**Unit & Integration Testing para ASP.NET en Web Forms**

**Unit Testing:**

Herramienta principal xUnit.

* **¿Por qué es útil?**
* Es una herramienta moderna y ligera, útil para realizar pruebas rápidas.
* Es compatible con muchos patrones avanzados.
* Está soportada por Microsoft.
* Soporta paralelismos para ejecutar las pruebas más rápido.
* **Herramientas de xUnit para casos específicos:**

| **Tipo de prueba** | **Mejor Herramienta** | **Justificación** |
| --- | --- | --- |
| Lógica del negocio | xUnit + Moq | xUnit para pruebas generales, Moq para crear mocks de dependencias. |
| Validación de datos | xUnit + FluentAssertions | FluentAssertions permite escribir aserciones legibles y expresivas. |
| Funciones utilitarias | xUnit | Sintaxis clara y directa, útil en funciones independientes. |
| Generación de datos de prueba | AutoFixture | Genera los datos de prueba necesarios sin necesidad de su ingreso manual. |

* **¿Por qué en Web Forms?**

| **Herramienta** | **Justificación** |
| --- | --- |
| xUnit | * Web Forms no tiene un enfoque modular por diseño, xUnit permite aislar clases para probar métodos individuales. * Su flexibilidad facilita la migración de pruebas si se decide pasar a un framework más moderno en el futuro. |
| Moq | * Permite probar lógicas como el acceso a datos en la capa de repositorios sin conectarse a una base de datos real. * Es útil para simular objetos como HttpContext, sesiones o cookies en tus pruebas. |
| FluentAssertions | * Hace que las pruebas sean más legibles, lo cual es ideal cuando se maneja lógica de validación en formularios y se requiere que otros desarrolladores las comprendan fácilmente. |
| AutoFixture | * Muy práctico para llenar objetos complejos como ViewModels o datos que tu aplicación usa en sus formularios. |

* **¿Cómo implementarlo?**
* **Preparación del entorno:**

1. Configuración del proyecto:

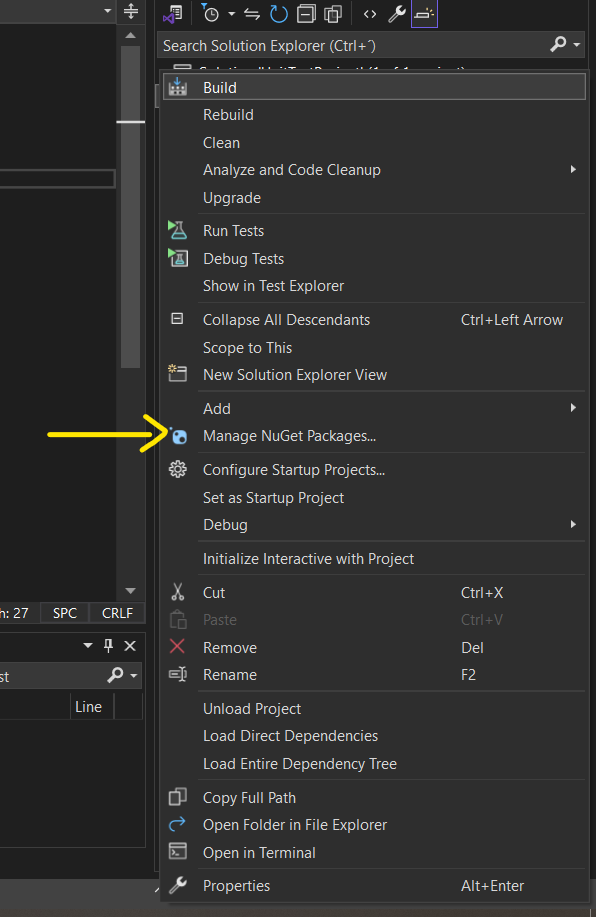
Crear dos proyectos en la misma solución:

* Un proyecto base (en este caso, Web Forms) para contener la lógica.
* Un proyecto de pruebas con xUnit (xUnit Test Project)

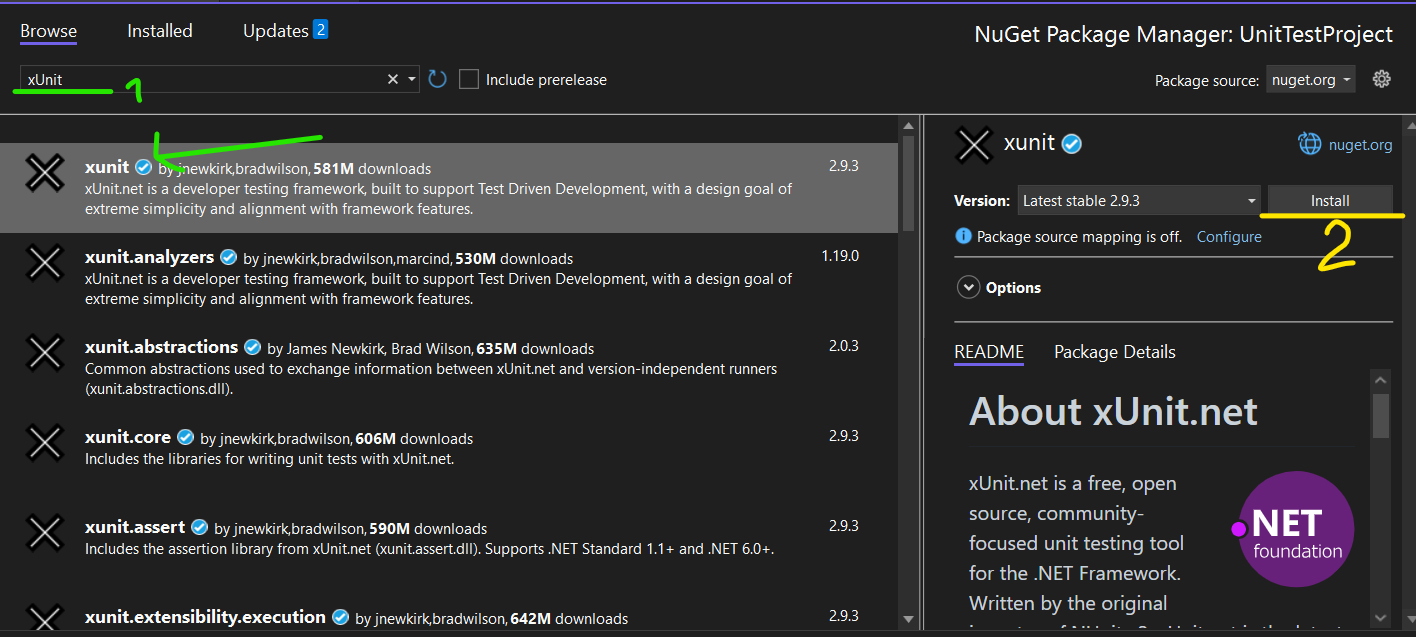
Enlazar el proyecto de pruebas con el proyecto base usando referencias.

1. Instalar herramientas necesarias:

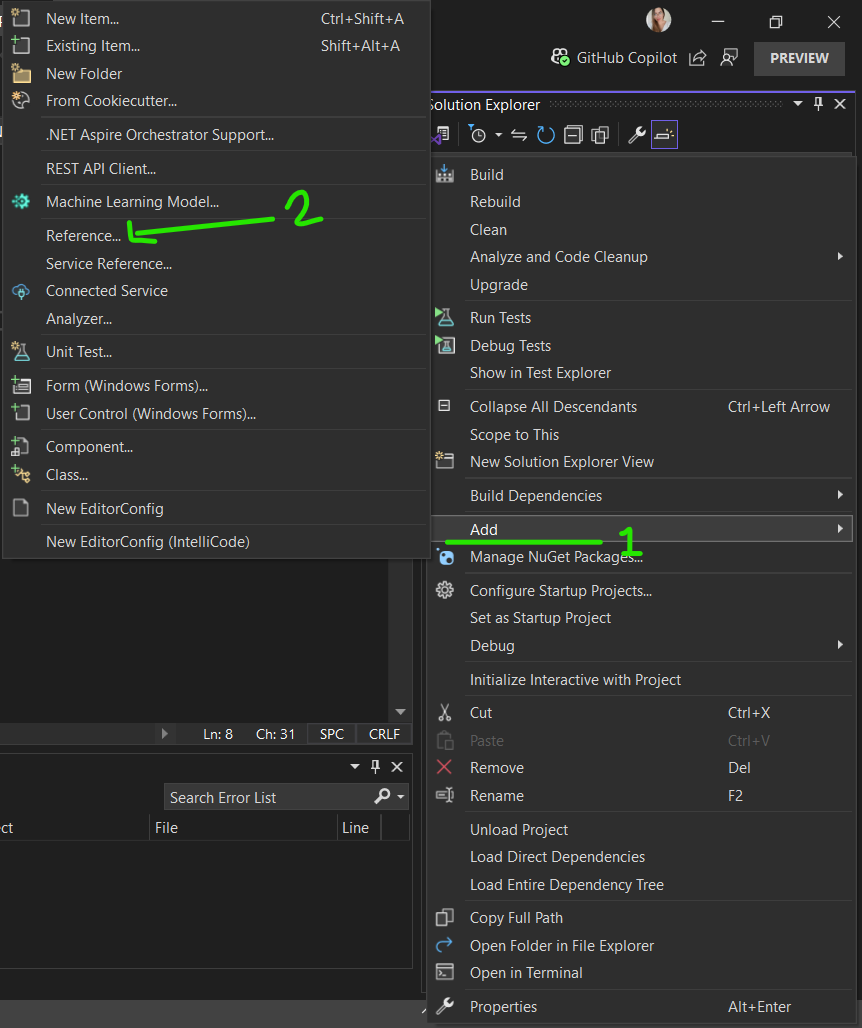
Click derecho en la solución del proyecto>Manage NuGet Packages…



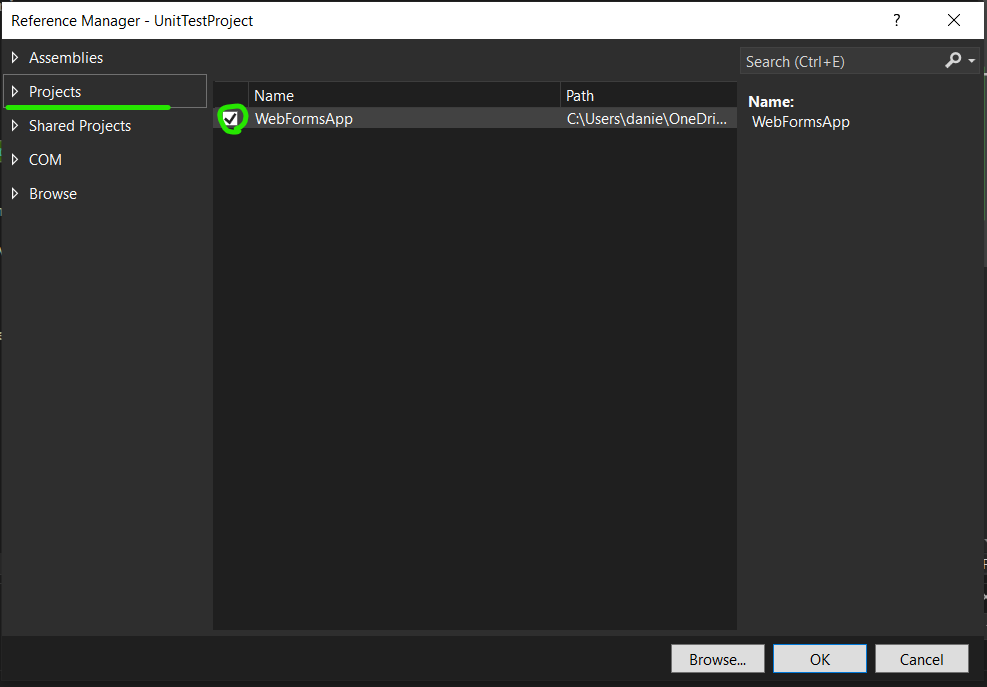
Instalar xUnit buscándolo en el área de examinar:



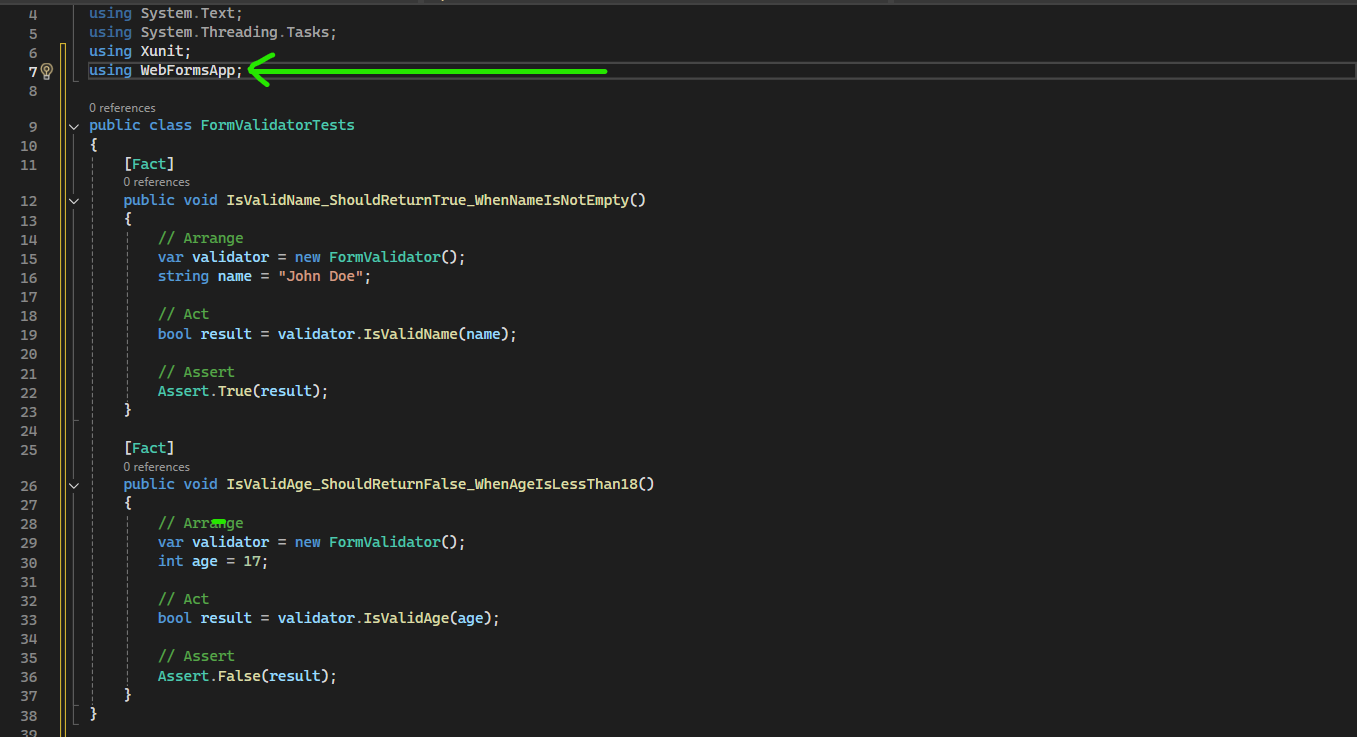
1. Enlazar el proyecto click derecho en el proyecto de unit testing (UnitTestProject) > Add > Reference…



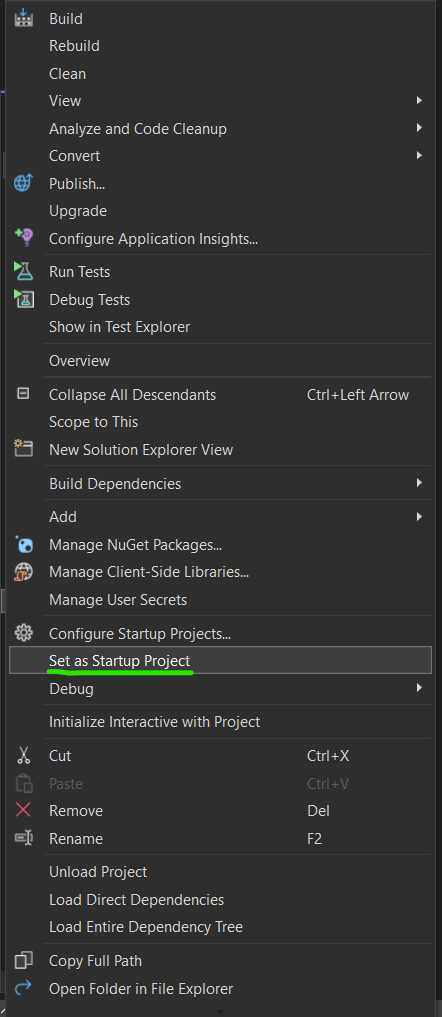
1. Buscar en la pestaña el proyecto que se desea agregar:



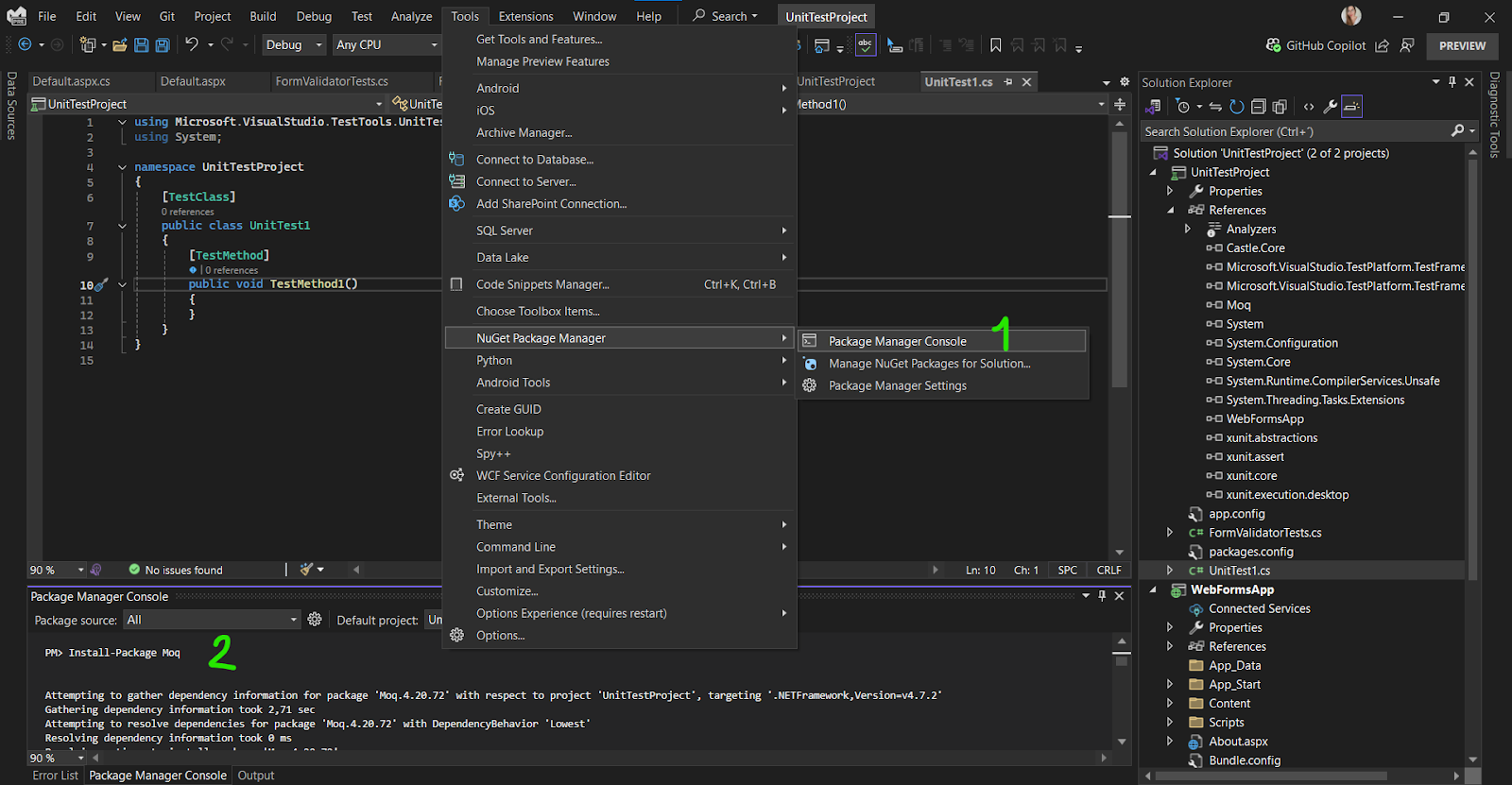
1. Agregar una nueva clase en el proyecto de pruebas para colocar la lógica de las mismas, haciendo referencia al proyecto que se agregó:



1. Establecer el proyecto agregado como el de inicialización, con click derecho en el proyecto>set as Startup Project:

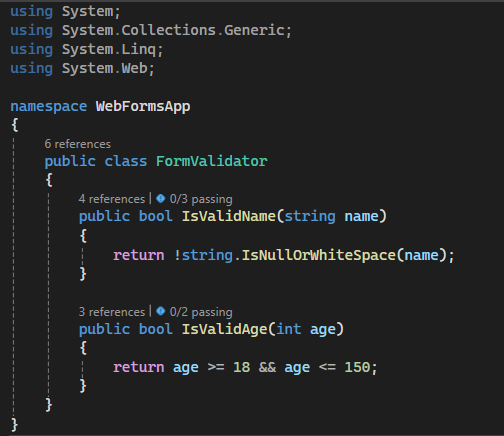


1. Instalar paquete de mocks con los paquetes NuGet:



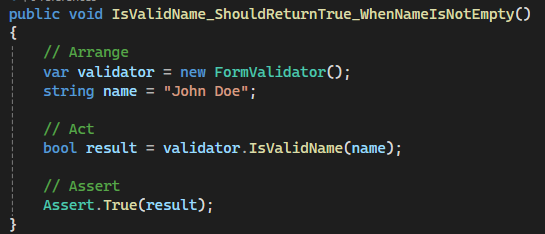
* **Ejemplo para Pruebas de funciones Utilitarias:**

#### Crear la clase base para la lógica: en el proyecto principal (Web Forms), se agregó la clase *FormValidator* con métodos para validar entradas:

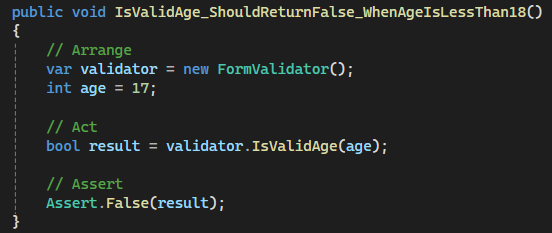


1. Crear pruebas unitarias con xUnit: en el proyecto de pruebas, agregar la clase FormValidatorTests para definir las pruebas:

* Caso 1: Nombre no vacío:

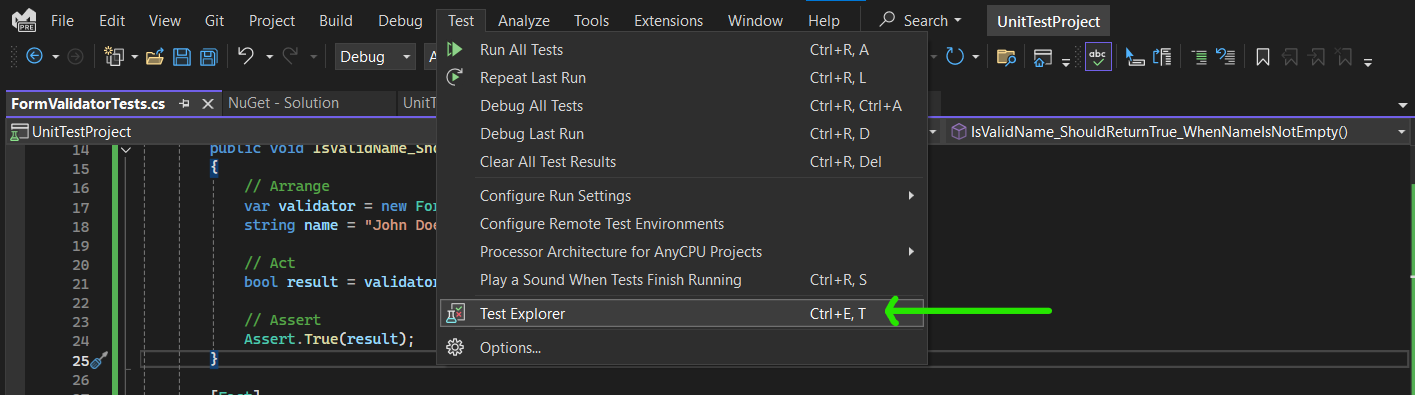


* Caso 2: Edad menor a 18:

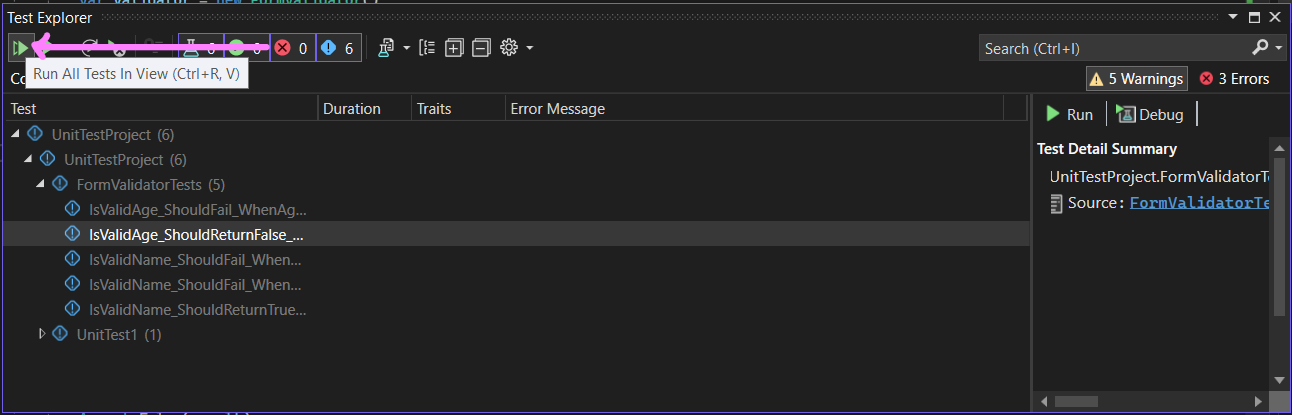


* **Ejecutar las pruebas:**

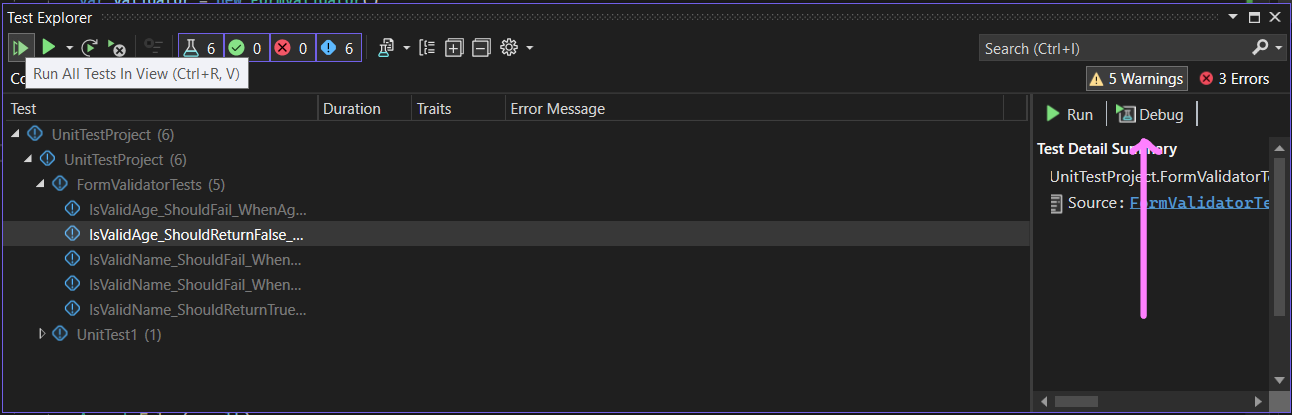
1. Abrir el Test Explorer en Visual Studio.



1. Ejecutar todas las pruebas con Run All.



1. Depurar las pruebas si es necesario.



* **¿Qué hace?** 
  + Pruebas manuales donde se definen explícitamente los valores de entrada (name y age).
  + Cada caso prueba un escenario específico con datos controlados.
* **Ejemplo de Generación de datos de prueba:**

**Instalar AutoFixture:**

1. Ir a NuGet Package Manager:

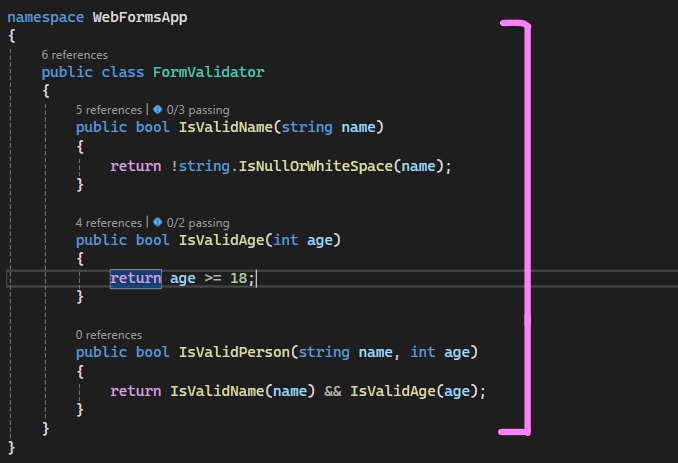
* Hacer clic derecho en el proyecto de pruebas > Manage NuGet Packages.

1. Buscar e instalar los siguientes paquetes:

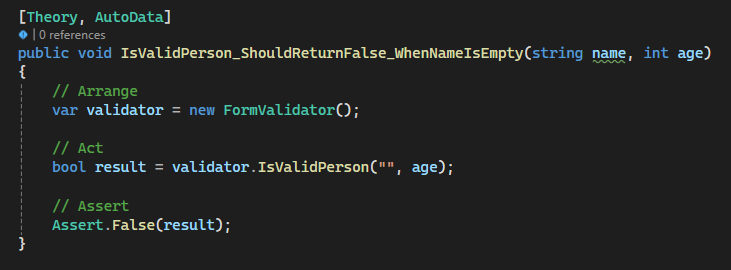
* AutoFixture
* AutoFixture.Xunit2
* **Creación de ejemplo para AutoFixture:**

**Escenario:** se probará una función que toma un nombre y una edad, y valida si ambos cumplen ciertas condiciones:

1. Crear la lógica de nuestras futuras pruebas



1. Crear pruebas con AutoFixture: en el proyecto de pruebas, usar AutoFixture para generar automáticamente valores para name y age.



1. Ejecutar de igual manera que con xUnit únicamente.

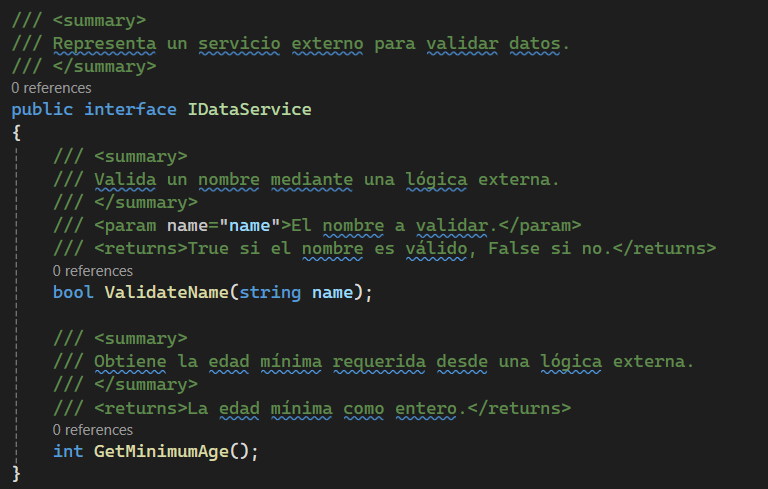
* **¿Qué hace?** 
  + Genera automáticamente valores de entrada aleatorios y variados para los parámetros de las pruebas.
  + Aumenta la cobertura al probar diferentes combinaciones sin necesidad de escribirlas manualmente.
* **Ejemplo de pruebas para Moqs:**

1. Configuración inicial para usar Moq:

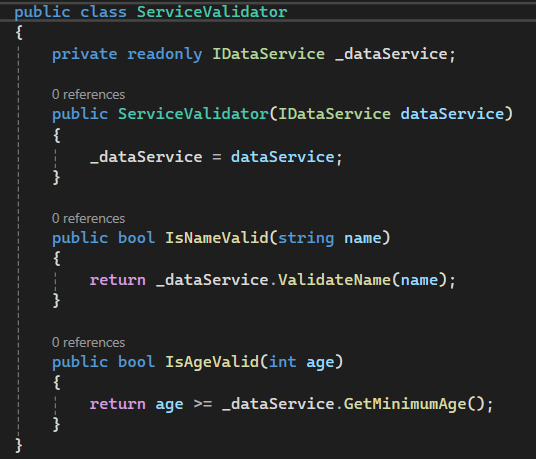
* Clic derecho en el proyecto de pruebas > Manage NuGet Packages.
* Buscar e instalar: Moq, AutoFixture.AutoMoq.

#### Creación de la interfaz para simulación:

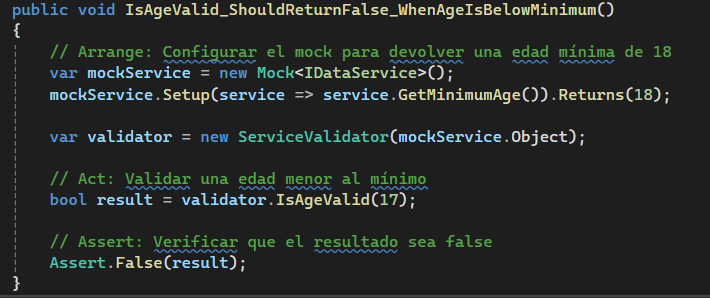
* En este caso en el proyecto Web Forms (donde está la lógica), se creará una interfaz que representa un servicio externo:



* Para este caso se hará una nueva clase de lógica para probar Moq (Service Validator), ahora l os datos dependen de la interfaz:



* Creación de pruebas para Moq:



1. Las pruebas se ejecutan de la misma manera que antes.

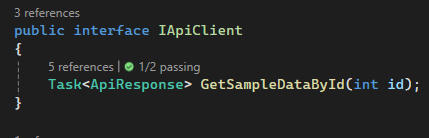
* **Ejemplo 2 con Moq:**

**Creación de la interfaz para simulación:**

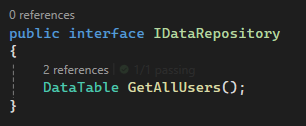
En este caso, en el proyecto Web Forms (donde está la lógica), se crearon dos interfaces que representan las dependencias externas:

1. IApiClient: Representa el cliente para consumir datos desde una API externa.
2. IDataRepository: Representa la capa de acceso a datos desde la base de datos.

Código de IApiClient:



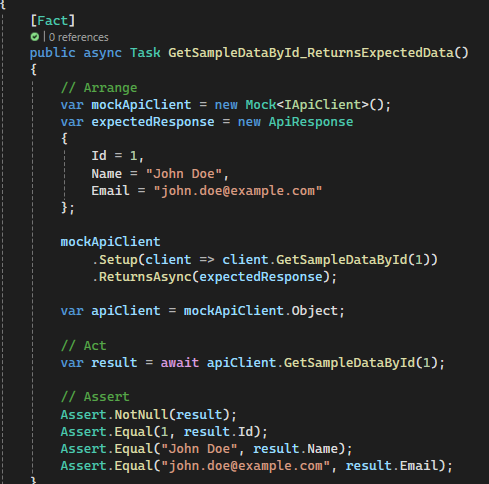
Código de IDataRepository:



### **Creación de lógica para pruebas con Moq:**

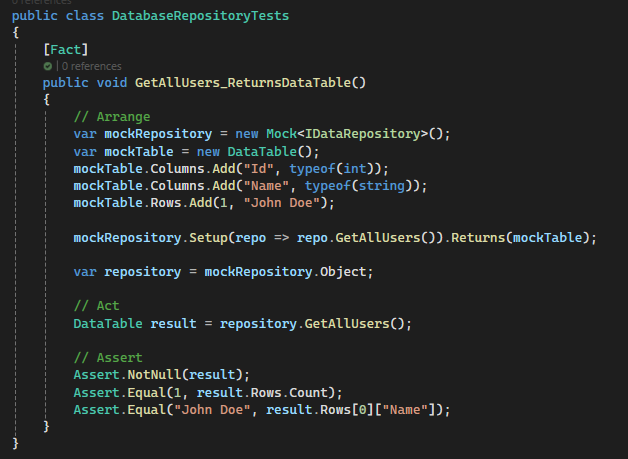
**Para la API:** el cliente real, ApiClient, fue simulado utilizando Moq. El siguiente ejemplo prueba que, al consumir datos desde la API con un ID válido, se obtienen los resultados esperados.

#### Código de prueba para IApiClient:



**Para la base de datos:** La capa de acceso a la base de datos (IDataRepository) fue simulada para devolver un conjunto de datos ficticio. Esto asegura que la lógica que consume estos datos funcione correctamente.

#### Código de prueba para IDataRepository:



### **Ejecución de pruebas:**

### Las pruebas se ejecutan de la misma manera que con xUnit:

### **¿Qué hace con las dependencias?**

* **Pruebas para la API:**
  + Verifica que al consumir datos desde la API con un ID válido, los datos devueltos coincidan con lo esperado.
  + Valida que se manejen correctamente los casos en que la API no devuelve datos.
* **Pruebas para la base de datos:**
  + Simula la capa de acceso a datos para garantizar que la lógica que consume estos datos funcione correctamente.
  + Genera datos ficticios para representar filas de una tabla.