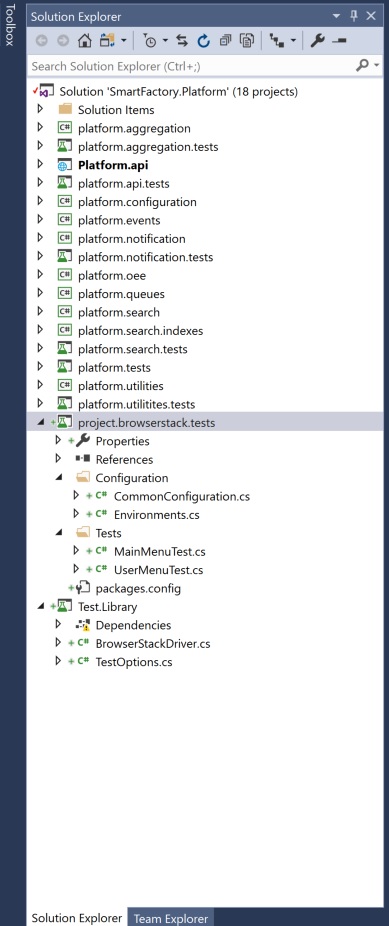
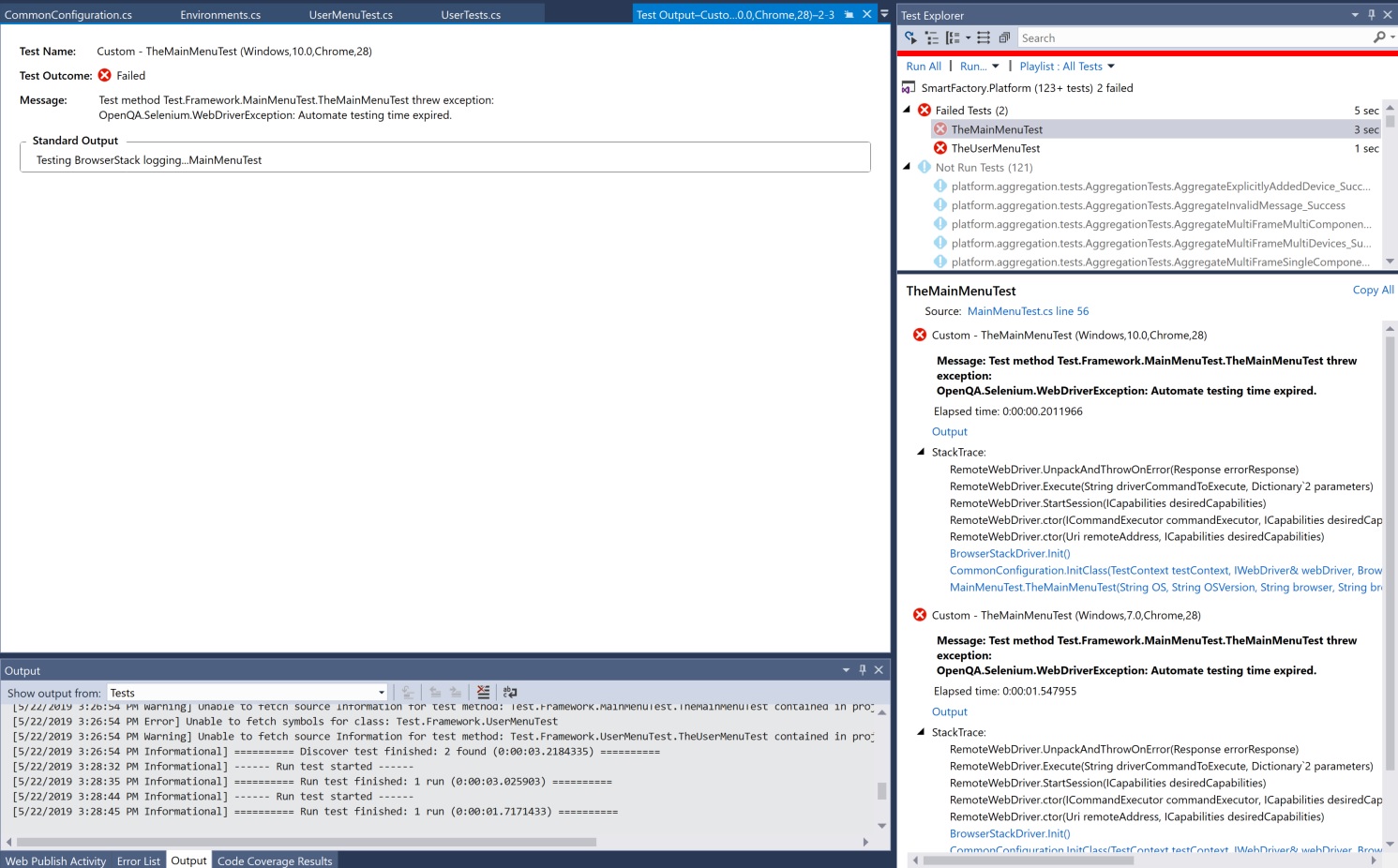
**Tips for Using the BrowserStack Testing Framework**

The BrowserStack testing framework has been integrated into an older version of the SmartFactory.platform codebase. This was done to complete the changes so that BrowserStack tests can be run alongside unit tests. You might choose to run them automatically with unit tests, but most likely once a full suite of tests has been developed it will be too slow to run all the time.

In the screenshot below, you can see that the projects project.browserstack.tests and Test.Library will eventually need to be merged with the latest version of the main codebase.



Tests can be run by right clicking the tests and selecting Run. Two examples are provided: TheMainMenuTest and TheUserMenuTest.



By highlighting TheMainMenuTest, we can see the details behind why the test failed. In this case, the reason is “Automate testing time expired.” This message tells us that we made it to the BrowserStack site, but our account was not allowed to continue. This is as far as we can go without a license.

Notice that the test ran twice: once for Windows 10 and once for Windows 7. These are the only two environments currently specified for testing.

Clicking the blue “Output” text causes the window shown in the left pane to appear. This is where Console.Writeline statements can be seen in the Visual Studio environment. In this case, the message “Testing BrowserStack Logging…MainMenuTest” is within the code. By placing these Console.Writeline statements in the environment with the rest of the codebase, it was planned that these statements would be viewable from DevOps with the other error messages. This provides an excellent opportunity to provide as much detail as possible on exactly why the test failed.

The InitClass function (under Configuration/CommonConfiguration) is where the connection with BrowserStack is setup:

public static void InitClass(TestContext testContext, out IWebDriver webDriver, out BrowserStackDriver browserStackDriver,

      String OS, String OSVersion, String browser, String browserVersion)

    {

      var options = new TestOptions

      {

        Os = OS,

        OsVersion = OSVersion,

        Browser = browser,

        BrowserVersion = browserVersion,

        BrowserStackUser = "billmelvin1",

        BrowserStackAccessKey = "xxxxxx",

        BrowserStackDebug = true,

        BrowserStackServer = "hub-cloud.browserstack.com"

      };

      \_bsDriver = new BrowserStackDriver(options);

      \_driver = \_bsDriver.Init();

      browserStackDriver = \_bsDriver;

      webDriver = \_driver;

    }

As you can see, this is connecting to the browser stack server at “hub-cloud.browserstack.com”.  The BrowserStackUser is “billmelvin1” which is why I have not checked it in.  With a licensed account, a “qa\_user” type of account with its own BrowserStackAccessKey should be possible.

Using this connection to the cloud, the sample script I gave has:

driver.Navigate().GoToUrl("[https://qa-mi-sfp-as-01.azurewebsites.net](https://eur02.safelinks.protection.outlook.com/?url=https%3A%2F%2Fqa-mi-sfp-as-01.azurewebsites.net&data=02%7C01%7C%7C686ffcbdc64c4684dd4108d6d77a56f8%7C1b16ab3eb8f64fe39f3e2db7fe549f6a%7C0%7C0%7C636933317789811716&sdata=tPlW6YpCzas0jZkZ%2Fkay39yCvnqPOau7xbXWI2ECKoo%3D&reserved=0)");

This tells BrowserStack to test against the web site at this URL. Any public facing instance can be specified here.

Another important area is Environments.cs (under Configuration). This is where the list of environments to be tested against is specified. Each test will be run against each environment contained in this file.

The two examples provided are very basic and are based on the output from a tool called Katalon (very similar to Selenium).

An important discovery was that some tests fail because a control is referenced before it has been loaded. So, some wait statements are needed in some areas to provide enough time. That is the intent of statements such as:

wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.ElementToBeClickable(

By.CssSelector(".svg-inline--fa:nth-child(3)")));

This code waits for “.svg-inline—fa:nth-child(3)” to be clickable before continuing. The wait command waits 10 seconds and the waitLonger command waits 60 seconds. Those are currently defined in the code and should be put someplace common as I suspect that most tests will need this technique.