Q>**WebdriverIo dependencies**?

A>

  "devDependencies": {

    "@types/chai-arrays": "^2.0.0",

    "@wdio/allure-reporter": "^8.11.0",

    "@wdio/cli": "^8.3.2",

    "@wdio/local-runner": "^8.3.2",

    "@wdio/mocha-framework": "^8.3.0",

    "@wdio/spec-reporter": "^8.3.0",

    "chromedriver": "^114.0.2",

    "wdio-chromedriver-service": "^8.0.1"

  }

Q> **How to config WDIO**?

🡪 npx wdio config

And then answer the questions it ask.

Q> **Where do we generally write test cases in webdriverIo project**?

A>

Make a directory in root folder called test then inside it /specs

Folder, where we will write all our test cases.

Each test file will be called a specs file

Q>**What is wdio.conf.js file**?

A>

Initiate by giving command npm init wdio .

1. This file is like a Test Runner.
2. Here we will give all the information like what test cases/ specs need to run.
3. How many instances we have to run.
4. The browser we will use to execute tests.
5. Base\_URL we will be using.
6. Connection timeout.
7. Test framework we will be using.

And many more…stuff

Q> **What is “Mocha”**?

A>

1. Mocha is a JavaScript framework for Node.js programs.
2. It features browser support, asynchronous testing, test coverage reports and use of any assertion library.
3. Mocha tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to correct test cases.
4. We need a testing framework to build a spec file and run our webdriverIo code.

describe('My Login application', () => {

    it('should login with valid credentials', async () => {

        //WebdriverIo code

    });

});

Here describe and it functions are coming from mocha test framework.

1. describe stands for test suite and its name is “*My Login application*”. There can be n number of cases that we can write inside it. It can be treated as Parent.
2. Each test case will be written inside it block. This particular test case name is “*should login with valid credentials*”. It can be treated as Child.
3. WebdriverIo automation code must be wrapped inside it block.

Q> **How to configure auto suggestions for WebdriverIo module in VS code**?

A>

1. Go to WebdriverIo official page and navigate to docs🡪integration🡪auto completion.
2. Add a jsconfig. json file at your project level and paste code from WebdriverIo site into it.

{

    "compilerOptions": {

        "types": [

            "node",

            "@wdio/globals/types",

            "@wdio/mocha-framework"

        ]

    },

    "include": [

        "test/specs/\*.js",

        "\*\*/\*.json",

        "node\_modules/@wdio/sync",

        "node\_modules/@wdio/mocha-framework"

      ]

}

1. The “compilerOptions” and “types” is from the WebdriverIo site but “include” is added extra because we want compilerOptions and types to get *detected in all .js spec files which we are writing currently in test/specs/ as well as in selected node modules*.

Q> **How to remove asynchronous nature of JavaScript while writing automation test**?

A>

**Problem Statement**:

1. While writing automation cases, we want to execute all steps of workflow, one by one, in a sequence, in order to replicate a test case scenario.
2. The problem with asynchronous nature of JavaScript code is that there is no guarantee that all the steps will execute in a sequence unlike other programming languages like python and java which are synchronous in nature.
3. JavaScript will start executing first line of code, let’s say opening the URL on the browser but it will quickly move to next line of code without even waiting for URL to fully load.
4. This will cause an issue, if next line of code is directly dependent on some information from URL like “Title of URL” and it gets executed without any information.

describe("Ecommerce Application", () => {

  it("Login Fail Page", () => {

    browser.url("https://rahulshettyacademy.com/loginpagePractise/");

    console.log(browser.getTitle());

  });

});

**async-await**:

1. We will use async-await syntax provided by JavaScript in which each line accompanied with await keyword will return one promise.
2. This promise will have one of these three status resolved / pending / rejected.
3. When *line execution is completed*, we say the promise is resolved.
4. If the promise is pending, JavaScript compiler knows, it has to come back to this line because it is not fully done and *goes to next step*.
5. When *JavaScript fails to execute a line* we say promise is rejected.

**Practical Solution**:

1. In order to make our JavaScript code behave in a synchronous way, we first need to make sure that *compiler should go to next line of code only when promise is resolved in present line of code*.
2. Write await before each line where you want compiler to wait until its promise is in resolved state.
3. Write async while writing function in which all asynchronous code is written (so that compiler treats it synchronously using await).

describe("Ecommerce Application", async () => {

  it("Login Fail Page", async () => {

    await browser.url("https://rahulshettyacademy.com/loginpagePractise/");

    console.log(await browser.getTitle());

  });

});

Q>**How to start testing / command in WebDriverIo?**

1. **In Terminal type,**

npx wdio run .\wdio.conf.js

🡪npx will search for wdio module in the node modules .bin folder

🡪run .\wdio.conf.js will start our test runner file located in root directory.

1. Configure our spec file name (*where we have written our test cases*) inside wdio.conf.js file

specs: ["./test/specs/\*\*/firstTest.js"],

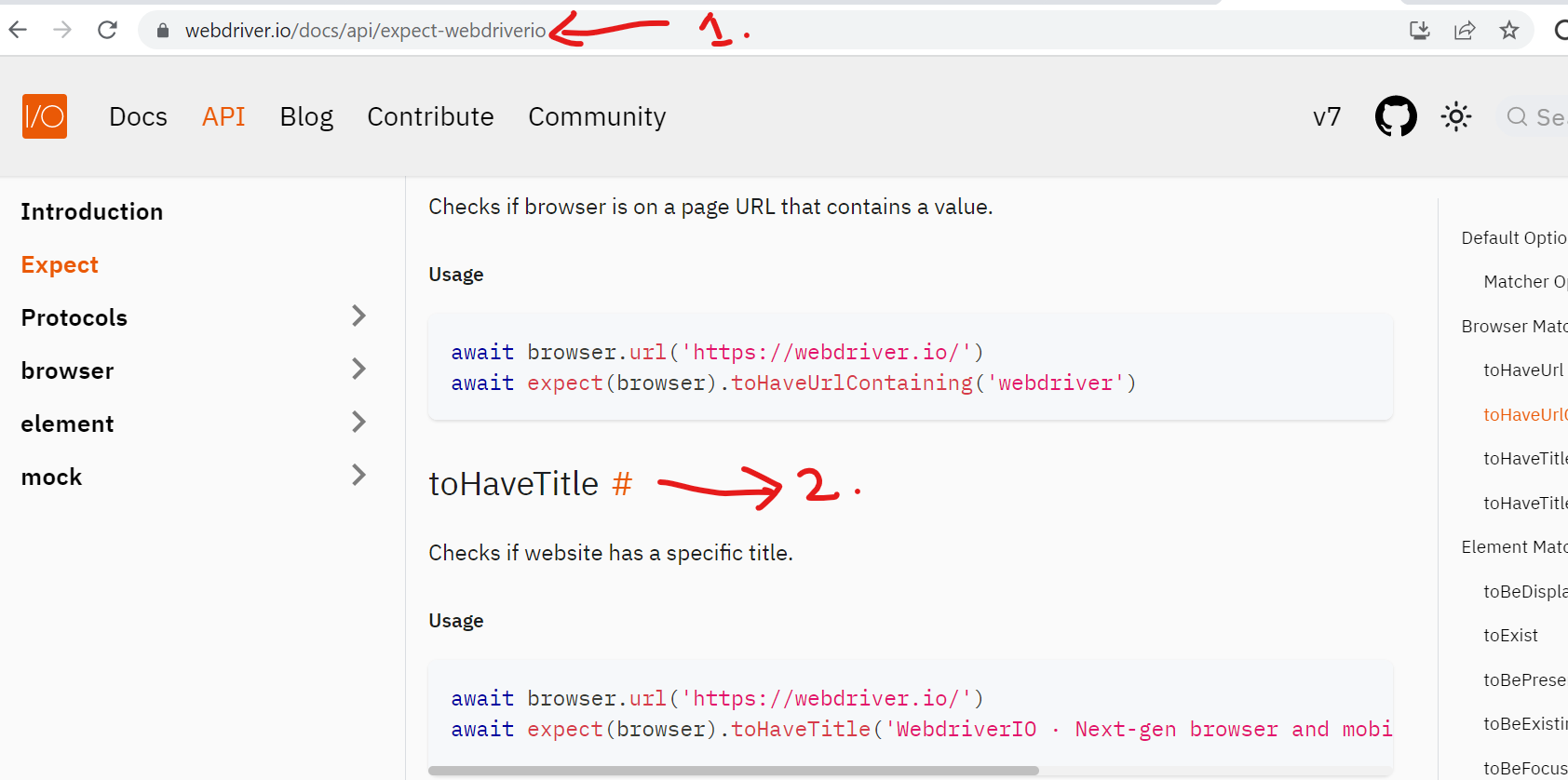
**Note:** specs is a property which will take the spec file which it has to execute.

 specs: ["./test/specs/\*\*/\*.js"],

In order to run all files inside specs folder.

Q> **What are assertions and Expect keyword used for, while writing code in WebDriverIo**?

1. **Expect:** When you are writing tests, you often need to check that values meet certain conditions, expect gives you access to a number of “matchers” that lets you validate different things on the browser or element object.
2. For example, you can get the title of the browser and compare it with the *expected* string and if both matches, test should pass.
3. Now, we need to bring these matchers or **assertions** in picture to compare *expected vs actual* value.



1. Go to <https://webdriver.io/docs/api/expect-webdriverio> and search for matcher or assertion that you want to use, in above picture we are searching for an assertion which helps us check title of a given website.
2. **Syntax:** use expect(browser). followed by assertion method, in our case its toHaveTitleContaining.

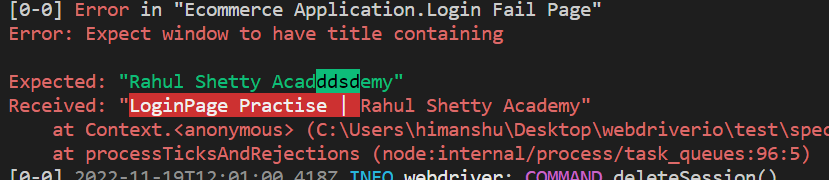
await expect(browser). toHaveTitleContaining("Rahul Shetty Acadddsdemy");

1. In case expected and actual are not same, compiler will wait and search for some time for the expected result. This time duration will depend on waitforTimeout value in our test runner wdio.conf.js file.

 // Default timeout for all waitFor\* commands.

  waitforTimeout: 10000,

**Note:** Below is the compiler showing mismatch of expected vs received title string. It even highlights the exact part where string is mismatched.

****

Q> **How do you use locators in WebDriverIo by creating a custom CSS selector or Xpath**?

A>

1. WebDriverIo uses two types of locators

**🡪**CSS Selectors

🡪XPATH

1. We can create *custom CSS selectors to be used as locator* from HTML element ID and CLASS

* In case of ID, use: - #id\_NAME
* In case of CLASS use: - .className.

1. After getting the locator, we can use this in our code by using $ syntax like $(“#username”). WebDriverIo will treat this as locator after opening the browser.
2. In case you *don’t have class or id in the HTML tag* in that case, you can **create custom locator** using below syntax.

**tagname[attribute=”value”]**

For example, in this input tag, we only have type and name attribute,

<input *type*=”text” *name*=” username”/>

So, the custom locator will be input [name=” username”]

1. You can select multiple locators for the same element at the same time

 await $("#username").setValue("Good Bye world");

    await browser. pause(3000);

    await $("input[name='username']").setValue("Hello world");

First it will type *Good Bye World*, wait for three seconds then clear out last text and type *Hello world* in the same input element.

1. To write the XPATH, we need to use below format,

**//tagname[@attribute=”value”]**

So you can choose either depending on your preference //input[@id='password’] or //input[@type='password'].

const password = $("//input[@id='password']");

    await password.setValue("learning");

Q> **How to get text present in any HTML element**?

A>

1. Locate the element using CSS selector or XPATH.
2. Use getText () method on it

await console.log($(".alert.alert-danger.col-md-12").getText());

**Note:** To get the locator information and validate use selectorsHub plugin in your browser.

Q> **How to define explicit waits in WebDriverIo**?

A>

1. Use waitUntil method which takes two arguments.
2. First argument is a ***condition*** which has to be satisfied for example *wait until a particular error message is displayed*. It has to be wrapped in the form of a function.
3. Second argument is the object and it mainly takes ***timeout*** value which is the *maximum duration we will wait for the condition to be true*. It has to be in JSON key-value form.

 await browser.waitUntil(

      async () => (await $("#signInBtn").getAttribute("value")) === "Sign In",

      {

        timeout: 5000,

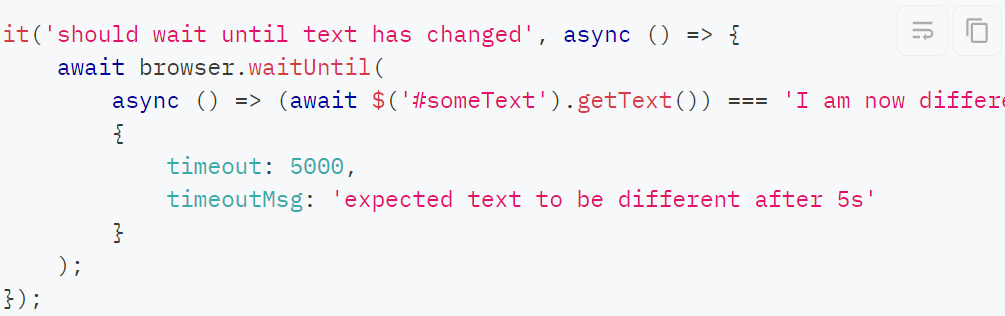
        timeoutMsg: "expected text should come in 5 seconds",

      }

    );

    await console.log(await $(".alert.alert-danger.col-md-12").getText());

Below code snippet is from WebDriverIo official documentation page.

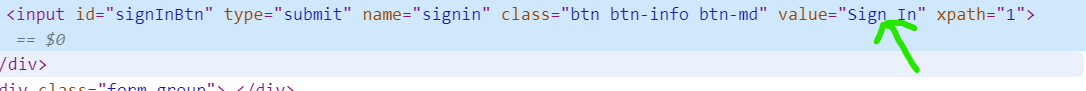
****

1. So writing code this way will help us wait dynamically for some predefined action without using sleep method.

**Note**: Always write async keyword before a function, if the function has some await statements inside it.

Q> **How to get any attribute value from an element?**

1. First inspect the element and observe if it has any unique attributes.
2. Now, locate the element using CSS or XPATH.
3. Use getAttribute method on this locator and

****

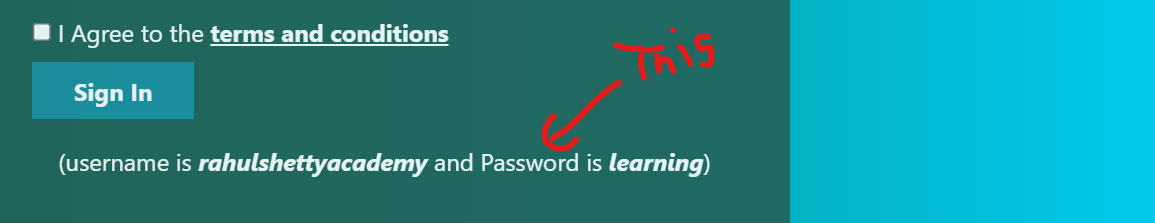
$("#signInBtn").getAttribute("value")

1. The above line of code will *return value of value attribute* as “Sign in”

Q> **How to pass or fail a test based on expected text value inside an element**?

A>

1. Take example of a <p> tag element which has some text inside it.



1. First locate this element, in my case I choose Xpath for this purpose.

const textElement = $("//p[@class='text-center text-white']");

1. Now we use expect method on this element along with toHaveTextContaining method.
2. Insert the text value as argument in toHaveTextContaining.

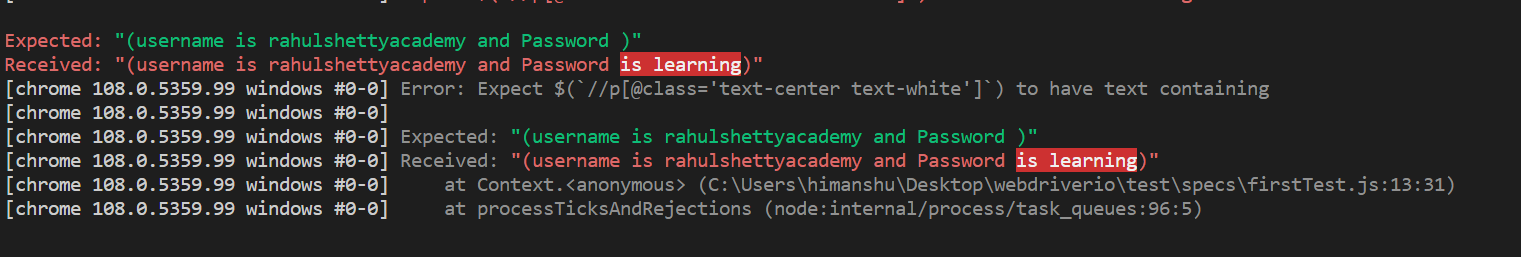
  const textElement = $("//p[@class='text-center text-white']");

    await expect(textElement).toHaveTextContaining(

      "(username is rahulshettyacademy and Password )"

    );

1. As you can see I passed the wrong text inside toHaveTextContaining method, so it will wait for timeout (while its searching for text…)and then will result Fail for this test case.



Above is the result we see in the console. The missing or wrong text is even highlighted.

**Note:** toHaveTextContaining . is actually used for partial text matching as a subtext but if you want exact word to word text match then use toHaveText method.

Q> **How to wait for a specific element to appear on page and then perform some operation on it**?

A>We can use waitForExist method on the element.

 //wait until checkout button is displayed

    const checkoutButton = $("//a[@class='nav-link btn btn-primary']");

    await checkoutButton.waitForExist({ timeout: 5000 });

    await checkoutButton.click();

Q> **How to check if a correct URL is present when a page is loaded?**

Look into assertions under expect API on official documentation page 🡪 toHaveUrlContaining



Q> **If you have multiple test cases in the same file, but you want to run only selected ones, how to do that**?

A>

Just put a small case ‘**x**’ before a test case.

describe("Ecommerce Application", async () => {

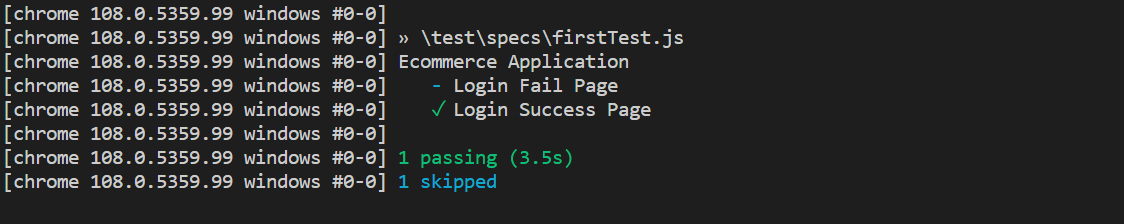
  xit("Login Fail Page", async () => {

    await browser.url("https://rahulshettyacademy.com/loginpagePractise/");

    console.log(await browser.getTitle());

    await expect(browser).toHaveTitleContaining("Rahul Shetty Academy");

the console will show this…



The case with **x** before it, will be skipped and will not be picked by webdriverIo at run time.

Q> **How to run same test in different browsers from the same code base**?

1. We can switch the browser at any time using selenium-standalone service.
2. Add this package in your package.json file using

npm install @wdio/selenium-standalone-service --save-dev

1. Now go to your wdio.conf.js and navigate to services and change its value to “selenium-standalone”
2. Inside capabilities array change browserName to your browser of choice.

browserName: "MicrosoftEdge",

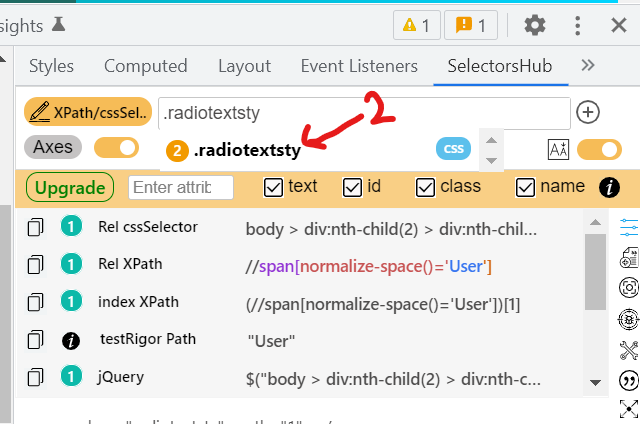


Run the test and now your selected browser will open and start executing test cases.

**Note:** Always use await syntax when you are assigning variables to a locator

Q>**Suppose same class name is used for two different elements, how can you choose to operate only one element while using the same className as locator**?

1. In this example we see two elements with same className



1. Catch all the elements with same className inside a variable using double dollar sign **$$** before locator

const radioButtons = await $$(".radiotextsty");

1. Now this variable, radioButtons is an array which holds two elements.
2. WebDriverIo reads the elements from top left to right, so first element will be radioButtons[0] and second one will be radioButtons[1].

In order to click the second element of radioButtons array,

await radioButtons[1].click();

    await browser.pause(4000);

Q>**Difference between waitForExist and waitForDisplayed**?

We use waitForExist to see if a particular element exists on our webpage or not.

waitForDisplayed is used for elements which exist on the webpage but are not visible or hidden

Q> **How to make browser pause/sleep/delay**?

A>

You can use implicit wait in the form of pause method

await **browser.pause**(4000)

It will stop the browser midway execution and you can use it to see the operations happening on browser.

Q> **How to make sure an element is selected or not**?

1. We can use webDriverIo’s, isSelected method to see whether or not a particular element is selected
2. It returns a true / false Boolean value.

 console.log(await $$(".customradio")[0].$("span").isSelected());

Q> **How to select Static dropdown values using WebdriverIo**?

A>

1. We can’t do direct operations on dropdown values since we get below error in selectors hub on inspecting one of the dropdown item.



1. So we inspect parent dropdown element and save it in a variable called dropdown.

const dropdown = await $("//select[@class='form-control']"); //parent element

There are three methods we can use to select a dropdown items

1. selectByAttribute
2. selectByVisibleText
3. selectByIndex

 await dropdown.selectByAttribute("value", "teach"); //select child element by its attribute

    await browser.pause(2000);

    await dropdown.selectByVisibleText("Consultant"); //select child element by visible text

    await browser.pause(2000);

    await dropdown.selectByIndex(0); //select child element by its index

    await browser.pause(2000);

**Note:** To verify we get the correct dropdown value at the end of execution we can use getValue assertion

 console.log(await dropdown.getValue());

Q> **How to select from Dynamic dropdowns?**

A>

1. By dynamic dropdown we mean the dropdown fields where we start typing and then dropdown items appear as per our input.

 it("dropdown\_practice", async () => {

    await browser.url("https://rahulshettyacademy.com/AutomationPractice/");

    const dropdownCountry = await $("#autocomplete");

    await dropdownCountry.setValue("ind");

    const dropDownItem = await $("//ul[@id='ui-id-1']//li[2]");

    await dropDownItem.waitForExist({ timeout: 5000 });

    await dropDownItem.getText();

    await dropDownItem.click();

    await browser.pause(2000);

  });

Refer to 🡪(**Lecture #39** for different method) . Where all the items in dropdown are contained inside one variable using **$$** sign and then we loop them one by one to check which one has Text that we want to select from dropdown menu.

Q>**When do we need “chai” assertions**?

1. We use webdriverIO assertions for matching title, URL, Text etc. which are browser/ UI related assertions.
2. But if want to compare two values like strings (basic/general assertions) we need to use another library called chai.
3. npm install chai and then import this chai module using require.

const expectChai = require("chai").expect;

1. webDriverIo uses expect method as well, so in case we need to use expect from chai, create a new variable expectChai as above.

expectChai(await dropdown.getValue()).to.equal("st");

Now we use this expectChai and based on this assertion our test will pass or fail.

Q> **How to scroll on page until a particular element appear**?

Use scrollIntoView command on a locator

await browser.url("https://rahulshettyacademy.com/AutomationPractice/");

    await $("#mousehover").scrollIntoView();

Q> **How to hover over an element**?

Use moveTo command to just hover over the item without clicking on it

await $("//button[@id='mousehover']").moveTo();

    await browser.pause(2000);

Q> **How to perform double click on a button**?

Use doubleClick method

 await browser.url(

      "https://only-testing-blog.blogspot.com/2014/09/selectable.html"

    );

    await $("button[ondblclick='myFunction()']").doubleClick();

Q> **How to make “Chai” assertion to see if alert is open or not**?-🡪 Not Working

We can use isAlertOpen method directly on browser lever to see if any alert is currently opened in our browser window or not.

If isAlertOpen returns true, then test should pass otherwise fail.

console.log(browser.isAlertOpen());

    expectChai(browser.isAlertOpen()).to.be.true;

    await browser.pause(2000);

Q> **How to open a new tab / child window from current browser.url and do some operations on it**?

A>

1. We have to use two browser methods for this purpose **switchToWindow** and **getWindowHandles**.
2. getWindowHandles will return an array of windows that are currently open while performing automation.

const handles = await browser.getWindowHandles();

1. This array “handles” gives information about parent window (handles[0]) and child window(handles[1]).
2. Now to switch to child window and perform operations on it we use switchToWindow method.

 await browser.switchToWindow(handles[1]);

**complete code:**

describe("Windows and Frames Miscelleneous", async () => {

  it("Parent and Child windows switch", async () => {

    await browser.url("https://rahulshettyacademy.com/loginpagePractise/");

    const blinkingText = await $(".blinkingText");

    await blinkingText.waitForClickable({ timeout: 5000 });

    await blinkingText.click();

    await browser.pause(2000);

    const handles = await browser.getWindowHandles();

    await browser.switchToWindow(handles[1]);

    const documentRequest = $("h1");

    console.log(await documentRequest.getText());

  });

});

Q>**What is the difference between switchToWindow and switchWindow methods?**

A>

1. In switchToWindow method *we navigate to a new tab/page which is opened by the application*.

For example, when we click on a link in our application and a new tab is opened, in that case we use switchToWindow method to do some operations on the newly opened tab.

1. In switchWindow method *we navigate to new tab/page which is opened by webdriverIo automation script*.

For example, below case

await browser.newWindow("https://www.google.com/");

    console.log(await browser.getTitle());

    await browser.switchWindow(

      "https://rahulshettyacademy.com/loginpagePractise/"

    );

    await $("#username").addValue("Helloiswitchedback");

    await browser.pause(3000);

Here, we first use newWindow method to open a new window and then in order to navigate to another window we use switchWindow mehthod.

**Note:** Once we switch using switchWindow it becomes our parent window.

Q>**How to find total number of elements of a specific type, let’s say all anchor tags on a page**?

1. First, collect all the elements of a specific type using $$, it will return an array.
2. Now find the length of the array which will be total number of elements of inside that array.

 it("Frames switching", async () => {

    await browser.url("https://rahulshettyacademy.com/AutomationPractice/");

   console.log(await $$("a").length);

    await browser.pause(2000);

  });

});

Q> **What is an iframe**?

1. An inline frame or iframe is a HTML element that loads another HTML page within its parent page.

*It essentially puts another webpage within the parent page.*

They are used for advertisements, embedded videos, web analytics and interactive content.

Q> **How to switch to an iframe and find some information, let’s say number of anchor tags inside this particular iframe**?

1. We can use switchToFrame method on browser in order to get iframe into focus of webdriverIo.(Make sure its visible on page)
2. Then we can combine all the anchor tags on this page inside an array and find its length to get number of anchor tags.

  it("Frames switching", async () => {

    await browser.url("https://rahulshettyacademy.com/AutomationPractice/");

    await $("#mousehover").scrollIntoView();

    console.log(await $$("a").length);

    await browser.pause(2000);

    await browser.switchToFrame(await $("#courses-iframe"));

    console.log(await $$("a").length);

    await browser.pause(2000);

  });

});

**Note:** If you want to switch back to parent window from iframe. We use switchToFrame method again but keep its argument as null

    await browser.switchToFrame(null);

    await $("#mousehover").click();

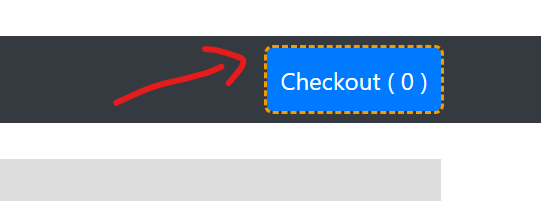
    await browser.pause(2000);

You will see that automation controls will be switched back to parent window and you will see click action happening there.

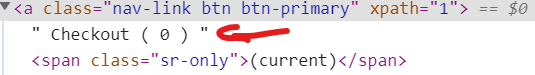
Q> **How to use a partial link text?**

A>

1. This locator *should only be used in case of links* (elements with anchor <a> tags)



For example, this Checkout button consist of an anchor tag with this text present



We can locate this element using partial text in this format

  //see if checkout button exist

    await $("\*=Checkout").waitForExist();

**Note:** Put \*= before the text that you want to locate.

Q> **Convert this CSS locator into XPATH locator?**

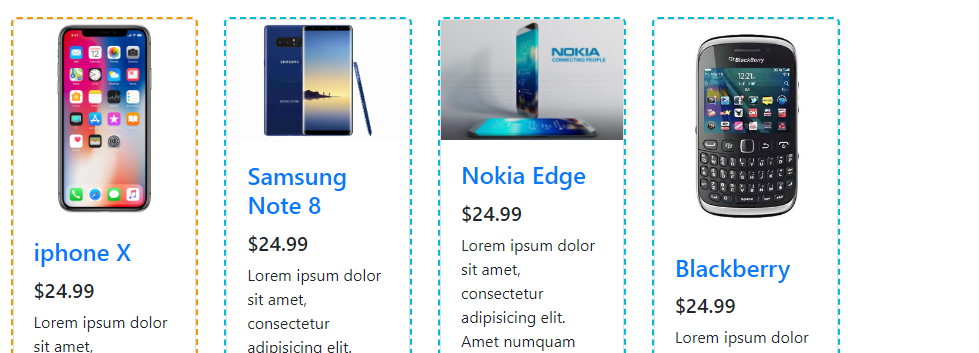
CSS locator = div[class=’card h-100’] div h4 a

*To convert into xpath* just put @ before class, double forward slash // before div and replace all spaces with forward slash /.

XPATH locator = //div[@class='card h-100']/div/h4/a

Q> **Perform click operation only on those cards which has title inside this array**?

 const products = ["iphone X", "Blackberry"];



Click operation should only occur on iphone X and Blackberry

describe("End to End Ecommerce application", async () => {

  it("Add items to cart", async () => {

    const products = ["iphone X", "Blackberry"];

    await browser.url("https://rahulshettyacademy.com/loginpagePractise/");

    await $("input[name='username']").setValue("rahulshettyacademy");

    const password = $("//input[@id='password']");

    await password.setValue("learning");

    await $("#signInBtn").click();

    //1. see if checkout button exist

    await $("\*=Checkout").waitForExist();

    //2. Collect all cards inside an array it will act as parent

    const cards = await $$(".card.h-100");

    await $(".card.h-100").waitForExist();

    //3. Loop goes through entire array

//make sure you use < not <= otherwise you will get undefined

    for (let i = 0; i < (await cards.length); i++) {

      //Title will be inside this variable

      const card = await cards[i].$("div h4 a");

      // 4. if products array includes card Title text it will return true

      if (products.includes(await card.getText())) {

        console.log("True");

        //perform click operation based on true IF condition

        await cards[i].$(".card-footer button").click();

      }

    }

  //5.Click on checkout cart to confirm if action is executed as per condition

    await $("\*=Checkout").click();

    await browser.pause(5000);

  });

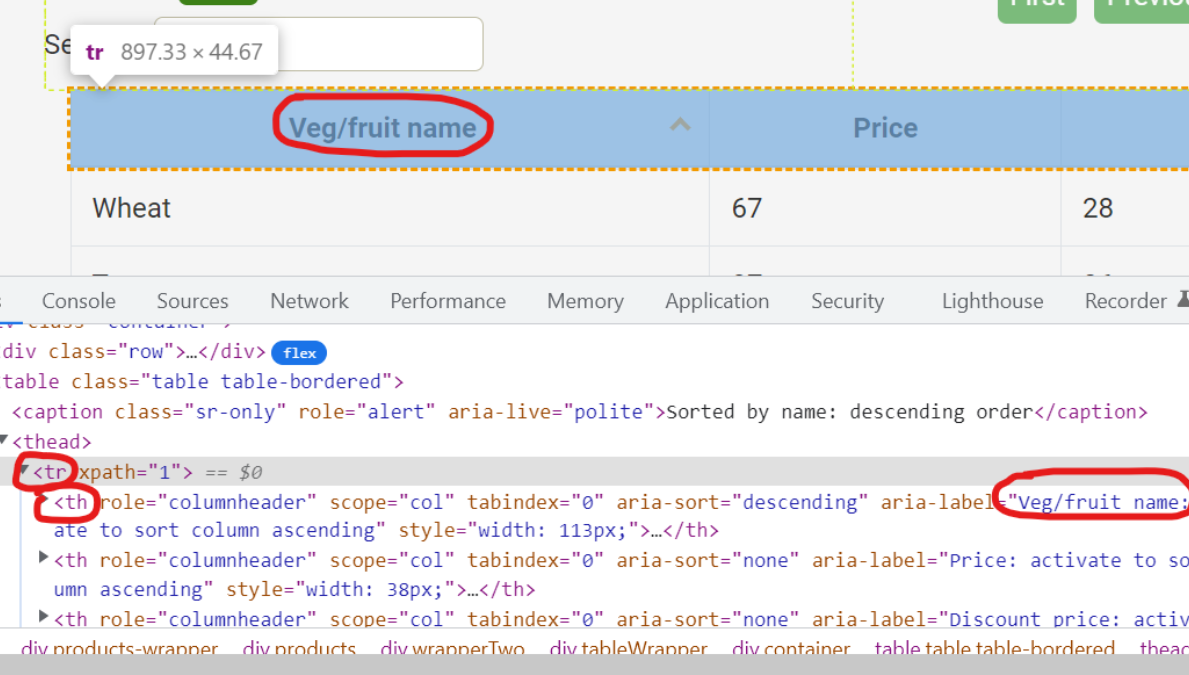
});

**Perform this same activity using different array methods like filter / map**

Note: If I use xpath instead of CSS selectors in above example, **code does not work expectedly in chrome browser but in msedge works properly!!!**

Q> **How to traverse from parent to child element with an example.**

1. Consider this page <https://rahulshettyacademy.com/seleniumPractise/#/offers>



**Notice** these tags <tr> which is table row and <th> which is table header. Here we want to traverse from <tr>(parent) to <th>(child)

Also, **notice** that there are three <th> child elements but we only want to target and click on the first one(Veg/Fruit name).

1. If we use xpath to traverse from parent to child, we will use this syntax

**//**parent**/**child

(Notice the double and single forward slash)

1. So in our case we will write locator like this

**//**tr**/**th (*but it locates all 3 headers*)

1. So, To target specifically first header **//**tr**/**th**[1]** (*array syntax*)

  it("check sorting", async () => {

    await browser.url(

      "https://rahulshettyacademy.com/seleniumPractise/#/offers"

    );

    (await $("//tr/th[1]")).click();

    await browser.pause(3000);

  });

Code will look like above

Q> **How sorting can be checked whether it is working correctly or not in the above table?**

A>

1. First thing after sorting the veg/fruit table after clicking on its table header, we will *retrieve* the list of veg/fruit names in an array; let us call it **array veggieNames**.
2. We will create a copy of the **array veggieNames** using slice method with no arguments, *so that while doing some kind of sorting on it, original array does not mutate*. This new array will be called **veggies**
3. Now we will *sort* the array and store its value in **array** called **sortedVeggies**.
4. Finally, we *compare* **array veggieNames** with **array sortedVeggies**. If they match, sorting has worked and Test case pass, otherwise not.

  it("check sorting", async () => {

    await browser.url(

      //1. Open the browser URL

      "https://rahulshettyacademy.com/seleniumPractise/#/offers"

    );

    (await $("//tr/th[1]")).click(); //2. Click the sort button

    const veggiesLocator = await $$("//tr/td[1]"); //3. Array of locators

    await browser.pause(3000);

    const veggieNames = await Promise.all(

      //4. All the actual values mapped from Array of locators

      veggiesLocator.map(async (veggie) => await veggie.getText())

    );

    const veggies = veggieNames.slice(); //5. Creates a copy of the above array

    console.log(veggies);

    const sortedVeggies = await veggies.slice().sort(); //6. Creates a sorted copy of array

    await browser.pause(3000);

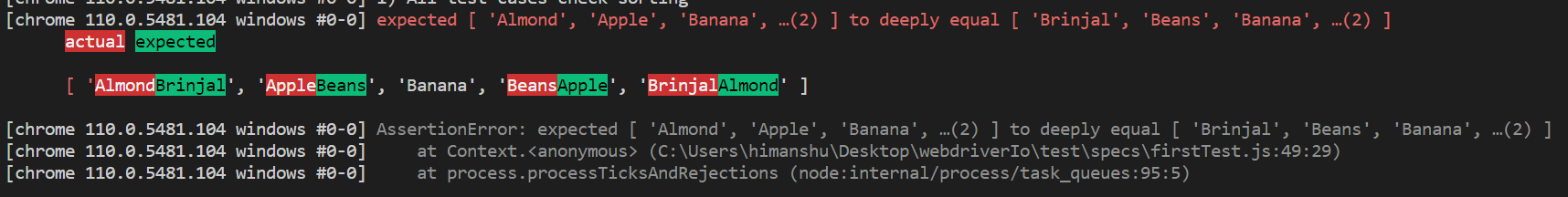
    expect(veggies).to.deep.equal(sortedVeggies); //7. deep equal assertion to match values at index of both arrays

To make sure that everything is working correctly I also created a reverse order of same array as **reverseVeggies** and compared with original array of objects. Test should fail with an assertion error in this case.

  const reverseVeggies = await veggies.slice().reverse();

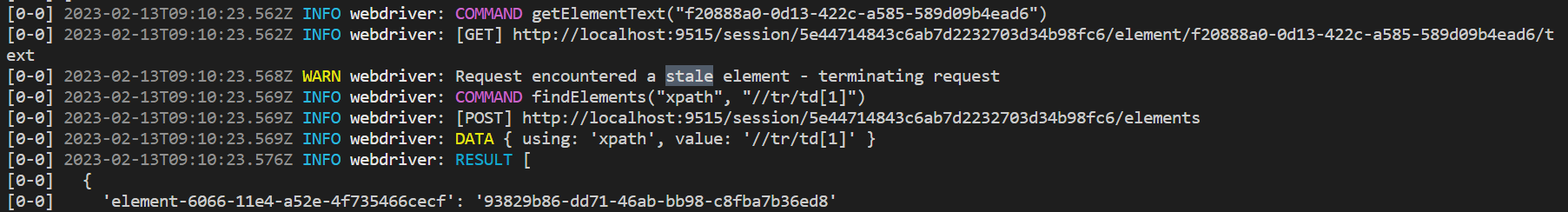
    expect(veggies).to.deep.equal(reverseVeggies); //7. deep equal assertion to match values at index of both arrays

**O/P:**



Q> **How do you solve staleElementException**?

A>



I encountered this error when I tried to getText from a table column *after sorting it*.

This webdriver warning arise when there has been a change on the DOM on the fly.

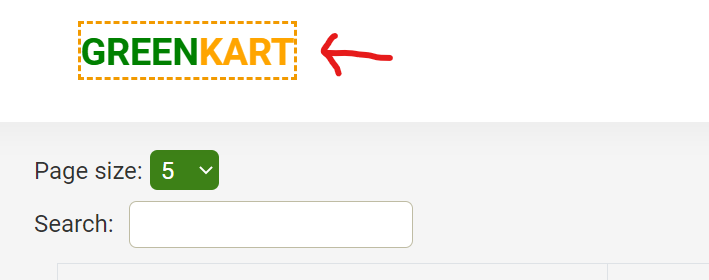
E.g., here sorting operation takes place, which removes old elements and add new DOM elements at the run time.

So when webdriver try to locate and get text out of these elements, it is unable to find it.

**Solution:** *Apply some implicit or explicit waits and let the elements load properly before applying any operation on them*.

Q> **How to use chai assertion to pass/fail a test depending on whether a particular string on our webpage matches with required string**?

Take this div on this page for example:



Here we will use assertion for given text “GREENKART”

1. First we will require expect module from chai library.

var expect = require("chai").expect;

1. Next we write the below code which invokes browser, URL, get text from target div and finally use assertion.

it("should check the brand logo name", async () => {

    await browser.url(

      "https://rahulshettyacademy.com/seleniumPractise/#/offers"

    );

    let logo = await $(

      "//div[@class='brand greenLogo']//div[@class='brand greenLogo']"

    );

    let logoName = await logo.getText();

    await browser.pause(3000); //just an implicit wait

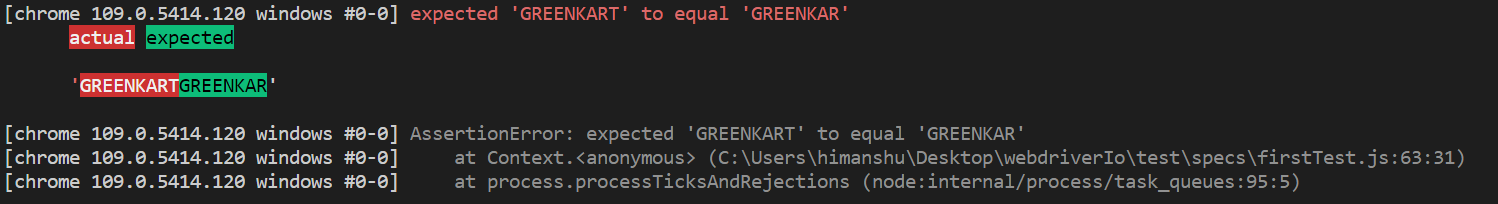
    await expect(logoName).to.equal("GREENKAR");

  })

1. In last expect statement, I used wrong string on purpose to show how webdriverIo handles assertion error. So we get below output and Test case fails.

**Note**: the syntax of assertion statement is

expect(foo).to.equal('bar');



1. In case of correct string in assertion statement test case is passed

Q> **How to use dragAndDrop method**?

A>

1. In this below example, we perform drag and drop operation on an element inside iframe so we first use switchToFrame command in order to switch into iframe.

Synatx of of the dragAndDrop method is

$(selector).dragAndDrop(target, { duration })

Here selector is the element you wish to drag and target is x, y coordinates of the drop location.

**Note:** In order to get x, y coordinates on a webpage, copy/paste below code in console

document.onmousemove = function(e){

var x = e.pageX;

var y = e.pageY;

e.target.title = "X is "+x+" and Y is "+y;

};

describe("All test cases", async () => {

  it("drag and drop", async () => {

    await browser.url("https://www.globalsqa.com/demo-site/draganddrop/");

    await browser.switchToFrame(

      await $("//iframe[@class='demo-frame lazyloaded']")

    );

    await $("//img[@alt='The peaks of High Tatras']").dragAndDrop({

      x: 500,

      y: 61,

    });

    await browser.pause(3000);

  });

});

Q> **How to handle this scenario when are getting Promise {<pending>} instead of actual values and we want to run some operations on those values**?

**Problem Statement**

const veggieNames = await veggiesLocator.map(

      async (veggie) => await veggie.getText()

    );

I am trying to map all the items from an array of locators veggiesLocator into another array veggieNames with only their text value.

After mapping, if I console.log(veggieNames[0]), I get

[0-0] Promise { 'Wheat' }

Instead, it should be simply [0-0] Wheat

I can also see this in console after running my program

[0-0] [

[0-0] Promise { <pending> },

[0-0] Promise { <pending> },

[0-0] Promise { <pending> },

[0-0] Promise { <pending> },

[0-0] Promise { <pending> }

[0-0] ]

**How to resolve it?**

A>

1. We need to await all the promises to be resolved for veggiesLocators array
2. Promise.all() method is useful when you have more than one promise and your code wants to know when all the operations that those promises represent have finished successfully.

So using this method as below, we get the intended output array

 const veggieNames = await Promise.all(

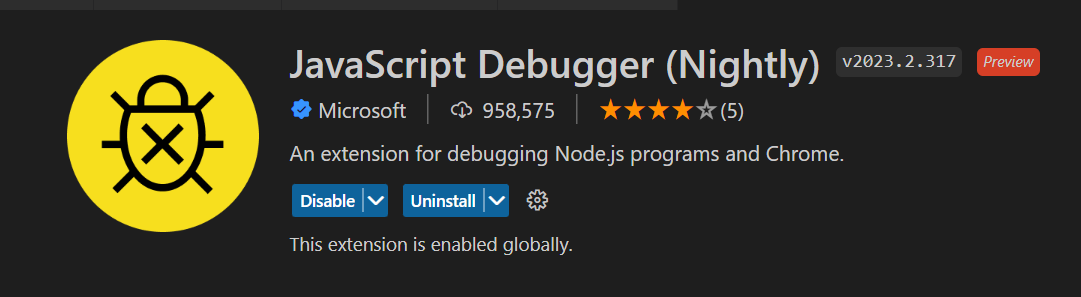
      veggiesLocator.map(async (veggie) => await veggie.getText())

    );

O/P: [0-0] Wheat

Q> **How to use debugger in vscode for debugging webdriverIo automation code**?

1. Install this extension



1. Create a folder called .vscode with filename launch.json and paste this code

{

  "configurations": [

    {

      "name": "webdriverIO test",

      "type": "node",

      "request": "launch",

      "args": ["wdio.conf.js"],

      "cwd": "${workspaceFolder}",

      "autoAttachChildProcesses": true,

      "program": "${workspaceRoot}/node\_modules/@wdio/cli/bin/wdio.js",

      "console": "integratedTerminal",

      "skipFiles": [

        "${workspaceFolder}/node\_modules/\*\*/\*.js",

        "${workspaceFolder}/lib/\*\*/\*.js",

        "<node\_internals>/\*\*/\*.js"

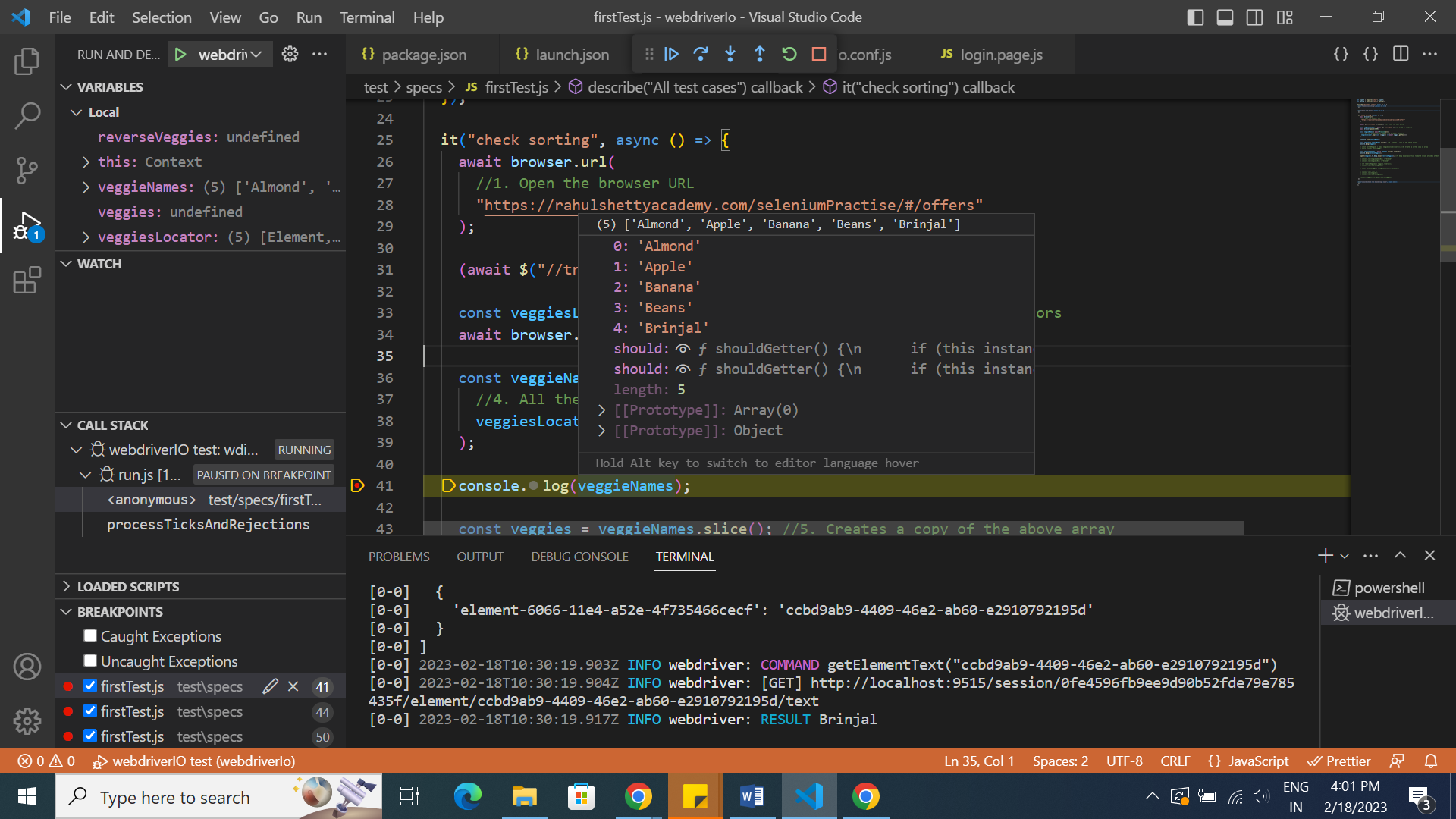
      ]

    }

  ]

}

1. Now insert breakpoints in code and start debugger.
2. If you hover over the code after debugging you can see the values associated with the code.



Q> **How to verify the text and total number of filtered results when we use a filter to search a specific item/items out of a list**?

A>

The complete program is below, read line-by-line comments to understand how everything is working.

  it("should validate filter result", async () => {

    await browser.url(

      //1. Open the browser URL

      "https://rahulshettyacademy.com/seleniumPractise/#/offers"

    );

    (await $("#search-field")).setValue("to"); //Enter value in filter to start searching

    const searchResultLocator = await $$("//tr/td[1]"); //All the filtered result locators saved in this variable

    await browser.pause(3000);

    const results = await Promise.all(

      //Get text from the locators

      searchResultLocator.map(async (result) => await result.getText())

    );

    function removeDuplicates(arr) {

      return [...new Set(arr)]; //Use Set method to remove duplicate items from the array

    }

    const newArray = removeDuplicates(results); //New array with only unique items

    console.log(removeDuplicates(results)); //Display this new array in console

    console.log(Object.keys(newArray).length); //Check length of this new array

    expect(newArray).to.have.lengthOf(2); //assertion for checking length of the array

    expect(newArray[1]).to.equal("Potato"); //assertion for checking value of string

  });

});

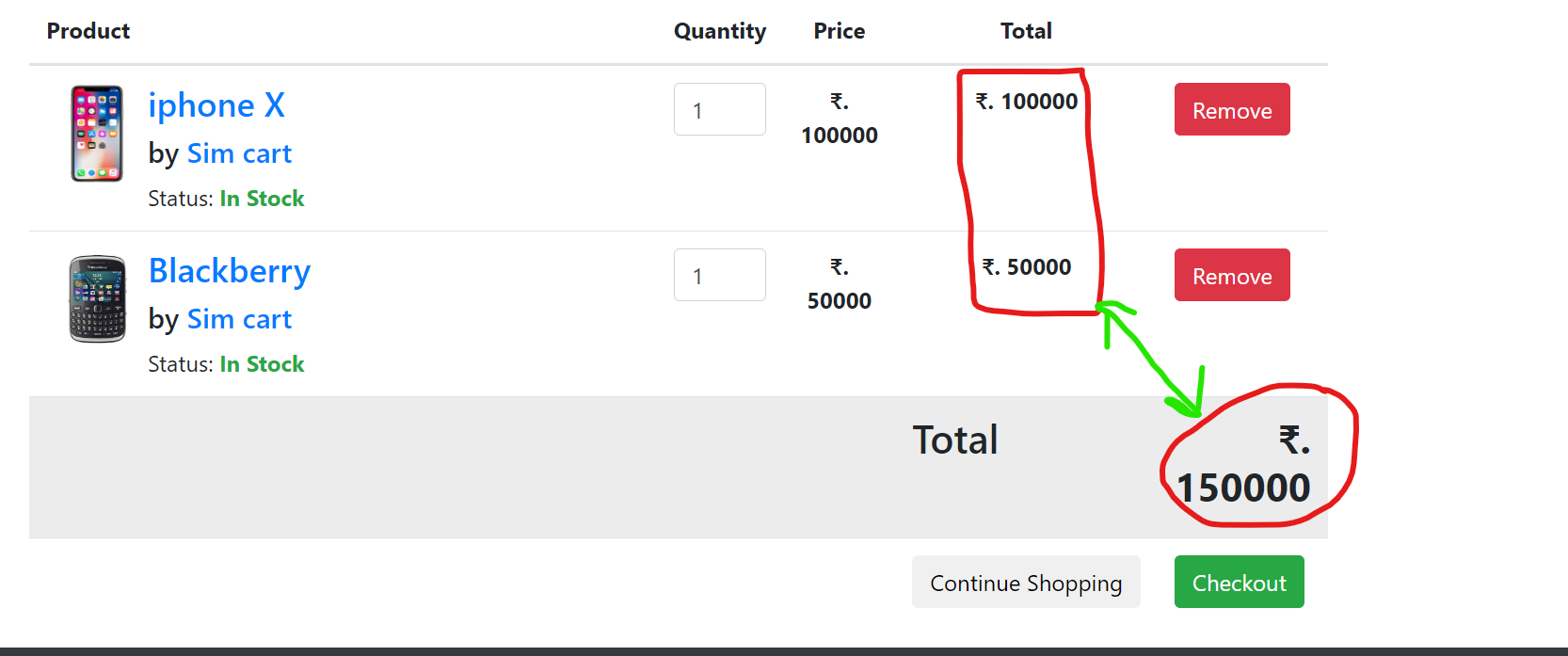
**Note: Try to do this without using implicit wait instead use explicit wait.**

Q> **Problem statement**

🡪Add 2 items in the cart

🡪Sum their individual prices

🡪Assert to check if sum of individual prices match with the Total price calculated by app.



A>

1. We need to first getText from the locators.
2. Now we can see the product price in form of *string*, which include a **₹** (rupee symbol), a dot (.) and some blank space. We need to remove these by using split and trim methods in order to get pure string.
3. After that, we get an array of strings, which we convert, to array of integers by using parseInt method.
4. Once we get array of integers, the sum of all items in this array will be calculated by using reduce method.

it("should validate sum of all item price in cart", async () => {

    const products = ["iphone X", "Blackberry"];

    await browser.url("https://rahulshettyacademy.com/angularpractice/shop");

    const cards = await $$("//div[@class='card h-100']");

    for (let i = 0; i < (await cards.length); i++) {

      //Title will be inside this variable

      const card = await cards[i].$("div h4 a");

      // 4. if products array includes card Title text it will return true

      if (products.includes(await card.getText())) {

        console.log("True");

        //perform click operation based on true IF condition

        await cards[i].$(".card-footer button").click();

      }

    }

    await $("\*=Checkout").click();

//PROBLEM STATEMENT STARTS FROM HERE

    const productsPrices = await $$("//tr//td[4]//strong");

    await browser.pause(3000);

    const sumOfProducts = await Promise.all(

      productsPrices.map(

        async (productPrice) =>

          await (await productPrice.getText()).split(".")[1].trim() //remove special character and whitespace from string

      )

    );

    const sumOfProductsNumbers = sumOfProducts.map((ele) => parseInt(ele)); //convert into string to integer

    const productPriceInt = sumOfProductsNumbers.reduce(

      (acc, sum) => acc + sum //sum of all items in array

    );

    const totalPrice = await (

      await $("td[class='text-right'] h3 strong")

    ) //get textTotal price locator

      .getText();

    const totalPriceInt = parseInt(totalPrice.split(".")[1].trim()); //split and trim

    await expect(productPriceInt).to.equal(totalPriceInt); //assertion

    await browser.pause(3000);

  });

Q> **How to make an explicit wait for an element to disappear or be hidden in order to do some operations**?

A>

You can use waitForDisplayed method with reverse: true

  const paginationButton = $("(//a[@role='button'])[6]");

    await paginationButton.waitForDisplayed({ reverse: true });

In this way, webDriver will wait until timeout period for the element to be removed from DOM and only then move to next line.

Q> **How to use chai-arrays to assert for arrays in webdriverIo**?

1. npm i --save-dev @types/chai-arrays.
2. To declare locally

let chai = require("chai");

const assertArrays = require("chai-arrays");

chai.use(assertArrays);

1. For more on chai-arrays

[**https://www.chaijs.com/plugins/chai-arrays/**](https://www.chaijs.com/plugins/chai-arrays/)

Q> **How to locate shadow DOM elements using webdriverIo**?

A>

1. If we inspect a webpage normally for Shadow DOM elements using browser in-built inspect tools, then we are unable to locate shadow elements.

The only way we can access these elements if we traverse from its parent to targeted shadow element.

1. This is the syntax we can use

$(Parent\_selector).shadow$(Child\_selector\_shadow)

As one can see, we chain the parent to child element and use a special keyword **shadow** at the junction.

**Real example:**



Here we can clearly see Parent light DOM and child Shadow DOM element.

1. To access this shadow element, we can write our test case this way.

  it("should input value in shadow element", async () => {

    await browser.url("https://books-pwakit.appspot.com/");

    await browser.pause(2000);

    const innerEl = await $("//book-app[@apptitle='BOOKS']").shadow$("#input");

    console.log(await innerEl.setValue("Hello"));

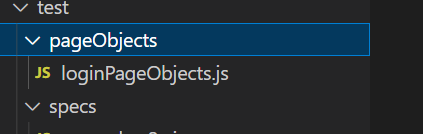
    await browser.pause(2000);

  });

Q> **Create a page object in webdriverIo and export properties and methods from this object into test file**.

A>

1. Create a new folder, which will hold page object files for respective test files. We call this folder as pageObjects.
2. Create a new file in this folder that will hold page objects for our login page. We call this loginPageObjects.js.



1. We start by declaring a simple class in our loginPageObjects.js file which we simply call as LoginPageObjects (first letter should be capital).

class LoginPageObjects {

1. Inside this class, we can define locators as class properties. We can define them simply inside a getter method.

get userName() {

    return $("//input[@id='username']");

  }

**Note:** *The get syntax binds an object property to a function, which we call when we need that property*.

1. If we want to access that property, which in our case is element locator, we simply need to call that getter funtion in this syntax.

console.log (**class\_name**.**function\_name**)

1. The getter functions will return locator values which can be called from anywhere in our project. However, we can also create separate methods, which uses these locators, and perform a specific task.

class LoginPageObjects {

  async browserUrl(path) {

    return await browser.url(path);

  }

  get userName() {

    return $("//input[@id='username']");

  }

  get userPassword() {

    return $("//input[@id='password']");

  }

  get signInButton() {

    return $("//input[@id='signInBtn']");

  }

  async Login(username, password) {

    await this.browserUrl("https://rahulshettyacademy.com/loginpagePractise/");

    await this.userName.setValue(username);

    await this.userPassword.setValue(password);

    await this.signInButton.click();

  }

}

Like this asynchronous Login method as above

1. After declaring and defining all these methods and properties in this class it is time to export this class as an object. We use below syntax

module.exports = new LoginPageObjects();

That is all from our page object file.

1. Now in order to make this class object accessible to our test files we will require it and assign a constant variable to it like this.

const login = require("../pageObjects/loginPageObjects");

1. And finally,

describe("Page object testing", async () => {

  it("should Open Browser URL and input credentials to access it", async () => {

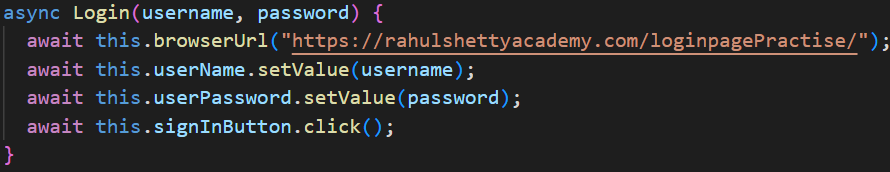
    await login.Login("rahulshettyacademy", "learning");

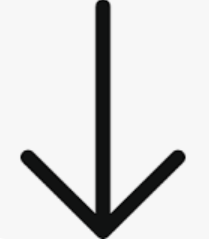
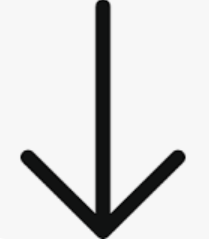
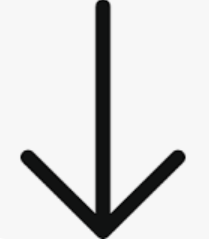
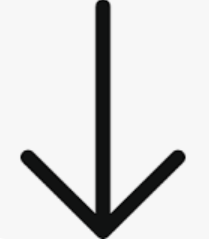
    await browser.pause(3000);

  });

});

As you can see this Login, method alone can perform four operations alone, which makes our code in test file more compact, easy to understand and most important *single place to modify* *in case frontend changes in future*.



 It becomes 



In our test file.

Q>**I want to take this a step further and even remove hardcoded credential values from my test file, how to do that**?

A>

We can do this by passing our credentials into a JSON file in *object-array* syntax and then parsing that JSON file’s content into our test file.

**Steps**:

1. Create a new folder called testData and create a new JSON file specific to a test spec file.
2. In our test file we have a test case by which we try to *login to an application using different username password combination*.
3. The JSON file will have different data sets, which will be, used to parametrize our test case. Let us call this file as LoginTest.json.

Inside this JSON file, we create two different datasets

[

  {

    "username": "rahulshettyacademy",

    "password": "learning123"

  },

  {

    "username": "rahulshettyacademy",

    "password": "learning"

  }

]

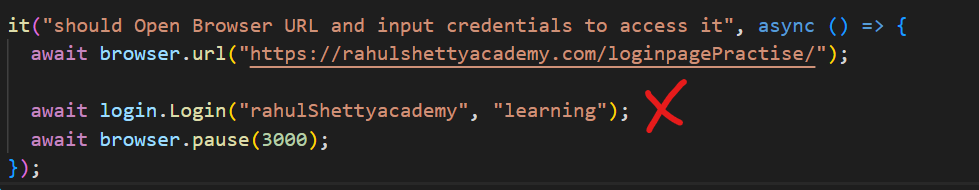
One uses incorrect password while the other one is correct one.

1. Now inside our test spec file it is time to parse contents of this LoginTest.json to get values of both data sets (‘*username’* and *‘password’*).
2. Therefore, first we use file system module or “fs” module of JavaScript and then call readFileSync method on it to get JSON file info. We save this information inside a variable called credentials.

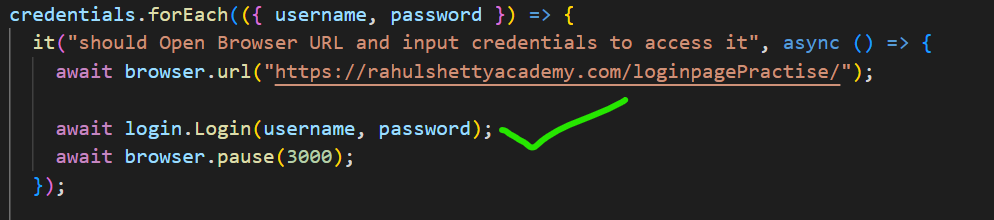
const fs = require("fs");

let credentials = JSON.parse(fs.readFileSync("test\_data/LoginTest.json"));

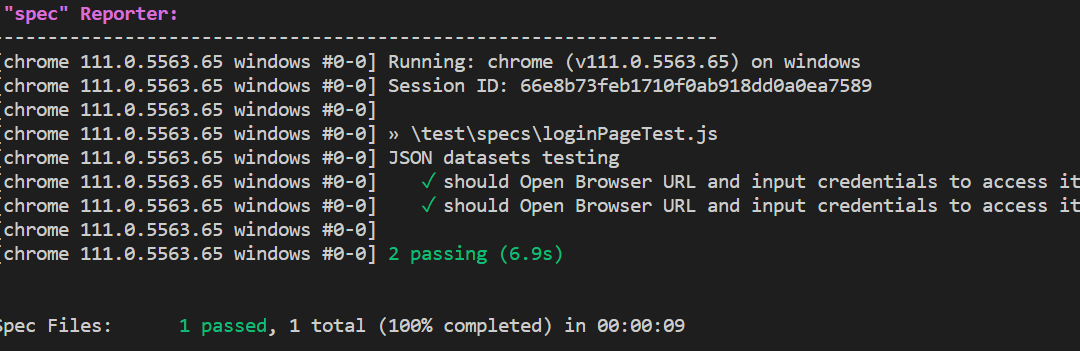
1. Our Original test case with hardcoded credentials looks like this



1. Therefore, we will use our JavaScript skills to feed data in a more *dynamic way* with the help for forEach method.



The same test case will run “*for each*” data set. Two times on our case.



**Running Tests in Parallel mode with effective utilization of capabilities**:

🡪 If we look at the value of maxInstances property in webdriverIo config file, it will tell us maximum number of browsers we can open at a time.

      maxInstances: 10,

🡪 But until now we have seen that only one browser is opening, It is because we have given only one spec file.

 specs: ["./test/specs/\*\*/loginPageTest.js"],

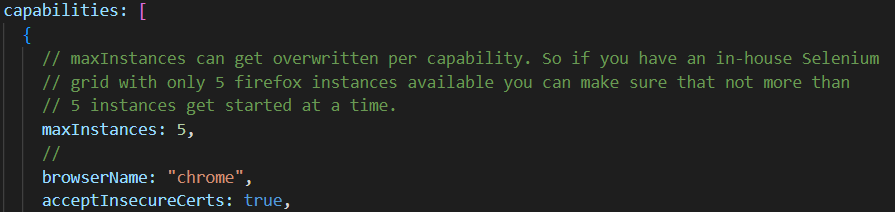
so if we give at most 10 spec files, their execution will start at parallel.

🡪To run all the files in specs folder

specs: ["./test/specs/\*\*/\*.js"]

Note: maxInstances are directly proportional to number of spec files that are run by webdriverIo **not** **test cases**.

🡪 There is another maxInstances property inside capabilities section



This maxInstances tells that only 5 instances can be opened on chrome, **what about rest**? Need to check(*maybe need to another object inside capabilities array for handling Firefox instance otherwise execution will not run in parallel but rather one by one*).

Q> **How to run chrome in headless mode**?

A> In Headless mode, test execution will start in the background but you will not see browser window invoking.

To enable headless mode, add this code below browserName,

      browserName: "chrome",

      'goog:chromeOptions': {

        args: ['headless', 'disable-gpu']

    },

Q> **Running selective tests using mocha grep options**?

A> Suppose we want to run sanity or smoke testing which includes only a few number of test cases or a specific product related test cases.

In this case we can add a keyword like ‘sanity’ or ‘smoke’ or ‘travel’ (*for travel related test cases*) in the it block like this,

  xit("check sorting -smoke", async () => {

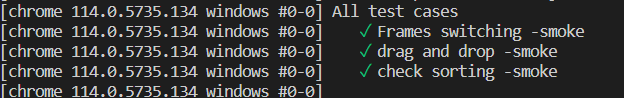
**‘-’** is not compulsory it is for representation purpose.

To run our test cases using webdriverIo framework we use this command

npx wdio run .\wdio.conf.js

Now to run test cases with -smoke in their title we will add another command line argument called –mochaOpts.grep smoke. So the complete command will look like this,

npx wdio run .\wdio.conf.js --mochaOpts.grep ***smoke***



Test cases with only –smoke keyword in their name are executed.

Q> **What is ‘bail’ parameter**?

A> Suppose we have 40 test cases running in parallel or one by one, But first 10 test cases are all showing failed which were passed earlier.

It is a sign that there is something seriously wrong in our application due to some recent changes or in our test suite.

*We can use ‘*bail’ *parameter to bail out of test execution by defining number of failed test cases*.

**Not working as expected**.

Q> **What is baseURL parameter**?

A>



Default baseURL is configured as <http://localhost>, So if all test cases in our test suite start with a particular URL like

https://rahulshettyacademy.com

we can provide the same URL here and remove it from test cases.



Q> **How to change logLevel** ?

A>

‘silent' is preferred, if you do not want too many logs.

Q> **How to select only some specific spec files while running test execution**?

A>

Create a new parameter in configuration file called suites,

  suites:{

    debitCard:['test/specs/firstTest.js','test/specs/loginPageTest.js']

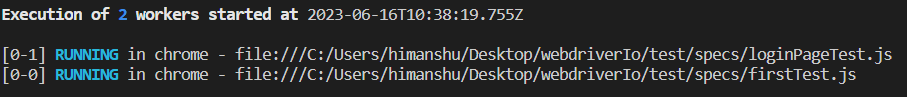
  },

Suppose we are running only debit Card related test cases which are in above two files.

In command line pass this command

npx wdio run .\wdio.conf.js --suite ***debitCard***

Only these two spec files will start executing which are in *debitCard* array of suites.



Q> **How to run any spec file or files from command line**?

A> Start by typing the run command

npx wdio run .\wdio.conf.js

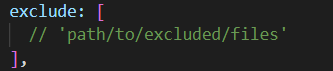
Then add a new command line parameter called spec followed by relative path of the spec file,



It will execute only that spec file. You can also add another spec file here by separating both files by a comma.

Q> **How to exclude a particular spec file from test execution**?

A> Use exclude parameter and copy the relative path of that particular spec file here.



Q> **Why we need multiple configuration files and how to create new configuration file**?

A>

Suppose we have two types of testing environment, QA and UAT for which have different base URL, timeout or some other parameter.

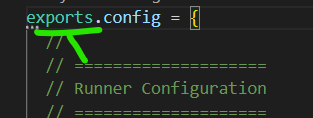
Until now if we want to change those parameter values, we have to change in our single WDIO configuration file which is a repetitive and time consuming process.

It is better to have different configuration file as per different testing environments so that we can switch from servers like Jenkins as per our requirement.

*How to do*:

🡪 Create a new file in root directory called wdio.*uat*.conf.js

🡪 We can inherit most of the properties from original wdio.conf.js file by merging parent config object and adding only new changes in wdio.*uat*.conf.js.



As you can see in our existing config file we are exporting config object. This object we have to *deep merge* with the new object.

🡪In our UAT config file, (npm install deepmerge)

const merge = require('deepmerge')

‘deepmerge’ is the node JS module to merge two objects together.

🡪 Add parent file,

const wdioConf = require('./wdio.conf.js')

🡪 use merge for merging *config* object in parent file with new object. Basically add new parameters you want to customize for this new config file.

merge(wdioConf.config, {

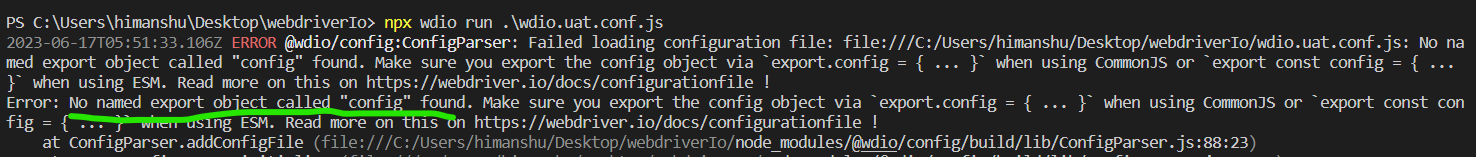
    baseUrl:'http://rahulshettyacademyUAT.com',

    waitforTimeout: 5000,

})

Here we want to change only baseUrl and waitforTimeout parameters.

🡪 run command npx wdio run .\**wdio.uat.conf.js**



🡪we see this error saying No named export object called config found. This means *we need to export data in form of object from UAT configuration file* and name it ‘config’.

exports.config = merge(wdioConf.config, {

    baseUrl:'http://rahulshettyacademyUAT.com',

    waitforTimeout: 5000,

    specs: ["./test/specs/\*\*/loginPageTest.js"]

})

Now run the same command again and we see test execution getting started.

Note: You can *grep* only selective test cases based on specific keywords in their test case title using mocha options.

    mochaOpts: {

        ui: "bdd",

        timeout: 60000,

        grep:'sanity'

      },

This way we do not have to mention those keywords in command line argument.

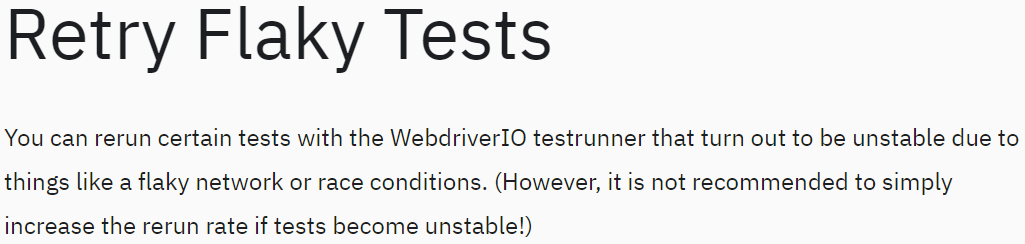
**Retry Mechanism in Mocha Framework to run Flaky tests**:

***What are flaky test***?

Test which fails too often or inconsistent due to environment issue(sometimes pass or fail). We can apply a retry mechanism in case it fails in order to run it again (*high probability that it might pass in second try*).

***How to apply retry mechanism*** ?

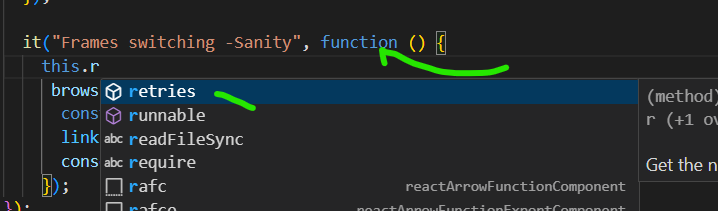
You can search official webdriverIo documentation



We will use this.retries() method and pass number of retries as argument.

Note1: If you want to retry entire spec file, call this method after *describe* statement at global level. In case you want to retry only the test case, add retries method after *it* statement.

Note2: This retries method will not appear with this keyword, if you have a fat arrow function, switch to JavaScript unbound function(*in fat arrow functions this is bound to current scope*).



 it("Frames switching -Sanity", function (){

    this.retries(2)

    browser.url("https://rahulshettyacademy.com/loginpagePractise/");

    const link = $("\*=Free");

    link.waitForExist();

**Code does not run as expected for retries**. My code passed even after giving wrong values because *I cannot use await statements inside unbound function*.

Q>**How to create our own scripts in package.json**?

A>

Suppose we want to run all UAT environment test cases, we write below command in console.

npx wdio run .\wdio.uat.conf.js

If we can shorten this command by giving it a simple name like uatenv, it would make our life a little easier.

🡪Open *package.json* file and navigate to “scripts” object.

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1",

  },

🡪Add a new parameter in this object “uatenv” and pair it with path that we write in console.

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1",

    "uatenv":"npx wdio run ./wdio.uat.conf.js"

  },

🡪 Now if we want to run this script, we simply have to write

npm run uatenv

and our UAT environment cases will start executing.

**Generating HTML reports through Allure package from webdriverIo**:

Official doc: <https://webdriver.io/docs/allure-reporter/>

🡪 Install allure reporter:

npm install @wdio/allure-reporter --save-dev

We will be able to see it in dev dependencies,

  "devDependencies": {

    "@wdio/allure-reporter": "^8.11.0",

🡪 set this configuration in our conf file.

  reporters: [

    [

      "allure",

      {

        outputDir: "allure-results", //.xml files will populate in this dir.

        disableWebdriverStepsReporting: true,

        disableWebdriverScreenshotsReporting: true,

      },

    ],

  ],

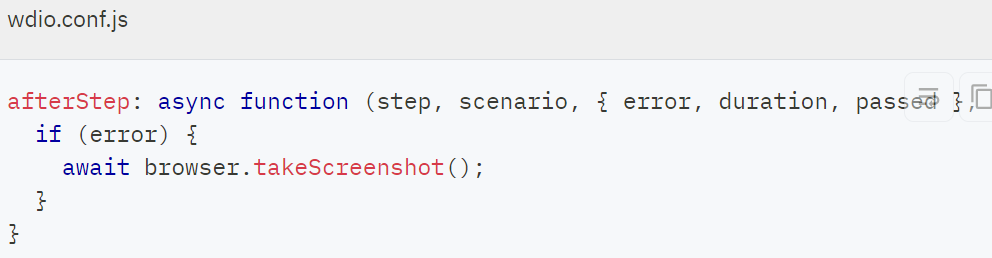
For each spec we will see a .xml file in output directory.

🡪 If you want to add the screenshot in report

disableWebdriverScreenshotsReporting: false,

🡪 and if you want to enable this feature only in case of failure, add this code in *afterTest* hook.()

In mocha we use *afterTest*, *in cucumber its afterStep hook*.



Change as per mocha…

  afterTest: async function (

    step,

    scenario,

    { error, duration, passed },

    context

  ) {

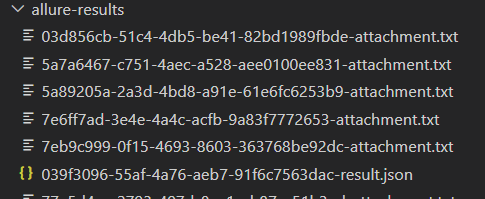
    if (error) {

      await browser.takeScreenshot();

    }

  },

🡪 Now we run test execution and we can see a folder allure-reports automatically generated.



🡪 There are many files in this folder, but we need a consolidated HTML report for that we need to download *allure command line tool*.

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

🡪 After installation is complete type this command for your directory,



In my case it is,

allure generate allure-results && allure open

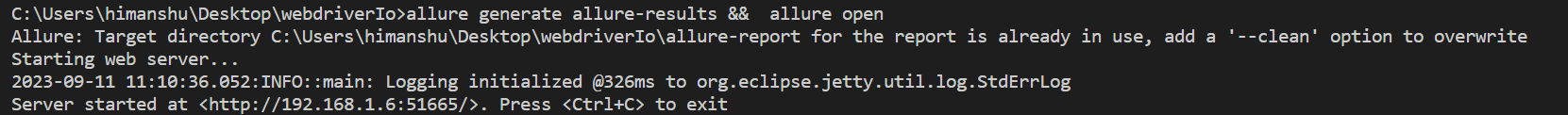
Note: && is not a valid argument if you run this command in powershell, use cmd or change the command

allure generate allure-results -and allure open

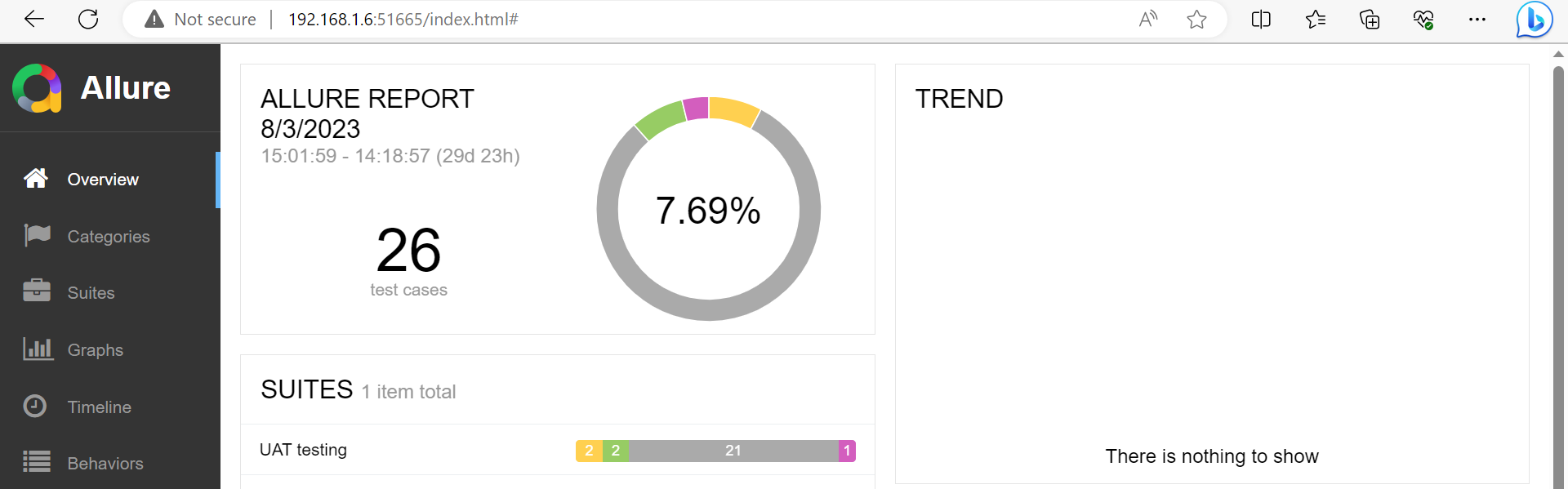
**Need to install Java Runtime Environment to run command line tool**.

After installing JRE, I ran this command in command prompt, inside my project directory,

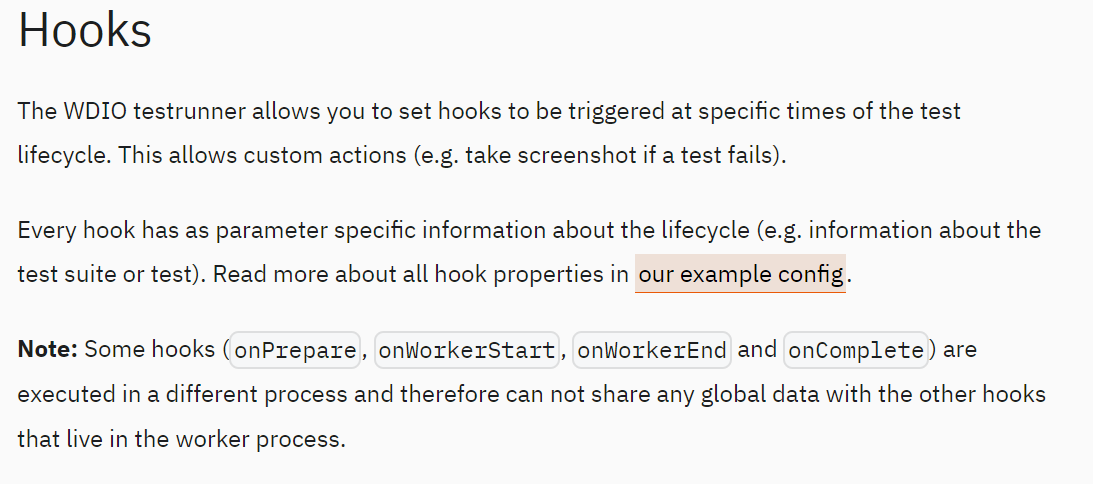
allure generate allure-results && allure open



And a webpage automatically opened at http://192.168.1.6:51665/index.html#,



What are hooks in webdriverIo?



In our test runner / config file we have hooks which is a piece of code that gets executed in the beginning or end of execution depending on type of that particular hook.

For example

  beforeSuite: function (suite) {

  },

This code will execute before any test suite is executed,

  beforeTest: function (test, context) {

  },

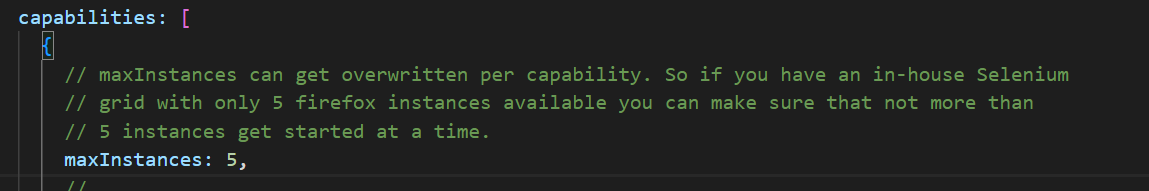
Code executed before any test is executed like delete cookies etc.

Q> **How many instances or spec files I can run at same time in webdriverIo**?

A>

Sometimes we need to run multiple tests in parallel

1. By default maxInstances property value is set to 10, which means we can run 10 different spec files at a time.



**Problem:**

Chromdriver issue, not able to create session

🡪 chromedriver --whitelisted-ips

This command will simply allow connection to all IPs (not recommended for security but can help solve the issue)

**Problem**:

I was running a custom WDIO config file

npx wdio run .\wdio.qa.conf.js

when I saw two worker process 0-0 and 1-0 being run



Although there was only one spec file and one test case I was currently executing.

Same case was executing two times!!!

**Solution**:

The issue was coming because capabilities array in main config file wdio.conf.js was not empty, It still had some values inside it like below,

  capabilities: [

    {

      browserName: "msedge",

      acceptInsecureCerts: true,

  ],

Making it empty solved the issue and now only one worker was started

  capabilities: [

  ],

**TO STUDY**

1. Chaining locators to reduce the scope of searching a locator throughout a webpage.
2. isSelected.
3. Handle pop ups. (lecture #35)
4. Take screenshot (getting TypeError: browser.saveScreenshot is not a function)
5. Chai assertions
6. Handle/Dismiss alerts

**Note:** to practice automation

<https://rahulshettyacademy.com/loginpagePractise/>

To practice all components on selectorsHub

<https://selectorshub.com/xpath-practice-page/>

To practice sorting

<https://rahulshettyacademy.com/seleniumPractise/#/offers>

**Every kind of XPATH available**

<https://www.lambdatest.com/blog/complete-guide-for-using-xpath-in-selenium-with-examples/#:~:text=XPath%2C%20also%20known%20as%20XML,page%20using%20HTML%20DOM%20structure>.

Topics need to cover

* XPath concepts and Writing XPath from basic to advance.
* Writing all kinds of XPath like XPath relative to other element, Relative XPath to Parent, index based xpath, XPath with attribute & text and absolute xpath.
* Detailed explanation about important functions of XPath like text(), . dot, normalize-space(), not(), !=
* Difference between //\* and //tagname
* Importance concepts about web like pseudo elements, comment, what is possible and what is not possible for web elements
* XPath 1.0 and XPath 2.0
* Shadow DOM, iframe, SVG, different technique to write xpath.
* How to automate and handle more complex scenarios like shadow dom inside iframe, iframe inside shadow dom, multiple shadow dom inside iframe etc.
* How to automate closed shadow dom
* Handling all kinds of web elements like dynamic, invisible dropdown and submenu etc.
* How to handle spin loader
* cssSelector, JS Path, jQuery
* Full training on SelectorsHub,
* How to write automation script smartly,
* Full training on TestCase Studio.
* How to complete manual testing without wasting much time and smartly.
* Bug reproducing techniques and a lot more.
* Interview preparation and questions on XPath & Selectors.
* After each class, practice questions and exercise will be shared. We will also discuss the exercise questions.
* Writing Axes Based XPath, XPath for Web Tables etc will be taught..