



# AZURE PLAYWRIGHT

La pièce maîtresse du spectacle  
automatisé !

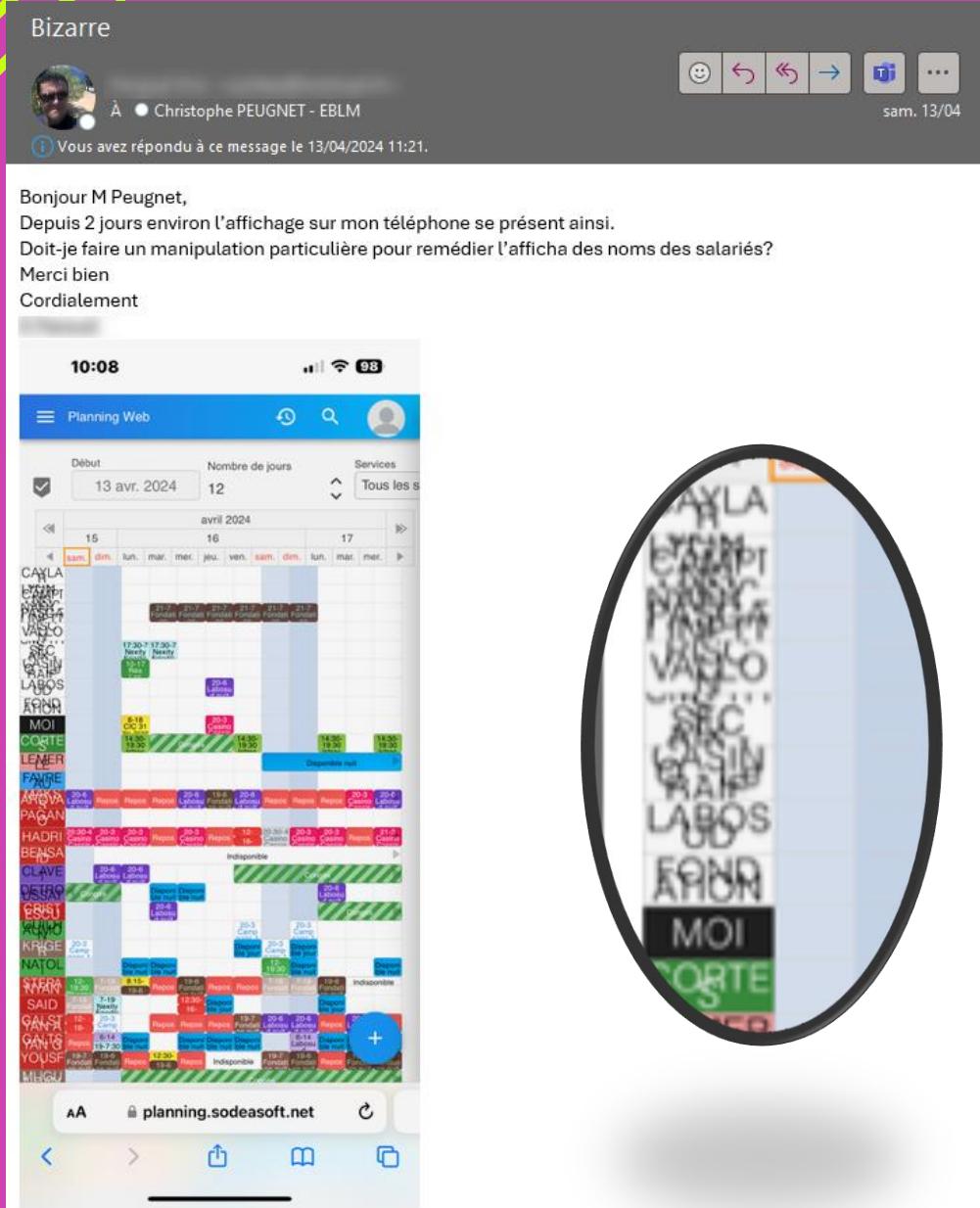




# Christophe Peugnet - Adrien Clerbois

MICROSOFT MVP, TECHNICAL ARCHITECT @ SENSE OF TECH

Follow us on social media @tossnet1 @aclerbois



Planning Web Version d'essai

Début 15/04/2024 Nombre de jours 7 Equipes toutes les équipes Afficher sur 24h tâches linéaires Aucun filtre

Planning

Vues individuelles Calculs des heures Gestion

Tâches prédefineds REU Libellé\* Réunion

Journée entière Durée (h) 10:00 ARRIÈRE PLAN POLICE

Début 17/04/2024 Fin 17/04/2024 Durée 12:00 Non verrouillée Tâche à valider

RÉPETER PLUS... ANNULER OK

Mon compte Démô Admin Faq A propos

REU

Color-coded grid:

- Red: Tony (row 1)
- Blue: Georges (row 2)
- Yellow: RTT (row 3)
- Pink: Administratif (row 4)
- Grey: (row 5)
- Orange: (row 6)
- Green: (row 7)
- Light Blue: (row 8)
- Light Green: (row 9)
- Light Orange: (row 10)
- Light Grey: (row 11)
- Dark Blue: (row 12)
- Dark Green: (row 13)
- Dark Orange: (row 14)
- Dark Grey: (row 15)

CE

DÉPLOYER UN VENDREDI, QU'ILS DISAIENT



TOUT IRA BIEN, QU'ILS DISAIENT

The screenshot displays a development environment for a .NET Core application named "WerbApp".

**Code Editor:** On the left, the `Counter.razor` file is open in the Visual Studio code editor. The code defines a component with a page title "Counter", an h1 header "Counter", a status message "Current count: @currentCount", and a primary button labeled "Click me" with an `onclick="IncrementCount"` event.

```
@page "/counter"
<PageTitle>Counter</PageTitle>
<h1>Counter</h1>
<p role="status">Current count: @currentCount</p>
<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>

@code {
    private int currentCount = 0;

    private void IncrementCount()
    {
        currentCount++;
    }
}
```

**Browser Preview:** In the center, a browser window titled "WerbApp" shows the application's home page. The page has a dark blue header with the "WerbApp" logo and navigation links for "Home", "Counter", and "Weather". The main content area displays the text "Hello, world!" and "Welcome to your new app.".

**Developer Tools:** At the bottom, the "Developer PowerShell" tab is active, providing a command-line interface for the project.

The screenshot shows a .NET Core web application development environment. On the left, the `Counter.razor` file is open in the code editor, displaying the following code:

```
@page "/counter"

<PageTitle>Counter</PageTitle>

<h1>Counter</h1>

<p role="status">Current count: <span id="currentCount">@currentCount</span></p>

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>

@code {
    private int currentCount = 0;

    private void IncrementCount()
    {
        currentCount++;
    }
}
```

The browser preview on the right shows the application running at `localhost:7154/counter`. The page title is "Counter". It displays the message "Current count: 3" and a button labeled "Click me".

## 1 Ajout d'un **projet de test**

```
dotnet new nunit -n PlaywrightTests -o Test
```

## 2 Ajout du **Nuget Microsoft.Playwright.xxx**

```
dotnet add package Microsoft.PlayWright.NUnit
```

## 3 Compiler ce projet

## 4 \bin\...\playwright.ps1 install

# RÉSUMONS

The background features a large, semi-transparent white circle centered on the left side. Overlaid on the bottom-left corner is a smaller, semi-transparent teal circle. The rest of the background is a solid medium blue.

**ÉCRIVONS NOTRE  
PREMIER TEST**

A screenshot of the Microsoft Visual Studio IDE interface, showing a .NET Core web application project named "WebApp".

The main window displays the code editor for a unit test file, `UnitTest1.cs`, which contains the following C# code:

```
namespace PlaywrightTests;
public class Tests
{
    [SetUp]
    public void Setup()
    {
    }

    [Test]
    public void Test1()
    {
        Assert.Pass();
    }
}
```

The Solution Explorer on the right shows the project structure, including the `PlaywrightTests` and `WebApp` projects, their files like `GlobalUsings.cs`, `UnitTest1.cs`, and various Razor pages (`Counter.razor`, `Error.razor`, etc.).

The Task List, Properties, and Other windows are also visible at the bottom of the interface.

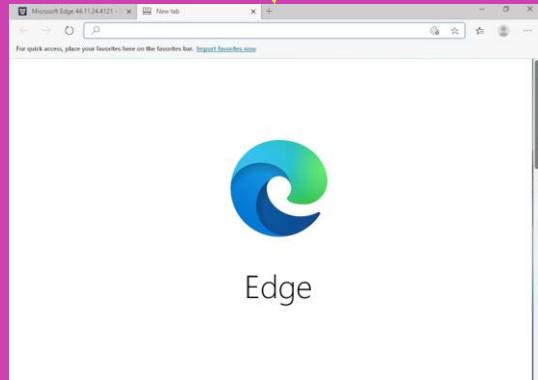


**Chrome Devtools Protocol**

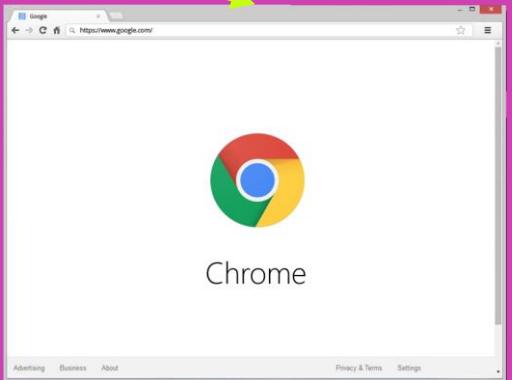
**Web InspectorProtocol**



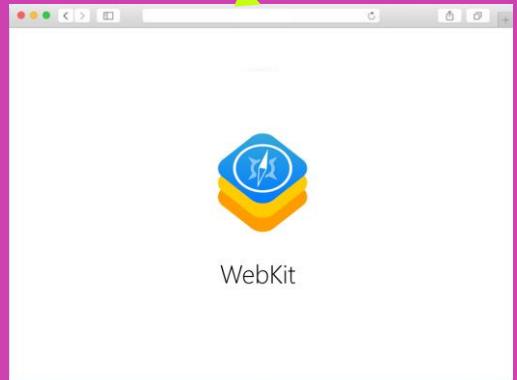
Firefox



Edge



Chrome

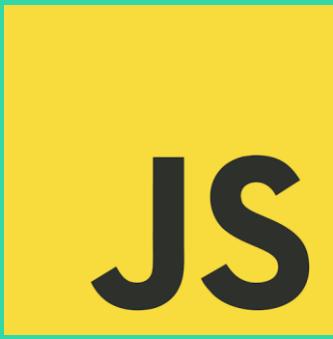


WebKit

# ISOLATION

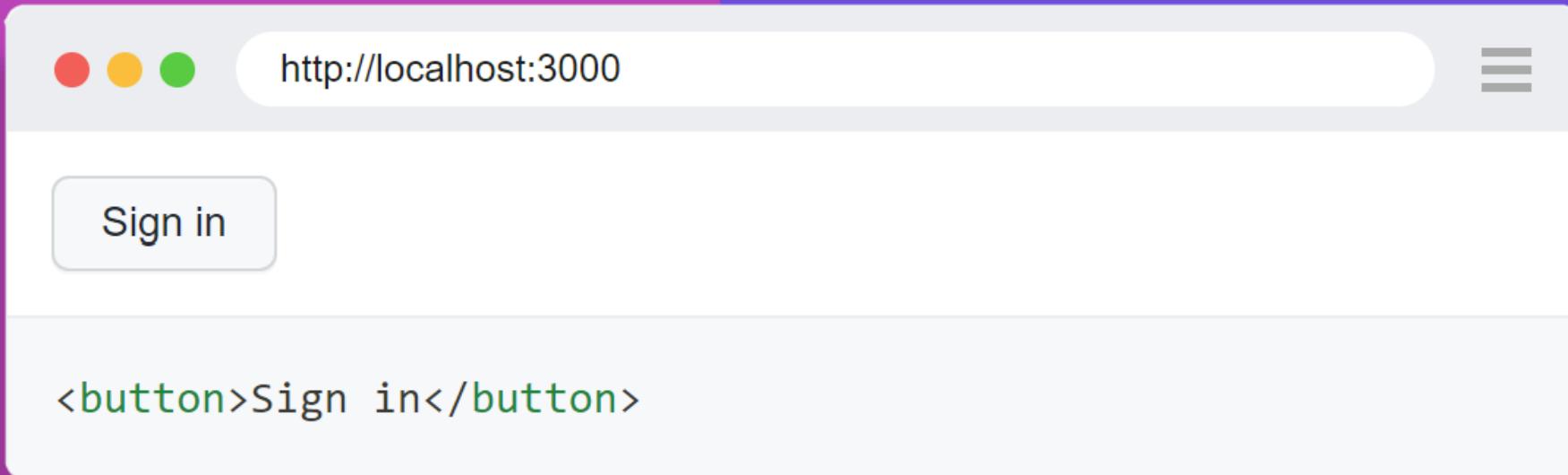


Tests written with Playwright execute in isolated clean-slate environments called browser contexts. This isolation model improves reproducibility and prevents cascading test failures.



# MULTI LANGUAGE

# LOCATORS



```
await page.GetByRole(AriaRole.Button, new() { Name = "Sign in" }).ClickAsync();
```

# LOCATORS

## Quick Guide

These are the recommended built in locators.

- `Page.GetByRole()` to locate by explicit and implicit accessibility attributes.
- `Page.GetByText()` to locate by text content.
- `Page.GetByLabel()` to locate a form control by associated label's text.
- `Page.GetByPlaceholder()` to locate an input by placeholder.
- `Page.GetByAltText()` to locate an element, usually image, by its text alternative.
- `Page.GetByTitle()` to locate an element by its title attribute.
- `Page.GetByTestId()` to locate an element based on its `data-testid` attribute (other attributes can be configured).

```
await page.GetByLabel("User Name").FillAsync("John");

await page.GetByLabel("Password").FillAsync("secret-password");

await page.GetByRole(AriaRole.Button, new() { Name = "Sign in" }).ClickAsync();

await Expect(Page.GetText("Welcome, John!")).ToBeVisibleAsync();
```

```
// Text input
await page.GetByRole(AriaRole.Textbox).FillAsync("Peter");
💡
// Check the checkbox
await page.GetByLabel("I agree to the terms above").CheckAsync();

// Assert the checked state
Assert.True(await page.GetByLabel("Subscribe to newsletter").IsCheckedAsync());

// Generic click
await page.GetByRole(AriaRole.Button).ClickAsync();

// Double click
await page.GetText("Item").DblClickAsync();

// Press keys one by one
await page.Locator("#area").TypeAsync("Hello World!");

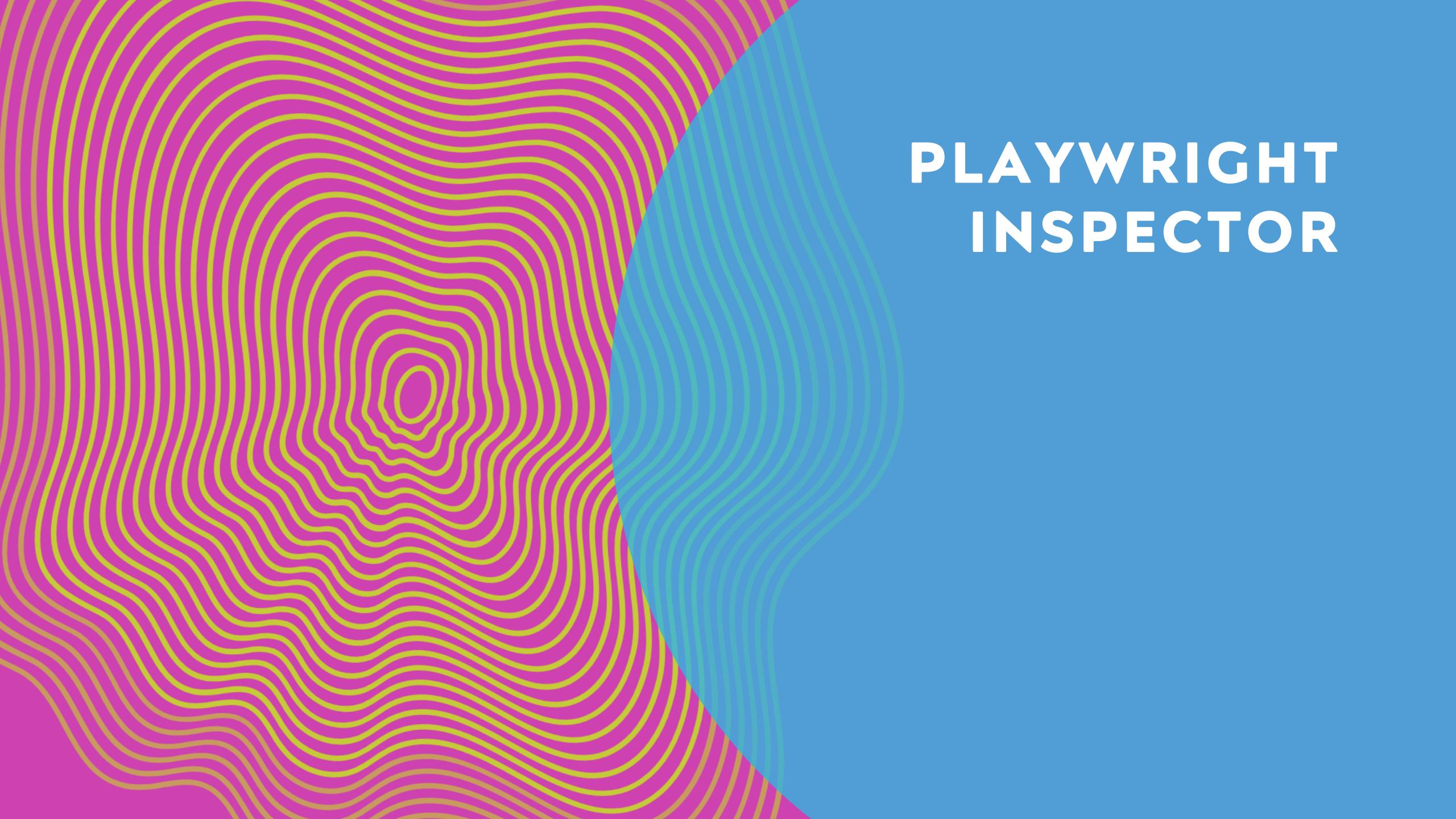
// Hit Enter
await page.GetText("Submit").PressAsync("Enter");

// Select multiple files
await page.GetByLabel("Upload files").SetInputFilesAsync(new[] { "file1.txt", "file12.txt" });
```

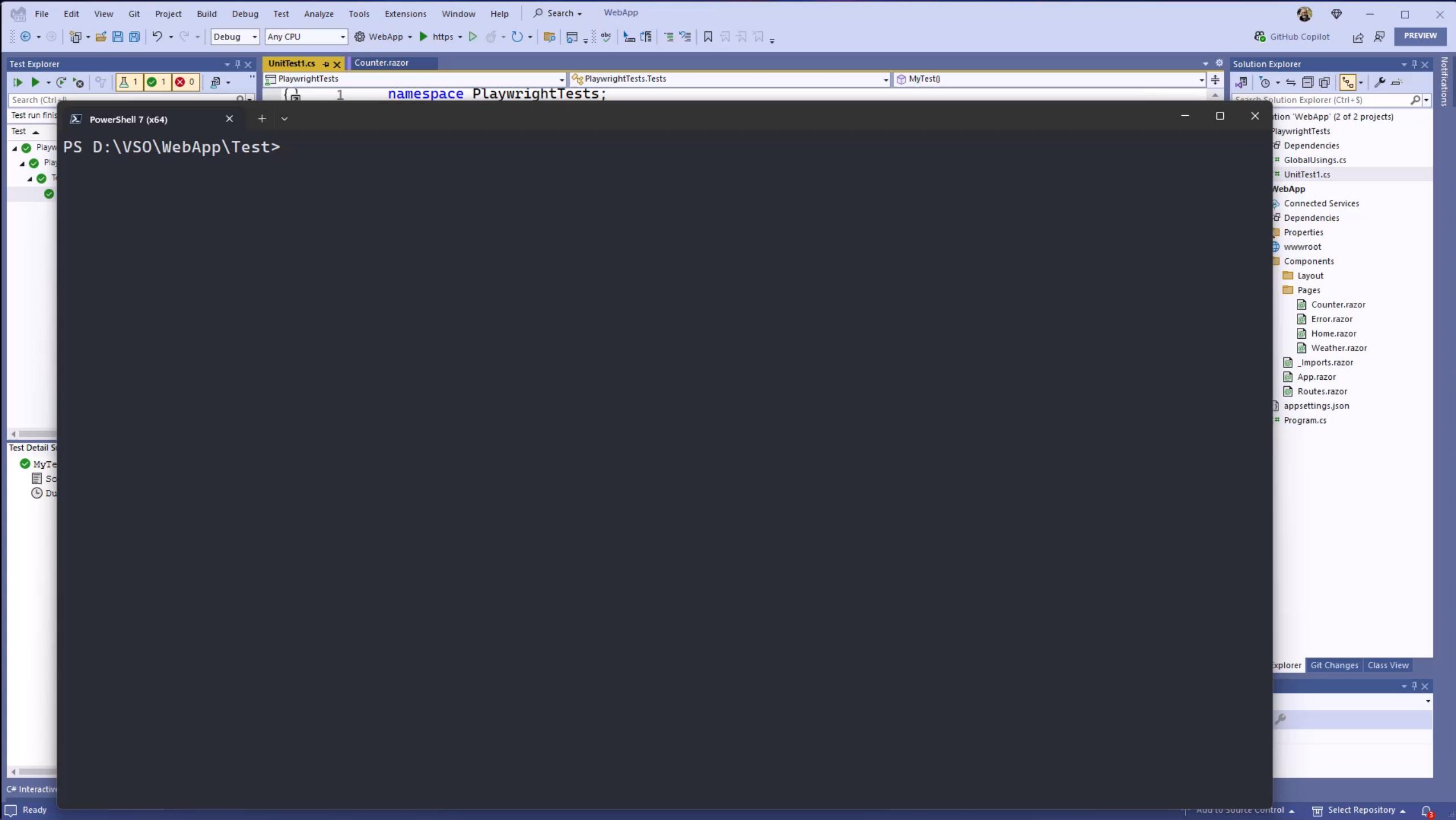
# ASSERTIONS

Assertion	Description
<a href="#">Expect(Locator).ToBeCheckedAsync()</a>	Checkbox is checked
<a href="#">Expect(Locator).ToBeDisabledAsync()</a>	Element is disabled
<a href="#">Expect(Locator).ToBeEditableAsync()</a>	Element is editable
<a href="#">Expect(Locator).ToBeEmptyAsync()</a>	Container is empty
<a href="#">Expect(Locator).ToBeEnabledAsync()</a>	Element is enabled
<a href="#">Expect(Locator).ToBeFocusedAsync()</a>	Element is focused
<a href="#">Expect(Locator).ToBeHiddenAsync()</a>	Element is not visible
<a href="#">Expect(Locator).ToBeVisibleAsync()</a>	Element is visible
<a href="#">Expect(Locator).ToContainTextAsync()</a>	Element contains text
<a href="#">Expect(Locator).ToHaveAttributeAsync()</a>	Element has a DOM attr.

Assertion	Description
<a href="#">Expect(Locator).ToHaveClassAsync()</a>	Element has a class property
<a href="#">Expect(Locator).ToHaveCountAsync()</a>	List has exact number of children
<a href="#">Expect(Locator).ToHaveCSSAsync()</a>	Element has CSS property
<a href="#">Expect(Locator).ToHaveIdAsync()</a>	Element has an ID
<a href="#">Expect(Locator).ToHaveJSPROPERTYAsync()</a>	Element has a JavaScript property
<a href="#">Expect(Locator).ToHaveTextAsync()</a>	Element matches text
<a href="#">Expect(Locator).ToHaveValueAsync()</a>	Input has a value
<a href="#">Expect(Locator).ToHaveValuesAsync()</a>	Select has options selected
<a href="#">Expect(Page).ToHaveTitleAsync()</a>	Page has a title
<a href="#">Expect(Page).ToHaveURLAsync()</a>	Page has a URL
<a href="#">Expect(ApiResponse).ToBeOKAsync()</a>	Response has an OK status

The background features a large, abstract graphic on the left side. It consists of two main color regions: a pink area on the top-left and a blue area on the bottom-right. Both regions contain a series of concentric, wavy lines that radiate from a central point, creating a sense of depth and motion. The lines are thin and light-colored, either yellow or green, which provides a high-contrast look against the pink and blue backgrounds.

# PLAYWRIGHT INSPECTOR



# UN SELFIE ?

## (AMAZING FEATURES)



This screenshot shows the Microsoft Visual Studio IDE interface with the following components visible:

- Top Bar:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, Search, WebApp.
- Solution Explorer:** Shows two projects: PlaywrightTests (2 files) and WebApp (8 files). The UnitTest1.cs file is selected in the PlaywrightTests project.
- Test Explorer:** Displays a test run summary: 1 Test (1 Pass, 0 Warnings, 0 Errors). The test MyTest has a duration of 1,8 sec.
- Code Editor:** The Counter.razor file is open, showing a Playwright test. The code includes a fixture setup and a test method named MyTest that navigates to a local host, clicks three buttons, and asserts the page contains the text "3".
- Developer PowerShell:** A terminal window showing the content root path and a warning about HTTPS redirection middleware.
- Properties:** A panel showing build configurations and other properties for the selected file.
- Bottom Bar:** C# Interactive (.NET Core), Output, Developer PowerShell, Error List, Ready.

```
[TestFixture]
public class Tests : PageTest
{
    [Test]
    public async Task MyTest()
    {
        await Page.GotoAsync("http://localhost:5154/");
        await Page.GetByRole(AriaRole.Link, new() { Name = "Counter" }).ClickAsync();
        await Page.GetByRole(AriaRole.Button, new() { Name = "Click me" }).ClickAsync();
        await Page.GetByRole(AriaRole.Button, new() { Name = "Click me" }).ClickAsync();
        await Page.GetByRole(AriaRole.Button, new() { Name = "Click me" }).ClickAsync();

        await Expect(Page.Locator("#currentCount")).ToContainTextAsync("3");
    }
}
```

```
Content root path: D:\VSO\WebApp\WebApp
warn: Microsoft.AspNetCore.HttpsPolicy.HttpsRedirectionMiddleware[3]
      Failed to determine the https port for redirect.
```

# LE DEBUG ?

The screenshot shows the Microsoft Visual Studio IDE interface with the following components visible:

- Top Bar:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, Search, GitHub Copilot, PREVIEW.
- Solution Explorer:** Shows the solution structure with two projects: PlaywrightTests and WebApp. Under PlaywrightTests, there are Dependencies, GlobalUsings.cs, UnitTest1.cs, and a folder named "Tests". Under WebApp, there are Connected Services, Dependencies, Properties, wwwroot, Components, Layout, Pages, and several Razor files like Counter.razor, Error.razor, Home.razor, Weather.razor, Imports.razor, App.razor, and Routes.razor. It also includes appsettings.json and Program.cs.
- Test Explorer:** Shows a test run summary: 1 Test (1 Passed, 0 Failed, 0 Skipped) run in 1,8 sec. The test is named MyTest.
- Unit Test Editor:** The current file is UnitTest1.cs, which contains a class Tests that inherits from PageTest. The code uses Playwright's API to navigate to a local host, take screenshots, click buttons, and assert the page content.
- Developer PowerShell:** A terminal window titled "Developer PowerShell" is open at the bottom left.
- Bottom Status Bar:** Displays the message "Test run finished: 1 Tests (1 Passed, 0 Failed, 0 Skipped) run in 1,8 sec".

```
[TestFixture]
public class Tests : PageTest
{
    [Test]
    public async Task MyTest()
    {
        await Page.GotoAsync("http://localhost:5154/");
        await Page.ScreenshotAsync(new()
        {
            Path = "screenshot1.png",
        });
        await Page.GetByRole(AriaRole.Link, new() { Name = "Counter" }).ClickAsync();
        await Page.GetByRole(AriaRole.Button, new() { Name = "Click me" }).ClickAsync();
        await Page.GetByRole(AriaRole.Button, new() { Name = "Click me" }).ClickAsync();
        await Page.GetByRole(AriaRole.Button, new() { Name = "Click me" }).ClickAsync();

        await Page.ScreenshotAsync(new()
        {
            Path = "screenshot2.png",
        });
        await Expect(Page.Locator("#currentCount")).ToContainTextAsync("3");
    }
}
```

# REACT LOCATORS

```
await page.Locator("_react=BookItem").ClickAsync();
```

match by **component**:

match by **component** and **exact property value, case-sensitive**:

match by **property value only, case-insensitive**:

match by **component** and **truthy property value**:

match by **component** and **boolean value**:

match by **property value substring**:

match by **component** and **multiple properties**:

match by **nested property value**:

match by **component** and **property value prefix**:

match by **component** and **property value suffix**:

match by **component** and **key**:

match by **property value regex**:

\_react=BookItem

\_react=BookItem[author = "Steven King"]

\_react=[author = "steven king" i]

\_react=MyButton[enabled]

\_react=MyButton[enabled = false]

\_react=[author \*= "King"]

\_react=BookItem[author \*= "king" i][year = 1990]

\_react=[some.nested.value = 12]

\_react=BookItem[author ^= "Steven"]

\_react=BookItem[author \$= "Steven"]

\_react=BookItem[key = '2']

\_react=[author = /Steven(\\s+King)?/i]

# OTHER LOCATORS

- CSS locator
- N-th element locator
- Parent element locator
- Combining two alternative locators
- Locating only visible elements
- Vue locator
- React locator
- XPath locatorLabel to form control retargeting
- Legacy text locator
- id, data-testid, data-test-id, data-test selectors
- Chaining selectors

# API TESTING

[API testing | Playwright .NET](#)

```
public class TestGitHubAPI : PlaywrightTest
{
    static string API_TOKEN = Environment.GetEnvironmentVariable("GITHUB_A

    private IAPIRequestContext Request = null;

    [SetUp]
    public async Task SetUpAPITesting()
    {
        await CreateAPIRequestContext();
    }

    private async Task CreateAPIRequestContext()
    {
        var headers = new Dictionary<string, string>();
        // We set this header per GitHub guidelines.
        headers.Add("Accept", "application/vnd.github.v3+json");
        // Add authorization token to all requests.
        // Assuming personal access token available in the environment.
        headers.Add("Authorization", "token " + API_TOKEN);

        Request = await this.Playwright.APIRequest.NewContextAsync(new() {
            // All requests we send go to this API endpoint.
            baseURL = "https://api.github.com",
            extraHTTPHeaders = headers,
        });
    }

    [TearDown]
    public async Task TearDownAPITesting()
    {
        await Request.DisposeAsync();
    }
}
```

# ACCESS -IBIL- ITY T E S TING

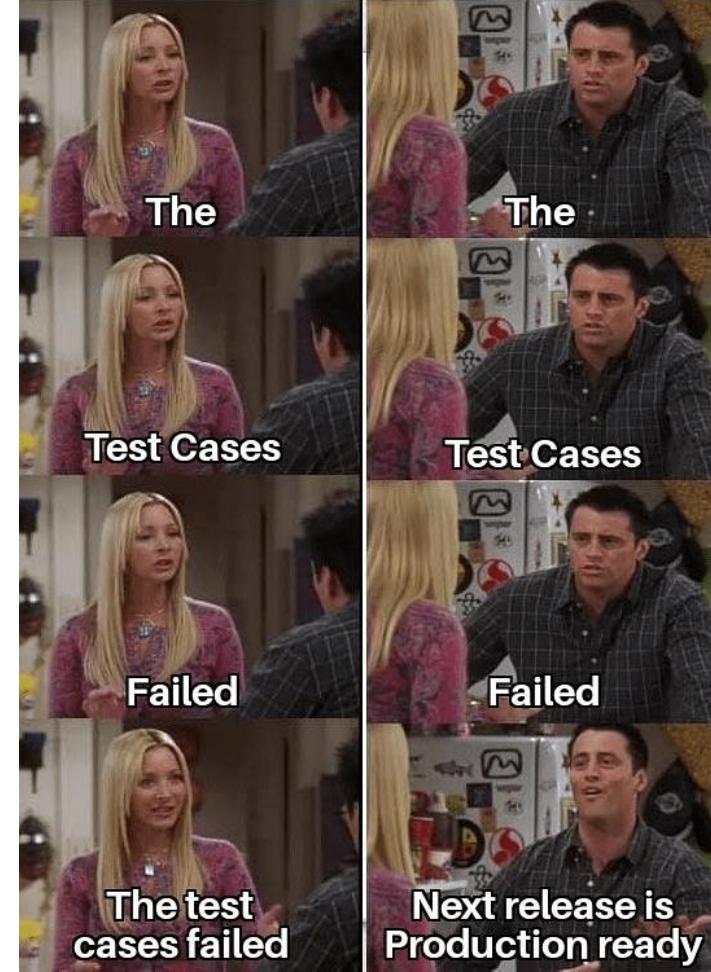
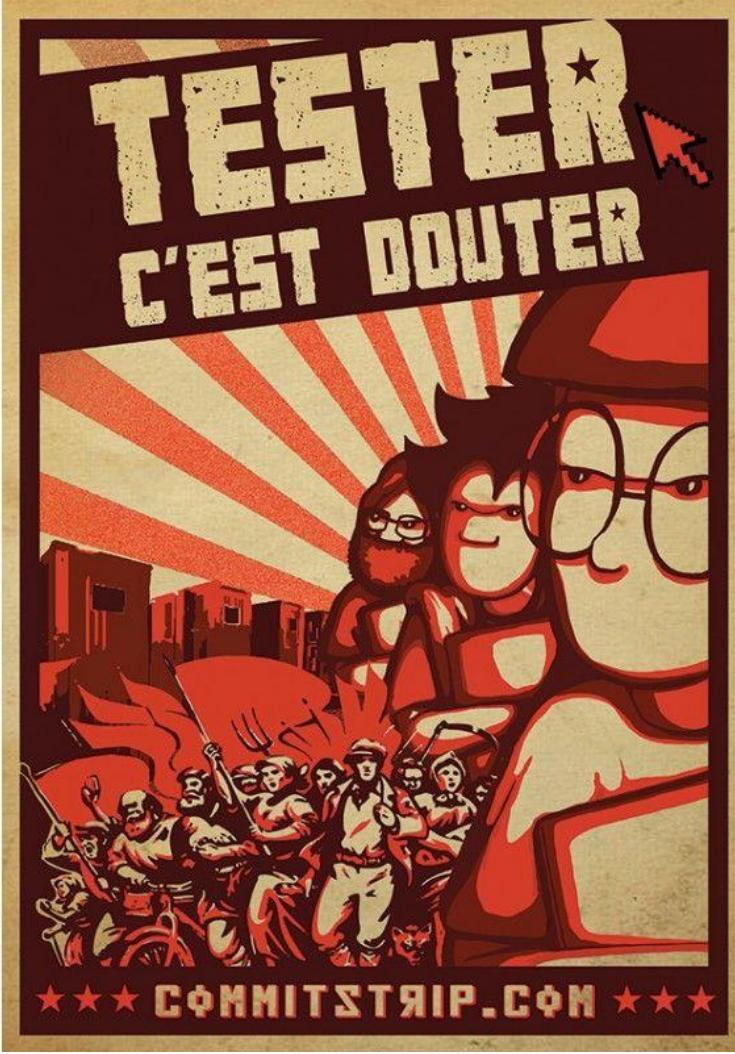
<https://npmjs.org/@axe-core/playwright>

```
const { AxeBuilder } = require('@axe-core/playwright');
const playwright = require('playwright');

(async () => {
  const browser = await playwright.chromium.launch({ headless: true });
  const context = await browser.newContext();
  const page = await context.newPage();
  await page.goto('https://dequeuniversity.com/demo/mars/');

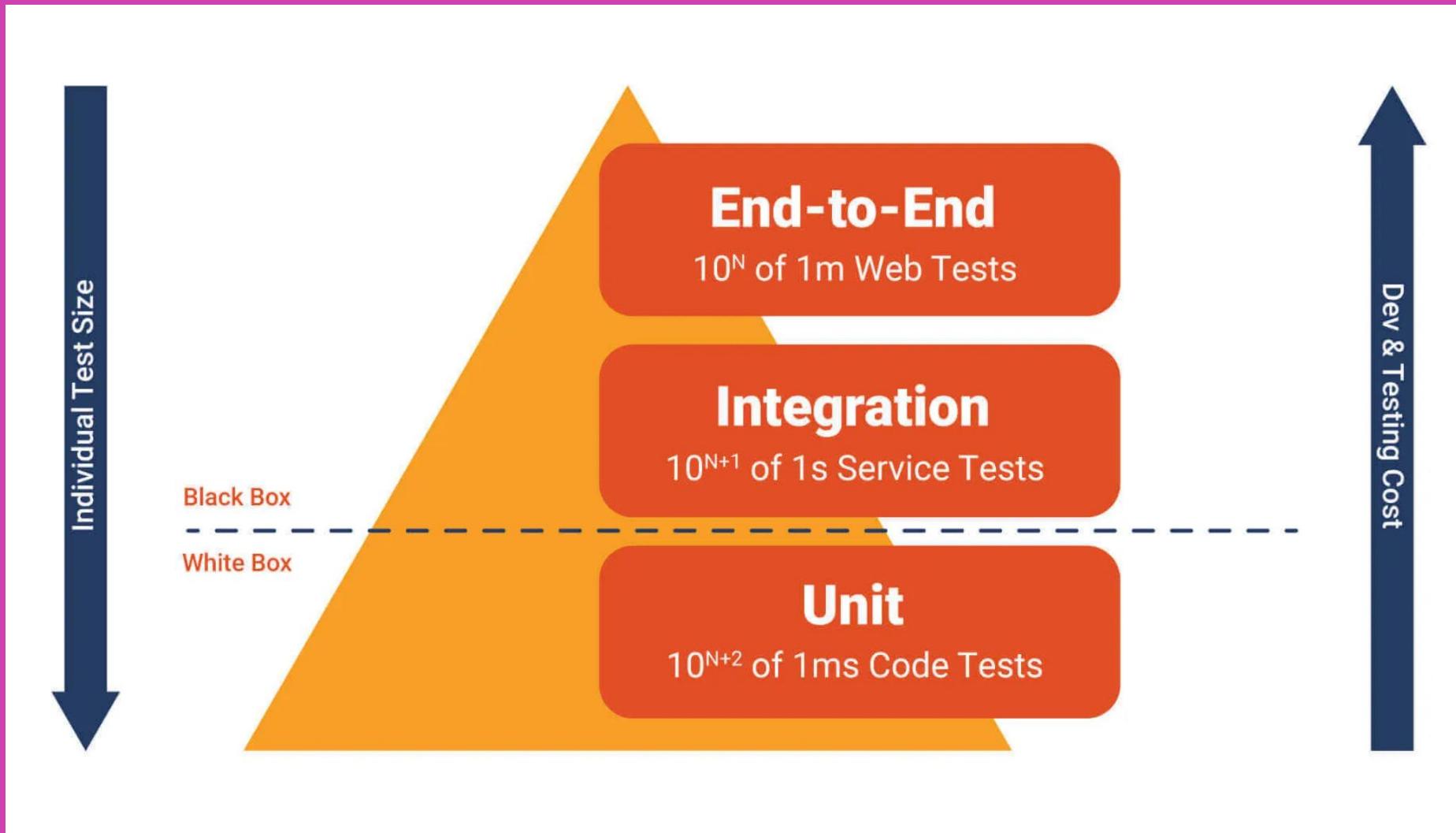
  try {
    const results = await new AxeBuilder({ page }).analyze();
    console.log(results);
  } catch (e) {
    // do something with the error
  }

  await browser.close();
})();
```



# TESTING ? TEST IS DOUBT

# TEST PYRAMID



# GET FEEDBACKS

- o Trace
- o Screenshots
- o Videos



## PLAYWRIGHT TESTING SERVICE

Exécuter des tests  
Playwright depuis Azure  
avec différentes  
combinaisons de  
navigateurs et d'OS

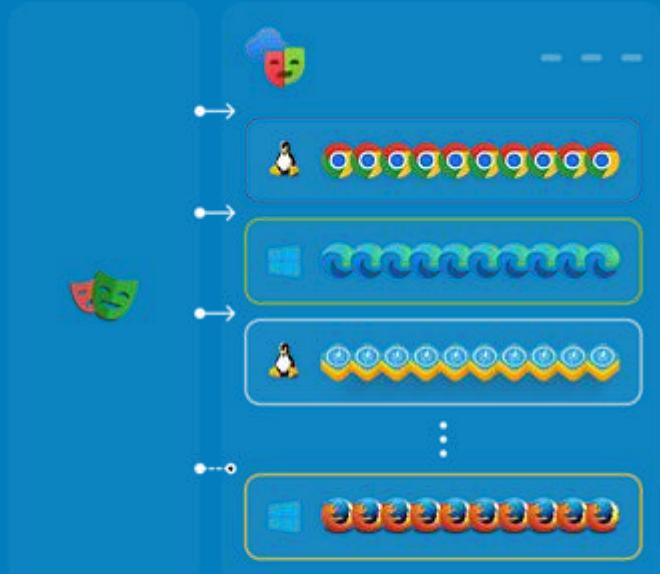




# RUN PLAYWRIGHT TESTS AT SCALE

## Exécuter les suites de tests plus rapidement

Distribuer les tests sur les navigateurs avec une parallélisation plus élevée dans CI pour exécuter les suites de tests plus rapidement.



# EXÉCUTER LES TESTS PLAYWRIGHT DE MANIÈRE FLEXIBLE



## Parallel browsers

Distribuez les tests pour qu'ils s'exécutent en parallèle sur un maximum de 50 navigateurs et dépasser les ressources locales pour accélérer les tests.



## Test scripts

Intégrer de manière transparente votre suite de tests existante sans modifier le code de test.



## OS and browser testing combinations

Utilisez n'importe quel appareil pour tester les systèmes d'exploitation Linux et Windows et les navigateurs modernes tels que Chromium, WebKit et Firefox.



## Endpoint testing

Testez des applications accessibles au public et en privé ou exécutez des tests sur un serveur de développement local.



## Consistent regional performance

Exécutez vos tests automatiquement sur des navigateurs hébergés dans la région Azure la plus proche.

# TARIFICATION



## Essai de 30 jours

Les 100 premières minutes de test sont gratuites avec un essai de 30 jours.

### Linux

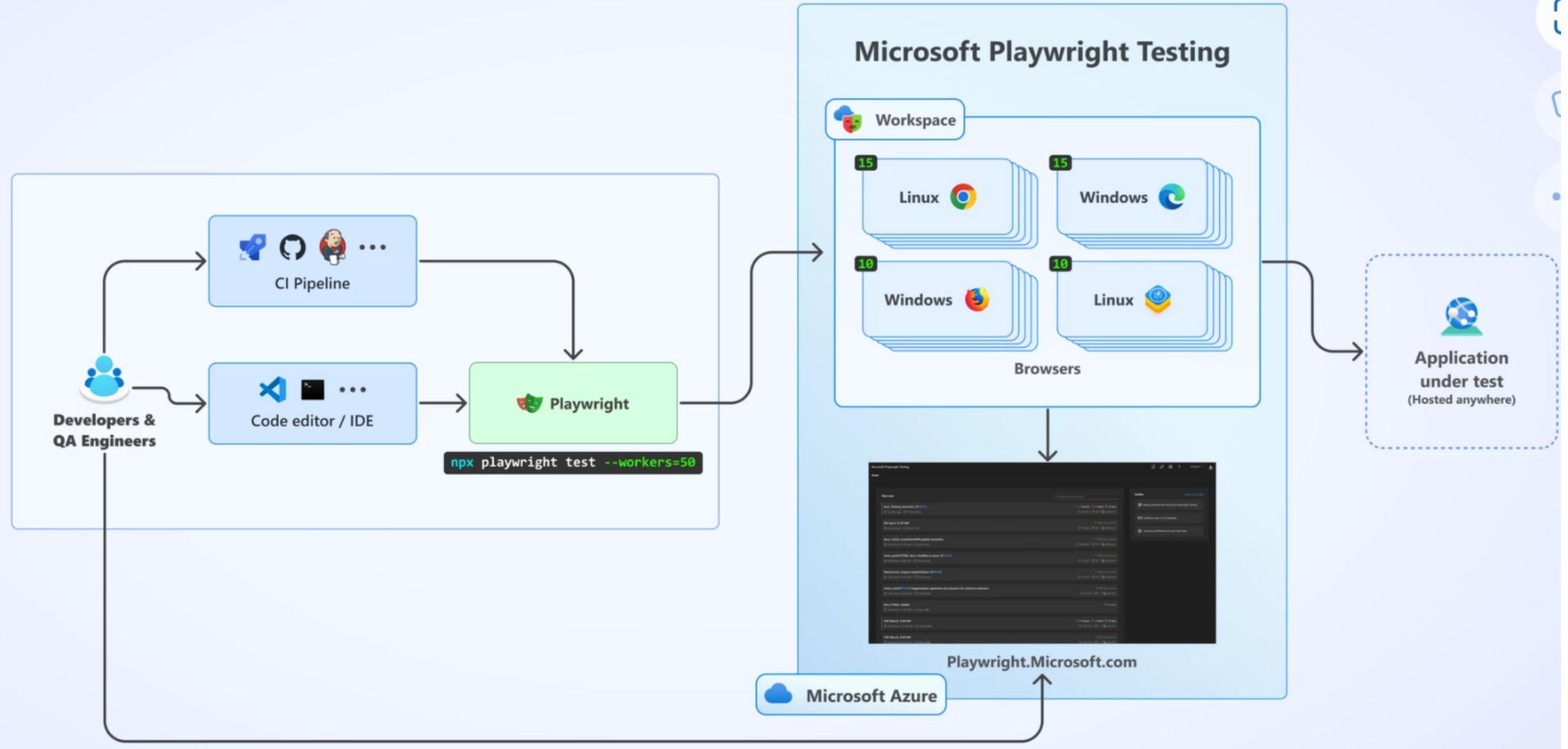
Système d'exploitation du navigateur cloud

**0,010 €** / 1 minute d'essai

### Windows

Système d'exploitation du navigateur cloud

**0,019 €** / 1 minute d'essai



# Azure Playwright Testing Service

The background features a dynamic, abstract design. On the left, a series of concentric, wavy lines in shades of pink, purple, and blue radiate from the bottom left towards the center. On the right, a large, semi-transparent green triangle is positioned at the top edge. The overall composition is modern and fluid.

# GOOD PRACTISES

# Trip & Tricks - Page object models

```
1  using System.Threading.Tasks;
2  using Microsoft.Playwright;
3
4  namespace BigCommerceApp.Tests.Models;
5
6  public class SearchPage
7  {
8      private readonly IPage _page;
9      private readonly ILocator _searchTermInput;
10
11     public SearchPage(IPage page)
12     {
13         _page = page;
14         _searchTermInput = page.Locator("[aria-label='Enter your search term']");
15     }
16
17     public async Task GotoAsync()
18     {
19         await _page.GotoAsync("https://bing.com");
20     }
21
22     public async Task SearchAsync(string text)
23     {
24         await _searchTermInput.FillAsync(text);
25         await _searchTermInput.PressAsync("Enter");
26     }
27 }
```

```
1  using BigCommerceApp.Tests.Models;
2
3  // in the test
4  var page = new SearchPage(await browser.NewPageAsync());
5  await page.GotoAsync();
6  await page.SearchAsync("search query");
```

# PLAYWRIGHT

reliable end-to-end testing for modern  
web apps

Any browser - Any platform - One API

Full Isolation - Fast execution

Powerful Tooling - Multi Language



# Christophe Peugnet - Adrien Clerbois

MICROSOFT MVP, TECHNICAL ARCHITECT @ SENSE OF TECH

Follow us on social media @tossnet1 @aclerbois