UI Automation Testing using Playwright with Java – Afour Tech

AFOUR TECHNOLOGIES
An ACL Digital Company
UI AUTOMATION TESTING
USING PLAYWRIGHT WITH JAVA
In this blog, we will explore playwright to see that how UI automation testing can be done using Playwright with Java and handle few of these common challenges.
READ MORE!
More Information
contact@afourtech.com
Visit Our Website
www.afourtech.com

A few common challenges with UI automation that increase maintenance and project cost are:

1. UI change, i.e., change in locators, e.g., Xpath, CSS
2. Flakiness due to loading issues
3. Cross-browser compatibility and maintenance
4. Debugging challenges

At **AFour Technologies**, we continuously try to address these issues by exploring new tools and innovating/creating customized features that help customers build robust solutions and reduce overall costs.
In this blog, we will explore Playwright to see how UI automation testing can be done using Playwright with Java and handle a few of these common challenges.

Introduction to Playwright

The Playwright is an open-source test automation library that Microsoft initially developed. It supports multiple browsers like Chromium, Firefox, and Web Kit. It supports multiple languages like Typescript, JavaScript, Java, Python, .Net, C#, and multiple platforms like

Windows, Linux, Mac OS, Android, and iOS.
Playwright aims to provide fast, reliable, user-friendly automation capabilities for web application testing. It offers a wide range of features that make it useful for various tasks.

There are three main components of a playwright:

1. **The browser driver**—It controls the browser engine and executes commands from the playwright API.
2. **The browser context**—Represents a single instance of a browser with its own cookies, cache, and other settings. Playwright supports multiple browser contexts within the same browser instance, which can be used for separate sets of tests or scenarios, and each browser context has its own set of pages.
3. **The page**—Represents a single web page within a browser context with its own DOM tree, frame hierarchy, and other content.

Example test:
When writing tests with Playwright, it offers "assertThat" overloads that intelligently wait until the expected condition is met.
There is a list of **assertions** such as:
.isChecked(), .isEnabled(), .isDisabled(), .isEditable(), .isVisible() & many more are there.
Locators are crucial in Playwright's auto-waiting and retry mechanism, enabling efficient element identification on web pages. They provide a means to locate elements and perform various actions like click(), fill(), and many more. Additionally, Playwright allows creation of custom Locators using the "page.locator()" method.
Also, Playwright provides multiple **actions**, for example ->
– To input text it provides fill() method,
– For checkboxes and radio buttons, it provides setChecked(),
– For selecting one or multiple options, it has selectOptions(),
– For clicking it has click(), dblClick() methods,
– It provides a method called type() which enters string character by character, and many more are there.

Playwright offers several features that make it an excellent choice for UI automation testing

Auto-waiting:
Playwright performs thorough checks on elements to ensure that actions behave as expected. It automatically waits for all necessary conditions to be met before executing the requested

action. The action fails if the required conditions are not met within the specified timeout. This approach guarantees that actions are performed at the right time, enhancing the reliability and accuracy of UI automation.
For example: If we want to perform a click() action on an element, then Playwright ensures that

1. Is an element attached to the DOM or not
2. Element is visible or not
3. Is it stable or not
4. It is enabled, and it receives events or not

And after performing all required checks, it performs the click action.

Headless and Non-headless mode:
Run your automated tests in headless mode or non-headless mode.

Debugging Tests:
The Playwright Inspector is a graphical user interface (GUI) tool that facilitates debugging your Playwright tests. It allows you to live-edit locators, pick locators, and view logs. Additionally, you can utilize the “page. pause()” method to debug your test cases effectively. This approach eliminates the need to manually navigate through each test action to reach the desired debugging point, streamlining the debugging process.

Test Generator:
Playwright includes a powerful test generation feature that allows you to perform actions in the browser. Playwright figures out the best locator on the page; for that, it prioritizes role, text, and testID locators.
If the element generator discovers multiple elements that match the same locator, it enhances the locator to ensure uniqueness, enabling us to identify each component accurately.
Command:
mvn exec: Java -e -D exec.mainClass=com.microsoft.playwright.CLI -D exec.args=” codegen demo. Playwright.dev/todomvc”
When running the Codegen command, two windows will open a browser window to interact with the page element and a playwright inspector to record a test.

Emulation:
We can use a test generator to generate tests using emulation so that we can generate a test for a specific viewport size, specific device, or specific color scheme; also, we can emulate geolocation, language, or timeZone.

First, let’s see **emulate viewport size:**
Playwright allows you to open a browser window with a specific width and height using the “viewport” option. This feature enables you to generate tests with different viewport sizes, providing flexibility in emulating various screen resolutions and responsive designs.
Command:
mvn exec:java -e -D exec.mainClass=com.microsoft.playwright.CLI -D exec.args=”codegen— viewport-size=800,600 playwright.dev”

Emulate Device:
We can use the command mentioned below:
mvn exec:java -e -D exec.mainClass=com.microsoft.playwright.CLI -D exec.args=’codegen— device=”iPhone 13″ playwright.dev’
We can use the —device option in the command, which sets the viewport size and user agent.
We can also write a script for this:
Playwright provides a helpful feature to emulate different devices and their characteristics, allowing you to test your web application’s behavior on various devices without needing the physical device. You can use the “setViewportSize” and “setUserAgent” methods to emulate the device properties. Here’s an example:

Intercepting Network request
Playwright offers a powerful feature that enables us to modify seamlessly or block network requests initiated by a web page. This can be useful for testing purposes like simulating slow network conditions, mocking server responses, or testing the web application’s behavior under specific conditions.
Example:
So, in this example, we are setting up a network route for the requested URL. The lambda expression is called when a matching request is made. We are setting custom messages, status codes, content type, and body as our custom messages.
Because we have set up a network route for the URL, our custom response will be returned instead of the regular page content, as shown below:

1. **Accessibility Testing**

Accessibility testing is essential to ensuring that a web application is usable for all users, including those with disabilities. Playwright provides built-in support for performing accessibility testing.

2. Authentication

Playwright provides an isolated environment called the browser context for executing tests. This allows tests to load an existing authenticated state, eliminating the need to log in for each test. By saving the authentication state in the context, it can be reused across all tests. This approach saves time by avoiding redundant login operations and ensures complete isolation between independent tests.

3. Downloads

With Playwright’s downloads feature, you can automate downloading files from a website. This feature helps test scenarios that involve verifying the correct download of files or automating tasks that involve downloading files. Furthermore, Playwright allows you to customize the behavior of the downloads feature by specifying the download path or filtering the types of files that can be downloaded.

4. Trace Viewer

The Playwright trace viewer is a tool that allows you to analyze and visualize the recorded trace data of a browser’s actions during test execution. Let’s see some options that provide detailed insights into the sequence of events, network requests, and other relevant information.

Command:

```
mvn exec: Java -e -D exec.mainClass=com.microsoft.playwright.CLI -D exec.args=” show-trace”
```

Actions:

Once the trace is opened in the Playwright trace viewer, you will find a list of actions that the Playwright performed on the left-hand side. Selecting each action from the list allows you to explore detailed information such as the action snapshot, action log, network log, and more.

Metadata:

The Playwright trace viewer offers comprehensive information, including the timing of actions, the browser engine utilized, viewport details, mobile device emulation, and the count of recorded pages, actions, and events.

Screenshots:

Each trace records a screenshot and renders it as a film strip. This film strip provides a visual representation of the recorded actions and states. Hovering over each frame in the film strip lets you view a magnified image corresponding to a specific action or state, allowing for detailed inspection and analysis.

Call:
The call option includes details such as the action name, the timestamp when it was called, the duration of the action's execution, the parameters passed to the action, the return value, and any associated log messages.

Console:
The Playwright's trace viewer Console panel offers a view of the console output generated during the recorded actions. It allows you to observe console logs, errors, warnings, and other relevant messages produced during the test case execution.

Network:
It provides a view of all network requests made by the browser, along with their corresponding responses, headers, timings, and other relevant information.

Selenium And Playwright

Both are popular choices for web automation testing. There are some key differences.

1. **Architecture:**

In Selenium, each command is transmitted as an HTTP request, and the corresponding JSON response is received. Every action, such as launching the browser, clicking an element, or entering text into a textbox, is transmitted as a specific HTTP request. This approach results in the termination of the connection between the server and client, which needs to be re-established for the subsequent request.
Connection termination after each request can lead to slower execution, which also introduces a layer of flakiness.
On the other hand, Playwright utilizes a single web socket connection to handle all requests during the test execution. This architectural approach minimizes potential points of failure and enables commands to be transmitted smoothly over a single connection. This makes Playwright a more stable tool as compared to Selenium.

2.Speed:
The Playwright is faster than Selenium when it comes to executing tests due to its architecture.

3.Flakiness:

Playwright is designed to be less flaky than Selenium again, thanks to its architecture that isolates each test in a separate browser context. This means that tests do not interface, reducing the chances of flakiness.

4.Cross-browser consistency:

Playwright is designed to provide a consistent API across multiple browsers, ensuring that the test can be written once and run across different browsers. Selenium requires browser-specific drivers, which may cause inconsistencies in behaviour and functionality across different browsers.

5.Debugging:

Both Selenium and Playwright provide good debugging capabilities, but Playwright offers additional features that make debugging easier. Playwright provides an interactive mode that allows you to interact with the browser as the test is running, which helps you to identify issues more quickly.

6.Testing framework:
Selenium and Playwright can be used with testing frameworks like Jest, Mocha, and Jasmine. However, Playwright has its test runner, making it easier to start with testing without learning a separate framework.

Conclusion:
In conclusion, Playwright, combined with Java, provides a robust and up-to-date solution for UI automation. Its unified API, ability to work across different browsers, and wide range of features streamline the automation process and improve testing efficiency. As a result, it is a popular choice for **Test Automation Services** and is often used as a foundation for **Test Automation Framework**. The architecture of Playwright, which utilizes browser-specific binaries and maintains persistent web socket connections, guarantees stability and reliability throughout test execution.
For detailed information and additional examples, please refer to the links in the references section.