Introduction to Playwright

Modern Browser Automation

November 2023

Written by

Prashanth Marappa
Senior Technology Consultant

EY
Building a better working world

Contents/agenda slide

1. What is Playwright?

2. What does the architecture of Playwright look like ?

3. What are the Key Features of Playwright?

4. Contrast the Playwright with Other testing tools

5. What are the challenges and limitations of Playwright ?

6. Conclusion

7. Demo

What is Playwright?

- Playwright is an open-source testing and automation framework
- Developed by Microsoft in 2020
- Supports Smoke, Regression, Data-driven, Behavioral –driven, E2E Testing, ADA Testing, API Testing

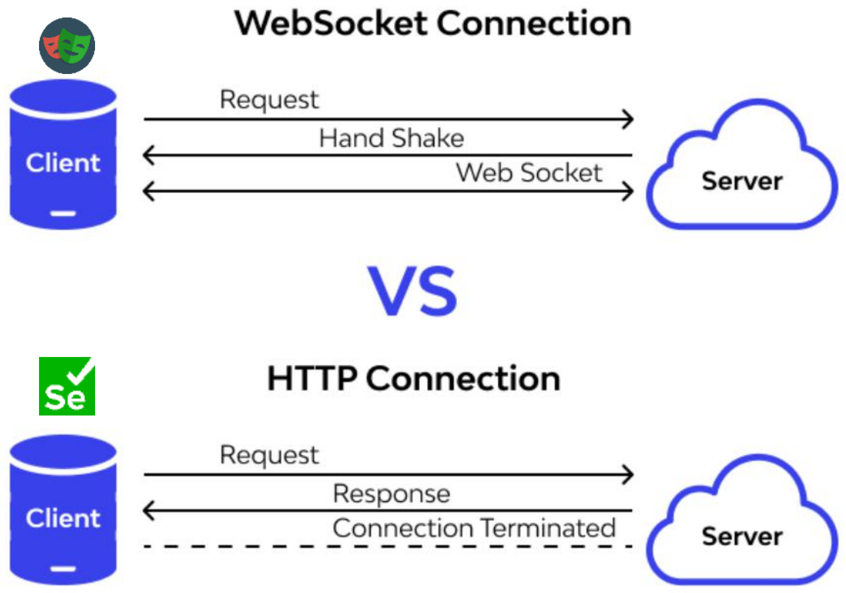
Poll time

The better the question. The better the answer.
The better the world works.

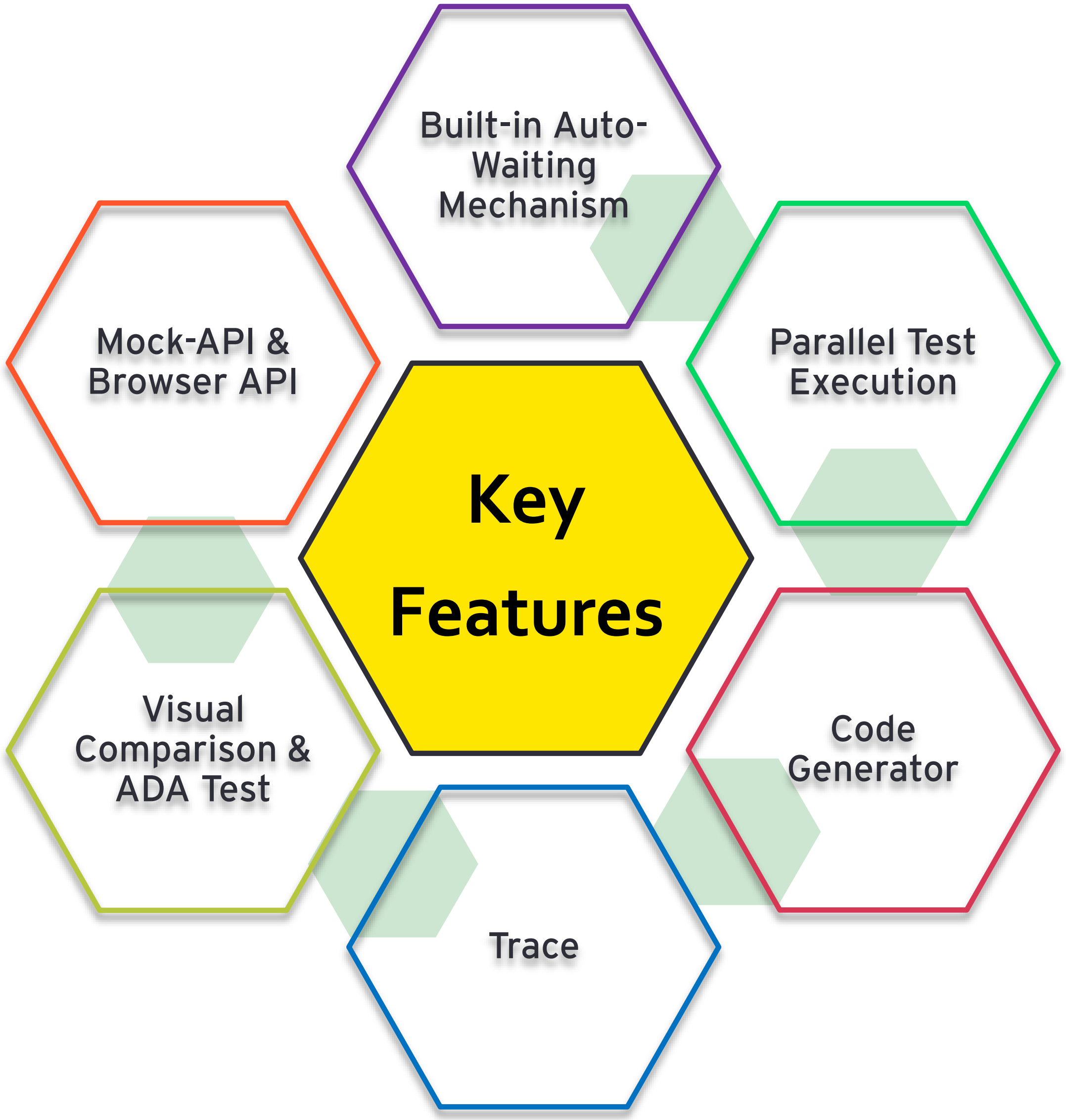
Building a better working world

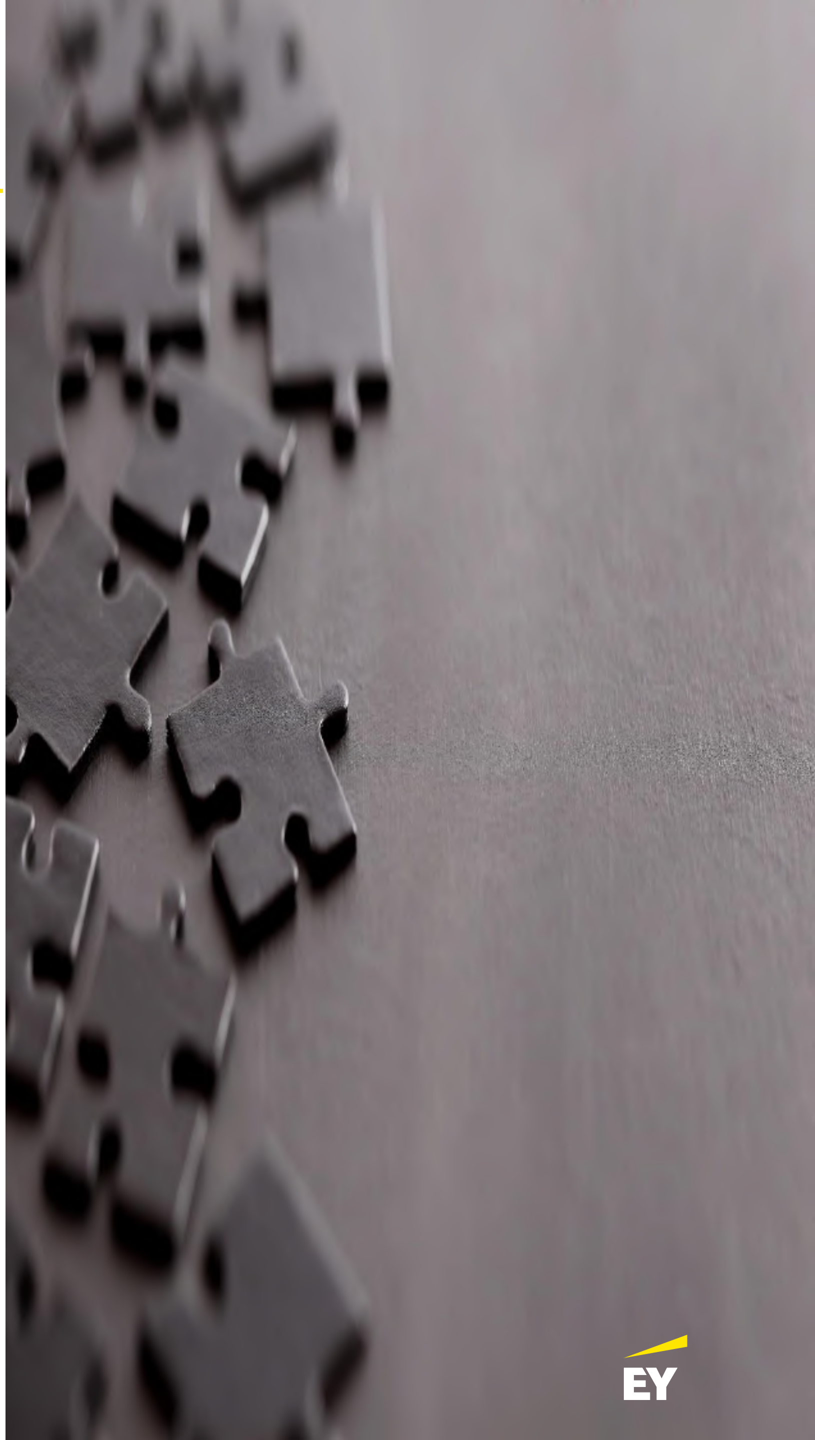
Architecture

Playwright's architecture works on web socket connection and communicates all requests through one single web socket connection. This helps in less failures due to bi-directional communication and faster execution



Features


Built-in Auto-Waiting Mechanism
Mock-API & Browser API
Parallel Test Execution
Key Features
Visual Comparison & ADA Test
Code Generator
Trace



Auto Waiting

Playwright runs several actionability checks on the elements to ensure that they behave as expected. It waits for all necessary checks to pass before performing the specified action.

For example, for page.click('#login'), Playwright will ensure that:

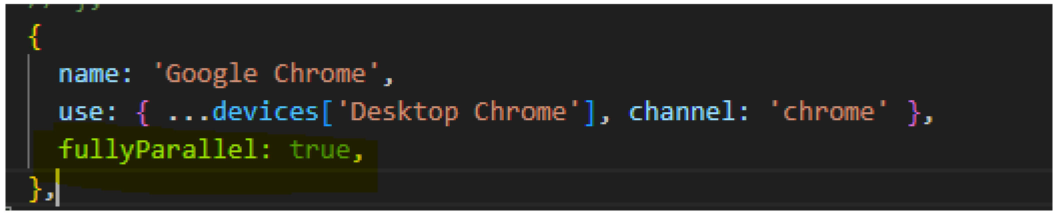
- ✓ element is Attached to the DOM
- ✓ element is Visible
- ✓ element is Stable, as in not animating or completed animation
- ✓ element Receives Events, as in not obscured by other elements
- ✓ element is Enabled

Playwright also gives the option to turn off auto-wait page.click('#login', force);

Parallel Test Execution

Playwright Test runs concurrently. All tests are executed in worker processes. These processes that run independently and are orchestrated by the test runner. Each worker has the same environment and launches their own browser.

For example, setting the parallel option true for chrome browser in configuration file



```
{
  name: 'Google Chrome',
  use: { ...devices['Desktop Chrome'], channel: 'chrome' },
  fullyParallel: true,
},
```

Execution Command:

npx playwright test --workers 4

Poll time

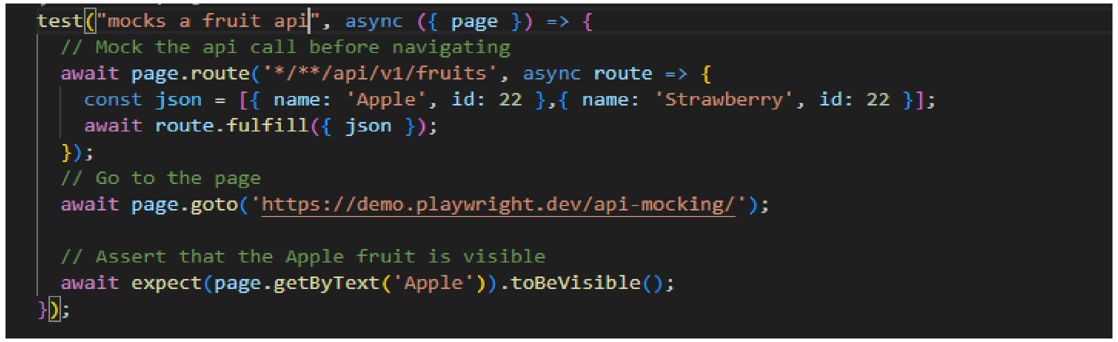
The better the question. The better the answer.
The better the world works.

Building a better
working world

Mock API

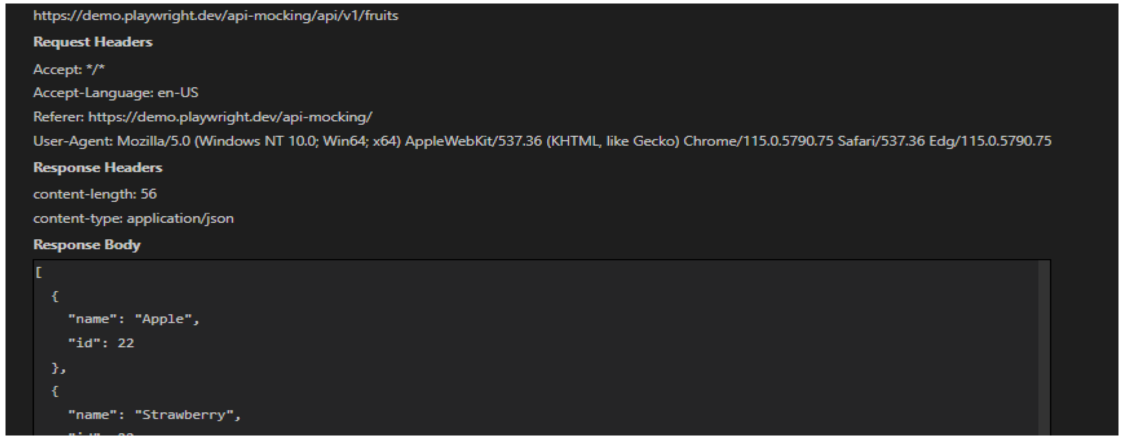
Playwright provides APIs for mocking and modifying HTTP and HTTPS network traffic. Any page request, can be tracked, edited, and mocked.

For example, Below fruits api is mocked with apple and strawberry data

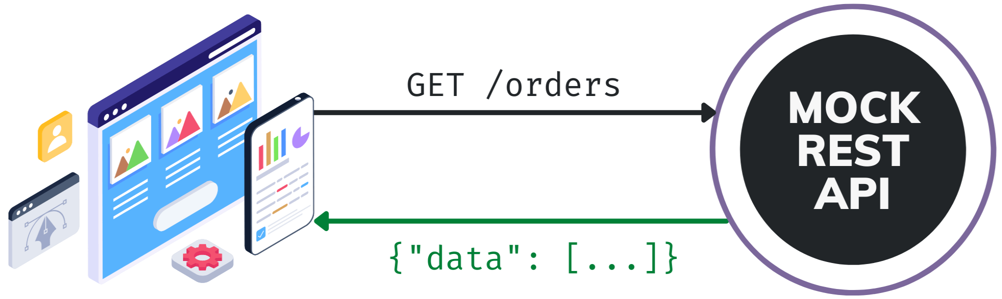


```
test("mocks a fruit api", async ({ page }) => {
  // Mock the api call before navigating
  await page.route('*/**/api/v1/fruits', async route => {
    const json = [{ name: 'Apple', id: 22 },{ name: 'Strawberry', id: 22 }];
    await route.fulfill({ json });
  });
  // Go to the page
  await page.goto('https://demo.playwright.dev/api-mocking/');

  // Assert that the Apple fruit is visible
  await expect(page.getByText('Apple')).toBeVisible();
});
```



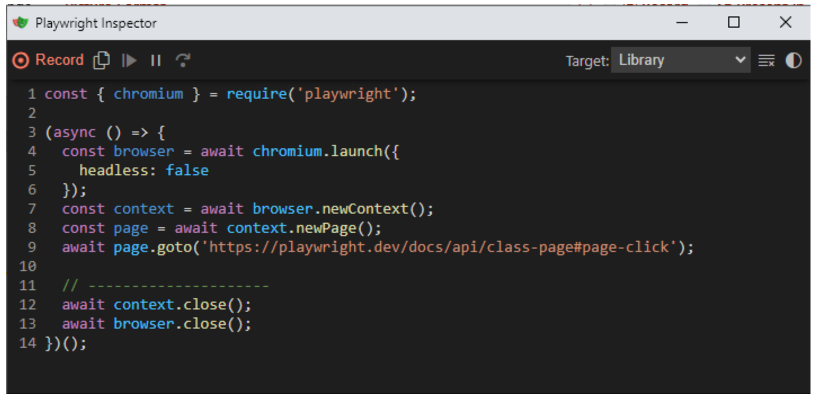
```
https://demo.playwright.dev/api-mocking/api/v1/fruits
Request Headers
Accept: */*
Accept-Language: en-US
Referer: https://demo.playwright.dev/api-mocking/
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.5790.75 Safari/537.36 Edg/115.0.5790.75
Response Headers
content-length: 56
content-type: application/json
Response Body
[
  {
    "name": "Apple",
    "id": 22
  },
  {
    "name": "Strawberry",
```



Code Generator

Playwright Codegen is a powerful utility that helps produce code snippets for any browser automation activities for several programming languages. This eliminates the need for you to manually develop all the automation code, saving you time and effort.

Command:
npx playwright codegen



```
const { chromium } = require('playwright');

(async () => {
  const browser = await chromium.launch({
    headless: false
  });
  const context = await browser.newContext();
  const page = await context.newPage();
  await page.goto('https://playwright.dev/docs/api/class-page#page-click');

  // ---------------------
  await context.close();
  await browser.close();
})();
```

Poll time

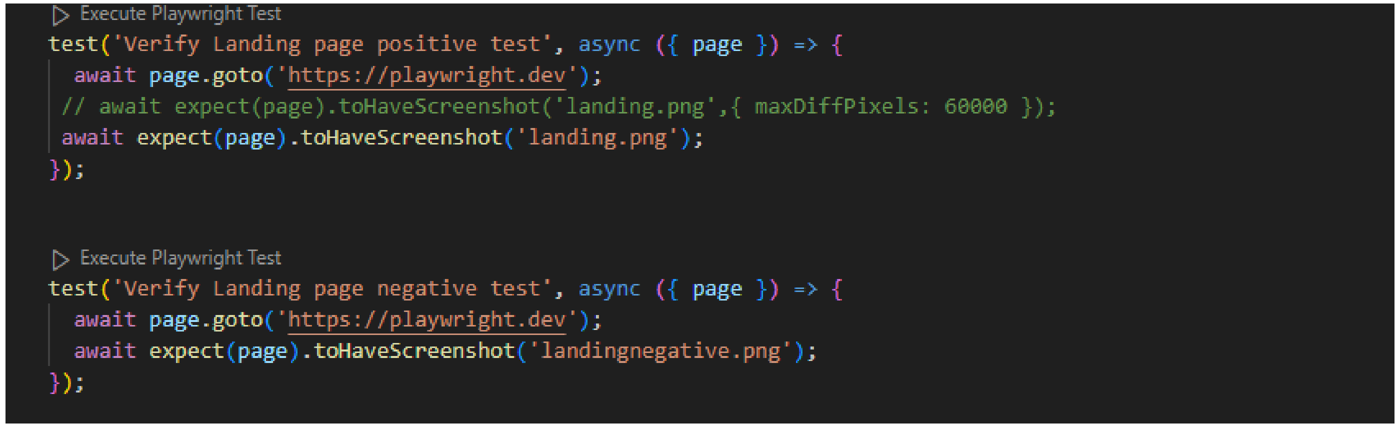
The better the question. The better the answer.
The better the world works.

Building a better
working world

Visual Comparison & ADA Testing

Playwright allows you to compare visual appearance of web pages across different browsers by using pixel by pixel comparison method. Playwright provides ability to ignore some pixels by setting up threshold. It can also take snapshots of dom elements and compare.

Playwright can be used to test your applications for many types of accessibility issues , it successfully matches criteria of Web Content Accessibility Guidelines (wcag2a, wcag2aa, wcag21a and wcag21aa)



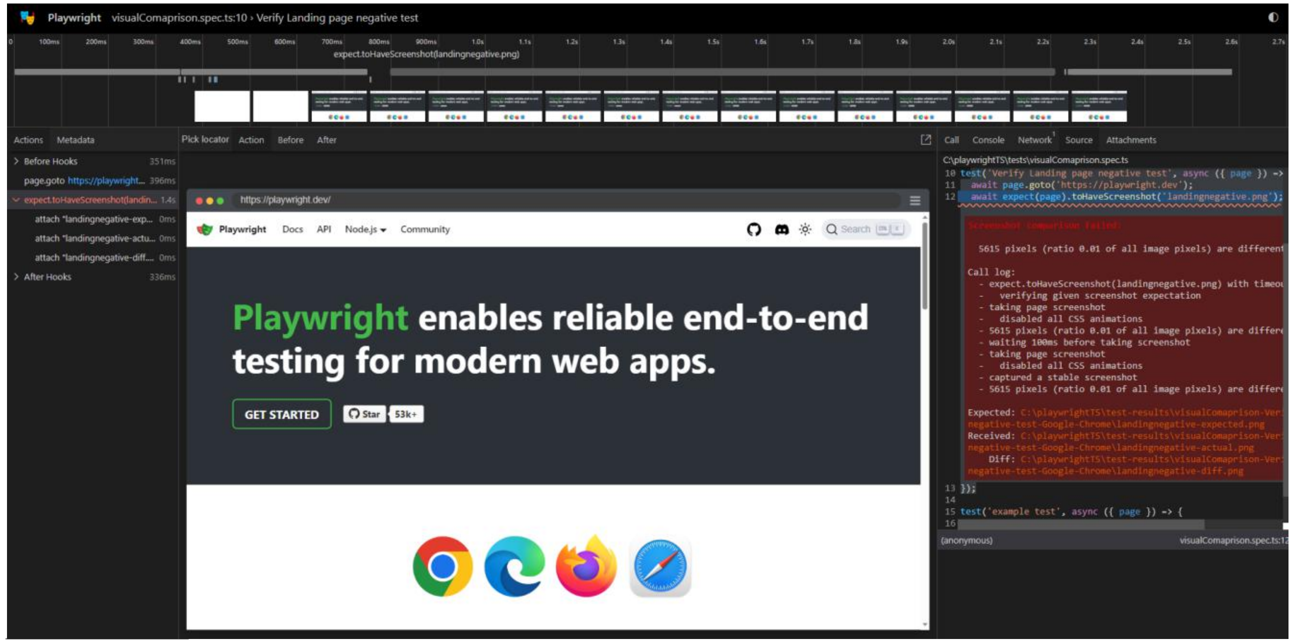
```
▷ Execute Playwright Test
test('Verify Landing page positive test', async ({ page }) => {
  await page.goto('https://playwright.dev');
 // await expect(page).toHaveScreenshot('landing.png',{ maxDiffPixels: 60000 });
 await expect(page).toHaveScreenshot('landing.png');
});


▷ Execute Playwright Test
test('Verify Landing page negative test', async ({ page }) => {
  await page.goto('https://playwright.dev');
  await expect(page).toHaveScreenshot('landingnegative.png');
});
```

Trace

Playwright Trace Viewer is a GUI tool that helps you explore recorded Playwright traces after the script has completed execution. This option is helpful for verifying test failures and documenting defects.

- Captures dom snapshots
- Captures dev logs with network components
- Captures action events


Playwright visualComaprison.spec.ts:10 › Verify Landing page negative test
expect.toHaveScreenshot(landingnegative.png)
Actions
Metadata
Pick locator
Action
Before
After
Call
Console
Network
Source
Attachments
https://playwright.dev/
Playwright
Docs
API
Node.js
Community
Search
Playwright enables reliable end-to-end testing for modern web apps.
GET STARTED
Star
53k+

```
Before Hooks                          351ms
page.goto https://playwright...       396ms
expect.toHaveScreenshot(landin...     1.4s
  attach "landingnegative-exp...      0ms
  attach "landingnegative-actu...     0ms
  attach "landingnegative-diff...     0ms
After Hooks                           336ms
```

C:\playwrightTS\tests\visualComaprison.spec.ts

```
test('Verify Landing page negative test', async ({ page }) =>
  await page.goto('https://playwright.dev');
  await expect(page).toHaveScreenshot('landingnegative.png');
```

```
Screenshot comparison failed:

  5615 pixels (ratio 0.01 of all image pixels) are different

Call log:
  - expect.toHaveScreenshot(landingnegative.png) with timeou
  -   verifying given screenshot expectation
  - taking page screenshot
  -   disabled all CSS animations
  - 5615 pixels (ratio 0.01 of all image pixels) are differe
  - waiting 100ms before taking screenshot
  - taking page screenshot
  -   disabled all CSS animations
  - captured a stable screenshot
  - 5615 pixels (ratio 0.01 of all image pixels) are differe

Expected: C:\playwrightTS\test-results\visualComaprison-Ver
negative-test-Google-Chrome\landingnegative-expected.png
Received: C:\playwrightTS\test-results\visualComaprison-Ver
negative-test-Google-Chrome\landingnegative-actual.png
    Diff: C:\playwrightTS\test-results\visualComaprison-Ver
negative-test-Google-Chrome\landingnegative-diff.png
```

```
});

test('example test', async ({ page }) => {
```

(anonymous) visualComaprison.spec.ts:12

Poll time

Comparison with other test tools

Feature	Playwright	Cypress	Selenium	TestCafe
Language Support	JavaScript, TypeScript, Python, C#, Java	JavaScript	Multiple languages through WebDriver	JavaScript
Browser Support	Chromium, Firefox, WebKit (cross-browser)	Chromium	Various browsers through WebDriver	Chromium, Firefox, Edge (cross-browser)
Architecture	Uses browser-specific drivers	Built-in Electron-based browser	Uses browser-specific drivers	Uses proxy-based approach
Async/Await	Fully supports async/await syntax	Supports async/await syntax	Requires explicit use of promises	Fully supports async/await syntax
Cross-Platform	Yes	Limited (focused on web and Electron apps)	Yes	Yes
Parallel Testing	Built-in support for parallel execution	parallel execution only allows in multiple machines	Can be achieved with external tools like TestNg	Built-in support for parallel execution
Headless Mode	Yes	Yes	Yes	Yes
Performance	Faster than all	Generally fast execution	Slower due to browser interaction overhead	Generally fast execution
DOM Snapshot	Supports capturing DOM snapshots	Built-in support for capturing screenshots	Limited support for capturing DOM snapshots	Supports capturing DOM snapshots
Auto-Waiting	Yes	Yes	Requires explicit waiting strategies	Yes
Community	Growing community and corporate support https://playwrightsolutions.com/	Active community and corporate support	Established community and wide adoption	Active community and continuous development
Learning Curve	Moderate to easy (APIs similar to Puppeteer)	Moderate (unique architecture)	Moderate (complex setup for distributed)	Moderate to easy (simple and intuitive)
Popularity	Growing popularity	Popular among certain dev circles	Widely used in industry	Growing popularity

Challenges and Limitations

While Playwright offers numerous benefits, it also has some challenges and limitations.

1. Limited Ecosystem or Community compared to Selenium
2. Community Size and Support
3. Mobile device Native app support missing
4. Not all programming languages supports Playwright native features

Conclusion

Playwright is a promising choice for web automation and testing, especially if cross-browser compatibility, design and execution speed is a priority and if you want to leverage features like parallel execution and headless mode. It will be very effective for In-Sprint automation.

Demo