MCP (Model Context Protocol)+ Copilot Setup in VSCode

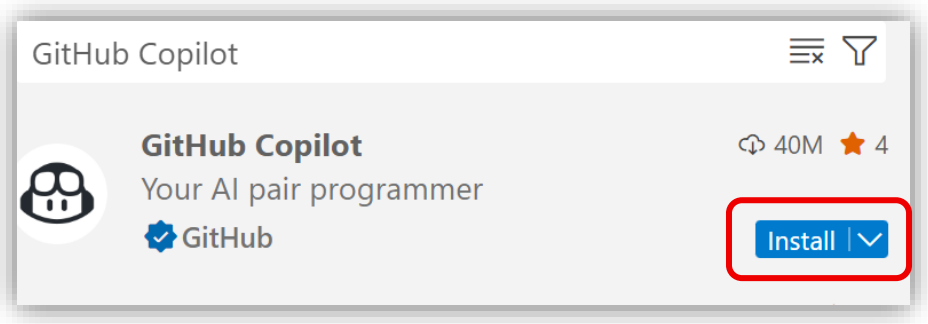
Prerequisites:

- Visual Studio Code (VS Code) installed
- Playwright VSCode Extension (Optional)
- GitHub account
- Playwright installation - npm init playwright@latest

✅ Step 1: Install GitHub Copilot Extension

Action:

1. Open **VS Code**
2. Go to **Extensions** tab (Ctrl+Shift+X)
3. Search for **GitHub Copilot**
4. Click **Install.**



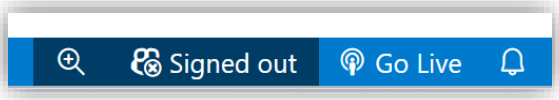
✅ Step 2: Sign-In to GitHub Copilot

Action:

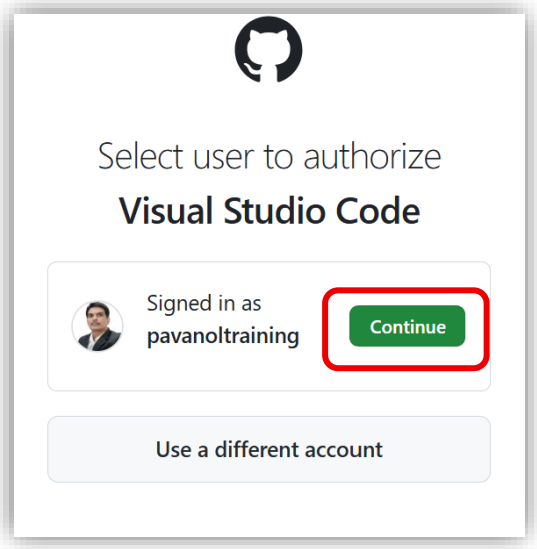
1. After installation, click **"Sign in"** (a pop-up might appear).
2. VS Code will redirect to your browser.
3. Authorize Copilot with your GitHub account.
4. Once authorized, you will see GitHub Copilot activated in VS Code.

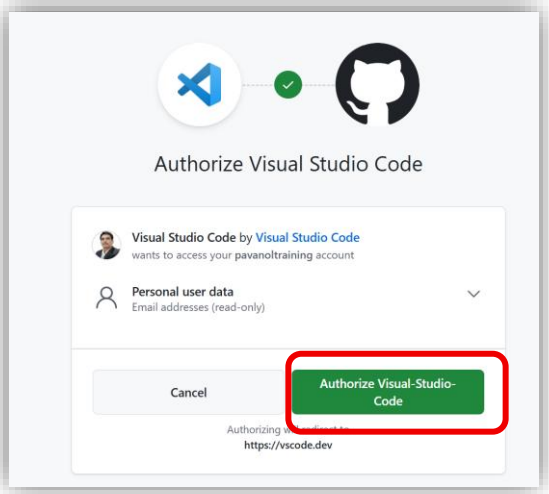
Capture the browser window showing authorization and the VS Code notification showing successful login (Follow the below screenshots).

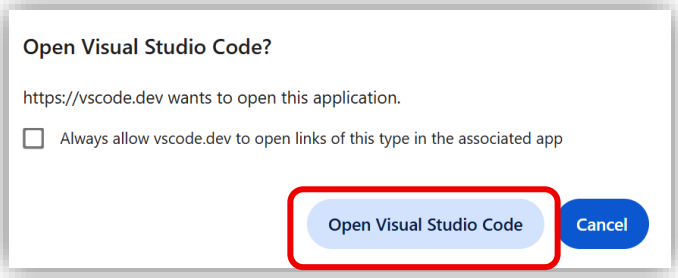
Sign-in To Copilot



Click-on to Continue.



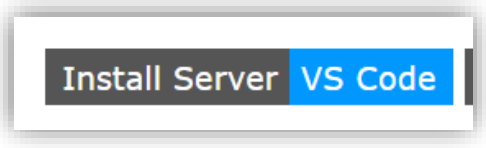


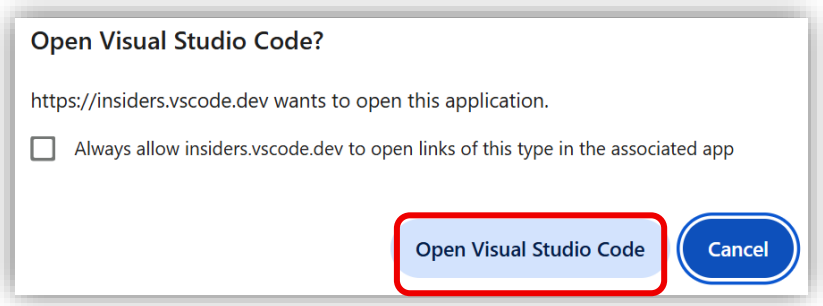


✅ Step 3: Install Playwright MCP Server

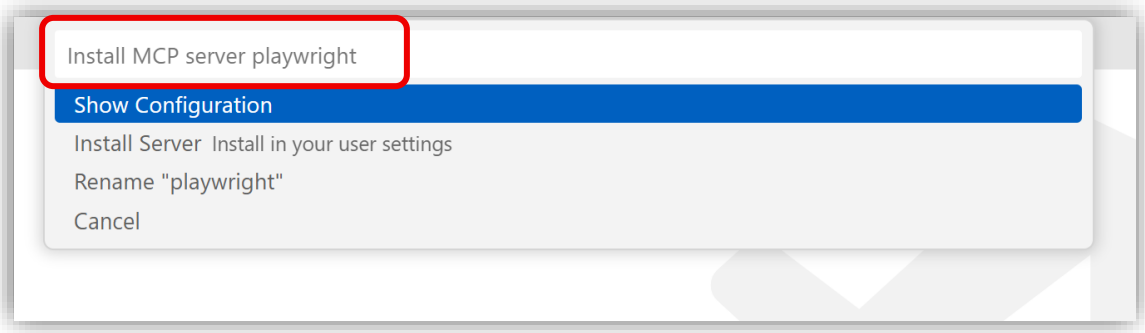
Action:

1. Go to this link: https://github.com/microsoft/playwright-mcp
2. Scroll down and look for installation instructions
3. Click on **Install Server VS Code button** (see below) then **Click on Open Visual Studio Code button** on **popup**.





4. Under the VS Code section, click "**Install MCP server for Playwright**"



✅ Step 4: Open Copilot Chat

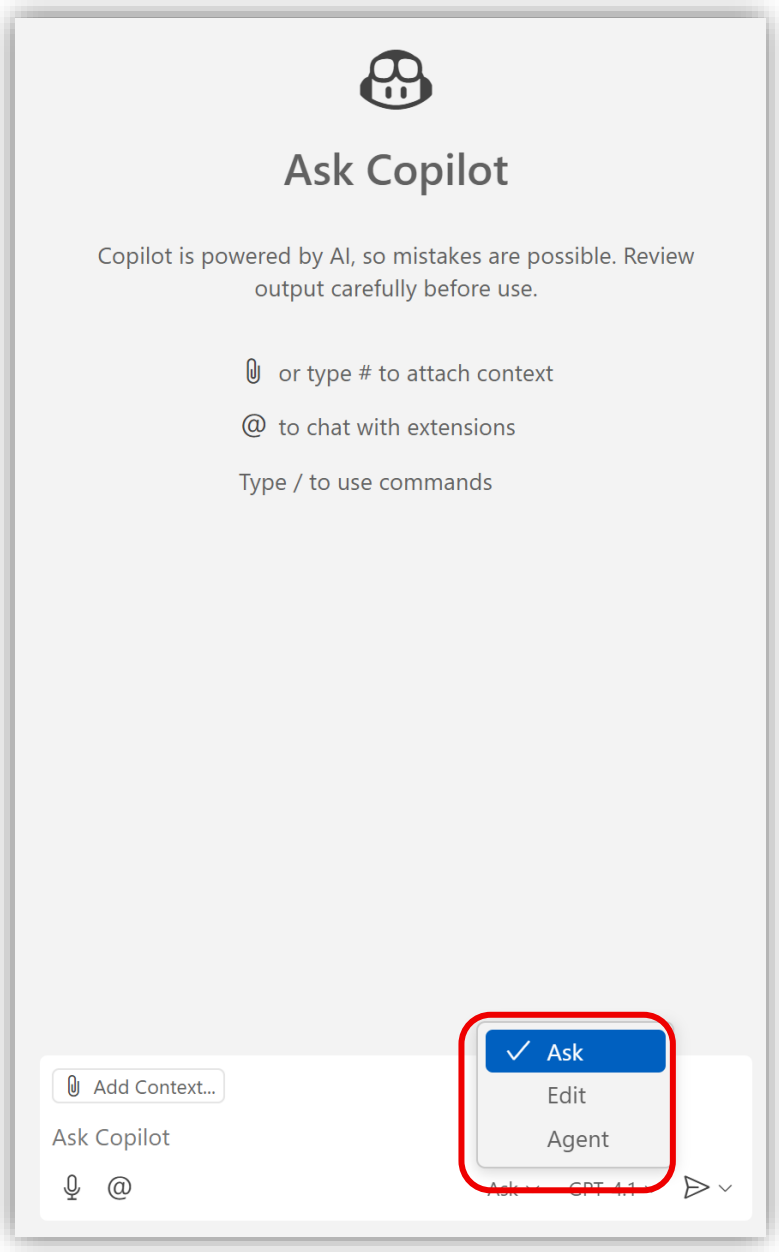
Action:

1. In the **Activity Bar**, click on **Copilot Chat icon**
2. A chat window will open where you can interact with Copilot

✅ Step 5: Select the Agent in GitHub Copilot

Action:

1. In the Copilot Chat, choose the **Agent** dropdown (top of chat window)
2. Select the **Claude 3.5/ChatGpt** from the list (may appear after installing MCP)



Generating Playwright Web Tests using MCP+Copilot:

Step 1: Create test context file

1. Inside your project folder, create a new file named: **generatewebtest.prompt.md**
2. Paste the following prompt:

```
You are a playwright test generator.
You are given a scenario and you need to generate a playwright test for it.
DO NOT generate test code based on the scenario alone.
DO run steps one by one using the tools provided by the Playwright MCP.
Only after all steps are completed, emit a Playwright TypeScript test that uses
@playwright/test based on message history
Save generated test file in the tests directory.
Execute the test file and iterate until the test passes.
```

Step 2: Add context file to Copilot Chat Context

In Copilot Chat, use the + icon or context menu to add the **context file to chat context.**

Step 3: Generate Playwright Test

Paste the following message/prompt into the Copilot chat:

```
Generate a Playwright test for the following scenario:
1. Navigate to http://www.automationpractice.pl/index.php
2. Search for 'T-shirts'
3. Verify the "Faded Short Sleeve T-shirts" in the list.
```

Generating Page Object Model(POM) using MCP+Copilot:

No test context needed.

Create a **POM** model for below steps:

1. Navigate to http://www.automationpractice.pl/index.php

2. Search for 'T-shirts'

3. Verify the "Faded Short Sleeve T-shirts" in the list.

Generating API Test using MCP+Copilot:

Step 1: Create test context file

1. Inside your project folder, create a new file named: **generateapicontext.txt**
2. Paste the following prompt:

You are an API test generator using Playwright MCP.

Use Playwright's `request` context and `@playwright/test` framework.

The test should:

- Send HTTP requests to the target API.

- Validate the status code, response body, and schema (if applicable).

- Use async/await syntax.

- Print useful logs for debugging if needed.

- Export the test to a `.spec.ts` file under the `/tests` folder.

Do not generate test code until all steps are fully explored and validated.

Step 2: Add context file to Copilot Chat Context

In Copilot Chat, use the + icon or context menu to add the **context file** to chat context.

Step 3: Generate Playwright API Test

Paste the following message/prompt into the Copilot chat:

Generate a Playwright API test for the following scenario:

1. Define the API endpoint URL: `https://fakestoreapi.com/products/1`.
2. Send a GET request to the endpoint.
3. Verify the response status is 200.
4. Validate the response contains these keys: `id`, `title`, `price`, `category`, and `description`.
5. Optionally validate the data types using a JSON Schema (Ajv).
6. Log the product title and price to the console.