**what is playwright?** it is a test automation tool that help us to write ui and api test automation script

**advantage of playwright?**

- autowait capability,

- support all major brower, will work on windows, macos, linux and native mobile emulation for google chrome on android and safari for ios enable

comprehensive testing across different devices and os

- provide laungage binding for js,tcs,java,python,c#

**advance feature?**

- tracing and debuging[record and playback]

- network interception

- browser context management [tranfer data to one browser to another]

- codegen tool[record and automatically write test]

**nodejs?**

it is open source server enviroment

**inilize a new node project?**

npm init

**inilize a new playwright project?**

npm init playwright

**filename should be?**

filename\_in\_camel\_case.spec.js

**test runner in playwright file is?**

playwright.config.js

**step to write basic test?**

import annotation in playwright module called test

require('@playwright/test') - store in object - called test -> const {test}=require('@playwright/test')

test function

test("first playwright test",function(){ });

test object should have two arugument first is "testname" and function where we going to write test

**js is async in nature?**

//step1

//step2

//step3

all above step will execute in a same time, that why js is called async

to stop this we need to tell js explicit wait using "await keyword"

if we use await we need to make the function async

**aynomoues function?**

function that does not contains name

async () => {} , instead of function we have arrow mark

**fixture?**

Browser will get inilize first -> context will create with own session,cookies,storage second -> page will be created mean new tab or window will created

brower -> invoke the browser and use it in function around curly bracket -> async({browser})=>{}

page -> will create a newcontext and newpage to automate -> async({browser,page})=>{}

expect –> will have assertion function

**create context?**

create a freash brower instance with proxy,plugin,cookies

context=await browser.newcontext()

await context.newPage(); -> newPage() method create a new page to automate

**how to get a page?** -> using goto() method on page fixture

Await page.goto(“enter you here”)

**Config file:**

testDir – indicate our test folder

testDir: './tests',

timeout – how many time the test should wait by default it will be 30

  /\* Maximum time one test can run for \*/

  timeout: 30 \* 1000,

expect - it is for assertion

reporter – how do we want to create a report

/\* Reporter to use. See https://playwright.dev/docs/test-reporters \*/

  reporter: 'html',

use – test cases will all the property inside the use property, what browser need to execute, screenshot, retry, etc.

  use: {

    /\* browser need to run \*/

    browserName:'chromium'

/\* mode of running \*/

    headless:false

  },

Except – we can give n times to wait for assertion

expect:{

    timeout: 5000

  }

**By default playwright will run on ?** headless mode

For run playwright test? We need to trigger our test runner config file to run test

**Command to run playwrigth?** Npx playwright test

* Npx playwright – find the playwright in our node module
* Test – find the test runner config file
* It will go to test folder and trigger all two test cases one by one

**Run test on headed mode?** Npx playwright test --headed

**Run only one single test?** Test.only(‘test name’,()=>{})

**Get title from page?** Awiat page.title() function help to get title

**Assert title?** Playwright has default assertion

Await expect(page).toHaveTitle(“Google”)

**Locators:**

It is from page fixture

Most of time we need to use css selector

**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 traversion from parent to child** = Css -> parenttagname >> childtagname

**If need to write the locator based on text** = Text =’ ‘

Example:-

test.only("third playwright test",async ({page})=>{

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

    await page.locator("#username").fill("rahulshetty")

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

    await page.locator("#signInBtn").click()

    //webdriverwait for error machnisms

    console.log("my message:- "+await page.locator("[name='loginForm'] div:nth-child(1) strong").textContent())

})

Enter text on input box: await.locator().fill() – fill method enter the value on text box

How to get the innertext on the element: await.locator.textContent() – get the text form the element

How to assert the value is present on the expected value: await expect(locator).toContains(expected value) – “toContains()” check the expected value is presented

How to erase the value in the inputbox:

 await page.locator("#username").fill("")

 await page.locator("#username").fill("rahulshettyacademy")

How to handle if a css selector has multiple element matches:

We can use “first() or nth(index number)”

console.log(await page.locator('.card .card-body h4 a').first().textContent())

    // or

    console.log(await page.locator('.card .card-body h4 a').nth(1).textContent())

How to get all innerText of the element

Using the allTextContents() return the array, it does not have syncronization

await page.locator('.card .card-body h4 a').allTextContents()