xUnit – .NET Core 3.1 – Web/HTTPs API Testing

Questions: ranganathan.palanisamy@gmail.com

Tutorial Index:

1. xUnit Testing:

xUnit is a free, open source, community-focused unit testing tool for the .NET /.NET Core Frameworks. Written by the original inventor of NUnit v2, xUnit.Net is the latest technology for the unit testing C#, F#, VB.Net and Other .NET Languages.

2. xUnit support 4 Types of Testing:

1. **Facts** – are tests which are always true. They test invariant conditions. (when we have some criteria that's always must be met, regardless of data example when we test a controller's action to see if it's returning the correct view)
2. **Theory** – are tests which are only true for a set of data (ex. Takes multiple different inputs and hold true for a set of a data, whereas a fact is always true, and tests invariant conditions)

   E.g.
   - [**Fact** (Skip="this is broken")]
   - [**Theory**(Skip="we should skip this too")]
3. **InlineData** - attributes that have sample argument values for each method parameter.

   E.g.
   - [**InlineData**("RL1")]
4. **Trait** - attribute can be used to categorize related test methods by assigning an arbitrary name/value pair for each defined "Trait"

   E.g. [**Trait**("Learning Resource Tests", "Adding LR")]

   public void TestAdd() { ... }

   - [**Trait**("Resource List Tests", "Adding RL")]
     public void TestAdd { ... }

3. xUnit Package References use:

1. Microsoft.NET.Test. Sdk , 2. Xunit , 3.xunit.runner.Visualstudio,4.coverlet.collector

4. Links:

a. https://xunit.net/

b. https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-dotnet-test

c. Video Learning

i. https://learn.microsoft.com/en-us/shows/visual-studio-toolbox/unit-testing-xunit

5. **AAA** Pattern (**A**rrange-**A**ct-**A**ssert)

Pattern is a common way of writing unit tests for a method under test

1. **Arrange** section of a unit test method initializes objects and sets the value of the data that is passed to the method under test
2. **Act** section invokes the method under test with the arranged parameters
3. **Assert** section verifies that the action of the method under test behaves as expected. For .NET, methods in the Assert class are often used for verification

6. Mocking, Integration Tests and More

1. **MemberData**: use the MemberData attribute to go beyond isolated test methods. This allows you to reuse test values for multiples methods in the test class.
2. **ClassData**: use the ClassData attribute to use your test data in multiple test classes. This allows you to specify a class that will pass a set of collections to your test method.

7. Package Reference

1. xunit - 2.4.1
2. xunit.runner.visualstudio - 2.4.3

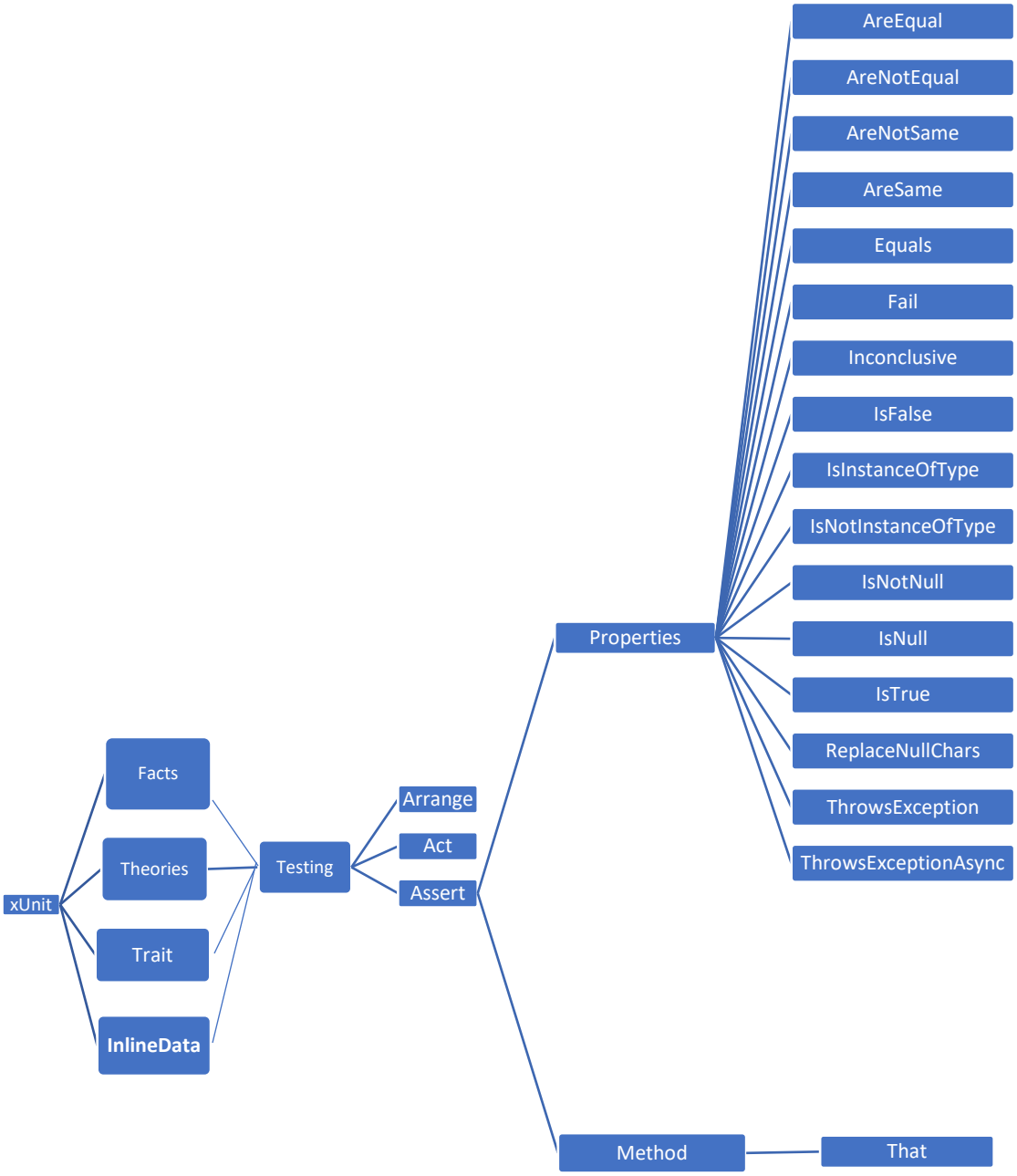
8. Mocking - Use a mocking framework (e.g. Moq) to mock external dependencies that you shouldn't need to test from your own code.

e.g.: **Moq** package (Daniel Cazzulino) **Version**: 4.16.1

9. Integration Tests - Use integration tests to go beyond isolated unit tests, to ensure that multiple components of your application are working correctly. This includes databases and file systems.

10. UI Tests - Test your UI components using a tool such as Selenium WebDriver or IDE in the language of your choice,

e.g. C#. For browser support, you may use Chrome or Firefox extensions, so this includes the new Chromium-based Edge browser.



xUnit – Assert		
No	**Functions**	**Clarification**
1	AreEqual(Double, Double, Double)	Tests whether the specified doubles are equal and throws an exception if they are not equal.
2	AreEqual(Double, Double, Double, String)	Tests whether the specified doubles are equal and throws an exception if they are not equal.
3	AreEqual(Double, Double, Double, String, Object[])	Tests whether the specified doubles are equal and throws an exception if they are not equal.
4	AreEqual(Object, Object)	Tests whether the specified objects are equal and throws an exception if the two objects are not equal. Different numeric types are treated as unequal even if the logical values are equal. 42L is not equal to 42.
5	AreEqual(Object, Object, String)	Tests whether the specified objects are equal and throws an exception if the two objects are not equal. Different numeric types are treated as unequal even if the logical values are equal. 42L is not equal to 42.
6	AreEqual(Object, Object, String, Object[])	Tests whether the specified objects are equal and throws an exception if the two objects are not equal. Different numeric types are treated as unequal even if the logical values are equal. 42L is not equal to 42.
7	AreEqual(Single, Single, Single)	Tests whether the specified floats are equal and throws an exception if they are not equal.
8	AreEqual(Single, Single, Single, String)	Tests whether the specified floats are equal and throws an exception if they are not equal.
9	AreEqual(Single, Single, Single, String, Object[])	Tests whether the specified floats are equal and throws an exception if they are not equal.
10	AreEqual(String, String, Boolean)	Tests whether the specified strings are equal and throws an exception if they are not equal. The invariant culture is used for the comparison.
11	AreEqual(String, String, Boolean, CultureInfo)	Tests whether the specified strings are equal and throws an exception if they are not equal.
12	AreEqual(String, String, Boolean, CultureInfo, String)	Tests whether the specified strings are equal and throws an exception if they are not equal.
13	AreEqual(String, String, Boolean, CultureInfo, String, Object[])	Tests whether the specified strings are equal and throws an exception if they are not equal.
14	AreEqual(String, String, Boolean, String)	Tests whether the specified strings are equal and throws an exception if they are not equal. The invariant culture is used for the comparison.
15	AreEqual(String, String, Boolean, String, Object[])	Tests whether the specified strings are equal and throws an exception if they are not equal. The invariant culture is used for the comparison.

16	AreEqual<T>(T, T)	Tests whether the specified values are equal and throws an exception if the two values are not equal. Different numeric types are treated as unequal even if the logical values are equal. 42L is not equal to 42.
17	AreEqual<T>(T, T, String)	Tests whether the specified values are equal and throws an exception if the two values are not equal. Different numeric types are treated as unequal even if the logical values are equal. 42L is not equal to 42.
18	AreEqual<T>(T, T, String, Object[])	Tests whether the specified values are equal and throws an exception if the two values are not equal. Different numeric types are treated as unequal even if the logical values are equal. 42L is not equal to 42.
19	AreNotEqual(Double, Double, Double)	Tests whether the specified doubles are unequal and throws an exception if they are equal.
20	AreNotEqual(Double, Double, Double, String)	Tests whether the specified doubles are unequal and throws an exception if they are equal.
21	AreNotEqual(Double, Double, Double, String, Object[])	Tests whether the specified doubles are unequal and throws an exception if they are equal.
22	AreNotEqual(Object, Object)	Tests whether the specified objects are unequal and throws an exception if the two objects are equal. Different numeric types are treated as unequal even if the logical values are equal. 42L is not equal to 42.
23	AreNotEqual(Object, Object, String)	Tests whether the specified objects are unequal and throws an exception if the two objects are equal. Different numeric types are treated as unequal even if the logical values are equal. 42L is not equal to 42.
24	AreNotEqual(Object, Object, String, Object[])	Tests whether the specified objects are unequal and throws an exception if the two objects are equal. Different numeric types are treated as unequal even if the logical values are equal. 42L is not equal to 42.
25	AreNotEqual(Single, Single, Single)	Tests whether the specified floats are unequal and throws an exception if they are equal.
26	AreNotEqual(Single, Single, Single, String)	Tests whether the specified floats are unequal and throws an exception if they are equal.
27	AreNotEqual(Single, Single, Single, String, Object[])	Tests whether the specified floats are unequal and throws an exception if they are equal.
28	AreNotEqual(String, String, Boolean)	Tests whether the specified strings are unequal and throws an exception if they are equal. The invariant culture is used for the comparison.
29	AreNotEqual(String, String, Boolean, CultureInfo)	Tests whether the specified strings are unequal and throws an exception if they are equal.

30	AreNotEqual(String, String, Boolean, CultureInfo, String)	Tests whether the specified strings are unequal and throws an exception if they are equal.
31	AreNotEqual(String, String, Boolean, CultureInfo, String, Object[])	Tests whether the specified strings are unequal and throws an exception if they are equal.
32	AreNotEqual(String, String, Boolean, String)	Tests whether the specified strings are unequal and throws an exception if they are equal. The invariant culture is used for the comparison.
33	AreNotEqual(String, String, Boolean, String, Object[])	Tests whether the specified strings are unequal and throws an exception if they are equal. The invariant culture is used for the comparison.
34	AreNotEqual<T>(T, T)	Tests whether the specified values are unequal and throws an exception if the two values are equal. Different numeric types are treated as unequal even if the logical values are equal. 42L is not equal to 42.
35	AreNotEqual<T>(T, T, String)	Tests whether the specified values are unequal and throws an exception if the two values are equal. Different numeric types are treated as unequal even if the logical values are equal. 42L is not equal to 42.
36	AreNotEqual<T>(T, T, String, Object[])	Tests whether the specified values are unequal and throws an exception if the two values are equal. Different numeric types are treated as unequal even if the logical values are equal. 42L is not equal to 42.
37	AreNotSame(Object, Object)	Tests whether the specified objects refer to different objects and throws an exception if the two inputs refer to the same object.
38	AreNotSame(Object, Object, String)	Tests whether the specified objects refer to different objects and throws an exception if the two inputs refer to the same object.
39	AreNotSame(Object, Object, String, Object[])	Tests whether the specified objects refer to different objects and throws an exception if the two inputs refer to the same object.
40	AreSame(Object, Object)	Tests whether the specified objects both refer to the same object and throws an exception if the two inputs do not refer to the same object.
41	AreSame(Object, Object, String)	Tests whether the specified objects both refer to the same object and throws an exception if the two inputs do not refer to the same object.
42	AreSame(Object, Object, String, Object[])	Tests whether the specified objects both refer to the same object and throws an exception if the two inputs do not refer to the same object.

43	Equals(Object, Object)	Static equals overloads are used for comparing instances of two types for reference equality. This method should not be used for comparison of two instances for equality. This object will always throw with Assert.Fail. Please
44	Fail()	Throws an AssertFailedException.
45	Fail(String)	Throws an AssertFailedException.
46	Fail(String, Object[])	Throws an AssertFailedException.
47	Inconclusive()	Throws an AssertInconclusiveException.
48	Inconclusive(String)	Throws an AssertInconclusiveException.
49	Inconclusive(String, Object[])	Throws an AssertInconclusiveException.
50	IsFalse(Boolean)	Tests whether the specified condition is false and throws an exception if the condition is true.
51	IsFalse(Boolean, String)	Tests whether the specified condition is false and throws an exception if the condition is true.
52	IsFalse(Boolean, String, Object[])	Tests whether the specified condition is false and throws an exception if the condition is true.
53	IsInstanceOfType(Object, Type)	Tests whether the specified object is an instance of the expected type and throws an exception if the expected type is not in the inheritance hierarchy of the object.
54	IsInstanceOfType(Object, Type, String)	Tests whether the specified object is an instance of the expected type and throws an exception if the expected type is not in the inheritance hierarchy of the object.
55	IsInstanceOfType(Object, Type, String, Object[])	Tests whether the specified object is an instance of the expected type and throws an exception if the expected type is not in the inheritance hierarchy of the object.
56	IsNotInstanceOfType(Object, Type)	Tests whether the specified object is not an instance of the wrong type and throws an exception if the specified type is in the inheritance hierarchy of the object.
57	IsNotInstanceOfType(Object, Type, String)	Tests whether the specified object is not an instance of the wrong type and throws an exception if the specified type is in the inheritance hierarchy of the object.
58	IsNotInstanceOfType(Object, Type, String, Object[])	Tests whether the specified object is not an instance of the wrong type and throws an exception if the specified type is in the inheritance hierarchy of the object.
59	IsNotNull(Object)	Tests whether the specified object is non-null and throws an exception if it is null.
60	IsNotNull(Object, String)	Tests whether the specified object is non-null and throws an exception if it is null.
61	IsNotNull(Object, String, Object[])	Tests whether the specified object is non-null and throws an exception if it is null.
62	IsNull(Object)	Tests whether the specified object is null and throws an exception if it is not.

63	IsNull(Object, String)	Tests whether the specified object is null and throws an exception if it is not.
64	IsNull(Object, String, Object[])	Tests whether the specified object is null and throws an exception if it is not.
65	IsTrue(Boolean)	Tests whether the specified condition is true and throws an exception if the condition is false.
66	IsTrue(Boolean, String)	Tests whether the specified condition is true and throws an exception if the condition is false.
67	IsTrue(Boolean, String, Object[])	Tests whether the specified condition is true and throws an exception if the condition is false.
68	ReplaceNullChars(String)	Replaces null characters ('\0') with "\0".
69	ThrowsException<T>()	Tests whether the code specified by delegate action throws exact given exception of type T (and not of derived type) and throws AssertFailedException if code does not throws exception or throws exception of type other than T.
70	ThrowsException<T>()	Tests whether the code specified by delegate action throws exact given exception of type T (and not of derived type) and throws AssertFailedException if code does not throws exception or throws exception of type other than T.
71	ThrowsException<T>(Action, String)	Tests whether the code specified by delegate action throws exact given exception of type T (and not of derived type) and throws AssertFailedException if code does not throws exception or throws exception of type other than T.
72	ThrowsException<T>(Action, String, Object[])	Tests whether the code specified by delegate action throws exact given exception of type T (and not of derived type) and throws AssertFailedException if code does not throws exception or throws exception of type other than T.
73	ThrowsException<T>(Func<Object>, String)	Tests whether the code specified by delegate action throws exact given exception of type T (and not of derived type) and throws AssertFailedException if code does not throws exception or throws exception of type other than T.
74	ThrowsException<T>(Func<Object>, String, Object[])	Tests whether the code specified by delegate action throws exact given exception of type T (and not of derived type) and throws AssertFailedException if code does not throws exception or throws exception of type other than T.
75	ThrowsExceptionAsync<T>()	Tests whether the code specified by delegate action throws exact given exception of type T (and not of derived type) and throws AssertFailedException if code does not throws exception or throws exception of type other than T.

76	ThrowsExceptionAsync<T>(Func<Task>)	Tests whether the code specified by delegate action throws exact given exception of type T (and not of derived type) and throws AssertFailedException if code does not throws exception or throws exception of type other than T.
77	ThrowsExceptionAsync<T>(Func<Task>, String, Object[])	Tests whether the code specified by delegate action throws exact given exception of type T (and not of derived type) and throws AssertFailedException if code does not throw exception or throws exception of type other than T.

Lambda Example Git Repo:

- https://github.com/ranganathanarul/AWSVisualStudio.git

API Example Git Repo:

- https://github.com/ranganathanarul/xUnitTesting.git