Playwright Overview

Playwright is an open-source framework for end-to-end testing of web applications. It supports multiple browsers, including Chromium, Firefox, and WebKit, and provides capabilities for testing modern web apps efficiently. Playwright is known for its cross-browser automation, powerful API, and robust support for testing in different languages like JavaScript, Python, C#, and Java.

Application Supported:- Web browser /apps,mobile web apps and API's.
Language Supported:- Javascript, Typescript, Java, Python and .Net .
Browsers supported:- Chromium ,Webkit(Safari) and Firebox(headed/headless).
OS supported:- Windows, MacOS, Linux ,Support CI Runs.

Features of Playwright

1. Free and open source
2. Multi Browser and Multi-language
3. Easy setup and configurations
4. Functional, API's and Accessibility Testing
5. Build-in reporters, Custom reports
6. CI ,CD and Docker
7. Parallel Testing
8. Auto Wait(No need of implicit and explicit Wait
9. Built in Assertions
10. Multi Tab and Multi Window

Advantage of Playwright

Multi-Browser Support:

- Playwright supports multiple browser engines, including Chromium, Firefox, and WebKit (used by Safari). This allows you to write tests that can be executed across different browsers.

Browser Contexts:

- Playwright introduces the concept of browser contexts, which are independent instances of a browser. Each context has its own cookies, cache, and other session-specific data, making it useful for scenarios where you need isolated browser sessions.

Headless and Headful Mode:

- Playwright allows you to run browsers in headless mode (without a graphical user interface) or headful mode (with a graphical user interface). This flexibility is useful for various testing and debugging scenarios.

Device Emulation:

- Playwright provides built-in device emulation, allowing you to simulate different devices and screen sizes. This is essential for testing the responsiveness of web applications.

Network Interception and Mocking:

- Playwright enables you to intercept and modify network requests, facilitating testing scenarios where you need to mock responses or simulate different network conditions

Parallel Test Execution:

- Playwright supports parallel test execution, improving the efficiency of your test suites by running tests concurrently.

Automated Screenshots and Videos:

- Playwright makes it easy to capture screenshots and record videos during test execution. This is useful for debugging and reviewing test results.

Selective Test Execution:

- You can selectively run specific tests or groups of tests using tags or patterns, allowing for targeted testing of specific features.

Page and Browser Events:

- Playwright provides events for various browser and page events, such as page load, network request/response, console messages, and more. You can listen to these events and take actions accordingly.

User Input Simulation:

- Playwright allows you to simulate user interactions, including clicks, keyboard inputs, and mouse movements, providing comprehensive coverage for testing user interfaces.

Visual Testing:

- Playwright supports visual testing, allowing you to compare screenshots or entire page layouts to detect visual regressions in your web application.

Integration with Test Runners:

- Playwright integrates seamlessly with popular test runners like Jest, Mocha, and Jasmine, making it easy to incorporate Playwright into your existing testing workflows.

Robust Selector Engine:

- Playwright's selector engine supports a wide range of selectors, including CSS, XPath, and custom selectors, making it versatile for locating elements on the page.

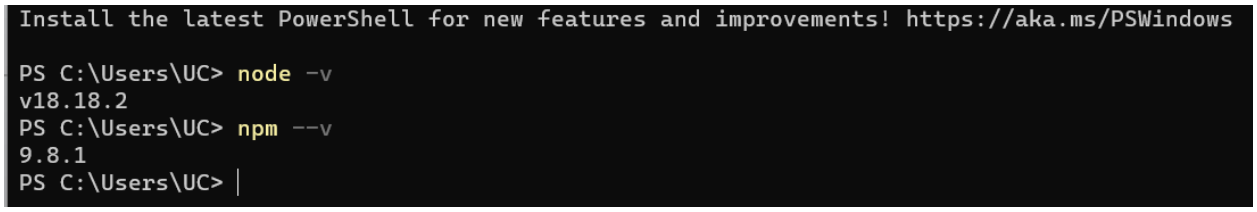
Method 1: Installations of Playwright with node.js (manually)

Step 1: Install Node.js and npm:

If you haven't already, install Node.js and npm from https://nodejs.org/.(I hope VS code is already installed, else install VS code as well.

Verify the installations by running the following commands in your terminal or command prompt:

1. *node -v*
2. *npm -v*



```
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\UC> node -v
v18.18.2
PS C:\Users\UC> npm --v
9.8.1
PS C:\Users\UC> |
```

Step 2: Install Visual Studio Code

Download and install Visual Studio Code from VS Code https://code.visualstudio.com/ .

Step 3: Create a New Project

Open Command Prompt and create a new directory for your Playwright project:

```
1. mkdir playwright-project
2. cd playwright-project
3. npm init -y
```

Step 4: Install Playwright

Install Playwright as a dependency for your project:

```
1. npm install playwright
```

Step 5: Create a Test File

Inside your project folder, create a new file, for example, ***test.js***. This file will contain your Playwright test script.In your test.js file, write a basic Playwright test.

Step 8: Run Your Playwright Test

Back in your Command Prompt, run the test script:

```
1. node test.js
```

(Note: if you are following above steps, make sure you have added *chromium* ***extension in your browser or you can go with below steps where all requirement added automatically with one click)***

Method 2: Installations of Playwright with node.js (with VS code extension)

Step 1. Install node.js and VS code
Step 2. Goto VS code market section
Step 3. Add the Playwright extension
Step 4. Tap on View CTA of VS code
Step 5. Tap on Command Palette(Ctrl+shift+p)
Step 6. Search “Playwright test “ and hit

Observed that all requirement start downloading Automatically (it’s take some Time in my case it’s take 18 min bcz i am using my ruler internet)
You can see the script automatically run in terminal:

```
PS C:\Users\UC\Desktop\Playwright with extention> npm init playwright@latest --yes -- --quiet
--browser=chromium --browser=firefox --browser=webkit --lang=js --gha
Getting started with writing end-to-end tests with Playwright:
Initializing project in '.'
Initializing NPM project (npm init -y)…
Wrote to C:\Users\UC\Desktop\Playwright with extention\package.json:

{
  "name": "playwright-with-extention",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

```
Installing Playwright Test (npm install --save-dev @playwright/test)…

added 3 packages, and audited 4 packages in 10s

found 0 vulnerabilities
Installing Types (npm install --save-dev @types/node)…

added 2 packages, and audited 6 packages in 12s

found 0 vulnerabilities
Downloading browsers (npx playwright install chromium firefox webkit)…
Downloading Chromium 120.0.6099.28 (playwright build v1091) from
https://playwright.azureedge.net/builds/chromium/1091/chromium-win64.zip
122 Mb [====================] 100% 0.0s
Chromium 120.0.6099.28 (playwright build v1091) downloaded to
C:\Users\UC\AppData\Local\ms-playwright\chromium-1091
Downloading FFMPEG playwright build v1009 from
https://playwright.azureedge.net/builds/ffmpeg/1009/ffmpeg-win64.zip
1.4 Mb [====================] 100% 0.0s
FFMPEG playwright build v1009 downloaded to
C:\Users\UC\AppData\Local\ms-playwright\ffmpeg-1009
Downloading Firefox 119.0 (playwright build v1429) from
https://playwright.azureedge.net/builds/firefox/1429/firefox-win64.zip
80.5 Mb [====================] 100% 0.0s
Firefox 119.0 (playwright build v1429) downloaded to
C:\Users\UC\AppData\Local\ms-playwright\firefox-1429
Downloading Webkit 17.4 (playwright build v1944) from
https://playwright.azureedge.net/builds/webkit/1944/webkit-win64.zip
46.4 Mb [====================] 100% 0.0s
Webkit 17.4 (playwright build v1944) downloaded to
C:\Users\UC\AppData\Local\ms-playwright\webkit-1944
```

```
Writing playwright.config.js.
Writing .github\workflows\playwright.yml.
Writing tests\example.spec.js.
Writing tests-examples\demo-todo-app.spec.js.
Writing package.json.
✔ Success! Created a Playwright Test project at C:\Users\UC\Desktop\Playwright with extention

Inside that directory, you can run several commands:

  npx playwright test
    Runs the end-to-end tests.

  npx playwright test --ui
    Starts the interactive UI mode.

  npx playwright test --project=chromium
    Runs the tests only on Desktop Chrome.

  npx playwright test example
    Runs the tests in a specific file.

  npx playwright test --debug
    Runs the tests in debug mode.

  npx playwright codegen
    Auto generate tests with Codegen.

We suggest that you begin by typing:

    npx playwright test

And check out the following files:
  - .\tests\example.spec.js - Example end-to-end test
  - .\tests-examples\demo-todo-app.spec.js - Demo Todo App end-to-end tests
  - .\playwright.config.js - Playwright Test configuration

Visit https://playwright.dev/docs/intro for more information. ✨

Happy hacking! 🎭
PS C:\Users\UC\Desktop\Playwright with extention>
```

Congratulations now you installed all packages (required and non-required both) and you can observe too many files added in your project Repo include test.js files.
Now you can see one default script created with name test.js

Step 7. Open the test.js file
observed some default tc's script already written for hint purpose you can remove it or you can modify it as per your requirement.
Step 8. Goto same terminal and run below script of tc's

1. *npx playwright test*
   Runs for the end-to-end tests.
2. *npx playwright test --ui*
   Starts the interactive UI mode.
3. *npx playwright test --project=chromium*
   Runs the tests only on Desktop Chrome.
4. *npx playwright test example*
   Runs the tests in a specific file.
5. *npx playwright test --debug*
   Runs the tests in debug mode.
6. *npx playwright codegen*
   Auto generate tests with Codegen.
7. *npx playwright test --project=chromium --headed*
8. *npx playwright test --project=chromium --headed --debug*
9. *npx playwright --version*

There are many commands you can use https://playwright.dev/docs/intro .

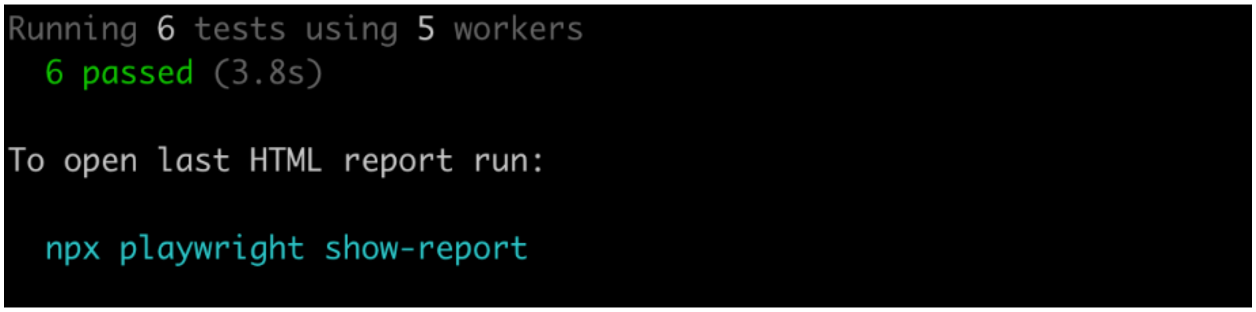
Method 3: Installing Playwright with Cmd:

Goto project dir and open it in VS code then open VS code terminal
Step 1: *npm init playwright@latest*
Run the install command and select the following to get started:

- Choose between TypeScript or JavaScript (default is TypeScript)
- Playwright will download the browsers needed as well as create the following files.

```
playwright.config.ts
package.json
package-lock.json
tests/
 example.spec.ts
tests-examples/
 demo-todo-app.spec.ts
```

Step 2: *npx playwright test*

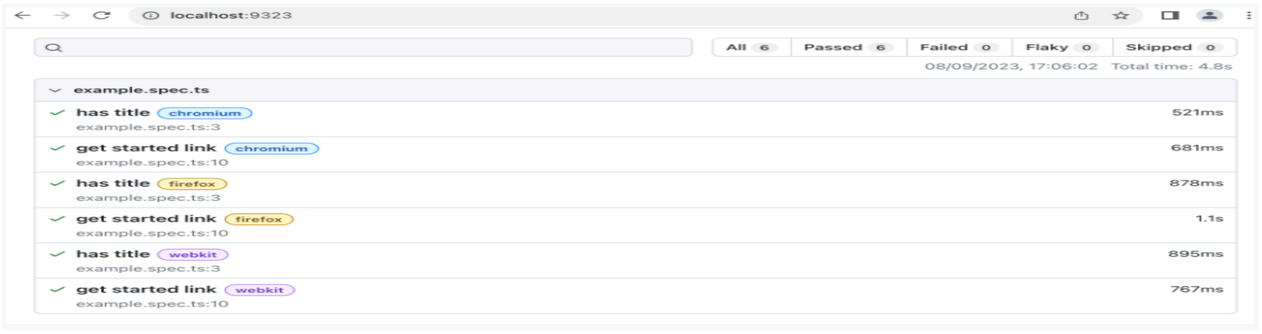


```
Running 6 tests using 5 workers
  6 passed (3.8s)

To open last HTML report run:

  npx playwright show-report
```

Step 3: *npx playwright show-report*



```
localhost:9323

All 6    Passed 6    Failed 0    Flaky 0    Skipped 0
                    08/09/2023, 17:06:02   Total time: 4.8s

example.spec.ts
✓ has title  chromium                                   521ms
  example.spec.ts:3
✓ get started link  chromium                            681ms
  example.spec.ts:10
✓ has title  firefox                                    878ms
  example.spec.ts:3
✓ get started link  firefox                             1.1s
  example.spec.ts:10
✓ has title  webkit                                     895ms
  example.spec.ts:3
✓ get started link  webkit                              767ms
  example.spec.ts:10
```

System requirements

- Node.js 16+
- Windows 10+, Windows Server 2016+ or Windows Subsystem for Linux (WSL).
- MacOS 12 Monterey or MacOS 13 Ventura.
- Debian 11, Debian 12, Ubuntu 20.04 or Ubuntu 22.04, with x86-64 or arm64 architecture.

Installation Command:

```
npm install playwright
```

1. **Install Browsers**: Playwright automatically downloads necessary browser binaries.

What's Installed

1. **Playwright CLI**: A command-line interface for running and debugging tests.
2. **Browser Binaries**: Chromium, Firefox, and WebKit binaries.
3. **Playwright Library**: APIs for browser automation.
4. **Example Test**: A sample test for reference.

Running the Example Test

Create a test file (e.g., `example.spec.js`):

```
const { test, expect } = require('@playwright/test');

test('basic test', async ({ page }) => {
    await page.goto('https://example.com');
    const title = await page.title();
    expect(title).toBe('Example Domain');
});
```

1.

Run the test:
```
npx playwright test
```

2.

HTML Test Reports

Playwright can generate detailed HTML reports to view test results:

Enable reporting:
```
npx playwright test --reporter=html
```
Open the report:
```
npx playwright show-report
```

1.

Running the Example Test in UI Mode

Start UI mode:
```
npx playwright test --ui
```

1. Interact with tests using a graphical interface.

Updating Playwright

To update Playwright to the latest version:

Update the package:
```
npm install playwright@latest
```
Check the installed version:
```
npx playwright --version
```

System Requirements

1. **Supported Platforms**: Windows, macOS, Linux.
2. **Node.js**: Version 16 or later.
3. **Browsers**: Automatically managed by Playwright, including Chromium, Firefox, and WebKit.

Writing Tests with Playwright

Playwright simplifies writing reliable and robust tests for web applications. Here's how you can structure and improve your tests effectively.

Introduction

Tests in Playwright are written using the Playwright Test framework. This framework includes built-in fixtures, powerful assertion libraries, and extensive support for test isolation, hooks, and reporting.

How to Write the First Test

Create a file, e.g., `first-test.spec.js`:

```
const { test, expect } = require('@playwright/test');


test('navigate to a page and check the title', async ({ page }) => {

    await page.goto('https://example.com');

    const title = await page.title();

    expect(title).toBe('Example Domain');

});
```

Run the test:

```
npx playwright test
```

How to Perform Actions

Playwright provides a robust API for performing user-like actions:

Clicking Elements:
```
await page.click('text="Get Started"');
```

Typing Text:
```
await page.fill('#username', 'testuser');
```

Navigating:
```
await page.goto('https://example.com');
```

How to Use Assertions

Assertions verify test outcomes and include various checks like text, attributes, or visibility.

Text Content:
```
expect(await page.textContent('h1')).toBe('Example Domain');
```

Element State:
```
expect(await page.isVisible('#submit')).toBe(true);
```

How Tests Run in Isolation

Each test starts with a fresh browser context, ensuring test independence:

```
test('test1', async ({ page }) => { /* Independent test */ });

test('test2', async ({ page }) => { /* Another independent test */ });
```

How to Use Test Hooks

Hooks manage setup and teardown logic:

Before Each Test:
```
test.beforeEach(async ({ page }) => {

    await page.goto('https://example.com');

});
```

After All Tests:
```
test.afterAll(async () => {

    console.log('All tests completed');

});
```

Actions and Navigation

Playwright supports user interaction and navigation:

- **Basic Actions**: Clicks, drags, and hover effects.
- **Page Navigation**: Navigate to URLs or simulate back/forward actions.

Interactions: Basic Actions

Checkboxes:
```
await page.check('#acceptTerms');
```

Drop-downs:
```
await page.selectOption('#dropdown', 'value');
```

Assertions

Playwright offers a rich set of assertions:

Assertion Description Lists

Assertion Type	Example	Description
Text Assertion	`expect(title).toBe('My Page');`	Matches the exact text.

State Check	`expect(page.isVisible('#login')).toBe(true);`	Checks element visibility.
URL Assertion	`expect(page.url()).toContain('/dashboard');`	Verifies URL content.

Test Isolation

- Each test runs in its own browser context.
- Shared state between tests is avoided by design.

Using Test Hooks

Hooks streamline pre- and post-test operations:

- **Global Setup**: Configure browsers or environments.
- **Per-Test Hooks**: Customize setup for individual tests.

Generating Tests with Playwright

Playwright simplifies test creation by enabling recording of user interactions and auto-generating locators, ensuring efficient test development.

Introduction

Playwright's built-in tools let you quickly record user interactions as test scripts. These recorded tests can be customized and enhanced for comprehensive test automation.

How to Record a Test

Playwright's test generator records actions performed in a browser and converts them into test scripts.

Start the Recorder:
```
npx playwright codegen
```

1. This launches a browser and a code generation UI.
2. **Perform Actions**:
   - Interact with the browser (e.g., navigate, click, type).
   - Actions are recorded in real time and displayed in the Playwright code editor.
3. **Save the Script**:
   - Once the interactions are complete, save the generated script.

Example Script:
```
const { test, expect } = require('@playwright/test');



test('recorded test', async ({ page }) => {

    await page.goto('https://example.com');

    await page.click('text=More information');

    expect(page.url()).toContain('https://example.com/info');

});
```

How to Generate Locators

Playwright generates precise and robust locators for web elements during test recording.

Start Locators Mode: Use the codegen tool:
```
npx playwright codegen
```

1. Hover over elements to view their locators.
2. **Manually Create Locators**:

CSS Selectors:
```
const locator = page.locator('.button-class');
```

Text-Based:
```
const locator = page.locator('text="Submit"');
```

XPath:
```
const locator = page.locator('//button[@id="submit"]');
```

3. **Best Practices for Locators**:
    - Prefer `role`, `text`, or `aria-label` attributes for accessibility and stability.
    - Avoid overly complex or brittle locators tied to dynamic attributes like IDs.

Generating Locators

Playwright ensures reliable locators for elements:

- **Using `codegen`**: Generate locators as part of a recorded script.

Locator Debugging:

```
npx playwright show-trace
```

- Visualize and fine-tune locators during test execution.

Running and Debugging Tests with Playwright

Playwright provides robust tools for executing and debugging tests efficiently, including CLI commands, debugging modes, and HTML reports for test results.

How to Run Tests from the Command Line

Run All Tests: Execute all tests in the `tests/` directory:

```
npx playwright test
```

Run a Specific Test File: Specify the file path:

```
npx playwright test example.spec.js
```

Filter Tests by Title: Use `-g` with a test title or pattern:

```
npx playwright test -g "basic test"
```

Run in a Specific Browser: Run tests in Chromium, Firefox, or WebKit:

```
npx playwright test --project=chromium
```

1. **Run in Parallel or Sequentially**:
    - Parallel: Enabled by default.

Sequential: Use `--workers=1`:

```
npx playwright test --workers=1
```

How to Debug Tests

Debugging with `--debug`: Pause the test runner at the first failure and open the inspector:

```
npx playwright test --debug
```

Run Tests in UI Mode: Interactively debug tests in a visual interface:

```
npx playwright test --ui
```

Debugging with Breakpoints: Add `debugger` statements in your code:

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

    await page.goto('https://example.com');

    debugger; // Pause here during debugging

    await page.click('text="Learn More"');

});
```

1. **Using Trace Viewer**: Enable tracing to analyze failures:

Record traces:

```
npx playwright test --trace=on
```

Open the trace:

```
npx playwright show-trace trace.zip
```

Step-by-Step Execution: Use the codegen tool for live debugging:

```
npx playwright codegen
```

How to Open the HTML Test Reporter

Generate the Report: After running tests with the HTML reporter enabled:

```
npx playwright test --reporter=html
```

Open the Report: Launch the HTML test report in your browser:

```
npx playwright show-report
```

1. **Navigate the Report**:
    - View test status (passed, failed, skipped).
    - Inspect error messages, logs, and stack traces for failed tests.

Playwright Trace Viewer

The **Playwright Trace Viewer** is a powerful debugging tool that provides detailed insights into test execution. It allows you to replay and analyze tests, inspect page interactions, and identify issues efficiently.

Introduction

- A **trace** is a record of test execution that includes:
    - Screenshots
    - DOM snapshots
    - Network requests
    - Console logs
- The Trace Viewer lets you visually inspect the trace to debug test failures or performance issues.

How to Record a Trace

Enable Tracing in the Test Runner: Use the `--trace` option to record traces during test execution:

```
npx playwright test --trace=on
```

Add Tracing Programmatically: Use Playwright's `trace` fixture in the test configuration:
javascript

```
const { test } = require('@playwright/test');



test('example with trace', async ({ page, context }) => {

    await context.tracing.start({ screenshots: true, snapshots: true
});
```

```
    await page.goto('https://example.com');

    await context.tracing.stop({ path: 'trace.zip' });

});
```

How to Open the HTML Report

Generate the HTML Report: Run tests with the `--reporter=html` option:

```
npx playwright test --reporter=html
```

Open the Report: View the report in a browser:

```
npx playwright show-report
```

1. **Inspect Traces**: For failed tests, the HTML report includes links to associated traces, allowing you to analyze them directly.

How to Open and View the Trace

Launch the Trace Viewer: Use the `show-trace` command to open a saved trace file:

```
npx playwright show-trace trace.zip
```

1. **Analyze the Trace**:
   - **Timeline**: Step through test actions in chronological order.
   - **Screenshots**: View screenshots taken during each interaction.
   - **Network**: Inspect network requests and responses.
   - **Console Logs**: Check for console errors or warnings.
   - **DOM Snapshots**: Examine the state of the DOM at each step.
2. **Debugging with the Trace Viewer**:
   - Recreate user interactions.
   - Identify issues like failed clicks, missing elements, or incorrect assertions.

Setting Up Continuous Integration (CI) for Playwright

Using CI tools like **GitHub Actions**, you can automate the execution of Playwright tests on every code change. This ensures consistent test coverage and faster feedback.

How to Set Up GitHub Actions

1. **Create a GitHub Workflow File**:
    - Add a `.yml` file in the `.github/workflows` directory, e.g., `playwright.yml`.

Example Workflow for Playwright:

```
name: Playwright Tests

on:

  push:

    branches:

      - main

  pull_request:



jobs:

  test:

    runs-on: ubuntu-latest

    steps:

      - name: Checkout code

        uses: actions/checkout@v3



      - name: Set up Node.js

        uses: actions/setup-node@v3
        with:

          node-version: '16'


      - name: Install dependencies

        run: npm install


      - name: Run Playwright tests

        run: npx playwright test
```

2. **Add Browser Dependencies**: Playwright automatically installs the required browsers during the test run.

How to View Test Logs

1. **Access Logs on GitHub Actions**:
    - Navigate to the **Actions** tab in your GitHub repository.
    - Select the workflow run to view logs for each step.
2. **Inspect Test Results**:
    - Check the output of the `npx playwright test` step to see test results, errors, or failures.

How to View the HTML Report

Generate the HTML Report: Modify the workflow to save the HTML report:

```
- name: Generate HTML report

  run: npx playwright test --reporter=html
```

Upload the Report: Use GitHub Actions' artifact functionality:

```
- name: Upload HTML report

  uses: actions/upload-artifact@v3

  with:
    name: playwright-report

    path: playwright-report
```

1. **Download the Report**:
   - Navigate to the workflow run.
   - Download the **playwright-report** artifact and open it locally.

How to View the Trace

Enable Tracing in CI: Record traces during test runs:
yaml

```
- name: Run Playwright tests with trace

  run: npx playwright test --trace=on
```

Upload Traces as Artifacts: Add a step to upload the trace:

```
- name: Upload trace

  uses: actions/upload-artifact@v3

  with:

    name: trace

    path: trace.zip
```

1.
2. **Download and Analyze**:
   - Download the trace from the workflow artifacts.

View it using the Trace Viewer:

```
npx playwright show-trace trace.zip
```

   -

How to Publish the Report on the Web

1. **Serve the Report via GitHub Pages**:

  - Generate the HTML report as part of the workflow.

Push the `playwright-report` folder to the `gh-pages` branch:

```
- name: Deploy to GitHub Pages

  uses: peaceiris/actions-gh-pages@v3

  with:

    github_token: ${{ secrets.GITHUB_TOKEN }}

    publish_dir: playwright-report
```

  -

2. **Access the Report**:
   - Visit the GitHub Pages URL (e.g., `https://<username>.github.io/<repository>`).
3. **Alternative: Use a Static File Host**:
   - Upload the report to platforms like Netlify or Vercel.

Setting Up Continuous Integration (CI) for Playwright on GitLab

GitLab CI/CD is a robust tool for automating Playwright tests. By configuring a `.gitlab-ci.yml` file, you can integrate Playwright into your GitLab pipelines.

How to Set Up GitLab CI for Playwright

1. **Create a `.gitlab-ci.yml` File**: Add this file to the root of your GitLab repository.

Example GitLab CI Configuration:

```
stages:

  - test



test_playwright:

  stage: test

  image: mcr.microsoft.com/playwright:v1.38.0-focal
  script:

    - npm install

    - npx playwright test

  artifacts:

    when: always

    paths:

      - playwright-report

      - trace.zip

    expire_in: 1 week
```

2. **Explanation**:
   - **Stages**: Define the pipeline stages, such as `test`.
   - **Image**: Use the official Playwright Docker image for easy setup.
   - **Script**: Includes commands to install dependencies and run tests.
   - **Artifacts**: Saves test reports and traces for download after the pipeline completes.

How to View Test Logs

1. **Access Logs in GitLab**:
   - Go to **CI/CD > Pipelines** in your GitLab project.
   - Select a pipeline run and view the logs for each job.
2. **Inspect Test Results**:
   - Check the `npx playwright test` step for test results, including failures.

How to View the HTML Report

Enable the HTML Reporter: Modify the script to generate an HTML report:

```
script:

  - npm install
  - npx playwright test --reporter=html
```

Save the Report as an Artifact: Add the `playwright-report` folder to artifacts:

```
artifacts:

  paths:

    - playwright-report
```

1. **Download and Open the Report**:
   - Navigate to the pipeline job.
   - Download the `playwright-report` artifact and open it locally in a browser.

How to View the Trace

Enable Tracing in Tests: Record traces during test execution:

```
script:

  - npx playwright test --trace=on
```

Save Traces as Artifacts: Add `trace.zip` to the artifacts section:

```
artifacts:

  paths:

    - trace.zip
```

1. **Download and Analyze the Trace**:
   - Download the `trace.zip` artifact from the pipeline.

Open the trace using the Playwright Trace Viewer:

```
npx playwright show-trace trace.zip
```

-

How to Publish the Report on the Web

Use GitLab Pages: Add a job to publish the `playwright-report` directory to GitLab Pages:

```
pages:
```

```
stage: deploy

script:

  - mv playwright-report public

artifacts:

  paths:

    - public
```

1. **Access the Report**:
   - Navigate to your GitLab Pages URL (e.g., `https://<username>.gitlab.io/<repository>`).
2. **Alternative Hosting**:
   - Use external static hosting platforms like Netlify if GitLab Pages isn't suitable.