HH Global QA Automation Framework Assignment

By Brad Hoyt, July 1st, 2025

This project demonstrates a basic Playwright-based QA automation framework for testing the HH Global homepage (https://www.hhglobal.com) across desktop and mobile devices.

Objectives

- Validate that key UI elements render and function as expected.
- Confirm navigation works across desktop and mobile views, including the hamburger menu.
- Establish a scalable foundation for broader test coverage.
- Showcase how Playwright can be used effectively for cross-device QA.

Project Structure

```
├── tests/
│   ├── desktop/
│   │   ├── homepage-desktop.spec.ts
│   │   ├── navigation-desktop.spec.ts
│   │   └── navigation-hamburger-desktop.spec.ts
│   ├── mobile/
│   │   ├── homepage-mobile.spec.ts
│   │   └── navigation-mobile.spec.ts
│   ├── home.spec.ts
│   └── ui-elements.spec.ts
├── helpers/
│   └── acceptCookies.ts
├── playwright.config.ts
└── README.md
```

The current folder structure separates desktop and mobile for clarity during this assignment, but future growth should consolidate tests by feature area, using tagging to target devices.

For easy reference, here's a blurb on setting up Playwright:

To run the tests, open a terminal, navigate to the hhglobal-playwright directory, and install dependencies with npm install. If you haven't already installed Playwright browsers, run npx playwright install. For the best development experience, you can also install the official Playwright Test for VS Code extension. Once everything is set up, use the commands below to execute the tests.

To run only desktop tests:

```
npx playwright test tests/desktop --project="chromium"
```

To run only mobile tests:

```
npx playwright test tests/mobile --project="Mobile Chrome"
```

Local Test Results:

Desktop Tests: Passed
When run in Parallel with 3 workers: All 9 tests Passed:

```
PS C:\Users\bradh\Desktop\automation\hhglobal-playwright> npx playwright test tests/desktop --project="chromium"

Running 9 tests using 4 workers
  9 passed (44.2s)
```

All passed individually and consecutively with one worker.

Mobile Tests: Passed
When run in Parallel with 3 workers: All 7 tests Passed:

```
PS C:\Users\bradh\Desktop\automation\hhglobal-playwright> npx playwright test tests/mobile --project="Mobile Chrome"

Running 7 tests using 4 workers
  7 passed (33.8s)
```

All passed individually and consecutively with one worker.

iOS and Cross Platform Browser Testing

While Playwright's device emulation reliably validates responsive layouts, especially with the Chromium and Mobile Chrome projects, I would recommend using a 3rd party tool like Browserstack for running tests on real devices. I would especially recommend this for iOS devices. Playwright's Mobile Safari emulation is not reliable, and the only way to test on Mac operating systems from a PC would be to use a tool like this. Also, using Browserstack can help catch touch-specific and rendering issues that emulators may miss. When running the tests for this project, I recommend running the desktop tests with Chromium (--project="chromium") and mobile tests with Mobile Chrome (--project="Mobile Chrome").

Overview

Desktop Tests

Below is a summary of each test file and its purpose.

`tests/home.spec.ts`

Purpose:

Basic smoke test confirming the homepage loads.

Checks:

- Page title includes "HH Global."

`tests/ui-elements.spec.ts`

Purpose:

Verifies that core homepage elements are present.

Checks:

- Navigation bar visibility.
- Hero banner visibility
- Footer visibility.

`tests/desktop/homepage-desktop.spec.ts`

Purpose:

Tests core homepage functionality on desktop viewport.

Checks:

- Hero Section: "Learn More" button navigation.
- Data Cards Section: Clicking the first data card.
- Contact Form: All fields are visible and fillable.
- Map Section: Clicking the UK map pin displays details.
- Footer: "Terms & Conditions" link navigation.

`tests/desktop/navigation-desktop.spec.ts`

Purpose:

Tests primary navigation menu and submenus on desktop.

Checks:

- Each top-level menu navigates correctly.
- Submenus open on hover and link correctly.

`tests/desktop/navigation-hamburger-desktop.spec.ts`

Purpose:

Validates the hamburger menu when the window is resized smaller on desktop.

Checks:

- Hamburger icon visibility and functionality.
- All parent and submenu links navigate correctly.

Mobile Tests

Below is a summary of each test file and its purpose.

`tests/home.spec.ts`

Purpose:

Basic smoke test confirming the homepage loads.

Checks:

- Page title includes "HH Global."

`tests/ui-elements.spec.ts`

Purpose:

Verifies that core homepage elements are present.

Checks:

- Navigation bar visibility.
- Hero banner visibility.
- Footer visibility.

`tests/mobile/homepage-mobile.spec.ts`

Purpose:

Verifies homepage elements and navigation on mobile devices.

Checks:

- Hero Section navigation.
- Data card navigation.
- Contact form fillability.
- Footer link navigation.

`tests/mobile/navigation-mobile.spec.ts`

Purpose:

Tests the mobile hamburger menu navigation.

Checks:

- Hamburger menu expands.
- Parent and submenu links function correctly.

Helper

`helpers/acceptCookies.ts`

Provides a function to dismiss the cookie consent overlay consistently across all tests.

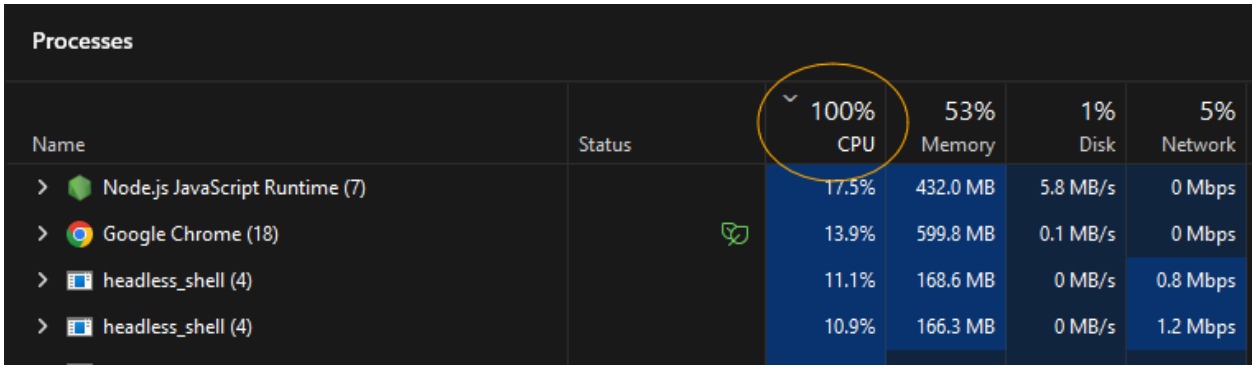
A note about the Playwright configuration file
Due to my computer performance, I made the following adjustments to the Playwright config file to compensate:

- I increased the timeout from 30000 to 90000.
- I changed the number of retries from 0 to 2.
- I limited the number of workers that would work in parallel to 3. (the default was 4)

```
export default defineConfig({
  testDir: './tests',
  timeout: 90000,
  fullyParallel: true,
  forbidOnly: !!process.env.CI,
  retries: 2, // Switch to 0 retries for clarity when writing tests
  reporter: 'html',
  workers: 3, // Limit to 3 workers due to CPU performance
```

Additional note: When writing and running new tests, I change the retries value to 0 so I can catch all failures.

I made these changes because the CPU on my laptop was maxing out which caused some tests to periodically not pass. See:



Scalability Strategy

Here are some key factors I would focus on when designing a scalable test framework:

Test Organization:
Tests could be structured by *feature area* rather than just desktop vs. mobile, making the suite intuitive and maintainable. For example:

```
/tests
  /homepage
    hero.spec.ts
    navigation.spec.ts
    cards.spec.ts
    footer.spec.ts
  /who-we-are
    company.spec.ts
    strategic-partners.spec.ts
    sustainable-growth.spec.ts
    inspiring-solutions.spec.ts
    leadership.spec.ts
  /what-we-do
    creative-digital.spec.ts
    conscious-creative.spec.ts
    technology.spec.ts
    procurement.spec.ts
    logistics.spec.ts
  /case-studies
    case-studies.spec.ts
  /news
    news.spec.ts
  /careers
    careers.spec.ts
  /common
    contact-forms.spec.ts
    language-selector.spec.ts
    footer-links.spec.ts
```

This makes it easy to expand as new pages, flows, or components are added without cluttering the project.

Tagging & Filtering:
Instead of having separate desktop and mobile folders with a lot of duplication across folders, I could apply `test.describe` tags or naming conventions and use Playwright's `--grep` flag to run targeted subsets (like desktop, mobile, smoke tests, regressions). So, for example, All `describe()` blocks would have either `@desktop`, `@mobile`, etc, and then use `--grep` to focus as needed when running tests. (Example: npx playwright test --grep @mobile)

Reusable Helpers:
Common actions, such as accepting cookies, interacting with the language selector, or validating contact forms, are consolidated into the `helpers/` directory. This avoids duplication and ensures consistency across all specs.

Page Object Models:
As the suite grows, implementing Page Objects for key pages like the homepage, navigation bar, and contact forms, etc. will further improve readability and maintainability. Page Objects put all the page's buttons, links, and actions together in one place. This makes your tests easier to understand and quicker to update when something on the page changes.

Device Cloud Integration: *(Using a 3rd party tool for testing on iOS and other actual devices)*
When ready to expand to real devices and test iOS devices more reliably, the framework should integrate seamlessly with BrowserStack or Sauce Labs. (My experience has been with Browserstack) We can switch to running tests on real hardware by updating the Playwright configuration to connect to a remote `wsEndpoint`, with no changes required to the test logic.

Continuous Integration:
Running tests through a CI pipeline (such as Jenkins, CircleCI, or GitHub Actions) increases reliability by ensuring tests always run in a clean, consistent environment. This helps catch issues early and reduces the chance of local setup differences causing problems. This also ensures that tests can be run in a clean environment whenever code changes, which would reduce the chance of missed issues.

🤖 AI Use Disclosure

To ensure transparency, here is how AI tools were used on this project:

- **Template Generation:** AI was used to help create the initial boilerplate code for Playwright test files.
- **Selector Research:** Occasionally used AI to help identify recommended page elements and selectors while inspecting the DOM.
- **Refactoring:** Assisted in consolidating the logic for accepting cookies into a reusable helper function.
- **Troubleshooting:** Provided recommendations for handling timing issues, overlays, and improving selector reliability.
- **Documentation:**
  Helped outline and refine the README and supporting notes.

All code, configurations, and documentation were either created, edited, or reviewed and finalized manually to ensure accuracy and maintain full ownership of the solution.

🛠️ My Contributions

Here's what I personally built and reviewed in this project:

- Designed the framework with desktop and mobile folders initially, and developed a feature-based folder strategy to support future scaling.
- Developed all test cases by analyzing the page DOM, finding stable selectors, and adding them to the project structure.
- Created and configured the Playwright setup, including adjustments to timeouts, retries, and worker limits for better reliability.
- Wrote a reusable helper function for accepting cookies to avoid duplication.
- Solved the challenge of testing the hamburger menu across desktop and mobile devices, making sure navigation worked consistently everywhere.
- Verified tests in Chromium and Mobile Chrome environments to cover multiple viewports.
- Researched and applied the right wait conditions to handle dynamic content and overlays.
- Created the Scalability Strategy to outline how the framework can grow, including plans for Page Objects and CI integration.
- Reviewed and finalized all code and documentation to ensure clarity, maintainability, and accuracy.

CONTACT

HH Global QA Automation Framework Assignment

By Brad Hoyt, July 1st, 2025
Email: bradhoyt@gmail.com
Phone: +420 734 702 874
LinkedIn: https://www.linkedin.com/in/bradjhoyt/