**Playwright Automation Tutorial**

**Introduction**

Playwright is test automation framework that can be used for

1. API testing
2. Modern web apps
3. Web apps on mobile

Language Supported- JS, TS, Java, Python and C#

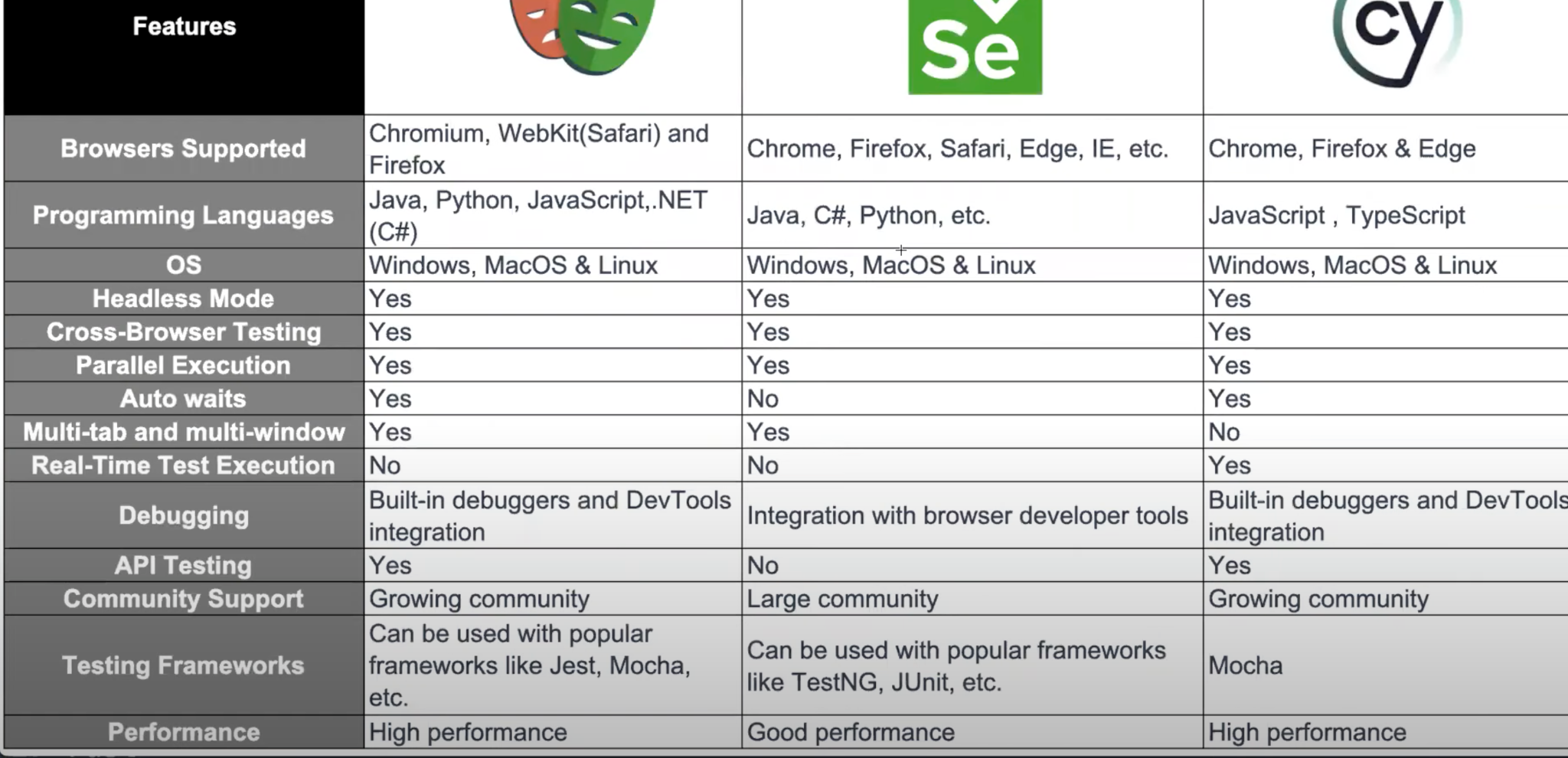
Browser Supported- Chromium based, Webkit, firefox [headed & Headless]

OS supported- Windows, Linux, Mac, CLI

**Features of PW**

1. Free & Open Source.
2. Multi browser, Multi Languages and Multi-OS
3. Setup & Config is easy
4. Can automate functional, API and accessibility testing(by third party plugins)
5. Built-In reporters, Custom reports
6. CI/CD & Docker integration
7. Supports parallel testing
8. Auto-wait
9. Built-in assertions
10. Less Flaky test
11. Multi-tab and Multi-window support
12. Iframe support and shadow-dom support
13. Support parameterizations for DDT
14. Emulate multiple devices
15. Faster than other automation tools
16. Codegen for generating test code in diff languages
17. Playwright inspector
18. Growing community

**Playwright vs Selenium vs Cypress**



**Playwright Installation**

Prerequisites: Node Js, npm , VS code

1. Create a new folder for your project
2. Open this folder from VS code
3. Install playwright using terminal npm init playwright@latest .
4. Follow the prompts with default settings. I have used javascript as language
5. Check using npm playwright -v

This will finish configuring playwright for your project and create some folders and config files that will be required for your testing

Package-json- Project management file to manage project dependencies just like POM.xml.

Playwright.config.js- Playwright related configurations like test folder, Parallel execution, browser name etc. Check the file for more details.

Test folder is created to store all tests

This can also be done using VS code directly.

1. Open the folder.
2. Go to extensions
3. Search Playwright Test for VS Code extension
4. Click Install
5. Go to folder
6. Click View-Command pallett- Search for Playwright
7. Install PW and follow prompts

**Playwright Common Commands**

npx playwright test —Runs the end-to-end tests.

npx playwright test --ui —Starts the interactive UI mode.

npx playwright test --project=chromium —Runs the tests only on Desktop Chrome.

npx playwright test example —Runs the tests in a specific file.

npx playwright test --debug —Runs the tests in debug mode.

npx playwright codegen –Auto generate tests with Codegen.

**Create Playwright Tests**

**We can create test using**

1. **By Writing our own script**
2. Open VS code editor
3. Click a new file in test folder With name as “Homepage.spec.js”

const {test, expect}= require ('@playwright/test')

test('Homepage',async({page})=>

{

await page.goto("https://www.demoblaze.com/index.html")

const pagetitle= await page.title()

console.log('Page title is', pagetitle)

await expect(page).toHaveTitle('STORE')

await expect(page).toHaveURL("https://www.demoblaze.com/index.html")

await page.close()

}

)

//async-await is to handle the async behavior of JS and to make sure that test execution happens in designated order or to handle JS promise. page.goto will wait until the page is loaded

//{test,expect} are imported modules required for test implementation

//page is a fixture that contains functions that are required for interacting with a web page

//Instead of const {test, expect}= require ('@playwright/test') you can also use import {test,expect} from '@playwright/test'

1. **CodeGen**

Run “npx playwright codegen” in the terminal : This will open the browser along with the PW inspector. You can run your test manually on the browser and the script will be recorded by itself in the PW inspector. You can also use “Pick Locator” to pick a locator for any element. You can then export this test code in different languages such as Java, JS, Python or .Net

Run npx playwright codegen –output tests/codegen.spec.js to generate test file as well in your test folder. Read PW doc for more details

**Running the Test**

Run this command in terminal

Headless mode for all tests in test folder- npx playwright test

Headed mode for all tests in test folder- npx playwright test –headed

To run a specific spec file- npx playwright test “testfilename.spec.js”

To run on a specific browser - npx playwright test –project=chromium

In debug mode- npx playwright test –debug

**Seeing the Test Reports**

To open last run report in HTML- npx playwright show-report

**Locating Web Elements in Playwright ( we can use selector hub to get locators)**

We can locate using a unique Property, CSS or Xpath of the element present in DOM.

Common syntax for executing a method on an element is

**link/button**

await page.locator(‘element locator’).click()

await page.click(‘locator of element’)

**inputbox**

.type() and .fill() are written like

await page.locator(‘locator’).type(‘value to be typed’)

await page.locator(‘locator’).fill(‘value to be typed’)

await page.type(‘Locator’,’Value to be typed’)

await page.fill(‘Locator’,’Value to be typed’)

*\*type() and .fill() are the same. It's just .type() mimics user keystrokes whereas .fill() fills the input at once.*

import {test,expect} from '@playwright/test'

test('Locators', async({page})=>

{

await page.goto('https://www.demoblaze.com/index.html')

await page.click('id=login2') // or await page.locator('id=login2').click()

await page.fill('#loginusername','pavanol') // or await page.locator('#loginusername').fill('pavanol')

await page.fill("input[id='loginpassword']", 'test@123')

await page.click("//button[normalize-space()='Log in']")

const logoutlink= await page.locator("//button[normalize-space()='Log out']")

await expect(logoutlink).toBeVisible

await page.close()

}

)

**Locate Multiple web elements**

I am declaring a const element and then getting all the elements using below

const element= await page.$$(‘locator’)

Below is a code to get number of elements as well as print text of the element

import { test, expect } from '@playwright/test';

test('LocateMultipleElements', async ({ page }) => {

// Navigate to the demo website

await page.goto('https://www.demoblaze.com/index.html');

// Find all anchor elements on the page

const links = await page.$$('a');

// Output the number of links found

console.log('Number of links:', links.length);

//get text from all the links

for(const link of links)

{const linktext=await link.textContent()

console.log(linktext)

}

const products = await page.$$("//div[@id='tbodyid']//div/h4/a");

// Output the number of products found

console.log('Number of products:', products.length);

//get text from all the products

for(const product of products)

{const productname=await product.textContent()

console.log(productname)

}

// Close the page after the test completes

await page.close();

});

**Playwright Built- In Locators**

page.getByRole() to locate by explicit and implicit accessibility attributes. (button, link or actionable elements)

page.getByText() to locate by text content.

page.getByLabel() to locate a form controlled by associated label's text. page.getByPlaceholder() to locate an input by placeholder.

page.getByAltText() to locate an element, usually image, by its text alternative.

page.getByTitle() to locate an element by its title attribute.

page.getByTestId() to locate an element based on its data-testid attribute (other attributes can be configured).

Below is a test case with some of the built in locators. For other locators that are not in test, please refer to the PW website.

**import { test, expect } from '@playwright/test'**

**test('BuiltInLocators', async({page})=>**

**{**

**await page.goto('https://opensource-demo.orangehrmlive.com/web/index.php/auth/login')**

**const logo= await page.getByAltText('company-branding')**

**await expect(logo).toBeVisible()**

**await page.getByPlaceholder('Username').fill('Admin')**

**await page.getByPlaceholder('Password').fill('admin123')**

**await page.getByRole('button',{type: 'submit'}).click()**

**const username=await page.locator('//p[@class="oxd-userdropdown-name"]').textContent()**

**await expect (await page.getByText(username)).toBeVisible()**

**await page.close()**

**}**

**)**

**Playwright Assertions**

There are two kinds of Assertion- Soft and Hard Assertions. Soft assertion does not terminate the execution whereas Hard assertions terminate the execution if the assertion fails.

There are a lot of assertions which can be learnt directly from the PW website. Here are 10 frequently used hard assertions.

To Negate these assertions you can use .not.assertion to verify negation logic for the element for e.g. not.toHaveUrl().

**import {test, expect} from '@playwright/test'**

**import express from 'express'**

**test('assertions', async({page})=>**

**{**

**await page.goto('https://demo.nopcommerce.com/register')**

**await expect (page).toHaveURL('https://demo.nopcommerce.com/register')**

**await expect (page).toHaveTitle('nopCommerce demo store. Register')**

**const logo = await page.locator('.header-logo')**

**await expect (logo).toBeVisible()**

**const searchbox = page.locator('#small-searchterms')**

**await expect (searchbox).toBeEnabled()**

**//similarly we can do toBeDisabled()**

**const maleradiobutton =page.locator('#gender-male')**

**await maleradiobutton.click()**

**expect(maleradiobutton).toBeChecked()**

**const Newsletter = await page.locator('#Newsletter')**

**await expect(Newsletter).toBeChecked()**

**const registerbutton = await page.locator('#register-button')**

**await expect(registerbutton).toHaveAttribute('type','submit')**

**const pagetitle= await page.locator('.page-title h1')**

**await expect(pagetitle).toHaveText('Register')**

**await expect(pagetitle).toContainText('iste')**

**const email = await page.locator('#Email')**

**await email.fill('textinput@test.com')**

**await expect(email).toHaveValue('textinput@test.com')**

**const months = await page.locator('select[name="DateOfBirthMonth"] option')**

**await expect(months).toHaveCount(13)**

**}**

**)**

**Soft Assertions** can be written in the same way as hard assertions. It's just you need to suffix except with .**soft.**  Like in below test, the middle assertion is failing but the script will still execute for all the assertions.

**import {test,expect} from '@playwright/test'**

**test('Soft Assertions',async({page})=>**

**{**

**await page.goto('https://www.demoblaze.com/index.html')**

**await expect.soft(page.locator('.navbar-brand')).toBeVisible()**

**await expect.soft(page).toHaveTitle('STORE123')**

**await expect.soft(page).toHaveURL('https://www.demoblaze.com/index.html')**

**})**

**WebElements- Input Box**

**import {test,expect} from '@playwright/test'**

**test('InputBox', async({page})=>**

**{**

**await page.goto('https://www.demoblaze.com/index.html')**

**await page.click('id=login2') // or await page.locator('id=login2').click()**

**await page.fill('#loginusername','pavanol') // or await page.locator('#loginusername').fill('pavanol')**

**await expect(page.locator("input[id='loginpassword']")).toBeVisible()**

**await expect(page.locator("input[id='loginpassword']")).toBeEmpty()**

**await expect(page.locator("input[id='loginpassword']")).toBeEditable()**

**await expect(page.locator("input[id='loginpassword']")).toBeEnabled()**

**await page.locator("input[id='loginpassword']").fill( 'test@123') //can also use .type()**

**await page.click("//button[normalize-space()='Log in']")**

**const logoutlink= await page.locator("//button[normalize-space()='Log out']")**

**await expect(logoutlink).toBeVisible**

**await page.waitForTimeout(5000)//if you want script to pause execution for some time. in this instance, its 5 seconds**

**await page.close()**

**}**

**)**

**WebElements- Radio Buttons**

**import{test,expect} from '@playwright/test'**

**test('',async({page})=>**

**{**

**await page.goto('https://www.dummyticket.com/dummy-ticket-for-visa-application/')**

**await page.waitForSelector('#product\_551', { state: 'visible' }) //.waitForSelector(locator) can be used to put wait for a specific element to be visible**

**const DummyHotelBooking=await page.locator('#product\_551')**

**await expect(DummyHotelBooking).not.toBeChecked()**

**await DummyHotelBooking.check()**

**await expect(DummyHotelBooking).toBeChecked()**

**await expect ((DummyHotelBooking).isChecked()).toBeTruthy()**

**//Uncheck will not work until you click on some other radio button. So basically, uncheck is not supported for radio buttons**

**}**

**)**

**WebElements- CheckBoxes**

**import {test,expect} from '@playwright/test'**

**import exp from 'constants'**

**test('Checkbox', async({page})=>**

**{**

**await page.goto('https://the-internet.herokuapp.com/checkboxes')**

**await expect(page.locator('input:nth-child(1)')).not.toBeChecked()**

**await expect(page.locator('input:nth-child(3)')).toBeChecked()**

**await page.locator('input:nth-child(1)').check() //check one checkbox**

**await expect(page.locator('input:nth-child(1)')).toBeChecked()**

**await expect(page.locator('input:nth-child(3)')).toBeChecked()**

**await page.locator('input:nth-child(3)').uncheck() //uncheck one checkbox**

**await expect(page.locator('input:nth-child(1)')).toBeChecked()**

**await expect(page.locator('input:nth-child(3)')).not.toBeChecked()**

**await page.locator('input:nth-child(1)').check() //Check multiple checkboxes**

**await page.locator('input:nth-child(3)').check()**

**await expect(page.locator('input:nth-child(1)')).toBeChecked()**

**await expect(page.locator('input:nth-child(3)')).toBeChecked()**

**await page.locator('input:nth-child(1)').uncheck() //uncheck multiple checkboxes**

**await page.locator('input:nth-child(3)').uncheck()**

**await expect(page.locator('input:nth-child(1)')).not.toBeChecked()**

**await expect(page.locator('input:nth-child(3)')).not.toBeChecked()**

**await page.locator('input:nth-child(n)').first().check() //Checking first check box**

**await expect(page.locator('input:nth-child(n)').first()).toBeChecked()**

**await page.locator('input:nth-child(n)').last().check() //Checking last checkbox**

**await expect(page.locator('input:nth-child(n)').last()).toBeChecked()**

**await page.locator('input:nth-child(n)').nth(0).check() //Checking checkbox using index value**

**await expect(page.locator('input:nth-child(n)').nth(0)).toBeChecked()**

**await page.locator('input:nth-child(n)').nth(1).check()**

**await expect(page.locator('input:nth-child(n)').nth(1)).toBeChecked()**

**//Checking using array and for loop**

**await page.goto('https://the-internet.herokuapp.com/checkboxes')**

**const checkboxes = page.locator('input[type="checkbox"]');**

**const count = await checkboxes.count(); // Get the number of checkboxes**

**for (let i = 0; i < count; i++) {**

**await checkboxes.nth(i).check(); // Check each checkbox**

**await expect(checkboxes.nth(i)).toBeChecked(); // Verify it is checked**

**}**

**// Optional: Uncheck all checkboxes**

**for (let i = 0; i < count; i++) {**

**await checkboxes.nth(i).uncheck(); // Uncheck each checkbox**

**await expect(checkboxes.nth(i)).not.toBeChecked(); // Verify it is unchecked**

**}**

**await page.close()**

**}**

**)**

**WebElement- DropDowns**

**import {test,expect} from '@playwright/test'**

**import exp from 'constants'**

**test('dropdown1', async({page})=>**

**{**

**//bootstap dropdown by label**

**await page.goto('https://testautomationpractice.blogspot.com/')**

**await page.locator('#country').selectOption({label:'India'})**

**await expect(page.locator('#country')).toHaveValue('india')**

**//bootstap dropdown by visible text**

**await page.goto('https://testautomationpractice.blogspot.com/')**

**await page.locator('#country').selectOption('India')**

**await expect(page.locator('#country')).toHaveValue('india')**

**//bootstap dropdown by dropdown value code**

**await page.locator('#country').selectOption({value:'uk'})**

**const country=await page.locator('#country').locator('option:checked').textContent()**

**await expect (country).toBe('United Kingdom')**

**//bootstap dropdown by index**

**await page.locator('#country').selectOption({index:1})**

**await expect(page.locator('#country')).toHaveValue('canada')**

**//bootstap dropdown without locator**

**await page.selectOption('#country',{index:1})**

**await expect(page.locator('#country')).toHaveValue('canada')**

**//Assert number of options in dropdown**

**const options =await page.locator('#country option') //Concatenating option will track list of options**

**expect(options).toHaveCount(10)**

**//Checking presence of an option**

**const content=await page.locator('#country').textContent()**

**await expect(content.includes('India')).toBeTruthy()**

**// Checking presence of an option using loop**

**const content2 = await page.$$('#country option'); // Get all option elements**

**let status = false;**

**for (const option of content2) {**

**const optionText = await option.textContent(); // Get the text of each option**

**if (optionText.includes('France')) {**

**status = true; // Set status to true if 'France' is found**

**break; // Exit loop if found**

**}**

**}**

**expect(status).toBeTruthy(); // Assert that status is true**

**// Selecting with for loop**

**const content3 = await page.$$('#country option'); // Get all option elements**

**for (const option of content3)**

**{**

**const optionText1 = await option.textContent(); // Get the text of each option**

**if (optionText1.includes('France'))**

**{**

**await page.selectOption('#country',optionText1)**

**break; // Exit loop if found**

**}**

**}**

**await expect(page.locator('#country')).toHaveValue('france')**

**//suggestive drop down after text input - to be done**

**//predictive dropdown such as google search - to be done**

**}**

**)**

**WebElement- Multiselect Dropdown**

**import{test,expect} from '@playwright/test'**

**test('Multiselect',async({page})=>**

**{**

**await page.goto('https://testautomationpractice.blogspot.com/')**

**await page.locator('#colors').selectOption(['Blue','Red','Yellow']) //or await page.selectOption('#colors',['Blue','Red','Yellow'])**

**//Assert number of options**

**const options= await page.locator('#colors option')**

**expect(options).toHaveCount(5)**

**//Assert using javascript**

**const jsoptions= await page.$$('#colors option')**

**console.log(jsoptions.length)**

**await expect(jsoptions.length).toBe(5)**

**//Assert presence of option**

**const content= await page.locator('#colors').textContent()**

**await expect(content.includes('Blue')).toBeTruthy()**

**await expect(content.includes('Black')).toBeFalsy()**

**}**

**)**

**WebElement- BootStrap Dropdown (dropdowns with no select tag)**

**import{test,expect} from '@playwright/test'**

**test('BootStrap Dropdown',async({page})=>**

**{**

**await page.goto('https://jquery-az.com/boots/demo.php?ex=63.0\_2')**

**await page.waitForTimeout(5000)**

**await page.locator('.multiselect').click()**

**const options =await page.locator('ul>li label input')**

**await expect(options).toHaveCount(11)**

**const optionslength =await page.$$('ul>li label input')**

**await expect(optionslength.length).toBe(11)**

**const optionslabel =await page.$$('ul>li label')**

**for (let label of optionslabel)**

**{**

**let val=await label.textContent()**

**console.log(val)**

**if (val.includes('Angular')|| val.includes('Java'))**

**{**

**await label.click()**

**}**

**}**

**for (let label of optionslabel)**

**{**

**let val=await label.textContent()**

**console.log(val)**

**if (val.includes('HTML')|| val.includes('CSS')||val.includes('Angular'))**

**{**

**await label.click()**

**}**

**}**

**await page.waitForTimeout(5000)**

**}**

**)**

**WebElement- AutoSuggest Dropdown**

**import {test,expect} from '@playwright/test'**

**test('AutoSuggestDropDown',async({page})=>**

**{**

**await page.goto('https://www.redbus.in/')**

**await page.locator('#src').fill('Delhi')**

**await page.waitForSelector("//li[contains(@class,'sc-iwsKbI')]/div/text[1]")**

**const source= await page.$$("//li[contains(@class,'sc-iwsKbI')]/div/text[1]")**

**for (let option of source )**

**{**

**let location= await option.textContent()**

**console.log(location)**

**}**

**for (let option of source )**

**{**

**let location= await option.textContent()**

**if (location.includes('Majnu Ka Tilla'))**

**{**

**await option.click()**

**break**

**}**

**}**

**await page.waitForTimeout(5000)**

**}**

**)**

**WebElement- Hidden Dropdowns (using selector hub debugger)**

**import{test,expect} from '@playwright/test'**

**import { KeyObject } from 'crypto'**

**test('HiddenDropDown', async({page})=>**

**{**

**await page.goto('https://opensource-demo.orangehrmlive.com/web/index.php/auth/login')**

**await page.locator("[name='username']").fill('Admin')**

**await page.locator("[name='password']").fill('admin123')**

**await page.click("[type='submit']")**

**await page.click("//span[normalize-space()='PIM']")**

**await page.click(".oxd-icon.bi-caret-up-fill.oxd-select-text--arrow")//need to confirm**

**//using selector hub debugger function**

**await page.waitForTimeout(3000)**

**const options= await page.$$("//div[@role='listbox']//span")**

**for(const option of options)**

**{**

**const jobtitle =await option.textContent()**

**console.log(jobtitle)**

**if (jobtitle.includes('QA Engineer'))**

**{await option.click()}**

**break**

**}**

**await page.waitForTimeout(5000)**

**}**

**)**

**WebElement- Dialog box**

**import{test,expect} from '@playwright/test'**

**test.skip('Alert with OK', async({page})=>**

**{**

**await page.goto('https://testautomationpractice.blogspot.com/')**

**//dialog window handler**

**await page.on('dialog', async dialog=>**

**{**

**expect (dialog.type()).toContain('alert')**

**expect(dialog.message()).toContain('I am an alert box!')**

**await dialog.accept()**

**})**

**await page.click('//button[normalize-space()="Alert"]')**

**})**

**test.skip('Confirmation Dialog with OK and cancel', async({page})=>**

**{**

**await page.goto('https://testautomationpractice.blogspot.com/')**

**//dialog window handler**

**await page.on('dialog', async dialog=>**

**{**

**expect (dialog.type()).toContain('confirm')**

**expect(dialog.message()).toContain('Press a button!')**

**//await dialog.accept()//close by using OK button**

**await dialog.dismiss()//close by using Cancel button**

**})**

**await page.click('//button[normalize-space()="Confirm Box"]')**

**//await expect(page.locator('//p[@id="demo"]')).toHaveText('You pressed OK!')**

**await expect(page.locator('//p[@id="demo"]')).toHaveText('You pressed Cancel!')**

**await page.waitForTimeout(5000)**

**})**

**test('Prompt Dialog', async({page})=>**

**{**

**await page.goto('https://testautomationpractice.blogspot.com/')**

**//dialog window handler**

**await page.on('dialog', async dialog=>**

**{**

**expect (dialog.type()).toContain('prompt')**

**expect(dialog.message()).toContain('Please enter your name:')**

**expect(dialog.defaultValue()).toContain('Harry Potter')**

**await dialog.accept('Gaurav Kumar')//close by using OK button and providing input in text field**

**})**

**await page.click('//button[normalize-space()="Prompt"]')**

**//await expect(page.locator('//p[@id="demo"]')).toHaveText('You pressed OK!')**

**await expect(page.locator('//p[@id="demo"]')).toHaveText('Hello Gaurav Kumar! How are you today?')**

**await page.waitForTimeout(5000)**

**})**

**WebElement- Frames**

**import{test,expect} from '@playwright/test'**

**test.skip ('iFrame using Name or URL', async({page})=>**

**{**

**await page.goto('https://ui.vision/demo/webtest/frames/')**

**const allframes=await page.frames()**

**console.log(allframes.length)**

**//using name or url of the frame**

**//const frame1=await page.frame('framename') //if name is present**

**const frame1=await page.frame({url:'https://ui.vision/demo/webtest/frames/frame\_1.html'}) //using frame url**

**//await frame1.locator("[name='mytext1']").fill('Hello') //or below**

**await frame1.fill("[name='mytext1']",'Hello')**

**}**

**)**

**test ('iFrame using Frame Locator', async({page})=>**

**{**

**await page.goto('https://ui.vision/demo/webtest/frames/')**

**const frame=await page.frameLocator('frame[src="frame\_1.html"]')**

**await frame.locator("[name='mytext1']").fill('Hello')**

**await page.waitForTimeout(3000)**

**}**

**)**

**WebElement- Inner/Nested Frames**

**import {test,expect} from '@playwright/test'**

**test('NestedFrames', async({page})=>**

**{**

**await page.goto('https://ui.vision/demo/webtest/frames/')**

**const frame3=await page.frame({url:'https://ui.vision/demo/webtest/frames/frame\_3.html'})**

**await frame3.locator("input[name='mytext3']").fill('Testing')**

**//Nested Frame inside frame 3**

**const childframes=await frame3.childFrames()**

**await childframes[0].locator("//\*[@id='i5']/div[3]/div").check()**

**await page.waitForTimeout(5000)**

**}**

**)**

**WebElement Tables**

**import{test,expect} from '@playwright/test'**

**test.skip('HandlingTable',async({page})=>**

**{**

**await page.goto('https://testautomationpractice.blogspot.com/')**

**const table= await page.locator('#productTable')**

**const columns= await table.locator('thead tr th')**

**console.log(await columns.count())**

**await expect(await columns.count()).toBe(4)**

**const rows= await table.locator('tbody tr')**

**console.log(await rows.count())**

**await expect(await rows.count()).toBe(5)**

**//Seelct checkbox for product4 using .filter()**

**const matchedrow= rows.filter({**

**has: page.locator('td'),**

**hasText: 'Product 4'**

**})**

**await matchedrow.locator('input').check()**

**//Select multiple products by reusable function in JS. assertion is also written as part of function**

**await selectproduct(rows,page,'Product 3')**

**await selectproduct(rows,page,'Product 2')**

**await page.waitForTimeout(5000)**

**}**

**)**

**test('Pagination',async({page})=>**

**{**

**//print all prod details**

**await page.goto('https://testautomationpractice.blogspot.com/')**

**const table= await page.locator('#productTable')**

**const columns= await table.locator('thead tr th')**

**const rows= await table.locator('tbody tr')**

**//print data from first page**

**for (let i=0;i<await rows.count();i++)**

**{**

**const currentrow=rows.nth(i)**

**const cells=rows.locator('td')**

**for (let j=0;j<await cells.count()-1;j++)**

**{**

**console.log(await cells.nth(j).textContent())**

**}**

**}**

**//print data from all pages- pagination**

**const pages= await page.locator('.pagination li a')**

**const pagecount=await pages.count()**

**console.log(pagecount)**

**for (let p=0; p<pagecount;p++)**

**{**

**if (p>0)**

**{**

**await pages.nth(p).click()**

**}**

**for (let i=0;i<await rows.count();i++)**

**{**

**const currentrow=rows.nth(i)**

**const cells=rows.locator('td')**

**for (let j=0;j<await cells.count()-1;j++)**

**{**

**console.log(await cells.nth(j).textContent())**

**}**

**}**

**await page.waitForTimeout(3000)**

**}**

**await page.waitForTimeout(5000)**

**}**

**)**

**//Reusable function**

**async function selectproduct(rows,page,name)**

**{**

**const matchedrow= rows.filter({**

**has: page.locator('td'),**

**hasText: name**

**})**

**await matchedrow.locator('input').check()**

**await expect(matchedrow.locator('input')).toBeChecked()**

**}**

**WebElement- Date Picker**

**import{test,expect} from '@playwright/test'**

**//date picker by text**

**test('DatePickerbytext',async({page})=>**

**{**

**await page.goto('https://testautomationpractice.blogspot.com/')**

**await page.locator('#datepicker').fill('06/09/2023')**

**await expect(page.locator('#datepicker')).toHaveValue('06/09/2023')**

**await page.waitForTimeout(5000)**

**}**

**)**

**//date picker by looping to year/month/date**

**test('DatePickerusingselect', async ({ page }) => {**

**const targetYear = '2022';**

**const targetMonth = 'August';**

**const targetDate = '21';**

**await page.goto('https://testautomationpractice.blogspot.com/');**

**await page.click('#datepicker');**

**while (true) {**

**const currentYear = await page.locator('.ui-datepicker-year').textContent();**

**const currentMonth = await page.locator('.ui-datepicker-month').textContent();**

**// Check if we have reached the correct year and month**

**if (currentYear === targetYear && currentMonth === targetMonth) {**

**const dates = await page.$$('.ui-state-default');**

**for (const dt of dates) {**

**if (await dt.textContent() === targetDate) {**

**await dt.click(); // Select the date**

**break;**

**}**

**}**

**break; // Exit the loop after selecting the date**

**}**

**// Determine whether to go to the next or previous month**

**const currentDate = new Date();**

**const currentYearNum = parseInt(currentYear);**

**const currentMonthNum = new Date(Date.parse(currentMonth + " 1, " + currentYearNum)).getMonth() + 1; // Months are 0-based in JS**

**const targetYearNum = parseInt(targetYear);**

**const targetMonthNum = new Date(Date.parse(targetMonth + " 1, " + targetYearNum)).getMonth() + 1; // Months are 0-based in JS**

**if (currentYearNum > targetYearNum || (currentYearNum === targetYearNum && currentMonthNum > targetMonthNum)) {**

**// If the current date is later than the target date, go to the previous month**

**await page.locator('[title="Prev"]').click();**

**} else {**

**// If the current date is earlier than the target date, go to the next month**

**await page.locator('[title="Next"]').click();**

**}**

**}**

**await page.waitForTimeout(5000); // Wait for 5 seconds (for demonstration)**

**});**

**//date picker by selecting date**

**test.only('DatePickerbylocatingdate', async ({ page }) => {**

**const date='24'**

**await page.goto('https://testautomationpractice.blogspot.com/');**

**await page.click('#datepicker');**

**await page.click(`//a[@class='ui-state-default'][text()='${date}']`);//mind the quote is done using till sign usually below escape button**

**await page.waitForTimeout(5000); // Wait for 5 seconds (for demonstration)**

**});**

**Mouse Actions**

**import{test,expect} from '@playwright/test'**

**test('MouseHover', async({page})=>**

**{**

**await page.goto('https://demo.opencart.com/')**

**const desktop=await page.locator('//a[normalize-space()="Desktops"]')**

**const macbook=await page.locator('//a[normalize-space()="Mac (1)"]')**

**await desktop.hover()**

**await macbook.hover()**

**await page.waitForTimeout(5000)**

**}**

**)**

**test('MouseRightClick', async({page})=>**

**{**

**await page.goto('https://swisnl.github.io/jQuery-contextMenu/demo.html')**

**const button= await page.locator('//span[normalize-space()="right click me"]')**

**await button.click({button: 'right'})**

**await page.waitForTimeout(5000)**

**}**

**)**

**test('MouseDoubleClick', async({page})=>**

**{**

**await page.goto('https://testautomationpractice.blogspot.com/')**

**const button= await page.locator('//button[normalize-space()="Copy Text"]')**

**await button.dblclick()**

**const secondfield=await page.locator('#field2')**

**await expect(secondfield).toHaveValue('Hello World!')**

**await page.waitForTimeout(5000)**

**}**

**)**

**test('MouseDrag&DrapApproach1', async({page})=>**

**{**

**await page.goto('http://www.dhtmlgoodies.com/scripts/drag-drop-custom/demo-drag-drop-3.html')**

**const source= await page.locator('#box6')**

**const target= await page.locator('#box106')**

**await source.hover()**

**await page.mouse.down()**

**await target.hover()**

**await page.mouse.up()**

**await page.waitForTimeout(5000)**

**}**

**)**

**test.only('MouseDrag&DrapApproach2', async({page})=>**

**{**

**await page.goto('http://www.dhtmlgoodies.com/scripts/drag-drop-custom/demo-drag-drop-3.html')**

**const source= await page.locator('#box6')**

**const target= await page.locator('#box106')**

**await source.dragTo(target)**

**await page.waitForTimeout(5000)**

**}**

**)**

**Keyboard actions**

**import{test,expect} from '@playwright/test'**

**test('Keyboardaction',async({page})=>**

**{**

**//When there are two text inputs on the web page and can be toggled using Tab key**

**await page.goto('https://gotranscript.com/text-compare')**

**await page.locator('[name="text1"]').fill('Welcome Gaurav')**

**await page.keyboard.press('Meta+A')// Control+A for windows**

**await page.keyboard.press('Meta+C')// Control+C for windows**

**//below two commands will switch focus from one text field to other text field**

**await page.keyboard.down('Tab')**

**await page.keyboard.up('Tab') //down and up are used when only single key has to be pressed. For more than one key, use press like above**

**await page.keyboard.press('Meta+V')// Control+C for windows**

**await expect(await page.locator('[name="text1"]').textContent()).toEqual(await page.locator('[name="text2"]').textContent())**

**await page.waitForTimeout(5000)**

**}**

**)**

**Upload Files**

**import {test,expect} from '@playwright/test'**

**test('Singlefileupload',async({page})=>**

**{**

**await page.goto('https://cgi-lib.berkeley.edu/ex/fup.html')**

**await page.waitForSelector('[name="upfile"]')**

**await page.locator('[name="upfile"]').setInputFiles('tests/UploadFiles/test1.pdf')**

**await page.locator('[name="note"]').fill('The file to be uploaded')**

**await page.locator('[type="submit"]').click()//This step will fail as click is navigating to a diff page.**

**await page.waitForTimeout(5000)**

**}**

**)**

**test.only('Multiplefileupload',async({page})=>**

**{**

**await page.goto('https://davidwalsh.name/demo/multiple-file-upload.php')**

**await page.locator('#filesToUpload').setInputFiles(['tests/UploadFiles/test1.pdf','tests/UploadFiles/test2.pdf'])**

**await page.waitForTimeout(3000)**

**expect(await page.locator('#fileList li:nth-child(1)')).toHaveText('test1.pdf')**

**expect(await page.locator('#fileList li:nth-child(2)')).toHaveText('test2.pdf')**

**await page.waitForTimeout(3000)**

**//removing the files**

**await page.locator('#filesToUpload').setInputFiles([])**

**expect(await page.locator('#fileList li:nth-child(1)')).toHaveText('No Files Selected')**

**await page.waitForTimeout(3000)**

**}**

**)**

**Playwright Hooks- BeforeEach/AfterEach**

**import {test,expect} from '@playwright/test'**

**let page; //we are creating a page fixture that will be used in all test cases. This fixture is being declared in beforeEach block**

**//also this will need fullyParallel to be false in config.js**

**test.beforeEach(async({browser})=>**

**{**

**page=await browser.newPage()**

**await page.goto('https://www.demoblaze.com/index.html')**

**await page.locator('#login2').click()**

**await page.locator('#loginusername').fill('pavanol')**

**await page.locator('#loginpassword').fill('test@123')**

**await page.locator('//button[normalize-space()="Log in"]').click()**

**}**

**)**

**test.afterEach(async()=>**

**{**

**await page.locator('#logout2').click()**

**}**

**)**

**test('Home Page',async()=>**

**{**

**await page.waitForSelector('.hrefch')**

**const prods= await page.$$('.hrefch')**

**expect (prods).toHaveLength(9)**

**}**

**)**

**test('Add Product to Card',async()=>**

**{**

**await page.locator('//a[normalize-space()="Samsung galaxy s6"]').click()**

**await page.waitForSelector('//a[normalize-space()="Add to cart"]')**

**await page.locator('//a[normalize-space()="Add to cart"]').click()**

**page.on('dialog',async dialog=>**

**{**

**expect(dialog.message).toContain('Product added.')**

**await dialog.accept()**

**}**

**)**

**}**

**)**

**Hooks- BeforeAll/AfterAll**

**import {test,expect} from '@playwright/test'**

**let page; //we are creating a page fixture that will be used in all test cases. This fixture is being declared in beforeEach block**

**//also this will need fullyParallel to be false in config.js**

**test.beforeAll(async({browser})=>**

**{**

**page=await browser.newPage()**

**await page.goto('https://www.demoblaze.com/index.html')**

**await page.locator('#login2').click()**

**await page.locator('#loginusername').fill('pavanol')**

**await page.locator('#loginpassword').fill('test@123')**

**await page.locator('//button[normalize-space()="Log in"]').click()**

**}**

**)**

**test.afterAll(async()=>**

**{**

**await page.locator('#logout2').click()**

**}**

**)**

**test('Home Page',async()=>**

**{**

**await page.waitForSelector('.hrefch')**

**const prods= await page.$$('.hrefch')**

**expect (prods).toHaveLength(9)**

**}**

**)**

**test('Add Product to Card',async()=>**

**{**

**await page.locator('//a[normalize-space()="Samsung galaxy s6"]').click()**

**await page.waitForSelector('//a[normalize-space()="Add to cart"]')**

**await page.locator('//a[normalize-space()="Add to cart"]').click()**

**page.on('dialog',async dialog=>**

**{**

**expect(dialog.message).toContain('Product added.')**

**await dialog.accept()**

**}**

**)**

**}**

**)**

**Grouping Test cases using describe block**

**import{test,expect} from '@playwright/test'**

**test.beforeAll(()=>{console.log('This is before All hook')})**

**test.afterAll(()=>{console.log('This is after All hook')})**

**test.beforeEach(()=>{console.log('This is before each hook')})**

**test.afterEach(()=>{console.log('This is after each hook')})**

**//Grouping test cases using describe block**

**test.describe('Group1',()=>**

**{**

**test('Test1',async({page})=>{console.log('This is test case 1')})**

**test('Test2',async({page})=>{console.log('This is test case 2')})**

**}**

**)**

**test.describe('Group2',()=>**

**{test('Test3',async({page})=>{console.log('This is test case 3')})**

**test('Test4',async({page})=>{console.log('This is test case 4')})**

**}**

**)**

**// .only and .skip will also work with test.describe for e.g. test.describe.skip, test.describe.only**

**Screenshot and Videos**

**import {test,expect} from '@playwright/test'**

**//default screenshots can be captured by declaring screenshot : 'on','off' or 'only-on-failure' in playwright.config.js**

**//default videos can be captured by declaring video : 'on','off', 'on-first-retry','retain-on-failure' or 'retry-with-video' in playwright.config.js**

**test('Page screenshot', async({page})=>**

**{**

**await page.goto('https://demo.opencart.com/')**

**//await page.screenshot({path: 'Homepage.png'})**

**await page.screenshot({path: 'tests/Screenshots/'+Date.now()+'Homepage.png'})**

**}**

**)**

**test('Full Page screenshot', async({page})=>**

**{**

**await page.goto('https://demo.opencart.com/')**

**await page.screenshot({path: 'tests/Screenshots/'+Date.now()+'Fullpage.png',fullPage:true})**

**}**

**)**

**test('Element screenshot', async({page})=>**

**{**

**await page.goto('https://demo.opencart.com/')**

**await page.locator('//\*[@id="content"]/div[2]/div[1]').screenshot({path: 'tests/Screenshots/'+Date.now()+'Element.png'})**

**}**

**)**

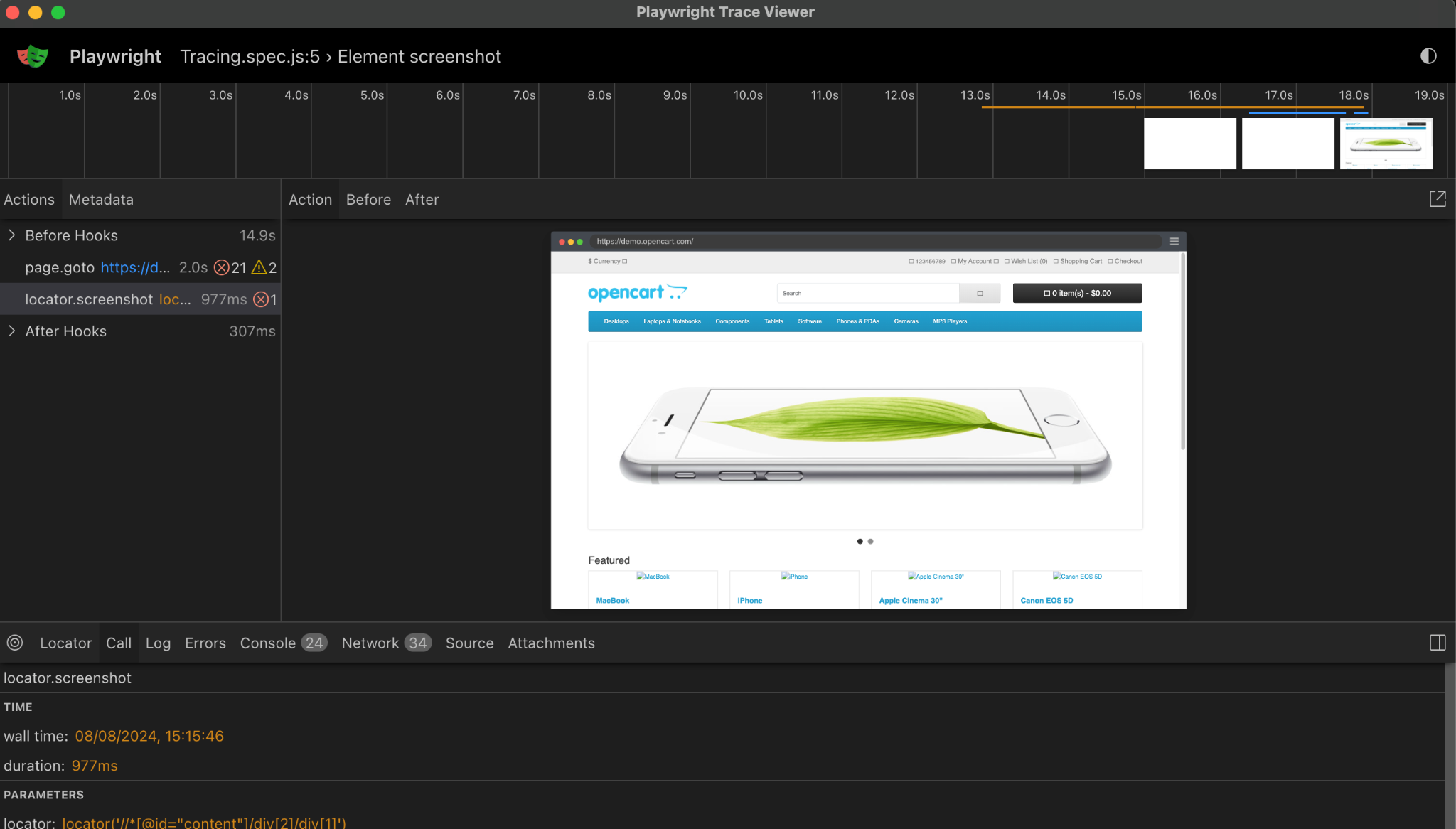
**Trace Viewer**

This can be enabled by enabling trace parameter in the config file. Trace will be stored in the test results folder as a zip file. Below is the command to see trace

npx playwright show-trace “relative path of trace.zip”

You can see information on the Actions and Metadata tab. There are various options that you can set for the config parameter, and accordingly, the trace will be generated.

If you have set the option as ‘retain-on-failure’, the trace will also be appended to the test report when the test case fails.



**Tags**

Tags help you to group your test case so that you can run tests selectively. You can define tags along with the test case name. You can also assign multiple tags to a single test case.

Command to run test cases using tags is

npx playwright test –grep @tagname or

npx playwright test testspecname.spec.js –grep @tagname

For e.g., refer below spec file and commands

import {test,expect} from '@playwright/test'

//Tags

test('test1@Sanity',async({page})=>{console.log('This is test 1')})

test('test2@Sanity',async({page})=>{console.log('This is test 2')})

test('test3@Reg',async({page})=>{console.log('This is test 3')})

test('test4@Reg',async({page})=>{console.log('This is test 4')})

test('test5@Sanity@Reg',async({page})=>{console.log('This is test 5')})

npx playwright test Annotations.spec.js --project=Chromium --headed --grep @sanity

npx playwright test Annotations.spec.js --project=Chromium --headed --grep @Reg

npx playwright test Annotations.spec.js --project=Chromium --headed --grep @sanity@Reg

npx playwright test Annotations.spec.js --project=Chromium --headed --grep @sanity –grep-invert @Reg (invert will exclude the test cases with mentioned tag)

**Annotations**

Annotations are given to manage the test run execution. They can be mentioned along with the “test” to define. Such as test.only, test.skip. You can also skip test cases based on certain values

import {test,expect} from '@playwright/test'

//Annotations

//only

test.only('test1',async({page})=>{console.log('This is test 1')})

//skip

test.skip('test2',async({page})=>{console.log('This is test 2')})

//skip based on value

test('test3',async({page,browserName})=>

{console.log('This is test 3')

if (browserName==='Chromium') {test.skip()}

}

)

//Fixme- Will be used to skip test cases with known issues. Similar to above but it will always skip Fixme is part of test block tag is there

test('test4',async({page})=>{

test.fixme()

console.log('This is test 4')

})

//Fail- when you expect a test case to fail. test case will fail if assertion in test case block is passing. will only pass if assertions fail

test('test5',async({page})=>{

test.fail()

console.log('This is test 5')

expect(1).toBe(1)

})

//Fail-Conditional. This will only consider test fail expectation when the conditions are met. Otherwise, test will skip this fail expectation and will work as is

test('test6',async({page,browserName})=>{

if (browserName==='chromium')

{test.fail()}

console.log('This is test 6')

expect(1).toBe(1)

})

//Slow- to increase the default test time for a specific test case by 3X. Default is 30 secs for all test cases

test.slow('test7',async({page})=>{

console.log('This is test 7')

await page.goto('https://www.demoblaze.com/index.html')

})

**Page Object Model**

The Page Object Model (POM) is a design pattern that helps organize your test code in a more manageable way, especially when working with testing frameworks like Playwright. In POM, you create a class for each page of your application, encapsulating the locators and methods related to that page.

The test case below is referencin to different classes for each page.

import {test,expect} from '@playwright/test'

import {LoginPage} from '../pages/LoginPage.js'

import {HomePage} from '../pages/HomePage.js'

import { CartPage } from '../pages/CartPage.js'

test('WithPOM',async({page})=>

{

//Login Page elements and methods will be referenced from login page class

const Login= new LoginPage(page)

await Login.gotoLoginpage()

await Login.login('pavanol','test@123')

await page.waitForTimeout(5000)

//Home Page elements and methods will be referenced from Home page class

const home= new HomePage(page)

await home.addProductToCart('HTC One M9')

await page.waitForTimeout(5000)

await home.gotocart()

//Cart Page elements and methods will be referenced from Cart page class

const cart=new CartPage(page)

await page.waitForTimeout(3000)

const prodstatus=await cart.checkProductIncart('HTC One M9')

expect(await prodstatus).toBe(true)

}

)

LoginPage class

exports.LoginPage=

class LoginPage

{

//Elements

constructor(page) {

this.page=page

this.loginlink='#login2'

this.usernameInput='#loginusername'

this.passwordInput='#loginpassword'

this.loginButton='//button[normalize-space()="Log in"]'

}

//Methods

async gotoLoginpage()

{

await this.page.goto('https://www.demoblaze.com/index.html')

}

async login(username,password)

{

await this.page.locator(this.loginlink).click()

await this.page.locator(this.usernameInput).fill(username)

await this.page.locator(this.passwordInput).fill(password)

await this.page.locator(this.loginButton).click()

}

}

HomePage Class

exports.HomePage =class HomePage

{

constructor(page)

{

//Elements

this.page=page

this.productList='//\*[@id="tbodyid"]/div/div/div/h4/a'

this.addToCartbtn='//a[normalize-space()="Add to cart"]'

this.cart='#cartur'

}

//Methods

async addProductToCart(productName)

{

const productList= await this.page.$$(this.productList)

for (const product of productList)

{

if (productName===await product.textContent())

{await product.click()

break

}

}

await this.page.on('dialog',async dialog=>

{

if(dialog.message().includes('added'))

{await dialog.accept()}

}

)

await this.page.locator(this.addToCartbtn).click()

}

async gotocart(){await this.page.locator(this.cart).click()}

}

CartPage Class

exports.CartPage=class CartPage{

constructor(page)

{

this.page=page

this.noOfProducts='//tbody[@id="tbodyid"]/tr/td[2]'

}

async checkProductIncart(productName)

{

const productsInCart=await this.page.$$(this.noOfProducts)

for (const product of productsInCart)

{

console.log(await product.textContent())

if (productName===await product.textContent())

{return true

break

}

}

}

}

**Handle Multiple Pages/Windows- By using browser context and page fixtures within**

**import {test,expect,chromium} from '@playwright/test' // We have imported browser here**

**//browser contains multiple context and context can contain multiple pages**

**//Scenario 1- When both pages are independent**

**test('Handlewindow/pages',async({})=>**

**{**

**const browser=await chromium.launch()**

**const context=await browser.newContext()**

**const page1= await context.newPage()**

**const page2= await context.newPage()**

**const allpages=context.pages()**

**console.log("Number of pages created is "+allpages.length)**

**//now page1 and page2 can be handled seperately and both will be for chromium browser.**

**//you can also import other browsers and crerate new context if you want to use other browsers parallely**

**await page1.goto("https://opensource-demo.orangehrmlive.com/web/index.php/auth/login")**

**await expect(page1).toHaveTitle('OrangeHRM')**

**await page1.waitForTimeout(5000)**

**await page2.goto("https://www.demoblaze.com/index.html")**

**await expect(page2).toHaveTitle('STORE')**

**await page2.waitForTimeout(5000)**

**await page1.close()**

**await page2.close()**

**}**

**)**

**//Scenario 2- When page 1 redirects/switches to a new page**

**test.only('HandleMultiplewindow/pages',async({})=>**

**{**

**const browser=await chromium.launch()**

**const context=await browser.newContext()**

**const page1= await context.newPage()**

**await page1.goto("https://opensource-demo.orangehrmlive.com/web/index.php/auth/login")**

**await expect(page1).toHaveTitle('OrangeHRM')**

**//Before clicking the element, we have create one event**

**const pagePromise=context.waitForEvent('page')//this will create a new page and below click will use that page**

**await page1.click('//a[normalize-space()="OrangeHRM, Inc"]')**

**const page2= await pagePromise**

**await expect(page2).toHaveTitle('Human Resources Management Software | OrangeHRM')**

**//after this page1 and page2 can be handled separately**

**await page1.waitForTimeout(3000)**

**await page2.waitForTimeout(3000)**

**//await page1.close()**

**//await page2.close()**

**await browser.close()**

**}**

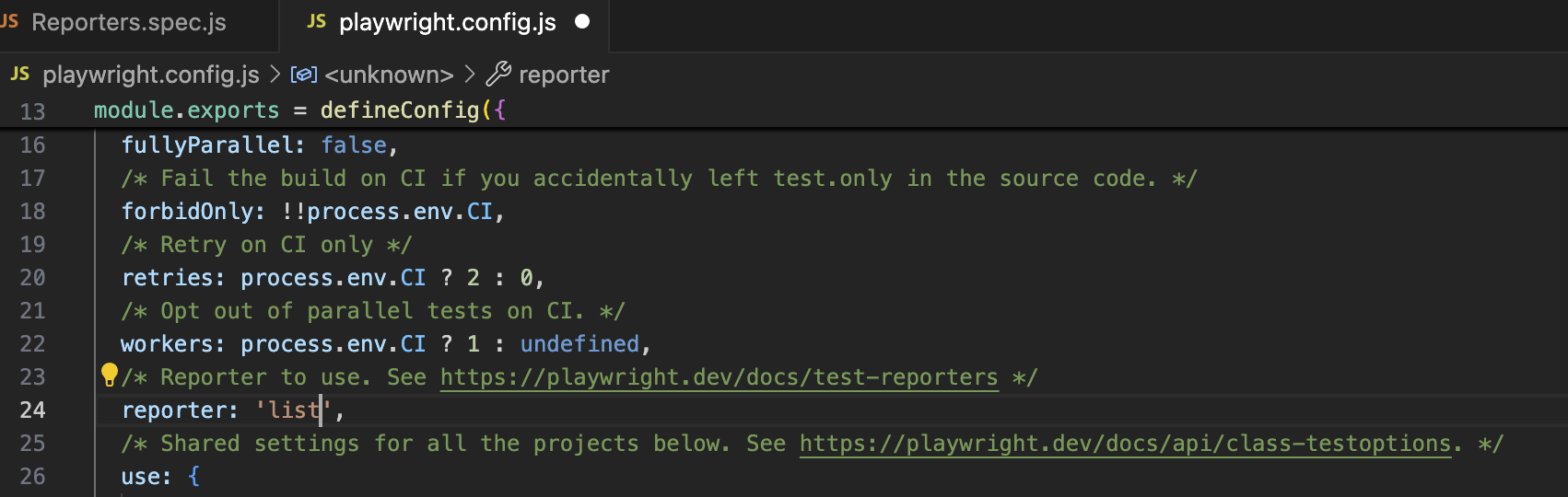
**)**

**PlayWright-Reporters**

Playwright supports a range of built in as well as third party reporters. These can be used in two ways - by defining in config file or by giving command from terminal

**List/Line/Dot Reporter- By updating playwright.config.js**

**Modify reporter parameter value as list, line, dot, html**

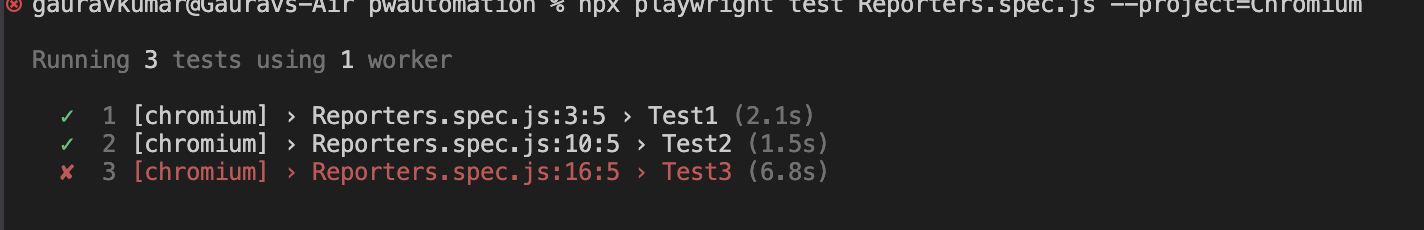


**List/Line/Dot Reporter by using command line**

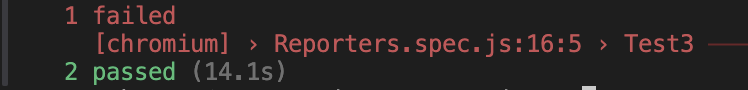
npx playwright test Reporters.spec.js --project=Chromium --reporter=list or line or dot or html

Just like List reporter, Line,dot,html reporter can also be configured from playwright.config.js or from command line

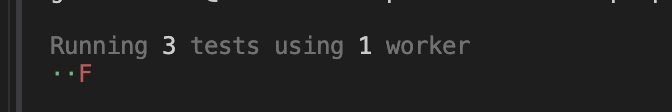
The report can be seen directly in terminal for line, list and dot reports like below



Line report looks like below

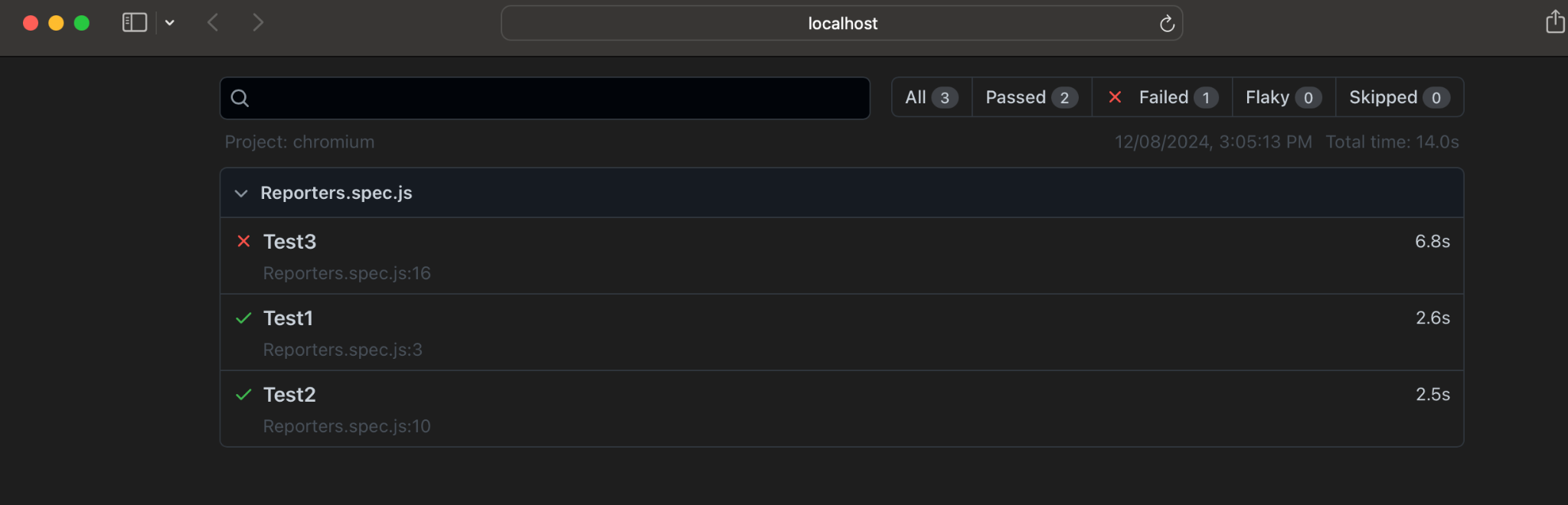


Dot report looks like below



HTML

Html reports are generated in a folder playwright-report and that report can be opened in any browser. Alternatively, you can also run command npx playwright show-report to generate the last run report

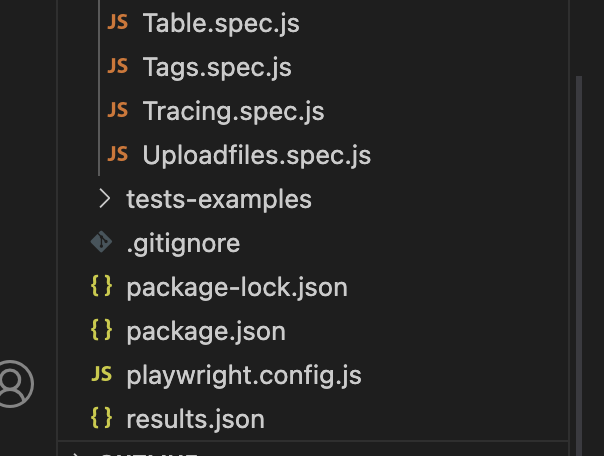


JSON Report

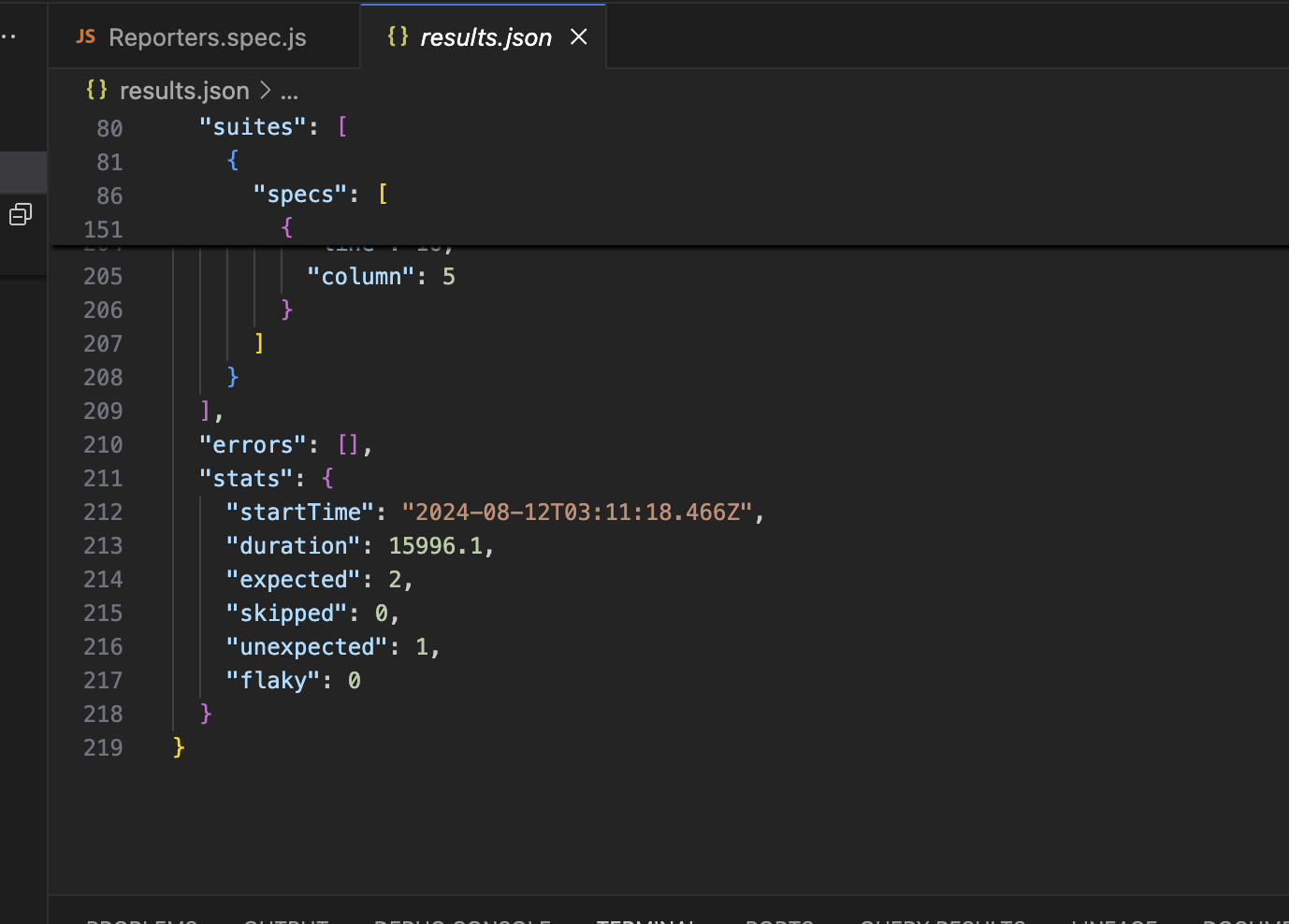
Json report can also be configured from playwright.config.js like below



Report will be generated with the same name that's given in config file



And the report looks like below

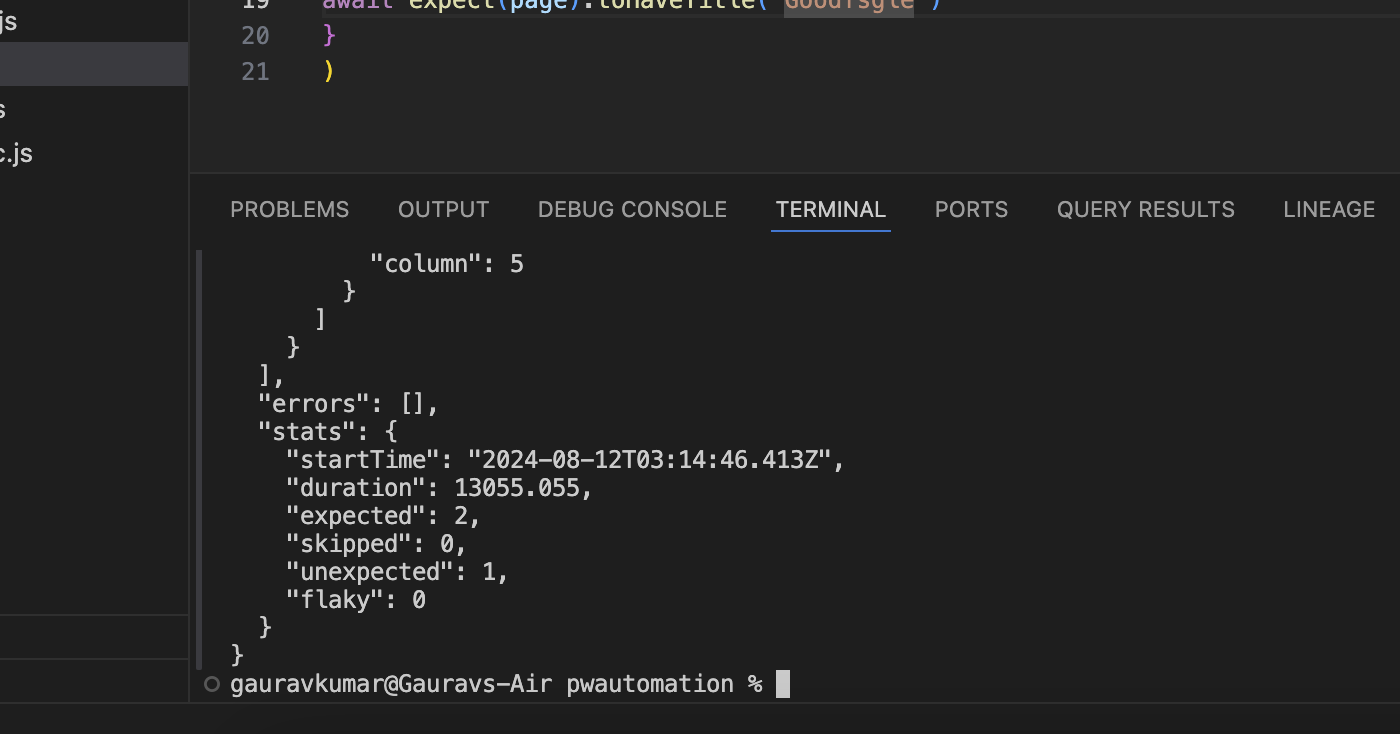


This can also be generated from CL using below command

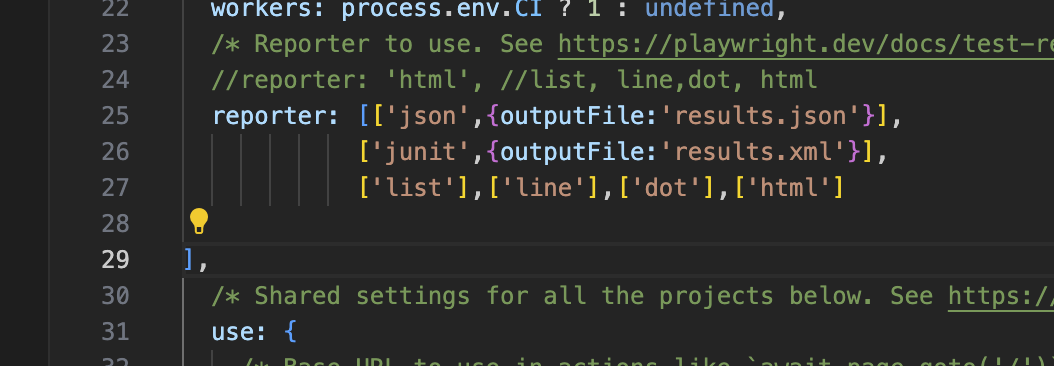
npx playwright test Reporters.spec.js --project=Chromium --reporter=json

And the report will be generated in the terminal itself as no output report name is given. To generate report in a file, you need to configure env variables for output report

Similar to json, Junit reports can also be configured and reports will be generated in xml format.



You can also generate report in multiple format by defining multiple formats in playwright.config.js in the form of array



**Third Party Reporters- Allure Reports**

**Install Allure-playwright module**

npm i -D @playwright/test allure-playwright

After successful installation, you will see a new dependency in package.json

**Install CLI for allure**

npm install -g allure-commandline --save-dev

Declare in playwright.config.js

reporter: [['allure-playwright',{outputFolder:'my-allure-results'}]],

You can also run directly from CLI without declaring in playwright.config.js

npx playwright test Reporters.spec.js --project=Chromium --reporter=allure-playwright

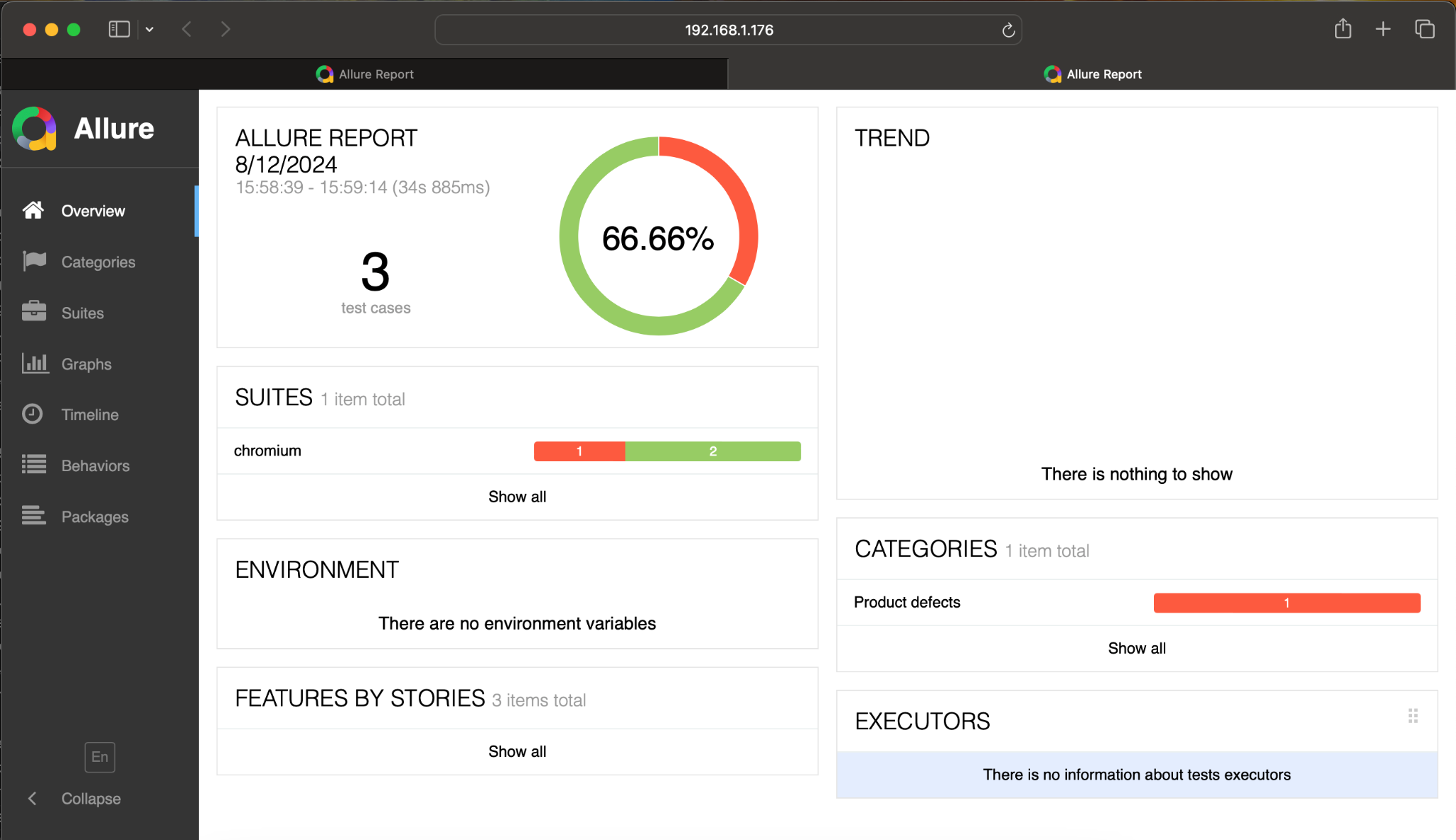
Allure json files will be created after text execution.

After this, we have to generate HTML report by consuming these files by running below commands

allure generate allure-results -o allure-report --clean

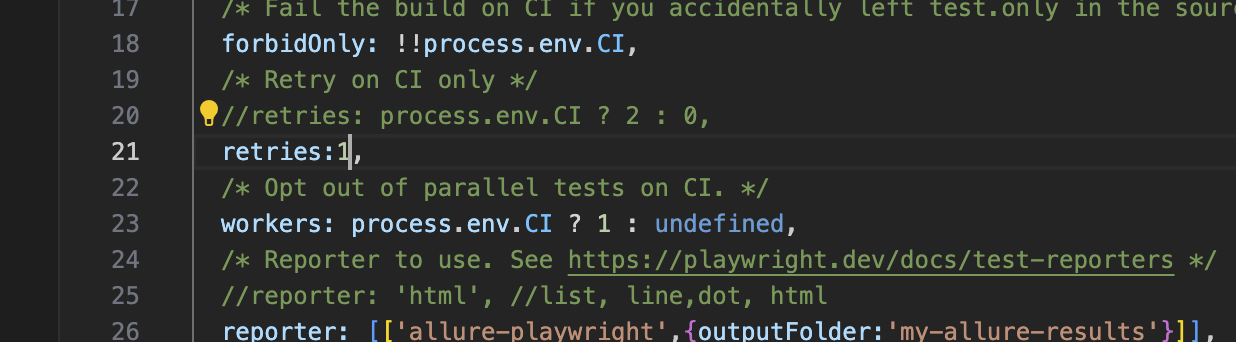
And then to open allure report run

allure open allure-report

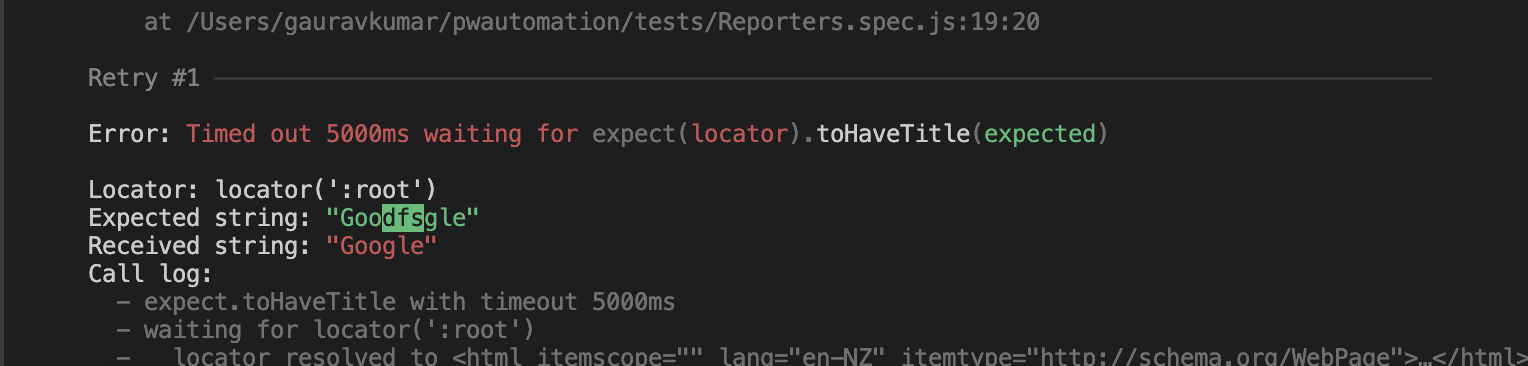


**Retry Failed Test Cases | Retries and Test Flakiness**

This can be achieved by defining retries parameter in playwright.config.js



And this will retry all the failed test cases one more time(depends on number of retries). That can be seen in terminal like below



Some test cases might give intermittent results and that's when we make use of this feature. These tests are also shown as Flaky on HTML report

Alternatively, we can also provide this in our runtime command

npx playwright test Reporters.spec.js --project=Chromium --retries=1

**API Testing using Playwright**

First of all, we need to set fullyparallel to false in playwright.config.js as API test case might have some dependencies

import {test,expect} from '@playwright/test'

//we use request fixture for API testing

var userid

test('Get Users',async({request})=>

{

const response=await request.get('https://reqres.in/api/users?page=2')

console.log(await response.json())

expect(response.status()).toBe(200)

})

test('Create User',async({request})=>{

const response=await request.post('https://reqres.in/api/users',

{

headers:{"Accept":"application/json"},

data:{"name":"Gaurav","job":"Learner"}

}

)

console.log(await response.json())

expect(response.status()).toBe(201)

var res=await response.json()

userid=res.id

})

test('Update User',async({request})=>

{

const response=await request.put('https://reqres.in/api/users/'+userid,

{

headers:{"Accept":"application/json"},

data:{"name":"Gaurav","job":"Learnerupdated"}

}

)

console.log(await response.json())

expect(response.status()).toBe(200)

}

)

test('Delete Users',async({request})=>

{

const response=await request.delete('https://reqres.in/api/users/'+userid)

expect(response.status()).toBe(204)

})

**Interview Questions**

1. **Async and await**

### **Asynchronous Nature of Playwright Operations**

Playwright interacts with web pages and browsers asynchronously. Operations such as navigating to a URL, clicking elements, or waiting for specific conditions to be met involve network requests and browser actions that take time to complete. JavaScript handles these operations asynchronously to ensure that the browser can continue to execute other tasks without being blocked.

### **Purpose of async Functions**

* **Declaration**: Functions marked as async are asynchronous functions. They always return a Promise, which allows them to work seamlessly with other asynchronous functions and operations.
* **Awaiting Promises**: Inside an async function, you can use the await keyword before a Promise-based function call. This makes JavaScript pause execution at that line until the Promise resolves, and then resumes execution with the resolved value.

### **Benefits in Playwright Tests**

1. **Clarity and Readability**: Using async and await makes the asynchronous flow more linear and readable compared to traditional callback-based or Promise-chaining approaches.
2. **Error Handling**: async functions simplify error handling with try-catch blocks, allowing you to catch and handle errors in a straightforward manner.
3. **Synchronization**: Playwright operations often depend on each other (e.g., waiting for an element to appear before interacting with it). await ensures that each step completes before moving on to the next, ensuring synchronization in your test logic.