What is Playwright

- Free & open Source Framework for web automation testing
- Created by Microsoft in 2020
- Apply for Web browser Apps and Mobile Web apps & API
- Supports for JS, TS, Java , Python , .NET (C#)
- All the modern browsers , Chromium , Webkit and firefox , IOS browsers in headed in headless modes
- Windows , MacOS , Linux
- Official website: https://playwright.dev/
- Related Courses: https://learn.microsoft.com/en-us/training/browse/?filter-products=play&products=playwright

Features

- Free and Open Source
- Multi browser , languages, OS
- Easy to setup and configure
- Functional Testing ,API testing ,Accessibility testing
- Built-in reports or custom reporters
- CI CD
- Parallel testing(no need third party library like TestNG)
- Auto waits, Timeouts (Ex; Loading page, finding elements)
- Capture screenshots and videos
- Fast test execution
- Built-in assertions

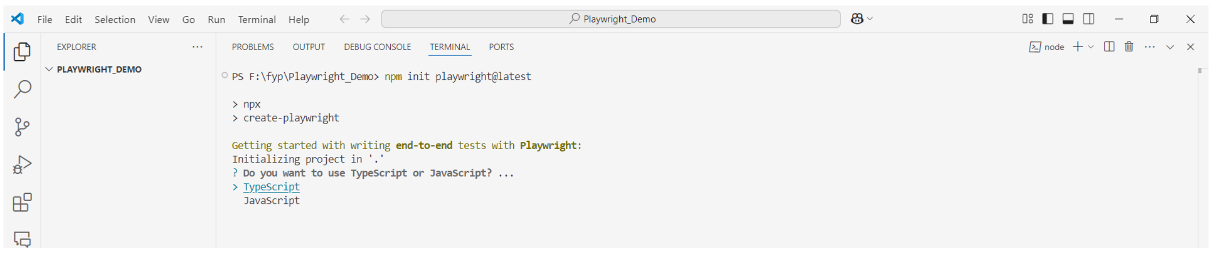
  Ex: When we try to click on login button playwright check whether;
    - The page is loaded
    - Elements are loaded
    - If the element is present
    - If the element is clickable

Prerequisites

- Node JS (npm -v) & (node -v)
- IDE (VS Code)

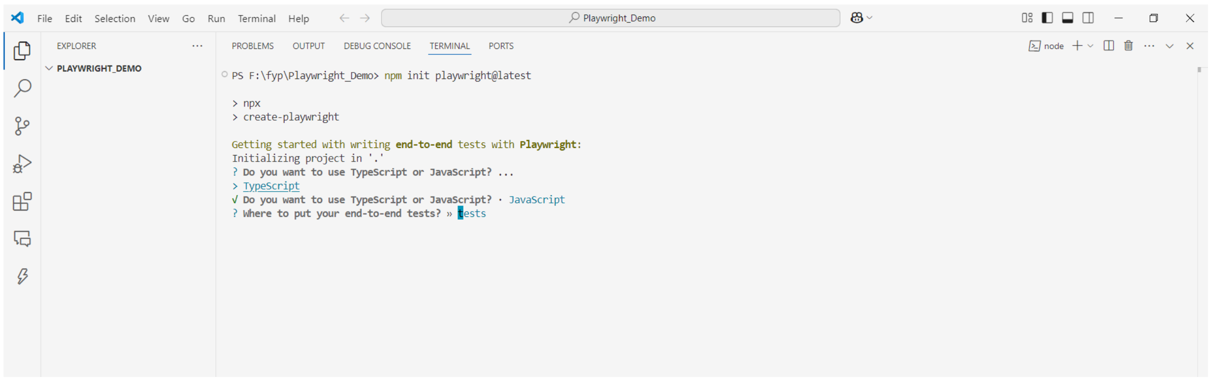
Installation- Approach 1

1. Create a folder in anywhere
2. Open folder in VS Code
3. Goto terminal and run command (npm init playwright@latest)


File Edit Selection View Go Run Terminal Help
Playwright_Demo
EXPLORER
PLAYWRIGHT_DEMO
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
node

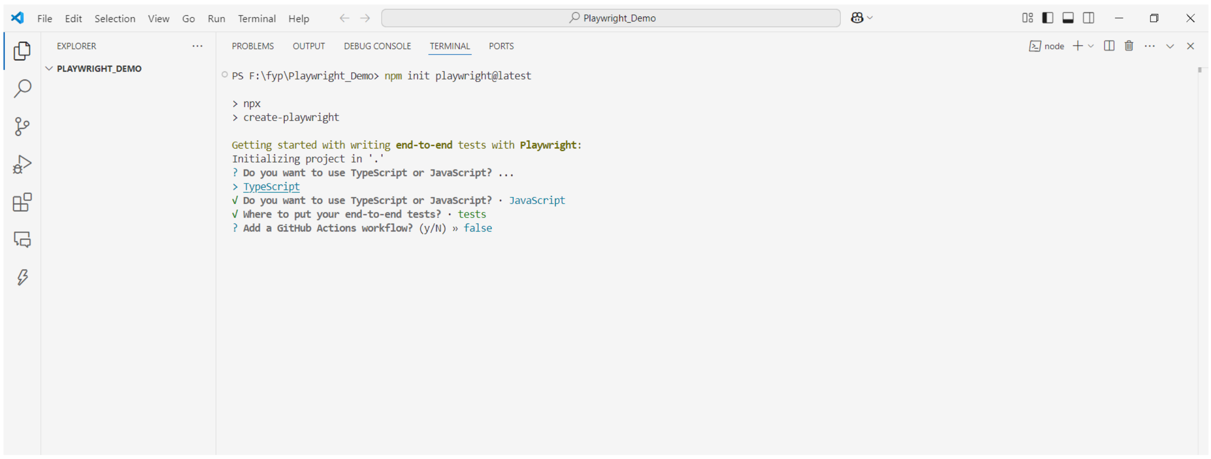
```
PS F:\fyp\Playwright_Demo> npm init playwright@latest

> npx
> create-playwright

Getting started with writing end-to-end tests with Playwright:
Initializing project in '.'
? Do you want to use TypeScript or JavaScript? ...
> TypeScript
  JavaScript
```

- Select a language



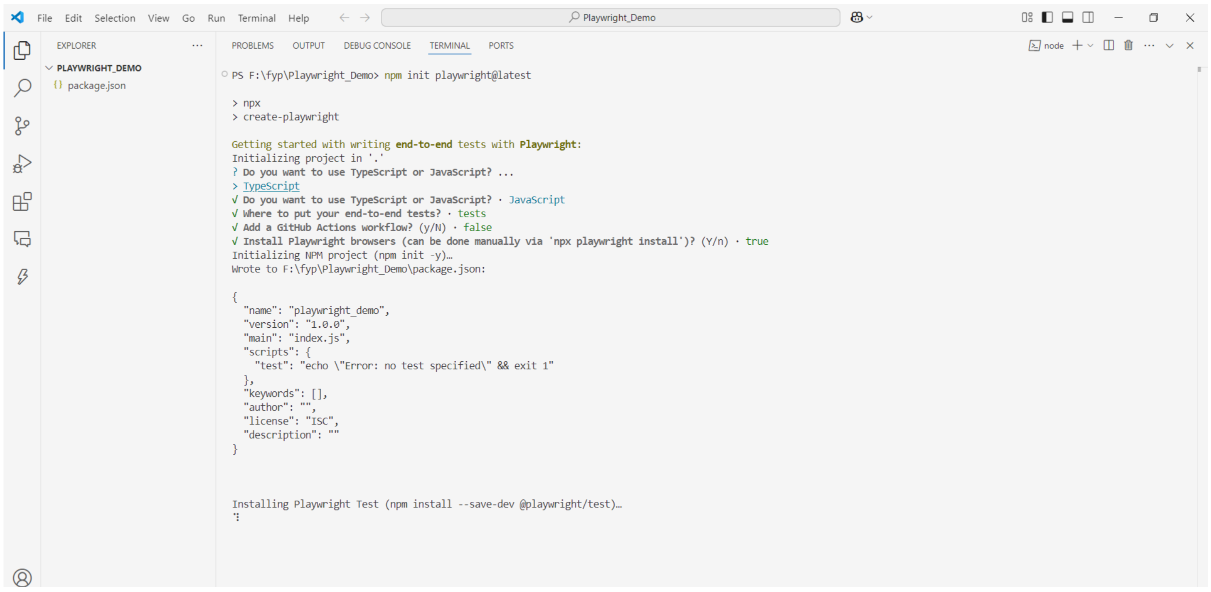
```
PS F:\fyp\Playwright_Demo> npm init playwright@latest

> npx
> create-playwright

Getting started with writing end-to-end tests with Playwright:
Initializing project in '.'
? Do you want to use TypeScript or JavaScript? ...
> TypeScript
√ Do you want to use TypeScript or JavaScript? · JavaScript
? Where to put your end-to-end tests? » tests
```

- Select JS with using arrow keys



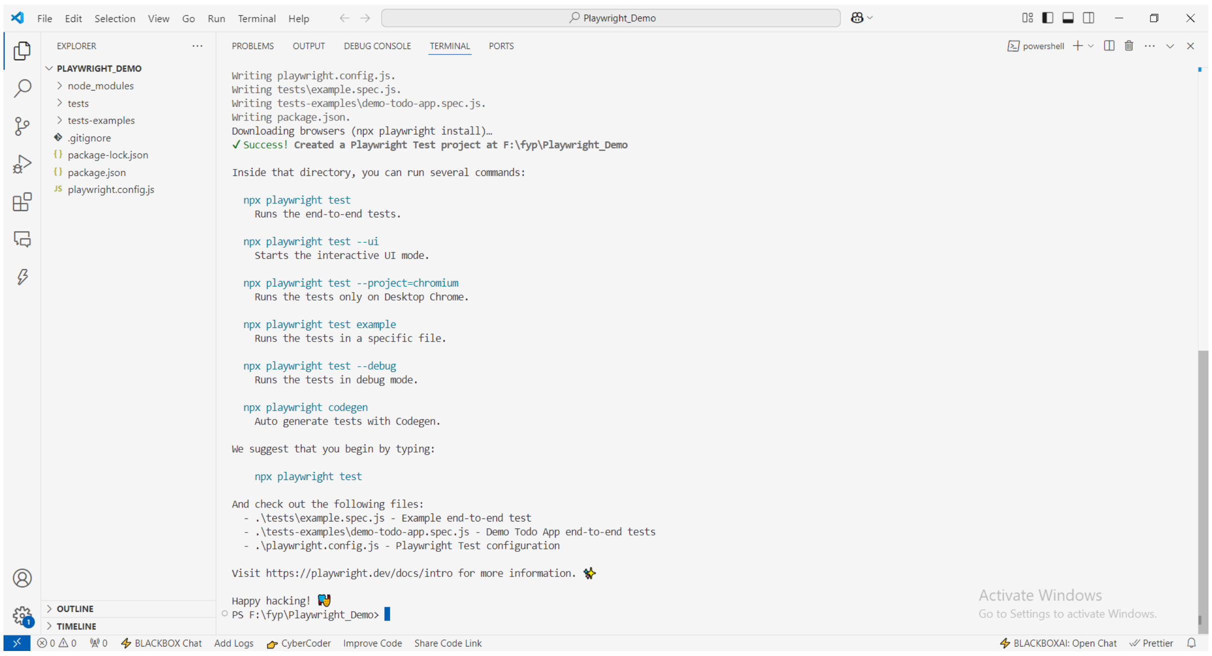
```
PS F:\fyp\Playwright_Demo> npm init playwright@latest

> npx
> create-playwright

Getting started with writing end-to-end tests with Playwright:
Initializing project in '.'
? Do you want to use TypeScript or JavaScript? ...
> TypeScript
√ Do you want to use TypeScript or JavaScript? · JavaScript
√ Where to put your end-to-end tests? · tests
? Add a GitHub Actions workflow? (y/N) » false
```

- Select whether need to create a github action flow



```
PS F:\fyp\Playwright_Demo> npm init playwright@latest

> npx
> create-playwright

Getting started with writing end-to-end tests with Playwright:
Initializing project in '.'
? Do you want to use TypeScript or JavaScript? ...
> TypeScript
√ Do you want to use TypeScript or JavaScript? · JavaScript
√ Where to put your end-to-end tests? · tests
√ Add a GitHub Actions workflow? (y/N) · false
√ Install Playwright browsers (can be done manually via 'npx playwright install')? (Y/n) · true
Initializing NPM project (npm init -y)…
Wrote to F:\fyp\Playwright_Demo\package.json:

{
  "name": "playwright_demo",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}



Installing Playwright Test (npm install --save-dev @playwright/test)…
```

- Give true to install browsers



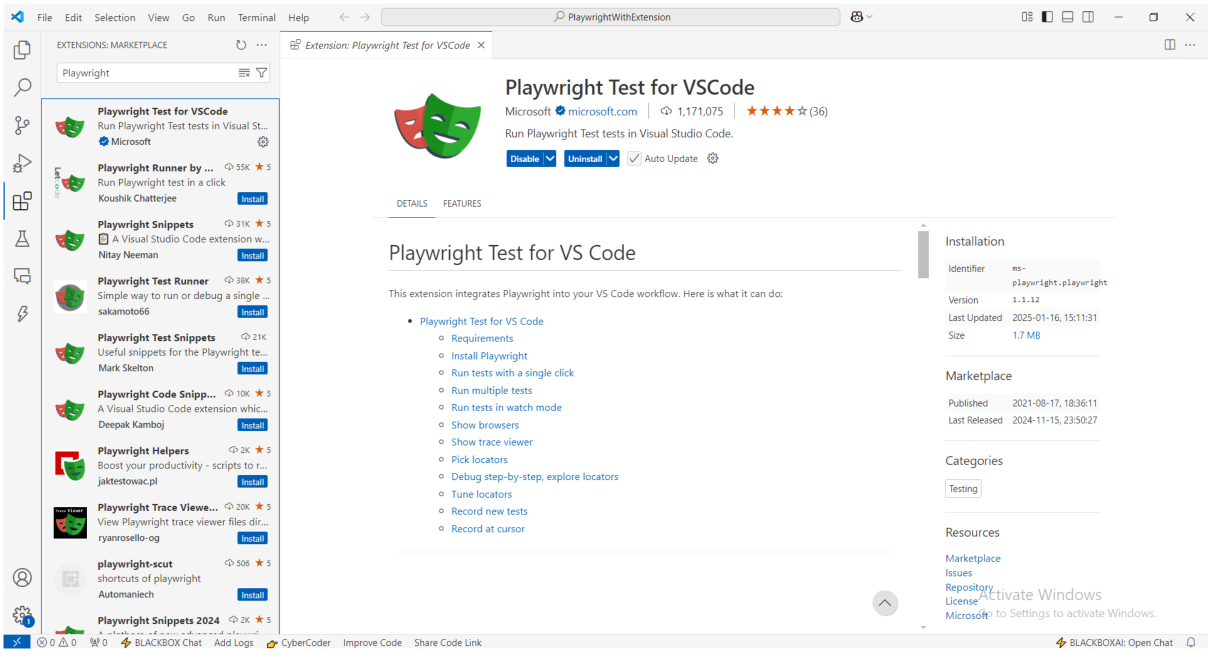
```
Writing playwright.config.js.
Writing tests\example.spec.js.
Writing tests-examples\demo-todo-app.spec.js.
Writing package.json.
Downloading browsers (npx playwright install)…
✓ Success! Created a Playwright Test project at F:\fyp\Playwright_Demo

Inside that directory, you can run several commands:

  npx playwright test
    Runs the end-to-end tests.

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

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

  npx playwright test example
    Runs the tests in a specific file.

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

  npx playwright codegen
    Auto generate tests with Codegen.

We suggest that you begin by typing:

    npx playwright test

And check out the following files:
  - .\tests\example.spec.js - Example end-to-end test
  - .\tests-examples\demo-todo-app.spec.js - Demo Todo App end-to-end tests
  - .\playwright.config.js - Playwright Test configuration

Visit https://playwright.dev/docs/intro for more information. ✨

Happy hacking! 🎭
PS F:\fyp\Playwright_Demo>
```

File Edit Selection View Go Run Terminal Help
Playwright_Demo
EXPLORER
PLAYWRIGHT_DEMO
node_modules
tests
tests-examples
.gitignore
package-lock.json
package.json
playwright.config.js
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
Activate Windows
Go to Settings to activate Windows.
BLACKBOX Chat Add Logs CyberCoder Improve Code Share Code Link
BLACKBOXAI: Open Chat Prettier

- Write a test script after generating the boiler template and after getting **“Happy hacking! 🎭”** message

<u>Create Project with Playwright Extension in VS Code - Approach 2</u>

1. Create a new folder and open in VS Code
2. Goto extensions section and install playwright extension from microsoft



Playwright Test for VSCode

Microsoft microsoft.com | 1,171,075 | ★★★★☆ (36)

Run Playwright Test tests in Visual Studio Code.

Playwright Test for VS Code

This extension integrates Playwright into your VS Code workflow. Here is what it can do:

- Playwright Test for VS Code
  - Requirements
  - Install Playwright
  - Run tests with a single click
  - Run multiple tests
  - Run tests in watch mode
  - Show browsers
  - Show trace viewer
  - Pick locators
  - Debug step-by-step, explore locators
  - Tune locators
  - Record new tests
  - Record at cursor

Installation

Identifier	ms-playwright.playwright
Version	1.1.12
Last Updated	2025-01-16, 15:11:31
Size	1.7 MB

Marketplace

Published	2021-08-17, 18:36:11
Last Released	2024-11-15, 23:50:27

Categories

Testing

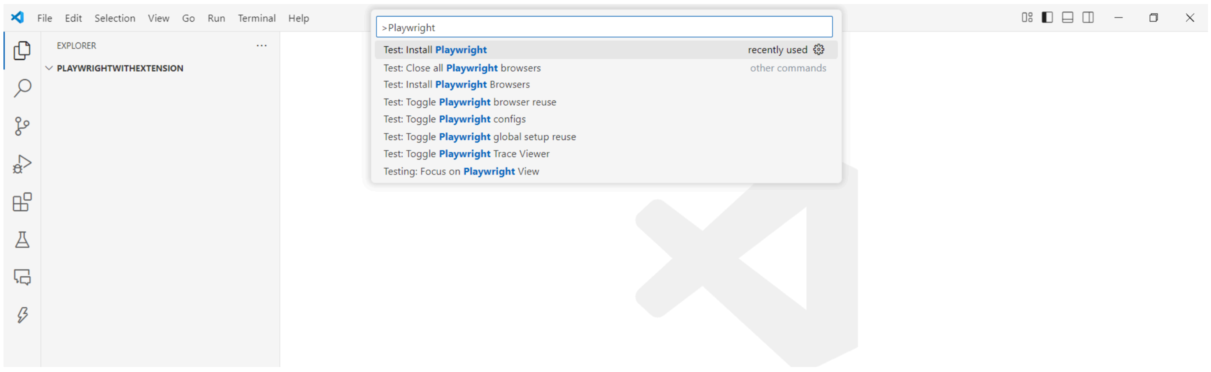
Resources

Marketplace
Issues
Repository
License
Microsoft

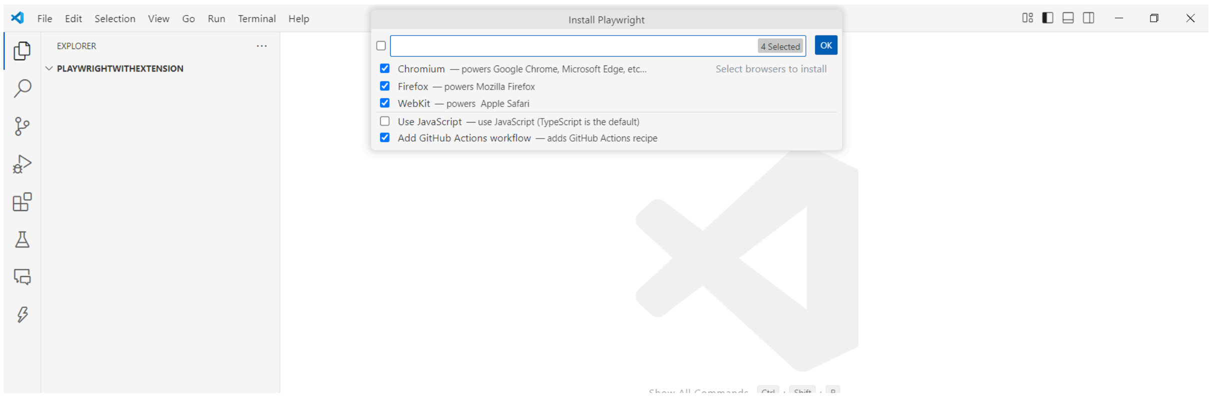
Extension	Downloads	Rating	Description	Publisher
Playwright Test for VSCode			Run Playwright Test tests in Visual St...	Microsoft
Playwright Runner by ...	55K	5	Run Playwright test in a click	Koushik Chatterjee
Playwright Snippets	31K	5	A Visual Studio Code extension w...	Nitay Neeman
Playwright Test Runner	38K	5	Simple way to run or debug a single ...	sakamoto66
Playwright Test Snippets	21K		Useful snippets for the Playwright te...	Mark Skelton
Playwright Code Snipp...	10K	5	A Visual Studio Code extension whic...	Deepak Kamboj
Playwright Helpers	2K	5	Boost your productivity - scripts to r...	jaktestowac.pl
Playwright Trace Viewe...	20K	5	View Playwright trace viewer files dir...	ryanrosello-og
playwright-scut	506	5	shortcuts of playwright	Automaniech
Playwright Snippets 2024	2K	5		

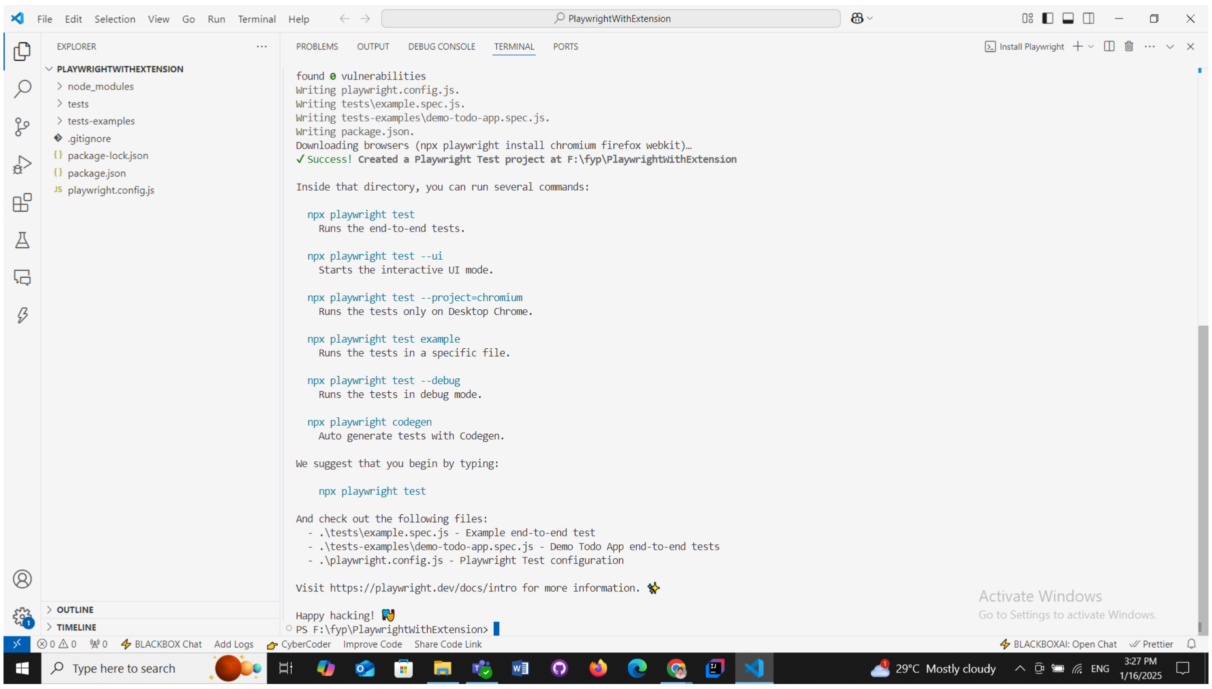
File Edit Selection View Go Run Terminal Help
PlaywrightWithExtension
EXTENSIONS: MARKETPLACE
Playwright
Extension: Playwright Test for VSCode
Disable
Uninstall
Auto Update
DETAILS
FEATURES
Install
Activate Windows
Go to Settings to activate Windows.
BLACKBOX Chat Add Logs CyberCoder Improve Code Share Code Link
BLACKBOXAI: Open Chat

3. View > Command Palette > type “Playwright”


```
>Playwright
Test: Install Playwright                          recently used
Test: Close all Playwright browsers               other commands
Test: Install Playwright Browsers
Test: Toggle Playwright browser reuse
Test: Toggle Playwright configs
Test: Toggle Playwright global setup reuse
Test: Toggle Playwright Trace Viewer
Testing: Focus on Playwright View
```

4. Select Install playwright and select the web browser options and CI CD options then press “OK”


```
Install Playwright
                                         4 Selected   OK
[✓] Chromium — powers Google Chrome, Microsoft Edge, etc...   Select browsers to install
[✓] Firefox — powers Mozilla Firefox
[✓] WebKit — powers Apple Safari
[ ] Use JavaScript — use JavaScript (TypeScript is the default)
[✓] Add GitHub Actions workflow — adds GitHub Actions recipe
```


```
found 0 vulnerabilities
Writing playwright.config.js.
Writing tests\example.spec.js.
Writing tests-examples\demo-todo-app.spec.js.
Writing package.json.
Downloading browsers (npx playwright install chromium firefox webkit)…
✓ Success! Created a Playwright Test project at F:\fyp\PlaywrightWithExtension

Inside that directory, you can run several commands:

  npx playwright test
    Runs the end-to-end tests.

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

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

  npx playwright test example
    Runs the tests in a specific file.

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

  npx playwright codegen
    Auto generate tests with Codegen.

We suggest that you begin by typing:

    npx playwright test

And check out the following files:
  - .\tests\example.spec.js - Example end-to-end test
  - .\tests-examples\demo-todo-app.spec.js - Demo Todo App end-to-end tests
  - .\playwright.config.js - Playwright Test configuration

Visit https://playwright.dev/docs/intro for more information. ✨

Happy hacking! 🎭
PS F:\fyp\PlaywrightWithExtension>
```

Project Structure

Package.json	Node project management file
Playwright.config.js	Configure file
Tests folder	Basic example test
Tests-example folder	Detailed example tests
.gitignore	To be used during git commit and push

Comparison between Selenium and Playwright

Aspect	Selenium	Playwright
Supported Languages	Java, Python, JavaScript, C#, Ruby, PHP, etc.	JavaScript/TypeScript, Python, Java, C#
Ecosystem	Older, mature, large community	Modern, growing, focused on speed and reliability
Browsers Supported	Chrome, Firefox, Safari, Edge	Chrome, Edge, Firefox, WebKit
Headless Mode	Supported, but requires separate configuration	Native
Execution Speed	Slower due to HTTP/WebDriver protocol overhead	Faster due to WebSocket-based communication
Auto-Waiting	Requires manual handling	Built-in
Cross-Browser Testing	Supported, but setup can be complex	Seamless, with fewer configurations
Flaky Tests	More common due to lack of built-in auto-waiting	Less common due to smart handling of waits

Options to Run theTests

npx playwright test	Run all tests on all browsers in headless mode
npx playwright test --workers 3	Run with 3 workers in parallel
npx playwright test ./tests/example.spec.js	Run a specific test file
npx playwright test example	Its pick the name of folder and run it
npx playwright test -g "has title"	Runs test with the title
npx playwright test --project=chromium	Runs on specific browser
npx playwright test --headed	Runs tests in headed mode

How to write tests

- Create a new file under the test folder
- Add module 'playwright/test'

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

- Create a test block -test(title,testFunction)

```
test('My demo test',async ({page})=>{
    await page.goto('https://www.google.co.uk/')
    await expect(page).toHaveTitle('Google')
})
```

Async = Before a function makes the function return a promise

Await = Before a function makes the function wait for the promise

Locate elements in Playwright
(https://playwright.dev/docs/locators)

- Property
- Xpath
- CSS

Locate Single element

Link/button
```
await page.locator('locator').click;
await page.click('locator');
```

Input Box
```
 await page.locator('locator').fill("value");
 await page.locator('locator').type("value");

 await page.fill('locator','value');
 await page.type(‘locator’,’value’);
```

Locate Multiple Web elements

```
Const elements = await page.$$(‘locator’);
```

Built-in Locators

- **page.getByRole()** to locate by explicit and implicit accessibility attributes.
- **page.getByText()** to locate by text content.
- **page.getByLabel()** to locate a form controlled by associated label's text.
- **page.getByPlaceholder()** to locate an input by placeholder.
- **page.getByAltText()** to locate an element, usually image, by its text alternative.
- **page.getByTitle()** to locate an element by its title attribute.
- **page.getByTestId()** to locate an element based on its `data-testid` attribute (other attributes can be configured).

Codegen in playwright
npx playwright codegen

Assertions

(https://testomat.io/blog/playwright-assertion-magic/#:~:text=If%20a%20hard%20assert%20fails,and%20all%20 assertions%20are%20 evaluated.)

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

await expect(page).toHaveURL()	Page has a URL
await expect(page).toHaveTitle()	Page has a title
await expect(locator).toBeVisible()	Element is visible
await expect(locator).toBeEnabled()	Element is enabled
await expect(locator).toBeChecked()	Checkbox is checked
await expect(locator).toHaveAttribute()	Element has a DOM attribute
await expect(locator).toHaveText()	Element matches text
await expect(locator).toContainText()	Element contains text
await expect(locator).toHaveValue()	Input has a value

Soft Assertions

If a soft assert fails, the test continues to execute, and all assertions are evaluated. Finally, a summary of assertion failures is provided.

```
 //Soft assertion
   await expect.soft(page).toHaveTitle('STORE1');
   await expect(page).toHaveURL("https://www.demoblaze.com/");
   await expect(page.locator('.navbar-brand')).toBeVisible();
```

Hard Assertions

If a hard assert fails, the test immediately stops, and the Playwright test case is marked as failed. Subsequent steps in the test script are not executed.

```
test('Soft Assertions',async({page})=>{
   await page.goto('https://www.demoblaze.com/');

   //Hard assertion
   await expect(page).toHaveTitle('STORE');
   await expect(page).toHaveURL("https://www.demoblaze.com/");
   await expect(page.locator('.navbar-brand')).toBeVisible();
 })
```

Mouse Actions

1. **Hover over**

```
test('Mouse Hover',async ({page})=>{
   await page.goto('https://practice.expandtesting.com/hovers');
   await expect(page.locator("//div[@class='figure'][1]")).toBeVisible();
   await page.hover("//div[@class='figure'][1]");
})
```

2. **Right Click**

```
test('Mouse Right Click',async ({page})=>{
   await
page.goto('https://swisnl.github.io/jQuery-contextMenu/demo.html');
   const Button = page.locator("//span[@class='context-menu-one btn
btn-neutral']");
   await Button.click({button:'right'});
})
```

3. **Double Click**

```
test('Mouse Double Click',async ({page})=>{
   await page.goto('https://testautomationpractice.blogspot.com/');
   const copyBtn = page.locator("//button[normalize-space()='Copy
Text']");
   await copyBtn.dblclick();
   const field2 = await page.locator('id=field2');
   await expect(field2).toHaveValue('Hello World!');
})
```

4. **Drag and Drop**

```
test('Drag and Drop',async ({page})=>{
   await page.goto('https://practice.expandtesting.com/drag-and-drop');
   const movable = page.locator('id=column-a');
   const droppable = page.locator('id=column-b');

   //Approach 1
   await movable.hover();
   await page.mouse.down();

   await droppable.hover();
   await page.mouse.up();

   //Approach 2
   await movable.dragTo(droppable);
})
```

Grouping tests

(https://playwright.dev/docs/test-annotations#group-tests)

Using describe we can create new test group

```
test.describe('Group 1',()=>{

   test('Mouse Hover',async ({page})=>{
      await page.goto('https://practice.expandtesting.com/hovers');
      await expect(page.locator("//div[@class='figure'][1]")).toBeVisible();
      await page.hover("//div[@class='figure'][1]");
   })
})
```

Using .only we can only run this block

```
test.describe.only('Group 2',()=>{
   test('Mouse Hover 1',async ({page})=>{
      await page.goto('https://practice.expandtesting.com/hovers');
      await expect(page.locator("//div[@class='figure'][1]")).toBeVisible();
      await page.hover("//div[@class='figure'][1]");
   })

   test('Mouse Hover 2',async ({page})=>{
      await page.goto('https://practice.expandtesting.com/hovers');
      await expect(page.locator("//div[@class='figure'][1]")).toBeVisible();
      await page.hover("//div[@class='figure'][1]");
   })
}
```

HOOKS

```
         test.beforeAll(async()=>{//once execute
            console.log('This is before all hook');
         })

         test.afterAll(async()=>{
            console.log('This is after all hook');
         })

         test.beforeEach(async()=>{
            console.log('This is before each hook');
         })

         test.afterEach(async()=>{
            console.log('This is after each hook');
         })
```

Capture screenshots

Approach 1

Using this we can customise when we need to capture the screenshots

- **Capture the page**

```
test('Page screenshot', async({page})=>{
   await page.goto('https://demo.nopcommerce.com/');
   await page.screenshot({path:'tests/screenshots/'+Date.now()+'HomePage.png'});
   page.close();
});
```

- **Capture the full page**

```
test('Full page screenshot', async({page})=>{
   await page.goto('https://demo.nopcommerce.com/');
   await
page.screenshot({path:'tests/screenshots/'+Date.now()+'HomePage_fullpage.png',full
Page:true});
   page.close();
});
```

- **Capture the particular element**

```
test('Element screenshot', async({page})=>{
   await page.goto('https://demo.nopcommerce.com/');
   await page.locator("//img[@title='Show products in category
Apparel']").screenshot({path:'tests/screenshots/'+Date.now()+'HomePage_Element.pn
g',fullPage:true});
   page.close();
});
```

Approach 2

This takes screenshots of all the test methods by default

```
use: {
    screenshot: 'on',
  },
```

This capture screenshots when test method got fail

```
use: {
    screenshot: 'only-on-failure',
  },
```

Tagging

```
test('test1 @sanity',async()=>{
   console.log('This is test 1');
})

test('test 2@reg',async()=>{
   console.log('This is test 2');
})
```

```
npx playwright test --grep "@reg"
```

Annotations

- test.skip() marks the test as irrelevant. Playwright does not run such a test. Use this annotation when the test is not applicable in some configuration.
- test.fail() marks the test as failing. Playwright will run this test and ensure it does indeed fail. If the test does not fail, Playwright will complain.
- test.fixme() marks the test as failing. Playwright will not run this test, as opposed to the `fail` annotation. Use `fixme` when running the test is slow or crashes.
- test.slow() marks the test as slow and triples the test timeout.

Allure Report Generation

1.Installation of ‘allure-playwright’ module

```
npm i -D @playwright/test allure-playwright
```

2.Installing allure command line

```
npm install -g allure-commandline - -save-dev
```

3.playwright.config.js

```
reporter =[‘allure-playwright’,{outputFolder:’allure-results’}]
```

4.Run the tests

5.Generate allure report:

```
allure generate allure-results -o test-report --clean
```

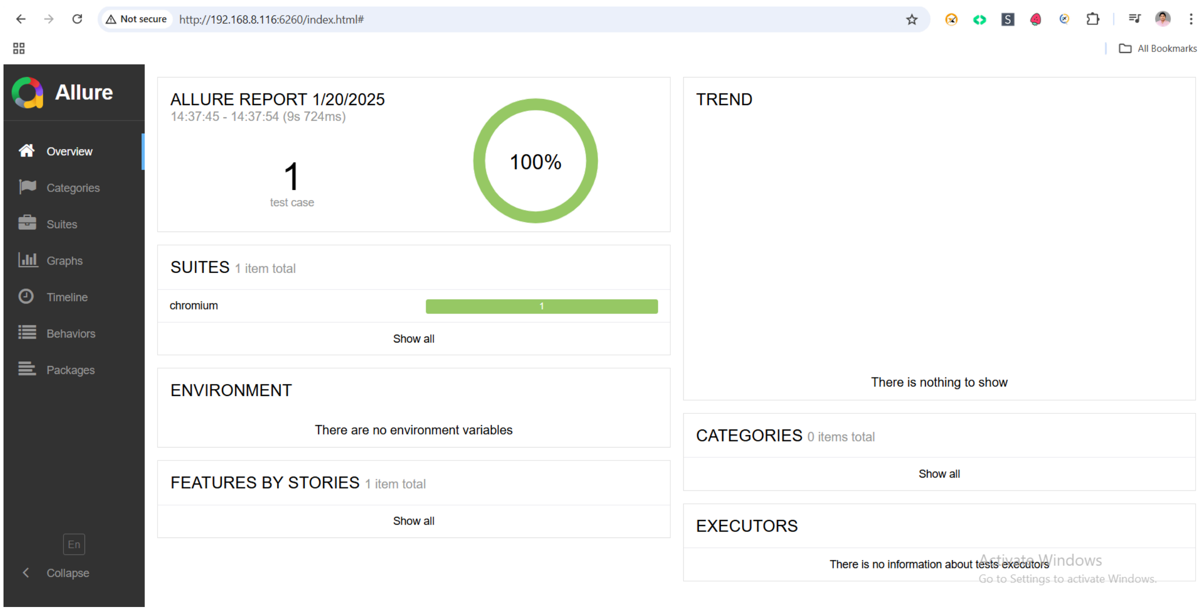
allure-results: where the json report generated

test-report:where i need to create my allure report

6.Open allure report

```
allure open test-report
```

test-report:where i created the allure report



ALLURE REPORT 1/20/2025
14:37:45 - 14:37:54 (9s 724ms)
1
test case
100%

SUITES 1 item total
chromium 1
Show all

ENVIRONMENT
There are no environment variables

FEATURES BY STORIES 1 item total
Show all

TREND
There is nothing to show

CATEGORIES 0 items total
Show all

EXECUTORS
There is no information about tests executors

POM

1. Create a new folder and open in IDE
2. Install & Setup Playwright by running **npm init playwright@latest**
3. Create a new folder “pages” that will contain all the page objects.

```
exports.LoginPage = class LoginPage {
  constructor(page) {
    this.page = page;
    this.username_textbox = page.getByPlaceholder("Username");
    this.password_textbox = page.getByPlaceholder("Password");
    this.login_button = page.getByRole("button", { name: "Login" });
  }

  //Action methods
  //   enterUsername(){//atomic methods
  //   }

  //   enterPassword(){
  //   }

  //   clickOnLogin(){
  //   }

  async gotoLoginPage() {
    await this.page.goto(

"https://opensource-demo.orangehrmlive.com/web/index.php/auth/login
"
    );
  }

  async login(username, password) {
    //single methods
    await this.username_textbox.fill(username);
    await this.password_textbox.fill(password);
    await this.login_button.click();
  }
};
```

4. **Create a new folder “tests” that will contain all the test cases**

```
import { test, expect } from "@playwright/test";
import { LoginPage } from "../pages/LoginPage";

test("Login with valid credentials", async ({ page }) => {
  const Login = new LoginPage(page);

  await Login.gotoLoginPage();
  await Login.login('Admin','admin123');
});
```

.env File

In Playwright, you can use `.env` files to store environment-specific configuration values like URLs, usernames, and passwords.

- **First install the dotenv dependency**

```
npm install dotenv
```

- **Configure the values in .env file**

```
baseUrl=https://opensource-demo.orangehrmlive.com/web/index.php/auth/logi
n
username=Admin
password=admin123
```

- **Import the dotenv and give the path to file**

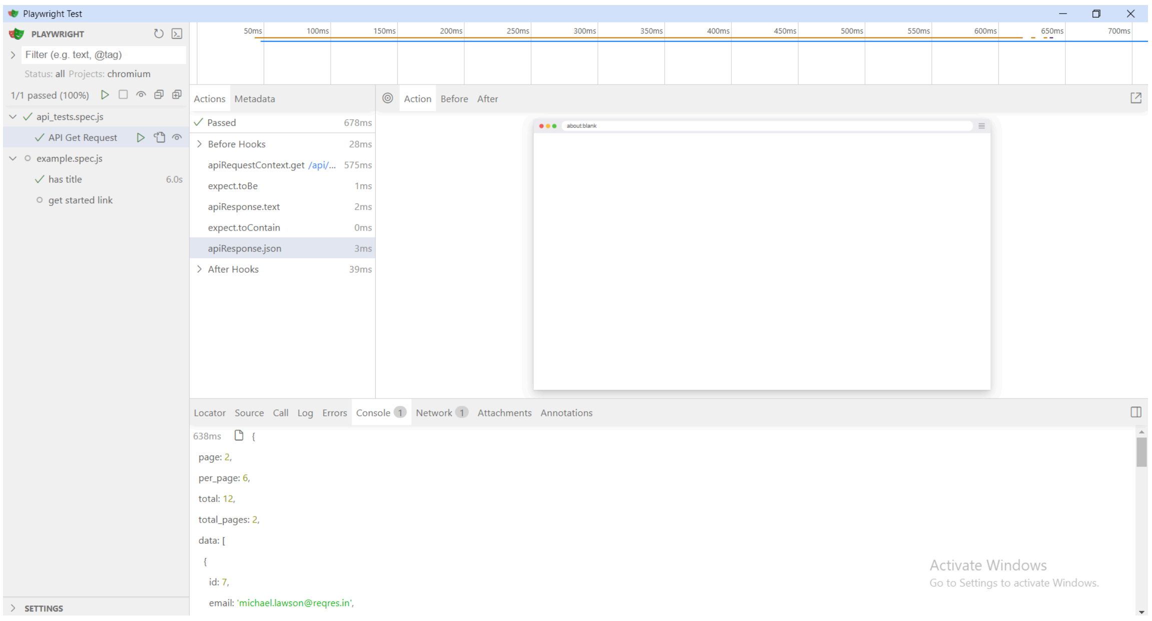
```
import { test, expect } from "@playwright/test";
import dotenv from "dotenv";
import path from "path";
dotenv.config({ path: path.resolve(__dirname, '..', '.env') });

const baseUrl = String(process.env.baseUrl);
const username = String(process.env.username);
const password = String(process.env.password);

test("Verify login for the OrangeHrm", async ({ page }) => {
  await page.goto(baseUrl);
  await page.getByPlaceholder("Username").fill(username);
  await page.getByPlaceholder("Password").fill(password);
  await page.getByRole("button", { name: "Login" }).click();
});
```

<u>API Testing with Playwright</u>

Resources:

Demo Site(https://reqres.in/)

- **Get the UI of playwright**
  npx playwright test --ui



```
✓ Passed                          678ms
> Before Hooks                    28ms
  apiRequestContext.get /api/...  575ms
  expect.toBe                     1ms
  apiResponse.text                2ms
  expect.toContain                0ms
  apiResponse.json                3ms
> After Hooks                     39ms
```

```
638ms  {
 page: 2,
 per_page: 6,
 total: 12,
 total_pages: 2,
 data: [
  {
   id: 7,
   email: 'michael.lawson@reqres.in',
```

- **Add imports**
  import {test,expect} from '@playwright/test'

- **Create a test using request context**
  test("API Get Request", async({request})=>{})

- **Send a GET request & store response in a variable**

const response = await request.get('<u>https://reqres.in/api/users?page=2</u>');

- **Verify the status code of the response is 200**
  expect (response.status()).toBe(200);

- **Check response contains some text**
  const text = await response.text();
  expect(text).toContain('Michael');

- **Write response on the console**
  console.log(await response.json());

The Full code

```
import { test, expect } from "@playwright/test";
import { request } from "http";

test("API Get Request", async ({ request }) => {
  const response = await
request.get("https://reqres.in/api/users?page=2");
  expect(response.status()).toBe(200);
  const text = await response.text();
  expect(text).toContain("Michael");
  console.log(await response.json());
});

test("API Post Request", async ({ request }) => {
  const response = await request.post('https://reqres.in/api/users', {
    data: {
      name: "Perera",
      job: "leader",
    },
  });
  expect(response.status()).toBe(201);
  const text = await response.text();
  expect(text).toContain("Perera");
  console.log(await response.json());
});

test("API PUT Request", async ({ request }) => {
    const response = await request.put('https://reqres.in/api/users/2',
{
      data: {
        name: "Ashani",
        job: "lawyer",
      },
    });
    expect(response.status()).toBe(200);
    const text = await response.text();
    expect(text).toContain("Ashani");
    console.log(await response.json());
});
```

```
test("API DELETE Request", async ({ request }) => {
    const response = await
request.delete('https://reqres.in/api/users/2');
    expect(response.status()).toBe(204);

});
```