JavaScript Playwright WordPress Plugin Automation Test

Author

Manan Ahmed Broti
https://ahmedmanan.com
https://github.com/AhmedManan
https://www.linkedin.com/in/ahmedmanan

December 25, 2025
version 1.0

Contents

1 Introduction

This documentation details the architecture, implementation, and deployment of a robust End-to-End automation framework designed to ensure the quality and reliability of some WP plugins. The core purpose of this document is to provide a comprehensive guide to the established testing environment. The framework utilizes JavaScript as the primary language, with Playwright for high-speed, reliable browser automation, and TestRunner as the powerful test runner. The structure follows strictly to the Page Object Model (POM) pattern, separating test logic from UI locators to maximize code reusability and significantly reduce maintenance overhead.

Finally, the entire testing lifecycle is integrated into a GitHub Actions CI/CD workflow, allowing tests to be executed automatically upon code changes, with detailed reports including HTML and Allure.

2 Project Setup

2.1 Prerequisites

Before running the playwright tests, ensure you have the following installed on your system:

- Node (Installed in your device)
- Java (Installed in your device)
- A Code Editor (VSCode is recommended)

2.2 Installation

- Clone this repository to your local machine.
- Install all prerequisites

To create a new playwright project, go to your project root directory and open terminal. Use the bellow command:

Listing 1: Terminal Session

```
npm init playwright@latest
```

When you first clone this repository, go to your project root directory and open terminal. Use the bellow command:

Listing 2: Terminal Session

```
npm install
```

You can also install the libraries manually.

Next step, install the browsers Playwright needs:

Listing 3: Terminal Session

```
npm install playwright
```

2.3 Setting Up Environment

A .env file is a plain text file used to store environment variables for an application, especially during local development or testing. It follows a simple key-value format, making it easy to manage configuration settings. To setup the project you need to create a .env file using the .env.example file provided in the project repository.

Listing 4: Default .env file used in the project

```
BASE_URL= 'Your Website URL'
ADMIN_USERNAME= 'Your Username'
ADMIN_PASSWORD= 'Your Password'
```

3 Run Tests & Report Generation

3.1 Running Tests

To execute the entire test suite across all browsers defined in your `playwright.config.js`, run:

Listing 5: Basic command to run all tests

```
npx playwright test
```

3.2 Running Specific Tests

To run a single test file:

Listing 6: Run specific spec file

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

To open the browser and visually watch the tests execute (useful for debugging):

Listing 7: Run tests in headed mode

```
npx playwright test --headed
```

3.3 Maintaining Test Sequence

By default, Playwright runs test files in parallel. To run tests within a file in a strict sequence (Serial Mode), use the following configuration:

Listing 8: Configuring serial execution in a test file

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

// This ensures tests run in order and fail-fast if one fails
test.describe.configure({ mode: 'serial' });

test('Step 1: Login', async ({ page }) => { /* ... */ });
test('Step 2: Checkout', async ({ page }) => { /* ... */ });
```

3.4 Generating Reports

3.4.1 Built-in HTML Report

To view the standard report after a run:

Listing 9: View Playwright HTML report

```
npx playwright show-report
```

3.4.2 Allure Report

If using Allure, generate and open the report using these commands:

Listing 10: Generate and open Allure report

```
npx allure generate allure-results --clean -o allure-report
npx allure open allure-report
```

4 Project Structure

This project utilizes the Page Object Model (POM) design pattern to enhance test maintainability and reduce code duplication. The directory structure is organized as follows:

4.1 tests/

Contains all executable test scripts, which use the `.spec.js` extension. These files focus exclusively on test logic, assertions, and workflow steps, while delegating low-level interactions to the Page Objects.

4.2 playwright.config.js

The central configuration file for the Playwright Test runner. It manages global settings, including browser projects (Chromium, Firefox, WebKit), timeouts, Base URLs, and reporter configurations (e.g., HTML, Allure).

4.3 pages/

Contains all Page Object Model (POM) classes. Each `.js` file represents a unique page or component of the application. These classes store locators and high-level action methods (e.g., `login()`, `checkout()`), abstracting the UI implementation away from the test scripts.

4.4 utils/

Stores helper scripts and custom utility functions. This includes common tasks such as custom authentication setups, date formatters, or data parsers used across the framework.

4.5 test-data/

Stores external data files, such as JSON or CSV files, used for data-driven testing. This separation allows test data to be updated without modifying the source code.

4.6 playwright-report/

The output directory for Playwright's native HTML reports. After a test run, this folder contains a self-contained report that can be viewed in any web browser.

4.7 test-results/

Contains artifacts captured during test execution, such as screenshots on failure, video recordings, and trace files. This is essential for debugging failed tests in CI environments.

4.8 package.json

The project manifest file for Node.js. It lists all required dependencies (e.g., `@playwright/test`, `allure-playwright`) and defines custom `npm` scripts for running tests and generating reports.

4.9 documents/

A dedicated directory for project documentation. This includes requirements, architecture diagrams, and the generated PDF versions of the test reports.

List of Figures