48. How to save session storage using Playwright and inject into new Browser

There is one method called storage state.

That method, after you log in. What it does is it will copy the entire content. What you see here cookies, session, local storage, tokens, everything. you can store and you can place it in one file called json file.

when browser is open, what we can do is previously once if you login with the UI, it will

store all that storage into one json file.

If you want to do any test related to orders page then they can directly login bcze we are used storagestate all the credentials stored and logged automatically.

Note : So make sure you put currly braces so that it identifies the fixture for (browser/page)

Created new file ( WebAPIPart2.spec.js ) and add the code in this for the (ClientApp.spec.js) then

Test before all method in this add the email password click and waitforloadstate cut and paste here in the before all method.

Once u write the code to path : state.json just run the test state.json file will be created and all the storage data should be displayed in the state.json file

49. How to debug the API steps in script using Visual code debugging

how to debug when there is both API and web mixed in one single test case.

If you want to debug the code then while writing the run command we added –debug

So the debugging start from that, the step one of your script.

Now this is your web UI, so this will help you to debug. What if you want debug api things also? So you cannot debug in playwright inspector, regarding that API calls.

So then how do you want to actually debug and see the logs, everything.

What happened when the API call is made?

First of all, you have to go to Package.json automatically this file will be created for you when you initially give the command of npx playwright project.

Inside the scripts, create one test property and give your command.

So once you have given this perfectly, then you will be able to debug all your code from the terminal itself.

“scripts” : {

“test” : “npx playwright test tests/WebAPIPart2.spec.js –headed”

 once you provided the above line from your keyboard type shift control P

select this debug npm script. this NPM will come to this package.JSON file and you are asking to debug NPM script. if you want to debug from Visual Studio.

Put the debug point anywhere in the api code then click shift ctrl P and select debug npm script from that breaking point that test will run and set by set we can able to do api debug

If you are test purely web, then you can depend upon playwright inspector.

50. Detailed view of Trace viewer to understand the API logging req/response

Once u run any API tests if we want to see the logs and run the test file then refresh the project and in the test-results for you can see the trace.zip file.

trace.playwright.dev – you need to upload the trace.zip file here then u can able see.

51. How to intercept Network response calls with Playwright route method

there is one scenario that you have to test is if there are no orders displaying on the page, you have to check for a message saying, sorry, there are no orders displayed.

can play right dynamically stop that response.And instead of this real response, can we inject a fake response and send it to the browser?

If you send the response of this with the empty data, obviously page looks like this. ( no orders) But that is only for that session.

Playwright is faking that response only for that automated browser session.

Once you close the browser, that will go off okay.

And again, all the others will show up as usual here only for that particular instance, we are altering the response

52. Understand the playwright route method and its parameters in intercepting demo

So basically, we need to log in until this page and we need to click on this order link and we have to make sure no orders is displaying here. That is how we need to start the flow.

So in playwright, we will tell like this page dot root, root means when you receive this URL, root

it to the way I want.

First argument. It's all about which URL we want to root, which network call we want to root.

And second argument is all about how you want to root now. ( function )

So route this the parameter and this is asynchronis function that why I give fat operator (=>)

Page dot request means we are turning our page mode to the API mode.

When you do page dot it will act on browser, but for if you want to turn your page to the mode where it can make API calls, then you have to do call request method on page.

this routing has to happen before you actually click on that orders button.

Now, see, before you click on orders button, you have to tell a playwright clearly that, whenever you encounter this, then do in this way.

make sure when you create any JavaScript object payload, convert it into Json.

Also, when you run into this error of disposal anytime, add this one more step asking for which call you want to wait for a response with this wildcard

Networktest.spec.js

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

const loginpayload = {userEmail: "anshika@gmail.com", userPassword: "Iamking@000"}

const orderPayload = {orders: [{country: "Cuba", productOrderedId: "6581ca399fd99c85e8ee7f45"}]}

const {APIUtils}=require('./utils/APIUtils');

const fakePayLoadOrders = { data: [], message: "No Orders" };

let response;

test.beforeAll(  async() =>

{

   //login API

   const apiContext = await request.newContext();

   const ApIUtils =new APIUtils(apiContext,loginpayload);

   response = await ApIUtils.CreateOrder(orderPayload);

});

//Create order is success

test('place the order', async ({page})=>    {

    page.addInitScript(value => {

      window.localStorage.setItem('token',value);  // key : value pair

    }, response.token);

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

 await page.route("https://rahulshettyacademy.com/api/ecom/order/get-orders-for-customer/\*",

 async route => {

   const response = await page.request.fetch(route.request());

   let body = JSON.stringify(fakePayLoadOrders);

   route.fulfill(

     {

       response,

       body,

     });

   //intercepting response -APi response-> { playwright fakeresponse}->browser->render data on front end

 });

await page.locator("button[routerlink\*='myorders']").click();

await page.waitForResponse("https://rahulshettyacademy.com/api/ecom/order/get-orders-for-customer/\*")

console.log(await page.locator(".mt-4").textContent());

});

54. How to intercept Network request calls with Playwright - Example demo

whenever you click on a view button, what is happening?

That order ID is concatenating and one get call is being made.

Forbidden error 403 status code.So forbidden means you are not authorized to view these details. So that's how you need to develop your application to make sure that this application will should not be able to access other people account.

NetworkTest2.spec.js

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

test('@QW Security test request intercept', async ({ page }) => {

    //login and reach orders 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');

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

    await page.locator("button[routerlink\*='myorders']").click();

    await page.route("https://rahulshettyacademy.com/api/ecom/order/get-orders-details?id=\*",

        route => route.continue({ url: 'https://rahulshettyacademy.com/api/ecom/order/get-orders-details?id=621661f884b053f6765465b6' }))

    await page.locator("button:has-text('View')").first().click();

    await expect(page.locator("p").last()).toHaveText("You are not authorize to view this order");

})

56. How to abort the Network calls with Playwright – Examples

So in intercepting means, just not only modifying, you can even stop the calls to reach the browser.

So your browser UI can be coded in such a way that when server is down it will tell you that looks like something is wrong at our end. Please give 24 hours.

There could be some error message directly on the screen.

\*\*/\* - means any URL ends with .css extension go and block it.

On method it will invoke when an event occurred. So that event what you have to write in the first argument. 2nd argument for that request it get the URL

UIBasics.spec.js

// page.route('\*\*/\*.css',route=>route.abort());

 //page.route('\*\*/\*.{jpg,png,jpeg}', route=> route.abort());

//page.on('request', request=> console.log(request.url()));

   //page.on('response',response=>console.log(response.url(), response.status()));