General para clasificar

# Azure DevOps

Link para editar los markdowns:

<https://www.markdown.es>

Learning about prompt Engineering:

# 

# 

# Playwright and TypeScript

### Installation

Use the command in the terminal:

**npm init playwright@latest**

Answer the questions for example:

* Do you want to use TypeScript or Javascript? A: TypeScript
* Where to put your end-to-end tests? A: tests (gonna be a folder)
* Add a GitHub Actions workflow? (y/N) A: false
* Install Playwright browsers A: true

You can open the Visual Studio Code by using the command “ **code .**”

Then install the plugins extensions

In these languages we have the operator ‘==’ called **Equality operator,** where you can verify that two variables are the same. For example we can have a **5 == ‘5’**, here will return **true** ‘cause are the same even if the type of data aren’t.

And if you use the operator ‘===’ called **Strict Equality Operator,** where you can verify if two variables are specifically the same, in the value they have and also the type of data. For example 5 === ‘5’ will return false, because the first one is an int type, and the second one is a string type.

The playwright configuration will be created in a **playwright.config.ts** file where we have all the configuration of the framework. Here you can have different parts:

* **test directory** in case you want to run a specific test.
* **fullyParallel:true/false**, where you can set if you want to run the tests parallel, normally this is gonna be true, because it is more effective in time consumption.
* “**retries**: process.env.CI ? 2: 0” or “**retries**: 2, “ also you can set a retry in case some test fails.When the process.env.CI is false, as defect is set to retry 0, but you can change it, and also set retry by defect using the second way shown.
* “**workers: process.env.CI ? 1 : undefined.**” Normally it is set as undefined and the playwright will take care of us.
* “**reporter**: ‘html’” by default we have the html, playwright has their own report in case the tests failed, and you can edit and create a personalized report, you can search more about it in <https://playwright.dev/docs/test-reporter#custom-reporters>
* “**use**: ” you can see what is the base URL in case all your tests have the same base page
* “**trace**” will help to understand our failed tests, as default we can set it as ‘on-first-retry’.
* “**screenshot**:” we can set it as ‘only-on-failure’
* “**OutputDir**” our test results folder.
* Then we have our projects, where you can set different navigators and/or environments.

### Package.json file

Here you can find all the dependencies you have installed on your project, and also you have a section called “scripts” where you can make scripts for your execution. This can help you to create terminal commands more useful based on your needs. For example:

* instead of using **npx playwright test tests/**(you can add the “ / ” to select more specifically the folder in case you have more than one folder starting with the same name, you can create a script called “**test:e2e**” and when use it in the terminal, it gonna do the same.
* If youv have a command “**npx playwright test tests --project=all-browsers-and-tests**”, you can create a script called “**test:e2e:all**” to resume it.
* You can create scripts that run a command with a specific tag, for example: “**npx playwright test tests-examples/ --grep @tagName --project=chromium**”
* If you want to run all the tests that does not have a specific tag(for example all the non-smoke tests) you can use the --grep-invert similar to the last one: “**npx playwright test tests-examples/ --grep-invert @tagNameThatYouDoNotWantToRun**”
* For the commands, you can also add parts like “**--ui**” or “--**debug**” to run the tests with more help and the debugging tools. You can see more of this in the section <https://playwright.dev/docs/running-tests>

Just a reminder, the section “**scripts**” in the package.json file is optional, this only helps you to use easier commands. And to run that script you only need to write “npx run” and then the script created in the package.json file.

### Test files

* First of all appears the imports in this format: “**import { test, expect } from ‘@playwright/test’;** ” the **test** is the test object for the test case, and the **expect** is the expect used for assertions(similar to Assert from TestNG)
* Then we have the test name, and the inherit page as well. Conceptually, in tests we have actions, and then the assertions.
* Always set an “**await**” at the beginning of each step, to ensure that the test is waiting until the element is ready to interact or is already displayed before completing an action and avoid wait errors.

### Used commands in playwright

##### npx playwright test

Run all the tests you have

##### npx playwright show-report

Open an HTML to show the results

### AppliTool for Playwright

Is an application that can do visual validations. To install it you can use the command “**npx i -D @applitools/eyes-playwright**”, and then you need to import the classes “**import { BatchInfo, Configuration, EyesRunner, VisualGridRunner, BrowserType, DeviceName, ScreenOrientation, Eyes, Target} from ‘@applitools/eyes-playwright’;**”

The other important step is set variables and constructors:

export const USE\_ULTRAFAST\_GRID: boolean = false;

export let Batch: BatchInfo;

#### Set a login authentication

<https://playwright.dev/docs/test-global-setup-teardown#configure-globalsetup-and-globalteardown>

##### Important points

* Is recommended to use project dependencies, because you can see the global configuration in the HTML report, the trace viewer can catch the configuration and it is possible to use fixtures.

##### Using dependencies

1. In playwright.config.ts we create a new project called ‘setup db’:



1. then we add the property ‘dependencies’ to our current project:



Now chromium with db depends on the ‘setup db’ project.

1. Initialize our db and create a setup test:



and

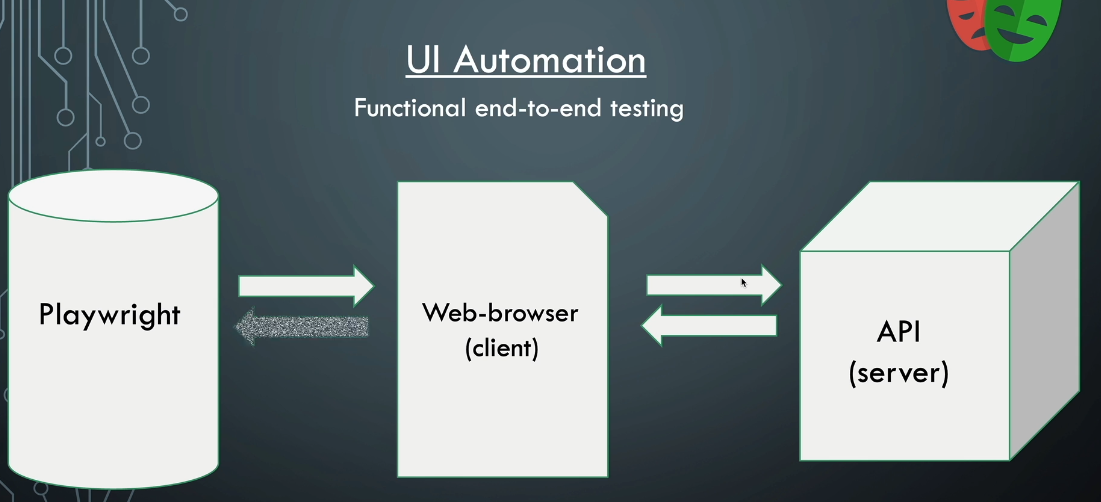


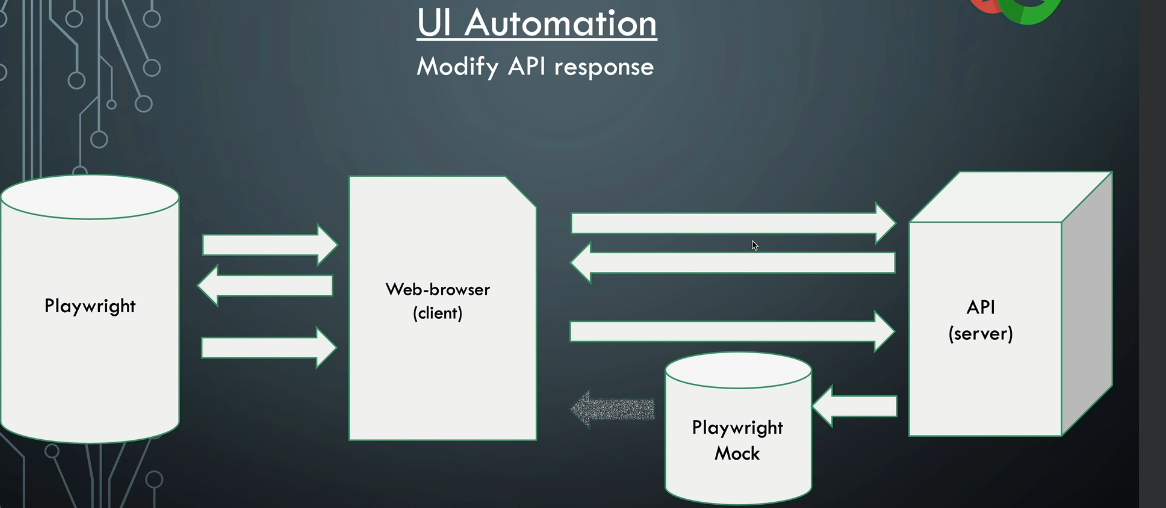
## APIs and Playwright?

Playwright can mock, intercept and interact with APIs directly.

### Types of interactions with playwright

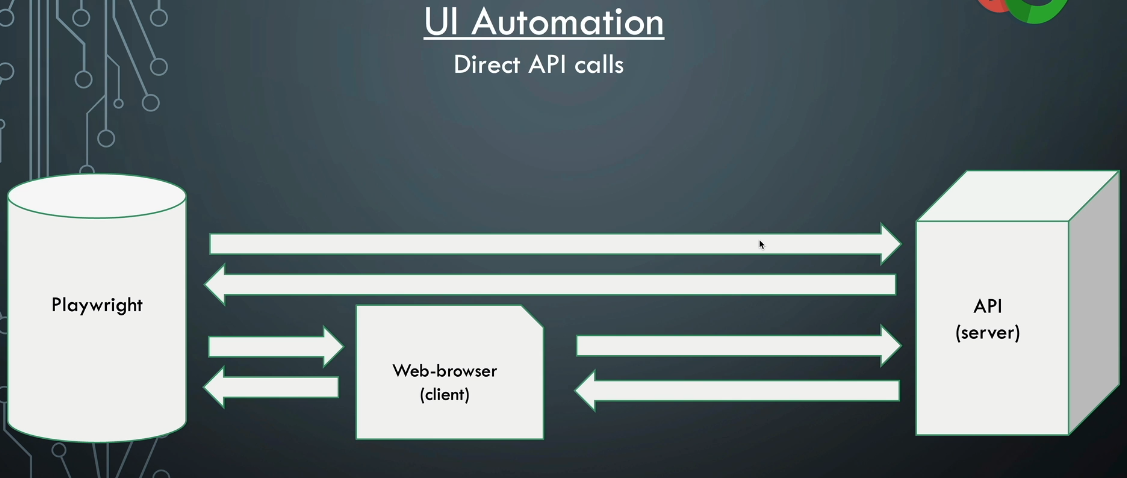
Playwright - Web-browser(client) - (when you need to modify the response) Playwright Mock - API(Server)





Direct API calls(useful when you don’t need to use an authorization):

Playwright - API(Server),



First you need to intercept the web using the page.route() before loading that webpage, normally this occurs in beforeEach test.

Also you need to create a .json file with the test-data, in order to avoid to hardcoding in the test files

## Good practices and good to have in mind in Typescript

* Remember to set retries in case something happens with the environment, we wouldn't want to have a false negative. With retries, we prevent this and other situations to happen and we have less flaky tests. However, it's important to take a look at recurring situations of retries and see if there is something that could be done to improve it.
* Normally the report will appear on-failure, so it could be important to change that to always.
* The command-line interface(cli) will override the playwright.config.ts file
* If you want to run all the tests while not having a certain tag, you only need to add the cli “**--grep-invert @tagName**”.
* If you do not use the “**{exact: ‘word’}**” Playwright will get all the elements that contain the text "word" anywhere and in case there is more than 1 will throw an error and fail the test.
* We have something similar to the “Rule:” in cucumber and java but called test.describe(‘Test name or description’, (parameters if needed) => { here you will write the tests })

Python

Docker in Playwright

To run test from docker:

Command to execute test inside COMMAND PROMPT - docker run -it --rm --name playwright-container -v "%cd%":/app mcr.microsoft.com/playwright:v1.380-jammy

Command to execute test inside POWER SHELL - docker run -it --rm --name playwright-container -v "$(pwd):/app" mcr.microsoft.com/playwright:v1.38.0-jammy

Once it opens the Bash inside the Docker run

cd /app --> To switch to app folder where all our code is copied

npm install --> To install all the dependencies mentioned in the package.json

npx playwright test--> To execute the test

Here with the current Playwright release I am using image name as playwright:v1.38.0-jammy. When you execute make sure to insert the latest playwright docker image name

===================================================================================

For MAC--> Execute all the commands mentioned below in the terminal

Command to execute test - docker run -it --rm --name playwright-container -v $PWD:/app mcr.microsoft.com/playwright:v1.38.0-jammy

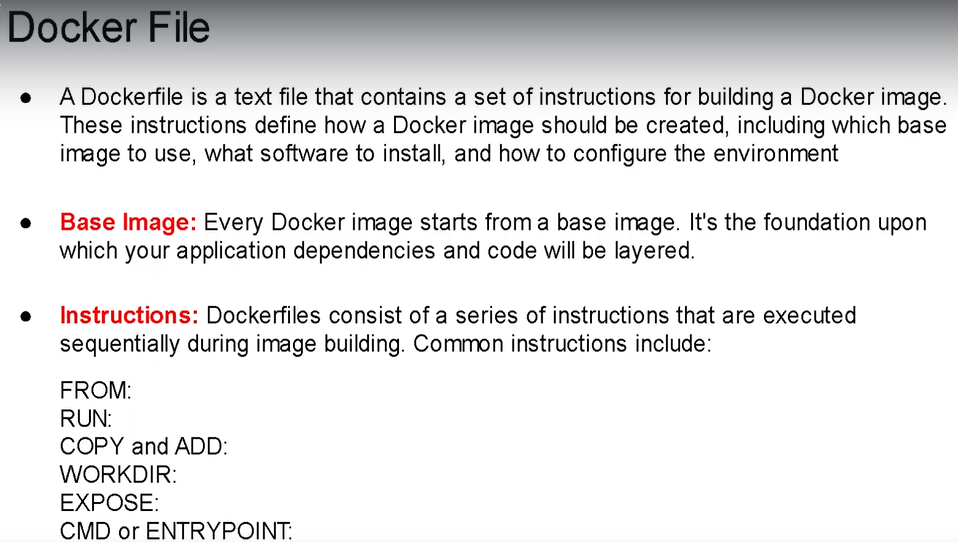
Once it opens the Bash inside the Docker run

cd /app --> To switch to app folder where all our code is copied

npm install --> To install all the dependencies mentioned in the package.json

npx playwright test--> To execute the test

Here with the current Playwright release I am using image name as playwright:v1.38.0-jammy. When you execute make sure to insert the latest playwright docker image name



# Use the Playwright image as the base image

FROM mcr.microsoft.com/playwright:v1.38.0-jammy

# Set the working directory inside the container

WORKDIR /app

# Copy your application code into the container

COPY . /app

# Install Java and other dependencies

RUN apt-get update && \\

apt-get install -y openjdk-11-jre-headless && \\

npm install

# Set environment variables or additional configuration if needed

ENV JAVA\_HOME=/usr/lib/jvm/java-11-openjdk-amd64

# Command to run your Playwright tests

CMD ["npm", "test"]

Docker Compose File

version: '3'

services:

playwright-test:

image: playwright

build: .

container\_name: playwright\_docker

command: npm run test

To run the docker compose file the command is

docker-compose -f docker-compose.yml up

Docker Compose File for WINDOWS

version: '3'

services:

playwright-test:

image: playwright

build: .

container\_name: playwright\_docker

volumes:

- ./:/app

command: npm run test

Docker Compose File for MAC

version: '3'

services:

playwright-test:

image: playwright

build: .

container\_name: playwright\_docker

volumes:

- $PWD:/app

command: npm run test

To run the docker compose file the command is

docker-compose -f docker-compose.yml up

AXE

Important points studying its GitHub Axe core:

* Axe is an accessibility testing engine for websites and other HTML-based user interfaces.
* You can find on average 57% of WCAG issues automatically.
* The library has all the rules and best practice in a .md file.
* Axe can return elements as incomplete due to detecting where cannot be certain and manual review is needed.
* The core has an .MD with best practices rules, the rules that are disabled by default, and WCAG 2.0 Level A and AA rules with its description, the impact, used tags per each rule, the type of issue, and the ACT rules. Even has the deprecated rules.
* The Readme.md recommend a Visual Studio Code extension that you can use to check on react native, angular, Vue, HTML and Markdown, and even can add it to your code and configure depending on your needs(Axe accessibility Linter) and you can integrate it into platforms asGitHub Actions, SonarQube, Jenkins, and Git pre-commit hook.
* Also has the possibility to build Axe with other languages.
* They have updates of security and new/better features each 3-5 months, so recommend scheduling a date to update Axe-core.

# Installation

1. Use this command “**npm install axe-core --save-dev**”
2. Include the Js

QA Studio

Resources:

Shadow locators (Shadow DOM):

<https://www.youtube.com/watch?v=-uMLqBO2x7c>

Java, Selenium WebDriver, Cucumber, and Gradle:

<https://globant.udemy.com/course/selenium-con-java-y-cucumber-el-curso-definitivo/learn/lecture/24420208#overview>

Playwright: Web Automation Testing From Zero to Hero:

<https://globant.udemy.com/course/playwright-from-zero-to-hero/learn/lecture/39698946?start=0#overview>

Playwright web automation with Test Automation University:

<https://testautomationu.applitools.com/playwright-advanced/>

Axe core:

<https://github.com/dequelabs/axe-core/blob/develop/README.md>

<https://github.com/dequelabs/axe-core/blob/develop/doc/rule-descriptions.md>

Axe Accessibility Linter:

<https://marketplace.visualstudio.com/items?itemName=deque-systems.vscode-axe-linter>

storageState in Playwright with Typescript:

<https://www.youtube.com/watch?v=QJL6uV7z-8I&ab_channel=CommitQuality>

Fixture for Playwright:

<https://www.youtube.com/watch?v=2O7dyz6XO2s&ab_channel=Checkly>

global setup and teardown:

<https://playwright.dev/docs/test-global-setup-teardown#configure-globalsetup-and-globalteardown>

Github repositories:  
<https://github.com/dilpreetj/playwright-course/blob/base-project/tests/contact.spec.ts>

In case an error occurs while installing Docker:

<https://learn.microsoft.com/en-us/windows/wsl/install-manual#step-4---download-the-linux-kernel-update-package>