***Playwright Automation***

Free and open source framework used for web automation testing. Created by Microsoft.

We can test web browser apps , mobile web apps, api

Languages used to implement playwright is JS, TSX,Java,Python, .NET

All modern engines like chromium, webkit and firefox are supported.(headless mode and headed mode) head less mode no physical looking headed mode means we have a physical look.

It is also supported in Windows, Linux OS locally or on CI.

**Features of Playwright:**

* Free and Open source
* Multi Browser and Multi Language and Multi Os
* Easy set up and configuration
* Functional , API and accessibility testing
* Built in reporters and custom reporters
* CI and CD and docker support is also present
* Recording , Debugging, and Explore Selectors
* Parallel testing
* Auto wait.
* Built in assertions | Less Flaky tests(flaky means not sure on passing and failing)
* Test retry and logs and screenshots and videos
* Multitab and multi window is supported
* It can also handles iframes and shadow DOM
* It can emulate mobile devices and switch geo locations
* Test Paremeterization
* Fast and Robust

By default all the tests in the playwright are executed in the headless mode and if we want to execute them in the headed mode we have to specify that using

***npx playwright test --headed*** – all the tests will run in headed mode.

***npm init playwright@latest*** – to initialize the npm project with playwright installed.

***npm install playwright*** – to install the playwright package to the project.

***npx playwright test*** – Runs the end to end tests.

***npx playwright test --workers 3*** – runs the test with 3 workers in parallel.

***npx playwright test --ui*** – Starts the interactive UI mode.

***npx playwright test --project=chromium*** – Runs the tests only the Desktop Chrome.

***npx playwright test example*** – Runs the tests in a specific file in which name contains example.

***npx playwright test [filepath]*** – runs the tests in the file given in the specified path.

***npx playwright test --debug*** -- Runs the tests in debug mode.

***npx playwright codegen*** – Auto generate test cases with Codegen.

***npx playwright show-report*** – to show the latest report.

***npx playwright --help*** – to get the help or a list of all commands that can be used with playwright.

***npm playwright -v –*** to check the playwright version.

***npx playwright test -g “title”*** – Runs the test with the given title.

Playwright test provides a test function to declare tests and expect function to write assertions so we have to import them to our file where we write tests.

***test(‘Test Title’,()=>{});*** -- syntax to create a test.

The keyword async before a function makes the function to return a promise.

Playwright comes with a tool called Codegen also called Test Generator. Can be used to record a test and generate test scripts.

To open the codegen we have to type npx playwright codegen url with that url it will open the window.

***npx playwright codegen*** – to open the codegen that will do as follows.

***npx playwright codegen --help*** – used to list all the options that can be used with codegen.

It opens 2 windows

1)A browser window to interact with the website.

2)Playwright Inspector window to record the test. Here we can see the playwright scripts.

***npx playwright codegen --target javascript -o [path of file]*** – sends the generated code to the file specified and --target specifies the language in which the code has to be generated.

Only thing we have to in the above file is we have to make sure to add the test block to the code generated because the code generated does not contain test block.

***npx playwright codegen --viewport-size=800,600*** – opens the playwright codegen in specific size.

***npx playwright codegen --device = “iPhone 11”*** – sets the emulated device.

***npx playwright codegen --color-scheme=dark*** – opens the browser in the dark mode.

***Trace Viewer in Playwright:***

GUI tool that helps viewing the executed test along with snapshots, timeline and other details(traces).

To on the trace viewer go to the config file in the playwright folder and in use:{} find the trace:”on-first-retry” by default and now you can have multiple values there. They are:

***‘on-first-retry’*** : Record trace when retrying a test for the first time.

***‘off’*** : Do not record a trace.

***‘on’*** : Record the trace for the each test. (Not recomonded as it is performance heavy).

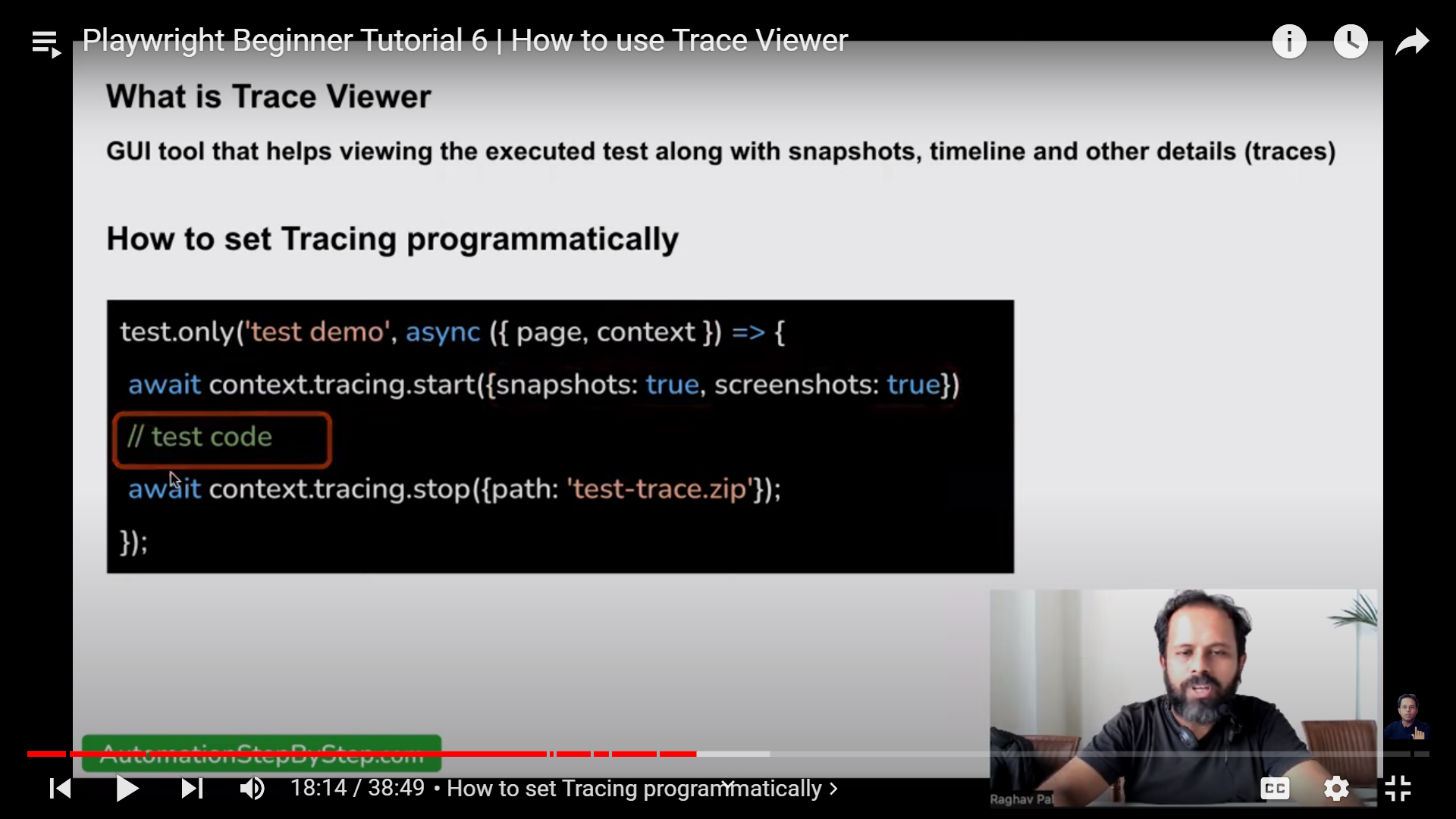
***‘retain on failure’*** : Record a trace for each test, but remove it from successful test runs.

We can view the trace of the test case by selecting the file in the *trace.playwright.dev*

To set the trace from the command line ***npx playwright test --trace on***

***npx playwright show-trace [path\_of\_trace.zip]*** – used to show the trace for the test executed.

We can also set the tracing in the application programmatically in the code by adding





We use beforeAll hooks and afterAll hooks in the application code we use beforeAll code is executed before all the tests run in our code and afterAll gets executed when all the tests completes its execution.

page.pause() will open the playwright inspector where we can use that for code-gen or to know the selectors and locators name and all.

***Note:***

*In latest versions of playwright npx playwright test ./tests/login\_demo.spec.js --headed will work and npx playwright test .\tests\login\_demo.spec.js --headed wont work we will pop out an error.*

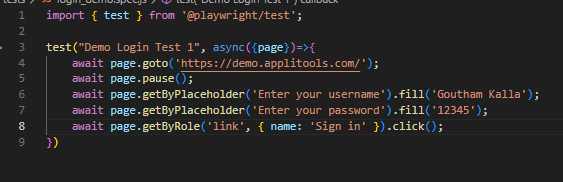
await page.locator(locator name) – older versions of playwright.

Example Code:



In newer versions the code is replaced :

Example code:



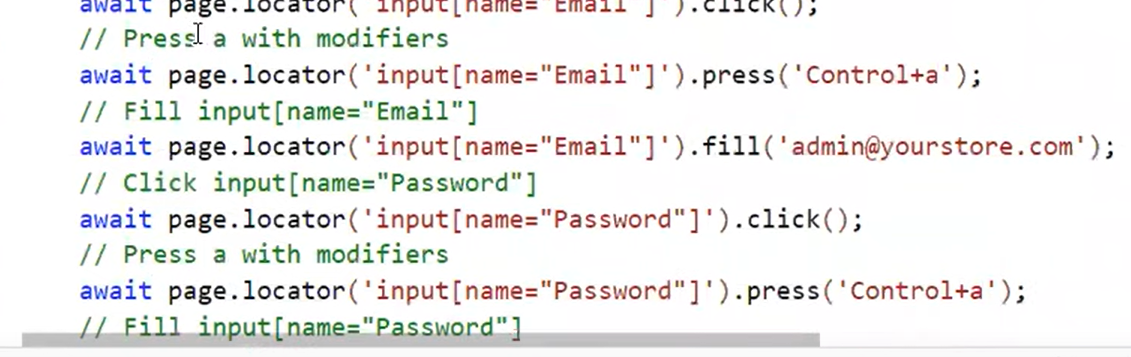
In playwright the commands like fill(), click() already has some in built assertions they check whether the element is present or not before the fill() and or else you can that do manually checking whether that is present or not.



In playwright when this document is done default timeout is 5 sec.

By default the timeout variable from playwright.config.js is taken by the each step but if you want to give some time for the specific step then you can give the ***{timeout: 5000}*** like in above example in order to give our custom timeout for that specific step. In this way each step also can be configured in playwright.

If there is ***test.only***() in a file then only that test is executed in that file.



We can clear the fields using .clear() of we can also press ctrl + a we can also record or simulate that actions also.

***Assertions in Playwright:***

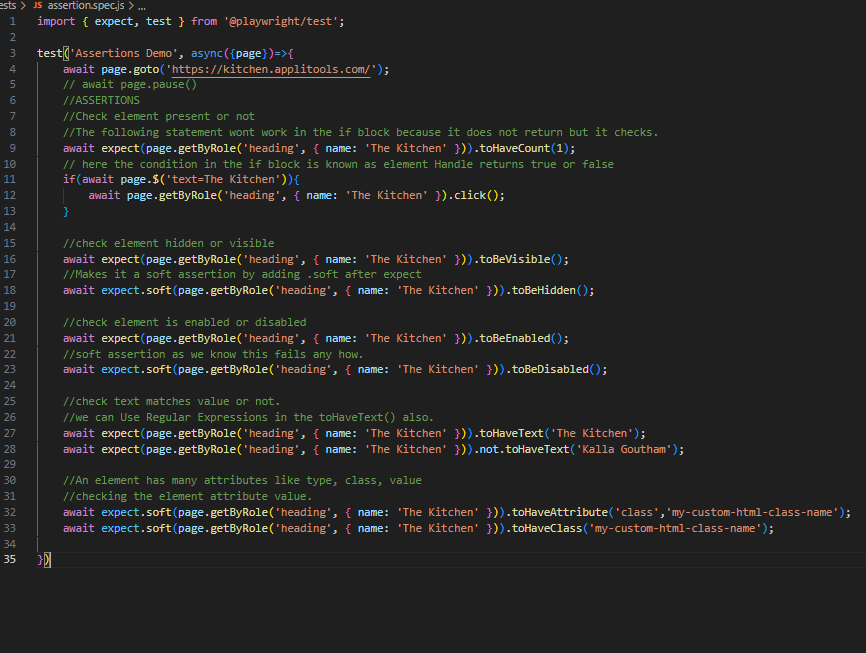
Checks or verification is also known as Assertion.

They will check actual = expected or not.

Assertions like present or not , visible or hidden or enabled or disabled or text match or does not match and element attribute , URL, titles and page matches and screen shots and soft assertions and visual matches in the UI also.

In general execution stops whenever the assertion fails in the given test.

To make execution continuing even after the assertion fails you have to make that assertion a soft assertion. In order to make an assertion soft you have to add ***.soft()*** after expect block in order to make that soft assertion.

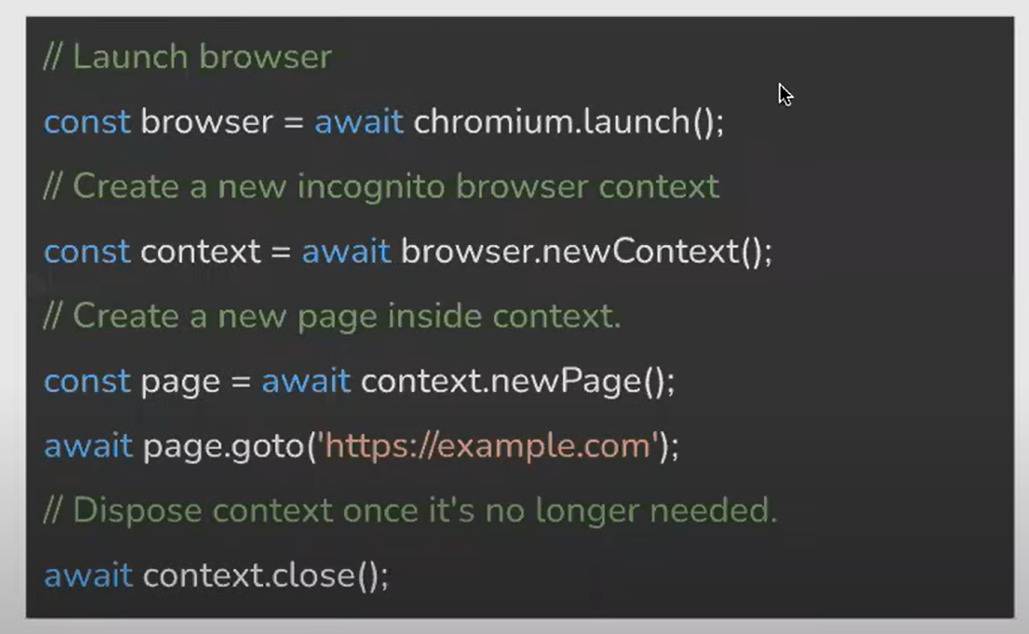


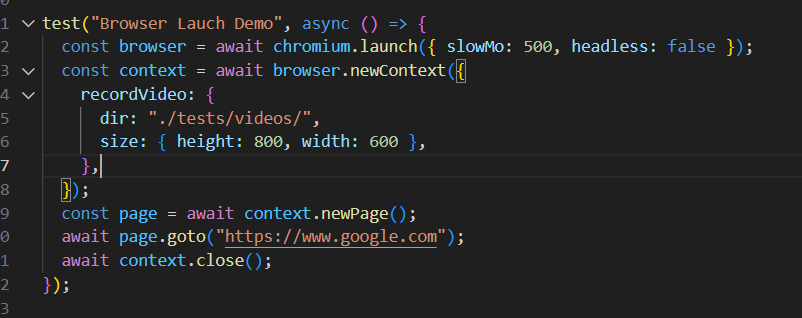
In playwright if there is no snapshot already present then it takes the new one when we run that for the first time and we will get a warning saying that a screenshot is taken but the test runs as expected and no interruption in the test happens.



By modifying the use in the playwright.config.js we can record and slow down the operations in the playwright as shown above.

In playwright we can also create our own context and browser pages and use that pages also for our testing.





Like mentioned in the above piece of code we can also make the configuration for a particular test and we can make that particular test to be running with mentioned configurations.

***Playwright Hooks and Groups:***

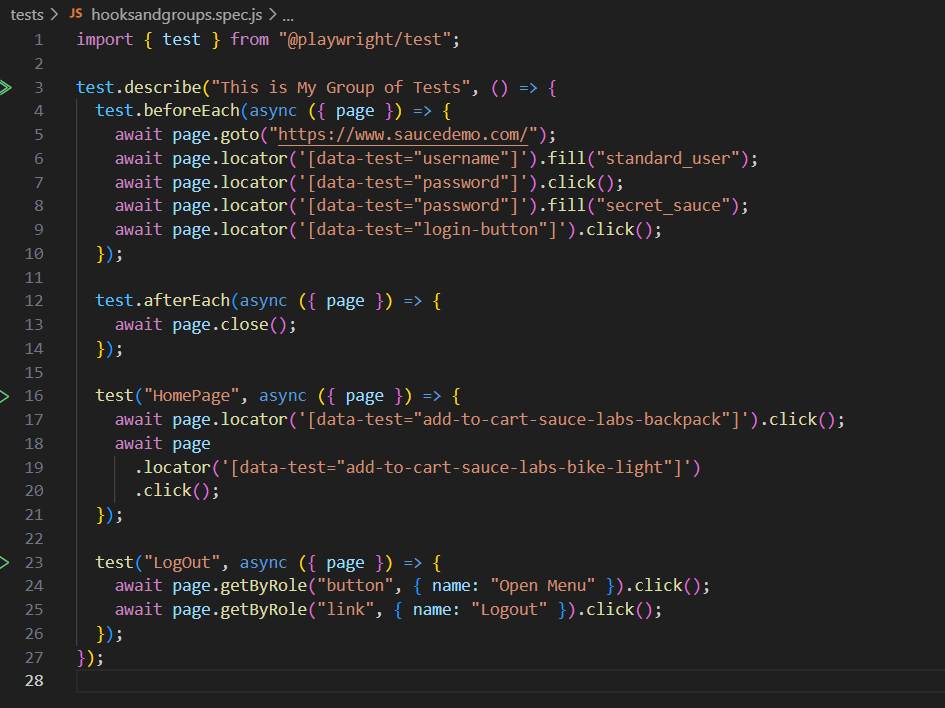




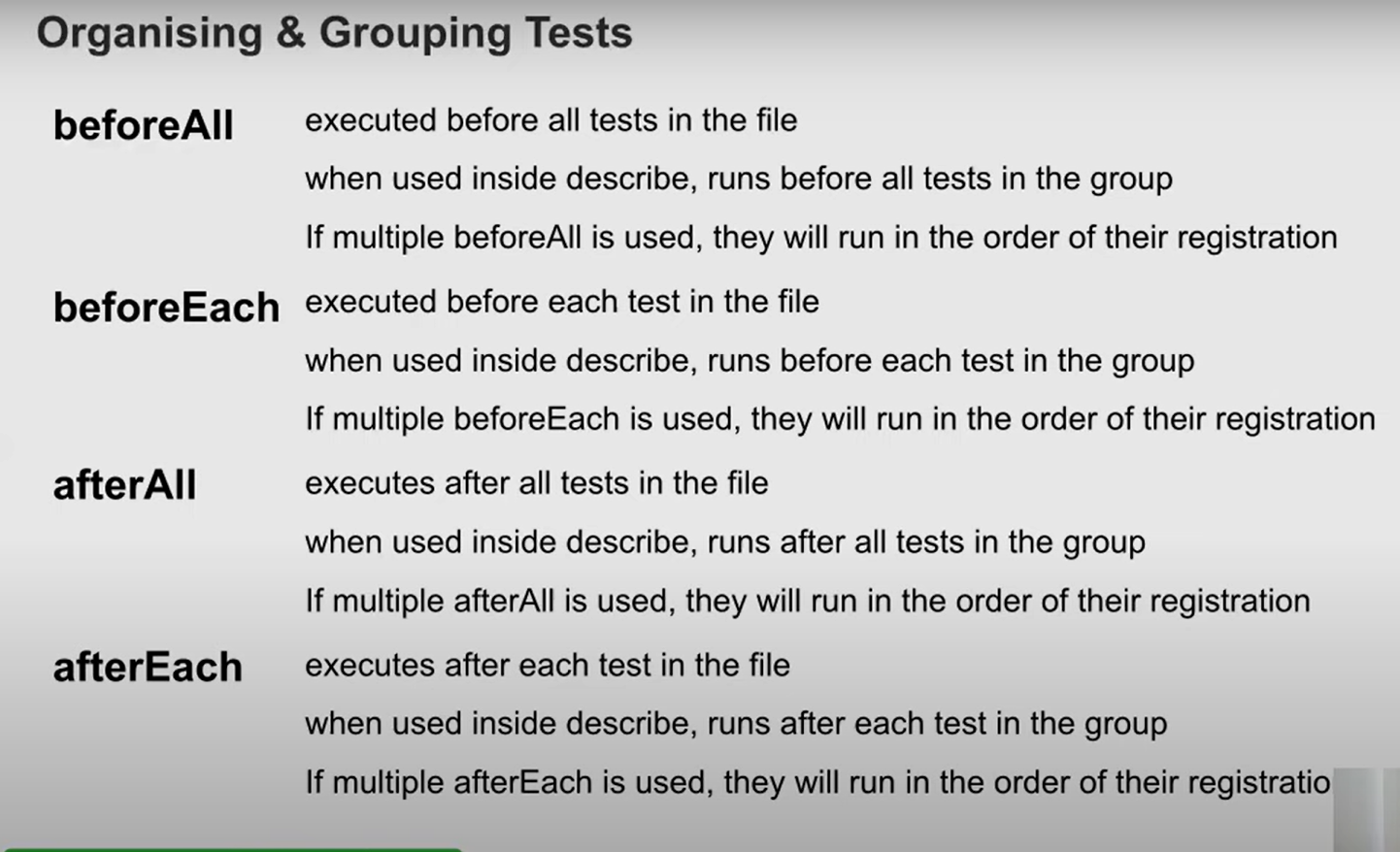
Now in the above code we have three tests one is login, Homepage and logout tests in all these three tests initially we have to login into the applications which involves the same steps so what we can do here is using the hooks afterEach, beforeEach, beforeAll and afterAll hooks in order to repeat the steps which are specified which decreases the time taken to run the tests.

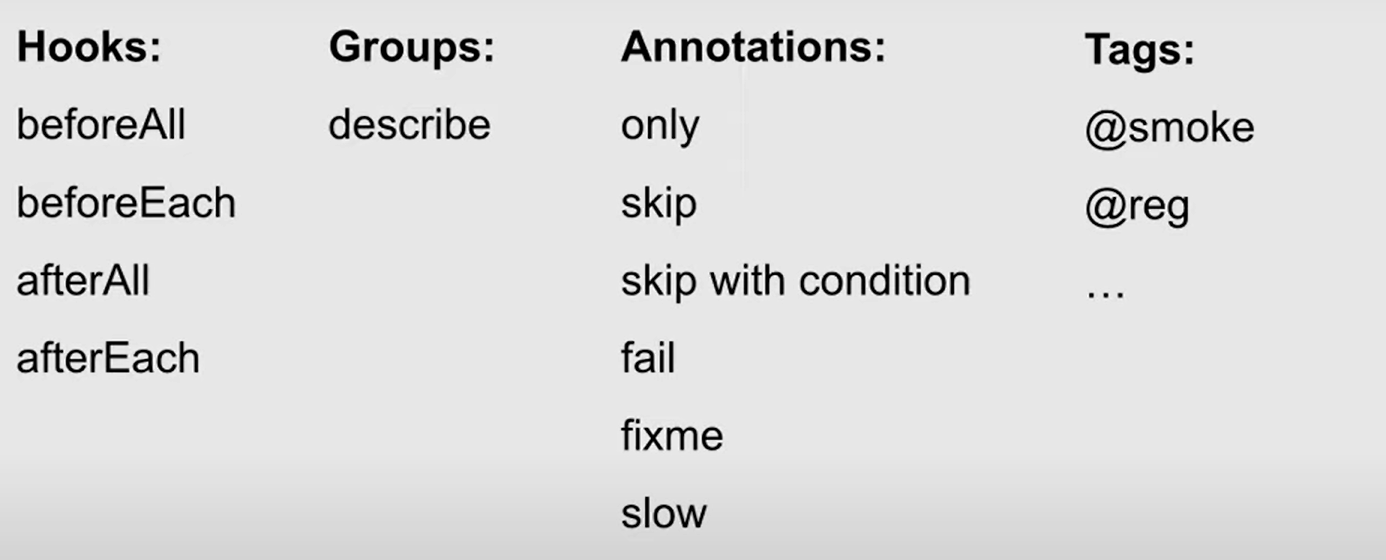


Now we have two pictures one is using hooks and another is without using hooks we can see both of the codes will perform similar tests and gives similar results but we can see there is a lot of redundant code got eliminated and we can also see our tests run much more faster than earlier.These are the uses of hooks.

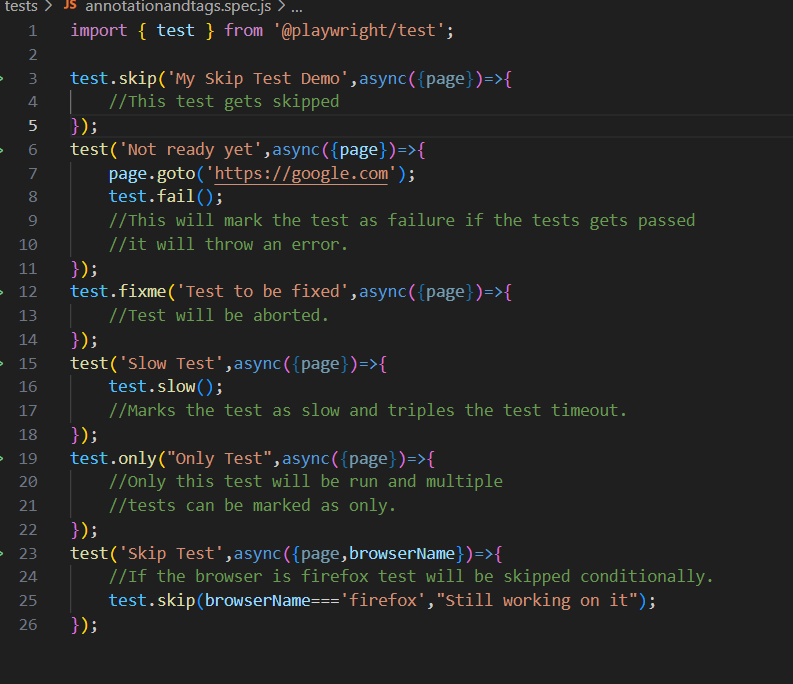


In this way we can group the given tests and we can have some custom beforeAll and beforeEach hooks for those group of tests which will improve the efficiency and response type of testing.

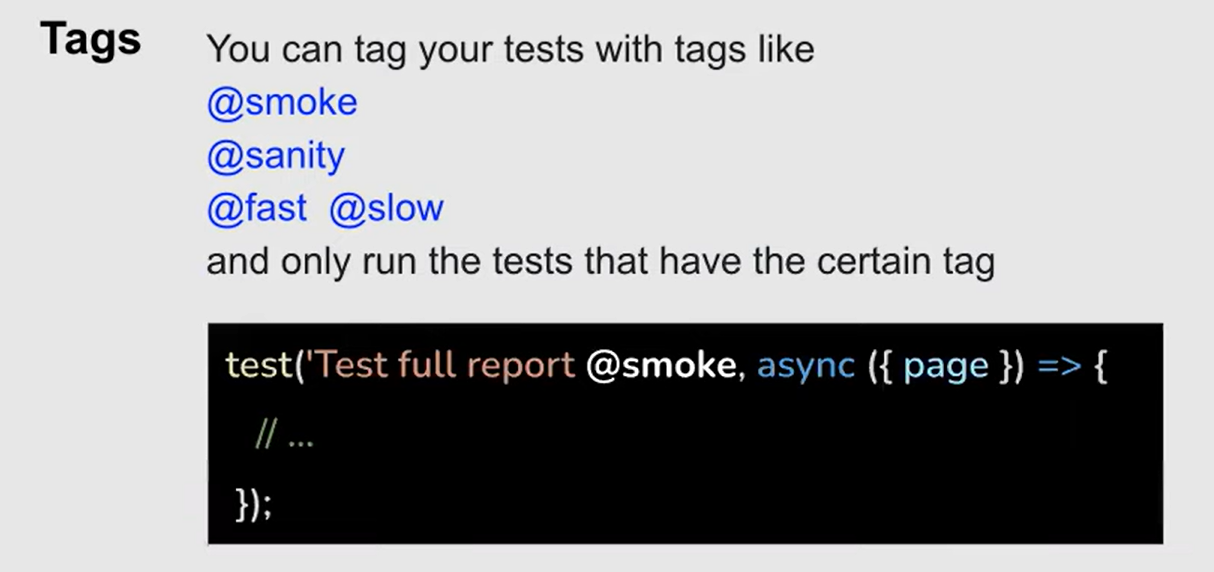


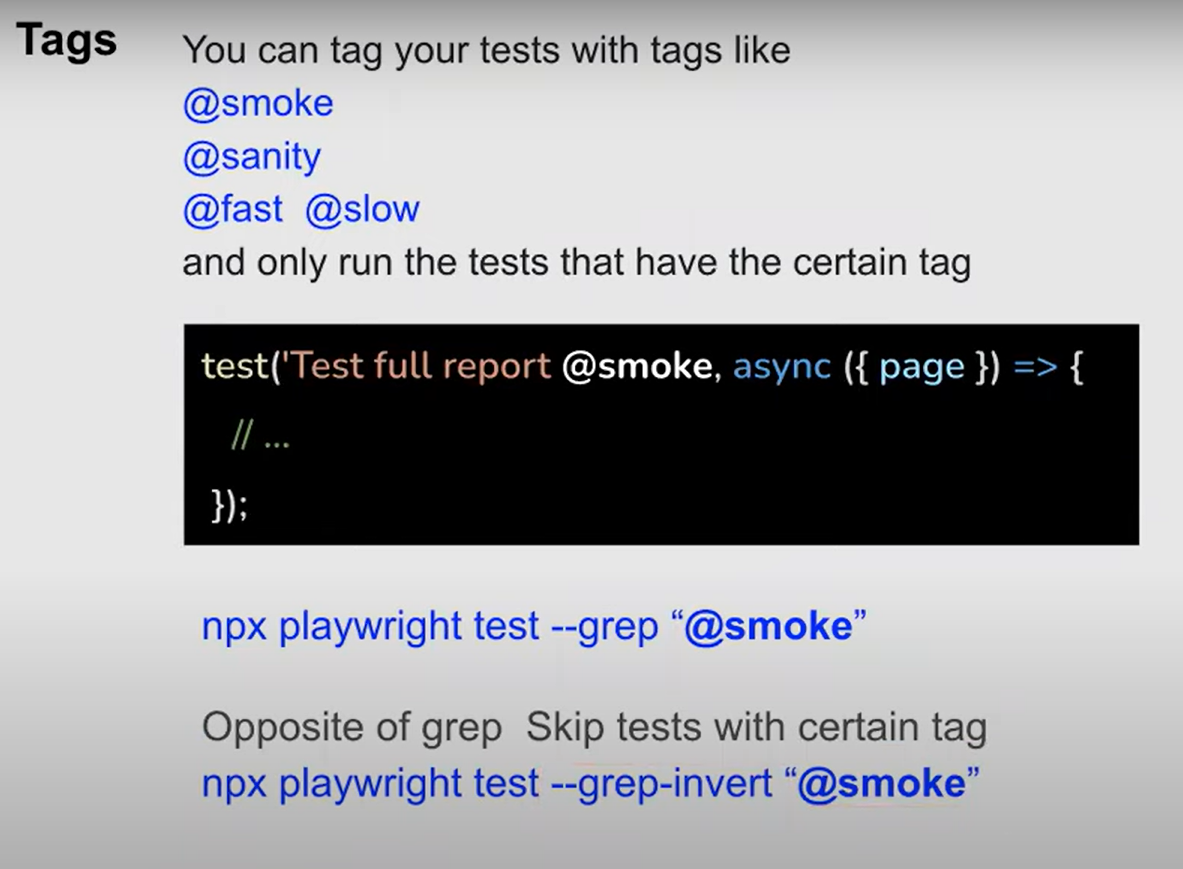


Annotations are keywords that contain some logical or conditional functionalities. Can be used in with test blocks to control execution of tests as needed.



The above codes will demo you the annotations in playwright.





Using tags in the test we can make sure only those will run when we use the command the tests tagged will only run. Just like profiles in the spring boot application.

***Playwright Page Object Model.***

Now let us suppose that we have a very big project where we have to execute many tests and there will be very long scripts and let us suppose that in future the developer changes the selector the same text box then we cant change the whole code inorder to fix the code so the best approach is to create a class for each page which contains all actions and attributes related to that page in that class which makes our work easier and only changing the attributes in that class makes our code working.

This is known as ***Playwright Page Object Model.***

Example code:

For Login Page:



And a test that uses Login Page:



In this way whenever the Login required we create and instantiate that object and use that object where we want and dividing this into one class for each makes its easier for us to do any changes that will happen.

***Playwright API Testing :***

Playwright can also be used to test the API’s also.

Page uses Browser Context where as request uses APIRequestContext.

For UI testing we will use page to simulate the actions and if we want to test the backend api’s we use request for that .

