|| Playwright using JavaScript ||

Playwright

Official Playwright Website : https://playwright.dev/docs/intro

[Read official doc for more detail information.]

why Playwright can be future of UI Automation :

Why Playwright?

- Playwright enables reliable end-to-end testing for modern web apps with its Auto-wait capability
- Works on major Browser which uses Chromium Engine, (Chrome & Edge) Firefox, Safari (Web kit) & Opera
- Works on any OS – Windows, MAC, Linux & Supports Native mobile emulation of Google Chrome for Android and Mobile IOS in Safari
- Works with any language – JavaScript, TypeScript, Java, Python, C#(.NET). – (*This course is made on JavaScript)*
- Playwright have excellent inbuilt features called Traces which can take automatic Screenshots, test video recording, Flaky test retry & Logging mechanism
- Playwright provides inspector tool which help us to monitor and debug every step of execution, see click points and verify page locators on fly
- Playwright has inbuilt API Testing libraries to fire the Network calls on fly within Web Application. (Test Edge Case scenarios with the mix of web & API testing)
- Playwright provides browser context feature which help to save and transfer the browser state to any other new browser.
- Playwright provides codegen tool which Generate test code by recording your actions. Save them into any language.

With all the above top features it has, Playwright is now tough competitor for Selenium & Cypress automation tools.

Introduction :

1. free & open-source framework for web automation testing, created by Microsoft.

2. Languages - JavaScript, TypeScript, Java, Python, .NET(C#).

3. Browsers - Chromium, Webkit, Firefox - headed or headless Mode

4. Auto-waiting : see chart it is imp : https://playwright.dev/docs/actionability

4. VS-Code Note : https://playwright.dev/docs/getting-started-vscode

Installations : https://playwright.dev/docs/intro#installing-playwright

1. Install node.js & vs-code in your system.

2. Run : npm init playwright@latest

3. Run the above install command and select the following to get started:

- Choose between TypeScript or JavaScript (default is TypeScript)

- Name of your Tests folder (default is tests or e2e if you already have a tests folder in your project)

- Add a GitHub Actions workflow to easily run tests on CI

- Install Playwright browsers (default is true)

- Note : If you Go inside node_modules/playwright/lib, here we can see all inbuilt methods & all.

Playwright Example Code :

example.spec.js U

e2e > example.spec.js > ...

Click here to ask Blackbox to help you code faster

```
// @ts-check
const { test, expect } = require('@playwright/test');

test('has title', async ({ page }) => {
  await page.goto('https://playwright.dev/');

  // Expect a title "to contain" a substring.
  await expect(page).toHaveTitle(/Playwright/);
});

test('get started link', async ({ page }) => {
  await page.goto('https://playwright.dev/');

  // Click the get started link.
  await page.getByRole('link', { name: 'Get started' }).click();

  // Expects page to have a heading with the name of Installation.
  await expect(page.getByRole('heading', { name: 'Installation' })).toBeVisible();
});
```

---

demo-todo-app.spec.js U

e2e > demo-todo-app.spec.js > test.describe('New Todo') callback

Click here to ask Blackbox to help you code faster

```
// @ts-check
const { test, expect } = require('@playwright/test');

test.beforeEach(async ({ page }) => {
  await page.goto('https://demo.playwright.dev/todomvc');
});

const TODO_ITEMS = [
  'buy some cheese',
  'feed the cat',
  'book a doctors appointment'
];

test.describe('New Todo', () => {
  test('should allow me to add todo items', async ({ page }) => {
    // create a new todo locator
    const newTodo = page.getByPlaceholder('What needs to be done?');

    // Create 1st todo.
    await newTodo.fill(TODO_ITEMS[0]);
    await newTodo.press('Enter');

    // Make sure the list only has one todo item.
    await expect(page.getByTestId('todo-title')).toHaveText([
      TODO_ITEMS[0]
    ]);

    // Create 2nd todo.
    await newTodo.fill(TODO_ITEMS[1]);
    await newTodo.press('Enter');

    // Make sure the list now has two todo items.
    await expect(page.getByTestId('todo-title')).toHaveText([
      TODO_ITEMS[0],
      TODO_ITEMS[1]
    ]);

    await checkNumberOfTodosInLocalStorage(page, 2);
  });
```

demo-todo-app.spec.js U

e2e > demo-todo-app.spec.js > test.describe('New Todo') callback

```
  test('should clear text input field when an item is added', async ({ page }) => {
    // create a new todo locator
    const newTodo = page.getByPlaceholder('What needs to be done?');

    // Create one todo item.
    await newTodo.fill(TODO_ITEMS[0]);
    await newTodo.press('Enter');

    // Check that input is empty.
    await expect(newTodo).toBeEmpty();
    await checkNumberOfTodosInLocalStorage(page, 1);
  });
```

demo-todo-app.spec.js U

e2e > demo-todo-app.spec.js > test.describe('New Todo') callback

```
54    test('should append new items to the bottom of the list', async ({ page }) => {
55      // Create 3 items.
56      await createDefaultTodos(page);
57
58      // create a todo count locator
59      const todoCount = page.getByTestId('todo-count')
60
61      // Check test using different methods.
62      await expect(page.getByText('3 items left')).toBeVisible();
63      await expect(todoCount).toHaveText('3 items left');
64      await expect(todoCount).toContainText('3');
65      await expect(todoCount).toHaveText(/3/);
66
67      // Check all items in one call.
68      await expect(page.getByTestId('todo-title')).toHaveText(TODO_ITEMS);
69      await checkNumberOfTodosInLocalStorage(page, 3);
70    });
71  });
72
73  test.describe('Mark all as completed', () => {
74    test.beforeEach(async ({ page }) => {
75      await createDefaultTodos(page);
76      await checkNumberOfTodosInLocalStorage(page, 3);
77    });
78
79    test.afterEach(async ({ page }) => {
80      await checkNumberOfTodosInLocalStorage(page, 3);
81    });
82
83    test('should allow me to mark all items as completed', async ({ page }) => {
84      // Complete all todos.
85      await page.getByLabel('Mark all as complete').check();
86
87      // Ensure all todos have 'completed' class.
88      await expect(page.getByTestId('todo-item')).toHaveClass(['completed', 'completed', 'completed']);
89      await checkNumberOfCompletedTodosInLocalStorage(page, 3);
90    });
91
```

demo-todo-app.spec.js U

e2e > demo-todo-app.spec.js > test.describe('Mark all as completed') callback > test('should allow me to mark all items as completed') callback

```
92    test('should allow me to clear the complete state of all items', async ({ page }) => {
93      const toggleAll = page.getByLabel('Mark all as complete');
94      // Check and then immediately uncheck.
95      await toggleAll.check();
96      await toggleAll.uncheck();
97
98      // Should be no completed classes.
99      await expect(page.getByTestId('todo-item')).toHaveClass(['', '', '']);
100   });
101
102   test('complete all checkbox should update state when items are completed / cleared', async ({ page }) => {
103     const toggleAll = page.getByLabel('Mark all as complete');
104     await toggleAll.check();
105     await expect(toggleAll).toBeChecked();
106     await checkNumberOfCompletedTodosInLocalStorage(page, 3);
107
108     // Uncheck first todo.
109     const firstTodo = page.getByTestId('todo-item').nth(0);
110     await firstTodo.getByRole('checkbox').uncheck();
111
112     // Reuse toggleAll locator and make sure its not checked.
113     await expect(toggleAll).not.toBeChecked();
114
115     await firstTodo.getByRole('checkbox').check();
116     await checkNumberOfCompletedTodosInLocalStorage(page, 3);
117
118     // Assert the toggle all is checked again.
119     await expect(toggleAll).toBeChecked();
120   });
121 });
122
123 test.describe('Item', () => {
124
125   test('should allow me to mark items as complete', async ({ page }) => {
126     // create a new todo locator
127     const newTodo = page.getByPlaceholder('What needs to be done?');
128
129     // Create two items.
130     for (const item of TODO_ITEMS.slice(0, 2)) {
131       await newTodo.fill(item);
132       await newTodo.press('Enter');
133     }
```

demo-todo-app.spec.js U

e2e > demo-todo-app.spec.js > test.describe('Mark all as completed') callback > test('should allow me to mark all items as completed') callback

```
135     // Check first item.
136     const firstTodo = page.getByTestId('todo-item').nth(0);
137     await firstTodo.getByRole('checkbox').check();
138     await expect(firstTodo).toHaveClass('completed');
139
140     // Check second item.
141     const secondTodo = page.getByTestId('todo-item').nth(1);
142     await expect(secondTodo).not.toHaveClass('completed');
143     await secondTodo.getByRole('checkbox').check();
144
145     // Assert completed class.
146     await expect(firstTodo).toHaveClass('completed');
147     await expect(secondTodo).toHaveClass('completed');
148   });
149
```

demo-todo-app.spec.js U

e2e > demo-todo-app.spec.js > test.describe('Mark all as completed') callback > test('should allow me to mark all items as completed') callback

```
150   test('should allow me to un-mark items as complete', async ({ page }) => {
151     // create a new todo locator
152     const newTodo = page.getByPlaceholder('What needs to be done?');
153
154     // Create two items.
155     for (const item of TODO_ITEMS.slice(0, 2)) {
156       await newTodo.fill(item);
157       await newTodo.press('Enter');
158     }
159
160     const firstTodo = page.getByTestId('todo-item').nth(0);
161     const secondTodo = page.getByTestId('todo-item').nth(1);
162     const firstTodoCheckbox = firstTodo.getByRole('checkbox');
163
164     await firstTodoCheckbox.check();
165     await expect(firstTodo).toHaveClass('completed');
166     await expect(secondTodo).not.toHaveClass('completed');
167     await checkNumberOfCompletedTodosInLocalStorage(page, 1);
168
169     await firstTodoCheckbox.uncheck();
170     await expect(firstTodo).not.toHaveClass('completed');
171     await expect(secondTodo).not.toHaveClass('completed');
172     await checkNumberOfCompletedTodosInLocalStorage(page, 0);
173   });
174
175   test('should allow me to edit an item', async ({ page }) => {
176     await createDefaultTodos(page);
177
178     const todoItems = page.getByTestId('todo-item');
179     const secondTodo = todoItems.nth(1);
180     await secondTodo.dblclick();
181     await expect(secondTodo.getByRole('textbox', { name: 'Edit' })).toHaveValue(TODO_ITEMS[1]);
182     await secondTodo.getByRole('textbox', { name: 'Edit' }).fill('buy some sausages');
183     await secondTodo.getByRole('textbox', { name: 'Edit' }).press('Enter');
184
185     // Explicitly assert the new text value.
186     await expect(todoItems).toHaveText([
187       TODO_ITEMS[0],
188       'buy some sausages',
189       TODO_ITEMS[2]
190     ]);
191     await checkTodosInLocalStorage(page, 'buy some sausages');
192   });
193 });
```

demo-todo-app.spec.js U

e2e > demo-todo-app.spec.js > test.describe('Mark all as completed') callback > test('should allow me to mark all items as completed') callback

```
194
195 test.describe('Editing', () => {
196   test.beforeEach(async ({ page }) => {
197     await createDefaultTodos(page);
198     await checkNumberOfTodosInLocalStorage(page, 3);
199   });
200
201   test('should hide other controls when editing', async ({ page }) => {
202     const todoItem = page.getByTestId('todo-item').nth(1);
203     await todoItem.dblclick();
204     await expect(todoItem.getByRole('checkbox')).not.toBeVisible();
205     await expect(todoItem.locator('label', {
206       hasText: TODO_ITEMS[1],
207     })).not.toBeVisible();
208     await checkNumberOfTodosInLocalStorage(page, 3);
209   });
210
211   test('should save edits on blur', async ({ page }) => {
212     const todoItems = page.getByTestId('todo-item');
213     await todoItems.nth(1).dblclick();
214     await todoItems.nth(1).getByRole('textbox', { name: 'Edit' }).fill('buy some sausages');
215     await todoItems.nth(1).getByRole('textbox', { name: 'Edit' }).dispatchEvent('blur');
216
217     await expect(todoItems).toHaveText([
218       TODO_ITEMS[0],
219       'buy some sausages',
220       TODO_ITEMS[2],
221     ]);
222     await checkTodosInLocalStorage(page, 'buy some sausages');
223   });
224
225   test('should trim entered text', async ({ page }) => {
226     const todoItems = page.getByTestId('todo-item');
227     await todoItems.nth(1).dblclick();
228     await todoItems.nth(1).getByRole('textbox', { name: 'Edit' }).fill('    buy some sausages    ');
229     await todoItems.nth(1).getByRole('textbox', { name: 'Edit' }).press('Enter');
230
231     await expect(todoItems).toHaveText([
232       TODO_ITEMS[0],
233       'buy some sausages',
234       TODO_ITEMS[2],
235     ]);
236     await checkTodosInLocalStorage(page, 'buy some sausages');
237   });
```

demo-todo-app.spec.js U

e2e > demo-todo-app.spec.js > test.describe('Mark all as completed') callback > test('should allow me to mark all items as completed') callback

```
239   test('should remove the item if an empty text string was entered', async ({ page }) => {
240     const todoItems = page.getByTestId('todo-item');
241     await todoItems.nth(1).dblclick();
242     await todoItems.nth(1).getByRole('textbox', { name: 'Edit' }).fill('');
243     await todoItems.nth(1).getByRole('textbox', { name: 'Edit' }).press('Enter');
244
245     await expect(todoItems).toHaveText([
246       TODO_ITEMS[0],
247       TODO_ITEMS[2],
248     ]);
249   });
```

demo-todo-app.spec.js U

e2e > demo-todo-app.spec.js > test.describe('Mark all as completed') callback > test('should allow me to mark all items as completed') callback

```
  test('should cancel edits on escape', async ({ page }) => {
    const todoItems = page.getByTestId('todo-item');
    await todoItems.nth(1).dblclick();
    await todoItems.nth(1).getByRole('textbox', { name: 'Edit' }).fill('buy some sausages');
    await todoItems.nth(1).getByRole('textbox', { name: 'Edit' }).press('Escape');
    await expect(todoItems).toHaveText(TODO_ITEMS);
  });
});

test.describe('Counter', () => {
  test('should display the current number of todo items', async ({ page }) => {
    // create a new todo locator
    const newTodo = page.getByPlaceholder('What needs to be done?');

    // create a todo count locator
    const todoCount = page.getByTestId('todo-count')

    await newTodo.fill(TODO_ITEMS[0]);
    await newTodo.press('Enter');
    await expect(todoCount).toContainText('1');

    await newTodo.fill(TODO_ITEMS[1]);
    await newTodo.press('Enter');
    await expect(todoCount).toContainText('2');

    await checkNumberOfTodosInLocalStorage(page, 2);
  });
});

test.describe('Clear completed button', () => {
  test.beforeEach(async ({ page }) => {
    await createDefaultTodos(page);
  });

  test('should display the correct text', async ({ page }) => {
    await page.locator('.todo-list li .toggle').first().check();
    await expect(page.getByRole('button', { name: 'Clear completed' })).toBeVisible();
  });
```

demo-todo-app.spec.js U

e2e > demo-todo-app.spec.js > test.describe('Mark all as completed') callback > test('should allow me to mark all items as completed') callback

```
  test('should remove completed items when clicked', async ({ page }) => {
    const todoItems = page.getByTestId('todo-item');
    await todoItems.nth(1).getByRole('checkbox').check();
    await page.getByRole('button', { name: 'Clear completed' }).click();
    await expect(todoItems).toHaveCount(2);
    await expect(todoItems).toHaveText([TODO_ITEMS[0], TODO_ITEMS[2]]);
  });

  test('should be hidden when there are no items that are completed', async ({ page }) => {
    await page.locator('.todo-list li .toggle').first().check();
    await page.getByRole('button', { name: 'Clear completed' }).click();
    await expect(page.getByRole('button', { name: 'Clear completed' })).toBeHidden();
  });
});

test.describe('Persistence', () => {
  test('should persist its data', async ({ page }) => {
    // create a new todo locator
    const newTodo = page.getByPlaceholder('What needs to be done?');

    for (const item of TODO_ITEMS.slice(0, 2)) {
      await newTodo.fill(item);
      await newTodo.press('Enter');
    }

    const todoItems = page.getByTestId('todo-item');
    const firstTodoCheck = todoItems.nth(0).getByRole('checkbox');
    await firstTodoCheck.check();
    await expect(todoItems).toHaveText([TODO_ITEMS[0], TODO_ITEMS[1]]);
    await expect(firstTodoCheck).toBeChecked();
    await expect(todoItems).toHaveClass(['completed', '']);

    // Ensure there is 1 completed item.
    await checkNumberOfCompletedTodosInLocalStorage(page, 1);

    // Now reload.
    await page.reload();
    await expect(todoItems).toHaveText([TODO_ITEMS[0], TODO_ITEMS[1]]);
    await expect(firstTodoCheck).toBeChecked();
    await expect(todoItems).toHaveClass(['completed', '']);
  });
});
```

demo-todo-app.spec.js U

e2e > demo-todo-app.spec.js > test.describe('Mark all as completed') callback > test('should allow me to mark all items as completed') callback

```
test.describe('Routing', () => {
  test.beforeEach(async ({ page }) => {
    await createDefaultTodos(page);
    // make sure the app had a chance to save updated todos in storage
    // before navigating to a new view, otherwise the items can get lost :(
    // in some frameworks like Durandal
    await checkTodosInLocalStorage(page, TODO_ITEMS[0]);
  });

  test('should allow me to display active items', async ({ page }) => {
    const todoItem = page.getByTestId('todo-item');
    await page.getByTestId('todo-item').nth(1).getByRole('checkbox').check();

    await checkNumberOfCompletedTodosInLocalStorage(page, 1);
    await page.getByRole('link', { name: 'Active' }).click();
    await expect(todoItem).toHaveCount(2);
    await expect(todoItem).toHaveText([TODO_ITEMS[0], TODO_ITEMS[2]]);
  });
```

demo-todo-app.spec.js U

e2e > demo-todo-app.spec.js > test.describe('Mark all as completed') callback > test('should allow me to mark all items as completed') callback

```
352   test('should respect the back button', async ({ page }) => {
353     const todoItem = page.getByTestId('todo-item');
354     await page.getByTestId('todo-item').nth(1).getByRole('checkbox').check();
355
356     await checkNumberOfCompletedTodosInLocalStorage(page, 1);
357
358     await test.step('Showing all items', async () => {
359       await page.getByRole('link', { name: 'All' }).click();
360       await expect(todoItem).toHaveCount(3);
361     });
362
363     await test.step('Showing active items', async () => {
364       await page.getByRole('link', { name: 'Active' }).click();
365     });
366
367     await test.step('Showing completed items', async () => {
368       await page.getByRole('link', { name: 'Completed' }).click();
369     });
370
371     await expect(todoItem).toHaveCount(1);
372     await page.goBack();
373     await expect(todoItem).toHaveCount(2);
374     await page.goBack();
375     await expect(todoItem).toHaveCount(3);
376   });
377
378   test('should allow me to display completed items', async ({ page }) => {
379     await page.getByTestId('todo-item').nth(1).getByRole('checkbox').check();
380     await checkNumberOfCompletedTodosInLocalStorage(page, 1);
381     await page.getByRole('link', { name: 'Completed' }).click();
382     await expect(page.getByTestId('todo-item')).toHaveCount(1);
383   });
384
385   test('should allow me to display all items', async ({ page }) => {
386     await page.getByTestId('todo-item').nth(1).getByRole('checkbox').check();
387     await checkNumberOfCompletedTodosInLocalStorage(page, 1);
388     await page.getByRole('link', { name: 'Active' }).click();
389     await page.getByRole('link', { name: 'Completed' }).click();
390     await page.getByRole('link', { name: 'All' }).click();
391     await expect(page.getByTestId('todo-item')).toHaveCount(3);
392   });
```

demo-todo-app.spec.js U

e2e > demo-todo-app.spec.js > test.describe('Mark all as completed') callback > test('should allow me to mark all items as completed') callback

```
394   test('should highlight the currently applied filter', async ({ page }) => {
395     await expect(page.getByRole('link', { name: 'All' })).toHaveClass('selected');
396
397     //create locators for active and completed links
398     const activeLink = page.getByRole('link', { name: 'Active' });
399     const completedLink = page.getByRole('link', { name: 'Completed' });
400     await activeLink.click();
401
402     // Page change - active items.
403     await expect(activeLink).toHaveClass('selected');
404     await completedLink.click();
405
406     // Page change - completed items.
407     await expect(completedLink).toHaveClass('selected');
408   });
409 });
410
411 async function createDefaultTodos(page) {
412   // create a new todo locator
413   const newTodo = page.getByPlaceholder('What needs to be done?');
414
415   for (const item of TODO_ITEMS) {
416     await newTodo.fill(item);
417     await newTodo.press('Enter');
418   }
419 }
420
421 /**
422  * @param {import('@playwright/test').Page} page
423  * @param {number} expected
424  */
425 async function checkNumberOfTodosInLocalStorage(page, expected) {
426   return await page.waitForFunction(e => {
427     return JSON.parse(localStorage['react-todos']).length === e;
428   }, expected);
429 }
```

demo-todo-app.spec.js U

e2e > demo-todo-app.spec.js > test.describe('Mark all as completed') callback > test('should allow me to mark all items as completed') callback

```
430
431 /**
432  * @param {import('@playwright/test').Page} page
433  * @param {number} expected
434  */
435 async function checkNumberOfCompletedTodosInLocalStorage(page, expected) {
436   return await page.waitForFunction(e => {
437     return JSON.parse(localStorage['react-todos']).filter(i => i.completed).length === e;
438   }, expected);
439 }
440
441 /**
442  * @param {import('@playwright/test').Page} page
443  * @param {string} title
444  */
445 async function checkTodosInLocalStorage(page, title) {
446   return await page.waitForFunction(t => {
447     return JSON.parse(localStorage['react-todos']).map(i => i.title).includes(t);
448   }, title);
449 }
450
```

■ Official Java Implementation Playwright GitHub :

1. https://github.com/microsoft/playwright-java

2. https://github.com/microsoft/playwright

■ Folder Structure :

1. package.json --> node project management file. OR his file contains metadata and dependencies for the project. It specifies the project's name, version, scripts, and dependencies required to run the project.

2. playwright.config.js --> configuration file. OR used to configure and customize Playwright settings for your project. It allows you to specify various options related to browser settings, test environment setup, test execution, and more.

3. e2e folder --> This folder contains all the test files.

4. package-lock.json --> Automatically generated file created by npm (Node Package Manager) to lock down the version of every package installed in a project. It serves as a record of the exact versions of dependencies installed, along with their transitive dependencies (dependencies of dependencies).

5. . gitignore --> used to specify intentionally untracked files that Git should ignore when performing version control operations. It allows you to exclude files or directories from being tracked by Git, which is useful for preventing sensitive or unnecessary files from being included in the repository.

6. playwright.yml --> This a configuration file used to define settings and options for running Playwright tests. It is commonly used in conjunction with continuous integration (CI) systems to automate the execution of Playwright tests as part of a CI/CD pipeline. The playwright.yml file typically contains configuration options related to test environment setup, test execution, and reporting.

7. Utils folder --> This folder contains utility functions, helpers, constants, or other shared code that is used across tests or page objects. It helps in keeping the code DRY (Don't Repeat Yourself) and improves maintainability.

- playwright.config.js :

When we are created new project then automatically configuration file came for you by default.

https://playwright.dev/docs/test-configuration

■ Playwright version : npm playwright -v

■ key fixtures or entities commonly used in Playwright tests :

https://playwright.dev/docs/test-fixtures

In Playwright, the term "fixture" typically refers to the setup and teardown functions used to initialize and clean up test environments. However, Playwright itself doesn't provide fixtures as built-in entities like it does with "context", "page", and "browser". Instead, it allows you to create and manage your own fixtures using testing frameworks such as Jest or Mocha.

That said, Playwright does provide a range of objects and entities that you interact with when writing tests. Here's a list of some key fixtures or entities commonly used in Playwright tests :

1. Browser: Represents a web browser instance. You can create and manipulate browser instances to perform actions such as opening new pages, navigating to URLs, and interacting with page content.

2. Context: Represents a browser context, which is an isolated browsing session. Each context has its own set of cookies, storage, and permissions. You can create multiple contexts within a single browser instance.

3. Page: Represents a web page within a browser context. You can interact with pages to perform actions such as clicking elements, filling forms, and evaluating JavaScript code.

4. Frame: Represents an HTML iframe element within a page. Frames allow you to interact with nested documents embedded within the main page.

5. ElementHandle: Represents a DOM element on a web page. You can use element handles to perform actions such as clicking, typing, and extracting text content from elements.

6. Request: Represents a network request made by a page. You can intercept and modify requests to simulate different network conditions or behaviors.

7. Response: Represents a network response received from a request. You can inspect response data, headers, and status codes to verify behavior or extract information.

8. Viewport: Represents the size and scale of the browser window or page viewport. You can configure viewports to simulate different device resolutions or screen sizes.

These are some of the key fixtures or entities used in Playwright tests. Depending on your testing framework and requirements, you may also define custom fixtures or setup functions to initialize and manage test environments.

Playwright.config.js :

The **playwright.config.js** file is a configuration file used to customize settings and options for running Playwright tests. It allows you to define various configurations related to browsers, test execution, environments, and more. This file is typically used when running tests locally or as part of a test automation setup.

https://playwright.dev/docs/test-configuration

1. testDir :

The `testDir` option in a Playwright configuration file (`playwright.config.js`) specifies the directory where Playwright should look for test files to execute. When running tests, Playwright will search for test files within the directory specified by `testDir`.

For example, if you set `testDir: "./e2e"`, Playwright will search for test files within the `e2e` directory relative to the location of the `playwright.config.js` file.

Here's a breakdown of what this option does:

- testDir: Specifies the directory where Playwright should look for test files.
- `"./e2e"`: Specifies the `e2e` directory relative to the location of the `playwright.config.js` file.

By setting `testDir`, you can organize your test files into a specific directory structure, making it easier to manage and execute tests. This can be particularly useful when you have multiple types of tests (e.g., end-to-end tests, unit tests) or when you want to separate tests into different folders based on functionality or features.

2. fullyParallel :

The `fullyParallel` option in a Playwright configuration file (`playwright.config.js`) enables fully parallel test execution, allowing tests to run concurrently across multiple workers or processes.

When `fullyParallel` is set to `true`, Playwright launches multiple test workers, each running tests independently. This can significantly reduce the overall test execution time, especially for large test suites or environments with multiple test files.

Here's how the `fullyParallel` option works:

- **fullyParallel**: Specifies whether to enable fully parallel test execution.
- `true`: Indicates that fully parallel test execution should be enabled.

When `fullyParallel` is set to `true`, Playwright dynamically distributes tests across available workers, maximizing CPU and resource utilization. This can lead to faster test execution and shorter feedback cycles, making it easier to identify and address issues in the application under test.

3. trace :

The `trace` option in a Playwright configuration file (`playwright.config.js`) allows you to enable or disable tracing for test execution. Tracing provides detailed information about browser interactions, network requests, and performance metrics during test execution, helping you debug and analyze test failures and performance issues.

Here's how the `trace` option works:

- **trace**: Specifies whether to enable tracing for test execution.
- `true`: Indicates that tracing should be enabled.
- `false`: Indicates that tracing should be disabled.

When `trace` is set to `true`, Playwright captures detailed information about browser interactions, network activity, and performance metrics during test execution. This information is then logged or outputted in a trace file, which you can analyze to troubleshoot issues, identify bottlenecks, and optimize test performance.

Tracing can be particularly useful for diagnosing complex issues such as flaky tests, intermittent failures, slow page loads, or network errors. By enabling tracing, you can gain insights into the inner workings of your tests and the application under test, helping you improve test reliability and efficiency.

4. timeout :

The `timeout` option in a Playwright configuration file (`playwright.config.js`) specifies the default timeout for various Playwright operations, such as waiting for page navigation, element visibility, or network requests. This timeout value is specified in milliseconds.

Here's how the `timeout` option works:

- **timeout**: Specifies the default timeout for Playwright operations.
- `30 * 1000`: Represents 30 seconds in milliseconds.

In this example, the `timeout` option is set to `30 * 1000`, which means that the default timeout for Playwright operations is 30 seconds. This timeout value is used when performing various operations in Playwright, such as waiting for elements to become visible, waiting for page navigation to complete, or waiting for network requests to finish.

Setting an appropriate timeout value is important to ensure that tests do not hang indefinitely and fail gracefully if a certain operation takes longer than expected. However, it's also important to balance between setting a timeout that is too short, leading to false positives due to premature timeouts, and setting a timeout that is too long, leading to slow test execution.

■ **To Run the cases :** https://playwright.dev/docs/test-cli

1. npx playwright test : Runs the end-to-end tests Or runs all tests on all browsers in headless mode
2. npx playwright test --headed : Runs tests in headed mode.
3. npx playwright test --ui : Start the interactive UI mode.
4. npx playwright test --project=chromium: Runs the tests only on Desktop Chrome
5. npx playwright test example: Runs the tests in specific file
6. npx playwright test --debug: Runs the tests in debug mode
7. npx playwright test filename.spec.js --debug: Debug specific tests file
8. npx playwright test filename.spec.js:21 --debug: Debug starting from specific line
9. npx playwright codegen: Auto generate tests with codegen - recording
10. npx playwright show-report : to see reports
11. npx playwright test HomePageTest --project=chromium
12. npx playwright test HomePageTest --project=chromium --headed
13. npx playwright test HomePageTest --project=chromium --headed --debug
14. npx playwright test --workers 3 : Runs with 3 workers in parallel
15. npx playwright test filename.spec.js : Runs a specific tests file
16. npx playwright test filename1.spec.js filename2.spec.js : Runs the files specified
17. npx playwright test filename1 filename2 : Runs the files that have filename1, filename2 in the file name
18. npx playwright test -g "check title" : Runs the test with title
19. npx playwright test tests/todo-page.spec.ts : Run a single test file
20. npx playwright test tests/todo-page/ tests/landing-page/ : Run a set of test files
21. npx playwright test --workers=1 : Disable parallelization
22. npx playwright test --reporter=dot : Choose a reporter
23. npx playwright test --help : Ask for help

■ CSS Selector : How to create CSS Selector...?

https://playwright.dev/docs/other-locators

1. If Id is present : tagname#id OR #id

2. If class attribute is present : tagname.class OR .class

3. Based on any attribute : [attribute='value']

4. With traversing from Parent to Child : parenttagname >> childtagname

5. Based on text : text=' '

■ To Record the cases :

npx playwright codegen : Run on terminal & just do steps manually.

npx playwright codegen google.com : Open google browser & do steps manually.

npx playwright codegen --help

npx playwright codegen -o tests/recordedTests.spec.js :create a file whatever we given & store recorded data in it.

npx playwright codegen --target javascript : To specify language to recode the code. js is default.

npx playwright codegen --target javascript -o tests/recordedTests.spec.js

npx playwright codegen --browser chromium : particular browser. It will by default chrome.

npx playwright codegen --device "iPhone 13" : particular device

npx playwright codegen --viewport-size "1280, 720" : specific browser window resolution or size x & y axis

npx playwright codegen --color-scheme=dark : open website in dark mode

npx playwright codegen --color-scheme=dark google.com : open website in dark mode

■ Locators : https://playwright.dev/docs/locators

1. page.getByRole() to locate by explicit and implicit accessibility attributes.

2. page.getByText() to locate by text content.

3. page.getByLabel() to locate a form control by associated label's text.

4. page.getByPlaceholder() to locate an input by placeholder.

5. page.getByAltText() to locate an element, usually image, by its text alternative.

6. page.getByTitle() to locate an element by its title attribute.

7. page.getByTestId() to locate an element based on its data-test-id attribute (other attributes can be configured).

■ Annotations : https://playwright.dev/docs/test-annotations

1. test.skip() marks the test as irrelevant. Playwright Test does not run such a test. Use this annotation when the test is not applicable in some configuration.

2. test.fail() marks the test as failing. Playwright Test will run this test and ensure it does indeed fail. If the test does not fail, Playwright Test will complain.

3. test.fixme() marks the test as failing. Playwright Test will not run this test, as opposed to the failure annotation. Use fixme when running the test is slow or crashes.

4. test.slow() marks the test as slow and triples the test timeout.

5. test.only() only that particular testcase will run. If you add .only for 2 tests, then 2 cases will run.

Assertions : https://playwright.dev/docs/test-assertions

```
/* Configure projects for major browsers */
projects: [
  {
    name: "chromium",
    use: { ...devices["Desktop Chrome"], headless: false },
  },
```

- If you use **headless : false** inside playwright.config.js file then all test cases will run on headed mode.

03-UI-basics-test.spec.js U

e2e > 03-UI-basics-test.spec.js > test("@Web Browser Context-Validating Error login") callback

Click here to ask Blackbox to help you code faster

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

test("@Web Browser Context-Validating Error login", async ({ browser }) => {
  const context = await browser.newContext(); // Creates a new browser context. It won't share cookies/cache with other browser contexts.
  const page = await context.newPage(); // Creates a new page in the browser context.
  // without writing above 2 lines we can also use page fixture - refer next (Web UI Controls) testcase

  await page.goto("https://rahulshettyacademy.com/loginpagePractise/"); // search particular url
  console.log(await page.title()); // To print the title of the webpage

  expect(await page.title()).toEqual(
    "LoginPage Practise | Rahul Shetty Academy"
  );
  await expect(page).toHaveTitle("LoginPage Practise | Rahul Shetty Academy");

```

Locator : https://playwright.dev/docs/locators

- Use selectorsHub OR ChroPath as a chrome extension for best selection of selectors on webpage.

- Here in below example :

- `[...]`: Denotes an attribute selector in CSS.
- `style`: Refers to the inline style attribute of HTML elements.
- `*=`: Represents the attribute substring matching operator, which selects elements where the attribute contains the specified substring.
- `'block'`: The substring being searched for within the `style` attribute. In this case, it's the string "block".

So, `[style*='block']` will select all elements that have inline styles containing the substring "block". For example, it would match an element like `<div style="display: block; color: red;">`, as the inline style contains the substring "block".

Code Continue…

```
03-UI-basics-test.spec.js U
e2e > 03-UI-basics-test.spec.js > test("@Web Browser Context-Validating Error login") callback
16  const userName = page.locator("#username"); // Rel CSS Selector - id
17  const signIn = page.locator("#signInBtn"); // Rel CSS Selector - id
18  await userName.fill("rahulshetty");
19  await page.locator("[type='password']").fill("learning"); // Based on any attribute i.e. [attribute='value']
20  await signIn.click();
21
22  console.log(await page.locator("[style*='block']").textContent()); // Incorrect username/password.
23  await expect(page.locator("[style*='block']")).toContainText("Incorrect"); // Verify Wrong Login Details.
```

→ Error box will blink like it will disappear after some second, so that's a reason we have used like that css selector & developer written code will change like this once alert-box blinks.

Before Alert-Box Came :

```
Elements  Console  Sources  Network  Performance  Memory  Application  Lighthouse
▼<div class="alert alert-danger col-md-12" style="display: none;"> == $0
   <strong>Incorrect</strong>
   " username/password."
 </div>
```

After Alert-Box Came :

```
Elements  Console  Sources  Network  Performance  Memory  Application  Lighthouse
▼<div class="alert alert-danger col-md-12" style="display: block;"> == $0
   <strong>Incorrect</strong>
   " username/password."
 </div>
```

■ Locators which extract multiple web elements in page :

Code Continue…

```
03-UI-basics-test.spec.js U
e2e > 03-UI-basics-test.spec.js > ...
25  //type - fill
26  await userName.fill(""); // clear existing content inside username field
27  await userName.fill("rahulshettyacademy"); // give proper username
28  await signIn.click();
29
30  const cardTitles = page.locator(".card-body a"); // Based on class attribute i.e. .class tagname as child
31
32  console.log(await cardTitles.first().textContent()); // get first element means index 0 --> iphone X
33  console.log(await cardTitles.nth(1).textContent()); // get second element means index 1 --> Samsung Note 8
```

■ how synchronization works :

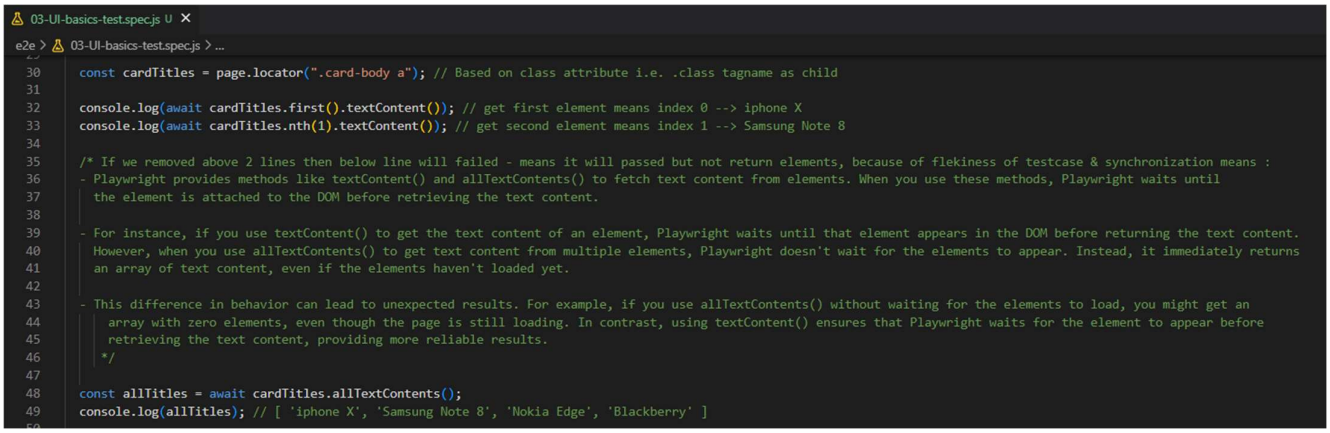
- If I want to grab text of all products inside one single step, because we cannot verify nth(x) if there is 50 products & more.

https://playwright.dev/docs/actionability

https://github.com/microsoft/playwright/issues/11023

https://playwright.dev/docs/api/class-locator#locator-click

Code Continue…



```
03-UI-basics-test.spec.js U
e2e > 03-UI-basics-test.spec.js > ...
30  const cardTitles = page.locator(".card-body a"); // Based on class attribute i.e. .class tagname as child
31
32  console.log(await cardTitles.first().textContent()); // get first element means index 0 --> iphone X
33  console.log(await cardTitles.nth(1).textContent()); // get second element means index 1 --> Samsung Note 8
34
35  /* If we removed above 2 lines then below line will failed - means it will passed but not return elements, because of flekiness of testcase & synchronization means :
36  - Playwright provides methods like textContent() and allTextContents() to fetch text content from elements. When you use these methods, Playwright waits until
37    the element is attached to the DOM before retrieving the text content.
38
39  - For instance, if you use textContent() to get the text content of an element, Playwright waits until that element appears in the DOM before returning the text content.
40    However, when you use allTextContents() to get text content from multiple elements, Playwright doesn't wait for the elements to appear. Instead, it immediately returns
41    an array of text content, even if the elements haven't loaded yet.
42
43  - This difference in behavior can lead to unexpected results. For example, if you use allTextContents() without waiting for the elements to load, you might get an
44      array with zero elements, even though the page is still loading. In contrast, using textContent() ensures that Playwright waits for the element to appear before
45      retrieving the text content, providing more reliable results.
46    */
47
48  const allTitles = await cardTitles.allTextContents();
49  console.log(allTitles); // [ 'iphone X', 'Samsung Note 8', 'Nokia Edge', 'Blackberry' ]
```

Assignment - 01 :

Action	Attached	Visible	Stable	Receives Events	Enabled	Editable
check	Yes	Yes	Yes	Yes	Yes	-
click	Yes	Yes	Yes	Yes	Yes	-
dblclick	Yes	Yes	Yes	Yes	Yes	-
tap	Yes	Yes	Yes	Yes	Yes	-
uncheck	Yes	Yes	Yes	Yes	Yes	-
hover	Yes	Yes	Yes	Yes	-	-
scrollIntoViewIfNeeded	Yes	Yes	Yes	-	-	-
screenshot	Yes	Yes	Yes	-	-	-
fill	Yes	Yes	-	-	Yes	Yes
selectText	Yes	Yes	-	-	-	-
dispatchEvent	Yes	-	-	-	-	-
focus	Yes	-	-	-	-	-
getAttribute	Yes	-	-	-	-	-

1. **Objective**: Verify that a user can successfully login to the web client application using valid credentials.
2. **Preconditions**:
   - Ensure that the web client application is accessible via the URL: https://rahulshettyacademy.com/client.
   - Have valid login credentials ready (email and password).

3. **Test Steps**:
   3.1. Navigate to the URL: https://rahulshettyacademy.com/client.
   3.2. Enter a valid email address into the "Email" field.
   3.3. Enter a valid password into the "Password" field.
   3.4. Click on the "Login" button.
   3.5. Wait for the page to load completely, ensuring that all API calls are made and the network is in an idle state.
   3.6. Verify that the user is redirected to the dashboard or home page of the application.
   3.7. Confirm that the list of product titles is displayed on the page.
   3.8. Compare the list of product titles with the expected list of titles (if available).
   3.9. Check if the specified product name ("zara coat 3" in this case) is present in the list of product titles.
   3.10. If the product name is found, it indicates that the login was successful.
   3.11. Log out of the application (if applicable) to complete the test.
4. **Expected Results**:
   - The user should be able to login successfully with the provided credentials.
   - After logging in, the user should be redirected to the dashboard or home page.
   - The list of product titles should be displayed on the page, indicating that the page has loaded successfully.
   - The specified product name ("zara coat 3") should be present in the list of product titles.
   - No errors or exceptions should occur during the login process.
5. **Test Data**:
   - Email: anshika@gmail.com
   - Password: Iamking@000
   - Expected product name: "zara coat 3"

04-ClientApp.spec.js U

e2e > 04-ClientApp.spec.js > test("@Web Client App login") callback

Click here to ask Blackbox to help you code faster

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

test("@Web Client App login", async ({ page }) => {
  //js file- Login js, DashboardPage
  const email = "anshika@gmail.com";
  const productName = "zara coat 3";
  const products = page.locator(".card-body");

  await page.goto("https://rahulshettyacademy.com/client");
  await page.locator("#userEmail").fill(email);
  await page.locator("#userPassword").fill("Iamking@000");
  await page.locator("[value='Login']").click();
  // Wait until all the api calls are made in network tab. 'networkidle' means wait until the network comes to idle state.
  await page.waitForLoadState("networkidle");
  // OR : Alternate way - simply wait for first element to be visible.
  await page.locator(".card-body b").first().waitFor();

  // allTextContents() : playwright will not wait for this method because of synchronization, it will return you empty array if you not write any one line out of above 2 lines.
  const titles = await page.locator(".card-body b").allTextContents();
  console.log(titles);
```

■ Handling UI Components (Dropdowns, Radio Button, Checkbox) :

- https://playwright.dev/docs/input#checkboxes-and-radio-buttons

- https://playwright.dev/docs/input#select-options

- await is required only when we are performing the actions.

const userName = page.locator("#username"); // Not Needed

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

- In Below code so in line: 67 actual action is performed outside this expect bracket only, so that's a reason we have given await at beginning.

- But In line: 72 actual action is performing inside this expect bracket only, so that's a reason we have given await inside the expect brackets. Outside we have only checking it's a true or false there is no action is happening.

03-UI-basics-test.spec.js U

e2e > 03-UI-basics-test.spec.js > ...

```
56  test("@Web UI Controls", async ({ page }) => {
57    await page.goto("https://rahulshettyacademy.com/loginpagePractise/");
58    const documentLink = page.locator("[href*='documents-request']");
59    const dropdown = page.locator("select.form-control");
60
61    await dropdown.selectOption("consult");
62    // await page.pause(); // Pause execution of the test to allow for manual inspection or debugging.
63
64    await page.locator(".radiotextsty").last().click(); // check user radio button
65    await page.locator("#okayBtn").click(); // click ok on popup or accept the popup
66    console.log(await page.locator(".radiotextsty").last().isChecked()); // true
67    await expect(page.locator(".radiotextsty").last()).toBeChecked();
68
69    await page.locator("#terms").click(); // Agree to the terms and conditions
70    await expect(page.locator("#terms")).toBeChecked();
71    await page.locator("#terms").uncheck(); // uncheck to the terms and conditions
72    expect(await page.locator("#terms").isChecked()).toBeFalsy();
73
74    // check whether the link is blinking or not on UI
75    await expect(documentLink).toHaveAttribute("class", "blinkingText");
76  });
```

■ Handling UI Components (Child Windows) :

- I want to know whether this link is blinking or not on UI, we are able to see its blinking, but we want to verify using automation.

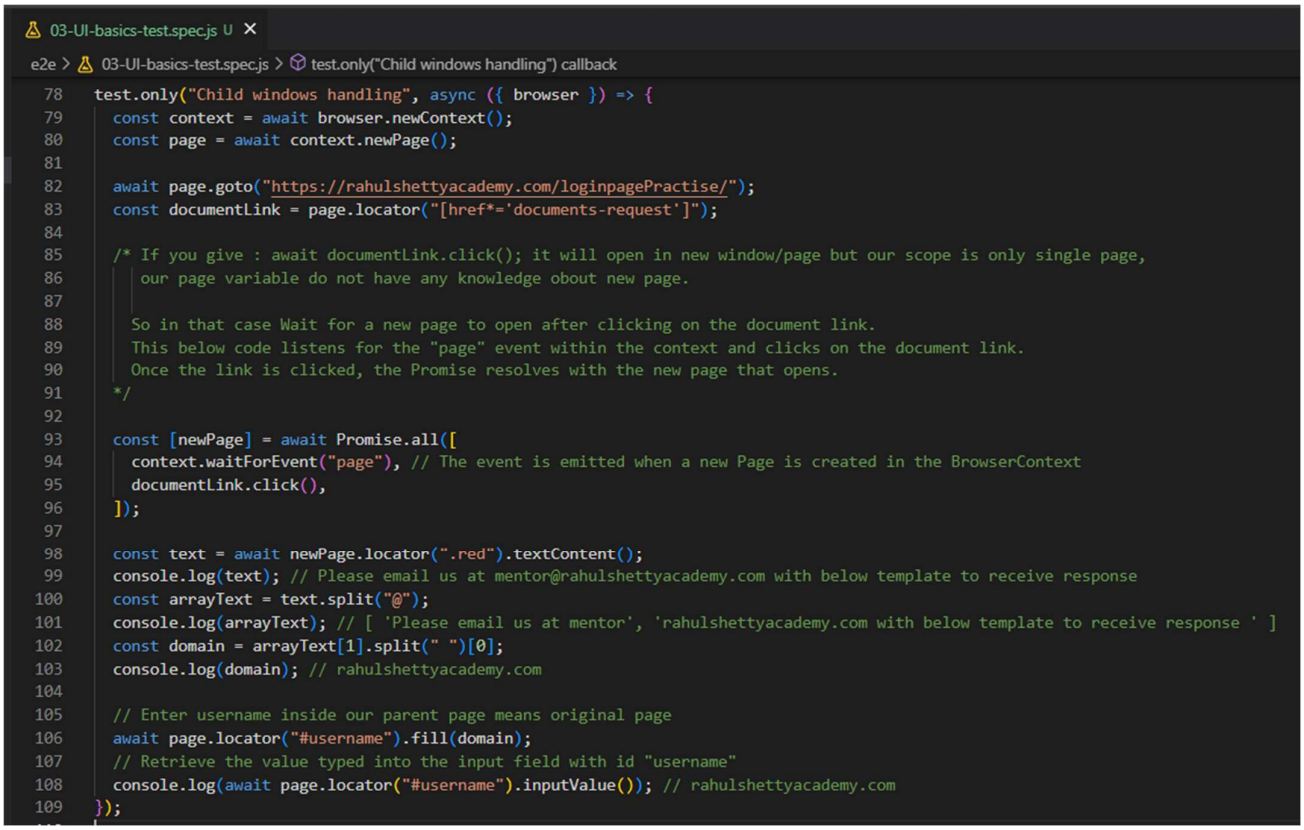
- If you wrote like below on line : 92 , so then also no working because when you click then only, we are able to see the new page, & also we are waiting for that new page that is also not possible because when we click on documentLink then only new page is open. So we need to do something so that both can run parallelly. So in that case we no need to use await because we are running synchronously. But we make sure that we want to make sure that wait until this parallel execution happen & then go to next step.

```
await context.waitForEvent("page");
await documentLink.click();
```

So for that we can use Promise.all() Method & list out all the steps whatever you want to parallelly do. So when execution start context.waitForEvent() step is started execution & moved in pending state, then perform click action, & it will not move out of scope until both promises will be fulfilled.

Also first line will context.waitForEvent() will return newPage & doc.click() is a just click operation it will not return anything.

```
const [newPage] = await Promise.all([
  context.waitForEvent("page"), // The event is emitted when a new Page is created in the BrowserContext
  documentLink.click(),
]);
```



- **Handling Inputs :** https://playwright.dev/docs/input
- **Handling Dialogs :** https://playwright.dev/docs/dialogs
- **Handling Frames :** https://playwright.dev/docs/frames

1. Auto-retrying assertions :

await expect(locator).toBeAttached()	Element is attached.
await expect(locator).toBeChecked()	Checkbox is checked.
await expect(locator).toBeDisabled()	Element is disabled.
await expect(locator).toBeEditable()	Element is editable.
await expect(locator).toBeEmpty()	Container is empty.
await expect(locator).toBeEnabled()	Element is enabled.
await expect(locator).toBeFocused()	Element is focused.
await expect(locator).toBeHidden()	Element is not visible.
await expect(locator).toBeInViewport()	Element intersects viewport.
await expect(locator).toBeVisible()	Element is visible.
await expect(locator).toContainText()	Element contains text.
await expect(locator).toHaveAttribute()	Element has a DOM attribute.
await expect(locator).toHaveClass()	Element has a class property.
await expect(locator).toHaveCount()	List has exact number of children.

await expect(locator).toHaveCSS()	Element has CSS property.
await expect(locator).toHaveId()	Element has an ID.
await expect(locator).toHaveJSProperty()	Element has a JavaScript property.
await expect(locator).toHaveScreenshot()	Element has a screenshot.
await expect(locator).toHaveText()	Element matches text.
await expect(locator).toHaveValue()	Input has a value.
await expect(locator).toHaveValues()	Select has options selected.
await expect(page).toHaveScreenshot()	Page has a screenshot.
await expect(page).toHaveTitle()	Page has a title.
await expect(page).toHaveURL()	Page has a URL.
await expect(response).toBeOK()	Response has an OK status.

2. non-retrying assertions :

expect(value).toBe()	Value is the same.
expect(value).toBeCloseTo()	Number is approximately equal.
expect(value).toBeDefined()	Value is not undefined.
expect(value).toBeFalsy()	Value is falsy, e.g. false, 0, null, etc.
expect(value).toBeGreaterThan()	Number is more than
expect(value).toBeGreaterThanOrEqual()	Number is more than or equal.
expect(value).toBeInstanceOf()	Object is an instance of a class.
expect(value).toBeLessThan()	Number is less than
expect(value).toBeLessThanOrEqual()	Number is less than or equal.
expect(value).toBeNaN()	Value is NaN
expect(value).toBeNull()	Value is null.
expect(value).toBeTruthy()	Value is truthy, i.e. not false, 0, null, etc.
expect(value).toBeUndefined()	Value is undefined.
expect(value).toContain()	String contains a substring.
expect(value).toContain()	Array or set contains an element.
expect(value).toContainEqual()	Array or set contains a similar element.
expect(value).toEqual()	Value is similar - deep equality and pattern matching.
expect(value).toHaveLength()	Array or string has length.
expect(value).toHaveProperty()	Object has a property.
expect(value).toMatch()	String matches a regular expression.
expect(value).toMatchObject()	Object contains specified properties.
expect(value).toStrictEqual()	Value is similar, including property types.

expect(value).toThrow() Function throws an error.

expect(value).any() Matches any instance of a class/primitive.

expect(value).anything() Matches anything.

expect(value).arrayContaining() Array contains specific elements.

expect(value).closeTo() Number is approximately equal.

expect(value).objectContaining() Object contains specific properties.

expect(value).stringContaining()String contains a substring.

expect(value).stringMatching() String matches a regular expression.

■ Difference Between fill & type method in Playwright :

Note : type is deprecated in latest version of playwright.

> `locator.type()` is marked as deprecated, with the suggestion `In most cases, you should use locator.fill(value[, options])` instead.
>
> `.type` focuses the input and puts in the text right **next to the previous value**.
> `.fill` on the other hand **clears up the input** and fills it with new text, behaving like
>
> ```
> await locator.clear()
> await locator.type()
> ```
>
> We have been using `.type` in our tests and thanks to that, found a bug in our codebase related to unwanted input. We would not have found it with using `.fill` — and thus not sure we should migrate to a less informative method.

➢ https://playwright.dev/docs/release-notes#version-138

- **`fill()`**: This method is used to set the value of form inputs directly. It simulates a user typing or pasting a complete value into the input field all at once. It does not trigger individual keyboard events for each character typed. As a result, some JavaScript-driven autocomplete or suggestion features that rely on keyboard events may not be triggered properly when using **`fill()`**. This can lead to issues where autocomplete suggestions do not appear as expected.
- **`type()`**: On the other hand, the **`type()`** method simulates user input by sending individual keyboard events for each character in the provided string. This closely mimics actual user typing behavior. Since **`type()`** triggers keyboard events, JavaScript-driven autocomplete features are more likely to be triggered correctly. As a result, autocomplete suggestions are more likely to appear when using **`type()`**.

1. fill method : This method is used to set the value of an input field by replacing its existing content. It clears the input field and then fills it with the specified text. For example, in JavaScript, you can use page.fill(selector, text) to fill an input field with the provided text.

```
await page.fill('input[name="username"]', 'your_username');
```

2. type method : The type method is used to simulate keyboard typing and can be used to interact with input fields, just like a real user typing on a keyboard. It can be used for entering text, keypress events, and other keyboard interactions. In JavaScript, you can use page.type(selector, text) to type text into an input field.

```
await page.type('input[name="password"]', 'your_password');
```

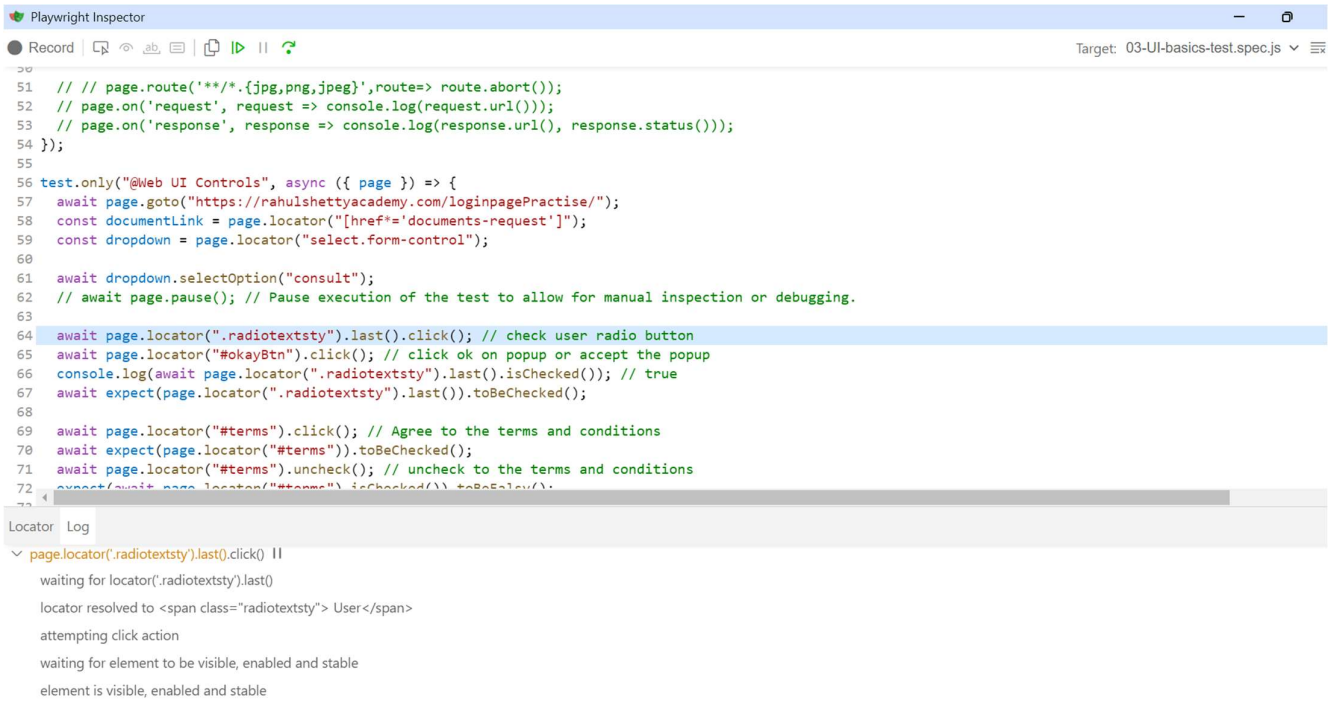
The key difference is that fill is typically used for replacing the entire content of an input field, while type simulates keyboard input, which can be useful for scenarios where you need to emulate typing character by character, triggering events like keypress or keyup, or for interacting with complex input fields, like date pickers or dropdowns.

■ Playwright Inspector :

https://playwright.dev/docs/debug

Run : npx playwright test e2e/03-UI-basics-test.spec.js --debug

- You can able to see all the logs inside the inspector.

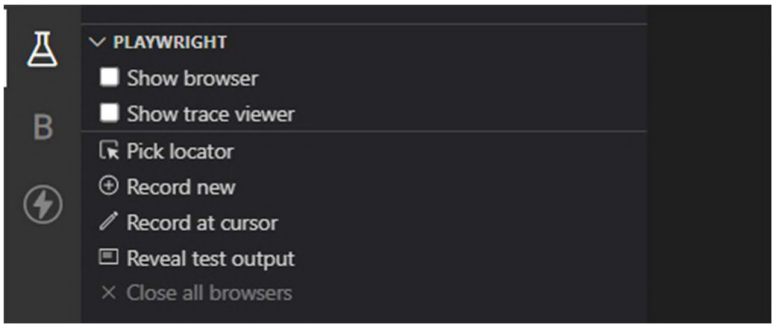


■ Playwright Record Automation Scripts :

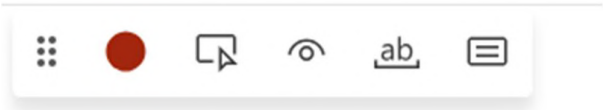
Method : 1 - Using Vs-Code Extension

1. Install above extension.

2. Click on Testing button & then Record new :



3. Perform the testcase & see code is generated inside vs-code file. Also you are able to pick a locator, assert the value, assert visibility, assert text, etc.



1. Red Button is for stop the Recording

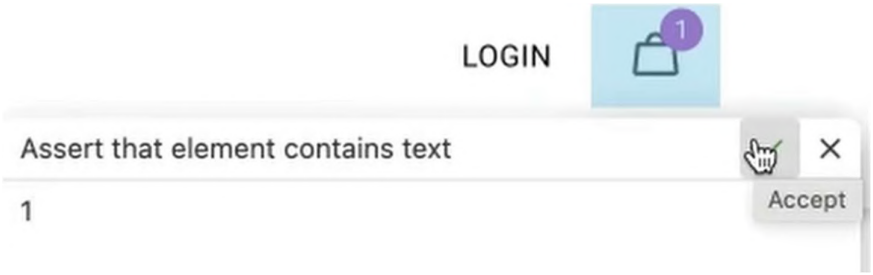
2. Pick a locator

3. Assert Visibility

4. Assert Text

5. Assert Visibility

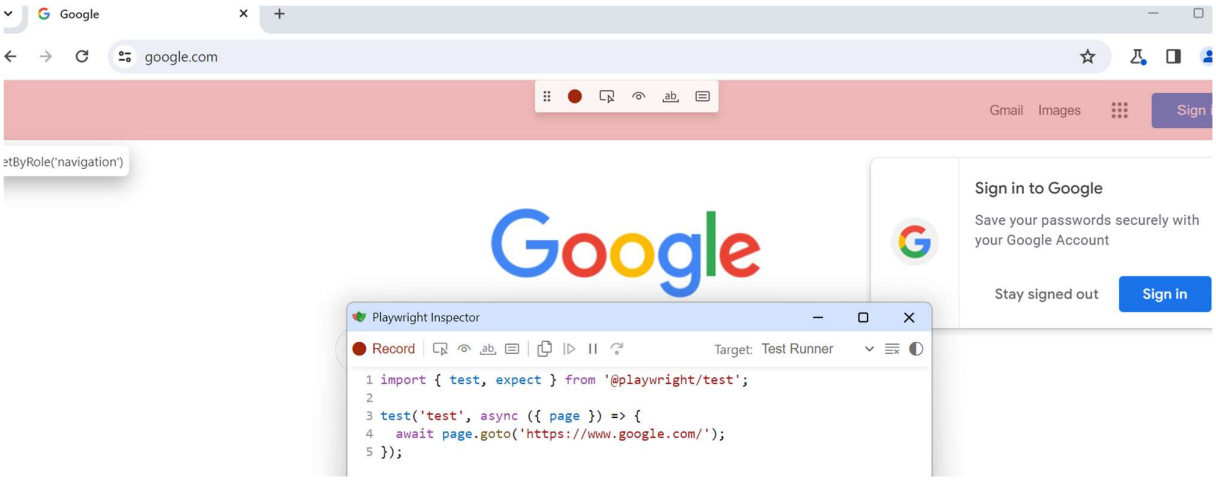
4. Also we can be able to give assertions for that element contains text in the textbox :



5. Also we can be able to run testcase from there itself using run button.

Method : 2 - Using Command Line :

Run : npx playwright codegen https://www.google.com



- Perform whatever automation you want to do & then Copy the script inside the Playwright Inspector & paste our code.

Playwright Reports : https://playwright.dev/docs/test-reporters

- If we want to see different reports, then add config accordingly OR run commands :

- If you want to create a screenshot for every steps. If you use 'only-on-failure' then it will capture screenshot only when test failed. If you use 'off' then it will not capture screenshots. & Stored inside playwright-report/ data folder. & also inside the report also.

- Trace option is used for allows you to enable or disable tracing for various events that occur during the execution of your tests. Tracing provides detailed information about the actions taken by Playwright during test execution, including network requests, page events, and more. Enabling tracing can be helpful for debugging, diagnosing issues, and understanding the behavior of your tests.

Trace is used to see what happened on each and every step of execution with complete log. For trace also there is below options : off, on, on-all-retries, on-first-retry, retain-on-failure, retain-on-first-failure, retry-with-trace. & we can be able to see traces inside playwright-report/ trace folder & also inside the report in zip format if we make on.

```
26  use: {
27    // to capture screenshot of tests
28    screenshot: "only-on-failure",
29    /* Collect trace when retrying the failed test. See https://playwright.dev/docs/trace-viewer */
30    trace: "on-first-retry",
31
```

1. npx playwright test --reporter=line OR npx playwright test reporters --reporter=line --project chromium :

It prints a line for each test being run. It uses a single line to report last finished test and prints failures when they occur. Line reporter is useful for large test suites where it shows the progress but does not spam the output by listing all the tests.

e.g. 3 passed (9.3s)

2. npx playwright test --reporter=list OR npx playwright test reporters --reporter=list --project chromium --headed :

It prints a line for each test being run & List reporter is default.

e.g.

✓ 1 [chromium] › reporters.spec.js:3:1 › Test-1 (2.8s)

✓ 2 [chromium] › reporters.spec.js:8:1 › Test-2 (1.5s)

✓ 3 [chromium] › reporters.spec.js:13:1 › Test-3 (2.1s)

3. npx playwright test --reporter=dot OR npx playwright test reporters --reporter=dot --project chromium :

Dot reporter is very concise - it only produces a single character per successful test run.

shows green dot for passed cases & capital F for failed cases.

e.g. ···

4. npx playwright test --reporter=html OR npx playwright test reporters --reporter=html --project chromium :

HTML reporter produces a self-contained folder that contains report for the test run that can be served as a web page.

By default, HTML report is opened automatically if some of the tests failed. You can control this behaviour via the open property in the Playwright config.

html report will generate folder named as playwright-report & inside that index.html file.

5. npx playwright show-report : A quick way of opening the last html test run report.

6. npx playwright test --reporter=blob OR npx playwright test reporters --reporter=blob --project chromium :

Blob reports contain all the details about the test run and can be used later to produce any other report. Their primary function is to facilitate the merging of reports from sharded tests.

It will create one folder named as blob-report & create file name ad report.zip.

7. npx playwright test --reporter=json OR npx playwright test reporters --reporter=json --project chromium :

JSON reporter produces an object with all information about the test run.

Above command will generate json report on terminal itself.

8. npx playwright test reporters --reporter=junit --project chromium

JUnit reporter produces a JUnit-style xml report.

Above command will generate json report on terminal itself.

9. If we want multiple reports then we want to do configurations in playwright.config file like below :

```
reporter: [['list'],
            ['html'],
            ['junit', { outputFile: 'results.xml' }],
        ['json', { outputFile: 'results.json' }]
            ],
```

How to Generate Allure Reports : https://www.npmjs.com/package/allure-playwright

1. Installation of "allure-playwright" module :

```
npm i -D @playwright/test allure-playwright
```

2. Installing allure command line :

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

OR

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

3. playwriht.config.js :

Either add allure-playwright into playwright.config.js :

```
{
  reporter: "allure-playwright";
}
```

OR Pass the same value via config file :

```
{
  reporter: [["line"], ["allure-playwright"]];
}
```

OR Pass the same value via command line :

```
npx playwright test --reporter=line,allure-playwright
```

4. Run the Tests:

just run below command in terminal after adding above code in config file :

```
npx playwright test tests/Reporters.spec.js
```

Specify location for allure results:

```
set ALLURE_RESULTS_DIR=my-allure-results
npx playwright test --reporter=line,allure-playwright
```

5. Generate Allure Report :

```
allure generate my-allure-results -o allure-report --clean
```

6. Open Allure Report :

```
allure open allure-report
```

End to End Web Automation Scenarios :

Test Case: Verify Successful Purchase of ZARA COAT 3

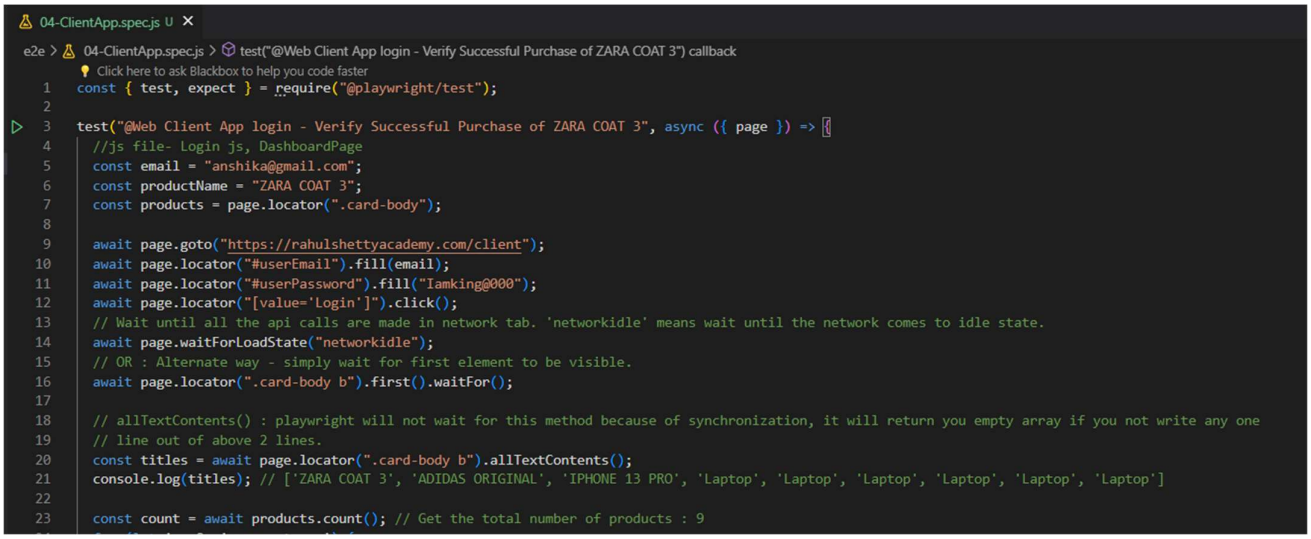
Steps:

1. Navigate to the web client application URL: https://rahulshettyacademy.com/client.
2. Enter the valid email address "anshika@gmail.com" into the email input field.
3. Enter the password "Iamking@000" into the password input field.
4. Click on the "Login" button.
5. Wait for the page to load and ensure successful login.
6. Click on the "Add To Cart" button next to the product "ZARA COAT 3".
7. Navigate to the cart by clicking on the "Cart" link or icon.
8. Wait for the cart items to load and ensure that "ZARA COAT 3" is added to the cart.
9. Click on the "Checkout" button to proceed to the checkout page.
10. Enter "ind" in the "Country" input field to select "India" from the autocomplete suggestions.
11. Wait for the autocomplete suggestions to load and ensure that "India" is displayed in the suggestions.
12. Click on the "India" option from the autocomplete suggestions.
13. Verify that the email address "anshika@gmail.com" is pre-filled in the billing information section.
14. Click on the "Place Order" button to complete the purchase.
15. Wait for the confirmation message "Thank you for the order." to appear.
16. Verify that the confirmation message is displayed.
17. Note down the order ID displayed on the confirmation page.
18. Click on the "My Orders" button to navigate to the orders page.
19. Wait for the orders page to load and ensure that the new order appears in the list of orders.
20. Find the newly placed order using the recorded order ID.
21. Click on the details button of the newly placed order to view order details.
22. Verify that the order details page displays the correct order ID matching the recorded order ID.

Expected Result:

- The user should be able to successfully purchase the "ZARA COAT 3" product.
- The checkout process should proceed smoothly without any errors.
- The confirmation message "Thank you for the order." should be displayed after placing the order.
- The newly placed order should appear in the list of orders on the orders page.
- The order details page should display the correct order ID matching the recorded order ID.

This test case ensures that the user can successfully complete a purchase of the "ZARA COAT 3" product on the web client application and that all order details are displayed accurately.



```
04-ClientApp.spec.js U
e2e > 04-ClientApp.spec.js > test("@Web Client App login - Verify Successful Purchase of ZARA COAT 3") callback
24    for (let i = 0; i < count; ++i) {
25      // Loop through each product
26      if ((await products.nth(i).locator("b").textContent()) === productName) {
27        // Check if the product name matches the desired product
28        // If there's a match, click on the "Add To Cart" button
29        await products.nth(i).locator("text= Add To Cart").click();
30        // Exit the loop since the desired product is found
31        break;
32      }
33    }
34
35    await page.locator("[routerlink*='cart']").click();
36    // await page.pause();
37
38    // Wait until the cart items are loaded and displayed - there is 4 li tag inside div but we want 1st so we use first.
39    await page.locator("div li").first().waitFor();
40    const bool = await page.locator("h3:has-text('zara coat 3')").isVisible();
41    expect(bool).toBeTruthy();
42    await page.locator("text=Checkout").click();
43
44    /**
45     * Autocomplete suggestions are not appearing when using fill(), but they do appear when using type(), it suggests that the autocomplete feature
46     * relies on keyboard events to trigger the suggestions. Therefore, using type() is the appropriate choice to ensure that the autocomplete suggestions
47     * are properly triggered during automation.
48     */
49    await page.locator("[placeholder*='Country']").type("ind");
50    // Select India as a country from the Auto-suggestion Dropdown
51    const dropdown = page.locator(".ta-results");
52    await dropdown.waitFor(); // Wait until the autocomplete results are loaded
53    const optionsCount = await dropdown.locator("button").count();
54
55    for (let i = 0; i < optionsCount; ++i) {
56      const text = await dropdown.locator("button").nth(i).textContent();
57      if (text === " India") {
58        await dropdown.locator("button").nth(i).click();
59        break;
60      }
61    }
```

```
04-ClientApp.spec.js U
e2e > 04-ClientApp.spec.js > ...
63    expect(page.locator(".user__name [type='text']").first()).toHaveText(email);
64    await page.locator(".action__submit").click(); // click on Place Order Button
65    await expect(page.locator(".hero-primary")).toHaveText(
66      " Thankyou for the order. "
67    );
68    const orderId = await page
69      .locator(".em-spacer-1 .ng-star-inserted")
70      .textContent();
71    console.log(orderId); // 661b717da86f8f74dcc0cc17
72
73    // Click on orders on top right to go back to dashboard and check whether the new order appears or not
74    await page.locator("button[routerlink*='myorders']").click();
75    await page.locator("tbody").waitFor();
76    const rows = await page.locator("tbody tr");
77
78    for (let i = 0; i < (await rows.count()); ++i) {
79      const rowOrderId = await rows.nth(i).locator("th").textContent();
80      if (orderId.includes(rowOrderId)) {
81        await rows.nth(i).locator("button").first().click();
82        break;
83      }
84    }
85    const orderIdDetails = await page.locator(".col-text").textContent();
86    expect(orderId.includes(orderIdDetails)).toBeTruthy();
87  });
88
```

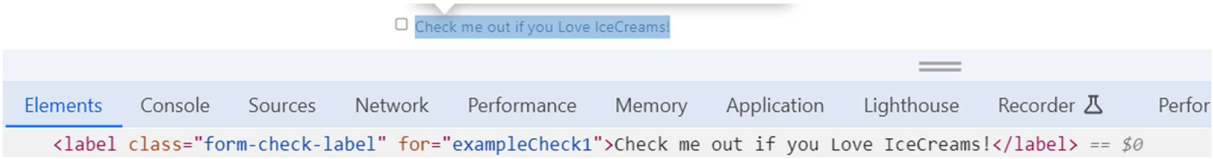
■ Playwright Unique GetBy Locators for Smart Testing & Test Runner usage :

- https://playwright.dev/docs/api/class-locator

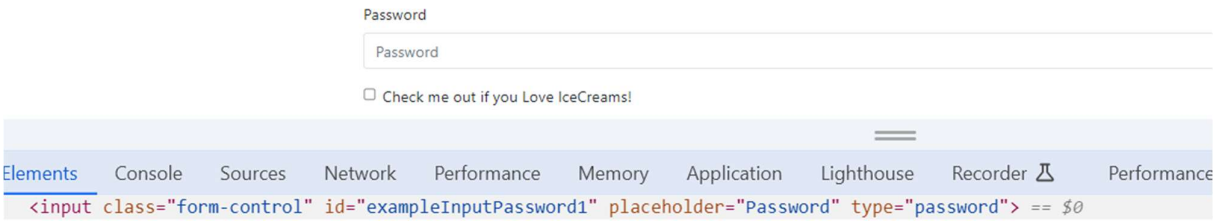
```
getAttribute
getByAltText
getByLabel
getByPlaceholder
getByRole
getByTestId
getByText
getByTitle
```

→ npx playwright test ./e2e/05-Special-Locators.spec.js --ui : Used to open Playwright test runner.

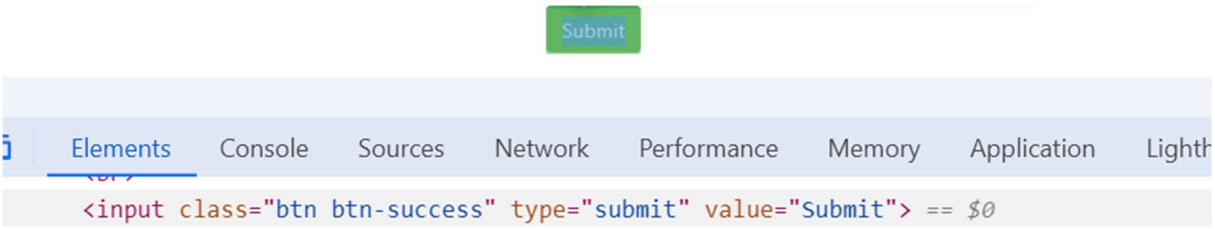
- getByLabel() :



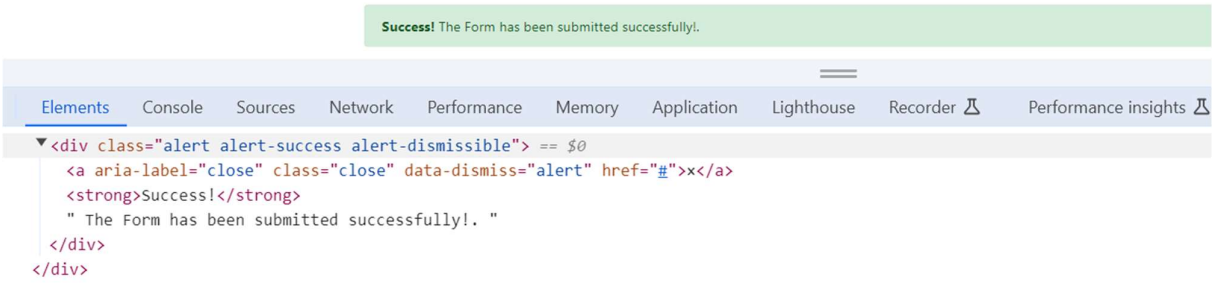
- getByPlaceholder() :



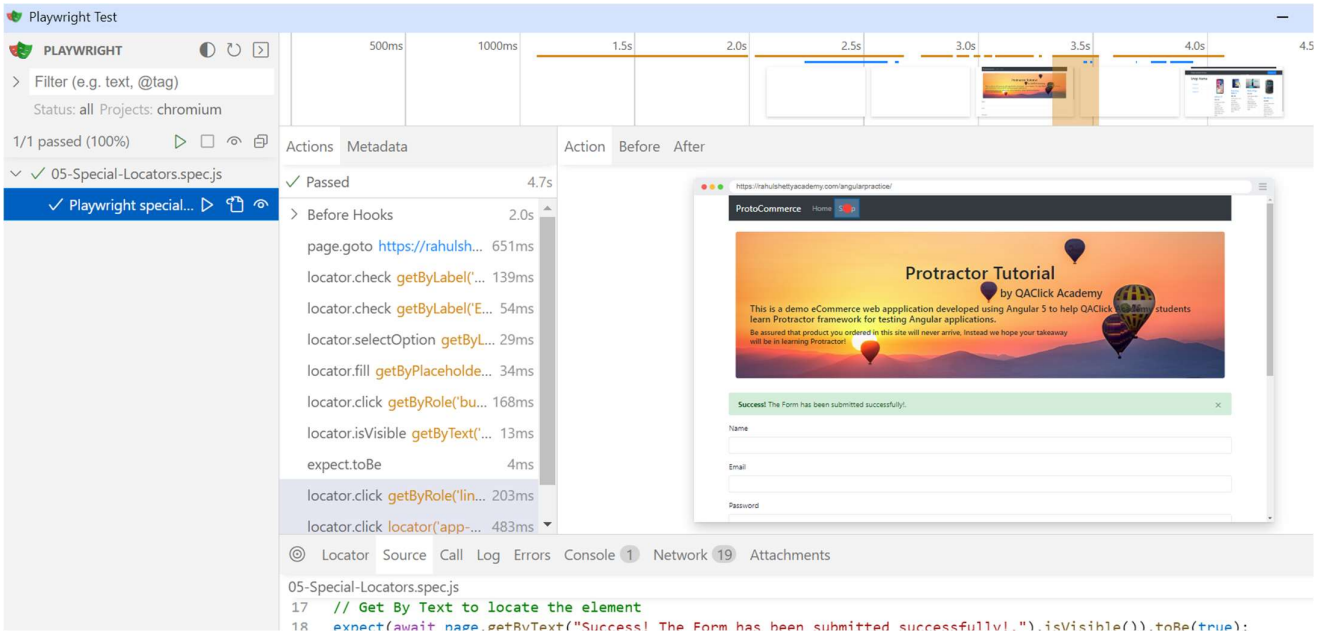
- getByRole() :



- getByText() :



→ Test Runner Demo :



05-Special-Locators.spec.js U

e2e > 05-Special-Locators.spec.js > ...

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

test("Playwright special locators", async ({ page }) => {
  await page.goto("https://rahulshettyacademy.com/angularpractice/");

  // Get By Label to locate the element
  await page.getByLabel("Check me out if you Love IceCreams!").check();
  await page.getByLabel("Employed").check();
  await page.getByLabel("Gender").selectOption("Female");

  // Get By Placeholder to locate the element
  await page.getByPlaceholder("Password").fill("abc123");

  // Get By Role to locate the element - role & name of that button
  await page.getByRole("button", { name: "Submit" }).click();

  // Get By Text to locate the element
  expect(await page.getByText("Success! The Form has been submitted successfully!.").isVisible()).toBe(true);

  // Get By Role to locate the element - role & name of that link
  await page.getByRole("link", { name: "Shop" }).click();

  // Get By Tag Name
  await page.locator('app-card').filter({hasText: 'Nokia Edge'}).getByRole('button').click();
  // await page.locator('app-card').filter({hasText: 'Nokia Edge'}).getByRole('button', {name: 'Add'}).click();
  // Here above if you not given {name: 'Add'} that is also fine because there is only one add button in Nokia card.
});
```

Playwright Handling Calendar :

- https://github.com/microsoft/playwright/issues/3578

- https://www.lambdatest.com/learning-hub/automate-date-pickers-with-playwright

- https://github.com/nhn/tui.calendar/blob/main/playwright.config.ts

06-Handling-Calender.spec.js U

e2e > 06-Handling-Calender.spec.js > ...

Click here to ask Blackbox to help you code faster

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

test.describe("Playwright Handling Calendar tests", () => {
  test("Calendar Validations", async ({ page }) => {
    const year = "2028";
    const monthNumber = "4"; // April is Month number 4 in the calendar
    const dayToSelect = "15";
    const expectedDate = [monthNumber, dayToSelect, year];

    await page.goto("https://rahulshettyacademy.com/seleniumPractise/#/offers");
    // Click on the date picker button to open the date picker calendar
    await page.locator(".react-date-picker__inputGroup").click();

    // Click on the year i.e. April 2024
    await page.locator(".react-calendar__navigation__label").click();
    // Click on the year i.e. 2024
    await page.locator(".react-calendar__navigation__label").click();

    // Select the Year given by user
    await page.getByText(year).click();

    // Click on the month on the calender
    await page
      .locator(".react-calendar__year-view__months__month")
      .nth(Number(monthNumber - 1))
      .click();

    // Click on the dat on the calender
    await page.locator("//abbr[text()='" + dayToSelect + "']").click();

    //
    const inputs = await page.locator(".react-date-picker__inputGroup input"); // value = "2024-04-14"
    for (let index = 0; index < inputs.length; index++) {
      const value = inputs[index].getAttribute("value");
      expect(value).toEqual(expectedDate[index]);
    }
  });
});
```

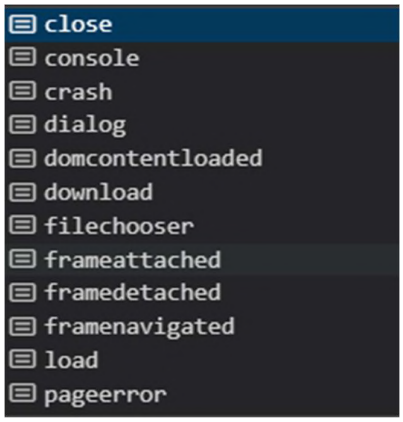
■ Handling Web dialogs, Frames & Event listeners with Playwright :

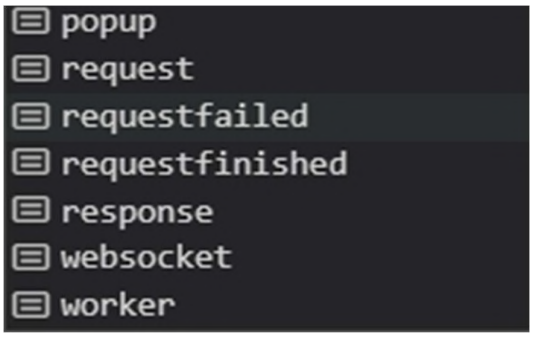
- https://playwright.dev/docs/dialogs

- https://playwright.dev/docs/api/class-framelocator

- https://playwright.dev/docs/events

- On Method Events List :





07-Handling-WebDialogs-Frames-and-Event-listeners.spec.js U

e2e > 07-Handling-WebDialogs-Frames-and-Event-listeners.spec.js > ...

Click here to ask Blackbox to help you code faster

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

test.describe("Playwright Handling & Validating Web dialogs, Frames & Event listeners", () => {
  test("@Web Popup validations", async ({ page }) => {
    await page.goto("https://rahulshettyacademy.com/AutomationPractice/");

    // Came back on AutomationPractice site
    await page.goto("http://google.com");
    await page.goBack();
    // await page.goForward();
    await page.reload();

    await expect(page.locator("#displayed-text")).toBeVisible(); // Check Hide/Show input is visible
    await page.locator("#hide-textbox").click(); // Click on Hide Button
    await expect(page.locator("#displayed-text")).toBeHidden(); // verify the element is hidden now
    // await page.pause();

    // Handle Confirm Popup dialog box
    /**
     * on method helps us to listen for events so it will emit when the event occurs in the webpage.
     * You can also able to give dismiss() instead of accept()
     *
     * Below line of code sets up an event listener for any dialog boxes that appear during the test execution. When a dialog box appears,
     * the event listener triggers the accept() method on the dialog, which simulates clicking the "OK" button to confirm the dialog.
     */
    page.on("dialog", (dialog) => dialog.accept()); // Hello , Are you sure you want to confirm? --> OK
    await page.locator("#confirmbtn").click();

    // Mouse Hover :
    await page.locator("#mousehover").hover();

    // Handle & automate frames
    const framesPage = page.frameLocator("#courses-iframe");
    await framesPage.locator(".new-navbar-highlighter").first().click(); // Click on All Access Plan link inside frame
    const textCheck = await framesPage.locator(".text h2").textContent(); // Join 13,522 Happy Subscibers!
    console.log(textCheck.split(" ")[1]); // 13,522
  });
```

■ API Testing with Playwright :

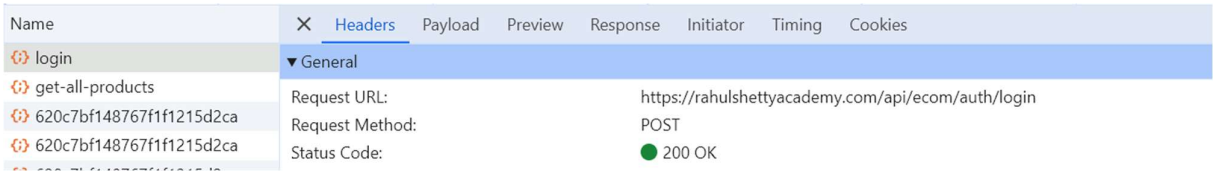
- https://playwright.dev/docs/mock-browser-apis

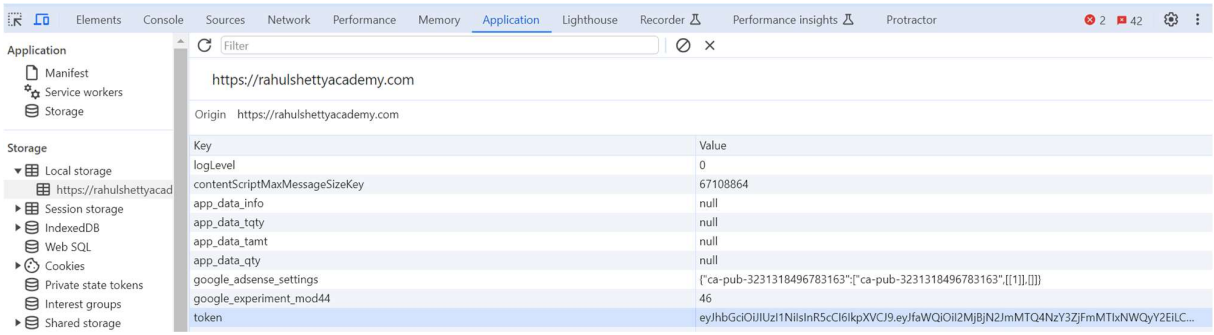
- https://playwright.dev/docs/mock#mock-api-requests

- https://playwright.dev/docs/network

- https://playwright.dev/docs/api/class-playwright

→ Once we login we are getting below API call :





- Login the Application Using API :

```
08-Web-API-Part1.spec.js U
e2e > 08-Web-API-Part1.spec.js > ...
Click here to ask Blackbox to help you code faster
const { test, expect, request } = require("@playwright/test");
const loginPayLoad = {
  userEmail: "anshika@gmail.com",
  userPassword: "Iamking@000",
};

let token;

test.beforeAll(async () => {
  const apiContext = await request.newContext();
  const loginResponse = await apiContext.post(
    "https://rahulshettyacademy.com/api/ecom/auth/login",
    {
      data: loginPayLoad,
    }
  ); //200,201,
  expect(loginResponse.ok()).toBeTruthy();
  const loginResponseJson = await loginResponse.json();
  token = loginResponseJson.token;
  console.log("Token : ", token);
});

test("@API Place the order", async ({ page }) => {
  page.addInitScript((value) => {
    window.localStorage.setItem("token", value);
  }, token);
```

- Create the Order Using API :

```
08-Web-API-Part1.spec.js U
e2e > 08-Web-API-Part1.spec.js > test("@API Place the order") callback
Click here to ask Blackbox to help you code faster
const { test, expect, request } = require("@playwright/test");
const loginPayLoad = {
  userEmail: "anshika@gmail.com",
  userPassword: "Iamking@000",
};
const orderPayLoad = {
  orders: [{ country: "India", productOrderedId: "6581ca399fd99c85e8ee7f45" }],
};
let token;
let orderId;

test.beforeAll(async () => {
  const apiContext = await request.newContext();
  // Login
  const loginResponse = await apiContext.post(
    "https://rahulshettyacademy.com/api/ecom/auth/login",
    {
      data: loginPayLoad,
    }
  ); //200,201,
  expect(loginResponse.ok()).toBeTruthy();
  const loginResponseJson = await loginResponse.json();
  token = loginResponseJson.token;
  console.log("Token : ", token);
```

```
  // Create Order
  const orderResponse = await apiContext.post(
    "https://rahulshettyacademy.com/api/ecom/order/create-order",
    {
      data: orderPayLoad,
      headers: {
        Authorization: token,
        "Content-Type": "application/json",
      },
    }
  );
  const orderResponseJson = await orderResponse.json();
  console.log(orderResponseJson);
  orderId = orderResponseJson.orders[0];
  console.log(orderResponseJson.message); // Order Placed Successfully
});

test("@API Place the order", async ({ page }) => {
  page.addInitScript((value) => {
    window.localStorage.setItem("token", value);
  }, token);
```

- Application Using API Utils :

08-Web-API-Part1.spec.js U

e2e > 08-Web-API-Part1.spec.js > test("@API Place the order") callback

Click here to ask Blackbox to help you code faster

```
const { test, expect, request } = require("@playwright/test");
const { APiUtils } = require("../utils/APiUtils");

// Define the login payload with user credentials
const loginPayLoad = {
  userEmail: "anshika@gmail.com",
  userPassword: "Iamking@000",
};

// Define the order payload with the country and product ID
const orderPayLoad = {
  orders: [{ country: "India", productOrderedId: "6581ca399fd99c85e8ee7f45" }],
};

let response;

// Run before all tests to authenticate and create an order
test.beforeAll(async () => {
  // Create a new context for API requests
  const apiContext = await request.newContext();
  // Initialize API utilities with the context and login payload
  const apiUtils = new APiUtils(apiContext, loginPayLoad);
  // Send API request to create an order and store the response
  response = await apiUtils.createOrder(orderPayLoad);
  console.log(response);
});

// Test case to verify successful order placement via API
test("@API Place the order", async ({ page }) => {
  // Add the authentication token to local storage before navigating
  page.addInitScript((value) => {
    window.localStorage.setItem("token", value);
  }, response.token);

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

  // Wait for navigation and pause for inspection - make manual login
  await page.pause();

  // Click on the "My Orders" button
  await page.locator("button[routerlink*='myorders']").click();

  // Wait for the table body to load
  await page.locator("tbody").waitFor();

  // Retrieve all rows in the orders table
  const rows = page.locator("tbody tr");
```

```
49
50    // Iterate through each row to find the order ID
51    for (let i = 0; i < (await rows.count()); ++i) {
52      const rowOrderId = await rows.nth(i).locator("th").textContent();
53      // If the order ID matches the created order, click on it
54      if (response.orderId.includes(rowOrderId)) {
55        await rows.nth(i).locator("button").first().click();
56        break;
57      }
58    }
59
60    // Get the order details displayed on the page
61    const orderIdDetails = await page.locator(".col-text").textContent();
62
63    // Verify if the created order ID is displayed in the details
64    expect(response.orderId.includes(orderIdDetails)).toBeTruthy();
65  });
66
67  //Verify if order created is showing in history page
68  // Precondition - create order -
69
```

APiUtils.js U

utils > APiUtils.js > ...

Click here to ask Blackbox to help you code faster

```
1   class APiUtils {
2     constructor(apiContext, loginPayLoad) {
3       this.apiContext = apiContext;
4       this.loginPayLoad = loginPayLoad;
5     }
6
7     async getToken() {
8       const loginResponse = await this.apiContext.post(
9         "https://rahulshettyacademy.com/api/ecom/auth/login",
10        {
11          data: this.loginPayLoad,
12        }
13      ); //200,201,
14      const loginResponseJson = await loginResponse.json();
15      const token = loginResponseJson.token;
16      console.log(token);
17      return token;
18    }
19
20    async createOrder(orderPayLoad) {
21      let response = {};
22      response.token = await this.getToken();
23      const orderResponse = await this.apiContext.post(
24        "https://rahulshettyacademy.com/api/ecom/order/create-order",
25        {
26          data: orderPayLoad,
27          headers: {
28            Authorization: response.token,
29            "Content-Type": "application/json",
30          },
31        }
32      );
33      const orderResponseJson = await orderResponse.json();
34      console.log(orderResponseJson);
35      const orderId = orderResponseJson.orders[0];
36      response.orderId = orderId;
37
38      return response;
39    }
40  }
41  module.exports = { APiUtils };
42
```

Session storage & Intercepting Network request/responses with Playwright :

09-Web-API-Part2.spec.js U

e2e > 09-Web-API-Part2.spec.js > ...

Click here to ask Blackbox to help you code faster

```
//Login UI  -> .json

//test browser-> .json , cart-,order, orderdetails,orderhistory
const { test, expect } = require("@playwright/test");
let webContext;

// Initialize a new browser context and page before running tests
test.beforeAll(async ({ browser }) => {
  // Create a new context and page for web interactions
  const context = await browser.newContext();
  const page = await context.newPage();

  // Navigate to the client page and login
  await page.goto("https://rahulshettyacademy.com/client");
  await page.locator("#userEmail").fill("rahulshetty@gmail.com");
  await page.locator("#userPassword").type("Iamking@000");
  await page.locator("[value='Login']").click();

  // Wait for the page to finish loading
  await page.waitForLoadState("networkidle");

  // Save the browser storage state to a file
  await context.storageState({ path: "state.json" });

  // Create a new browser context with the saved storage state for API tests
  webContext = await browser.newContext({ storageState: "state.json" });
});

// Test case for API interactions
test("@API Test case 2", async () => {
  // Create a new page in the API test context
  const page = await webContext.newPage();

  // Navigate to the client page in the API test context
  await page.goto("https://rahulshettyacademy.com/client");

  // Wait for the page to finish loading
  await page.waitForLoadState("networkidle");

  // Retrieve and log the titles of card body elements
  const titles = await page.locator(".card-body b").allTextContents();
  console.log(titles);
});
```

state.json U

state.json > ...

Click here to ask Blackbox to help you code faster


```
{
  "cookies": [
    {
      "name": "FTNT-EP-FG180FTK20900275",
      "value": "tKWlpbUwiO-npaWlgaWlpePx6_GI4PWI4-KUnZXj8e6X1ZyV1ZeSkIjQ1JeRkZCV1JeTkQ==",
      "domain": ".i.ibb.co",
      "path": "/",
      "expires": -1,
      "httpOnly": true,
      "secure": true,
      "sameSite": "None"
    },
    {
      "name": "FTNT-EP-FG180FTK20900275",
      "value": "tKWlpbhSie-kpaWloqWlpdzHytfKycC1",
      "domain": ".corp.capgemini.com",
      "path": "/",
      "expires": -1,
      "httpOnly": true,
      "secure": true,
      "sameSite": "None"
    }
  ],
  "origins": [
    {
      "origin": "https://rahulshettyacademy.com",
      "localStorage": [
        {
          "name": "token",
          "value": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2Mjc0MjU0OWUyNmI3ZTFhMTBlOWZjZTAiLCJ1c2VyRW1haWwiOiJyYWh1bHNoZXR0eUBnbWFpbC5jb20iLCJ1c2VyTW9iaW
        }
      ]
    }
  ]
```

- How to Debug API Steps in script using Visual Code debugging :

- Put a Breakpoint on where we want to debug on vs code line , then Press Control + Shift + P → Select Debug: Debug npm Script & debug step by step.

Before making sure to add below things inside package.json file :

```
package.json U
package.json > ...
  "scripts": {
    "test": "npx playwright test ./e2e/09-Web-API-Part2.spec.js --headed"
  },
```

When you are debugging make sure to increase timeout inside the config.js file for better debugging.

Detailed view of Trace viewer to understand the API logging req/responses :

Give trace : 'on'; inside playwright.config.js & check zip once run

playwright route method and its parameters in intercepting :

```
10-Network-Test-Intercept.spec.js U
e2e > 10-Network-Test-Intercept.spec.js > ...
const loginPayLoad = {
  userEmail: "anshika@gmail.com",
  userPassword: "Iamking@000",
};
const orderPayLoad = {
  orders: [{ country: "India", productOrderedId: "6581ca399fd99c85e8ee7f45" }],
};
const fakePayLoadOrders = { data: [], message: "No Orders" };

let response;
test.beforeAll(async () => {
  const apiContext = await request.newContext();
  // Login
  const apiUtils = new APiUtils(apiContext, loginPayLoad);
  // Create Order
  response = await apiUtils.createOrder(orderPayLoad);
});

//create order is success
test("@SP Place the order", async ({ page }) => {
  page.addInitScript((value) => {
    window.localStorage.setItem("token", value);
  }, response.token);
  await page.goto("https://rahulshettyacademy.com/client");

  // Temp Solution : Login Manually //
  await page.pause();

  // Intercepting network requests using Playwright's page.route() method
  await page.route(
    "https://rahulshettyacademy.com/api/ecom/order/get-orders-for-customer/*",
    async (route) => {
      // Fetching the original request
      const response = await page.request.fetch(route.request());
      // Creating a fake payload for orders
      let body = JSON.stringify(fakePayLoadOrders);
      // Fulfilling the intercepted request with the fake response
      route.fulfill({
        response,
        body,
      });
      // Interception process: API response -> Playwright fake response -> browser -> rendering data on front end
    }
  );

  // Clicking on a button with a specific router link
  await page.locator("button[routerlink*='myorders']").click();

  // Waiting for the response from the intercepted API request
  await page.waitForResponse(
    "https://rahulshettyacademy.com/api/ecom/order/get-orders-for-customer/*"
  );

  console.log(await page.locator(".mt-4").textContent()); // You have No Orders to show at this time. Please Visit Back Us
});
```

- How to intercept Network request calls with Playwright :

Whenever you are clicking on view on orders so that specific orderId is concatenated with that API call & make a GET API call in the backend where you will be able to see all the details.

- https://playwright.dev/docs/api/class-route

```
11-Network-Test-Security.spec.js U
e2e > 11-Network-Test-Security.spec.js > ...
Click here to ask Blackbox to help you code faster
const { test, expect } = require("@playwright/test");

test("@QW Security test request intercept", async ({ page }) => {
  //login and reach orders page
  await page.goto("https://rahulshettyacademy.com/client");
  await page.locator("#userEmail").fill("anshika@gmail.com");
  await page.locator("#userPassword").type("Iamking@000");
  await page.locator("[value='Login']").click();
  await page.waitForLoadState("networkidle");
  await page.locator(".card-body b").first().waitFor();

  await page.locator("button[routerlink*='myorders']").click();
  // Intercepting network requests using Playwright's page.route() method
  await page.route(
    // Intercepting requests to get order details API endpoint
    "https://rahulshettyacademy.com/api/ecom/order/get-orders-details?id=*",
    // Callback function to handle intercepted requests
    (route) =>
      // Continuing the intercepted request with a modified URL
      route.continue({
        url: "https://rahulshettyacademy.com/api/ecom/order/get-orders-details?id=621661f884b053f6765465b6",
      })
  );

  // Clicking on the first button element that contains the text 'View'
  await page.locator("button:has-text('View')").first().click();

  // Waiting for the last 'p' element and expecting it to have specific text
  await expect(page.locator("p").last()).toHaveText(
    "You are not authorize to view this order"
  );
});
```

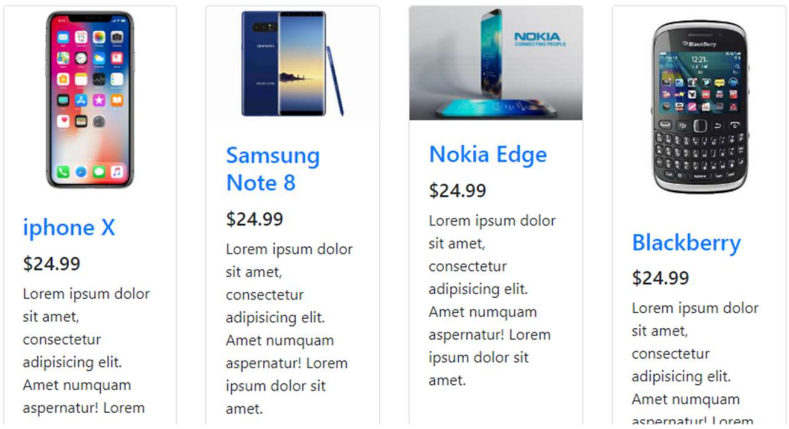
- How to abort Network calls with Playwright :

- Server is Down means backend servers are not responding & not giving any response back. So using playwright we can down server & abort or block the calls.

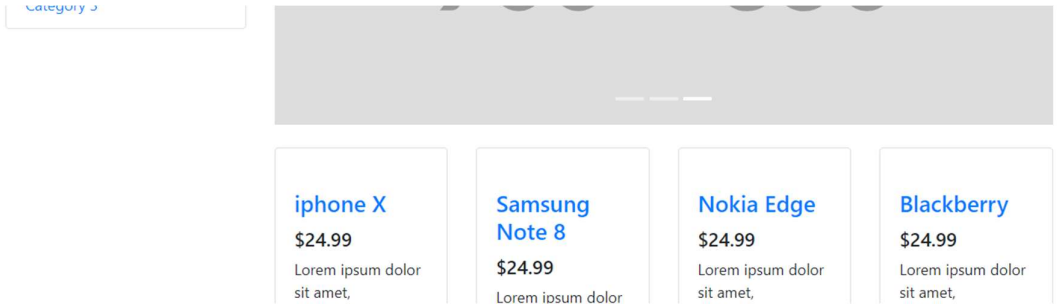
- So when you are refresh there is lots of calls happening here, but I don't want get the particular api call so we can block that call using playwright. It will Not impact because we have block images only from call it will not impact. It will only see UI without images.

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

- Before Blocking :



- After Blocking the Call :



- Playwright tracks every request, so we can try to print all api calls request URL & status.

→ These Below lines of code set up event listeners to capture and log information about each request and response made by the page. The 'request' event logs the URL of each request, while the 'response' event logs both the URL and status code of each response.

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

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

03-UI-basics-test.spec.js U

e2e > 03-UI-basics-test.spec.js > ...

```
111  test("@Web Browser Context - Validating Login Functionality", async ({
112    browser,
113  }) => {
114    const context = await browser.newContext();
115    const page = await context.newPage();
116    const userName = page.locator("#username");
117    const signIn = page.locator("#signInBtn");
118
119    /** This line of code intercepts network requests for image files with extensions .jpg, .png, and .jpeg using Playwright's page.route() method
120     * and aborts these requests. This can be useful for excluding image requests from affecting the test execution or for testing scenarios where
121     * image loading needs to be prevented.
122     * */
123    page.route("**/*.{jpg,png,jpeg}", (route) => route.abort());
124
125    /**
126     * To Print all the API calls URL & status
127     */
128    // Listening for all requests made by the page using the 'request' event
129    page.on("request", (request) => {
130      // Logging the URL of each request to the console
131      console.log(request.url());
132    });
133
134    // Listening for all responses received by the page using the 'response' event
135    page.on("response", (response) => {
136      // Logging the URL and status code of each response to the console
137      console.log(response.url(), response.status());
138    });
139
140    await page.goto("https://rahulshettyacademy.com/loginpagePractise/");
141    await userName.fill("rahulshettyacademy");
142    await page.locator("[type='password']").fill("learning");
143    await signIn.click();
144    await page.pause();
145  });
146
```

■ Capture Screenshots with Playwright on page & partial Element level :

- https://playwright.dev/docs/screenshots#capture-into-buffer

- https://playwright.dev/docs/test-snapshots

```
07-Handling-WebDialogs-Frames-and-Event-listeners.spec.js U
e2e > 07-Handling-WebDialogs-Frames-and-Event-listeners.spec.js > ...
39  test("Screenshot & Visual comparision", async ({ page }) => {
40    await page.goto("https://rahulshettyacademy.com/AutomationPractice/");
41
42    // Verifying that Hide/Show Input Box is visible on the page
43    await expect(page.locator("#displayed-text")).toBeVisible();
44
45    // Taking a screenshot of the visible element Hide/Show Input Box & Stored in same directory
46    await page
47      .locator("#displayed-text")
48      .screenshot({ path: "partialScreenshot.png" });
49    // Clicking on the hide button to hide the Hide/Show Input Box
50    await page.locator("#hide-textbox").click();
51
52    // Taking a screenshot of the entire page after hiding the textbox and stored in same directory
53    await page.screenshot({ path: "screenshot.png" });
54    await expect(page.locator("#displayed-text")).toBeHidden();
55  });
```

Visual Testing using Playwright :

- https://playwright.dev/docs/api/class-snapshotassertions#snapshot-assertions-to-match-snapshot-2

Visual testing in Playwright involves capturing screenshots of web pages or specific elements and comparing them against baseline images to detect any visual differences.

Here's how visual testing works in Playwright:

1. **Capturing Screenshots**: Playwright allows you to capture screenshots of entire web pages or specific elements using the `screenshot()` method.
2. **Baseline Images**: The initial screenshots captured during test execution serve as baseline images. These baseline images represent the expected visual appearance of the web page or elements under test.
3. **Comparison**: During subsequent test runs, Playwright captures new screenshots of the same web pages or elements. These screenshots are compared against the baseline images using pixel-by-pixel comparison algorithms.
4. **Detecting Differences**: Playwright identifies any visual differences between the new screenshots and the baseline images. These differences may include changes in layout, styling, or content.
5. **Reporting**: Playwright generates detailed reports highlighting any visual differences detected during the test execution. These reports help testers identify and analyze potential issues affecting the visual appearance of their web applications.

```
07-Handling-WebDialogs-Frames-and-Event-listeners.spec.js U
e2e > 07-Handling-WebDialogs-Frames-and-Event-listeners.spec.js > ...
57    test("Visual Testing", async ({ page }) => {
58      //make payment when you 0 balance
59      await page.goto("https://rahulshettyacademy.com/loginpagePractise/");
60      expect(await page.screenshot()).toMatchSnapshot("landing.png");
61    });
62  });
```

■ how to manipulate edit and write excel files using pure Java Script : [No playwright comes into picture here]

- https://www.npmjs.com/package/exceljs

- install exceljs : npm i exceljs

- Traversing rows and columns of excel worksheet :

- The **`excelTest()`** function provides a mechanism to open an Excel file, access its contents, and log the values of each cell to the console. It can be used for various purposes such as data analysis, data validation, or data migration.

Excel-Demo.js U

edit-write-excel-using-JavaScript > Excel-Demo.js > ...

Click here to ask Blackbox to help you code faster

```
1  const ExcelJS = require("exceljs");
2
3  // Function to read data from an Excel file
4  async function excelTest() {
5    // Create a new Excel workbook object
6    const workbook = new ExcelJS.Workbook();
7
8    // Read the Excel file from disk
9    await workbook.xlsx.readFile("./download.xlsx");
10
11   // Get the first worksheet in the workbook
12   const worksheet = workbook.getWorksheet("Sheet1");
13
14   // Iterate over each row in the worksheet
15   worksheet.eachRow((row) => {
16     // Iterate over each cell in the row
17     row.eachCell((cell) => {
18       // Log the value of each cell to the console
19       console.log(cell.value);
20     });
21   });
22 }
23
24 // Call the excelTest function to read data from the Excel file
25 excelTest();
26
27 // To RUN : node .\edit-write-excel-using-JavaScript\Excel-Demo.js
```

- Build Util functions to read and update excel file strategically :

- Now I want to verify that if apple is present in excel sheet, then give me the coordinates i.e. row number & column number.

- Also we are able to change or update Apple to IPhone inside excel file.

Excel-Demo.js U

edit-write-excel-using-JavaScript > Excel-Demo.js > ...

Click here to ask Blackbox to help you code faster

```
1  const ExcelJS = require("exceljs");
2
3  // Function to read data from an Excel file and find the coordinates of a specific value
4  async function excelTest() {
5    // Create a new Excel workbook object
6    const workbook = new ExcelJS.Workbook();
7
8    // Read the Excel file from disk
9    await workbook.xlsx.readFile("./download.xlsx");
10
11   // Get the first worksheet in the workbook
12   const worksheet = workbook.getWorksheet("Sheet1");
```

```
    // Iterate over each row in the worksheet
    worksheet.eachRow((row, rowNumber) => {
      // Iterate over each cell in the row
      row.eachCell((cell, colNumber) => {
        // Check if the cell value is "Apple"
        if (cell.value === "Apple") {
          // Log the row number where "Apple" is found
          console.log("Row Number:", rowNumber); // 3
          // Log the column number where "Apple" is found
          console.log("Column Number:", colNumber); // 2
        }
      });
    });

    // Update data as Apple to IPhone
    const cell = worksheet.getCell(3, 2);
    cell.value = "IPhone";
    await workbook.xlsx.writeFile("./download.xlsx");
  }

  // Call the excelTest function to read data from the Excel file and find "Apple"
  excelTest();

  // To RUN : node .\edit-write-excel-using-JavaScript\Excel-Demo.js
```

- Without Hardcode values we can segregate the code :

```
Excel-Demo.js U
edit-write-excel-using-JavaScript > Excel-Demo.js > excelTest
Click here to ask Blackbox to help you code faster
const ExcelJS = require("exceljs");

// Function to read data from an Excel file and find the coordinates of a specific value
async function excelTest(searchText) {
  let output = { row: -1, column: -1 };
  // Create a new Excel workbook object
  const workbook = new ExcelJS.Workbook();

  // Read the Excel file from disk
  await workbook.xlsx.readFile("./download.xlsx");

  // Get the first worksheet in the workbook
  const worksheet = workbook.getWorksheet("Sheet1");

  // Iterate over each row in the worksheet
  worksheet.eachRow((row, rowNumber) => {
    // Iterate over each cell in the row
    row.eachCell((cell, colNumber) => {
      if (cell.value === searchText) {
        output.row = rowNumber;
        output.column = colNumber;
      }
    });
  });

  // Update data as Banana to Republic
  const cell = worksheet.getCell(output.row, output.column);
  cell.value = "Republic";
  await workbook.xlsx.writeFile("./download.xlsx");
}

// Call the excelTest function to read data from the Excel file and find "Apple"
excelTest("Banana");
```

- get and update the data from excel based on filter search criteria :

- Now we can change Price of Orange from 399 to 490.

- We have used change as object, output.column + change.colChange because we are on the same row & we want to go 2 col forward inside excel file.

Excel-Demo.js U

edit-write-excel-using-JavaScript > Excel-Demo.js > ...

Click here to ask Blackbox to help you code faster

```
const ExcelJS = require("exceljs");

// Function to read data from an Excel file and find the coordinates of a specific value
async function writeExcelTest(searchText, replaceText, change, filePath) {
  const workbook = new ExcelJS.Workbook();
  await workbook.xlsx.readFile(filePath);
  const worksheet = workbook.getWorksheet("Sheet1");
  const output = await readExcel(worksheet, searchText);

  // Used to access the cell in the updated column where the replaceText will be written.
  const cell = worksheet.getCell(output.row, output.column + change.colChange);
  cell.value = replaceText;
  await workbook.xlsx.writeFile(filePath);
}

async function readExcel(worksheet, searchText) {
  let output = { row: -1, column: -1 };
  worksheet.eachRow((row, rowNumber) => {
    row.eachCell((cell, colNumber) => {
      if (cell.value === searchText) {
        output.row = rowNumber;
        output.column = colNumber;
      }
    });
  });
  return output;
}

// //update Orange Price to 350.
writeExcelTest("Orange", 350, { rowChange: 0, colChange: 1 }, "./download.xlsx");
```

■ **how to inject playwright code and perform end to end automation for downloading excel file-> edit excel -> upload excel file back into Web portal using Playwright :**

12-Upload-Download.spec.js U

e2e > 12-Upload-Download.spec.js > ...

Click here to ask Blackbox to help you code faster

```
const ExcelJS = require("exceljs");
const { test, expect } = require("@playwright/test");

async function writeExcelTest(searchText, replaceText, change, filePath) {
  const workbook = new ExcelJS.Workbook();
  await workbook.xlsx.readFile(filePath);
  const worksheet = workbook.getWorksheet("Sheet1");
  const output = await readExcel(worksheet, searchText);

  const cell = worksheet.getCell(output.row, output.column + change.colChange);
  cell.value = replaceText;
  await workbook.xlsx.writeFile(filePath);
}

async function readExcel(worksheet, searchText) {
  let output = { row: -1, column: -1 };
  worksheet.eachRow((row, rowNumber) => {
    row.eachCell((cell, colNumber) => {
      if (cell.value === searchText) {
        output.row = rowNumber;
        output.column = colNumber;
      }
    });
  });
  return output;
}

test("Upload download excel validation", async ({ page }) => {
  const textSearch = "Papaya";
  const updateValue = "890";
  await page.goto(
    "https://rahulshettyacademy.com/upload-download-test/index.html"
  );

  // Create a promise to wait for the download event
  const downloadPromise = page.waitForEvent("download");

  // Click on Download button to initiate the download of an xlsx file
  await page.getByRole("button", { name: "Download" }).click();

  // Wait for the download to complete by resolving the download promise
  await downloadPromise;
```

12-Upload-Download.spec.js U

e2e > 12-Upload-Download.spec.js > ...

```
  //update Papaya Price to 890.
  writeExcelTest(
    textSearch,
    updateValue,
    { rowChange: 0, colChange: 2 },
    "./download.xlsx"
  );

  // Click on Upload choose file button & upload file
  await page.locator("#fileinput").click();
  await page.locator("#fileinput").setInputFiles("./download.xlsx");

  // Find the locator for the text to search for, in this case, 'Papaya'
  const textlocator = page.getByText(textSearch);

  // Find the desired row containing the searched text
  const desiredRow = await page.getByRole("row").filter({ has: textlocator });

  // Expect the cell with id "cell-4-undefined" in the desired row to contain the updateValue
  await expect(desiredRow.locator("#cell-4-undefined")).toContainText(
    updateValue
  );
});
```

- https://playwright.dev/docs/input#upload-files

■ Page Object Patterns & Data driven Parameterization for Playwright Tests :

- https://playwright.dev/docs/pom

13-ClientApp-Page-Object.spec.js U

e2e > 13-ClientApp-Page-Object.spec.js > ...

Click here to ask Blackbox to help you code faster

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

const { POManager } = require("../Page-Objects/POManager");
//Json->string->js object
const dataset = JSON.parse(
  JSON.stringify(require("../utils/placeorderTestData.json"))
);

for (const data of dataset) {
  test(`@Webs Client App login for ${data.productName}`, async ({ page }) => {
    const poManager = new POManager(page);
    //js file- Login js, DashboardPage
    const loginPage = poManager.getLoginPage();
    await loginPage.goTo();
    await loginPage.validLogin(data.username, data.password);
    const dashboardPage = poManager.getDashboardPage();
    await dashboardPage.searchProductAddCart(data.productName);
    await dashboardPage.navigateToCart();

    const cartPage = poManager.getCartPage();
    await cartPage.VerifyProductIsDisplayed(data.productName);
    await cartPage.Checkout();

    const ordersReviewPage = poManager.getOrdersReviewPage();
    await ordersReviewPage.searchCountryAndSelect("ind", "India");
    const orderId = await ordersReviewPage.SubmitAndGetOrderId();
    console.log(orderId);
    await dashboardPage.navigateToOrders();
    const ordersHistoryPage = poManager.getOrdersHistoryPage();
    await ordersHistoryPage.searchOrderAndSelect(orderId);
    expect(orderId.includes(await ordersHistoryPage.getOrderId())).toBeTruthy();
  });
}
```

13-ClientApp-Page-Object.spec.js U

e2e > 13-ClientApp-Page-Object.spec.js > ...

```
35
36  customtest(`Client App login`, async ({ page, testDataForOrder }) => {
37    const poManager = new POManager(page);
38    //js file- Login js, DashboardPage
39    const loginPage = poManager.getLoginPage();
40    await loginPage.goTo();
41    await loginPage.validLogin(
42      testDataForOrder.username,
43      testDataForOrder.password
44    );
45    const dashboardPage = poManager.getDashboardPage();
46    await dashboardPage.searchProductAddCart(testDataForOrder.productName);
47    await dashboardPage.navigateToCart();
48
49    const cartPage = poManager.getCartPage();
50    await cartPage.VerifyProductIsDisplayed(testDataForOrder.productName);
51    await cartPage.Checkout();
52  });
53
54  // test files will trigger parallel
55  // individual tests in the file will run in sequence
56
```

CartPage.js U

Page-Objects > CartPage.js > ...

Click here to ask Blackbox to help you code faster

```
1  const { expect } = require("@playwright/test");
2  class CartPage {
3    constructor(page) {
4      this.page = page;
5      this.cartProducts = page.locator("div li").first();
6      this.productsText = page.locator(".card-body b");
7      this.cart = page.locator("[routerlink*='cart']");
8      this.orders = page.locator("button[routerlink*='myorders']");
9      this.checkout = page.locator("text=Checkout");
10   }
11
12   async VerifyProductIsDisplayed(productName) {
13     await this.cartProducts.waitFor();
14     const bool = await this.getProductLocator(productName).isVisible();
15     expect(bool).toBeTruthy();
16   }
17
18   async Checkout() {
19     await this.checkout.click();
20   }
21
22   getProductLocator(productName) {
23     return this.page.locator("h3:has-text('" + productName + "')");
24   }
25 }
26 module.exports = { CartPage };
27
```

DashboardPage.js U

Page-Objects > DashboardPage.js > ...

Click here to ask Blackbox to help you code faster

```
class DashboardPage {
  constructor(page) {
    this.page = page;
    this.products = page.locator(".card-body");
    this.productsText = page.locator(".card-body b");
    this.cart = page.locator("[routerlink*='cart']");
    this.orders = page.locator("button[routerlink*='myorders']");
  }

  async searchProductAddCart(productName) {
    const titles = await this.productsText.allTextContents();
    console.log(titles);
    const count = await this.products.count();
    for (let i = 0; i < count; ++i) {
      if (
        (await this.products.nth(i).locator("b").textContent()) === productName
      ) {
        //add to cart
        await this.products.nth(i).locator("text= Add To Cart").click();
        break;
      }
    }
  }

  async navigateToOrders() {
    await this.orders.click();
  }

  async navigateToCart() {
    await this.cart.click();
  }
}
module.exports = { DashboardPage };
```

LoginPage.js U

Page-Objects > LoginPage.js > ...

Click here to ask Blackbox to help you code faster

```
class LoginPage {
  constructor(page) {
    this.page = page;
    this.signInbutton = page.locator("[value='Login']");
    this.userName = page.locator("#userEmail");
    this.password = page.locator("#userPassword");
  }

  async goTo() {
    await this.page.goto("https://rahulshettyacademy.com/client");
  }

  async validLogin(username, password) {
    await this.userName.type(username);
    await this.password.type(password);
    await this.signInbutton.click();
    await this.page.waitForLoadState("networkidle");
  }
}
module.exports = { LoginPage };

```

OrdersHistoryPage.js U

Page-Objects > OrdersHistoryPage.js > ...

Click here to ask Blackbox to help you code faster

```
class OrdersHistoryPage {
  constructor(page) {
    this.page = page;
    this.ordersTable = page.locator("tbody");
    this.rows = page.locator("tbody tr");
    this.orderdIdDetails = page.locator(".col-text");
  }
  async searchOrderAndSelect(orderId) {
    await this.ordersTable.waitFor();
    for (let i = 0; i < (await this.rows.count()); ++i) {
      const rowOrderId = await this.rows.nth(i).locator("th").textContent();
      if (orderId.includes(rowOrderId)) {
        await this.rows.nth(i).locator("button").first().click();
        break;
      }
    }
  }

  async getOrderId() {
    return await this.orderdIdDetails.textContent();
  }
}
module.exports = { OrdersHistoryPage };
```

OrdersReviewPage.js U

Page-Objects > OrdersReviewPage.js > ...

Click here to ask Blackbox to help you code faster

```
const { expect } = require("@playwright/test");

class OrdersReviewPage {
  constructor(page) {
    this.page = page;
    this.country = page.locator("[placeholder*='Country']");
    this.dropdown = page.locator(".ta-results");
    this.emailId = page.locator(".user__name [type='text']").first();
    this.submit = page.locator(".action__submit");
    this.orderConfirmationText = page.locator(".hero-primary");
    this.orderId = page.locator(".em-spacer-1 .ng-star-inserted");
  }
  async searchCountryAndSelect(countryCode, countryName) {
    await this.country.type(countryCode, { delay: 100 });
    await this.dropdown.waitFor();
    const optionsCount = await this.dropdown.locator("button").count();
    for (let i = 0; i < optionsCount; ++i) {
      const text = await this.dropdown.locator("button").nth(i).textContent();
      if (text.trim() === countryName) {
        await this.dropdown.locator("button").nth(i).click();
        break;
      }
    }
  }

  async VerifyEmailId(username) {
    await expect(this.emailId).toHaveText(username);
  }

  async SubmitAndGetOrderId() {
    await this.submit.click();
    await expect(this.orderConfirmationText).toHaveText(
      " Thankyou for the order. "
    );
    return await this.orderId.textContent();
  }
}
module.exports = { OrdersReviewPage };
```

POManager.js U

Page-Objects > POManager.js > ...

Click here to ask Blackbox to help you code faster

```
const { LoginPage } = require("./LoginPage");
const { DashboardPage } = require("./DashboardPage");
const { OrdersHistoryPage } = require("./OrdersHistoryPage");
const { OrdersReviewPage } = require("./OrdersReviewPage");
const { CartPage } = require("./CartPage");
class POManager {
  constructor(page) {
    this.page = page;
    this.loginPage = new LoginPage(this.page);
    this.dashboardPage = new DashboardPage(this.page);
    this.ordersHistoryPage = new OrdersHistoryPage(this.page);
    this.ordersReviewPage = new OrdersReviewPage(this.page);
    this.cartPage = new CartPage(this.page);
  }

  getLoginPage() {
    return this.loginPage;
  }

  getCartPage() {
    return this.cartPage;
  }

  getDashboardPage() {
    return this.dashboardPage;
  }
  getOrdersHistoryPage() {
    return this.ordersHistoryPage;
  }

  getOrdersReviewPage() {
    return this.ordersReviewPage;
  }
}
module.exports = { POManager };
```

placeorderTestData.json U

utils > placeorderTestData.json > ...

Click here to ask Blackbox to help you code faster

```
[
  {
    "username": "anshika@gmail.com",
    "password": "Iamking@000",
    "productName": "ZARA COAT 3"
  }
]
```

test-base.js U

utils > test-base.js > ...

Click here to ask Blackbox to help you code faster

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

exports.customtest = base.test.extend({
  testDataForOrder: {
    username: "anshika@gmail.com",
    password: "Iamking@000",
    productName: "ADIDAS ORIGINAL",
  },
});
```

Project Configurations, & Config options for robust Framework design :

- https://playwright.dev/docs/test-configuration#basic-configuration

- https://playwright.dev/docs/release-notes#breaking-change-custom-config-options

playwright.config.js U

playwright.config.js > <unknown> > projects

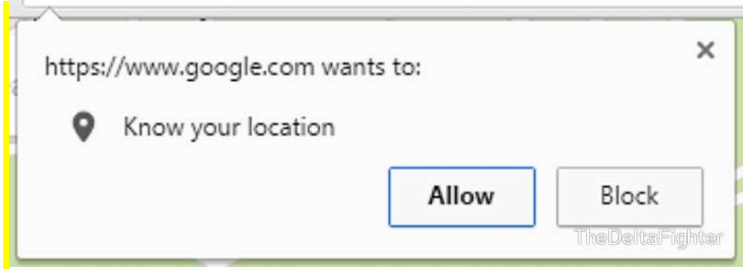
```
/* Configure projects for major browsers */
projects: [
  {
    name: "chromium",
    use: { ...devices["Desktop Chrome"], headless: false },
  },

  {
    name: "firefox",
    use: { ...devices["Desktop Firefox"] },
  },

  {
    name: "webkit",
    use: { ...devices["Desktop Safari"] },
  },

  /* Test against mobile viewports. */
  {
    name: 'Mobile Chrome',
    use: { ...devices['Pixel 5'] },
  },
  {
    name: 'Mobile Safari',
    use: { ...devices['iPhone 12'] },
  },

  /* Test against branded browsers. */
  {
    name: 'Microsoft Edge',
    use: { ...devices['Desktop Edge'], channel: 'msedge' },
  },
  {
    name: 'Google Chrome',
    use: { ...devices['Desktop Chrome'], channel: 'chrome' },
  },
],
```

- SSL Certification Error :


```
{
  name: "chromium",
  use: {
    ...devices["Desktop Chrome"],
    headless: false,

    // To handle SSL Certificate error Automatically
    ignoreHTTPSErrors: true,
```

- go location pop up:

- https://playwright.dev/docs/emulation#geolocation




```
projects: [
  {
    name: "chromium",
    use: {
      ...devices["Desktop Chrome"],
      headless: false,

      // want to handle go-location
      permissions: ["geolocation"],
```

- Record the video :

- https://playwright.dev/docs/videos#record-video

■ Test Retries , Serial & Parallel execution & Tagging Tests in Playwright :

- https://playwright.dev/docs/test-retries#retries

- https://playwright.dev/docs/api/class-test#test-describe-configure

- https://playwright.dev/docs/test-configuration

- If you give retries: 2 then once your test fail it will run 2 times again, means total 3 time.

```
module.exports = defineConfig({
  testDir: "./e2e",
  /* Retries the testcase 1 times once failed */
  retries: 2,
```

- Understand how playwright run tests in serial & parallel mode and update setting :

- https://playwright.dev/docs/test-retries#serial-mode

- https://playwright.dev/docs/test-parallel#serial-mode

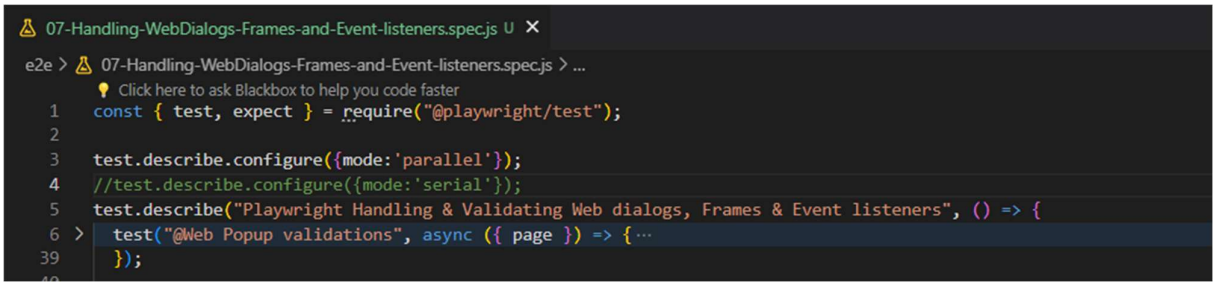
- https://playwright.dev/docs/test-parallel

- By Default If we run a single spec file, our testcases will execute one by one means serial mode. But If we run total e2e OR tests folder then test files will trigger run at parallel but Individual tests present in that file that is run in serial.

```
54   // test files will trigger parallel
55   // individual tests in the file will run in sequence
56
```

- If I give simple "npx playwright test" then it will run entire tests inside all folder structure. , make sure you have not use test.only anywhere.

- Also we can be able to run tests inside file in serial or parallel mode as below : If you give serial mode one by one testcase will be execute.



- But In serial for example 1st test passed & then 2nd failed then 3rd will be automatically skipped by the playwright.

- Race Condition :

- https://playwright.dev/docs/writing-tests#introduction

- For example you are working on same site & running parallel & in between one test adding items in cart & one removing so that time race condition will be overcome, so then :

To overcome race conditions when running tests in parallel, especially when multiple tests interact with the same resources (like adding and removing items from a cart), you can employ various strategies:

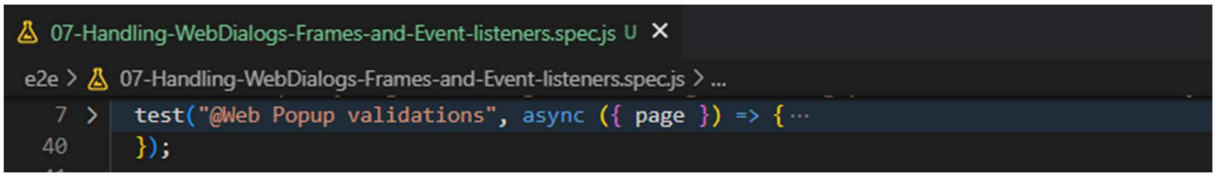
1. **Test Data Isolation:** Ensure that each test case uses its own set of test data. For example, if one test adds items to the cart, the other test should use a separate cart or clear the cart before running. This prevents interference between tests.
2. **Synchronization:** Implement synchronization mechanisms such as waits or assertions to ensure that each test step is completed before moving on to the next step. For example, you can use `page.waitFor` to wait for an element to appear before interacting with it.
3. **Test Ordering:** Organize your tests in such a way that they don't interfere with each other. If one test adds items to the cart, make sure that the test that removes items from the cart runs after the first one.
4. **Session Isolation:** Consider running each test case in its own browser session or context to prevent interference between tests. Playwright provides features for creating isolated browser contexts (`browser.newContext`) that you can use for this purpose.
5. **Resource Cleanup:** Ensure that resources are cleaned up properly after each test. For example, remove any items added to the cart at the end of the test so that the cart is in a consistent state for the next test.
6. **Retry Mechanism:** Implement retry mechanisms for tests that are known to be flaky due to race conditions. This allows the test to be retried multiple times until it passes consistently.

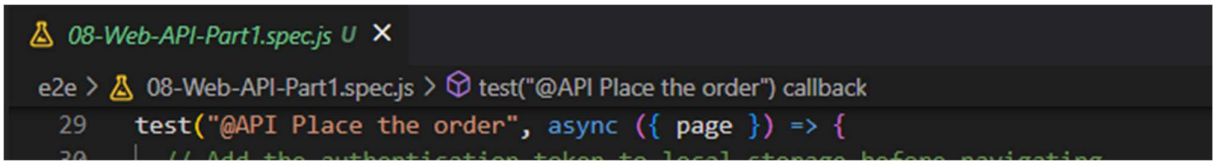
By following these strategies, you can mitigate race conditions and ensure that your tests run reliably in parallel without interfering with each other.

- How to tag tests and control the execution from the command line parameters :

- https://playwright.dev/docs/test-annotations#tag-tests

- Here's how you can tag tests and control their execution :





→ To Run : npx playwright test --grep "@API"

→ npx playwright test --grep "@Web"

1. **Tagging Tests:**
   You can tag tests using annotations in your test files. For example, you can add annotations like `@test` or `@focus` to tag specific tests.

```javascript
test('Test Case 1', async ({ page }) => {
    // Test logic...
});

test('Test Case 2', async ({ page }) => {
    // Test logic...
}).only(); // Use .only() to focus on this test

test('Test Case 3', async ({ page }) => {
    // Test logic...
}).fail(); // Use .fail() to mark this test as expected to fail
```

2. **Controlling Execution:**
   You can control the execution of tests based on these tags using command-line parameters. For example:
   - To run only tests with the `@focus` tag, use the `--focus` flag:

```bash
npx playwright test --focus
```

   - To run only tests without the `@focus` tag, use the `--grep` flag with a negative pattern:

```less
npx playwright test --grep "^(?!.*@focus).*"
```

   - To run only tests marked as expected to fail, use the `--expected-flaky` flag:

```bash
npx playwright test --expected-flaky
```

   - To run only tests with specific tags, use the `--grep` flag with a regular expression:

```perl
npx playwright test --grep "@tagName"
```

3. **Combining Tags:**
   You can combine tags to further refine the selection of tests. For example:
   - To run tests with both `@tag1` and `@tag2`, use:

```perl
npx playwright test --grep "@tag1.*@tag2"
```

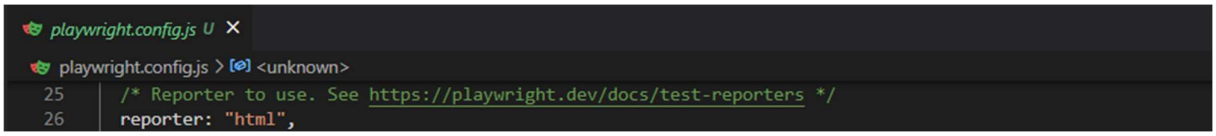
   - To run tests with either `@tag1` or `@tag2`, use:

```perl
npx playwright test --grep "@tag1|@tag2"
```

■ HTML & Allure Reporting & CI/CD Jenkins Integration :

- https://playwright.dev/docs/trace-viewer-intro#opening-the-html-report
- https://playwright.dev/docs/intro#html-test-reports
- https://playwright.dev/docs/ci-intro#html-report
- https://playwright.dev/docs/ci-intro#downloading-the-html-report
- https://playwright.dev/docs/test-reporters#html-reporter

- https://playwright.dev/docs/test-reporters#third-party-reporter-showcase
- https://playwright.dev/docs/release-notes#playwright-test
- https://playwright.dev/docs/best-practices#run-tests-on-ci
- When we give reporter as html inside playwright.config.js file then report will generate automatically.



- Allure Report :

- https://www.npmjs.com/package/allure-playwright

Installation

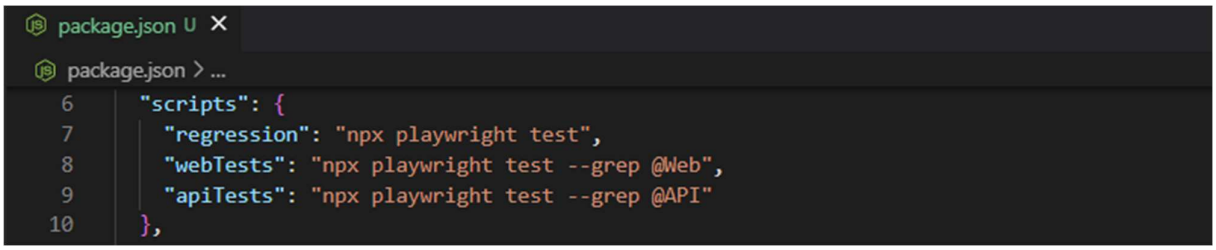
```
npm i -D @playwright/test allure-playwright
```

or via yarn:

```
yarn add @playwright/test allure-playwright --dev
```

- npx playwright test --grep "@Web" --reporter=line,allure-playwright
- allure generate ./allure-results --clean
- allure open ./allure-report

- Create custom scripts to trigger the tests from package.json file :



- Integrate the Playwright framework with Jenkins :

- Download Generic Java Package(.war) from https://www.jenkins.io/download/

- Now open command prompt & navigate to the path where your Jenkins.jar file located.

- Type Command : java -jar Jenkins.war -httpPort=9090 → To start the Jenkinson localhost 9090

- Open localhost 9090 on browser & give username as admin & password & login.

- Paste the password you copied from the initialAdminPassword file into the text box and click "Continue".

- The initial password is stored in a file named initialAdminPassword, which is typically located in the Jenkins installation directory. You'll need to open this file and copy the password.

- Now Create new Job : New Items → Enter Name → Freestyle Project → Ok

- If you have git repo ten use that otherwise if local then choose option as : Use custom workspace → /Users/YBOROLE/Playwright_Testing/Playwright-Udemy

- Below Add Build Step → For window choose Execute window bash command & for Mac choose Execute shell → npm run regression (OR Any other if other) → Save

- Once you are able to see Build Now → you are able to see everything like logs passed/failed etc.

- If you want to parameterized the build then : go to Configure → Click on This project is Parameterized → Choose Choice Parameter → Give Name as a Script & Choices as regression or .. OR regression \n webTests \n apiTests → Inside window bash command → add only npm run "%Script%" → Save

- Now you are able to see Build with Parameters → Choose options → Build

■ Playwright Cucumber Framework Integration :

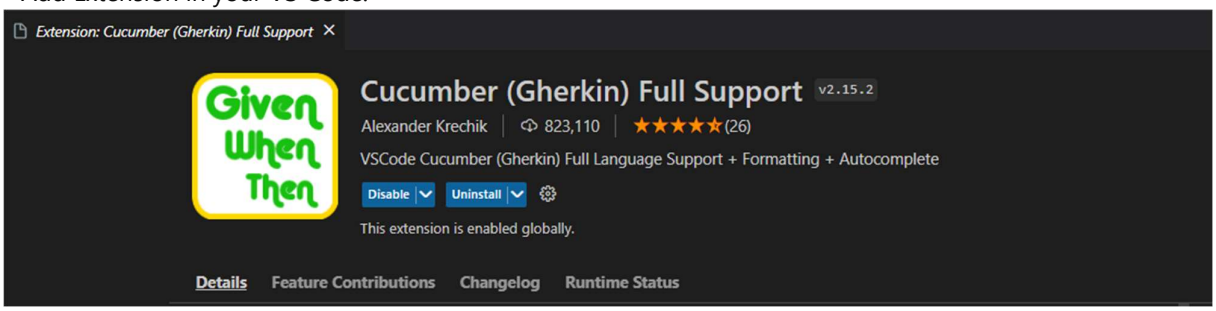
- https://cucumber.io/docs/installation/javascript/

- https://github.com/cucumber/cucumber-js

- https://vitalets.github.io/playwright-bdd/#/

- Install Cucumber : npm install @cucumber/cucumber

- Add Extension in your VS Code:



- .Feature File :

- Feature → It's like a test suite like a folder which will hold the multiple test cases.

- Scenario Outline → Each scenario outline means one test in cucumber.

- Given, when, Then → It's a test steps.

- https://github.com/cucumber/cucumber-js/blob/main/docs/support_files/world.md

- https://github.com/cucumber/cucumber-js/blob/main/docs/support_files/step_definitions.md

- https://github.com/cucumber/cucumber-js/blob/main/docs/support_files/timeouts.md

- Note : Read about World Constructor in cucumber

- Code :

Ecommerce.feature U

features > Ecommerce.feature

Click here to ask Blackbox to help you code faster

```
Feature: Ecommerce validations

    Scenario Outline: Placing the Order
        Given a login to Ecommerce application with "<username>" and "<password>"
        When Add "ZARA COAT 3" to Cart
        Then Verify "ZARA COAT 3" is displayed in the Cart
        When Enter valid details and Place the Order
        Then Verify order is present in the OrderHistory

        Examples:
            | username          | password    |
            | anshika@gmail.com | Iamking@000 |
```

steps.js U

features > step_definitions > steps.js > ...

Click here to ask Blackbox to help you code faster

```
const { When, Given, Then } = require("@cucumber/cucumber");
const { POManager } = require("../../Page-Objects/POManager");
const { expect } = require("@playwright/test");
const playwright = require("@playwright/test");

Given(
  "a login to Ecommerce application with {string} and {string}",
  { timeout: 100 * 1000 },
  async function (username, password) {
    const browser = await playwright.chromium.launch();
    const context = await browser.newContext();
    const page = await context.newPage();
    this.poManager = new POManager(page);
    //js file- Login js, DashboardPage
    const loginPage = this.poManager.getLoginPage();
    await loginPage.goTo();
    await loginPage.validLogin(username, password);
  }
);

When("Add {string} to Cart", async function (productName) {
  // Write code here that turns the phrase above into concrete actions
  this.dashboardPage = this.poManager.getDashboardPage();
  await this.dashboardPage.searchProductAddCart(productName);
  await this.dashboardPage.navigateToCart();
});

Then("Verify {string} is displayed in the Cart", async function (productName) {
  // Write code here that turns the phrase above into concrete actions
  this.cartPage = this.poManager.getCartPage();
  await this.cartPage.VerifyProductIsDisplayed(productName);
  await this.cartPage.Checkout();
});

When("Enter valid details and Place the Order", async function () {
  // Write code here that turns the phrase above into concrete actions
  const ordersReviewPage = this.poManager.getOrdersReviewPage();
  await ordersReviewPage.searchCountryAndSelect("ind", "India");
  this.orderId = await ordersReviewPage.SubmitAndGetOrderId();
  console.log(this.orderId);
});

Then("Verify order is present in the OrderHistory", async function () {
  // Write code here that turns the phrase above into concrete actions
  await this.dashboardPage.navigateToOrders();
  const ordersHistoryPage = this.poManager.getOrdersHistoryPage();
  await ordersHistoryPage.searchOrderAndSelect(this.orderId);
  expect(
    this.orderId.includes(await ordersHistoryPage.getOrderId())
  ).toBeTruthy();
});
```

- To Run type : npx cucumber-js

- After Run you are able to see execution done but not able to see terminal, terminal still hanging there so to resolve that :

Run : npx cucumber-js –exit

- OR Directly Run : npx cucumber-js --exit

- https://github.com/cucumber/cucumber-js/blob/main/docs/deprecations.md

→ To generate the step_defination file skeleton write only feature file & Run you are abler to see skeleton is generated in terminal by cucumber.

→ **World Constructor :**

World, is an isolated scope for each scenario, exposed to the steps and most hooks as `this`. It allows you to set variables in one step and recall them in a later step. All variables set this way are discarded when the scenario concludes. It is managed by a world class, either the default one or one you create. Each scenario is given an new instance of the class when the test starts, even if it is a retry run.

The world is not available to the hooks `BeforeAll` or `AfterAll` as each of these executes outside any particular scenario.

Here's some simple usage of the world to retain state between steps:

```
const { Given, Then } = require('@cucumber/cucumber')

Given("my color is {string}", function(color) {
  this.color = color
})

Then("my color should not be red", function() {
  if (this.color === "red") {
    throw new Error("Wrong Color");
  }
});
```

Important Note: The following will NOT work as arrow functions do not have their own bindings to `this` and are not suitable for the apply method Cucumber uses internally to call your step definitions and hooks.

```
// This WON'T work!!
Then("my color should not be blue", () => {
  if (this.color === "red") {
    throw new Error("Wrong Color");
  }
});
```

- Hooks :

- https://github.com/cucumber/cucumber-js/blob/main/docs/support_files/hooks.md

- https://playwright.dev/docs/api/class-test

1. Whenever we are writing hooks in test file, we should not pass page fixture in tests because we already created page instance.

2. Whenever we are writing hooks in test file, we want to make sure in config file this parameter should be : fullyParallel: false

- Implement Cucumber Tags for features :

- We are giving tags inside feature file start using @ symbol. E.g. @validation & etc..

- To Run : npx cucumber-js --tags "@Validations" --exit

- https://cucumber.io/docs/cucumber/api/?lang=java

- https://cucumber.io/docs/cucumber/api/?lang=java#tag-expressions

- Tagged Hooks : https://github.com/cucumber/cucumber-js/blob/main/docs/support_files/hooks.md

- A scenario called have n number of tags you can give like @validation @foo

- Demo :

```
@Validations
Scenario Outline: Say hello
```

- Parameterization with Scenario outline & run tests Parallel in Playwright :

- Important Note : If you want to use parameterized then you want to give Scenario Outline then only cucumber will be recognized that this is a parameterized.

- If you give multiple data inside example in feature file then it will run that much time.

- Otherwise only Scenario is fine for without parameterization.

```
@Validations
Scenario Outline: Say hello
    Given a login to Ecommerce2 application with "<username>" and "<password>"
    Then Verify Error message is displayed

    Examples:
        | username          | password    |
        | anshika@gmail.com | Iamking@000 |
```

- Generate HTML reports for Cucumber Playwright & Rerun failed Scenarios :

- Cucumber can run only Scenarios in parallel, not feature files.

- So for e.g. there is 2 feature files then we cannot run these 2 files parallelly that is the limitation of cucumber, but you can run multiple Scenarios parallelly.

- If there is 2 Scenarios inside you feature file & we want to run Parallelly then : (Mentioned how many scenarios you want to run parallel here 2) :

Run : npx cucumber-js features/greeting.feature --parallel 2 --exit

→ Now if you want to **generate html report** then it is very easy, Run :

npx cucumber-js features/greeting.feature --parallel 2 --exit --format html:cucumber-report.html

→ **Retry :**

- https://github.com/cucumber/cucumber-js/blob/main/docs/retry.md

If you have a flaky scenario (e.g. failing 10% of the time for some reason), you can use *Retry* to have Cucumber attempt it multiple times until either it passes or the maximum number of attempts is reached. You enable this via the `retry` configuration option, like this:

- In a configuration file `{ retry: 1 }`
- On the CLI `cucumber-js --retry 1`

The number you provide is the number of retries that will be allowed after an initial failure.

Note: Retry isn't recommended for routine use, but can be a good trade-off in some situations where you have a scenario that's too valuable to remove, but it's either not possible or not worth the effort to fix the flakiness.

Some notes on how Retry works:

- Only relevant failing scenarios are retried, not the whole test run.
- When a scenario is retried, it runs all hooks and steps again from the start with a fresh World - nothing is retained from the failed attempt.
- When a scenario passes on a retry, it's treated as a pass overall in the results, although the details of each attempt are emitted so formatters can access them.

- To Retry : npx cucumber-js --retry 1 --exit

→ Also we are able to add inside package.json & Run : npm run cucumberRegression

package.json U

```
"cucumberRegression": "npx cucumber-js --tags \"@Regression\" --retry 1 --exit --format html:cucumber-report.html"
```

- BDD Main Code :

hooks.js U

```
const { After, Before, AfterStep, Status } = require("@cucumber/cucumber");
const playwright = require("@playwright/test");

// Before hook: Executed before all scenarios
Before(async function () {
  console.log("Launching browser and initializing test environment...");
  const browser = await playwright.chromium.launch({
    headless: false, // Launch the browser in non-headless mode for debugging
  });
  // Create a new browsing context & new page within the context
  const context = await browser.newContext();
  this.page = await context.newPage(); // world class constructor concept is used here
});

// AfterStep hook: Executed after each step
AfterStep(async function ({ result }) {
  // This hook will be executed after all steps, and take a screenshot on step failure
  if (result.status === Status.FAILED) {
    const buffer = await this.page.screenshot(); // Capture a screenshot
    await this.page.screenshot({ path: "screenshot1.png" }); // Save the screenshot to a file
    this.attach(buffer.toString("base64"), "base64:image/png"); // Attach the screenshot to the test report
    console.log("Step failed! Screenshot captured."); // Log message indicating that a screenshot is captured on step failure
  }
});

// After hook: Executed after all scenarios
After(async function () {
  console.log("Test execution completed. Closing browser..."); // Log message indicating completion of test execution
});

// In selenium webdriver we want to quit browser at last using driver.quit(); But In Playwrite its automatically closed & clear all
```

Ecommerce.feature U

```
Feature: Ecommerce validations

    @Regression
    Scenario Outline: Placing the Order
        Given a login to Ecommerce application with "<username>" and "<password>"
        When Add "ZARA COAT 3" to Cart
        Then Verify "ZARA COAT 3" is displayed in the Cart
        When Enter valid details and Place the Order
        Then Verify order is present in the OrderHistory

        Examples:
            | username          | password    |
            | anshika@gmail.com | Iamking@000 |
```

ErrorValidation.feature U

```
Feature: Login Failed Validation

    @Validations
    Scenario Outline: Verify Error message is displayed after login
        Given a login to Ecommerce2 application with "<username>" and "<password>"
        Then Verify Error message is displayed

        Examples:
            | username          | password    |
            | anshika@gmail.com | Iamking@000 |
```

greeting.feature U

features > greeting.feature

Click here to ask Blackbox to help you code faster

```
Feature: Greeting

    @Regression
    Scenario Outline: Say hello
        Given a login to Ecommerce application with "<username>" and "<password>"
        When Add "ZARA COAT 3" to Cart
        Then Verify "ZARA COAT 3" is displayed in the Cart
        When Enter valid details and Place the Order
        Then Verify order is present in the OrderHistory

        Examples:
            | username          | password    |
            | anshika@gmail.com | Iamking@000 |

    @Regression
    Scenario Outline: Say bye
        Given a login to Ecommerce2 application with "<username>" and "<password>"
        Then Verify Error message is displayed

        Examples:
            | username          | password    |
            | anshika@gmail.com | Iamking@000 |
```

verify_table_step.feature U

features > verify_table_step.feature

Click here to ask Blackbox to help you code faster

```
Feature: Verify table step

    Scenario: Verify table step
        Given a table step
            | Product   | Quantity |
            | Apricot   | 5        |
            | Brocolli  | 2        |
            | Cucumber  | 10       |
```

steps.js U

features > step_definitions > steps.js > ...

Click here to ask Blackbox to help you code faster

```
const { When, Given, Then } = require("@cucumber/cucumber");
const { expect } = require("@playwright/test");
const { POManager } = require("../../Page-Objects/POManager");
const assert = require("assert");
let poManager;

// Step definition for verifying order in order history
Then("Verify order is present in the OrderHistory", async function () {
  await this.dashboardPage.navigateToOrders();
  const ordersHistoryPage = poManager.getOrdersHistoryPage();
  await ordersHistoryPage.searchOrderAndSelect(this.orderId);
  expect(
    this.orderId.includes(await ordersHistoryPage.getOrderId())
  ).toBeTruthy();
});

// Step definition for placing an order
When("Enter valid details and Place the Order", async function () {
  await this.cartPage.Checkout();

  const ordersReviewPage = poManager.getOrdersReviewPage();
  await ordersReviewPage.searchCountryAndSelect("ind", "India");
  this.orderId = await ordersReviewPage.SubmitAndGetOrderId();
  console.log(this.orderId);
});

// Step definition for verifying a product in the cart
Then("Verify {string} is displayed in the Cart", async function (productName) {
  this.cartPage = poManager.getCartPage();
  await this.cartPage.VerifyProductIsDisplayed(productName);
});

// Step definition for adding a product to the cart
When("Add {string} to Cart", async function (productName) {
  this.dashboardPage = poManager.getDashboardPage();
  await this.dashboardPage.searchProductAddCart(productName);
  await this.dashboardPage.navigateToCart();
});
```

steps.js U

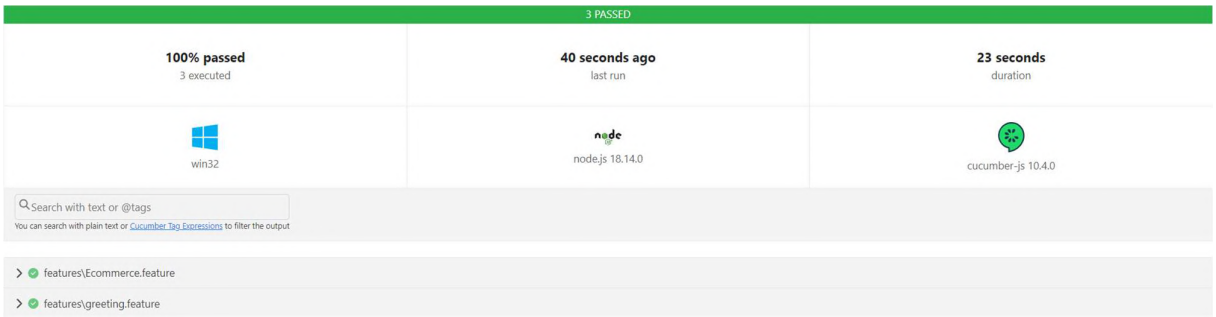
features > step_definitions > steps.js > ...

```
40  // Step definition for logging in to the Ecommerce application
41  Given(
42    "a login to Ecommerce application with {string} and {string}",
43    { timeout: 100 * 1000 },
44    async function (username, password) {
45      poManager = new POManager(this.page);
46      //js file- Login js, DashboardPage
47      const loginPage = poManager.getLoginPage();
48      await loginPage.goTo();
49      await loginPage.validLogin(username, password);
50    }
51  );
52
53  // Step definition for a table step
54  Given(/^a table step$/, function (table) {
55    const expected = [
56      ["Apricot", "5"],
57      ["Brocolli", "2"],
58      ["Cucumber", "10"],
59    ];
60    assert.deepEqual(table.rows(), expected);
61  });
62
63  // Step definition for logging in to the Ecommerce2 application
64  Given(
65    "a login to Ecommerce2 application with {string} and {string}",
66    { timeout: 100 * 1000 },
67    async function (username, password) {
68      const userName = this.page.locator("#username");
69      const signIn = this.page.locator("#signInBtn");
70      await this.page.goto("https://rahulshettyacademy.com/loginpagePractise/");
71      console.log(await this.page.title());
72      await userName.type("rahulshetty");
73      await this.page.locator("[type='password']").type("learning");
74      await signIn.click();
75    }
76  );
77
78  // Step definition for verifying error message display
79  Then("Verify Error message is displayed", async function () {
80    await expect(this.page.locator("[style*='block']")).toContainText(
81      "Incorrect"
82    );
83  });
```

steps.js U

features > step_definitions > steps.js > ...

```
84
85  //npx cucumber-js features/ErrorValidation.feature --exit
86  //npx cucumber-js --parallel 2 --exit --format html:cucumber-report.html
87  //npx cucumber-js features/greeting.feature --parallel 2 --exit --format html:cucumber-report.html
88
```



➢ **Playwright Run Report : Date 17th April : 03:10 PM**

Run : npx playwright test

Total Tests Run : 44

```
  4 flaky
    [chromium] › 08-Web-API-Part1.spec.js:29:1 › @API Place the order ──────────────────────
    [chromium] › 10-Network-Test-Intercept.spec.js:22:1 › @SP Place the order ──────────────
    [chromium] › 12-Upload-Download.spec.js:28:1 › Upload download excel validation ────────
    [chromium] › 13-ClientApp-Page-Object.spec.js:11:3 › @Webs Client App login for ZARA COAT 3 ──
  1 skipped
  39 passed (1.6m)
```

- Total Passed Run : 39

- Total Skipped Run : 01

- Total Flaky Run : 4

- Flaky means that cases failed when first run but when again run 1 time that pass.

➢ **Playwright BDD Cucumber Run Report : Date 17th April : 03:20 PM**

```
PS C:\Users\YBOROLE\Playwright_Testing\Playwright-Udemy> npx cucumber-js features --exit
Launching browser and initializing test environment...
..[
  'ZARA COAT 3',
  'ADIDAS ORIGINAL',
  'IPHONE 13 PRO',
  'Laptop',
  'Laptop',
  'Laptop',
  'Laptop',
  'Laptop',
  'Laptop',
  'Laptop',
  'Laptop'
]
.. | 661f9ab8a86f8f74dcc4702c |
..Test execution completed. Closing browser...
.Launching browser and initializing test environment...
.LoginPage Practise | Rahul Shetty Academy
..Test execution completed. Closing browser...
.Launching browser and initializing test environment...
..Test execution completed. Closing browser...
.

5 scenarios (5 passed)
15 steps (15 passed)
0m27.110s (executing steps: 0m27.066s)
PS C:\Users\YBOROLE\Playwright_Testing\Playwright-Udemy>
```

|| Playwright using JavaScript ||

- **To Run Playwright tests :** npx playwright test

- **Generate Report :** npx playwright show-report

Playwright.config.js:

playwright.config.js

```
13  module.exports = defineConfig({
14    testDir: "./e2e",
15    /* Run tests in files in parallel */
16    fullyParallel: true,
17    /* Fail the build on CI if you accidentally left test.only in the source code. */
18    forbidOnly: !!process.env.CI,
19    /* Retry on CI only */
20    // retries: process.env.CI ? 2 : 0,
21    /* Retries the testcase 1 times once failed */
22    retries: 1,
23    /* Opt out of parallel tests on CI. */
24    workers: process.env.CI ? 1 : undefined,
25    /* Reporter to use. See https://playwright.dev/docs/test-reporters */
26    reporter: "html",
27    // reporter: [["html", { open: "always" }]],
28    // reporter: [["html", { outputFolder: "my-report" }]],
29    // reporter: "dot",
30    // reporter: "list",
31    // reporter: [["list"], ["json", { outputFile: "test-results.json" }]],
32    /* For custom Reports : See https://playwright.dev/docs/test-reporters#custom-reporters */
33
34    /* Shared settings for all the projects below. See https://playwright.dev/docs/api/class-testoptions. */
35    use: {
36      /* Collect trace when retrying the failed test. See https://playwright.dev/docs/trace-viewer */
37      trace: "on-first-retry",
38      /* Capture screenshot of failed test */
39      screenshot: "only-on-failure",
40
41      /* Base URL to use in actions like `await page.goto('/')`. */
42      // baseURL: 'http://127.0.0.1:3000',
43
44      headless: false,
45      ignoreHTTPSErrors: true,
46      viewport: { width: 1280, height: 720 },
47      video: "on-first-retry",
48    },
```

playwright.config.js

```
13  module.exports = defineConfig({
49    /* Maximum time one test can run for. See https://playwright.dev/docs/test-timeouts */
50    timeout: 30 * 1000,
51    /* Maximum time expect() should wait for condition to be met. */
52    expect: {
53      timeout: 5000,
54    },
55
56    /* Folder for test artifacts such as screenshots, videos, tarces, etc. */
57    // outputDir: "test-results/",
58
```

```
  /* Configure projects for major browsers */
  projects: [
    {
      name: "chromium",
      use: {
        ...devices["Desktop Chrome"],
        headless: false,

        // want to handle go-location
        permissions: ["geolocation"],
        // To handle SSL Certificate error Automatically
        ignoreHTTPSErrors: true,
        // To change the browser view
        // viewport: { width: 720, height: 720 },
      },
    },
```

Playwright Run Options:

- https://playwright.dev/docs/running-tests

Playwright First Test:

```
 * 1. Open the page
 * 2. Click at Get started
 * 3. Mouse hover the language dropdown
 * 4. Click at  Java
 * 5. Check the URL
 * 6. Check the text "Installing Playwright" is not being displayed
 * 7. Check the text below is displayed
 *
```

14-Playwright-Website-Tests.spec.js

```
e2e > 14-Playwright-Website-Tests.spec.js > ...
Click here to ask Blackbox to help you code faster
const { test, expect } = require("@playwright/test");

test("check Java page", async ({ page }) => {
  await page.goto("https://playwright.dev/");
  await page.getByRole("link", { name: "Get started" }).click();
  await page.getByRole("button", { name: "Node.js" }).hover();
  await page.getByText("Java", { exact: true }).click();
  // await page.getByRole('navigation', { name: 'Main' }).getByText('Java').click(); // in case the locator above doesn't work, you can
  await expect(page).toHaveURL("https://playwright.dev/java/docs/intro");
  await expect(
    page.getByText("Installing Playwright", { exact: true })
  ).not.toBeVisible();
  const javaDescription = `Playwright is distributed as a set of Maven modules. The easiest way to use it is to add one dependency to yo
  await expect(page.getByText(javaDescription)).toBeVisible();
});
```

Playwright Locators:

https://playwright.dev/python/docs/locators

```
getAttribute    (method) Page.getAttribute(selector: stri…
getByAltText
getByLabel
getByPlaceholder
getByRole
getByTestId
getByText
getByTitle
```

1. getByRole() :

Consider the following DOM structure.

```
<h3>Sign up</h3>
<label>
  <input type="checkbox" /> Subscribe
</label>
<br/>
<button>Submit</button>
```

You can locate each element by it's implicit role:

```
await expect(page.getByRole('heading', { name: 'Sign up' })).toBeVisible();

await page.getByRole('checkbox', { name: 'Subscribe' }).check();

await page.getByRole('button', { name: /submit/i }).click();
```

```
(method) Page.getByRole(role: "alert" | "alertdialog" | "application" | "article" | "banner" |
"blockquote" | "button" | "caption" | "cell" | "checkbox" | "code" | "columnheader" | "combobox" |
"complementary" | ... 67 more ... | "treeitem", options?: {
    ...;
} | undefined): Locator
```

Allows locating elements by their ARIA role, ARIA attributes and accessible name.

2. getByText() :

Consider the following DOM structure:

```
<div>Hello <span>world</span></div>
<div>Hello</div>
```

You can locate by text substring, exact string, or a regular expression:

```
// Matches <span>
page.getByText('world');

// Matches first <div>
page.getByText('Hello world');
```

```
// Matches second <div>
page.getByText('Hello', { exact: true });

// Matches both <div>s
page.getByText(/Hello/);

// Matches second <div>
page.getByText(/^hello$/i);
```

```
(method) Page.getByText(text: string | RegExp, options?: {
    exact?: boolean | undefined;
} | undefined): Locator
```

Allows locating elements that contain given text.

See also locator.filter([options]) that allows to match by another criteria, like an accessible role, and then filter by the text content.

3. getByAttribute() :

```
(method) Page.getAttribute(selector: string, name: string, options?: {
    strict?: boolean | undefined;
    timeout?: number | undefined;
} | undefined): Promise<string | null>
```

NOTE Use locator-based locator.getAttribute(name[, options]) instead. Read more about locators.

Returns element attribute value.

@param `selector`
A selector to search for an element. If there are multiple elements satisfying the selector, the first will be used.

@param `name` — Attribute name to get the value for.

@param `options`

4. **getByAltText() :**

```
(method) Page.getByAltText(text: string | RegExp, options?: {
    exact?: boolean | undefined;
} | undefined): Locator
```

Allows locating elements by their alt text.

Usage

For example, this method will find the image by alt text "Playwright logo":

```
<img alt='Playwright logo'>
await page.getByAltText('Playwright logo').click();
```

5. **getByLabel() :**

```
(method) Page.getByLabel(text: string | RegExp, options?: {
    exact?: boolean | undefined;
} | undefined): Locator
```

Allows locating input elements by the text of the associated `<label>` or `aria-labelledby` element, or by the `aria-label` attribute.

Usage

For example, this method will find inputs by label "Username" and "Password" in the following DOM:

```
<input aria-label="Username">
<label for="password-input">Password:</label>
<input id="password-input">
```

```
await page.getByLabel('Username').fill('john');
await page.getByLabel('Password').fill('secret');
```

@param `text` — Text to locate the element for.

6. **getByPlaceholder() :**

```
(method) Page.getByPlaceholder(text: string | RegExp, options?: {
    exact?: boolean | undefined;
} | undefined): Locator
```

Allows locating input elements by the placeholder text.

Usage

For example, consider the following DOM structure.

```
<input type="email" placeholder="name@example.com" />
```

You can fill the input after locating it by the placeholder text:

```
await page
    .getByPlaceholder('name@example.com')
    .fill('playwright@microsoft.com');
```

@param `text` — Text to locate the element for.

7. **getByTestId() :**

```
(method) Page.getByTestId(testId: string | RegExp): Locator
```

Locate element by the test id.

Usage

Consider the following DOM structure.

```
<button data-testid="directions">Itinéraire</button>
```

You can locate the element by it's test id:

```
await page.getByTestId('directions').click();
```

Details

By default, the `data-testid` attribute is used as a test id. Use selectors.setTestIdAttribute(attributeName) to configure a different test id attribute if necessary.

```
// Set custom test id attribute from

*@playwright*
/test config:
import { defineConfig } from '@playwright/test';

export default defineConfig({
use: {
testIdAttribute: 'data-pw'
},
});
```

@param `testId` — Id to locate the element by.

8. getByTitle() :

```
(method) Page.getByTitle(text: string | RegExp, options?: {
    exact?: boolean | undefined;
} | undefined): Locator
```

Allows locating elements by their title attribute.

Usage

Consider the following DOM structure.

```
<span title='Issues count'>25 issues</span>
```

You can check the issues count after locating it by the title text:

```
await expect(page.getByTitle('Issues count')).toHaveText('25 issues');
```

@param `text` — Text to locate the element for.

➪ **In Our first test we have used below locators :**

a -> link role

```
<a class="getStarted_Sjon" href="/java/docs/intro">Get started</a> == $0
```

```
await page.getByRole("link", { name: "Get started" }).click();
```

role = "button"

```
<a href="#" aria-haspopup="true" aria-expanded="false" role="button" class="navbar__link">…</a> == $0
```

```
await page.getByRole("button", { name: "Node.js" }).hover();
```

by Text -> Java

```
<a href="/java/docs/intro" target="_self" rel="noopener noreferrer" class="dropdown__link undefined dropdown__link--active"
data-language-prefix="/java/">Java</a> == $0
```

```
await page.getByText("Java", { exact: true }).click();
```

by Role -> navigation

```
await page.getByRole('navigation', { name: 'Main' }).getByText('Java').click();
```

Playwright Basic Tests:

```
test("has title", async ({ page }) => {
  await page.goto("https://playwright.dev/");
  // Expect a title "to contain" a substring.
  await expect(page).toHaveTitle(/Playwright/);
});

test("get started link", async ({ page }) => {
  await page.goto("https://playwright.dev/");
  // Click the get started link.
  await page.getByRole("link", { name: "Get started" }).click();
  // Expects the URL to contain intro.
  await expect(page).toHaveURL(/.*intro/);
});
```

Playwright Template:

15-Playwright-Template.spec.js

```
e2e > 15-Playwright-Template.spec.js > ...
Click here to ask Blackbox to help you code faster
import { test, expect } from "@playwright/test";

//AAA Pattern

// [Arrange]
// [Act]
// [Assert]

const password = process.env.PASSWORD;

test.beforeAll(async ({ playwright }) => {
  test.skip(
    !!process.env.PROD,
    "Test intentionally skipped in production due to data dependency."
  );
  // start a server before all tests start to be run
  // create a db connection before all tests start to be run
  // reuse a sign in state before all tests start to be run
});

test.beforeEach(async ({ page }, testInfo) => {
  console.log(`Running ${testInfo.title}`);
  // open a URL before each test start to be run
  // clean up the DB before each test start to be run
  // create a page object before each test start to be run
  // dismiss a modal before each test start to be run
  // load params before each test start to be run
});

test.afterAll(async ({ page }, testInfo) => {
  console.log("Test file completed.");
  // close a DB connection after all test executions done.
});
```

15-Playwright-Template.spec.js

```
e2e > 15-Playwright-Template.spec.js > ...
test.afterEach(async ({ page }, testInfo) => {
  console.log(`Finished ${testInfo.title} with status ${testInfo.status}`);

  if (testInfo.status !== testInfo.expectedStatus)
    console.log(`Did not run as expected, ended up at ${page.url()}`);
  // clean up all the data we created for this test through API calls
});

// test.describe('Test Cases outline', () => {
// test.describe.only('Only this test case will run', () => {
test.describe.skip("This testcases inside this block is not execute, alss cases will be skipped ", () => {
  test("Test Scenario One", async ({ page }) => {
    await test.step("Step One", async () => {
      // ...
    });

    await test.step("Step Two", async () => {
      // ...
    });

    // ...
  });

  test("Test Scenario Two", async ({ page }) => {
    // Arrange
    // Act
    // Assert
  });
  /**
    test.only('Test Scenario Three', async ({ page }) => {
      // Arrange
      // Act
      // Assert
    });
  */
```

```
    });
  */
  /**
    test.skip('Test Scenario Four', async ({ page }) => {
      // Arrange
      // Act
      // Assert
    });
  */
});
```

Playwright Page Object Model automation using TypeScript:

Home-Page.ts

Page-Objects > Home-Page.ts > ...

Click here to ask Blackbox to help you code faster

```typescript
import { type Locator, type Page, expect } from "@playwright/test";

export class HomePage {
  // Variables
  readonly page: Page;
  readonly getStartedButton: Locator;
  readonly pageTitle: RegExp;

  // Constructor
  constructor(page: Page) {
    this.page = page;
    this.getStartedButton = page.getByRole("link", { name: "Get started" });
    this.pageTitle = /Playwright/;
  }

  // Methods
  async clickGetStarted(): Promise<void> {
    await this.getStartedButton.click();
  }

  async assertPageTitle(): Promise<void> {
    await expect(this.page).toHaveTitle(this.pageTitle);
  }
}

export default HomePage;
```

Top-Menu-Page.ts

Page-Objects > Top-Menu-Page.ts > ...

Click here to ask Blackbox to help you code faster

```typescript
import { expect, Locator, Page } from "@playwright/test";

export class TopMenuPage {
  readonly page: Page;
  readonly getStartedLink: Locator;
  readonly nodeLink: Locator;
  readonly javaLink: Locator;
  readonly nodeLabel: Locator;
  readonly javaLabel: Locator;
  readonly nodeDescription: string = "Installing Playwright";
  readonly javaDescription: string = `Playwright is distributed as a set of Maven modules. The easiest way to use it is to add one depend

  constructor(page: Page) {
    this.page = page;
    this.getStartedLink = page.getByRole("link", { name: "Get started" });
    this.nodeLink = page.getByRole("button", { name: "Node.js" });
    this.javaLink = page
      .getByRole("navigation", { name: "Main" })
      .getByText("Java");
    this.nodeLabel = page.getByText(this.nodeDescription, { exact: true });
    this.javaLabel = page.getByText(this.javaDescription);
  }

  async hoverNode(): Promise<void> {
    await this.nodeLink.hover();
  }

  async clickJava(): Promise<void> {
    await this.javaLink.click();
  }

  async assertPageUrl(pageUrl: RegExp): Promise<void> {
    await expect(this.page).toHaveURL(pageUrl);
  }

  async assertNodeDescriptionNotVisible(): Promise<void> {
    await expect(this.nodeLabel).not.toBeVisible();
  }

  async assertJavaDescriptionVisible(): Promise<void> {
    await expect(this.javaLabel).toBeVisible();
  }
}
export default TopMenuPage;
```

```ts
// 16-Playwright-POM-Using-TypeScript.spec.ts
// e2e > 16-Playwright-POM-Using-TypeScript.spec.ts > ...
// Click here to ask Blackbox to help you code faster
import { test, type Page } from "@playwright/test";
import { HomePage } from "../Page-Objects/Home-Page";
import { TopMenuPage } from "../Page-Objects/Top-Menu-Page";

const URL = "https://playwright.dev/";
let homePage: HomePage;
let topMenuPage: TopMenuPage;
const pageUrl = /.*intro/;

test.beforeEach(async ({ page }) => {
  await page.goto(URL);
  homePage = new HomePage(page);
});

async function clickGetStarted(page: Page) {
  await homePage.clickGetStarted();
  topMenuPage = new TopMenuPage(page);
}

test.describe("Playwright website", () => {
  test("has title", async () => {
    await homePage.assertPageTitle();
  });

  test("get started link", async ({ page }) => {
    // Act
    await clickGetStarted(page);
    // Assert
    await topMenuPage.assertPageUrl(pageUrl);
  });
```

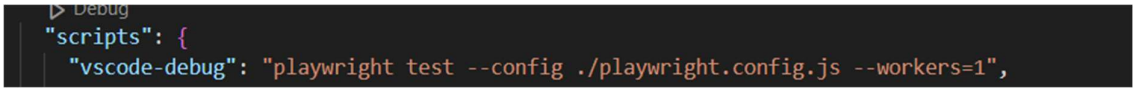
```ts
  test("check Java page", async ({ page }) => {
    await test.step("Act", async () => {
      await clickGetStarted(page);
      await topMenuPage.hoverNode();
      await topMenuPage.clickJava();
    });

    await test.step("Assert", async () => {
      await topMenuPage.assertPageUrl(pageUrl);
      await topMenuPage.assertNodeDescriptionNotVisible();
      await topMenuPage.assertJavaDescriptionVisible();
    });
  });
});
```

Playwright Visual Regression Testing with Applitools:

- npm i -D @applitools/eyes-playwright
- https://applitools.com/docs/api-ref/sdk-api/playwright/js-intro/checksettings
- https://applitools.com/docs/api-ref/sdk-api/playwright/js-intro/checksettings#region-match-levels
- https://applitools.com/tutorials/quickstart/web/playwright/typescript/integrate

Playwright setup a VSCode launch config for your Playwright tests:

```json
// launch.json
// .vscode > launch.json > ...
// Click here to ask Blackbox to help you code faster
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "node",
      "request": "launch",
      "name": "Playwright tests",
      "console": "integratedTerminal",
      "skipFiles": ["<node_internals>/**"],
      "runtimeExecutable": "npm",
      "runtimeArgs": ["run-script", "vscode-debug", "--", "${fileBasename}"]
    },
    {
      "type": "node",
      "request": "launch",
      "name": "Playwright Inspector",
      "console": "integratedTerminal",
      "skipFiles": ["<node_internals>/**"],
      "runtimeExecutable": "npm",
      "runtimeArgs": ["run-script", "vscode-debug", "--", "${fileBasename}"],
      "env": {
        "PWDEBUG": "1"
      }
    }
  ]
}
```



➔ https://dev.to/ryanroselloog/easiest-way-to-setup-a-vscode-launch-config-for-your-playwright-tests-2p9b

Playwright Extra Resources:

Test Configuration (playwright.config.ts)

1. https://playwright.dev/docs/test-configuration

Command Line

1. https://playwright.dev/docs/test-cli

Environment Variables

1. https://playwright.dev/docs/test-parameterize#passing-environment-variables

Locators

1. https://playwright.dev/docs/api/class-framelocator
2. https://playwright.dev/docs/best-practices#use-locators

Assertions

1. https://playwright.dev/docs/best-practices#use-web-first-assertions

Regular Expressions (RegExp)

1. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp

Auto-waiting

1. https://playwright.dev/docs/actionability

Async functions and Await

1. Why do we write await in Playwright

Arrange Act Assert (AAA)

1. https://medium.com/@pjbgf/title-testing-code-ocd-and-the-aaa-pattern-df453975ab80

Test Info

1. https://playwright.dev/docs/api/class-testinfo

Test Step

1. https://playwright.dev/docs/api/class-test#test-step

Page Object Model (POM)

1. https://playwright.dev/docs/pom

Locators

1. https://playwright.dev/docs/locators

Actions

1. https://playwright.dev/docs/input

BDD

1. Playwright, BDD, Cucumber and my opinion about it

Most used commands

1. .click()
2. .fill()
3. .type()
4. .check()
5. .uncheck()
6. .keyboard.press()

Most used asserts

1. .toHaveText()
2. .toBeVisible()
3. .toEqual()
4. .toContainText()
5. .toBeEnabled()
6. .toBe()
7. .toBeGreaterThan()
8. .toHaveCount()
9. .not

Playwright Visual Comparisons

1. https://playwright.dev/docs/test-snapshots

Applitools

1. https://auth.applitools.com/users/register
2. https://applitools.com/tutorials/quickstart/web/playwright/typescript

3. https://applitools.com/docs/api-ref/category/playwright-javascript-class-index
4. https://applitools.com/docs/topics/overview/using-the-ultrafast-grid.html
5. https://applitools.com/docs/topics/overview/overview-writing-tests-with-eyes-runner.html
6. https://applitools.com/docs/api-ref/sdk-api/playwright/javascript/
7. https://applitools.com/docs/api-ref/sdk-api/playwright/js-intro/checksettings
8. https://applitools.com/docs/api-ref/sdk-api/playwright/js-intro/runners
9. https://applitools.com/docs/api-ref/sdk-api/playwright/js-intro/ufg

More about visual testing

1. https://testingwithrenata.com/blog/visual-regression/visual-regression-testing-with-applitools/
2. https://testingwithrenata.com/blog/visual-regression/how-to-structure-visual-regression-testing-based-on-business-needs/

Playwright

1. https://playwright.dev/docs/trace-viewer-intro
2. https://playwright.dev/docs/test-ui-mode
3. https://playwright.dev/docs/debug
4. https://playwright.dev/docs/codegen-intro
5. https://playwright.dev/docs/test-configuration
6. https://playwright.dev/docs/test-global-setup-teardown#configure-globalsetup-and-globalteardown
7. https://playwright.dev/docs/test-global-setup-teardown#capturing-trace-of-failures-during-global-setup
8. https://playwright.dev/docs/trace-viewer
9. https://playwright.dev/docs/auth#multiple-signed-in-roles
10. https://playwright.dev/docs/auth#testing-multiple-roles-together
11. https://playwright.dev/docs/auth#authenticate-with-api-request
12. https://playwright.dev/docs/test-fixtures
13. https://playwright.dev/docs/api/class-apirequestcontext
14. https://playwright.dev/docs/api/class-apirequest
15. https://playwright.dev/docs/api/class-apiresponse
16. https://playwright.dev/docs/test-parameterize#passing-environment-variables
17. https://playwright.dev/docs/mock#mock-api-requests
18. https://playwright.dev/docs/mock-browser-apis
19. https://playwright.dev/docs/test-parameterize#create-tests-via-a-csv-file
20. https://testautomationu.applitools.com/playwright-intro/chapter4.html
21. https://auth.applitools.com/users/register
22. https://applitools.com/tutorials/quickstart/web/playwright/typescript
23. https://applitools.com/docs/api-ref/sdk-api/playwright/js-intro/checksettings
24. https://playwright.dev/docs/ci
25. https://playwright.dev/docs/test-parallel
26. https://testautomationu.applitools.com/github-actions-for-testing/

27. https://timdeschryver.dev/blog/using-playwright-test-shards-in-combination-with-a-job-matrix-to-improve-your-ci-speed
28. https://playwrightsolutions.com/whats-an-easy-way-to-tell-how-many-workers/

GitHub repos worth taking a look :

1. https://github.com/raptatinha/tau-introduction-to-playwright
2. https://github.com/MarcusFelling/demo.playwright
3. https://github.com/SwathiVisagn123/Playwright_TS
4. https://github.com/PauloGoncalvesBH/running-playwright-on-aws-lambda
5. https://github.com/raptatinha/blog-playwright-bdd
6. https://github.com/vitalets/playwright-bdd

- **Advanced Playwright:**

➔ https://github.com/raptatinha/tau-advanced-playwright

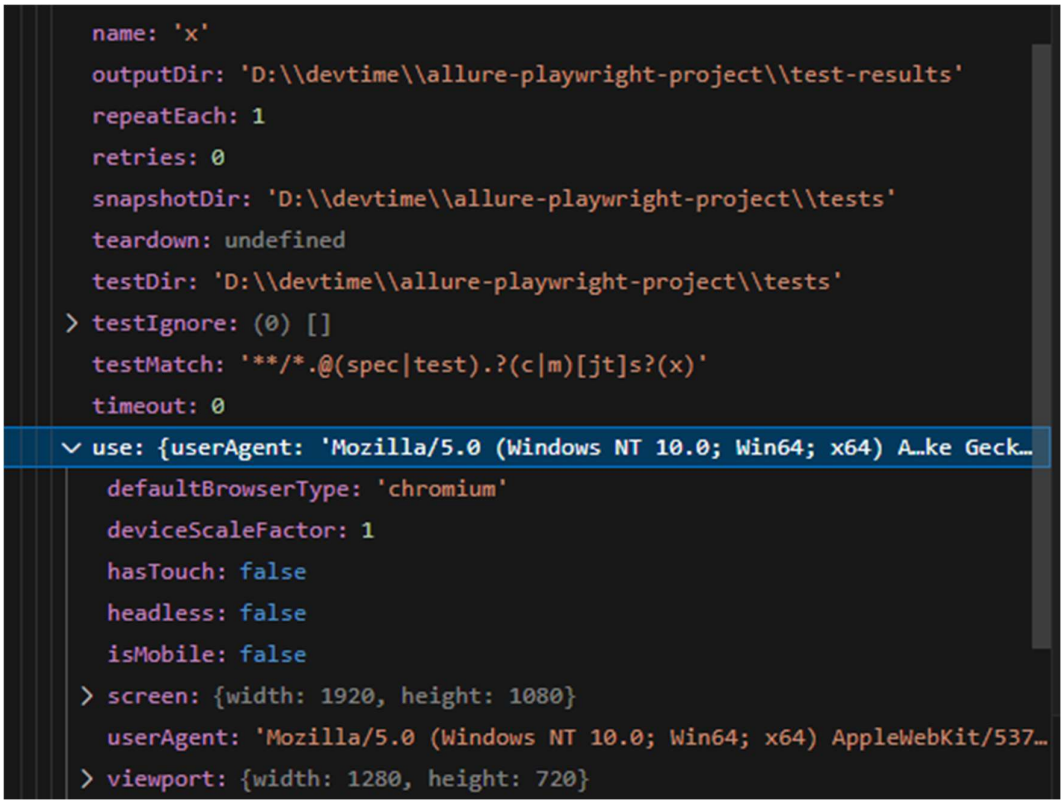
- **Playwright Get more details about your test during test run:**

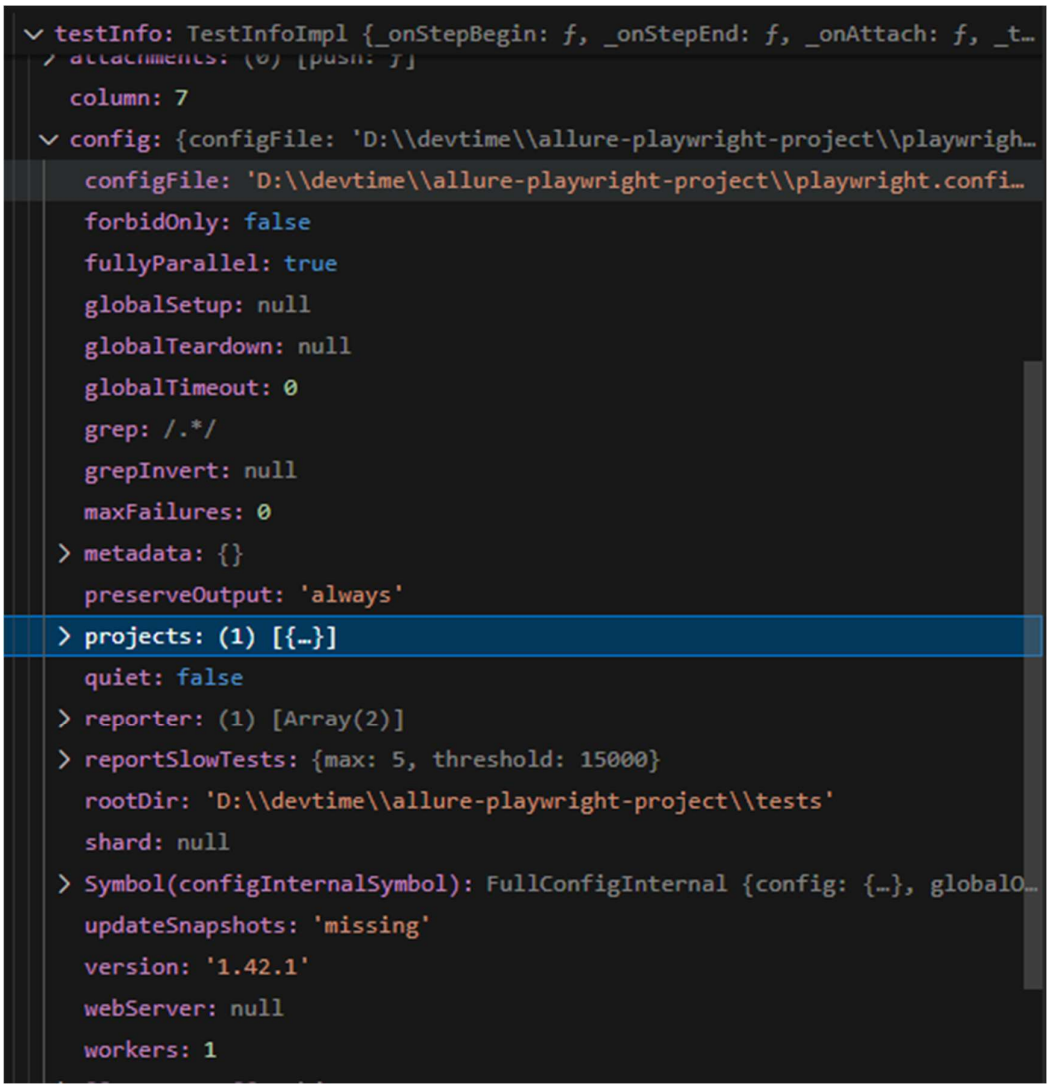
There is a way that you can access some particular values related to your test in real time. Say you have a complex project that has a dynamic configuration based on environments, test-data or whatever other details, and you want to see during test run what particular values are set. You can do that by accessing `testInfo` object within the test. Here is a snippet of code example to see how to access it:

```
import { test } from "@playwright/test";

test.describe('test suite name', () => {

  test("test name", async ({ page }, testInfo) => {
    console.log(`test name: ${testInfo.title}`)
    console.log(`parallel index :${testInfo.parallelIndex}`)
    console.log(`shard index: ${JSON.stringify(testInfo.config.shard)}`
  });
});
```

Below are some screenshots of other values you can access via `testInfo`

```
testInfo: TestInfoImpl {_onStepBegin: ƒ, _onStepEnd: ƒ, _onAttach: ƒ, _t…
  fn: async ({\n    page\n  }, testInfo) => {\n    console.log(`test name…
  line: 6
  outputDir: 'D:\\devtime\\allure-playwright-project\\test-results\\paral…
  parallelIndex: 0
  project: {grep: /.*/, grepInvert: null, outputDir: 'D:\\devtime\\allure…
    __projectId: 'x'
  dependencies: (0) []
  grep: /.*/
  grepInvert: null
  metadata: undefined
```


name: 'x'
outputDir: 'D:\\devtime\\allure-playwright-project\\test-results'
repeatEach: 1
retries: 0
snapshotDir: 'D:\\devtime\\allure-playwright-project\\tests'
teardown: undefined
testDir: 'D:\\devtime\\allure-playwright-project\\tests'
testIgnore: (0) []
testMatch: '**/*.@(spec|test).?(c|m)[jt]s?(x)'
timeout: 0
use: {userAgent: 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) A…ke Geck…
defaultBrowserType: 'chromium'
deviceScaleFactor: 1
hasTouch: false
headless: false
isMobile: false
screen: {width: 1920, height: 1080}
userAgent: 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537…
viewport: {width: 1280, height: 720}


testInfo: TestInfoImpl {_onStepBegin: ƒ, _onStepEnd: ƒ, _onAttach: ƒ, _t…
column: 7
config: {configFile: 'D:\\devtime\\allure-playwright-project\\playwrigh…
configFile: 'D:\\devtime\\allure-playwright-project\\playwright.confi…
forbidOnly: false
fullyParallel: true
globalSetup: null
globalTeardown: null
globalTimeout: 0
grep: /.*/
grepInvert: null
maxFailures: 0
metadata: {}
preserveOutput: 'always'
projects: (1) [{…}]
quiet: false
reporter: (1) [Array(2)]
reportSlowTests: {max: 5, threshold: 15000}
rootDir: 'D:\\devtime\\allure-playwright-project\\tests'
shard: null
Symbol(configInternalSymbol): FullConfigInternal {config: {…}, globalO…
updateSnapshots: 'missing'
version: '1.42.1'
webServer: null
workers: 1

```ts
// e2e > TS 17-test-details-reporting.spec.ts > ...
// Click here to ask Blackbox to help you code faster
import { test } from "@playwright/test";

test.describe("Test Details Reporting", () => {
  test("Display Test Information", async ({ page }, testInfo) => {
    // Displaying test name
    console.log(`Test Name: ${testInfo.title}`); // Display Test Information

    // Displaying parallel index
    console.log(`Parallel Index: ${testInfo.parallelIndex}`); // 0

    // Displaying shard index
    console.log(`Shard Index: ${JSON.stringify(testInfo.config.shard)}`); // null
  });
});
```

How to use Playwright to test multiple browser windows? :

Here is a way you can test multiple windows. **Not multiple tabs** in the same window. But multiple windows, each with its own storage and cookies. A use case would be if your website has implemented chat functionalities, and you want to see if the messages are delivered correctly. You can have two browsers logged in with two users and have them talk with each other. How would we achieve that in Playwright in one single test?

By using our `browser` and `page` objects. Here is an example of code

```ts
// e2e > TS 18-Cross-site-windows-test.spec.ts > ...
// Click here to ask Blackbox to help you code faster
import { test } from "@playwright/test";

test.describe("Cross-Site Functionality Test", () => {
  test("Testing multiple browser windows functionality between two different websites", async ({
    browser,
  }) => {
    // Open two browsers, each with its own context and page
    const user1Context = await browser.newContext();
    const user1Page = await user1Context.newPage();
    const user2Context = await browser.newContext();
    const user2Page = await user2Context.newPage();

    // Open different websites for both users
    await user1Page.goto("https://www.google.com");
    await user2Page.goto("https://www.yahoo.com");
  });
});
```

How Playwright handles multiple tabs in the same browser? :

For instance where an element would have a property like `target="_blank"` that upon clicking it, will open a new tab, then refer to this at playwright docs. If you find it hard to understand the `const pagePromise = context.waitForEvent('page')` just think of it as an event listener so it will not stop your test, it will just listen. Now right after you perform the click that opens the new tab put `const newPage = await pagePromise` there and from then on, you can use `newPage` the same as `user2Page` from my example above at point 2. Now you can cycle via `newPage` object or initial page object without the need to do any extra actions. Those of you who have done Selenium remember the switch back and forth with commands such as `driver.switchTo().window(actual)`, no need to do that here anymore. Here you have each page with its own object.

Remember that:

> browser.newContext() = new window (not yet complete browser, it still needs a tab)

> context.newPage() = new tab

Here are more examples to understand. Read the comments please.

```
import { test } from "@playwright/test";

test("Multiple windows and tabs default way", async ({ page }) => {
  // Default way of using playwright
  // page comes with values about the browser you have setup in config.
  // Ready to go, no need for extra actions.
  // this opens a window (context) and a tab (a page)
  await page.goto("https://duckduckgo.com/")
  });
```

```
import { test } from "@playwright/test";

test("Multiple windows and tabs", async ({ browser }) => {
  // this creates a new window but you can't perform actions
  // with page2Context since its not yet complete, it still needs a tab
  const page2Context = await browser.newContext()

  // we have a browser, we have a window we only need a tab. You do thi
  const page2 = page2Context.newPage()
  });
```

```
test("Multiple windows and tabs mix it", async ({ page, context, browser }) => {
  // this opens normally a full browser with window and tab (default way)
  await page.goto("https://duckduckgo.com/")

  // this creates a new tab from the same window (context) as line above
  const page2 = await context.newPage()
  await page2.goto("https://martioli.com/")

  // this sets up a new browser window with a tab.
  // Independent from the lines above
  const page3Context = await browser.newContext()
  const page3 = await page3Context.newPage()
  await page3.goto("https://github.com/adrianmaciuc")
  });
```

➔ https://playwright.dev/docs/pages?ref=blog.martioli.com#handling-new-pages

How to handle multiple types of browsers inside a test ? :

What I want to showcase below is not a way to test multiple browsers. There are far more efficient ways to do that. I am not sure if I will write about that because its fairly simple and the internet is full of such tutorials. But for the sake of getting our hands dirty and understanding how our browser instances are created, see below how you can play with various browsers directly inside your test scope.

```
test("Multiple browser drivers", async () => {
  const browser = await webkit.launch(); — 576ms
  const context = await browser.newContext(); — 676ms
  const page = await context.newPage(); — 926ms
  await page.goto("https://martioli.com/"); — 3118ms

  const browser2 = await firefox.launch(); — 1616ms
  const context2 = await browser2.newContext(); — 78ms
  const page2 = await context2.newPage(); — 2804ms
  await page2.goto("https://martioli.com/"); — 3062ms
});
```

Notice that there is no more { browser, page } . What happened here is we've pulled the webkit and firefox object inside the test scope. Its a bit of a stretch to do this. But for the purpose of us to understand and maybe develop in the future some creative ideas, it's good to know how it works.

Remember that, in a normal setup where you just use `test` , and then destructure `{ page }` the object will come with the values about the browser, that you have set inside your playwright config file or with the values that can be set dynamically in a terminal command or a pipeline.

Can I override Playwright configurations from within my test ? :

→ https://playwright.dev/docs/test-configuration?ref=blog.martioli.com

→ https://playwright.dev/docs/test-projects?ref=blog.martioli.com

→ https://playwright.dev/docs/test-use-options?ref=blog.martioli.com

→ https://playwright.dev/docs/test-configuration?ref=blog.martioli.com#advanced-configuration

We all know that our file called **playwright.config** holds the configuration and helps us set the project up, and all of our tests will run with those configs. But what if **I want to override the configurations only for one test** or a suite of tests ?

What if I want a suite of tests to run with a set of configs and another with other set of configs ?

You can do that in **two ways**:

The simple way. Create a project and have all the use options written for that project

An example for the first option, would be like this

```
import { defineConfig, devices } from '@playwright/test';

export default defineConfig({
  projects: [
      name: 'whateverNameForProject',
      use: {
        ...devices['Desktop Chrome'],
        colorScheme: 'dark',
        locale: 'fr-FR',
        httpCredentials: {
          username: 'yourUser',
          password: 'yourPass',
        },
        testIdAttribute: 'data-testid',
      }
    // any other config you would like goes here
  ],
});
```

Whenever you run your tests it will by default run all tests against all projects. So be sure to specify which project you want to run with `--project=whateverNameForProject`

The dirty way. Override config values from within your tests

For the second option, imagine that you want a particular test suite or a spec file to have tests with a special setting for geolocation or viewports, maybe a test has some simple login credentials you have to bypass. In other words, whatever you see here, any of these values can be manipulated inside our tests. This is how you do it

```
import { test } from "@playwright/test";

test.use({
    geolocation: { longitude: 36.095388, latitude: 28.0855558 },
    userAgent: 'my super secret Agent value'
  })

test("Override config", async ({ page }) => {
    await page.goto("https://martioli.com/")
})
```

This will keep the rest of the settings and just override what you need.

> Easter Egg -> let me know the name of the location I used here in the geolocation and you shall win the title as my golden reader

If you want to have in the same spec file, multiple suites that each come with their own extra config you can do like this

```
import { test, expect, webkit, firefox, chromium } from "@playwright/te

test.describe('Override suite 1', () => {
  test.use({
    viewport: { width: 400, height: 810 },
    geolocation: { longitude: 36.095388, latitude: 28.0855558 },
    userAgent: 'my super secret Agent value'
  });

  test("Override test 1", async ({ page }) => {
    await page.goto("https://martioli.com/")
  })

});

test.describe('Override suite 2', () => {
  test.use({
    viewport: { width: 768, height: 1024 },
    geolocation: { longitude: 36.095388, latitude: 28.0855558 },
    userAgent: 'my second super secret Agent value nr 2'
  });

  test("Override test 2", async ({ page }) => {
    await page.goto("https://martioli.com/")
  })

});
```

If you want the same override config at spec file level, just move the test.use() at the top of the page and it will apply to all the suites inside the file.

Configurations can be also done via globalSetup. Its a more elegant and advanced way of doing things. I will probably do a blog post about it in the future.

Not that elegant but still another way to do it, you can pass in configuration override using context.

```
import { test, devices } from "@playwright/test";

test("Override test 1", async ({ browser }) => {
  const context = await browser.newContext({
    ...devices['iPhone 13'],
    isMobile: true
  })
  const page = await context.newPage()
  await page.goto("https://martioli.com/")
})
```

Promise.all() in Playwright :

I'll use the `waitForResponse()` method to showcase

Let's say, we have a Search Input field and a button that triggers the search, eventually making a request to an API (https://example.com/api/search, the search term is in the request body)

You would probably write something like this

```
await page.locator("button").click() // search button

await page.waitForResponse("https://example.com/api/search")
```

With the above code, there's a (high) chance that we already received a response from `https://example.com/api/search` before we reached the `await page.waitForResponse(" https://example.com/api/search ")` line. The `.click()` method doesn't resolve immediately, but performs a range of (time-consuming) steps before resolving the await promise and continuing to the next line.

Await executes code asynchronous in sequence, one after another.

What we want here is for `await page.locator("button").click()` and `await page.waitForResponse(" https://example.com/api/search ")` to be executed at the same time - so that both can do their job properly.
That's where Promise.all() comes into play.

Promise.all() executes promises concurrently, meaning,

```
const [response] = await Promise.all([
  page.locator("button").click(),
  page.waitForResponse("https://example.com/api/search")
]);
```

Executes both `.click()` and `.waitForResponse()` at the same time. The await Promise.all() as a whole only resolves when all of its argument promises passed. The issue we've noticed here is called race condition.

Many Playwright events `(.waitForRequest(), .waitForResponse(), .waitForEvent(), ...)` must execute concurrently with their triggers using Promise.all.

How to find a child element if only its parents have unique ids :

For example if you have `uniqueIDParent > div1 > div2 > span` (where span has the "text you want") , but there are multiple children (spans) with different text, if you give to playwright its parent or grandparent or greatgreat, it will traverse all of its children and extract all text. `expect(uniqueID).toHaveText("text you want")` will work. In the same time `page.getByTestId(uniqueIDParent).filter({ hasText: "text you want" })` will work just fine if you target only the child that has the text you want. Filtering works miracles.

→ https://playwright.dev/docs/locators?ref=blog.martioli.com#filtering-locators

Why do I get sometimes in Playwright the error browser has been closed :

One reason may be, because you probably forgot an `await` somewhere. Playwright works in asynchronous way, meaning that its all promises, but you need your test steps to run sequentially, top to bottom and to perform the actions on that exact order. This is achieved by using the `await` key that will resolve the promise and this is how Playwright will keep your steps in order, to avoid issues about race conditions. Sometimes VS Code will suggest to you that you don't need some awaits, disregard that, you **do** need them.

Playwright Auto-waits :

Here are some typical examples of assertions from Selenium that you do not need to perform anymore:

- You don't have to assert element is visible before interacting with it. Playwright will do the following before an interaction:

> - it will check if the element is visible
>
> - if its attached to DOM
>
> - and if its stable, animation is completed

- When you open a page or when you click a link to redirect to a page, you don't have to assert the page is loaded before interacting with elements, playwright will wait for the page to fully load first
- You don't have to write explicit waits for an element when you are waiting for it to appear or waiting for it to disappear. Playwright has built-in timeouts (these explicit waits you know from Selenium) that will wait for an element and try to find it for a set interval of time (eg: by default is 5 seconds for an `expect(locator).toBeVisible()`)

There are very particular situations sometimes where the timeouts are not working for you :

→ https://playwright.dev/docs/next/api/class-frame?ref=blog.martioli.com#frame-wait-for-timeout

Or you just can't figure it out what to do about it, you still have a last option to simply wait a set number of seconds using the waitForTimeout() . This is discouraged in general, but sometimes you just have no other option.

How do I assert an array of strings in Playwright ? :

`expect(locator).toHaveText(array)` - this can actually take an array and it will, under the hood, iterate over it to assert the items exist

How to handle multiple elements in Playwright ? :

Playwright locators will find both one element or multiple elements with the same method, depends on what you give it. If you want to deal with multiple elements (in Selenium you would do `findElements` and it will return an array of elements) , in Playwright if you do `page.locator(multipleElements)` this will return multiple elements but **JUST** for playwright under the hood, not for you (at least not as simple array of objects). All you get is one single locator (object). **Why this happens?** Because you may want to find one single element and try maybe to click it, so when you perform the action, in case the attribute belongs to multiple elements it will not let you click on multiple elements and it will throw an error, suggesting you how to fix your test. But if you *really* want to deal with all the elements, how do I do that ? You have to add `.all()` at the end. Example: `page.locator(multipleElements).all()`

How to handle bigger chunks of text ? :

Sometimes you can have multiple items that have text, if you give to playwright the parent of all the items, then playwright can extract those texts as an array. You can achieve that with `page.locator(parentOfElementsWithText).allTextContents()` or with `.allInnerTexts()` . The downside of this is that it is not recommended to assert exact match of text, mainly because sometimes it will fetch new lines (/n) or commas or extra spaces, but it can be useful to use this with an `expect(locator).toContain()`

How do I assert the absence of an element in Playwright ? :

You can find this little hack on stackoverflow `expect(locator).toHaveCount(0)` . This is similar to Selenium `findElements` , and if element not found, it just returns an empty array. This is good because it will not fail the test. However I do recommend the Playwright builtin NOT operator , which is what you should actually use. Example `locator(element).not.toBeVisible()` . As a general practice all assert methods can be used with **NOT.**

How do I deal with the situation where simple use of just getByTestId() is not enough ? :

There are various scenarios where you have maybe tables, where you need to combine parents and children web elements to reach your desired data. So, before going to `locator()` to build your super complex `css selectors` or `xpath` , think about using the and operator , for example `page.getByText(elem).and(page.getByText(elem)` or you can do `page.getByTestId(elem).getByTestId(elem)` . Another option would be to use filtering locators. Filters also can be used with the mix of NOT operator

Using one parent element to do multiple actions :

You can store the element in a `const element = page.locator()` . Notice that this scenario is one of the few times when you don't need the `await` in front of `page.locator()` . You can look at it as a placeholder. Later in your code you can do `element.click()` or search thru its children `element.getByTestId(child)` . The *cool* feature about this is that every time you call the element it will re-query the DOM.

■ Do not use $(locator) or $$(multiple) :

Use of $ or $$ in Playwright is element handle and its highly discouraged by playwright. The main idea about this is that you can encounter situations where using $ will just reference an element from a previous version of the DOM. You may end up seeing the well known Selenium error `StaleElementReferenceException`

→ https://playwright.dev/docs/api/class-elementhandle?ref=blog.martioli.com

■ What if you have to wait for the app response and it takes longer than the usual 5 seconds ?:

You may encounter this, for example when you perform an action, a click, a submit, and it takes a longer time to get an answer, either you will have the 'spinning loader' or similar. If you know this happens you can increase the timeout to wait for a response on just one particular action, for example `expect(locator).toBeVisible({ timeout: 20000 })` . By passing the timeout inside the method this will override the default playwright configs only for that single line of code (hopefully you will no longer see the error `error: timed out 5000ms waiting for expect(locator).tobevisible()` and your test will pass)

■ Why sometimes expect does not have the usual toHaveText() method ? :

But if I try `toBe()` then it works. Because it depends what you are giving to `expect()` as an object. If you give it a standard locator it will have all the web first assertions but if you give it a modified object, something similar to `page.locator(elementWithText).innerText()` , because of the `innerText()` method this is no longer a standard playwright object, and with the rest of the objects it uses the jest expect methods. The playwright expect object will detect what type of object you give it and will work with either the jest expect like `toBe()` or the playwright methods like `toHaveText()` . Just remember that the playwright methods have auto-wait retry and the jest one does not.

→ https://playwright.dev/docs/best-practices?ref=blog.martioli.com#use-web-first-assertions

■ Multiple data test ID attributes on web elements :

We can configure in our playwright to use a custom test id attribute. When using the method `getByTestId()` Playwright will default to `data-testid="selector"` , so in case you have a different kind of default id locator set in your web app, you have to set up your testIdAttribute first. This includes if your website has unique ids under the format `id="selector"` . But **can you config to use multiple test id** attributes? Answer is **NO**. However, there is a small hack. You can use a different `testIdAttribute` in your general config and have a different one in your projects. Or you can have different projects with different IDs. This is used when you have two teams or group of teams that developed one app using one type of unique ID and another with a different one. It may be a situation of migrating away from a legacy app. Here is below an example

→ https://playwright.dev/docs/api/class-testoptions?ref=blog.martioli.com#test-options-test-id-attribute

```
projects: [
  {
    name: "new-mega-awesome-app",
    testDir: "./tests",
    use: {
      ...devices["Desktop Chrome"],
      testIdAttribute: "id",
      baseURL: "https://newapp.domain.com",
    },
  },
  {
    name: "legacy-app",
    testDir: "./tests-for-legacy-app",
    use: {
      ...devices["Desktop Chrome"],
      testIdAttribute: "data-testid",
      baseURL: "https://legacyapp.domain.com",
    },
  },
],
```

How to handle an element that appears after full page load in playwright :

One of those rare cases where an element will appear in DOM only after some conditions or scripts are executed. Even if Playwright will wait for the page to fully load, you may encounter situations where your element will appear later than that. In this situation here are a few tricks I recommend:

- First option, wait for network to be idle: `page.goto('https://playwright.dev/', {waitUntil: 'networkidle'})`
- Second option is to wait for the elements particular state. Use element.waitFor(state). Default value is "visible" but you can change to, for example, be present in the DOM, by doing first `const element = page.locator('#locator')` and then `await element.waitFor("attached")`

→ https://playwright.dev/docs/api/class-locator?ref=blog.martioli.com#locator-wait-for

Custom expect message :

→ https://playwright.dev/docs/test-assertions?ref=blog.martioli.com#custom-expect-message

Did you know that you can have your own custom expect message when an assertion fails? `await expect(locatorOrValue, 'Failed to perform something').toBe()`. This can be very useful when you use them in custom built methods or page methods and you want to have a more detailed message of what failed.

Is there a way to fail an assertion but not fail the test? :

→ https://playwright.dev/docs/test-assertions?ref=blog.martioli.com#soft-assertions

→ https://playwright.dev/docs/test-assertions?ref=blog.martioli.com#expecttopass

Or maybe you are expecting something that appears at random intervals of time. Playwright has an expect method that can repeat expect assertions (multiple yes) until they ALL pass, and its called expect.toPass(). Just group all assertions inside an expect.toPass(). How this method works is very similar to what I've explained above at expect.poll() but this time inside our expect method we can perform one or multiple expect() that may fail a couple of times before they succeed. Condition is they ALL have to pass. So the block of code repeats until the condition is met (or it timesout). Here is an example :

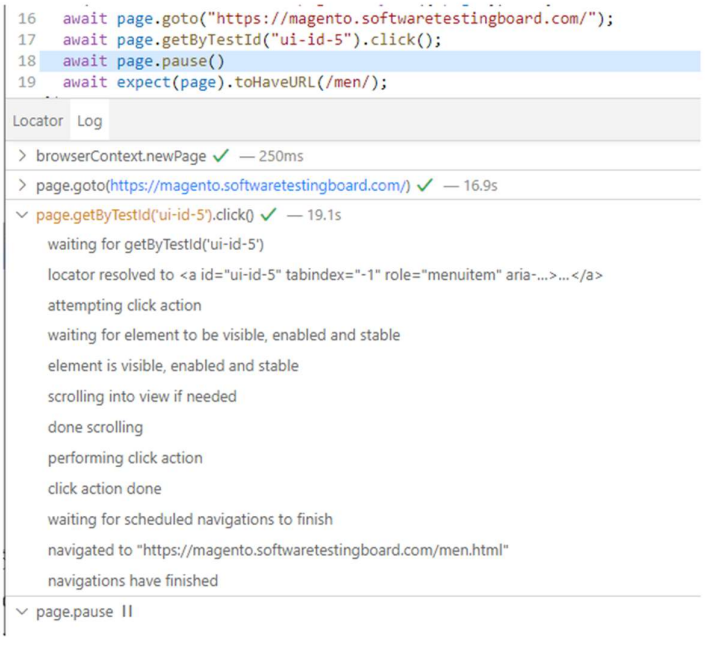
```
test("Element appears after 5 seconds", async ({ page }) => {
    await page.goto("https://webdriveruniversity.com/Accordion/index.ht
    await expect(async () => {
        await expect(page.getByText("LOADING")).toBeVisible()
        await expect(page.getByText("COMPLETE.")).toBeVisible()
    }).toPass({
        intervals: [1_000, 5_000, 10_000],
        timeout: 60_000
    });
});
```

Notice configuration now goes inside the toPass({})

How to debug like a pro in playwright ? :

➔ https://blog.martioli.com/debugging-tests-with-playwright/
➔ https://playwright.dev/docs/debug?ref=blog.martioli.com#playwright-inspector

This topic actually deserves its own blog post, and I've written it here. But just wanted to point out one thing that probably almost everyone missed out. Playwrights own debugger called the inspector can give you a LOT more than you think. Let me show you what I mean. Lets say you have one line of code that you just can't figure it out why you can't achieve what you want. Here's what you can do, start your test with `npx playwright test path/to/yourTest.spec.js --debug`, then go to the inspector and open the action you want to find out more about, and behold all of the actions Playwright is performing under the hood:



How to get text and store in a variable in playwright for later reuse ? :

Playwright will recommend to give an element to expect() and use its toHaveText() method to assert if an element has a certain text, but sometimes you just need to fetch that text from an element and later reuse it for other actions. This happens when values are not set and come dinamically. You can achieve this with `const elementText = await page.locator(locator).innerText()` . This will give you a value as a string inside elementText to later use.

> Note that innerText() will give you only visible text. So if text is inside the DOM but is not visible then it will not return that value. To include even hidden text use textContent()

How can I intercept a network call in playwright ? :

➔ https://playwright.dev/docs/api/class-page?ref=blog.martioli.com#page-route
➔ https://smartbear.com/learn/automated-testing/how-to-perform-end-to-end-testing/?ref=blog.martioli.com#vertical-vs-horizontal-endtoend-testing
➔ https://playwright.dev/docs/api/class-page?ref=blog.martioli.com#page-wait-for-response

There are two types of end to end testing, horizontal and vertical. Most common one is horizontal where you test for example purchasing a product. This is an user flow. And the less known one is vertical, where you test `user action -> Front End (UI) -> Back End -> database` or `database -> Back End -> Front End -> what user sees` .

Take for example an invoice that is generated if you click a button and the values are displayed at UI level in your account. You may have the front end perform a computation of data received from the backend to display the values. What if values are displayed incorrectly, who is to blame? front end devs will say its a backend problem and backend devs will say its a frontend problem. Sounds familiar? In order to catch this kind bug you do it like this:

- for the front end you put an assertion of value that will be displayed in the UI
- for the backend you listen to network calls and assert **what** the backend sends via the API **before** it reaches frontend

This will give you validation at two levels/layers . So, how do we intercept a call in playwright? With the use of waitForResponse . Let me give you a snippet of code for further clarification:

```
// the preceding double asterix is used to have dinamic wait on multipl
// Notice waitForResponse has no await. It just marks SINCE WHEN you wa
const invoiceCall = page.waitForResponse("**/invoices/*");
await page.getByText("Generate Invoice").click();

// It will default wait for 30 seconds for the call
const response = await invoiceCall

// now make that response as json to be easily read
const responseAsJson = await response.json();
await expect(responseAsJson.invoice.value).toBe("355");
```

> Why should I care about the SINCE WHEN part? This comes in handy when you want to intercept multiple calls from same endpoint. More on that soon.

How can I assert receiving of an email with playwright ? :

➔ https://en.wikipedia.org/wiki/Polling_(computer_science)?ref=blog.martioli.com
➔ https://playwright.dev/docs/test-assertions?ref=blog.martioli.com#expectpoll

Imagine that you have a registration form, and upon submitting that form an email is being sent with the confirmation number/link. Since the email does not always appear instant in your inbox, you can perform a polling technique , using expect.poll from playwright. Read the comments below as I try to explain how it works.

```
await expect.poll(async () => {
  const allEmails = await page.request.get('https://api.email.com/allEmails')
  // do some logic here, for example search the emails for the one that includes the registration form data of username/email
  for (email of allEmails){
      if (email.to.includes("registrationForm@email.com"))
          // once email found write the logic to extract link/code from email contents
          return emailCode
  }
  return false;
}, {
  // this part is the configuration of your polling and they are all optional
  // Custom error message, triggered after timeout runs out.
  message: 'Failed to find confirmation link in email',
  // Probe, wait 1s, probe, wait 2s, probe, wait 10s, probe, wait 10s, probe
  // ... Defaults to [100, 250, 500, 1000].
  intervals: [1_000, 2_000, 10_000],
  // last value in interval is repeated until timeout. And it will throw an error with the custom message
  timeout: 60_000
}).toBeTruthy();
```

Publish your playwright reports to GitHub pages:

➔ https://playwright.dev/docs/test-assertions?ref=blog.martioli.com#expectpoll
➔ https://github.com/marketplace/actions/github-pages-action?ref=blog.martioli.com
➔ https://playwright.dev/docs/test-sharding?ref=blog.martioli.com#merging-reports-from-multiple-shards
➔ https://playwrightsolutions.com/whats-an-easy-way-to-tell-how-many-workers/?ref=blog.martioli.com
➔ https://playwright.dev/docs/test-sharding?ref=blog.martioli.com
➔ https://playwright.dev/docs/test-sharding?ref=blog.martioli.com#merging-reports-from-multiple-shards
➔ https://playwright.dev/docs/test-sharding?ref=blog.martioli.com

Full parallelization in Playwright :

➔ https://blog.martioli.com/full-parallelization-with-playwright/

A Comparative Analysis of Playwright Adoption vs Cypress and Selenium :

➔ https://ray.run/blog/the-rapid-adoption-of-playwright-test-in-software-qa
➔ https://medium.com/version-1/playwright-vs-selenium-which-is-faster-and-more-stable-c41ba1a89c0a

PARAMETER	PLAYWRIGHT	SELENIUM
Applicability	Large scale Web scraping	Large scale Web scraping
Parallel Testing	Supports	Supports
Browsers	Cross Browser Support- Chrome, Edge, Firefox	Cross Browser Support- Chrome, Edge, Firefox, Safari, Opera.
Language	TypeScript, JavaScript, Python, .Net, Java	Multi Language Support- java, python, C#, Ruby, PHP, Perl, JavaScript
Mode	Headless and Headed mode	Headless and Headed mode
Mobile Automation?	Yes	Yes
Wait	Auto wait feature	Built in support for all types of wait. Needs to be coded.
Screenshots and Screen Recording	Built in support	Built in support for screenshot. No screen recording feature available.
Community	Low to Moderate Usage- Growing Community	Largest Community and Usage. Wide range of resources. Hence, greater capability.
Documentation	Comprehensive Documentation	Extensive Documentation and Mature
OS	Windows, Mac OS, Linux	Windows, Mac OS, Linux, Solaris
Types of Testing	Functional and API only	Functional, API, Non-functional (small scale), Network calls, etc
Updates	Regular	Regular
Real/Virtual Devices	Does not support real devices out-of-the-box.	All Real and Virtual Devices
iFrames	Yes	Yes

Running Playwright Tests via Azure DevOps Pipeline :

➔ https://medium.com/@info_70421/running-playwright-tests-via-azure-devops-pipeline-6b39e34e361b

Strategic Tagging: Optimizing Your Playwright Test Suit :

→ https://medium.com/@merisstupar11/strategic-tagging-optimizing-your-playwright-test-suit-4ab109343fed

Content Security Policy in Playwright Automation :

Are you diving into the realm of Microsoft Playwright automation using Dotnet? If so, understanding Content Security Policy (CSP) and its implications when working with JavaScript is crucial. Let's delve into this topic and explore how to effectively navigate CSP challenges in your automation workflows.

Content Security Policy (CSP) is a critical aspect of web security, often set to true by default for certain applications. CSP serves as a cross-site scripting policy, explicitly blocking JavaScript execution to mitigate against potential security threats. While CSP enhances security, it can also pose challenges for automation engineers working with JavaScript-dependent applications.

To execute JavaScript code within applications bound by CSP restrictions, bypassing CSP becomes essential. Fortunately, Microsoft Playwright provides mechanisms to bypass CSP seamlessly. Let's explore a code snippet that demonstrates how to bypass CSP effectively:

```
public static async Task<(IBrowserContext, IPage)> GetPage(IBrowser?
browser)
{
    if (browser == null)
        throw new Exception("Error: Browser object is null");
    var context = await browser.NewContextAsync(new
BrowserNewContextOptions { BypassCSP = true });
    //var context = await browser.NewContextAsync();
    //var page = await browser.NewPageAsync(new BrowserNewPageOptions {
BypassCSP = true });
    var page = await context.NewPageAsync();
    return (context, page);
}
```

> In the provided code snippet, we utilize the BypassCSP option to circumvent CSP restrictions, allowing seamless execution of JavaScript code within applications.

Bypassing CSP in Context:

It's worth noting that CSP can be bypassed in both context and page options. If you encounter issues with trace file generation or face challenges with JavaScript execution, consider adding the CSP bypass option in the context rather than the page.

By incorporating these strategies into your Microsoft Playwright automation workflows, you'll enhance efficiency and overcome CSP-related obstacles with ease. Stay tuned for more insightful tips and tricks as we continue to explore the exciting world of Microsoft Playwright automation with Dotnet. Happy automating!

Playwright Latest Releases & Bugs :

→ https://github.com/microsoft/playwright/releases

→ https://github.com/microsoft/playwright/issues

Playwright's getByRole is 1.5x slower than CSS selectors :

In the realm of web automation and testing with Playwright, understanding the performance of various locator strategies is key. This article delves into the `getByRole` locator's efficiency compared to CSS selectors, offering insights into the technical workings and practical implications of these choices.

I wanted to use `Page#getByRole`'s underlying CSS selectors in our codebase. But the `getByRole` locator was 1.5 times slower compared to the standard CSS selectors, prompting an investigation into the root cause. This performance discrepancy, likely stemming from Playwright's use of `querySelectorAll('*')` and matching elements by the accessible name, raises essential considerations for developers prioritizing speed in their automation scripts.

Deep Dive: How `getByRole` Works

The `getByRole` function in Playwright is more than just a method to locate web elements; it's a complex mechanism with multiple layers of interaction within the Playwright architecture. Let's demystify this process with an example. Consider this code:

```
1  await page.getByRole('button', { name: 'Enter address manually' }).click();
```

This command sets off a cascade of actions within Playwright:

- Page#getByRole creates a `Locator`
- Locator#click delegates call to `Frame#click` passing the `Locator#_selector`
- Frame#click delegates to `Channel#click`. `Frame` inherits `_channel` from `ChannelOwner`. `ChannelOwner#_channel` is a JS Proxy object based on the `EventEmitter`
- Client `Frame` dispatches an event to the server `Frame#click`
- FrameSelector#resolveInjectedForSelector injects the `FrameExecutionContext#injectedScript` script to the page, controlled by Playwright. The `InjectedScript#constructor` adds the engine for `getByRole` locator.
- createRoleEngine calls `parseAttributeSelector` and `queryRole`
- queryRole calls `querySelectorAll('*')` and matches the element

Compared to the `getByRole`, the locator with the regular CSS selector just traverses DOM and is 1.5 times faster.

Performance

Comparative tests reveal that a regular locator using CSS selectors outperforms `getByRole` by 1.5 times. Interestingly, the `$.then` method trailed, being 2x slower in our tests.

```
console.time("getByRole");
for (let i = 0; i < 100; i++) {
  const textContent = await page.getByRole('button', { name: 'Enter address manually' }
}
console.timeEnd("getByRole");

console.time("locator");
for (let i = 0; i < 100; i++) {
  const textContent = await page.locator('.ektjNL').textContent()
}
console.timeEnd("locator");

console.time("$.then");
for (let i = 0; i < 100; i++) {
  const textContent = await page.$('.ektjNL').then(e => e.textContent())
}
console.timeEnd("$.then");

// Output:
// getByRole: 677.5ms
// locator: 497.306ms
// $.then: 1.135s
```

Conclusion

In web automation, understanding Playwright's locators — `getByRole` and CSS selectors — is key. `getByRole` excels in clarity, while CSS selectors win in speed. Choose wisely: `getByRole` for testing accessibility and/or user-facing attributes of elements, CSS selectors for efficiency.

Playwright's test.slow() and slowMo :

Introduction: In the world of automated testing with Playwright, efficiency and accuracy are key. Playwright offers various features to help developers manage and debug their tests effectively. Two such features are `test.slow()` and the `slowMo` option within `launchOptions`. Although they might seem similar at first glance, they serve distinct purposes in the testing lifecycle.

Understanding `test.slow()`: `test.slow()` is a method used within Playwright Test. This method is primarily used to adjust the expectations around the duration of a test. It's particularly useful for tests that are inherently slow due to complex interactions or heavy resource usage.

```
// put yourconst { test, expect } = require('@playwright/test');

test.describe('Slow motion example', () => {
  test.slow(); // This will slow down the test

  test('Example test with slowMo', async ({ browser }) => {
    // Launch the browser with the slowMo option
    const context = await browser.newContext({ slowMo: 50 }); // slowMo value in milliseconds
    const page = await context.newPage();

    // Your test steps go here
    await page.goto('https://example.com');
    await expect(page).toHaveTitle(/Example Domain/);

    // More interactions can be added here

    // Closing the browser context
    await context.close();
  });
});
```

- `test.slow()` adjusts the timeout settings for a test.
- It is used to mark a test as slower than usual, helping avoid false negatives due to timeouts.
- It does not slow down the test's execution but rather manages the test's time expectations.

Exploring `launchOptions: { slowMo: 1000 }`: On the other hand, the `slowMo` option within `launchOptions` is a configuration setting for launching a browser. This option adds a specified delay (in milliseconds) after each browser action.

Key Points:

- `slowMo` literally slows down browser interactions.
- It is invaluable for debugging, allowing developers to observe the browser's actions step-by-step.
- The delay is applied to actions like clicks, typing, and navigation.

Comparative Analysis: While `test.slow()` is about test management, `slowMo` focuses on the pace of interaction within the browser. `test.slow()` does not change how the test runs, but rather how its duration is interpreted. Conversely, `slowMo` changes the speed of the browser's actions, making it easier to track and debug.

Practical Applications:

- Use `test.slow()` when dealing with tests that are naturally slower to prevent timeout issues.
- Apply `slowMo` when you need to observe the browser's actions in real-time for debugging.

Conclusion: Both `test.slow()` and `slowMo` are essential tools in a developer's arsenal when working with Playwright. Understanding the distinction and appropriate use cases for each can significantly enhance your testing strategy, ensuring both efficiency and accuracy in your automated testing process.

- **Playwright Fixtures & Page Object Model (POM) :**

Page Object Model (POM)

The page object model is a design pattern that helps to separate the page logic from the tests. POM is a class-based object that represents a page. It contains all the methods and properties that are used to interact with a given page.

```ts
import { Page, Locator } from '@playwright/test';

export default class LoginPage {

  readonly page: Page;

  // Locators
  readonly usernameField: Locator;
  readonly passwordField: Locator;
  readonly loginButton: Locator;


  constructor(page: Page) {
    this.page = page;

    // Locators
    this.usernameField = this.page.locator('#user-name');
    this.passwordField = this.page.locator('#password');
    this.loginButton = this.page.locator('#login-button');
  }

  async navigate(): Promise<void> {
    await this.page.goto('/');
  }

  async login(username: string, password: string): Promise<void> {
    await this.usernameField.type(username);
    await this.passwordField.type(password);
    await this.loginButton.click();
  }
}
```

Creating POM fixture

You can create your own fixture in Playwright by extending the base test and use it to create instances of your page classes - as many as you wish. Thus the instance of your page object along with its methods and properties become accessible to your tests.

```ts
import { test as base } from '@playwright/test'
import LoginPage from '../pages/login.page'
import CartPage from '../pages/cart.page'
import InventoryPage from '../pages/inventory.page'

interface Pages {
  loginPage: LoginPage;
  cartPage: CartPage;
  inventoryPage: InventoryPage;
}

export const test =
  base.extend <
  Pages >
  {
    loginPage: async ({ page }, use) => {
      await use(new LoginPage(page))
    },
    cartPage: async ({ page }, use) => {
      await use(new CartPage(page))
    },
    inventoryPage: async ({ page }, use) => {
      await use(new InventoryPage(page))
    },
  }

export const expect = test.expect
```

Before using POM Fixture

```
import { expect, test, Page } from '@playwright/test'
import LoginPage from '../pages/login.page'
import InventoryPage from '../pages/inventory.page'
import CartPage from '../pages/cart.page'

test.describe('Demo Tests', () => {
  test('Add single item to shopping cart', async ({ page }: { page: Page }) => {
    const loginPage = new LoginPage(page)
    const inventoryPage = new InventoryPage(page)
    const cartPage = new CartPage(page)

    await loginPage.navigate()
    await loginPage.login('standard_user', 'secret_sauce')
    const titleInventoryPage = await inventoryPage.getInvetoryItemTitleByIndex(
      0
    )
    await inventoryPage.addItemToCartByIndex(0)
    expect(await inventoryPage.getShoppingCartCount()).toEqual('1')
    await inventoryPage.clickShoppingCartIcon()
    expect(await cartPage.getCartItemsCount()).toEqual(1)
    expect(await cartPage.getCartItemTitleByIndex(0)).toEqual(
      titleInventoryPage
    )
  })
})
```

After using POM Fixture

```
import { expect, test } from '../fixtures/pomFixture'

test.describe('Demo Tests', () => {
  test('Add single item to shopping cart', async ({
    loginPage,
    inventoryPage,
    cartPage,
  }) => {
    await loginPage.navigate()
    await loginPage.login('standard_user', 'secret_sauce')
    const titleInventoryPage = await inventoryPage.getInvetoryItemTitleByIndex(
      0
    )
    await inventoryPage.addItemToCartByIndex(0)
    expect(await inventoryPage.getShoppingCartCount()).toEqual('1')
    await inventoryPage.clickShoppingCartIcon()
    expect(await cartPage.getCartItemsCount()).toEqual(1)
    expect(await cartPage.getCartItemTitleByIndex(0)).toEqual(
      titleInventoryPage
    )
  })
})
```

Playwright Tricky UI Automation Scenarios :

Alerts

We will start with Alerts. You have probably seen them many times before.

Alert Box

The first is the simple “Alert Box”. We can just close it and nothing more.
How do you do that in Playwright? Quite simply

All we need to do is subscribe to the ‘dialogue’ event and call ‘dismiss’ from it once this event has been triggered. In our case we trigger it with the click.

> *page.on(‘dialog’, async dialog => await dialog.dismiss());*

```
test('Click on JS Alert button - Should display alert', async ({ page, javaScriptAlertsPage }) => {
    //Arrange
    page.on('dialog', async dialog => await dialog.dismiss());
    const expectedText = 'You successfully clicked an alert';

    //Act
    await javaScriptAlertsPage.clickForJsAlert();

    //Assert
    expect(await javaScriptAlertsPage.getResultText()).toBe(expectedText);
});
```

Confirm Box

The next is “Confirm Box” with “OK” and “Cancel” buttons.

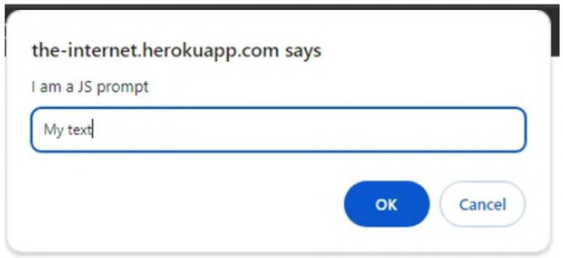
The idea is quite similar. Subscribe on ‘dialog’ and call “accept” or “dismiss” methods

> *page.on(‘dialog’, async dialog => await dialog.accept());*

```
test('Click on JS Confirm button and confirm - Should display alert', async ({ page, javaScriptAlertsPage }) => {
    //Arrange
    page.on('dialog', async dialog => await dialog.accept());
    const expectedText = 'You clicked: Ok';

    //Act
    await javaScriptAlertsPage.clickForJsConfirm();

    //Assert
    expect(await javaScriptAlertsPage.getResultText()).toBe(expectedText);
});
```

Prompt Box

The last item on the list is the “Prompt Box”. In addition to the previous box, it contains an “Input” field where you can add a text.



But it is not very difficult. Subscribe again and call an accept method with a text you want to pass inside the input field.

> *page.on(‘dialog’, async dialog => await dialog.accept(‘Test’));*

IFrames

Yes. We all hate it a bit. Some less and some more :)

But we definitely face it and we have to solve it. So we have a page with some frames

- If we look inside of the HTML markup, we may my two top level Iframes(**frame_top** and **frame_bottom**).
- Inside of the **frame_top** we also have 3 nested frames (**frame_left**, **frame_middle** and **frame_right**)

Let's try to get text from each of these frames.

- Finding top-level frame locators. We need to use a page and a "frameLocator" method. The parameter of this method is the selector of the frame.

> *this.frameTop = page.frameLocator('frame[name="frame-top"]');*
>
> *this.frameBottom = page.frameLocator('frame[name="frame-bottom"]');*

- Now. We can find nested frames for **frame_top** frame. We just need to call "frameLocator" method again for the found top level locator. And will chain it.

> *this.innerTopLeftFrame = this.frameTop.frameLocator('frame[name="frame-left"]');*
>
> *this.innerTopRightFrame = this.frameTop.frameLocator('frame[name="frame-right"]');*
>
> *this.innerTopMiddleFrame = this.frameTop.frameLocator('frame[name="frame-middle"]');*

- Finally we create a methods that gets a text using already prepared frame locators

Top level frame

> *async getFrameBottomText(): Promise<string> {*
>
> *return await this.frameBottom.locator('body').innerText();*
>
> *}*

Nested frame

> *async getInnerTopLeftFrameText(): Promise<string> {*
>
> *return await this.innerTopLeftFrame.locator('body').innerText();*
>
> *}*

```
export default class NestedFramesPage extends BasePage {
    frameTop: FrameLocator;
    frameBottom: FrameLocator;
    innerTopLeftFrame: FrameLocator;
    innerTopRightFrame: FrameLocator;
    innerTopMiddleFrame: FrameLocator;

    constructor(page: Page) {
        super(page);
        this.frameTop = page.frameLocator('frame[name="frame-top"]');
        this.frameBottom = page.frameLocator('frame[name="frame-bottom"]');

        this.innerTopLeftFrame = this.frameTop.frameLocator('frame[name="frame-left"]');
        this.innerTopRightFrame = this.frameTop.frameLocator('frame[name="frame-right"]');
        this.innerTopMiddleFrame = this.frameTop.frameLocator('frame[name="frame-middle"]');
    }

    async getFrameBottomText(): Promise<string> {
        return await this.frameBottom.locator('body').innerText();
    }

    async getInnerTopLeftFrameText(): Promise<string> {
        return await this.innerTopLeftFrame.locator('body').innerText();
    }

    async getInnerTopRightFrameText(): Promise<string> {
        return await this.innerTopRightFrame.locator('body').innerText();
    }

    async getInnerTopMiddleFrameText(): Promise<string> {
        return await this.innerTopMiddleFrame.locator('body').innerText();
    }
}
```

We can call these methods in the tests and make sure they pass.

```
test('Nested Frames sub-page - Get text from different IFrames', async ({ framesPage, page }) => {
    //Arrange
    const nestedFramesPage: NestedFramesPage = await framesPage.navigateToNestedFramesPage();

    //Act
    const textFromFrameBottom: string = await nestedFramesPage.getFrameBottomText();
    const textFromInnerTopLeftFrame: string = await nestedFramesPage.getInnerTopLeftFrameText();
    const textFromInnerTopRightFrame: string = await nestedFramesPage.getInnerTopRightFrameText();
    const textFromInnerTopMiddleFrame: string = await nestedFramesPage.getInnerTopMiddleFrameText();

    //Assert
    expect.soft(textFromFrameBottom).toEqual('BOTTOM');
    expect.soft(textFromInnerTopLeftFrame).toEqual('LEFT');
    expect.soft(textFromInnerTopRightFrame).toEqual('RIGHT');
    expect.soft(textFromInnerTopMiddleFrame).toEqual('MIDDLE');
});
```

As you can see, it is not that difficult. What I like is that you don't have to go in and out of the frames as you do with WebDriver.

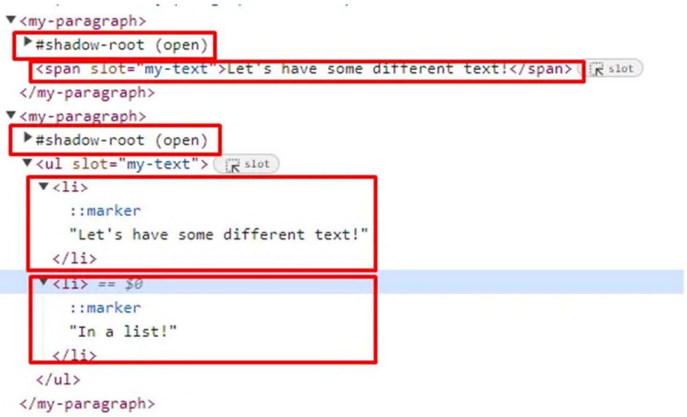
Shadow Dom

Also one of the most popular features on the list. But we will beat it too.

But first. What is Shadow Dom?

The Shadow DOM is a web standards technology designed to help web developers encapsulate their code. The primary benefit is encapsulation: styles and scripts defined inside the shadow DOM won't conflict with the styles and scripts in the main document or other shadow DOMs.

- Here we have a two simple paragraphs. Each of these paragraphs have 'shadow-root'. In a first case it is just a 'span' and in second it is a list 'ul' with two items 'li'.



So let's try to read these texts. All we need to do is take the component name and the rest of the selector

First paragraph span text:

> *async getFirstParagraphText(): Promise<string> {*
>
> *return await this.page.locator('my-paragraph span').textContent() as string;*
>
> *}*

Second paragraph list text:

> *async getSecondParagraphText(): Promise<string[]> {*
>
> *return await this.page.locator('my-paragraph ul li').allInnerTexts() as string[];*
>
> *}*

```
export default class ShadowDomPage extends BasePage {
    constructor(page: Page) {
        super(page);
    }

    async getFirstParagraphText(): Promise<string> {
        return await this.page.locator('my-paragraph span').textContent() as string;
    }

    async getSecondParagraphText(): Promise<string[]> {
        return await this.page.locator('my-paragraph ul li').allInnerTexts() as string[];
    }
}
```

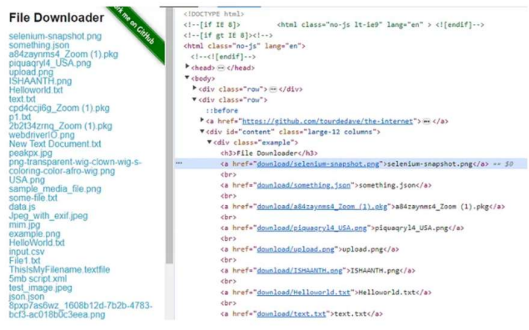
Let's call this method and check inside the test

```
test.describe.parallel('Shadow Dom page tests - Verify page logic', async () => {
    test.beforeEach(async ({ mainPage, baseURL }) => {
        await mainPage.navigateUrl(new PageDataConstants(baseURL as string).shadowDomPage.getUrlPath());
        await mainPage.waitForPageLoad();
    });

    test('Verify shadow dom', async ({ shadowDomPage }) => {
        //Arrange
        const expectedFirstParagraphText = `Let's have some different text!`;
        const expectedSecondParagraphText = [`Let's have some different text!`, `In a list!`]

        //Act
        const firstParagraphText: string = await shadowDomPage.getFirstParagraphText();
        const secondParagraphText: string[] = await shadowDomPage.getSecondParagraphText();

        //Assert
        expect.soft(firstParagraphText).toBe(expectedFirstParagraphText);
        expect.soft(secondParagraphText).toEqual(expectedSecondParagraphText);
    });
});
```

As you can see, this is even easier than using IFrames.

File Download

This is a more interesting case. Maybe for me :) We want to download a file using a href link.



What we need to do is to break this logic down into a few parts

- First we need to initialize the download and wait for the download event.

> *const [download] = await Promise.all([*
>
> *this.page.waitForEvent('download'), // Wait for the download event*
>
> *this.page.click(`a[href="download/${expectedFileName}"]`) // Click the download link*
>
> *]);*

- Secondly, we need to save our file

> *await download.saveAs(savePath);*

```
async downloadFile(expectedFileName: string, savePath: string) {
    // Start download
    const [download] = await Promise.all([
        this.page.waitForEvent('download'), // Wait for the download event
        this.page.click(`a[href="download/${expectedFileName}"]`) // Click the download link
    ]);

    await download.saveAs(savePath);

    return download;
}
```

The method is ready. Now let's call it in the test.

- Within the test, we have prepared a filename. Because we will use it in our selector.
- And then we need to provide a save path

> *const expectedFileName = 'USA.png';*
>
> *const downloadFolderPath = path.resolve(__dirname, `../test-data`); // Update this path*
>
> *const savePath = path.join(downloadFolderPath, expectedFileName);*

- The next step is simply to call the prepared method and assert

> *await downloadFilePage.downloadFile(expectedFileName, savePath);*
>
> *expect(fs.existsSync(savePath)).toBeTruthy();*

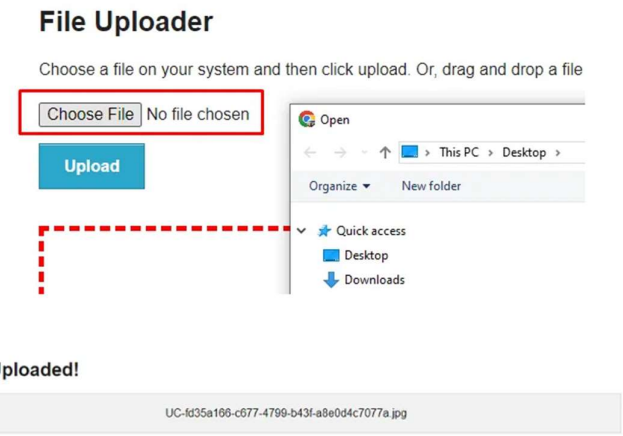
- And of course it is good practice to clean up after yourself :)

> *// Clean up: remove downloaded file*
>
> *fs.unlinkSync(savePath);*

```
test.describe.parallel('File Download page tests - Verify page logic', async () => {
    test.beforeEach(async ({ mainPage, baseURL }) => {
        await mainPage.navigateUrl(new PageDataConstants(baseURL as string).fileDownloadPage.getUrlPath());
        await mainPage.waitForPageLoad();
    });

    test('Download file', async ({ downloadFilePage }) => {
        // Arrange
        const expectedFileName = 'USA.png';
        const downloadFolderPath = path.resolve(__dirname, `../test-data`); // Update this path
        const savePath = path.join(downloadFolderPath, expectedFileName);

        // Act
        await downloadFilePage.downloadFile(expectedFileName, savePath);

        // Assert
        expect(fs.existsSync(savePath)).toBeTruthy();

        // Clean up: remove downloaded file
        fs.unlinkSync(savePath);
    });
});
```

File Upload

Now we want to do the opposite. We want to upload a file



File Uploaded!

UC-fd35a166-c677-4799-b43f-a8e0d4c7077a.jpg

How to do it? In the case of this training site implementation, we need to perform 2 operations:

1. Upload a file from the specified location using the setInputFiles method. As you can see, it takes the selector and the local file location path.

> *await this.page.setInputFiles('#file-upload', filePath);*

2. We need to click the Upload button to confirm the operation

> *await this.page.getByRole('button', { name: 'Upload' }).click();*

Additionally: I created a method that takes the uploaded file name on the file upload screen.

> *public async getUploadedFileName(): Promise<string> {*
>
> *return await this.page.textContent('#uploaded-files').then(text => text?.trim()) as string;*
>
> *}*

```
export default class UploadFilePage extends BasePage {
    constructor(page: Page) {
        super(page);
    }

    public async uploadFile(filePath: string): Promise<UploadFilePage> {
        if (!fs.existsSync(filePath)) {
            throw new Error(`File not found: ${filePath}`);
        }

        await this.page.setInputFiles('#file-upload', filePath);
        await this.page.getByRole('button', { name: 'Upload' }).click();

        return this;
    }

    public async getUploadedFileName(): Promise<string> {
        return await this.page.textContent('#uploaded-files').then(text => text?.trim()) as string;
    }
```

As usual, we need to call it from within the test.

1. Before you do this, you need to create a directory and put a file in it. You could potentially create this file during the test run. But I decided not to make it too complicated for our example
2. Next we need to resolve the file path. I have also defined 'fileName' because I want to check that it is at the end.

> *const fileName = 'upload_file.txt';*
>
> *const filePath = path.resolve(__dirname, `../test-data/${fileName}`);*

3. Finally, we can call an upload method that we prepared earlier

> *await uploadFilePage.uploadFile(filePath);*

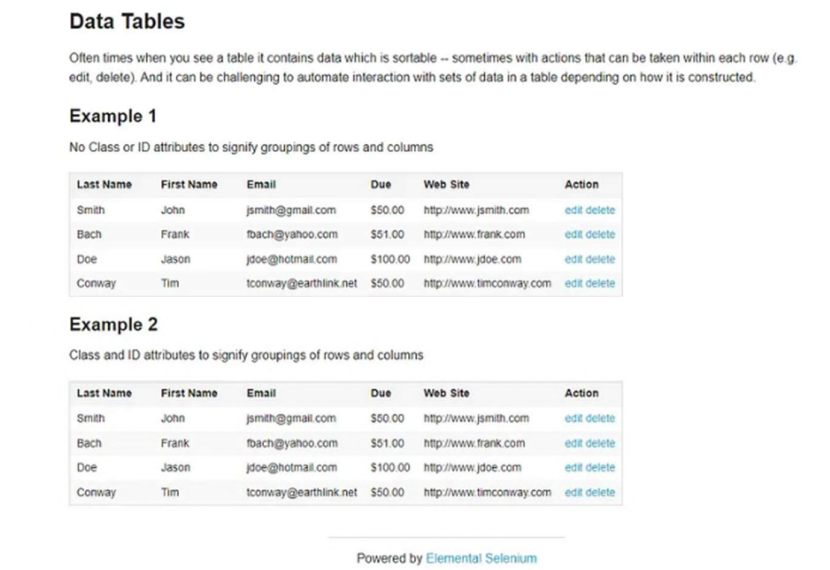
4. And check that the file has been uploaded

> *expect(await uploadFilePage.getUploadedFileName()).toBe(fileName);*

```
test.describe.parallel('File Upload page tests - Verify page logic', async () => {
    test.beforeEach(async ({ mainPage, baseURL }) => {
        await mainPage.navigateUrl(new PageDataConstants(baseURL as string).fileUploadPage.getUrlPath());
        await mainPage.waitForPageLoad();
    });

    test('Upload file', async ({ uploadFilePage }) => {
        //Arrange
        const fileName = 'upload_file.txt';
        const filePath = path.resolve(__dirname, `../test-data/${fileName}`); //File should be in the specified folder

        //Act
        await uploadFilePage.uploadFile(filePath);

        //Assert
        expect(await uploadFilePage.getUploadedFileName()).toBe(fileName);
    });
});
```

Tables

And last but not least. Yes, tables. I have heard questions about how to parse them, even from experienced people. Today we will only look at a table with static fields. Yes, they are easier to parse. You always know which header is coming next.



Data Tables

Often times when you see a table it contains data which is sortable -- sometimes with actions that can be taken within each row (e.g. edit, delete). And it can be challenging to automate interaction with sets of data in a table depending on how it is constructed.

Example 1

No Class or ID attributes to signify groupings of rows and columns

Last Name	First Name	Email	Due	Web Site	Action
Smith	John	jsmith@gmail.com	$50.00	http://www.jsmith.com	edit delete
Bach	Frank	fbach@yahoo.com	$51.00	http://www.frank.com	edit delete
Doe	Jason	jdoe@hotmail.com	$100.00	http://www.jdoe.com	edit delete
Conway	Tim	tconway@earthlink.net	$50.00	http://www.timconway.com	edit delete

Example 2

Class and ID attributes to signify groupings of rows and columns

Last Name	First Name	Email	Due	Web Site	Action
Smith	John	jsmith@gmail.com	$50.00	http://www.jsmith.com	edit delete
Bach	Frank	fbach@yahoo.com	$51.00	http://www.frank.com	edit delete
Doe	Jason	jdoe@hotmail.com	$100.00	http://www.jdoe.com	edit delete
Conway	Tim	tconway@earthlink.net	$50.00	http://www.timconway.com	edit delete

Powered by Elemental Selenium

There are several ways to do it. But I still prefer the method I use for WebDriver.

- First, we will create a model (interface with a table row names)

> *export default interface ExampleOneTableModel {*
>
> *lastName: string | null;*
>
> *firstName: string | null;*
>
> *email: string | null;*
>
> *due: string | null;*
>
> *webSite: string | null;*
>
> *}*

```
export default interface ExampleOneTableModel {
    lastName: string | null;
    firstName: string | null;
    email: string | null;
    due: string | null;
    webSite: string | null;
}
```

- The second step is the implementation of the method logic. In short, we can divide it into two parts

1. We need to find a line. Just do what we normally do. Not nothing:) Find a locator.

> *const tableRows = this.page.locator(this.tableRowsSelector);*
>
> *const rowCount = await tableRows.count();*

2. We want to use the for operator to fill cells for each row.

a) So we take the cells for the row using the “td” selector.

b) We know exactly how many cells we have and what each cell number belongs to. So we use indexes to get values and assign them to the appropriate cell.

```
async getTableData(): Promise<ExampleOneTableModel[]> {
    const tableData: ExampleOneTableModel[] = [];

    // Wait for table rows
    const tableRows = this.page.locator(this.tableRowsSelector);
    const rowCount = await tableRows.count();

    for (let rowIndex = 0; rowIndex < rowCount; rowIndex++) {
      const row = tableRows.nth(rowIndex);
      const cells = row.locator('td');

      const model: ExampleOneTableModel = {
        lastName: await cells.nth(0).textContent(),
        firstName: await cells.nth(1).textContent(),
        email: await cells.nth(2).textContent(),
        due: await cells.nth(3).textContent(),
        webSite: await cells.nth(4).textContent()
      };

      tableData.push(model);
    }

    return tableData;
}
```

We want to call a method we have prepared.

> *const tableResults: ExampleOneTableModel[] = await (await dataTablesPage.tableOne.sortTableByField(TableHeaderNames.Email)).getTableData();*

Note: I also have a method that sorts the results by a specified field. This is more related to the random results returned by the table in it's default state. I will provide a repo at the end of this article and you may find this implementation too :)

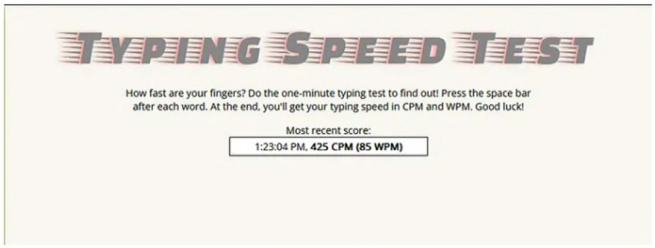
```
import { test, expect } from "../fixture-extention";
import { TableHeaderNames } from "../pages/data-tables-page/fragements/example-one-table-fragment";
import ExampleOneTableModel from "../pages/data-tables-page/models/example-one-table-model";
import PageDataConstants from "../pages/pages-constants";

test.describe.parallel('Sortable tables page tests - Verify page logic', async () => {
    test.beforeEach(async ({ mainPage, baseURL }) => {
        await mainPage.navigateUrl(new PageDataConstants(baseURL as string).sortableDataTablesPage.getUrlPath());
        await mainPage.waitForPageLoad();
    });

    test('Verify table 1 can be sorted by email', async ({ dataTablesPage }) => {
        //Arrange
        const expectedContent: ExampleOneTableModel[] = [
            { lastName: "Conway", firstName: "Tim", email: "tconway@earthlink.net", due: "$50.00", webSite: "http://www.timconway.com" },
            { lastName: "Smith", firstName: "John", email: "jsmith@gmail.com", due: "$50.00", webSite: "http://www.jsmith.com" },
            { lastName: "Doe", firstName: "Jason", email: "jdoe@hotmail.com", due: "$100.00", webSite: "http://www.jdoe.com" },
            { lastName: "Bach", firstName: "Frank", email: "fbach@yahoo.com", due: "$51.00", webSite: "http://www.frank.com" }
        ].sort((a, b) => a.email.localeCompare(b.email));

        //Act
        const tableResults: ExampleOneTableModel[] =  await (await dataTablesPage.tableOne.sortTableByField(TableHeaderNames.Email)).getTableData();

        //Assert
        expect(tableResults).toEqual(expectedContent);
    });
});
```

Automate Speed Typing With Playwright :

This simple script to write all the words from a page will pass the words and count the entire list of words and print each word typed. At the end, the test will check a simple assertion whether the message is printed for your specific score that you achieved.

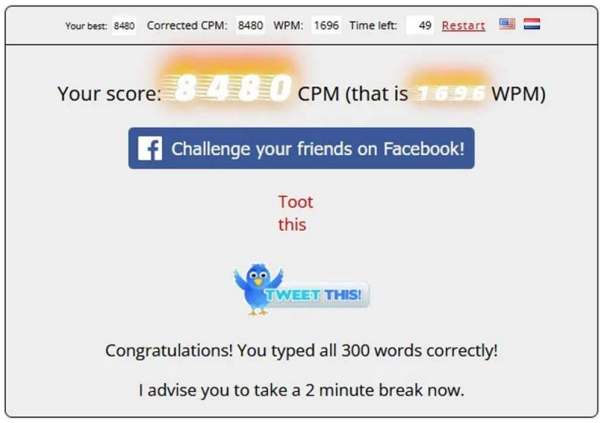
```
19-Speed-Typing-Test.spec.ts M
e2e > 19-Speed-Typing-Test.spec.ts > ...
Click here to ask Blackbox to help you code faster
import { expect, test } from '@playwright/test'

test('Automate Speed Typewriting', async ({ page }) => {
  // Marking this test case as slow to increase its timeout
  test.slow()
  await page.goto('https://typing-speed-test.aoeu.eu/')
  // Waiting for the input area to appear on the page
  await page.waitForSelector('#input')

  // Counting the total number of words present on the page
  const totalWords = (await page.$$('.nextword')).length
  console.log(`Total words: ${totalWords}`)

  // Iterating through each word on the page
  for (let i = 0; i <= totalWords; i++) {
    // Finding the current word element on the page
    const wordElement = await page.locator('#currentword')
    // Extracting the text of the current word
    const word = await wordElement.innerText()

    // Typing the current word into the input area and pressing Space
    await page.fill('#input', word)
    await page.keyboard.press('Space')
    // Logging the word that was typed
    console.log(`Typed word ${i + 1}: ${word}`)
  }
  // Finding the final message element which contains the test result
  const finalMessage = await page.locator('#result')

  // Verifying that the final message contains the expected format of the test result
  await expect(finalMessage).toContainText(
    /Your score: \d+ CPM \(that is \d+ WPM\)/
  )
  // Verifying that the final message is visible on the page
  await finalMessage.isVisible()
})
```

Average Score:



Average people score

Score with Playwright:



Playwright score

- **Advanced Snapshot Testing in Playwright :**

→ https://mikestopcontinues.hashnode.dev/advanced-snapshot-testing-in-playwright?utm_source=substack&utm_medium=email

- **NPM and Node Packages :**

→ https://medium.com/@oroz.askarov/all-you-need-to-know-about-npm-and-packages-as-a-beginner-b6fcea8b3519

- **Keyboard actions :** https://playwright.dev/docs/api/class-keyboard
- **Mouse Actions :** https://playwright.dev/docs/input#mouse-click
- **Test Recording :** https://playwright.dev/docs/codegen
- **Recording Videos :** https://playwright.dev/docs/videos
- **Take Screenshots :** https://playwright.dev/docs/screenshots
- **Tag Tests :** https://playwright.dev/docs/test-annotations#tag-tests

- Bug for this concept : https://github.com/microsoft/playwright/issues/9667

Integrate the Playwright tests with Jenkins :

Step 1 : Download Jenkins Server:

- Download Generic Java Package(.war) from https://www.jenkins.io/download/ & store inside same directory.

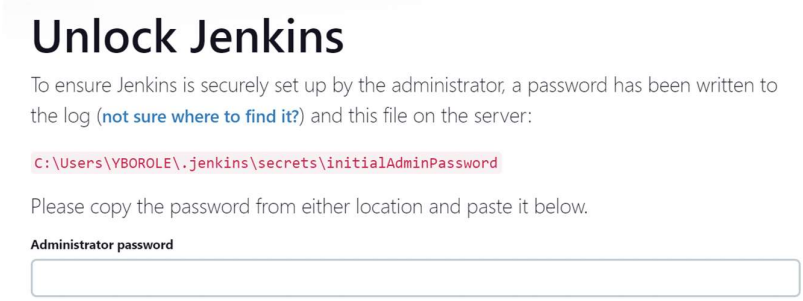
Step 2 : Run Jenkins Server:

- Now open command prompt or in vs code terminal & navigate to the path where your Jenkins.jar file located.

- Type Command : java -jar jenkins.war --httpPort=4080 --enable-future-java → To start the Jenkins on localhost 4080

- Open localhost 4080 on browser & give password displayed on terminal.

```
Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:

1bbe191274584879824 91a969fb35c39
```



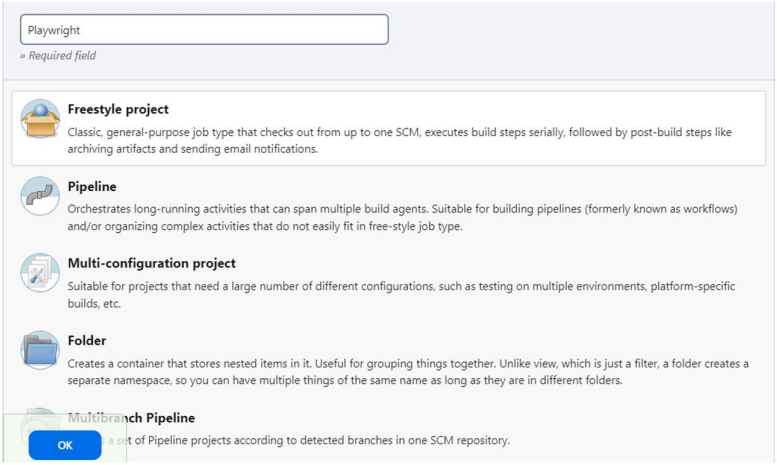
- Click on Install Suggested Plugin

- Skip and continue as admin

- enter localhost link & save & then start

Step 3 : Create Jenkins Build:

- New Item → Enter Name → Freestyle Project → Ok

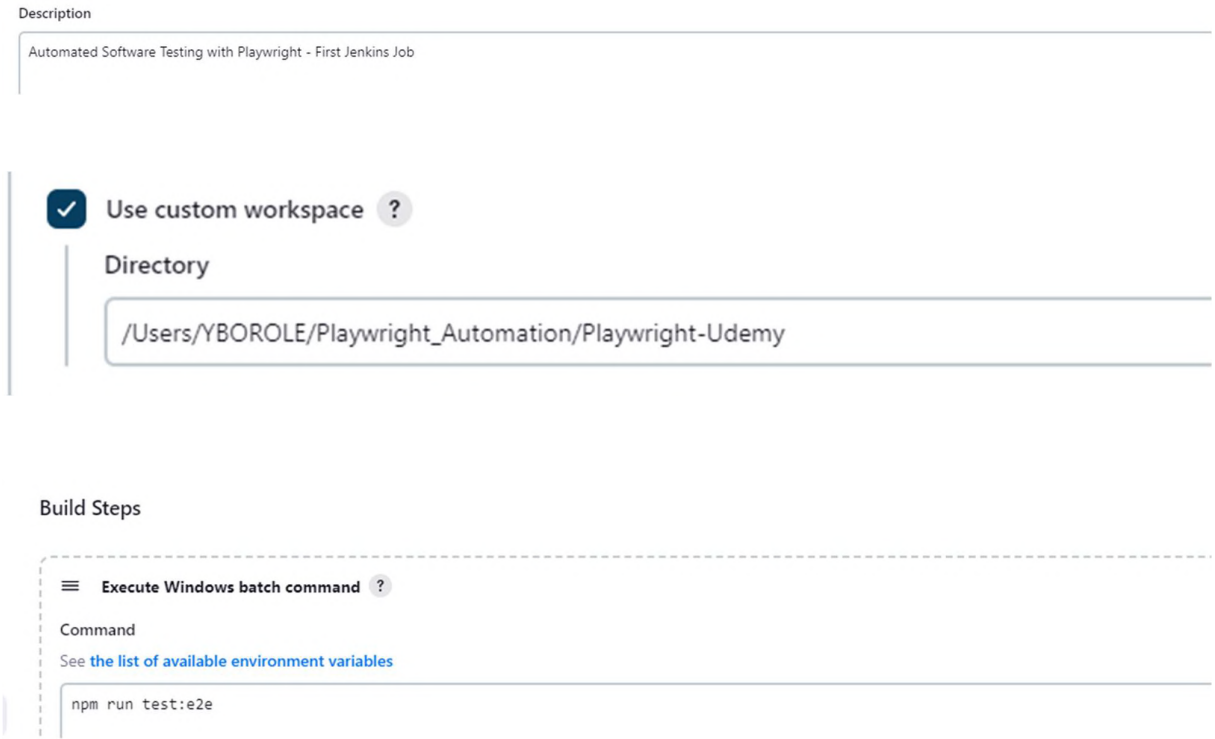


- Enter description → Advanced → use custom workspace

- If you have git repo then use that otherwise if local then choose option as : Use custom workspace → Enter Directory for your code → e.g. /Users/YBOROLE/Playwright_Testing/Playwright-Udemy

- Below Add Build Step → For window choose Execute window bash command & for Mac choose Execute shell

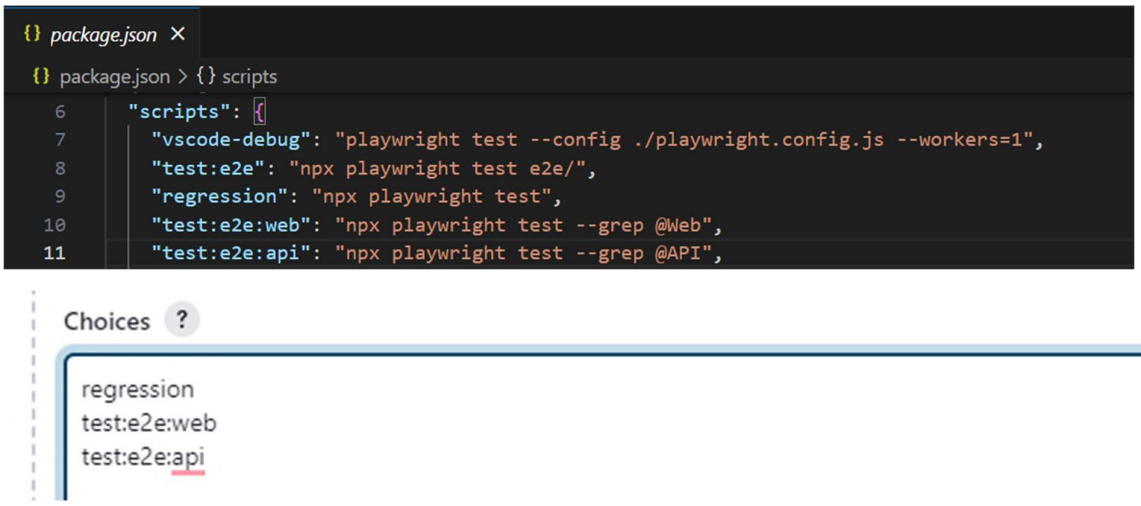
- Add run command & Save



- Once saved we can see Build Now option, click on that & schedule job → you are able to see everything like logs passed/failed etc.

Step 4 : Parameterized Jenkins Build:

- Go to Configure → Click on This project is Parameterized option → Choose Choice Parameter → Give Name as a script & description as any e.g. select name of the node script from your package.json & enter Choices as regression or .. OR regression \n webTests \n apiTests ..etc →



& then Inside window bash command → add only npm run "%script%" OR npm run "$script" → Save

- Now you are able to see Build with Parameters → Choose specific options → Build

Step 4 : Jenkins Server Node Script:

package.json M

```
{} package.json > ...
  6   "scripts": {
  7     "jenkins-server": "java -jar jenkins.war --httpPort=4080 --enable-future-java",
```

- For Run Purpose give command in cmd as : npm run jenkins-server

---

■ Playwright Advanced Tricks & Tips :

e2e-tips-and-tricks.spec.ts U

```
e2e > e2e-tips-and-tricks.spec.ts > ...
      Click here to ask Blackbox to help you code faster
  1  import { expect, test } from "@playwright/test";
  2
  3  test.describe("Advanced Tips & Tricks Section", () => {
  4    test("Test Info Object", async ({ page }, testInfo) => {
  5      await page.goto("http://www.example.com");
  6      console.log(testInfo);
  7      console.log(testInfo.title); // TestInfo Object
  8      console.log(testInfo.status); // passed
  9    });
 10
 11    test("Test Skip Browser Annotation", async ({ page, browserName }) => {
 12      // Skip the test if the browser is Firefox
 13      test.skip(
 14        browserName === "firefox", // Condition to skip the test
 15        "Feature not ready to run inside firefox browser" // Reason for skipping the test
 16      );
 17
 18      // Proceed with test execution if the browser is not Firefox
 19      await page.goto("http://www.example.com");
 20    });
 21
 22    test("Test Fixme Annotation", async ({ page, browserName }) => {
 23      // Mark the test as needing revision if the browser is Firefox
 24      test.fixme(
 25        browserName === "firefox", // Condition to mark the test as needing revision
 26        "Test is not stable, needs revision" // Reason for marking the test as needing revision
 27      );
 28
 29      // Proceed with test execution regardless of browser
 30      await page.goto("http://www.example.com");
 31    });
```

e2e-tips-and-tricks.spec.ts U

```
e2e > e2e-tips-and-tricks.spec.ts > ...
  3  test.describe("Advanced Tips & Tricks Section", () => {
 33    // Parametrized Tests :
 34    const peoples = ["Yogesh", "Sama", "Dhivya", "Akshitha", "Varun"];
 35    for (const name of peoples) {
 36      test(`Running test for: ${name}`, async ({ page }) => {
 37        await page.goto("http://zero.webappsecurity.com/");
 38        await page.fill("#searchTerm", `${name}`);
 39        await page.keyboard.press("Enter");
 40        await expect(page.locator("div[class='top_offset'] h2")).toContainText(
 41          "Search Results:"
 42        );
 43      });
 44    }
```

e2e-tips-and-tricks.spec.ts U

e2e > e2e-tips-and-tricks.spec.ts > ...

```
3   test.describe("Advanced Tips & Tricks Section", () => {
46    test("Mouse Movement Simulation", async ({ page }) => {
47      await page.goto("http://www.example.com");
48      await page.mouse.move(0, 0);
49      await page.mouse.down();
50      await page.mouse.move(0, 100);
51      await page.mouse.up();
52    });
53
54    test("Multiple Browser Tabs inside one browser", async ({ browser }) => {
55      const context = await browser.newContext();
56      const page1 = await context.newPage();
57      await page1.goto("http://www.example.com");
58      const page2 = await context.newPage();
59      await page2.goto("http://www.example.com");
60      await page1.waitForTimeout(3000);
61    });
62
63    test("Data Helpers - Get Random Number & String", async ({ browser }) => {
64      const randomNumber = await getRandomNumber();
65      console.log(randomNumber);
66
67      const randomString = await getRandomString(5);
68      console.log(randomString);
69    });
70  });
```

e2e-tips-and-tricks.spec.ts U

e2e > e2e-tips-and-tricks.spec.ts > ...

```
71
72  // Generates a random number between 1 and 1000 (inclusive).
73  async function getRandomNumber() {
74    return Math.floor(Math.random() * 1000 + 1);
75  }
76
77  async function getRandomString(length) {
78    const characters =
79      "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
80    let result = "";
81
82    for (let i = 0; i < length; i++) {
83      const randomIndex = Math.floor(Math.random() * characters.length);
84      result += characters.charAt(randomIndex);
85    }
86
87    return result;
88  }
89
90  // Device Emulation : npx playwright open --device="iPhone 11" wikipedia.org
91  // Generate PDF Files : npx playwright pdf http://www.example.com example-file.pdf
92  // Generate Customized Screenshots : npx playwright screenshot --device="iPhone 11" --color-scheme=dark --wait-for-timeout=3000 http://www.example.com example-img.png
93  // Emulate Browser Language & Timezone : npx playwright open --lang="en-US" --timezone="Asia/Kolkata" http://www.example.com
94
```

■ Playwright Professional Custom Reporting :

test-result.json U

{} test-result.json > ...

Click here to ask Blackbox to help you code faster

```
1  {
2    "test": "Positive Scenario for login & logout",
3    "status": "passed",
4    "executionTime": 8589,
5    "errors": []
6  }
```

TS custom-reporter.ts U

TS custom-reporter.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import {
  Reporter,
  FullConfig,
  Suite,
  FullResult,
} from '@playwright/test/reporter';
import * as fs from 'fs';

class MyReporter implements Reporter {
  onBegin(config: FullConfig<{}, {}>, suite: Suite) {
    console.log(`Execution of ${suite.allTests().length} tests`);
  }

  onEnd(result: FullResult) {
    console.log(`Execution finished with status of ${result.status}`);
  }

  onTestBegin(test) {
    console.log(`Execution of ${test.title} started.`);
  }

  onTestEnd(test, result) {
    const execTime = result.duration;

    const data = {
      test: test.title,
      status: result.status,
      executionTime: execTime,
      errors: result.errors,
    };

    const dataToString = JSON.stringify(data, null, 2);
    console.log(dataToString);

    fs.writeFileSync('test-result.json', dataToString);
  }
}

export default MyReporter;

// npx playwright test --reporter=custom-reporter.ts
// npx playwright test e2e/e2e-login.spec.ts --reporter=custom-reporter.ts
```

Playwright API tests :

e2e-api.spec.ts U

e2e > e2e-api.spec.ts > ...

Click here to ask Blackbox to help you code faster

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

test.describe.parallel('API Testing', () => {
  const baseUrl = 'https://reqres.in/api';

  test('@API assert response status', async ({ request }) => {
    const response = await request.get(`${baseUrl}/users/2`);
    expect(response.status()).toBe(200);

    const responseBody = JSON.parse(await response.text());
    console.log(responseBody);
  });

  test('@API assert invalid endpoint', async ({ request }) => {
    const response = await request.get(
      `${baseUrl}/users/non-existing-endpoint`,
    );

    expect(response.status()).toBe(404);
  });

  test('@API Get Request - get user details', async ({ request }) => {
    const response = await request.get(`${baseUrl}/users/1`);
    const responseBody = JSON.parse(await response.text());

    expect(response.status()).toBe(200);
    expect(responseBody.data.id).toBe(1);
    expect(responseBody.data.email).toBe('george.bluth@reqres.in');
    expect(responseBody.data.first_name).toBe('George');
    expect(responseBody.data.last_name).toBe('Bluth');
  });
```

e2e-api.spec.ts U

e2e > e2e-api.spec.ts > ...

```
3   test.describe.parallel('API Testing', () => {
33    test('@API Post Request - create new user', async ({ request }) => {
34      const response = await request.post(`${baseUrl}/users`, {
35        data: {
36          id: 46171542,
37          name: 'Yogesh Borole',
38          job: 'Software Engineer',
39        },
40      });
41      const responseBody = JSON.parse(await response.text());
42      console.log(responseBody);
43
44      expect(responseBody.id).toBe(46171542);
45      expect(responseBody.createdAt).toBeTruthy();
46    });
47
48    test('@API Post Request - login successful', async ({ request }) => {
49      const response = await request.post(`${baseUrl}/login`, {
50        data: {
51          email: 'eve.holt@reqres.in',
52          password: 'cityslicka',
53        },
54      });
55      const responseBody = JSON.parse(await response.text());
56
57      expect(response.status()).toBe(200);
58      expect(responseBody.token).toBeTruthy();
59    });
60
61    test('@API Post Request - login fail', async ({ request }) => {
62      const response = await request.post(`${baseUrl}/login`, {
63        data: {
64          email: 'peter@klaven',
65        },
66      });
67      const responseBody = JSON.parse(await response.text());
68
69      expect(response.status()).toBe(400);
70      expect(responseBody.error).toBe('Missing password');
71    });
```

e2e-api.spec.ts U

e2e > e2e-api.spec.ts > ...

```
3   test.describe.parallel('API Testing', () => {
73    test('@API Put Request - update user', async ({ request }) => {
74      const response = await request.put(`${baseUrl}/users/2`, {
75        data: {
76          name: 'Sama Dharma',
77          job: 'Angular Developer',
78        },
79      });
80      const responseBody = JSON.parse(await response.text());
81
82      expect(response.status()).toBe(200);
83      expect(responseBody.name).toBe('Sama Dharma');
84      expect(responseBody.job).toBe('Angular Developer');
85      expect(responseBody.updatedAt).toBeTruthy();
86    });
87
88    test('@API Delete Request - delete user', async ({ request }) => {
89      const response = await request.delete(`${baseUrl}/users/2`);
90      expect(response.status()).toBe(204);
91    });
92  });
93
```

Playwright Web UI tests :

e2e-currency-exchange.spec.ts U

e2e > e2e-currency-exchange.spec.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import { test, expect } from "@playwright/test";

test.describe("Currency Exchange Form", () => {
  // Before Hook
  test.beforeEach(async ({ page }) => {
    await page.goto("http://zero.webappsecurity.com/");
    await page.click("#signin_button");
    await page.fill("#user_login", "username");
    await page.fill("#user_password", "password");
    await page.getByText("Sign in").click();

    await page.goBack();
    await page.click("#onlineBankingMenu");
    await page.click("#account_summary_link");
  });

  test("should make currency exchange", async ({ page }) => {
    await page.click("#pay_bills_tab");
    await page.click("a[href='#ui-tabs-3']");
    await page.selectOption("#pc_currency", "CHF"); // select Switzerland (franc)

    const todaysSellRate = await page.locator("#sp_sell_rate");
    await expect(todaysSellRate).toContainText(
      "1 franc (CHF) = 1.1383 U.S. dollar (USD)"
    );
    await page.fill("#pc_amount", "5000");
    await page.click("#pc_inDollars_true");
    await page.click("#pc_calculate_costs");

    const convertedAmount = await page.locator("#pc_conversion_amount");
    await expect(convertedAmount).toContainText(
      "4392.52 franc (CHF) = 5000.00 U.S. dollar (USD)"
    );

    await page.click("#purchase_cash");

    const successMessage = await page.locator("#alert_content");
    await expect(successMessage).toBeVisible();
    await expect(successMessage).toContainText(
      "Foreign currency cash was successfully purchased."
    );
  });
});
```

e2e-filter-transactions.spec.ts U

e2e > e2e-filter-transactions.spec.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import { test, expect } from "@playwright/test";

test.describe("Filter Transactions", () => {
  // Before Hook
  test.beforeEach(async ({ page }) => {
    await page.goto("http://zero.webappsecurity.com/");
    await page.click("#signin_button");
    await page.fill("#user_login", "username");
    await page.fill("#user_password", "password");
    await page.getByText("Sign in").click();

    await page.goBack();
    await page.click("#onlineBankingMenu");
    await page.click("#account_summary_link");
  });

  test("verify the results for each accounts", async ({ page }) => {
    await page.click("#account_activity_tab");
    await page.selectOption("#aa_accountId", "3");
    const savingAccount = await page.locator(
      "#all_transactions_for_account tbody tr"
    );
    await expect(savingAccount).toHaveCount(3);

    await page.selectOption("#aa_accountId", "4");
    const loanAccount = await page.locator(
      "#all_transactions_for_account tbody tr"
    );
    await expect(loanAccount).toHaveCount(2);

    await page.selectOption("#aa_accountId", "5");
    const noResult = await page.locator(".well");
    await expect(noResult).toBeVisible();
  });
});
```

e2e-login.spec.ts U

e2e > e2e-login.spec.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import { test, expect } from "@playwright/test";

test.describe("Login/Logout Flow", () => {
  // Before Hook
  test.beforeEach(async ({ page }) => {
    await page.goto("http://zero.webappsecurity.com/");
  });

  test("Negative Scenario for login", async ({ page }) => {
    await page.click("#signin_button");
    await page.fill("#user_login", "invalid-user");
    await page.fill("#user_password", "invalid-pass");
    await page.getByText("Sign in").click();

    const errorMsg = await page.locator(".alert-error");
    await expect(errorMsg).toContainText("Login and/or password are wrong.");
  });

  test("Positive Scenario for login & logout", async ({ page }) => {
    await page.click("#signin_button");
    await page.fill("#user_login", "username");
    await page.fill("#user_password", "password");
    await page.getByText("Sign in").click();

    await page.goBack();
    await page.click("#onlineBankingMenu");
    await page.click("#account_summary_link");
    const accountSummaryTab = await page.locator("#account_summary_tab");
    await expect(accountSummaryTab).toBeVisible();
  });
});
```

e2e-payment.spec.ts U

e2e > e2e-payment.spec.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import { test, expect } from "@playwright/test";

test.describe("New Payment", () => {
  // Before Hook
  test.beforeEach(async ({ page }) => {
    await page.goto("http://zero.webappsecurity.com/");
    await page.click("#signin_button");
    await page.fill("#user_login", "username");
    await page.fill("#user_password", "password");
    await page.getByText("Sign in").click();

    await page.goBack();
    await page.click("#onlineBankingMenu");
    await page.click("#account_summary_link");
  });

  test("should send new payment", async ({ page }) => {
    await page.click("#pay_bills_tab");
    await page.selectOption("#sp_payee", "bofa");
    await page.click(".icon-question-sign");

    const bankAccount = await page.locator("#sp_payee_details");
    await expect(bankAccount).toContainText(
      "For 47844181491040 Bank of America account"
    );

    await page.selectOption("#sp_account", "4");
    await page.fill("#sp_amount", "10000");
    await page.fill("#sp_date", "2024-04-30");
    await page.fill("#sp_description", "Payment Done");
    await page.click("#pay_saved_payees");

    const successMsg = await page.locator("#alert_content");
    await expect(successMsg).toBeVisible();
    await expect(successMsg).toContainText(
      "The payment was successfully submitted."
    );
  });
});
```

e2e-search.spec.ts U

e2e > e2e-search.spec.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import { test, expect } from "@playwright/test";

test.describe("Search Results", () => {
  // Before Hook
  test.beforeEach(async ({ page }) => {
    await page.goto("http://zero.webappsecurity.com/");
  });

  test("should find search results", async ({ page }) => {
    await page.fill("#searchTerm", "Bank");
    await page.keyboard.press("Enter");

    const numberOfLinks = await page.locator("li>a");
    await expect(numberOfLinks).toHaveCount(2);
  });
});
```

e2e-submit-feedback-form.spec.ts U

e2e > e2e-submit-feedback-form.spec.ts > ...

Click here to ask Blackbox to help you code faster

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

test.describe("Feedback Form", () => {
  // Before Hook
  test.beforeEach(async ({ page }) => {
    await page.goto("http://zero.webappsecurity.com/");
  });

  test("Reset Feedback form", async ({ page }) => {
    await page.click("#feedback");
    await page.fill("#name", "some name");
    await page.fill("#email", "some email@gmail.com");
    await page.fill("#subject", "some subject");
    await page.fill("#comment", "some nice comment about the application");
    await page.click("input[name='clear']");

    const selectors = ["#name", "#email", "#subject"];

    selectors.forEach(async (selector) => {
      const input = await page.locator(selector);
      await expect(input).toBeEmpty();
    });
  });

  test("submit feedback form", async ({ page }) => {
    await page.click("#feedback");
    await page.fill("#name", "some name");
    await page.fill("#email", "some email@gmail.com");
    await page.fill("#subject", "some subject");
    await page.fill("#comment", "some nice comment about the application");
    await page.click("input[name='submit']");

    const verifyMsg = await page.locator(".offset3.span6").textContent();
    console.log(verifyMsg);
    expect(verifyMsg).toContain("Thank you for your comments, some name.");
    expect(verifyMsg).toContain(
      "They will be reviewed by our Customer Service staff and given the full attention that they deserve."
    );
  });
});

```

e2e-transfer-funds.spec.ts U

e2e > e2e-transfer-funds.spec.ts > ...

Click here to ask Blackbox to help you code faster

```
import { test, expect } from "@playwright/test";

test.describe("Transfer funds and make payment", () => {
  // Before Hook
  test.beforeEach(async ({ page }) => {
    await page.goto("http://zero.webappsecurity.com/");
    await page.click("#signin_button");
    await page.fill("#user_login", "username");
    await page.fill("#user_password", "password");
    await page.getByText("Sign in").click();

    await page.goBack();
    await page.click("#onlineBankingMenu");
    await page.click("#account_summary_link");
  });

  test("transfer funds from saving to loan account", async ({ page }) => {
    await page.click("#transfer_funds_tab");
    await page.selectOption("#tf_fromAccountId", "3");
    await page.selectOption("#tf_toAccountId", "4");
    await page.fill("#tf_amount", "700");
    await page.fill("#tf_description", "sent from saving to loan account");
    await page.click("#btn_submit");

    const boardHeader = await page.locator("h2.board-header");
    await expect(boardHeader).toContainText(
      "Transfer Money & Make Payments - Verify"
    );
    await page.click("#btn_submit");

    const submitMsg = await page.locator(".alert-success");
    await expect(submitMsg).toContainText(
      "You successfully submitted your transaction."
    );
  });
});
```

Playwright Page Object Model UI tests :

→ Page Objects :

TS ZeroBank-AbstractPage.ts U

Page-Objects > TS ZeroBank-AbstractPage.ts > ...

```
Click here to ask Blackbox to help you code faster
import { Page } from "@playwright/test";

export class AbstractPage {
  // Define selectors
  readonly page: Page;

  // Initialize selectors using constructor
  constructor(page: Page) {
    this.page = page;
  }

  // Define home page methods
  async wait(time: number): Promise<void> {
    await this.page.waitForTimeout(time);
  }
}
```

TS ZeroBank-FeedbackPage.ts U

Page-Objects > TS ZeroBank-FeedbackPage.ts > ...

```
Click here to ask Blackbox to help you code faster
import { Page, Locator, expect } from "@playwright/test";

export class FeedbackPage {
  readonly page: Page;
  readonly nameInput: Locator;
  readonly emailInput: Locator;
  readonly subjectInput: Locator;
  readonly commentInput: Locator;
  readonly feedbackButton: Locator;
  readonly clearButton: Locator;
  readonly submitButton: Locator;
  readonly feedbackSubmitMsg: Locator;

  constructor(page: Page) {
    this.page = page;
    this.nameInput = page.locator("#name");
    this.emailInput = page.locator("#email");
    this.subjectInput = page.locator("#subject");
    this.commentInput = page.locator("#comment");
    this.feedbackButton = page.locator("#feedback");
    this.clearButton = page.locator("input[name='clear']");
    this.submitButton = page.locator("input[name='submit']");
    this.feedbackSubmitMsg = page.locator(".offset3.span6");
  }

  async openFeedbackForm(): Promise<void> {
    await this.feedbackButton.click();
  }

  async fillFeedbackForm(
    name: string,
    email: string,
    subject: string,
    comment: string
  ): Promise<void> {
    await this.nameInput.fill(name);
    await this.emailInput.fill(email);
    await this.subjectInput.fill(subject);
    await this.commentInput.fill(comment);
  }
```

TS ZeroBank-FeedbackPage.ts U

Page-Objects > TS ZeroBank-FeedbackPage.ts > ...

```
export class FeedbackPage {
  async clearFeedbackForm(): Promise<void> {
    await this.clearButton.click();
  }

  async submitFeedbackForm(): Promise<void> {
    await this.submitButton.click();
  }

  async getVerifyMessage(): Promise<string> {
    const message = await this.feedbackSubmitMsg.textContent();
    return message ?? ""; // Return an empty string if message is null
  }

  async assertReset(selectors: string[]): Promise<void> {
    for (const selector of selectors) {
      const input = await this.page.locator(selector);
      await expect(input).toBeEmpty();
    }
  }

  async verifyFeedbackSentMsg(): Promise<void> {
    const verifyMsg = await this.getVerifyMessage();
    expect(verifyMsg).toContain("Thank you for your comments, some name.");
    expect(verifyMsg).toContain(
      "They will be reviewed by our Customer Service staff and given the full attention that they deserve."
    );
  }
}
```

ZeroBank-HomePage.ts U

Page-Objects > ZeroBank-HomePage.ts > HomePage > assertNumberOfLinks

Click here to ask Blackbox to help you code faster

```ts
import { Locator, Page, expect } from "@playwright/test";

export class HomePage {
  // Define selectors
  readonly page: Page;
  readonly signInBtn: Locator;
  readonly searchBox: Locator;
  readonly links: Locator;
  readonly onlineBankingMenu: Locator;
  readonly accountSummaryLink: Locator;

  // Initialize selectors using constructor
  constructor(page: Page) {
    this.page = page;
    this.signInBtn = page.locator("#signin_button");
    this.searchBox = page.locator("#searchTerm");
    this.links = page.locator("li>a");
    this.onlineBankingMenu = page.locator("#onlineBankingMenu");
    this.accountSummaryLink = page.locator("#account_summary_link");
  }

  // Define home page methods
  async clickOnSigninButton(): Promise<void> {
    await this.signInBtn.click();
  }

  async searchFor(searchTerm: string): Promise<void> {
    await this.searchBox.fill(searchTerm);
    await this.page.keyboard.press("Enter");
  }

  async assertNumberOfLinks(expectedCount: number): Promise<void> {
    await expect(this.links).toHaveCount(expectedCount);
  }

  async navigateToAccountSummary(): Promise<void> {
    await this.onlineBankingMenu.click();
    await this.accountSummaryLink.click();
  }
}
```

ZeroBank-LoginPage.ts U

Page-Objects > ZeroBank-LoginPage.ts > LoginPage > visit

Click here to ask Blackbox to help you code faster

```ts
import { Locator, Page, expect } from "@playwright/test";
import { AbstractPage } from "./ZeroBank-AbstractPage";

export class LoginPage extends AbstractPage {
  // Define selectors
  // readonly page: Page;
  readonly usernameInput: Locator;
  readonly passwordInput: Locator;
  readonly loginButton: Locator;
  readonly errorMessage: Locator;

  // Initialize selectors using constructor
  constructor(page: Page) {
    // this.page = page;
    super(page);
    this.usernameInput = page.locator("#user_login");
    this.passwordInput = page.locator("#user_password");
    this.loginButton = page.getByText("Sign in");
    this.errorMessage = page.locator(".alert-error");
  }

  // Define login page methods
  /**
   * Navigate to the login page
   */
  async visit(): Promise<void> {
    await this.page.goto("http://zero.webappsecurity.com/");
  }

  /**
   * Perform login operation
   * @param username - The username to fill in the username input field
   * @param password - The password to fill in the password input field
   */
  async login(username: string, password: string): Promise<void> {
    await this.usernameInput.fill(username);
    await this.passwordInput.fill(password);
    await this.loginButton.click();
  }
```

ZeroBank-LoginPage.ts U

Page-Objects > ZeroBank-LoginPage.ts > LoginPage > visit

```ts
export class LoginPage extends AbstractPage {

  /**
   * Assert error message displayed on the login page
   */
  async assertErrorMsg(): Promise<void> {
    await expect(this.errorMessage).toContainText(
      "Login and/or password are wrong."
    );
  }
}
```

TS ZeroBank-PaymentPage.ts U

Page-Objects > TS ZeroBank-PaymentPage.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import { Locator, Page, expect } from "@playwright/test";

export class PaymentPage {
  // Define selectors
  readonly page: Page;
  readonly paySelectBox: Locator;
  readonly questionSignIcon: Locator;
  readonly payeeDetail: Locator;
  readonly accountSelectBox: Locator;
  readonly amountInput: Locator;
  readonly dateInput: Locator;
  readonly descriptionInput: Locator;
  readonly submitPaymentButton: Locator;
  readonly successMsg: Locator;

  // Initialize selectors using constructor
  constructor(page: Page) {
    this.page = page;
    this.paySelectBox = page.locator("#sp_payee");
    this.questionSignIcon = page.locator(".icon-question-sign");
    this.payeeDetail = page.locator("#sp_payee_details");
    this.accountSelectBox = page.locator("#sp_account");
    this.amountInput = page.locator("#sp_amount");
    this.dateInput = page.locator("#sp_date");
    this.descriptionInput = page.locator("#sp_description");
    this.submitPaymentButton = page.locator("#pay_saved_payees");
    this.successMsg = page.locator("#alert_content");
  }
```

TS ZeroBank-PaymentPage.ts U

Page-Objects > TS ZeroBank-PaymentPage.ts > ...

```ts
export class PaymentPage {
  // Define home page methods
  async createPayment(
    payee: string,
    expectedText: string,
    accountType: string,
    amount: string,
    date: string,
    description: string
  ): Promise<void> {
    await this.paySelectBox.selectOption(payee);
    await this.questionSignIcon.click();
    await this.assertPayeeDetailText(expectedText);
    await this.accountSelectBox.selectOption(accountType);
    await this.amountInput.fill(amount);
    await this.dateInput.fill(date);
    await this.descriptionInput.fill(description);
    await this.submitPaymentButton.click();
  }

  async assertPayeeDetailText(expectedText: string): Promise<void> {
    await expect(this.payeeDetail).toContainText(expectedText);
  }

  async assertSuccessMsg(): Promise<void> {
    await expect(this.successMsg).toBeVisible();
    await expect(this.successMsg).toContainText(
      "The payment was successfully submitted."
    );
  }
}
```

→ Components :

TS ZeroBank-Navbar.ts U

Page-Objects > Components > TS ZeroBank-Navbar.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import { Locator, Page, expect } from "@playwright/test";

export class Navbar {
  // Define selectors
  readonly page: Page;
  readonly accountSummary: Locator;
  readonly accountActivity: Locator;
  readonly transferFunds: Locator;
  readonly PayBills: Locator;
  readonly myMoneyMap: Locator;
  readonly onlineStatements: Locator;

  // Initialize selectors using constructor
  constructor(page: Page) {
    this.page = page;
    this.accountSummary = page.locator("#account_summary_tab");
    this.accountActivity = page.locator("#account_activity_tab");
    this.transferFunds = page.locator("#transfer_funds_tab");
    this.PayBills = page.locator("#pay_bills_tab");
    this.myMoneyMap = page.locator("#money_map_tab");
    this.onlineStatements = page.locator("#online_statements_tab");
  }

  // Define methods
  async assertAccountSummaryTabVisible(): Promise<void> {
    await expect(this.accountSummary).toBeVisible();
  }
```

TS ZeroBank-Navbar.ts U

Page-Objects > Components > TS ZeroBank-Navbar.ts > ...

```ts
export class Navbar {

  async clickOnTab(tab: string): Promise<void> {
    switch (tab) {
      case "Account Summary":
        await this.accountSummary.click();
        break;
      case "Account Activity":
        await this.accountActivity.click();
        break;
      case "Transfer Funds":
        await this.transferFunds.click();
        break;
      case "Pay Bills":
        await this.PayBills.click();
        break;
      case "My Money Map":
        await this.myMoneyMap.click();
        break;
      case "Online Statements":
        await this.onlineStatements.click();
        break;
      default:
        throw new Error("This tab does not exist..!!");
    }
  }
}
```

→ Page Object Model Test-Files :

e2e-pom-login.spec.ts U

e2e > e2e-pom-login.spec.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import { test } from "@playwright/test";
import { LoginPage } from "../Page-Objects/ZeroBank-LoginPage";
import { HomePage } from "../Page-Objects/ZeroBank-HomePage";
import { Navbar } from "../Page-Objects/Components/ZeroBank-Navbar";

test.describe("Login/Logout Flow", () => {
  let loginPage: LoginPage;
  let homePage: HomePage;
  let navbar: Navbar;

  // Before Hook
  test.beforeEach(async ({ page }) => {
    loginPage = new LoginPage(page);
    homePage = new HomePage(page);
    navbar = new Navbar(page);

    await loginPage.visit();
  });

  test("Negative Scenario for login", async () => {
    await homePage.clickOnSigninButton();
    await loginPage.login("invalid-user", "invalid-pass");
    await loginPage.assertErrorMsg();
  });

  test("Positive Scenario for login & logout", async ({ page }) => {
    await homePage.clickOnSigninButton();
    await loginPage.login("username", "password");
    await page.goBack();
    await loginPage.wait(3000); // wait method called from abstract class
    await homePage.navigateToAccountSummary();
    await navbar.assertAccountSummaryTabVisible();
  });
});

```

e2e-pom-search.spec.ts U

e2e > e2e-pom-search.spec.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import { test } from "@playwright/test";
import { LoginPage } from "../Page-Objects/ZeroBank-LoginPage";
import { HomePage } from "../Page-Objects/ZeroBank-HomePage";

test.describe("Search Results", () => {
  let loginPage: LoginPage;
  let homePage: HomePage;
  // Before Hook
  test.beforeEach(async ({ page }) => {
    loginPage = new LoginPage(page);
    homePage = new HomePage(page);
    await loginPage.visit();
  });

  test("should find search results", async () => {
    await homePage.searchFor("Bank");
    await homePage.assertNumberOfLinks(2);
  });
});

```

e2e-pom-payment.spec.ts U

e2e > e2e-pom-payment.spec.ts > ...

Click here to ask Blackbox to help you code faster

```ts
import { test } from "@playwright/test";
import { LoginPage } from "../Page-Objects/ZeroBank-LoginPage";
import { HomePage } from "../Page-Objects/ZeroBank-HomePage";
import { Navbar } from "../Page-Objects/Components/ZeroBank-Navbar";
import { PaymentPage } from "../Page-Objects/ZeroBank-PaymentPage";

test.describe("New Payment", () => {
  let loginPage: LoginPage;
  let homePage: HomePage;
  let navbar: Navbar;
  let paymentPage: PaymentPage;

  // Before Hook
  test.beforeEach(async ({ page }) => {
    loginPage = new LoginPage(page);
    homePage = new HomePage(page);
    navbar = new Navbar(page);
    paymentPage = new PaymentPage(page);

    await loginPage.visit();
    await homePage.clickOnSigninButton();
    await loginPage.login("username", "password");
    await page.goBack();
    await homePage.navigateToAccountSummary();
  });

  test("should send new payment", async () => {
    // await page.click("#pay_bills_tab");
    await navbar.clickOnTab("Pay Bills");
    await paymentPage.createPayment(
      "bofa",
      "For 47844181491040 Bank of America account",
      "4",
      "10000",
      "2024-04-30",
      "Payment Done"
    );
    await paymentPage.assertSuccessMsg();
  });
});
```

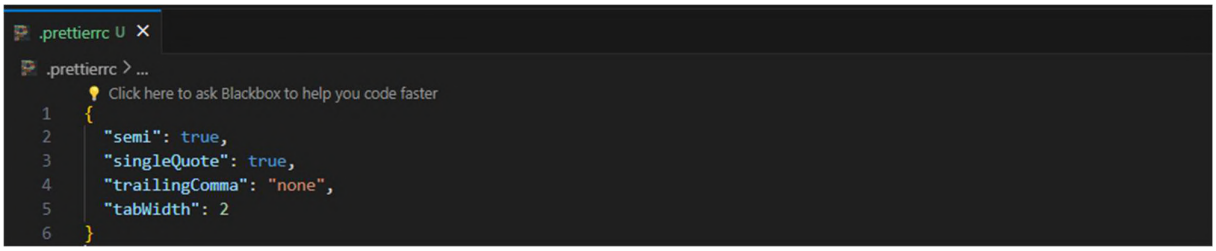
e2e-pom-submit-feedback-form.spec.ts U

e2e > e2e-pom-submit-feedback-form.spec.ts > ...

```ts
const { test } = require("@playwright/test");

import { FeedbackPage } from "../Page-Objects/ZeroBank-FeedbackPage";
import { LoginPage } from "../Page-Objects/ZeroBank-LoginPage";
import { HomePage } from "../Page-Objects/ZeroBank-HomePage";

test.describe("Feedback Form", () => {
  let loginPage: LoginPage;
  let homePage: HomePage;
  let feedbackPage: FeedbackPage;
  // Before Hook
  test.beforeEach(async ({ page }) => {
    loginPage = new LoginPage(page);
    homePage = new HomePage(page);
    feedbackPage = new FeedbackPage(page);

    await loginPage.visit();
    await feedbackPage.openFeedbackForm();
  });

  test("Reset Feedback form", async () => {
    await feedbackPage.fillFeedbackForm(
      "some name",
      "some email@gmail.com",
      "some subject",
      "some nice comment about the application"
    );
    await feedbackPage.clearFeedbackForm();

    const selectors = ["#name", "#email", "#subject"];
    await feedbackPage.assertReset(selectors);
  });

  test("submit feedback form", async () => {
    await feedbackPage.fillFeedbackForm(
      "some name",
      "some email@gmail.com",
      "some subject",
      "some nice comment about the application"
    );
    await feedbackPage.submitFeedbackForm();
    await feedbackPage.verifyFeedbackSentMsg();
  });
});
```

Prettier :

```bash
npm install --save-dev prettier
```



Eslint Configurations :

1. **Install ESLint**: First, you need to install ESLint and its necessary plugins as devDependencies in your project. You can do this using npm or yarn.

```bash
npm install eslint @typescript-eslint/parser @typescript-eslint/eslint-plugin --save-de
```

2. **Initialize ESLint Configuration**: After installing ESLint, you can initialize an ESLint configuration file using the following command:

```bash
npx eslint --init
```

This command will guide you through creating a `.eslintrc.js` file where you can define your ESLint configuration. You can choose the options that best suit your project's requirements.

5. **Run ESLint**: Once you have configured ESLint and defined your rules, you can run ESLint to perform linting checks on your project files. You can do this using the following command:

```bash
npx eslint .
```

This command will analyze all JavaScript and TypeScript files in your project directory and report any linting errors or warnings based on your ESLint configuration.

6. **Integrate ESLint with Your IDE (Optional)**: Many code editors and IDEs provide ESLint integration, allowing you to see linting errors and warnings directly in your editor as you write code. You may need to install ESLint extensions or plugins for your specific editor.

.eslintrc.js U

.eslintrc.js > ...

Click here to ask Blackbox to help you code faster

```
module.exports = {
  root: true,
  env: {
    node: true,
    es6: true,
    jest: true
  },
  extends: ['eslint:recommended'],
  parserOptions: {
    ecmaVersion: 2021,
    sourceType: 'module'
  },
  rules: {
    'no-unused-vars': 'warn',
    'no-console': 'warn',
    'no-undef': 'error',
    'prefer-const': 'warn',
    'no-var': 'error',
    semi: ['error', 'never'],
    indent: ['error', 2],
    quotes: ['error', 'single']
  }
}
```

■ Important Resources related to Playwright :

→ https://playwright.dev/

→ https://medium.com/@dneprokos/sdet-introduction-to-the-first-playwright-project-40c9a816b114

→ https://medium.com/@dneprokos/playwright-with-net-what-is-it-useful-bafc85898b5

→ https://medium.com/@dneprokos/sdet-quick-introduction-to-allure-reports-with-playwright-397d5c9986b5

→ https://medium.com/@dneprokos/automation-stories-usefulness-of-playwright-fixtures-cced0fd45a26

→ https://medium.com/@dneprokos/automation-stories-ui-automation-split-webelements-to-components-24c0e2241370

→ https://github.com/raptatinha/tau-introduction-to-playwright

→ https://testingwithrenata.com/

→ https://github.com/raptatinha/tau-introduction-to-playwright/issues

→ https://github.com/microsoft/playwright

→ https://www.youtube.com/channel/UC46Zj8pDH5tDosqm1gd7WTg

→ https://dev.to/playwright

→ https://stackoverflow.com/questions/tagged/playwright

→ https://twitter.com/playwrightweb

→ https://medium.com/@dneprokos/sdet-introduction-to-the-first-playwright-project-40c9a816b114

→ https://serpapi.com/blog/web-scraping-with-css-selectors-using-python/#selectors_types

→ https://ceroshjacob.medium.com/playwrights-retry-feature-for-individual-test-steps-fb866f63bed2

→ https://medium.com/@merisstupar11/decoding-the-importance-of-api-automation-testing-27c1cf79a0e7

→ https://medium.com/@merisstupar11/effortless-ci-integration-running-playwright-tests-with-github-actions-9df48837d68f

→ https://medium.com/@oroz.askarov/building-a-robust-automation-framework-in-playwright-typescript-version-b13be4e4bf56

→ https://medium.com/@avsaryagmurr/playwright-and-cucumber-automation-using-typescript-guide-3-enhancing-add-items-to-cart-e149301f9b23

→ https://medium.com/@avsaryagmurr/playwright-and-cucumber-automation-using-typescript-guide-2-enhancing-login-scenarios-4456eaa0eaf4

→ https://playwright.dev/docs/test-parameterize

→ https://github.com/microsoft/playwright/issues/23068

→ https://hackernoon.com/playwright-vs-cypress-for-rest-api-automated-tests-who-comes-out-on-top

→ https://medium.com/lingvano/how-we-reduced-testing-time-by-70-by-moving-from-cypress-to-playwright-32a731d468ba

→ https://medium.com/trendyol-tech/testing-nirvana-trendyol-coupon-team-reveals-the-secrets-of-playwrights-magic-wand-f45fa3e0be00

→ https://www.houseful.blog/posts/2023/playwright-standards/

→ https://playwrightsolutions.com/playwright-ask-me-anything-debrief-will-playwright-replace-cypress/?utm_campaign=Software%2BTesting%2BWeekly&utm_medium=email&utm_source=Software_Testing_Weekly_201

→ https://blog.martioli.com/debugging-tests-with-playwright/?utm_campaign=Software%2BTesting%2BWeekly&utm_medium=email&utm_source=Software_Testing_Weekly_209

→ https://afsalbacker.medium.com/access-playwright-trace-viewer-reports-online-via-amazon-s3-51fd365d80f6

→ https://www.linkedin.com/pulse/test-automation-how-bypass-re-login-playwright-python-nir-tal-cfnnf/

→ https://dev-tester.com/the-value-of-automated-regression-testing-for-high-quality-applications/?utm_campaign=Software%2BTesting%2BWeekly&utm_medium=email&utm_source=Software_Testing_Weekly_216

→ https://blog.martioli.com/selectors-in-automation-frameworks-with-cypress/

→ https://ray.run/blog/tips-for-writing-efficient-playwright-test-scripts

→ https://ceroshjacob.medium.com/playwright-locators-targeting-elements-efficiently-ba2352ba6d76

→ https://medium.com/@oroz.askarov/javascript-vs-typescript-for-test-automation-framework-6b5ae1b4089a

→ https://medium.com/@oroz.askarov/all-you-need-to-know-about-npm-and-packages-as-a-beginner-b6fcea8b3519

→ https://medium.com/@oroz.askarov/building-a-robust-automation-framework-in-playwright-typescript-version-b13be4e4bf56

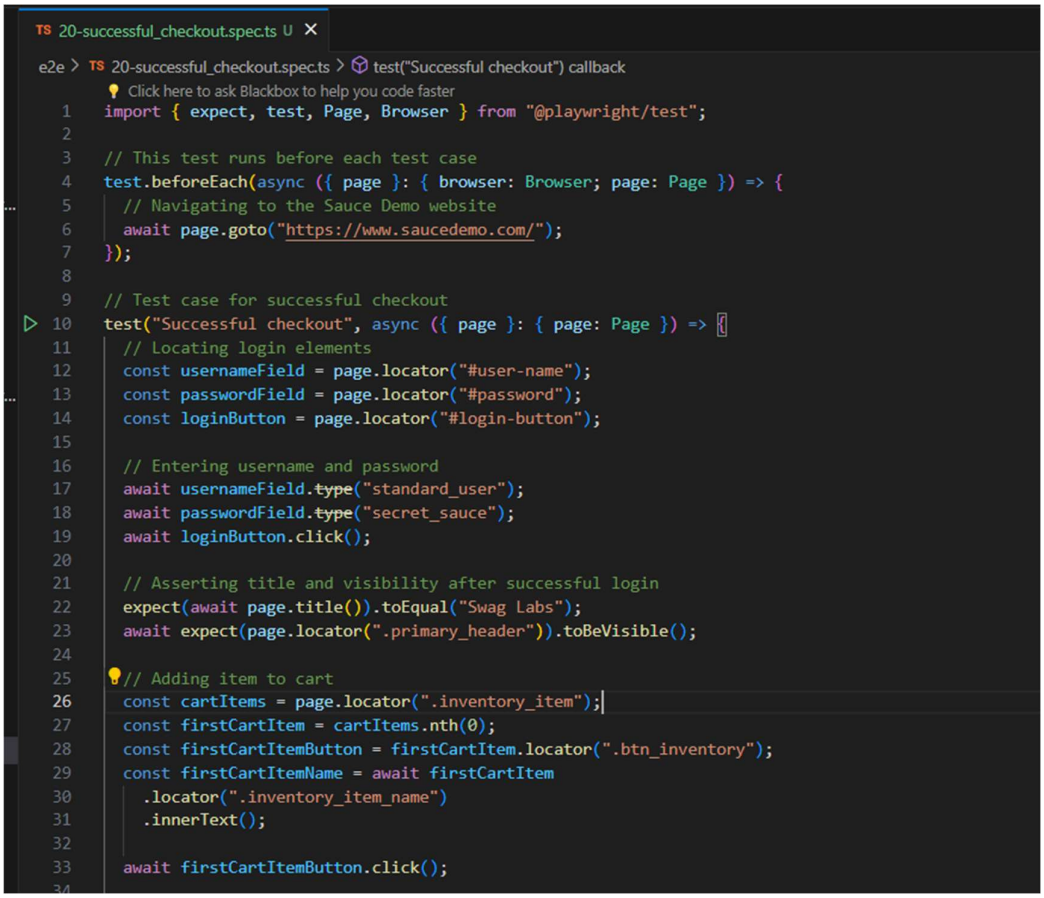
→ https://adventuresinqa.com/2024/02/27/elevating-automated-testing-playwright-meets-chatgpt/?utm_source=Coding_Jag&utm_medium=Email&utm_campaign=Coding_jag_181

→ https://www.browserstack.com/docs/percy/integrate/playwright

→ https://github.com/percy/percy-playwright

- 💻 Practiced Code :

1. Checkout Test :



```
TS 20-successful_checkout.spec.ts U ×
e2e > TS 20-successful_checkout.spec.ts > test("Successful checkout") callback
Click here to ask Blackbox to help you code faster
import { expect, test, Page, Browser } from "@playwright/test";

// This test runs before each test case
test.beforeEach(async ({ page }: { browser: Browser; page: Page }) => {
  // Navigating to the Sauce Demo website
  await page.goto("https://www.saucedemo.com/");
});

// Test case for successful checkout
test("Successful checkout", async ({ page }: { page: Page }) => {
  // Locating login elements
  const usernameField = page.locator("#user-name");
  const passwordField = page.locator("#password");
  const loginButton = page.locator("#login-button");

  // Entering username and password
  await usernameField.type("standard_user");
  await passwordField.type("secret_sauce");
  await loginButton.click();

  // Asserting title and visibility after successful login
  expect(await page.title()).toEqual("Swag Labs");
  await expect(page.locator(".primary_header")).toBeVisible();

  // Adding item to cart
  const cartItems = page.locator(".inventory_item");
  const firstCartItem = cartItems.nth(0);
  const firstCartItemButton = firstCartItem.locator(".btn_inventory");
  const firstCartItemName = await firstCartItem
    .locator(".inventory_item_name")
    .innerText();

  await firstCartItemButton.click();
```

```
  // Asserting cart badge and cart content
  expect(await page.locator(".shopping_cart_badge").innerText()).toEqual("1");
  await page.locator(".shopping_cart_link").click();
  expect(await page.locator(".cart_list .cart_item").count()).toEqual(1);
  expect(
    await page
      .locator(".cart_list .cart_item")
      .nth(0)
      .locator(".inventory_item_name")
      .innerText()
  ).toEqual(firstCartItemName);

  // Proceeding to checkout
  await page.locator("#checkout").click();
  await page.locator("#first-name").type("Yogesh");
  await page.locator("#last-name").type("Borole");
  await page.locator("#postal-code").type("0001");
  await page.locator("#continue").click();

  // Asserting checkout details
  expect(await page.locator(".inventory_item_name").innerText()).toEqual(
    firstCartItemName
  );
  expect(await page.locator(".cart_quantity").innerText()).toEqual("1");
  await page.locator("#finish").click();

  // Asserting completion message and redirect back to products
  await expect(page.locator("#checkout_complete_container")).toBeVisible();
  expect(await page.locator(".complete-header").innerText()).toEqual(
    "Thank you for your order!"
  );
  expect(await page.locator(".complete-text").innerText()).toEqual(
    "Your order has been dispatched, and will arrive just as fast as the pony can get there!"
  );
  await expect(page.locator('img[alt="Pony Express"]')).toBeVisible();
  await page.locator("#back-to-products").click();

  // Asserting URL after returning to products page
  expect(page.url()).toContain("inventory.html");
});
```

2. Alerts :

3. Assertions :

```ts
// 22-assertions.spec.ts U
// e2e > 22-assertions.spec.ts > ...
// Click here to ask Blackbox to help you code faster
import { test, expect } from "@playwright/test";

test("Assertions", async ({ page }) => {
  await page.goto("http://www.saucedemo.com");
  expect(page.locator("#user-name")).toContainText("");
  await expect.soft(page.locator("#password")).toBeEditable();
  await expect(page.locator("#login-button")).toHaveCount(1);
  await expect(page.locator("#skillmea")).not.toBeVisible();
});
```

4. Element State :

```ts
// 23-elementState.spec.ts U
// e2e > 23-elementState.spec.ts > ...
// Click here to ask Blackbox to help you code faster
import { test, expect } from "@playwright/test";

test("Element state", async ({ page }) => {
  await page.goto("http://www.saucedemo.com");
  console.log(await page.locator("#user-name").isEditable()); // true
  console.log(await page.locator("#password").isVisible()); // true
  console.log(await page.locator("#login-button").isHidden()); // false
});
```

5. Mouse Actions :

```ts
// 24-mouse.spec.ts U
// e2e > 24-mouse.spec.ts > ...
// Click here to ask Blackbox to help you code faster
import { test, expect } from "@playwright/test";

test.describe("Mouse", () => {
  test.beforeEach(async ({ page }) => {
    await page.goto("https://jspaint.app/");
  });

  test("Mouse paint", async ({ page }) => {
    await page.mouse.move(200, 200);
    await page.mouse.down();
    await page.mouse.move(400, 200);
    await page.mouse.move(400, 400);
    await page.mouse.move(200, 400);
    await page.mouse.move(200, 200);
    await page.mouse.up();
  });
});
```

6. Screenshots :

```ts
// e2e > TS 25-screenshots.spec.ts > ...
// Click here to ask Blackbox to help you code faster
import { test } from "@playwright/test";

test.describe("Screenshots", () => {
  test.beforeEach(async ({ page }) => {
    const userNameInput = page.locator("#user-name");
    const passwordInput = page.locator("#password");
    const loginButton = page.locator("#login-button");

    await page.goto("http://www.saucedemo.com");
    await userNameInput.fill("standard_user");
    await passwordInput.fill("secret_sauce");
    await loginButton.click();
  });

  test("Viewport screenshot", async ({ page }) => {
    await page.screenshot({ path: "screenshots/viewport.png" });
  });

  test("Full page screenshot", async ({ page }) => {
    await page.screenshot({ path: "screenshots/fullpage.png", fullPage: true });
  });

  test("Element screenshot", async ({ page }) => {
    await page
      .locator("#item_4_img_link")
      .screenshot({ path: "screenshots/element.png" });
  });
});
```

7. File Upload :

```ts
// e2e > TS 27-upload.spec.ts > ...
// Click here to ask Blackbox to help you code faster
import { test, expect } from "@playwright/test";

test.describe("Upload", () => {
  test("Upload file", async ({ page }) => {
    await page.goto("https://demoqa.com/upload-download");
    await page.locator("#uploadFile").setInputFiles(["./download.xlsx"]);
    await expect(page.locator("#uploadedFilePath")).toBeVisible();
  });
});
```

8. Tabs :

```ts
// e2e > TS 26-tabs.spec.ts > ...
// Click here to ask Blackbox to help you code faster
import { test } from "@playwright/test";

test.describe("Tabs", () => {
  test("Multi tabs", async ({ page, context }) => {
    await page.goto("https://demoqa.com");

    // create second tab
    const newTab = await context.newPage();
    await newTab.goto("http://saucedemo.com");

    // bring demoqa to the front
    await page.bringToFront();

    await newTab.locator("#login-button").click();
    await newTab.close();
  });
});
```

8. Visual Test :

TS 28-visual.spec.ts U ✕

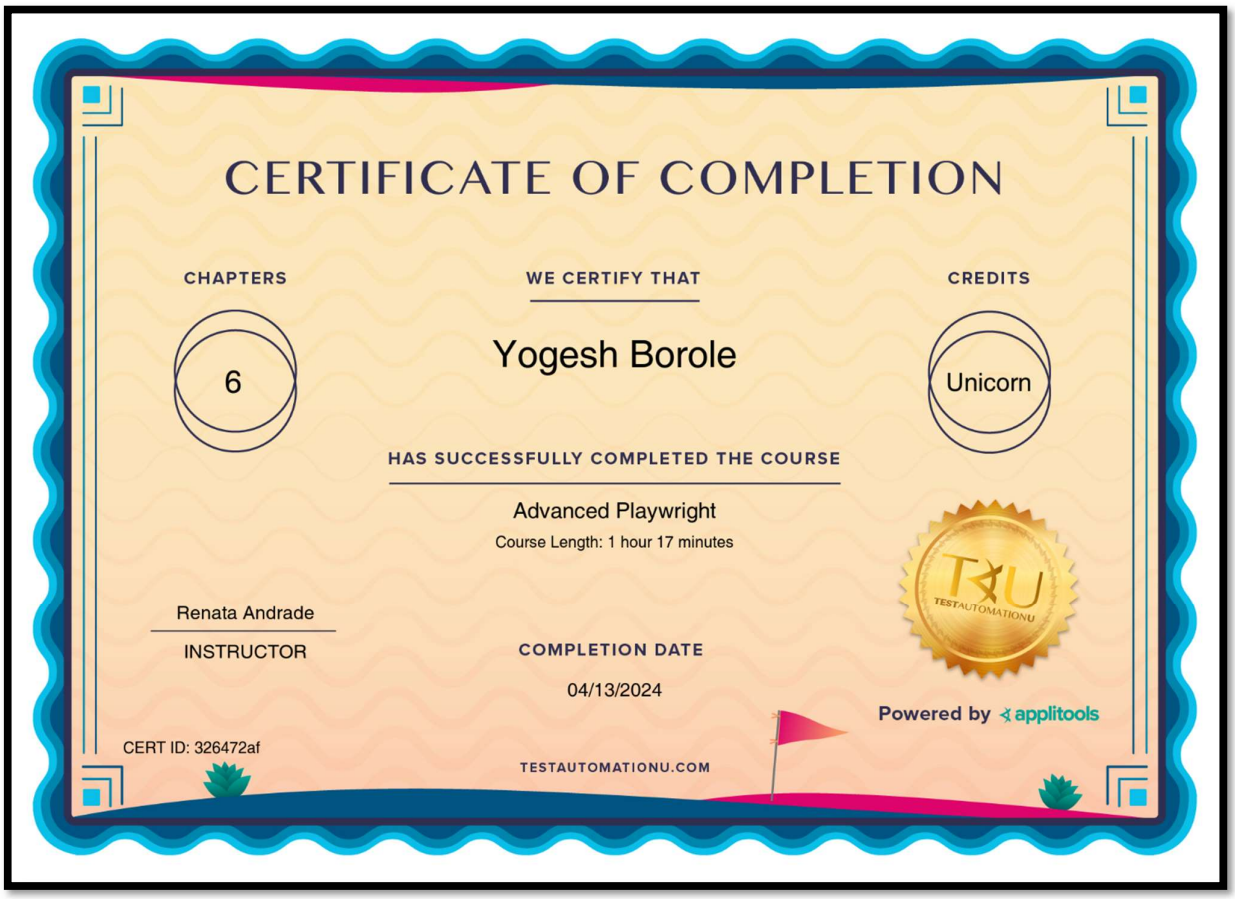
e2e > TS 28-visual.spec.ts > ...

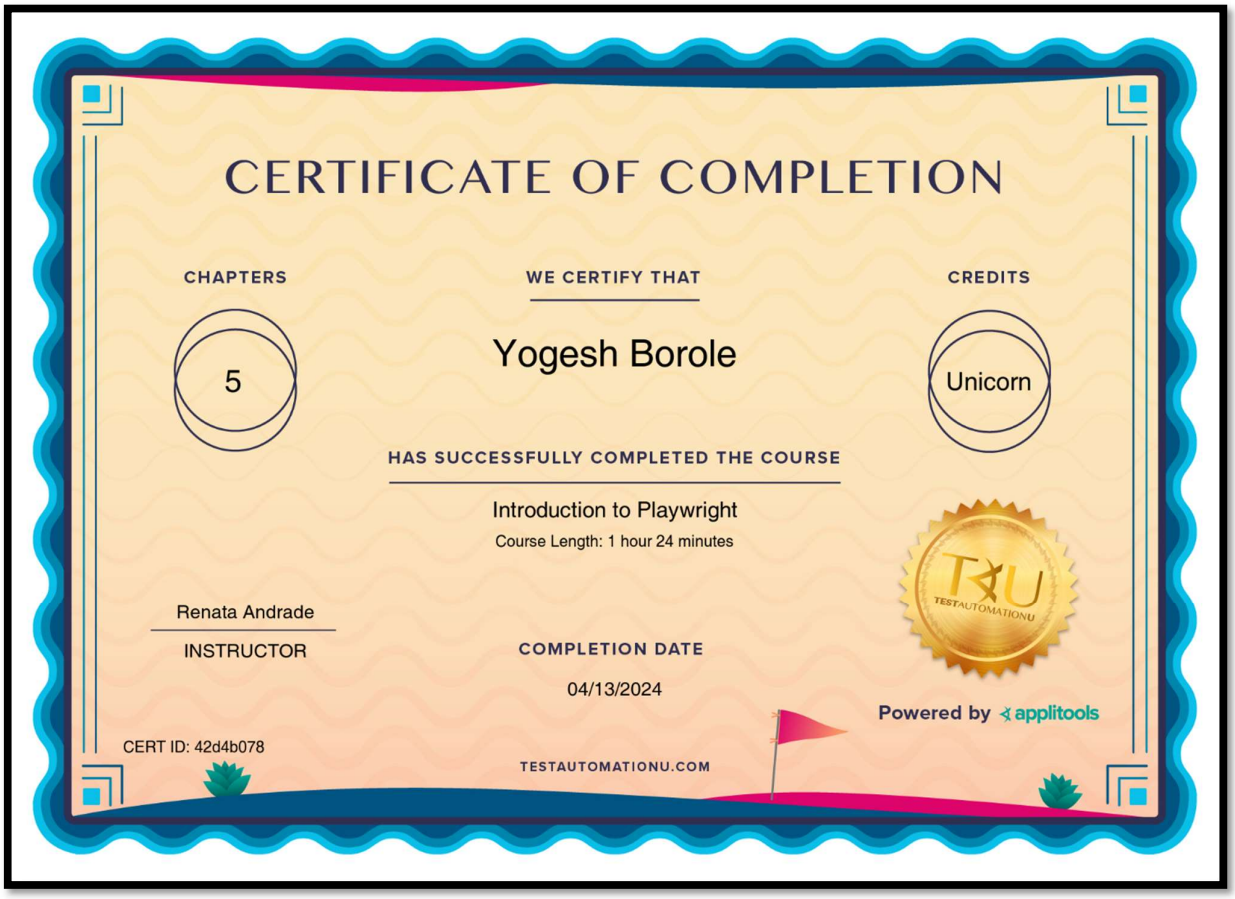
Click here to ask Blackbox to help you code faster

```ts
import { test, expect } from "@playwright/test";

test.describe("Visual testing", () => {
  test.beforeEach(async ({ page }) => {
    await page.goto("http://www.saucedemo.com");
  });

  test("Visual test - login page", async ({ page }) => {
    await expect(page).toHaveScreenshot({ maxDiffPixels: 100 });
  });
});

```

There is no end to education...!!

==========


CERTIFICATE OF COMPLETION
CHAPTERS
6
WE CERTIFY THAT
Yogesh Borole
CREDITS
Unicorn
HAS SUCCESSFULLY COMPLETED THE COURSE
Advanced Playwright
Course Length: 1 hour 17 minutes
Renata Andrade
INSTRUCTOR
COMPLETION DATE
04/13/2024
Powered by applitools
CERT ID: 326472af
TESTAUTOMATIONU.COM


CERTIFICATE OF COMPLETION
CHAPTERS
5
WE CERTIFY THAT
Yogesh Borole
CREDITS
Unicorn
HAS SUCCESSFULLY COMPLETED THE COURSE
Introduction to Playwright
Course Length: 1 hour 24 minutes
Renata Andrade
INSTRUCTOR
COMPLETION DATE
04/13/2024
Powered by applitools
CERT ID: 42d4b078
TESTAUTOMATIONU.COM

udemy

Certificate no: UC-8dcc74a3-1f1e-4473-9c61-242d70f227b1
Certificate url: ude.my/UC-8dcc74a3-1f1e-4473-9c61-242d70f227b1
Reference Number: 0004

CERTIFICATE OF COMPLETION

Playwright JS Automation Testing from Scratch with Framework

Instructors **Rahul Shetty**

Yogesh Borole

Date **Nov. 15, 2023**
Length **16.5 total hours**

udemy

Certificate no: UC-df70c659-dabf-4f9f-b556-13aec2d29ff2
Certificate url: ude.my/UC-df70c659-dabf-4f9f-b556-13aec2d29ff2
Reference Number: 0004

CERTIFICATE OF COMPLETION

Automated Software Testing with Playwright

Instructors **Kaniel Outis**

Yogesh Borole

Date **May 5, 2024**
Length **12.5 total hours**