Playwright: Introduction

Playwright

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Have a Question?

Software University

sli.do

#QA-FrontEnd

Table of Contents

Web UI Testing Overview

Web UI Testing

- **Web UI testing** tests components which **users interact** with
- Aims to determine if APIs meet requirements for
  - **User Experience**
    - UI == first point of contact with app for users
  - **Functionality**
    - Ensures all visual components work as expected
  - **Compatibility**
    - Checks whether all devices display web app correctly
  - **Performance**
    - Tests how UI performs under different conditions

Introduction to Playwright

Playwright

- Playwright == an **open-source** automation **library**
  - Used for **testing web applications** across different browsers and platforms
- Provides a **unified API**
- Enables automating **interactions with browsers**
  - Supports modern rendering engines
- Supports **various programming languages**
  - JavaScript, TypeScript, Python and C#

Features

- **Multi-Browser Support**
  - Chromium
    - Google Chrome
    - Microsoft Edge
  - Firefox
  - Safari
- **Cross-Platform**
  - Runs on Windows, Linux and macOS
  - Suitable for **different environments**, supporting **CI/CD**

Features

- Supports **Headless & Headed Mode**
  - Headless mode (no UI)
  - Headed mode (normal browser window)
- **Auto-Wait**
  - **Automatically** waits for **elements** to be **ready before** performing checks on them
- **Network Interception**
- Rich **set of APIs**

Features

- **Snapshot Testing**
  - Ensures that UI doesn't undergo **unintended** changes
- **Built-in Test Runner**
  - **Own** test runner
    - Optimized for parallel text execution
    - Can handle complex test setups
- **Browser Context** and **Pages**
  - Allows running tests in **isolation** from one another

Use Cases

- **Automated Regression Testing**
  - Ensures new features don't break existing functionality
- **Cross-Browser Testing**
  - Checks if apps perform consistently across different browsers
- **Performance Testing**
  - Allows analyzing performance of web apps under various conditions
- **Accessibility Testing**
  - Ensures web apps meat compliance standards

Example

- Playwright test

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

test('Page has Playwright in title', async ({ page }) => {
  await page.goto('https://playwright.dev/');
  const title = await page.title();
  expect(title).toBe('Fast and reliable end-to-end testing for
modern web apps | Playwright');
  const getStarted = await page.textContent('text=Get Started');
  expect(getStarted).toBeTruthy();
});
```

Playwright Setup

Setting Up Playwright

- Setting up Playwright is **straightforward** and requires **minimal** configuration
- Typically installed by **npm**

```
npm install @playwright/test
```

  - Playwright provides drivers for popular browsers, eliminating the need for separate installations
- You can start writing tests **immediately** after installation
  - Place the tests in a folder, named tests in the project's directory

Setting Up Playwright

- Create a **test directory** in the project's directory
- Create a **JS file** to hold the **Playwright** code
- Using **CommonJS**, import the function module

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

// Define the test suite
test('Page has Playwright in title', async ({ page }) => {
  await page.goto('https://playwright.dev/');
  const title = await page.title();
  // Define the test cases
  expect(title).toBe('Fast and reliable end-to-end testing
for modern web apps | Playwright');
}
```

Setting Up Playwright

- **Run** the tests
  - Using **npm script**
    - Add the command in the **package.json** file

```
"scripts": {
   "test": "playwright test"
}
```

    - Run the command

```
npm playwright test
```

Setting Up Playwright

Software University

File Edit Selection View ···

15-Playwright-Introduction-Lab-Solution

EXPLORER

15-PLAYWRIGHT-INTRODUCTION-LAB...
- node_modules
- test-results
- tests
  - todo.test.js
- package-lock.json
- package.json

OUTLINE
TIMELINE
VS CODE PETS

todo.test.js

tests > todo.test.js > ...

```
1  const { test, expect } = require('@playwright/test');
2
3  test('homepage has Playwright in title and learn link', async ({ page }) =>
4    await page.goto('https://playwright.dev/');
5    const title = await page.title();
6    expect(title).toBe('Fast and reliable end-to-end testing for modern web ap
7    const getStarted = await page.textContent('text=Get Started');
8    expect(getStarted).toBeTruthy();
9  });
10
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

node
powershell

```
PS D:\SoftUni [illegible]
 npm test

> new-folder@1.0.0 test
> playwright test


Running 1 test using 1 worker

  ✓  1 tests\todo.test.js:3:1 › homepage has Playwright in title and learn link (3.
4s)
```

0 0 0 Ln 12, Col 1 Spaces: 2 UTF-8 CRLF {} JavaScript Go Live

Working with Playwright

Test Functions

- **Test functions** are created using the `test()` function
  - Takes two arguments
    - **Description of the test**
    - **Callback function**, containing **the test logic**

Test Description

Callback function

```
test('Home page title test', async ({page}) => {
    await page.goto('https://example.com');
    const title = await page.title();
    expect(title).toBe('Expected Page Title', 'Page
title should match the expected title');
});
```

Test logic

Working with Playwright

- Writing tests with Playwright involves **defining test scenarios** and **executing them** against the web application
- Playwright offers a rich set of **APIs**
  - **Interacting** with web elements
  - **Simulating** user actions
  - **Asserting** expected outcomes
- Test **scripts** can be organized into **suites**
  - Enables **modular** and **scalable test automation**

Test Suites

- **Group related tests** together for a better organization

```
test.describe('name', () => {
    // Test case #1
    // Test case #2
});

// Test case #3
// Test case #4
```

Test Cases

- **Define a single test,** including its description and logic

```
test(title, [details], body)
```

  - **`title`** (string)
    - Title of the test
  - **`details`**
    - Additional configuration or metadata
  - **body** (callback function)
    - Function that performs the test

Playwright Functions

Navigation

- Navigate to a specified URL

```
page.goto(url, options)
```

- Reload current page

```
page.reload(options)
```

- Navigate the session history
  - Similar to pressing the browser's back and forward buttons

```
page.goBack(options)
```

```
page.goForward(options)
```

Interactions

- Click an element, specified by a selector

```
page.click(selector, options)
```

- Fill input field with specified text

```
page.fill(selector, value, options)
```

- Simulate a key press event on a specific element

```
page.press(selector, key, options)
```

- Perform drag-and-drop actions between elements

```
page.dragAndDrop(sourceSelector, targetSelector, options)
```

Assertions

- Strict equality check

```
expect.toBe(expected)
```

- Check if a string contains a specific substring

```
expect.toContain(substring)
```

- Assert that an element or page contains the specified text

```
expect.toHaveText(text, options)
```

- Assert visibility status of elements

```
expect.toBeVisible()
```

```
expect.toBeHidden()
```

Element Handling

- Return the first element, matching the selector

```
page.$(selector)
```

- Check if a string contains a specific substring

```
page.$$(selector)
```

- Assert that an element or page contains the specified text

```
page.textContent(selector, options)
```

- Assert visibility status of elements

```
page.innerHTML(selector, options)
```

Waiting & Conditions

- Wait for an element to appear in the DOM

```
page.waitForSelector(selector, options)
```

- Wait for a specified period
  - Useful for JS timeouts

```
page.waitForTimeout(millisecond)
```

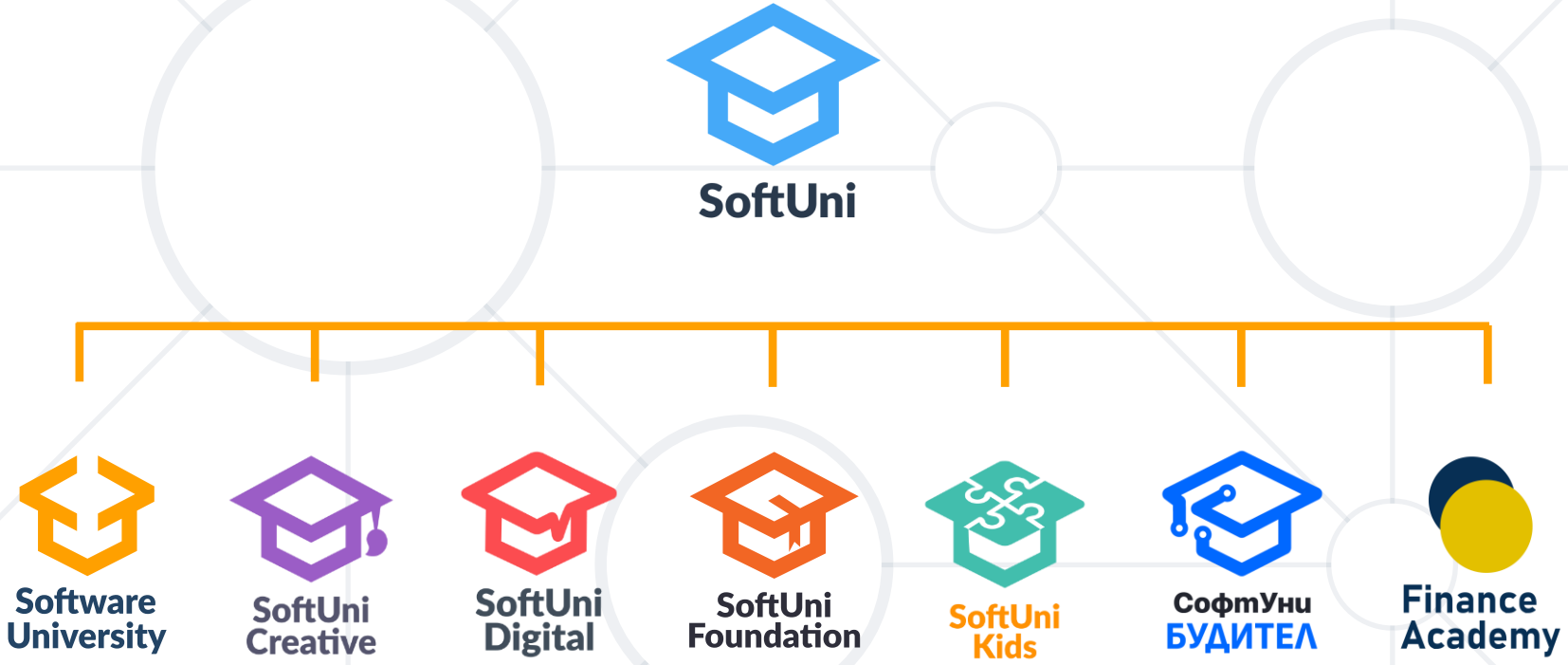
- Wait for the provided function to return a truthy value

```
page.waitForFunction(function[, arg, options])
```

Summary

- **UI testing** is essential for delivering high-quality web applications to end-users
- **Playwright** provides a robust framework for automating UI tests with JavaScript, offering **cross-browser** compatibility and powerful features
- Incorporating **Playwright** into your testing strategy can **enhance productivity**, accelerate release cycles, and ensure a seamless user experience

Questions?



Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

createX

Postbank

top EMPLOYER България Bulgaria 2024 FOR A BETTER WORLD OF WORK

DRAFT KINGS THE CROWN IS YOURS

VIVACOM

INDEAVR Serving the high achievers

AMBITIONED

PHAR VISION

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg
- © Software University – https://softuni.bg