System Integration™

Playwright for .Net

BMV SYSTEM INTEGRATION PRIVATE LIMITED

Idea... Implementation... Innovation...

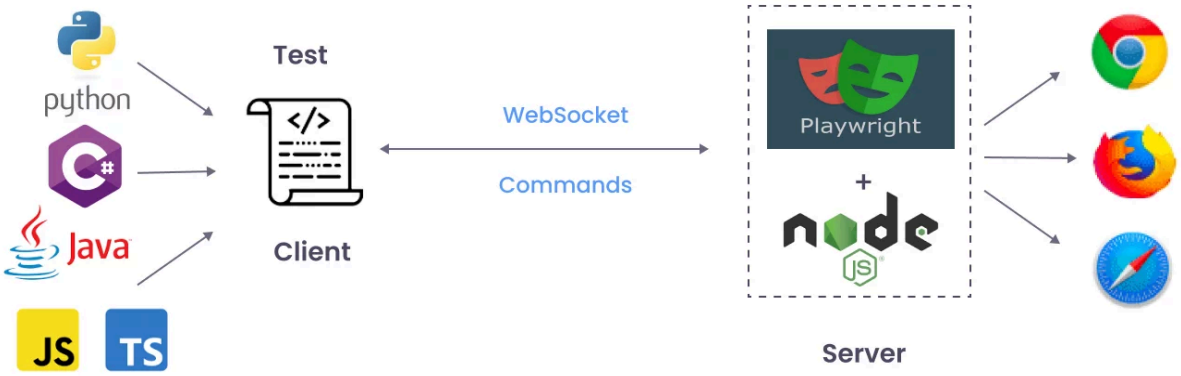
:: CORPORATE HEAD OFFICE ::

A503, The First,
Behind The ITC Narmada Hotel & Keshavbaug Party Plot,
Off 132 ft Road, Vastrapur, Ahmedabad.
Gujarat- 380015
Phone: +91 (79) 40 30 53 02
Website: www.systemintegration.in
Mail: info@systemintegration.in

Table of Content

Introduction

Playwright is an open-source Node.js library that enables efficient E2E testing of web apps.



Key Features

Cross-Browser, Cross-Platform, Cross-Language: Supports all modern rendering engines (Chromium, WebKit, Firefox) across Windows, Linux, and macOS. Can be used with TypeScript, JavaScript, Python, .NET, and Java.

Test Mobile Web: Native mobile emulation for Android (Chrome) and iOS (Safari), ensuring consistent testing experience across desktop and mobile environments.

Resilient Testing: Auto-wait feature eliminates flaky tests by waiting for elements to be actionable before performing actions. Web-first assertions and tracing capabilities enhance test reliability.

Full Isolation, Fast Execution: Creates a new browser context for each test, equivalent to a fresh browser profile, ensuring full test isolation with minimal overhead.

Powerful Tooling: Includes code generation for test creation, Playwright inspector for page inspection and test debugging, and trace viewer for comprehensive investigation of test failures.

Installation:

1. Start by creating a new project with dotnet new. This will create the PlaywrightTests directory which includes a UnitTest1.cs file:

NUnit:

```
dotnet new nunit -n PlaywrightTests
cd PlaywrightTests
```

MSTest:

```
dotnet new mstest -n PlaywrightTests
cd PlaywrightTests
```

2. Install the necessary Playwright dependencies:

NUnit:

```
dotnet add package Microsoft.Playwright.NUnit
```

MSTest:

```
dotnet add package Microsoft.Playwright.MSTest
```

3. Build the project so the playwright.ps1 is available inside the bin directory:

```
dotnet build
```

4. Install required browsers by replacing netX with the actual output folder name, e.g. net8.0:

```
pwsh bin/Debug/netX/playwright.ps1 install
```

If pwsh is not available, you have to install PowerShell.

Writing Tests:

The Playwright NUnit and MSTest test framework base classes will isolate each test from each other by providing a separate Page instance, where every test gets a fresh environment, even when multiple tests run in a single Browser.

NUnit:

```
using System.Threading.Tasks;
using Microsoft.Playwright.NUnit;
using NUnit.Framework;

namespace PlaywrightTests;

[Parallelizable(ParallelScope.Self)]
[TestFixture]
public class Tests : PageTest
{
    [Test]
    public async Task BasicTest()
    {
        await Page.GotoAsync("https://playwright.dev");
    }
}
```

Using Test Hooks

You can use SetUp/TearDown in NUnit or TestInitialize/TestCleanup in MSTest to prepare and clean up your test environment:

```
using System.Threading.Tasks;
using Microsoft.Playwright.NUnit;
using NUnit.Framework;

namespace PlaywrightTests;

[Parallelizable(ParallelScope.Self)]
[TestFixture]
public class Tests : PageTest
{
    [Test]
    public async Task MainNavigation()
    {
        // Assertions use the expect API.
        await Expect(Page).ToHaveURLAsync("https://playwright.dev/");
    }

    [SetUp]
    public async Task SetUp()
    {
        await Page.GotoAsync("https://playwright.dev");
    }
}
```

Generating Tests:

Playwright comes with the ability to generate tests out of the box and is a great way to quickly get started with testing

Running Codegen

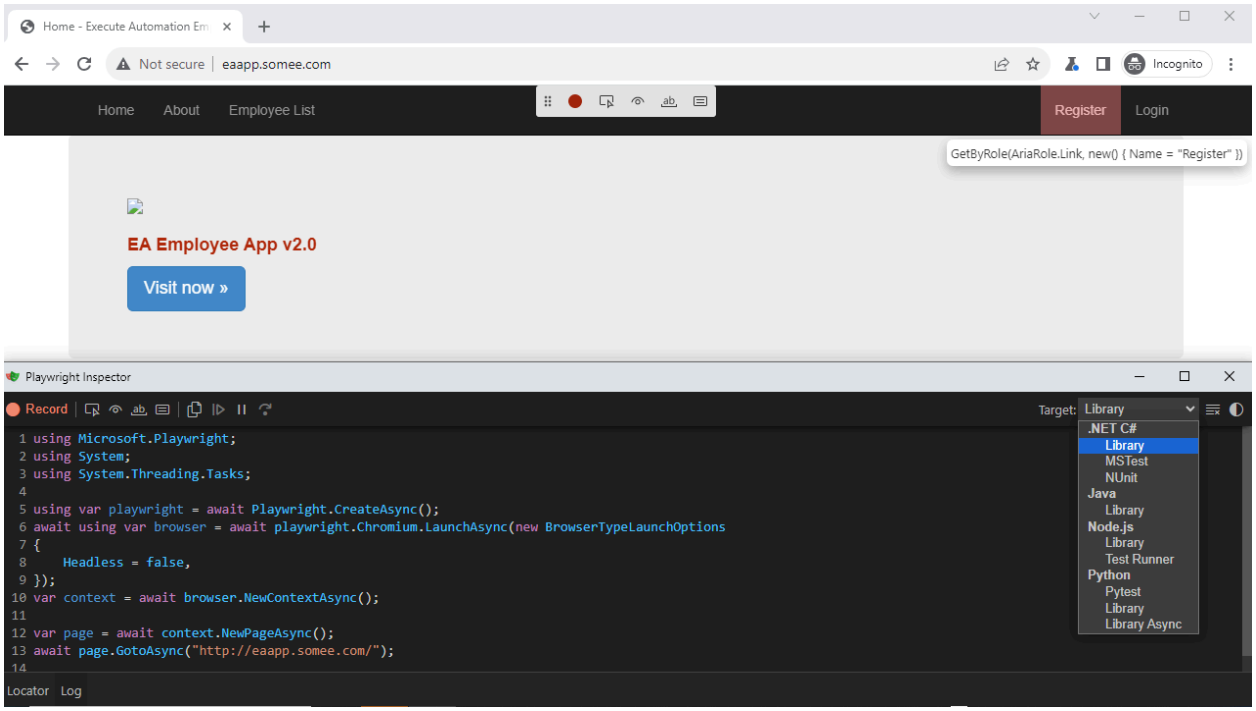
Use the codegen command to run the test generator followed by the URL of the website you want to generate tests for.

```
pwsh bin/Debug/netX/playwright.ps1 codegen demo.playwright.dev/todomvc
```

OR

open PowerShell in the directory where the "playwright.ps1" is located, and then execute the command: **./playwright.ps1 codegen url**

Here's a concise guide on how to record tests using Playwright's codegen:



Interact with the Page: Simply interact with the webpage as you would during regular testing, performing actions like clicks or filling forms.

Add Assertions: Utilize the toolbar to add assertions for elements on the page. This includes assertions for visibility, text content, or specific values.

Stop Recording: When finished interacting with the page, press the 'record' button to halt the recording process.

Copy Generated Code: Click the 'copy' button to obtain the generated code, which encompasses the interactions and assertions recorded during the session.

Clear Code: If needed, utilize the 'clear' button to reset the code and begin recording a new test session.

Generate Locators: Easily generate locators for elements on the page by clicking the 'Pick Locator' button. This allows for precise identification of elements within the generated code.

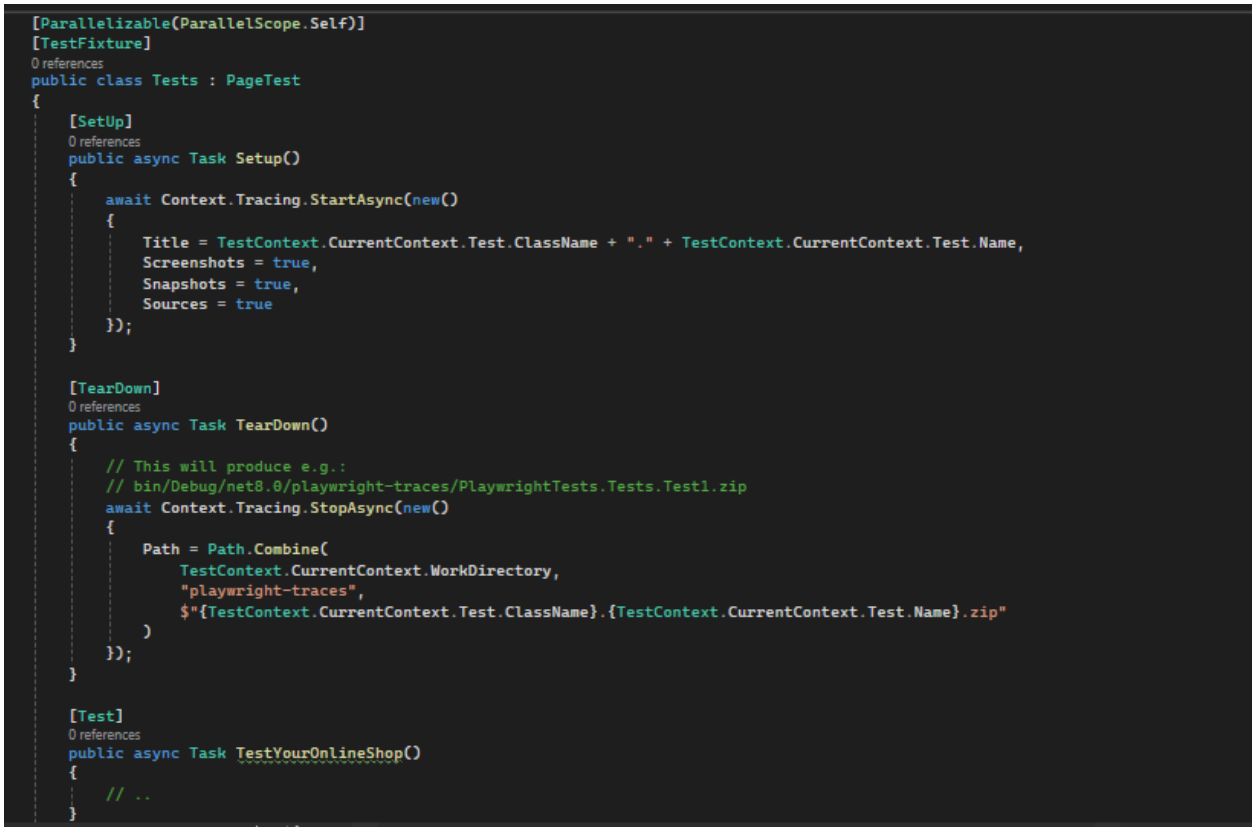
Fine-Tune Locators: Refine the generated locators in the playground to ensure accurate targeting of elements within the browser window.

Trace viewer

Playwright Trace Viewer is a GUI tool that lets you explore recorded Playwright traces of your tests meaning you can go back and forward though each action of your test and visually see what was happening during each action.

Recording a trace

Traces can be recorded using the BrowserContext.Tracing API as follows:



```
[Parallelizable(ParallelScope.Self)]
[TestFixture]
0 references
public class Tests : PageTest
{
    [SetUp]
    0 references
    public async Task Setup()
    {
        await Context.Tracing.StartAsync(new()
        {
            Title = TestContext.CurrentContext.Test.ClassName + "." + TestContext.CurrentContext.Test.Name,
            Screenshots = true,
            Snapshots = true,
            Sources = true
        });
    }

    [TearDown]
    0 references
    public async Task TearDown()
    {
        // This will produce e.g.:
        // bin/Debug/net8.0/playwright-traces/PlaywrightTests.Tests.Test1.zip
        await Context.Tracing.StopAsync(new()
        {
            Path = Path.Combine(
                TestContext.CurrentContext.WorkDirectory,
                "playwright-traces",
                $"{TestContext.CurrentContext.Test.ClassName}.{TestContext.CurrentContext.Test.Name}.zip"
            )
        });
    }

    [Test]
    0 references
    public async Task TestYourOnlineShop()
    {
        // ..
    }
```

This will record the trace and place it into the bin/Debug/net8.0/playwright-traces/ directory.

Opening the trace

You can open the saved trace using the Playwright CLI or in your browser on **trace.playwright.dev**. Make sure to add the full path to where your trace.zip file is located. This should include the test-results directory followed by the test name and then trace.zip.

```
pwsh bin/Debug/netX/playwright.ps1 show-trace "full_path_to_trace.zip"
```

Viewing Traces

Click through actions or hover on the timeline to see page changes. Inspect logs, source, and network. The viewer generates a DOM snapshot for full interaction.