Student Guide: C# Selenium Form Submission Project

Introduction

This guide will walk you through creating a C# console application that uses Selenium WebDriver to interact with a login form on https://the-internet.herokuapp.com/login, submit credentials, and handle the response.

Prerequisites

- Visual Studio 2022 (Community, Professional, or Enterprise edition)
- .NET SDK (6.0 or later recommended)
- Internet connection

Step 1: Installing Visual Studio 2022 (if not already installed)

1. Download Visual Studio 2022 from Microsoft's official website
2. Run the installer and select the ".NET desktop development" workload
3. Complete the installation process

Step 2: Creating a New Console Application

Using Visual Studio UI

1. Open Visual Studio 2022
2. Click on "Create a new project"
3. Search for "Console App" and select "Console App (.NET Core)" or "Console App (.NET)" with C# language
4. Click "Next"
5. Name your project "FormSubmissionDemo"
6. Choose a location to save your project
7. Select your preferred .NET version (recommend .NET 6.0 or later)
8. Click "Create"

Using CLI (Command Line Interface)

1. Open Command Prompt or PowerShell
2. Navigate to your desired project directory:

```
cd C:\Path\To\Your\Projects\Folder
```

3. Create a new console application:

```
dotnet new console -n FormSubmissionDemo
```

4. Navigate to the new project directory:

```
cd FormSubmissionDemo
```

Step 3: Installing Required NuGet Packages

Using Visual Studio UI

1. Right-click on your project in Solution Explorer
2. Select "Manage NuGet Packages"
3. Click on the "Browse" tab
4. Search for and install the following packages:
   - `Selenium.WebDriver`
   - `Selenium.Support`
   - `WebDriverManager` (for automatic driver management)

Using CLI

1. In the project directory, run the following commands:

```
dotnet add package Selenium.WebDriver
dotnet add package Selenium.Support
dotnet add package WebDriverManager
```

Step 4: Setting Up the Project Structure

Replace the contents of `Program.cs` with the following code:

```
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using WebDriverManager;
using WebDriverManager.DriverConfigs.Impl;
using System;
using System.Threading;

namespace FormSubmissionDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Starting Selenium WebDriver...");

            // Setup ChromeDriver using WebDriverManager
            new DriverManager().SetUpDriver(new ChromeConfig());

            // Initialize Chrome browser
```

```
            IWebDriver driver = new ChromeDriver();

            try
            {
                // Maximize browser window
                driver.Manage().Window.Maximize();

                // Navigate to the login page
                Console.WriteLine("Navigating to the login page...");
                driver.Navigate().GoToUrl("https://the-
internet.herokuapp.com/login");

                // Wait for page to load
                Thread.Sleep(1000);

                Console.WriteLine("Page title: " + driver.Title);

                // Find username and password fields
                IWebElement usernameField = driver.FindElement(By.Id("username"));
                IWebElement passwordField = driver.FindElement(By.Id("password"));
                IWebElement loginButton =
driver.FindElement(By.CssSelector("button[type='submit']"));

                // Enter credentials (provided by the website)
                usernameField.SendKeys("tomsmith");
                passwordField.SendKeys("SuperSecretPassword!");

                Console.WriteLine("Submitting the login form...");

                // Submit the form
                loginButton.Click();

                // Wait for redirection
                Thread.Sleep(2000);

                // Check if login was successful
                // The page URL changes after successful login
                if (driver.Url.Contains("/secure"))
                {
                    Console.WriteLine("\nLogin Successful!");

                    // Get the success message
                    IWebElement flashMessage = driver.FindElement(By.Id("flash"));
                    Console.WriteLine("Message: " + flashMessage.Text);

                    // Find and click the logout button
                    IWebElement logoutButton =
driver.FindElement(By.CssSelector(".button.secondary"));
                    Console.WriteLine("\nClicking logout button...");
                    logoutButton.Click();

                    // Wait for redirection
                    Thread.Sleep(1000);
```

```
                    Console.WriteLine("Returned to login page: " +
driver.Url.Contains("/login"));
                }
                else
                {
                    Console.WriteLine("\nLogin Failed!");

                    // Get the error message
                    IWebElement flashMessage = driver.FindElement(By.Id("flash"));
                    Console.WriteLine("Error Message: " + flashMessage.Text);
                }

                // Now let's try with invalid credentials
                Console.WriteLine("\nTrying with invalid credentials...");
                driver.Navigate().GoToUrl("https://the-
internet.herokuapp.com/login");

                // Find username and password fields again after page refresh
                usernameField = driver.FindElement(By.Id("username"));
                passwordField = driver.FindElement(By.Id("password"));
                loginButton =
driver.FindElement(By.CssSelector("button[type='submit']"));

                // Enter invalid credentials
                usernameField.SendKeys("invaliduser");
                passwordField.SendKeys("wrongpassword");

                // Submit the form
                loginButton.Click();

                // Wait for response
                Thread.Sleep(1000);

                // Get the error message
                IWebElement errorMessage = driver.FindElement(By.Id("flash"));
                Console.WriteLine("Error Message: " + errorMessage.Text);

                Console.WriteLine("\nPress any key to exit...");
                Console.ReadKey();
            }
            catch (Exception ex)
            {
                Console.WriteLine("An error occurred: " + ex.Message);
            }
            finally
            {
                // Close the browser
                driver.Quit();
            }
        }
    }
}
```

Step 5: Building and Running the Application

Using Visual Studio UI

1. Press F5 or click the "Start" button (green play button) to build and run your application
2. A Chrome browser window will open, navigate to the login page, and perform the form submission steps

Using CLI

1. In the project directory, run:

```
dotnet build
dotnet run
```

Step 6: Understanding the Code

Key Components

1. **WebDriver Setup:**

```
new DriverManager().SetUpDriver(new ChromeConfig());
IWebDriver driver = new ChromeDriver();
```

This initializes the Chrome WebDriver using WebDriverManager to automatically download the correct driver version.

2. **Navigation:**

```
driver.Navigate().GoToUrl("https://the-internet.herokuapp.com/login");
```

This commands the browser to navigate to the specified URL.

3. **Finding Form Elements:**

```
IWebElement usernameField = driver.FindElement(By.Id("username"));
IWebElement passwordField = driver.FindElement(By.Id("password"));
IWebElement loginButton =
driver.FindElement(By.CssSelector("button[type='submit']"));
```

These lines locate the form elements using their IDs and CSS selectors.

4. **Interacting with Form Elements:**

```
// Enter credentials
usernameField.SendKeys("tomsmith");
passwordField.SendKeys("SuperSecretPassword!");

// Submit the form
loginButton.Click();
```

The code enters text into the form fields and clicks the submit button.

5. **Verifying Results:**

```
if (driver.Url.Contains("/secure"))
{
    Console.WriteLine("\nLogin Successful!");

    // Get the success message
    IWebElement flashMessage = driver.FindElement(By.Id("flash"));
    Console.WriteLine("Message: " + flashMessage.Text);
}
```

After submission, the code checks the URL and message to verify the login result.

Step 7: Advanced Form Handling Techniques

1. Handling Select Dropdowns

```
using OpenQA.Selenium.Support.UI;

// Find the dropdown element
IWebElement dropdown = driver.FindElement(By.Id("dropdown-id"));

// Create a Select object
SelectElement selectElement = new SelectElement(dropdown);

// Select by visible text
selectElement.SelectByText("Option Text");

// Or select by value
selectElement.SelectByValue("value");

// Or select by index
selectElement.SelectByIndex(1);
```

2. Handling Checkboxes and Radio Buttons

```
// Find the checkbox
IWebElement checkbox = driver.FindElement(By.Id("checkbox-id"));

// Check if it's selected
bool isSelected = checkbox.Selected;

// Click to toggle
if (!isSelected)
{
    checkbox.Click();
}
```

3. Working with Multiple Forms

```
// Find a specific form
IWebElement form = driver.FindElement(By.Id("form-id"));

// Find elements within this form
IWebElement inputField = form.FindElement(By.Name("username"));

// Submit the specific form
form.Submit();
```

Step 8: Adding Explicit Waits (Recommended Enhancement)

To make your code more robust, replace the `Thread.Sleep()` with explicit waits:

```
using OpenQA.Selenium.Support.UI;

// Add this after submitting the form
WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));

// Wait for URL to change
wait.Until(d => d.Url.Contains("/secure"));

// Or wait for an element to be visible
wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.ElementIsVisible(By.Id("f
lash")));
```

To use the above, add the following NuGet package:

```
dotnet add package DotNetSeleniumExtras.WaitHelpers
```

Step 9: Form Validation Handling

Handling Client-Side Validation

```
// Submit an empty form to trigger validation
loginButton.Click();

// Find validation messages
IReadOnlyCollection<IWebElement> validationMessages =
driver.FindElements(By.CssSelector(".error-message"));

foreach (var message in validationMessages)
{
    Console.WriteLine("Validation error: " + message.Text);
}
```

Handling CAPTCHA (Conceptual - requires manual intervention)

```
// Find the CAPTCHA element
IWebElement captchaImage = driver.FindElement(By.Id("captcha-image"));

// Display image to user
Console.WriteLine("Please solve the CAPTCHA displayed in the browser");
Console.Write("Enter CAPTCHA solution: ");
string captchaSolution = Console.ReadLine();

// Enter the CAPTCHA solution
IWebElement captchaInput = driver.FindElement(By.Id("captcha-input"));
captchaInput.SendKeys(captchaSolution);
```

Troubleshooting

1. **Element Not Found Exceptions:**
    - Use try-catch blocks to handle cases when elements can't be found
    - Implement explicit waits to ensure elements are loaded before accessing
    - Check if selectors are correct using browser developer tools
2. **Form Submission Issues:**
    - Verify that all required fields are filled
    - Check for any client-side validation triggering
    - Ensure the form is actually submitting (watch network activity in browser dev tools)
3. **StaleElementReferenceException:**
    - This occurs when an element becomes detached from the DOM
    - Re-locate elements after page navigation or DOM changes

```
try {
    element.Click();
} catch (StaleElementReferenceException) {
    // Re-locate the element
    element = driver.FindElement(By.Id("element-id"));
    element.Click();
}
```

Conclusion

You have now created a C# console application using Selenium WebDriver to interact with a web form. This project demonstrates how to:

1. Navigate to a login page
2. Locate form elements
3. Input data into form fields
4. Submit the form
5. Handle the response (success or failure)
6. Process feedback messages

This knowledge forms the foundation for automating any web form interaction, including registration forms, search forms, contact forms, and more.

Next Steps

1. Try automating a more complex form with different input types (dropdowns, checkboxes, etc.)
2. Implement proper explicit waits throughout the code
3. Create a reusable framework for form testing
4. Implement data-driven testing with multiple test credentials
5. Add screenshot capture for failed login attempts