Visual Testing using Cucumber.JS, Playwright, JavaScript, and Pixel Match

- Create a new folder, **VisualTestingPlaywright,** this will be our Project root
- Open this folder via VSCode
- In the VSCode terminal initiate a new NPM project – **npm init.** This will create a package.json file in the **VisualTestingPlaywright** folder.
- Install Cucumber -- **npm install --save-dev @cucumber/cucumber**
- Install Playwright -- **npm install --save-dev playwright**
- Install Playwright Test -- **npm install --save-dev @playwright/test**
- Install Chai Assertion Library -- **npm install --save-dev chai**
- Install Pixel Match -- **npm install --save-dev pixelmatch**
- Install PNGJS -- **npm install --save-dev pngjs**
- Install FS Jetpack -- **npm install --save-dev fs-jetpack**
- Install adm-zip -- **npm install --save-dev adm-zip**

Folders Creation:

Create the below folders

- VisualTestingPlaywright/features
- VisualTestingPlaywright/features/images/
- VisualTestingPlaywright/features/images/base_images
- VisualTestingPlaywright/features/images/test_images
- VisualTestingPlaywright/features/reports/
- VisualTestingPlaywright/features/step_definitions/
- VisualTestingPlaywright/features/support/

The base_images folder will contain the Base images and ZIP's of previous Base images. The test_images folder will contain the Current images taken as the test is running. All the feature files will be in the features folder and step definitions will be in the VisualTestingPlaywright/step_definitions folder.

Running commands:

- **.\node_modules\.bin\cucumber-js .\features\VisualTesting.feature --format html:.\features\reports\Report.html**
- **.\node_modules\.bin\cucumber-js --tags "@VisualTests" --format html:.\features\reports\Report.html**

Test summary:

Visual testing involves verifying whether a webpage is rendered correctly. It's different from Functional testing as Visual testing ensures that the look of a webpage is not compromised. For

example, we have a button in a webpage, if the icon for the button is not loaded functional tests will not catch it as the button works properly, but visual tests will catch it.

Visual tests compare the current webpage with an image of an existing image, called a base image, to compare and if there are any differences in the images, it flags it.

In this repo, we have 3 tests:

- BaseImageTaker.feature
- VisualTesting.feature
- VisualTestingWithErrors.feature

The tests are done for the below pages in https://www.saucedemo.com/

- Login Page
- Product Page
- Specific Product Page
- Cart Page
- Checkout Page
- Checkout Confirmation Page
- Checkout Finish Page

The BaseImageTaker.feature takes the Base images of the above pages and should only be run when there's any intended change in the above pages, as it updates the images that we use to compare against.

The VisualTesting.feature has the regular scenarios which will check all the above pages and compare it with the base images. It uses the normal login.

The VisualTestingWithErrors.feature has the same scenarios as VisualTesting.feature which will check all the above pages and compare it with the base images. It uses the visual_user login, it’s a login provided by Sauce Demo, which has some UI issues. Using this user login, we can validate the accuracy of our tests.

Folder Structure: