==

Tutor: **Rahul Shetty**

Reference: **UDEMY**

Course: **Playwright JS Automation Testing from Scratch with Framework**

==

1. Course URL: https://www.udemy.com/course/playwright-tutorials-automation-testing/
2. Document prepared by: **Rajat Verma**
    a. https://www.linkedin.com/in/rajat-v-3b0685128/
    b. https://github.com/rajatt95
    c. https://rajatt95.github.io/

Softwares:

1. Programming language - Javascript
2. IDE - VS Code
    a. https://code.visualstudio.com/download
    b. Plugin
        i. Playwright Test for VSCode
3. Engine:
    a. https://nodejs.org/en/download/
4. Playwright:
    a. https://playwright.dev/
    b. npm init playwright

----

1. **Learnings from Course (UDEMY - RS - Playwright-Javascript)**
    a. Links:
        i. Playwright:
            1. https://playwright.dev/
            2. https://playwright.dev/docs/intro
            3. https://playwright.dev/docs/actionability
            4. https://www.npmtrends.com/playwright
            5. https://github.com/Microsoft/playwright
            6. https://playwright.dev/docs/navigations
            7. https://playwright.dev/docs/selectors#text-selector
            8. https://playwright.dev/docs/api/class-test
            9. https://playwright.dev/docs/test-annotations
            10. https://trace.playwright.dev/
            11. https://playwright.dev/docs/screenshots

        12. https://playwright.dev/docs/videos
        13. https://playwright.dev/docs/api/class-testoptions#test-options-video
        14. https://playwright.dev/docs/test-reporters
    ii. Assertions:
        1. https://playwright.dev/docs/test-assertions
        2. https://playwright.dev/docs/test-assertions#locator-assertions-to-have-attribute

—-----------------

b. Playwright:

- **i.** Written on the Node.js platform.
- ii. Browsers:
    1. Chromium Engine (Chrome, Edge)
    2. WebKit -
        - a. **WebKit is an in-built browser that uses the Safari engine**
        - b. This works on WIN OS as well
    3. Firefox
    4. Opera
- iii. OS:
    1. WIN, MAC, Linux
- iv. Programming languages:
    1. Javascript, Typescript, Java, Python, C#
- v. Features:
    1. Auto-wait capability
    2. **Native Mobile automation**
        - a. Android - Chrome
        - b. iOS - Safari
    3. **Traces, Videos**
    4. **Inspector tool** - Debug mode
    5. API testing
        - **a. Making calls and extract response**
        - **b. Intercepting**
    6. **Browser Context (Example: Inject Cookies to Browser)**
    7. **Codegen tool** (Generates code for many languages)
    8. **Parallel Cross Browser testing**
    9. **Reporting:**
        - **a. Playwright-report**
        - **b. Integration with Allure**

—-----------------

—-----------------

c. **Basic concepts:**
   i. Importance of ***async-await***
      1. await is required when actual action is performed.
   ii. ***Browser Context*** and ***Page Fixture***
   iii. Fixtures:
      1. Browser, page, ddsdssd, dsadsadasd
   iv. How to select the Browser for tests execution

—-----------------

d. **Playwright configuration file:**
   i. playwright.config.js
      1. Configuration for everything is a part of this config object
         a. There are many properties in this config object:
      2. Properties
         a. **testDir** -> Where all the tests are present
         b. **timeout** ->
            i. Maximum time one test can run for
            ii. If the test in hanged due to some reason, then, it will be reported as a Failure
         c. **Block- expect** -> This is related to Assertions
            i. **timeout**
         d. **reporter** ->
            i. How do you want to report your test results
            ii. Other ways: JSON, etc.
         e. **Block- use** ->
            i. **trace -** This is for tracing (reported in Playwright report)
            ii. **headless -** This is for the execution mode
            iii. **screenshot -** To take screenshots at a different level
            iv. **viewport -** To set the dimension of the Browser
            v. **ignoreHttpsErrors -** This is to handle SSL certifications
            vi. **permissions : ['geolocation'] -** This is to give the permission for Location access
            vii. **video -** This is to record the videos of tests

—-----------------

—-----------------

e. **Playwright**:

   i. Default, it starts the test execution in headless mode.

      1. We can set the mode of execution

         a. During runtime,

            i. **npx playwright test --headed** (Runs the E2E tests in headed mode)

         b. In the configuration file,

            i. **use: {**

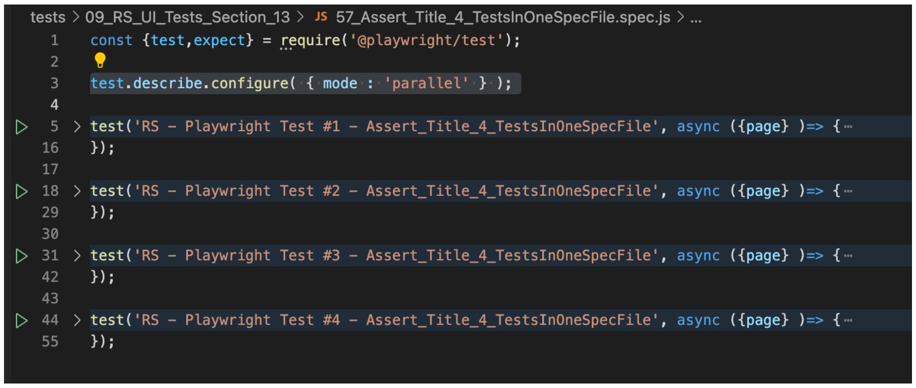
               headless:false/true

               }

   ii. Tests present in one spec file will execute sequentially, but, spec files will execute parallelly

      1. If you want to execute test cases in parallel:

         a. **test.describe.configure({ mode : 'parallel' });**

         b. 

```
tests > 09_RS_UI_Tests_Section_13 > JS 57_Assert_Title_4_TestsInOneSpecFile.spec.js > ...
const {test,expect} = require('@playwright/test');

test.describe.configure( { mode : 'parallel' } );

test('RS - Playwright Test #1 - Assert_Title_4_TestsInOneSpecFile', async ({page} )=> {…
});

test('RS - Playwright Test #2 - Assert_Title_4_TestsInOneSpecFile', async ({page} )=> {…
});

test('RS - Playwright Test #3 - Assert_Title_4_TestsInOneSpecFile', async ({page} )=> {…
});

test('RS - Playwright Test #4 - Assert_Title_4_TestsInOneSpecFile', async ({page} )=> {…
});
```

      2. Execution modes:

         a. Parallel: **npm run test_single_RS_4_TestsInOneSpecFile_Parallel**

         b. Default: **npm run test_single_RS_4_TestsInOneSpecFile_Default**

         c. Serial: **npm run test_single_RS_4_TestsInOneSpecFile_Serial**

      3. NOTES:

         a. In the **Serial (Inter-Dependent)** mode case,

            i. If 2nd test case is failed,

               1. Then, the 3rd and 4th test case will be skipped.

            ii. In the default mode,

               1. Test cases will not be skipped.

   iii. If you want to run only 1 test case present in the spec file, then.

      1. **test.only**

—-----------------

—-----------------

f. **Test Annotations:**
   i. https://playwright.dev/docs/test-annotations
      1. **test.beforeAll(() => {});**
         a. Executes 1st in the spec file or before any test case
      2. **test.beforeEach(() => {});**
         a. Executes before each and every test case
      3. **Test.only**

—-----------------

g. **Playwright methods:**
   i. Navigate to application:
      1. **page.goto("https://www.google.com/");**
   ii. Go Back and Forward:
      1. **await page.goBack();**
      2. **await page.goForward();**
   iii. Fill value in textbox
      1. **await page.locator('#password').type('learning');**
   iv. Type in textbox slowly:
      1. **drpdwn_selectCountry.type('ind', { delay:1000 });**
   v. Clear and then, fill value in textbox
      1. **await page.locator('#password').fill('learning');**
   vi. Click on element
      1. **await page.locator('#signInBtn').click();**
   vii. Click on Visible element:
      1. **page.locator(" li a[href*='lifetime-access']:visible").click();**
   viii. Mouse Hover on element
      1. **await page.locator('#signInBtn').hover();**
   ix. Extract the text of element:
      1. **await page.locator('[style*=block]').textContent());**
   x. Get first element from multiple elements
      1. **console.log(await page.locator('.card-body a').nth(0).textContent());**
      2. **console.log(await page.locator('.card-body a').first().textContent());**
   xi. Get last element from multiple elements
      1. **console.log(await page.locator('.card-body a').last().textContent());**
   xii. Get text of all the elements found:
      1. **await title_products.allTextContents());**
         a. // -> This will get the title of all the elements and put into one array
         b. // -> Playwright does not auto-wait for this method; This will return an empty Array

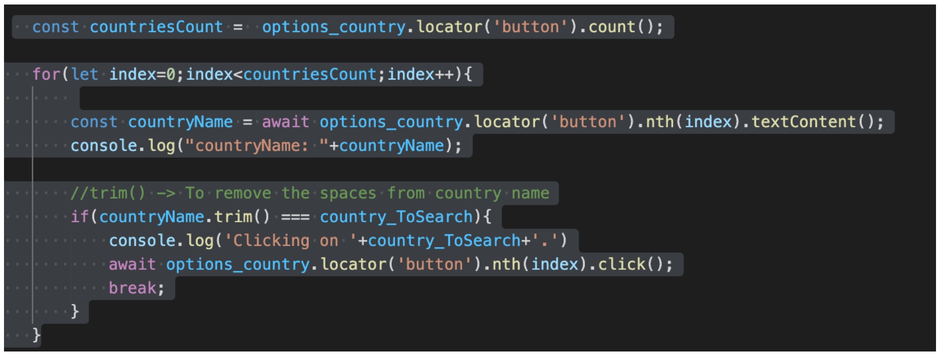
xiii. Wait for Page to load

1. **await page.waitForLoadState('networkidle');**
2. **await page.waitForLoadState('domcontentloaded');**
3. **await page.waitForLoadState('load');**

xiv. Wait for element:

1. **await page.locator('div li').waitFor();**
   a. This looks for one element

xv. Dropdown:

1. Static:
   a. //consult : <option value="consult">Consultant</option>
   await drpdwn_role.selectOption('consult');
2. Auto-Suggestive:
   a. 

```
  const countriesCount =  options_country.locator('button').count();

  for(let index=0;index<countriesCount;index++){
      
      const countryName = await options_country.locator('button').nth(index).textContent();
      console.log("countryName: "+countryName);

      //trim() -> To remove the spaces from country name
      if(countryName.trim() === country_ToSearch){
          console.log('Clicking on '+country_ToSearch+'.')
          await options_country.locator('button').nth(index).click();
          break;
      }
  }
```

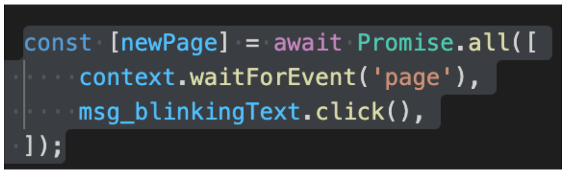
xvi. Radio button:

1. **console.log(await radioBtn_user.isChecked());**

xvii. Checkbox:

1. **console.log(await checkBox_terms.isChecked());**

xviii. Child window:

1. 

```
const [newPage] = await Promise.all([
    context.waitForEvent('page'),
    msg_blinkingText.click(),
]);
```

xix. Elements count:

1. **page.locator('.card-body').count();**

xx. Ways to find element on the page using text

1. **page.locator("text=Add to Cart")**
2. **page.locator("h3:has-text("+productName+")")**

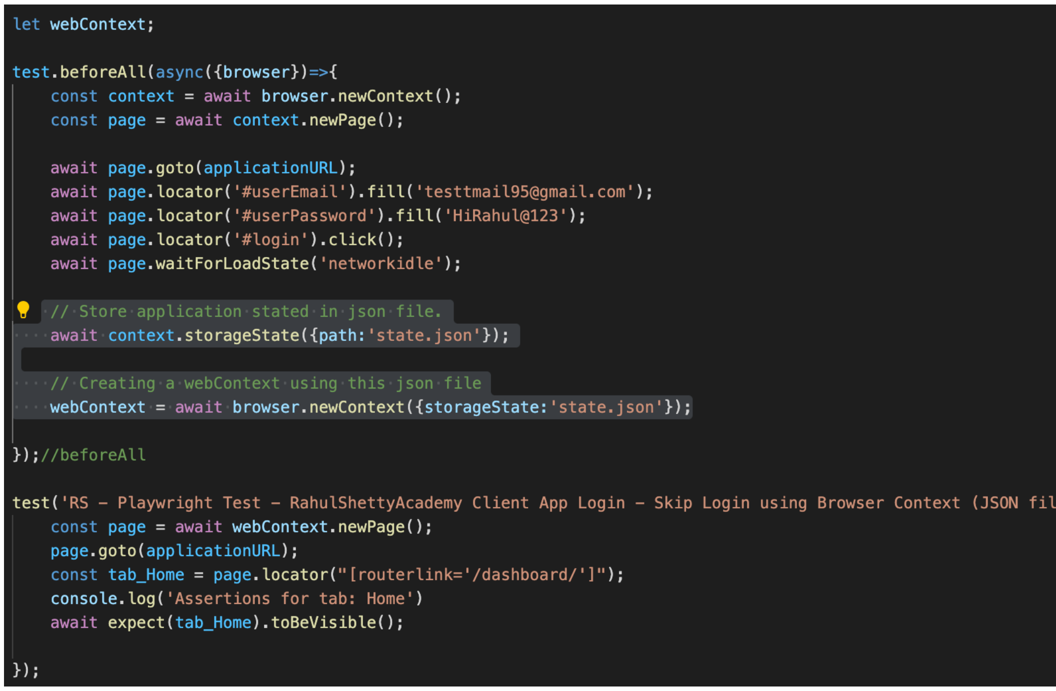
xxi. Accept/Dismiss Alert/Popup/Dialog:

1. **page.on('dialog' , dialog => dialog.accept());**
2. **page.on('dialog' , dialog => dialog.dismiss());**

xxii. Frames:

1. **const frame_courses = page.frameLocator('#courses-iframe');**
   a. //courses-iframe -> This is the ID of this frame

xxiii. Use Browser state (Local storage, Session storage, Cookies)

1. 

```
let webContext;

test.beforeAll(async({browser})=>{
    const context = await browser.newContext();
    const page = await context.newPage();

    await page.goto(applicationURL);
    await page.locator('#userEmail').fill('testmail95@gmail.com');
    await page.locator('#userPassword').fill('HiRahul@123');
    await page.locator('#login').click();
    await page.waitForLoadState('networkidle');

    // Store application stated in json file.
    await context.storageState({path:'state.json'});

    // Creating a webContext using this json file
    webContext = await browser.newContext({storageState:'state.json'});

});//beforeAll

test('RS - Playwright Test - RahulShettyAcademy Client App Login - Skip Login using Browser Context (JSON fil
    const page = await webContext.newPage();
    page.goto(applicationURL);
    const tab_Home = page.locator("[routerlink='/dashboard/']");
    console.log('Assertions for tab: Home')
    await expect(tab_Home).toBeVisible();

});
```

   a. Store application stated in json file:
      i. **await context.storageState({path:'state.json'});**
   b. Now, we have to invoke the Browser using this JSON file:
      i. Creating a webContext using this json file
      ii. **webContext = await browser.newContext({storageState:'state.json'});**
   c. Use this **webContext** to create the page fixture in the Test case:
      i. **const page = await webContext.newPage();**
      ii. **page.goto(applicationURL);**

xxiv. Capture screenshot and put in some folder:

1. Full Page:
   a. **await page.screenshot({path:'./screenshots/Screenshot_FullPage.png', fullPage: true});**
2. Visible Screen:
   a. **await page.screenshot({path:'./screenshots/Screenshot_FullPage.png'});**
3. Element level:
   a. **await btn_hide.screenshot({path:'./screenshots/Screenshot_FullPage.png'});**
4.

xxv. Compare 2 images:

1. **expect(await page.screenshot()).toMatchSnapshot('uk.flightaware-prod.png');**

—------------------

h. **Assertions:**

   i. Page title:
      1. **await expect(page).toHaveTitle('Google');**

   ii. Element Text:
      1. **await expect(page.locator('[style*=block]')).toHaveText('Incorrect username/password.');**

   iii. Element Partial Text:
      1. **await expect(page.locator('[style*=block]')).toContainText('Incorrect username/password.');**

   iv. Radio button to be checked:
      1. **await expect(radioBtn_user).toBeChecked();**

   v. Checkbox to be checked:
      1. **await expect(checkBox_terms).toBeChecked();**

   vi. Expecting false
      1. **expect(await checkBox_terms.isChecked()).toBeFalsy();**

   vii. Expecting true
      1. **expect(await checkBox_terms.isChecked()).toBeTruthy();**
      2. **expect(orderID.includes(orderID_order_summary_page)).toBeTruthy();**

   viii. Attribute value:
      1. **await expect(btn_SignIn).toHaveAttribute('name', 'signin');**

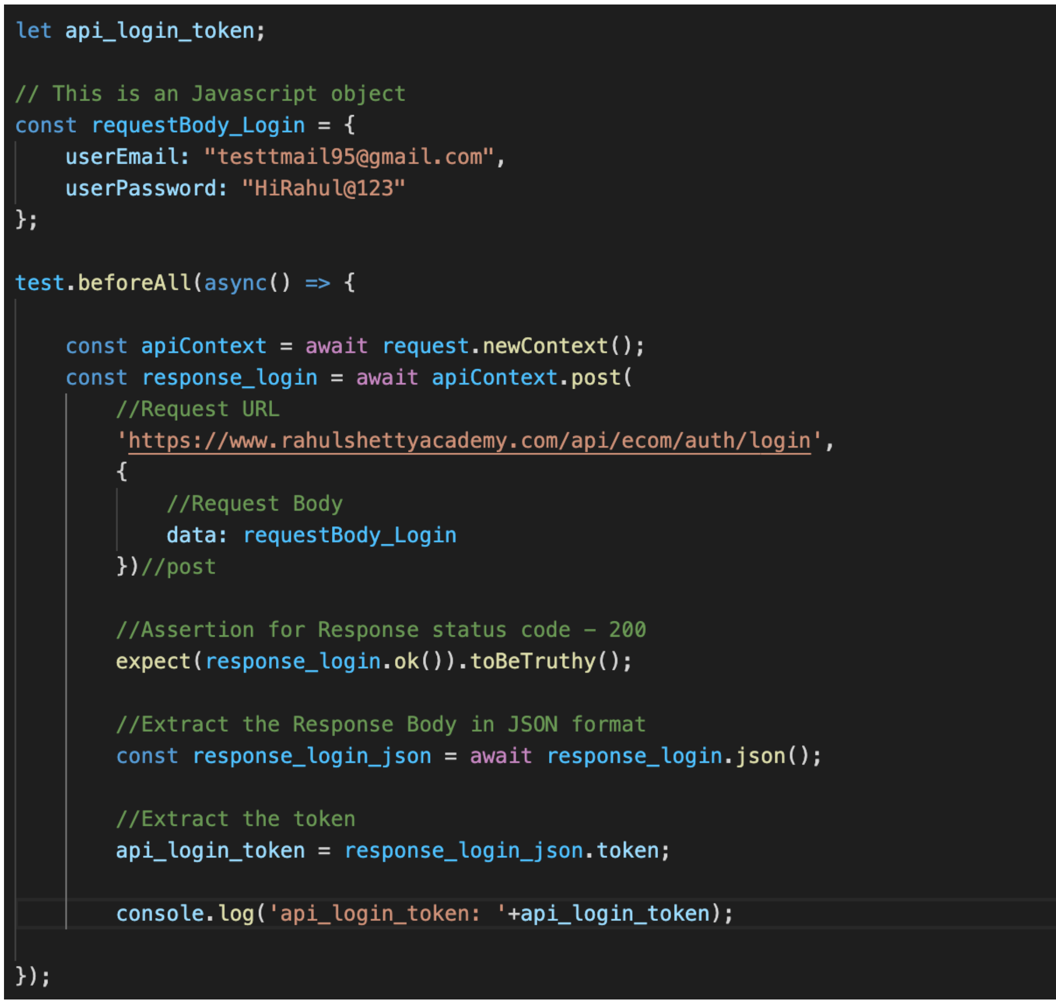
   ix. Element Visible or Hidden:
      1. **await expect(txtBox_hide_show_example).toBeVisible();**
      2. **await expect(txtBox_hide_show_example).toBeHidden();**

—------------------

—-----------------

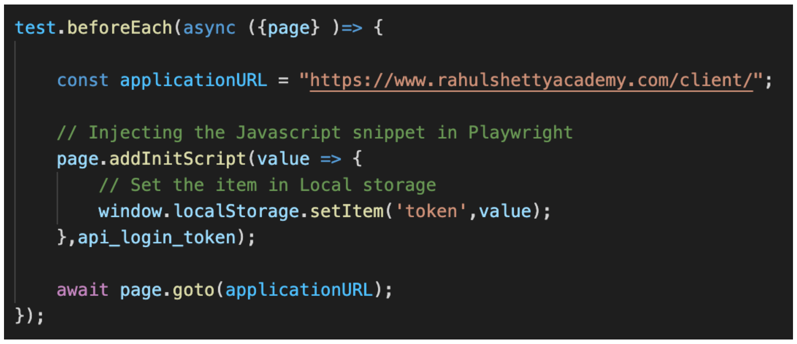
i. **Playwright with API:**

   i. Call Login API and extract the token from the Response body:

      1. 

```
let api_login_token;

// This is an Javascript object
const requestBody_Login = {
    userEmail: "testtmail95@gmail.com",
    userPassword: "HiRahul@123"
};

test.beforeAll(async() => {

    const apiContext = await request.newContext();
    const response_login = await apiContext.post(
        //Request URL
        'https://www.rahulshettyacademy.com/api/ecom/auth/login',
        {
            //Request Body
            data: requestBody_Login
        })//post

        //Assertion for Response status code - 200
        expect(response_login.ok()).toBeTruthy();

        //Extract the Response Body in JSON format
        const response_login_json = await response_login.json();

        //Extract the token
        api_login_token = response_login_json.token;

        console.log('api_login_token: '+api_login_token);

});
```

   ii. Inject the token into Browser's local storage:

      1. 

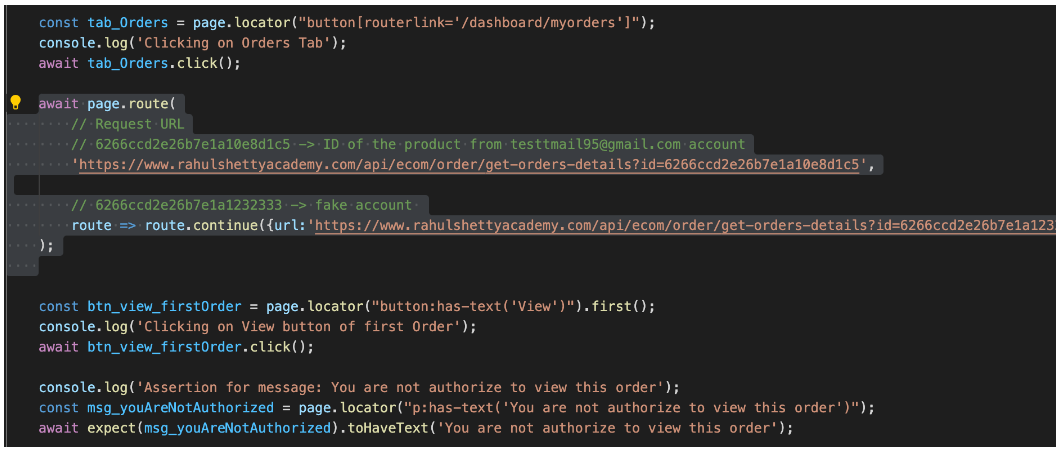
```
test.beforeEach(async ({page} )=> {

    const applicationURL = "https://www.rahulshettyacademy.com/client/";

    // Injecting the Javascript snippet in Playwright
    page.addInitScript(value => {
        // Set the item in Local storage
        window.localStorage.setItem('token',value);
    },api_login_token);

    await page.goto(applicationURL);
});
```

—-----------------

—-----------------

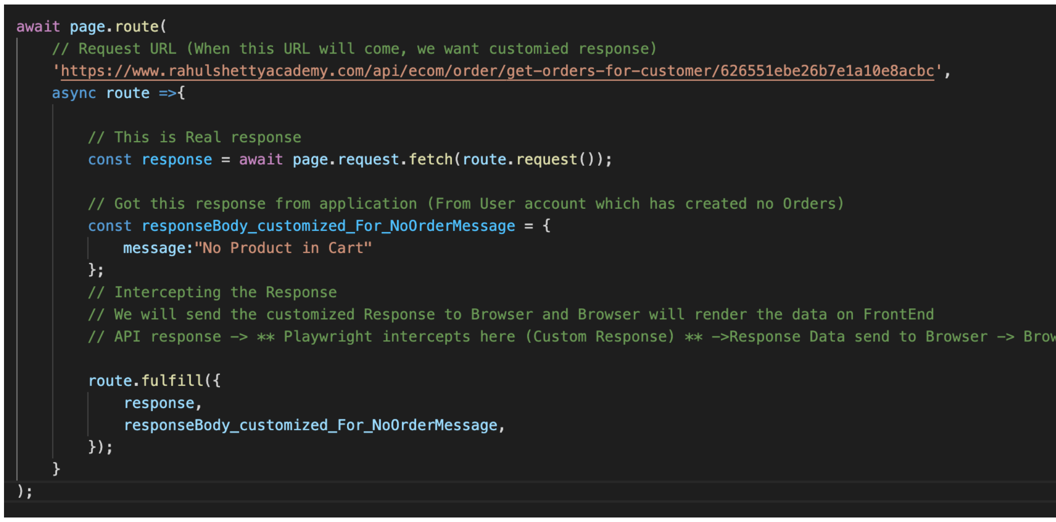
j. Intercepting:

i. Customize/Alter Request URL:

1. 

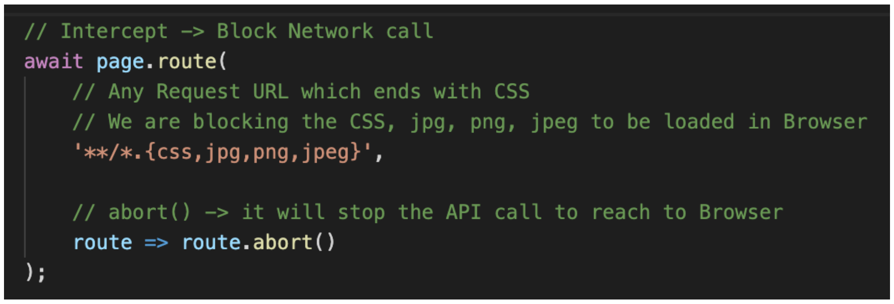
```
const tab_Orders = page.locator("button[routerlink='/dashboard/myorders']");
console.log('Clicking on Orders Tab');
await tab_Orders.click();

await page.route(
        // Request URL
        // 6266ccd2e26b7e1a10e8d1c5 -> ID of the product from testtmail95@gmail.com account
        'https://www.rahulshettyacademy.com/api/ecom/order/get-orders-details?id=6266ccd2e26b7e1a10e8d1c5',

        // 6266ccd2e26b7e1a1232333 -> fake account
        route => route.continue({url:'https://www.rahulshettyacademy.com/api/ecom/order/get-orders-details?id=6266ccd2e26b7e1a123
    );

const btn_view_firstOrder = page.locator("button:has-text('View')").first();
console.log('Clicking on View button of first Order');
await btn_view_firstOrder.click();

console.log('Assertion for message: You are not authorize to view this order');
const msg_youAreNotAuthorized = page.locator("p:has-text('You are not authorize to view this order')");
await expect(msg_youAreNotAuthorized).toHaveText('You are not authorize to view this order');
```

ii. Customize/Alter Response body:

1. 

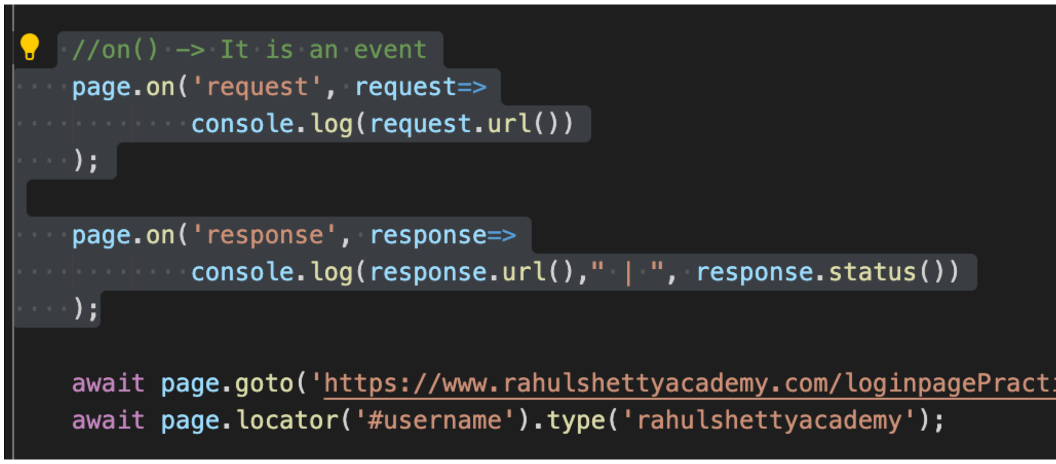
```
await page.route(
    // Request URL (When this URL will come, we want customied response)
    'https://www.rahulshettyacademy.com/api/ecom/order/get-orders-for-customer/626551ebe26b7e1a10e8acbc',
    async route =>{

        // This is Real response
        const response = await page.request.fetch(route.request());

        // Got this response from application (From User account which has created no Orders)
        const responseBody_customized_For_NoOrderMessage = {
            message:"No Product in Cart"
        };
        // Intercepting the Response
        // We will send the customized Response to Browser and Browser will render the data on FrontEnd
        // API response -> ** Playwright intercepts here (Custom Response) ** ->Response Data send to Browser -> Brow

        route.fulfill({
            response,
            responseBody_customized_For_NoOrderMessage,
        });
    }
);
```

iii. Abort Network calls: Blocking CSS to be loaded in Browser:

1. 

```
// Intercept -> Block Network call
await page.route(
    // Any Request URL which ends with CSS
    // We are blocking the CSS, jpg, png, jpeg to be loaded in Browser
    '**/*.{css,jpg,png,jpeg}',

    // abort() -> it will stop the API call to reach to Browser
    route => route.abort()
);
```

iv. Log all the Request URLs and Response status codes:

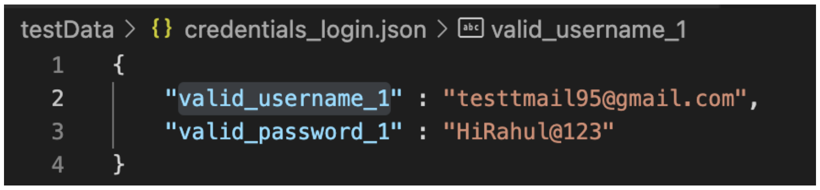
1. 

```
//on() -> It is an event
page.on('request', request=>
          console.log(request.url())
);

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

await page.goto('https://www.rahulshettyacademy.com/loginpagePract
await page.locator('#username').type('rahulshettyacademy');
```

—------------------

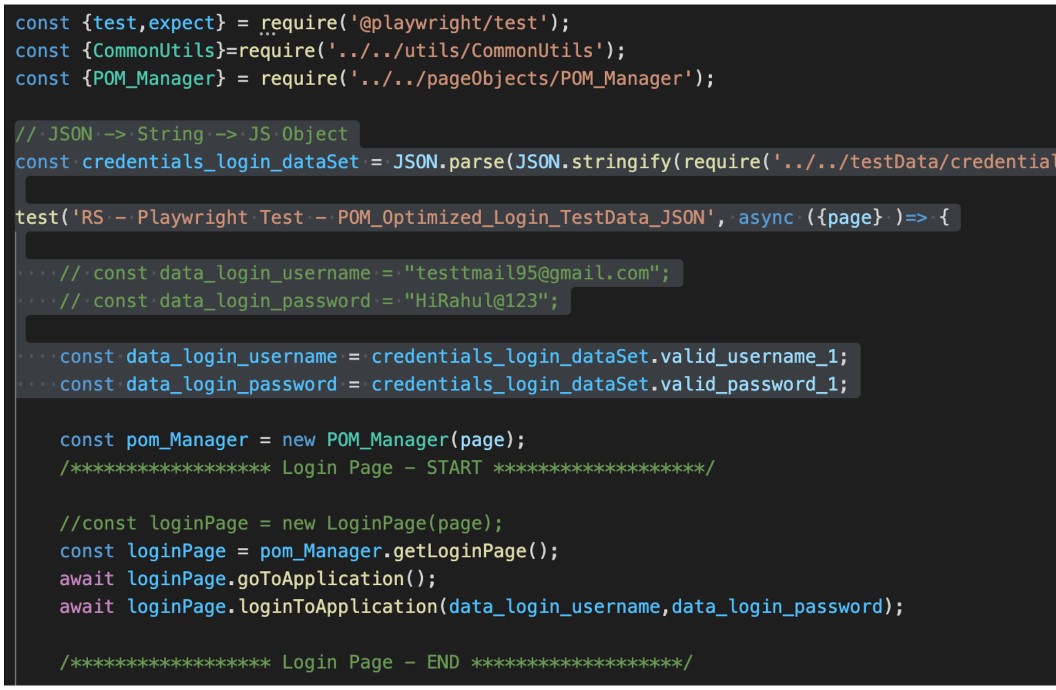
k. Data-Driven:

i. Data-Driven:

1. 

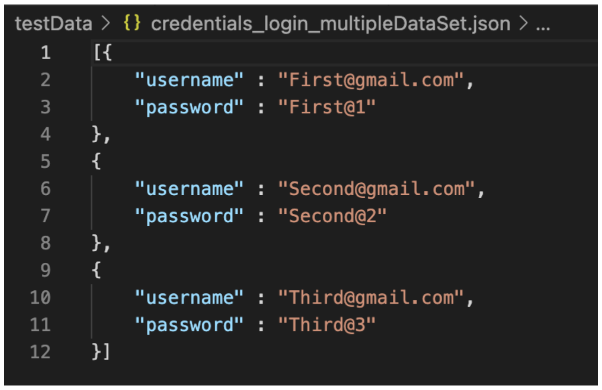
testData > {} credentials_login.json > valid_username_1

```
{
    "valid_username_1" : "testmail95@gmail.com",
    "valid_password_1" : "HiRahul@123"
}
```

2. 

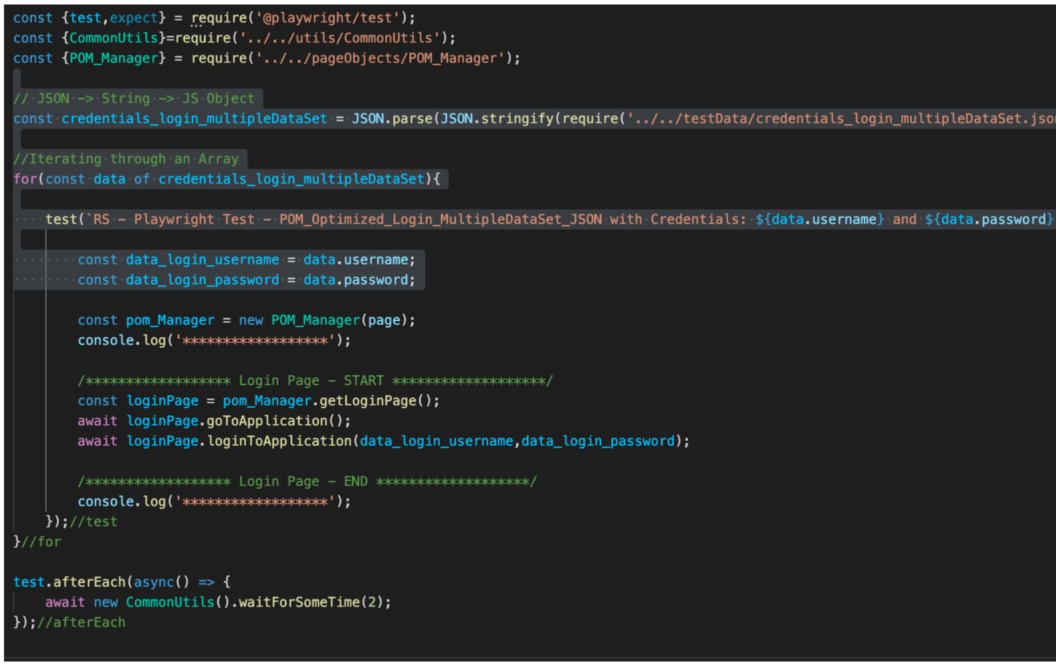
```
const {test,expect} = require('@playwright/test');
const {CommonUtils}=require('../../utils/CommonUtils');
const {POM_Manager} = require('../../pageObjects/POM_Manager');

// JSON -> String -> JS Object
const credentials_login_dataSet = JSON.parse(JSON.stringify(require('../../testData/credentia

test('RS - Playwright Test - POM_Optimized_Login_TestData_JSON', async ({page} )=> {

    // const data_login_username = "testmail95@gmail.com";
    // const data_login_password = "HiRahul@123";

    const data_login_username = credentials_login_dataSet.valid_username_1;
    const data_login_password = credentials_login_dataSet.valid_password_1;

    const pom_Manager = new POM_Manager(page);
    /****************** Login Page - START ******************/

    //const loginPage = new LoginPage(page);
    const loginPage = pom_Manager.getLoginPage();
    await loginPage.goToApplication();
    await loginPage.loginToApplication(data_login_username,data_login_password);

    /****************** Login Page - END ******************/
```

ii. Test script with Multiple Data Sets

1. 

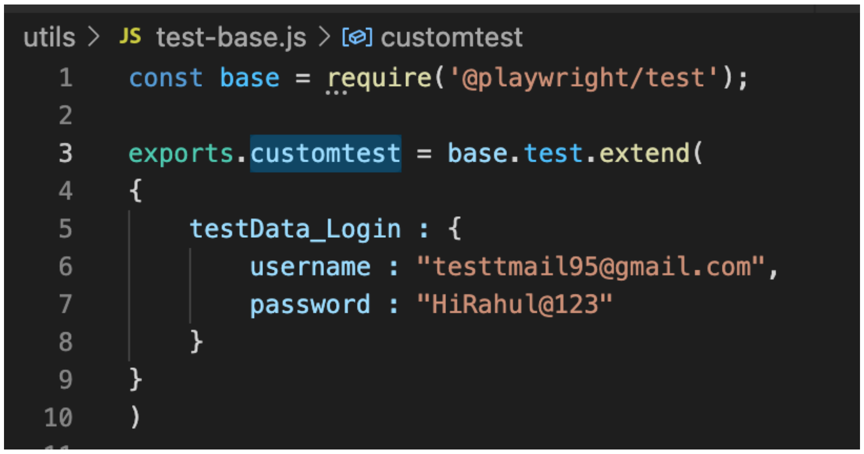
testData > {} credentials_login_multipleDataSet.json > ...

```json
[{
    "username" : "First@gmail.com",
    "password" : "First@1"
},
{
    "username" : "Second@gmail.com",
    "password" : "Second@2"
},
{
    "username" : "Third@gmail.com",
    "password" : "Third@3"
}]
```

2. 

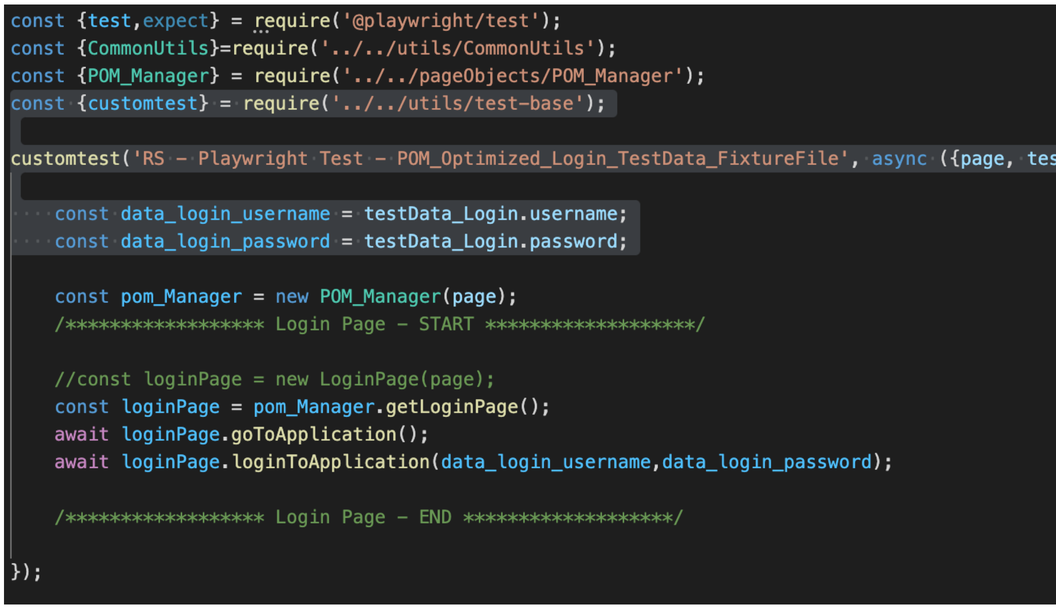
```javascript
const {test,expect} = require('@playwright/test');
const {CommonUtils}=require('../../utils/CommonUtils');
const {POM_Manager} = require('../../pageObjects/POM_Manager');

// JSON -> String -> JS Object
const credentials_login_multipleDataSet = JSON.parse(JSON.stringify(require('../../testData/credentials_login_multipleDataSet.jso

//Iterating through an Array
for(const data of credentials_login_multipleDataSet){

    test(`RS - Playwright Test - POM_Optimized_Login_MultipleDataSet_JSON with Credentials: ${data.username} and ${data.password}

        const data_login_username = data.username;
        const data_login_password = data.password;

        const pom_Manager = new POM_Manager(page);
        console.log('*****************');

        /***************** Login Page - START ******************/
        const loginPage = pom_Manager.getLoginPage();
        await loginPage.goToApplication();
        await loginPage.loginToApplication(data_login_username,data_login_password);

        /***************** Login Page - END *****************/
        console.log('*****************');
    });//test
}//for

test.afterEach(async() => {
    await new CommonUtils().waitForSomeTime(2);
});//afterEach
```

iii. Test script using Fixture:

1. 

utils > JS test-base.js > customtest

```javascript
const base = require('@playwright/test');

exports.customtest = base.test.extend(
{
    testData_Login : {
        username : "testtmail95@gmail.com",
        password : "HiRahul@123"
    }
}
)
```

2. 

```
const {test,expect} = require('@playwright/test');
const {CommonUtils}=require('../../utils/CommonUtils');
const {POM_Manager} = require('../../pageObjects/POM_Manager');
const {customtest} = require('../../utils/test-base');

customtest('RS - Playwright Test - POM_Optimized_Login_TestData_FixtureFile', async ({page, tes

    const data_login_username = testData_Login.username;
    const data_login_password = testData_Login.password;

    const pom_Manager = new POM_Manager(page);
    /***************** Login Page - START ******************/

    //const loginPage = new LoginPage(page);
    const loginPage = pom_Manager.getLoginPage();
    await loginPage.goToApplication();
    await loginPage.loginToApplication(data_login_username,data_login_password);

    /***************** Login Page - END ******************/

});
```

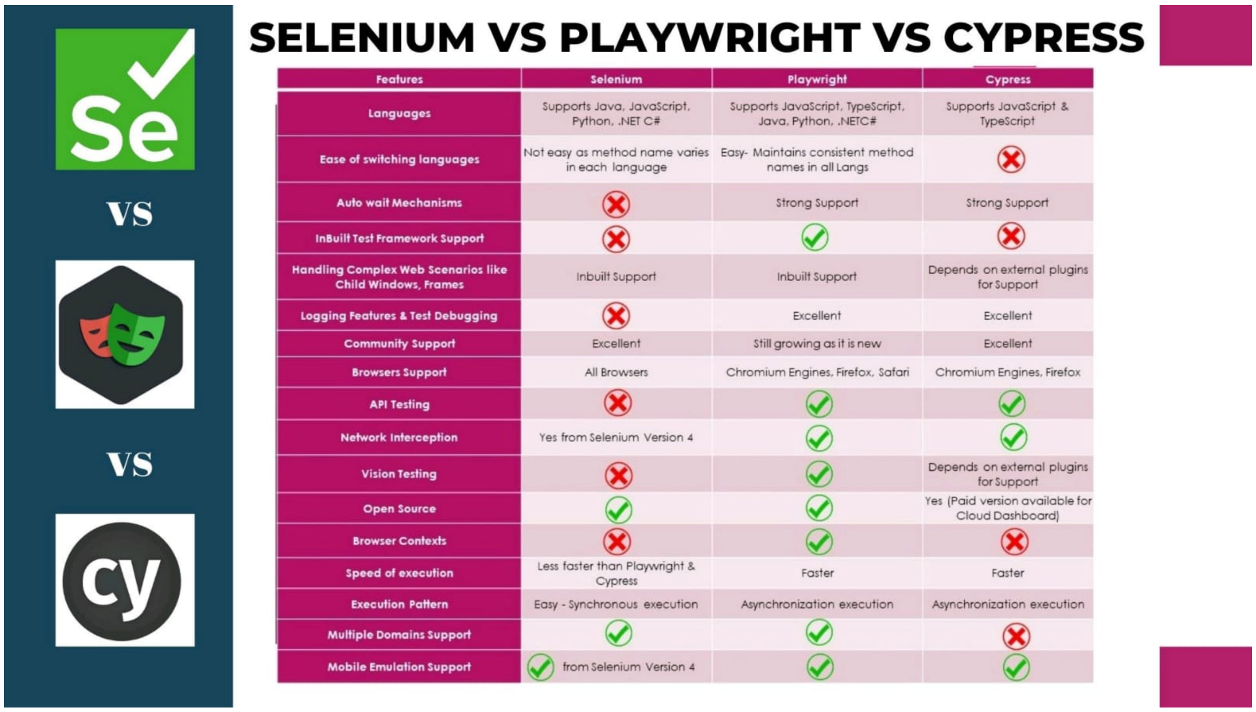
—-----------------

l. **Commands**:
   i. **npx playwright test** (Runs the E2E tests in headless mode)
      1. If we have marked any test case as
         a. test.only
            i. Then, only those test cases will run
      2. It will take all the projects
         a. Chromium, Safari, Firefox
   ii. **npx playwright test --headed** (Runs the E2E tests in headed mode)
   iii. **npx playwright test --project=chromium** (Runs the tests only on Desktop Chrome)
   iv. **npx playwright test tests/example.spec.js** (Runs the tests of a specific file)
   v. **npx playwright test --debug** (Runs the tests in debug mode)
   vi. **npx playwright test --grep @Sanity** (Runs the Sanity tests)
   vii. **npx playwright codegen** https://www.google.com/ (This will start the Recording your actions over the application)
   viii. **npx playwright show-report** (To open last HTML report run)
   ix. **npx playwright test tests/example.spec.js --config playwright.config-custom.js** (Runs with specific config file)
   x. If you have added the scripts in the package.json file, then,
         a. **npm run open_reports**
         b. **npm run test_single_assert_title_headed**
   xi. Allure:
      1. https://www.npmjs.com/package/allure-playwright
      2. Commands:
         a. npm i -D @playwright/test allure-playwright
         b. playwright test tests/04_RS_UI_Tests_Section_7/*.spec.js --headed --reporter=line,allure-playwright

i. Now, you will see allure-results folder is generated in your project.

3. **allure generate ./allure-results --clean** (To generate the final report)
    a. Now, you will see allure-report folder is generated in your project. This folder has the final report
4. **allure open ./allure-report** (To open the report)

==



SELENIUM VS PLAYWRIGHT VS CYPRESS

Features	Selenium	Playwright	Cypress
Languages	Supports Java, JavaScript, Python, .NET C#	Supports JavaScript, TypeScript, Java, Python, .NETC#	Supports JavaScript & TypeScript
Ease of switching languages	Not easy as method name varies in each language	Easy- Maintains consistent method names in all Langs	✗
Auto wait Mechanisms	✗	Strong Support	Strong Support
InBuilt Test Framework Support	✗	✓	✗
Handling Complex Web Scenarios like Child Windows, Frames	Inbuilt Support	Inbuilt Support	Depends on external plugins for Support
Logging Features & Test Debugging	✗	Excellent	Excellent
Community Support	Excellent	Still growing as it is new	Excellent
Browsers Support	All Browsers	Chromium Engines, Firefox, Safari	Chromium Engines, Firefox
API Testing	✗	✓	✓
Network Interception	Yes from Selenium Version 4	✓	✓
Vision Testing	✗	✓	Depends on external plugins for Support
Open Source	✓	✓	Yes (Paid version available for Cloud Dashboard)
Browser Contexts	✗	✓	✗
Speed of execution	Less faster than Playwright & Cypress	Faster	Faster
Execution Pattern	Easy - Synchronous execution	Asynchronization execution	Asynchronization execution
Multiple Domains Support	✓	✓	✗
Mobile Emulation Support	✓ from Selenium Version 4	✓	✓

==