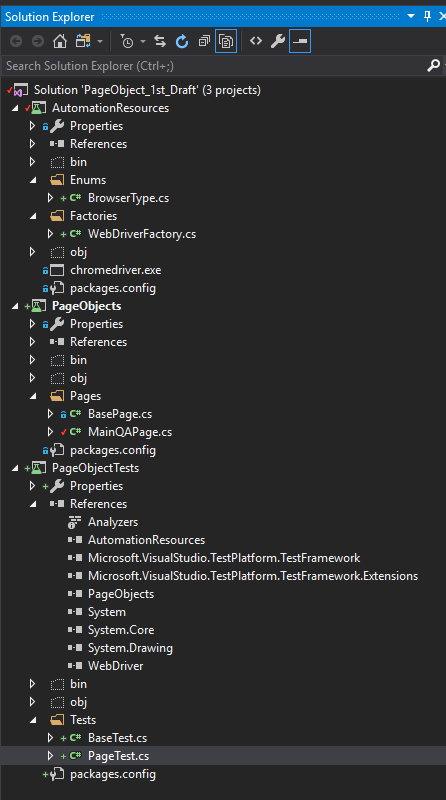
**Best Practices**

**Solution Layout**

***Projects:***

1. *Automation Resources*
   1. This project should contain items that are shared by all page objects
2. *Page Objects*
   1. This project should contain items that are related to a single “page” of an application
      1. In the case of a SPA (single page application), this could be broken up into “sections” (e.g. Left Nav, Map, Toolbox, etc)
3. *Page Object Tests*
   1. This project should only be concerned with running tests

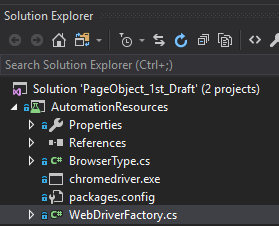
Solution Layout:



**Automation Resources**

The *only* place that contains the actual chrome driver itself.

Has single place where the web driver gets created.



Web Driver Factory example:

using System;

using System.IO;

using System.Reflection;

using AutomationResources.Enums;

using OpenQA.Selenium;

using OpenQA.Selenium.Chrome;

using OpenQA.Selenium.Edge;

using OpenQA.Selenium.Firefox;

namespace AutomationResources

{

public class WebDriverFactory

{

public IWebDriver Create(BrowserType browserType)

{

switch (browserType)

{

case BrowserType.Chrome:

return GetChromeDriver();

case BrowserType.FireFox:

return GetFireFoxDriver();

case BrowserType.Edge:

return GetEdgeDriver();

}

throw new NotImplementedException("No such browser currently exists!!!");

}

//--gets a string where the driver is located

//--example - "D:\\repositoryname\\solutionname\\projectname\\bin\\Debug"

private readonly string \_outputDirectory = Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);

private IWebDriver GetEdgeDriver()

{

return new EdgeDriver(\_outputDirectory);

}

private IWebDriver GetFireFoxDriver()

{

return new FirefoxDriver(\_outputDirectory);

}

private IWebDriver GetChromeDriver()

{

return new ChromeDriver(\_outputDirectory);

}

}

}

**Page Objects:**

The [Page Object Model](https://martinfowler.com/bliki/PageObject.html) is a common design pattern in software automation for websites. Essentially, it tries to decouple the testing of the page from the markup / layout of the page itself – so when the website’s style or layout changes, our test doesn’t break. This also means that when the website does change, we only have 1 place to change. If the specifics of the page were in our tests, then we could potentially have multiple things that needed to be changed.

*Base Page:*

using OpenQA.Selenium;

namespace PageObject\_1st\_Draft.Pages

{

public class BasePage

{

public BasePage(IWebDriver driver)

{

Driver = driver;

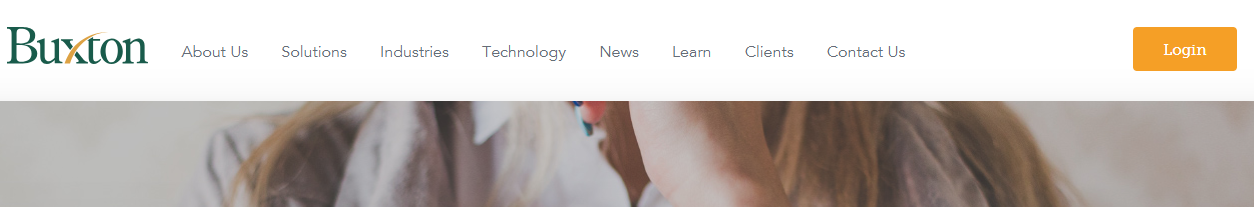
}

protected IWebDriver Driver { get; set; }

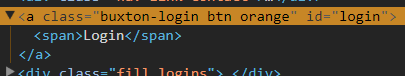
}

}

Using [www.buxtonco.com](http://www.buxtonco.com) as an example:



The login button has a class of orange:



Because the website can (and will) change it’s style and layout, we would never target this button based on the fact that it is orange (because if it changed to green, our test would no longer work) – in this case we’d use the id. But it’s more than this.

Since for the foreseeable future, our clients will get access to SCOUT via logging in through our website, we’d create a test to perform the following steps:

1. Navigate to the Buxtonco website
2. Click the Login button
3. Click the appropriate Platform
4. Proceed……………

*Child Page:*

The Buxtonco page would then take this form

using OpenQA.Selenium;

namespace PageObject\_1st\_Draft.Pages

{

public class Buxtonco : BasePage

{

public Buxtonco(IWebDriver driver) : base(driver)

{

}

public void GoTo()

{

Driver.Navigate().GoToUrl("https://www.buxtonco.com");

}

public void FindLoginButton\_AndClick()

{

Driver.FindElement(By.Id("login")).Click();

}

public void FindPlatformLink\_AndClick()

{

Driver.FindElement(By.Id("ap-login")).Click();

}

}

}

This means that if anything in the website changes, we change it in one place, here – on the Page Object instance. If we were finding elements and clicking on them from a test itself, then we’d potentially have multiple places where we’d need to make a change.

**Page Object Tests:**

*Base Test:*

The Base Test is what ALL child tests should inherit from.

This class must contain two attributes, and should be the only test class with these attributes:

* **[TestInitialize]**
  + Identifies the method run *before* a test
  + Allocates and configures resources needed by all tests in the **[TestClass]**
* **[TestCleanup]**
  + Identifies a method that contains code that must be used *after* the test has run
  + Frees resources obtained by all tests in the **[TestClass]**
  + Will run even if an exception is thrown

using AutomationResources.Enums;

using AutomationResources.Factories;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using OpenQA.Selenium;

namespace PageObject\_1st\_Draft.Tests

{

public class BaseTest

{

public IWebDriver BaseDriver { get; set; }

[TestInitialize]

public void Setup()

{

var factory = new WebDriverFactory();

BaseDriver = factory.Create(BrowserType.Chrome);

BaseDriver.Manage().Window.Maximize();

}

[TestCleanup]

public void TearDown()

{

BaseDriver.Quit();

}

}

}

*Child Test:*

Notice, in this test, we are not using the driver to find anything. That is abstracted out into methods found in the Page Object.

using Microsoft.VisualStudio.TestTools.UnitTesting;

using PageObject\_1st\_Draft.Pages;

namespace PageObject\_1st\_Draft.Tests

{

[TestClass]

[TestCategory("Where is this showing up")]

public class PageTest : BaseTest

{

[TestMethod]

// 1 - go to main qa site

// 2 - search for complicated page

// 3 - assert you're on the right page

public void TestMethod1()

{

var x = new MainQaPage(BaseDriver);

x.GoToMainPage();

x.ClickSignInLinkAndSearch("complicated page");

Assert.AreEqual(BaseDriver.Url, "https://www.ultimateqa.com/?s=complicated+page");

}

}

}