Playwright

Link - https://playwright.dev/

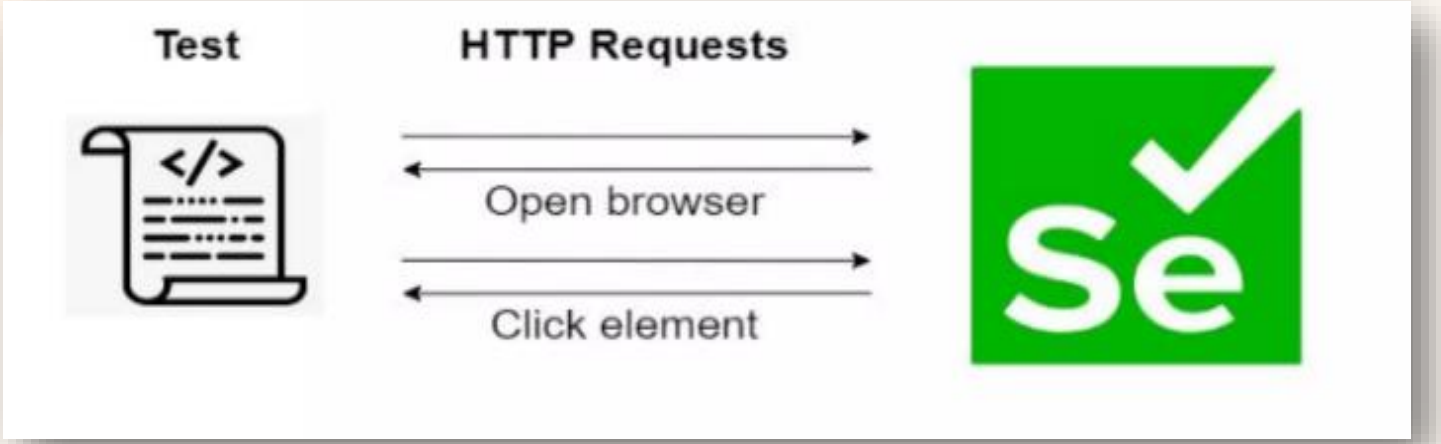
Modules

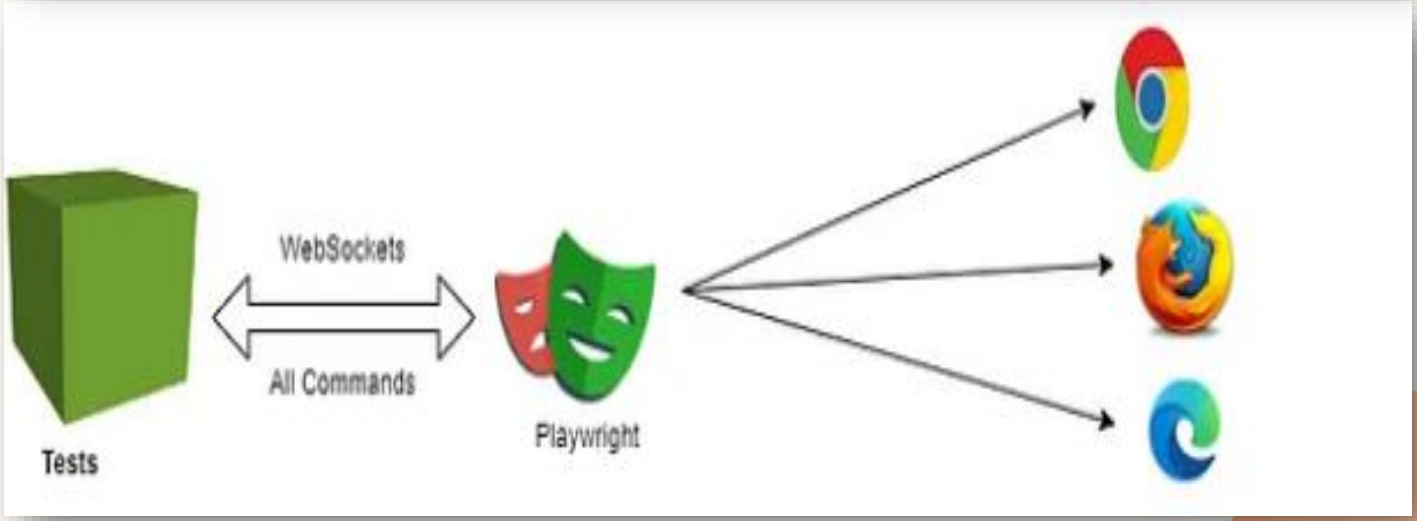
1. **Introduction to PlayWright**
2. **Installation of PlayWright**
3. **First Test in PlayWright**
4. **Locators**
5. **WebElement Interaction / User Actions**
6. **Assertions**
7. **Codegen - Record Script in PlayWright**
8. **Screenshots** and **videos** in PlayWright
9. **Reports** in PlayWright

Introduction To Playwright

- Playwright is a E2E open source testing and automation framework for modern web apps that allows you to do testing in any browser, device or platform using the language of your choice.
- Playwright is an open source test automation library initially developed by Microsoft contributors.
- It allows testing Chromium, Firefox and WebKit with a single API.
- Playwright supports programming languages such as Java, Python, C# and Node JS with JavaScript.
- Playwright It is built to enable cross-browser web automation that is reliable, fast and capable.

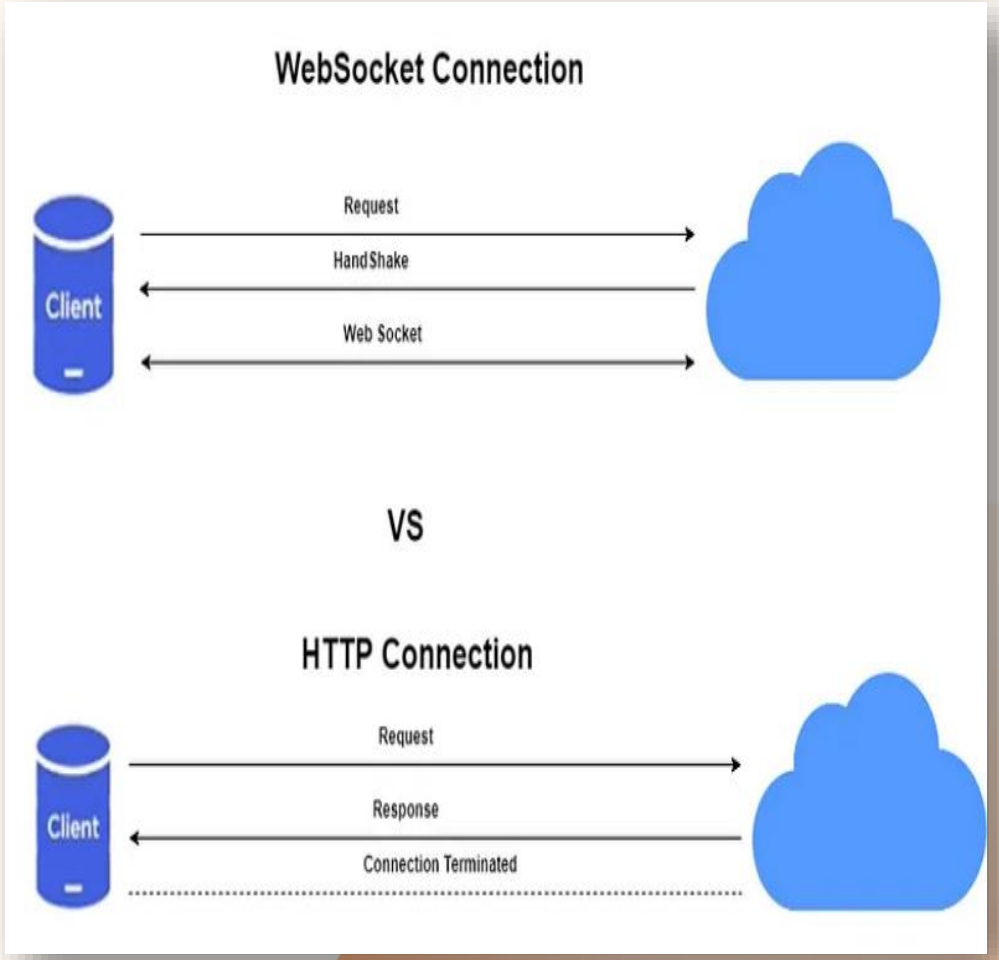
Playwright Architecture





Playwright Architecture

- Selenium sends each command as a separate HTTP request and receives JSON responses.
- So, every action, such as opening the browser, clicking an element, or sending keys in a text box, is sent as a separate HTTP request.
- Additionally, after completion of every request, the connection between server and client will be terminated, which needed to be re-established for the next request.
- Connection termination after every request result in slower execution which introduces a layer of flakiness.
- Playwright, on the other hand, communicates all requests through a single Web socket connection, which stays in place until test execution is completed.
- This reduces the points of failure and allows commands to be sent quickly on a single connection.



Playwright Vs Selenium

StarAgile

Considerations	Playwright	Selenium
Operating Systems	Windows, Linux, Mac OS	Windows, Mac OS, Linux, Solaris
Browser Support	Chromium, WebKit, and Firefox	Chrome, Edge, Firefox, Safari
Browser Drivers	Built-in drivers	Separate web drivers
Language Support	Typescript, Python, JavaScript, Java, .NET	Python, Java, JavaScript, Ruby, C#
Headless Mode	for supported browsers	for Chrome and Firefox
Speed and Performance	Faster than Selenium	Comparatively slower than the Playwright
Community Support	Growing community	Larger community

Playwright Vs Selenium Which to Choose

- Both Playwright and Selenium have advantages and limitations, which means choosing between them is subjective to the scenario they will be used for.
- Playwright offers fast testing in complex web applications with headless architecture and requires NodeJS as a prerequisite, it is relatively new.
- It lacks support on various levels, such as community, browsers, real devices, language options, and integrations.
- Selenium has all of this to offer
- Each supports CI/CD for a software project with due accuracy.
- Playwright has the upper hand in complex web applications, but has limited coverage.
- On the contrary, Selenium offers comprehensive coverage, scalability, flexibility, and strong community support.
- It depends on the project requirements and the priorities to choose one among these two testing frameworks.
- When the architecture is very complex with limited coverage, then Playwright is the option to select for testing.
- In a scenario that requires wider coverage, Selenium is the best.

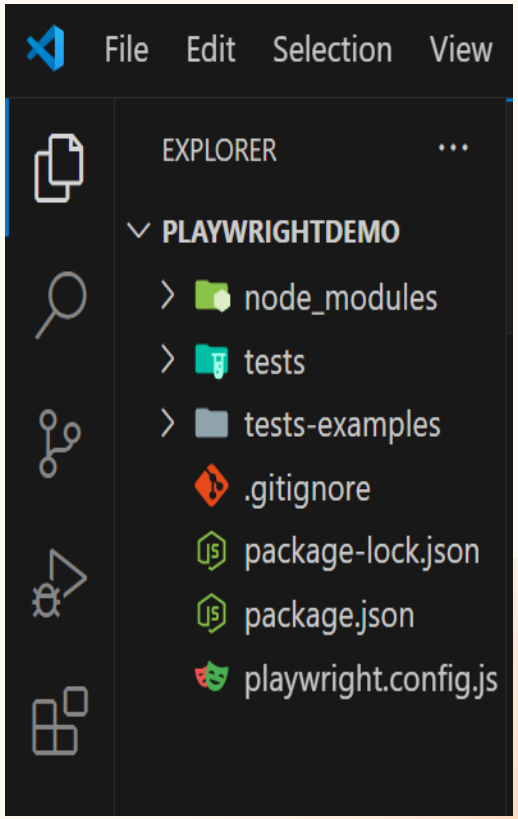
Key Features of the playwright Framework

- Support for cross-browser platforms built on Chromium, WebKit, and Firefox - which includes Chrome, Edge, Firefox, Opera, and Safari.
- Auto-wait built-in, smart assertions that retry until the element is found, and test data tracing - keep track of logs, videos and screenshots easily.
- Support for cross-language, including JavaScript, TypeScript, Python, Java, and .NET- write tests in the environment that suits you while still covering all areas and formats.
- Support for cross-platform execution on Windows, Linux, and macOS.
- Built with modern architecture and no restrictions - interact with multi-page, multi-tab websites like a real user, and tackle frames and browser events with ease.

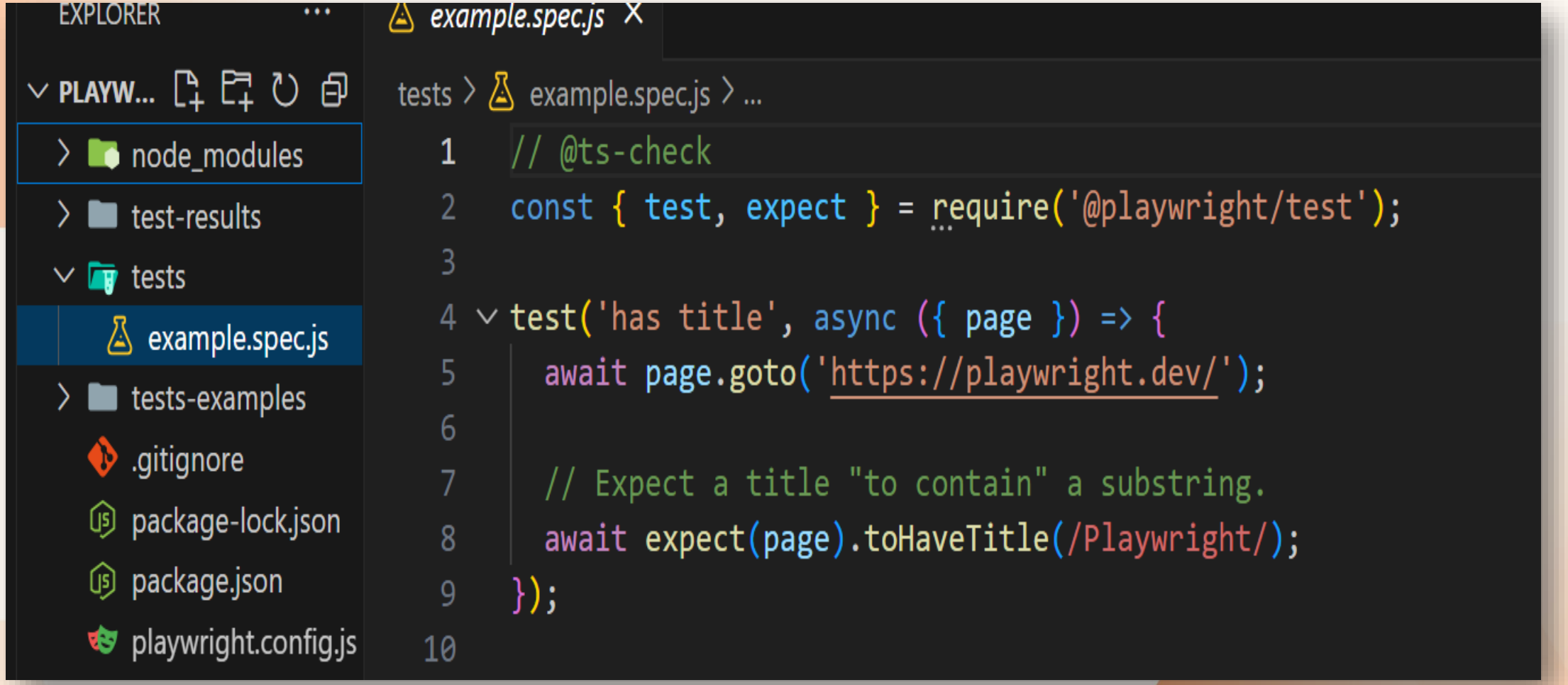
Playwright Installation and Its Project Structure

- Here are the prerequisites to install Playwright:
- NodeJS
- VS Code Editor
- **Steps to install the Playwright:**
- After the prerequisites, Create a folder with "Playwright_Demo" as folder name.
- Launch VS Code Editor and Open the "Playwright_Demo" Folder.
- Open your terminal and Run this command:

  npm init playwright@latest.
- Playwright will download the browsers needed as well as create the following files i.e. playwright.config.ts, package. json and package-lock.json.



First Test Case In Playwright



Test Execution In Playwright

- Run this command to run your tests:
- npx playwright test
- This command will run your tests in headless mode for all three browsers i.e. Chromium, Firefox and WebKit.ast read
- To see the report, One can run this command that will open report in your browser.
- npx playwright show-report

```
PS C:\Users\Ganes\OneDrive\Desktop\Learning\PlaywrightDemo> npx playwright test

Running 6 tests using 6 workers
  6 passed (23.4s)

To open last HTML report run:

  npx playwright show-report

PS C:\Users\Ganes\OneDrive\Desktop\Learning\PlaywrightDemo>
```

Playwright Locators

- **Link - Locators | Playwright**
- **https://github.com/DD-TestEningeer/web-automation-using-playwright/wiki**
- Locators are the central piece of Playwright's auto-waiting and retry-ability. In a nutshell, locators represent a way to find element(s) on the page at any moment.
- These are the **recommended** built-in locators.
- **page.getByRole()** :to locate by explicit and implicit accessibility attributes.
- **page.getByText()** :to locate by text content.
- **page.getByLabel():** to locate a form control by associated label's text.
- **page.getByPlaceholder()** :to locate an input by placeholder.
- **page.getByAltText():** to locate an element, usually image, by its text alternative.
- **page.getByTitle()** :to locate an element by its title attribute.
- **page.getByTestId()** :to locate an element based on its data-testid attribute (other attributes can be configured).

Record Script In Playwright

- Link - Test generator | Playwright
- The Playwright Code Generator is a tool that helps you create automated tests for web applications by recording your interactions in a browser.
- As you perform actions (like clicking buttons, filling forms, etc.), the code generator generates corresponding Playwright script code in real-time.
- This feature simplifies the process of writing tests by providing a visual way to create scripts.
- In Playwright, you can record scripts using the Playwright CLI tool or the Playwright code generator.
- Here's how to do it:
- Using the Playwright Code Generator
- Run the Code Generator: You can start the Playwright code generator by running:
- **npx playwright codegen <URL>**
- Replace <URL> with the website you want to test.
- For example:
- **npx playwright codegen https://example.com**

Viewport In Playwright

- **Link - [Emulation | Playwright]**
- In Playwright, a "viewport" refers to the visible area of a web page that is rendered in the browser.
- It's essentially the size of the browser window when it displays the webpage.
- Setting the viewport size allows you to simulate how a webpage will look on different devices and screen resolutions.
- You can configure the viewport size when launching a browser instance or when creating a page.

playwright.config.ts

```ts
import { defineConfig, devices } from '@playwright/test';

export default defineConfig({
  projects: [
    {
      name: 'chromium',
      use: {
        ...devices['Desktop Chrome'],
        // It is important to define the `viewport` property after destructuring `devices`,
        // since devices also define the `viewport` for that device.
        viewport: { width: 1280, height: 720 },
      },
    },
  ]
});
```

tests/example.spec.ts

```ts
import { test, expect } from '@playwright/test';

test.describe('specific viewport block', () => {
  test.use({ viewport: { width: 1600, height: 1200 } });

  test('my test', async ({ page }) => {
    // ...
  });
});
```

File Upload In Playwright

- **Link -** Actions | Playwright

```
import { test, expect } from "@playwright/test";

test("TC001 - Upload File Demo", async ({ page }) => {

  await page.goto("https://dd-demo-tau.vercel.app/web_elements.html");
  await page.getByRole("link", { name: "File Upload" }).click();
  await page.getByRole("button", { name: "Choose file" }).click();
  await page.getByRole("button", { name: "Choose file“ }).setInputFiles("testData.json");
  await expect(page.locator("#fileUploadMsg")).toContainText("File Selected: testData.json");
  await page.waitForTimeout(5000);
});
```

Dropdown In Playwright

- **Link -** Locator | Playwright
- **Handling dropdowns in Playwright can be done in several ways, depending on whether you're working with <select> elements or custom dropdown implementations.**
- **For standard <select> elements, use the selectOption method.**
- **For custom dropdowns, simulate clicks to open and select options.**
- **Selects one or multiple options in the <select> element with locator.selectOption().**
- **You can specify option value, or label to select. Multiple options can be selected.**
- **// Single selection matching the value or label**
- **await page.getByLabel('Choose a color').selectOption('blue');**
- **// Single selection matching the label**
- **await page.getByLabel('Choose a color').selectOption({ label: 'Blue' });**
- **// Multiple selected items**
- **await page.getByLabel('Choose multiple colors').selectOption(['red', 'green', 'blue']);**

Screenshot In Playwright

- **Link -** Screenshots | Playwright
- **Capturing screenshots in Playwright is straightforward and can be done using the screenshot() method.**
- **Here’s how you can take screenshots of a webpage or specific elements.**
- **Full page screenshot is a screenshot of a full scrollable page, as if you had a very tall screen and the page could fit it entirely.**
- **await page.screenshot({ path: 'screenshot.png', fullPage: true });**

- **Element screenshot**
- **Sometimes it is useful to take a screenshot of a single element.**
- **await page.locator('.header').screenshot({ path: 'screenshot.png' });**

Videos In Playwright

- **Link -** Videos | Playwright
- **Record video**
- **Playwright Test can record videos for your tests, controlled by the video option in your Playwright config. By default videos are off.**
- **'off' - Do not record video.**
- **'on' - Record video for each test.**
- **'retain-on-failure' - Record video for each test, but remove all videos from successful test runs.**
- **'on-first-retry' - Record video only when retrying a test for the first time.**
- **Video files will appear in the test output directory, typically test-results.**
- **Videos are saved upon browser context closure at the end of a test.**
- **If you create a browser context manually, make sure to await browserContext.close().**

Popups In Playwright

- **Link -** Pages | Playwright

- **A **popup** is a new browser window or tab that opens when triggered by actions like clicking a link or button. It can contain HTML elements that you can interact with, such as filling out forms or clicking buttons. You can control and automate interactions with popups using Playwright by listening to the `popup` event.**

•**Type**: Can be new browser windows or tabs (e.g., a new tab opens when you click a link).

•**Interaction**: You can interact with elements inside the popup, like clicking buttons, filling out forms, and more.

•**Trigger**: Often triggered by actions like clicking on a link (<a target="_blank">) or a button that opens a new window/tab.)

•**Control**: You have full control over the content and can automate interactions with popups.

Popups In Playwright

- **What is a Popup?**
  - New browser window/tab triggered by a link or button.
  - Can contain forms, buttons, or other HTML elements.
- **Trigger:** <a target="_blank"> or buttons opening new windows/tabs.
- **Interaction:** Fill forms, click buttons, read content inside popup.
- **Control:** Playwright can fully automate interactions using popup event

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

test('handle link-triggered popup', async ({ page }) => {
  // Navigate to the demo page
  await page.goto('https://dd-demo-tau.vercel.app/web_elements.html');
  // Click on the "Links" section link
  await page.getByRole('link', { name: 'Links' }).click();
  // Prepare to wait for the popup (new tab/window)
  const page1Promise = page.waitForEvent('popup');
  // Click the external link that opens a new window
  await page.getByRole('link', { name: 'External Link' }).click();
  // Wait for the popup page to load
  const page1 = await page1Promise;
  // Verify the popup heading is visible
  await expect(page1.getByRole('heading', { name: 'Example Domain' })).toBeVisible();
  // Verify the popup heading contains the expected text
  await expect(page1.getByRole('heading')).toContainText('Example Domain');
});
```

Alerts In Playwright

- **Link -** Dialogs | Playwright

Reports In Playwright

- **Link - Running and debugging tests | Playwright**
- **Playwright Test comes with a few built-in reporters for different needs and ability to provide custom reporters.**
- **The easiest way to try out built-in reporters is to pass --reporter command line option.**
- **npx playwright test --reporter=line**

- **Built-in reporters**
  - **List reporter**
  - **Line reporter**
  - **Dot reporter**
  - **HTML reporter - npx playwright test <specific test file > --reporter=html**
  - **JSON reporter**

- **Third Party - Allure Report**
  - allure-js/packages/allure-playwright at main · allure-framework/allure-js · GitHub
  - **https://github.com/DD-TestEningeer/web-automation-using-playwright/wiki/Allure-%E2%80%90-Reports**

Framework Enhancement

- Pre-req
- Installation
- Project Setup
- Config
- Screenshots, video, trace, report config
- Adding Test in single file
- Adding Test using POM pattern
- Running Test in different Ways using terminal
- Running Test on the GitHub Actions