**Playwrights Automation**

**Install the Playwright with command:**

npm init playwright@latest

*Note: Ensure Node.js (v14+) is installed. For yarn users, run yarn create playwright instead.*  
*Additional Note: Use npm install playwright if you only need the library without test scaffolding.*

**Before writing test cases, we need to import:**

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

*Use require for CommonJS modules (default in Node.js) and import for ES modules (add "type": "module" in package.json).*  
*Tip: Use import syntax for TypeScript projects (rename files to .ts and configure tsconfig.json).*

**test:** Defines a test case block.  
**expect:** Used for assertions (validating outcomes).

**Example test structure:**

test('Home Page', async ({ page })=>{

await page.goto(‘https://www.google.com’)

})

*Fixtures like page are injected automatically. Fixtures manage state and provide APIs (e.g., page for browser interactions).*  
***Warning:*** *Avoid using page outside test blocks (e.g., in global scope) to prevent state leaks.*

**Fixtures:**

* Predefined objects (e.g., page, context, browser) with methods for actions like navigation, clicks, and input.
* Reusable across tests, ensuring isolation. Example:

test(\'Login\', async ({ page }) => {

await page.goto(\'/login\');

});

*Advanced: Create custom fixtures (e.g., authenticated users) via test.extend().*

**JavaScript Promises:**

* Use async/await to handle asynchronous operations (e.g., network requests, element interactions).
* Playwright methods return promises; await ensures sequential execution.  
  Tip: Use Promise.all() for parallel actions (e.g., clicking multiple elements simultaneously).

**Run commands:**

* npx playwright test: Runs all tests in headless mode.
* npx playwright test Homepage.spec.js: Runs a specific file.
* npx playwright test --project=chromium: Limits execution to Chromium.
* npx playwright test --headed: Runs tests visibly.
* npx playwright test --debug: Launches the Playwright Inspector for step-by-step debugging.
* npx playwright test --workers=1: Runs tests sequentially (disable parallel execution).

**Locators:**

* Best Practices: Prefer semantic locators (getByRole, getByText) over XPath/CSS for stability.
* Types:
* Property: page.getByTestId(\'id\')
* CSS: page.locator(\'.class\')
* XPath: page.locator(\'//button\') (use sparingly).  
  Pro Tip: Use testId attributes (e.g., data-testid="submit-button") for resilient element targeting.

**Built-in Locators:**

* [page.getByRole()](https://playwright.dev/docs/locators#locate-by-role) to locate by explicit and implicit accessibility attributes.
* [page.getByText()](https://playwright.dev/docs/locators#locate-by-text) to locate by text content.
* [page.getByLabel()](https://playwright.dev/docs/locators#locate-by-label) to locate a form control by associated label's text.
* [page.getByPlaceholder()](https://playwright.dev/docs/locators#locate-by-placeholder) to locate an input by placeholder.
* [page.getByAltText()](https://playwright.dev/docs/locators#locate-by-alt-text) to locate an element, usually image, by its text alternative.
* [page.getByTitle()](https://playwright.dev/docs/locators#locate-by-title) to locate an element by its title attribute.
* [page.getByTestId()](https://playwright.dev/docs/locators#locate-by-test-id) to locate an element based on its data-testid attribute (other attributes can be configured).

**Code Gen:**

* npx playwright codegen
* npx playwright codegen -o tests/mytestfile.spec.js
* npx playwright codegen --device “iphone 13”

**Hard Assertions:**

Hard assertion failed assertion will terminate test execution.

*// Hard Assertions - if the assertion fails, the test will stop and the test will fail*

  await expect(*page*).toHaveTitle('STORE');

  await expect(*page*).toHaveURL('https://demoblaze.com/index.html');

  await expect(*page*.locator('.navbar-brand')).toBeVisible();

**Soft Assertions:**

soft assertions: failed soft assertions do not terminate test execution, but mark the test as failed.

*// Soft Assertions - if the assertion fails, the test will continue and the test will fail*

  await expect.soft(*page*).toHaveTitle('STORE123');

  await expect.soft(*page*).toHaveURL('https://demoblaze.com/index.html');

  await expect.soft(*page*.locator('.navbar-brand')).toBeVisible();

**Element Interactions:**

**Locate multiple web elements:**

* const element = await page.$$(locator)

**For Click:**

* await page.locator(‘locator’).click()
* await page.click (‘locator’)

**Input Box:**

* await page.locator(‘locator’).fill(‘value’)
* await page.locator(‘locator’).type(‘value’)
* await page.fill(‘locator’, ‘value’)
* await page.type(‘locator’, ‘value’)

**Input Box:**

*// input first name*

  await expect(*page*.locator('#name')).toBeVisible();

  await expect(*page*.locator('#name')).toBeEmpty();

  await expect(*page*.locator('#name')).toBeEditable();

  await expect(*page*.locator('#name')).toBeEnabled();

  await *page*.locator('#name').fill('John');

  await expect(*page*.locator('#name')).toHaveValue('John');

**Radio Button:**

*// radio button*

  await *page*.locator('#male').check();

  await expect(*page*.locator('#male')).toBeChecked();

  await expect(*page*.locator('#male').isChecked()).toBeTruthy();

*// check female is unchecked*

  await expect(*page*.locator('#female')).not.toBeChecked();

  await expect(*page*.locator('#male').isVisible()).toBeTruthy();

  await expect(*page*.locator('#male').isEnabled()).toBeTruthy();

  await expect(*page*.locator('#male').isEditable()).toBeTruthy();

*// check male is not disabled and not hidden*

  const isDisabled = await *page*.locator('#male').isDisabled();

  expect(isDisabled).toBeFalsy();

  const isHidden = await *page*.locator('#male').isHidden();

  expect(isHidden).toBeFalsy();

**CheckBox:**

* await page.locator(‘locator’).check()
* await page.locator(‘locator’).toBeChecked()
* await page.locator(‘locator’).isChecked() return True False{.toBeTruthy(), .toBeFalsy()}
* await page.locator(‘locator’).unCheck()

**Dropdown Menu:**

* **By label:**
* await page.locator('#country').selectOption({label: 'India'})
* **By Text:**
* await page.locator('#country').selectOption('India')
* await page.selectOption("#country", 'India')
* **By Value:**
* await page.locator('#country').selectOption({value: 'uk'})
* **By Index:**
* await page.locator('#country').selectOption({index: 1})

**Multi select Dropdown:**

* await page.selectOption('#colors', ['Blue', 'red', 'Yellow'])

**Auto Suggest Dropdown**

*//1) Select the city*

  await *page*.locator('#src').fill('Delhi');

  await *page*.waitForSelector("//li[contains(@class,'sc-iwsKbI')]/div/text[1]");

  await *page*.$$("//li[contains(@class,'sc-iwsKbI')]/div/text[1]");

  const fromCityOptions = await *page*.$$("//li[contains(@class,'sc-iwsKbI')]/div/text[1]");

  for (const option of fromCityOptions) {

    const value = await option.textContent();

    console.log(value);

    if (value.includes('Anand Vihar')) {

      await option.click();

      break;

    }

**Alert:**

By default, dialogs are auto-dismissed by Playwright, so you don't have to handle them. However, you can register a dialog handler before the action that triggers the dialog to either **dialog**.**accept()** or **dialog.dismiss()** it.

* Alert with OK

*page*.on('dialog', async *dialog* => {

    expect(*dialog*.type()).toContain('alert');

    expect(*dialog*.message()).toContain('I am an alert box!');

    await *dialog*.accept();

  });

  await *page*.click("#alertBtn")

* Confirmation Dialog-Alert with Ok and Cancel

*page*.on('dialog', async *dialog* => {

      expect(*dialog*.type()).toContain('confirm');

      expect(*dialog*.message()).toContain('Press a button!');

      await *dialog*.accept(); *// close by using ok button*

*//await dialog.dismiss(); // close by using cancel button*

    });

    await *page*.click("//button[@id='confirmBtn']");

    await expect(*page*.locator("//p[@id='demo']")).toHaveText('You pressed OK!');

* Prompt Dialog-Alert with Ok and Cancel

*page*.on('dialog', async *dialog* => {

      expect(*dialog*.type()).toContain('prompt');

      expect(*dialog*.message()).toContain('Please enter your name:');

      expect(*dialog*.defaultValue()).toContain('Harry Potter');

      await *dialog*.accept('John'); *// close by using ok button*

*//await dialog.dismiss(); // close by using cancel button*

    });

    await *page*.click("//button[@id='promptBtn']");

    await expect(*page*.locator("//p[@id='demo']")).toHaveText('Hello John! How are you today?');

**Frames:**

* await page.**frame**({url: 'https://ui.vision/demo/webtest/frames/frame\_1.html'}).fill("[name='mytext1']", 'Aakash')
* await page. **frameLocator**("frame[src='frame\_1.html']").locator("[name='mytext1']").fill('Aakash')

**Inner/Nested Frames**

await *page*.goto('https://ui.vision/demo/webtest/frames/');

    const frame4 = await *page*.frame({url: 'https://ui.vision/demo/webtest/frames/frame\_3.html'});

    await frame4.fill("[name=mytext3]",'Hello');

*// nested frames*

    const childFrame = await frame4.childFrames();

    await childFrame[0].locator("//\*[@id='i6']/div[3]/div").check(); *// childFrame[0] is the first child frame, childFrame[1] is the second child frame*

**Table:**

* const table = await page.locator('#productTable')
* const colums = await table.locator('thead tr th')
* const rows = await table.locator('tbody tr')

**Mouse Actions:**

* await page.locator("//a[normalize-space()='Desktops']").**hover**()
* await page.locator("//span[@class='context-menu-one btn btn-neutral']").**click**({button: 'right'})
* await page.locator("//button[normalize-space()='Copy Text']").**dblclick**()
* await page.locator("//button[normalize-space()='Copy Text']").**press**('Control+Shift+KeyR')
* Approach 1: Using **dragTo**() method
  + await page.locator('#item-to-be-dragged').dragTo(page.locator('#item-to-drop-at'))
* Approach 2: **Dragging manually**
* await *page*.locator('#draggable').hover();
* await *page*.mouse.down();
* await *page*.locator('#droppable').hover();
* await *page*.mouse.up();
* *// verify the text*
* await expect(*page*.locator('#droppable')).toHaveText('Dropped!');

**Upload Files:**

* Select one file

await page.getByLabel('Upload file').setInputFiles(path.join(\_\_dirname, 'myfile.pdf'));

await *page*.locator("#singleFileInput").setInputFiles("tests/uploadFiles/testfile1.pdf");

* Select multiple files

await page.getByLabel('Upload files').setInputFiles([ path.join(\_\_dirname, 'file1.txt'), path.join(\_\_dirname, 'file2.txt'),]);

await *page*.locator("#multipleFilesInput").setInputFiles(['tests/uploadFiles/testfile1.pdf', 'tests/uploadFiles/testfile2.pdf']);

**Playwright Hooks:**

* **beforeEach:** This hook is executed before each individual test.
* **afterEach:** This hook is executed after each individual test.
* **beforeAll:** This hook is executed once before any of the tests start running.
* **afterAll:** This hook is executed once after all the tests have been run.

**Grouping:**

After Hookes

test.describe('Group 1', () => {

    test('Test 1', async ({ *page* }) => {

        console.log('Test 1');

    });

    test('Test 2', async () => {

        console.log('Test 2');

    });

});

**Tracing:**

*// set the trace to on in playwright.config.js*

*// Run the command: npx playwright test Tracing.spec.js --project=chromium*

*// trace.zip file will be created in the test-results folder*

*// Right click on the trace.zip file and copy the relative path*

*// Run the command: npx playwright show-trace test-results\Tracing-Video-Recording-chromium\trace.zip*

*// This will open the trace in the browser*

**Tags:**

test('Test1@sanity', async ({ *page* }) => {

    console.log('Test 1');

});

test('Test3@regression', async ({ *page* }) => {

    console.log('Test 3');

});

test('Test5@sanity@regression', async ({ *page* }) => {

    console.log('Test 5');

});

*// Run the command: npx playwright test Tags.spec.js --project=chromium*

*// ----It will run all the tests----*

*// Run the command: npx playwright test Tags.spec.js --project=chromium --grep "@sanity"*

*// ----It will run all the sanity tests----*

*// Run the command: npx playwright test Tags.spec.js --project=chromium --grep "@regression"*

*// ----It will run all the regression tests----*

*// Run the command: npx playwright test Tags.spec.js --project=chromium --grep "@sanity@regression"*

*// ----It will run only the test in which both @sanity@regression tag is present----*

*// Run the command: npx playwright test Tags.spec.js --project=chromium --grep-invert "@sanity"*

*// ----It will run all the tests except the sanity tests----*

**Annotations:**

*//This test will be skipped for chromium browser*

test('Test3', async ({ *page*, *browserName* }) => {

    console.log('Test 3');

    if (*browserName* === 'chromium') {

        test.skip();

    }

});

*// Fixme: This test will be skipped for chromium browser*

test('Test4', async ({ *page*, *browserName* }) => {

    test.fixme();

    console.log('Test 4');

});

*// Fail: This test will fail*

test('Test5', async ({ *page* }) => {

    test.fail();

    console.log('Test 5');

    expect(1).toBe(1);

});

**Handling Windows:**

import { test, expect, chromium } from '@playwright/test';

test('Handle Page/Windows', async ({})=> {

    const browser = await chromium.launch()

    const context = await browser.newContext()

    const page1 = await context.newPage()

    const page2 = await context.newPage()

    const allPages = context.pages()

    console.log("No of Pages created:", allPages.length)

    await page1.goto("https://opensource-demo.orangehrmlive.com/web/index.php/auth/login")

    await expect(page1).toHaveTitle("OrangeHRM")

    await page2.goto("https://orangehrm.com")

    await expect(page2).toHaveTitle("Human Resources Management Software | OrangeHRM")

});

test('Handle Multiple Page/Windows', async ({})=> {

    const browser = await chromium.launch()

    const context = await browser.newContext()

    const page1 = await context.newPage()

    await page1.goto("https://opensource-demo.orangehrmlive.com/web/index.php/auth/login")

    await expect(page1).toHaveTitle("OrangeHRM")

    const pagePromise = context.waitForEvent('page')

    await page1.locator('//a[normalize-space()="OrangeHRM, Inc"]').click()

    const newPage = await pagePromise;

    await expect(newPage).toHaveTitle("Human Resources Management Software | OrangeHRM")

    await browser.close()

});

**Reporters:**

*// For list: npx playwright test Reporters.spec.js --project=chromium --reporter=list*

*// For line: npx playwright test Reporters.spec.js --project=chromium --reporter=line*

*// For Dot: npx playwright test Reporters.spec.js --project=chromium --reporter=dot*

*// For html: npx playwright test Reporters.spec.js --project=chromium --reporter=html*

*// For Json: npx playwright test Reporters.spec.js --project=chromium --reporter=json but first set the environment variable*

*// For JUnit: npx playwright test Reporters.spec.js --project=chromium --reporter=junit but first set the environment variable*

**Allure Reports:**

1) Installation of "allure-playwright" module

npm i -D @playwright/test allure-playwright

2) Installing Allure command line

Ref: https://www.npmjs.com/package/allure-commandline

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

(or)

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

3) playwright.config.js

reporter: [['allure-playwright', {outputFolder: 'Alure-Reports'}]],

(or)

npx playwright test --reporter=allure-playwright

4) Run the tests

npx playwright test tests/Reporters.spec.js

5) Open Allure Report:

allure serve allure-results

**Retries and Test Flakiness:**

1) Passed - No retry

2) Failed - Retry - Failed

3) Failed - Retry - Passed // Fleky tests

# Give failing tests 3 retry attempts

npx playwright test RetryTest.spec.js --project=chromium --headed --retries=3

**API Test:**

import { test, expect } from '@playwright/test';

var userId;

test('Get Users', async ({ *request* }) => {

    const response = await *request*.get('https://reqres.in/api/users?page=2')

    console.log(await response.json())

    expect(response.status()).toBe(200)

});

test('Create User', async ({ *request* }) => {

    const response = await *request*.post('https://reqres.in/api/users', {

        data: {

            "name": "shehroz",

            "job": "trainer"

        },

        headers:{

            "Accept":"application/json"

        }

    });

    const jsonResponse = await response.json(); *// Await the JSON response*

    console.log(jsonResponse); *// Log full response to check structure*

    expect(response.status()).toBe(201);

    userId = jsonResponse.id; *// Extract id properly*

    console.log("User Creation ID:", userId);

});

test('Update User', async ({ *request* }) => {

    const response = await *request*.put('https://reqres.in/api/users/' + userId, {

        data: {

            "name": "shehroz",

            "job": "engineer"

        },

        headers:{

            "Accept":"application/json"

        }

    });

    const jsonResponse = await response.json(); *// Await the JSON response*

    console.log(jsonResponse); *// Log full response to check structure*

    expect(response.status()).toBe(200);

});

test('Delete User', async ({ *request* }) => {

    const response = await *request*.delete('https://reqres.in/api/users/' + userId);

    console.log("Delete User Response Status:", response.status());

    expect(response.status()).toBe(204);

});