

Playwright 101

For beginner to intermediate



Introduction to software testing

Introduction to software testing outline

- Software development life cycle
- Software testing life cycle
- Test pyramid
- Manual testing vs Automation testing
- Software tester

Planning

Defining

**Software
Development
Life
Cycle**

Designing

Developing

Testing

Deployment

Maintenance

Software Development Life Cycle

Planning

Requirement analysis is the most important and fundamental stage in SDLC. It is performed with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas. Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

Defining

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through an **SRS (Software Requirement Specification) document which consists of all the product requirements to be designed and developed during the project life cycle.**

Designing

Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - **Design Document Specification**. This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity, budget and time constraints, the best design approach is selected for the product. A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules

Developing

The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Maintenance

Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment.

After the product is released in the market, its maintenance is done for the existing customer base.

Deployment

Once the product is tested and ready to be deployed it is released formally in the appropriate market.

Sometimes product deployment happens in stages as per the business strategy of that organization.

The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

Testing

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC.

However, this stage refers to the testing only stage of the product **where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.**

Software Testing Life Cycle

**Test environment
Setup**

**Test case
Development**

Test planing

**Requirement
Analysis**

Test Execution

Test Closure

Planning

Maintenance

Deployment

Testing

Defining

Designing

Developing

Software Testing Life Cycle

Test case Development

Requirement Analysis

Test planning

Requirement Analysis is the first step of the Software Testing Life Cycle (STLC). **In this phase quality assurance team understands the requirements like what is to be tested.** If anything is missing or not understandable then the quality assurance team meets with the stakeholders to better understand the detailed knowledge of requirements.

In this phase **manager of the testing, team calculates the estimated effort and cost for the testing work.** This phase gets started once the requirement-gathering phase is completed.

In this phase testing team notes down the detailed test cases. **The testing team also prepares the required test data for the testing.** When the test cases are prepared then they are reviewed by the quality assurance team.

Software Testing Life Cycle

Test Closure

The testing team should have a clear understanding of the software's quality and reliability, and any defects or issues that were identified during testing should have been resolved. The test closure stage also includes documenting the testing process and any lessons learned so that they can be used to improve future testing processes.

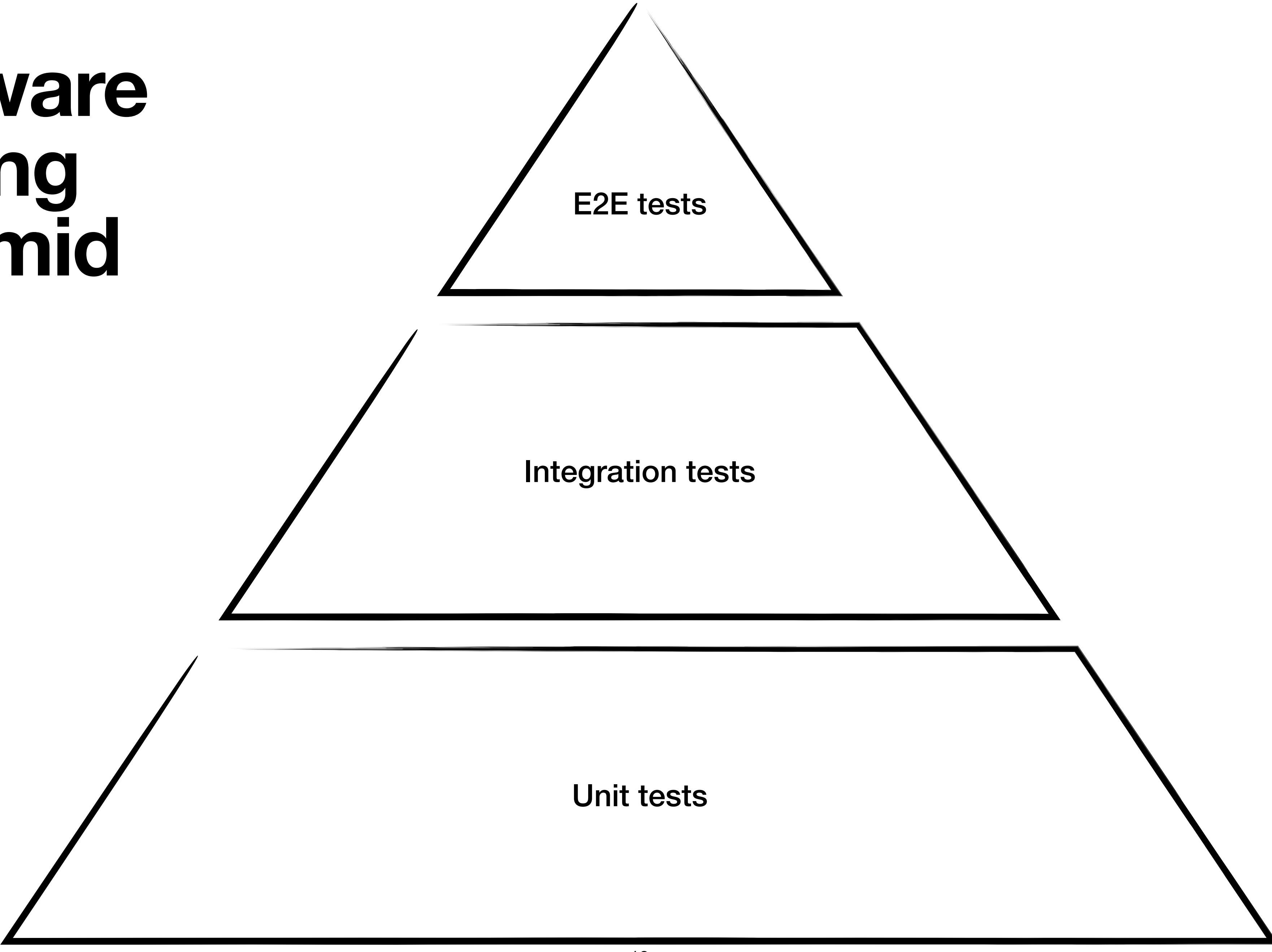
In this phase testing team starts executing test cases based on prepared test cases in the earlier step.

Test Execution

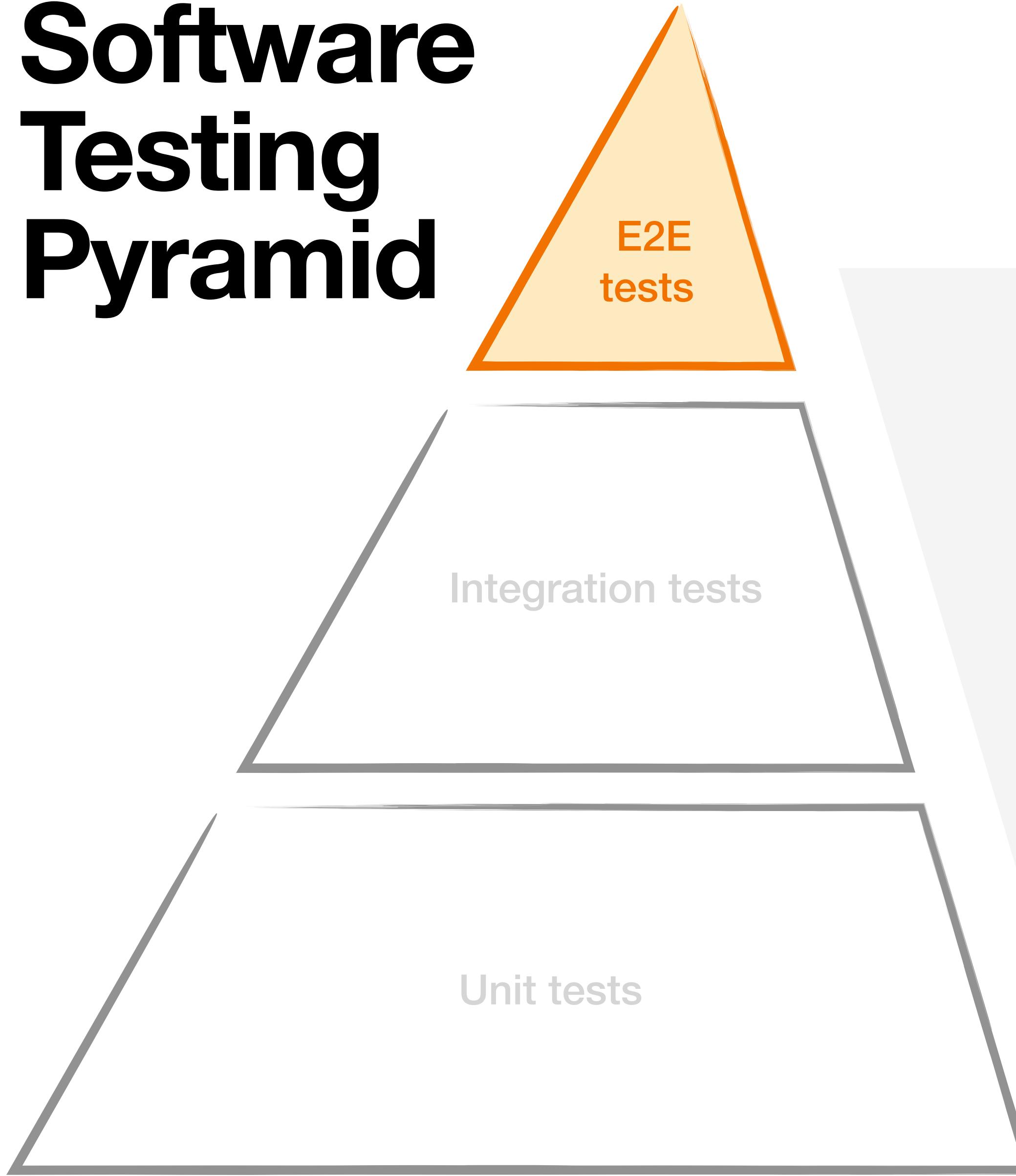
Basically, the test environment decides the conditions on which software is tested. This is independent activity and can be started along with test case development. In this process, the testing team is not involved. Either the developer or the customer creates the testing environment.

Test environment Setup

Software Testing Pyramid



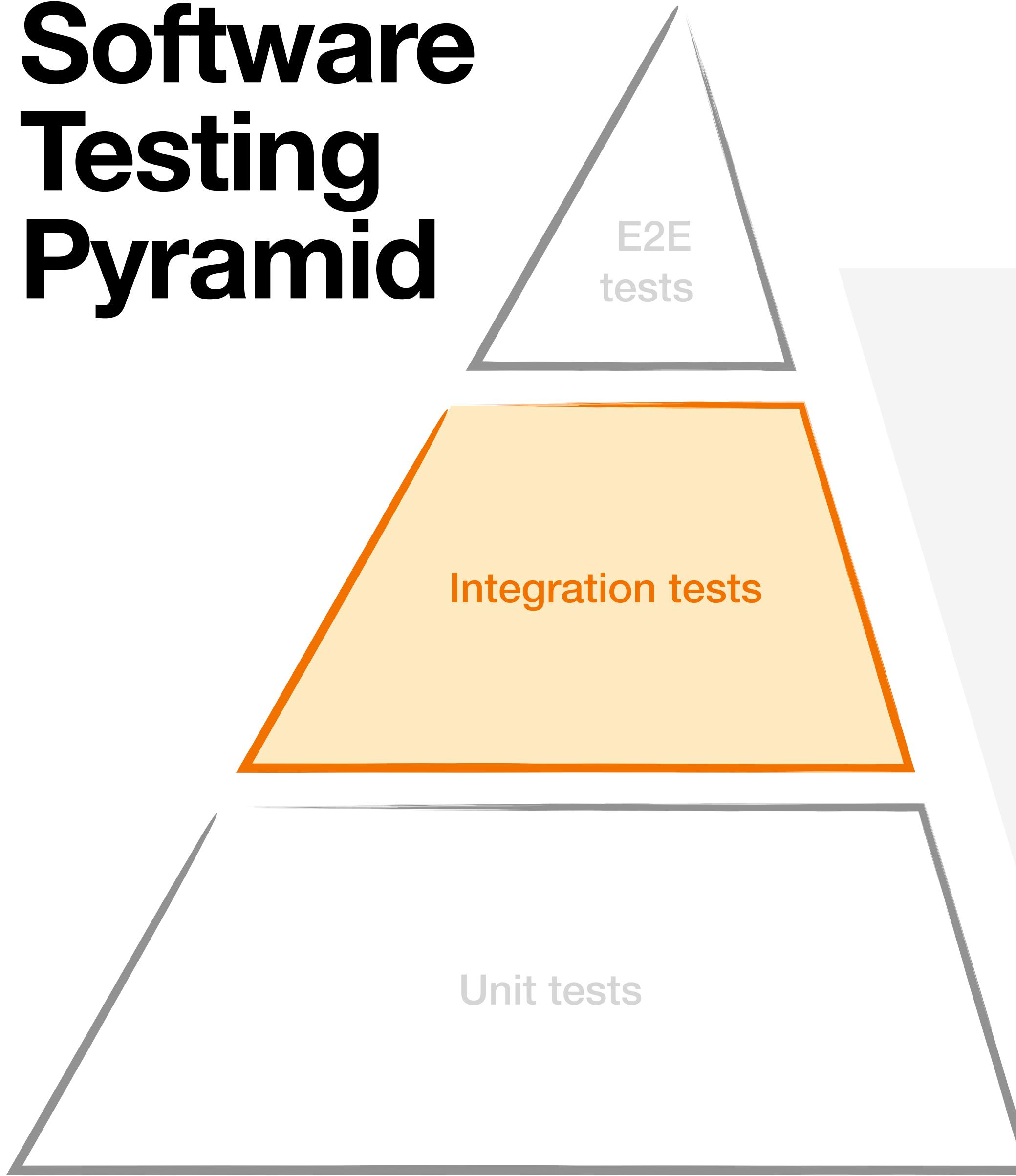
Software Testing Pyramid



The aim of this type of End-to-End (E2E) testing is to evaluate whether the system complies with the end-user requirements and if it is ready for deployment. The scope of acceptance testing ranges from simply finding spelling mistakes and cosmetic errors, to uncovering bugs that could cause a major error in the application.

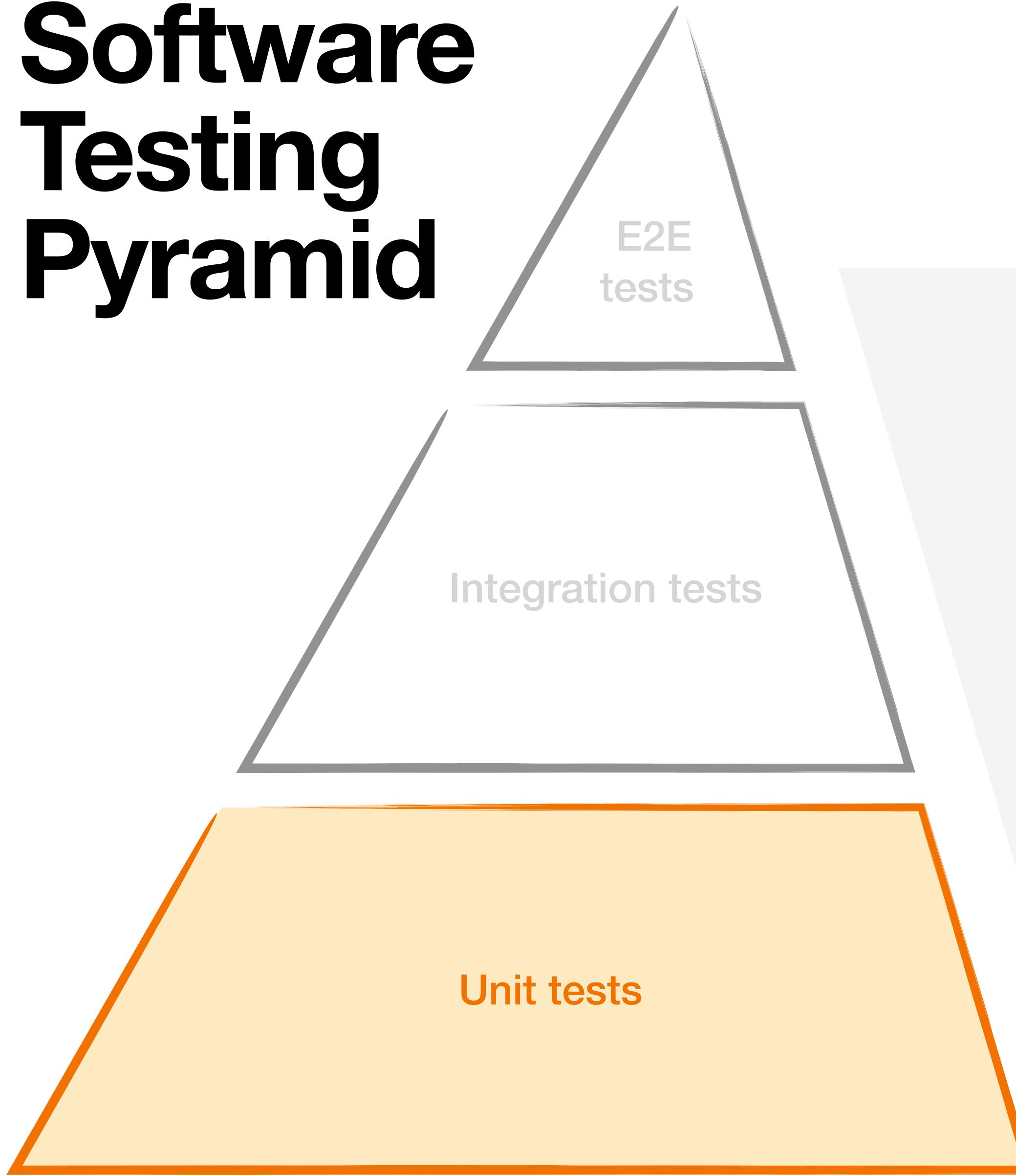
By performing acceptance tests, the testing team can find out how the product will perform when it is installed on the user's system. There are also various legal and contractual reasons why acceptance testing has to be carried out.

Software Testing Pyramid



Integration testing aims to test different parts of the system in combination in order to assess if they work correctly together. By testing the units in groups, any faults in the way they interact together can be identified.

Software Testing Pyramid



Unit testing aims to verify each part of the software by isolating it and then perform tests to demonstrate that each individual component is correct in terms of fulfilling requirements and the desired functionality.

This type of testing is performed at the earliest stages of the development process, and in many cases it is executed by the developers themselves before handing the software over to the testing team.

The advantage of detecting any errors in the software early in the day is that by doing so the team minimize software development risks, as well as time and money wasted in having to go back and undo fundamental problems in the program once it is nearly completed.

End-to-End Tests

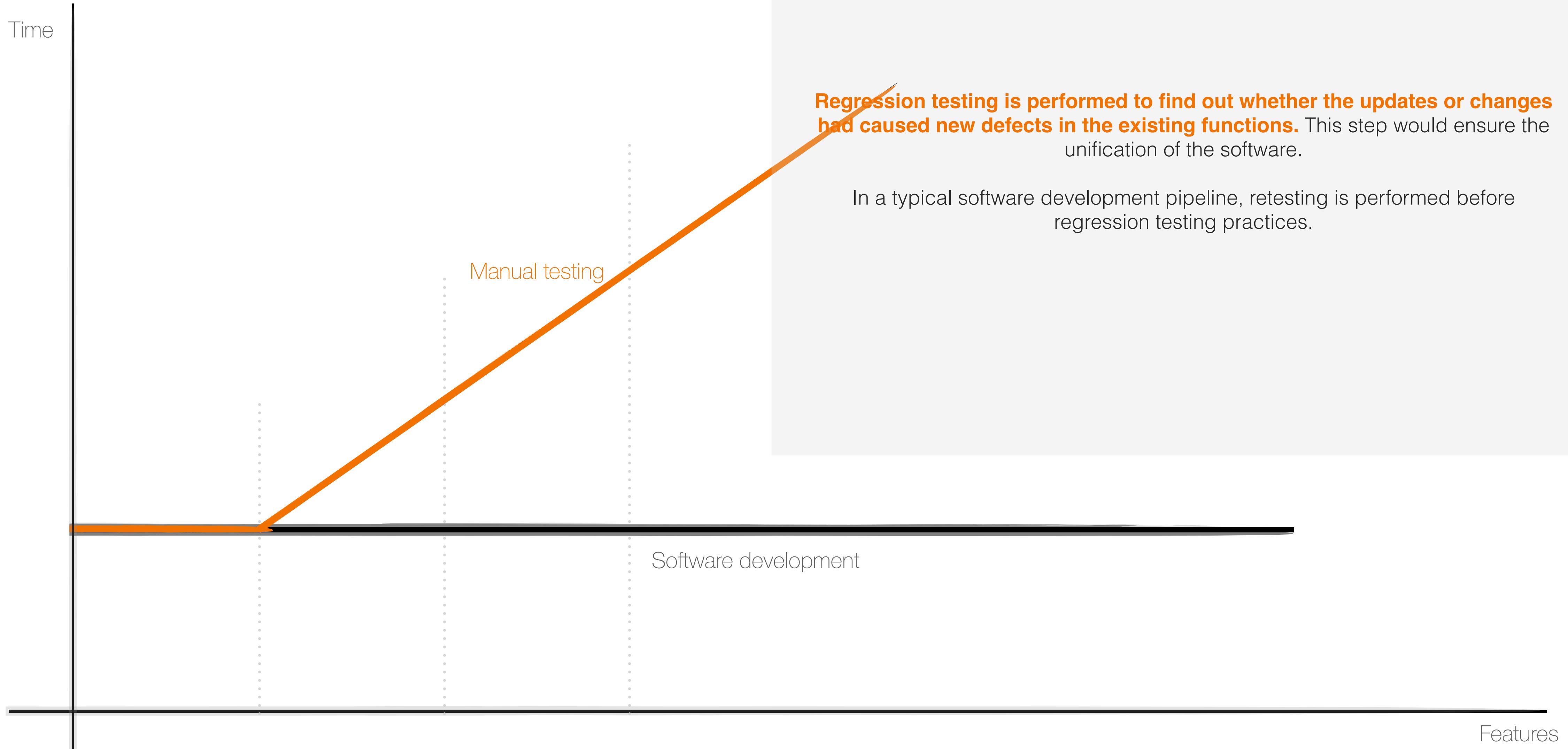
Strengths

- **Realistic Testing:** E2E tests mimic real user interactions with the application, making them ideal for assessing how the software behaves in a production-like environment.
- **Comprehensive Coverage:** E2E tests can cover a wide range of functionalities, ensuring that the entire system functions correctly from end to end. This can help catch integration issues and ensure the different components of the application work together seamlessly.
- **User Perspective:** These tests focus on the user's perspective, helping identify usability and user experience issues that might not be apparent in other testing methods.
- **Regression Testing:** E2E tests are useful for regression testing, as they can quickly uncover issues introduced by new code changes or updates.
- **Validation of Critical Paths:** They are particularly effective at testing critical paths in the application, ensuring that essential functionality is working as intended.
- **Detection of Business Logic Errors:** E2E tests can uncover business logic errors that might not be caught by unit tests or other testing methods.

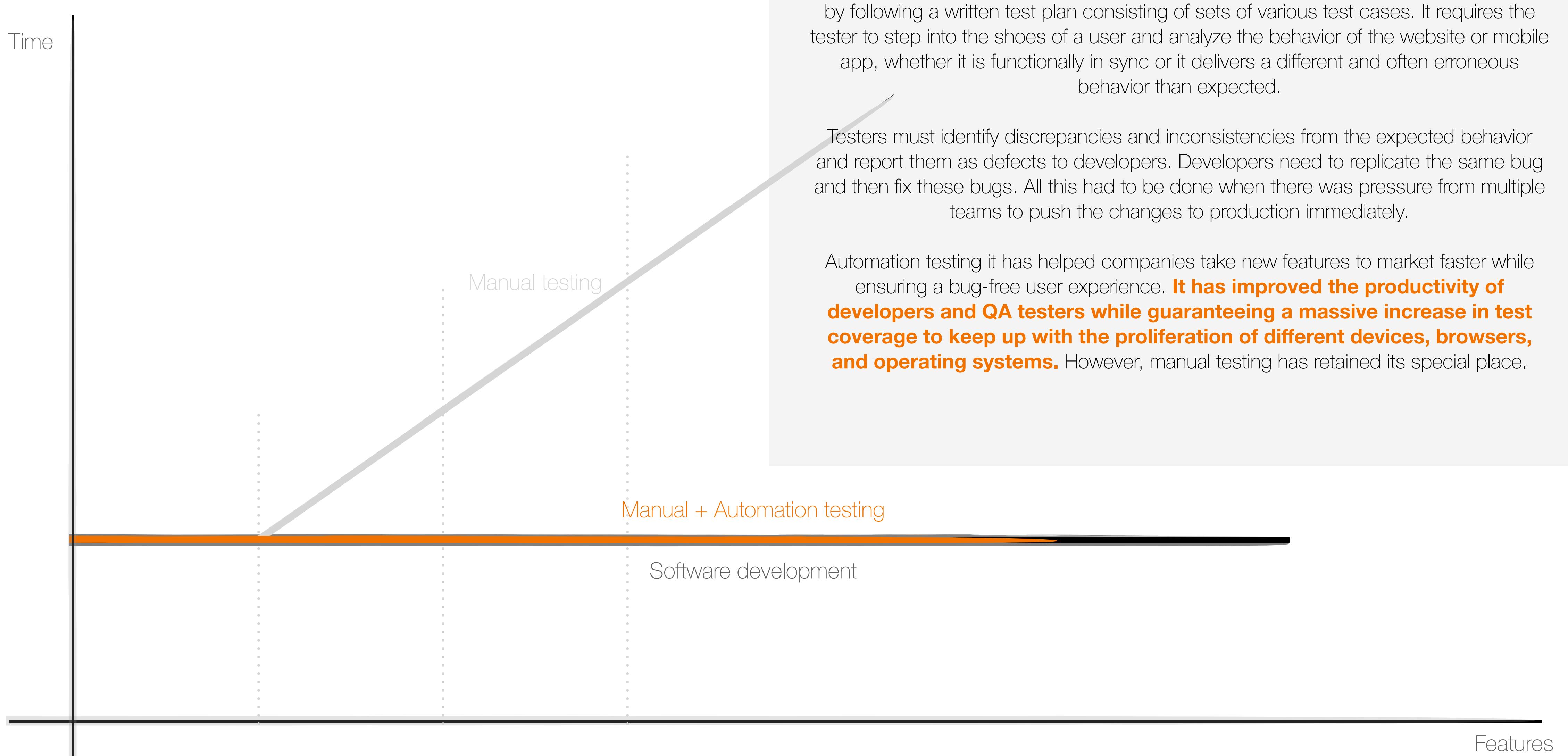
Weaknesses

- **Complexity:** Writing and maintaining E2E tests can be complex and time-consuming. Test scripts often require significant effort to develop and keep up-to-date as the application evolves.
- **Fragility:** E2E tests can be fragile and sensitive to changes in the UI or application structure. Minor UI changes can lead to test failures, requiring constant test maintenance.
- **Resource-Intensive:** Running E2E tests typically consumes more resources (time, hardware, and software) compared to other types of tests, which can slow down the development and testing process.
- **Limited Debugging Information:** When E2E tests fail, they may not provide detailed information about the root cause of the issue, making it challenging to diagnose and fix problems.
- **Late Discovery of Issues:** E2E tests are often conducted later in the development cycle, which means issues may be discovered after significant work has been done, leading to costly rework.
- **Not Suitable for All Testing Needs:** E2E tests may not be suitable for every testing scenario. They are less effective at testing individual components or units in isolation, which is better suited for unit tests.
- **Cost:** The cost of setting up and maintaining an E2E testing infrastructure, including test environments and tools, can be high.

Manual Testing

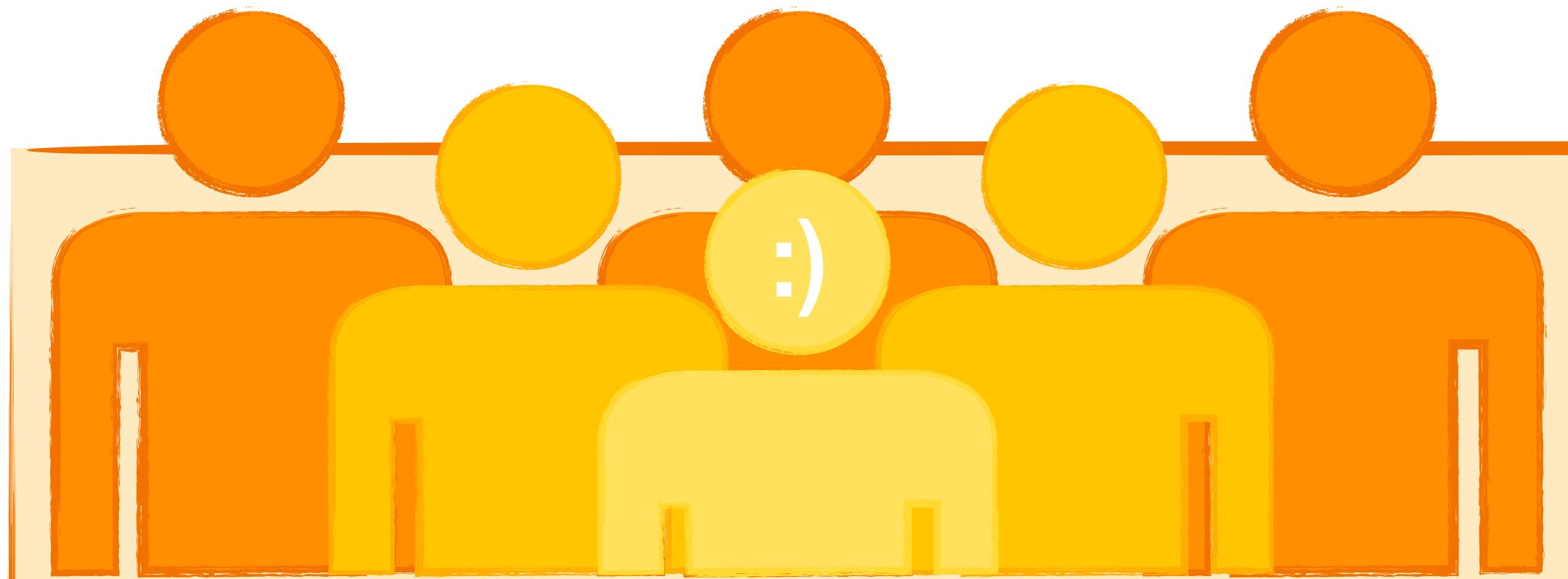


Automation Testing



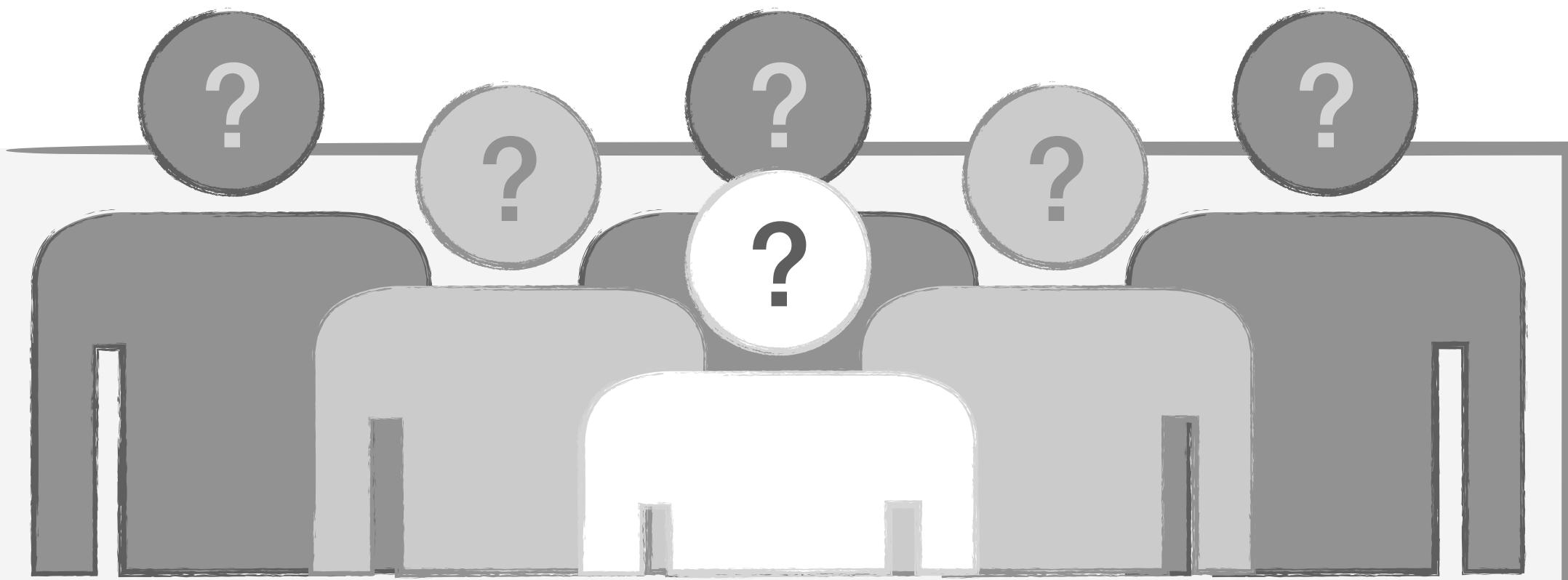
Tester

Alpha



Alpha Testing is a type of acceptance testing; **performed to identify all possible issues and bugs before releasing the final product to the end users**. Alpha testing is carried out by the testers who are internal employees of the organization. The main goal is to identify the tasks that a typical user might perform and test them.

Beta



Beta Testing is **performed by “real users” of the software application in “real environment”** and it can be considered as a form of external User Acceptance Testing. It is the final test before shipping a product to the customers. Direct feedback from customers is a major advantage of Beta Testing. This testing helps to test products in customer's environment.

Playwright

An open-source browser automation library developed by Microsoft.

It supports multiple web browsers, including Chromium, Firefox, and WebKit.

Playwright allows developers to automate browser tasks such as page navigation, form submission, and interaction with web elements.

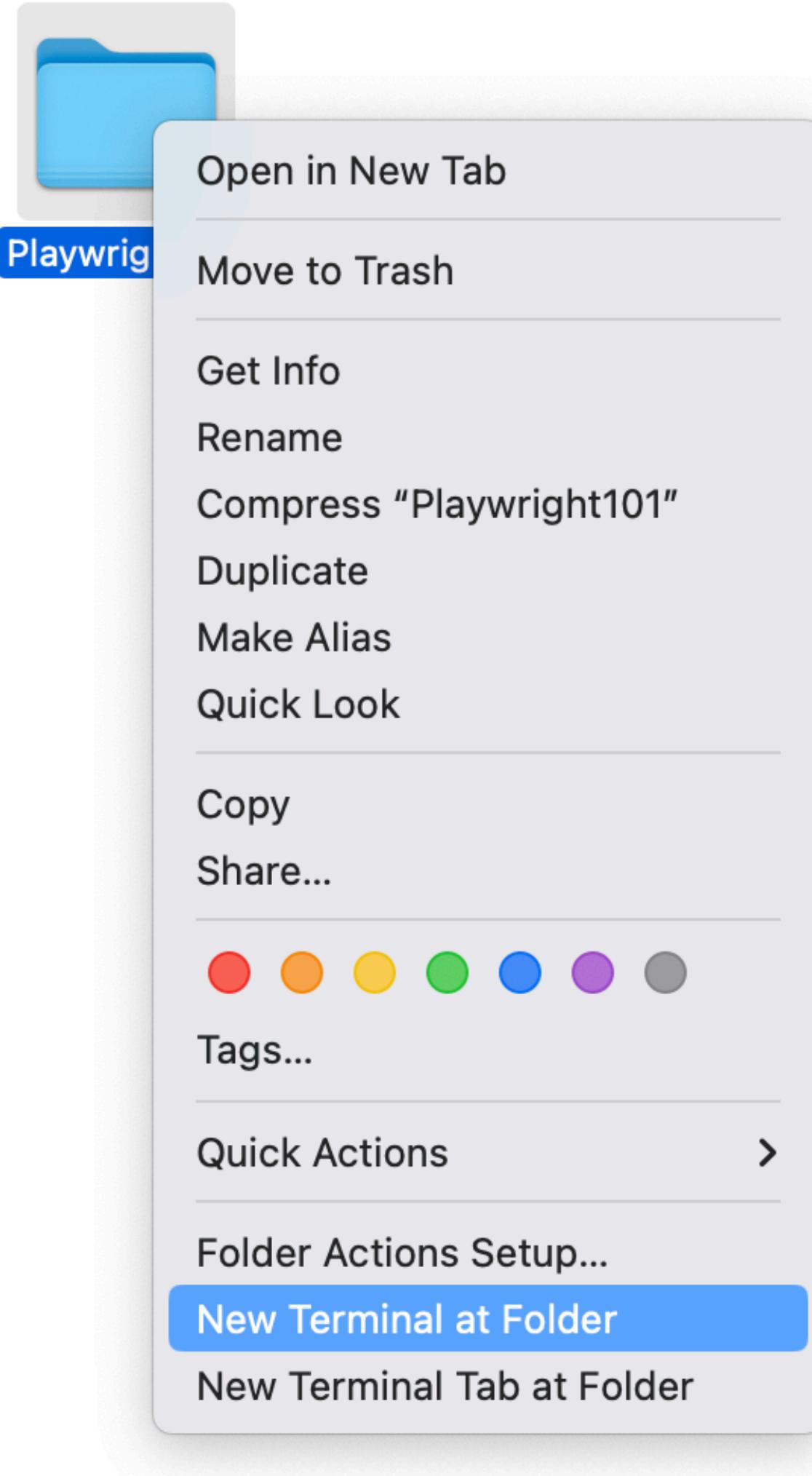


Frameworks Comparison



Aspect	Robot Framework	Playwright	Cypress
Type	Keyword-driven test automation framework	Node.js library for browser automation	JavaScript end-to-end testing framework
Language	Uses its own domain-specific language (keywords)	JavaScript, Python, .NET APIs available	JavaScript
Focus	General-purpose, supports various types of testing	Browser automation across different browsers	Web application end-to-end testing
Browser Support	Supports multiple browsers through Selenium	Supports Chromium, Firefox, WebKit	Primarily focused on Chrome
User Interfaces Support	Supports web, mobile, and desktop automation	Web and browser automation only	Web browser automation only
Parallel Execution	Supports parallel execution of tests	Supports parallel execution	Supports parallel execution
Community & Ecosystem	Active community and many libraries	Rapidly growing ecosystem and active development	Active community and plugin ecosystem
Testing Level	Supports unit, integration, and acceptance testing	Primarily focused on end-to-end testing	Primarily focused on end-to-end testing
Flexibility	Highly flexible due to keyword-driven nature	Flexible and powerful due to code-based approach	Flexible and powerful due to code-based approach
Maintenance	Requires maintenance of keyword libraries	Requires maintenance of codebase	Requires maintenance of codebase
Popularity	Widely used in test automation	Gaining popularity, especially for modern web apps	Popular choice for web testing

Initial Playwright project



Run command:

npm init playwright@latest

A screenshot of a terminal window titled "~/Repositories/Playground/Playwright101". The terminal shows the following output:

```
Last login: Wed Nov 15 21:18:09 on ttys014
[jojo@Sattawats-MacBook-Air Playwright101 % npm init playwright@latest
Need to install the following packages:
  create-playwright@1.17.131
Ok to proceed? (y)
Getting started with writing end-to-end tests with Playwright:
Initializing project in '.'
? Do you want to use TypeScript or JavaScript? ...
> TypeScript
  JavaScript
```

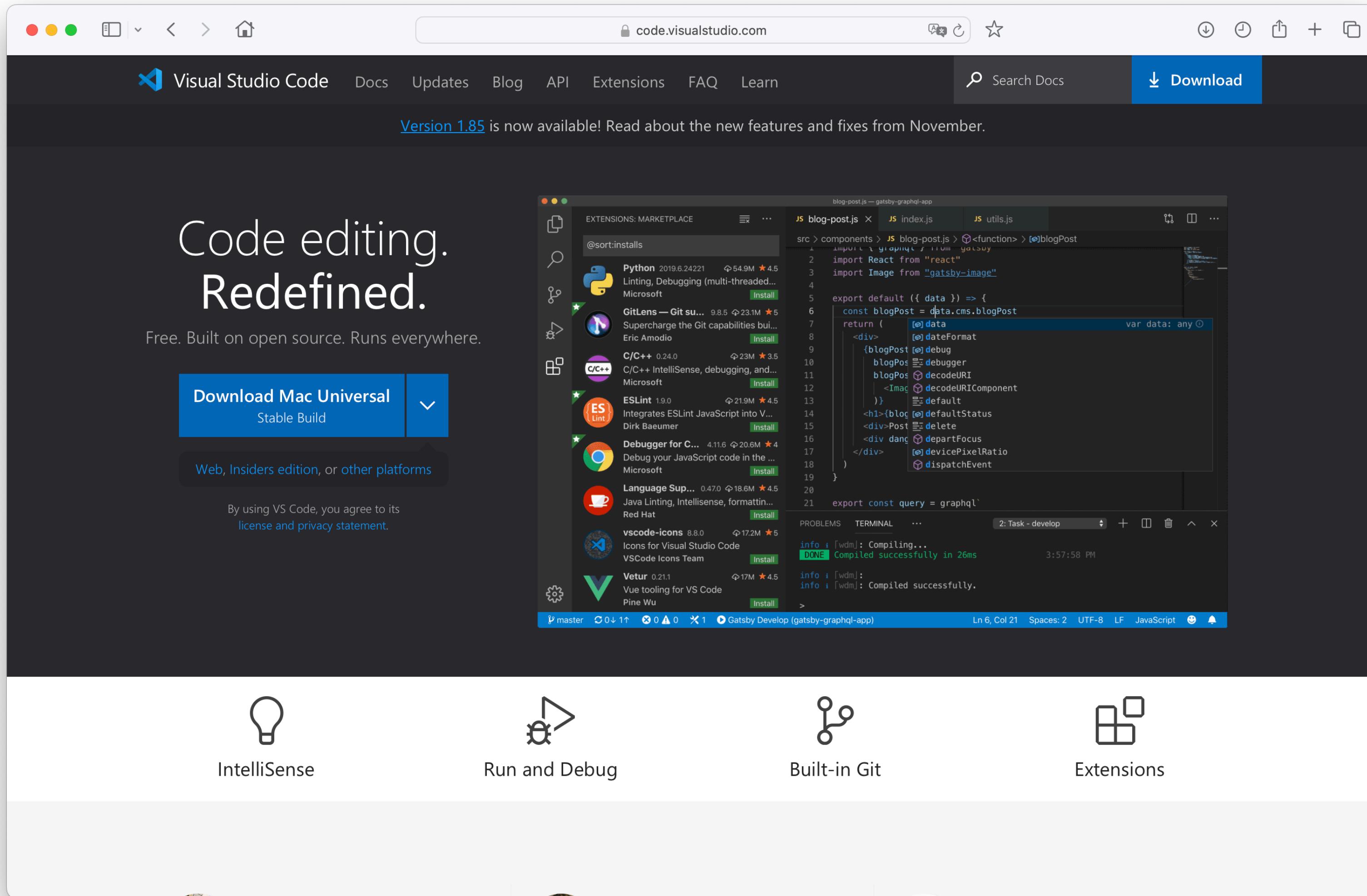
Installations

- VSCode
- Windows
 - Node.js
- Mac
 - Brew
 - Node.js

Install VSCode

Go to this official website => <https://code.visualstudio.com/>

It has already suggested the compatible version with your environment



Windows: Download Node.js

Go to this official website => <https://nodejs.org/en/download> , then download the Windows version

The screenshot shows the Node.js download page. At the top, there are two tabs: "LTS" (Recommended For Most Users) and "Current" (Latest Features). The "LTS" tab is selected. Below the tabs, there are three main download options:

- LTS (Windows Installer):** node-v20.10.0-x64.msi
- Current (macOS Installer):** node-v20.10.0.pkg
- Current (Source Code):** node-v20.10.0.tar.gz

Below these, there is a table showing download links for additional platforms:

	32-bit	64-bit	ARM64
Windows Installer (.msi)	node-v20.10.0-win32-ia32.msi	node-v20.10.0-win32-x64.msi	node-v20.10.0-win64-ia32.msi
Windows Binary (.zip)	node-v20.10.0-win32.zip	node-v20.10.0-win32-x64.zip	node-v20.10.0-win64-ia32.zip
macOS Installer (.pkg)	node-v20.10.0-macosx-ia32.pkg	node-v20.10.0-macosx-x64.pkg	node-v20.10.0-macos-arm64.pkg
macOS Binary (.tar.gz)	node-v20.10.0-macosx-ia32.tar.gz	node-v20.10.0-macosx-x64.tar.gz	node-v20.10.0-macos-arm64.tar.gz
Linux Binaries (x64)	node-v20.10.0-linux-x64-unknown.tar.gz	node-v20.10.0-linux-x64-unknown-musl.tar.gz	
Linux Binaries (ARM)	node-v20.10.0-linux-armv7-unknown.tar.gz	node-v20.10.0-linux-armv8-unknown.tar.gz	
Source Code	node-v20.10.0.tar.gz		

Below the table, there is a section titled "Additional Platforms" with links to Docker images and other platforms.

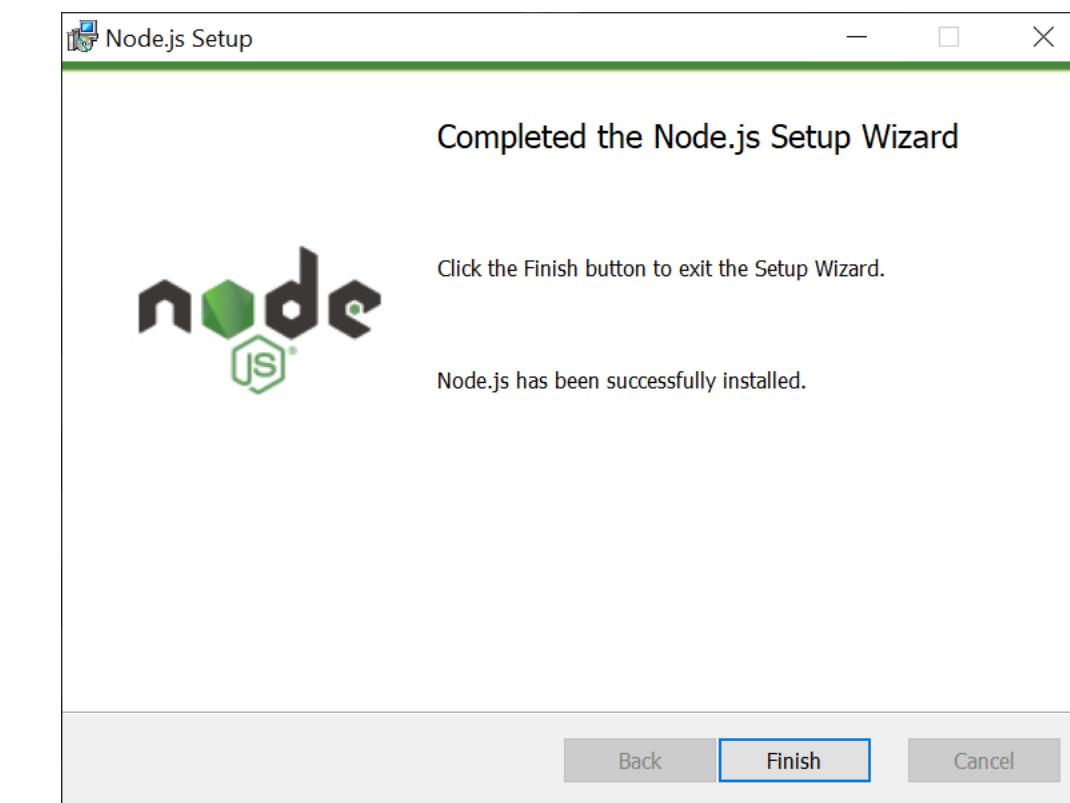
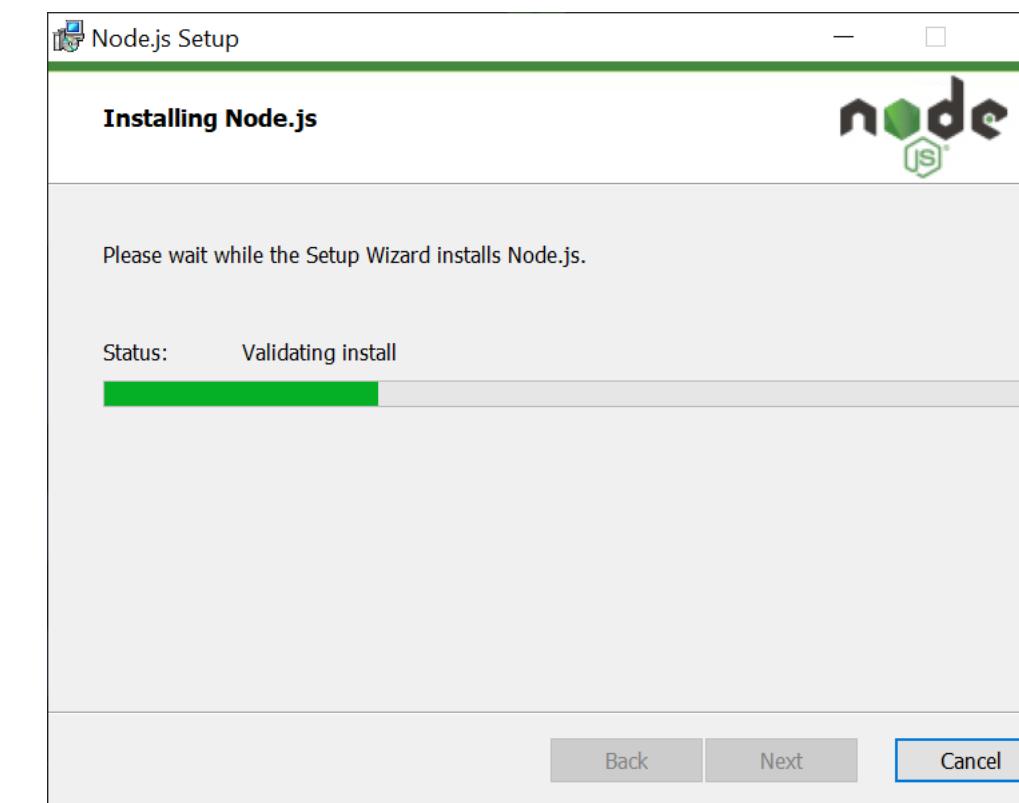
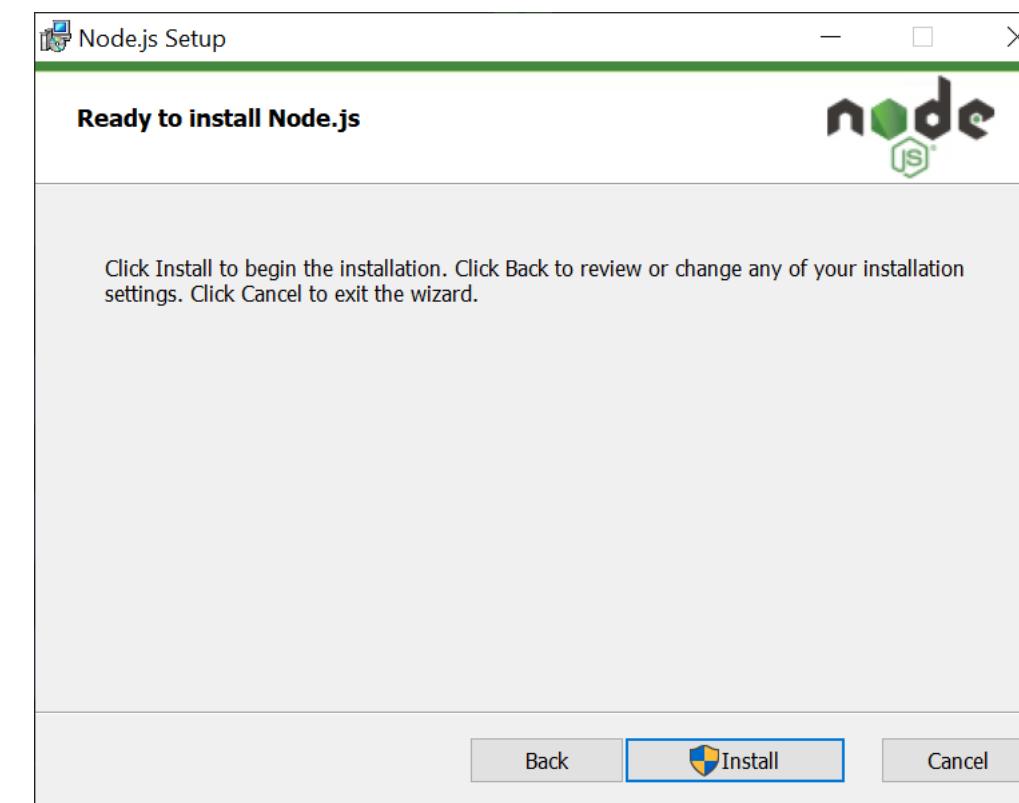
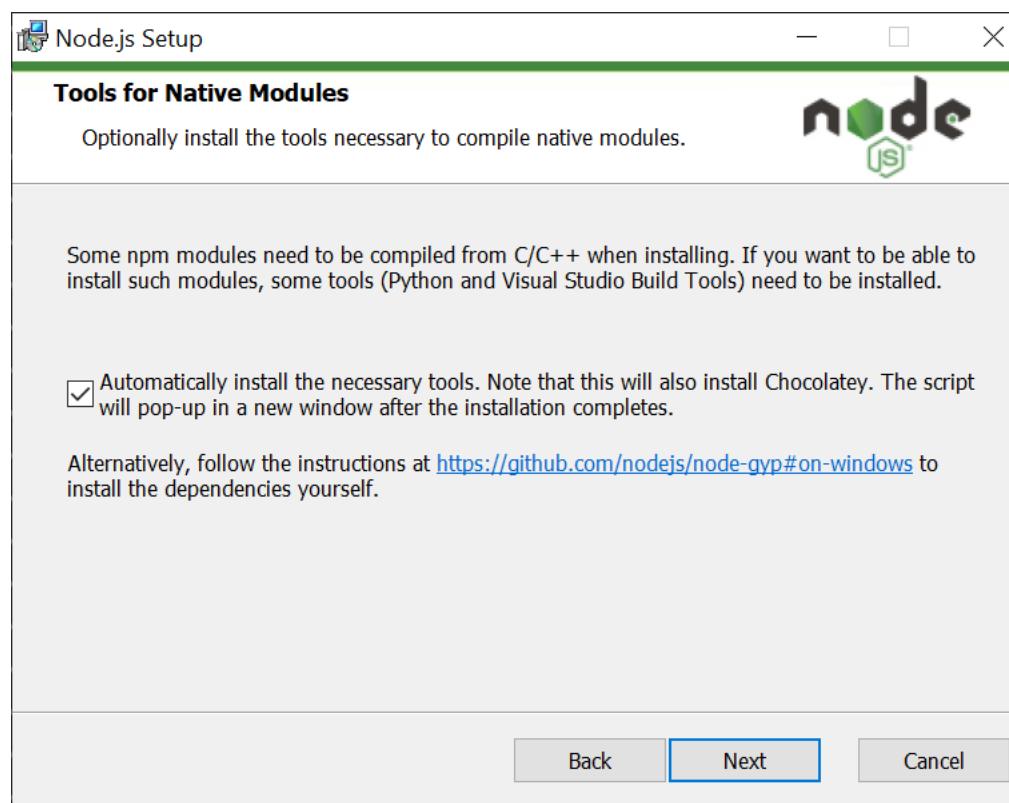
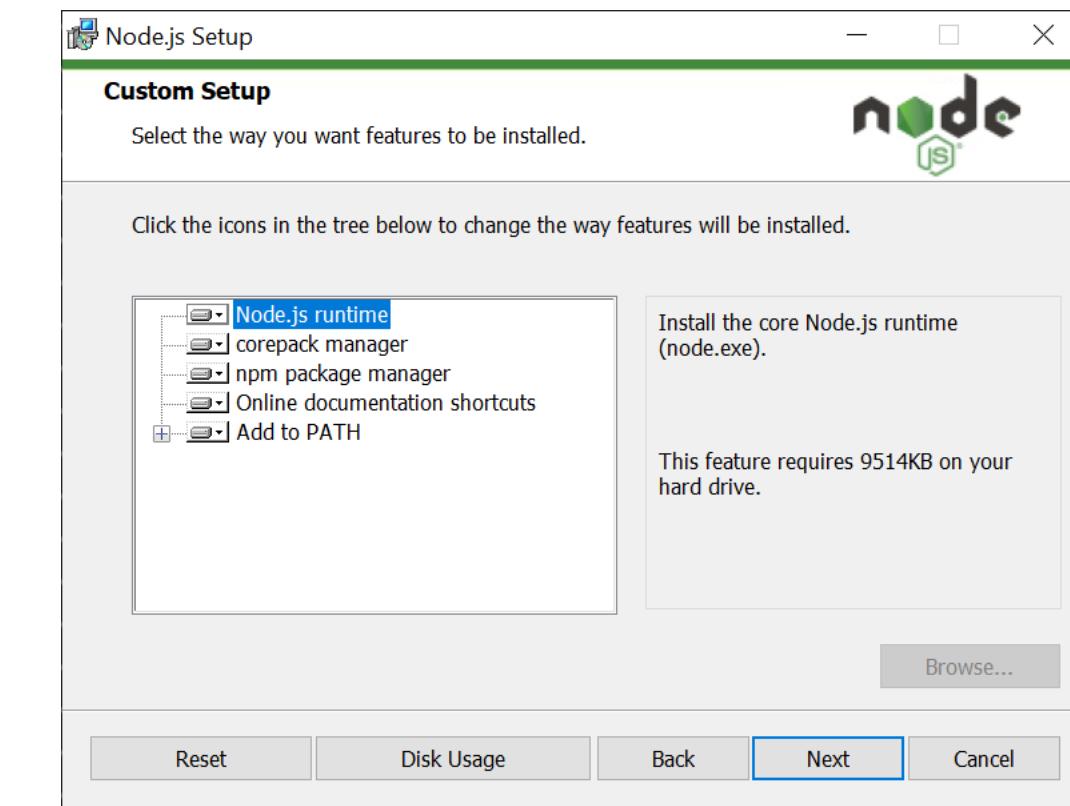
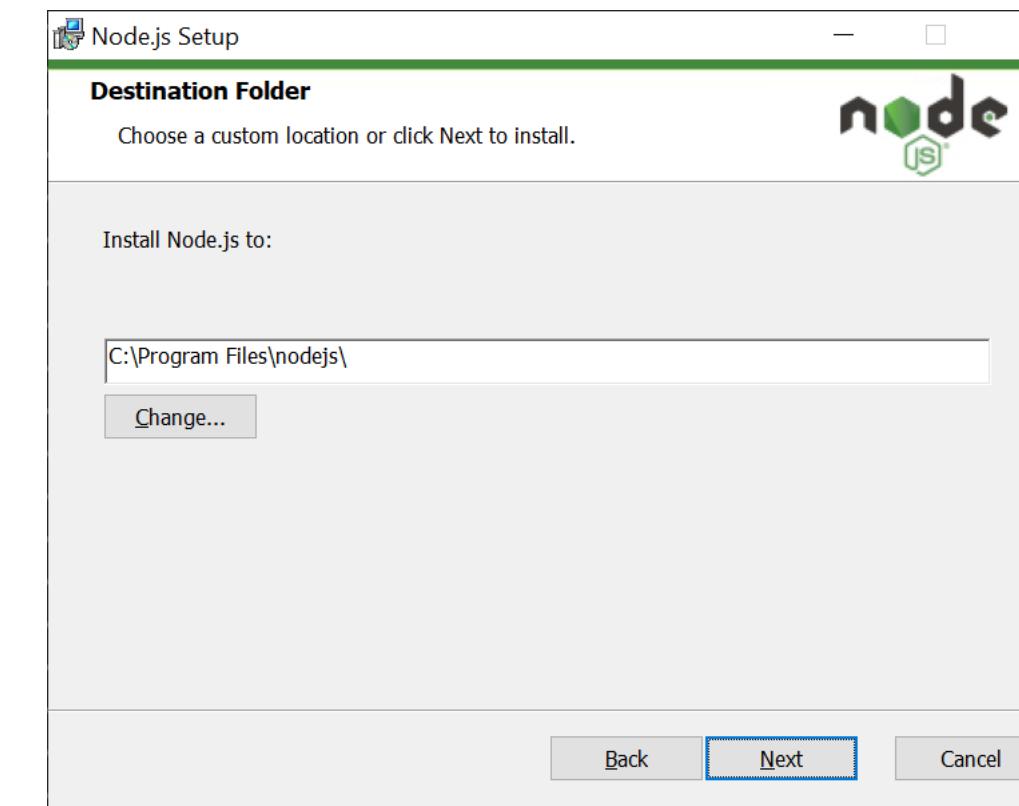
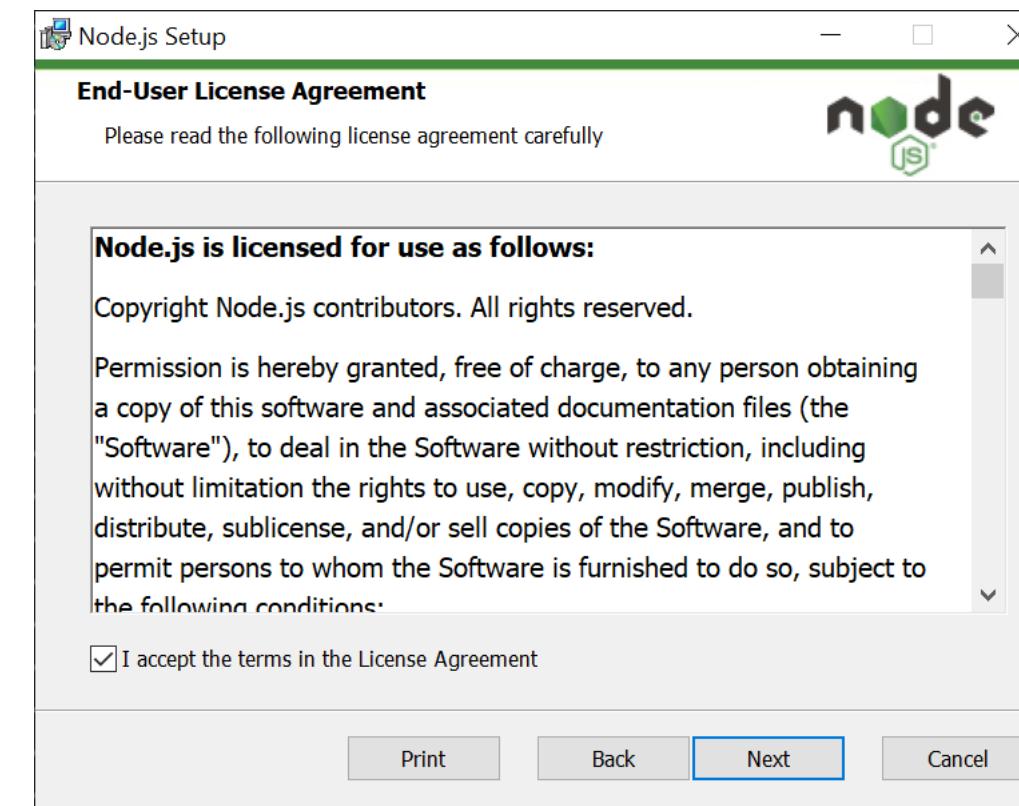
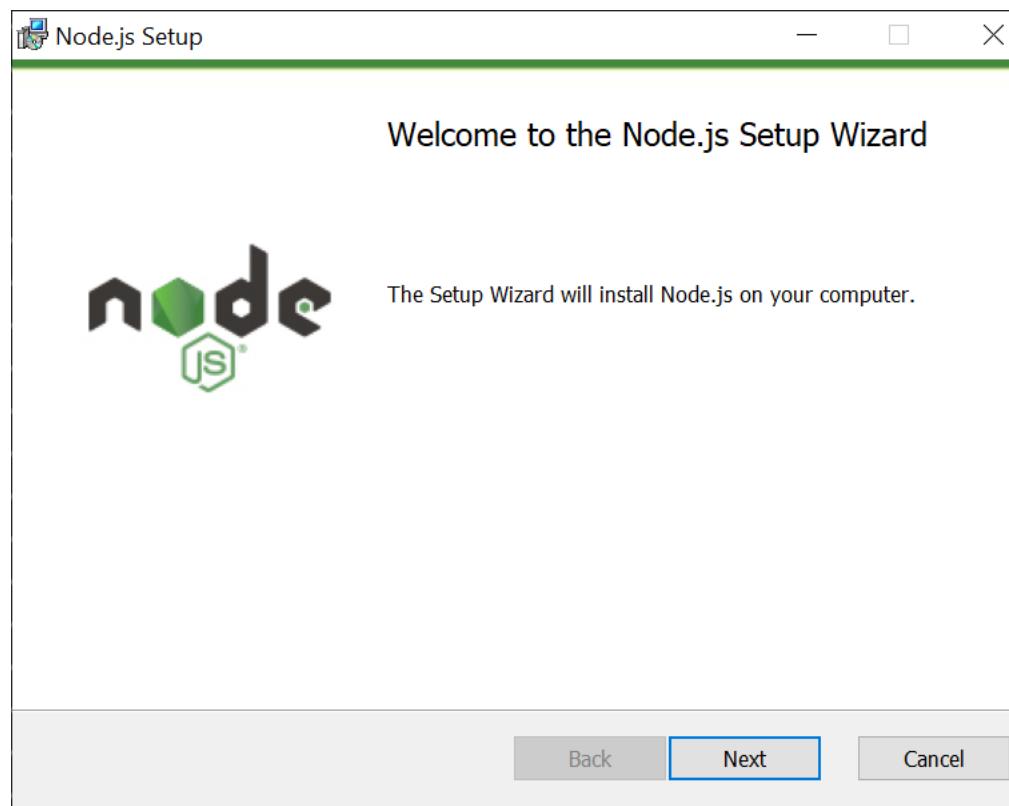
At the bottom left, there is a link: <https://nodejs.org/dist/v20.10.0/node-v20.10.0-x64.msi>

At the bottom right, there is a list of links:

- Signed SHASUMS for release files ([How to verify](#))
- All download options
- Installing Node.js via package manager
- Previous Releases
- Nightly builds

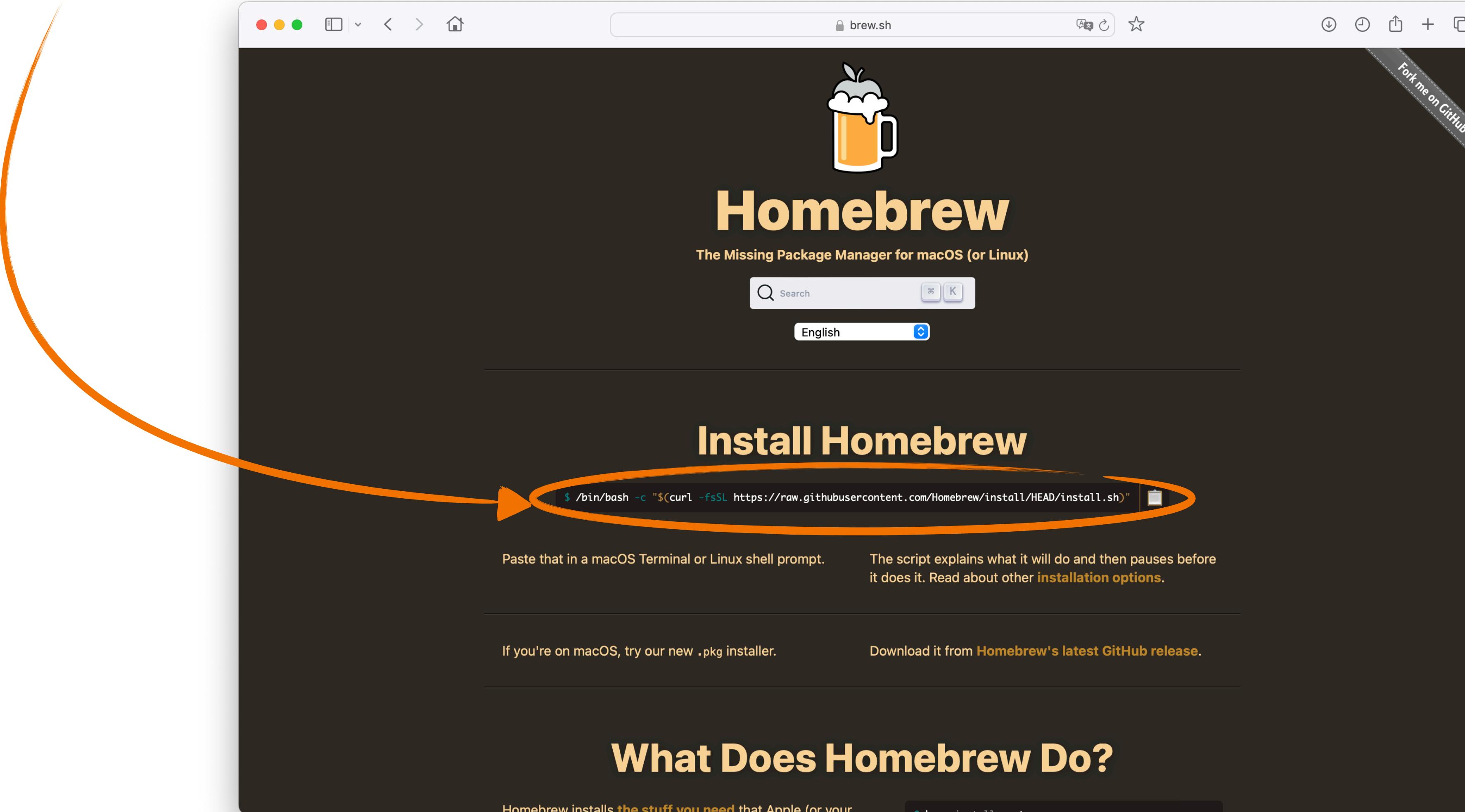
Windows: Install Node.js

So easy! Next, Next, Next, and Finish.



Mac: Homebrew website

At first, it might be a bit challenging, but your life will become more comfortable with Homebrew
Copy that command and paste it to your terminal



Mac: Install Homebrew

At first, it might be a bit challenging, but your life will become more comfortable with Homebrew
Copy that command and paste it to your terminal

After done installation, it will show you the next steps for setting brew path

```
[testuser@Sattawats-MacBook-Air ~ % /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"  
==> Checking for `sudo` access (which may request your password)...  
Password:  
[testuser@Sattawats-MacBook-Air ~ % /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"  
==> Checking for `sudo` access (which may request your password)...  
| Password:  
| ==> This script will install:  
| /opt/homebrew/bin/brew  
| /opt/homebrew/share/doc/homebrew  
| /opt/homebrew/share/man/man1/brew.1  
| /opt/homebrew/share/zsh/site-functions/_brew  
| /opt/homebrew/etc/bash_completion.d/brew  
| /opt/homebrew  
| Press RETURN/ENTER to continue or any other key to abort:  
[testuser@Sattawats-MacBook-Air ~ % brew untap homebrew/cask  
Warning: /opt/homebrew/bin is not in your PATH.  
Instructions on how to configure your shell for Homebrew  
can be found in the 'Next steps' section below.  
==> Installation successful!  
==> Homebrew has enabled anonymous aggregate formulae and cask analytics.  
Read the analytics documentation (and how to opt-out) here:  
https://docs.brew.sh/Analytics  
No analytics data has been sent yet (nor will any be during this install run).  
==> Homebrew is run entirely by unpaid volunteers. Please consider donating:  
https://github.com/Homebrew/brew#donations  
==> Next steps:  
- Run these two commands in your terminal to add Homebrew to your PATH:  
  (echo; echo 'eval "$(/opt/homebrew/bin/brew shellenv)"') >> /Users/testuser/.zprofile  
  eval "$(/opt/homebrew/bin/brew shellenv)"  
- Run brew help to get started  
- Further documentation:  
  https://docs.brew.sh
```

Mac: Set Homebrew path

Copy that provided 2 lines, and paste to terminal

```
testuser — zsh — 80x24
Warning: /opt/homebrew/bin is not in your PATH.
Instructions on how to configure your shell for Homebrew
can be found in the 'Next steps' section below.
==> Installation successful!

==> Homebrew has enabled anonymous aggregate formulae and cask analytics.
Read the analytics documentation (and how to opt-out) here:
https://docs.brew.sh/Analytics
No analytics data has been sent yet (nor will any be during this install run).

==> Homebrew is run entirely by unpaid volunteers. Please consider donating:
https://github.com/Homebrew/brew#donations

==> Next steps:
- Run these two commands in your terminal to add Homebrew to your PATH:
  (echo; echo 'eval "$( /opt/homebrew/bin/brew shellenv )"' ) >> /Users/testuser/.zprofile
  eval "$( /opt/homebrew/bin/brew shellenv )"
- Run brew help to get started
- Further documentation:
  https://docs.brew.sh

testuser@Sattawats-MacBook-Air ~ % (echo; echo 'eval "$( /opt/homebrew/bin/brew shellenv )"' ) >> /Users/testuser/.zprofile
```

```
testuser — zsh — 80x24
Instructions on how to configure your shell for Homebrew
can be found in the 'Next steps' section below.
==> Installation successful!

==> Homebrew has enabled anonymous aggregate formulae and cask analytics.
Read the analytics documentation (and how to opt-out) here:
https://docs.brew.sh/Analytics
No analytics data has been sent yet (nor will any be during this install run).

==> Homebrew is run entirely by unpaid volunteers. Please consider donating:
https://github.com/Homebrew/brew#donations

==> Next steps:
- Run these two commands in your terminal to add Homebrew to your PATH:
  (echo; echo 'eval "$( /opt/homebrew/bin/brew shellenv )"' ) >> /Users/testuser/.zprofile
  eval "$( /opt/homebrew/bin/brew shellenv )"
- Run brew help to get started
- Further documentation:
  https://docs.brew.sh

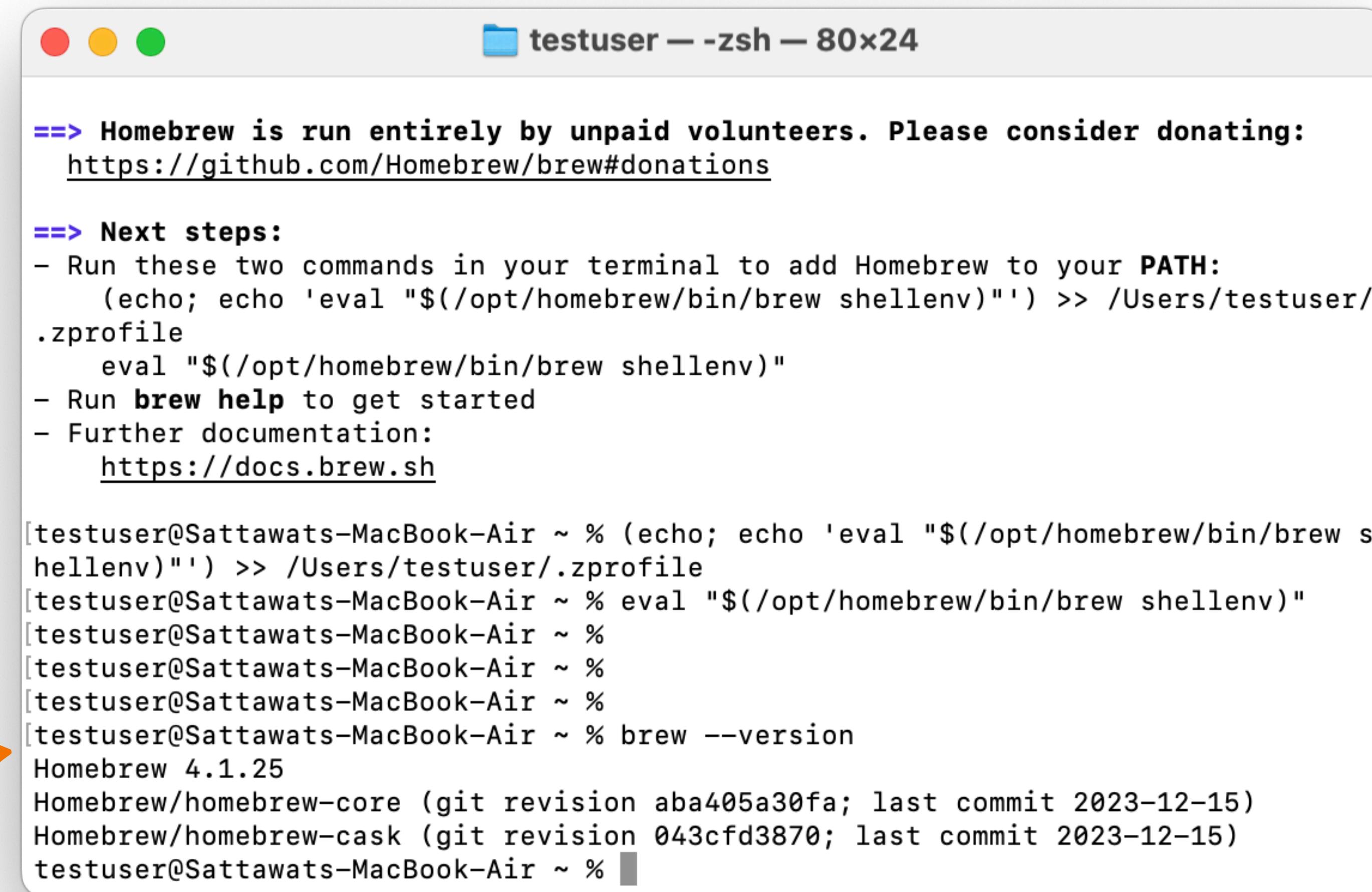
[testuser@Sattawats-MacBook-Air ~ % (echo; echo 'eval "$( /opt/homebrew/bin/brew shellenv )"' ) >> /Users/testuser/.zprofile
testuser@Sattawats-MacBook-Air ~ % eval "$( /opt/homebrew/bin/brew shellenv )"
```

Mac: Check Homebrew version

After set the brew path, check brew version with command "brew --version"

Check brew version:

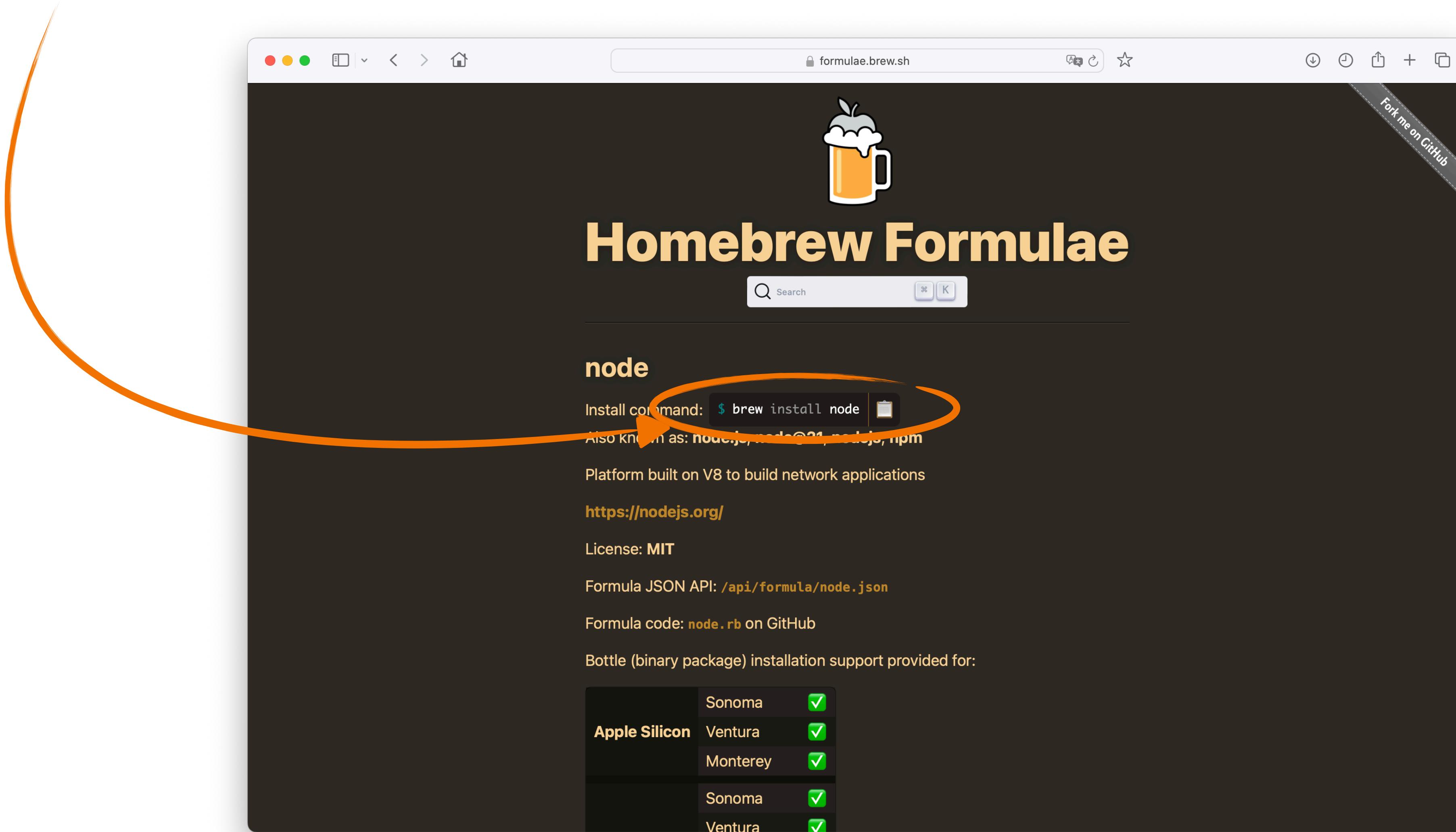
brew --version



```
==> Homebrew is run entirely by unpaid volunteers. Please consider donating:  
https://github.com/Homebrew/brew#donations  
  
==> Next steps:  
- Run these two commands in your terminal to add Homebrew to your PATH:  
  (echo; echo 'eval "$( /opt/homebrew/bin/brew shellenv )"' ) >> /Users/testuser/.zprofile  
  eval "$( /opt/homebrew/bin/brew shellenv )"  
- Run brew help to get started  
- Further documentation:  
  https://docs.brew.sh  
  
[testuser@Sattawats-MacBook-Air ~ % (echo; echo 'eval "$( /opt/homebrew/bin/brew shellenv )"' ) >> /Users/testuser/.zprofile  
[testuser@Sattawats-MacBook-Air ~ % eval "$( /opt/homebrew/bin/brew shellenv )"  
[testuser@Sattawats-MacBook-Air ~ %  
[testuser@Sattawats-MacBook-Air ~ %  
[testuser@Sattawats-MacBook-Air ~ %  
[testuser@Sattawats-MacBook-Air ~ % brew --version  
Homebrew 4.1.25  
Homebrew/homebrew-core (git revision aba405a30fa; last commit 2023-12-15)  
Homebrew/homebrew-cask (git revision 043cf3d3870; last commit 2023-12-15)  
testuser@Sattawats-MacBook-Air ~ %
```

Mac: Install Node.js

Just run, and wait



Playwright project structure

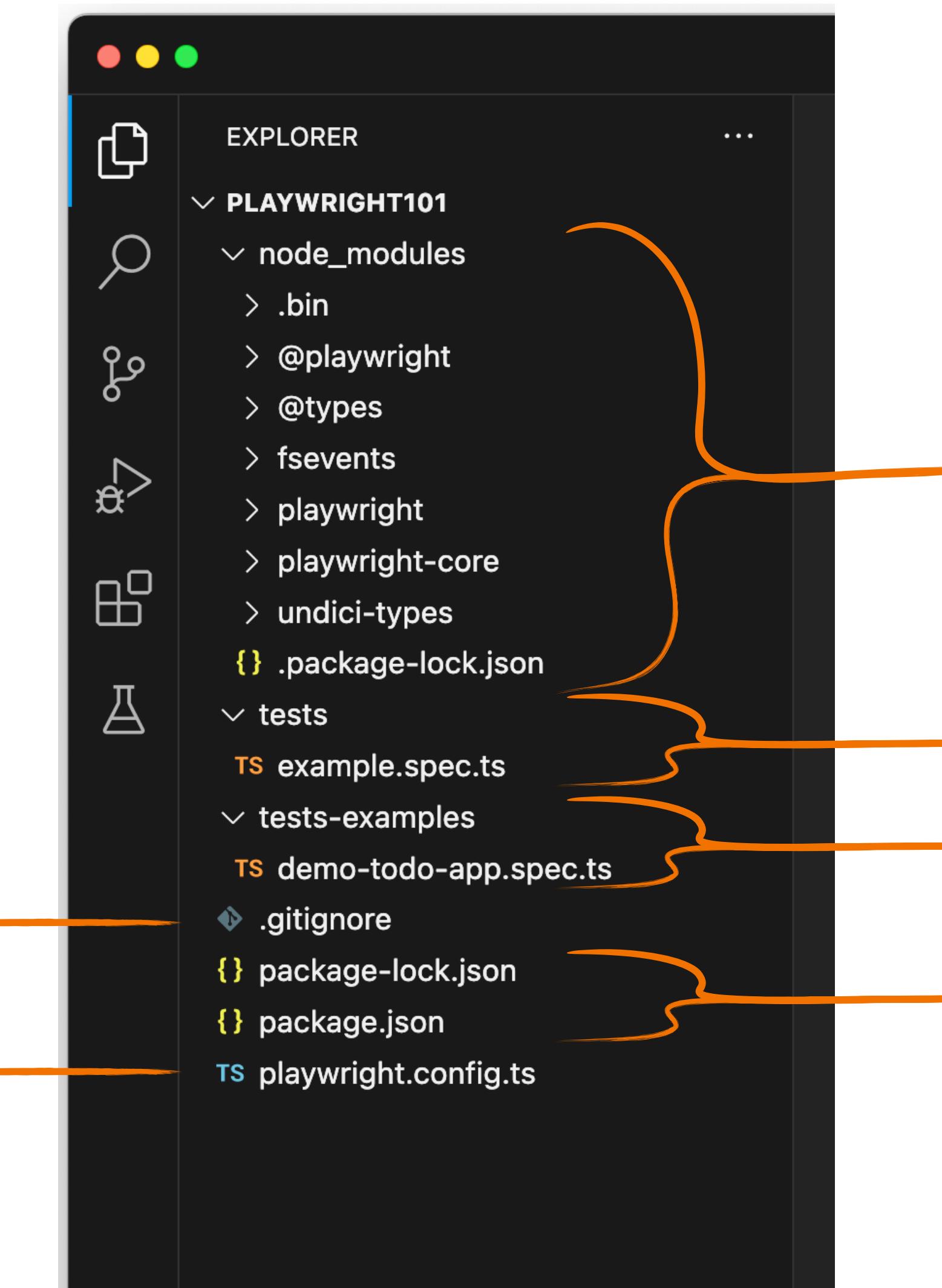
and

Example scenarios from init

Project structure

Specifies intentionally untracked files that Git should ignore

options to configure how your tests are run



Installed Node.js libraries

Our test scenarios will be implemented here

Example test scenarios from Playwright init

All you need to know about what's required in your project's dependency

Example test cases

The screenshot shows a dark-themed instance of VS Code with the following details:

- Explorer View:** Shows the project structure under "PLAYWRIGHT101". The "tests" folder contains "example.spec.ts", which is currently selected.
- Editor View:** Displays the content of "example.spec.ts". The code uses Playwright's test API to navigate to a page and check its title and a heading.
- Bottom Status Bar:** Shows file counts (0△0), a launch status (Launch Program (Playwright101)), and the current file information (Ln 19, Col 1, Spaces: 2, UTF-8, LF, {}, TypeScript).
- Bottom Navigation Bar:** Includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TEST RESULTS (which is active), TERMINAL, PORTS, and ROBOT OUTPUT.
- Bottom Right Panel:** Shows the "TEST RESULTS" panel with a list of test runs and their outcomes. It includes entries for "Test run at 11/21/2023, 9:50:..." (with "has title" and "get started link" checked) and "Test run at 11/21/2023, 9:50:48 PM" (unchecked).

Example test cases: has title

The screenshot shows a VS Code interface with the following details:

- EXPLORER:** Shows the project structure under "PLAYWRIGHT101". A tooltip "First test case named 'has title'" is shown over the "tests" folder.
- EDITOR:** The file "example.spec.ts" is open, displaying the following code:

```
1 import { test, expect } from '@playwright/test';
2
3 test('has title', async ({ page }) => {
4   await page.goto('https://playwright.dev/');
5
6   // Expect a title "to contain" a substring.
7   await expect(page).toHaveTitle(/Playwright/);
8 });
9
10 test('get started link', async ({ page }) => {
11   await page.goto('https://playwright.dev/');
12
13   // Click the get started link.
14   await page.getByRole('link', { name: 'Get started' }).click();
15
16   // Expects page to have a heading with the name of Installation.
17   await expect(page.getByRole('heading', { name: 'Installation' })).toBeVisible();
18 });
19
```
- TEST RESULTS:** A tooltip shows the results of a recent test run:
 - Test run at 11/21/2023, 9:50:... (Success)
 - has title (Success)
 - get started link (Success)
 - Test run at 11/21/2023, 9:50:48 PM (Success)
- STATUS BAR:** Shows "Ln 19, Col 1" and "Spaces: 2" and "TypeScript".

Test scenario structure

test : A function (method) named 'test' requires two arguments: 'title' and 'testFunction'

```
~ 3 test(  
~ 4   ,  
~ 5   |  
~ 6   |  
~ 7   |  
~ 8   );
```

title : A test scenario name must be unique within the same test suite

'has · title'

testFunction : A function that contains our test steps, marked as an **asynchronous (async)** function that will wait for any step marked with 'await'

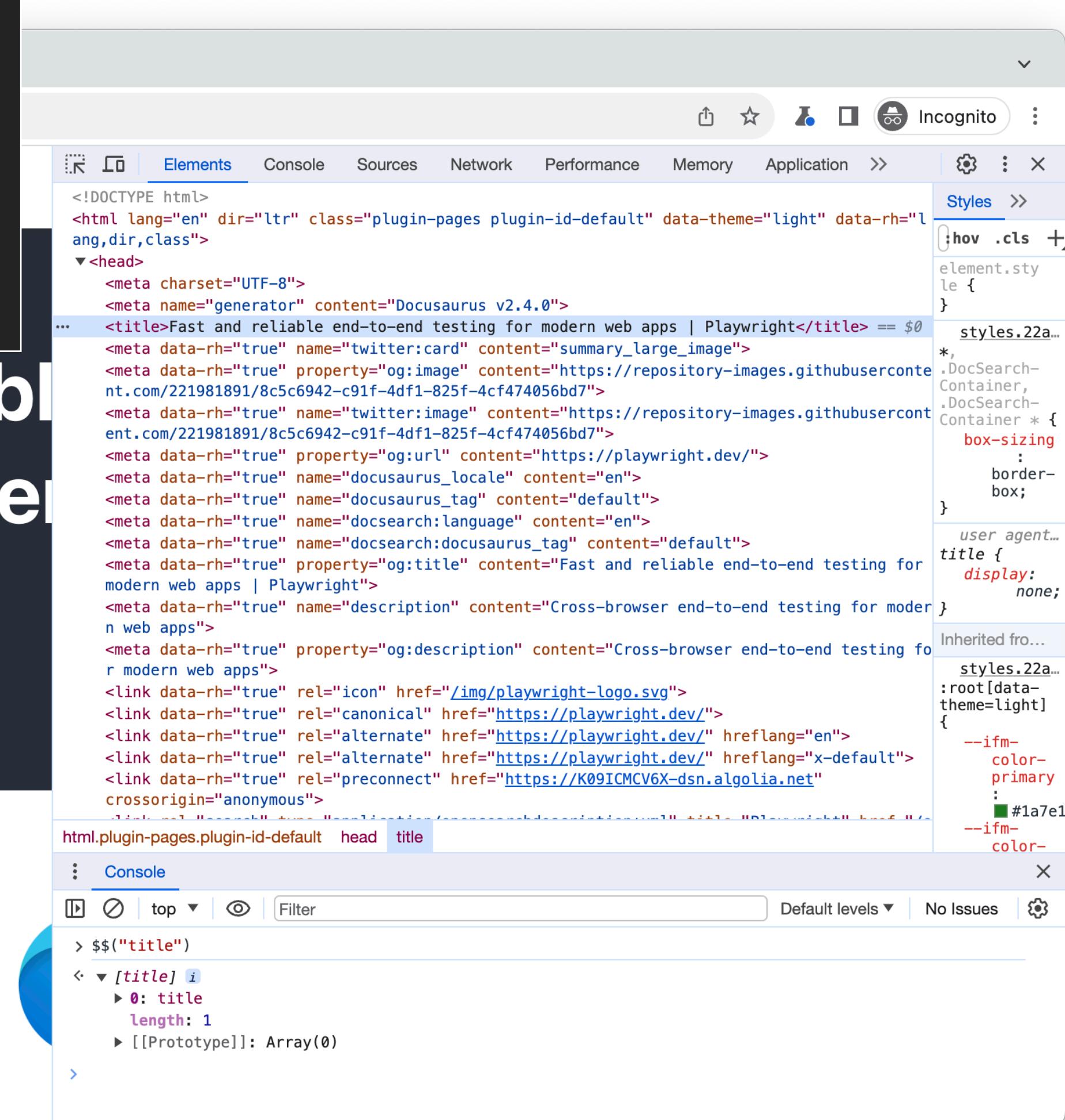
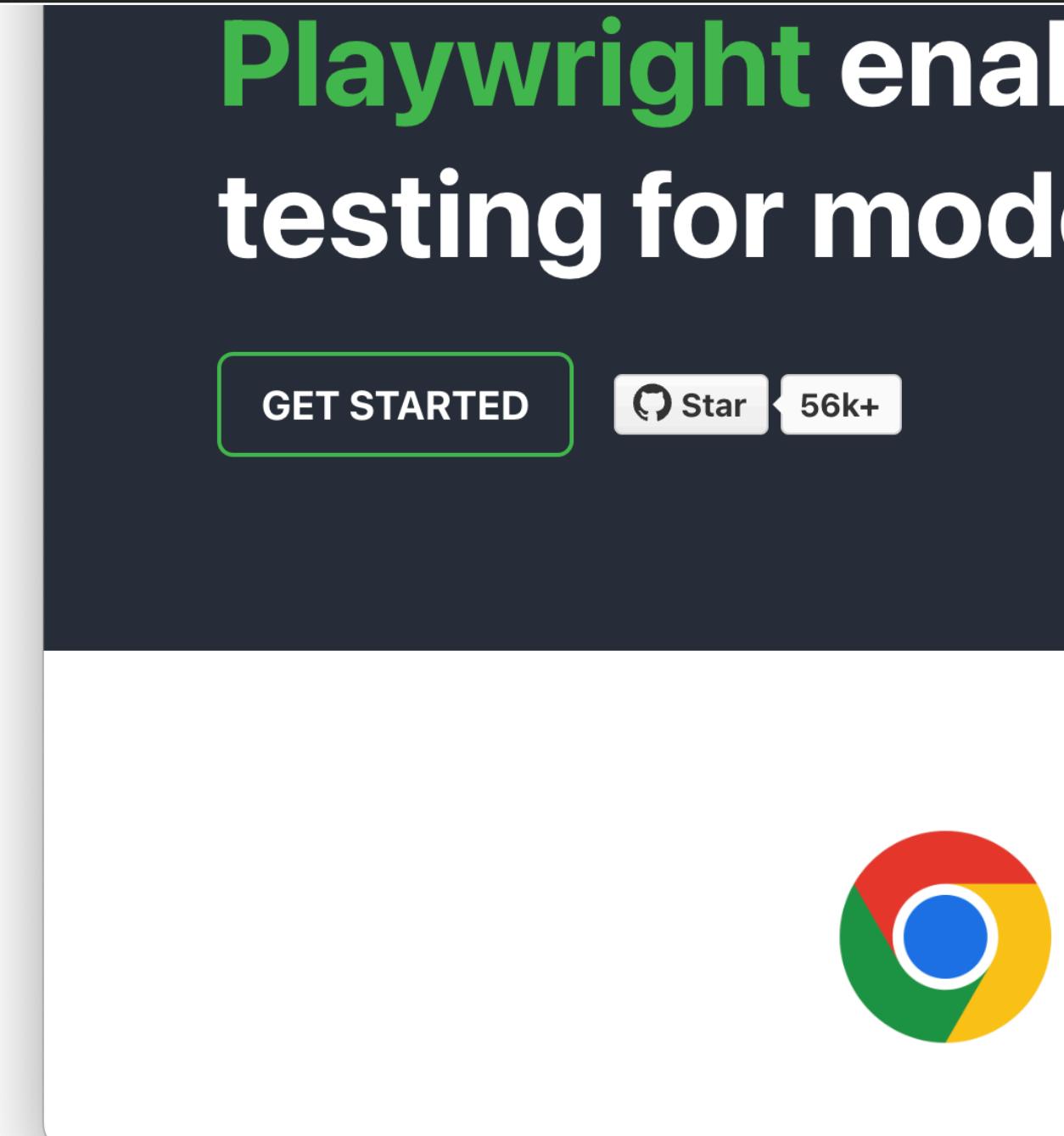
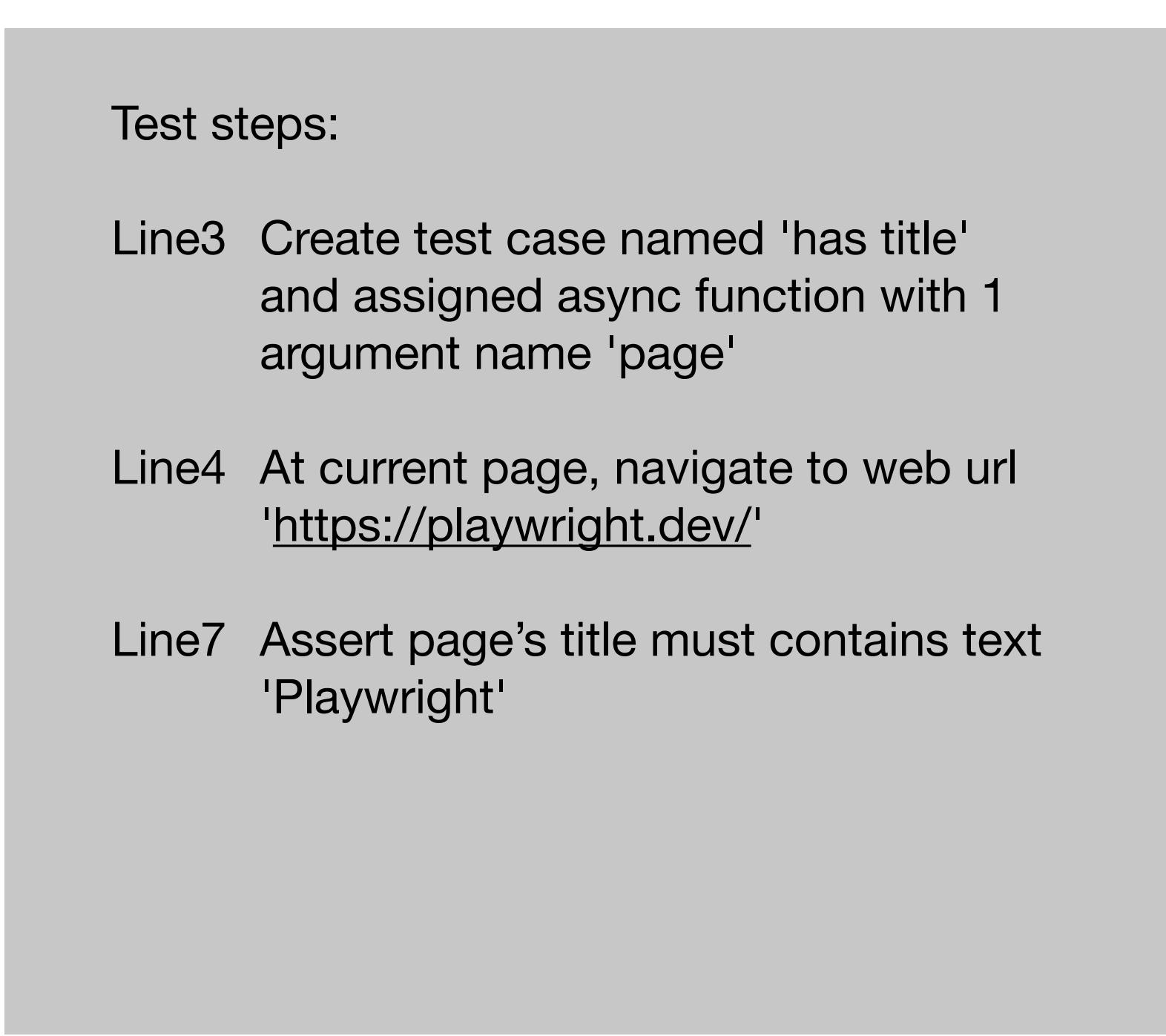
```
~ 3   ·async ·(      ) ·=> ·{  
~ 4   |  
~ 5   |  
~ 6   |  
~ 7   |  
~ 8   }
```

```
~ 3 test('has · title', ·async ·({ ·page ·}) ·=> ·{  
~ 4   ·· await ·page.goto('https://playwright.dev/'); - 3462ms  
~ 5   ·· // ·Expect ·a ·title ·"to ·contain" ·a ·substring.  
~ 6   ·· await ·expect(page).toHaveTitle('/Playwright/'); - 22ms  
~ 7   ·}  
~ 8   );
```

Example test cases: has title

```
2
3  test('has title', async ({ page }) => {
4    await page.goto('https://playwright.dev/'); - 3462ms
5
6    // Expect a title "to contain" a substring.
7    await expect(page).toHaveTitle(/Playwright/); - 22ms
8  });

```



Example test cases: get started link



The screenshot shows a VS Code interface with the following details:

- Explorer View:** Shows a project structure for "PLAYWRIGHT101". A tooltip is displayed over the "tests" folder, containing the text: "Second test case named 'get started link'".
- Code Editor:** The file "example.spec.ts" is open, showing TypeScript code for playwright tests. One specific test case is highlighted:

```
test('get-started-link', async ({ page }) => {
  await page.goto('https://playwright.dev/');
  // Click the get-started-link.
  await page.getByRole('link', { name: 'Get started' }).click();
  // Expects page to have a heading with the name of Installation.
  await expect(page.getByRole('heading', { name: 'Installation' })).toBeVisible();
});
```
- Test Results:** A sidebar on the right displays the results of a recent test run:
 - Test run at 11/21/2023, 9:50:...
 - has title
 - get started link
 - Test run at 11/21/2023, 9:50:48 PM
- Bottom Status Bar:** Shows file statistics (Ln 19, Col 1), code analysis (0△0), and other development tools.

Example test cases: get started link

The screenshot shows a browser window with the URL `playwright.dev/docs/intro`. The page title is "Installation". The left sidebar has sections like "Getting Started", "Installation" (which is selected), "Writing tests", "Generating tests", "Running and debugging tests", "Trace viewer", "CI GitHub Actions", "Getting started - VS Code", and "Release notes". The main content area has a heading "Introduction" and some text about Playwright's capabilities. A developer tools panel is open at the bottom, showing the "Elements" tab with the DOM tree and a "Styles" panel on the right. The test code in the console is:

```
10 test('get-started-link', async ({page}) => {
11   await page.goto('https://playwright.dev/'); - 3073ms
12
13   // Click the get-started link.
14   await page.getByRole('link', {name: 'Get started'}).click(); - 66ms
15
16   // Expects page to have a heading with the name of Installation.
17   await expect(page.getByRole('heading', {name: 'Installation'})).toBeVisible(); - 371ms
18 });
19
```

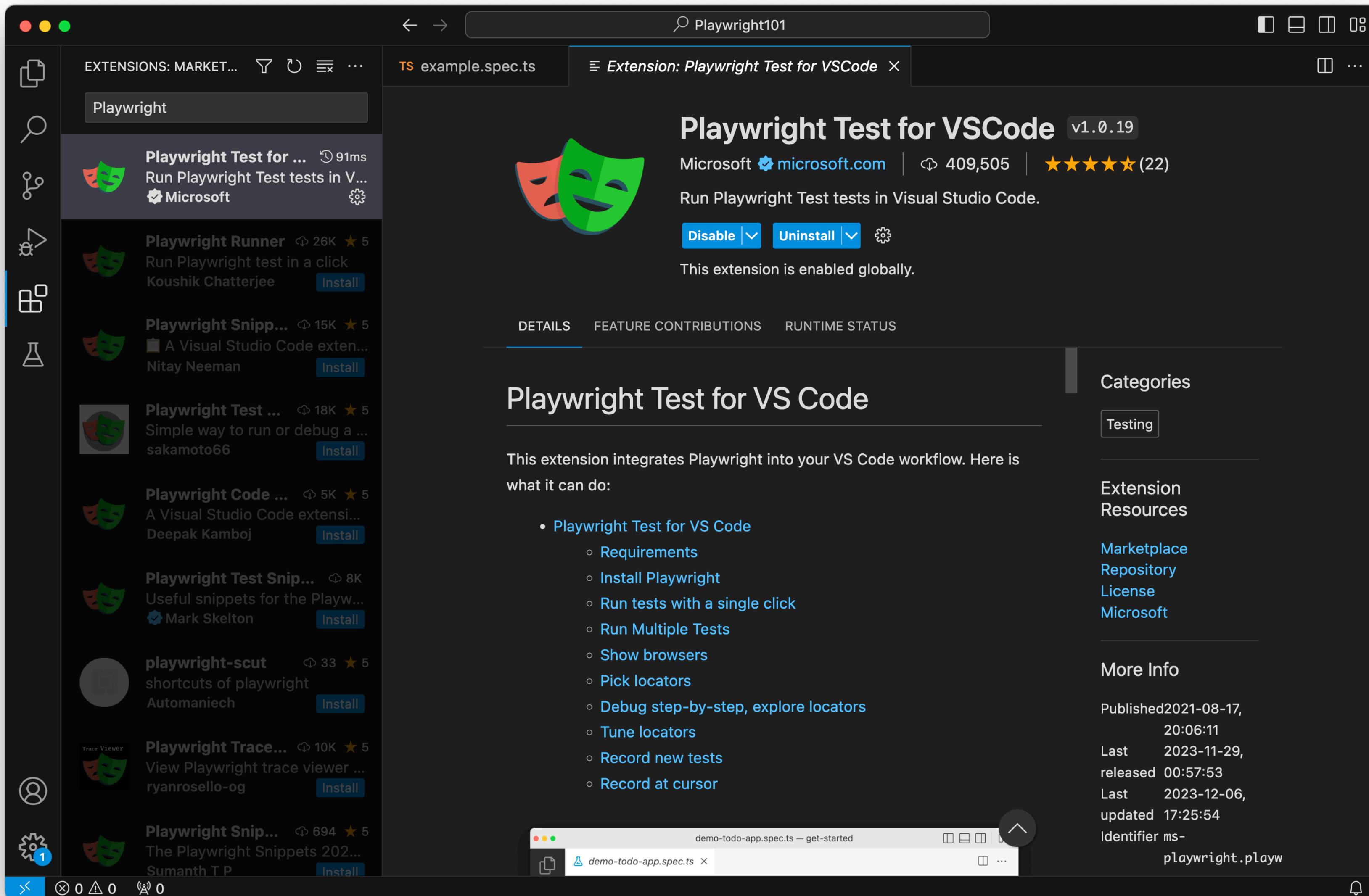
Test steps:

- Line10 Create test case named 'get started link' and assigned `async` function with 1 argument name 'page'
- Line14 At current page, navigate to web url `'https://playwright.dev/'`
- Line17 In expect's bracket, current page find and get page element with role 'heading' (e.g. h1, h2, ...) that contains text 'Installation'. Then it should be visible

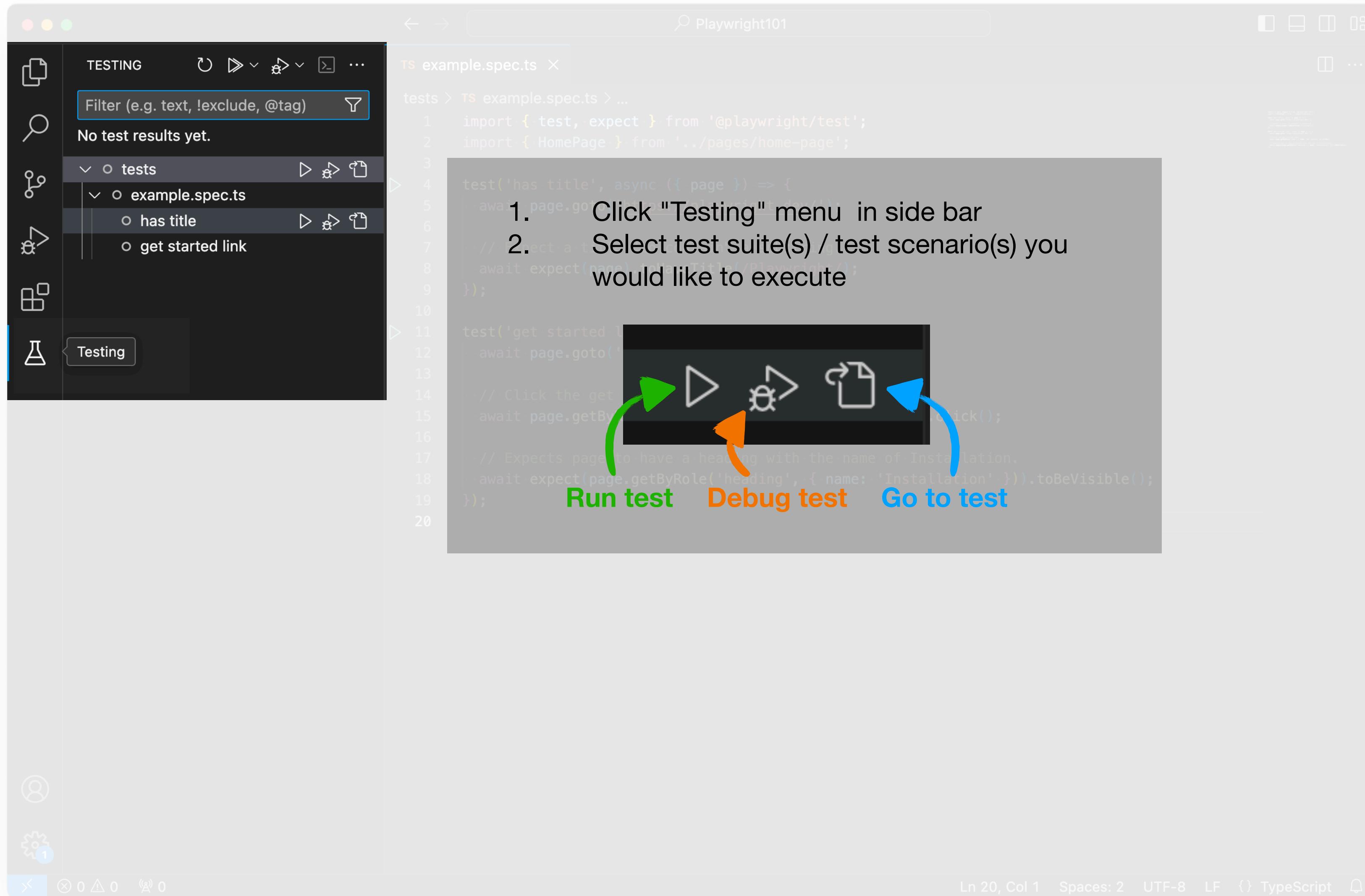
Test execution and Debug mode

- VSCode extension
- Execute / debug test(s) from VSCode
- Execute / debug test(s) from Command

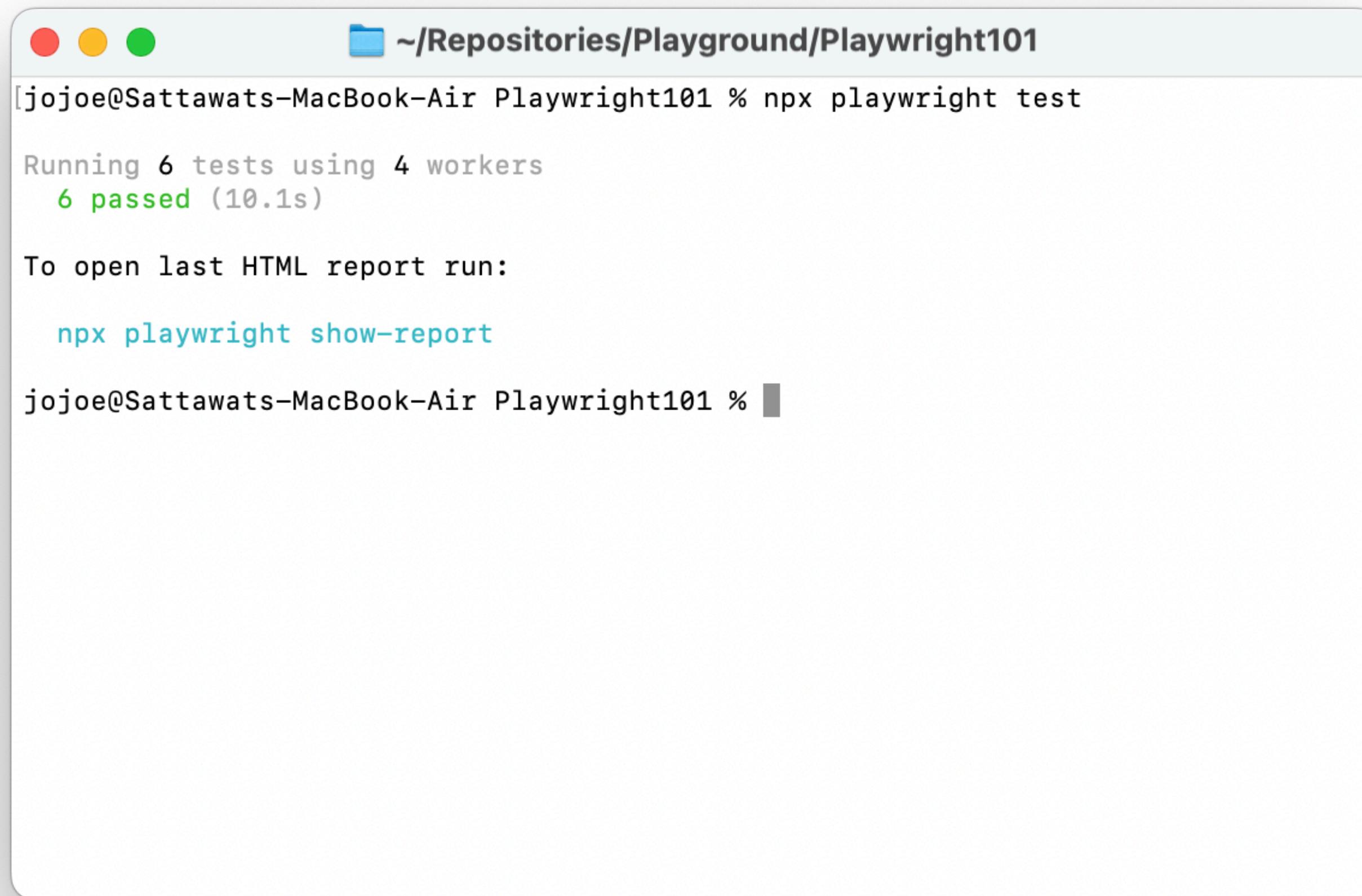
VSCode extension



Execute / debug tests from VSCode



Execute tests from command



A screenshot of a macOS terminal window. The title bar shows the path `~/Repositories/Playground/Playwright101`. The terminal output is as follows:

```
[jojoe@Sattawats-MacBook-Air Playwright101 % npx playwright test
Running 6 tests using 4 workers
6 passed (10.1s)

To open last HTML report run:
npx playwright show-report
jojoe@Sattawats-MacBook-Air Playwright101 %
```

Execute all test suite and scenarios:

```
npx playwright test
```

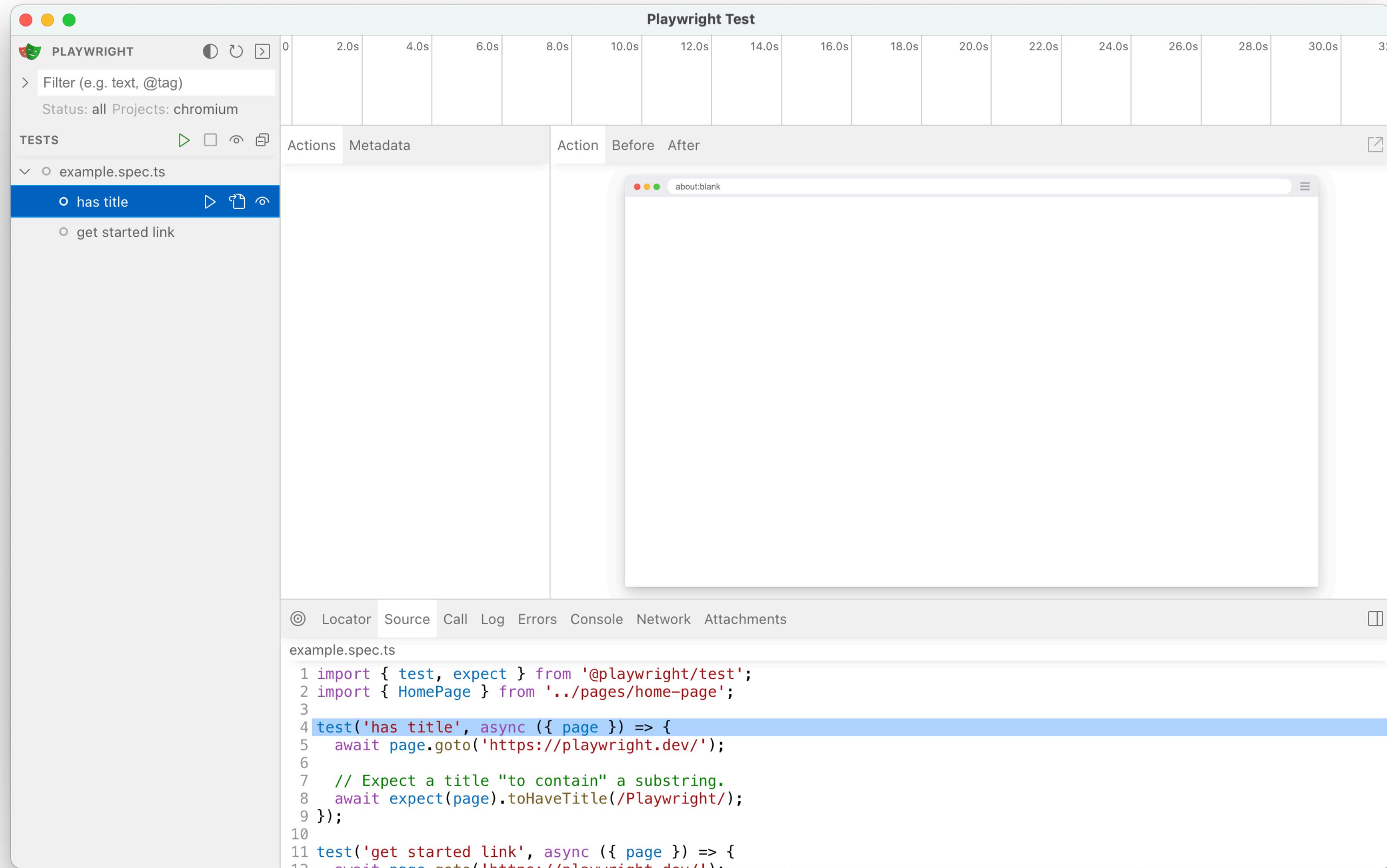
Execute single test suite:

```
npx playwright test tests/example.spec.ts
```

Execute single test scenarios:

```
npx playwright test -g "has title"
```

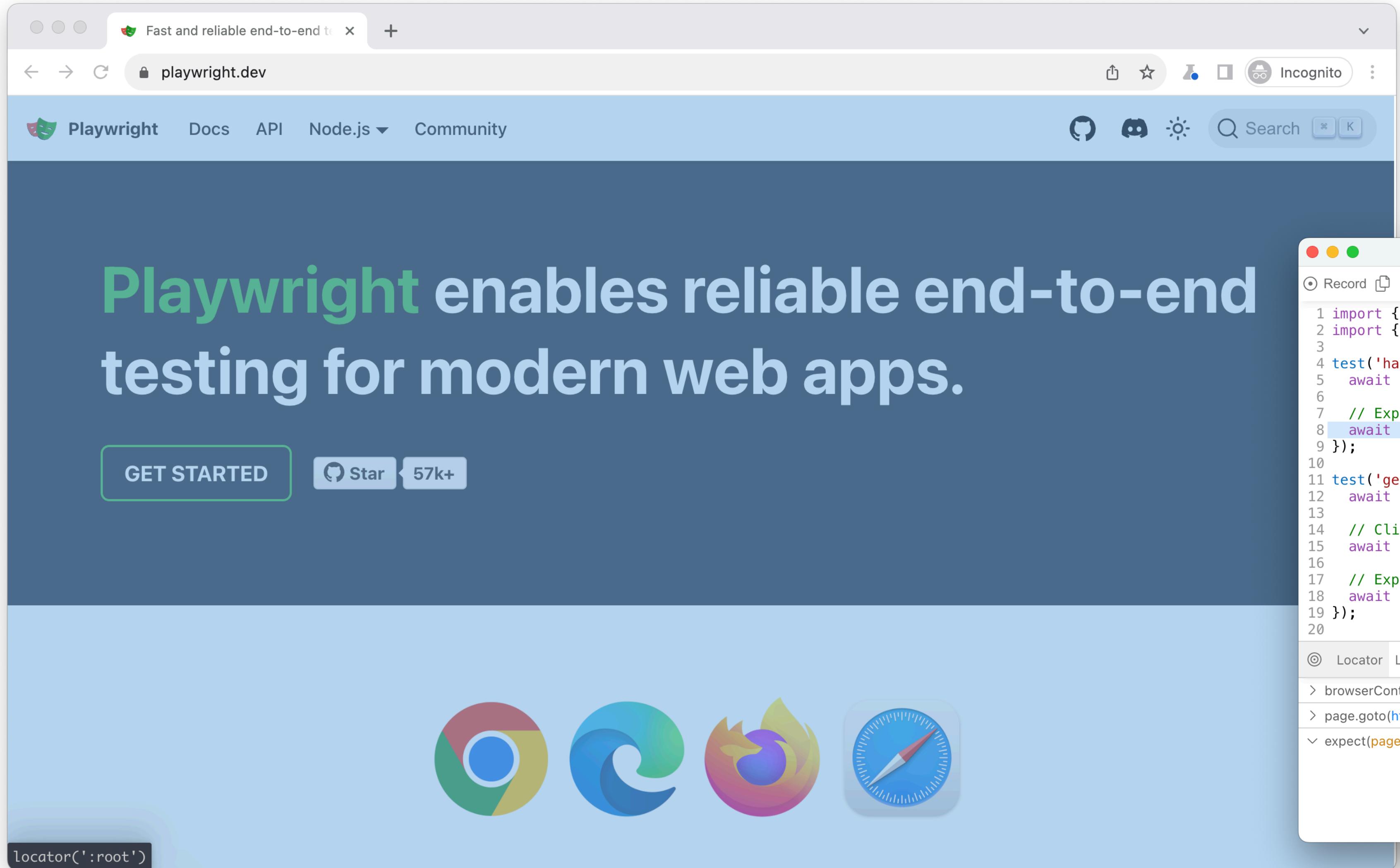
Execute tests in UI mode from command



Open inspector UI mode:

npx playwright test --ui

Execute tests in debug mode from command



Open inspector debug mode:

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

The screenshot shows the 'Playwright Inspector' tool window. It has a code editor on the left containing a TypeScript test file with syntax highlighting. The code is as follows:

```
1 import { test, expect } from '@playwright/test';
2 import { HomePage } from '../pages/home-page';
3
4 test('has title', async ({ page }) => {
5   await page.goto('https://playwright.dev/');
6
7   // Expect a title "to contain" a substring.
8   await expect(page).toHaveTitle(/Playwright/);
9 });
10
11 test('get started link', async ({ page }) => {
12   await page.goto('https://playwright.dev/');
13
14   // Click the get started link.
15   await page.getByRole('link', { name: 'Get started' }).click();
16
17   // Expects page to have a heading with the name of Installation.
18   await expect(page.getByRole('heading', { name: 'Installation' }));
19 });
20
```

Below the code editor is a log panel with the following entries:

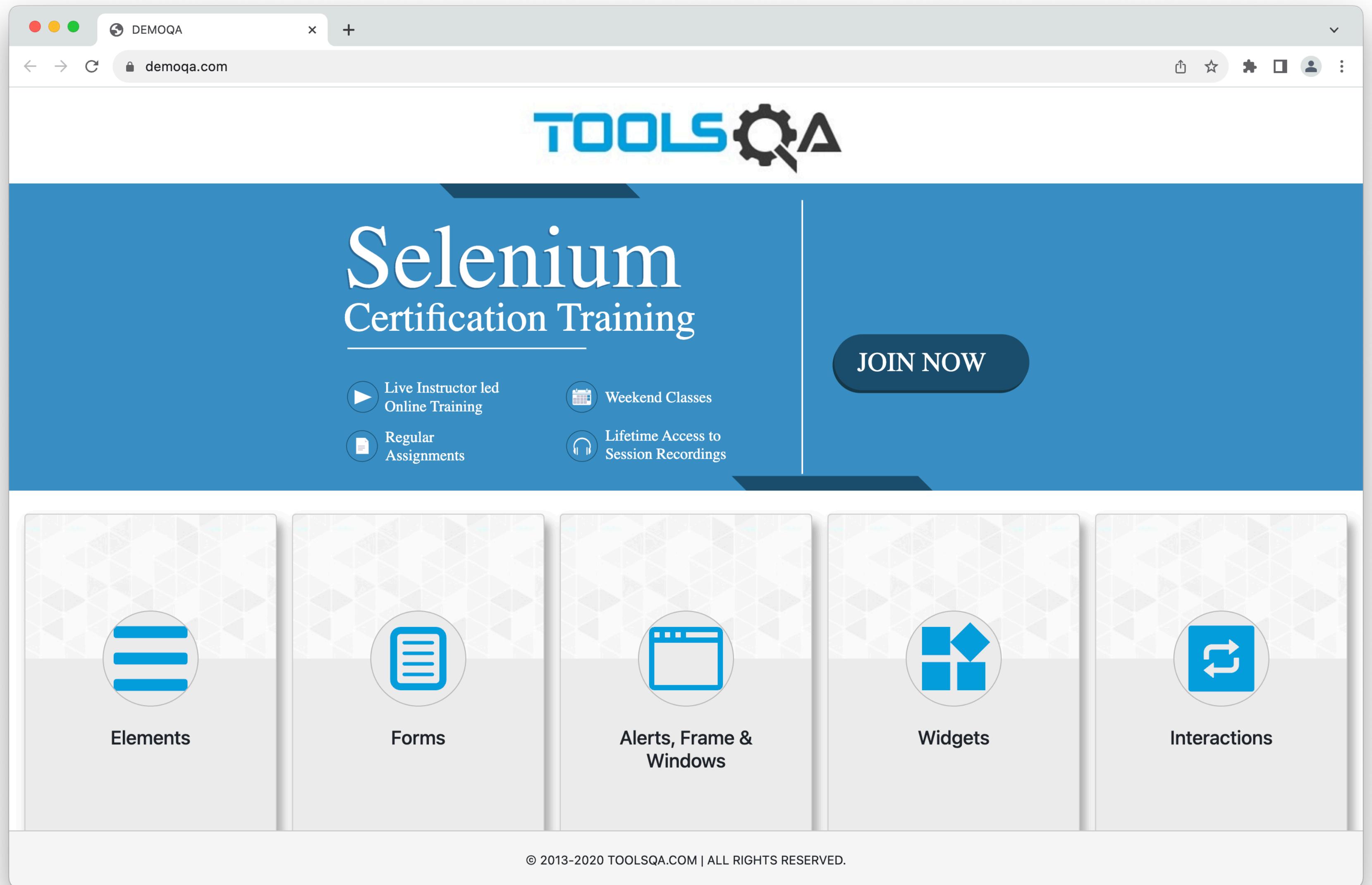
- > browserContext.newPage ✓ — 261ms
- > page.goto('https://playwright.dev/') ✓ — 2.4s
- ✓ expect(page.locator(':root')).toHaveTitle() II

Locators

- Chrome inspect and console
- HTML Tags
 - ID
 - CSS
 - XPath
 - data-testid
 - Chain locator

Tools QA : Our playground in this session :)

Open Chrome browser
Go to this url: demoqa.com



Chrome Inspect, Elements, or Console

The image shows two screenshots of a web browser window displaying the ToolsQA Selenium Certification Training website. The left screenshot shows the context menu open after right-clicking on the page, with the 'Inspect' option highlighted and circled in orange. The right screenshot shows the Chrome DevTools interface with the 'Elements' tab selected, displaying the page's HTML structure. A large orange arrow points from the 'Inspect' button in the context menu to the 'Elements' tab in the DevTools.

Open inspect and console steps:

1. Right click on page and select "Inspect" or press "F12"
2. In the inspect tab, select "Elements" or "Console"

Identify the unique locator

The screenshot shows a web browser window for 'DEMOQA' at 'demoqa.com'. The main content is a landing page for 'Selenium Certification Training'. On the left, there are four icons: 'Live Instructor led Online Training', 'Weekend Classes', 'Regular Assignments', and 'Lifetime Access to Session Recordings'. A large 'JOIN NOW' button is centered. On the right, there's a section titled 'Elements' with a sub-section 'Forms' featuring a document icon. The bottom of the page has a footer with the text '© 2013-2020 TOOLSQA.COM | ALL RIGHTS RESERVED.' To the right of the browser window is the open 'Elements' tab of the Chrome DevTools. It displays the HTML structure of the page, highlighting the element under inspection. The element being inspected is an 'h5' heading with the text 'Elements heading'.

Steps to identify locator of some elements:

1. Open Chrome inspect, and go to Elements tab
2. Use this tool, it is the easiest way
3. Point the arrow at the element you want, then click
4. In the Elements tab, it will bring you to the line of HTML of the element

However, how do you know that your selected element is unique?

Look closer

The screenshot shows a web browser window with the URL demoqa.com. The page content is a promotional banner for 'Selenium Certification Training'. On the left, there's a large blue box with the 'TOOLSQA' logo at the top. Below it, the text 'Selenium Certification Training' is displayed. To the right of this text is a 'JOIN NOW' button. Further down, there are four categories represented by cards: 'Elements' (highlighted with a purple dashed border), 'Forms', 'Tables', and 'JavaScript'. Each category has a small icon and a brief description. At the bottom of the page, there's a copyright notice: '© 2013-2020 TOOLSQA.COM | ALL RIGHTS RESERVED.' The browser's developer tools are open, with the 'Elements' tab selected. A tooltip from the developer tools points to the 'Elements' card, providing its dimensions: 'div.card.mt-4.top-card 356.8 x 400'. The DOM tree in the developer tools shows the structure of the page, including various divs, spans, and other HTML elements.

What if we want to click this locator, which one is unique?

- A. css = div.avatar.mx-auto.white
xpath = //div[@class='avatar mx-auto white']
- B. css = div.card-body
xpath = //div[@class='card-body']
- C. css = h5
xpath = //h5
- D. CSS can't point to the unique element 'Elements'
xpath = //h5[text()='Elements']

This example shows you two kinds of locators:
CSS and XPath locators

Let's try

Chrome console to identity selectors

The screenshot shows a browser window with the URL `demoqa.com`. The main content is the **ToolsQA Selenium Certification Training** page, featuring sections for **Elements**, **Forms**, **Alerts, Frame & Windows**, and **Widgets**. A specific element in the Widgets section is highlighted with a purple dashed border and labeled `div.avatar.mx-auto.white 100x100`.

The bottom portion of the screenshot shows the **Console** tab of the Chrome DevTools. The console output shows the result of running the CSS selector `$$("div.avatar.mx-auto.white")`, which returns a NodeList with 6 items. An orange circle highlights the first few lines of the console output, and an orange arrow points from this circle to the text "Seem it's not unique selector" located at the bottom left.

```
> $$("div.avatar.mx-auto.white")
< (6) [div.avatar.mx-auto.white, div.avatar.mx-auto.white, div.avatar.mx-auto.white, div.avatar.mx-auto.whi
  div.avatar.mx-auto.white, div.avatar.mx-auto.white] i
    ▷ 0: div.avatar.mx-auto.white
    ▷ 1: div.avatar.mx-auto.white
    ▷ 2: div.avatar.mx-auto.white
    ▷ 3: div.avatar.mx-auto.white
    ▷ 4: div.avatar.mx-auto.white
    ▷ 5: div.avatar.mx-auto.white
    length: 6
    ▷ [[Prototype]]: Array(0)
```

Locator assignment:

In case you would like to fine the CSS selector
use this sign:

```
$$("div.avatar.mx-auto.white")
```

**In case you would like to fine the XPath locator
use this sign:**

```
$x("//div[@class='avatar mx-auto white']")
```

Seem it's not unique selector

This page is the best practice for selector finding

Full Name
Jojoe

Email
name@example.com

Current Address
Current Address

Permanent Address

Name: Jojoe

Submit

ZeroStep
Build Playwright

© 2013-2020 TOOLSQA.COM | ALL RIGHTS RESERVED.

id = userForm

css = form#userForm

id = userName

css = input#userName

xpath = //input[@id='userName']

xpath = //input[
@placeholder='Full Name']

data-testid locator

data-testid is an attribute commonly used in web development to facilitate testing of user interfaces. It is not a native HTML attribute, but rather a **custom attribute that developers add to their HTML elements to make it easier to select and interact with those elements in automated tests.**

When writing tests, developers often need a way to uniquely identify elements on a page, especially when using testing libraries or frameworks like Jest, React Testing Library, or others. Adding data-testid attributes to elements allows developers to target specific elements in their tests, regardless of the underlying structure or styling of the page.

```
html
```

 Copy code

```
<button data-testid="submit-button">Submit</button>
```

Chain locator

The screenshot shows a browser window with a form and its corresponding DOM structure in the developer tools' Elements tab.

Form Fields:

- Full Name: Jojoe
- Email: name@example.com
- Current Address: (empty)
- Permanent Address: (empty)

Locators:

- id = userForm**: Points to the `<form id="userForm" class="mt-2 row">` element in the DOM.
- css = form#userForm**: Points to the `<form id="userForm" class="mt-2 row">` element in the DOM.
- id = userName**: Points to the `<input id="userN`ame" class="mr-sm-2 form-control"> element in the DOM.
- css = input#userName**: Points to the `<input id="userN`ame" class="mr-sm-2 form-control"> element in the DOM.
- xpath = //input[@id='userName']**: Points to the `<input id="userN`ame" class="mr-sm-2 form-control"> element in the DOM.
- xpath = //input[@placeholder='Full Name']**: Points to the `<input id="userN`ame" class="mr-sm-2 form-control"> element in the DOM.

DOM Structure:

```
<div id="aygound-body" class="aygound-body">
  <div class="row">
    <div class="col-12 mt-4 col-md-6">
      <div class="text-field-container">
        <form id="userForm" class="mt-2 row">
          <div id="userN
          ame-wrapper" class="col-md-3 col-sm-12">
            <label class="form-label" id="userN
            ame-label">Full Name</label>
            <input autocomplete="off" placeholder="Full Name" type="text" id="userN
            ame" class="mr-sm-2 form-control">
          </div>
          <div id="userEmail-wrapper" class="mt-2 row">
            <div class="col-md-9 col-sm-12">
              <input type="text" id="userEmail" class="form-control">
            </div>
          </div>
          <div id="currentAddress-wrapper" class="mt-2 row">
            <div class="col-md-9 col-sm-12">
              <input type="text" id="currentAddress" class="form-control">
            </div>
          </div>
          <div id="permanentAddress-wrapper" class="mt-2 row">
            <div class="col-md-9 col-sm-12">
              <input type="text" id="permanentAddress" class="form-control">
            </div>
          </div>
          <div class="mt-2 justify-content-end row">
            <div class="text-right col-md-2 col-sm-12">
              <button id="submit" type="button" class="btn btn-primary">Submit</button>
            </div>
          </div>
          <div id="output" class="mt-4 row">
            <div class="border col-md-12 col-sm-12">
              <p id="name" class="mb-1">
                "Name:<br/>"<br/>"Jojoe"
              </p>
            </div>
          </div>
        </form>
      </div>
    </div>
  </div>
</div>
```

It is not quite hard to connect more than one selector together

A chain locator is an approach to create a new unique locator by combining different kinds of selectors.

For example, if you cannot find a unique selector:

css = userForm >> xpath = //input[@id='userName']

The sign **>>** is a chain connector that links between:

- 1st locator: css = userForm
- 2nd locator: xpath = //input[@id='userName']

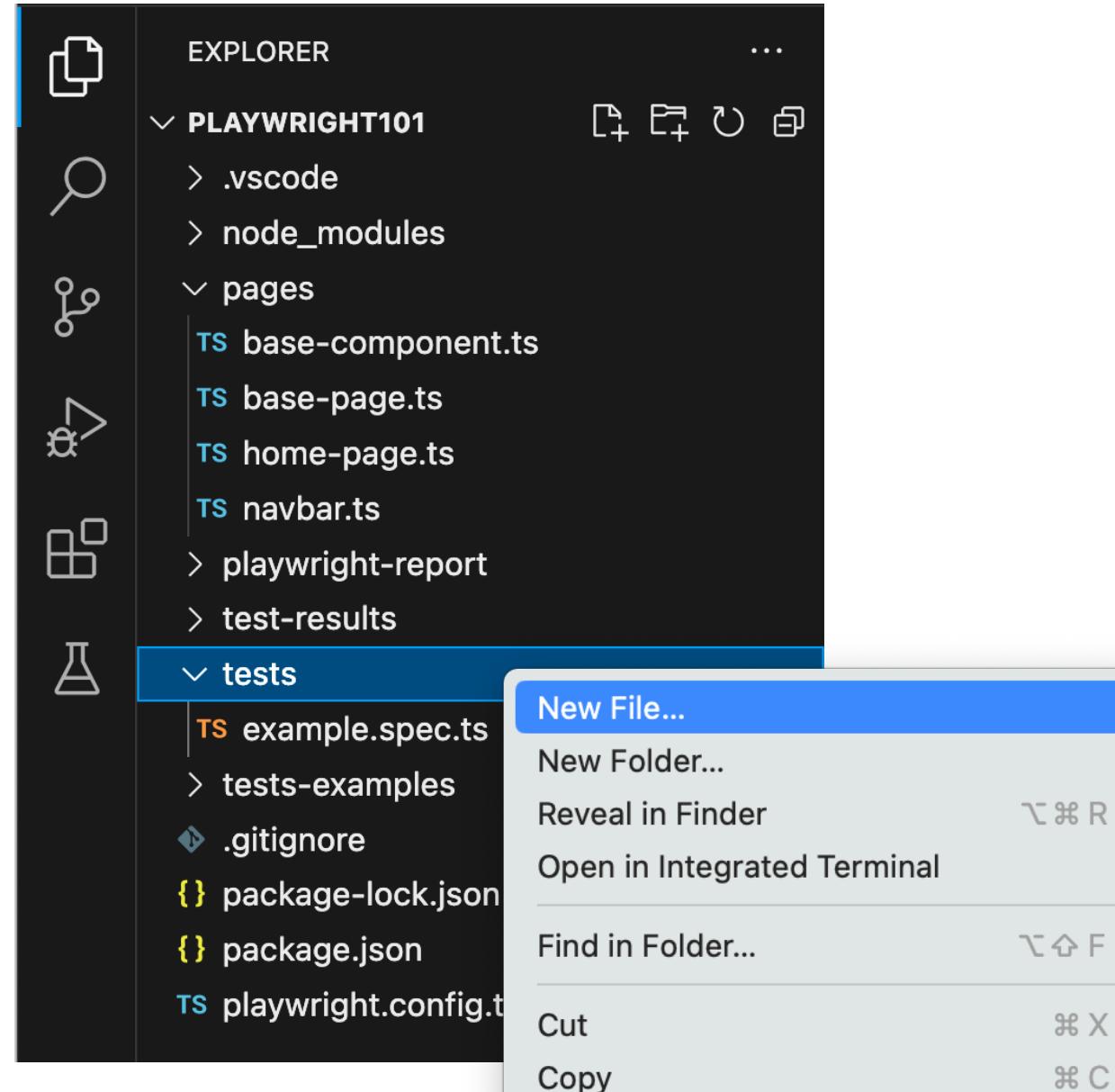
The true form of above chain locator is

```
page.locator("css=userForm")
  .locator("xpath='//input[@id='userName']'")
```

Create first test

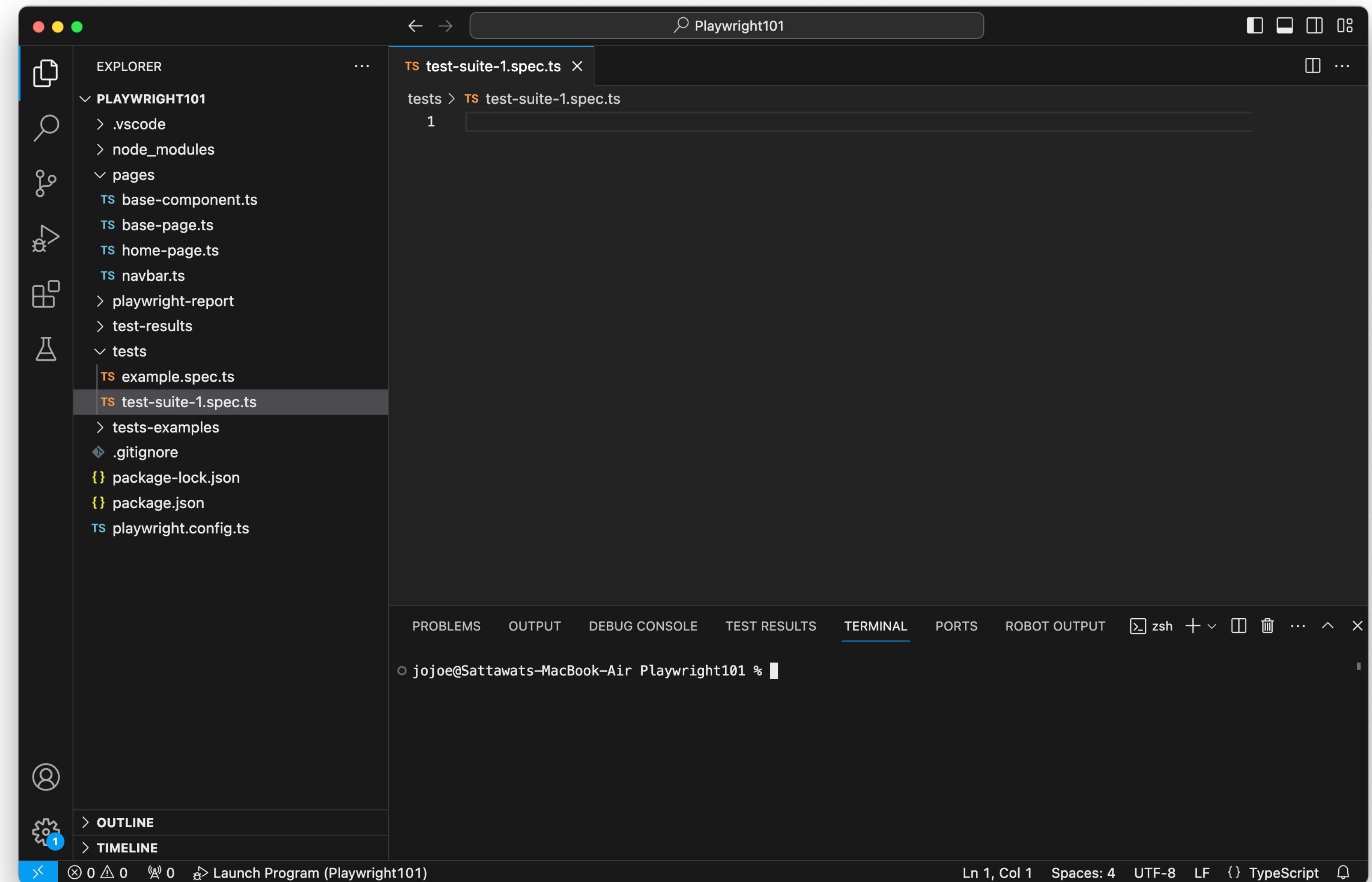
Suite
and
Scenario

Create first test suite file



Create new test file (Test suite) steps:

1. Right click at folder "tests"
2. Select "New File..."
3. Naming your test file, this file must ending with .spec.ts



Create first test scenario

The screenshot shows the Visual Studio Code (VS Code) interface with a dark theme. The title bar displays "Playwright101". The left sidebar has icons for TESTING, SEARCH, TESTS, and TERMINAL. The TESTING section shows a "tests" folder with "example.spec.ts" and "test-suite-1.spec.ts". The "example.spec.ts" file is open in the main editor, showing the following code:

```
1 import { test } from '@playwright/test';
2
3 test("Your first test scenario", async () => {
4
5});
```

The terminal at the bottom shows the command "jojoe@Sattawats-MacBook-Air Playwright101 %". The status bar at the bottom indicates "Ln 5, Col 4" and "Spaces: 4".

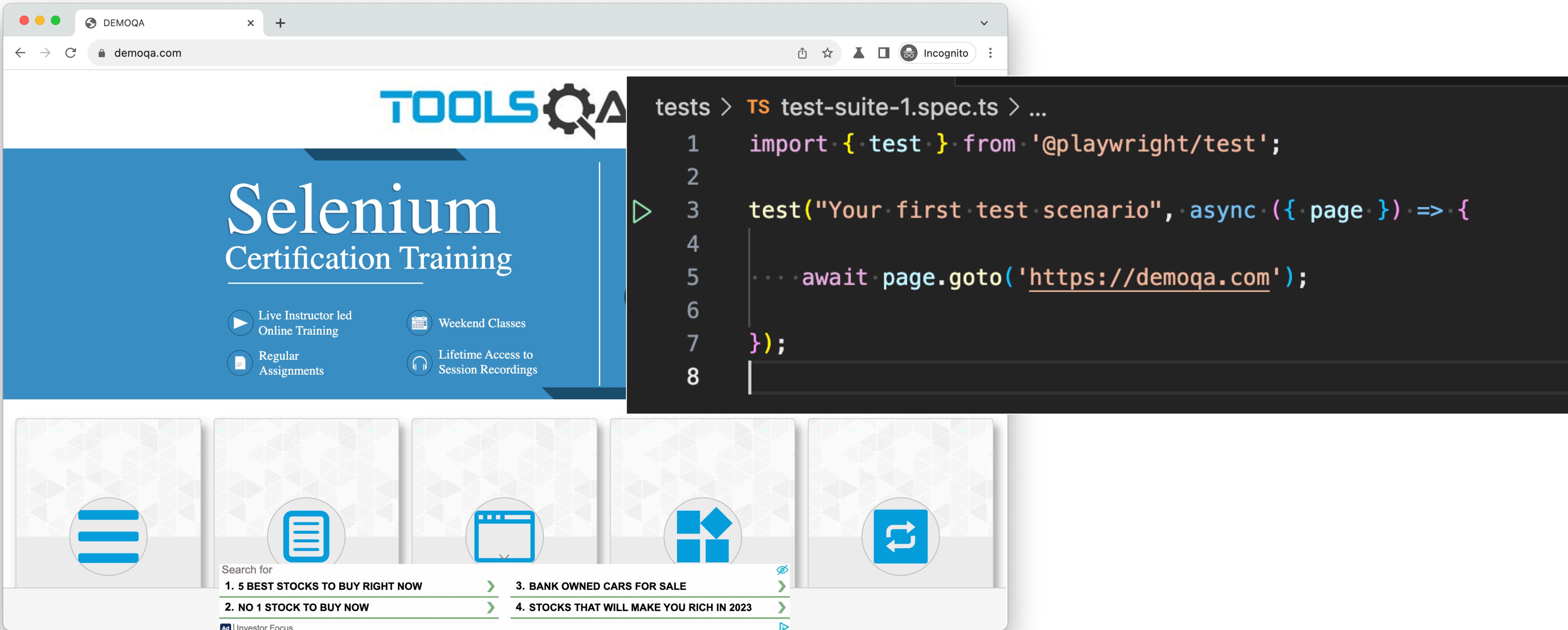
Actions

- Page navigation
- Mouse click
- Text input
- Checkbox & Radio button
- Dropdown selection
- Keys and shortcuts
- Upload / download file
- Drag and Drop

Page navigation

Do you remember our playground?

In this part, we will let's our test scenario page open that website.



Mouse click

>> Elements > Buttons

Identify the locator of the element we need and click on it!

The image shows a browser window with two main panes. The left pane displays a landing page for 'Automation Training' from demoqa.com. It features a blue header with icons for 'Live Instructor led Online Training', 'Weekend Classes', 'Regular Assignments', and 'Lifetime Access to Session Recordings'. Below the header are three cards: 'Elements' (selected), 'Forms', and 'Alerts, Frame & Windows'. The 'Elements' card has a blue border and its locator is highlighted in a tooltip: `locator('xpath=//h5[text()='Elements']')`. The right pane shows a code editor with a dark theme, displaying a TypeScript test script. The script imports the playwright/test module and defines a test function for a scenario named 'Your first test scenario'. Inside the test function, the code uses await.page.goto to navigate to the demoqa URL and then await.page.locator to click on the 'Elements' button. The code editor highlights the URL and the locator selector.

```
tests > TS test-suite-1.spec.ts > ...
1 import { test } from '@playwright/test';
2
3 test("Your first test scenario", async ({ page }) => {
4
5   await page.goto('https://demoqa.com');
6
7   await page.locator("xpath=//h5[text()='Elements']").click();
8
9 });
10
```

A screenshot of a web browser showing the ToolsQA website at demoqa.com/buttons. The page title is 'Buttons'. On the left, a sidebar menu titled 'Elements' lists various UI components: Text Box, Check Box, Radio Button, Web Tables, Buttons, Links, and Broken Links - Images. The 'Buttons' item is highlighted. The main content area displays three blue buttons: 'Double Click Me', 'Right Click Me', and 'Click Me'. An orange circle highlights the 'Click Me' button, which is connected by an orange arrow to the text 'Mouse click assignment:' in the adjacent block.

Well done
Your test go through
that website as well

Mouse click assignment:
Go to this page then, find and click on their buttons.

Text input

>> Elements > Text Box

Fill in the text boxes, then click the 'Submit' button.

Try both valid and invalid email formats to observe any errors.

We will address error handling and assertions in the next session.

The screenshot shows a browser window with the URL `demoqa.com/text-box`. On the left, there is a sidebar with a navigation menu under 'Elements' containing links for Text Box, Check Box, Radio Button, Web Tables, Buttons, Links, and Broken Links - Images. The main content area is titled 'Text Box'. It contains four text input fields: 'Full Name' (placeholder 'Full Name'), 'Email' (placeholder 'name@example.com'), 'Current Address' (placeholder 'Current Address'), and 'Permanent Address' (placeholder 'Permanent Address'). At the bottom is a blue 'Submit' button. On the right, a code editor displays a TypeScript test script:

```
tests > TS test-suite-1.spec.ts > ⚡ test("Your first test scenario") callback
1 import { test } from '@playwright/test';
2
3 test("Your first test scenario", async ({ page }) => {
4
5     await page.goto('https://demoqa.com');
6     await page.locator("xpath=//h5[text()='Elements']").click();
7     await page.locator("xpath=//span[@class='text' and text()='Text Box']").click();
8
9     await page.locator("id=userName").fill("Jojoe"); // Input full name
10    await page.locator("...").fill("..."); // Input email
11    await page.locator("...").fill("..."); // Input current address
12    await page.locator("...").fill("..."); // Input permanent address
13
14    await page.locator("id=submit").click(); // Click submit button
15
16 });
17
```

Checkbox

>> Elements > Check Box

Check! Check! Check! Uncheck! Uncheck!

The screenshot shows a browser window with the URL `demoqa.com/checkbox`. On the left, there is a sidebar menu with the following items:

- Elements
- Text Box
- Check Box
- Radio Button
- Web Tables
- Buttons
- Links
- Broken Links - Images

The "Check Box" item is selected, indicated by a blue border around its icon. The main content area displays a file tree under the heading "Check Box". The tree structure is as follows:

- Home
 - Desktop
 - Documents
 - Downloads
 - Word File.doc
 - Excel File.doc

On the right side of the browser window, there is a code editor showing a TypeScript test script:

```
tests > TS test-suite-1.spec.ts > ...
1 import { test } from '@playwright/test';
2
3 test("Your first test scenario", async ({ page }) => {
4
5     await page.goto('https://demoqa.com');
6     await page.locator("xpath=//h5[text()='Elements']").click();
7     await page.locator("xpath=//span[@class='text' and text()='Check Box']").click();
8
9     // Expand check box tree
10    await page.locator("xpath=//div[@id='tree-node']/ol/li/span/button").click();
11    // Check check box name "Home"
12    await page.locator("xpath=//label[@for='tree-node-home']/span[@class='rct-checkbox']").check();
13
14    // Uncheck check box name "Document"
15    await page.locator("xpath=//label[@for='tree-node-documents']/span[@class='rct-checkbox']").uncheck();
16
17 });
18
```

At the bottom of the browser window, there is a footer bar with the text "© 2013-2020 TOOLSQA.COM | ALL RIGHTS RESERVED."

DEMOQA

demoqa.com/checkbox

TOOLS QA

Check Box

Elements

Text Box

Check Box

Radio Button

Web Tables

Buttons

Links

Broken Links - Images

Check Box

- Home
- Desktop
 - Notes **Check this box!**
 - Commands
- > Documents
- Downloads
 - Word File.doc **Check this box!**
 - Excel File.doc **Check this box!**

You have selected : notes downloads wordFile excelFile

YOUR TURN!
Seem this page is
the best practice

Check box assignment 1:
Go to this page then, find and check their boxes.

DEMOQA

demoqa.com/checkbox

TOOLS QA

Check Box

Elements

Text Box

Check Box

Radio Button

Web Tables

Buttons

Links

Broken Links - Images

Check Box

- Home
 - Desktop
 - Notes
 - Commands
 - > Documents
- > Downloads
 - Word File.doc
 - Excel File.doc

You have selected : notes downloads wordFile excelFile

One more, one more :D

Try this function with some check boxes

Check box assignment 2:

Try this `.isChecked()` function to their box types

- Checked box
- Half checked box
- Uncheck box

Add this code next to the comment
// Uncheck check box name "Document"

```
17 | ... // Get "Home" check box status and log it out
18 | ... const boxStatus = await page.locator("xpath=/label[@for='tree-node-home']//span[@class='rct-checkbox']").isChecked();
19 | ... console.log(boxStatus);
```

Let's see, What the final result is

Playwright101

TS test-suite-1.spec.ts X

tests > TS test-suite-1.spec.ts > ⚡ test("Your first test scenario") callback

```

3  test("Your first test scenario", async ({ page }) => {
4
5    await page.goto('https://demoqa.com'); - 6585ms
6    await page.locator("xpath=//h5[text()='Elements']").click(); - 194ms
7    await page.locator("xpath=//span[@class='text' and text()='Check Box']").click(); - 469ms
8
9    // Expand check box tree
10   await page.locator("xpath=//div[@id='tree-node']/ol/li/span/button").click(); - 50ms
11   // Check check box name "Home"
12   await page.locator("xpath=//label[@for='tree-node-home']//span[@class='rct-checkbox']").check(); - 45ms
13
14   // Uncheck check box name "Document"
15   await page.locator("xpath=//label[@for='tree-node-documents']//span[@class='rct-checkbox']").uncheck(); - 48ms
16
17   // Get "Home" check box status and log it out
18   const homeBoxStatus = await page.locator("xpath=//label[@for='tree-node-home']//span[@class='rct-checkbox']").isChecked(); - 3ms
19   console.log("Home check box status: ", homeBoxStatus);
20
21   // Expand check box "Desktop" tree
22   await page.locator("xpath=//label[@for='tree-node-desktop']//button").click(); - 29ms
23   await page.locator("xpath=//label[@for='tree-node-notes']//span[@class='rct-checkbox']").check(); - 5ms
24   await page.locator("xpath=//label[@for='tree-node-commands']//span[@class='rct-checkbox']").uncheck(); - 25ms
25
26   // Expand check box "Downloads" tree
27   await page.locator("xpath=//label[@for='tree-node-downloads']//button").click(); - 35ms
28   await page.locator("xpath=//label[@for='tree-node-wordFile']//span[@class='rct-checkbox']").check(); - 4ms
29   await page.locator("xpath=//label[@for='tree-node-excelFile']//span[@class='rct-checkbox']").check(); - 3ms
30
31   const desktopBoxStatus = await page.locator("xpath=//label[@for='tree-node-notes']//span[@class='rct-checkbox']").isChecked(); - 2ms
32   console.log("Desktop check box status: ", desktopBoxStatus);
33
34   const commandsBoxStatus = await page.locator("xpath=//label[@for='tree-node-commands']//span[@class='rct-checkbox']").isChecked(); - 6ms
35   console.log("Commands check box status: ", commandsBoxStatus);
36
37 });

```

PROBLEMS OUTPUT DEBUG CONSOLE TEST RESULTS TERMINAL PORTS ROBOT OUTPUT

-project=chromium --repeat-each 1 --retries 0 --timeout 0 --workers 1

Running 1 test using 1 worker

Home check box status: false

Desktop check box status: true

Commands check box status: false

1 passed (8.3s)

Ln 37, Col 1 S

Radio button

>> Elements >> Radio Button

You can select only one of it.

DEMOQA

demoqa.com/radio-button

TOOL

Radio B

Do you like the site?

Yes Impressive No

You have selected Impressive

Elements

Text Box

Check Box

Radio Button

Web Tables

Buttons

Links

Broken Links - Images

© 2013-2020 TOOLSQA.COM

```
tests > TS test-suite-1.spec.ts > ...
1 import { test } from '@playwright/test';
2
3 test("Your first test scenario", async ({ page }) => {
4
5     await page.goto('https://demoqa.com');
6     await page.locator("xpath=//h5[text()='Elements']").click();
7     await page.locator("xpath=//span[@class='text' and text()='Radio Button']").click();
8
9     // Collect locator to one variable make it easy to reuse
10    const impressiveRadioLocator = page.locator("xpath=//input[@id='impressiveRadio']");
11
12    // Check status of impressive radio button
13    const impressiveRadioStatus = await impressiveRadioLocator.isDisabled();
14    console.log("Is impressive radio disable:", impressiveRadioStatus);
15
16    // Chain locator out 1 step then click, due to have something wrong with radio button
17    await impressiveRadioLocator.locator("./").click();
18
19    const noRadioLocator = page.locator("xpath=//input[@id='noRadio']");
20    const noRadioStatus = await noRadioLocator.isDisabled();
21    console.log("Is no radio disable:", noRadioStatus);
22
23 });
24
```

PROBLEMS OUTPUT TEST RESULTS DEBUG CONSOLE TERMINAL PORTS ... Filter (e.g. text, !exclude)

```
/opt/homebrew/bin/node ./node_modules/@playwright/test/cli.js test -c playwright.config.ts /Users/jojoel/ht101/tests/test-suite-1\\.spec\\.ts:3 --headed --project=chromium --repeat-each 1 --retries 0 --timeout 0
```

Running 1 test using 1 worker

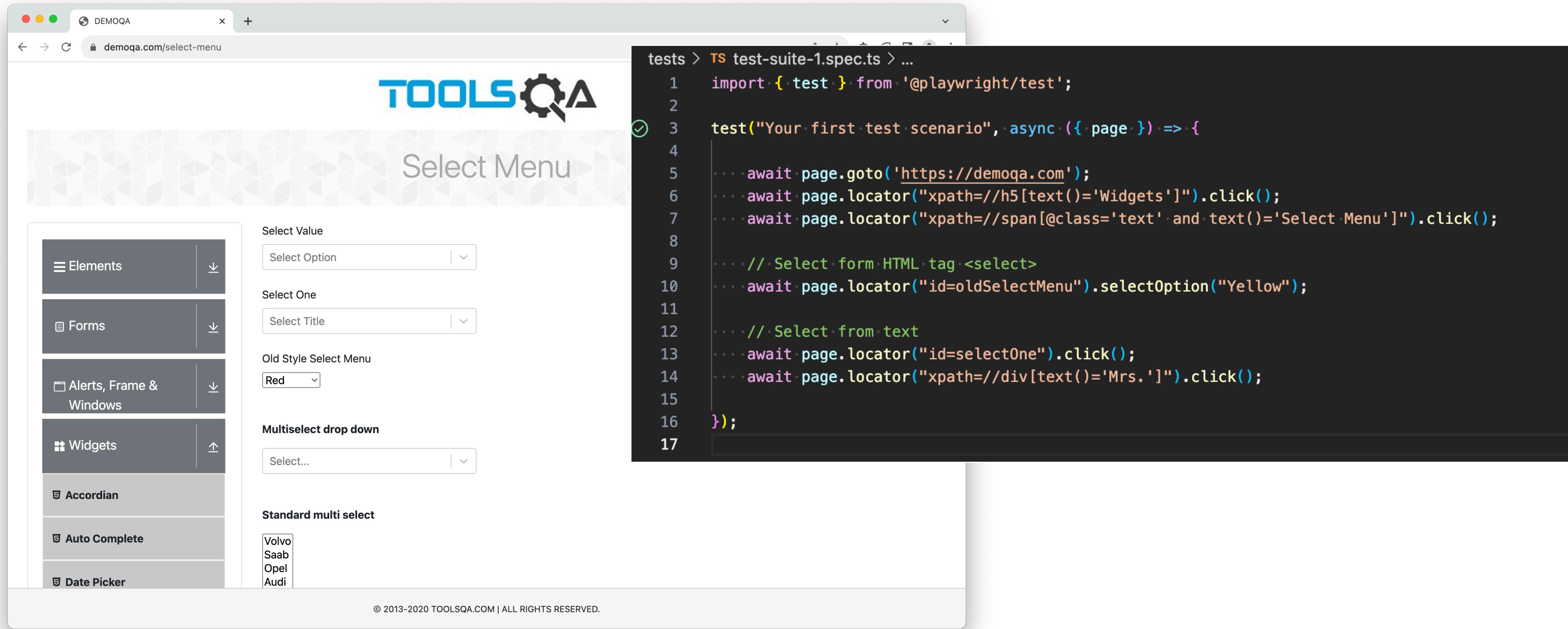
Is impressive radio disable: false

Is no radio disable: true

1 passed (812ms)

Dropdown selection

>> Widgets > Select Menu



The image shows a screenshot of a web browser displaying the ToolsQA Select Menu page. The page has a sidebar with various UI component categories like Elements, Forms, Alerts, Frame & Windows, Widgets, Accordion, Auto Complete, and Date Picker. The main content area is titled "Select Menu" and contains several dropdown examples: "Select Value" (a standard dropdown menu), "Select One" (a dropdown menu with a placeholder "Select Title"), "Old Style Select Menu" (a dropdown menu with an option "Red" selected), and "Multiselect drop down" (a dropdown menu with a placeholder "Select..."). To the right of the browser window is a terminal window showing a TypeScript code snippet for a playwright test. The code demonstrates how to navigate to the demoqa website, click on the "Widgets" menu item, and then click on the "Select Menu" item. It then performs two selects: one from a form HTML tag's select element and another from a text input field.

```
tests > TS test-suite-1.spec.ts > ...
1 import { test } from '@playwright/test';
2
3 test("Your first test scenario", async ({ page }) => {
4
5     await page.goto('https://demoqa.com');
6     await page.locator("xpath=//h5[text()='Widgets']").click();
7     await page.locator("xpath=//span[@class='text' and text()='Select Menu']").click();
8
9     // Select form HTML tag <select>
10    await page.locator("id=oldSelectMenu").selectOption("Yellow");
11
12    // Select from text
13    await page.locator("id=selectOne").click();
14    await page.locator("xpath=//div[text()='Mrs.']").click();
15
16 });
17
```

The screenshot shows a web browser window for 'DEMOQA' at 'demoqa.com/select-menu'. The page title is 'Select Menu'. On the left, there's a sidebar with navigation items: 'Elements', 'Forms', 'Alerts, Frame & Windows', 'Widgets', 'Accordion', 'Auto Complete', and 'Date Picker'. The main content area contains several examples of select menus:

- Select Value**: A single-select dropdown labeled 'Select Option'.
- Select One**: A single-select dropdown labeled 'Select Title'.
- Old Style Select Menu**: A dropdown menu showing 'Red'.
- Multiselect drop down**: A multiselect dropdown labeled 'Select...' containing the options 'Volvo', 'Saab', 'Opel', and 'Audi'.
- Standard multi select**: A standard multiselect dropdown labeled 'Select...' containing the same four options.

A large orange oval highlights the 'Multiselect drop down' section, and an arrow points from it to the text 'Drop down assignment 1: Select 2 or more options from "Multiselect drop down"'.

© 2013-2020 TOOLSQA.COM | ALL RIGHTS RESERVED.

Challenge

Select options from that dropdown

Drop down assignment 1:
Select 2 or more options from
"Multiselect drop down"

Remark: Locator of this drop down quite hard to identify, but you can do it :)

Keys and shortcuts

>> Google

Its how to press your keyboard in Playwright.

The image shows a split-screen interface. On the left is a screenshot of a web browser displaying the Google homepage. On the right is a code editor showing a TypeScript file named `test-suite-1.spec.ts`. The code in the editor is as follows:

```
tests > TS test-suite-1.spec.ts > ...
1 import { test } from '@playwright/test';
2
3 test("Your first test scenario", async ({ page }) => {
4
5     await page.goto("https://google.com");
6
7     await page.locator("xpath=//textarea[@role='combobox']").press("H");
8     await page.locator("xpath=//textarea[@role='combobox']").press("e");
9     await page.locator("xpath=//textarea[@role='combobox']").press("l");
10    await page.locator("xpath=//textarea[@role='combobox']").press("l");
11    await page.locator("xpath=//textarea[@role='combobox']").press("o");
12
13    await page.locator("xpath=//textarea[@role='combobox']").press("!");
14    await page.locator("xpath=//textarea[@role='combobox']").press("Backspace");
15
16    await page.locator("xpath=//textarea[@role='combobox']").press("Enter");
17
18 });
19
```

The browser window shows the Google search bar with "Hello" typed in, and the code editor highlights the line `await page.goto("https://google.com");`.

Upload file

>> Elements > Upload and Download

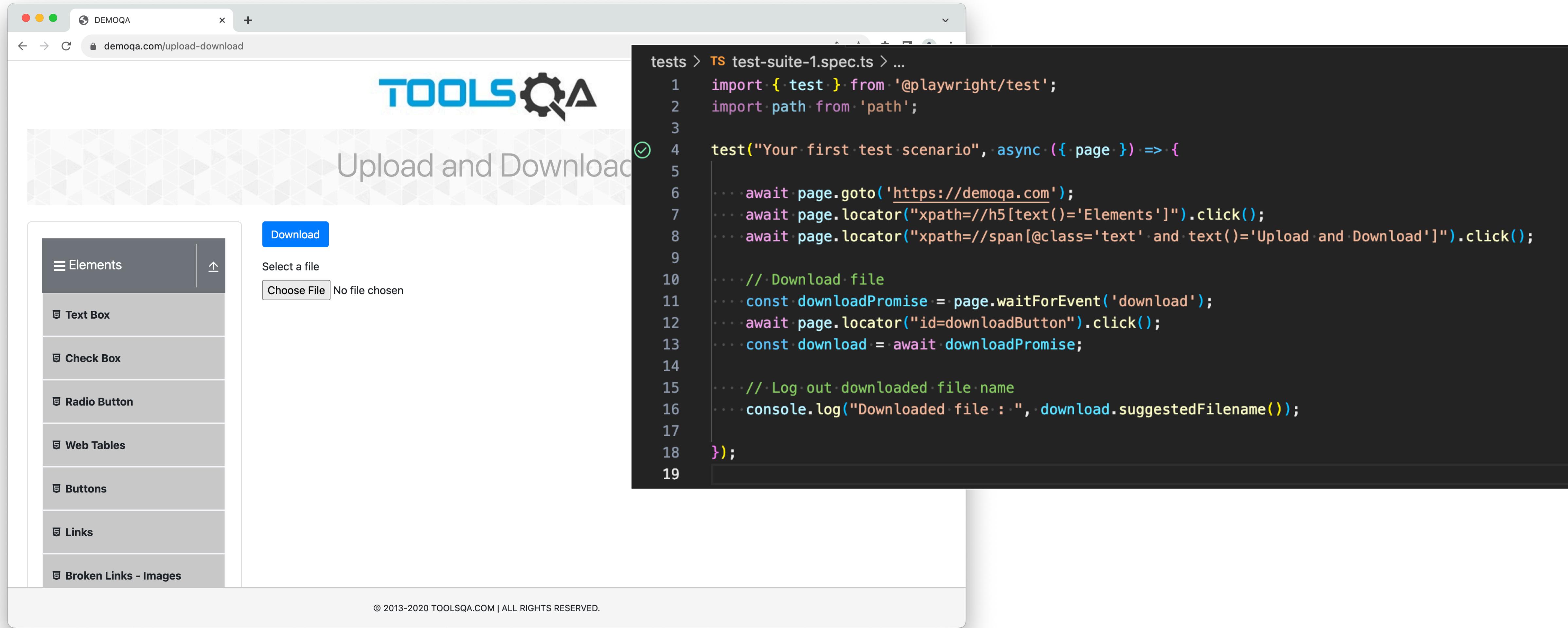
The screenshot shows a web browser window for 'DEMOQA' at 'demoqa.com/upload-download'. The page title is 'Upload and Download'. On the left, a sidebar menu lists 'Elements', 'Text Box', 'Check Box', 'Radio Button', 'Web Tables', 'Buttons', 'Links', and 'Broken Links - Images'. A 'Download' button is visible. In the center, there's a 'Select a file' input field with 'Choose File' and 'No file chosen' buttons. The right side of the browser window displays a terminal-like interface with a code editor showing a TypeScript file:

```
tests > ts test-suite-1.spec.ts > ...
1 import { test } from '@playwright/test';
2 import path from 'path';
3
4 test("Your first test scenario", async ({ page }) => {
5
6   await page.goto('https://demoqa.com');
7   await page.locator("xpath=//h5[text()='Elements']").click();
8   await page.locator("xpath=//span[@class='text' and text()='Upload and Download']").click();
9
10 // Upload file
11 await page.locator("id=uploadFile").setInputFiles(path.join(__dirname, '../assets/sampleFile.jpeg'));
12
13 });
14
```

Below the browser window, a note states: `__dirname` : Path of the current directory file that is running.

Download file

>> Elements > Upload and Download

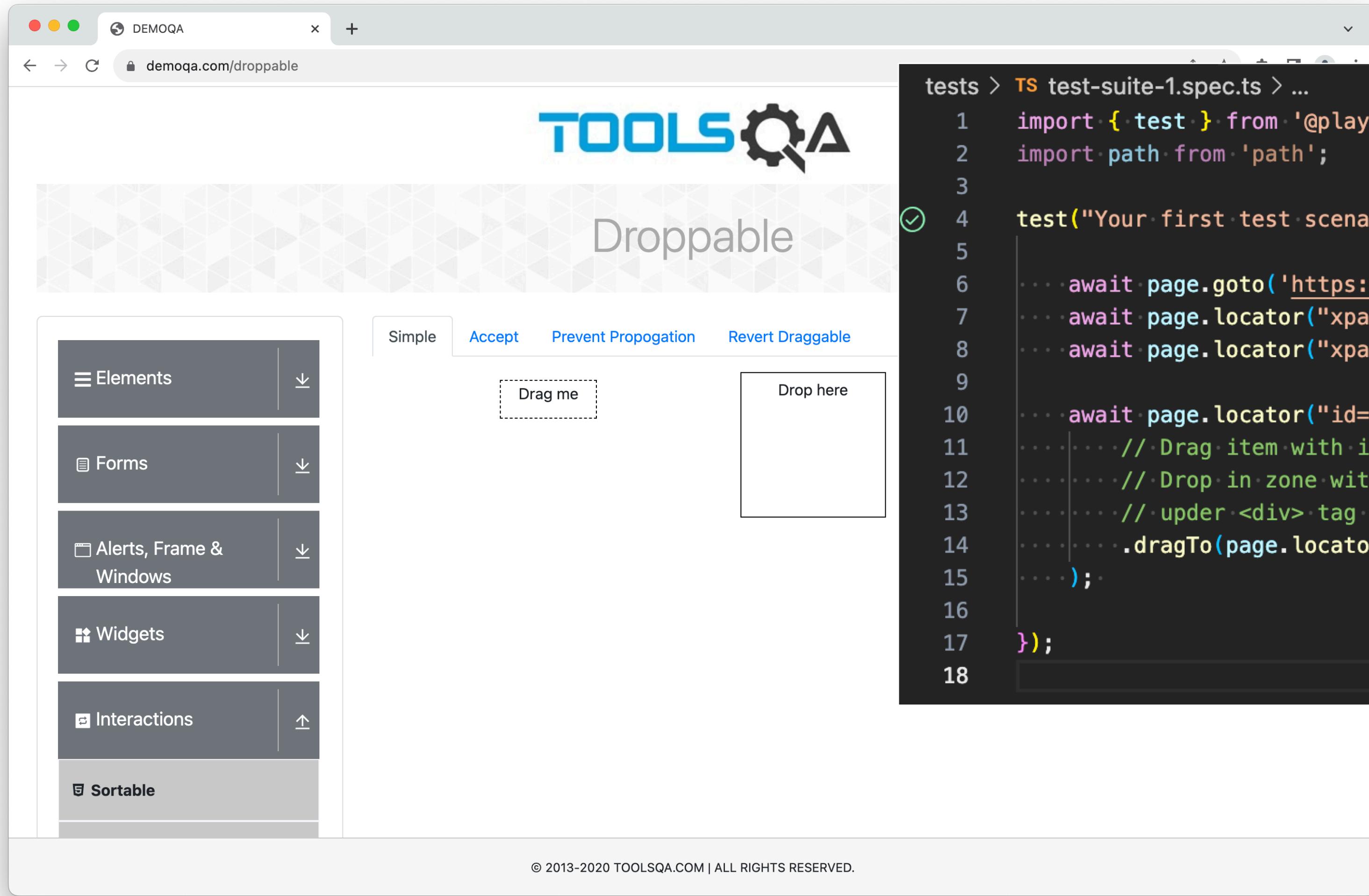


```
tests > TS test-suite-1.spec.ts > ...
1 import { test } from '@playwright/test';
2 import path from 'path';
3
4 test("Your first test scenario", async ({ page }) => {
5
6     await page.goto('https://demoqa.com');
7     await page.locator("xpath=/h5[text()='Elements']").click();
8     await page.locator("xpath=/span[@class='text' and text()='Upload and Download']").click();
9
10    // Download file
11    const downloadPromise = page.waitForEvent('download');
12    await page.locator("id=downloadButton").click();
13    const download = await downloadPromise;
14
15    // Log out downloaded file name
16    console.log("Downloaded file ::", download.suggestedFilename());
17
18});
19
```

Drag and drop

>> Interactions > Droppable

Drag me to drop zone plz.



The screenshot shows a browser window with the title 'DEMOQA' and the URL 'demoqa.com/droppable'. The page has a header 'TOOLS QA' and a main section titled 'Droppable'. On the left, there's a sidebar with categories: Elements, Forms, Alerts, Frame & Windows, Widgets, Interactions (which is expanded), and Sortable. In the center, there are two boxes: a dashed box labeled 'Drag me' and a solid box labeled 'Drop here'. Above these boxes are buttons: Simple, Accept, Prevent Propagation, and Revert Draggable. A checkmark icon is next to the 'Simple' button. To the right of the page content is a terminal window showing a TypeScript test script:

```
tests > TS test-suite-1.spec.ts > ...
1 import { test } from '@playwright/test';
2 import path from 'path';
3
4 test("Your first test scenario", async ({ page }) => {
5
6     await page.goto('https://demoqa.com');
7     await page.locator("xpath=//h5[text()='Interactions']").click();
8     await page.locator("xpath=//span[@class='text' and text()='Droppable']").click();
9
10    await page.locator("id=draggable").
11        // Drag item with id "draggable"
12        // Drop in zone with id "droppable"
13        // under <div> tag with class "simple-drop-container"
14        .dragTo(page.locator("css=div.simple-drop-container > id=droppable"))
15    ;
16
17 });
18
```

Scraping data

- Get text from web
- Get data in table
- Get current page url

Get data from web

>> Elements > Text Box

DEMOQA

demoqa.com/text-box

Text Box

Full Name: Jojoe

Email: jojoe@gmail.com

Current Address: 123 road avenue city city nowhere

Permanent Address: homeless

Name:Jojoe
Email:jojoe@gmail.com
Current Address :123 road avenue city city nowhere
Permananet Address :homeless

```
tests > TS test-suite-1.spec.ts > ...
1 import { test } from '@playwright/test';
2
3 test("Get data", async ({ page }) => {
4
5     await page.goto("https://demoqa.com");
6     await page.locator("xpath=//h5[text()='Elements']").click();
7     await page.locator("xpath=//span[@class='text' and text()='Text Box']").click();
8
9     // Fill in data to form and submit
10    await page.locator("id=userName").fill("Jojoe");
11    await page.locator("id=submit").click();
12
13    // Get data from some HTML tag with id "name"
14    const name = await page.locator("id=name").innerText();
15    console.log("Name: ", name);
16
17 });
18
```

PROBLEMS OUTPUT TEST RESULTS DEBUG CONSOLE TERMINAL PORTS ROBOT OUTPUT

```
/opt/homebrew/bin/node ./node_modules/@playwright/test/cli.js test -c playwright.config.ts /Use
1\spec\ts:3 --headed --project=chromium --repeat-each 1 --retries 0 --timeout 0 --workers 1

Running 1 test using 1 worker
Name: Name:Jojoe
1 passed (12.1s)
```

Get data in table

>> Elements > Web Tables

First Name	Last Name	Age	Email	Salary	Department
Cierra	Vega	39	cierra@exa...	10000	Insurance
Alden	Cantrell	45	alden@exa...	12000	Compliance
Kierra	Gentry	29	kierra@exa...	2000	Legal

```
tests > TS test-suite-1.spec.ts > ...
23
24  test("Get data", async ({ page }) => {
25
26    await page.goto("https://demoqa.com/webtables");
27    await page.locator("xpath=//h5[text()='Elements']").click();
28    await page.locator("xpath=//span[@class='text' and text()='Web Tables']").click();
29
30    // Rows locator
31    const rowLocator = page.locator(
32      `xpath=//div[@class='rt-tbody']//div[@role='row' and (@class!='rt-tr--padRow--even' and @class!='rt-tr--padRow--odd')]`);
33
34    const rowCount = await rowLocator.count();
35    const rowElements = new Array<EmployeeRowElement>(); // Array for collecting rows locator
36    for (let i = 0; i < rowCount; i++) {
37      rowElements.push(new EmployeeRowElement(page, rowLocator.nth(i))); // Get each row by index
38    }
39
40    for await (const row of rowElements) {
41      const fName = await row.getFirstName(); // Get data from row
42      console.log(fName);
43    }
44  });
45
46 });
47
```

PROBLEMS OUTPUT TEST RESULTS DEBUG CONSOLE TERMINAL PORTS ... Filter (e.g. text, !exclude)

```
/opt/homebrew/bin/node ./node_modules/@playwright/test/cli.js test -c playwright.config.ts /Users/jojo/Reht101/tests/test-suite-1\spec.ts:24 --headed --project=chromium --repeat-each 1 --retries 0 --timeout 0
```

Running 1 test using 1 worker

Cierra
Alden
Kierra

1 passed (19.6s)

More code in next page

Don't worry about this code, we will discuss it again in session **Page Object Model**

```
tests > TS test-suite-1.spec.ts > ...
1 import { Locator, Page, test } from '@playwright/test';
2
3 class BaseElement {
4
5   protected page: Page;
6   protected elementLocator: Locator;
7   constructor(page: Page, elementLocator: Locator) {
8     this.page = page;
9     this.elementLocator = elementLocator;
10  }
11 }
12
13 class EmployeeRowElement extends BaseElement {
14
15   private _fName: Locator = this.elementLocator.locator("xpath=/div[@role='gridcell'][1]");
16   private _delete: Locator = this.elementLocator.locator("xpath=/div[@role='gridcell'][8]");
17
18   public getFirstName() {
19     return this._fName.innerText();
20   }
21
22 }
```

Get current page url

```
tests > TS test-suite-1.spec.ts > ...
1   import { Locator, Page, test } from '@playwright/test';
2
3  ⚡ 3  test("Get data", async ({ page }) => {
4
5    ... await page.goto("https://demoqa.com");
6    ... await page.locator("xpath=//h5[text()='Elements']").click();
7
8    ... // Get current page url
9    const currentPageUrl = page.url();
10   console.log("Current page: ", currentPageUrl);
11
12 });
13
```

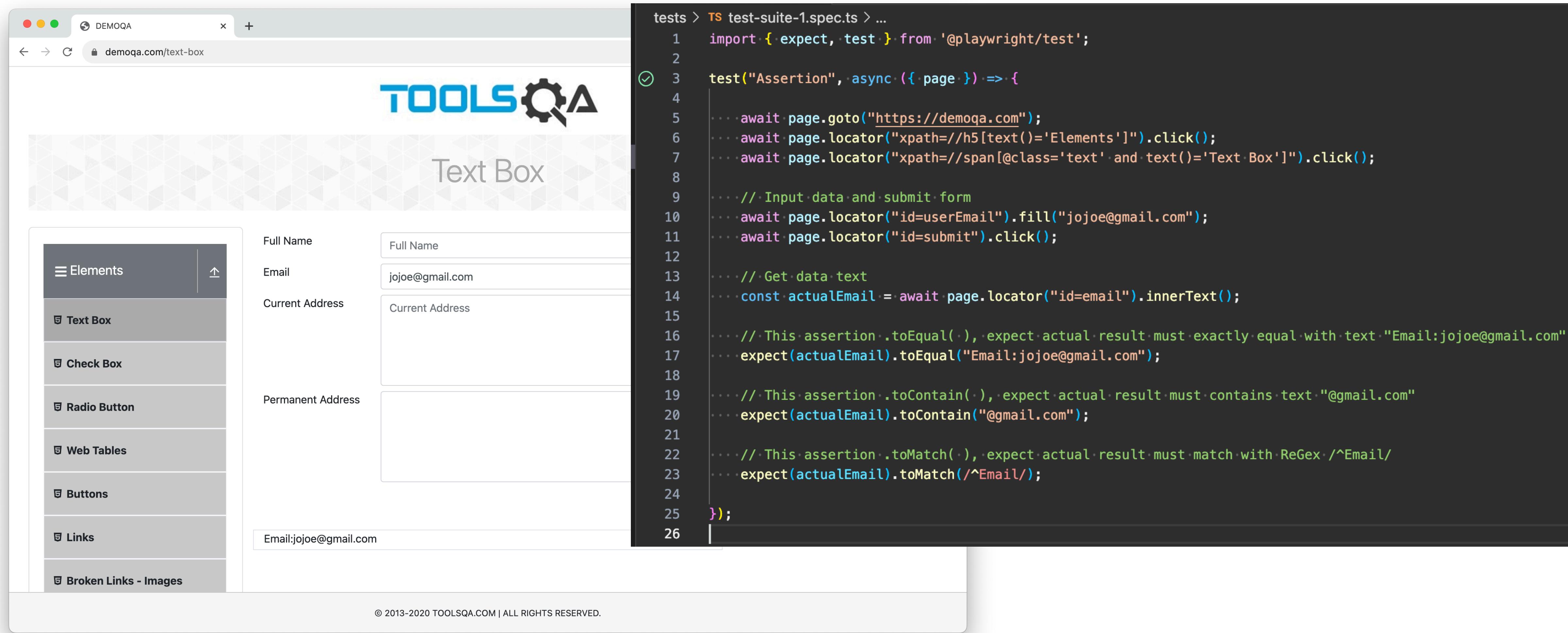
```
PROBLEMS OUTPUT TEST RESULTS DEBUG CONSOLE TERMINAL PORTS ROBOT
/opt/homebrew/bin/node ./node_modules/@playwright/test/cli.js test -c playwright101/tests/test-suite-1\spec\ts:3 --headed --project=chromium --repeat-e
Running 1 test using 1 worker
Current page: https://demoqa.com/elements
1 passed (5.9s)
```

Assertion

- Text
- Number
- Date and time

Text assertion

>> Elements > Text Box



The screenshot shows a browser window with the URL demoqa.com/text-box. The page title is "Text Box". On the left, there's a sidebar with a navigation menu:

- Elements
- Text Box (selected)
- Check Box
- Radio Button
- Web Tables
- Buttons
- Links
- Broken Links - Images

The main content area has four input fields:

- Full Name: Full Name
- Email: jojoe@gmail.com
- Current Address: Current Address
- Permanent Address: (empty)

At the bottom, there's a footer with the text "Email:jojoe@gmail.com".

To the right of the browser window is a code editor displaying a TypeScript test script:

```
tests > TS test-suite-1.spec.ts > ...
1 import { expect, test } from '@playwright/test';
2
3 test("Assertion", async ({ page }) => {
4
5     await page.goto("https://demoqa.com");
6     await page.locator("xpath=//h5[text()='Elements']").click();
7     await page.locator("xpath=//span[@class='text' and text()='Text Box']").click();
8
9     // Input data and submit form
10    await page.locator("id=userEmail").fill("jojoe@gmail.com");
11    await page.locator("id=submit").click();
12
13    // Get data text
14    const actualEmail = await page.locator("id=email").innerText();
15
16    // This assertion .toEqual(), expect.actual.result must exactly equal with text "Email:jojoe@gmail.com"
17    expect(actualEmail).toEqual("Email:jojoe@gmail.com");
18
19    // This assertion .toContain(), expect.actual.result must contains text "@gmail.com"
20    expect(actualEmail).toContain("@gmail.com");
21
22    // This assertion .toMatch(), expect.actual.result must match with ReGex /^Email/
23    expect(actualEmail).toMatch(/^Email/);
24
25 });
26
```

Number assertion

```
tests > TS test-suite-1.spec.ts > ...
1   import { expect, test } from '@playwright/test';
2
3
4   test("Assertion", async () => {
5     const num = 123.55;
6
7     expect(num).toEqual(123.55); - 2ms
8
9     expect(num).toBeLessThanOrEqual(150); - 0ms
10
11    expect(num).toBeGreaterThanOrEqual(100); - 0ms
12
13
14 })
15
```

Test steps:

Line6 Create a constant "num" and assign value for 123.55

Line8 Assert "num" must equal to 123.55

Line10 Assert "num" must less than or equal to 150

Line12 Assert "num" must greater than or equal to 100

Date time assertion

>> Widgets > Date Picker

Unfortunately, Playwright unable to assert Date and Time directly

So, we must convert DateTime to Number format via function `.getTime()`

That will return us the time since Thursday, 1 January 1970 07:00:00 GMT+07:00
In second

The screenshot shows a browser window with the ToolsQA logo at the top. Below it is a "Date Picker" example page. On the left, a sidebar menu lists various UI components: Elements, Forms, Alerts, Frame & Windows, Widgets, Accordion, Auto Complete, and Date Picker. The "Widgets" item is currently selected. The main content area displays a date input field with the value "12/12/2023" and a date-time input field below it with the value "December 12, 2023 11:5". To the right of the browser window is the DevTools Variables panel. It shows a variable named "currentDate" with the value "Wed Dec 13 2023 00:00:00...". The "WATCH" section is empty. Below the browser window, a code editor window shows the following TypeScript code:

```
test("Assertion", async ({ page }) => {
    await page.goto("https://demoqa.com");
    await page.locator("xpath=//h5[text()='Widgets']").click();
    await page.locator("xpath=//span[@class='text' and text()='Date Picker']").click();

    // Get current date picker value
    const currentDateText = await page.locator("id=datePickerMonthYearInput").inputValue();

    // Assert currentDate as text
    expect(currentDateText).toEqual("12/13/2023");

    // Assert currentDate as DateTime type
    const currentDate = new Date(currentDateText);
    const time = currentDate.getTime();
    expect(time).toBeGreaterThanOrEqual(1702400400000 - 5000);
    expect(time).toBeLessThanOrEqual(1702400400000 + 5000);
});
```

Test steps:

- Line16 Convert currentDateText from string to DateTime format
- Line17 Get time in second
- Line18 Assert by expect the time in second must greater than or equal 1702400400000 - 5000
- Line19 Assert by expect the time in second must less than or equal 1702400400000 - 5000

The screenshot shows the homepage of epochconverter.com. At the top, there's a navigation bar with a clock icon, the title "EpochConverter", and a subtitle "Epoch & Unix Timestamp Conversion Tools". Below this, a message says "The current Unix epoch time is 1702402124". A section titled "Convert epoch to human-readable date and vice versa" includes a timestamp input field (1702402121), a "Timestamp to Human date" button, and a "batch convert" link. It also mentions supported timestamp formats: seconds, milliseconds, microseconds, and nanoseconds. Below this is a date and time picker with fields for Year (Yr), Month (Mon), Day, Hour (Hr), Minute (Min), and Second (Sec). The current selection is 2023-12-12 17:28:41 GMT. A "Human date to Timestamp" button is also present. Further down, there's another timestamp input field (Tue, 12 Dec 2023 17:28:41 GMT), a "Human date to Timestamp" button, and a "batch convert" link. A note specifies the input format as RFC 2822, D-M-Y, M/D/Y, Y-M-D, etc. Strip 'GMT' to convert to local time. At the bottom, a link says "Prefer a 24-hour clock? Go to preferences."

You can learn more about Current Time Stamp with this website

<https://www.epochconverter.com/>

Pages

- Home
- Preferences
- Toggle theme

Tools ^

- Epoch converter
- Batch converter
- Time zone converter
- Timestamp list
- LDAP converter
- WebKit/Chrome timestamp
- Unix hex timestamp
- Cocoa Core Data timestamp
- Mac HFS+ timestamp
- SAS timestamp
- Seconds/days since year 0
- Bin/Oct/Hex converter
- Countdown in seconds
- Epoch clock

Date and Time ^

- Week numbers
- Weeks by year
- Day numbers
- Days by year

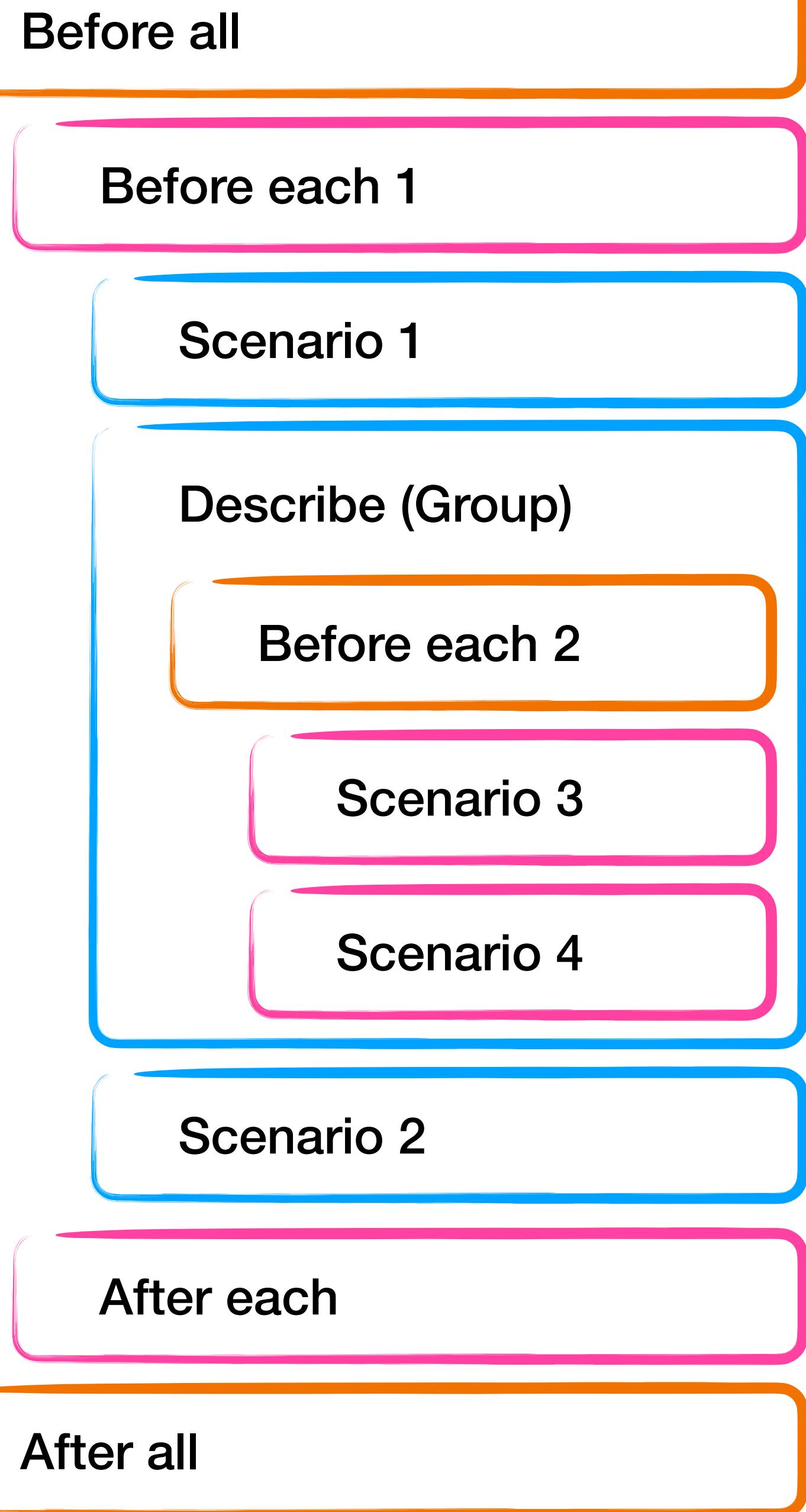
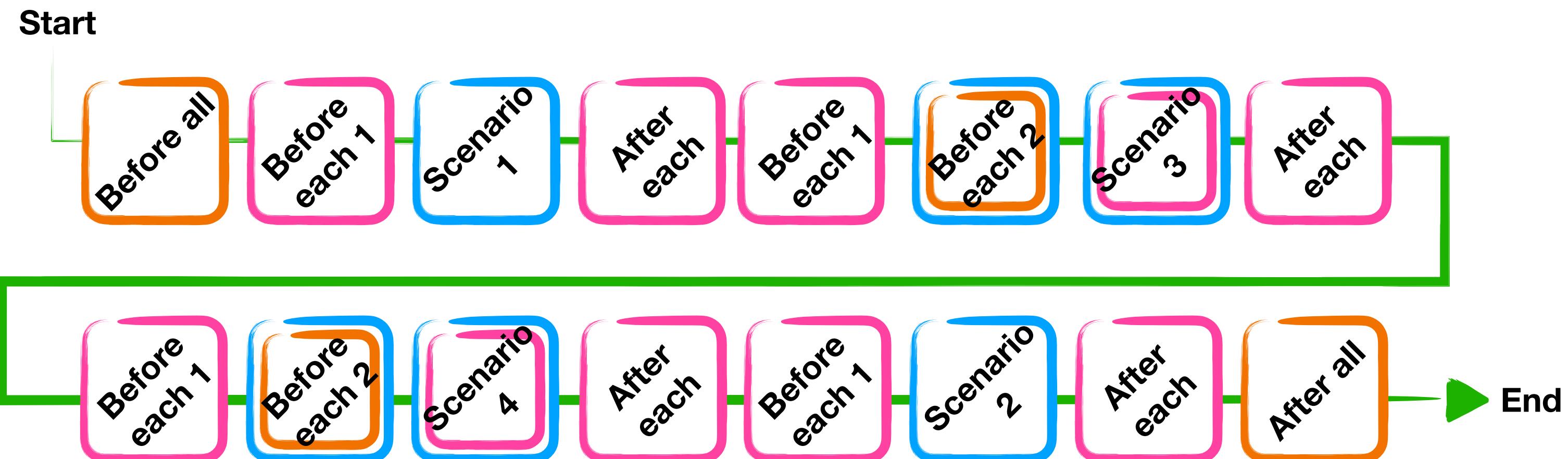
Test suite

- Test setup / teardown
 - `test.beforeAll()`
 - `test.beforeEach()`
 - `test.afterAll()`
 - `test.afterEach()`
- Groups
 - `test.describe()`
- Template

Test setup and teardown

Test setup and teardown are procedures performed before and after the execution of a test or a set of tests to ensure a controlled and **consistent environment for testing**.

These procedures help in preparing the test environment, performing any necessary configurations, and cleaning up after the tests are executed.



Example all test setup and teardown in a single test suite

Similar to the previous slide,
here is the code that shows how the test scenarios,
descriptions, and test setup/teardown run together.

You can see that the **test setup/teardown**
inside `.describe()` block will not affect test scenarios
outside the block.

```
tests > TS test-suite-1.spec.ts > ...
1 import { test } from '@playwright/test';
2
3 test.beforeAll(async () => {
4   // 9ms
5   console.log('Before all');
6 }
7 test.beforeEach(async () => {
8   // 6ms
9   console.log('Before each 1');
10 }
11 test("Scenario 1", async () => {
12   // scenario 1
13 }
14
15 test.describe("Describe", async () => {
16
17   test.beforeEach(async () => {
18     // 0ms
19     console.log('Before each 2');
20   }
21   test("Scenario 3", async () => {
22     // scenario 3
23   }
24
25   test("Scenario 4", async () => {
26     // scenario 4
27   }
28 }
29
30 test("Scenario 2", async () => {
31   // scenario 2
32 }
33
34 test.afterEach(async () => {
35   // 0ms
36   console.log('After each');
37 }
38
39 test.afterAll(async () => {
40   // 5ms
41   console.log('After all');
42 }
```

```
Filter (e.g. text, !exclude)
/opt/homebrew/bin/node ./node_modules/playwright/config.ts /Users/jojo/Repos/tests/test-suite-1\spec\ts --headless
1 --retries 0 --timeout 0 --workers 1
Running 4 tests using 1 worker
Before all
Before each 1
scenario 1
After each
Before each 1
Before each 2
scenario 3
After each
Before each 1
Before each 2
scenario 4
After each
Before each 1
scenario 2
After each
After all
4 passed (434ms)
```

Test suite structure

It's quite straightforward in the naming of their functions

.describe() a group of test scenarios

```
tests > ts test-suite-1.spec.ts > ...
1  import { test } from '@playwright/test';
2
3  test.beforeAll(async () => {
4    console.log("Before all");
5  });
6
7  test.beforeEach(async () => {
8    console.log("Before each 1");
9  });
10
11 test("Scenario 1", async () => {
12   console.log("scenario 1");
13 });
14
15 test.describe("Describe", async () => {
16
17   test.beforeEach(async () => {
18     console.log("Before each 2");
19   });
20
21   test("Scenario 3", async () => {
22     console.log("scenario 3");
23   });
24
25   test("Scenario 4", async () => {
26     console.log("scenario 4");
27   });
28 });
29
30 test("Scenario 2", async () => {
31   console.log("scenario 2");
32 });
33
34 test.afterEach(async () => {
35   console.log("After each");
36 });
37
38 test.afterAll(async () => {
39   console.log("After all");
40 });
41
42
```

.beforeAll() will run at the beginning of the test suite execution

.beforeEach() will run every time before each test scenario is executed

This .beforeEach() will run every time before each test scenario in it's block is executed

.afterEach() will run every time after each test scenario is executed

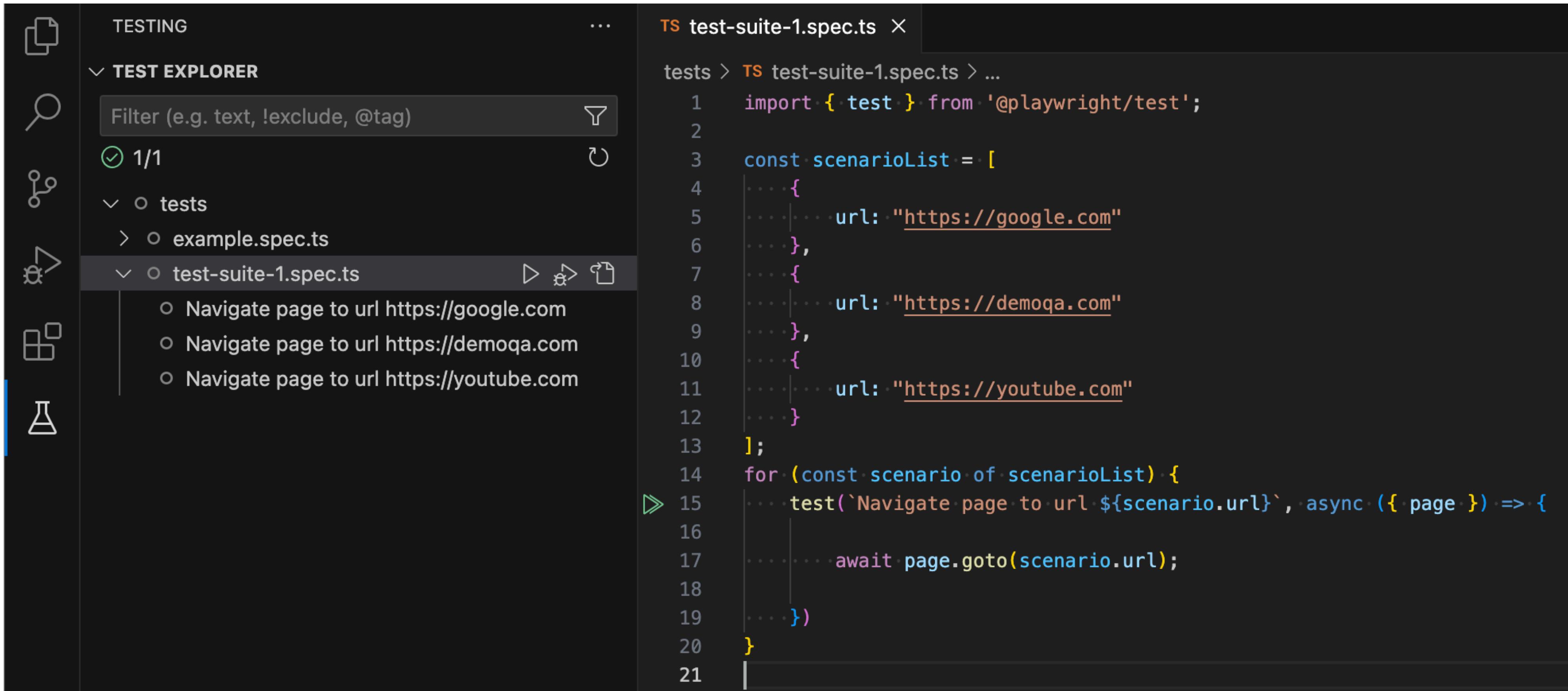
.afterAll() will run at the end of the test suite execution

```
tests > TS test-suite-1.spec.ts > ⚘ test.describe("Describe") callback > ⚘ test.afterAll
10
11  test("Scenario 1", async () => {
12  |... console.log("scenario 1");
13  });
14
15  test.describe("Describe", async () => {
16  |... test.beforeAll(async () => {
17  |...|... console.log("Before all 2");
18  |...});
19
20  |... test.beforeEach(async () => {
21  |...|... console.log("Before each 2");
22  |...});
23
24  |... test.beforeEach(async () => {
25  |...|... console.log("Before each 3");
26  |...});
27
28  |... test("Scenario 3", async () => {
29  |...|... console.log("scenario 3");
30  |...});
31
32
33  |... test("Scenario 4", async () => {
34  |...|... console.log("scenario 4");
35  |...});
36
37  |... test.afterEach(async () => {
38  |...|... console.log("After each 2");
39  |...});
40
41  |... test.afterAll(async () => {
42  |...|... console.log("After all 2");
43  |...});
44
45 });
46
47 test("Scenario 2", async () => {
48 |... console.log("scenario 2");
49 });
50
51 test.afterEach(async () => {
52 |... console.log("After each");
53 });
54
```

What if?
You have more than one test setup/teardown steps

Try add this, this, and this steps to your code
,then see what is going on

Test scenario template



The screenshot shows the Visual Studio Code interface with the following details:

- TESTING** sidebar icon.
- TEST EXPLORER** section:
 - Filter input field: "Filter (e.g. text, !exclude, @tag)".
 - Count: 1/1.
 - Tree view:
 - tests
 - > example.spec.ts
 - > test-suite-1.spec.ts
 - o Navigate page to url https://google.com
 - o Navigate page to url https://demoqa.com
 - o Navigate page to url https://youtube.com- Code Editor**:
 - File: `test-suite-1.spec.ts`.
 - Content:

```
1 import { test } from '@playwright/test';
2
3 const scenarioList = [
4   {
5     url: "https://google.com"
6   },
7   {
8     url: "https://demoqa.com"
9   },
10  {
11    url: "https://youtube.com"
12  }
13];
14 for (const scenario of scenarioList) {
15   test(`Navigate page to url ${scenario.url}`, async ({ page }) => {
16     await page.goto(scenario.url);
17   })
18 }
19
20
21
```

**Are you still alright?
Day 1 end at this page.**

API tests

- Chrome inspect and network
- HTTP header
- Requests
 - GET
 - POST / PUT / PATCH
 - DELETE
- JSON parse

Chrome Inspect and Network

The screenshot shows two instances of the Reqres website (reqres.in) in a browser. The left instance has a context menu open over the page content, with the 'Inspect' option highlighted and circled in orange. The right instance shows the Chrome DevTools Network tab, which is also circled in orange. The Network tab displays a list of network requests, with the first request's response body visible:

```
1 { "data": { "id": 2, "email": "janet.weaver@reqres.in", "first_name": "Janet", "last_name": "Weaver", "avatar": "https://reqres.in/img/faces/2-image.jpg" }, "support": { "url": "https://reqres.in/#support-heading", "text": "To keep BeepBeep free, contributions towards server costs are appreciated!" }}
```

Open inspect and console steps:

1. Right click on page and select "Inspect" or press "F12"
2. In the inspect tab, select "Network"

REQRES : New playground with fake data :/

Open Chrome browser
Go to this url: <https://reqres.in/>

The screenshot shows a Chrome browser window with the title "Reqres - A hosted REST-API re..." and the URL "reqres.in". The left side of the interface is a sidebar listing various API endpoints categorized by method (GET, POST, PUT, PATCH, DELETE) and resource type (LIST USERS, SINGLE USER, etc.). On the right, there are two main sections: "Request" and "Response". The "Request" section shows a purple button labeled "GET" next to the URL "/api/users?page=2". The "Response" section shows a green "200" status code and a JSON object representing a paginated list of users. The JSON response is as follows:

```
{  
  "page": 2,  
  "per_page": 6,  
  "total": 12,  
  "total_pages": 2,  
  "data": [  
    {  
      "id": 7,  
      "email": "michael.lawson@reqres.in",  
      "first_name": "Michael",  
      "last_name": "Lawson",  
      "avatar": "https://reqres.in/img/faces/7.jpg"  
    },  
    {  
      "id": 8,  
      "email": "lindsay.ferguson@reqres.in",  
      "first_name": "Lindsay",  
      "last_name": "Ferguson",  
      "avatar": "https://reqres.in/img/faces/8.jpg"  
    },  
    {  
      "id": 9,  
      "email": "tobias.funke@reqres.in",  
      "first_name": "Tobias",  
      "last_name": "Funke",  
      "avatar": "https://reqres.in/img/faces/9.jpg"  
    },  
    {  
      "id": 10,  
      "email": "byron.fields@reqres.in",  
      "first_name": "Byron",  
      "last_name": "Fields",  
      "avatar": "https://reqres.in/img/faces/10.jpg"  
    }]  
}
```

HTTP header

The screenshot shows the Network tab in the Chrome DevTools. A specific request for 'users?page=2' is selected. In the 'Request Headers' section, an orange warning message reads: '⚠ Provisional headers are shown. Disable cache to see full headers. [Learn more](#)'. Below this, several headers are listed:

Name	Value
Content-Type	application/json
Referer	https://reqres.in/
Sec-Ch-Ua	"Google Chrome";v="119", "Chromium";v="119", "Not?A_Brand";v="24"
Sec-Ch-Ua-Mobile	?0
Sec-Ch-Ua-Platform	"macOS"
User-Agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.0.0 Safari/537.36

```
tests > TS test-suite-1.spec.ts > ...
1 import { test, request } from '@playwright/test';
2
3 test("HTTP Request", async () => {
4
5     // Create HTTP header
6     const httpContext = await request.newContext({
7         extraHTTPHeaders: {
8             "Authorization": "" // Add your accessToken here if required
9         }
10    });
11
12 });
13 }
```

Here is the **API request header**; most of them are automatically added.

However, in some special cases, you may need to assign a key and value to the header for extras such as

- Authorization (accessToken)
- accepted file types

and so on.

Request GET

```
tests > TS test-suite-1.spec.ts > ...
1   import { test, request } from '@playwright/test';
2
3  ⚡ test("HTTP Request", async () => {
4    // Create HTTP header
5    const httpContext = await request.newContext({
6      extraHTTPHeaders: {
7        "Authorization": "" // Add your accessToken here if required
8      }
9    });
10
11   // HTTP request GET from domain https://reqres.in and path /api/users/2
12   const httpResponse = await httpContext.get("https://reqres.in/api/users/2");
13   console.log(`HTTP request GET response code: ${httpResponse.status()} ${httpResponse.statusText()}`);
14
15
16 });
17
```

Test steps:

- Line5 Create new HTTP context and add http header
- Line8 Add item name "Authorization" with empty string to header (In this case, no need any access token)
- Line13 Sent request GET with created HTTP context to url <https://reqres.in/api/users/2>
- Line14 Log out response status and text to console

PROBLEMS OUTPUT DEBUG CONSOLE TEST RESULTS TERMINAL PORTS ROBOT OUTPUT Filter (e.g. text, !exclude)

```
/opt/homebrew/bin/node ./node_modules/@playwright/test/cli.js test -c playwright.config.ts /Users/jojoee/Repositories/1\spec\ts:3 --headed --project=chromium --repeat-each 1 --retries 0 --timeout 0 --workers 1

Running 1 test using 1 worker
HTTP request GET response code: 200 OK
1 passed (550ms)
```

The screenshot shows a list of API endpoints on the left and a detailed view of a specific request on the right. The list includes:

- GET LIST USERS
- GET SINGLE USER
- GET SINGLE USER NOT FOUND
- GET LIST <RESOURCE>
- GET SINGLE <RESOURCE>
- GET SINGLE <RESOURCE> NOT FOUND
- POST CREATE
- PUT UPDATE
- PATCH UPDATE
- DELETE DELETE
- POST REGISTER - SUCCESSFUL
- POST REGISTER - UNSUCCESSFUL
- POST LOGIN - SUCCESSFUL
- POST LOGIN - UNSUCCESSFUL

The detailed view shows a request to `/api/users/23` with a status of `404` and an empty response body `{}`.

API Request

Sent GET request to this path

After the previous slide show you how to GET user info from url <https://reqres.in/api/users/2>

API Request assignment 1:
Sent request to get the non-exist user with id=23

It should response you with status **404 Not Found**

Request POST

```
tests > TS test-suite-1.spec.ts > ...
1  import { test, request } from '@playwright/test';
2
3  test("HTTP Request", async () => {
4
5    // Create HTTP header
6    const httpContext = await request.newContext({
7      extraHTTPHeaders: {
8        "Authorization": "" // Add your accessToken here if required
9      }
10     });
11
12    // HTTP request GET from domain https://reqres.in and path /api/users
13    const httpResponse = await httpContext.post("https://reqres.in/api/users", {
14      data: {
15        "name": "morpheus",
16        "job": "leader"
17      }
18    });
19    console.log(`HTTP request POST response code: ${httpResponse.status}`);
20
21  });
22
```

Test steps:

- Line5 Create new HTTP context and add http header
- Line8 Add item name "Authorization" with empty string to header (In this case, no need any access token)
- Line13 Sent request POST with created HTTP context to url <https://reqres.in/api/users>
- Line14 Add data payload to request POST
- Line19 Log out response status and text to console

PROBLEMS OUTPUT DEBUG CONSOLE TEST RESULTS TERMINAL PORTS RO
/opt/homebrew/bin/node ./node_modules/@playwright/test/cli.js test -c pl
1\\.spec\\.ts:4 --headed --project=chromium --repeat-each 1 --retries 0 --t

Running 1 test using 1 worker
HTTP request POST response code: 201 Created
1 passed (1.1s)

Request PUT / PATCH

```
tests > TS test-suite-1.spec.ts > ...
1  import { test, request } from '@playwright/test';
2
3  test("HTTP Request", async () => {
4
5    // Create HTTP header
6    const httpContext = await request.newContext({ - 1ms
7    extraHTTPHeaders: {
8      "Authorization": "" // Add your accessToken here if required
9    }
10   });
11
12  // HTTP request GET from domain https://reqres.in and path /api/users/2
13  const httpResponse = await httpContext.put("https://reqres.in/api/users/2", { - 847ms
14  data: {
15    name: "morpheus",
16    job: "leader update"
17  }
18  );
19  console.log(`HTTP request PUT response code: ${httpResponse.status}`);
20
21 });
22 |
```

Test steps:

- Line5 Create new HTTP context and add http header
- Line8 Add item name "Authorization" with empty string to header (In this case, no need any access token)
- Line13 Sent request PUT with created HTTP context to url <https://reqres.in/api/users/2>
- Line14 Add data payload to request POST
- Line19 Log out response status and text to console

PROBLEMS OUTPUT DEBUG CONSOLE TEST RESULTS TERMINAL PORTS RO

```
/opt/homebrew/bin/node ./node_modules/@playwright/test/cli.js test -c pl
1\spec\ts:3 --headed --project=chromium --repeat-each 1 --retries 0 --
Running 1 test using 1 worker
HTTP request PUT response code: 200 OK
1 passed (1.2s)
```

The screenshot shows the Reqres API documentation. On the left, a sidebar lists various endpoints categorized by method (GET, POST, PUT, PATCH, DELETE) and resource type (LIST USERS, SINGLE USER, etc.). A large orange circle highlights the PATCH endpoint for user ID 2. To the right, a detailed view of this endpoint is shown, including the Request URL (/api/users/2), Response status (200), and the JSON response body:

```
{  
  "name": "morpheus",  
  "job": "zion resident"  
}
```

The Response body is also highlighted with an orange circle. An orange arrow points from the highlighted text in the question below to this highlighted area.

Guess? How PATCH use Sent PATCH request to this path

After the previous slide show you how to PATCH user info from url <https://reqres.in/api/users/2>

API Request assignment 2:
Sent request to patch the user with id=2

It should response you with status **200 OK**

Request DELETE

```
tests > TS test-suite-1.spec.ts > ...
1  import { test, request } from '@playwright/test';
2
3  test("HTTP Request", async () => {
4    // Create HTTP header
5    const httpContext = await request.newContext({ - 2ms
6      extraHTTPHeaders: {
7        "Authorization": "" // Add your accessToken here if required
8      }
9    });
10
11   // HTTP request GET from domain https://reqres.in and path /api/users/2
12   const httpResponse = await httpContext.delete("https://reqres.in/api/users/2"); - 808ms
13   console.log(`HTTP request DELETE response code: ${httpResponse.status()} ${httpResponse.statusText()}`);
14
15 });
16
17
```

Test steps:

- Line5 Create new HTTP context and add http header
- Line8 Add item name "Authorization" with empty string to header (In this case, no need any access token)
- Line13 Sent request DELETE with created HTTP context to url <https://reqres.in/api/users/2>
- Line14 Log out response status and text to console

PROBLEMS OUTPUT DEBUG CONSOLE TEST RESULTS TERMINAL PORTS ROBOT OUTPUT

Filter (e.g. text, !exclude)

```
/opt/homebrew/bin/node ./node_modules/@playwright/test/cli.js test -c playwright.config.ts /Users/jojo/Repositories/1\spec\ts:3 --headed --project=chromium --repeat-each 1 --retries 0 --timeout 0 --workers 1
```

```
Running 1 test using 1 worker
HTTP request DELETE response code: 204 No Content
1 passed (1.2s)
```

JSON parse

Test steps:

Line13 After receive the response of HTTP
GET

Line14 Log out response status and text to
console

Line16 Convert the response body as a
string to JSON form

Line17 Log out the response data

```
tests > TS test-suite-1.spec.ts > ...
1 import { test, request } from '@playwright/test';
2
3 test("HTTP Request", async () => {
4
5     // Create HTTP header
6     const httpContext = await request.newContext({ - 1ms
7     extraHTTPHeaders: {
8         "Authorization": "" // Add your accessToken here if required
9     }
10 });
11
12 // HTTP request GET from domain https://reqres.in and path /api/users/2
13 const httpResponse = await httpContext.get("https://reqres.in/api/users/2"); - 157ms
14 console.log(`HTTP request GET response code: ${httpResponse.status()} ${httpResponse.statusText()}`);
15
16 const data = JSON.parse(await httpResponse.body().toString()); - 1ms
17 console.log("Response data is :", data.data);
18
19 });
20
```

The screenshot shows a browser developer tools interface with several tabs: Elements, Console, Sources, Network, Performance, Memory, Application, Security, and Lighthouse. The Network tab is active, showing a list of network requests. One specific request, identified by an orange circle and arrow, has its response body expanded in the Response panel. The response body is a JSON object:

```
{
  "id": 2,
  "email": "janet.weaver@reqres.in",
  "first_name": "Janet",
  "last_name": "Weaver",
  "avatar": "https://reqres.in/img/faces/2-image.jpg"
}
```

The Console tab shows the output of the test script. It includes the command run, the test name, and the results of the assertions:

```
Running 1 test using 1 worker
HTTP request PUT response code: 200 OK
Response data is : {"id": 2, "email": "janet.weaver@reqres.in", "first_name": "Janet", "last_name": "Weaver", "avatar": "https://reqres.in/img/faces/2-image.jpg"}
arg1: {"id": 2, "email": "janet.weaver@reqres.in", "first_name": "Janet", "last_name": "Weaver", "avatar": "https://reqres.in/img/faces/2-image.jpg", "email": "janet.weaver@reqres.in", "first_name": "Janet", "id": 2, "last_name": "Weaver"}> [[Prototype]]: Object
1 passed (21 ms)
```

Page object model

- What is page object?
- Create page object
 - BasePage class
 - BaseComponent class
 - Page object class

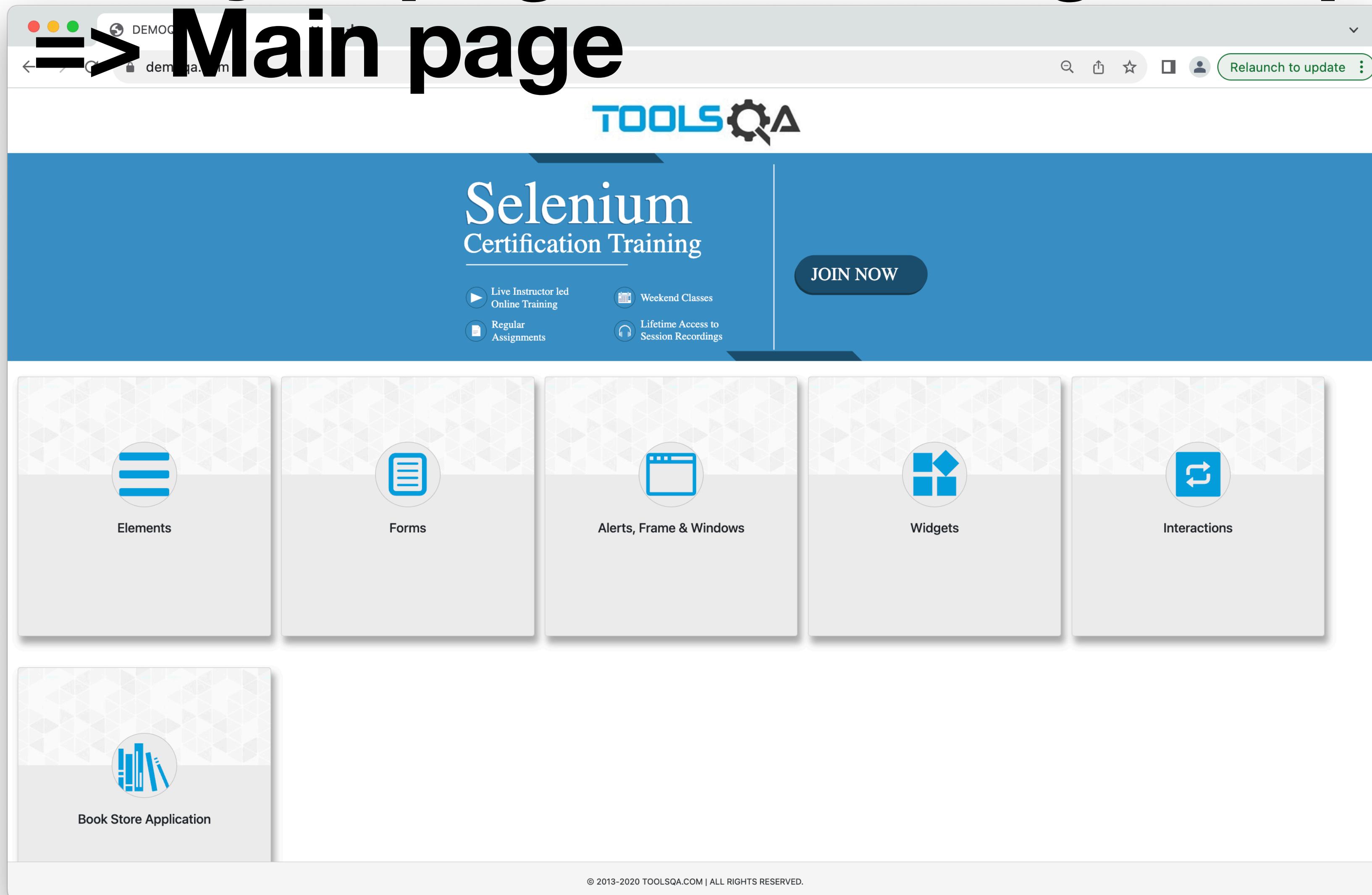
What is page object?

The Page Object Model (POM) is a design pattern commonly used in test automation to enhance the maintainability and readability of test scripts. While the Page Object Model is not specific to any particular testing framework or tool, it can be applied to various technologies and platforms, including Playwright.

In the context of Playwright, the idea of the Page Object Model involves organizing your test automation code in a way that separates the abstraction of web pages from the actual test logic. This helps to achieve better code organization, reusability, and maintainability.

Analyze page containing components

Main page



Main page components

xpath = //h5[text()='Elements']

xpath = //h5[text()='Forms']

xpath = //h5[text()='Alerts, ...']

xpath = //h5[text()='Widgets']

xpath = //h5[text()='Interactions']

xpath = //h5[text()='Book Store ...']

Analyze page containing components

Text Box page

The screenshot shows a web application interface for testing various UI components. The title bar says "Text Box". On the left, a sidebar titled "Elements" lists components such as Text Box, Check Box, Radio Button, Web Tables, Buttons, Links, Broken Links - Images, Upload and Download, Dynamic Properties, Forms, Alerts, Frame & Windows, and Widgets. The main content area has a red header with the British Council IELTS logo. It contains form fields for "Full Name" (with placeholder "name@example.com"), "Email", "Current Address", and "Permanent Address". A "Submit" button is at the bottom right. To the right of the form is an advertisement for "ZeroStep" featuring "Build Playwright Tests using AI" and a "Try for free →" button. Below that is an Adobe Creative Cloud promotion for "ส่วนลด 40% ในปีแรก สำหรับ Creative Cloud" with a colorful flower graphic.

Left panel component

css = div.accordion

Text box page components

id = userName

id = userEmail

id = currentAddress

Analyze page containing components

Web Tables page

The screenshot shows a web application interface for testing web components. The left sidebar lists various UI elements: Elements, Text Box, Check Box, Radio Button, Web Tables, Buttons, Links, Broken Links - Images, Upload and Download, Dynamic Properties, Forms, Alerts, Frame & Windows, and Widgets. The main content area is titled 'Web Tables' and contains a table with the following data:

First Name	Last Name	Age	Email	Salary	Department	Action	
Cierra	Vega	39	cierra@example.com	10000	Insurance		
Alden	Cantrell	45	alden@example.com	12000	Compliance		
Kierra	Gentry	29	kierra@example.com	2000	Legal		

Below the table are navigation buttons: Previous, Page 1 of 1, 10 rows, and Next.

Left panel component

css = div.accordion

Web tables page components

id = addNewRecordButton

id = searchBox

css = div.input-group-append

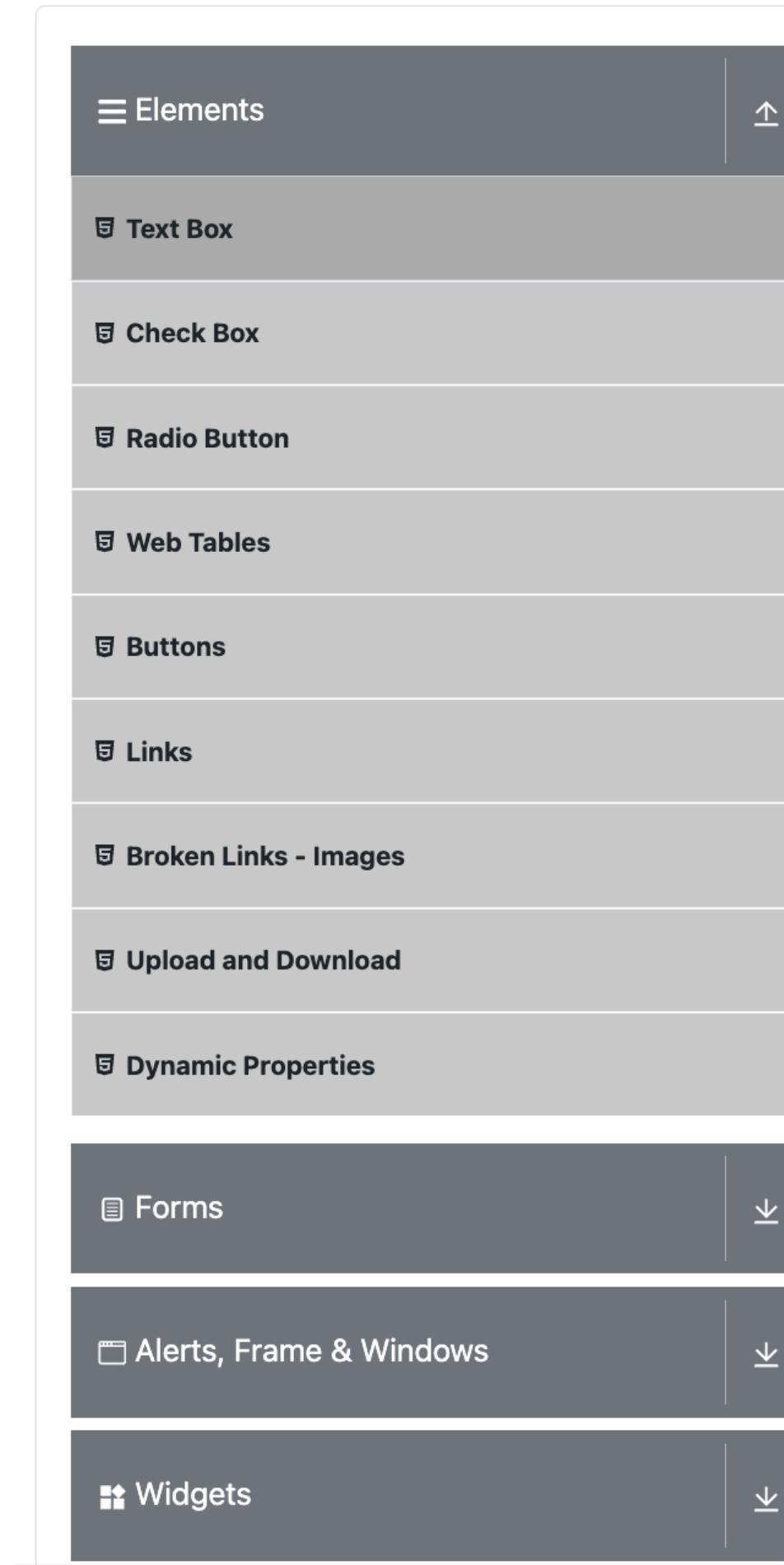
Table component

css = div.rt-thead

css = div.rt-tbody

css = div.rt-tbody >>
xpath = //div[@role='row' and
not(contains(@class, '-padRow'))]

Analyze page containing components => Side bar



Group of menu list

css = div.header-text >> text = 'Elements'

css = span.text >> text = 'Text Box'

css = span.text >> text = 'Check Box'

css = span.text >> text = 'Radio Button'

:

css = div.header-text >> text = 'Forms'

css = div.header-text >> text = 'Alerts, Frame ...'

Analyze page containing components

=> Table, columns & rows

First Name	Last Name	Age	Email	Salary	Department	Action
Cierra	Vega	39	cierra@example.com	10000	Insurance	 
Alden	Cantrell	45	alden@example.com	12000	Compliance	 
Kierra	Gentry	29	kierra@example.com	2000	Legal	 

Previous Page 1 of 1 10 rows Next

Row component

```
css = div.rt-tbody >>  
xpath = //div[@role='row' and  
not(contains(@class, '-padRow'))]
```

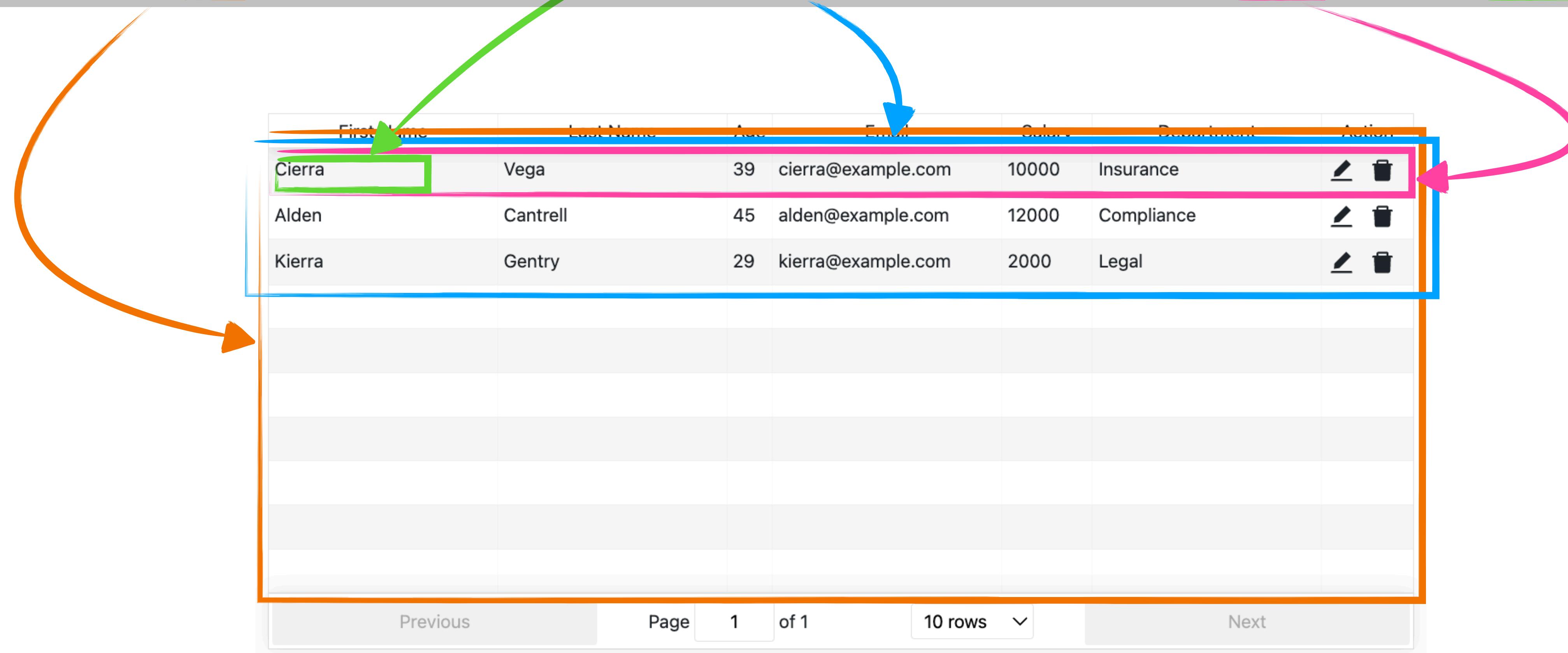
Row elements

```
css = div.rt-tbody >> xpath = //div[@role='row' and not(contains(@class, '-padRow'))]  
>> nth=1 >> xpath = //div[1]
```

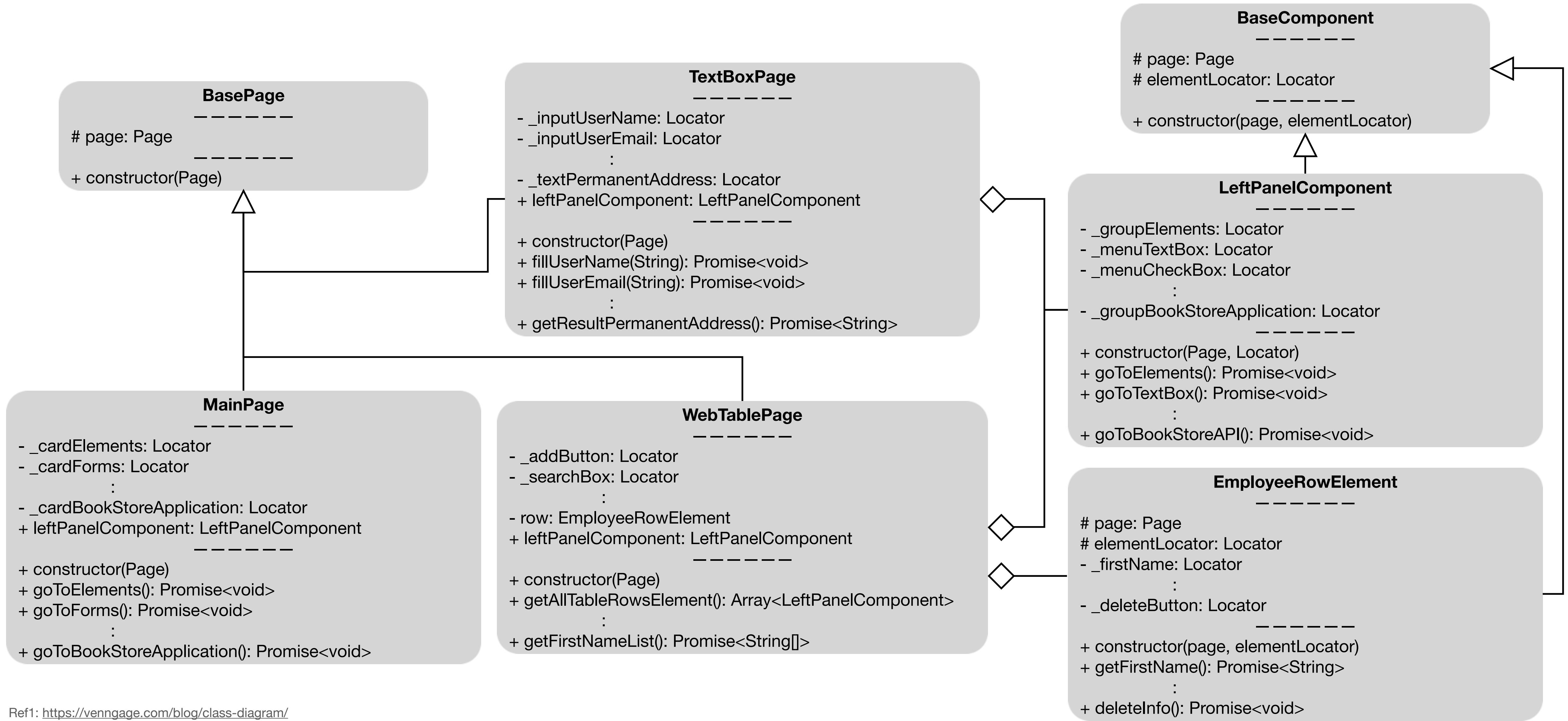
```
css = div.rt-tbody >> xpath = //div[@role='row' and not(contains(@class, '-padRow'))]  
>> nth=1 >> xpath = //div[2]
```

Explain chain locator of row elements

```
css = div.rt-tbody >> xpath = //div[@role='row' and not(contains(@class, '-padRow'))] >> nth = 1 >> xpath = //div[1]
```



Class diagram of DemoQA pages



Test case: Fill text box form

```
tests > TS test-suite-1.spec.ts > ...
1 import { expect, test } from '@playwright/test';
2 import { MainPage } from '../pages/main-page';
3 import { TextBoxPage } from '../pages/text-box-page';
4
5 test("Fill in text box form", async ({ page }) => {
6
7   await page.goto("https://demoqa.com");
8
9   const mainPage = new MainPage(page);
10  await mainPage.goToElements();
11
12  const textboxPage = new TextBoxPage(page);
13  await textboxPage.leftPanal.goToTextBox();
14
15  await textboxPage.fillUserName("Jojoe");
16  await textboxPage.submitForm();
17
18  const resultName = await textboxPage.getUserName();
19  expect(resultName).toEqual("Name: Jojoe")
20
21 });
22 |
```

Test steps:

- Line7 Let page go to url "<https://demoqa.com>"
- Line9 Create new object of MainPage by passing one parameter "page" as required
- Line10 With MainPage object, call function "goToElements"
- Line12 Create new object of TextBoxPage by passing one parameter "page" as required
- Line13 With TextBoxPage object, refer to property name "leftPanal" and call function "goToTextBox" from leftPanal
- Line15 With TextBoxPage object, call function "fillUserName" and pass 1 parameter as string "Jojoe"
- Line16 With TextBoxPage object, call function "submitForm"
- Line18 With TextBoxPage object, call function "getUserName" then assign return value to variable name "resultName"
- Line19 Assert value of resultName must equal to expected value

MainPage

```
pages > TS main-page.ts > ...
1 import { Locator, Page } from "@playwright/test";
2 import { BasePage } from "./base-page";
3
4 export class MainPage extends BasePage {
5
6   ...private readonly _cardElement: Locator = this.page.locator("xpath=//h5[text()='Elements']");
7   ...private readonly _cardForms: Locator = this.page.locator("xpath=//h5[text()='Forms']");
8   ...private readonly _cardWidgets: Locator = this.page.locator("xpath=//h5[text()='Widgets']");
9   ...private readonly _cardAlertsFrameWindows: Locator = this.page.locator("xpath=//h5[text()='Alerts, Frame & Windows']");
10  ...private readonly _cardInteractions: Locator = this.page.locator("xpath=//h5[text()='Interactions']");
11  ...private readonly _cardBookStoreApplication: Locator = this.page.locator("xpath=//h5[text()='Book Store Application']");
12
13  constructor(page: Page) {
14    ...super(page);
15  }
16
17  ...public async goToElements() {
18    ...await this._cardElement.click();
19  }
20
21  > ...public async goToForms() ...
22  ...
23
24  > ...public async goToWidgets() ...
25  ...
26
27  > ...public async goToAlertsFrameWindows() ...
28  ...
29
30  > ...public async goToInteractions() ...
31  ...
32
33  > ...public async goToBookStoreApplication() ...
34  ...
35
36
37  > ...public async goToBookStoreApplication() ...
38  ...
39
40 }
```

Test steps:

- Line4 Create class name "MainPage" and inherit class properties from class "BasePage"
- Line6-11 Declare private property and fixed it as readonly
- Line13 Call class constructor with require one argument type "Page"
- Line14 Pass value of parameter "Page" to super class "BasePage"
- Line17 Create async function name "goToElements"
- Line18 Click at locator "_cardElement"

TextBoxPage

```
pages > ts text-box-page.ts > ...
1 import { Locator } from '@playwright/test';
2 import { BasePage } from './base-page';
3 import { LeftPanalComponent } from './left-panal-component';
4
5 export class TextBoxPage extends BasePage {
6
7   private readonly _inputUserName: Locator = this.page.locator("id=userName");
8   private readonly _inputUserEmail: Locator = this.page.locator("id=userEmail");
9   private readonly _inputCurrentAddress: Locator = this.page.locator("id=currentAddress");
10  private readonly _inputPermanentAddress: Locator = this.page.locator("id=permanentAddress");
11  private readonly _submitButton: Locator = this.page.locator("id=submit");
12  private readonly _textResultUserName: Locator = this.page.locator("id=name");
13  private readonly _textResultUserEmail: Locator = this.page.locator("id=email");
14  private readonly _textResultCurrentAddress: Locator = this.page.locator("id=currentAddress");
15  private readonly _textResultPermanentAddress: Locator = this.page.locator("id=permanentAddress");
16
17  public readonly leftPanal = new LeftPanalComponent(this.page, this.page.locator("css=div.left-pannel"));
18
19  public async fillUserName(text: string) {
20    await this._inputUserName.fill(text);
21  }
22
23  > public async fillUserEmail(text: string) {
24  }
25
26
27  > public async fillCurrentAddress(text: string) {
28  }
29
30
31  > public async fillPermanentAddress(text: string) {
32  }
33
34
35  public async submitForm() {
36    await this._submitButton.click();
37  }
38
39  public async getUserName(): Promise<String> {
40    const isVisible = await this._textResultUserName.isVisible();
41    if (isVisible) {
42      const text = await this._textResultUserName.innerText();
43      return text;
44    }
45    return "";
46  }
47
48  > public async getUserEmail(): Promise<String> {
```

```
19   ...     public async fillUserName(text: string) {
20   ...       await this._inputUserName.fill(text);
21   ...
22
23   35     public async submitForm() {
24   ...       await this._submitButton.click();
25   ...
26
27   39     public async getUserName(): Promise<String> {
28   ...       const isVisible = await this._textResultUserName.isVisible();
29   ...       if (isVisible) {
30   ...         const text = await this._textResultUserName.innerText();
31   ...         return text;
32   ...
33   ...
34   ...
35   ...
36   ...
37   ...
38   ...
39   ...
40   ...
41   ...
42   ...
43   ...
44   ...
45   ...
46   ...
47   ...
48   ... }
```

Test steps:

Line39 Create async function name "getUserName" that will result result as string type

Line40 Check visible of locator "_textResultUserName" then assign to variable "isVisible"

Line41-44 If "isVisible" is True, will get "_textResultUserName" text and assign to variable "text" then return "text" as string

Line45 Else, return empty string

LeftPanalComponent

```
pages > ts left-panel-component.ts > ...
1 import { Locator } from '@playwright/test';
2 import { BaseComponent } from './base-component';
3
4 export class LeftPanalComponent extends BaseComponent {
5     // Group of Elements
6     private readonly _elements: Locator = this.elementLocator.locator("css=div.header-text >> text='Elements'");
7     private readonly _textBox: Locator = this.elementLocator.locator("css=span.text >> text='Text Box'");
8     private readonly _checkBox: Locator = this.elementLocator.locator("css=span.text >> text='Check Box'");
9     private readonly _webTable: Locator = this.elementLocator.locator("css=span.text >> text='Web Tables'");
10    // ...
11
12    // Group of Widgets
13    private readonly _widgets: Locator = this.elementLocator.locator("css=div.header-text >> text='Widgets'");
14    private readonly _selectMenu: Locator = this.elementLocator.locator("css=span.text >> text='Select Menu'");
15    // ...
16
17    public async goToTextBox() {
18        const isVisible = await this._textBox.isVisible();
19        if (isVisible) {
20            await this._textBox.click();
21        } else {
22            await this._elements.click();
23            await this._textBox.click();
24        }
25    }
26
27 }
28
29 > ... public async goToCheckBox() ...
30 > ... public async goToWebTable() ...
31 > ... public async goToSelectMenu() ...
32 }
```

In this case, the constructor function is automatically invoked and receives the required parameters (page, elementLocator) when creating an object of the "LeftPanelComponent" class.

This is facilitated by the super keyword, which ensures that the constructor of the superclass "BaseComponent" is appropriately called with the provided parameters, maintaining the necessary initialization for the inherited class

Test steps:

- Line4 Create class name "LeftPanalComponent" and inherit class properties from class "BaseComponent"
- Line6-15 Declare private property and fixed it as readonly
- Line17 Create async function name "goToTextBox"
- Line18 Check visible of locator "_textBox" then assign to variable "isVisible"
- Line19-21 If "isVisible" is True, will click at "_textBox"
- Line19-21 Else, click at "_elements" first, then click at "_textBox"

Test case: Get row detail from table

```
tests > TS test-suite-1.spec.ts > ...
1 import { expect, test } from '@playwright/test';
2 import { MainPage } from '../pages/main-page';
3 import { WebTablesPage } from '../pages/web-tables-page';
4
5 test("Get row detail from table", async ({ page }) => {
6
7   await page.goto("https://demoqa.com");
8
9   const mainPage = new MainPage(page);
10  await mainPage.goToElements();
11
12  const webTablesPage = new WebTablesPage(page);
13  await webTablesPage.leftPanal.goToWebTable();
14
15  const firstNameList = await webTablesPage.getFirstNameList();
16  expect(firstNameList).toContain("Alden");
17
18 });
19 |
```

Test steps:

- Line7 Let page go to url "<https://demoqa.com>"
- Line9 Create new object of MainPage by passing one parameter "page" as required
- Line10 With MainPage object, call function "goToElements"
- Line12 Create new object of WebTablesPage by passing one parameter "page" as required
- Line13 With WebTablesPage object, refer to property name "leftPanal" and call function "goToWebTable" from leftPanal
- Line13 With WebTablesPage object, call function name "getFirstNameList" that will return array of string then assign value to variable name "firstNameList"
- Line19 Assert value of firstNameList must contain the expected value "Alden"

WebTablesPage

```
pages > TS web-tables-page.ts > ...
1 import { Locator } from "@playwright/test";
2 import { BasePage } from "./base-page";
3 import { LeftPanalComponent } from "./left-panal-component";
4 import { EmployeeRowElement } from "./employee-row-element";
5
6 export class WebTablesPage extends BasePage {
7
8     private readonly _rows: Locator = this.page.locator(`css=div.rt-tbody >> xpath=/div[@role='row' .
9     and not(contains(@class, '-padRow'))]`);
10
11    public readonly leftPanal = new LeftPanalComponent(this.page, this.page.locator("css=div.left-pannel"));
12
13    public async getAllRowsElement() {
14
15        const rowCount = await this._rows.count();
16        const rowElements = new Array<EmployeeRowElement>();
17        for (let i = 0; i < rowCount; i++) {
18            rowElements.push(new EmployeeRowElement(this.page, this._rows.nth(i)));
19        }
20
21        return rowElements;
22    }
23
24    public async getFirstNameList() {
25        const rowElements = await this.getAllRowsElement();
26        const firstNameList = new Array<String>();
27        for await (const row of rowElements) {
28            const fName = await row.getFirstName();
29            firstNameList.push(fName);
30        }
31
32        return firstNameList;
33    }
34
35 }
36
```

```
13     public async getAllRowsElement() {
14
15         const rowCount = await this._rows.count();
16         const rowElements = new Array<EmployeeRowElement>();
17         for (let i = 0; i < rowCount; i++) {
18             rowElements.push(new EmployeeRowElement(this.page, this._rows.nth(i)));
19         }
20
21         return rowElements;
22     }
23
```

Test steps:

Line13 Create async function name
"getAllRowsElement"

Line15 Count locator match with _rows

Line16 Create new array of object
"EmployeeRowElement" assign to variable
rowElements

Line17 For loop following rowCount
i = 0 -> rowCount

Line18 Collect object of EmployeeRowElement to
rowElements

Line21 Return rowElements

WebTablesPage

```
pages > TS web-tables-page.ts > ...
1 import { Locator } from '@playwright/test';
2 import { BasePage } from './base-page';
3 import { LeftPanalComponent } from './left-panal-component';
4 import { EmployeeRowElement } from './employee-row-element';
5
6 export class WebTablesPage extends BasePage {
7
8     private readonly _rows: Locator = this.page.locator(`css=div.rt-tbody >> xpath=/div[@role='row']
9     and not(contains(@class, '-padRow'))]`);
10
11    public readonly leftPanal = new LeftPanalComponent(this.page, this.page.locator("css=div.left-pannel"));
12
13    public async getAllRowsElement() {
14
15        const rowCount = await this._rows.count();
16        const rowElements = new Array<EmployeeRowElement>();
17        for (let i = 0; i < rowCount; i++) {
18            rowElements.push(new EmployeeRowElement(this.page, this._rows.nth(i)));
19        }
20
21        return rowElements;
22    }
23
24    public async getFirstNameList() {
25        const rowElements = await this.getAllRowsElement();
26        const firstNameList = new Array<String>();
27        for await (const row of rowElements) {
28            const fName = await row.getFirstName();
29            firstNameList.push(fName);
30        }
31
32        return firstNameList;
33    }
34
35 }
36
```

```
24     public async getFirstNameList() {
25         const rowElements = await this.getAllRowsElement();
26         const firstNameList = new Array<String>();
27         for await (const row of rowElements) {
28             const fName = await row.getFirstName();
29             firstNameList.push(fName);
30         }
31
32         return firstNameList;
33     }
34
35 }
```

Test steps:

- Line13 Create async function name "getFirstNameList"
- Line15 Call function getAllRowsElement and assign return value to variable rowElement
- Line16 Create new array of String assign to variable firstNameList
- Line18 For loop in array rowElements
- Line18 As a row that represent to object of EmployeeRowElement, call function "getFirstName" and assign return value to variable fName
- Line18 Collect string from fName variable to array firstNameList
- Line18 Return firstNameList

EmployeeRowElement

```
pages > ts employee-row-element.ts > ...
1 import { Locator } from '@playwright/test';
2 import { BaseComponent } from './base-component';
3
4 export class EmployeeRowElement extends BaseComponent {
5
6   ... private readonly _fName: Locator = this.elementLocator.locator("xpath=/div[@role='gridcell'][1]");
7   ... private readonly _lName: Locator = this.elementLocator.locator("xpath=/div[@role='gridcell'][2]");
8   ... private readonly _age: Locator = this.elementLocator.locator("xpath=/div[@role='gridcell'][3]");
9   ... private readonly _email: Locator = this.elementLocator.locator("xpath=/div[@role='gridcell'][4]");
10  ... private readonly _salary: Locator = this.elementLocator.locator("xpath=/div[@role='gridcell'][5]");
11  ... private readonly _department: Locator = this.elementLocator.locator("xpath=/div[@role='gridcell'][6]");
12  ... private readonly _editButton: Locator = this.elementLocator.locator("xpath=/div[@role='gridcell'][7]");
13  ... private readonly _deleteButton: Locator = this.elementLocator.locator("xpath=/div[@role='gridcell'][8]");
14
15  ... public async getFirstName() {
16    ... const text = await this._fName.innerText();
17    ... return text;
18  }
19
20  > ... public async getLastname() { ... }
21  ...
22
23  > ... public async getAge() { ... }
24  ...
25
26  > ... public async getEmail() { ... }
27  ...
28
29  > ... public async getSalary() { ... }
30  ...
31
32  > ... public async getDepartment() { ... }
33  ...
34
35  > ... public async edit() { ... }
36  ...
37
38  > ... public async delete() {
39    ... await this._deleteButton.click();
40  }
41
42
43
44
45  > ... public async edit() { ... }
46  ...
47
48
49  > ... public async delete() {
50    ... await this._deleteButton.click();
51  }
52
53
54 }
```

Test steps:

Line4 Create class name "EmployeeRowElement" and inherit class properties from class "BaseComponent"

Line6-13 Declare private property and fixed it as readonly

Test steps:

Line15 Create async function name "getFirstName"

Line16 At locator _fName, get text and assign value to variable text

Line17 Return variable text

Test steps:

Line49 Create async function name "delete"

Line50 At locator _deleteButton, click it

BasePage & BaseComponent

```
pages > TS base-page.ts > ...
```

```
1 import { Page } from "@playwright/test";
2
3 export class BasePage {
4   ...
5   protected page: Page;
6
7   constructor(page: Page) {
8     ...
9     this.page = page;
10    ...
11 }
```

```
pages > TS base-component.ts > ...
```

```
1 import { Locator, Page } from "@playwright/test";
2
3 export class BaseComponent {
4   ...
5   protected page: Page;
6   ...
7   protected elementLocator: Locator;
8
9   constructor(page: Page, elementLocator: Locator) {
10    ...
11      this.page = page;
12      this.elementLocator = elementLocator;
13    ...
14 }
```

Object creation steps

```
tests > ts test-suite-1.spec.ts > ...
1 import { expect, test } from '@playwright/test';
2 import { MainPage } from '../pages/main-page';
3 import { TextBoxPage } from '../pages/text-box-page';
4
5 test("Fill in text box form", async ({ page }) => {
6   await page.goto("https://demoqa.com");
7
8   const mainPage = new MainPage(page);
9   await mainPage.goToElements();
10
11  const textBoxPage = new TextBoxPage(page);
12  await textBoxPage.leftPanal.goToTextBox();
13
14  await textBoxPage.fillUserName("Jojoe");
15  await textBoxPage.submitForm();
16
17  const resultName = await textBoxPage.getUserName();
18  expect(resultName).toEqual("Name:Jojoe");
19
20
21 });
22
```

1. In test scenario create new object of MainPage

2. When create object constructor will be called first

8. Return create object MainPage

4. Value passed from child class
5. Call constructor
6. Assign value to variable page

```
pages > TS main-page.ts > ...
1 import { Locator, Page } from '@playwright/test';
2 import { BasePage } from './base-page';
3
4 export class MainPage extends BasePage {
5
6   private readonly _cardElement: Locator = this.page.locator("xpath=/h5[text()='Element']");
7   private readonly _cardForms: Locator = this.page.locator("xpath=/h5[text()='Forms']");
8   private readonly _cardWidgets: Locator = this.page.locator("xpath=/h5[text()='Widgets']");
9   private readonly _cardAlertsFrameWindows: Locator = this.page.locator("xpath=/h5[text()='Alerts/Frame/Windows']");
10  private readonly _cardInteractions: Locator = this.page.locator("xpath=/h5[text()='Interactions']");
11  private readonly _cardBookStoreApplication: Locator = this.page.locator("xpath=/h5[text()='Book Store Application']");
12
13  constructor(page: Page) {
14    super(page);
15  }
16
17  public async goToElements() {
18    await this._cardElement.click();
19  }
20}
```

3. Pass value to super class

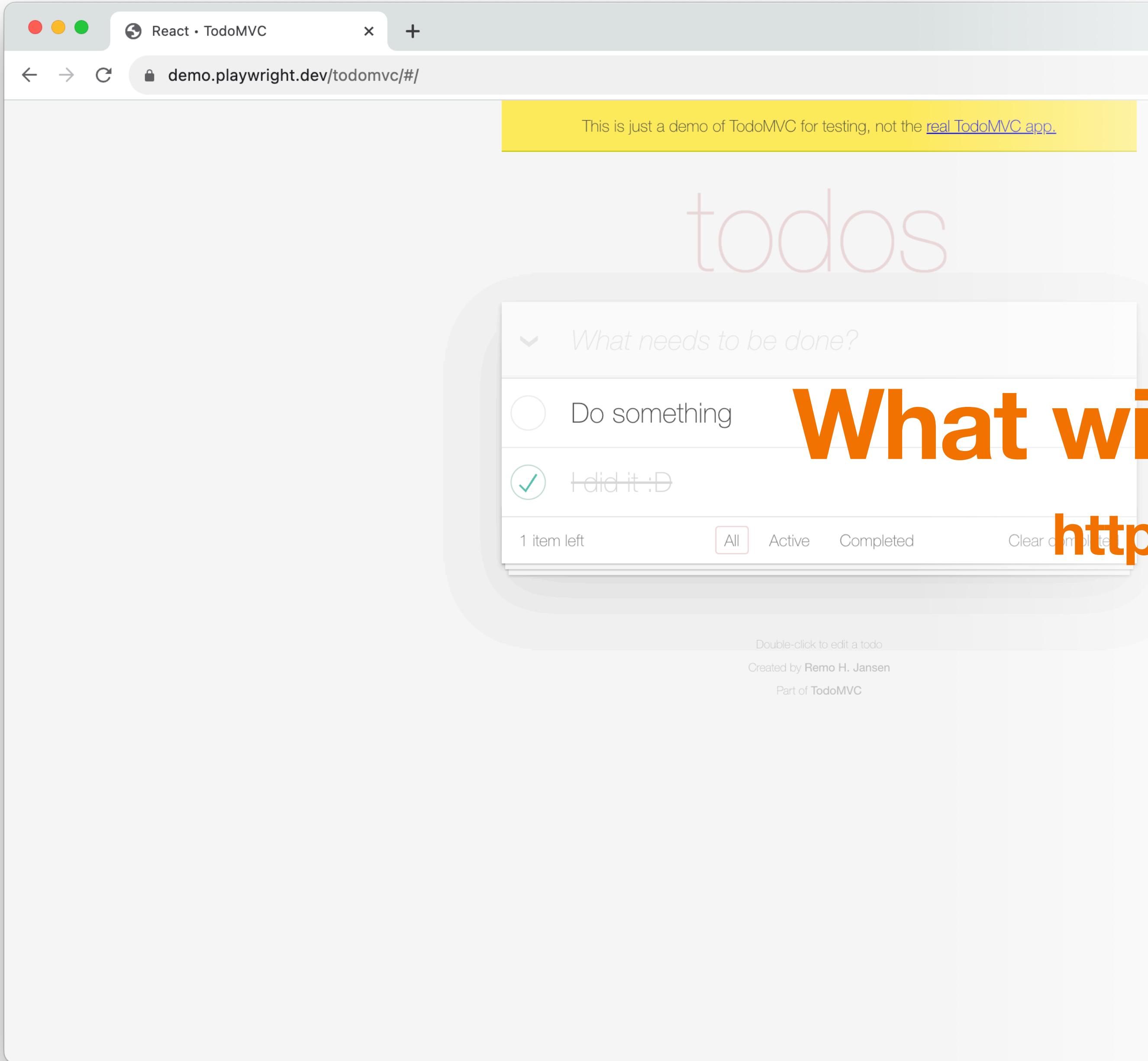
```
pages > TS base-page.ts > ...
1 import { Page } from '@playwright/test';
2
3 export class BasePage {
4   protected page: Page;
5
6   constructor(page: Page) {
7     this.page = page;
8   }
9
10 }
11
```

7. this.page is a property of super class BasePage inherited to MainPage class



Don't be afraid
Now we are friend

Playwright Workshop



**As a QA,
What will you do with this site
<https://demo.playwright.dev/todomvc/#/>**

Now it's workshop time

Do anything you want :)

- Go to this
<https://demo.playwright.dev/todomvc/#/>
- Click it
- Fill it
- Check it
- Bla bla bla ~

LEVEL UP ↑

You are unlocked a new skill
End-to-End Automated Testing with Playwright