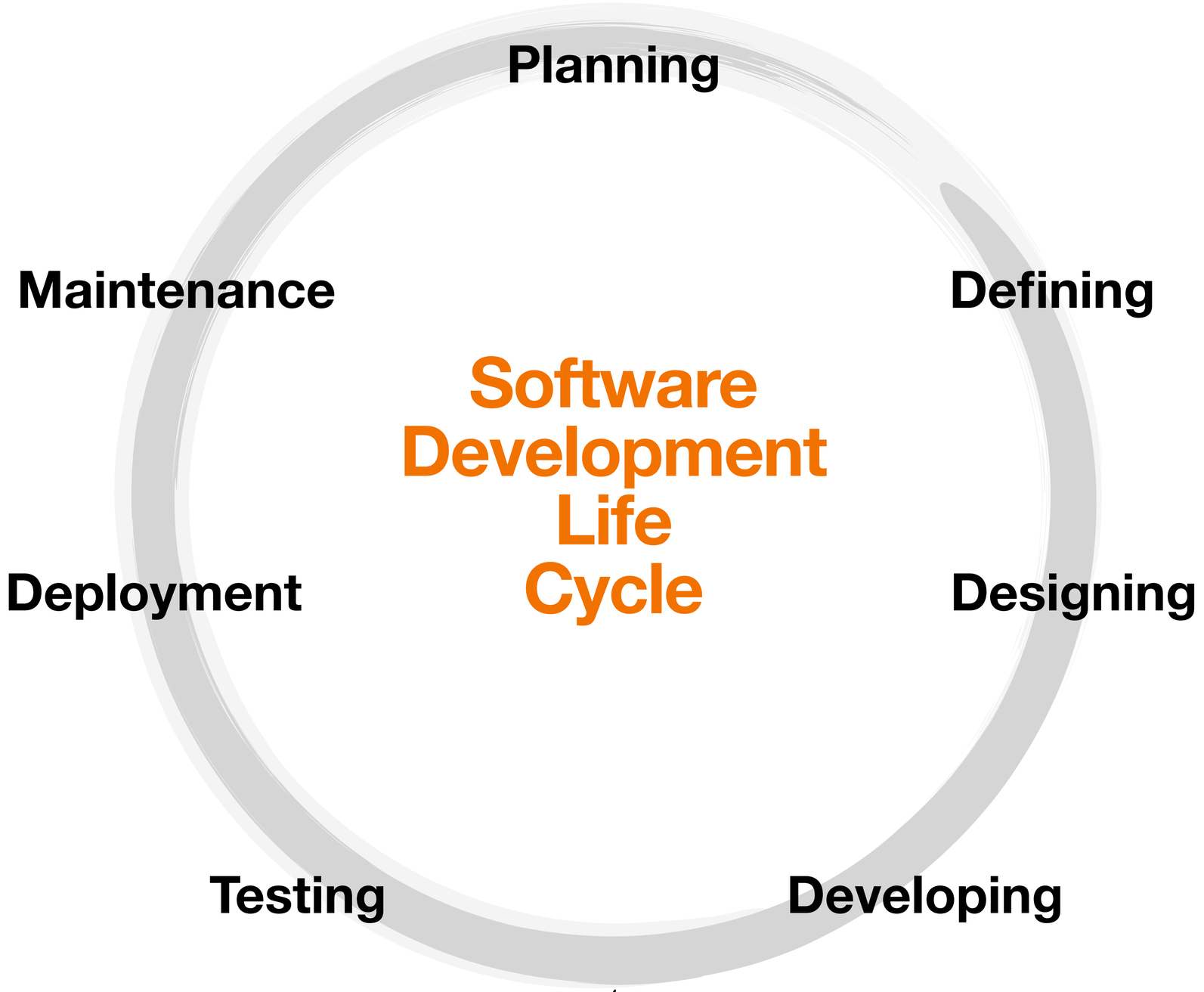
Playwright 101

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
Designing
Developing
Testing
Deployment
Maintenance
Software Development Life Cycle

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.

Ref: https://www.tutorialspoint.com/sdlc/sdlc_overview.htm

Software Development Life Cycle

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.

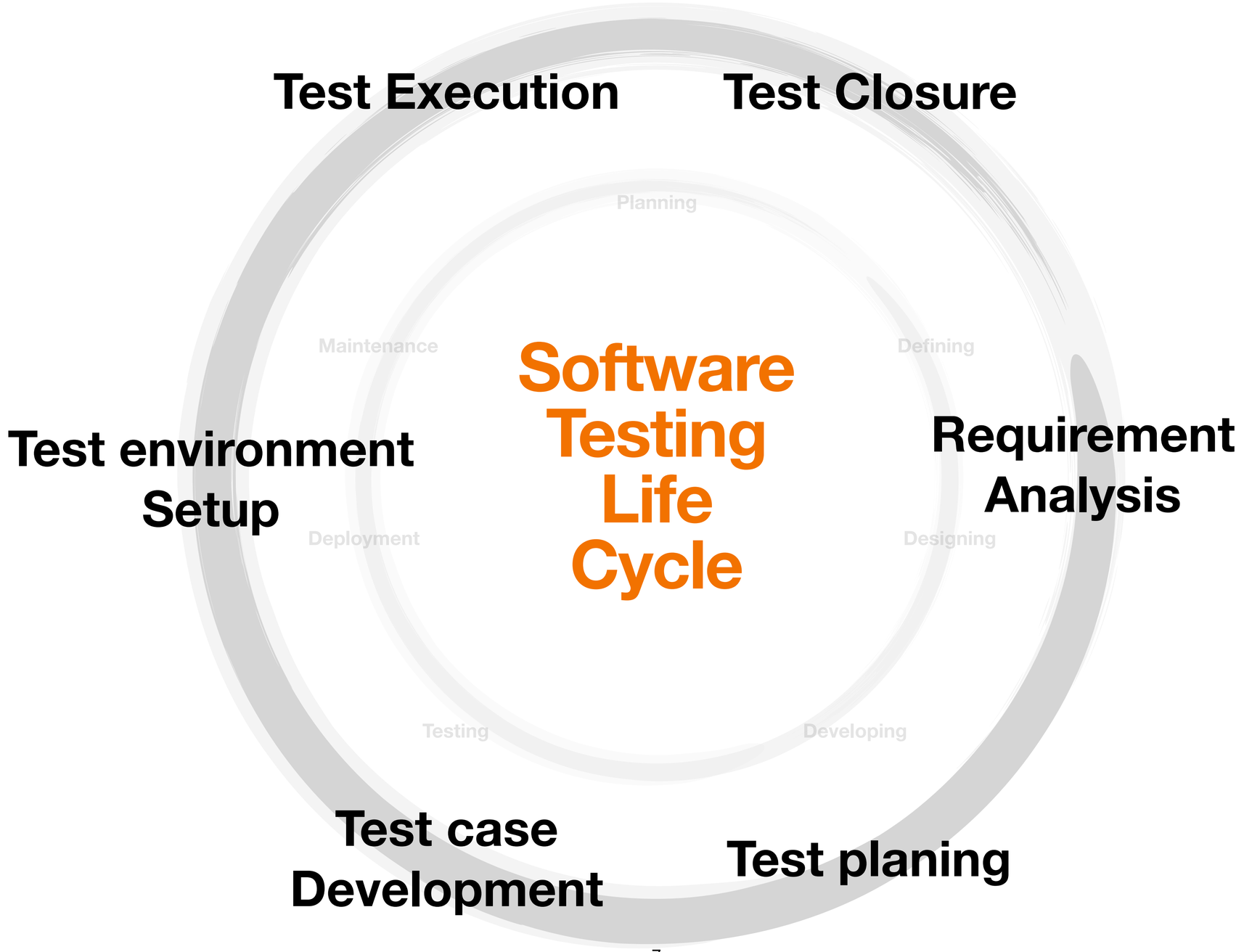
Deployment

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.**

Ref: https://www.tutorialspoint.com/sdlc/sdlc_overview.htm


Test Execution
Test Closure
Planning
Maintenance
Defining
Software Testing Life Cycle
Test environment Setup
Requirement Analysis
Deployment
Designing
Testing
Developing
Test case Development
Test planing

Software Testing Life Cycle

Requirement Analysis

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.

Test planing

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.

Test case Development

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.

Ref: https://www.geeksforgeeks.org/software-testing-life-cycle-stlc/

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

Test Execution

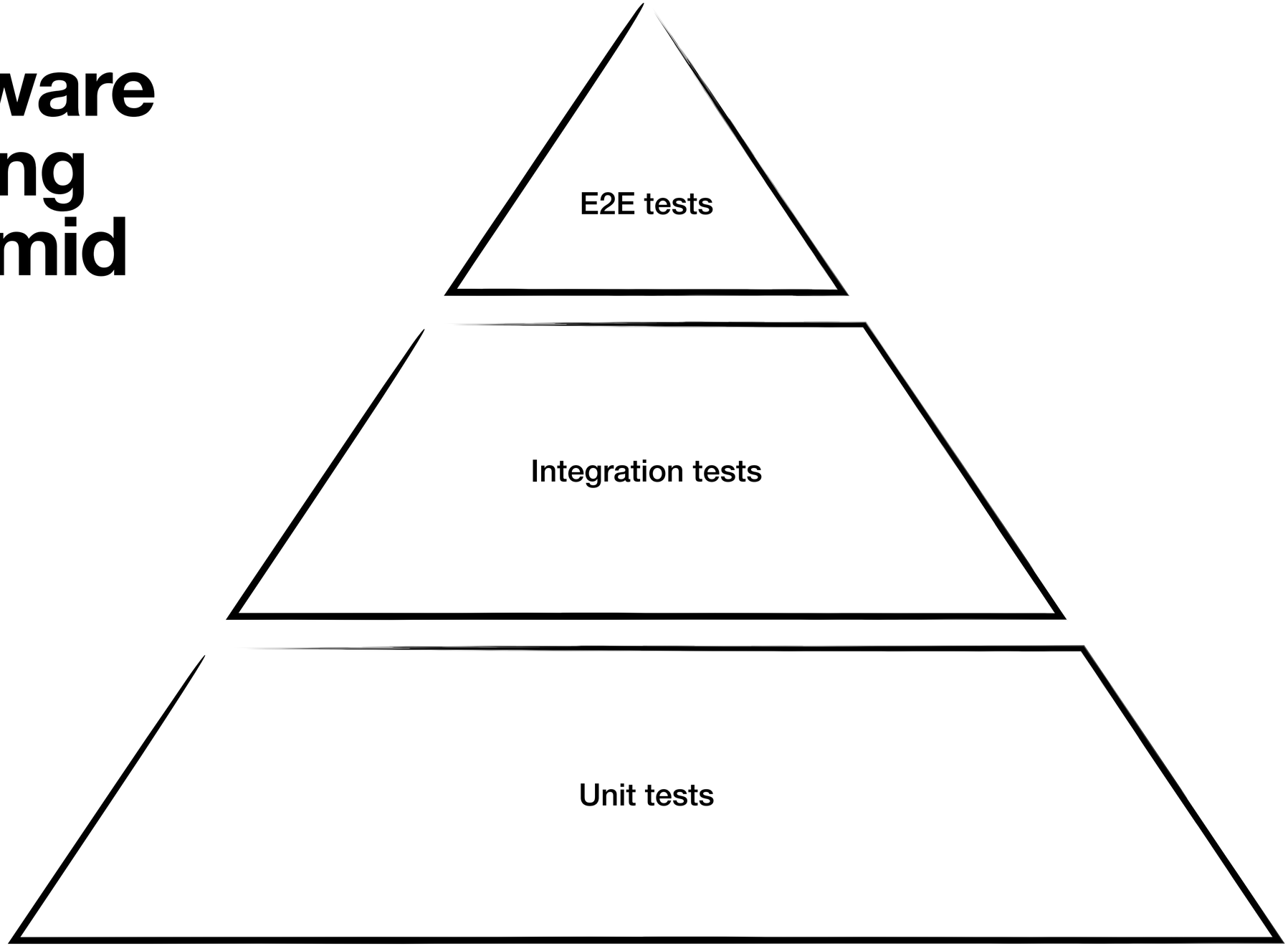
In this phase testing team **starts executing test cases based on prepared test cases** in the earlier step.

Test environment Setup

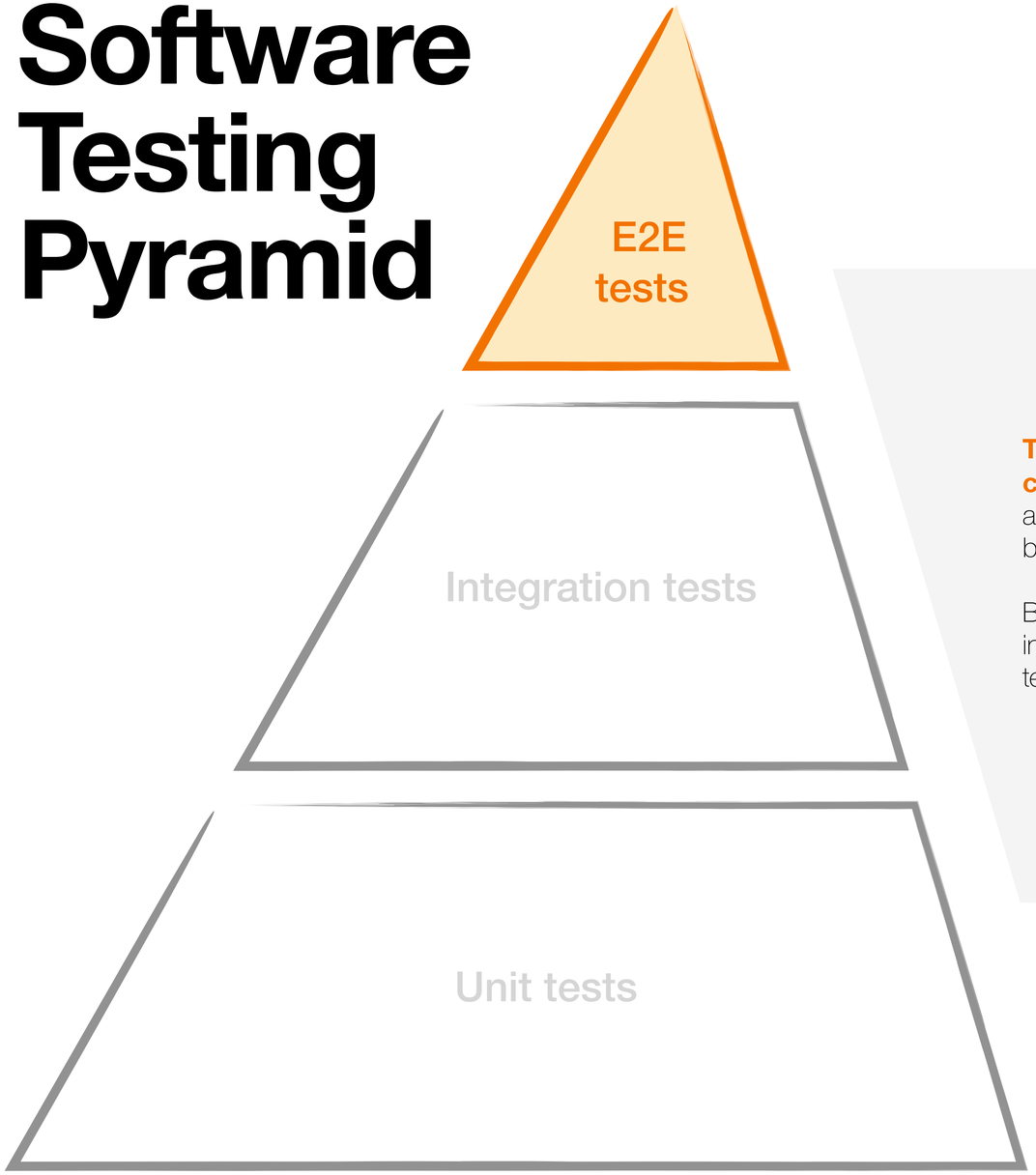
Basically, the test environment decides the conditions on which software is tested. This is independent activity and can be started along with test case development. In this process, the testing team is not involved.
Either the developer or the customer creates the testing environment.

Ref: https://www.geeksforgeeks.org/software-testing-life-cycle-stlc/

Software Testing Pyramid



Software Testing Pyramid

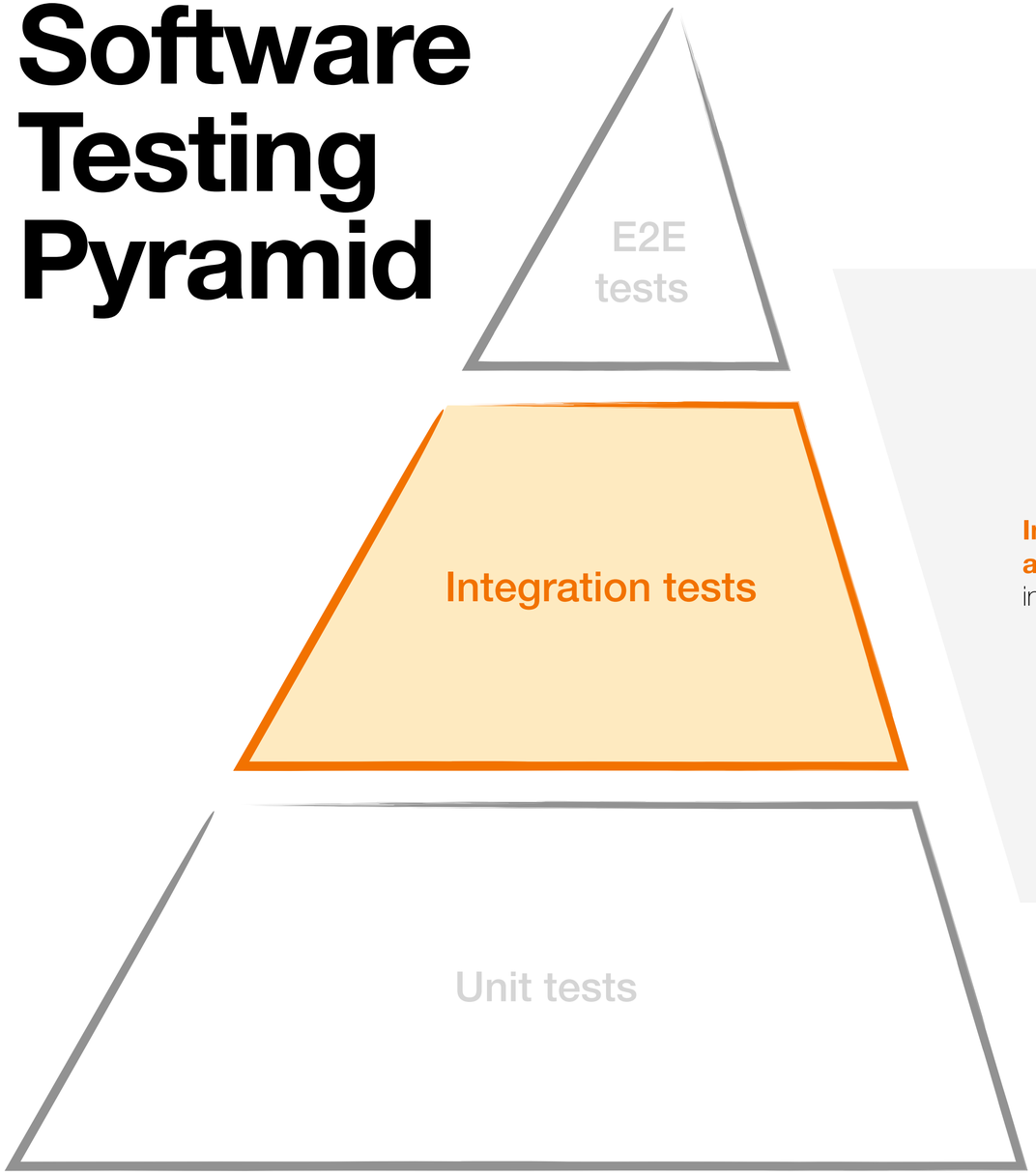


The aim of this type of End-to-End (E2E) testing is to evaluate whether the system complies with the end-user requirements and if it is ready for deployment. The scope of acceptance testing ranges from simply finding spelling mistakes and cosmetic errors, to uncovering bugs that could cause a major error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

Ref: https://worldofagile.com/blog/agile-test-pyramid/

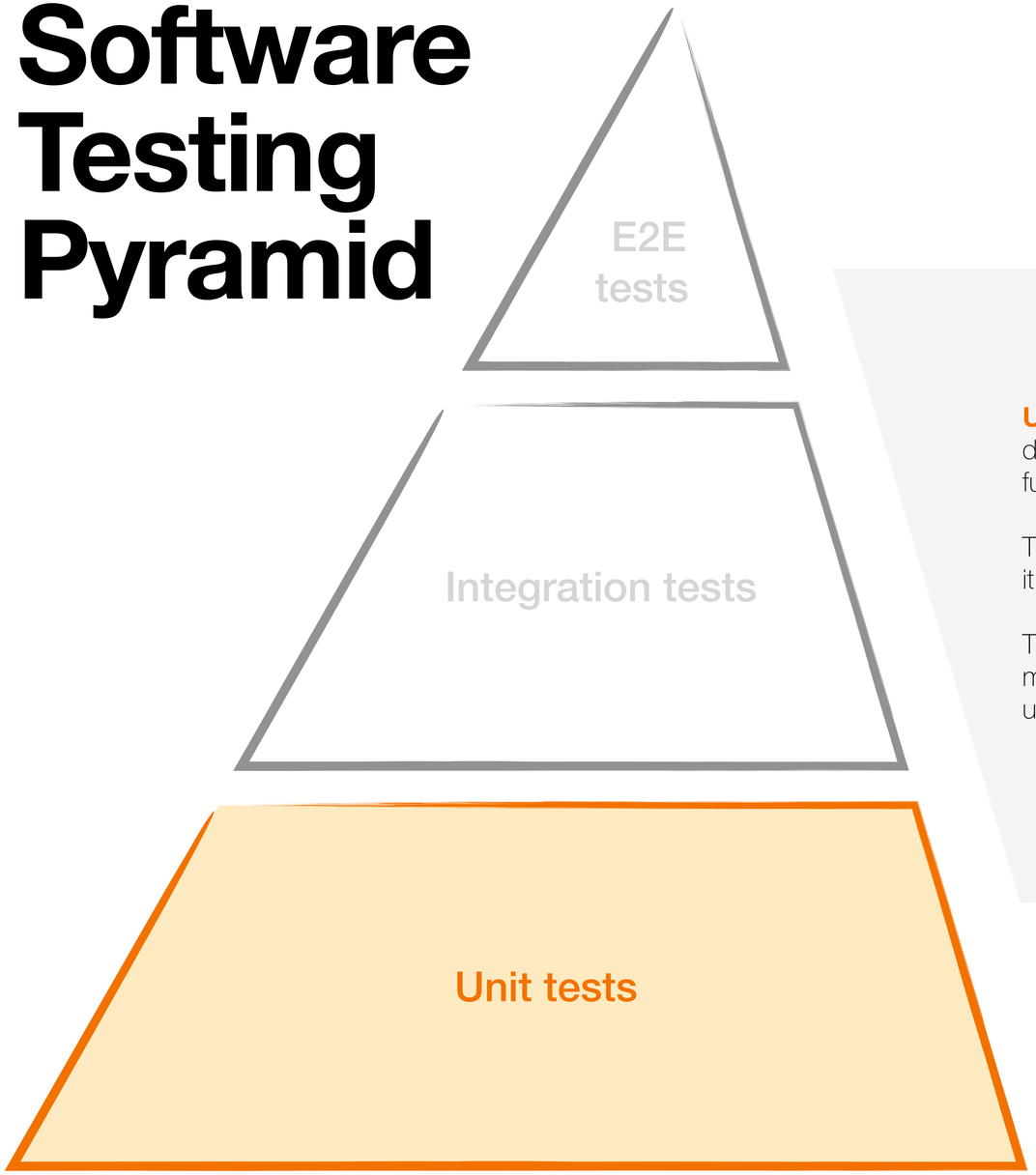
Software Testing Pyramid



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.

Ref: https://worldofagile.com/blog/agile-test-pyramid/

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.

Ref: https://worldofagile.com/blog/agile-test-pyramid/

End-to-End Tests

Strengths

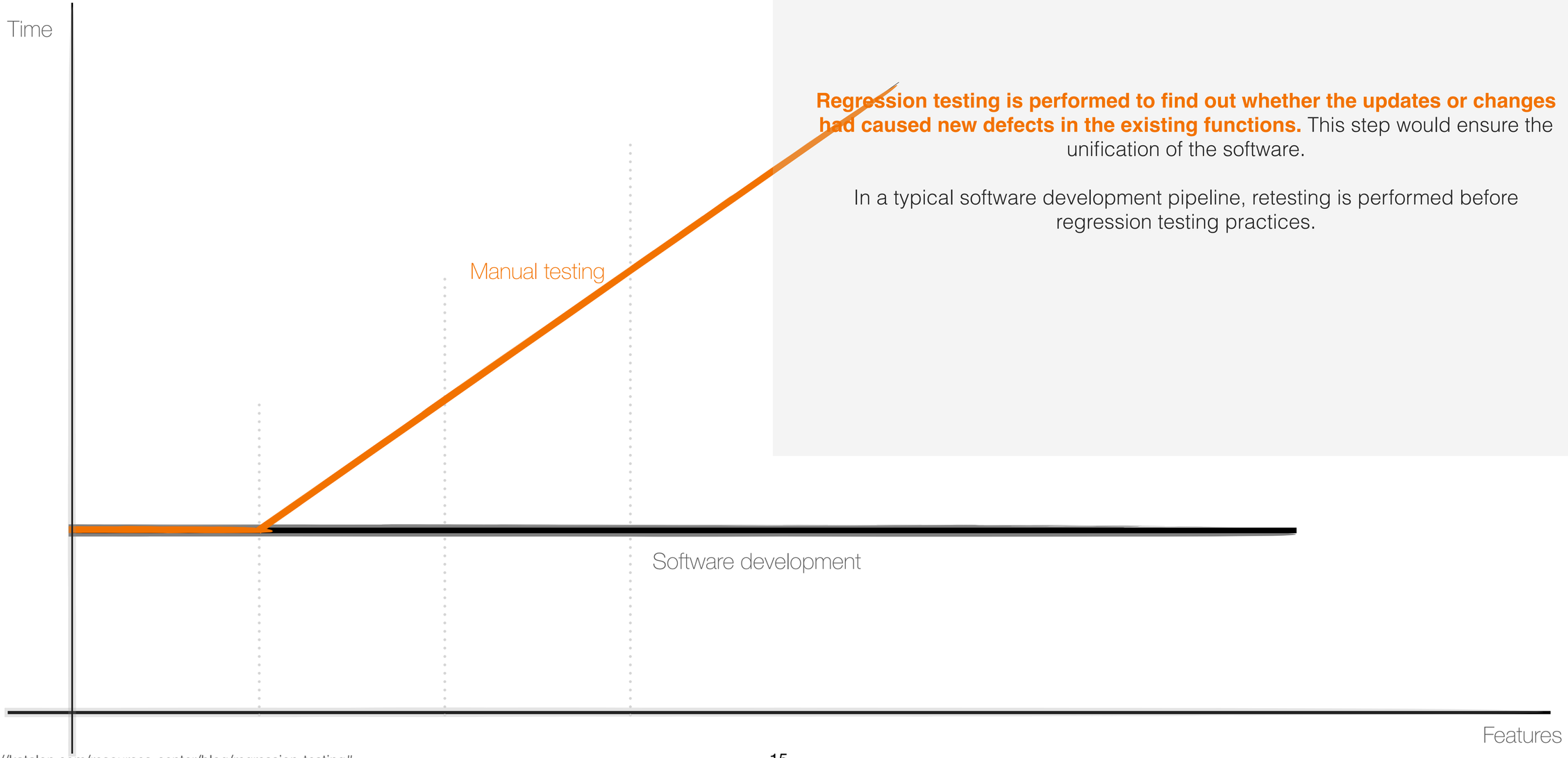
- **Realistic Testing**: E2E tests mimic real user interactions with the application, making them ideal for assessing how the software behaves in a production-like environment.
- **Comprehensive Coverage**: E2E tests can cover a wide range of functionalities, ensuring that the entire system functions 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.

Ref: "Software Testing: Principles and Practices", Srinivasan Desikan and Gopalaswamy Ramesh

Manual Testing

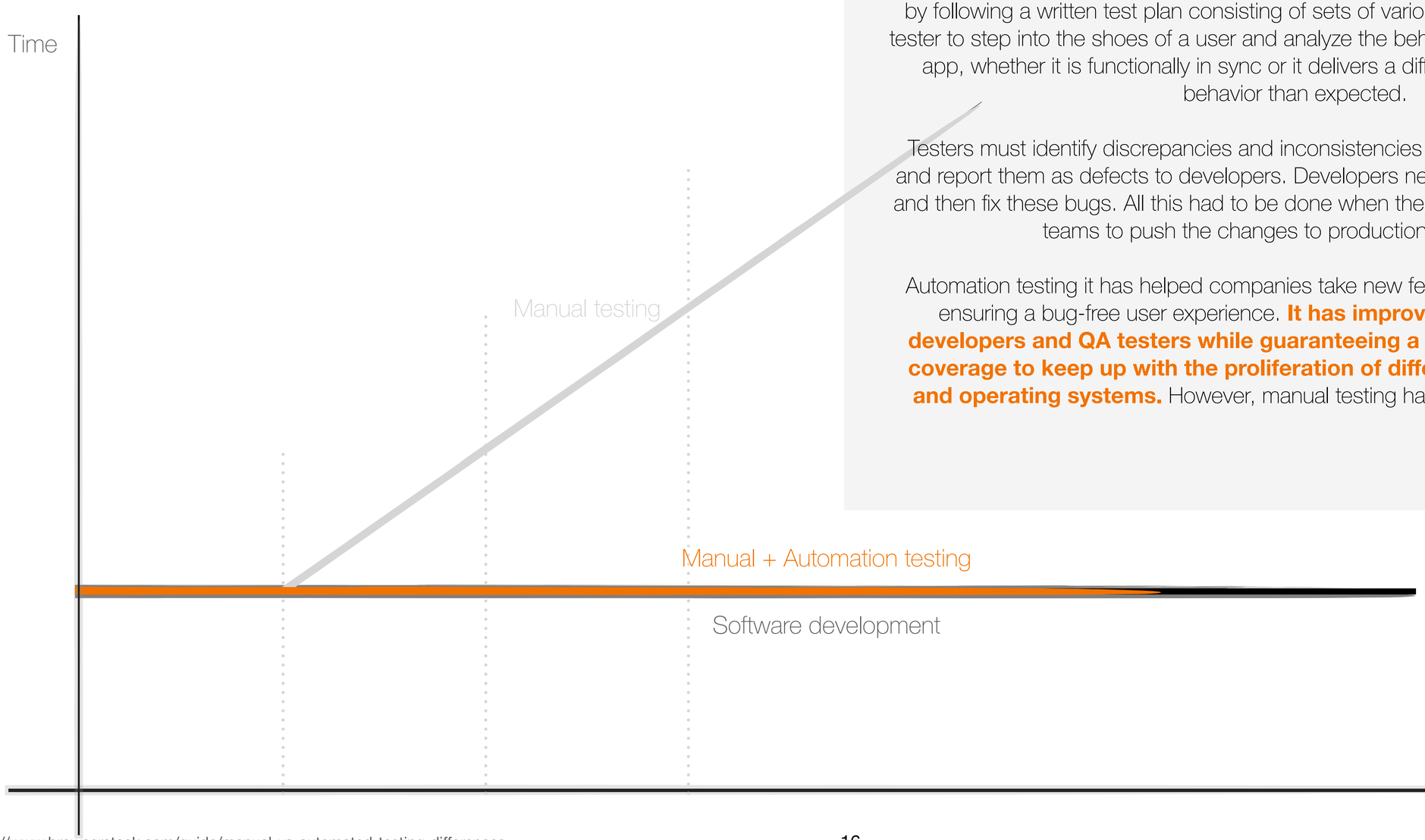


Regression testing is performed to find out whether the updates or changes had caused new defects in the existing functions. This step would ensure the unification of the software.

In a typical software development pipeline, retesting is performed before regression testing practices.

Ref: https://katalon.com/resources-center/blog/regression-testing#

Automation Testing



With traditional manual testing, the testers manually look out for defects in the software by following a written test plan consisting of sets of various test cases. It requires the tester to step into the shoes of a user and analyze the behavior of the website or mobile app, whether it is functionally in sync or it delivers a different and often erroneous behavior than expected.

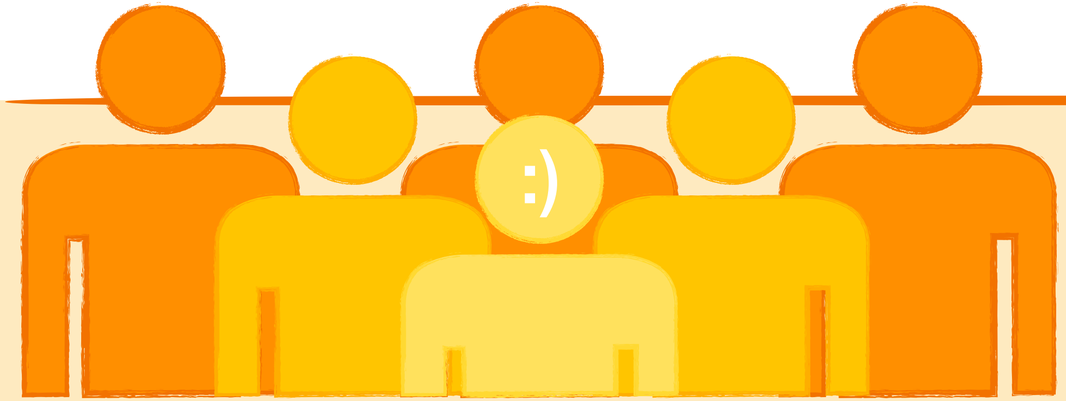
Testers must identify discrepancies and inconsistencies from the expected behavior and report them as defects to developers. Developers need to replicate the same bug and then fix these bugs. All this had to be done when there was pressure from multiple teams to push the changes to production immediately.

Automation testing it has helped companies take new features to market faster while ensuring a bug-free user experience. **It has improved the productivity of developers and QA testers while guaranteeing a massive increase in test coverage to keep up with the proliferation of different devices, browsers, and operating systems.** However, manual testing has retained its special place.

Ref: https://www.browserstack.com/guide/manual-vs-automated-testing-differences

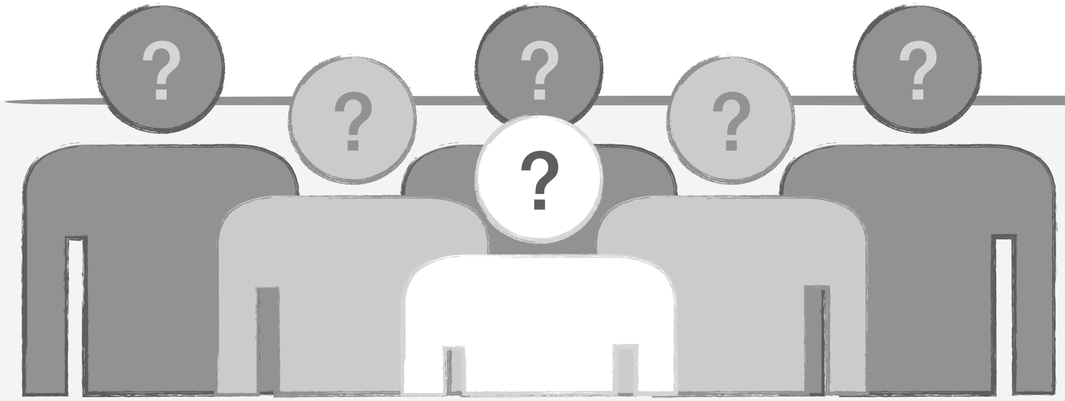
Tester

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.

Ref: https://www.guru99.com/alpha-beta-testing-demystified.html

Playwright

An open-source browser automation library developed by Microsoft.

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

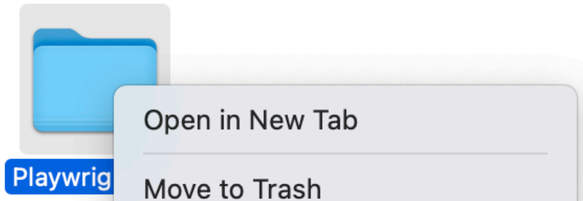
Playwright allows developers to automate browser tasks such as page navigation, form submission, and interaction with web elements.

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

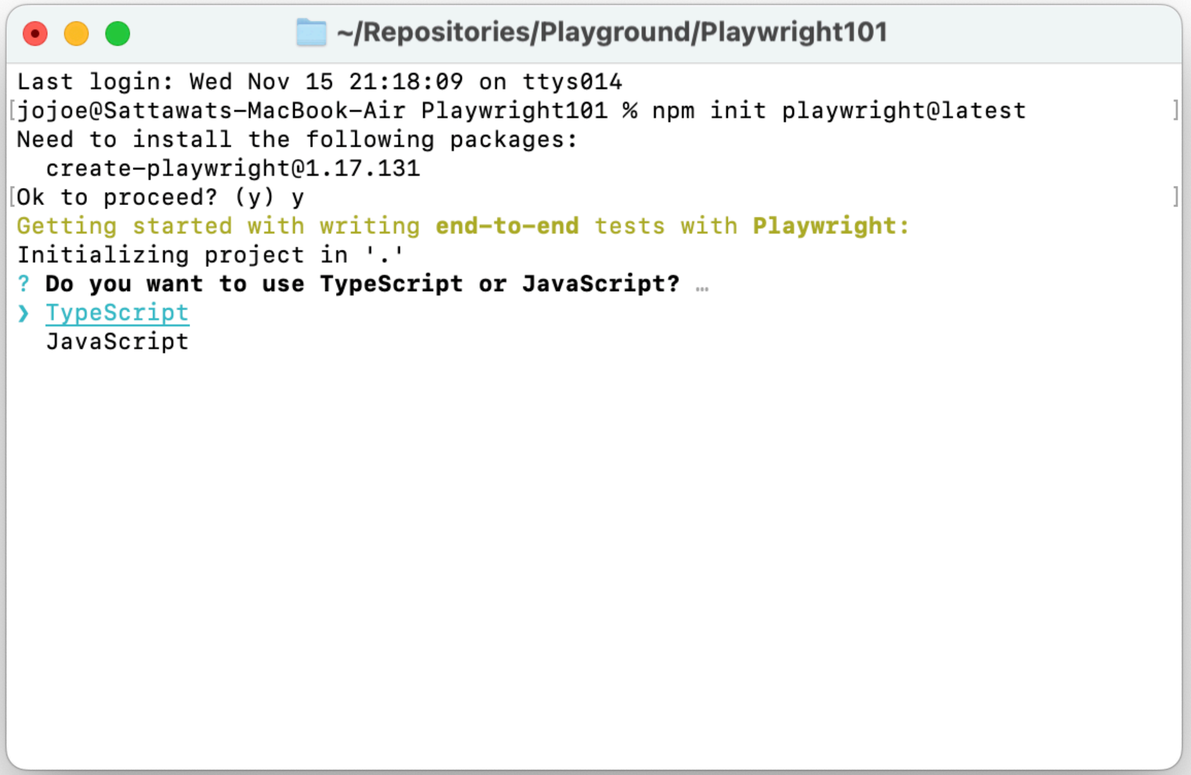
Ref1: "Robot Framework Test Automation" by Sumit Bisht and Manoj Kumar
Ref2: "Mastering Selenium Robot Framework" by Marko Kocic
Ref3: "Mastering Playwright" by Eran Peer
Ref4: "Cypress.io Practical Automation Testing For Everyone" by Filip Hric

Initial Playwright project



Run command:

```
npm init playwright@latest
```

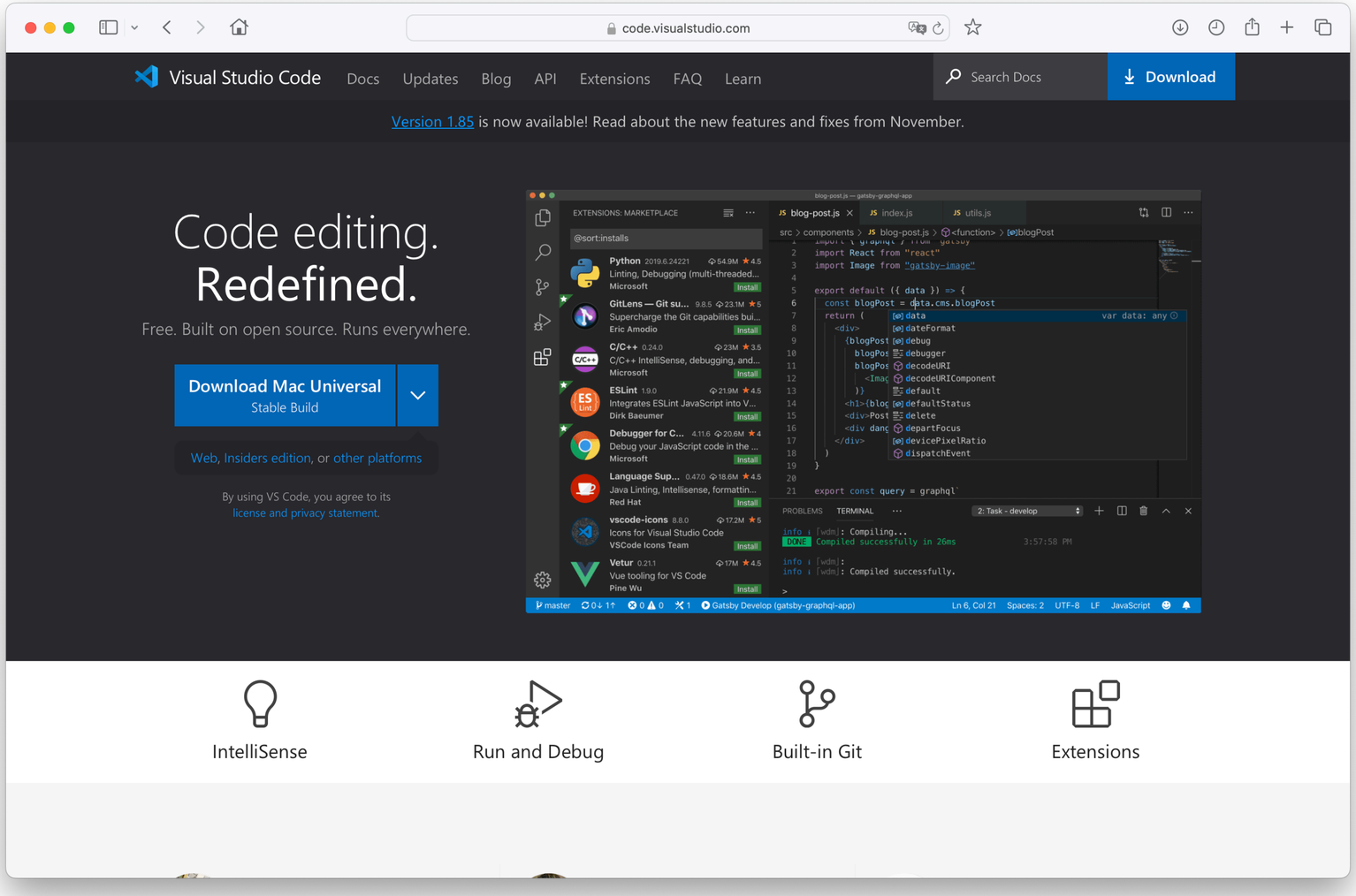


Installations

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

Install VSCode

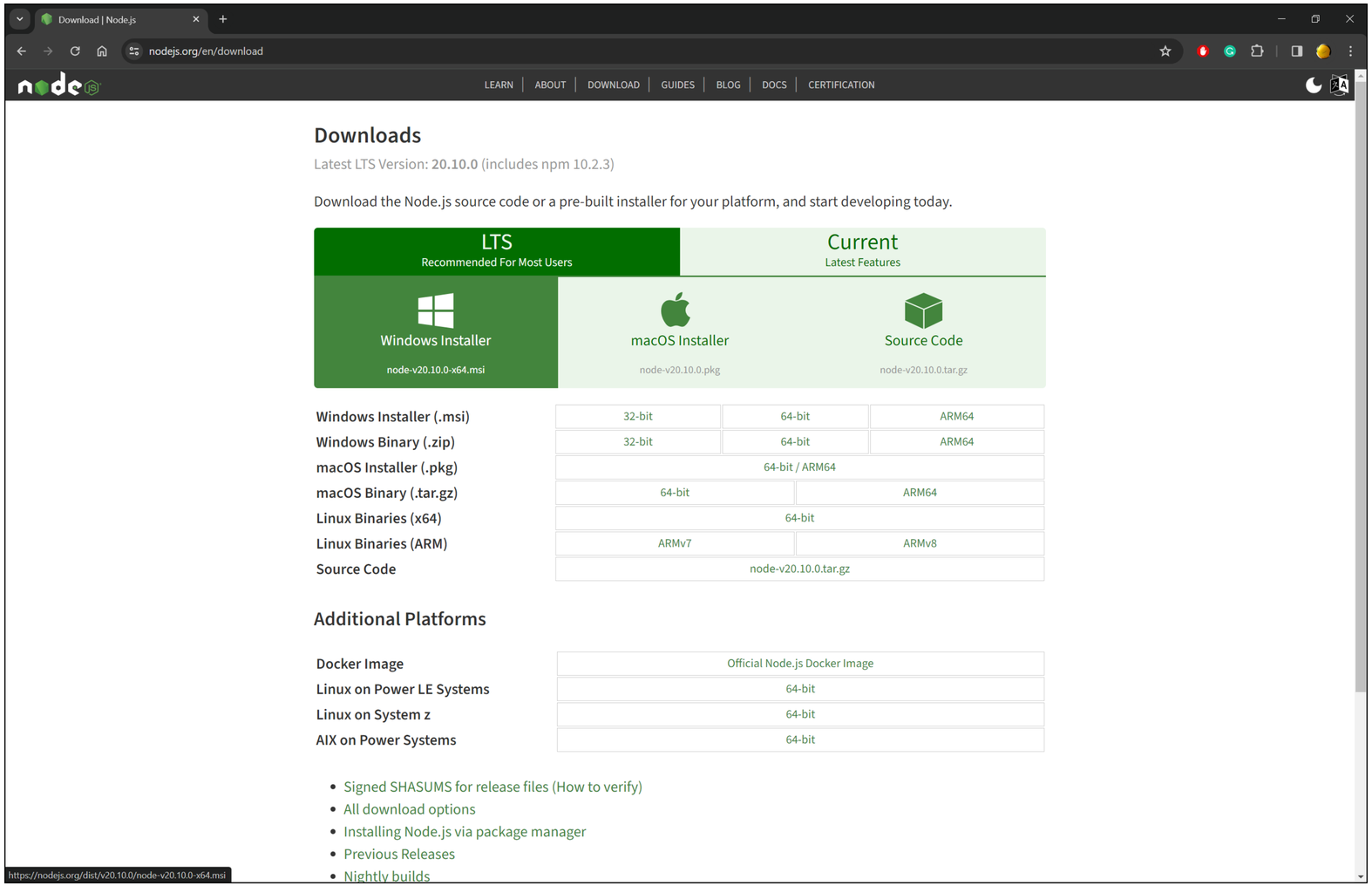
Go to this official website => https://code.visualstudio.com/
It has already suggested the compatible version with your environment



Ref1: https://code.visualstudio.com/

Windows: Download Node.js

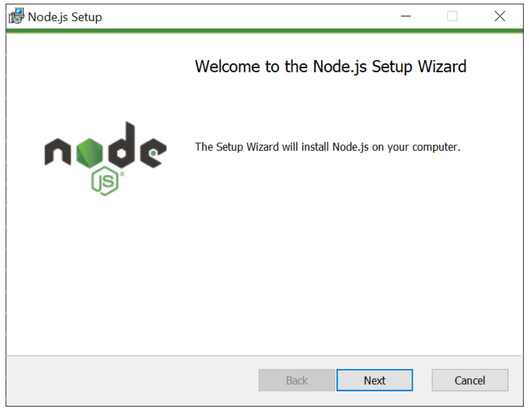
Go to this official website => https://nodejs.org/en/download , then download the Windows version

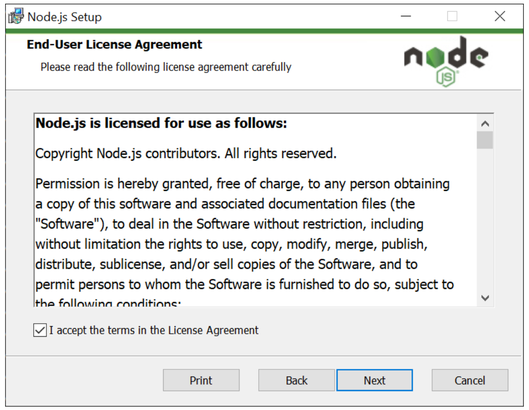


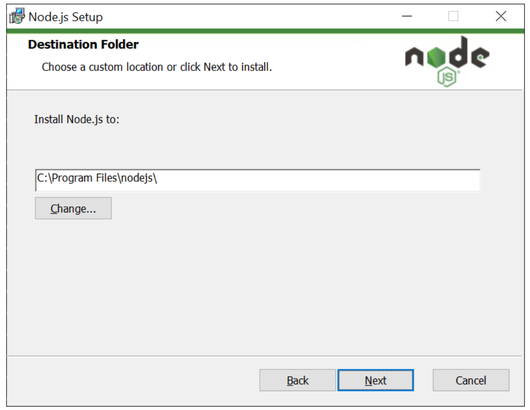
Ref1: https://nodejs.org/en/download

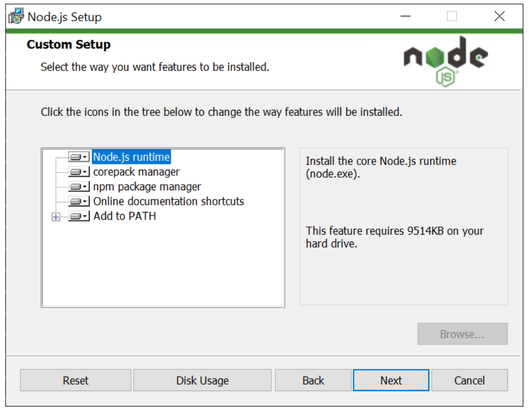
Windows: Install Node.js

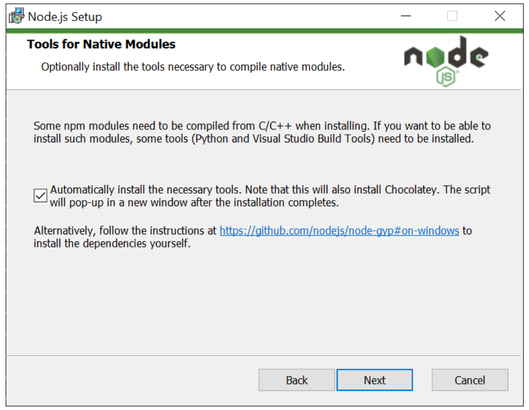
So easy! Next, Next, Next, and Finish.

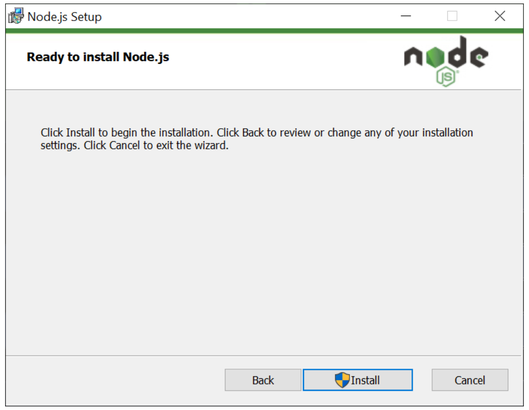


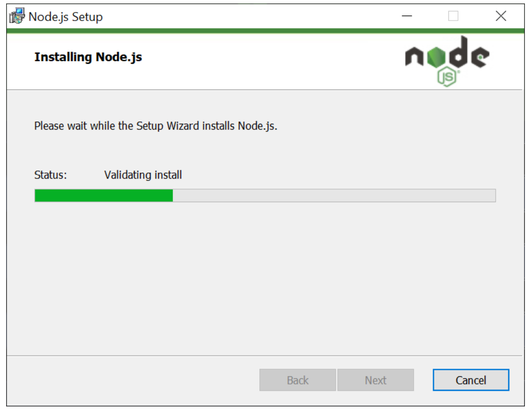


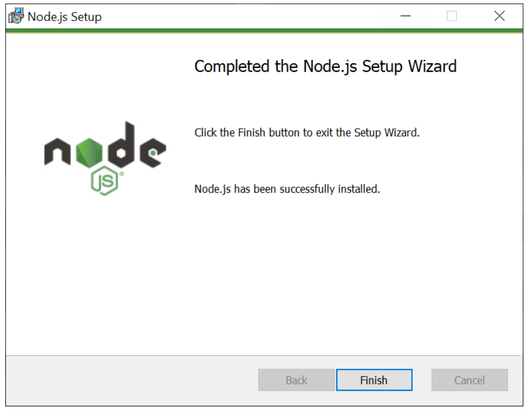






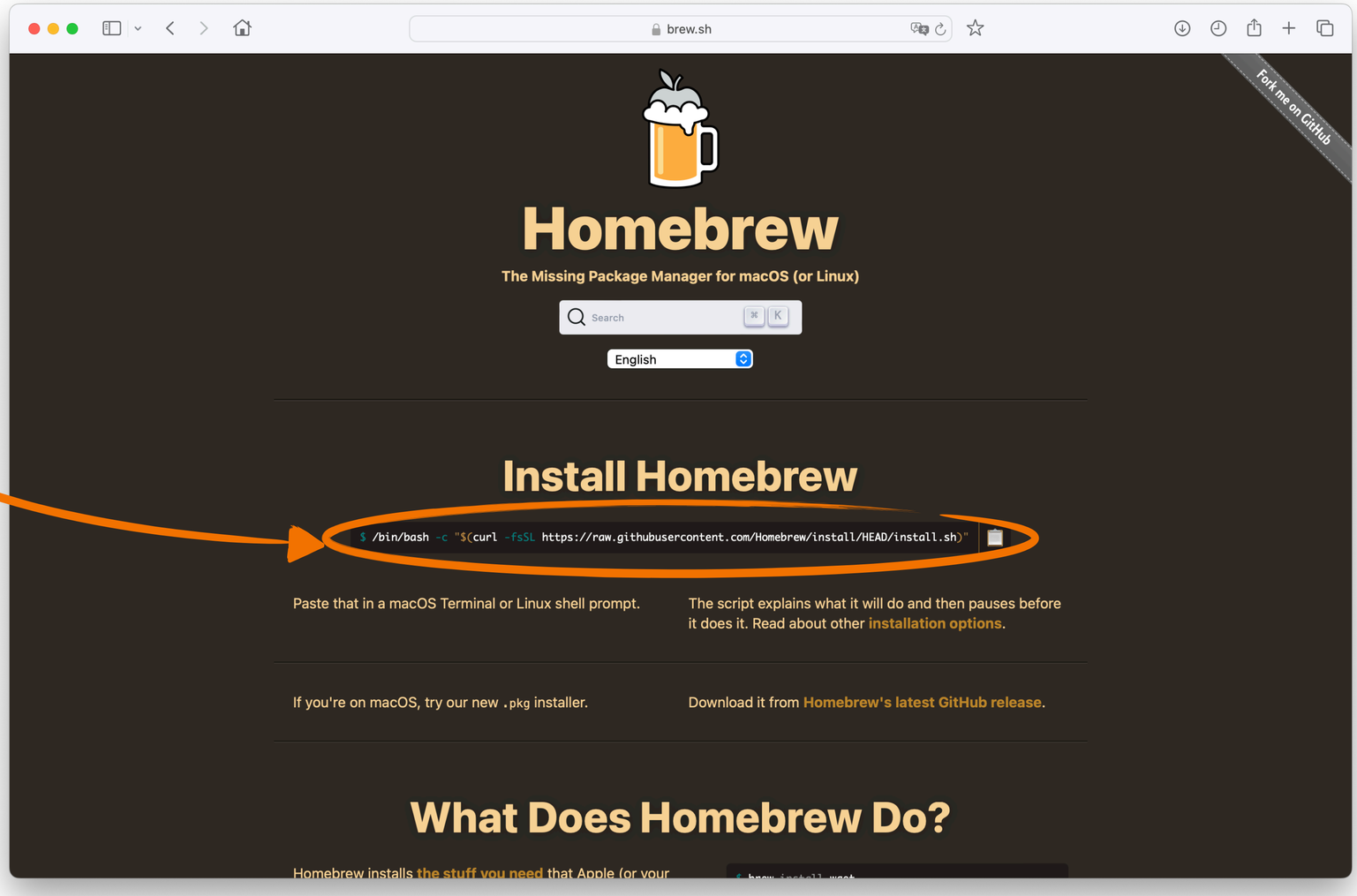






Mac: Homebrew website

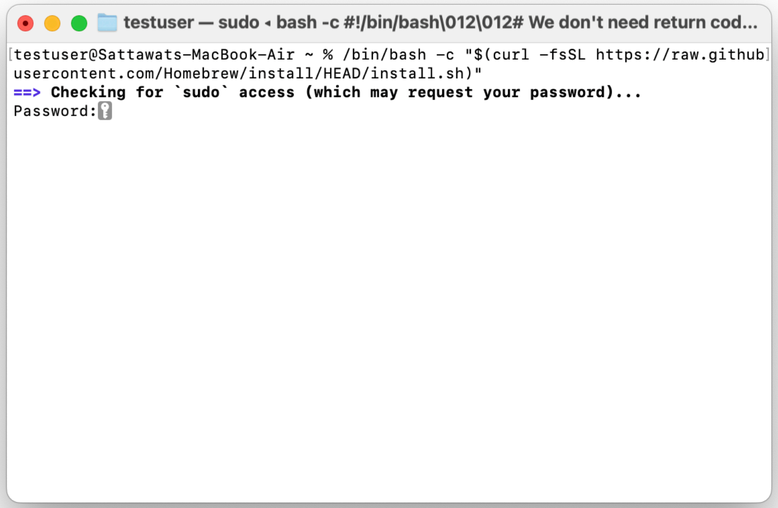
At first, it might be a bit challenging, but your life will become more comfortable with Homebrew
Copy that command and paste it to your terminal

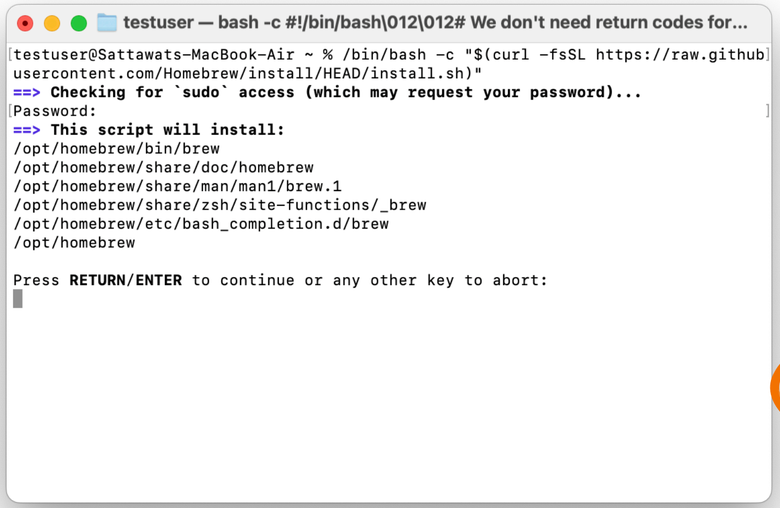


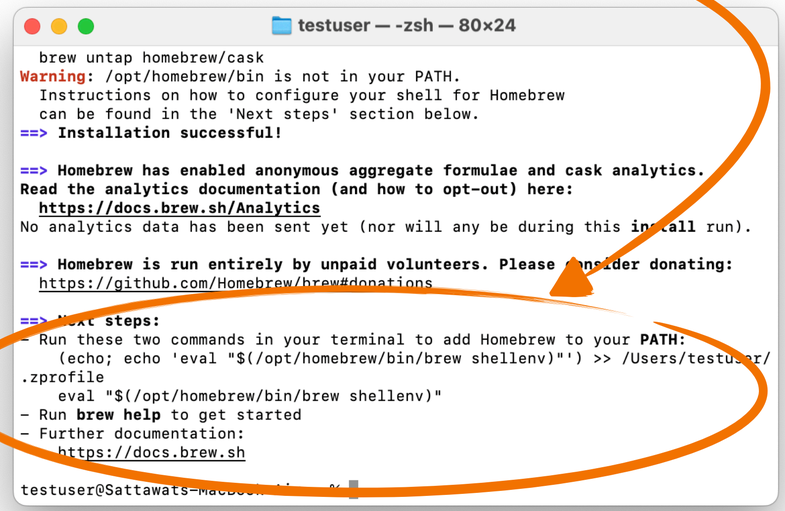
Mac: Install Homebrew

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

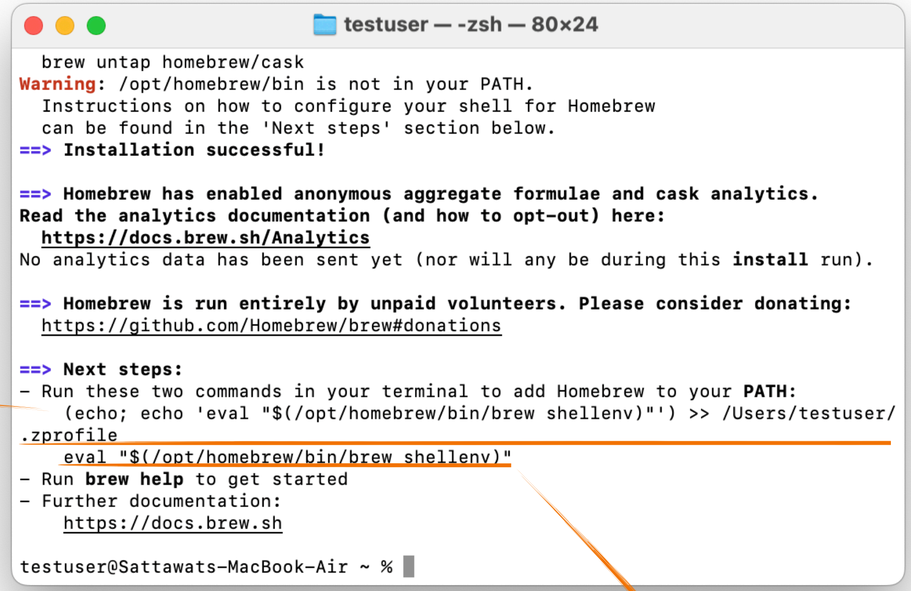


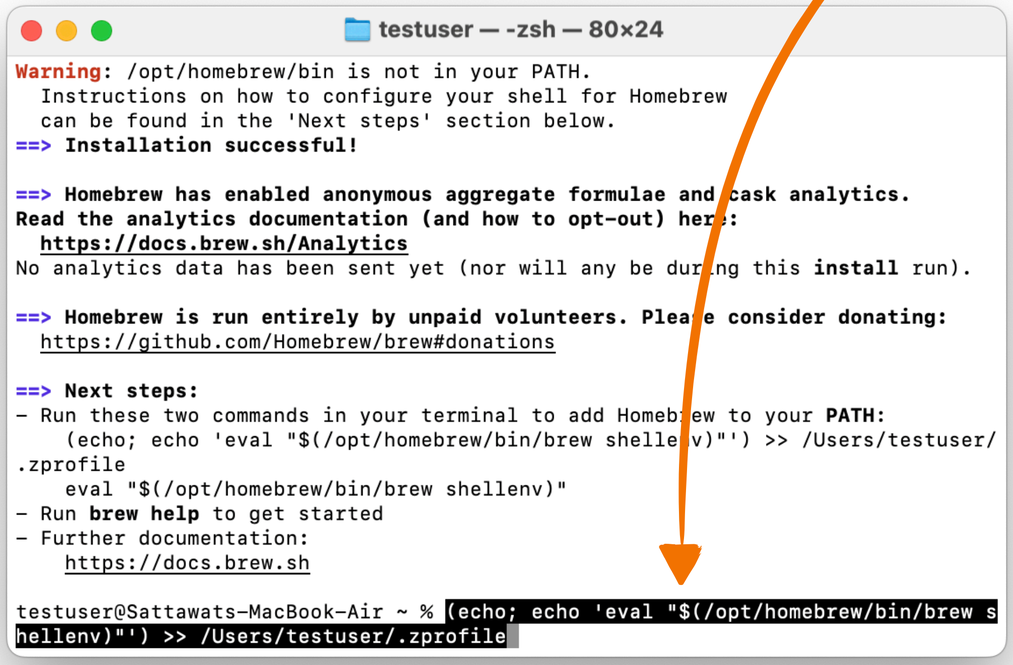


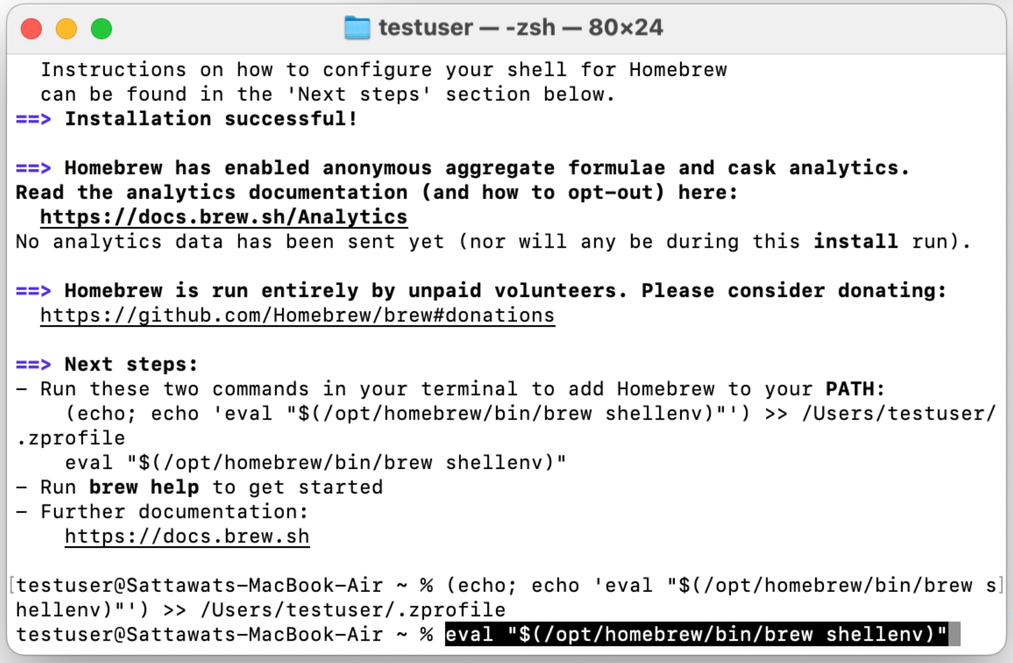


Mac: Set Homebrew path

Copy that provided 2 lines, and paste to terminal





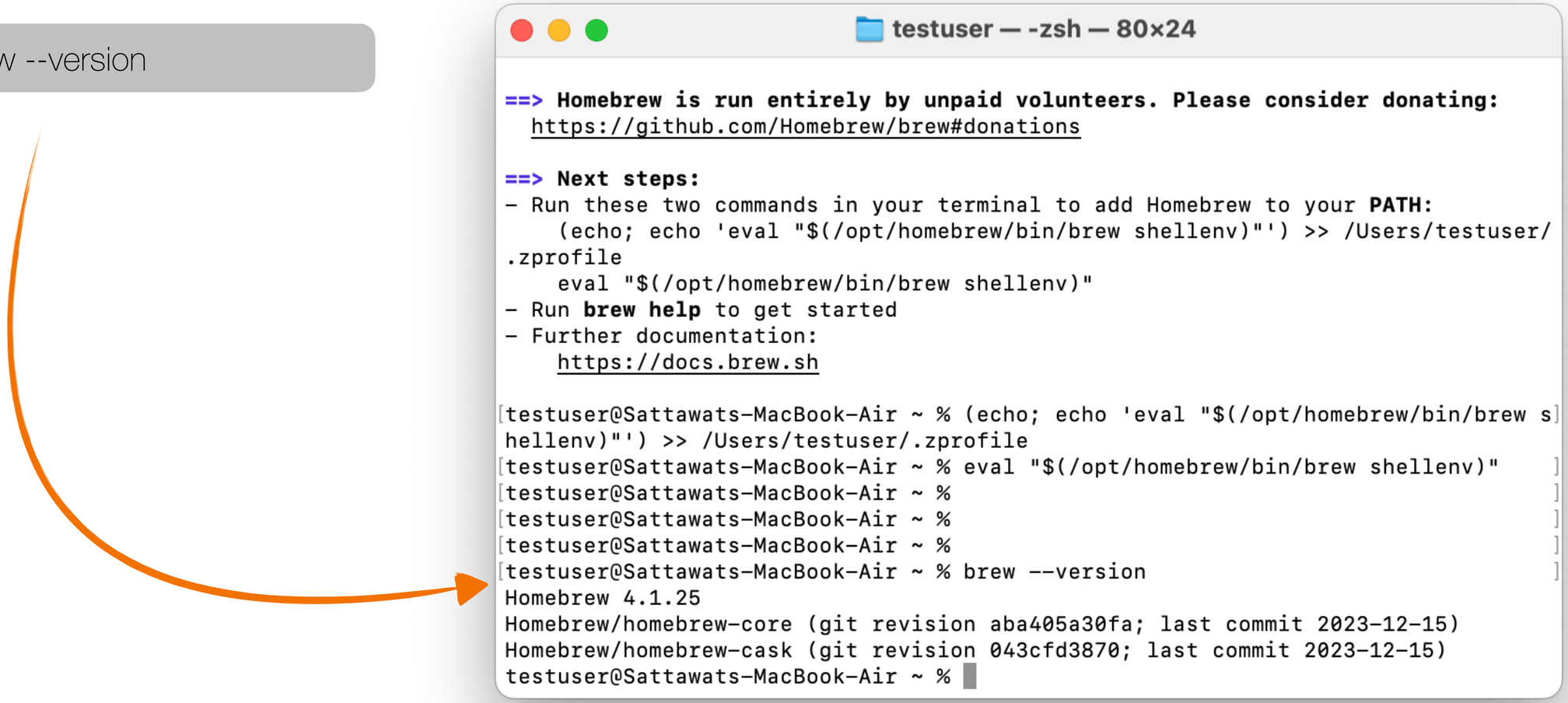


Mac: Check Homebrew version

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

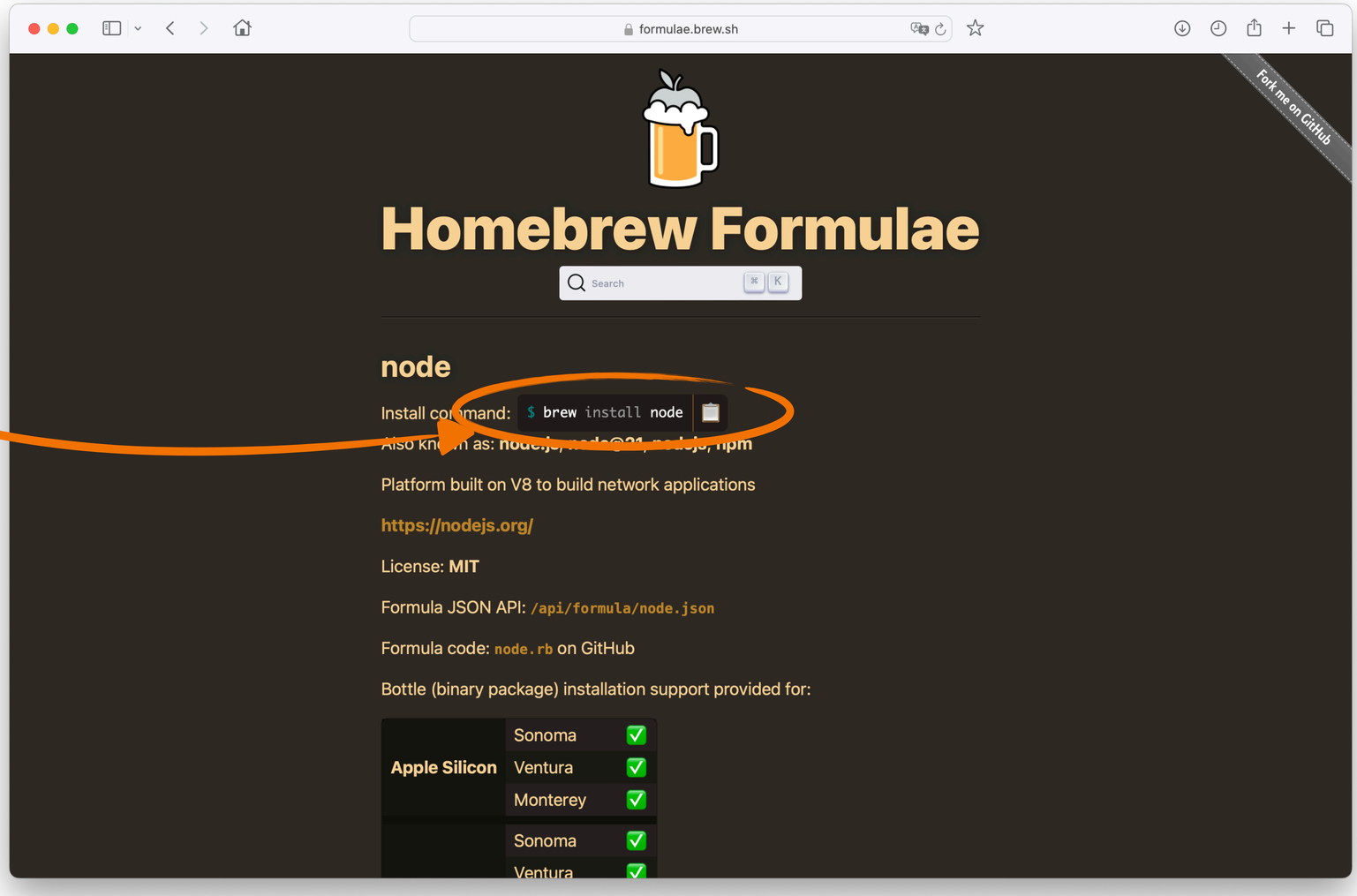
Check brew version:

```
brew --version
```



Mac: Install Node.js

Just run, and wait

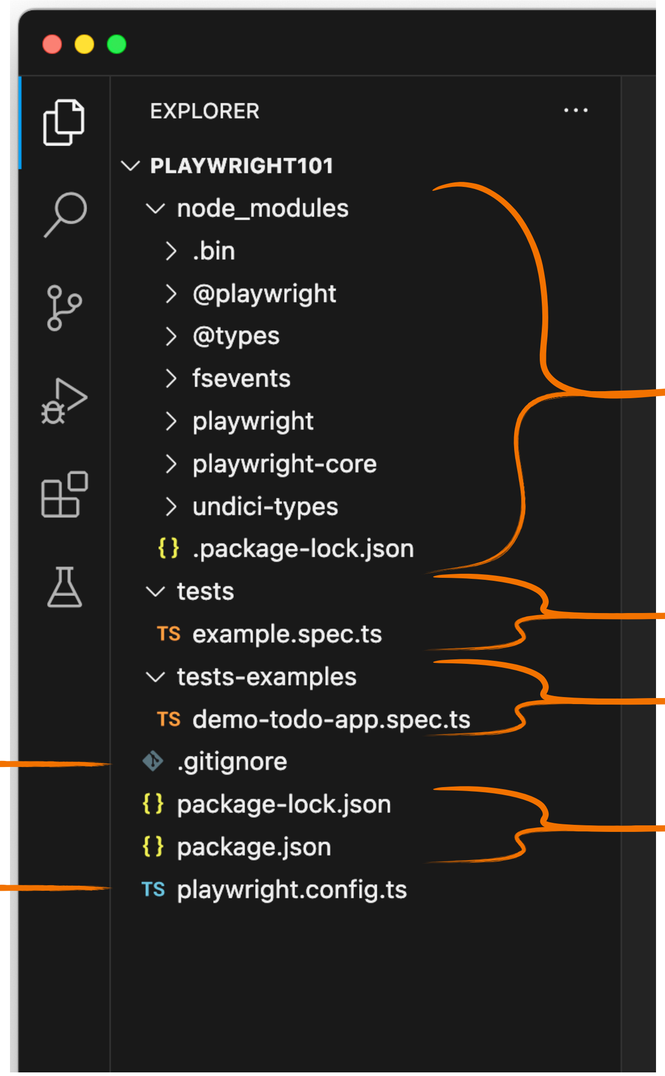


Playwright project structure

and

Example scenarios from init

Project structure



Installed Node.js libraries

Our test scenarios will be implement here

Example test scenarios from Playwright init

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

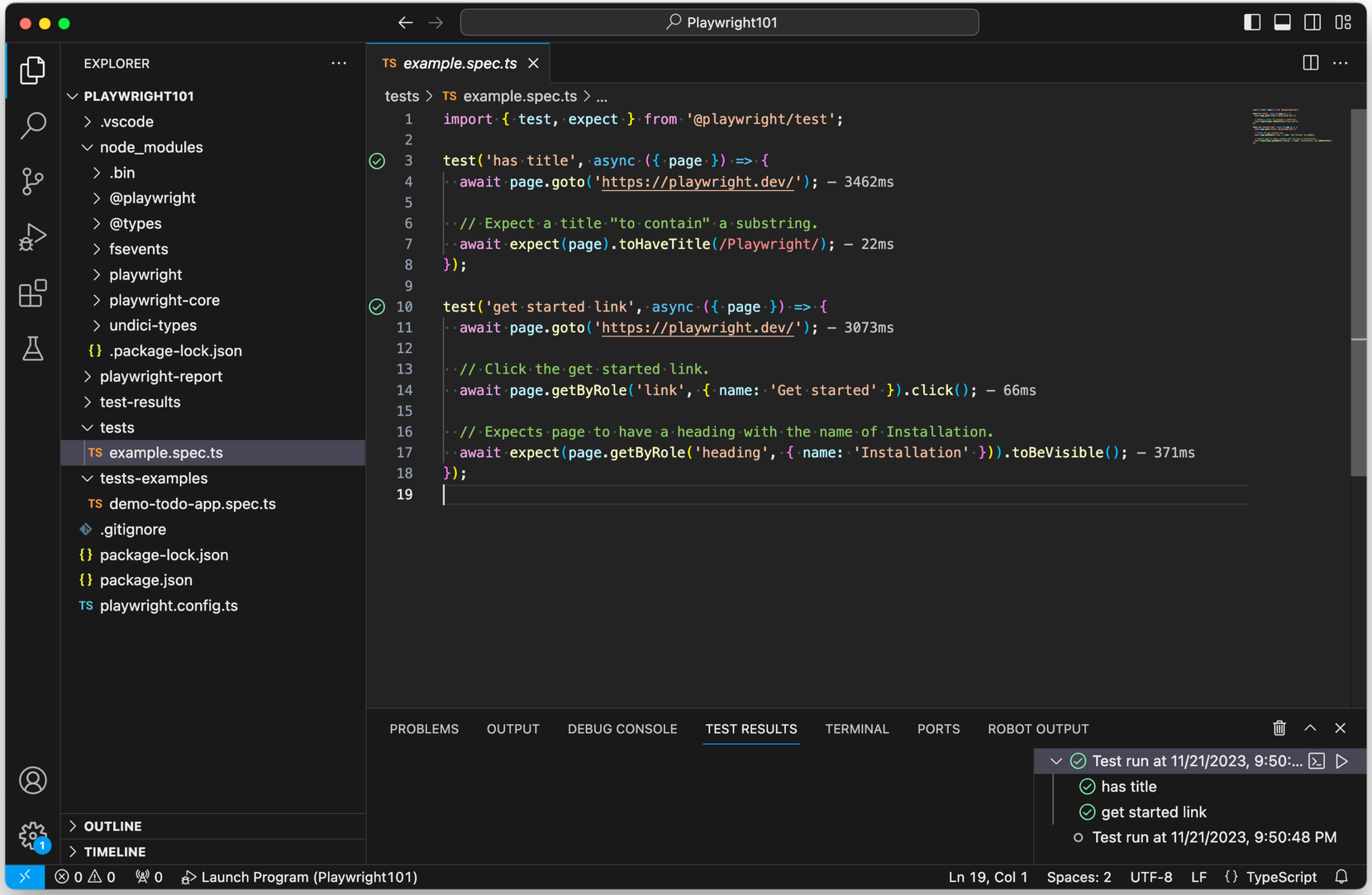
Specifies intentionally untracked files that Git should ignore

options to configure how your tests are run

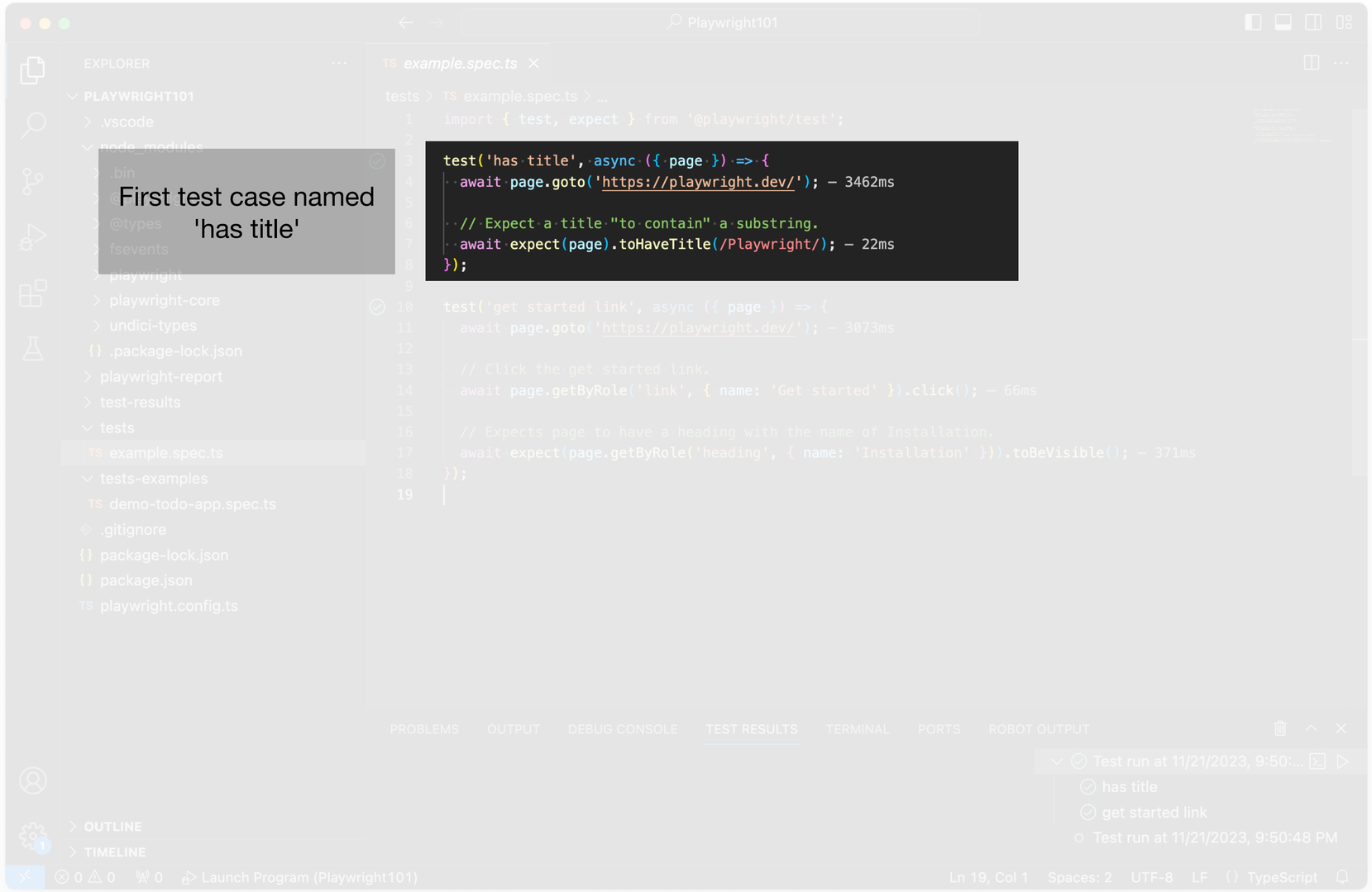
Ref1: package.json: https://docs.npmjs.com/cli/v10/configuring-npm/package-json
Ref2: package-lock.json: https://docs.npmjs.com/cli/v10/configuring-npm/package-lock-json

Ref3: gitignore: https://git-scm.com/docs/gitignore
Ref4: Test configuration: https://playwright.dev/docs/test-configuration

Example test cases



Example test cases: has title



Test scenario structure

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

```
test(          ,
);
```

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**'

```
async (          ) => {
}
```

```
test('has title', async ({ page }) => {
  await page.goto('https://playwright.dev/'); — 3462ms

  // Expect a title "to contain" a substring.
  await expect(page).toHaveTitle(/Playwright/); — 22ms
});
```

Example test cases: has title

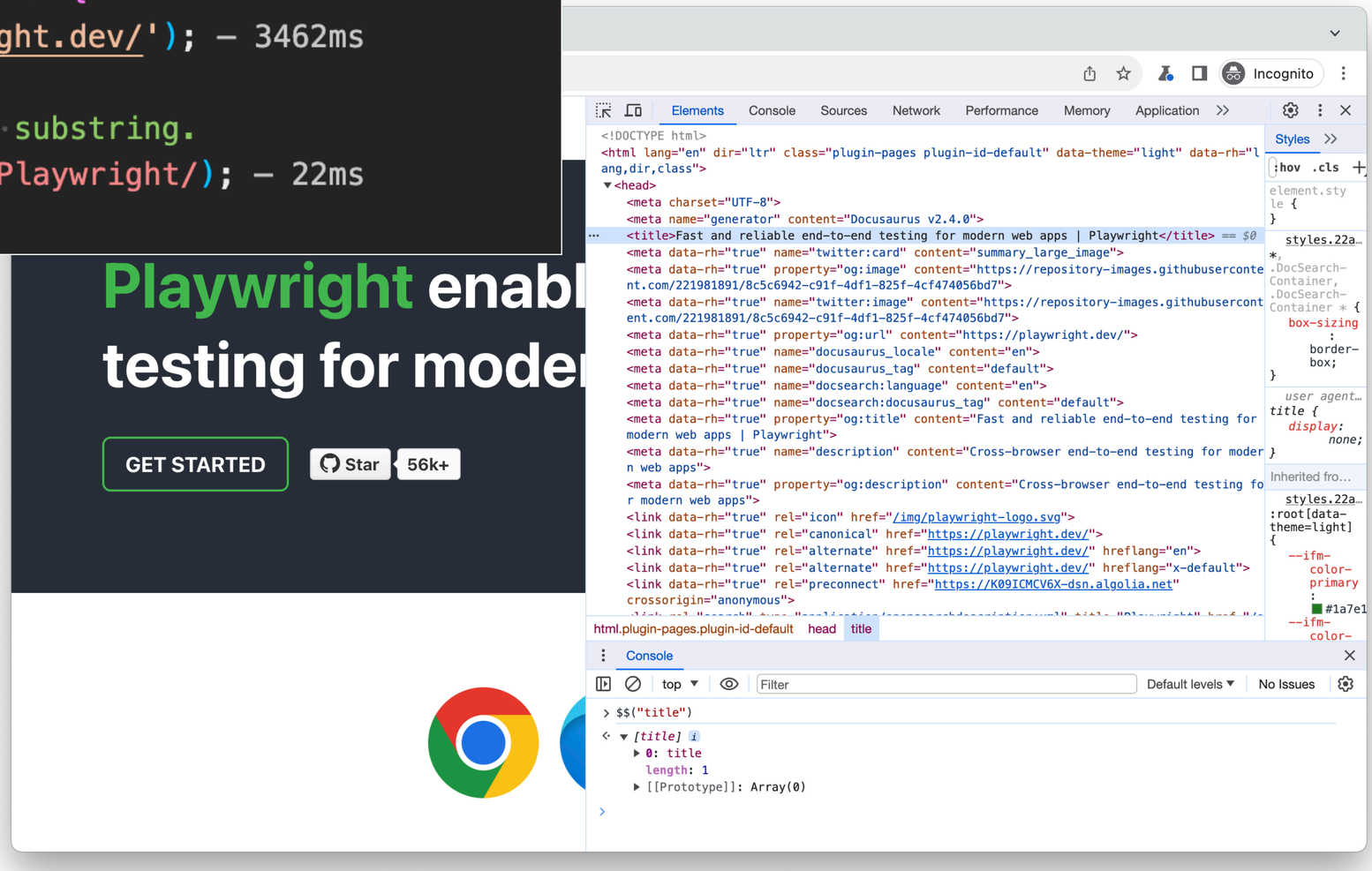
```
3  test('has title', async ({ page }) => {
4    await page.goto('https://playwright.dev/'); — 3462ms
5
6    // Expect a title "to contain" a substring.
7    await expect(page).toHaveTitle(/Playwright/); — 22ms
8  });
```

Test steps:

Line3 Create test case named 'has title' and assigned async function with 1 argument name 'page'

Line4 At current page, navigate to web url 'https://playwright.dev/'

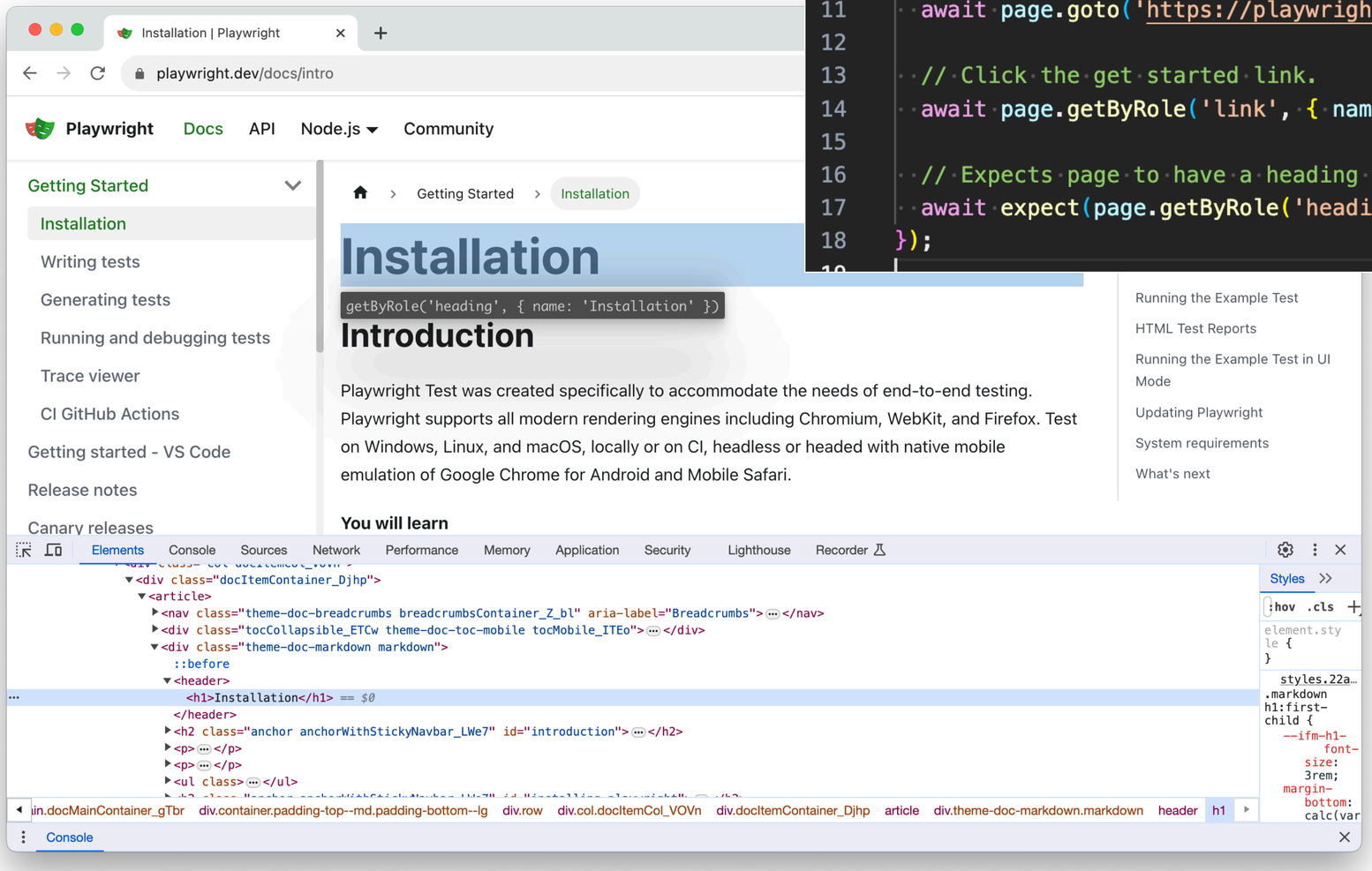
Line7 Assert page's title must contains text 'Playwright'



Example test cases: get started link

Example test cases: get started link

```
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  });
```



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

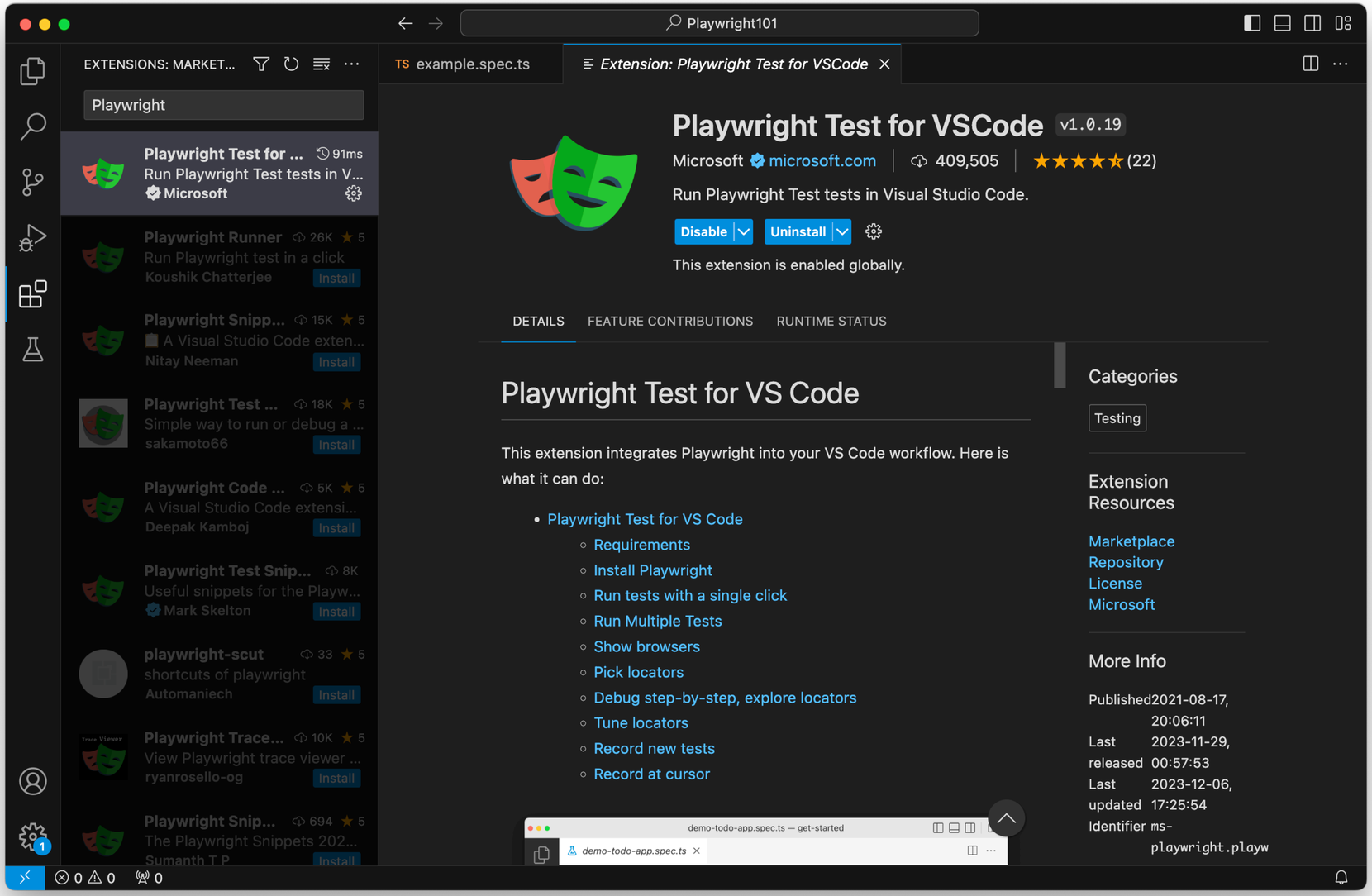
Debug mode

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

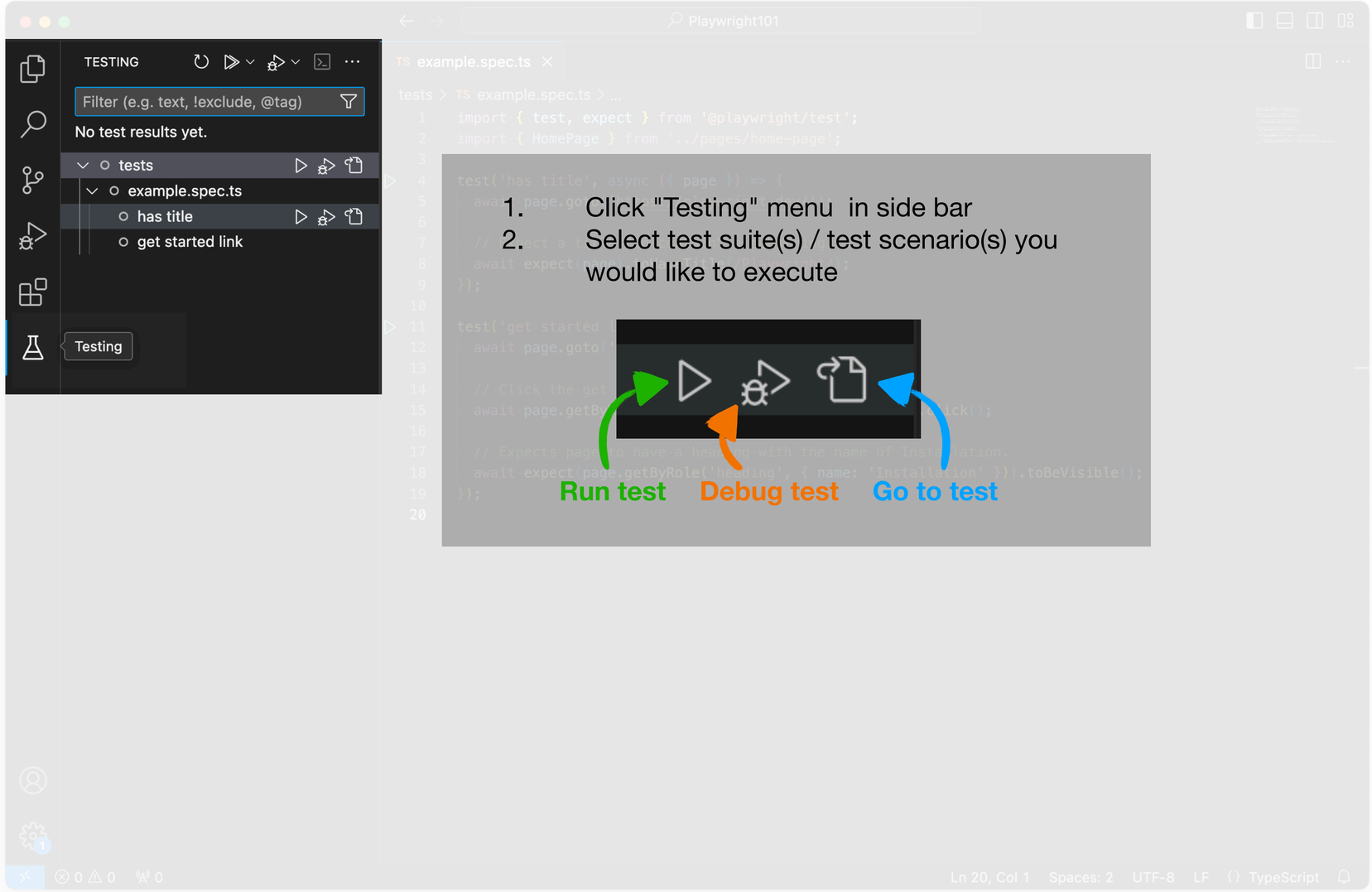
Ref1: https://playwright.dev/docs/intro#running-the-example-test-in-ui-mode
Ref2: https://playwright.dev/docs/test-ui-mode

Ref3: https://playwright.dev/docs/debug

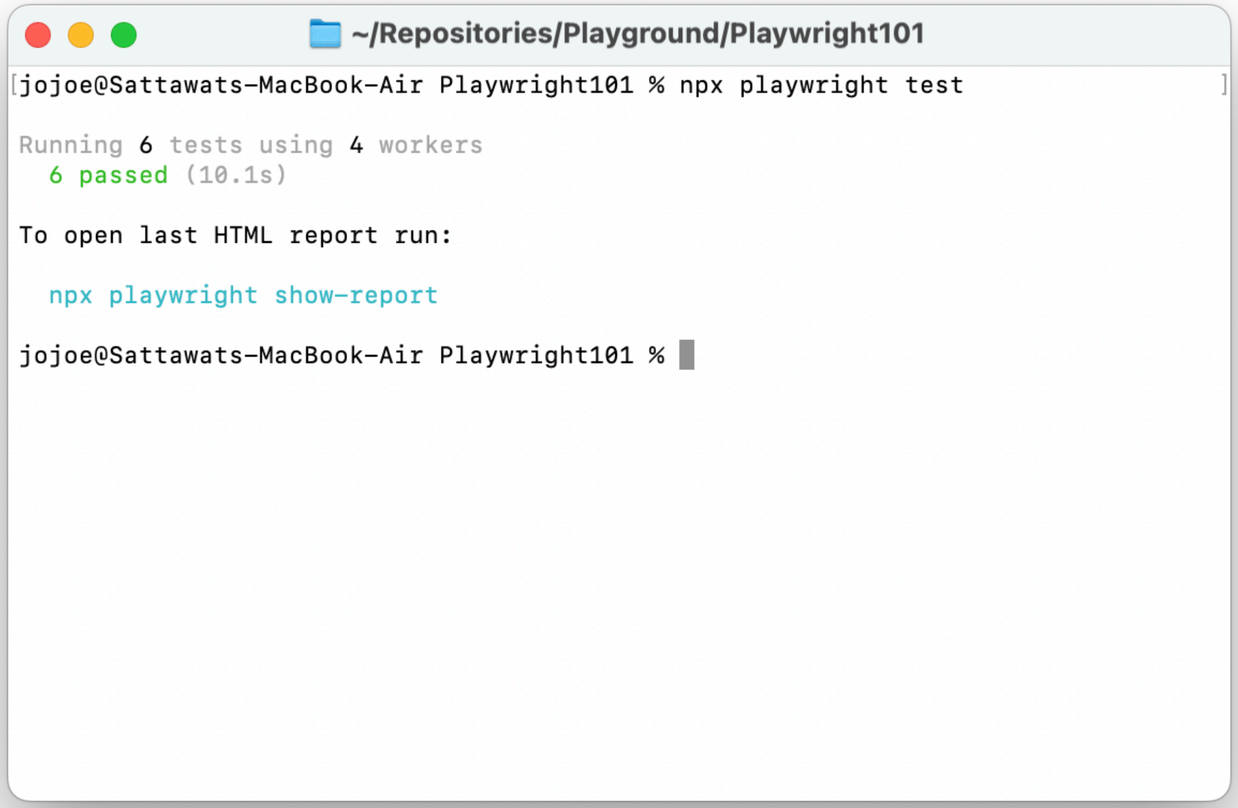
VSCode extension



Execute / debug tests from VSCode



Execute tests from command



Execute all test suite and scenarios:

```
npx playwright test
```

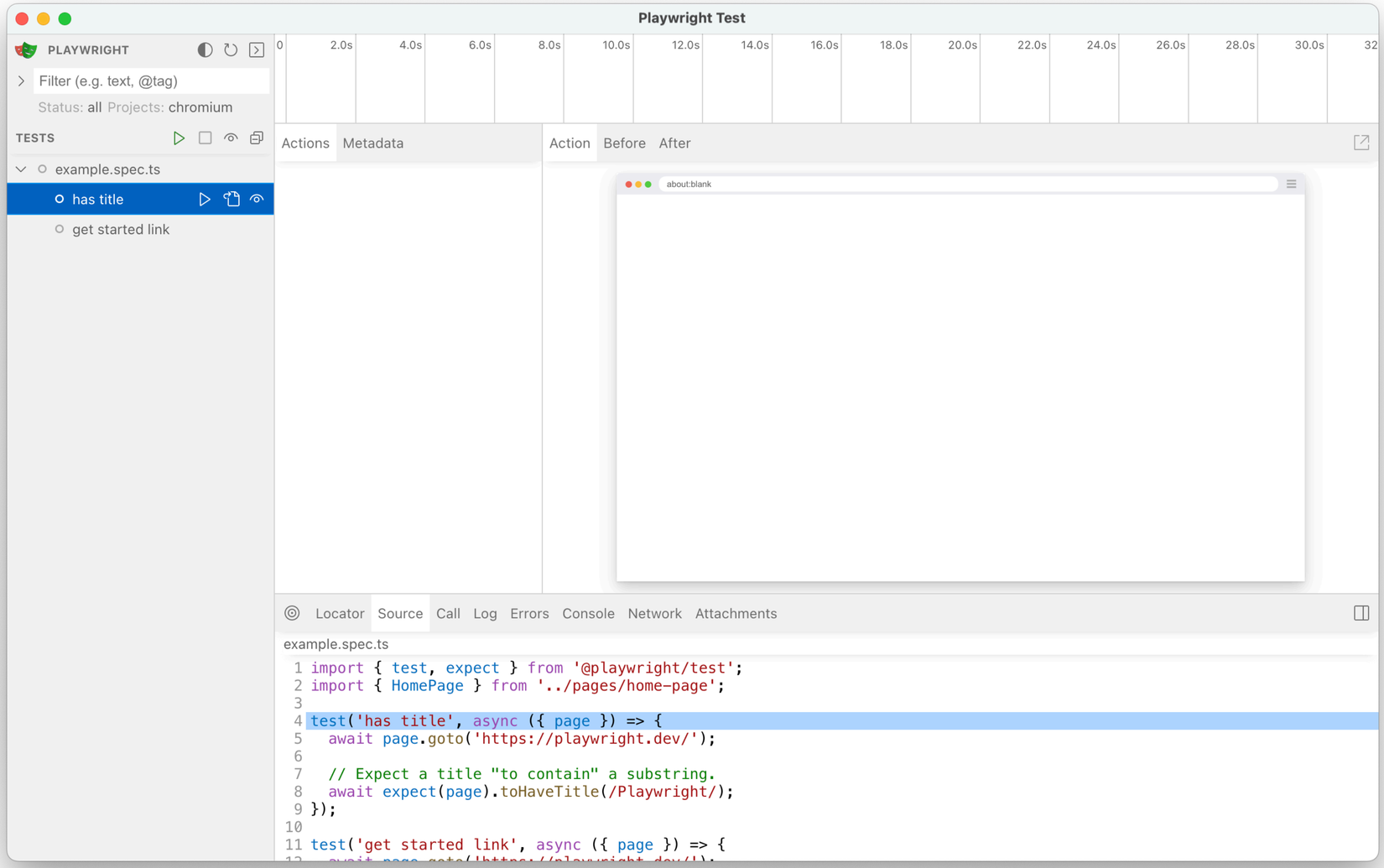
Execute single test suite:

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

Execute single test scenarios:

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

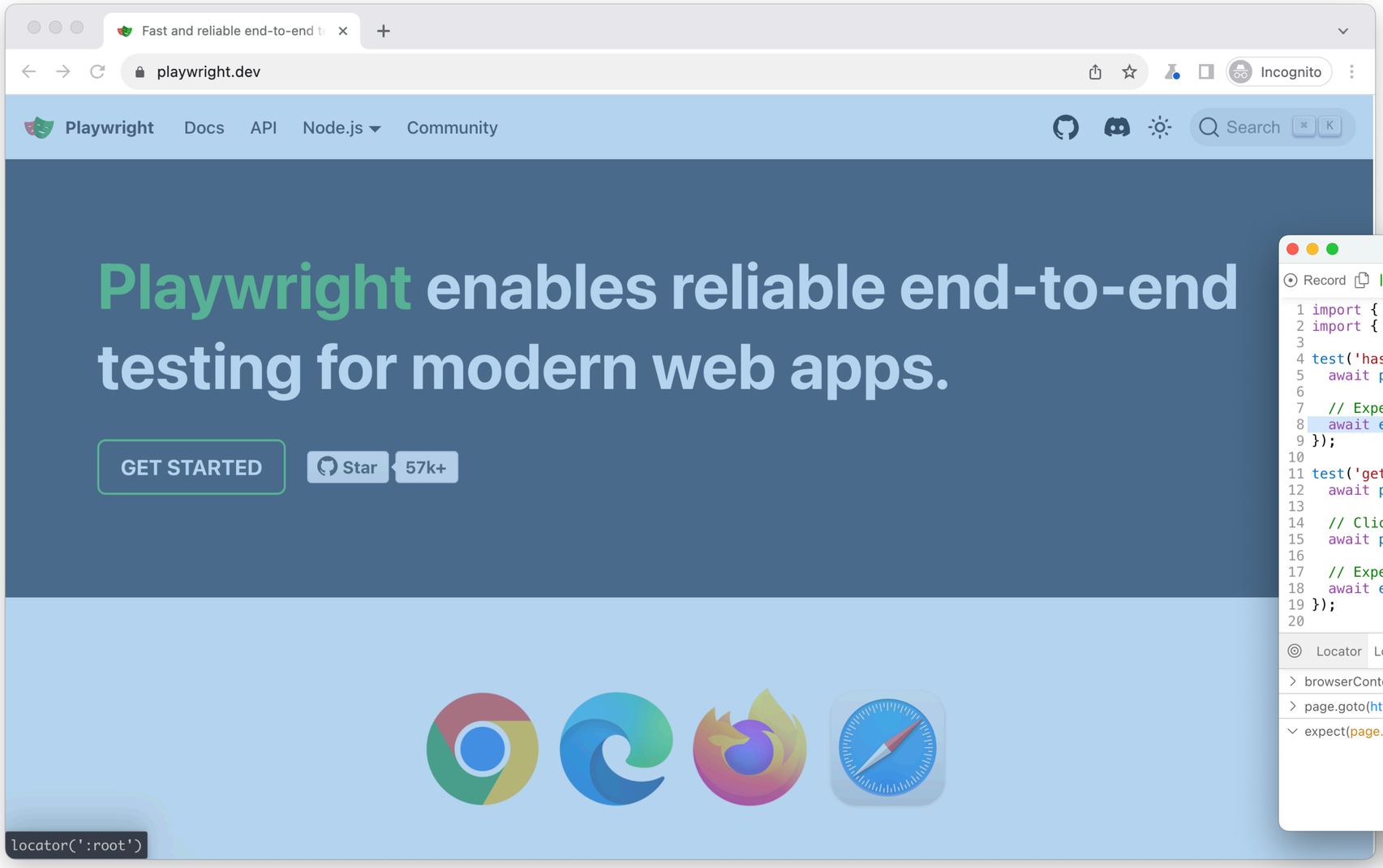
Execute tests in UI mode from command



Open inspector UI mode:

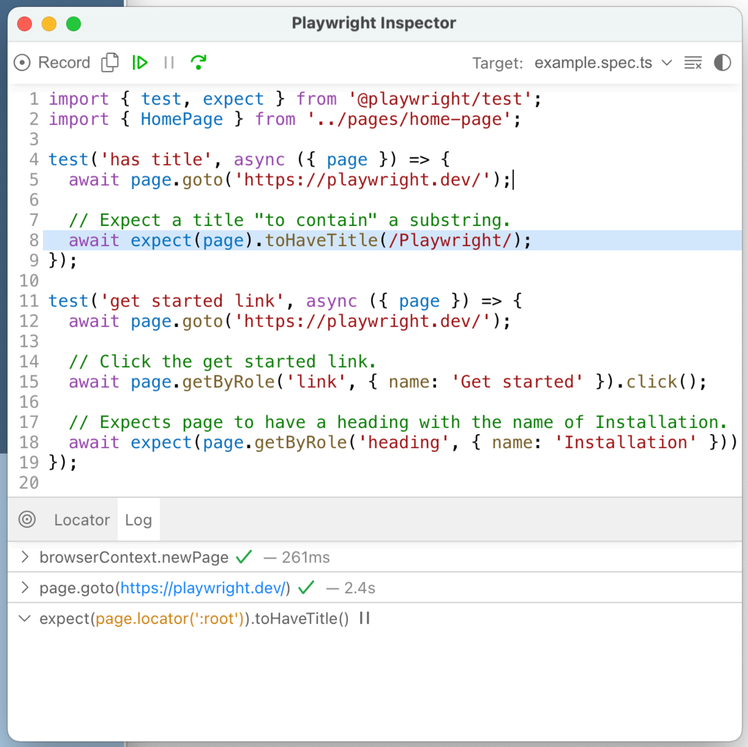
```
npx playwright test --ui
```

Execute tests in debug mode from command



Open inspector debug mode:

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



Locators

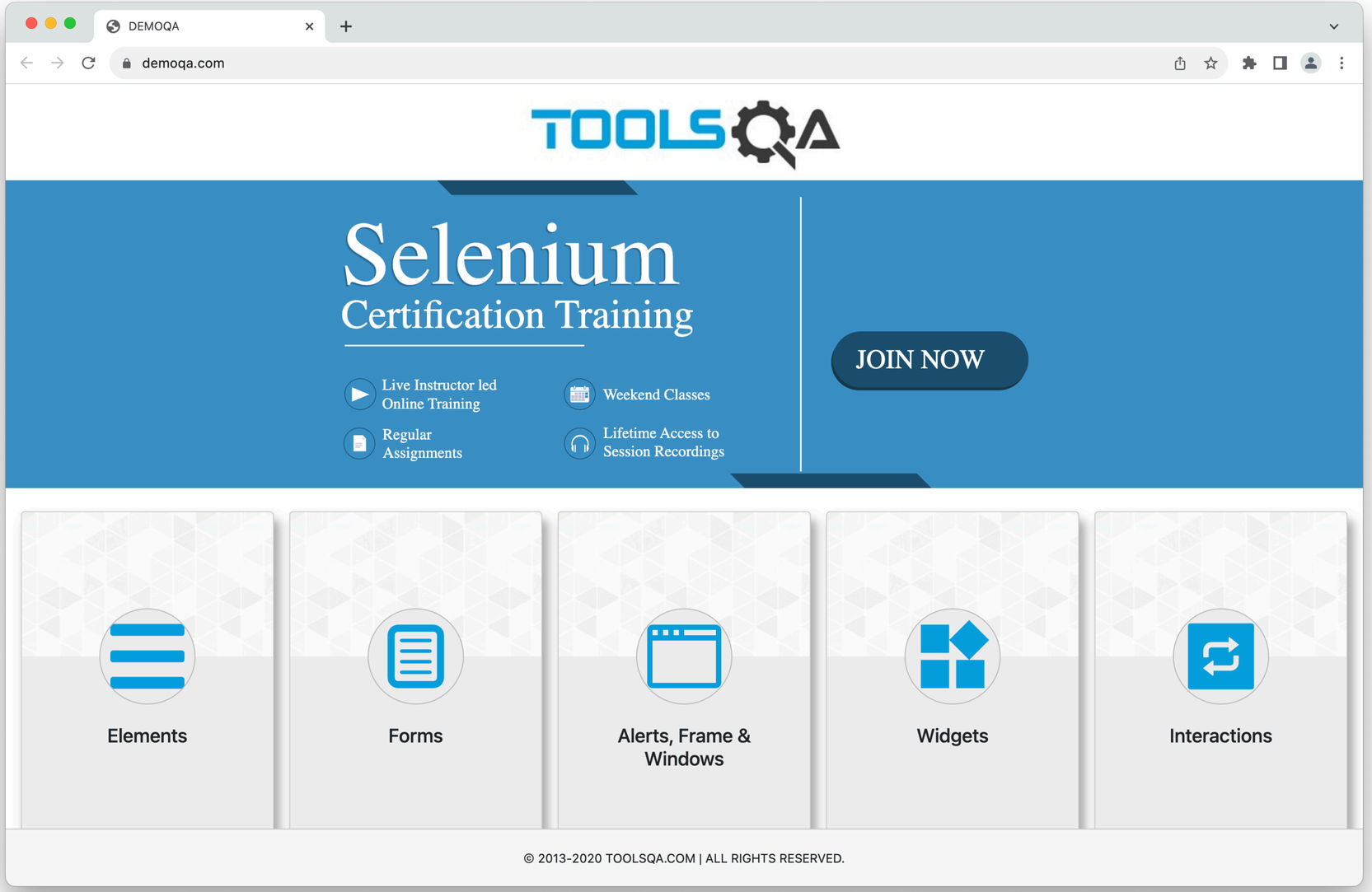
- Chrome inspect and console
- HTML Tags
  - ID
  - CSS
  - XPath
  - data-testid
  - Chain locator

Ref1: https://playwright.dev/docs/other-locators

Tools QA : Our playground in this session :)

Open Chrome browser

Go to this url: demoqa.com

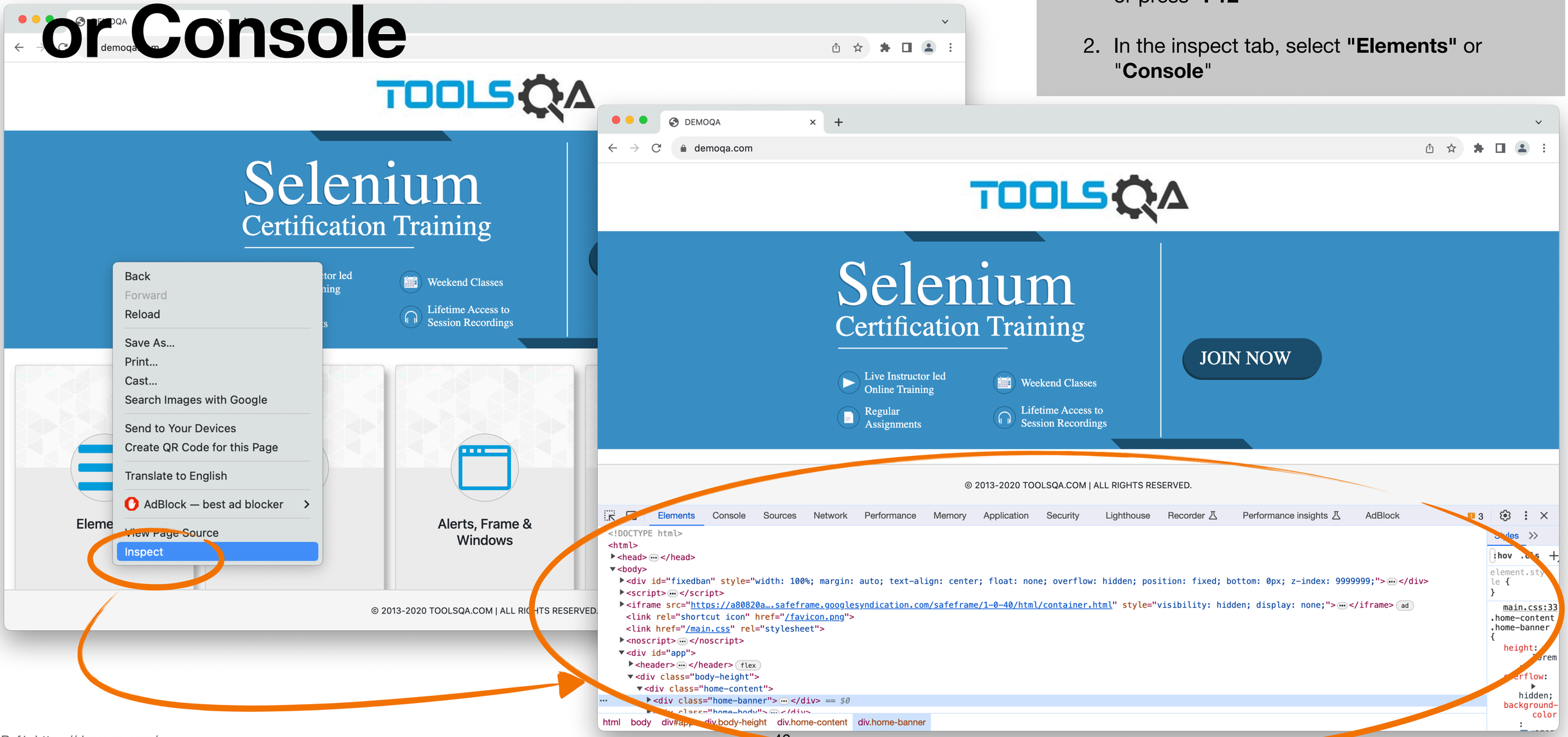


Ref1: https://demoqa.com/

Chrome Inspect, Elements, or Console

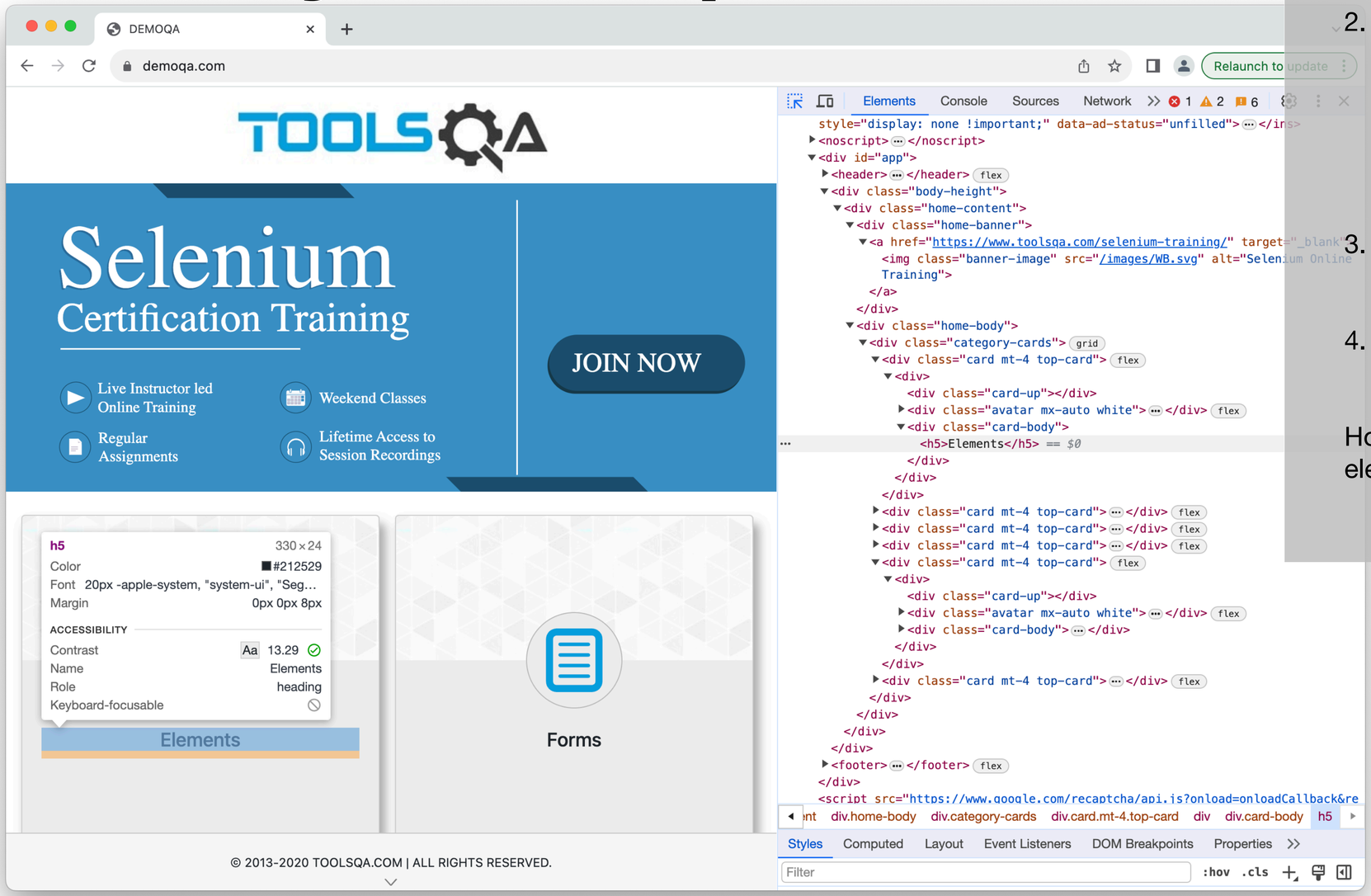
Open inspect and console steps:

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



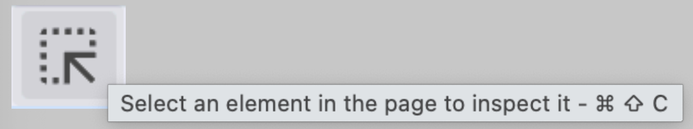
Ref1: https://demoqa.com/

Identify the unique locator



Steps to identify locator of some elements:

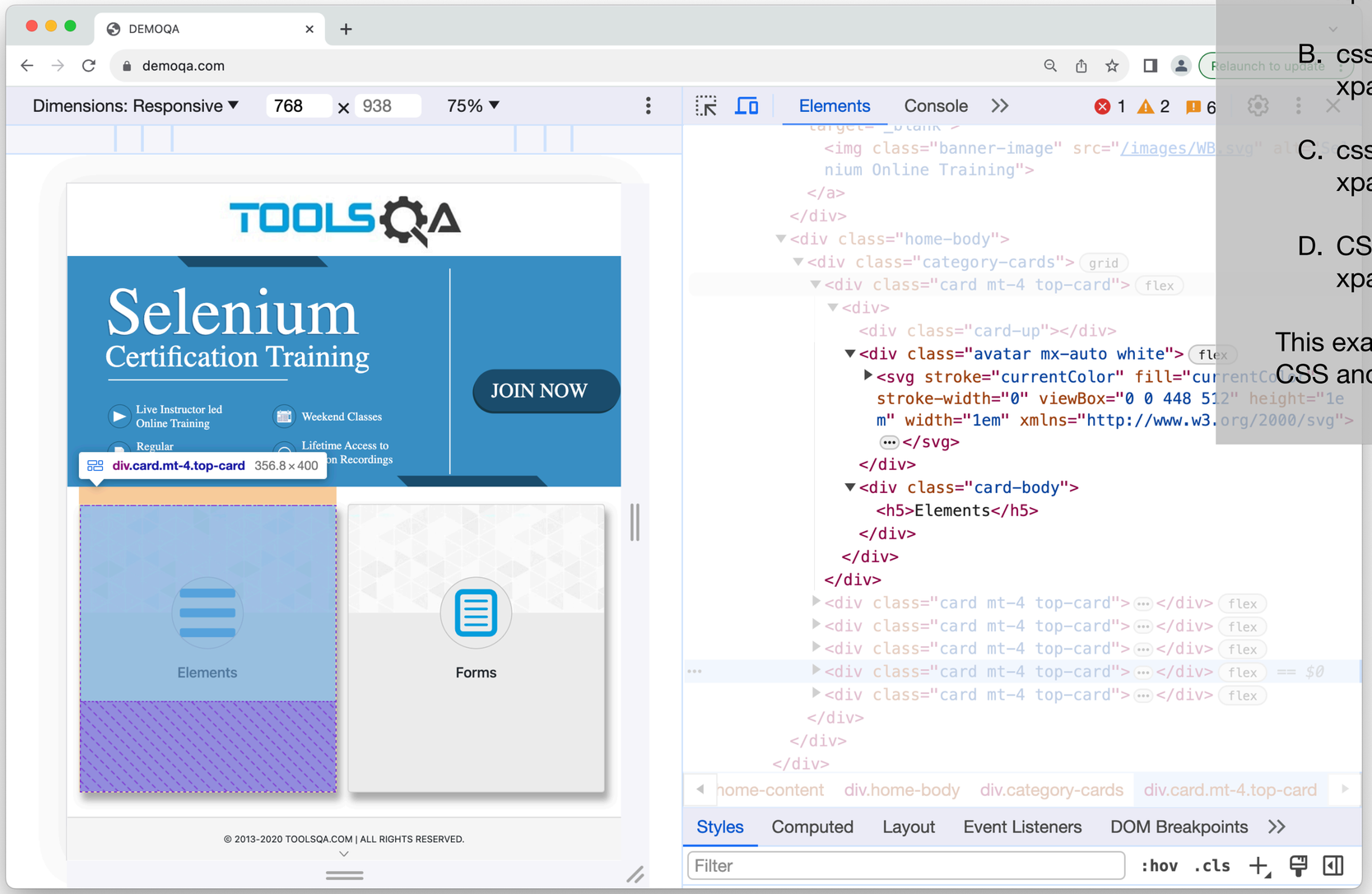
1. Open Chrome inspect, and go to Elements tab
2. Use this tool, it is the easies way



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



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']

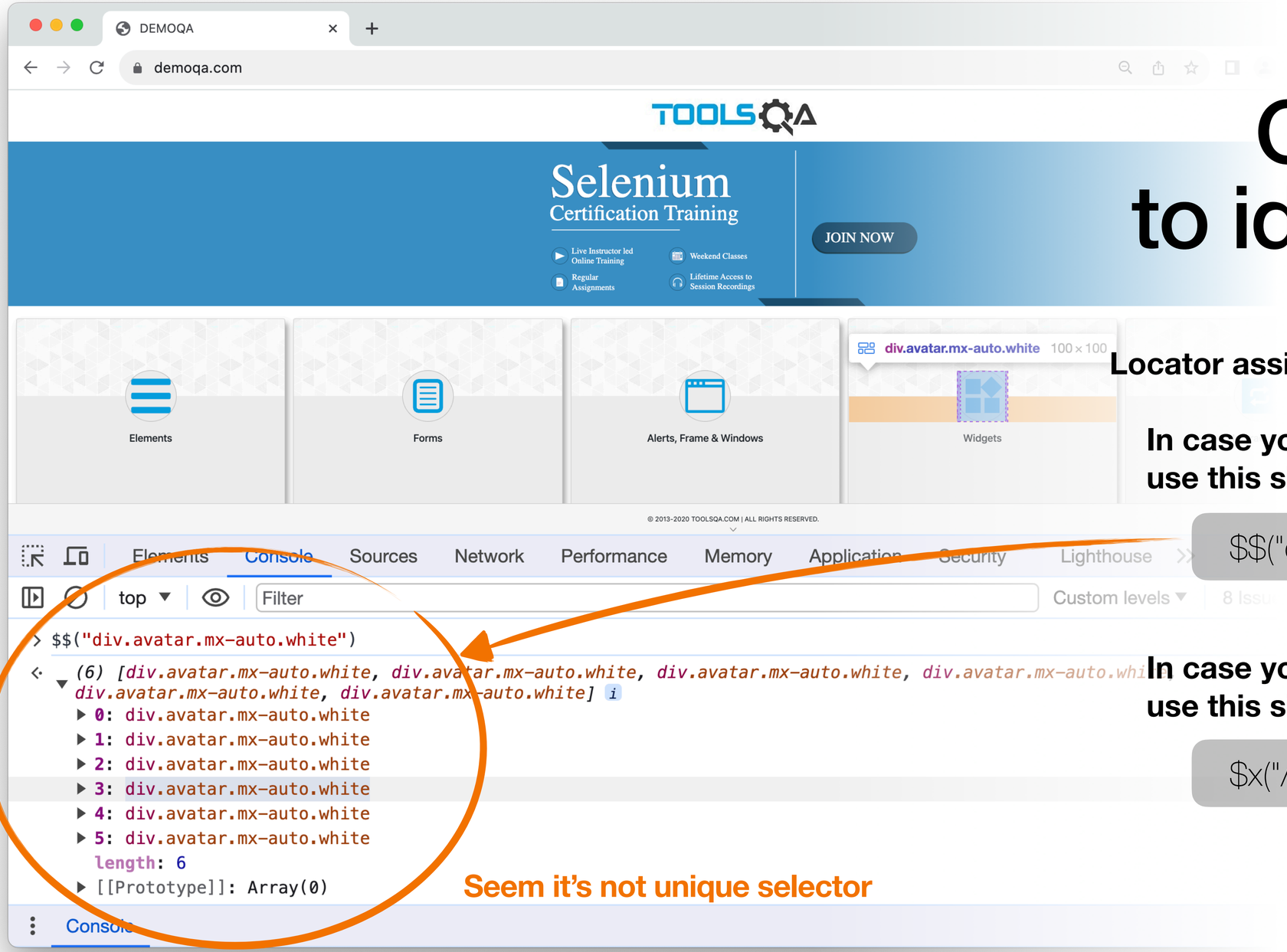
B. css = div.card-body
   xpath = //div[@class='card-body']

C. css = h5
   xpath = //h5

D. CSS is 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



Locator assignment:

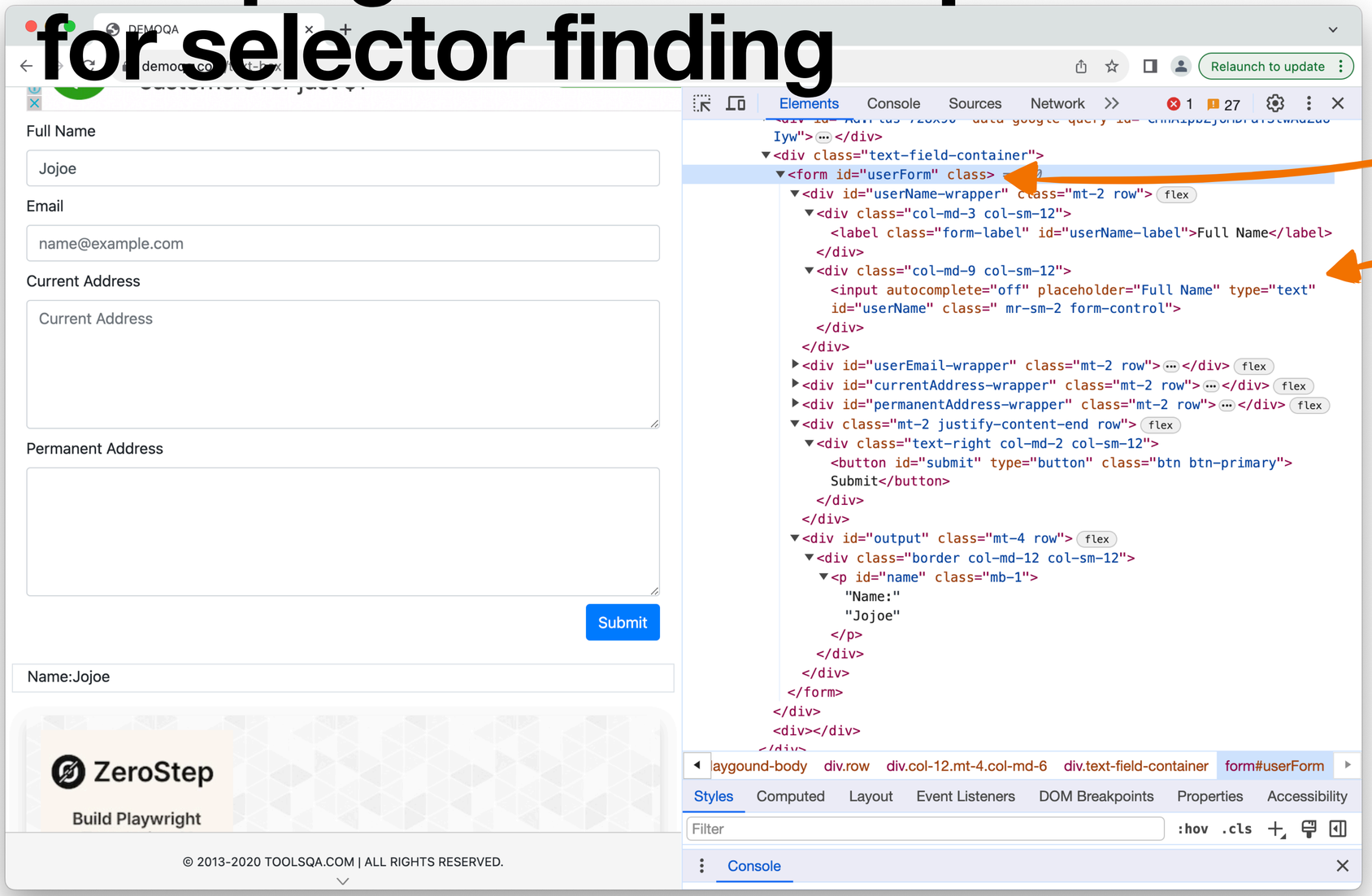
In case you would like to fine the CSS selector use this sign:

```
$$("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']")
```

This page is the best practice for selector finding



id = userForm

css = form#userForm

id = userName

css = input#userName

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

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

data-testid locator

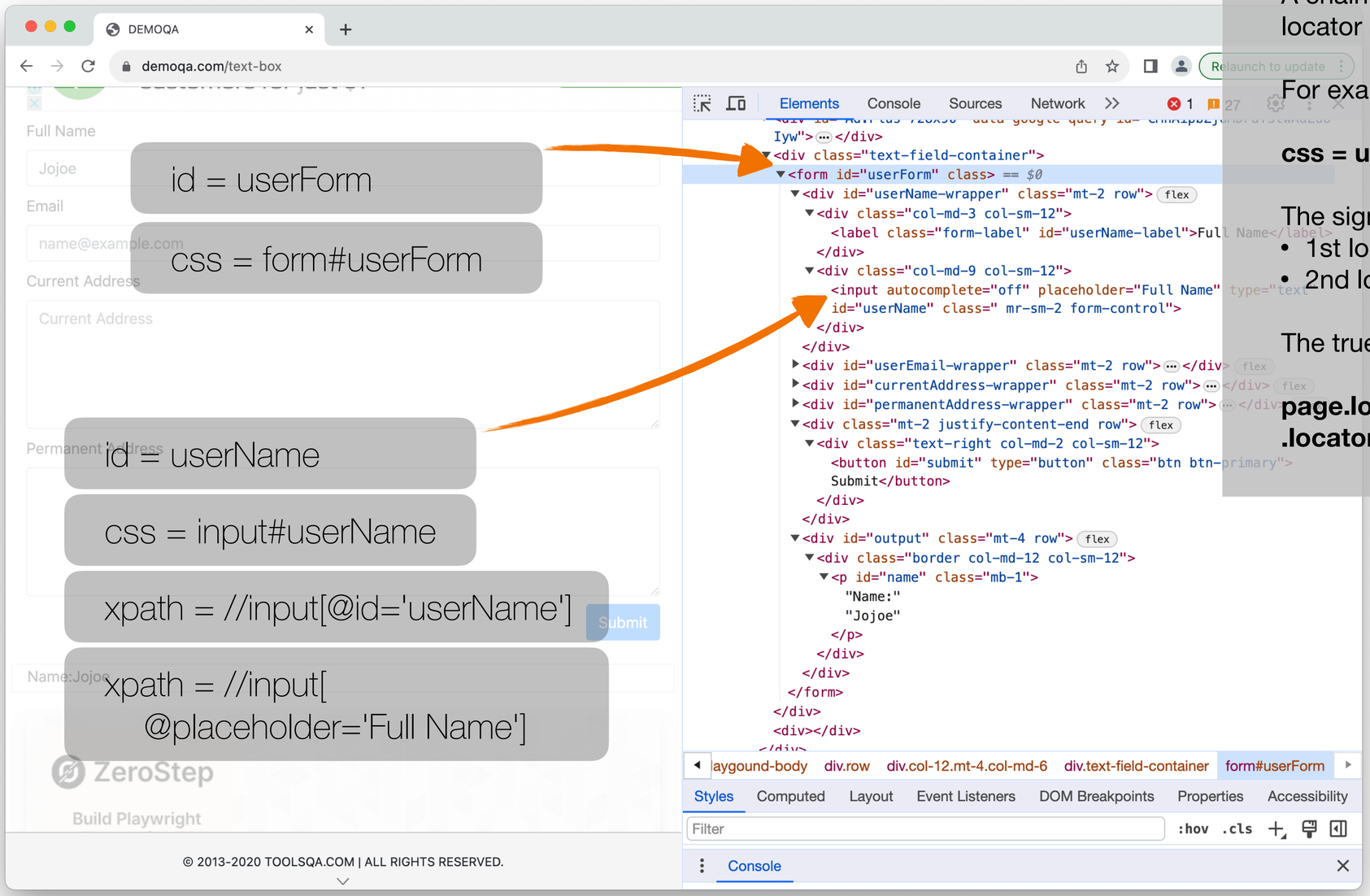
data-testid is an attribute commonly used in web development to facilitate testing of user interfaces. It is not a native HTML attribute, but rather a **custom attribute that developers add to their HTML elements to make it easier to select and interact with those elements in automated tests.**

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

Ref1: OpenAI. (2023). "What is data-testid and how to naming it."

Chain locator



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']'")

Ref1: https://playwright.dev/docs/other-locators#chaining-selectors

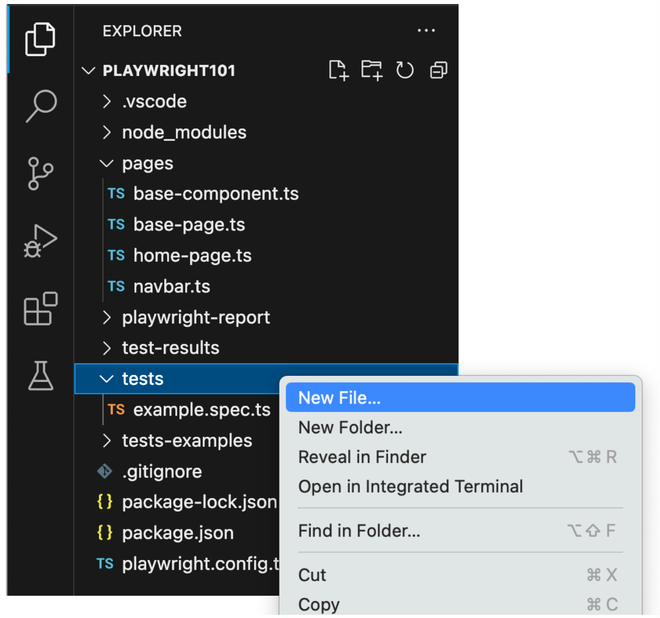
Create first test

Suite

and

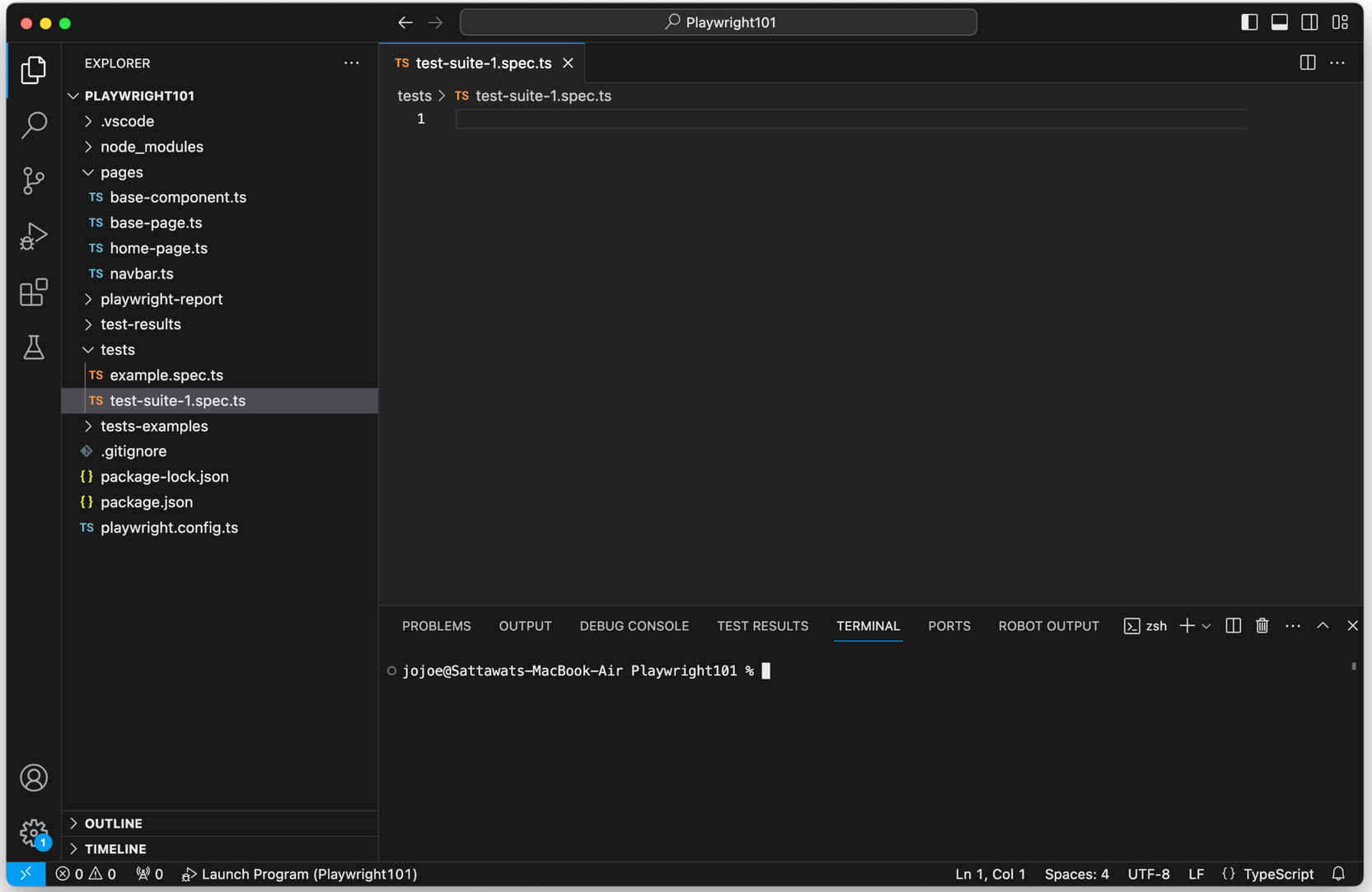
Scenario

Create first test suite file

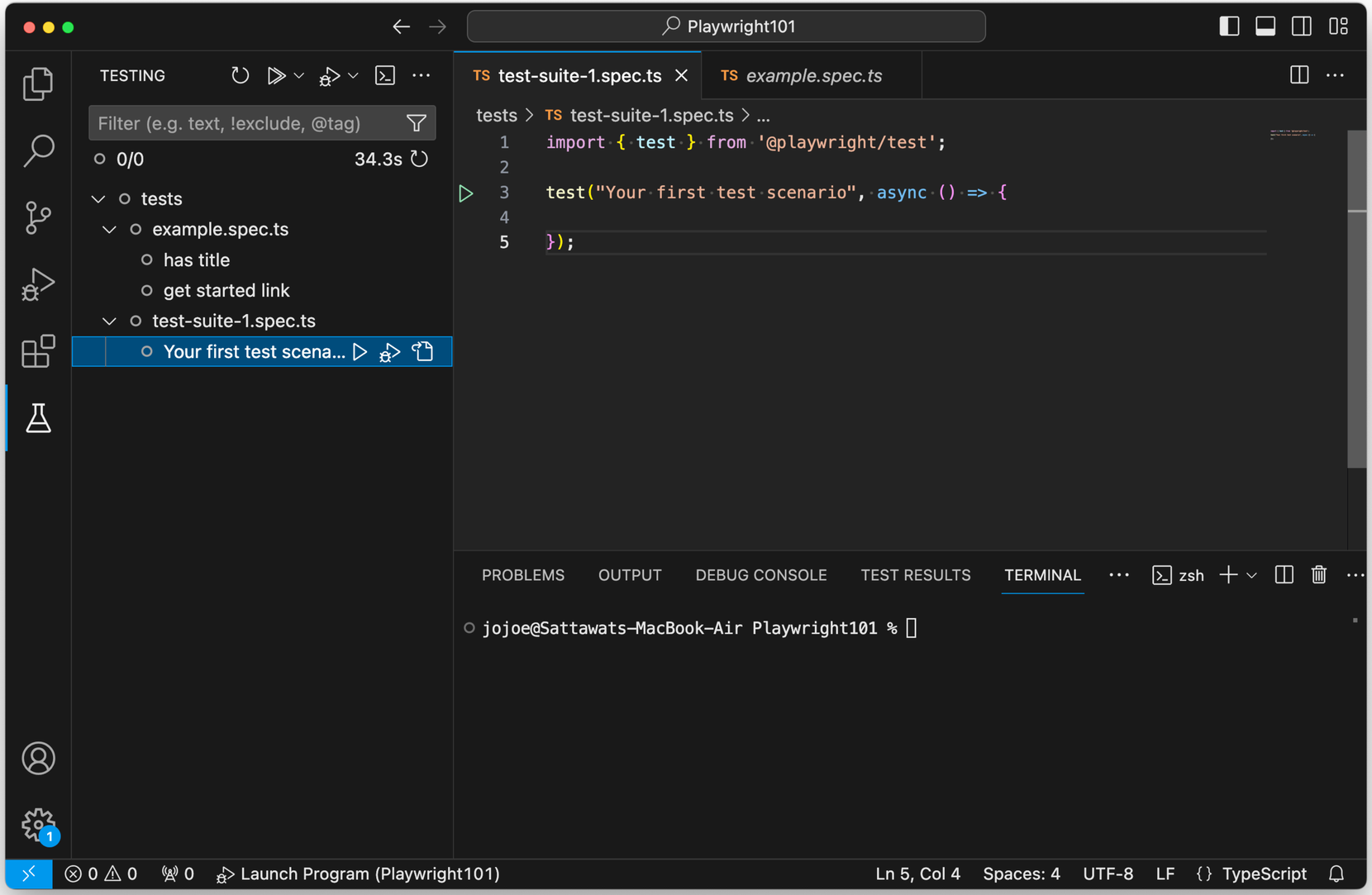


Create new test file (Test suite) steps:

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



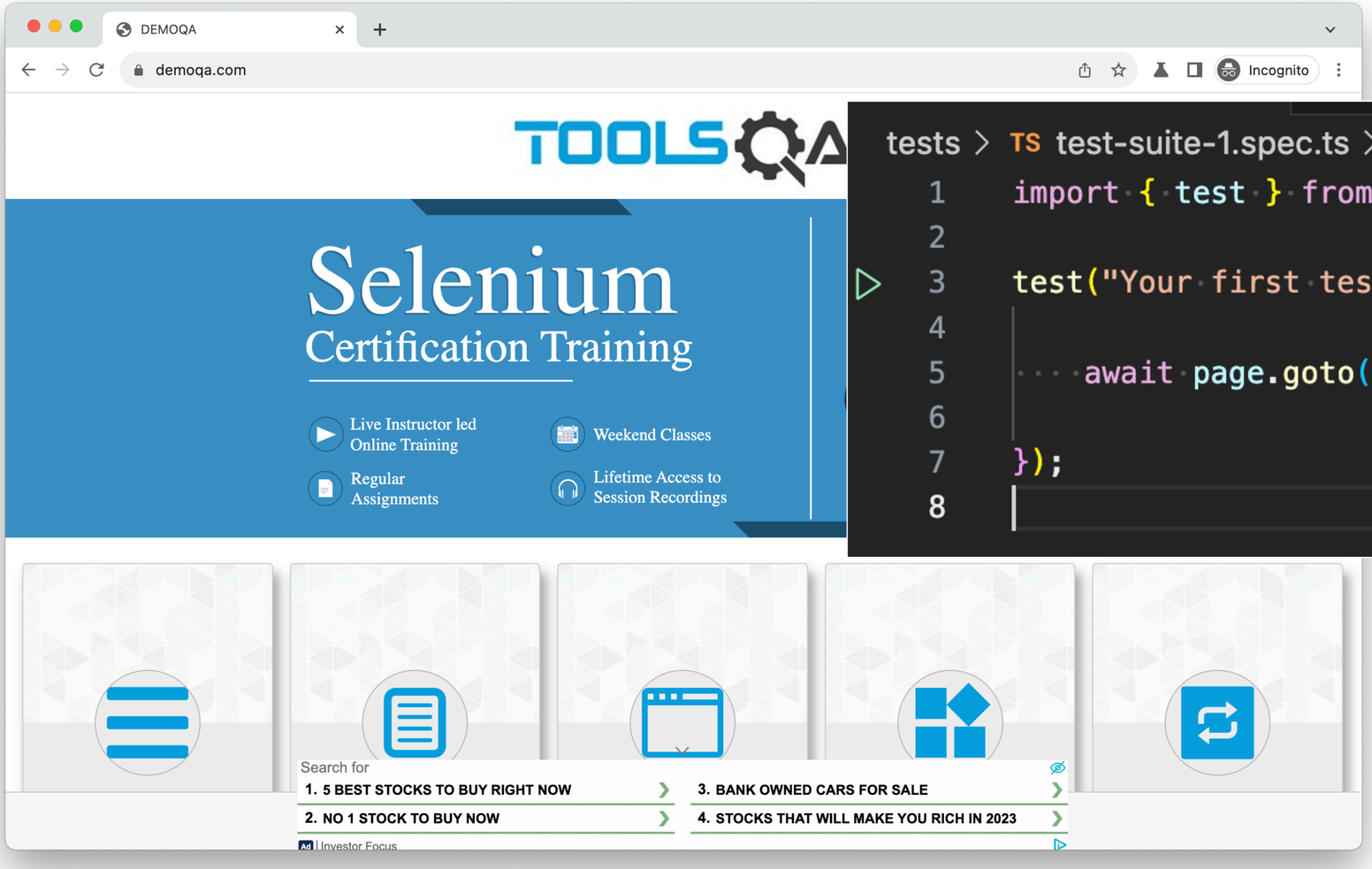
Actions

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

Ref1: https://playwright.dev/docs/navigations
Ref2: https://playwright.dev/docs/input

Page navigation

Do you remember our playground?
In this part, we will let's our test scenario page open that website.



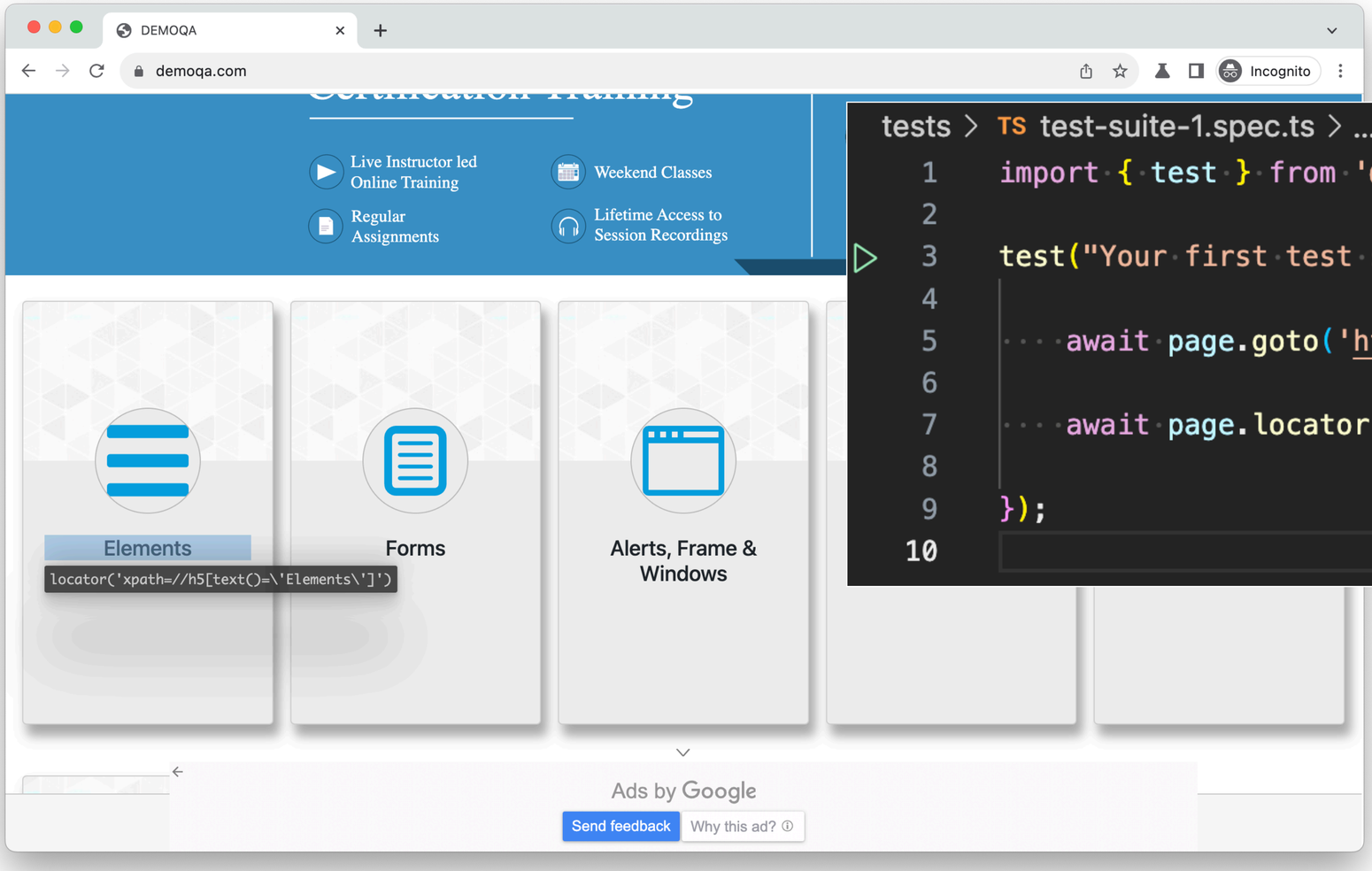

```
tests > TS test-suite-1.spec.ts > ...
1  import { test } from '@playwright/test';
2
3  test("Your first test scenario", async ({ page }) => {
4
5      await page.goto('https://demoqa.com');
6
7  });
8
```

Ref1: https://playwright.dev/docs/navigations

Mouse click

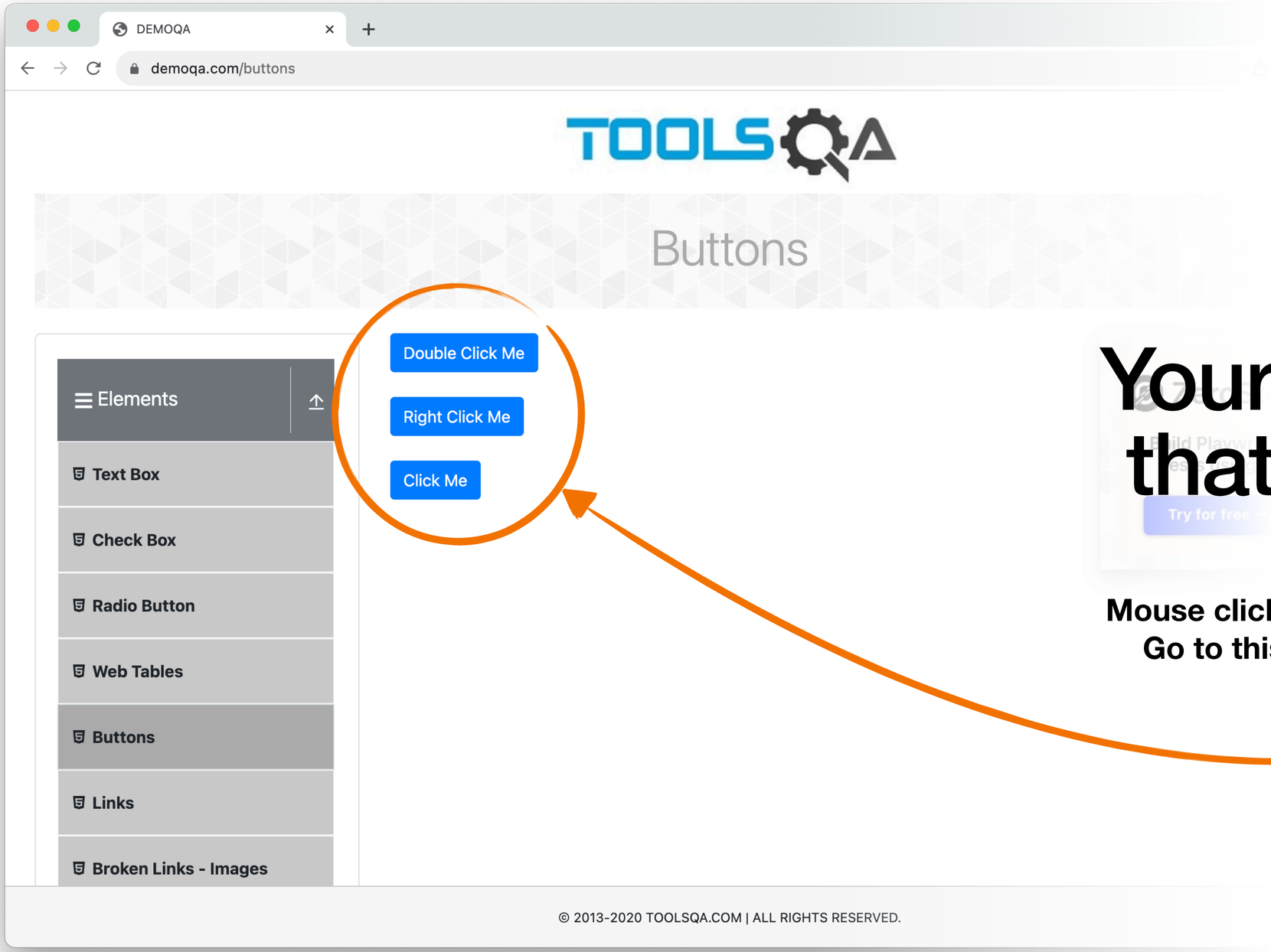
>> Elements > Buttons

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




```
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
```

Ref1: https://playwright.dev/docs/input



Well done

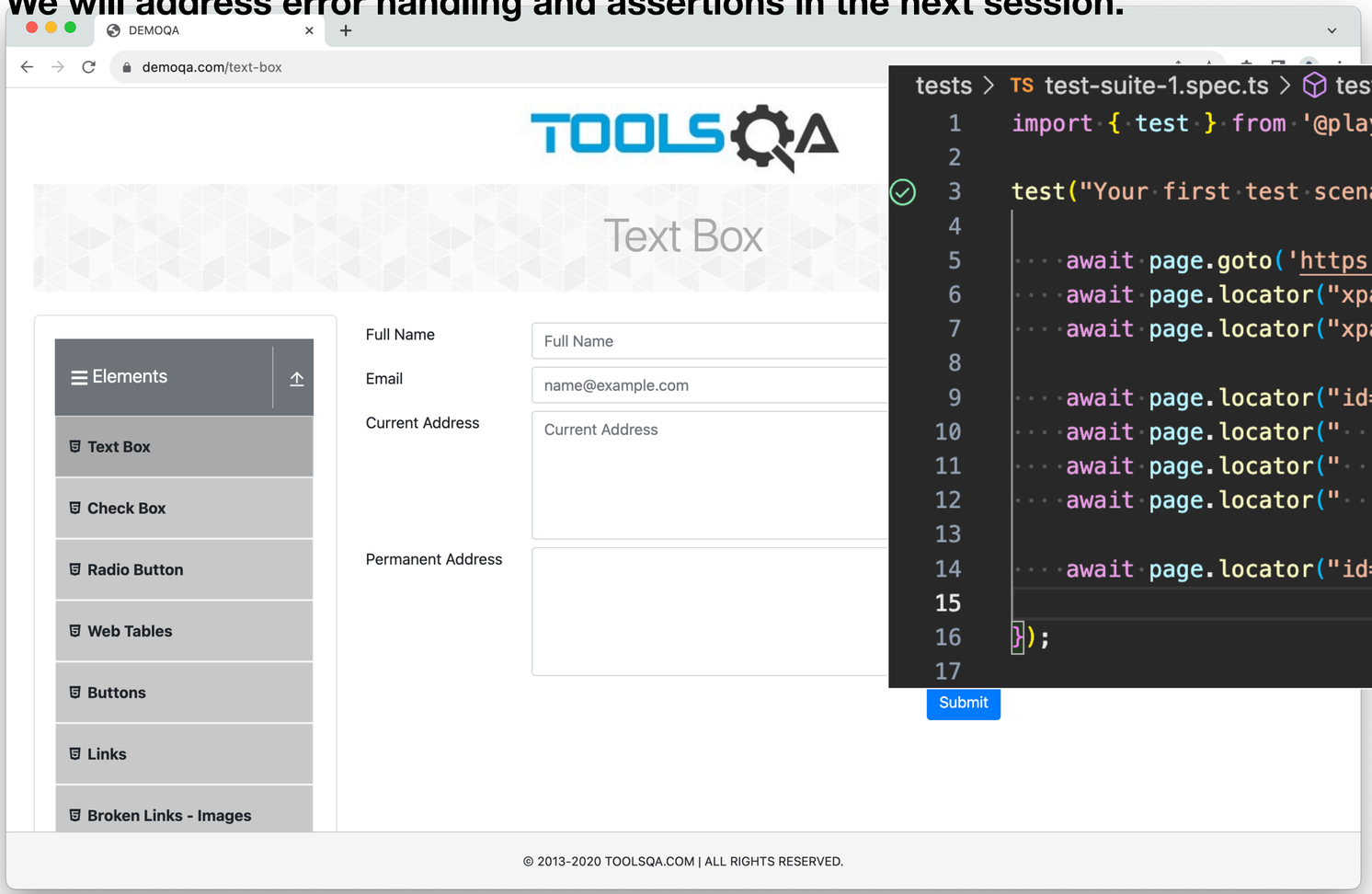
Your test go through that website as well

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.

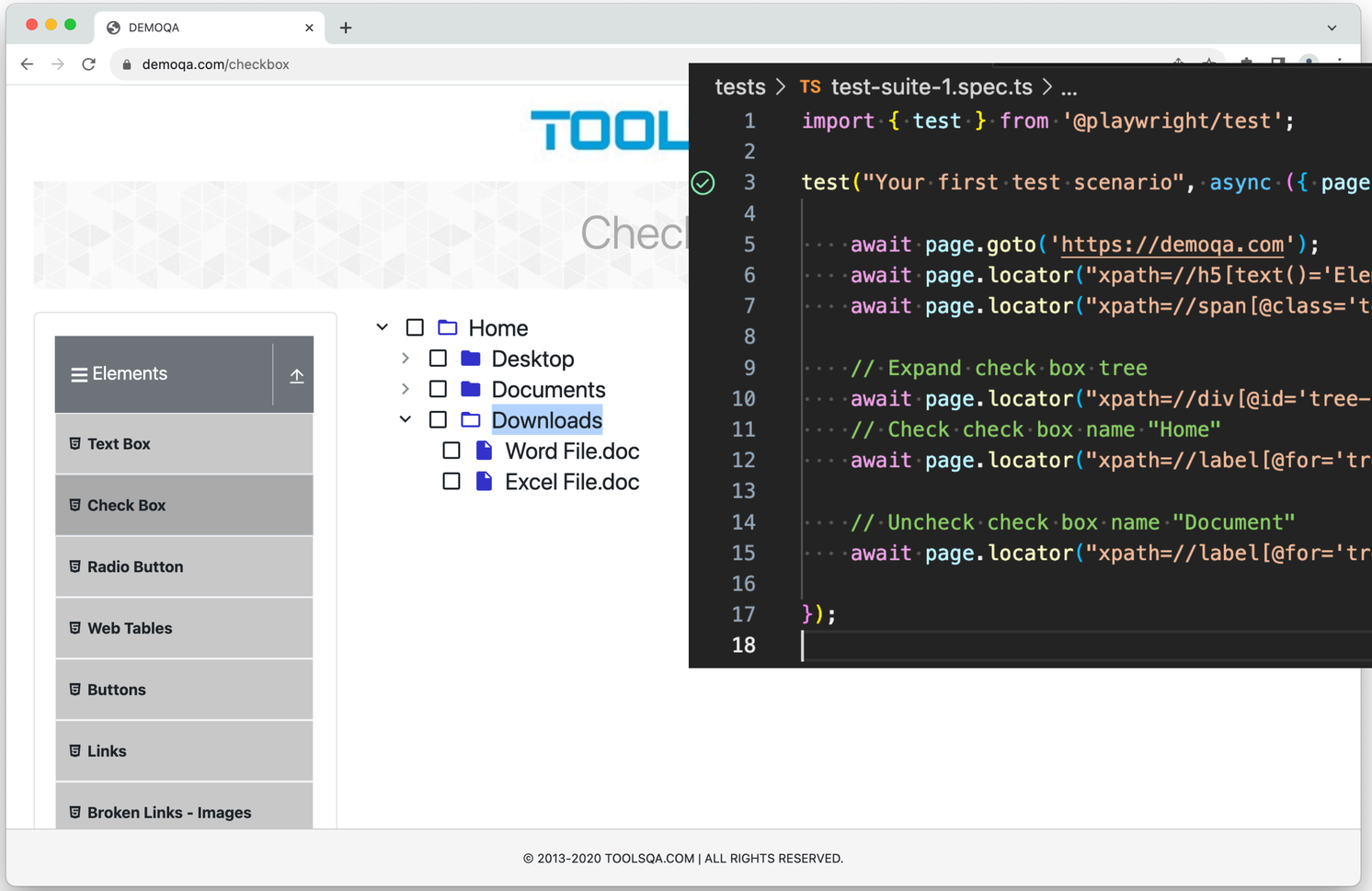



```
tests > TS test-suite-1.spec.ts > test("Your first test scenario") callback
1  import { test } from '@playwright/test';
2
3  test("Your first test scenario", async ({ page }) => {
4
5      await page.goto('https://demoqa.com');
6      await page.locator("xpath=//h5[text()='Elements']").click();
7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

Ref1: https://playwright.dev/docs/input

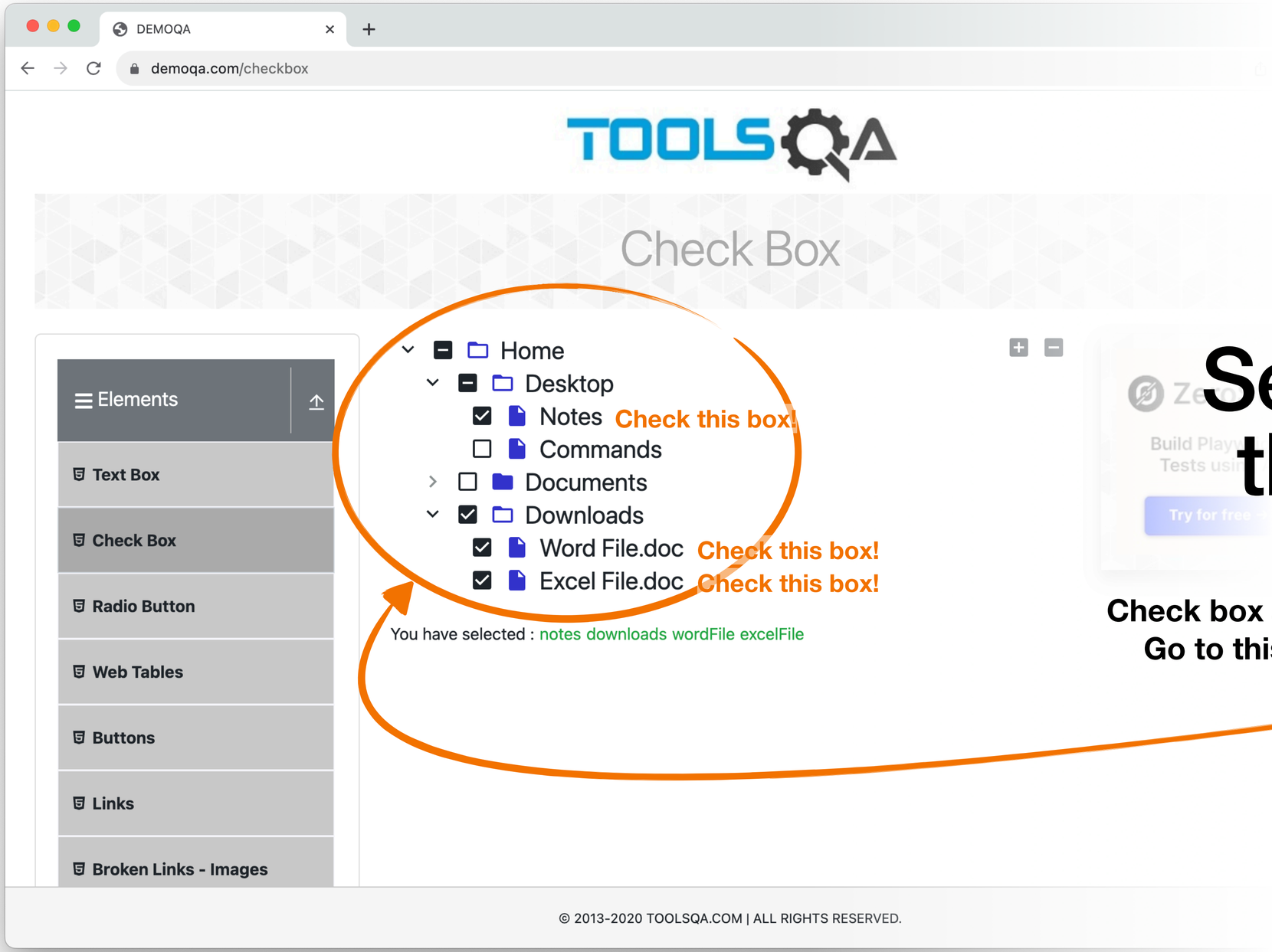
Checkbox

>> Elements > Check Box
Check! Check! Check! Uncheck! Uncheck!




```
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
```

Ref1: https://playwright.dev/docs/input

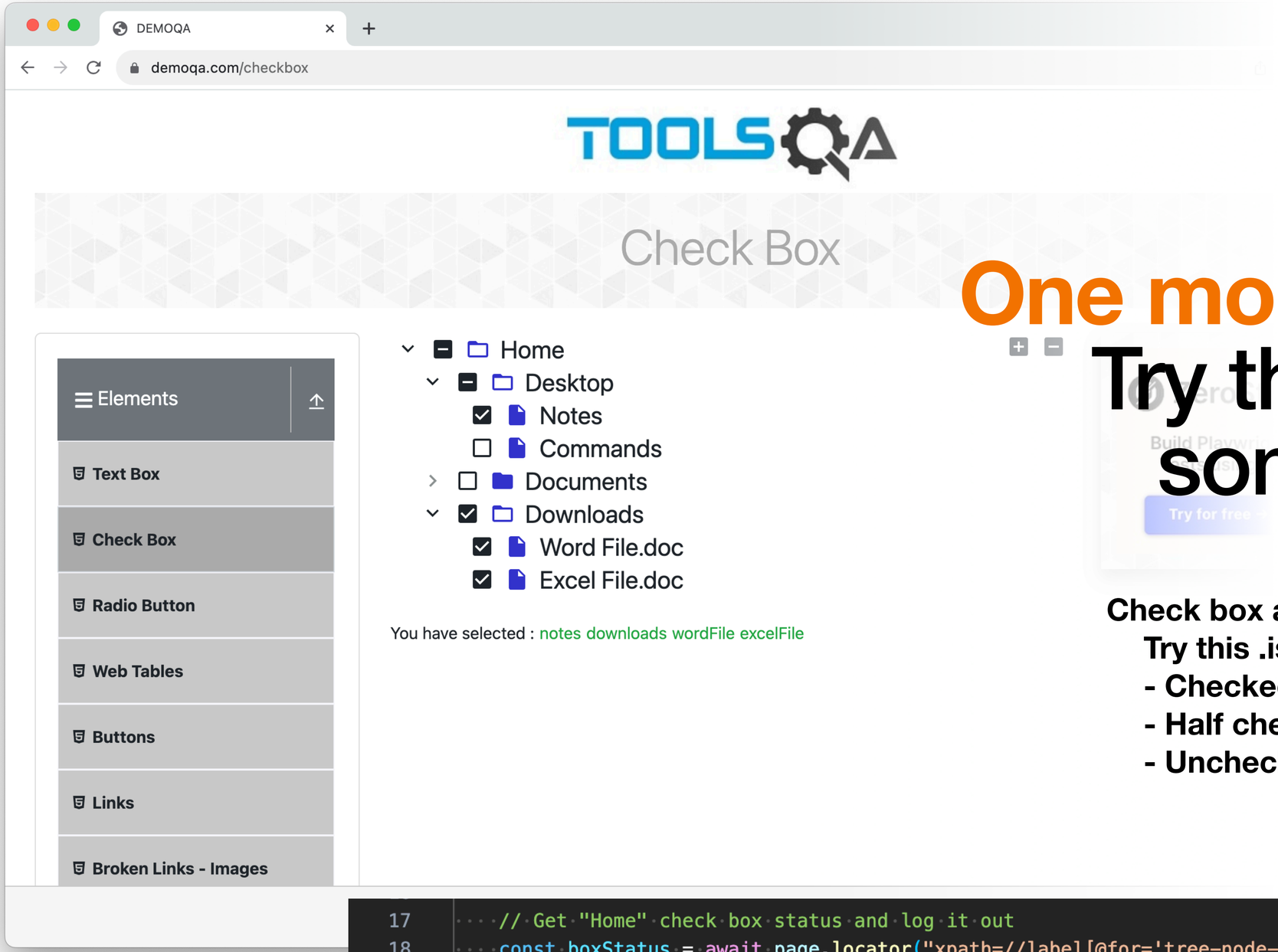


YOUR TURN!

Seem this page is the best practice

Check box assignment 1:

Go to this page then, find and check their boxes.



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

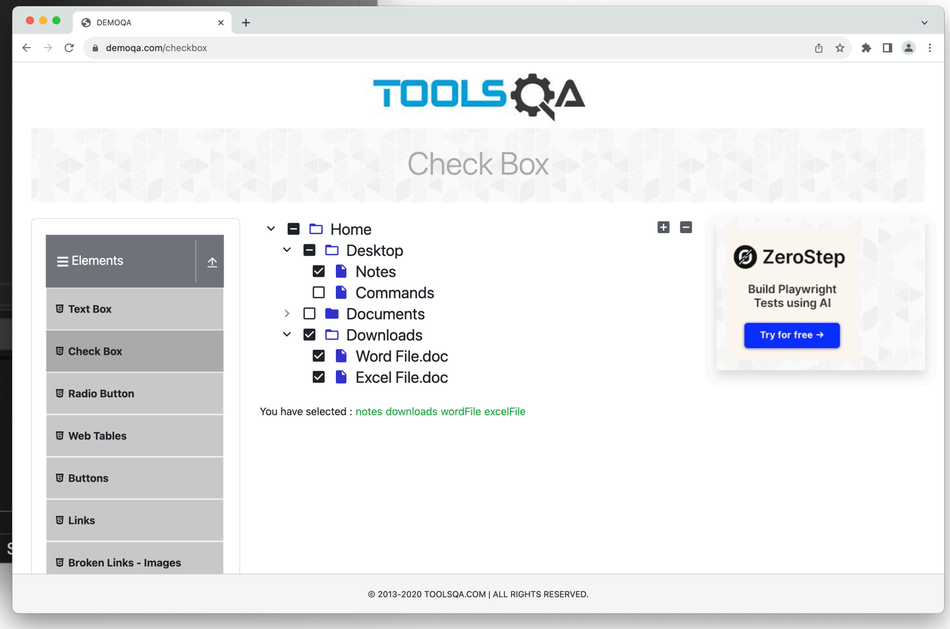
Add this code next to the comment
// Uncheck check box name "Document"

```
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

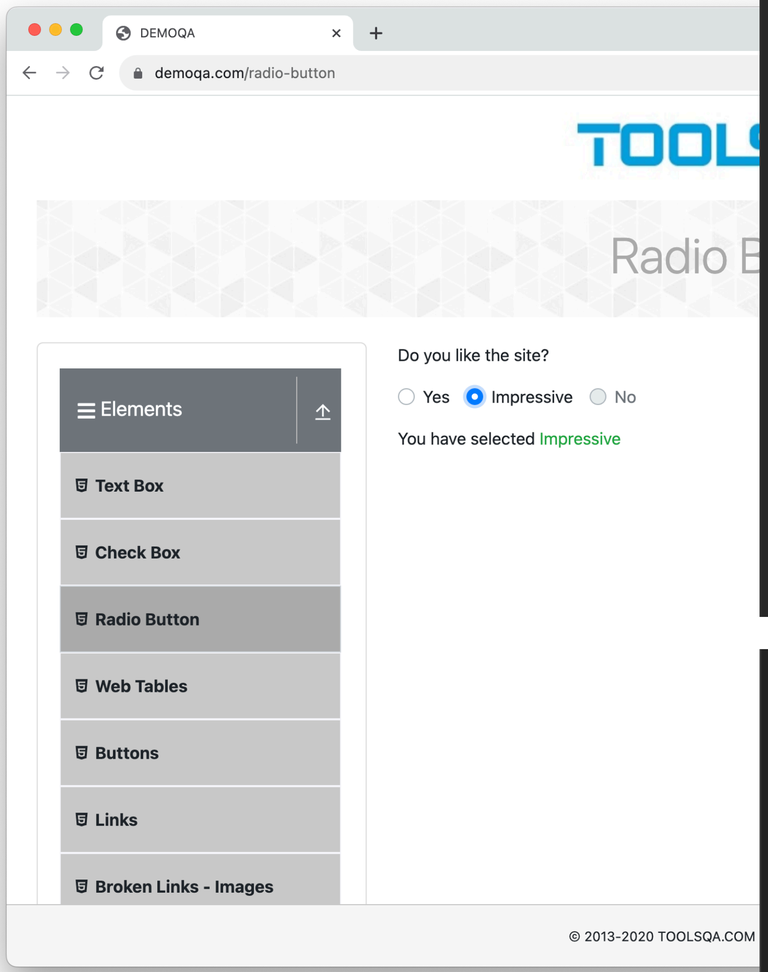
```
test("Your first test scenario", async ({ page }) => {

    await page.goto('https://demoqa.com'); – 6585ms
    await page.locator("xpath=//h5[text()='Elements']").click(); – 194ms
    await page.locator("xpath=//span[@class='text' and text()='Check Box']").click(); – 469ms

    // Expand check box tree
    await page.locator("xpath=//div[@id='tree-node']/ol/li/span/button").click(); – 50ms
    // Check check box name "Home"
    await page.locator("xpath=//label[@for='tree-node-home']//span[@class='rct-checkbox']").check(); – 45ms

    // Uncheck check box name "Document"
    await page.locator("xpath=//label[@for='tree-node-documents']//span[@class='rct-checkbox']").uncheck(); – 48ms

    // Get "Home" check box status and log it out
    const homeBoxStatus = await page.locator("xpath=//label[@for='tree-node-home']//span[@class='rct-checkbox']").isChecked(); – 3ms
    console.log("Home check box status: ", homeBoxStatus);

    // Expand check box "Desktop" tree
    await page.locator("xpath=//label[@for='tree-node-desktop']/../button").click(); – 29ms
    await page.locator("xpath=//label[@for='tree-node-notes']//span[@class='rct-checkbox']").check(); – 5ms
    await page.locator("xpath=//label[@for='tree-node-commands']//span[@class='rct-checkbox']").uncheck(); – 25ms

    // Expand check box "Downloads" tree
    await page.locator("xpath=//label[@for='tree-node-downloads']/../button").click(); – 35ms
    await page.locator("xpath=//label[@for='tree-node-wordFile']//span[@class='rct-checkbox']").check(); – 4ms
    await page.locator("xpath=//label[@for='tree-node-excelFile']//span[@class='rct-checkbox']").check(); – 3ms

    const desktopBoxStatus = await page.locator("xpath=//label[@for='tree-node-notes']//span[@class='rct-checkbox']").isChecked(); – 2ms
    console.log("Desktop check box status: ", desktopBoxStatus);

    const commandsBoxStatus = await page.locator("xpath=//label[@for='tree-node-commands']//span[@class='rct-checkbox']").isChecked(); – 6ms
    console.log("Commands check box status: ", commandsBoxStatus);

});
```

```
-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)
```



Radio button

>> Elements >> Radio Button
You can select only one of it.



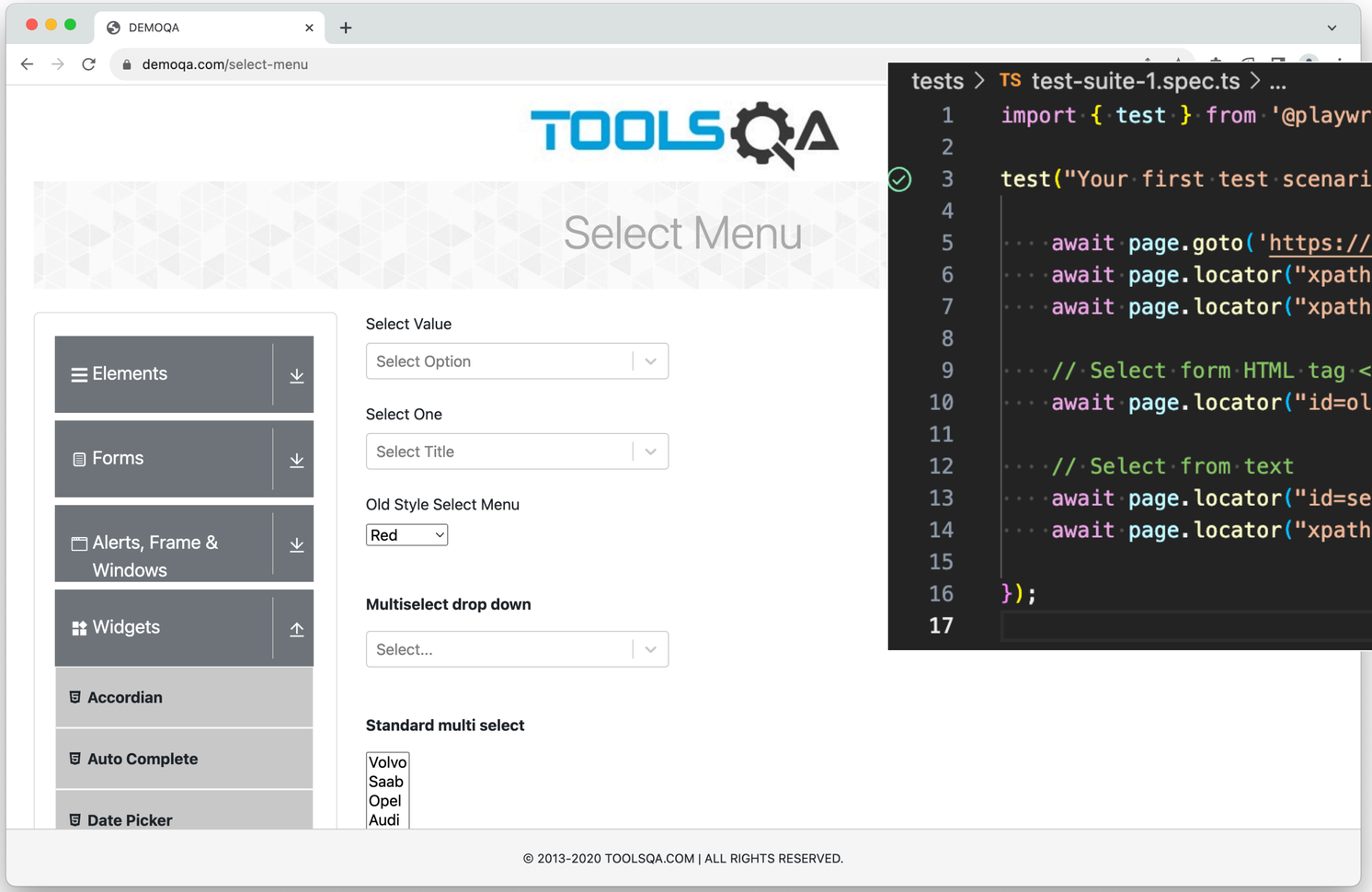

```
tests > TS test-suite-1.spec.ts > ...
1  import { test } from '@playwright/test';
2
3  test("Your first test scenario", async ({ page }) => {
4
5      await page.goto('https://demoqa.com');
6      await page.locator("xpath=//h5[text()='Elements']").click();
7      await page.locator("xpath=//span[@class='text'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

Ref1: https://playwright.dev/docs/input

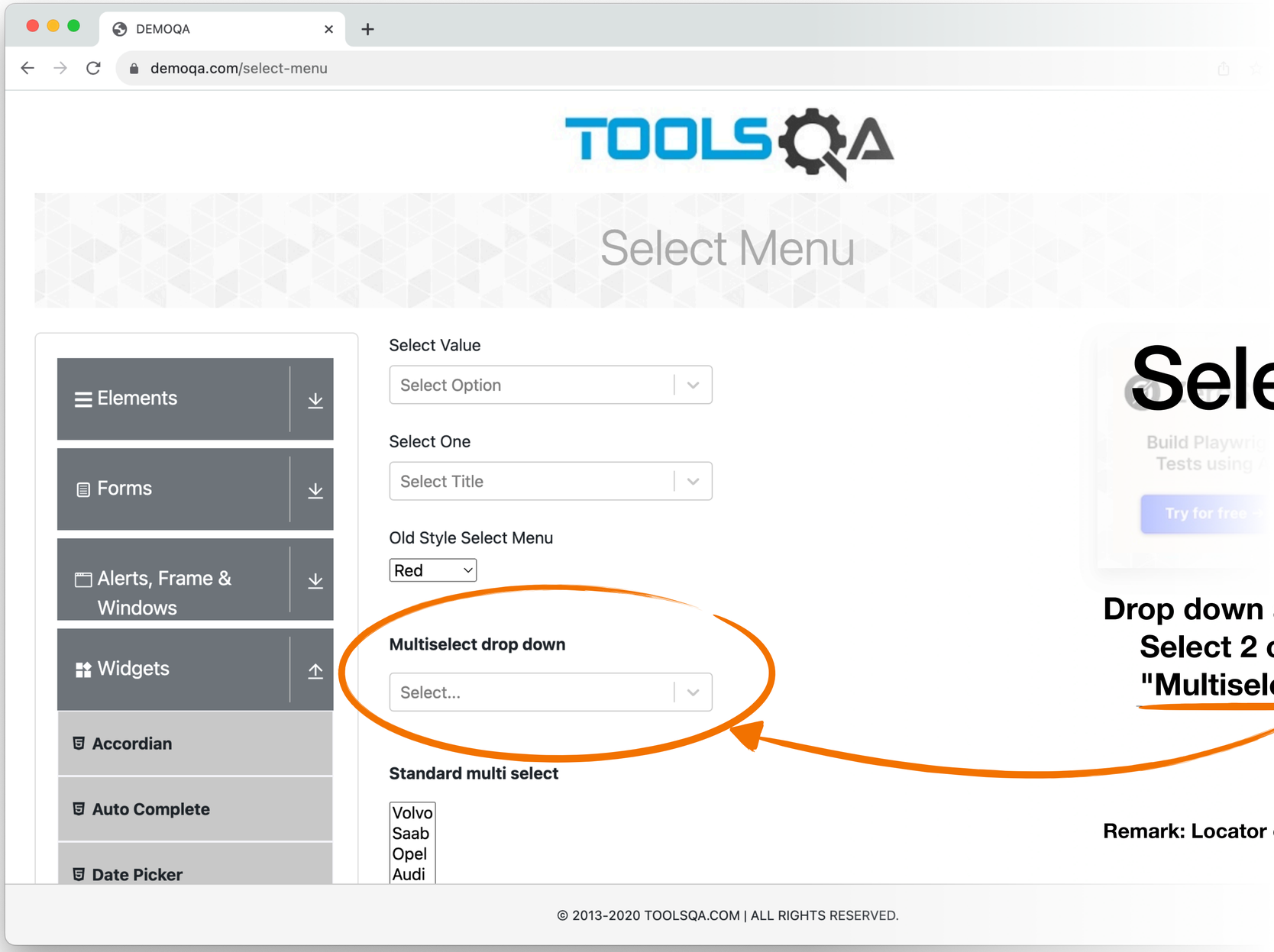
Dropdown selection

>> Widgets > Select Menu




```
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
```

Ref1: https://playwright.dev/docs/input



Challenge

Select options from that dropdown

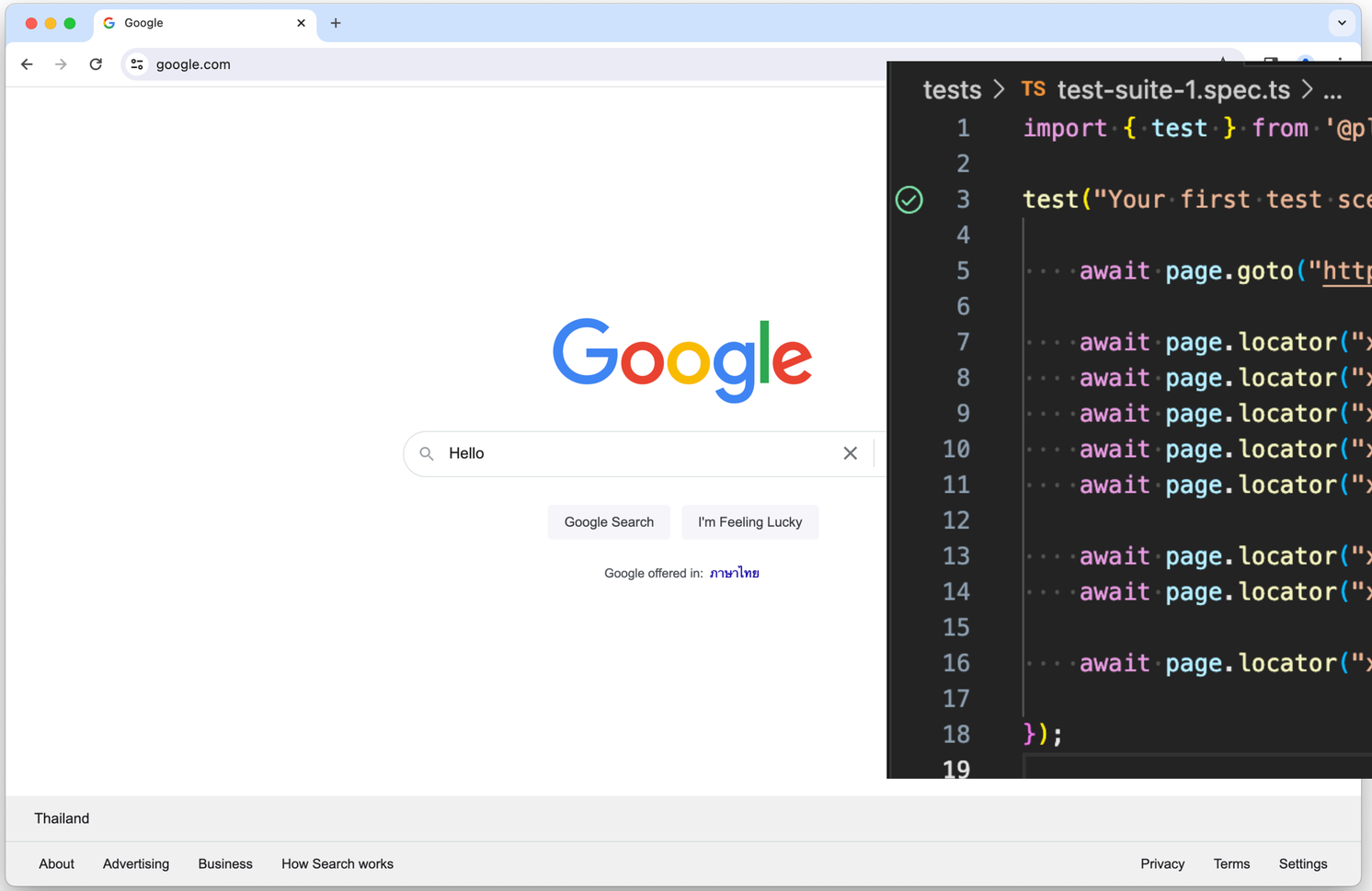
Drop down assignment 1:
Select 2 or more options from "Multiselect drop down"

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.

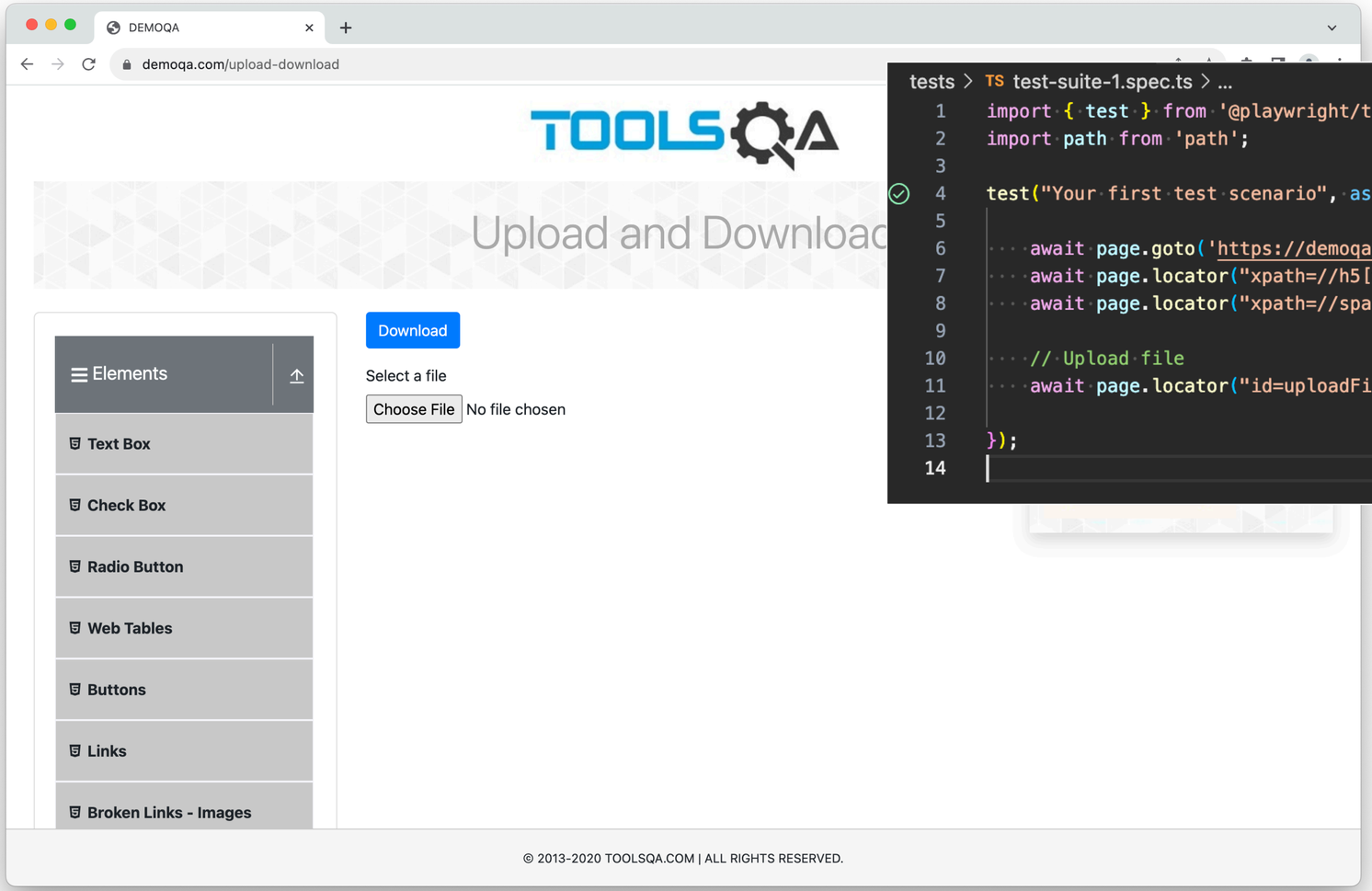



```
tests > TS test-suite-1.spec.ts > ...
1   import { test } from '@playwright/test';
2
3   test("Your first test scenario", async ({ page }) => {
4
5       await page.goto("https://google.com");
6
7       await page.locator("xpath=//textarea[@role='combobox']").press("H");
8       await page.locator("xpath=//textarea[@role='combobox']").press("e");
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
```

Ref1: https://playwright.dev/docs/input

Upload file

>> Elements > Upload and Download



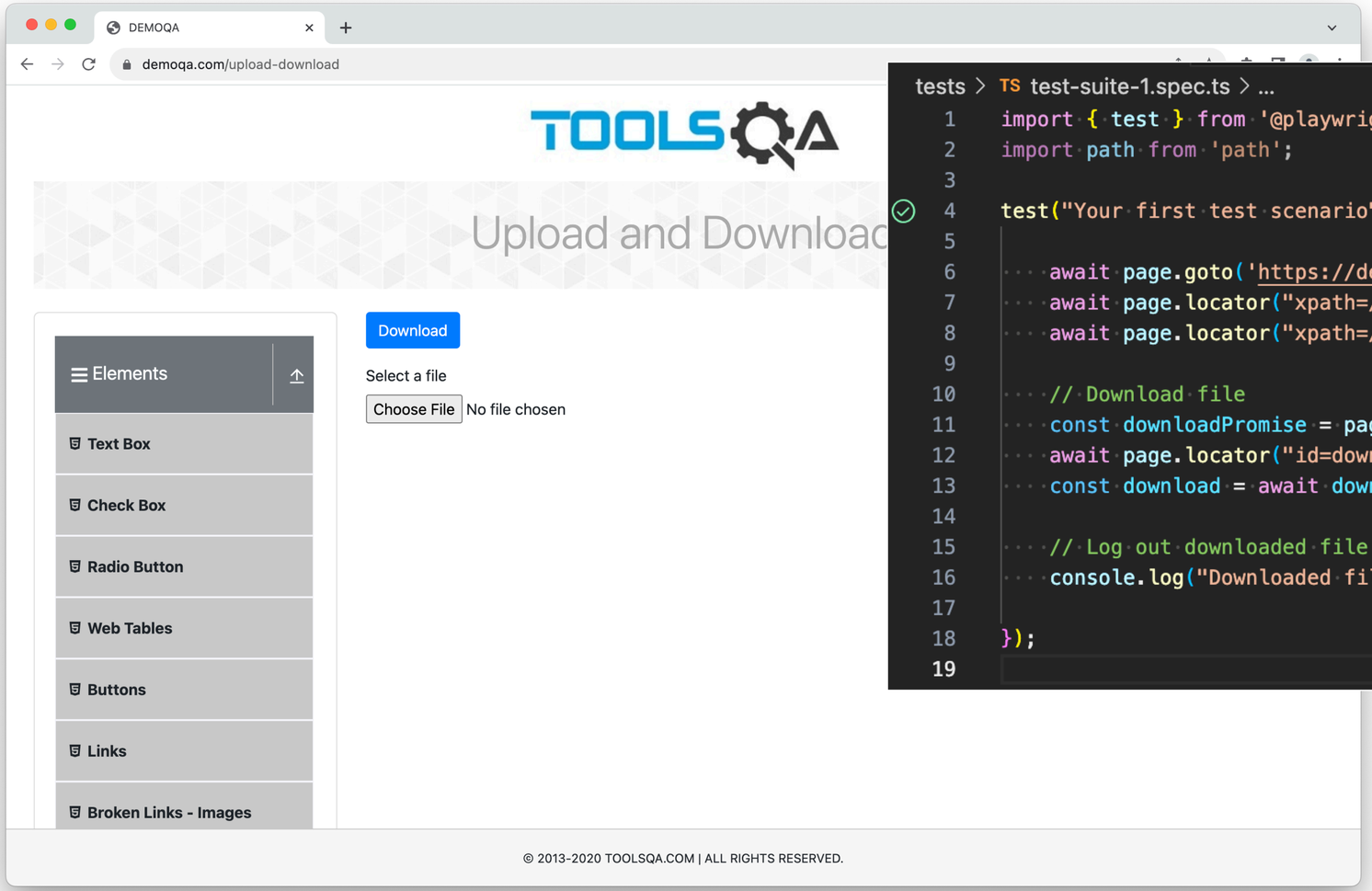

```
tests > TS test-suite-1.spec.ts > ...
1  import { test } from '@playwright/test';
2  import path from 'path';
3
4  test("Your first test scenario", async ({ page }) => {
5
6      await page.goto('https://demoqa.com');
7      await page.locator("xpath=//h5[text()='Elements']").click();
8      await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

__dirname : Path of the current directory file that is running

Ref1: https://playwright.dev/docs/input

Download file

>> Elements > Upload and Download



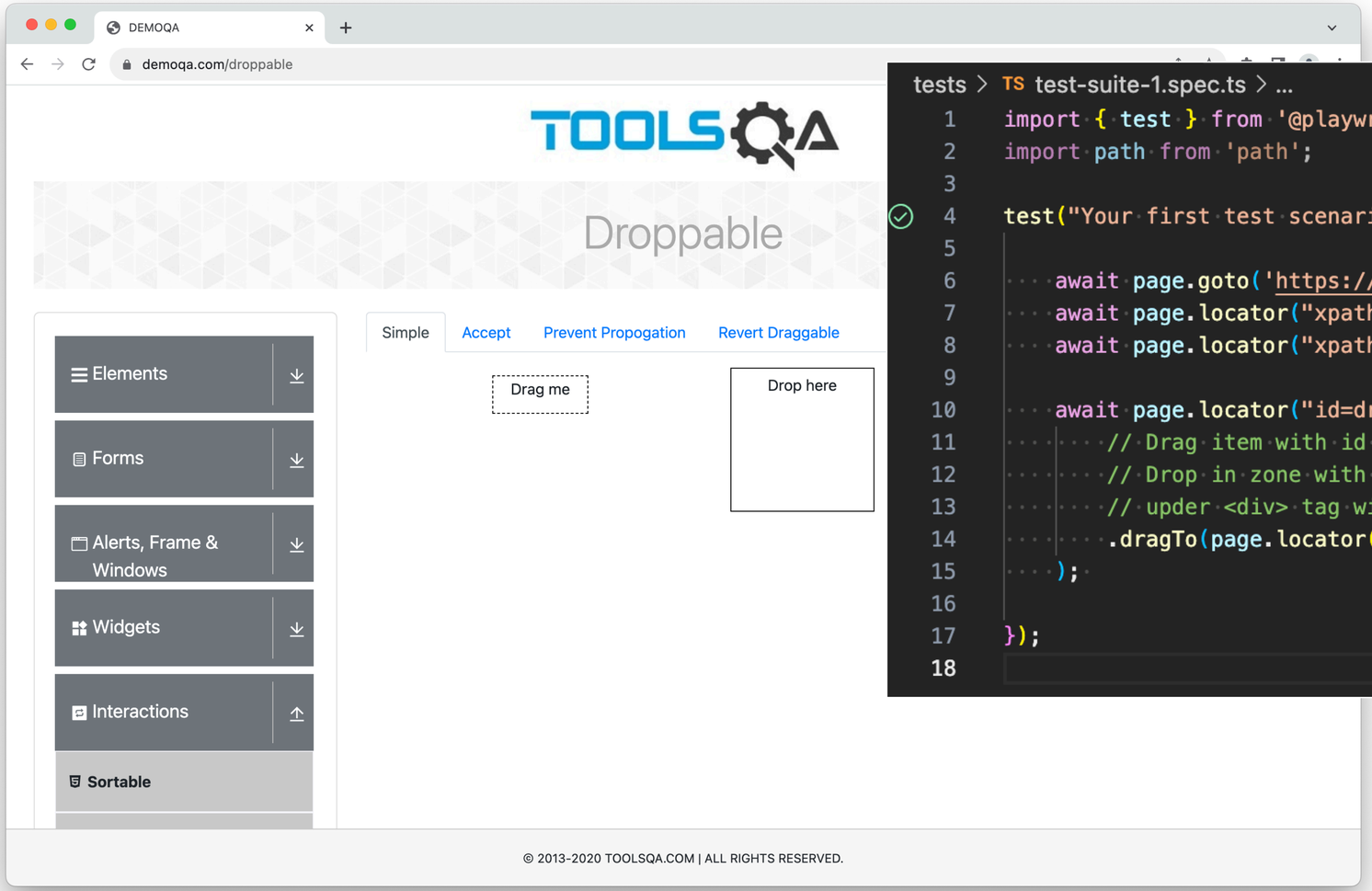
tests > TS test-suite-1.spec.ts > ...

```
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
```

Ref1: https://playwright.dev/docs/downloads

Drag and drop

>> Interactions > Droppable
Drag me to drop zone plz.




```
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         // uper <div> tag with class "simple-drop-container"
14         .dragTo(page.locator("css=div.simple-drop-container >> id=droppable")
15     );
16
17 });
18
```

Ref1: https://playwright.dev/docs/input

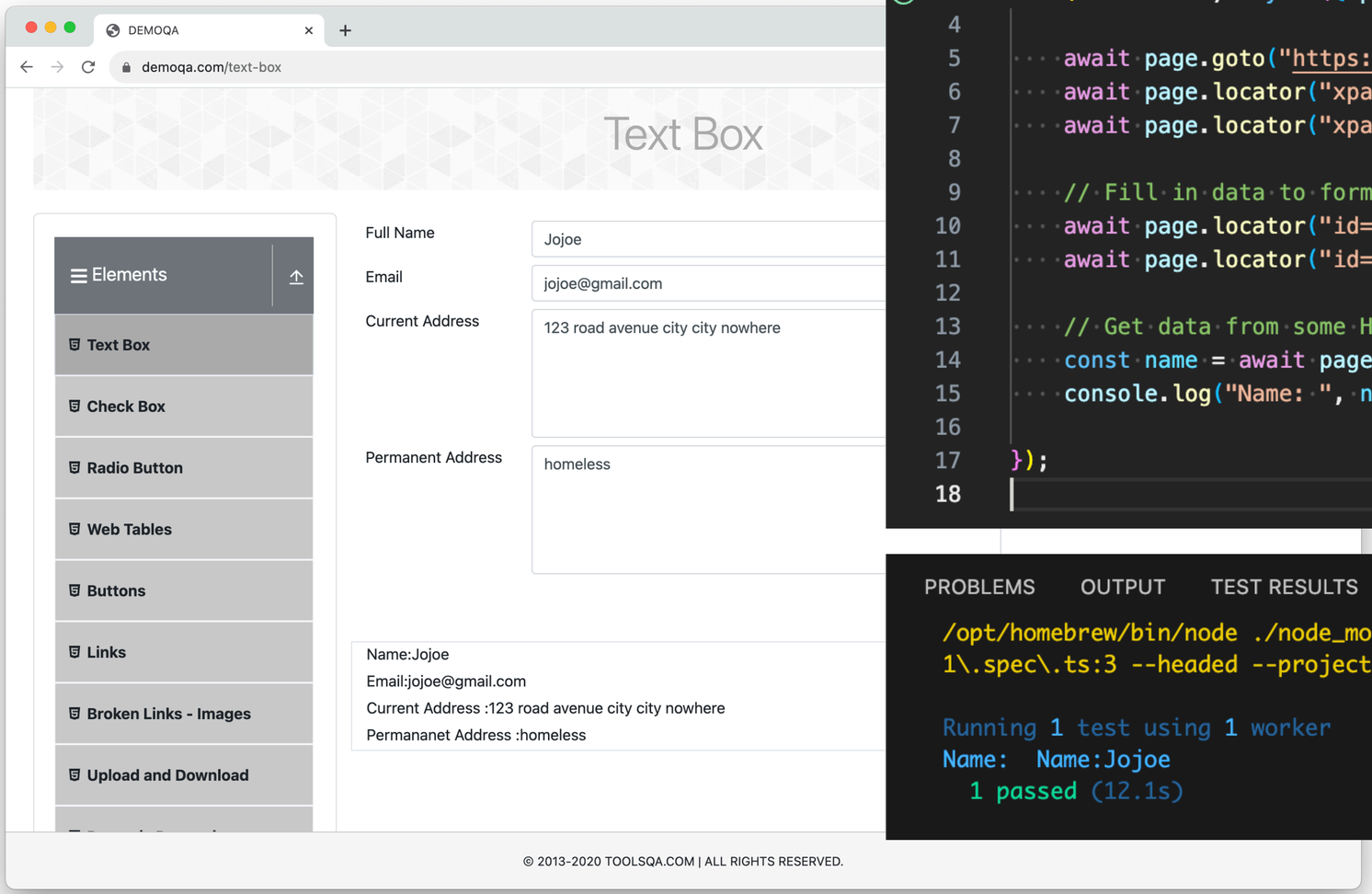
Scraping data

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

Ref1: https://playwright.dev/docs/api/class-elementhandle

Get data from web

>> Elements > Text Box

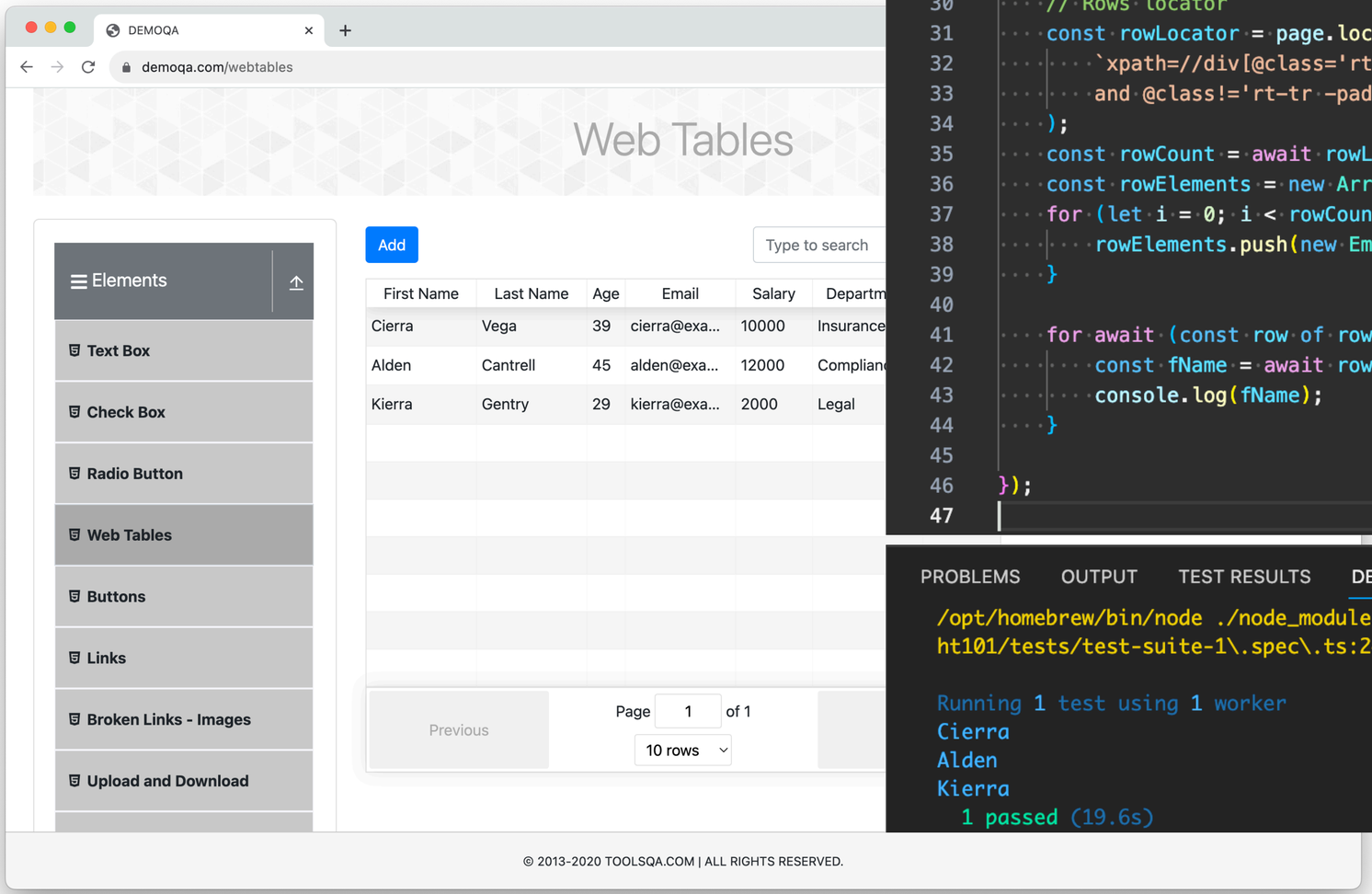



```
tests > TS test-suite-1.spec.ts > ...
1  import { test } from '@playwright/test';
2
3  test("Get data", async ({ page }) => {
4
5      await page.goto("https://demoqa.com");
6      await page.locator("xpath=//h5[text()='Elements']").click();
7      await page.locator("xpath=//span[@class='text' and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




```
tests > TS test-suite-1.spec.ts > ...
23
24  test("Get data", async ({ page }) => {
25
26      await page.goto("https://demoqa.com");
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'
33          and @class!='rt-tr -padRow -odd')]`
34      );
35      const rowCount = await rowLocator.count();
36      const rowElements = new Array<EmployeeRowElement>(); // Array for collect rows locator
37      for (let i = 0; i < rowCount; i++) {
38          rowElements.push(new EmployeeRowElement(page, rowLocator.nth(i))); // Get each row by index
39      }
40
41      for await (const row of rowElements) {
42          const fName = await row.getFirstName(); // Get data from row
43          console.log(fName);
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/jojoe/Re
ht101/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 > ...

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

class BaseElement {

    protected page: Page;
    protected elementLocator: Locator;
    constructor(page: Page, elementLocator: Locator) {
        this.page = page;
        this.elementLocator = elementLocator;
    }
}

class EmployeeRowElement extends BaseElement {

    private _fName: Locator = this.elementLocator.locator("xpath=//div[@role='gridcell'][1]");
    private _delete: Locator = this.elementLocator.locator("xpath=//div[@role='gridcell'][8]");

    public getFirstName() {
        return this._fName.innerText();
    }

}
```

Get current page url

tests > TS test-suite-1.spec.ts > ...

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

test("Get data", async ({ page }) => {

    await page.goto("https://demoqa.com");
    await page.locator("xpath=//h5[text()='Elements']").click();

    // Get current page url
    const currentUrl = page.url();
    console.log("Current page: ", currentUrl);

});
```

PROBLEMS OUTPUT TEST RESULTS DEBUG CONSOLE TERMINAL PORTS ROBO

```
/opt/homebrew/bin/node ./node_modules/@playwright/test/cli.js test -c playw
ht101/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)
```

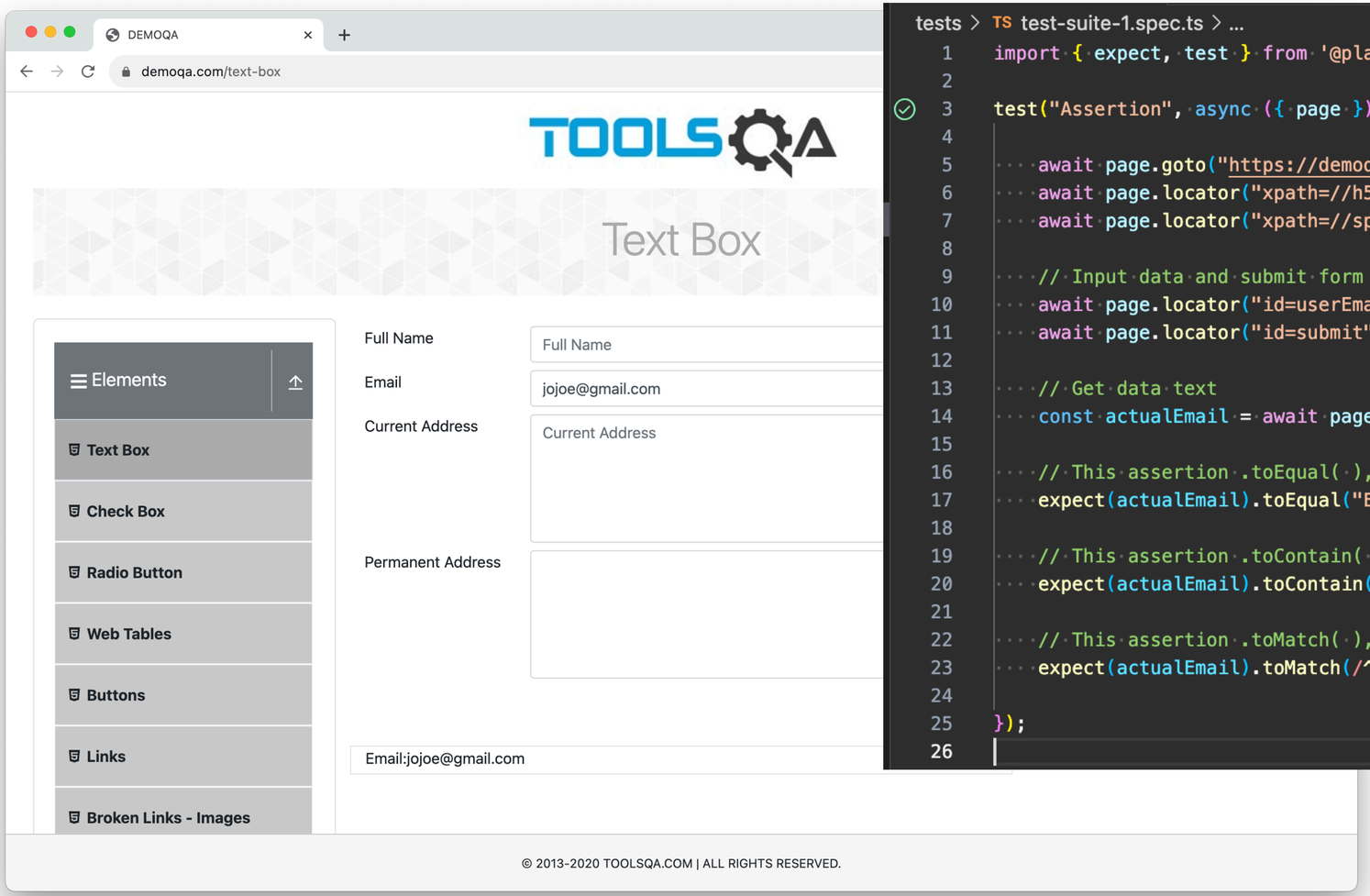
Assertion

- Text
- Number
- Date and time

Ref1: https://playwright.dev/docs/test-assertions

Text assertion

>> Elements > Text Box




```
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
6       const num = 123.55;
7
8       expect(num).toEqual(123.55); — 2ms
9
10      expect(num).toBeLessThanOrEqual(150); — 0ms
11
12      expect(num).toBeGreaterThanOrEqual(100); — 0ms
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

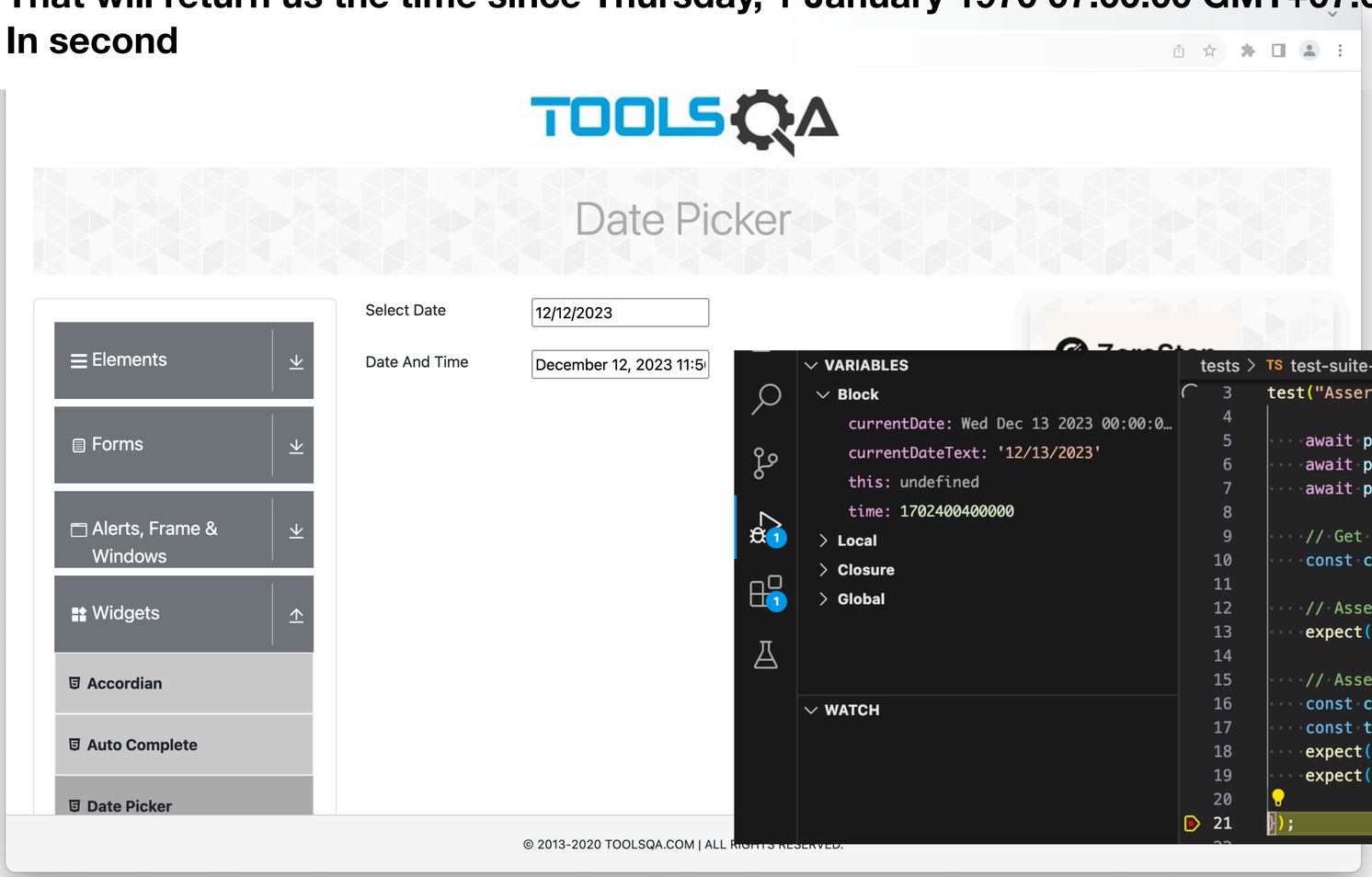
Test steps:

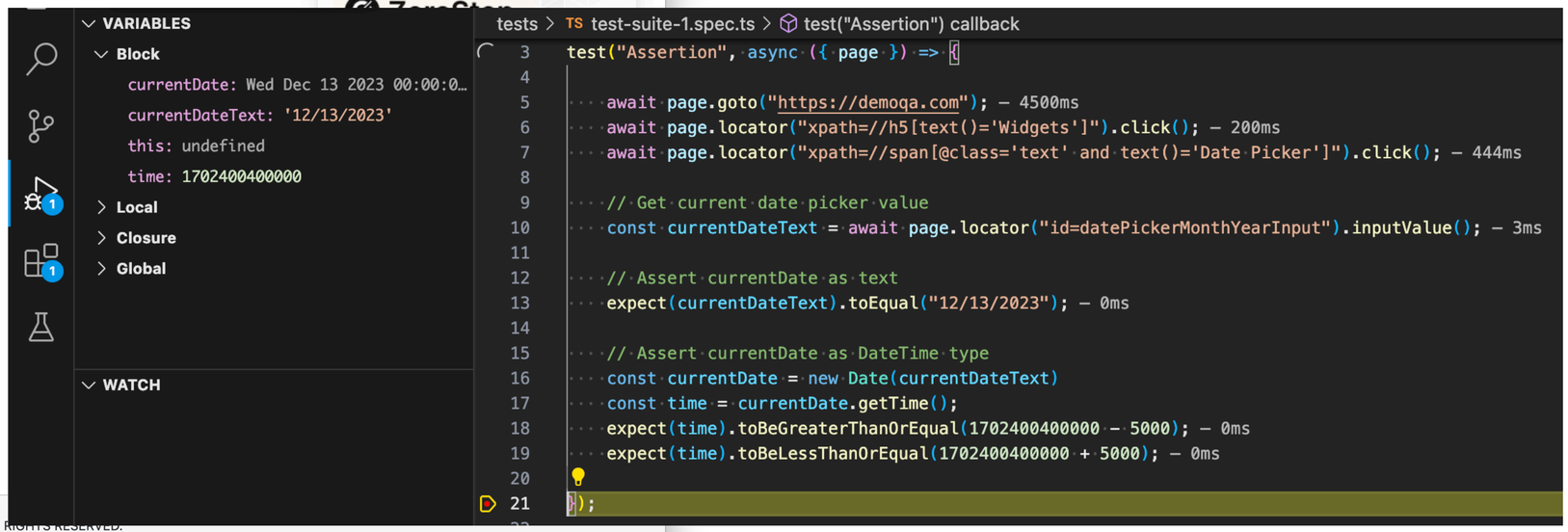
Line16 Convert currentDateText from string to DateTime format

Line17 Get time in second

Line18 Assert by expect the time in second must greater than or equal 1702400400000 - 5000

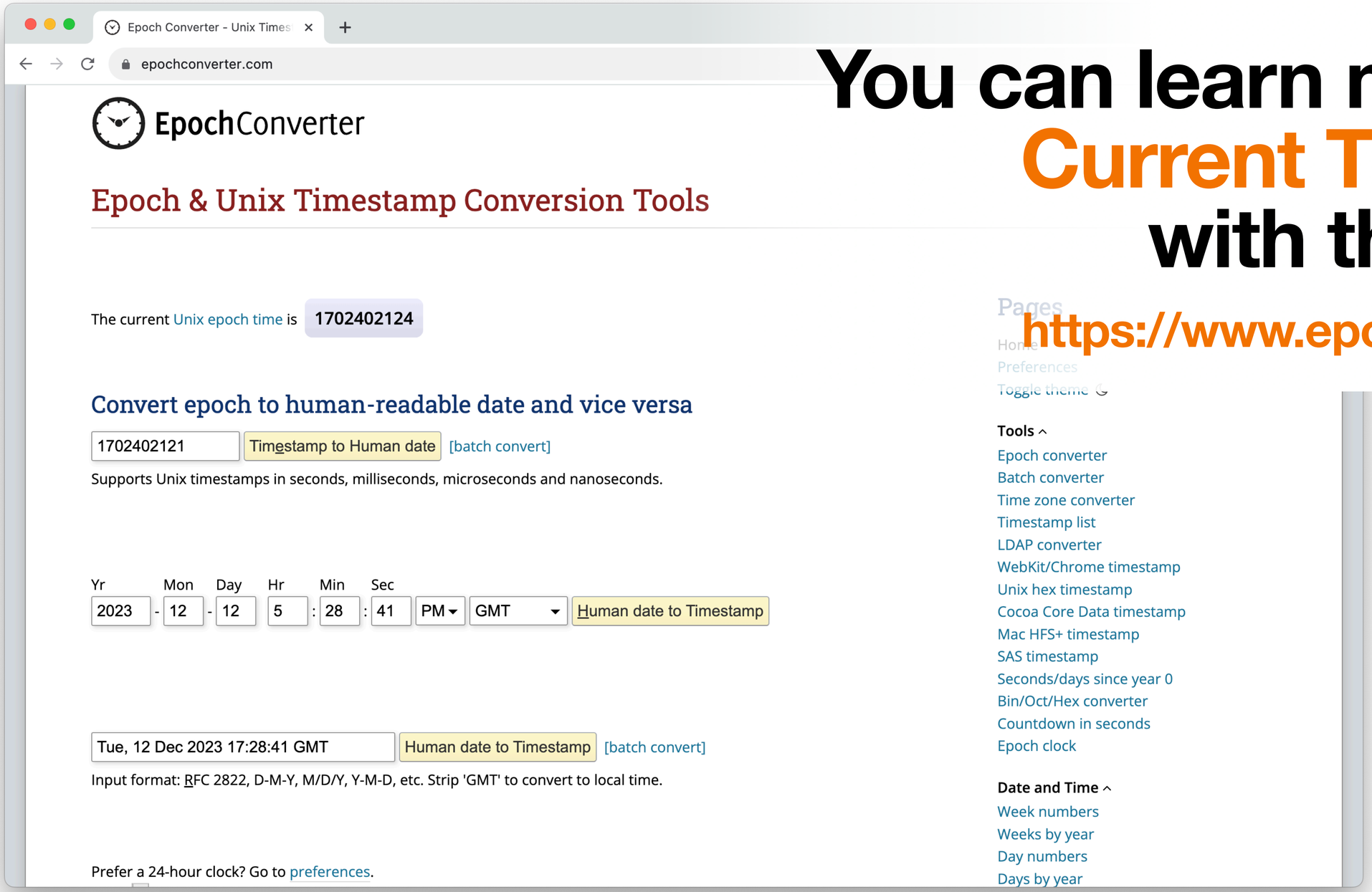
Line19 Assert by expect the time in second must less than or equal 1702400400000 - 5000





You can learn more about Current Time Stamp with this website

https://www.epochconverter.com/



Ref1: https://www.epochconverter.com/

Test suite

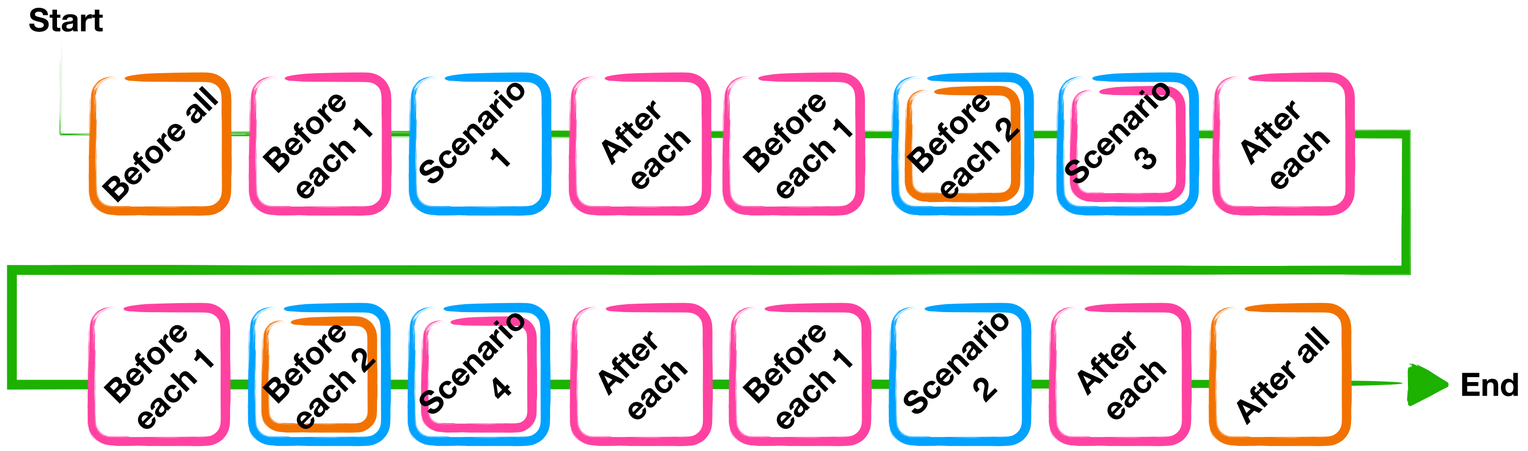
- Test setup / teardown
  - test.beforeAll( )
  - test.beforeEach( )
  - test.afterAll( )
  - test.afterEach( )
- Groups
  - test.describe( )
- Template

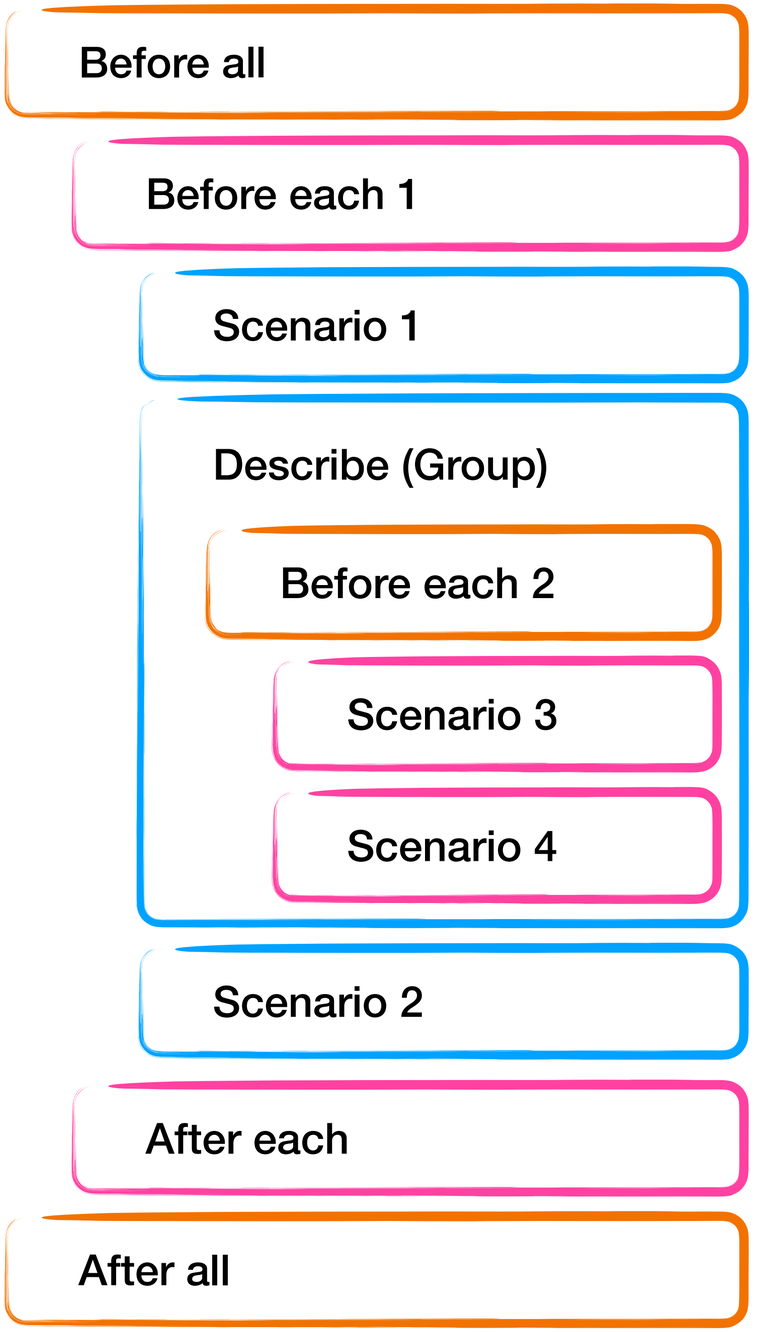
Ref1: https://playwright.dev/docs/api/class-test

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 > ...

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

test.beforeAll(async () => { – 9ms
    console.log("Before all");
});

test.beforeEach(async () => { – 6ms
    console.log("Before each 1");
});

test("Scenario 1", async () => {
    console.log("scenario 1");
});

test.describe("Describe", async () => {

    test.beforeEach(async () => { – 0ms
        console.log("Before each 2");
    });

    test("Scenario 3", async () => {
        console.log("scenario 3");
    });

    test("Scenario 4", async () => {
        console.log("scenario 4");
    });

});

test("Scenario 2", async () => {
    console.log("scenario 2");
});

test.afterEach(async () => { – 0ms
    console.log("After each");
});

test.afterAll(async () => { – 5ms
    console.log("After all");
});
```

Filter (e.g. text, !exclude)

```
/opt/homebrew/bin/node ./node_module
laywright.config.ts /Users/jojoe/Rep
tests/test-suite-1\.spec\.ts --heade
1 --retries 0 --timeout 0 --workers

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 () => { – 9ms
4      console.log("Before all");
5  });
6
7  test.beforeEach(async () => { – 6ms
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 () => { – 0ms
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
31 test("Scenario 2", async () => {
32     console.log("scenario 2");
33 });
34
35 test.afterEach(async () => { – 0ms
36     console.log("After each");
37 });
38
39 test.afterAll(async () => { – 5ms
40     console.log("After all");
41 });
42
```

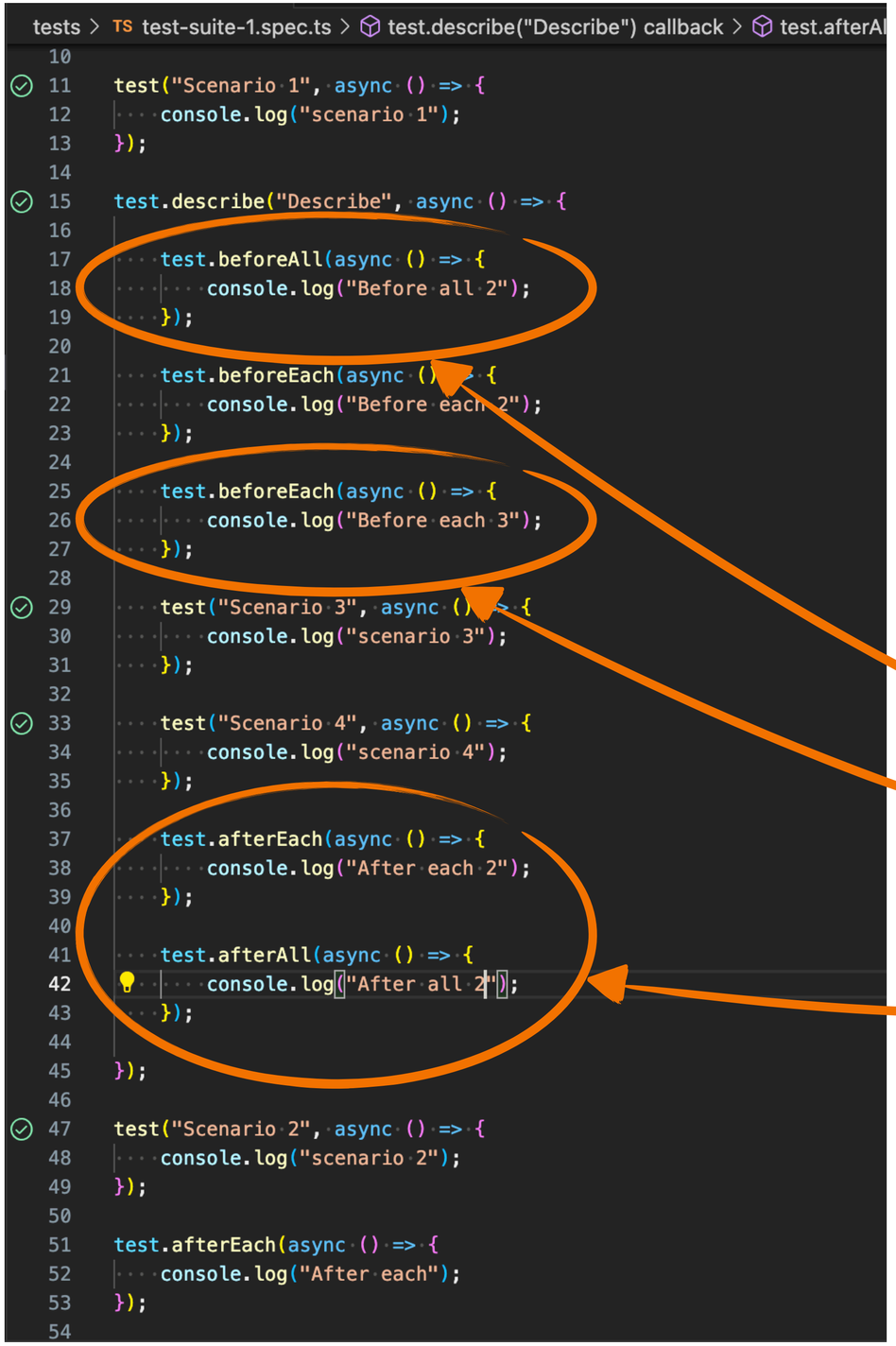
.beforeAll( ) will run at the beginning of the test suite execution

.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



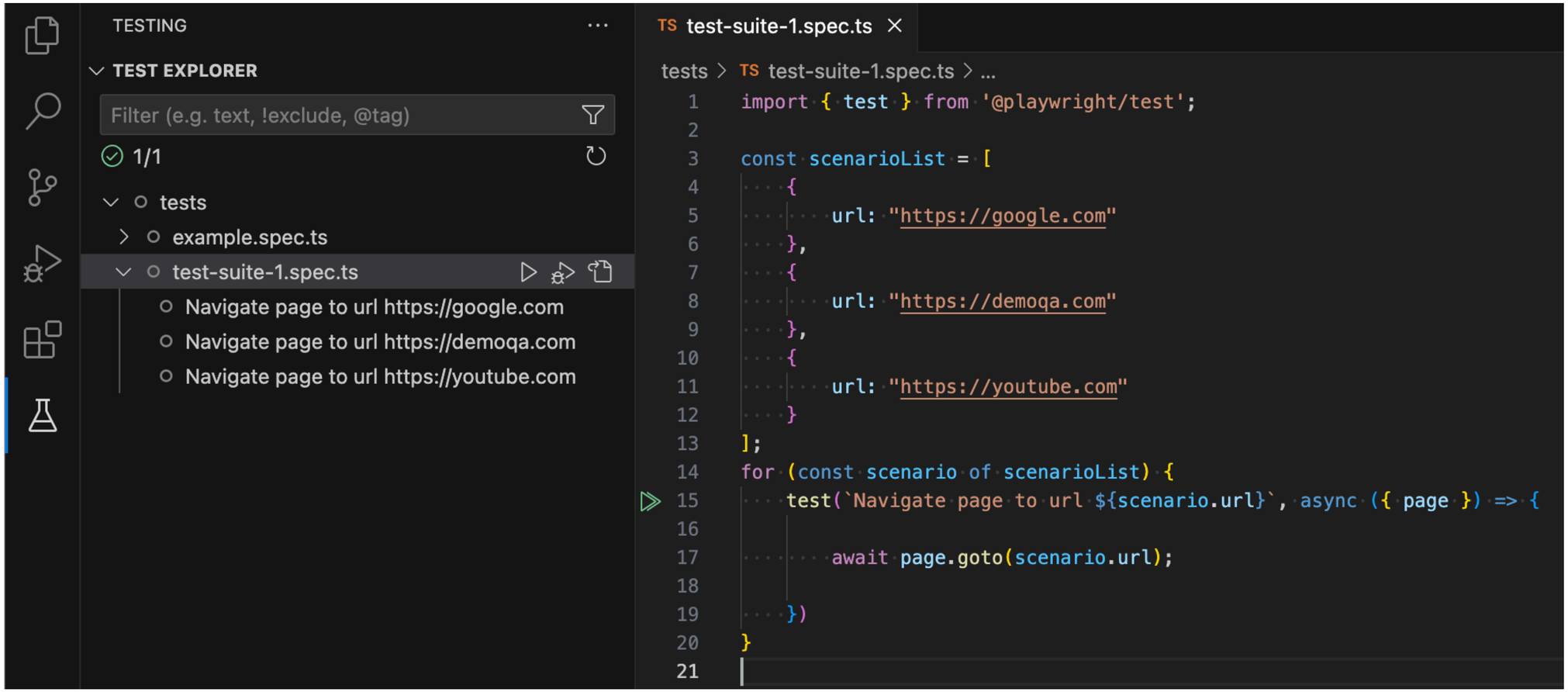
What if?

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



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

API tests

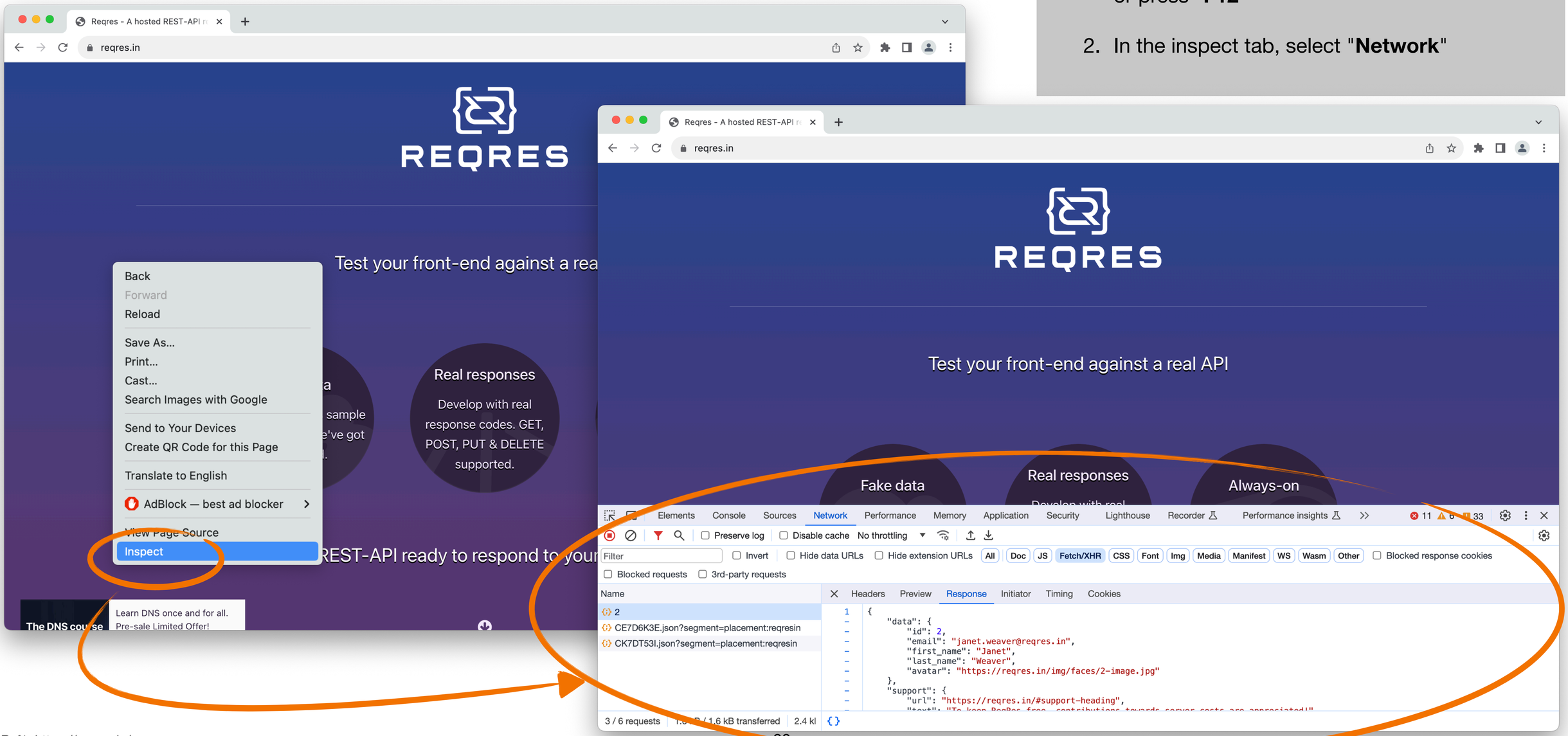
- Chrome inspect and network
- HTTP header
- Requests
  - GET
  - POST / PUT / PATCH
  - DELETE
- JSON parse

Ref1: https://playwright.dev/docs/api-testing

Chrome Inspect and Network

Open inspect and console steps:

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

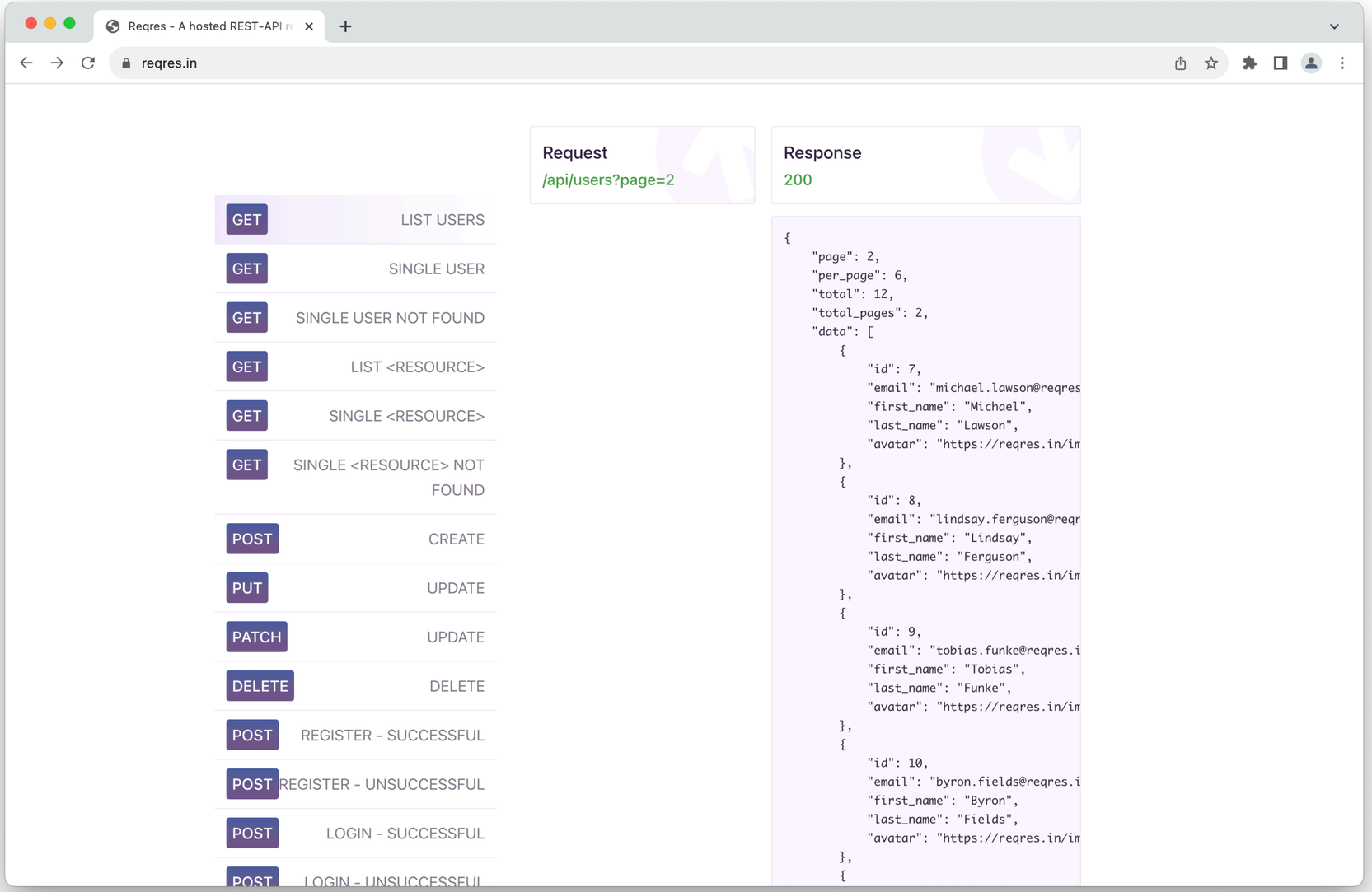


Ref1: https://reqres.in/

REQRES : New playground with fake data :/

Open Chrome browser

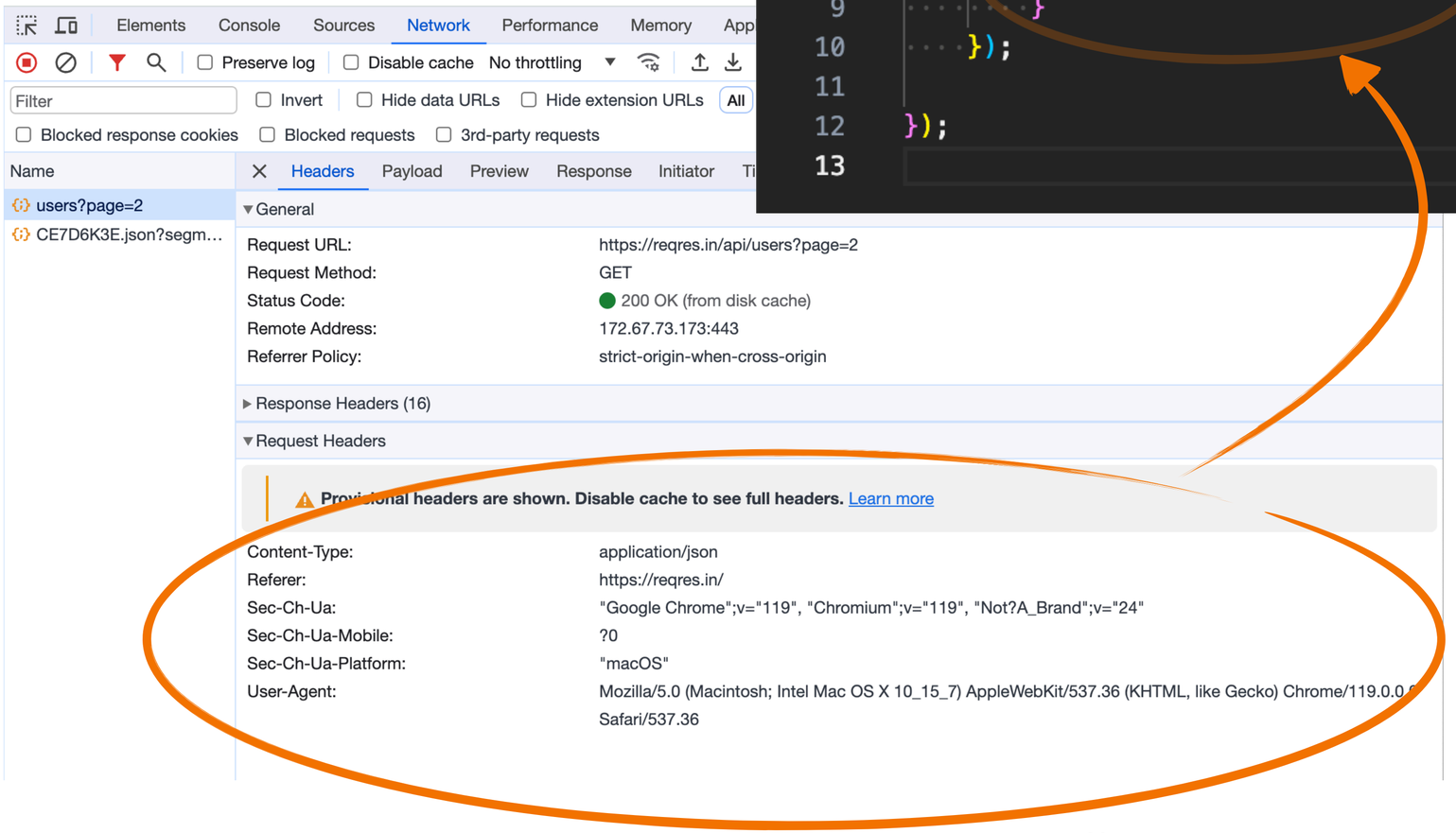
Go to this url: https://reqres.in/



Ref1: https://reqres.in/

HTTP header

```
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.

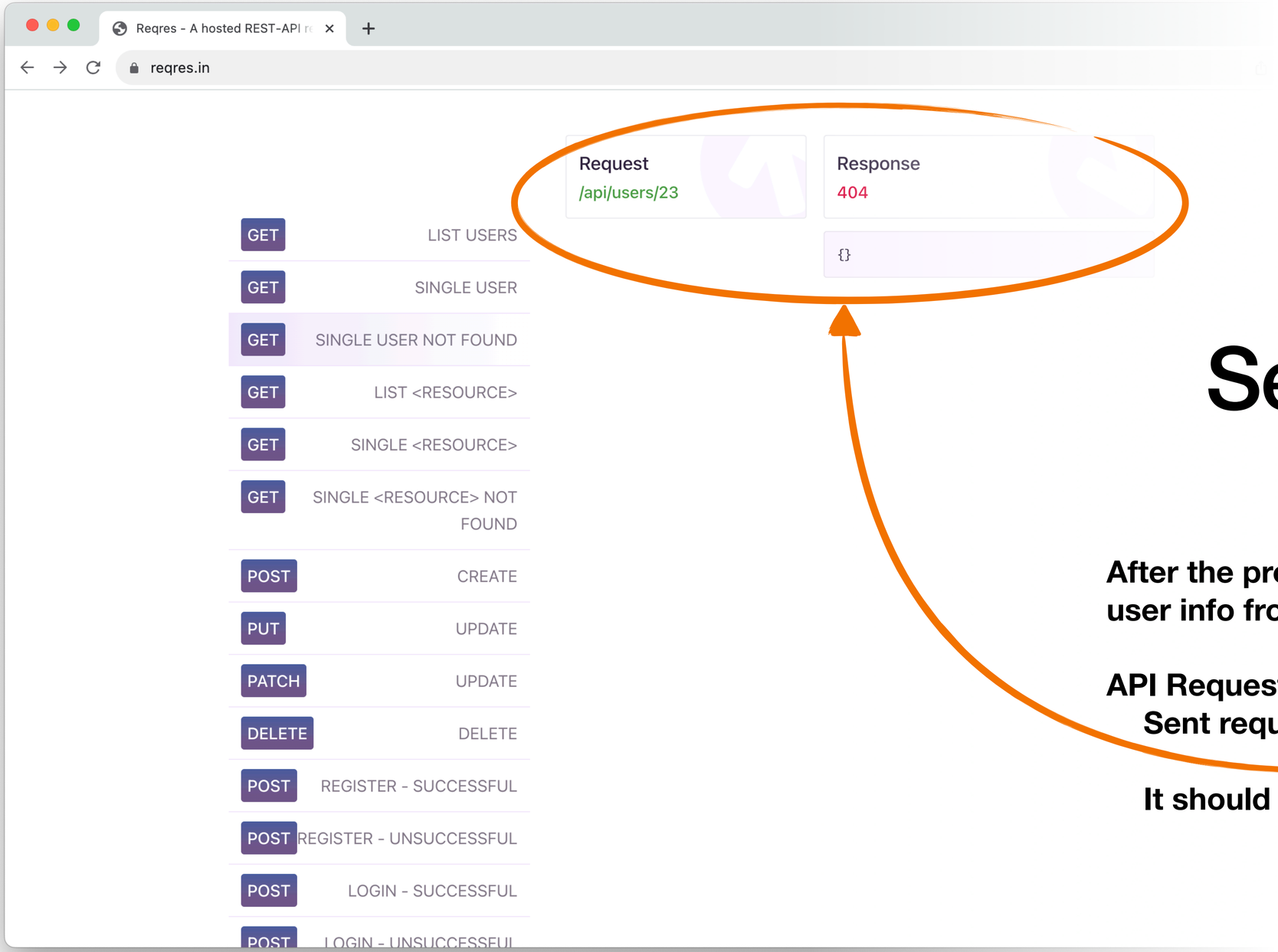
Request GET

```
tests > TS test-suite-1.spec.ts > ...
1   import { test, request } from '@playwright/test';
2
3   test("HTTP Request", async () => {
4
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/2
13      const httpResponse = await httpContext.get("https://reqres.in/api/users/2");
14      console.log(`HTTP request GET response code: ${httpResponse.status()} ${httpResponse.statusText()}`);
15
16  });
17
```

```
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/jojoe/Repositorie
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)
```

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



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: ${h
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 plo
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: ${ht
20
21  });
22
```

Test steps:

Line5 Create new HTTP context and add http header

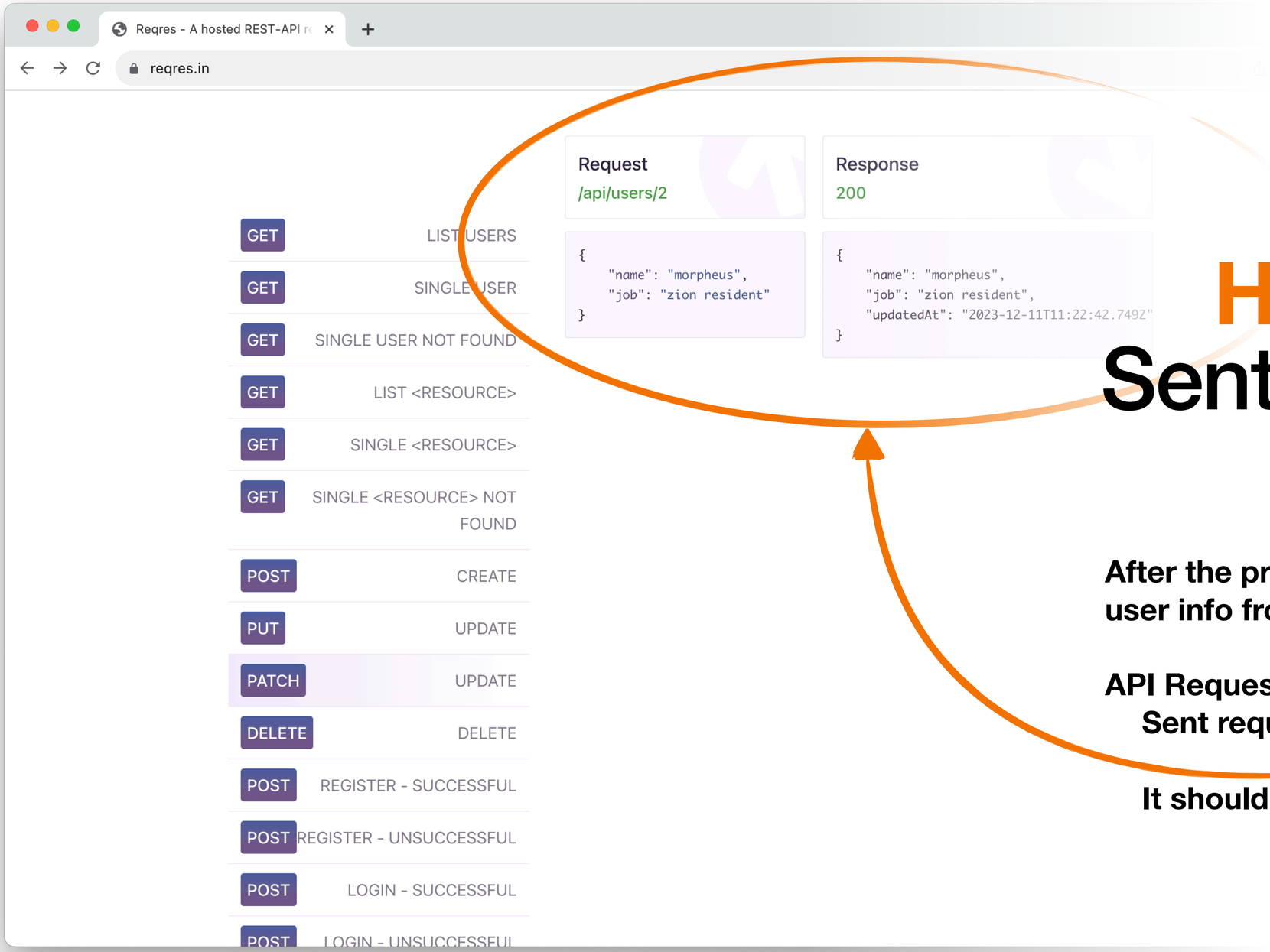
Line8 Add item name "Authorization" with empty string to header (In this case, no need any access token)

Line13 Sent request PUT with created HTTP context to url https://reqres.in/api/users

Line14 Add data payload to request POST

Line19 Log out response status and text to console

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TEST RESULTS  TERMINAL  PORTS  RC

/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)
```



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
5       // Create HTTP header
6       const httpContext = await request.newContext({ - 2ms
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.delete("https://reqres.in/api/users/2"); - 808ms
14      console.log(`HTTP request DELETE response code: ${httpResponse.status()} ${httpResponse.statusText()}`);
15
16  });
17
```

```
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/jojoe/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)
```

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

JSON parse

Test steps:

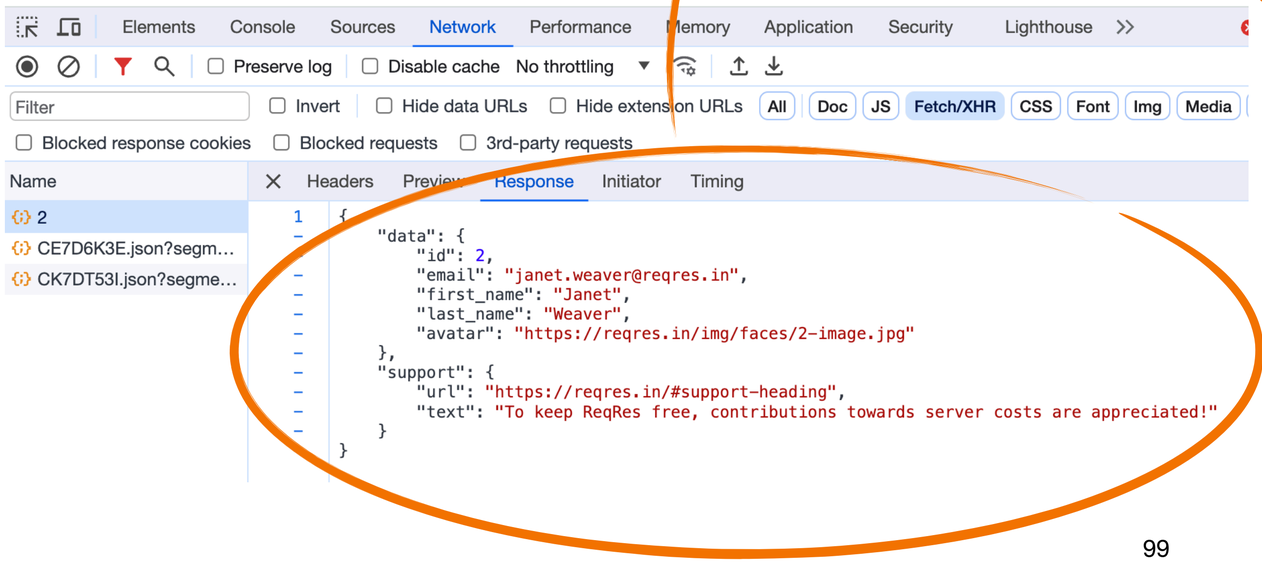
Line13 After receive the response of HTTP GET

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
```



Page object model

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

Ref1: https://playwright.dev/docs/pom

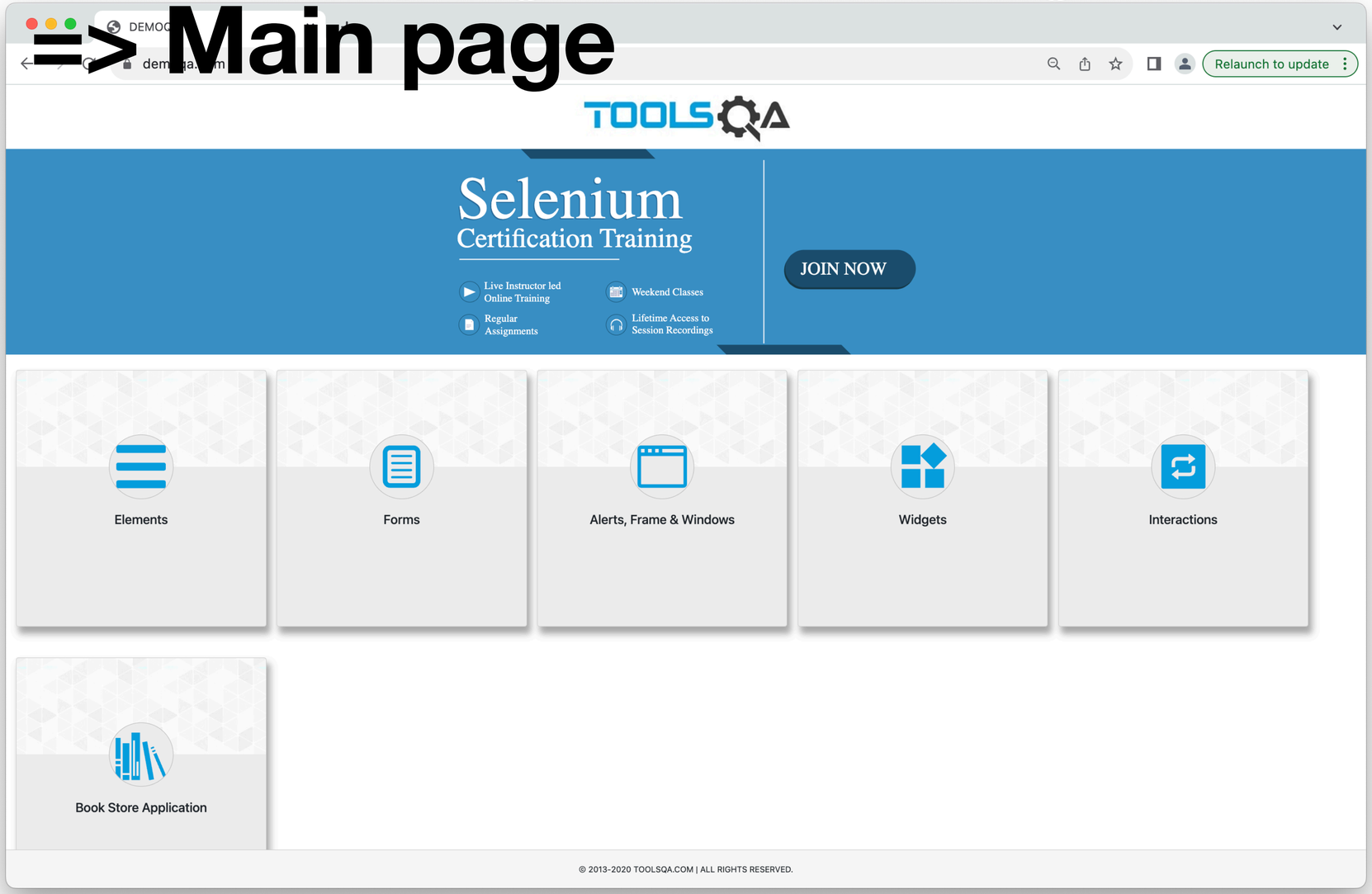
What is page object?

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.

Ref1: OpenAI. (2023). "What is idea of page object model in Playwright."
Ref2: https://playwright.dev/docs/pom

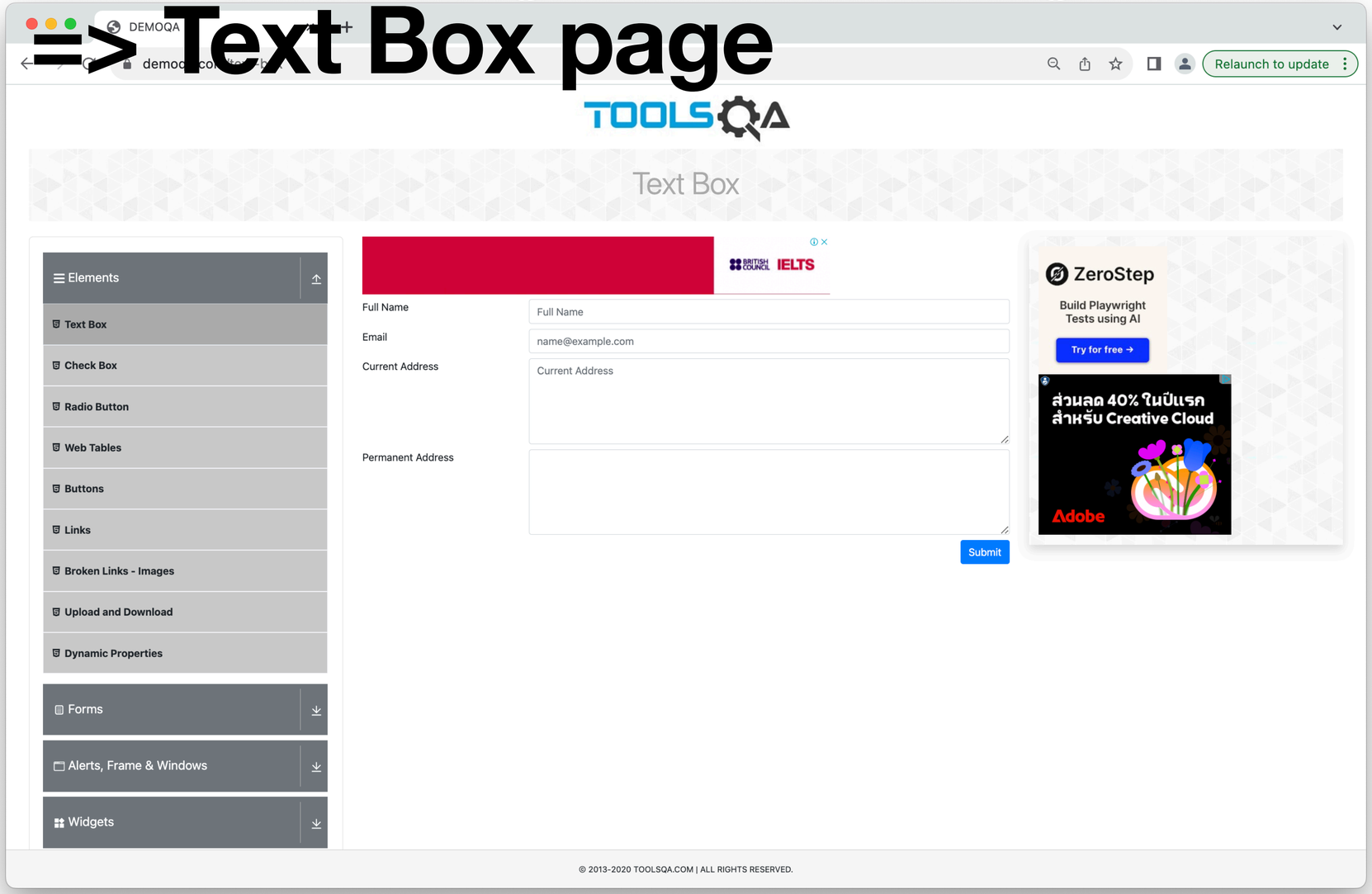
Analyze page containing components => Main page



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



Left panel component

css = div.accordion

Text box page components

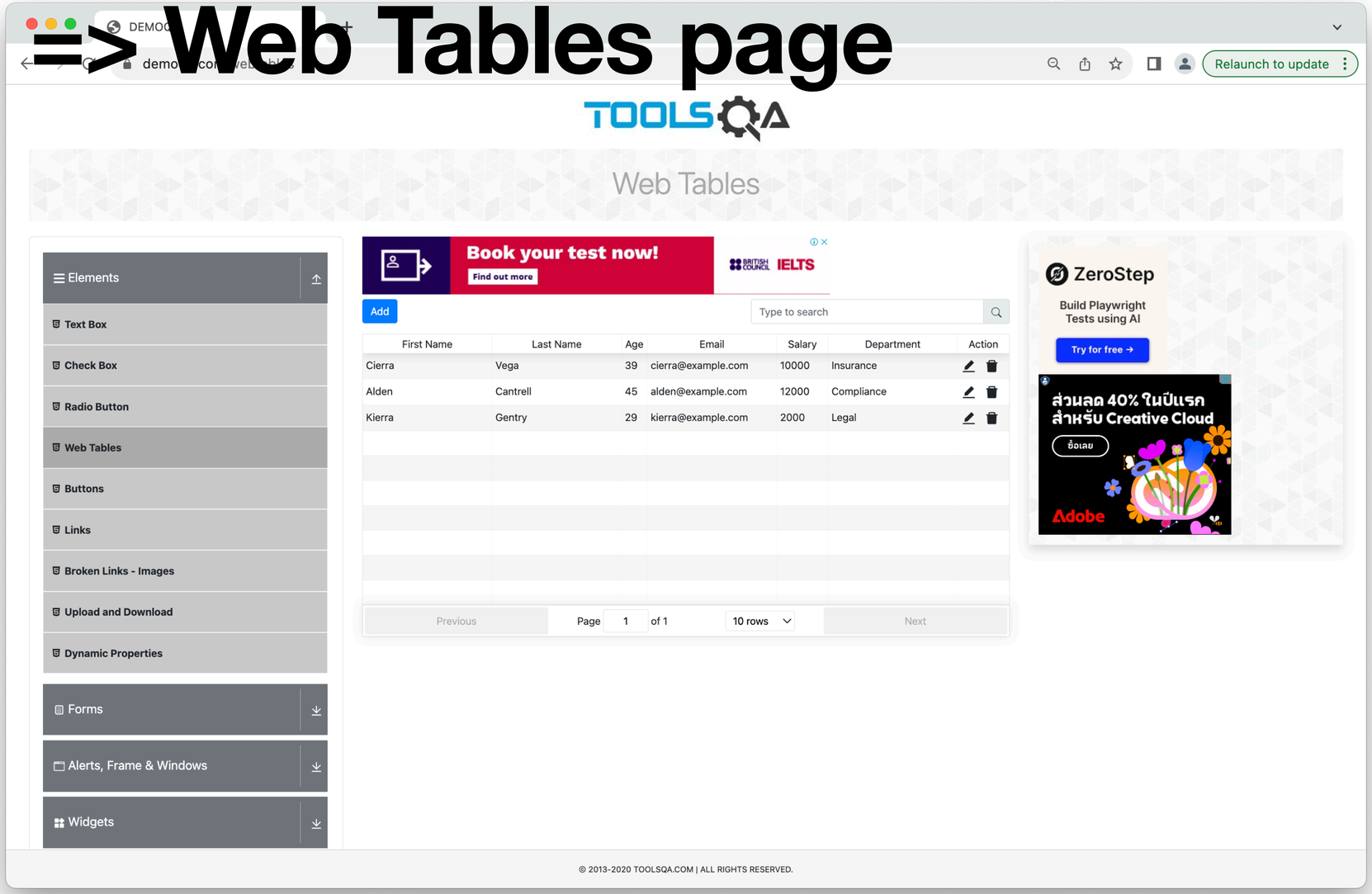
id = userName

id = userEmail

id = currentAddress

:

Analyze page containing components => Web Tables page



Left panel component

css = div.accordion

Web tables page components

id = addNewRecordButton

id = searchBox

css = div.input-group-append

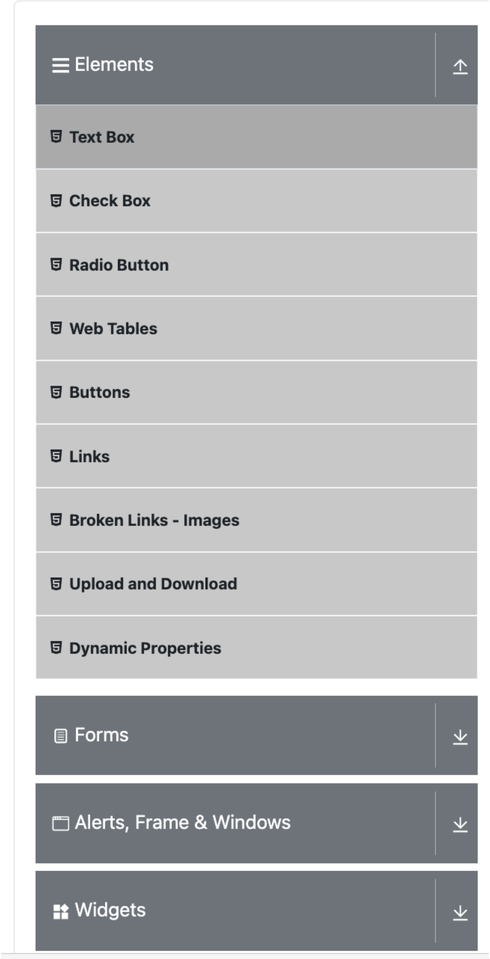
Table component

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'

:

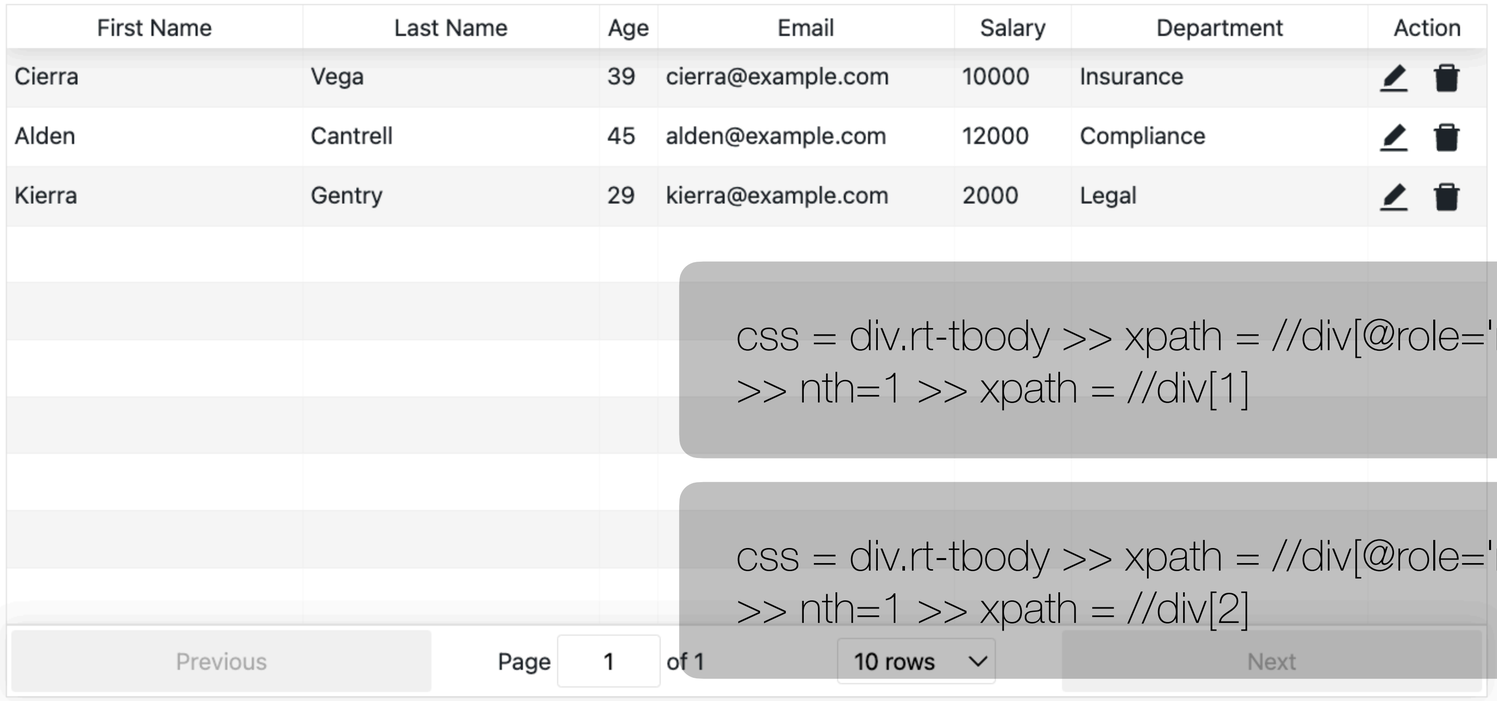
css = div.header-text >> text = 'Forms'

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

Analyze page containing components => Table, columns & rows

Row component

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



Row elements

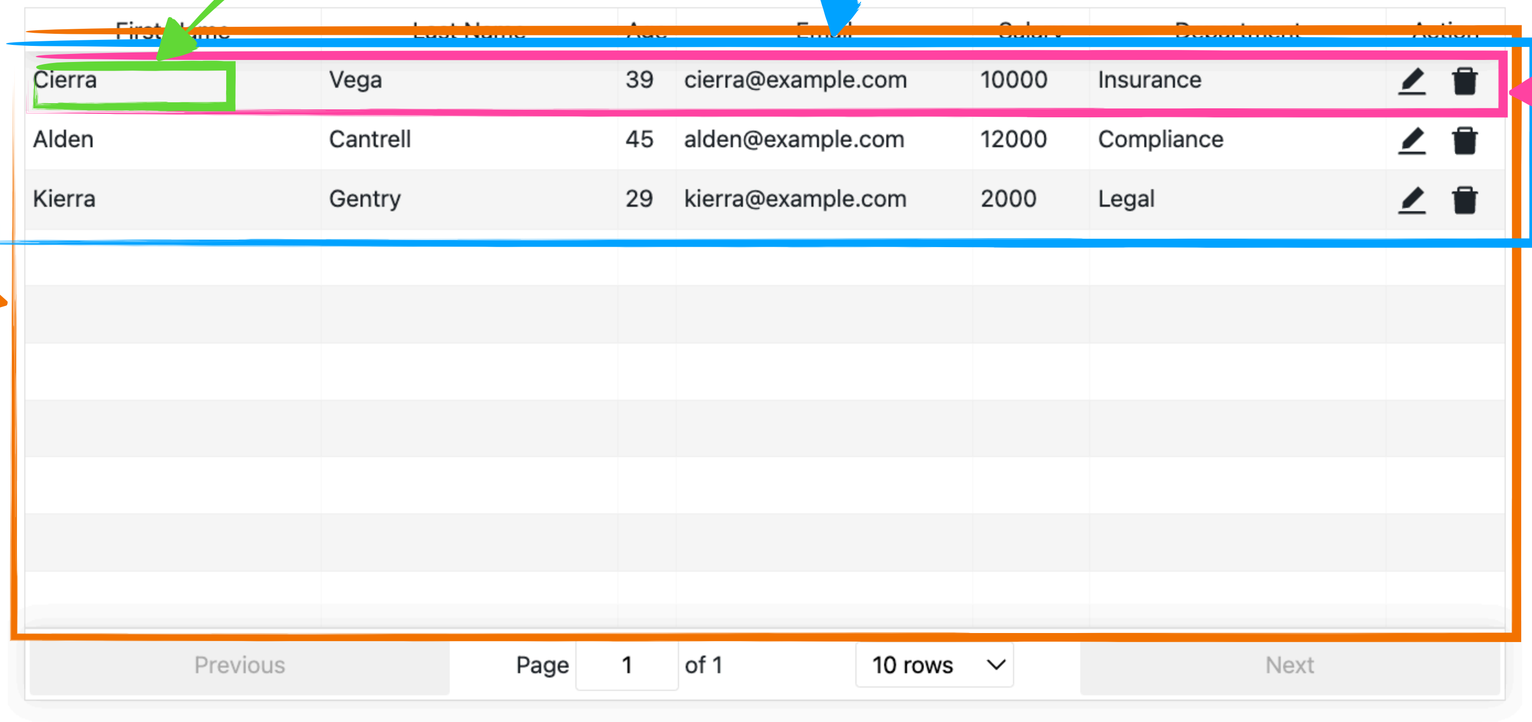
```
css = div.rt-tbody >> xpath = //div[@role='row' and not(contains(@class, '-padRow'))]
>> nth=1 >> xpath = //div[1]
```

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

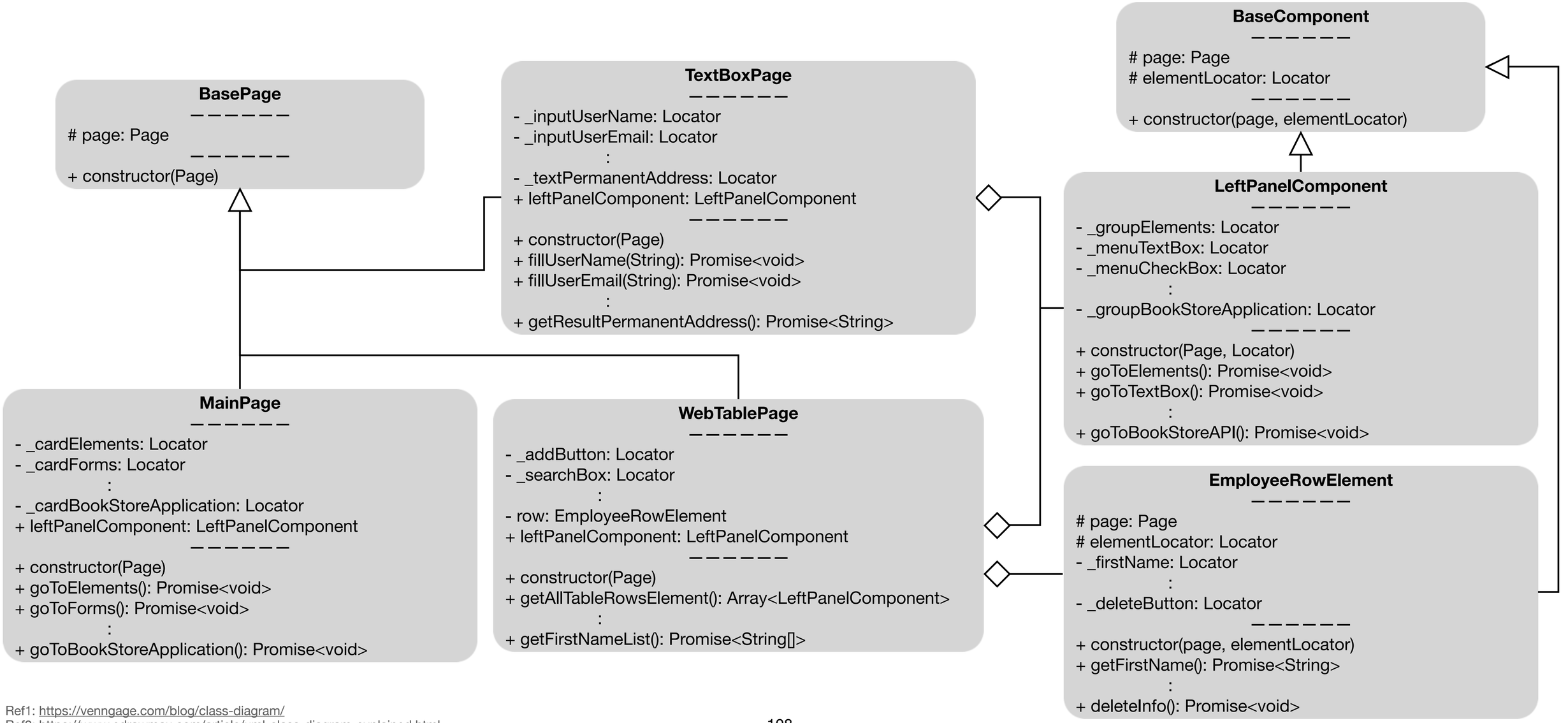
:

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



Ref1: https://venngage.com/blog/class-diagram/
Ref2: https://www.edrawmax.com/article/uml-class-diagram-explained.html

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 > ...

```ts
import { Locator, Page } from "@playwright/test";
import { BasePage } from "./base-page";

export class MainPage extends BasePage {

    private readonly _cardElement: Locator = this.page.locator("xpath=//h5[text()='Elements']");
    private readonly _cardForms: Locator = this.page.locator("xpath=//h5[text()='Forms']");
    private readonly _cardWidgets: Locator = this.page.locator("xpath=//h5[text()='Widgets']");
    private readonly _cardAlertsFrameWindows: Locator = this.page.locator("xpath=//h5[text()='Alerts, Frame & Windows']");
    private readonly _cardInteractions: Locator = this.page.locator("xpath=//h5[text()='Interactions']");
    private readonly _cardBookStoreApplication: Locator = this.page.locator("xpath=//h5[text()='Book Store Application']");

    constructor(page: Page) {
        super(page);
    }

    public async goToElements() {
        await this._cardElement.click();
    }

    public async goToForms() {…
    }

    public async goToWidgets() {…
    }

    public async goToAlertsFrameWindows() {…
    }

    public async goToInteractions() {…
    }

    public async goToBookStoreApplication() {…
    }
}
```

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 > ...

```ts
import { Locator } from "@playwright/test";
import { BasePage } from "./base-page";
import { LeftPanalComponent } from "./left-panal-component";

export class TextBoxPage extends BasePage {

    private readonly _inputUserName: Locator = this.page.locator("id=userName");
    private readonly _inputUserEmail: Locator = this.page.locator("id=userEmail");
    private readonly _inputCurrentAddress: Locator = this.page.locator("id=currentAddress");
    private readonly _inputPermanentAddress: Locator = this.page.locator("id=permanentAddress");
    private readonly _submitButton: Locator = this.page.locator("id=submit");
    private readonly _textResultUserName: Locator = this.page.locator("id=name");
    private readonly _textResultUserEmail: Locator = this.page.locator("id=email");
    private readonly _textResultCurrentAddress: Locator = this.page.locator("id=currentAddress");
    private readonly _textResultPermanentAddress: Locator = this.page.locator("id=permanentAddress");

    public readonly leftPanal = new LeftPanalComponent(this.page, this.page.locator("css=div.left-pannel"));

    public async fillUserName(text: string) {
        await this._inputUserName.fill(text);
    }

    public async fillUserEmail(text: string) {…
    }

    public async fillCurrentAddress(text: string) {…
    }

    public async fillPermanentAddress(text: string) {…
    }

    public async submitForm() {
        await this._submitButton.click();
    }

    public async getUserName(): Promise<String> {
        const isVisible = await this._textResultUserName.isVisible();
        if (isVisible) {
            const text = await this._textResultUserName.innerText();
            return text;
        }
        return "";
    }

    public async getUserEmail(): Promise<String> {…
```

```ts
19    public async fillUserName(text: string) {
20        await this._inputUserName.fill(text);
21    }
```

```ts
35    public async submitForm() {
36        await this._submitButton.click();
37    }
```

```ts
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    }
```

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-panal-component.ts > ...

```ts
import { Locator } from "@playwright/test";
import { BaseComponent } from "./base-component";

export class LeftPanalComponent extends BaseComponent {

    // Group of Elements
    private readonly _elements: Locator = this.elementLocator.locator("css=div.header-text >> text='Elements'");
    private readonly _textBox: Locator = this.elementLocator.locator("css=span.text >> text='Text Box'");
    private readonly _checkBox: Locator = this.elementLocator.locator("css=span.text >> text='Check Box'");
    private readonly _webTable: Locator = this.elementLocator.locator("css=span.text >> text='Web Tables'");
    // ...

    // Group of Widgets
    private readonly _widgets: Locator = this.elementLocator.locator("css=div.header-text >> text='Widgets'");
    private readonly _selectMenu: Locator = this.elementLocator.locator("css=span.text >> text='Select Menu'");
    // ...

    public async goToTextBox() {
        const isVisible = await this._textBox.isVisible();
        if (isVisible) {
            await this._textBox.click();
        }
        else {
            await this._elements.click();
            await this._textBox.click();
        }
    }

    public async goToCheckBox() {…
    }

    public async goToWebTable() {…
    }

    public async goToSelectMenu() {…
    }

}
```

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"

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 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 > ...

```ts
import { Locator } from "@playwright/test";
import { BasePage } from "./base-page";
import { LeftPanalComponent } from "./left-panal-component";
import { EmployeeRowElement } from "./employee-row-element";

export class WebTablesPage extends BasePage {

    private readonly _rows: Locator = this.page.locator(`css=div.rt-tbody >> xpath=//div[@role='row'
        and not(contains(@class, '-padRow'))]`);

    public readonly leftPanal = new LeftPanalComponent(this.page, this.page.locator("css=div.left-pannel"));

    public async getAllRowsElement() {

        const rowCount = await this._rows.count();
        const rowElements = new Array<EmployeeRowElement>();
        for (let i = 0; i < rowCount; i++) {
            rowElements.push(new EmployeeRowElement(this.page, this._rows.nth(i)));
        }

        return rowElements;
    }

    public async getFirstNameList() {
        const rowElements = await this.getAllRowsElement();
        const firstNameList = new Array<String>();
        for await (const row of rowElements) {
            const fName = await row.getFirstName();
            firstNameList.push(fName);
        }

        return firstNameList;
    }

}
```

```ts
    public async getAllRowsElement() {

        const rowCount = await this._rows.count();
        const rowElements = new Array<EmployeeRowElement>();
        for (let i = 0; i < rowCount; i++) {
            rowElements.push(new EmployeeRowElement(this.page, this._rows.nth(i)));
        }

        return rowElements;
    }
```

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 > ...
```

```ts
import { Locator } from "@playwright/test";
import { BasePage } from "./base-page";
import { LeftPanalComponent } from "./left-panal-component";
import { EmployeeRowElement } from "./employee-row-element";

export class WebTablesPage extends BasePage {

    private readonly _rows: Locator = this.page.locator(`css=div.rt-tbody >> xpath=//div[@role='row'
        and not(contains(@class, '-padRow'))]`);

    public readonly leftPanal = new LeftPanalComponent(this.page, this.page.locator("css=div.left-pannel"));

    public async getAllRowsElement() {

        const rowCount = await this._rows.count();
        const rowElements = new Array<EmployeeRowElement>();
        for (let i = 0; i < rowCount; i++) {
            rowElements.push(new EmployeeRowElement(this.page, this._rows.nth(i)));
        }

        return rowElements;
    }

    public async getFirstNameList() {
        const rowElements = await this.getAllRowsElement();
        const firstNameList = new Array<String>();
        for await (const row of rowElements) {
            const fName = await row.getFirstName();
            firstNameList.push(fName);
        }

        return firstNameList;
    }

}
```

```ts
    public async getFirstNameList() {
        const rowElements = await this.getAllRowsElement();
        const firstNameList = new Array<String>();
        for await (const row of rowElements) {
            const fName = await row.getFirstName();
            firstNameList.push(fName);
        }

        return firstNameList;
    }
```

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 > ...

```ts
import { Locator } from "@playwright/test";
import { BaseComponent } from "./base-component";

export class EmployeeRowElement extends BaseComponent {

    private readonly _fName: Locator = this.elementLocator.locator("xpath=//div[@role='gridcell'][1]");
    private readonly _lName: Locator = this.elementLocator.locator("xpath=//div[@role='gridcell'][2]");
    private readonly _age: Locator = this.elementLocator.locator("xpath=//div[@role='gridcell'][3]");
    private readonly _email: Locator = this.elementLocator.locator("xpath=//div[@role='gridcell'][4]");
    private readonly _salary: Locator = this.elementLocator.locator("xpath=//div[@role='gridcell'][5]");
    private readonly _department: Locator = this.elementLocator.locator("xpath=//div[@role='gridcell'][6]");
    private readonly _editButton: Locator = this.elementLocator.locator("xpath=//div[@role='gridcell'][7]");
    private readonly _deleteButton: Locator = this.elementLocator.locator("xpath=//div[@role='gridcell'][8]");

    public async getFirstName() {
        const text = await this._fName.innerText();
        return text;
    }

    public async getLastName() {…
    }

    public async getAge() {…
    }

    public async getEmail() {…
    }

    public async getSalary() {…
    }

    public async getDepartment() {…
    }

    public async edit() {…
    }

    public async delete() {
        await this._deleteButton.click();
    }

}

```

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 > ...

```ts
import { Page } from "@playwright/test";

export class BasePage {

    protected page: Page;

    constructor(page: Page) {
        this.page = page;
    }
}

```

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

```ts
import { Locator, Page } from "@playwright/test";

export class BaseComponent {

    protected page: Page;
    protected elementLocator: Locator;

    constructor(page: Page, elementLocator: Locator) {
        this.page = page;
        this.elementLocator = elementLocator;
    }

}

```

Object creation steps

```ts
// tests > TS test-suite-1.spec.ts > ...
import { expect, test } from '@playwright/test';
import { MainPage } from '../pages/main-page';
import { TextBoxPage } from '../pages/text-box-page';

test("Fill in text box form", async ({ page }) => {

    await page.goto("https://demoqa.com");

    const mainPage = new MainPage(page);
    await mainPage.goToElements();

    const textboxPage = new TextBoxPage(page);
    await textboxPage.leftPanal.goToTextBox();

    await textboxPage.fillUserName("Jojoe");
    await textboxPage.submitForm();

    const resultName = await textboxPage.getUserName();
    expect(resultName).toEqual("Name:Jojoe")

});
```

1. In test scenario create new object of mainPage

```ts
// pages > TS base-page.ts > ...
import { Page } from "@playwright/test";

export class BasePage {

    protected page: Page;

    constructor(page: Page) {
        this.page = page;
    }
}
```

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

```ts
// pages > TS main-page.ts > ...
import { Locator, Page } from "@playwright/test";
import { BasePage } from "./base-page";

export class MainPage extends BasePage {

    private readonly _cardElement: Locator = this.page.locator("xpath=//h5[text()='Element
    private readonly _cardForms: Locator = this.page.locator("xpath=//h5[text()='Forms']")
    private readonly _cardWidgets: Locator = this.page.locator("xpath=//h5[text()='Widgets
    private readonly _cardAlertsFrameWindows: Locator = this.page.locator("xpath=//h5[text
    private readonly _cardInteractions: Locator = this.page.locator("xpath=//h5[text()='In
    private readonly _cardBookStoreApplication: Locator = this.page.locator("xpath=//h5[te

    constructor(page: Page) {
        super(page);
    }

    public async goToElements() {
        await this._cardElement.click();
    }
```

2. When create object constructor will be called first

3. Pass value to super class

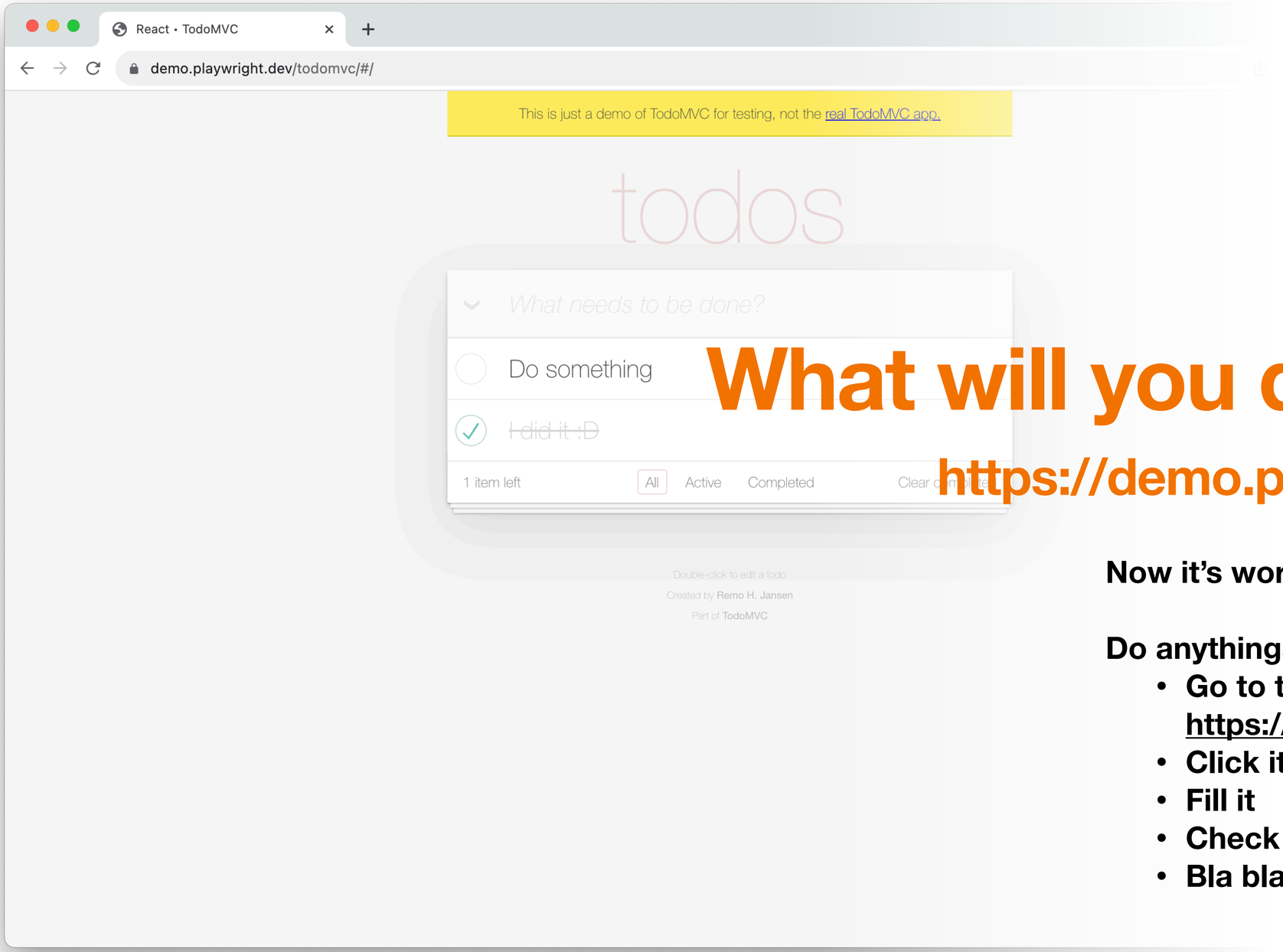
7. this.page is a property of super class BasePage inherited to MainPage class

8. Return create object MainPage

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 anythings 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