**Playwright**

* Playwright is a powerful end-to-end testing framework that enables reliable automation for modern web applications.
* **Why Playwright?**

1. **Reliable End-to-end Testing:** 
   * Playwrigth’s Auto-wait capability ensures reliable and stable end-to-end testing for modern web applications, even in the face of dynamic and complex user interactions.
   * [**Auto-waiting:**](https://playwright.dev/docs/actionability#:~:text=On%20this%20page-,Auto%2Dwaiting,-Introduction%E2%80%8B)

The playwright performs a range of actionability checks on the elements before making actions to ensure these actions behave as expected. It auto-waits for all the relevant checks to pass and only then performs the requested actions. If required checks do not pass within the given **timeout**, action fails with the **TimeoutError.**

* + Before execution for that element playwright check some conditions are eligible on that element or not like it visible, stable or enable or not

**A screenshot of a computer screen

Description automatically generated**

1. **Cross-Browser Compatibility:**
   * Playwright supports all major browsers, including Chrome, Edge, Firefox, Safari and Opera, allowing you to test your web applications across a wide range of browsers and platforms
2. **Multiplatform Support:**
   * Playwright works seamlessly on Windows, macOS, and Linux, and also supports native mobile emulation for Google chrome on Android and Safari on iOS, enabling comprehensive testing across different devices and operating systems.
3. **Multilingual Flexibility:**
   * Playwright provides language binding for JavaScript, TypeScript, Java, Python and C# allowing you to choose the programming language that best fits your team’s preferences and expertise.

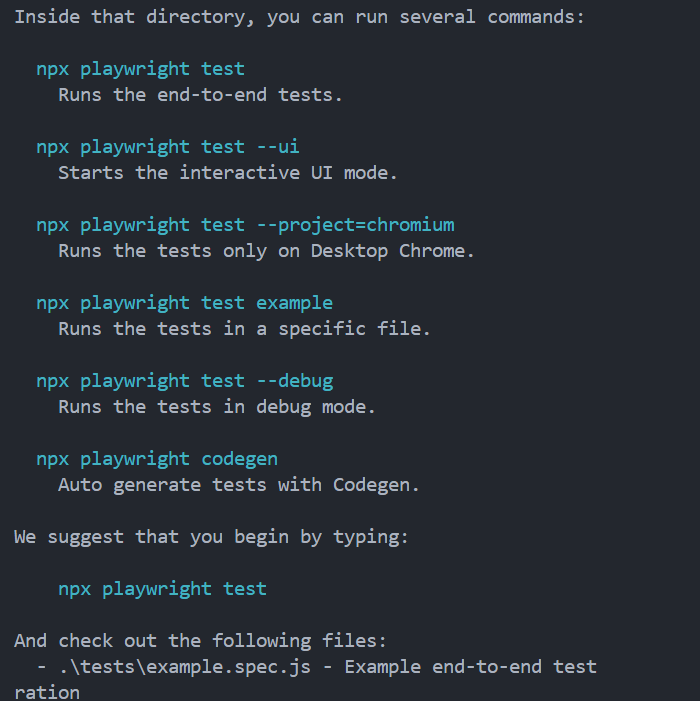
* **Playwright’s Advanced Features:**

1. **Tracing and Debugging:**
   * Playwright has built-in tracing and debugging capabilities, including automatic screenshots, test video recording and comprehensive logging, to simplify the process of identifying and resolving issues in your test suite.
2. **Network Interception:**
   * Utilize playwright’s API testing libraries to intercept and validate network calls within your web application, enabling you to test edge case scenarios and ensure the resilience of your application’s network interaction.
3. **Browser Context Management:**
   * Explore playwright’s browser context feature, which allows you to save and transfer browser state across your test suite, improving test efficiency and reducing the overhead of setting up the same browser state for each test case.
4. **Codegen Tool:**
   * It can generate code by recording your actions, saving your time and effort in creating initial test case and providing a starting point for further customization.

* **To create playwright project, we use following command**

**npm init playwright**

* + This command is creating playwright structure project with all playwright dependencies
  + **Playwright.config.json** file is important file which is also called as playwright test runner file



* **Sample test cases:**

A computer screen shot of a program code

Description automatically generated

We can manually create **page** using **browser** fixture or we can directly use **page** fixture to goto URL.

* To set which browser should execute we should mention in Playwright.config.json file.

A black background with red and green text

Description automatically generated

* To execute playwright project we use following command

**npx playwright test**

* Playwright by default run in headless mode means we won’t see anything of what is happening in the browser when our script runs. If we want to see browser execution then we should run in **headed** mode using following command

**npx playwright test –headed**

**or**

add in configuration file

A computer screen shot of a computer code

Description automatically generated

* To generate test report:

**npx playwright show-report**

* To execute in debug mode:

**npx playwright test --debug**

* To verify title of webpage we have assertion

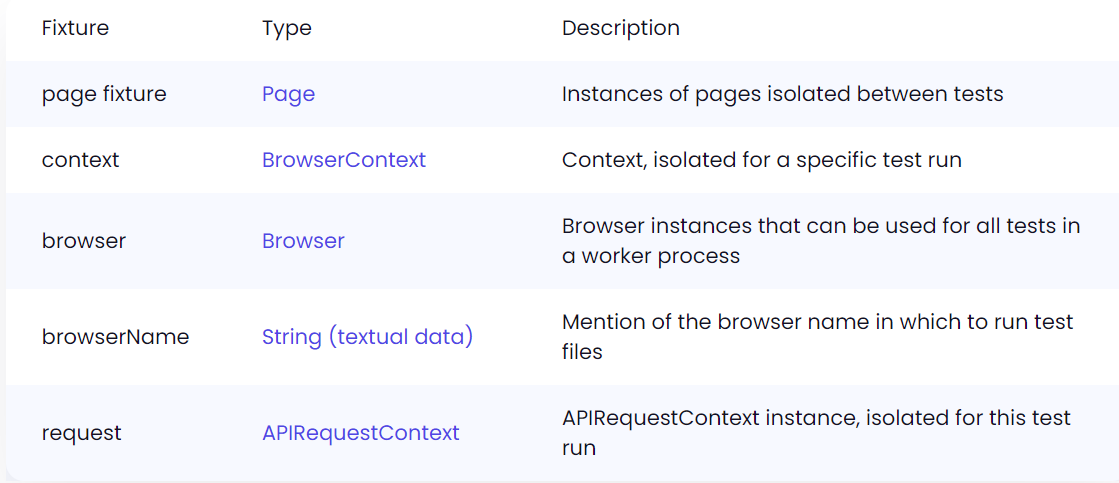


It will check and verify title of webpage if match then **passes** otherwise **failed**

* **Note: webkit** is a default playwright specific browser engine.
* **Fixtures:** [**🔗**](https://testomat.io/blog/what-is-the-use-of-fixtures-in-playwright/)
  + Playwright commonly uses with the **Page Object Model,** which is one of the approaches to structuring tests to optimize their speed and efficiency. In the framework under consideration, fixtures simplify grouping tests and are considered a way manage them as the page object method.
  + Test fixtures are preliminary conditions or steps that are executed before running a test. The testing concept of fixtures is based on their use – existing fixtures create a precise environment for each test, avoiding anything unnecessary.
  + In playwright, test fixtures allow you to reuse code for different test cases. In essence, a fixture is a function that wraps the inheritance of classes. It’s a convenient way to encapsulate your testing functionality and its data in separate blocks and call it when needed.

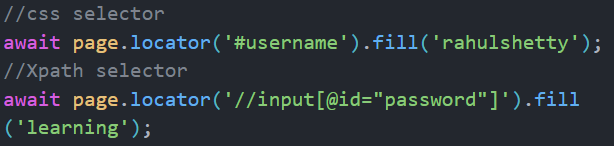
****

* + Below, we present a list of fixtures used when writing a base test in playwright:

****

* **Different types of methods in Playwright:**

1. **.fil():** 
   * **.fill()** method is use to fill/pass data in input filled

****

1. **.click():**
   * **.click()** method is used to perform click event on elements like button.

****

1. **.textContent():**
   * **.textContent()** method is used to extract text from element

****

1. **. toContainsText():**
   * **.toContainsText()** check given text available in there or not



It will check given text available in that located element or not that’s basis it will passes or failed the case

This method is assertion method.

1. **.waitFor():**
   * **.waitFor()** method is primarily used with locators to paused the execution until a specific condition is met on a web page element.
   * It helps in synchronizing your test scripts with dynamic content changes.

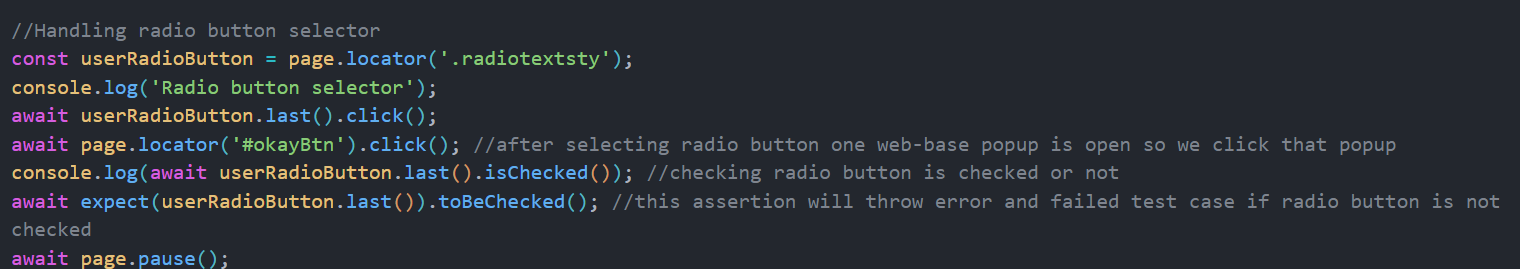
****

* + This method works on single element if multiple element is detect it won’t work. If we want to work we should select only one element by using **.first() / .last()** function.

1. **.selectOption():**
   * **.selectOption()** method is used to select option in dropdown menu

****

1. **Handling Radio button and popup:**

****

* + **.toBeChecked()** assertion is passed case if radio button is checked otherwise it failed this test case.
  + **.isChecked()** method is check button is checked or not checked.

1. **Handling checkbox:**

**A screen shot of a computer code

Description automatically generated**

* + **.toBeFalsy()** check given condition return falsy value or not. It failed test case when condition return **true**
  + **.toBeTruthy()** method failed test case if condition return **false.**
  + **.uncheck()** method is used to uncheck a button/checkbox

1. **Handle child window:**

**A screen shot of a computer code

Description automatically generated**

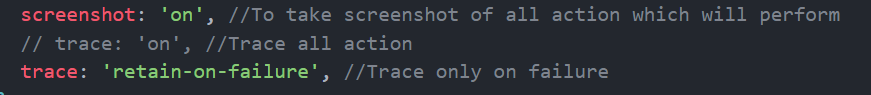
* + **Promise.all([])** create array of new pages by executing conditions written inside it
  + **.waitForEvent()** this is browser context method which can be used to wait for a specific event to occur on a page. You can use this method to wait for an event, such as a DOM element being clicked, a form being submitted, a page is loaded, or a network request being made.

1. **Generate code use Codegen tool:**

**npx playwright codegen url**

* + This command is used to open link in recording mode.

1. **To generate screenshot and trace report:**

****

* + Add this two properties in **playwright.config.json** file in use object
* **Playwright Inbuilt Locators:**

1. **.getByLabel():**
   * It is used to locate the control by its associated label using **page.getByLabel()**

**A screen shot of a computer program

Description automatically generated**

* + Use this locators when locating form field.

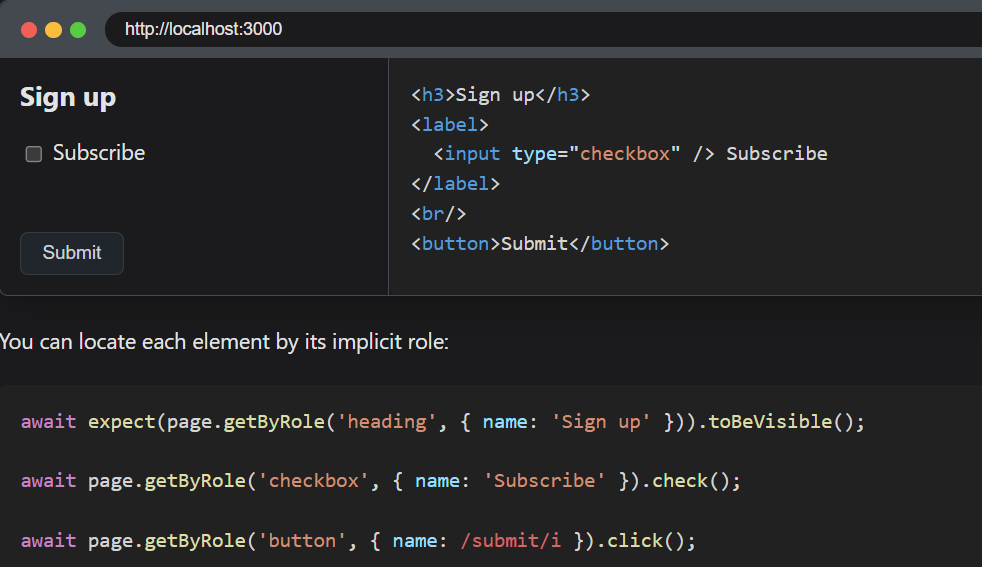
1. **.getByPlaceholder():**
   * Input may have a placeholder attribute to hint to the user what value should be entered. You can locate such an input using **page.getByPlaceholder().**

**A screenshot of a computer program

Description automatically generated**

* + Use this locator when locating form elements that do not have labels but have placeholder texts.

1. **.getByRole():**
   * The **page.getByRole()** locator reflects how users and assistive technology perceive the page, for example whether some element is a button or a checkbox. When locating by role, you should usually pass the accessible name as well, so that the locator pinpoints the exact element.

****

* + Role locators include buttons, checkboxes, headings, links, lists, tables and may more elements.
  + Note that many html elements have an implicitly defined role that is recognized by the role locator.

1. **.getByText():**
   * Find an element by the text it contains. You can match by a substring, exact string, or a regular expression when using **page.getByText().**

****

* + Matching by text always normalizes whitespace