Now here in the pages and features

Login page

Registration page

ACCOUNT SETTINGS page (Update Account Info)

My Products page

Add product page/Update Product page [same page structure, just title change]

Browse products page

view products page

Delete product (just feature, present in my product page)

Buy/Rent product (just feature, present in view product page)

Prompt for login:
@/Users/maisha-chowa/Automation-Project/Automation-task-Sazim

goal: generate test case for login feature(data-driven approach)

step 1: using playwright test agents planner to capture login page elements @pages/login_page.py

step 2: using playwright test agents generator to generate test cases for login page @tests/test_login.py -> during login test case generation handle combination of positve, negative tests cases, ui validation

step 3: also generates data for login tests @test_data/login.json

step 4: using playwright test agents healer executes the test suite and automatically repairs failing tests.

strictly follow pom structure.
After successful login, it will redirect to https://ehsanur-rahman-sazim.github.io/my-products

Prompt for login 2:

goals: add more negative test cases

negative_wrong_password

negative_unregistered_email

negative_invalid_email_format

blank_email

blank_password

blank_both

generate data for the above test cases @test_data/login.json

and add the test cases in the @tests/test_login.py as well

Prompt for Registration:

@/Users/maisha-chowa/Automation-Project/Automation-task-Sazim/

goal: generate test case for register feature(data-driven approach)

After visting to base url just below login form there is a Sign Up btn to navigate to Registration page

step 1: using playwright test agents planner to capture Registration page elements @pages/registration_page.py

step 2: using playwright test agents generator to generate test cases for registration page @tests/test_registration.py -> during registration test case generation handle all combination of positve, negative tests cases, ui validation

step 3: also generates data for registration tests @test_data/registration.json

step 4: using playwright test agents healer executes the test suite and automatically repairs failing tests.
After successful registration, it will redirect to
https://ehsanur-rahman-sazim.github.io/my-products

strictly follow pom structure and project structure. for reference follow the @tests/test_login.py @test_data/login.json @pages/login_page.py

Prompt for Update Account
goal 1: store positive login info in auth state to reuse now generate a positive login test function,

and store the auth state so that i don't to login to visit my-products page every time. then storing the login info in the auth state, to reuse. use already existed postive login function in @tests/test_login.py file or create a new file in utils folder goal 2: generate test case Update Account info(data-driven approach) After login -> redirect to my-products url -> then click ACCOUNT SETTINGS from the navbar -> it will redirect to BASE_URL/account-settings step 1: using playwright test agents planner to capture ACCOUNT SETTINGS page elements @pages/account_settings_page.py step 2: using playwright test agents generator to generate test cases for ACCOUNT SETTINGS page @tests/test_account_settings.py -> during update ACCOUNT SETTINGS test case generation handle all combination of positve, negative tests cases, ui validation for positive test data - after clicking the update btn - > click the My Products btn -> then again click the ACCOUNT SETTINGS btn -> to validate account updated with new data step 3: also generates data for registration tests @test_data/account_settings.json step 4: using playwright test agents healer executes the test suite and automatically repairs failing tests. strictly follow pom structure and project structure. for reference follow the @tests/test_registration.py @test_data/registration.json @pages/registration_page.py

Fixing add product error

@test_data/add_update_product.json there is 6 input field, it's not handling the rent duration dropdown,

that's why positive test is failing.

Goal: Add the missing input field in product

step 1: using playwright test agents planner to capture Add product page elements @pages/add_update_product_page.py

step 2: using playwright test agents generator to generate test cases for add product page @tests/test_add_update_product.py -> during registration test case generation handle all combination of positve, negative tests cases, ui validation

step 3: also update data for create products tests @test_data/add_update_product.json

step 4: using playwright test agents healer executes the test suite and automatically repairs failing tests.

step 5: After successful add product, it will redirect to https://ehsanur-rahman-sazim.github.io/my-products, check in this page product added or not

after fixing it , only run add Product test

Prompt for add product

Goal: Generate test cases for Add product

step 1: After login it will redirect to my product page click the add product btn

step 2: using playwright test agents planner to capture Add product page elements @pages/add_update_product_page.py

step 3: using playwright test agents generator to generate test cases for add product page @tests/test_add_update_product.py -> during registration test case generation handle all combination of positve, negative tests cases, ui validation

step 4: also generates data for create products tests @test_data/add_update_product.json

step 5: using playwright test agents healer executes the test suite and automatically repairs failing tests.

step 6: After successful add product, it will redirect to https://ehsanur-rahman-sazim.github.io/my-products, check in this page product added or not

strictly follow pom structure and project structure. for reference follow the @tests/test_registration.py @pages/registration_page.py @test_data/registration.json

Prompt for update product

Goal: Generate test cases for update product, that are already created.

step 1: After login it will redirect to my product page, open the product that is already created, searched with title present in the @test_data/add_update_product.json positive test date

step 2: using playwright test agents planner to capture update product page elements @pages/add_update_product_page.py [same as the add product page-just title changed]

step 3: using playwright test agents generator to generate test cases for add product page @tests/test_add_update_product.py -> during update test case generation handle all combination of positve, negative tests cases, ui validation

step 4: also generates data for update products tests @test_data/add_update_product.json

step 5: using playwright test agents healer executes the test suite and automatically repairs failing tests.

step 6: After successful update product, it will redirect to BASE_URL/my-products, check in this page product added or not

strictly follow pom structure and project structure. for reference follow the @pages/add_update_product_page.py @test_data/add_update_product.json @tests/test_add_update_product.py

Goal: Generate test cases for browse product step
1: After login it will redirect to my product page, click the Browse products btn from navbar step 2: using playwright test agents planner to capture Browse products page elements @pages/browse_products_page.py . also store the all the 7 products temporay, sothat it can be used to create test data step 3: using playwright test agents generator to generate test cases for browse products page 4 different category filter options available search by title, search by category, buy with max-min, rent with max min @tests/test_browse_products.py -> during browse test case generation handle all combination of positve, negative tests cases, ui validation. after every filter tests, click the clear btn then start the next test step 4: also generates data for search combination tests 4 different category filter options available search by title, search by category, buy with max-min, rent with max min, @test_data/browse_products.json step 5: using playwright test agents healer executes the test suite and automatically repairs failing tests. strictly follow pom structure and project structure.

4 tags

Available
Sold
Rented
You Own the product

If available, then Buy and Rent btn is there
If sold, then Buy and Rent btn is not there
If rented, then Buy and Rent btn is not there
If You Own the product, then Buy and Rent btn is not there
If You both Available, Own the product, then Buy and Rent btn is not there

If Available, click the Buy btn, it shows modal with yes and not btn,
    If click, tag should changed to sold,
    If click, tag should remains to available

If Available, click the Rented btn, it shows modal with start and not end date, also with Book rent and Cancel btn

If give valid rented date [from today to upcoming date] and click Book Rent, tag should changed to Rented,

If give valid rented duration max 1 week and click Book Rent, tag should changed to Rented,

If give invalid date rented date [past date, Start Date = End Date,End Date Before Start Date, Empty Inputs] -> can't click Book Rent, tag should remain Available,

If give invalid date rented duration [more then 1 week]-> can't click Book Rent, tag should remain Available,

If click cancel, tag should remains to available

Owner Can't Edit/Delete the product, Once it rented or sold