CYDEO

Playwright Automation Day01

Content

- Introduction to Playwright
- Environment Setup
- @playwright/test package
- Fixtures
- Working with browsers
- Interacting with Web-Elements

What is Playwright?

- An automation tool for automating web browsers
- Developed by Microsoft
- Provides support for multiple programming languages
- Supports chromium, firefox, and webKit browsers

Playwright

Why Use Playwright?

- Reliable and fast browser automation
- Cross-browser testing
- Auto-waiting functionality
- Easier setup and maintenance

Setting up Playwright project

- **Steps:**
  1. Create a new folder for VS Code project
  2. Open the folder in VS Code
  3. Open VS Code terminal
  4. Give the npm init playwright@latest command
  5. When asked Okay to proceed? type y
  6. Select JavaScript for the language
  7. Select the default folder for end-to-end tests by simply pressing the Enter key
  8. Select false for adding GitHub Actions workflow by simply pressing the Enter key
  9. Select true for installing playwright browsers by simply pressing the Enter key
  10. Wait for the installations to be completed
  11. Open the terminal and give the npx playwright test command (By default it runs the tests in headless mode
  12. Open the terminal and and give the npx playwright test --headed command

Recommended Naming Conventions

- For folder names: lowercase
- For test files: lowerCamelCase or Kebab-case
  - lowerCamelCase: loginTest.spec.js, loginTest.test.js
  - Kebab-case: login-test.spec.js, login-test.test.js
- For page object files: UpperCamelCase
  - UpperCamelCase: LoginPage.js

@playwright/test package

@playwright/test package

- Designed specifically for end-to-end testing
- Uses a specialized test runner and framework
- Comes with its own built-in reporters
- Introduces the concept of fixtures
- Has its own global configurations
- It includes built-in assertions

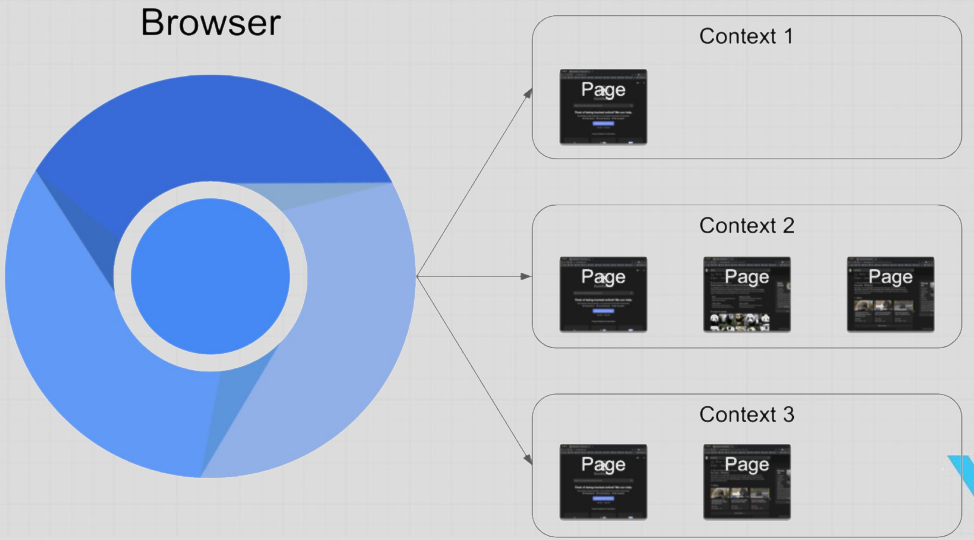
End-to-End Testing

- Testing the entire flow of an application from start to finish
- Ensures all integrated parts work together as expected
- The end-to-end testing in Playwright mainly focuses on UI
- Verifies the application's user interface and workflows from the user's perspective
- Helps identify and resolve problems before they affect users

E2E Testing

Test Runner

- Playwright uses fixtures to provide reusable, isolated test environments
- Simplifies the test setup and teardown process, making the tests more reliable and maintainable
- Common fixtures includes:
  - page
  - browser
  - context
  - viewport
  - browserName
  - request
  - baseURL



Test Function Declaration

- The test() function is the core building block of Playwright tests
- It takes two arguments: a string description and an async callback function
  - description: Describes what the test is checking
  - async callback function: Takes fixture(s) and contains the codes of the test
- Each test function runs in isolation, Playwright creates a new browser context for each test

```
import {test} from '@playwright/test';

test('test name', async ({ fixtureName }) => {

  // Test Codes

});
```

Test Groups

- The test.describe() is used for creating test groups.
- Test groups allow us to organize related tests together
- It improves readability and maintainability of our test suite
- Test Groups can be nested for further organization
- Test groups can have their own **hooks** that apply to all tests within the group

```
test.describe('Group name', () => {

  test('Test 1', async ({fixtureName}) => {
    // Test1 codes
  });

  test('Test 2', async ({fixtureName}) => {
    // Test2 codes
  });

});
```

Hooks

- Allow us to setup and teardown test environments for specific groups of tests.
- Helps with organizing and managing test setup and cleanup more efficiently.
- Reduces code duplication and improves readability
- Enables efficient management of resources for related tests
- Types of hooks in test groups are:
  - beforeEach(): Runs before each test in the test group
  - afterEach(): Runs after each test in the test group
  - beforeAll(): Runs once before all tests in the test group
  - afterAll(): Runs once after all tests in the test group

```
test.describe('User Authentication', () => {

  test.beforeAll(async ({ browser }) => {
    // code that runs one time before all tests
  });

  test.afterAll(async () => {
    // code that runs one time after all tests
  });

  test.beforeEach(async ({ page }) => {
    // code that runs before each test
  });

  test.afterEach(async ({ page }) => {
    // code that runs after each test
  });

  test('Test 1', async ({ page }) => {
    // Test2 codes
  });

  test('Test 2', async ({ page }) => {
    // Test1 codes
  });

});
```

Flow of Test Hooks and Structures

Execution order:

1. test.beforeAll()
2. For each test:
    - 2.1 test.beforeEach()
    - 2.2 Test Code
    - 2.3 test.afterEach()
3. test.afterAll()

```
test.describe('Group', () => {
    ├── test.beforeAll(() => { ... });
    ├── test('Test 1', async ({ page }) => { ... });
    │   ├── test.beforeEach(() => { ... });
    │   ├── Actual test code
    │   └── test.afterEach(() => { ... });
    ├── test('Test 2', async ({ page }) => { ... });
    │   ├── test.beforeEach(() => { ... });
    │   ├── Actual test code
    │   └── test.afterEach(() => { ... });
    └── test.afterAll(() => { ... });
});
```

The page Fixture

- A powerful tool for web automation and testing.
- An isolated page instance for each test
- Provides a clean, consistent starting point for each test
- Automatically created and destroyed
- Provides methods for navigation, interaction, and assertions

Common Methods of page Object

Method Name	Description
goto(url)	Navigate the browser to a specified URL
title()	Returns the title of the current page as a string
url()	Returns the current URL of the page as a string
setViewportSize({w, l})	sets the size of the browser viewport to specified width and height values
setDefaultTimeout(milliseconds)	sets the default maximum time (in milliseconds) the test on the page can take before timing out
page.locator(selector)	Creates a locator for the given selector, which can be used to perform actions like click, type

Locators

- Powerful tools for element interaction and assertion.
- An object representing a way to find element(s) on the page
- Provides a robust and reliable way to interact with page elements
- Locator Selectors:
    - CSS selectors: page.locator('button.primary')
    - XPath: page.locator('//button[contains(text(), "Submit")]')
    - Text content: page.locator('text=Submit')
    - TestID: page.locator('data-testid=submit-button')

Common Methods of Locator Object

Methods - Actions	Methods - Retrieval	Methods - State
click(url)	textContent()	isVisible()
fill()	innerText()	isEnabled()
type()	inputValue()	isChecked()
type()	getAttribute()	isDisabled()
check()		
uncheck()		
selectOption()		