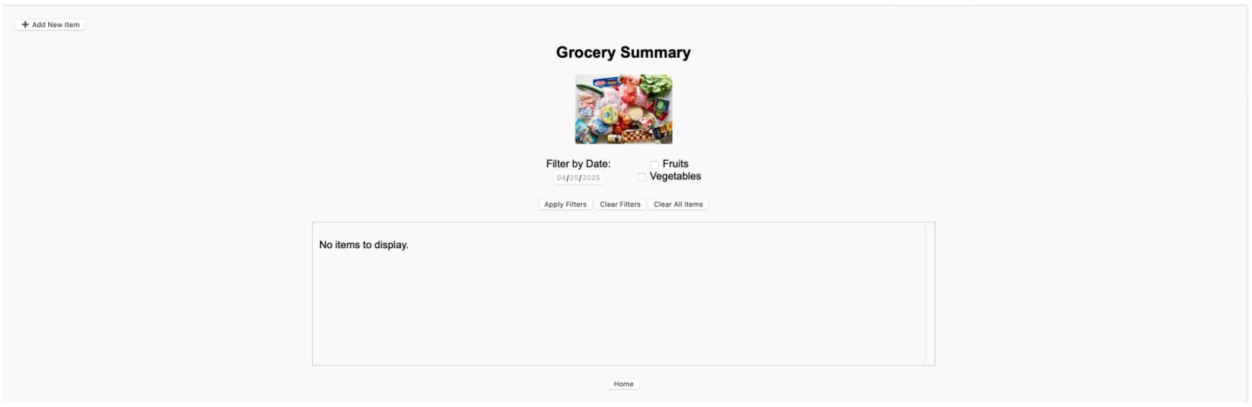


Figure 2 : Snippet of add page



No items to display.

Figure 3 : Snippet of summary page

Version - 2

The three changes in the index page of version 2 are, change in image size from 300px to 200px, change in the location of the form, change in orientation of Add Grocery and View Summary buttons.

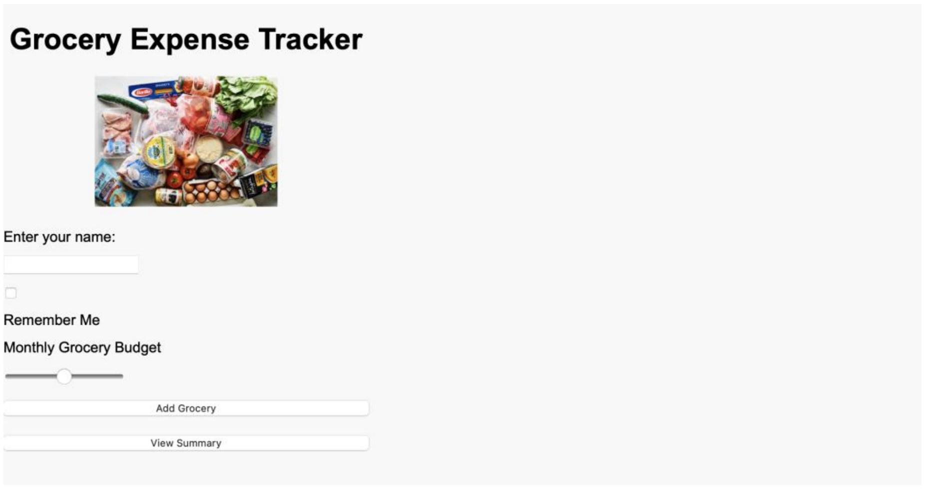


Figure 4 : Snippet of index page

The three changes in the index page of version 2 are change in image size from 300px to 220px, change in width of the form, change in orientation of Submit, Home, Clear button**s.**

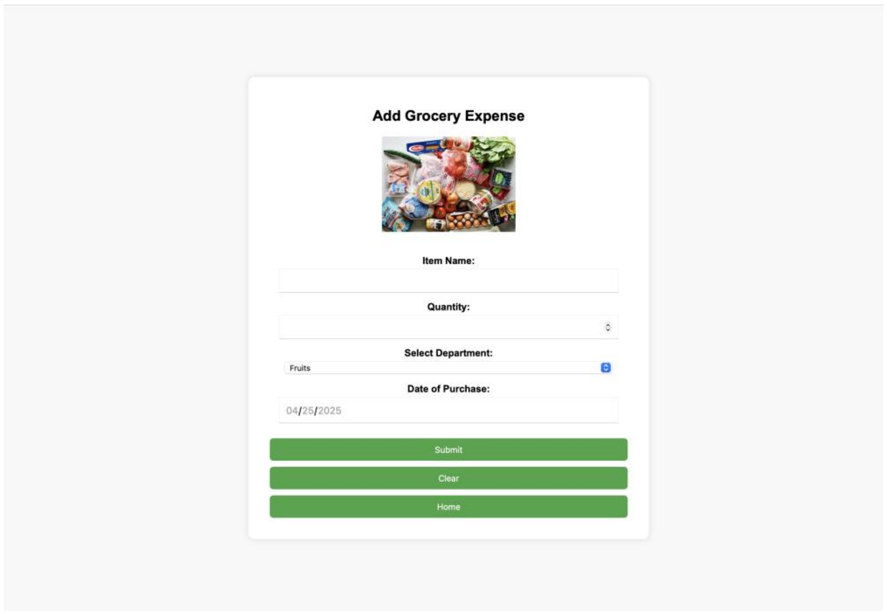


Figure 5 : Snippet of add page

The three changes in the below summary page of version 2 are change in size of scroll box, change in the location of the form, change in orientation of Filter, Fruits and vegetables check box.

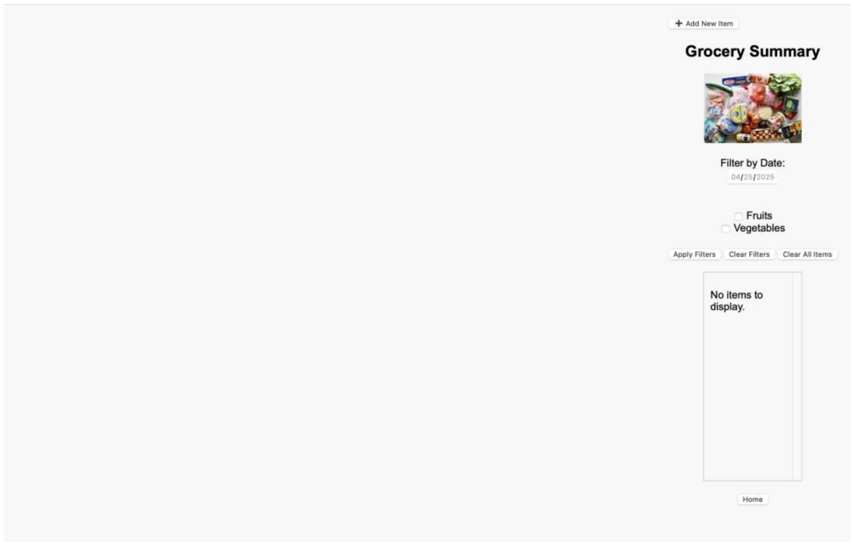


Figure 6 : Snippet of summary page

Task 2 - Description of the tool selected for GUI Testing

I have used a open-source tool called Playwright developed by Microsoft for GUI testing. This tool is easy to use with simpler syntax and can be installed using pip command.

Features (Johnson,2023) **:**

1. Cross-browser support (Chromium, WebKit, Firefox)
2. Cross-platform testing (web, mobile, desktop)
3. Mobile emulation (geolocation, screen size, device characteristics)
4. Multi-language support (JavaScript, TypeScript, Python, Java, C#)
5. Headless and headful execution modes
6. Isolated browser contexts for multi-session testing
7. Automated screenshots and video recording
8. Network request interception and mocking
9. Robust, auto-waiting assertion API
10. Automation of user interactions (clicks, forms, navigation)

Scope (Saini, 2024) **:**

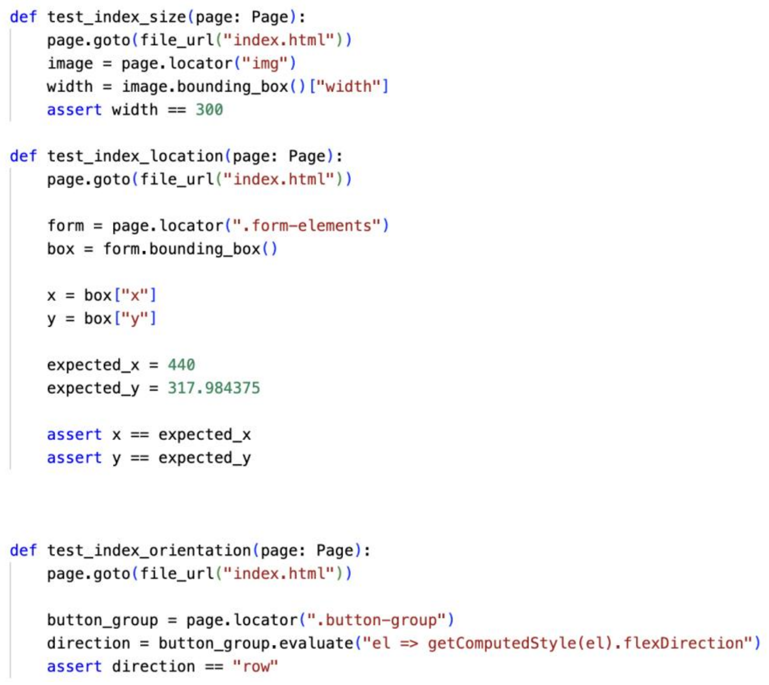
Cross-browser testing, API testing, performance testing, headless and headed mode execution, automatic waiting, parallel execution, network interception, multi-layer testing across UI, API, and performance.

Area of Usage (Johnson,2023) **:**

1. Using explicit waits (e.g., waitForSelector) to handle delayed element rendering due to network or animations
2. Leveraging browser contexts to isolate tests and prevent cross-test interference
3. Optimizing test performance by minimizing navigation, using headless mode, and running tests in parallel
4. Utilizing Playwright's debugging tools such as verbose logs, Inspector, and video recording for troubleshooting
5. Integrating Playwright into CI/CD pipelines for automated testing on every code commit

Task 3 - Description of test cases developed

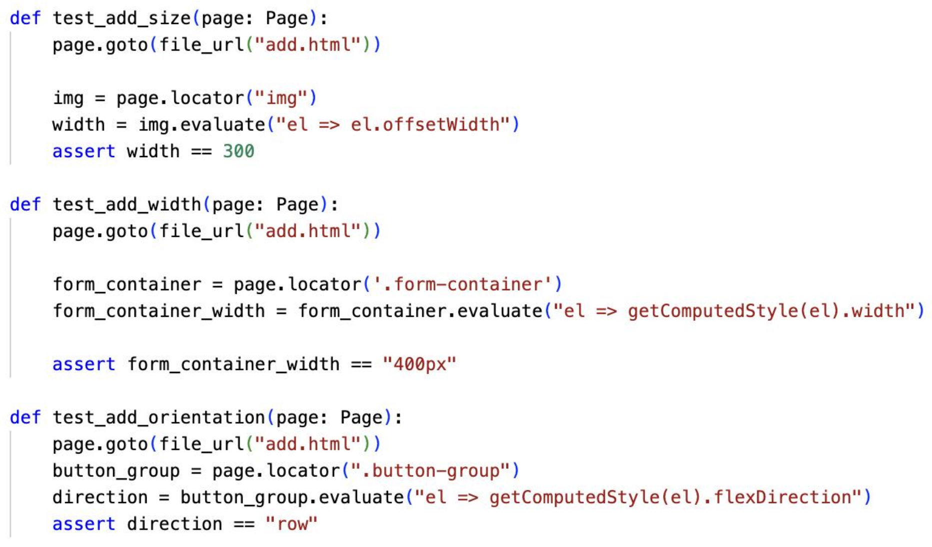
The below test cases test_index_size() checks for any changes made to the image, Test_index_location() checks for any changes made to location of the form, test_index_orientation() checks for the changes made to form orientation in home page in version 2.



```python
def test_index_size(page: Page):
    page.goto(file_url("index.html"))
    image = page.locator("img")
    width = image.bounding_box()["width"]
    assert width == 300

def test_index_location(page: Page):
    page.goto(file_url("index.html"))

    form = page.locator(".form-elements")
    box = form.bounding_box()

    x = box["x"]
    y = box["y"]

    expected_x = 440
    expected_y = 317.984375

    assert x == expected_x
    assert y == expected_y


def test_index_orientation(page: Page):
    page.goto(file_url("index.html"))

    button_group = page.locator(".button-group")
    direction = button_group.evaluate("el => getComputedStyle(el).flexDirection")
    assert direction == "row"
```

Figure 7 : Snippet of test cases for testing the 3 changes made to the index page

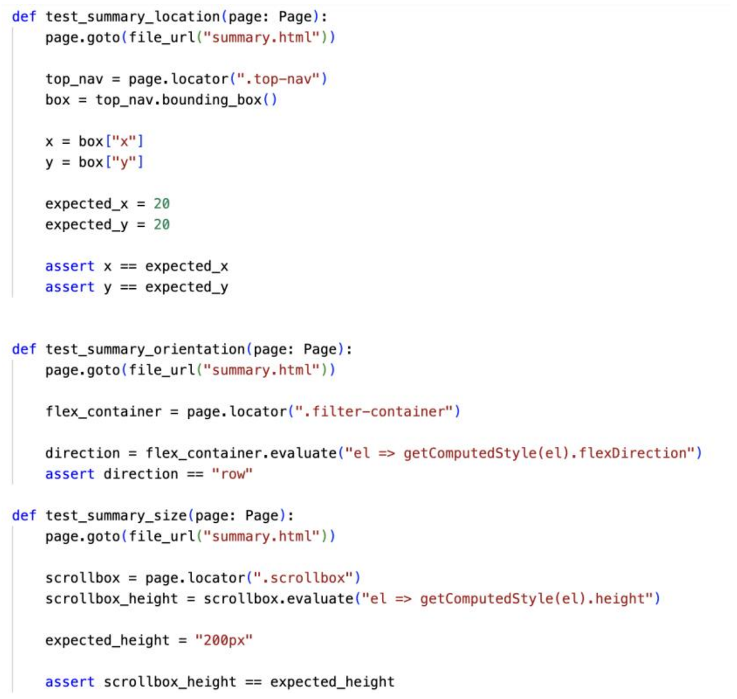
The below test cases test_add_size() checks for any changes made to the image, Test_add_width() checks for any changes made to size of the form, test_add_orientation() checks for the changes made to form orientation in add page in version 2.



```python
def test_add_size(page: Page):
    page.goto(file_url("add.html"))

    img = page.locator("img")
    width = img.evaluate("el => el.offsetWidth")
    assert width == 300

def test_add_width(page: Page):
    page.goto(file_url("add.html"))

    form_container = page.locator('.form-container')
    form_container_width = form_container.evaluate("el => getComputedStyle(el).width")

    assert form_container_width == "400px"

def test_add_orientation(page: Page):
    page.goto(file_url("add.html"))
    button_group = page.locator(".button-group")
    direction = button_group.evaluate("el => getComputedStyle(el).flexDirection")
    assert direction == "row"
```

Figure 8 : Snippet of test cases for testing the 3 changes made to the add page

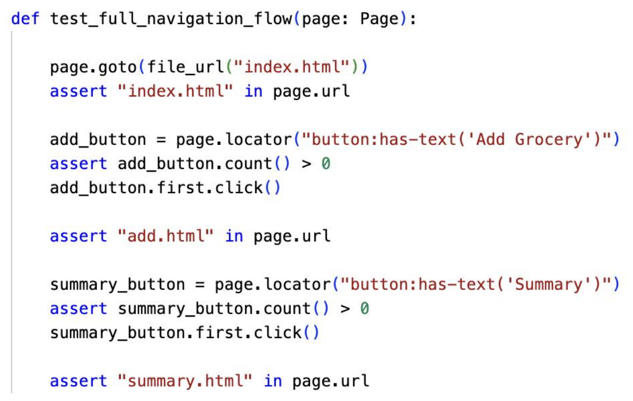
The below test cases, test_summary_location() checks for any changes made to the location of the form, test_summary_orientation() checks for any changes made to orientation of the form, test_summary_size() checks for the changes made to size of the form in summary page in version 2.



```python
def test_summary_location(page: Page):
    page.goto(file_url("summary.html"))

    top_nav = page.locator(".top-nav")
    box = top_nav.bounding_box()

    x = box["x"]
    y = box["y"]

    expected_x = 20
    expected_y = 20

    assert x == expected_x
    assert y == expected_y


def test_summary_orientation(page: Page):
    page.goto(file_url("summary.html"))

    flex_container = page.locator(".filter-container")

    direction = flex_container.evaluate("el => getComputedStyle(el).flexDirection")
    assert direction == "row"

def test_summary_size(page: Page):
    page.goto(file_url("summary.html"))

    scrollbox = page.locator(".scrollbox")
    scrollbox_height = scrollbox.evaluate("el => getComputedStyle(el).height")

    expected_height = "200px"

    assert scrollbox_height == expected_height
```

Figure 9 : Snippet of test cases for testing the 3 changes made to the summary page

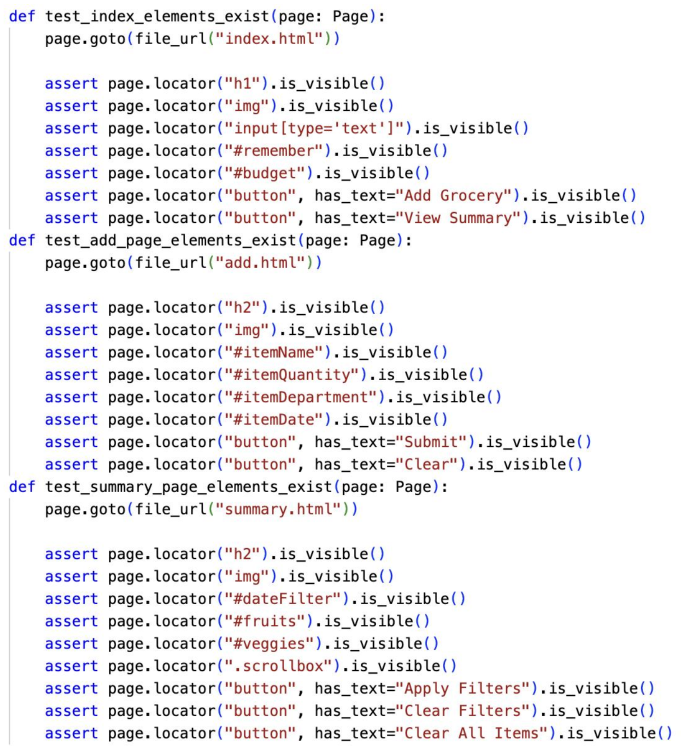
The below test case is used to check the path change from index to add and add to summary pages.



```python
def test_full_navigation_flow(page: Page):

    page.goto(file_url("index.html"))
    assert "index.html" in page.url

    add_button = page.locator("button:has-text('Add Grocery')")
    assert add_button.count() > 0
    add_button.first.click()

    assert "add.html" in page.url

    summary_button = page.locator("button:has-text('Summary')")
    assert summary_button.count() > 0
    summary_button.first.click()

    assert "summary.html" in page.url
```

Figure 10 : Snippet of test cases for testing the path flow

The below test cases checks for the existence of GUI elements(h1 tag, h2 tag, image, buttons, drop box, drop elements, filter) in index, add, summary pages.

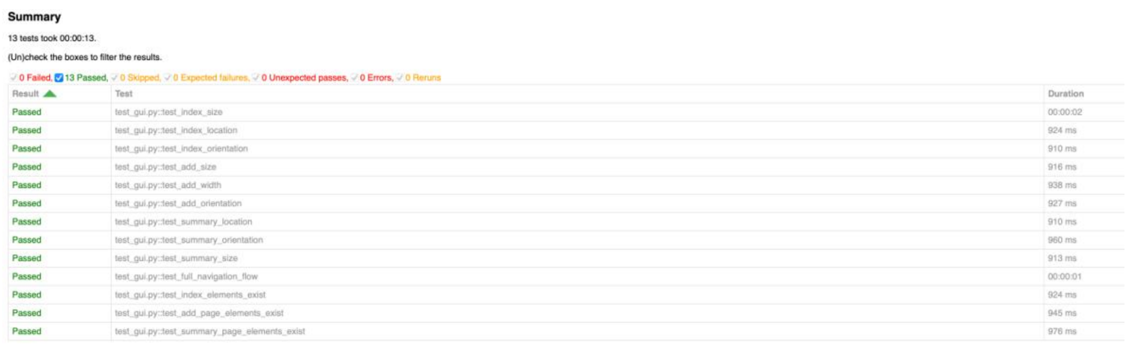


```python
def test_index_elements_exist(page: Page):
    page.goto(file_url("index.html"))

    assert page.locator("h1").is_visible()
    assert page.locator("img").is_visible()
    assert page.locator("input[type='text']").is_visible()
    assert page.locator("#remember").is_visible()
    assert page.locator("#budget").is_visible()
    assert page.locator("button", has_text="Add Grocery").is_visible()
    assert page.locator("button", has_text="View Summary").is_visible()
def test_add_page_elements_exist(page: Page):
    page.goto(file_url("add.html"))

    assert page.locator("h2").is_visible()
    assert page.locator("img").is_visible()
    assert page.locator("#itemName").is_visible()
    assert page.locator("#itemQuantity").is_visible()
    assert page.locator("#itemDepartment").is_visible()
    assert page.locator("#itemDate").is_visible()
    assert page.locator("button", has_text="Submit").is_visible()
    assert page.locator("button", has_text="Clear").is_visible()
def test_summary_page_elements_exist(page: Page):
    page.goto(file_url("summary.html"))

    assert page.locator("h2").is_visible()
    assert page.locator("img").is_visible()
    assert page.locator("#dateFilter").is_visible()
    assert page.locator("#fruits").is_visible()
    assert page.locator("#veggies").is_visible()
    assert page.locator(".scrollbox").is_visible()
    assert page.locator("button", has_text="Apply Filters").is_visible()
    assert page.locator("button", has_text="Clear Filters").is_visible()
    assert page.locator("button", has_text="Clear All Items").is_visible()
```

Figure 11 : Snippet of test cases for checking the existence of GUI elements

Task 4 - Explaination of test results

The version 1 of the GUI application passed all 13 test cases(Figure 7 - 11) which checks the existence of elements in index, add, summary pages and checks for any changes made to size, location and orientation. There is no failure in test cases as changes in size, location and orientation are made to version 2 of GUI application.



Summary

13 tests took 00:00:13.

(Un)check the boxes to filter the results.

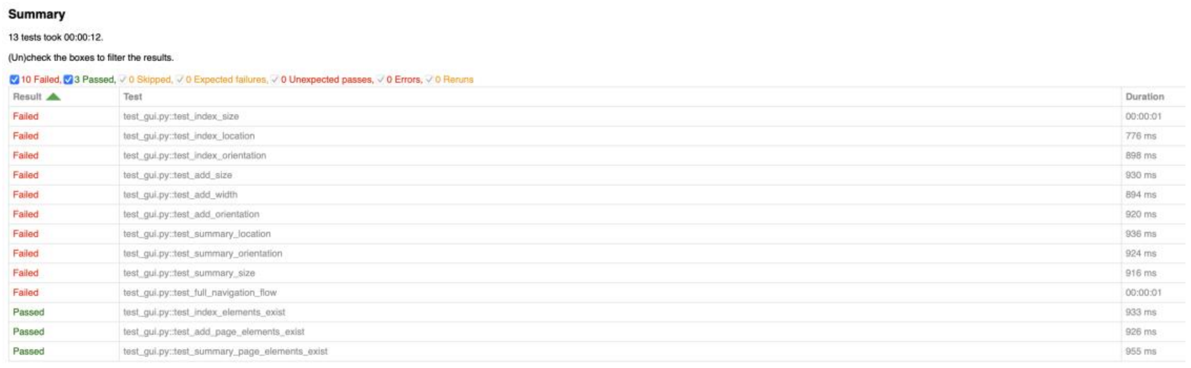
✓ 0 Failed, ☑ 13 Passed, ✓ 0 Skipped, ✓ 0 Expected failures, ✓ 0 Unexpected passes, ✓ 0 Errors, ✓ 0 Reruns

Result ▲	Test	Duration
Passed	test_gui.py::test_index_size	00:00:02
Passed	test_gui.py::test_index_location	924 ms
Passed	test_gui.py::test_index_orientation	910 ms
Passed	test_gui.py::test_add_size	916 ms
Passed	test_gui.py::test_add_width	938 ms
Passed	test_gui.py::test_add_orientation	927 ms
Passed	test_gui.py::test_summary_location	910 ms
Passed	test_gui.py::test_summary_orientation	960 ms
Passed	test_gui.py::test_summary_size	913 ms
Passed	test_gui.py::test_full_navigation_flow	00:00:01
Passed	test_gui.py::test_index_elements_exist	924 ms
Passed	test_gui.py::test_add_page_elements_exist	945 ms
Passed	test_gui.py::test_summary_page_elements_exist	976 ms

Figure 12 : Snippet of test results for version 1

We can observe that only 3 test cases which can be seen in Figure 10 which checks for the existence of GUI elements, but not their size, location or orientation have passed, while the changes in size, location and orientation made to index, add, summary pages have failed.

The version 2 failed test case test_index_size as checks for width of image in index page to be 300, but the true image size is 200, test_index_location failed as it got 0 x coordinate instead of 440, test_index_orientation failed as the orientation of the form shoud in row but got column. test_add_size failed as the width of image in add page is 220 but should be 300, test_add_width failed as the width of the form shoud be 400px instead the form was 600px, test_add_orientation failed as the orientation of form in add page is in column but it should be in row. test_summary_location failed as the x-coordinate should be 20 but the form was at 956.9921875, test_summary_orientation failed as the direction of the form should be in row but the form was in column, test_summary_size failed as the height of the form should be 200 px but got 300px. test_full_navigation_flow failed as the path from index --> add -->summary has been change to index --> add --> index --> summary.



Summary

13 tests took 00:00:12.

(Un)check the boxes to filter the results.

✓ 10 Failed, ✓ 3 Passed, ✓ 0 Skipped, ✓ 0 Expected failures, ✓ 0 Unexpected passes, ✓ 0 Errors, ✓ 0 Reruns

Result	Test	Duration
Failed	test_gui.py::test_index_size	00:00:01
Failed	test_gui.py::test_index_location	776 ms
Failed	test_gui.py::test_index_orientation	898 ms
Failed	test_gui.py::test_add_size	930 ms
Failed	test_gui.py::test_add_width	894 ms
Failed	test_gui.py::test_add_orientation	920 ms
Failed	test_gui.py::test_summary_location	936 ms
Failed	test_gui.py::test_summary_orientation	924 ms
Failed	test_gui.py::test_summary_size	916 ms
Failed	test_gui.py::test_full_navigation_flow	00:00:01
Passed	test_gui.py::test_index_elements_exist	933 ms
Passed	test_gui.py::test_add_page_elements_exist	926 ms
Passed	test_gui.py::test_summary_page_elements_exist	955 ms

Figure 13 : Snippet of test results for version 2

Task 5 - Assessment of the tool

a. set of features and functionalities provided - Playwright is simple to use, easy to install, integrates well with pytest to generate HTML based test report

b. type of coverage - Checks for the existence of most of all the GUI elements, path navigation from pages, validates precise data needed.

c. reuse of test cases - The test cases generated for one version can be used to all the other versions with, simple change in the path to the files.

d. test results produced - Integrating with pytest, conviniently generates a well represented HTML page with all the failed and passed test cases. The terminal version of test report with out integrating with pytest is convinient as well.

e. ease of usage - The syntax for using playwright is very simple and easy to grasp.

f. type of GUI elements that can be tested - Mostly all of the elements can be tested which include buttons, forms, checkboxes, and more.

References

Johnson, E. (2023, November 20). What is Playwright? Its features, advantages, and disadvantages. Test Automation Tools. https://testautomationtools.dev/playwright-overview/

Saini, M. (2024, October 12). Why I chose Playwright for test automation. The Testing Hub. https://medium.com/the-testing-hub/why-i-chose-playwright-for-test-automation-f4c1dbdaf7be