

Testing using Selenium WebDriver and NUnit in C#

- [Overview](#)
- [Description](#)
- [Tips](#)
- [References](#)

Overview

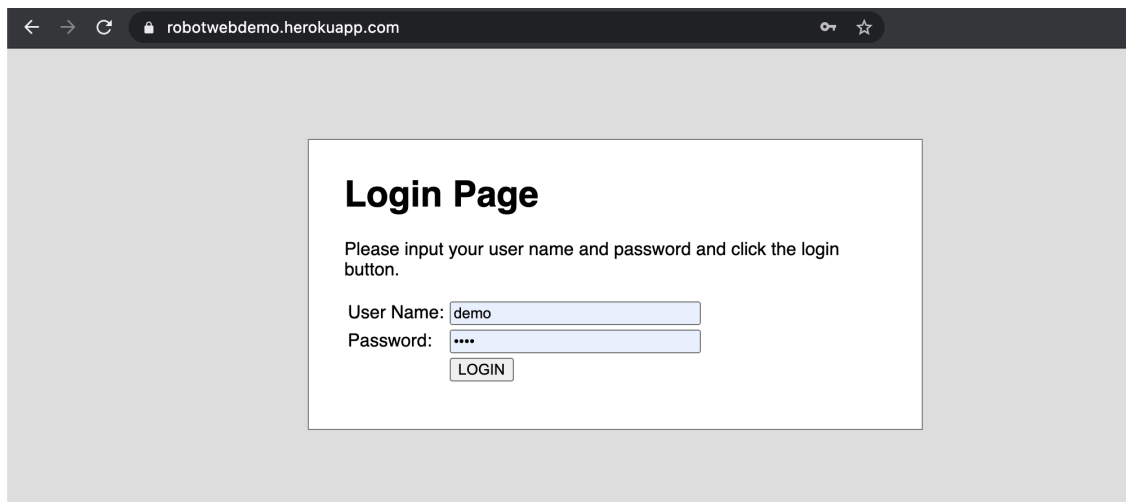
In this tutorial, we will create some UI tests using NUnit and Selenium WebDriver for browser automation.

Source-code for this tutorial

Code is available in [GitHub](#); the repo contains some additional tests beyond the scope of this tutorial and some auxiliary scripts.

Description

Our target application is a simple website providing a login page that we aim to test using positive and negative test scenarios.



We start by creating a #C .NET project, with NUnit and Selenium WebDriver dependencies.

nunit_webdriver_tests.csproj

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>

    <IsPackable>false</IsPackable>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="DotNetSeleniumExtras.PageObjects" Version="3.11.0" />
    <PackageReference Include="DotNetSeleniumExtras.PageObjects.Core" Version="3.12.0" />
    <PackageReference Include="NUnit" Version="3.13.1" />
    <PackageReference Include="NUnit.Console" Version="3.12.0" />
    <PackageReference Include="NUnit3TestAdapter" Version="3.16.1">
      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.NET.Test.Sdk" Version="16.5.0" />
    <PackageReference Include="Selenium.WebDriver" Version="3.141.0" />
  </ItemGroup>

</Project>
```

Before implementing the tests, we need to choose an approach for abstracting our website.

Interaction with the login page is abstracted using the page objects model. There's a object for the login page itself and another for the page containing the result.

Tests **may** be annotated with the "Requirement" property, if you wish to link the corresponding Test issue to an existing requirement/story issue in Jira.

The following code snippet shows two test scenarios: one for a successful login and another for an invalid login attempt due to incorrect credentials.

WebdemoTests.cs

```
using System;
using NUnit.Framework;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using Webdemo.PageObjects;

namespace SeleniumWebdriver
{
    [TestFixture]
    public class WebdemoTests
    {
        private IWebDriver driver;

        [SetUp]
        public void SetupTest()
        {
            ChromeOptions options = new ChromeOptions();
            options.AddArgument("--no-sandbox"); // Bypass OS security model, to run in Docker
            options.AddArgument("--headless");
            driver = new ChromeDriver(options);
        }

        [TearDown]
        public void TeardownTest()
        {
            try
            {
                driver.Quit();
            }
            catch (Exception)
            {
                // Ignore errors if unable to close the browser
            }
        }

        [Test, Property("Requirement", "CALC-2")]
        public void ValidLogin()
        {
            LoginPage loginPage = new LoginPage(driver).Open();
            LoginResultsPage loginResultsPage = loginPage.Login("demo", "mode");
            Assert.AreEqual(loginResultsPage.Title, "Welcome Page");
            Assert.IsTrue(loginResultsPage.Contains("Login succeeded"));
        }

        [Test, Property("Requirement", "CALC-2")]
        public void InvalidLogin()
        {
            LoginPage loginPage = new LoginPage(driver).Open();
            LoginResultsPage loginResultsPage = loginPage.Login("demo", "invalid");
            Assert.AreEqual(loginResultsPage.Title, "Error Page");
            Assert.IsTrue(loginResultsPage.Contains("Login failed"));
        }
    }
}
```

Running the tests can be done using dotnet utility.

example of a Bash script to run the tests

```
dotnet test -s nunit.runsettings --filter WebdemoTests
```

We can specify a configuration file to fine-tune NUnit behaviour, such as the output directory for the automation results report.

nunit.runsettings

```
<?xml version="1.0" encoding="utf-8"?>
<RunSettings>
  <NUnit>
    <WorkDirectory>./TestResults</WorkDirectory>
    <TestOutputXml>./</TestOutputXml>
  </NUnit>
</RunSettings>
```

After successfully running the Test Case and generating the NUnit XML report (e.g., [nunit_webdriver_tests.xml](#)), it can be imported to Xray via a CI tool (e.g. [Jenkins](#)), or the REST API, or by using the **Import Execution Results** action within the Test Execution.

example of a Bash script to import results

```
#!/bin/bash
FILE="TestResults/nunit_webdriver_tests.xml"
PROJECT=BOOK
JIRA_BASEURL=https://jiraserver.example.com
JIRA_USERNAME=someuser
JIRA_PASSWORD=somepass

curl -H "Content-Type: multipart/form-data" -u $JIRA_USERNAME:$JIRA_PASSWORD -F "file=@$FILE" "$JIRA_BASEURL/rest/raven/2.0/import/execution/nunit?projectKey=$PROJECT"
```



CALC / CALC-14

Execution results - nunit_webdriver_tests.xml - [1621936829736]

Edit Comment Assign More To Do In Progress Done Admin

Details

Type: Test Execution Status: TO DO (View Workflow)
Priority: Trivial Resolution: Unresolved
Labels: None
Test Plan: None
Test Environments: None

Description

Execution results imported from external source

Tests

+ Add

Overall Execution Status

2 PASS

Total Tests: 2

Filter(s)



Show 100 entries

Columns

	Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Status	
	1	CALC-12	InvalidLogin	Generic	1	0	Xpand IT Admin	PASS	
	2	CALC-10	ValidLogin	Generic	1	0	Xpand IT Admin	PASS	

Showing 1 to 2 of 2 entries

First Previous 1 Next Last

A Test Execution issue will be created in this case. The first time results are imported, Test issues are autoprovisioned; one per each NUnit test; in subsequent imports, results (i.e. Test Runs) are created for the already existing Tests so that these are reused and not duplicated.

Each NUnit's test is mapped to a Generic Test in Jira, and the **Generic Test Definition** field contains the name of the namespace, class, and the method name that implements the test.

In the details of execution screen, we can see the Test Run, showing the overall result and also the duration of the Test.

CALC / Test Execution: CALC-14 / Test: CALC-10

ValidLogin

Export Test as Text

Return to Test Execution

Execute with Exploratory App

4 Previous

Execution Status

PASS

Started On: 25/May/21 10:00 AM

Finished On: 25/May/21 10:00 AM

Assignee: Xpand IT Admin

Executed By: Xpand IT Admin

Tests environments:

Version: -

Revision: -

Comment

Preview Comment

Execution Defects (0)

Create Defect

Create Sub-Task

Add Defects

Execution Evidence (0)

Add Evidence

Execution Details

Test Description

None

Test Issue Links (1)

tests

CALC-2 As a user, I can login the website

TO DO

Custom Fields

There are no Test Run Custom Fields defined.

Test Details

Test Type: Generic

Definition: SeleniumWebdriver.WebdemoTests.ValidLogin

Results

Context	Output	Duration	Status
Testcase 0-1004 - ValidLogin	-	795.851 ms	PASS

In this case, we can also see that the Test was linked automatically to the existing user story (i.e. CALC-2).

Therefore, in the Story issue screen we can track the impacts of the test results on the calculated coverage, which in this case shows our Story as being "OK" due to the passing tests.



CALC / CALC-2

As a user, I can login the website

[Edit](#) [Comment](#) [Assign](#) [More](#) [To Do](#) [In Progress](#) [Done](#) [Admin](#)

Details

Type: [Story](#) Status: **TO DO** ([View Workflow](#))
Priority: ☐ Trivial Resolution: **Unresolved**
Labels: **None**
Requirement Status: **OK**

Description

[Click to add description](#)

Test Coverage

[Create Test](#) [Create Sub-Test Execution](#) [+ Link](#)

TEST COVERAGE FOR THE FOLLOWING ANALYSIS SCOPE

Scope: Version; **Version:** None - latest execution; **Environment:** All Environments

OK

[Filter\(s\)](#)



Show **10** entries Columns

P	Status	Resolution	Key	Summary	Test Runs	Test Status
<input type="checkbox"/>	TO DO	<i>Unresolved</i>	CALC-10	ValidLogin		PASS
<input type="checkbox"/>	TO DO	<i>Unresolved</i>	CALC-12	InvalidLogin		PASS

Showing 1 to 2 of 2 entries

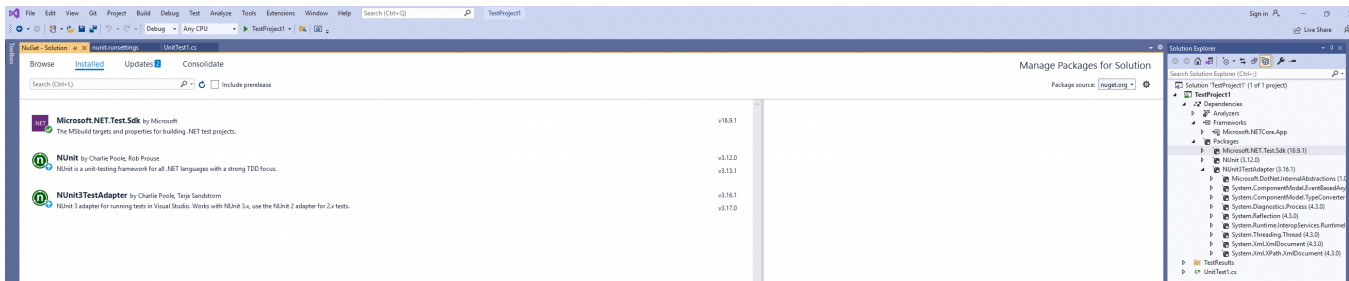
[First](#) [Previous](#) **1** [Next](#) [Last](#)

Tips

If you're using Visual Studio as your IDE, you need to have some dependencies/packages installed.

- NUnit
- NUnit3TestAdapter

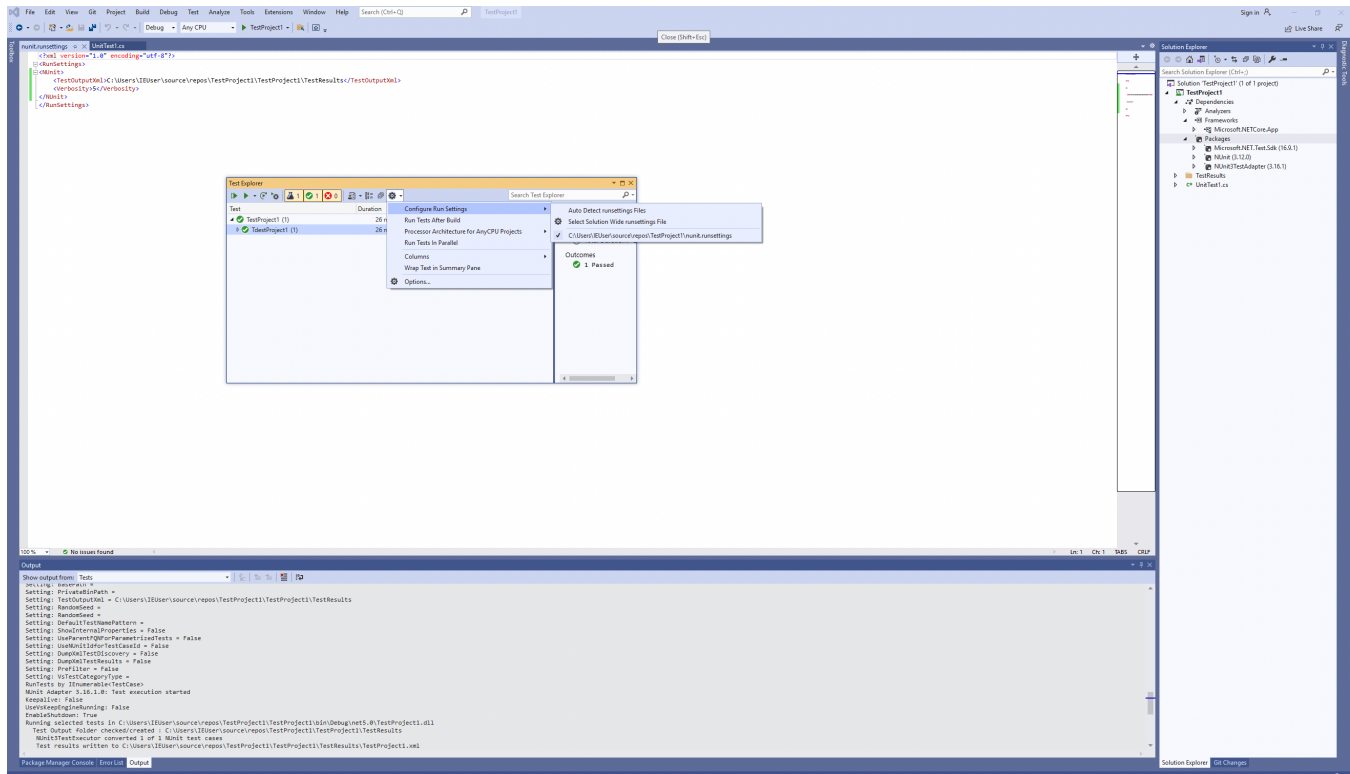
These can be installed from **Tools>NuGet Package Manager** (using the console or the manager's UI).



Then you can configure the Test Explorer to run the NUnit tests while at the same time producing a NUnit XML report.

nunit.runsettings

```
<?xml version="1.0" encoding="utf-8"?>
<RunSettings>
  <NUnit>
    <TestOutputXml>C:\TestResults</TestOutputXml>
  </NUnit>
</RunSettings>
```



References

- [GitHub repository for this tutorial](#)
- <https://github.com/nunit/docs/wiki>
- http://www.seleniumhq.org/docs/03_webdriver.jsp
- <http://www.dotnetcatch.com/2016/11/23/selenium-with-net-core/>
- <http://toolsqa.wpengine.com/selenium-webdriver/c-sharp/iwebdriver-browser-commands-in-c-sharp/>
- [Configure unit tests by using a .runsettings file](#)