11. Locators supported by playwright and how to type into elements on page

5 sec is a expect timeout ( timeout : 5000 ) timeout for all assertions is 5 secs

30 sec is a timeout for overall test – global timeout

( .locator ) This is the method which will help us to locate any element on the page.

 most of the times we have to rely only on CSS selector when you are dealing with playwright.

Download selector hub in chrome browser

If ID is present CSS -> tagname#id (or) #id

If class attribute is present CSS -> tagname.class (or) .class

Write css based on any Attribute CSS -> [attribute =’value’]

Write Css with traversing from parent to child CSS -> parenttagname >> childtagname

If needs to write the locator based on text -> text = ‘ ‘

Await page.locator(‘#username’).fill(“rahulshetty”);

Await page.locator(“[type=’password’]”).fill(“learning”);

Await page.locator(“#signInBtn”).click();

12. Extracting the text from browser and inserting valid expect assertions in test

When error message is showing up, this style attribute value is changing.

So developers have dynamically updated the HTML attributes to decide when to display and when not to display.

we give the wrong credentials and extract the error msg and do some assertion.

If you want to grab/extract the text present on the error message use ( dot text content())

console.log(await page.locator("[style\*='block']").textContent());

await expect(page.locator("[style\*='block']")).toContainText('Incorrect username/password.');  //assertion

    // correct credetials

13. How to work with locators which extract multiple webelements in page

if you want to clear the existing content in the text, then this “fill” method will give us handy to perform that operation.

await is not required. await is required only when you are performing the actual action.

So here actual action is performing type, click, text content, everything.

So here we are just storing our CSS selector into one username

Const username = page.locator(‘#username’);

there is a method called fill which does the same purpose of what type it does.

But here the additional feature, what we have with the fill is if you just enter a blank value here,

then it simply wipes off the existing content what is already present.

await userName.fill("");

14. Understanding how wait mechanism works if list of elements are returned

alltextContents() - This method will not wait until elements are showing of the page that 30sec timeout.It won't wait the page.

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

/\*send that browser as a parameter to your test to function first

and that parameter will be passed of to inside your body\*/

test.only('first playwright test', async ({browser})=>    //

{

   // chrome - plugins/cookie(parameter) browser : Browser

   const context = await browser.newContext();   // context,page is a variable

   const page = await context.newPage();

   const userName = page.locator('#username');

   const signInBtn = page.locator("#signInBtn");

   const cardTitles = page.locator(".card-body a"); // parent to child

   await page.goto("https://rahulshettyacademy.com/loginpagePractise/");

   console.log(await page.title());

   //css   // wrong credentials and extract the error msg and do the assertion

   await userName.fill("rahulshetty");

   await page.locator("[type='password']").fill("learning");

   await signInBtn.click();

   console.log(await page.locator("[style\*='block']").textContent());

   await expect(page.locator("[style\*='block']")).toContainText('Incorrect username/password.');  //assertion

    // correct credetials

    await userName.fill("");

    await userName.fill("rahulshettyacademy");

    await signInBtn.click();

    console.log(await cardTitles.first().textContent());

    console.log(await cardTitles.nth(1).textContent());

 // All the products text in the list format

  const allTitles = await cardTitles.allTextContents(); // this method won't wait 30sec bcze it's an array array will be  0 or 10

console.log(allTitles);

});

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

{

   await page.goto("https://google.com");

   //get title - assertion

   console.log(await page.title());

   await expect(page).toHaveTitle("Google");

});

15. Techniques to wait dynamically for new page in Service based applications

npx playwright test tests/filename - it will only run that specific file

waitfor( ) – methods will only work when your locator returns only single element

If it is a click here, you need not give any step because playwright have an auto wait mechanism for the click action.

So for all these actions you never need to write any synchronization step just because all text content method is not here.

So that's why we ended up having a one separate step to handle that wait mechanism.

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

test.only('@Web Client App Login', async ({page})=>    {

 await page.goto("https://rahulshettyacademy.com/client");

 await page.locator("#userEmail").fill("anshika@gmail.com");

 await page.locator("#userPassword").fill("Iamking@000");

 await page.locator("[value='Login']").click();

 //await page.waitForLoadState("networkidle");   // loaded all the API network calls but due to some flaky that why we use wait for method

or

await page.locator(".card-body b").first().waitFor();

 const graballTitles = await page.locator(".card-body b").allTextContents();

 console.log(graballTitles);

});