**Pruebas unitarias con xUnit en .NET Core**

**Introducción**

Unit Testing es un patrón de diseño de software que se utiliza para probar los componentes más pequeños en la fase de desarrollo de software.

Las pruebas unitarias se utilizan para validar la funcionalidad que consiste en crear el resultado esperado antes de pasar al entorno de producción y al equipo de control de calidad.

Ayuda a detectar problemas en la fase inicial del ciclo de desarrollo de software.

Hay muchas herramientas de prueba unitaria que ya están presentes al usar .NET Framework como xUnit, NUnit y muchas más.

**xUnit**

xUnit es un marco de pruebas unitario gratuito y de código abierto para el desarrollo de .NET

xUnit tiene muchas características que permiten escribir un caso de prueba unitario limpio y bueno.

Tiene muchos atributos como Hecho, Teoría y muchos más para escribir casos de prueba de manera efectiva y limpia y también proporciona un mecanismo para crear nuestro propio atributo.

**Atributos de xUnit**

xUnit utiliza el atributo [Fact] en .NET que identifica el método para la prueba unitaria

[Fact]

public void EvenNumberTest() {

//Arrange

var num = 6;

//Act

bool result = Mathematics.IsEvenNumber(num);

//Assert

Assert.True(result);

}

El atributo [Theory] se utiliza para proporcionar parámetros al método de prueba.

[Theory]

[InlineData(5)]

public void OddNumberTest(int num) {

//Act

bool result = Mathematics.IsOddNumber(num);

//Assert

Assert.True(result);

}

C#

**Patrón de prueba**

Arrange-Act-Assert es una excelente manera de escribir casos de prueba unitarios limpios y más legibles

**Arrange**

En la sección de organización configuramos y declaramos algunas entradas y variables de configuración.

**Act**

En la sección **Act**, colocamos las cosas y funcionalidades principales, como llamadas a métodos, llamadas a API y algo así.

**Assert**

Assert verifica los resultados esperados y verifica si coincidirán con nuestros requisitos funcionales o no.

Comencemos con un ejemplo práctico usando .NET Core y xUnit.

**Paso 1**

Abra VS Code y cree un directorio "UnitTesting" y luego cree una nueva solución allí usando el siguiente comando

dotnet new sln

**Paso 2**

Cree un nuevo proyecto de biblioteca de clases TestDemo

dotnet new classlib -o TestDemo

**Paso 3**

Luego vaya al directorio donde está el archivo de la solución y luego agregue el proyecto TestDemo al archivo de la solución.

dotnet sln add .\TestDemo\TestDemo.csproj

BÁSICO

**Etapa 4**

Más adelante, cree el proyecto xUnit TestDemo.Tests

dotnet new xunit -o TestDemo.Tests

BÁSICO

**Paso 5**

Agregue el proyecto TestDemo.Tests xUnit al archivo de solución

dotnet sln add ./TestDemo.Tests/TestDemo.Tests.csproj

BÁSICO

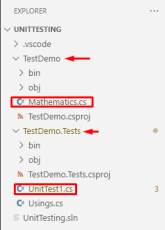
**Paso 6**

Agregue la referencia del proyecto TestDemo al proyecto TestDemo.Tests que está relacionado con xUnit

dotnet add ./TestDemo.Tests/TestDemo.Tests.csproj reference ./TestDemo/TestDemo.csp roj

**Paso 7**

Estructura del proyecto



**Paso 8**

Cree una clase de Matemáticas en el Proyecto TestDemo como se muestra a continuación

namespace TestDemo;

public static class Mathematics {

public static bool IsEvenNumber(int num) {

return num % 2 == 0 ? true : false;

}

public static bool IsOddNumber(int num) {

return num % 2 == 1 ? true : false;

}

public static int SquareOfNumber(int num) {

int sum = 0;

for (int count = 1; count <= num; count++) {

sum = count \* count;

}

return sum;

}

public static double Addition(double num1, double num2) { return (num1 + num2);

}

public static double Subtraction(double num1, double num2) { return (num1 - num2);

}

public static double Multiplication(double num1, double num2) {

return (num1 \* num2);

}

public static double Division(double num1, double num2) {

return (num1 / num2);

}

}

C#

**Paso 9**

A continuación, cree la clase UnitTest1 en TestDemo.Tests en la que escribimos diferentes casos de prueba.

namespace TestDemo.Tests;

public class UnitTest1 {

[Fact]

public void EvenNumberTest() {

//Arrange

var num = 6;

//Act

bool result = Mathematics.IsEvenNumber(num);

//Assert

Assert.True(result);

}

[Fact]

public void OddNumberTest() {

//Arrange

var num = 5;

//Act

bool result = Mathematics.IsOddNumber(num);

//Assert

Assert.True(result);

}

[Fact]

public void SquareOfNumberTest() {

//Arrange

var num = 5;

var expectedResult = 25;

//Act

var result = Mathematics.SquareOfNumber(num);

//Assert

Assert.Equal(expectedResult, result);

}

[Fact]

public void AdditionOfNumber() {

//Arrange

var num1 = 5;

var num2 = 2;

var expectedResult = 7;

//Act

var result = Mathematics.Addition(num1, num2);

//Assert

Assert.Equal(expectedResult, result);

}

[Fact]

public void SubtractionOfNumber() {

//Arrange

var num1 = 5;

var num2 = 2;

var expectedResult = 3;

//Act

var result = Mathematics.Subtraction(num1, num2); //Assert

Assert.Equal(expectedResult, result);

}

[Fact]

public void MultiplicationOfNumber() {

//Arrange

var num1 = 5;

var num2 = 2;

var expectedResult = 10;

//Act

var result = Mathematics.Multiplication(num1, num2); //Assert

Assert.Equal(expectedResult, result);

}

[Fact]

public void DivisionOfNumber() {

//Arrange

var num1 = 10;

var num2 = 2;

var expectedResult = 5;

//Act

var result = Mathematics.Division(num1, num2);

//Assert

Assert.Equal(expectedResult, result);

}

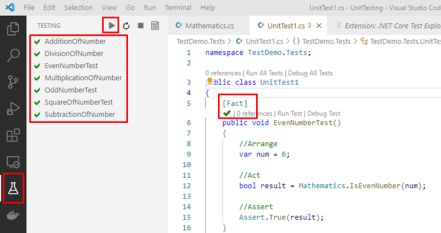
}

C#

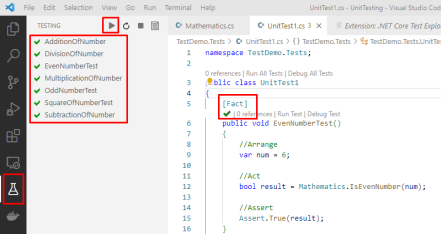
**Paso 10**

6 / 8

Le sugeriré que instale .NET Core Test Explorer para el caso de prueba.

 **Paso 11**

Finalmente, puede ejecutar su caso de prueba como se muestra a continuación y puede obtener el resultado de todos sus casos de prueba si se ejecuta correctamente, entonces el clic verde aparece delante del nombre del caso de prueba y debajo del atributo [Fact] en la clase de Matemáticas. y si la aplicación no produce el resultado esperado, el signo cruzado aparece delante del caso de prueba y debajo del atributo [Fact]



**Conclusión**

Discutimos qué es un caso de prueba unitaria y su propósito, y más adelante, la introducción de xUnit, atributos y algunos patrones. Además, la implementación práctica usando .NET Core 6 en VS Code paso a paso.

8 / 8