**Playwright Setup and Testing for Blazor Web Applications**

Contents

[1. Installation and Setup 1](#_Toc178602264)

[Node.js Requirement 2](#_Toc178602265)

[**Initializing Playwright** 2](#_Toc178602266)

[**Project Structure** 2](#_Toc178602267)

[**2. Playwright Management and Updates** 2](#_Toc178602268)

[**Starting Playwright (New Project Setup)** 2](#_Toc178602269)

[**Updating Playwright** 2](#_Toc178602270)

[**Installing Browser Dependencies** 2](#_Toc178602271)

[**Checking Playwright Version** 3](#_Toc178602272)

[**3. Running Tests** 3](#_Toc178602273)

[**Running All Tests** 3](#_Toc178602274)

[**Running a Specific Test** 3](#_Toc178602275)

[**Viewing HTML Reports** 3](#_Toc178602276)

[**Interactive Debugging with UI** 3](#_Toc178602277)

[**4. Running Tests for Specific Browsers** 3](#_Toc178602278)

[**WebKit (Safari Engine)** 4](#_Toc178602279)

[**Firefox** 4](#_Toc178602280)

[**Chromium (Chrome/Edge)** 4](#_Toc178602281)

[**5. Additional Notes and Useful Commands** 4](#_Toc178602282)

[**Running Tests in Headed Mode** 4](#_Toc178602283)

[**Using Test Filters** 4](#_Toc178602284)

[**Debugging with Breakpoints** 4](#_Toc178602285)

[**Taking Screenshots for Visual Testing** 4](#_Toc178602286)

[6. Testing Code 5](#_Toc178602287)

[Basic 5](#_Toc178602288)

[Interacting with UI Elements 6](#_Toc178602289)

[**Breakdown of the Code:** 6](#_Toc178602290)

[Dynamic/Blazor Specific Components 7](#_Toc178602291)

[**Breakdown of the Code:** 7](#_Toc178602292)

# 1. Installation and Setup

### Node.js Requirement

* Ensure you have the latest version of **Node.js** installed before starting with Playwright.
* Download it from [Node.js official website](https://nodejs.org/).
* To check if Node.js is installed or verify the version, run the following command in your terminal:

node -v

**Initializing Playwright**

1. Go to your Blazor project’s main folder.
2. Run the following command to initialize Playwright:

npm init playwright@latest

1. This command will:
   * Install Playwright dependencies.
   * Set up a **tests** folder containing .spec.js files (test scripts).
   * Download necessary browser binaries (Chromium, Firefox, WebKit).

**Project Structure**

* Once Playwright is initialized, the default folder structure will look like this:

tests/

├── example.spec.js (\*Sample test file generated by Playwright\*)

└── <your custom test files>

**2. Playwright Management and Updates**

**Starting Playwright (New Project Setup)**

* When setting up a new project or integrating Playwright into an existing Blazor project, initialize Playwright with:

npm init playwright@latest

**Updating Playwright**

* To update Playwright to the latest version, run:

npm install -D @playwright/test@latest

**Installing Browser Dependencies**

* If Playwright requires additional browser dependencies (e.g., for CI environments), install them using:

npx playwright install --with-deps

**Checking Playwright Version**

* To verify the current version of Playwright installed, run:

npx playwright --version

**3. Running Tests**

**Running All Tests**

* To run all tests across multiple browsers (Chromium, Firefox, WebKit), use:

npx playwright test

**Running a Specific Test**

* To run a specific test file (e.g., weather.spec.js), run:

npx playwright test weather.spec.js

**Viewing HTML Reports**

* After running tests, Playwright generates a detailed **HTML report**. To view the report, use:

npx playwright show-report

**Interactive Debugging with UI**

* For an more interactive test/running experience with real-time updates, use:

npx playwright test --ui

* + This will open a UI where you can browse test results, inspect errors, and re-run tests.

**4. Running Tests for Specific Browsers**

Playwright supports testing across multiple browsers, which is essential for cross-browser testing. Below are commands to run tests on specific browsers.

**WebKit (Safari Engine)**

* To run tests on the **WebKit** engine (Safari), use:

npx playwright test --project=webkit

**Firefox**

* To run tests in **Firefox**, use:

npx playwright test --project=firefox

**Chromium (Chrome/Edge)**

* To run tests on the **Chromium** engine (Chrome, Edge), use:

npx playwright test --project=chromium

**5. Additional Notes and Useful Commands**

**Running Tests in Headed Mode**

* Playwright runs in headless mode (no browser UI) by default. If you want to run it with the browser UI visible for debugging purposes, use:

npx playwright test --headed

**Using Test Filters**

* Playwright allows you to filter and run specific tests based on keywords. To run tests that match a particular keyword, use:

npx playwright test --grep "Login"

**Debugging with Breakpoints**

* You can pause test execution at any point using the following command in your test script:

await page.pause();

* + This opens the browser and pauses the test, allowing you to manually inspect the state.

**Taking Screenshots for Visual Testing**

* You can capture screenshots of specific elements or entire pages during your tests:

await page.screenshot({ path: 'example.png' });

\*Screenshots are useful for visual regression testing, ensuring that the UI does not change unexpectedly between releases.\*

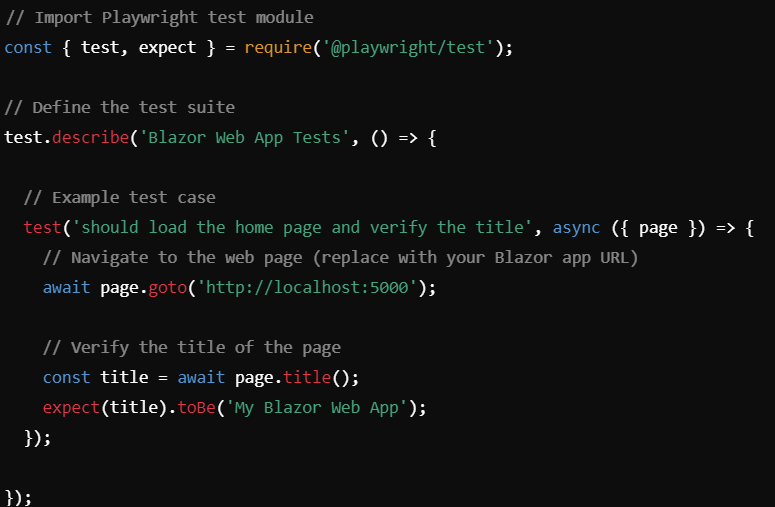
## 6. Testing Code

### Basic

A Playwright test script typically follows this structure:

* **Launching a browser**.
* **Navigating to a webpage**.
* **Interacting with UI elements**.
* **Verifying expected behaviour** (using assertions).

**Example code:**

****

**Breakdown of the Code:**

**Imports**: The test and expect functions from Playwright are imported to define tests and perform assertions.

**Test Suite**: test.describe() groups related tests into a suite. You can have multiple test cases inside a suite.

**Test Case**: test() defines a single test case. In this example, we are checking if the home page of the Blazor app loads and if the page title matches the expected value.

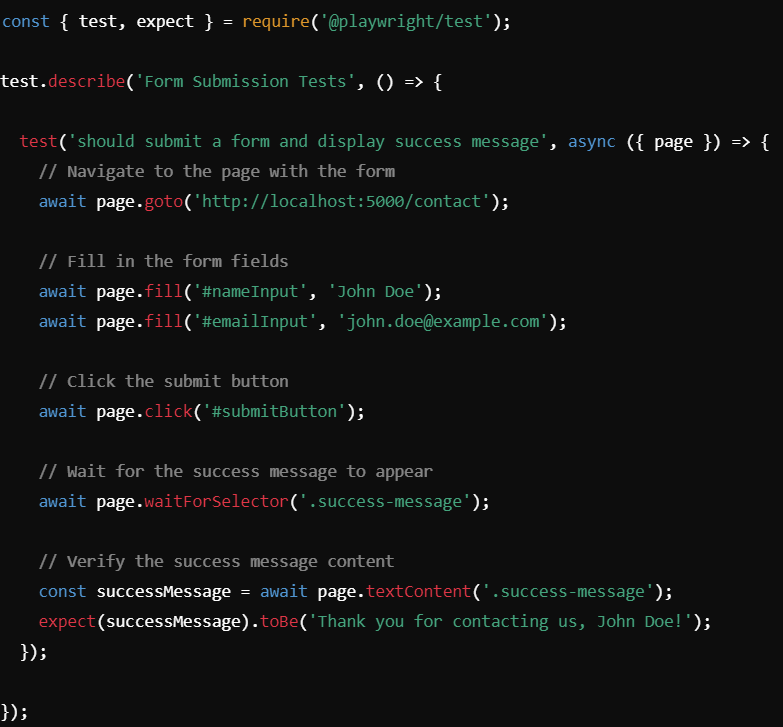
**Navigating to the Web Page**: page.goto() opens the specified URL in the browser.

**Assertion**: expect() checks that the page’s title matches the expected string. If the assertion fails, Playwright will mark the test as failed.

### Interacting with UI Elements

Playwright allows you to simulate user interactions, such as clicking buttons, typing into text fields, or selecting elements.

**Example code:**

****

**Breakdown of the Code:**

**Form Interactions**: page.fill(selector, value) is used to input text into form fields, where selector refers to the HTML element’s CSS selector (e.g., #nameInput) and value is the text to input.

**Clicking Buttons**: page.click(selector) simulates clicking the form’s submit button.

**Waiting for Elements**: page.waitForSelector(selector) waits for an element (such as the success message) to appear before proceeding with further actions.

**Assertions**: After submitting the form, the script checks the text content of the success message and asserts that it matches the expected string using expect().

### Dynamic/Blazor Specific Components

Since Blazor components render dynamically, there may be cases where the UI takes time to update due to server-side interactions or WebAssembly (WASM) loading times. Playwright can handle these with built-in waiting mechanisms.



**Breakdown of the Code:**

**Initial State Check**: page.textContent() retrieves the text content of the counter component to verify its initial value.

**Interacting with Blazor Components**: After clicking the increment button, Playwright waits for the counter value to update using page.waitForSelector().

**Dynamic Content Handling**: Since Blazor components may take time to re-render, we ensure that Playwright waits for the DOM to update before verifying the new counter value.