17. Handling static Select dropdown options with Playwright

Now there are two kind of dropdowns.

One is a static dropdown where options will already be there. So such dropdowns are called select dropdowns. ( if you do inspect  you will see that tag name for that dropdown as a select).

execution will be very faster that we could not even see that result.

So what you can do is you can have one method here, page.pause();

So what happens is your execution will pause before closing your test. ( this open new playwright inspector)

if there are more than two, let's say five, and if you want second one, then you have to do nth one. ( radio button selection )

18. Selecting radio buttons, Checkboxes and implement expect assertions

19. Using async await with Assertions and understand validating the attributes

On each line of code starting “await” - is required only when you are performing the action.

20. Handling Child windows & Tabs with Playwright by switching browser context\

So in this case when we say operation it's a promise. So basically there are three states of promise

1. Promise pending 2. Promise reject 3. Promise fullfill

Promise - So when you execute any step through JavaScript, it will immediately return something about the status of that step, status of that operation, whatever it returns, we call it as a promise.

pending - that means step is still in the process of execution, but it still won't wait

here. It will just go Next step if you don't have await

 rejected - that means that step is failed due to some reason. Probably object not found or something. This is not successfully executed.

Fulfill - That means this step is successfully executed and it returned.

Whatever this guy is expecting in output.

promise dot all.

So that means wherever you think a set of steps needs to be parallelly, go and wait until these steps are successfully accomplished. Then those steps you can wrap in one array.

The expectation of this array is it has to return um fulfilled promises into an index.

Interview que : What if you have dependency that multiple steps in your playwright script needs to go asynchronously?

We know right to stop asynchronous to make it synchronous. There is a concept called async await. We know that what if there are any steps?

If they want to go asynchronously, that means together parallelly how you can control making sure those two are fulfilled before proceeding next.

In that case you use promise dot all and you put all the steps in this array like this.

So that way it will iterate until all these promises are fulfilled successfully and then only it proceeds to next step.

test('UI Controls', async ({page})=>

{

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

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

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

   const documentLink = page.locator("[href\*='documents-request']");  // bliking text

   const staticdropdown = page.locator("select.form-control");

   await staticdropdown.selectOption("consult");

  // await page.locator(".radiotextsty").last().click();

  await page.locator(".radiotextsty").nth(1).click();

  await page.locator("#okayBtn").click();  //popup

 //Assertions

//console.log(await page.locator(".radiotextsty").nth(1).isChecked()); //Boolean(true/false)

  await expect(page.locator(".radiotextsty").nth(1)).toBeChecked(); // pass fail

  await page.locator("#terms").click(); // agree checkbox

  await expect(page.locator("#terms")).toBeChecked();

  await page.locator("#terms").uncheck();

  expect (await page.locator("#terms").isChecked()).toBeFalsy();

   //await page.pause();

   await expect(documentLink).toHaveAttribute("class","blinkingText"); // blinking text link

});

test.only('child window handle', async ({browser})=>

{

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

   const page = await context.newPage();

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

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

   const documentLink = page.locator("[href\*='documents-request']");  // bliking text

   const [newPage] = await Promise.all([

context.waitForEvent('page'), // this method will be invoked when there is another backgroundpage is created in your original context.

documentLink.click(),

   ])

   const text = await newPage.locator(".red").textContent();

   const arrayText = text.split("@");

   const domain = arrayText[1].split(" ")[0]

   console.log(domain);

   await page.locator("#username").type(domain);

   page.pause();

   console.log(await page.locator("#username").textContent());

});