Cross-Browser Automation with Selenium Using a Different Programming Language

Name: Hoang Anh Le

Course: Introduction to Selenium

Contents

I. Objective 3
II. Steps to complete the project 3
1. Choose a different programming language 3
2. Choose a different browser 4
3. Set up the environment 4
4. Rewrite an Existing Script 7
5. Run and debug the Script 9
III. Conclusion 10
1. Result 11
2. Challenges 11
3. Enhancements 11

I. Objective

Selenium WebDriver with a different browser and programming language than those usedin the course. This involved adapting and rewriting an automation script to run in a newenvironment, reinforcing my understanding of Selenium's core concepts. A key aspect wasrecognizing the nuances and differences in syntax, setup, and browser behavior whenswitching programming languages and browsers. This project reinforces Selenium's cross-language and cross-browser capabilities, providing hands-on experience in adapting to newtools and environments.

II. Steps to complete the project

1. Choose a different programming language

Choosing **C# for Selenium WebDriver** automation can be advantageous for several reasons, particularly within a Microsoft-centric development environment:

- Integration with .NET Ecosystem:

If your application is built on the .NET framework or you are using Microsoft technologies like Azure DevOps, C# naturally integrates into this ecosystem, streamlining development and testing workflows.

- Powerful IDE Support (Visual Studio):

Visual Studio, a comprehensive Integrated Development Environment (IDE), offers robust features for C# development, including IntelliSense for code completion, powerful debugging tools, and refactoring capabilities, which can significantly enhance productivity in writing and maintaining Selenium tests.

- Strong Typing and Code Maintainability:

C# is a strongly-typed language, which can lead to more robust and maintainable code by catching potential errors at compile time rather than runtime. This can be especially beneficial for large and complex test suites.

- Modern Testing Frameworks:

C# integrates well with popular testing frameworks like NUnit and MSTest, providing a structured environment for organizing and executing Selenium tests. It also supports Behavior-Driven Development (BDD) frameworks like SpecFlow.

- Performance:

C# offers good performance for running Selenium tests, especially when integrated within the .NET framework, making it suitable for projects requiring efficient test execution.

- Community and Resources:

While Java boasts a larger community for Selenium, C# also has a substantial and active community, with numerous online resources, tutorials, and forums available to support learning and problem-solving.

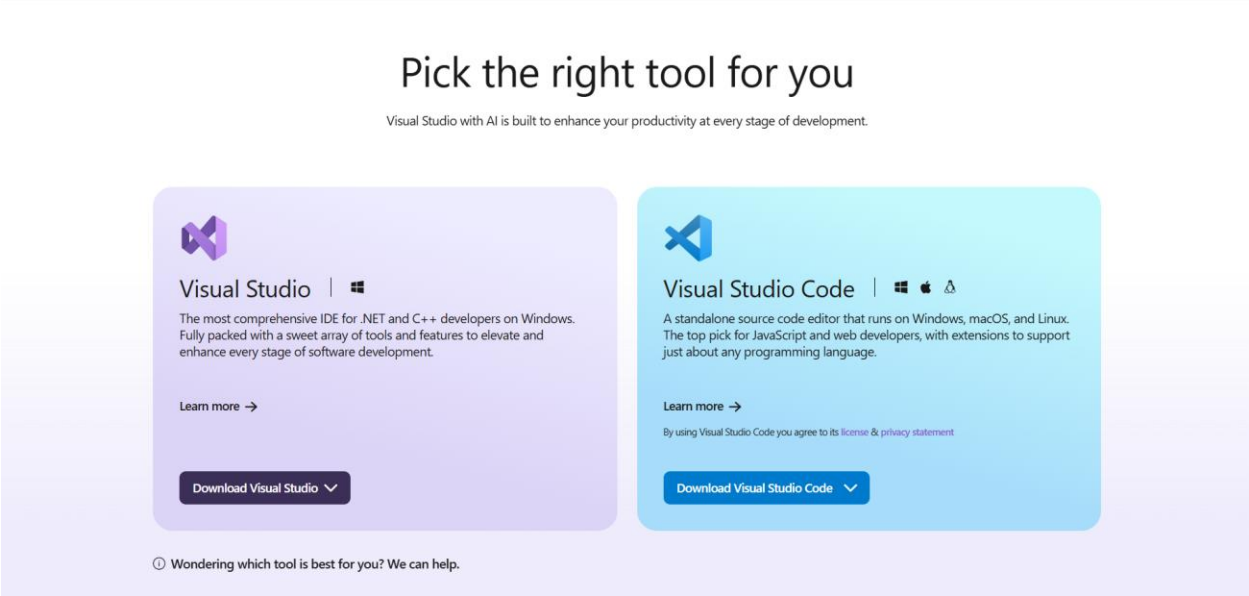
2. Choose a different browser

I choose Chrome for Selenium WebDriver because of its large user base, excellent documentation, fast performance, and robust features like headless mode, Chrome DevTools Protocol (CDP) access, and browser-specific capabilities that provide comprehensive testing control. This combination of popularity and advanced functionality makes it a preferred choice for web automation.

3. Set up the environment

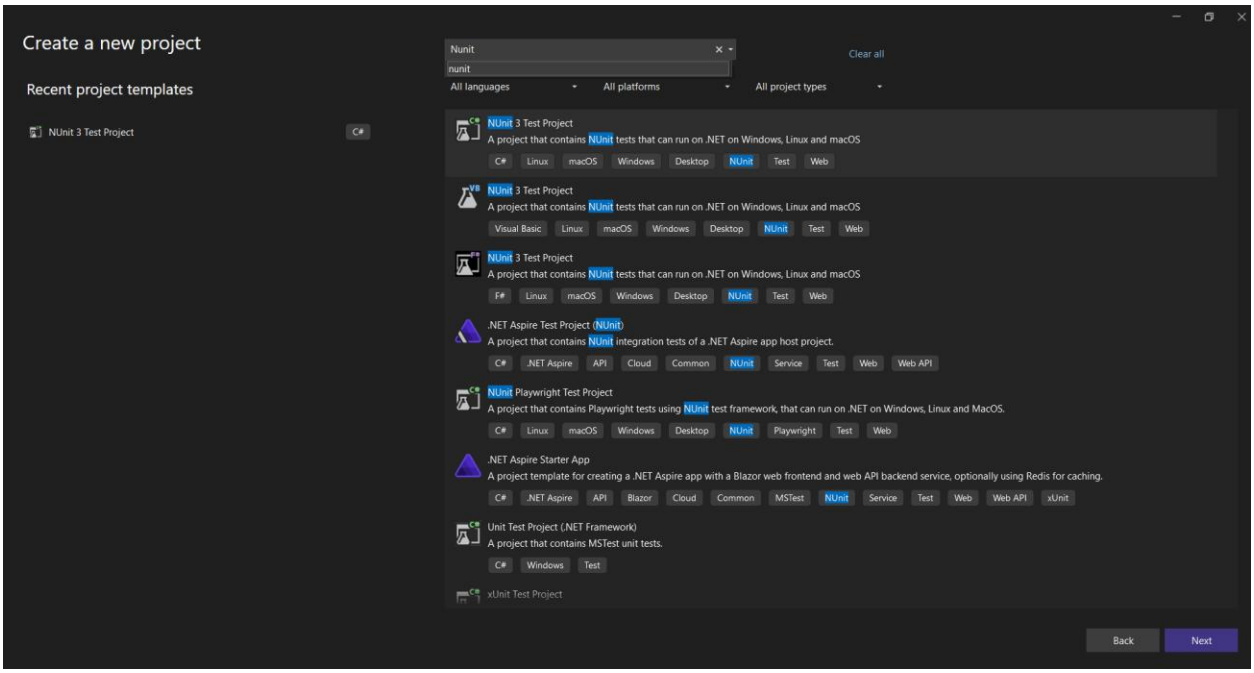
In this case, I am using Visual Studio as the IDE throughout the process.

- Download Visual Studio on website: https://visualstudio.microsoft.com/
- On this website, scroll down until you see this part

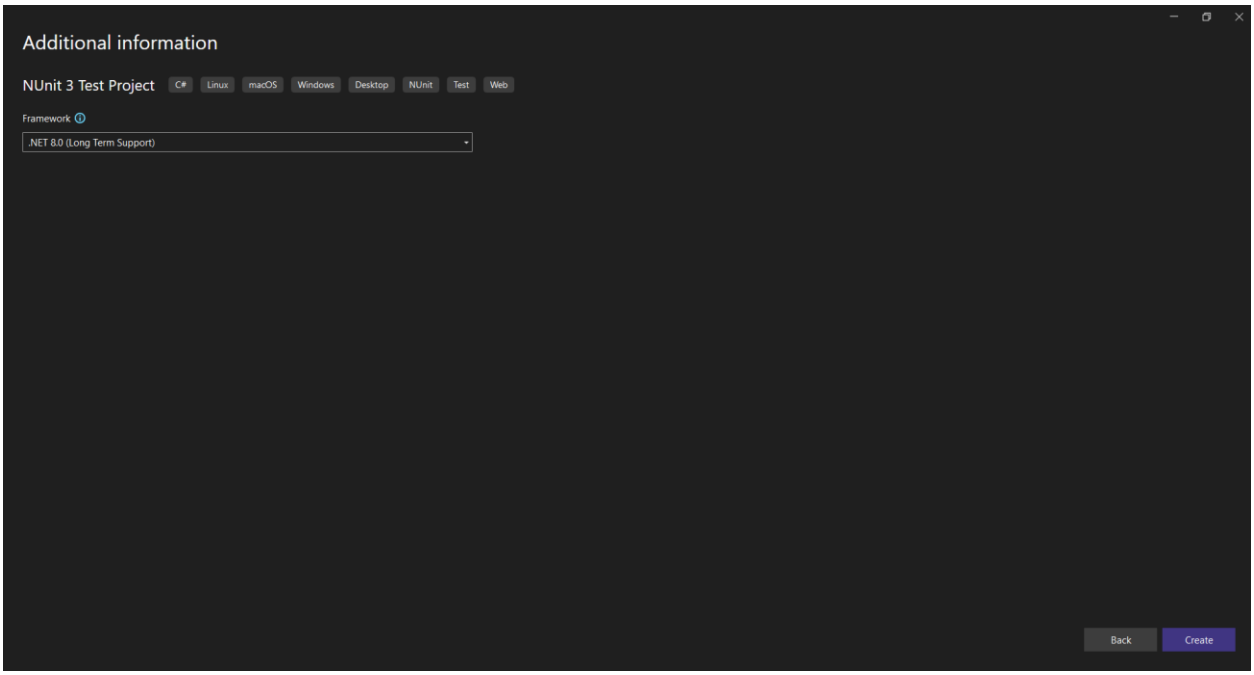


Choosing the purple one and set up as the tutorial.

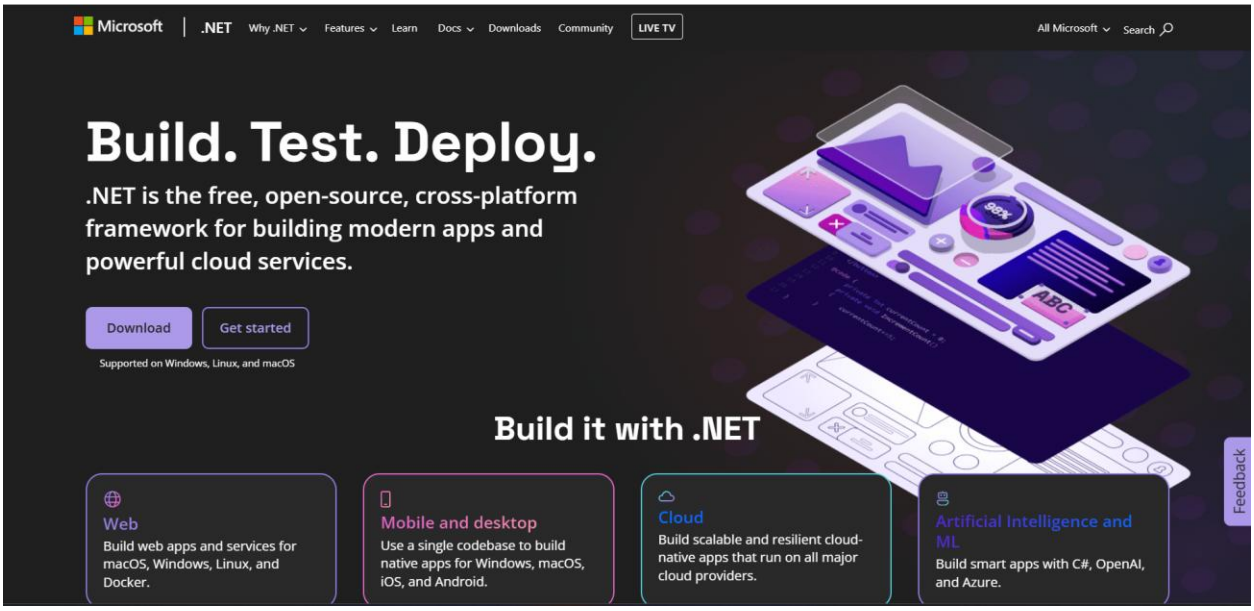
- Create a new project choosing NUnit Project



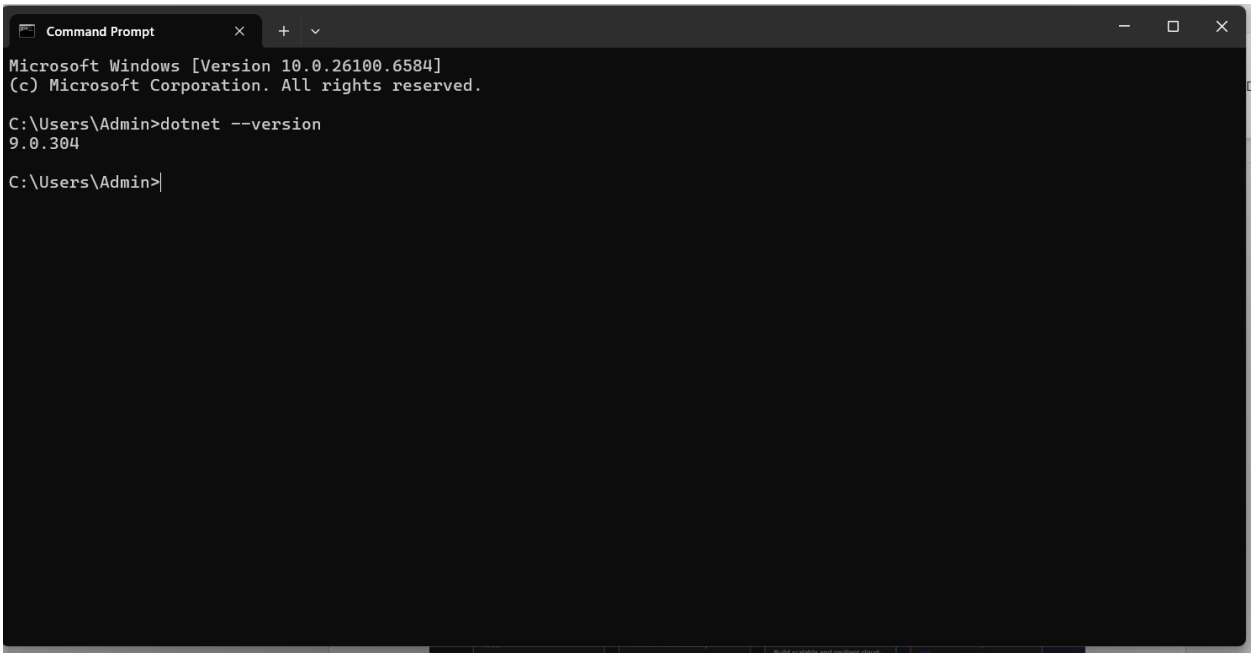
- In framework, I choose .NET 8 but it depends on the time that you are using and also your needs.



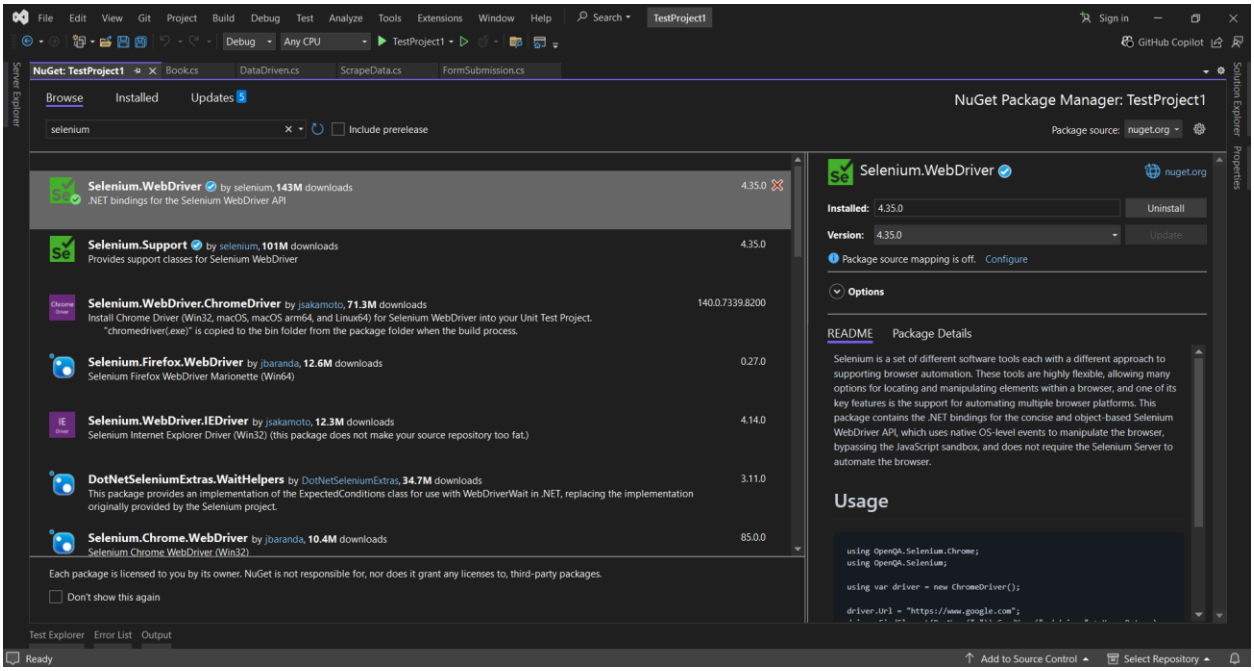
- However, you have to install .NET in computer through https://dotnet.microsoft.com/en-us/



- Make sure you got .NET by “dotnet --version”



- Back to Visual Studio Code, go to “Manage Nuget Package” (can be found with the search bar), search for Selenium WebDriver and then install it.



4. Rewrite an Existing Script

The instructor was teaching Javascript Selenium so I just rewrite the code from him different in some parts. Based on the tasks, I writed these code below:

- Automate a form submission

```
[Test]
0 references
public void Test()
{
    driver.Navigate().GoToUrl("https://scholar.google.com/?hl=vi");
    TestContext.WriteLine("This test gonna form submission from GG Scholar");

    IWebElement searchBox = driver.FindElement(By.Name("q"));
    searchBox.SendKeys("GNN" + Keys.Enter);
    Thread.Sleep(3000);

    IWebElement yearGap = driver.FindElement(By.CssSelector("#gs_res_sb_yyl > li:nth-child(4) > a"));
    yearGap.Click();
    Thread.Sleep(3000);

    Assert.IsTrue(driver.Title.Contains("GNN"), "Page did not contain the search text");
}
```

- Scrape data form a website

```
[Test]
0 references
public void Test()
{
    driver.Navigate().GoToUrl("https://books.toscrape.com/");

    WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));

    int countPageNumber = 3;
    TestContext.WriteLine("Number of testing pages: " + countPageNumber);

    int n = 1;

    while (n!= countPageNumber+1)
    {
        wait.Until(drv => drv.FindElements(By.CssSelector(".product_pod")).Count > 0);

        IList<IWebElement> books = driver.FindElements(By.CssSelector(".product_pod"));

        foreach (var book in books)
        {
            string title = book.FindElement(By.CssSelector("h3 > a")).GetAttribute("title");
            string price = book.FindElement(By.CssSelector(".price_color")).Text;
            string availability = book.FindElement(By.CssSelector(".availability")).Text.Trim();

            string line = $"\"{title}\",\"{price}\",\"{availability}\"\n";
            File.AppendAllText(csvFile, line);

            TestContext.WriteLine(line);
        }

        IWebElement nextButton = driver.FindElement(By.CssSelector(".next > a"));
        nextButton.Click();
        n++;
    }

    TestContext.WriteLine($"Data saved to {csvFile}");
}
```

- Implement data-driven testing

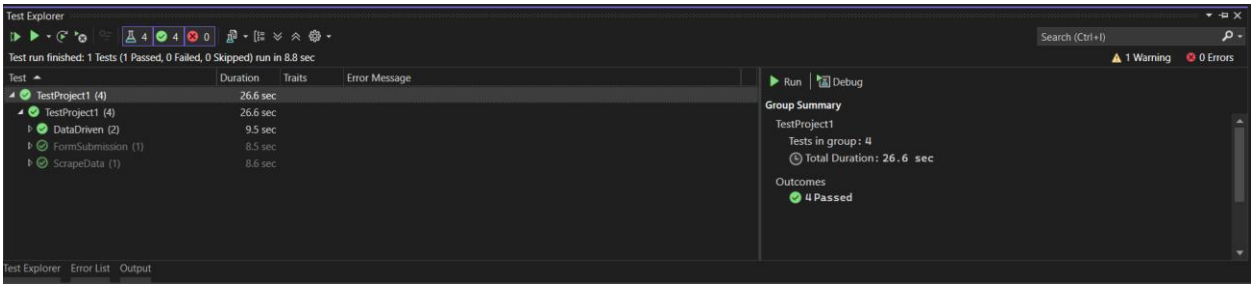
```
[Test]
0 references
public void Test()
{
    driver.Navigate().GoToUrl("https://books.toscrape.com/");

    // Open the CSV file once with StreamWriter
    using (var writer = new StreamWriter(csvFile, false)) // false = overwrite file
    {
        writer.WriteLine("Title,SavedPrice,CurrentPrice");

        WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));
        int n = 1;
        int countPageNumber = 3;

        List<Book> bookData = getBookDataFromFile("BooksData.csv");
        TestContext.WriteLine("Length of book list: " + bookData.Count);
        int lengthOfBookData = 0;

        while (n != countPageNumber + 1)
        {
            wait.Until(drv => drv.FindElements(By.CssSelector(".product_pod")).Count > 0);
            IList<IWebElement> books = driver.FindElements(By.CssSelector(".product_pod"));

            foreach (var book in books)
            {
                string title = book.FindElement(By.CssSelector("h3 > a")).GetAttribute("title");
                string price = book.FindElement(By.CssSelector(".price_color")).Text;

                if (lengthOfBookData < bookData.Count)
                {
                    Book savedBook = bookData[lengthOfBookData];

                    if (savedBook.IsDifferentPrice(price) && savedBook.isTheSameTitle(title))
                    {
                        string line = $"\"{title}\",\"{savedBook.Price}\",\"{price}\"";
                        TestContext.WriteLine(line);
                        writer.WriteLine(line);
                    }
                }
                else
                {
                    TestContext.WriteLine($"No saved book found for index {lengthOfBookData}");
                }

                lengthOfBookData++;
            }

            IWebElement nextButton = driver.FindElement(By.CssSelector(".next > a"));
            nextButton.Click();
            n++;
        }
    }
}
```

- For more details, please visit:

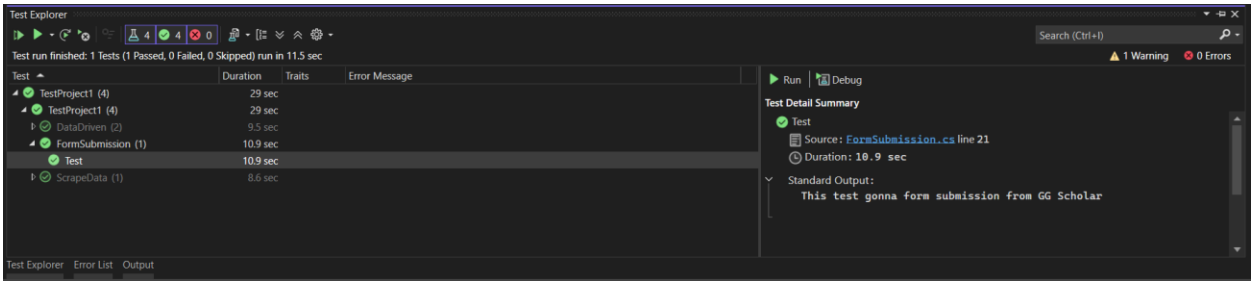
https://github.com/anhleh33/SeleniumDemo.git

5. Run and debug the Script

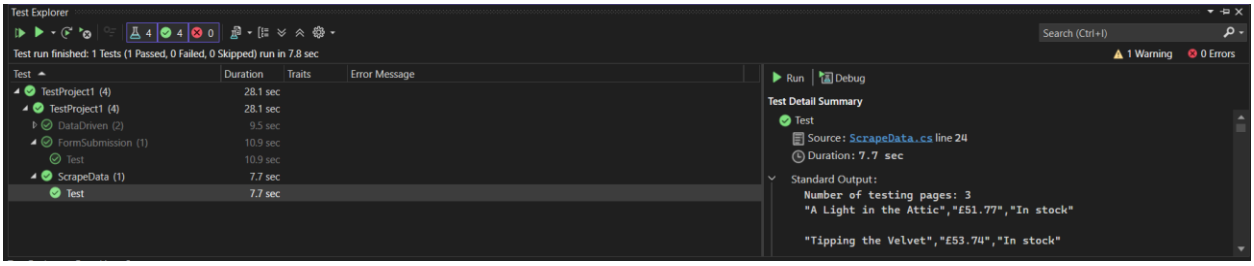
My Test Explorer:



- Automate a form submission



- Scrape data form a website



- Implement data-driven testing

III. Conclusion

1. Result

Through this project, I successfully demonstrated Selenium's cross-browser and cross-language flexibility by rewriting automation scripts in C# and running them on Chrome instead of the original setup. The scripts were able to perform form submissions, extract data from websites, and execute data-driven tests within the Visual Studio environment using NUnit. This confirmed my ability to adapt Selenium tests to different environments while maintaining their core functionality.

Overall, this project not only reinforced my understanding of Selenium's core concepts but also gave me valuable hands-on experience in adapting automation workflows to new programming languages and browser environments.

2. Challenges

During the process, I faced challenges such as configuring the .NET environment, handling differences in syntax between JavaScript and C#, and resolving compatibility issues between Selenium WebDriver, ChromeDriver, and .NET. Debugging in a new programming language also requires extra time to understand error messages and best practices.

3. Enhancements

To further improve this project, I could:

- Expand the test suite to cover more complex user interactions such as file uploads, dynamic elements, and multi-tab handling.
- Integrate the tests with a Continuous Integration (CI) pipeline like Jenkins or GitHub Actions for automated execution.
- Explore cross-platform testing with other browsers (Firefox, Edge, Safari) to maximize Selenium's cross-browser capabilities.
- Apply parallel test execution to reduce run time and increase efficiency.