Testing Web Applications with Selenium POM using C# and NUnit

Office: Karposh bb (direkcija na “Granit”), 7000 Bitola, Macedonia.
Tel: +389 (0) 47 221 914, fax + 398 (0) 47 240 010
e-mail:interworks@interworks.com.mk

© Copyright InterWorks 2017.

Date: Jan, 2017

CONTENTS

1 INTRODUCTION 3

1.1 WHAT IS SELENIUM? 3
1.2 SELENIUM IDE 3
1.3 SELENIUM WEB DRIVER 3
1.4 SELENIUM GRID 3

2 BUILD SELENIUM TESTS USING NUNIT FRAMEWORK 5

2.1 SETTING UP THE PROJECT 5
2.2 BUILDING THE SELENIUM CODE 7
2.2.1 Setting up the main class 8
2.2.2 The PageActions class 10
2.2.3 The BasicMethods class 11
2.2.4 The Validations class 11
2.3 PAGE OBJECT MODEL (POM) 12
2.4 NUNIT TESTING FRAMEWORK 13
2.4.1 Assertions 13
2.4.2 Constraints 14
2.4.3 Attributes 14

Office: Karposh bb (direkcija na “Granit”), 7000 Bitola, Macedonia.
Tel: +389 (0) 47 221 914, fax + 398 (0) 47 240 010
e-mail:interworks@interworks.com.mk
© Copyright InterWorks 2017.

INTRODUCTION

1.1 What is Selenium?

Selenium is an open source suite that is used for web applications test automation. It allows recording or scripting automated test flows. It uses a test domain-specific language Selenese. Tests can be written in a number of popular programming languages, including C#, Java, PHP, Python, Ruby, Groovy and Scala. Automation is supported for multiple web browsers as well multiple platforms.

The Selenium suite consists of the following components: Selenium IDE, Selenium RC, Web Driver and Selenium Grid. The Selenium RC and Web driver have been merged into a single framework Selenium 2, which is widely accepted and used.

1.2 Selenium IDE

Selenium Integrated Development Environment (IDE) is the simplest framework in the Selenium suite and is the easiest one to learn. It is a Firefox plugin that can be installed and used for recording the flows inside web application. Because of its simplicity, it is used as prototyping tool.

1.3 Selenium Web Driver

Selenium WebDriver is used for creation of robust, browser-based regression automation suites and tests that can be run in different browsers by leveraging specific browser drivers for Chrome, Firefox and Internet Explorer. At highest level, Selenium WebDriver, simply, accepts programmed commands, sends them to a browser and interacts with the HTML elements on the page.

However, before driver can interact with the element, it need to be located (find) in the page. The WebDriver's method findElement() is used for that purpose. The driver can find elements on the web page using several types of locators:

- By Id
- By Class Name
- By Tag Name
- By Name
- By Link Text
- By CSS
- By XPath

These locators will search the web page's HTML and find particular element(s) based on the supplied Id, XPath, ClassName etc.
After driver, finds element, it is returned as WebElement object. WebElement objects, expose methods that simulate "human" interaction with the page. Methods that often used are: click(), sendKeys(), submit(), getText(), etc.

1.4 Selenium Grid

Selenium Grid is a server that allows to run tests on different machines against different browsers in parallel. The server acts as a hub, while all browser instances contact that hub.

Running the tests with Selenium Grid mainly gives the following advantages:

- Running the tests against different browsers, operating systems and machines at the same time. This will ensure that the application is fully compatible with range of browser/OS combinations

Office: Karposh bb (direkcija na "Granit"), 7000 Bitola, Macedonia.
Tel: +389 (0) 47 221 914, fax + 398 (0) 47 240 010
e-mail:interworks@interworks.com.mk
© Copyright InterWorks 2017.

- Saves time in the execution of test suites

Figure 1. Selenium Grid uses a hub-node concept where you only run the test on a single machine called a **hub**, but the execution will be done by different machines called **nodes**

Office: Karposh bb (direkcija na "Granit"), 7000 Bitola, Macedonia.
Tel: +389 (0) 47 221 914, fax + 398 (0) 47 240 010
e-mail:interworks@interworks.com.mk
© Copyright InterWorks 2017.

BUILD SELENIUM TESTS USING NUNIT FRAMEWORK

Note: In this tutorial we will build an example for Smoke Test on LinkedIn web application. The scenario will cover user log in, navigation to the edit profile page, elements validation on the profile page and log out.

1.5 Setting up the project

Microsoft Visual Studio, preferably the latest version, is required for building and running this test project.

First of all, the NUnit should be installed in Visual Studio. For that purpose, navigate to Tools -> Extensions and Updates and search for NUnit. Install the “NUnit 3 Test Adapter” and “Nunit Templates for Visual Studio”. After installing these extensions, you will be asked to restart the Visual Studio in order to make these extensions available for use.

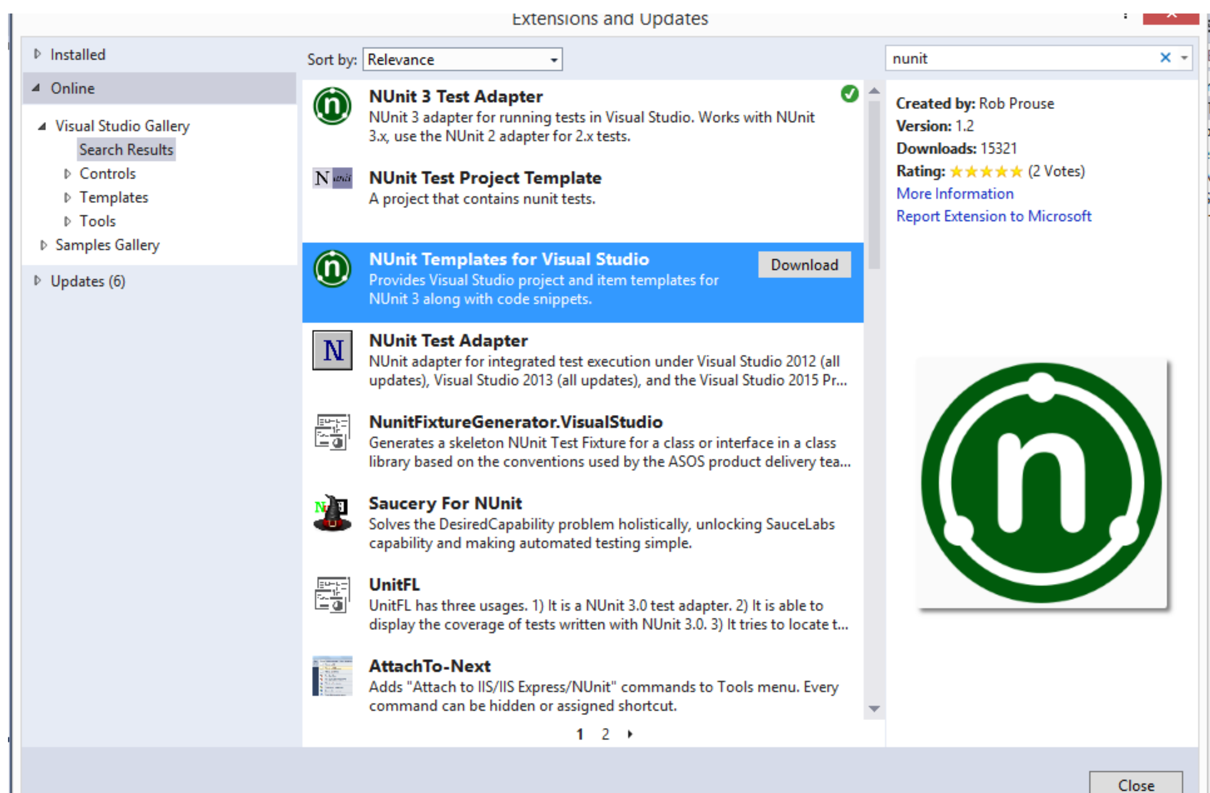


Figure 2. Installing NUnit in Microsoft Visual Studio

Next, you need to create the project. Click File->New->Project. On the left side of the pop up window choose Test and from the templates choose “NUnit 3 Unit Test Project”. Give a name to your project and location to be saved to.

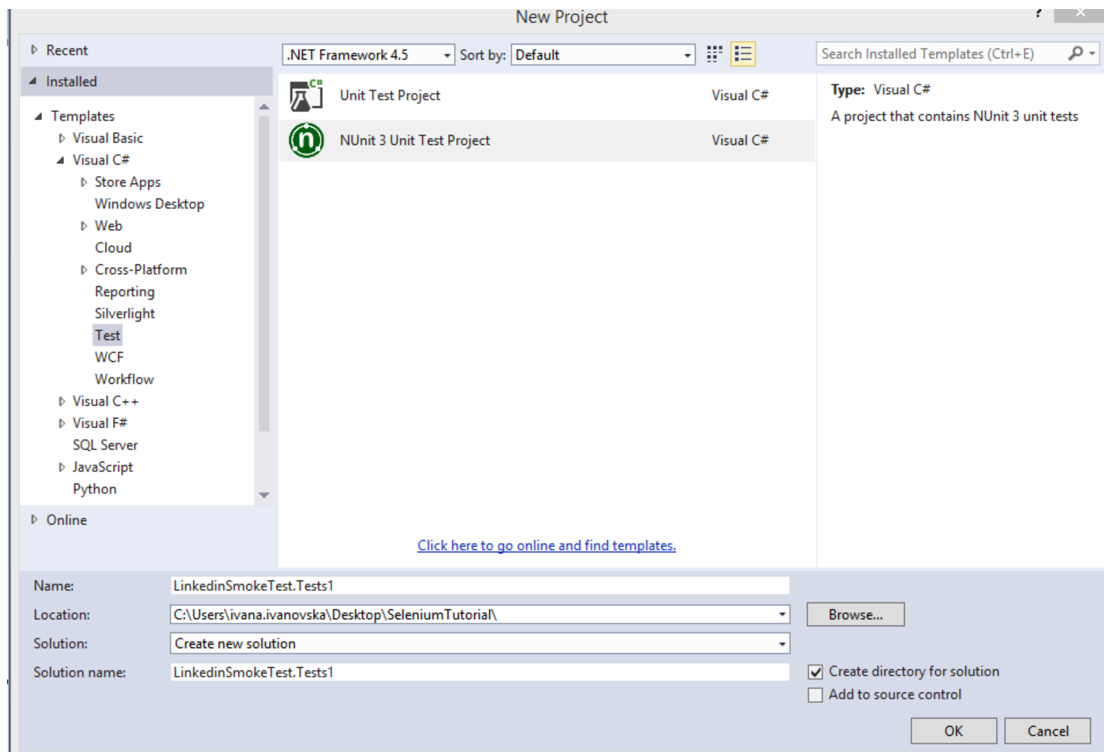


Figure 3. Create test project

In the newly created test Class, you will notice the [TextFixture] and [Test] annotations. Beside these, the annotations [SetUp] and [TearDown] are widely used in NUnit in order to mark some methods that are used as test case execution lifecycle callbacks.

Each test case execution includes the following phases:

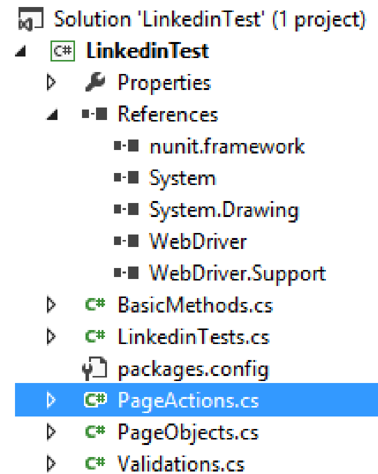
1. [SetUp] – The method under this annotation opens the browser. It executes before every test case
2. [Test] – Every test case should have this annotation in order to be executed
3. [TearDown] – The method under this annotation executes after each test case and it usually closes the browser, makes some changes in DB etc.

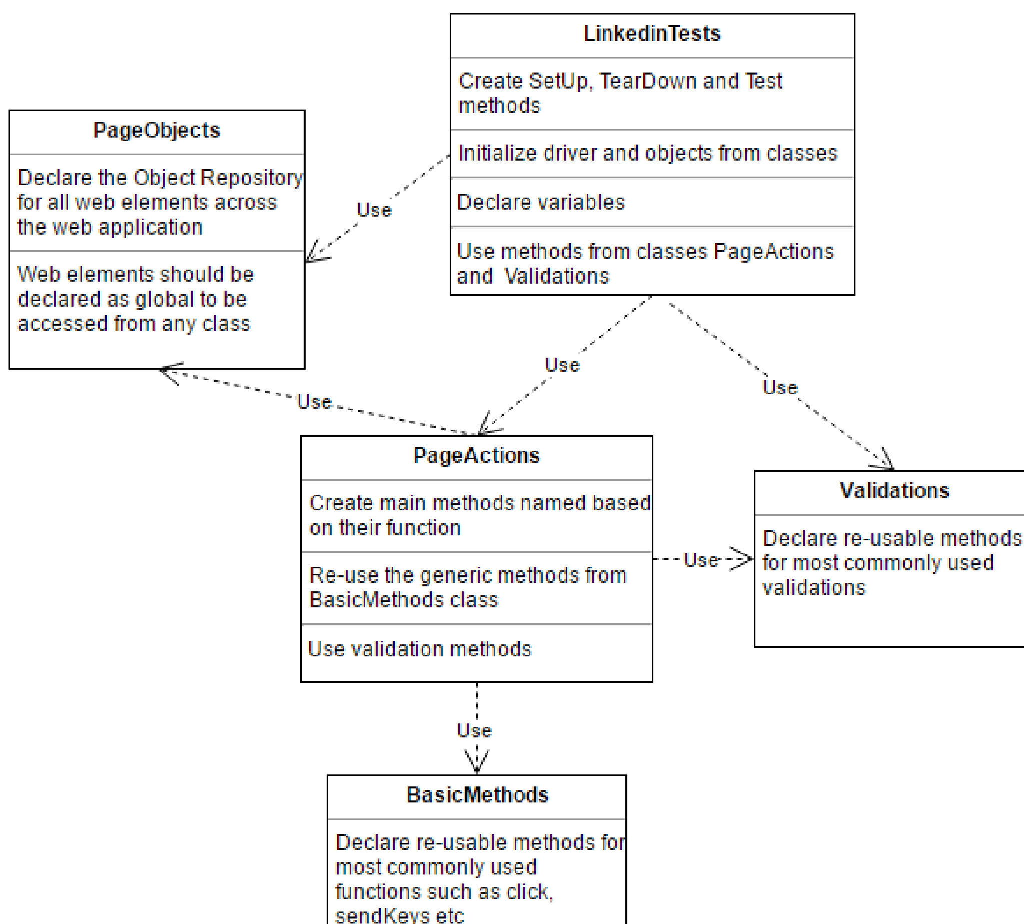
1.6 Building the Selenium Code

The final version of the automated Selenium project will consist of the following classes: LinkedinTests, PageActions, BasicMethods, Validations and PageObjects.

The separation of code when writing automated tests is a good practice because it provides easier and quicker updates of tests. In this example, all page specific methods are declared in one class (PageActions), all validations are defined in other class (Validations). The class BasicMethods contains the most common user actions that are not page specific. Finally, in the class LinkedinTests, all methods are reused and combined in several specific test scenarios.

The class PageObjects contains all hard-coded page elements values (names, ids etc.). So, if any change is made to the page HTML, updates need to be done only in the class that corresponds to that page, without modification in all tests where that page is used.





© Copyright InterWorks 2017.

Figure 5. Class Model

1.6.1 Setting up the main class

The main class LinkedinTests is the main entry point in the testing fixture and contains SetUp, TearDown, SmokeTest and RegressionTest methods.

```
1.  using LinkedinTest.Tests1;
2.  using NUnit.Framework;
3.  using OpenQA.Selenium;
4.  using OpenQA.Selenium.Chrome;
5.  using System;
6.
7.  namespace LinkedinTest
8.  {
9.      [TestFixture]
10.     public class LinkedinTests
11.     {
12.         static IWebDriver driver;
13.         PageActions pageActions = new PageActions();
14.         Validations validations = new Validations();
15.
16.         public String email = "testaccount@gmail.com"; // Replace with valid regi
    stered email on linkedin
17.         public String password = "pass"; // Replace with valid password used when
    registering
18.         public String username = "JohnWack"; // Replace with valid username used
    when registering
19.         public String fullName = "John Wack";
20.         public String locality = "Macedonia";
21.         public String linkedinURL = "https://www.linkedin.com/";
22.
23.         [SetUp]
24.         public void SetUp()
25.         {
26.             driver = new ChromeDriver();
27.             driver.Manage().Timeouts().ImplicitlyWait(TimeSpan.FromSeconds(30));

28.         }
29.
30.         [Test]
31.         public void SmokeTest()
32.         {
33.             pageActions.navigateToLinkedin(driver, linkedinURL);
34.             pageActions.Login(driver, email, password);
35.             pageActions.navigateThroughMenu(driver, PageObjects.ProfileLinkMenu,
    PageObjects.editProfileLink);
36.             validations.validateElementIsPresent(driver, PageObjects.addPhotoElem
    ent);
37.             validations.validateTextInElement(driver, PageObjects.fullNameElement
    , fullName);
38.             validations.validateTextInElement(driver, PageObjects.localityElement
    , locality);
39.             pageActions.Logout(driver);
40.         }
41.
42.         [Test]
43.         [Ignore("Ignore a test")]
```

Office: Karposh bb (direkcija na “Granit”), 7000 Bitola, Macedonia.
Tel: +389 (0) 47 221 914, fax + 398 (0) 47 240 010
e-mail:interworks@interworks.com.mk
© Copyright InterWorks 2017.

```
44.         public void RegressionTest()
45.         {
46.
47.         }
48.
49.         [TearDown]
50.         public void TearDown()
51.         {
52.             driver.Quit();
53.         }
54.     }
55. }
```

In the **SetUp** method, the driver is initialized as new **ChromeDriver**. In order to use the ChromeDriver, it must be installed first and added as a reference in the project. For that purpose, click on References -> Manage NuGet Packages. Search for “WebDriver” and “Selenium.WebDriver.ChromeDriver”. Install these packages to the project.

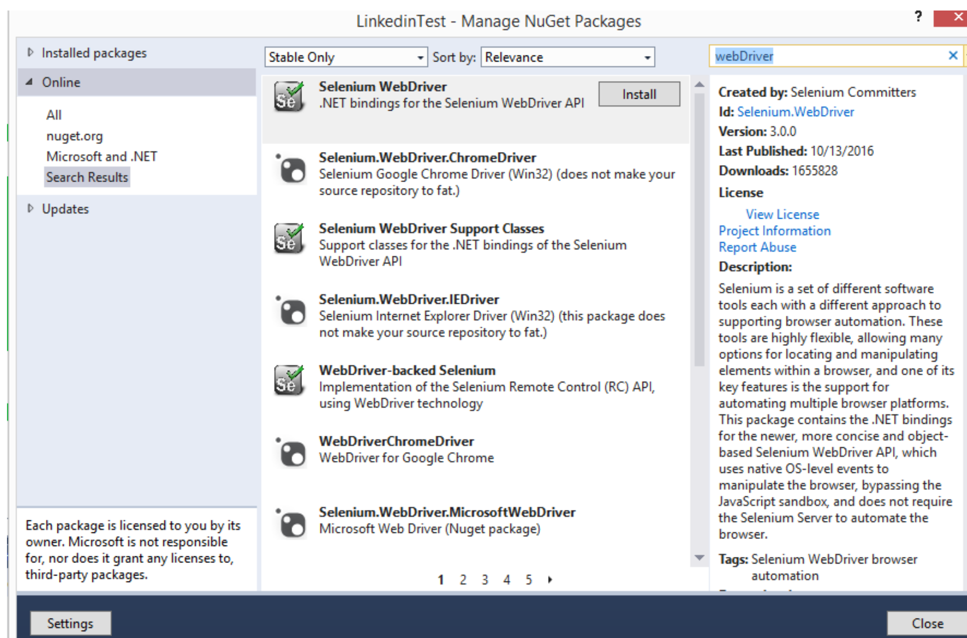


Figure 6: Installing WebDriver

In order Chrome driver to be used, add reference: `using OpenQA.Selenium.Chrome;` and with this setup, the Chrome browser will be opened at the beginning of each test. Please note that after driver initialization, is added wait period of 30 seconds in order to anticipate some delay if the browser can’t open immediately.

The method **SmokeTest** method is calling other methods from classes *PageActions* and *Validations*. Since the used methods are created to be re-usable, specific details are sent as parameters. For example to Login, email and password used in the test, are sent as parameters to the `pageActions.Login` method. Please also note, that public variables from *PageObjects* class, for example **`PageObjects.fullNameElement`**, are sent as parameters. The `fullNameElement` is declared as global variable in class PageObjects, and defines path to particular page object.

© Copyright InterWorks 2017.

Since the scope of this tutorial is smoke test, only SmokeTest method is defined. The execution of the method **RegressionTest** is skipped, by annotating it with the "[Ignore("Ignore a test")]".

Finally, the **TearDown** method just closes the browser at the end of each test.

1.6.2 The PageActions class

The PageActions class, defines methods that "mimics" user interactions with the web site under test. For example, method `navigateToLinkedin` simulates navigating to LinkedIn home page, while Login simulates logging into LinkedIn, navigating to home page and validating existence of particular elements (News Feed). This class can be extended with methods for additional user actions, like ConnectToFriend, EndorseFriend, LikePost etc.

```
1.  using OpenQA.Selenium;
2.  using OpenQA.Selenium.Interactions;
3.  using System;
4.
5.  namespace LinkedinTest.Tests1
6.  {
7.      public class PageActions
8.      {
9.          BasicMethods basicMethod = new BasicMethods();
10.         Validations validations = new Validations();
11.
12.         public void navigateToLinkedin(IWebDriver driver, String linkedinURL)
13.         {
14.             driver.Navigate().GoToUrl(linkedinURL);
15.             validations.validateScreenByUrl(driver, linkedinURL);
16.             validations.validateElementIsPresent(driver, PageObjects.forgotPassLi
    nk);
17.         }
18.
19.         public void Login(IWebDriver driver, String email, String password)
20.         {
21.             basicMethod.clickElement(driver, PageObjects.loginEmailInput);
22.             basicMethod.sendKeys(driver, PageObjects.loginEmailInput, email);
23.             basicMethod.clickElement(driver, PageObjects.loginPassInput);
24.             basicMethod.sendKeys(driver, PageObjects.loginPassInput, password);
25.             basicMethod.clickElement(driver, PageObjects.loginButton);
26.             // Wait up to 30s for the user to login and to be redirected to Home
    screen
27.             driver.Manage().Timeouts().ImplicitlyWait(TimeSpan.FromSeconds(30));

28.             validations.validateElementIsPresent(driver, PageObjects.newsFeedElem
    ent);
29.         }
30.
31.         public void navigateThroughMenu(IWebDriver driver, By menu, By submenu)
32.         {
33.             Actions builder = new Actions(driver);
34.             builder.MoveToElement(driver.FindElement(menu)).Perform();
35.             builder.MoveToElement(driver.FindElement(submenu)).Click().Perform();

36.         }
37.
38.         public void Logout(IWebDriver driver)
```

Office: Karposh bb (direkcija na "Granit"), 7000 Bitola, Macedonia.
Tel: +389 (0) 47 221 914, fax + 398 (0) 47 240 010
e-mail:interworks@interworks.com.mk
© Copyright InterWorks 2017.

```
39.         {
40.             Actions builder = new Actions(driver);
41.             builder.MoveToElement(driver.FindElement(PageObjects.navProfilePhoto)
    ).Perform();
42.             builder.MoveToElement(driver.FindElement(PageObjects.logoutButton)).C
    lick().Perform();
43.         }
44.
45.     }
46. }
```

1.6.3 The BasicMethods class

The BasicMethods class, simulates simple user actions like clicking a button or typing text in a text box element.

```
1.  using OpenQA.Selenium;
2.
3.  namespace LinkedinTest.Tests1
4.  {
5.      public class BasicMethods
6.      {
7.          public void clickElement(IWebDriver driver, By element)
8.          {
9.              driver.FindElement(element).Click();
10.         }
11.
12.         public void sendKeys(IWebDriver driver, By element, string keys)
13.         {
14.             driver.FindElement(element).SendKeys(keys);
15.         }
16.
17.     }
18. }
```

The method `clickElement` finds particular element, that is passed as parameter, and “click” on it. Similar, method sendKeys finds the element and then sends string of characters that are also passed as parameter.

1.6.4 The Validations class

The Validations class implements necessary test validations. In this example, Validation class implements three generic validations on particular elements, which are passed as parameters. The method `validateScreenByUrl` validates page on which the user is navigated, `validateElementIsPresent` validates if particular element is present on the page and `validateTextInElement` validates text that is displayed in particular element.

```
1.  using NUnit.Framework;
2.  using OpenQA.Selenium;
3.  using System;
4.
5.  namespace LinkedinTest.Tests1
6.  {
7.      public class Validations
8.      {
9.          public void validateScreenByUrl(IWebDriver driver, String screenUrl)
```

Office: Karposh bb (direkcija na “Granit”), 7000 Bitola, Macedonia.
Tel: +389 (0) 47 221 914, fax + 398 (0) 47 240 010
e-mail:interworks@interworks.com.mk
© Copyright InterWorks 2017.

```
10.         {
11.             String currentURL = driver.Url;
12.             Assert.IsTrue(currentURL.Equals(screenUrl));
13.         }
14.
15.         public void validateElementIsPresent(IWebDriver driver, By element)
16.         {
17.             IWebElement findElement = driver.FindElement(element);
18.             Assert.IsTrue(findElement.Displayed);
19.         }
20.
21.         public void validateTextInElement(IWebDriver driver, By element, String t
    ext)
22.         {
23.             String findElement = driver.FindElement(element).Text;
24.             Assert.IsTrue(findElement.Equals(text));
25.         }
26.     }
27. }
```

Validations are implemented using the **NUnits Assert** class. For example, the `Assert.IsTrue(currentURL.Equals(screenUrl));` validates that value of variable `currentURL` is same with value `screenUrl`. If these two match, the condition will return true, which is exactly what `Assert.IsTrue` expects. In other words, the assertion will pass.

1.7 Page Object Model (POM)

In the PageObjects class in this example, implements Page Object Model (POM) paradigm for implementing web automated tests using Selenium. In this class are defined global variables for web elements that will be further used in all tests. The purpose of this class, is to have the web elements defined in one single place, so if any web element is changed in the future, the tests will be easily to maintain. Also, with object declaration in one class, every object will be reusable around the entire project.

As good practice, for easy test maintenance, web objects on particular page can be separated in “groups”. Also depending on complexity of implemented tests, test project can implement multiple PageObjects classes – one per each page.

```
1.  using OpenQA.Selenium;
2.
3.  namespace LinkedinTest
4.  {
5.      public static class PageObjects
6.      {
7.          // Login screen objects
8.          public static By loginEmailInput = By.Id("login-email");
9.          public static By loginPassInput = By.Id("login-password");
10.         public static By loginButton = By.Id("login-submit");
11.         public static By forgotPassLink = By.LinkText("Forgot password?");
12.
13.         // Home screen objects
14.         public static By newsFeedElement = By.Id("ozfeed");
15.
16.         // Profile screen objects
17.         public static By fullNameElement = By.ClassName("full-name");
18.         public static By localityElement = By.ClassName("locality");
19.
20.         // Menu objects
```


© Copyright InterWorks 2017.

```
21.         public static By ProfileLinkMenu = By.XPath(".//*[@id='main-site-
   nav']/ul/li[2]/a");
22.         public static By editProfileLink = By.XPath(".//*[@id='profile-sub-
   nav']/li[1]/a");
23.         public static By addPhotoElement = By.ClassName("edit-photo-content");
24.         public static By navProfilePhoto = By.ClassName("nav-profile-photo");
25.         public static By logoutButton = By.XPath(".//*[@id='account-sub-
   nav']/div/div[2]/ul/li[1]/div/span/span[3]/a");
26.     }
27. }
```

Benefits of using POM paradigm, can be summarized as:

- The Page Object Model helps make code more readable, maintainable and reusable.
- It is a design pattern to create Object Repository for web UI elements.
- The names of the variables and methods should be given as per the task they are performing. For example `loginEmailInput – Web UI element ; Login – method`
- Page Object Pattern says operations and flows in the UI should be separated from verification. This concept makes our code cleaner and easy to understand.
- The object repository is independent of test cases, so we can re-use the same object repository, and once a change is made to web element only one change is code is needed. This is one of the greatest benefits of the pattern.

1.8 NUnit testing framework

NUnit is an open source unit testing framework for Microsoft .NET. It serves the same purpose as JUnit does in the Java world, and is one of the many programs in the xUnit family. The frameworks includes different assertions, constraints and attributes.

1.8.1 Assertions

Assertions are central in automated tests, since those check if some test output parameter has expected value. In a case when assertion fails, an error is reported, that can be further propagated in automated test reports.

NUnit offers many assertions, as methods, such as: Equality Asserts, Identity Asserts, Condition Asserts, Comparison Asserts, Type Asserts, Exception Asserts, Utility Methods, String Asserts, Collection Asserts, File Asserts, and Directory Asserts.

The Equality Assert and Condition Assert are most commonly used. Below are some examples of these type of assertions:

- Assert.AreEqual(int expected, int actual); // type can be decimal, float, double, object etc.
- Assert.AreEqual(int expected, int actual, string message);
- Assert.AreNotEqual(int expected, int actual); // type can be decimal, float, double, object etc.
- Assert.AreNotEqual(int expected, int actual, string message);
- Assert.IsTrue(bool condition);

Office: Karposh bb (direkcija na “Granit”), 7000 Bitola, Macedonia.
Tel: +389 (0) 47 221 914, fax + 398 (0) 47 240 010
e-mail:interworks@interworks.com.mk
© Copyright InterWorks 2017.

- Assert.IsTrue(bool condition, string message);
- Assert.IsFalse(bool condition);
- Assert.IsFalse(bool condition, string message);
- Assert.IsNull(object anObject);
- Assert.IsNull(object anObject, string message);
- Assert.IsEmpty(string aString);
- Assert.IsEmpty(string aString, string message);
- Assert.IsNotEmpty(string aString);
- Assert.IsNotEmpty(string aString, string message);

1.8.2 Constraints

The constraint-based Assert model uses a single method of the Assert class for all assertions. The logic necessary to carry out each assertion is embedded in the constraint object passed as the second parameter to that method.

An example of using constraint model is assertion:

```
1.  Assert.That( myString, Is.EqualTo("Hello") );
```

The second argument in this assertion uses one of NUnit's syntax helpers to create an EqualConstraint. The same assertion could also be made in this form:

```
1.  Assert.That( myString, new EqualConstraint("Hello") );
```

Using this model, all assertions are made using one of the forms of the Assert.That() method, which can be overloaded.

1.8.3 Attributes

NUnit has custom attributes such as: Setup, Test, Teardown, TestFixture, Ignore, Category, Description, Exception etc. The **Setup**, **Test**, **Teardown**, **TestFixture** and **Ignore** attributes were already used in the example above and their purpose was already explained.

The **Category** attribute provides an alternative to suites for dealing with groups of tests. Individual tests or fixtures can be identified as belonging to a particular category. When this attribute is used, only particular tests that belongs to particular Category will be run.

```
1.  namespace NUnit.Tests
2.  {
3.  using System;
4.  using NUnit.Framework;
5.  
6.  [TestFixture]
7.  public class SuccessTests
8.  {
9.  [Test]
10. [Category("Long")]
11. public void VeryLongTest()
12. { /* ... */ }
13. }
```


© Copyright InterWorks 2017.

The **Description** attribute is used to apply descriptive text to a Test, TestFixture or Assembly. The text appears in the XML output file with test results and is shown in the Test Properties dialog.

```
1.  [assembly: Description("Assembly description here")]
2.
3.  namespace NUnit.Tests
4.  {
5.    using System;
6.    using NUnit.Framework;
7.
8.    [TestFixture, Description("Fixture description here")]
9.    public class SomeTests
10.   {
11.     [Test, Description("Test description here")]
12.     public void OneTest()
13.     { /* ... */ }
14.   }
15. }
```

The **Exception** attribute specifies if particular test is expected to throw an exception. The attribute gives option to specify the exact type of expected exception, but also to include the exact text of the exception or the custom error message. The use of this attribute helps in testing negative scenarios.

```
1.  [ExpectedException( typeof( ArgumentException ), ExpectedMessage="expected messag
    e" )]
```

For more details on NUnit framework please refer to: http://www.nunit.org/

Office: Karposh bb (direkcija na “Granit”), 7000 Bitola, Macedonia.
Tel: +389 (0) 47 221 914, fax + 398 (0) 47 240 010
e-mail:interworks@interworks.com.mk
© Copyright InterWorks 2017.