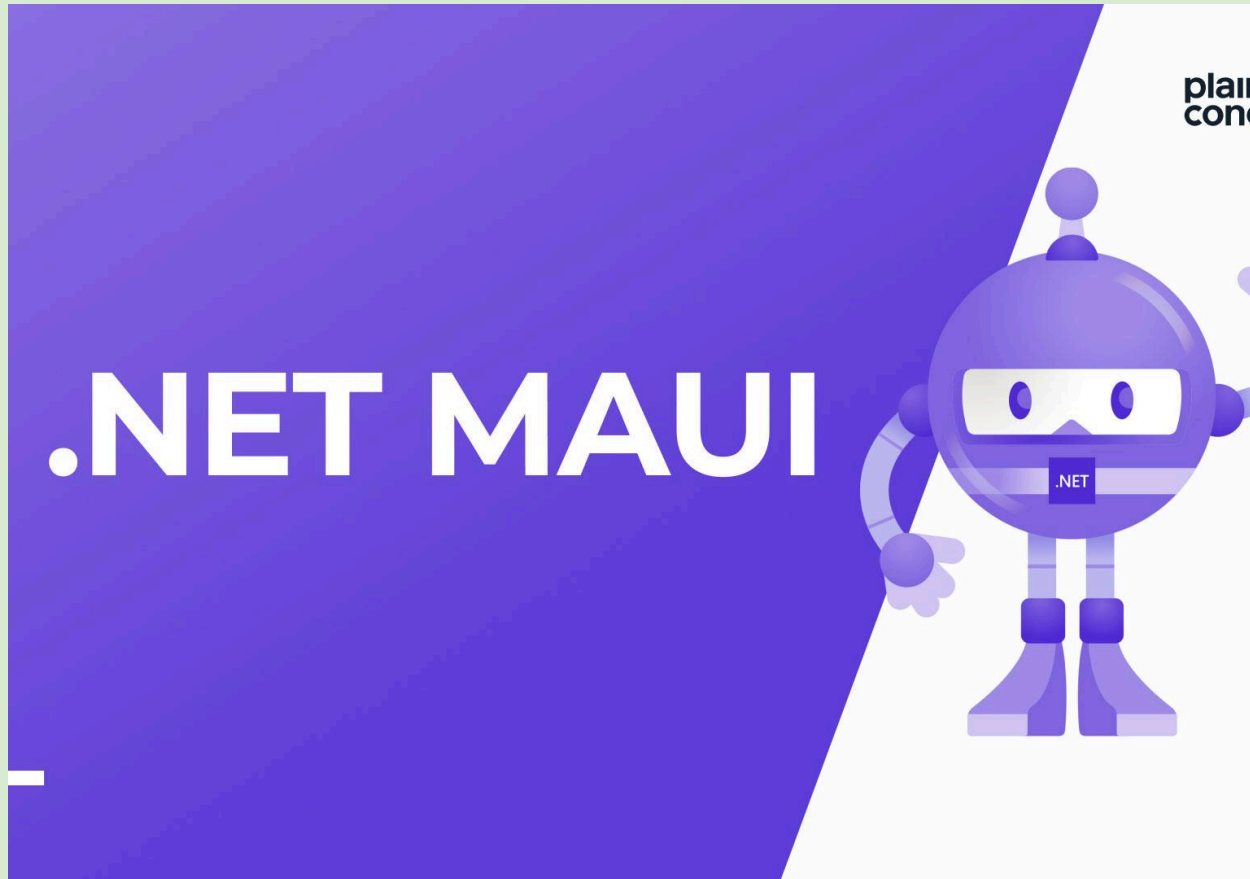


# UNIDAD 04 ACTIVIDAD 02



**Azael Morell Martínez**


29/01/2025

Desarrollo de Interfaces

# ÍNDICE:

---

[ÍNDICE:](#)

[INTRODUCCIÓN](#) 

[OBJETIVOS](#) 

[PLANTEAMIENTOS PREVIOS](#) 

[DIAGRAMA DE GANTT:](#) 

[RESULTADOS](#) 

[VISTA GENERAL DEL PROYECTO:](#)


[SUGERENCIA:](#)

[EL CÓDIGO .XAML:](#)

[EL CÓDIGO .XAML.CS:](#)

[Descripción Técnica:](#)

[UnitTest1.xaml.cs Y TESTING](#)

[PROBLEMA ENCONTRADO](#) 

[SUGERENCIAS PARA HACER LA APP](#) 

[CONCLUSIÓN](#) 

[WEBGRAFÍA:](#) 

## INTRODUCCIÓN

En este proyecto, he desarrollado una aplicación que implementa operaciones matemáticas básicas utilizando C# y el marco de trabajo .NET. La lógica principal de las operaciones se encuentra en una clase llamada Operaciones, y la interfaz de usuario está gestionada por la clase MainViewModel. Además, se han realizado pruebas unitarias para garantizar la correcta funcionalidad del código utilizando el marco de trabajo xUnit.

## OBJETIVOS

El objetivo principal de este proyecto es proporcionar una herramienta sencilla y eficiente para realizar operaciones matemáticas básicas como suma, resta, multiplicación y división. Además, se busca mantener una arquitectura de código limpio y modular que facilite la extensión y el mantenimiento del proyecto. Otro objetivo crucial es garantizar la calidad del código mediante la implementación de pruebas unitarias exhaustivas.

## PLANTEAMIENTOS PREVIOS

Antes de comenzar, era fundamental tener una comprensión sólida de:

- La estructura y funcionamiento de .NET MAUI.
- Cómo trabajar con XAML para diseñar la interfaz de usuario.
- La implementación de métodos asíncronos en C#.

## DIAGRAMA DE GANTT:

<u>Tarea</u>	<u>1-3 días</u>	<u>4-7 días</u>	<u>más de una semana</u>
Desarrollo del código y de la interfaz	xxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxx		
Desarrollo de la documentación	xxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxx		
Testing del funcionamiento	xxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxx		

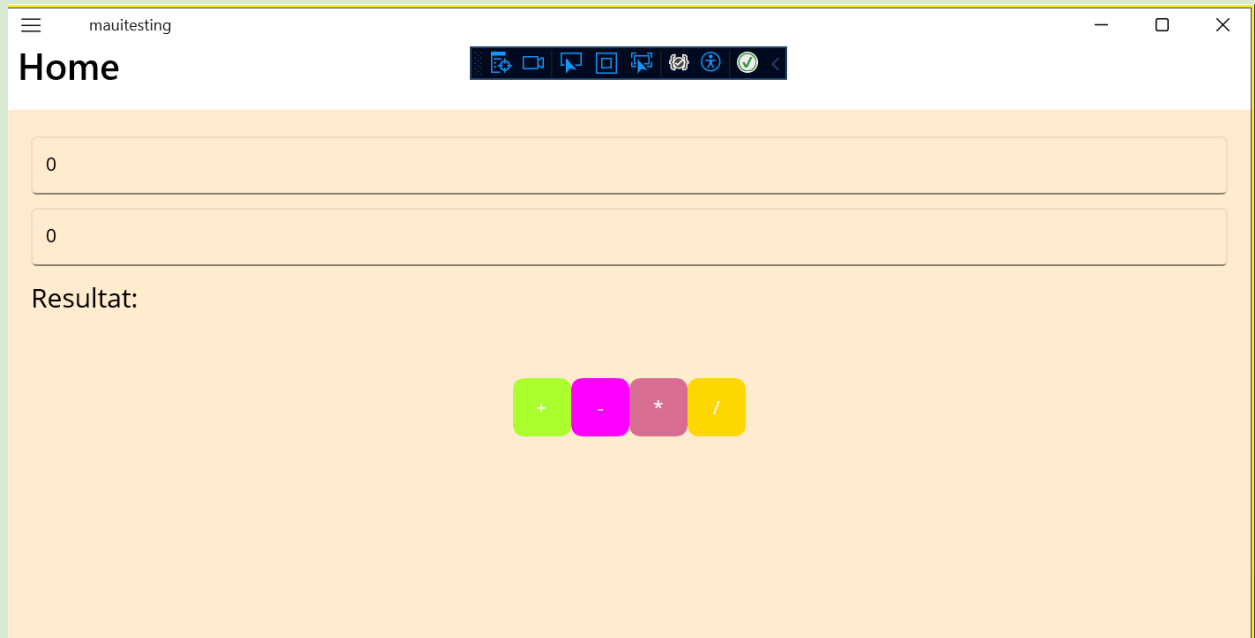
## RESULTADOS 💪

### VISTA GENERAL DEL PROYECTO:

✨ Estoy súper emocionado de compartir el diseño de mi app de calculadora. La he llevado a cabo en .NET MAUI y ha quedado genial. ¡Déjame contarte más al respecto!

#### 📝 Diseño de la App:

- **Interfaz Sencilla** 🎨: El fondo tiene un color beige claro, que le da un aspecto limpio y agradable.
- **Título** 📄: Arriba, el título dice "Home" en letras negras sobre un fondo blanco. Muy claro y directo.
- **Campos de Entrada** 📱: Justo debajo, hay dos campos de entrada con el valor inicial de "0" para ingresar los números que se desean operar.
- **Resultado** 📊: Debajo de los campos de entrada, hay un texto que dice "Resultat:" donde se mostrará el resultado de la operación.
- **Botones Funcionales** ☀️:
  - Botón verde que dice "+" ➕ para sumar.
  - Botón rosa que dice "-" ➖ para restar.
  - Botón rojo que dice "\*" ✖ para multiplicar.
  - Botón amarillo que dice "/" ➗ para dividir.



## SUGERENCIA:

SI QUIERES VER EL FUNCIONAMIENTO DE ESTA APP. RECOMIENDO VER EL VIDEO LLAMADO "TestingAPP\_ListaTareas" QUE ESTÁ SUBIDO EN ESTE MISMO REPOSITORIO DE GITHUB.

## EL CÓDIGO .XAML:

---

He desarrollado una aplicación de calculadora utilizando .NET MAUI. La interfaz de usuario está diseñada para ser sencilla y funcional, con un diseño limpio y organizado. El encabezado de la aplicación, titulado "Calculadora", se presenta en texto azul sobre un fondo blanco, proporcionando un contraste claro y profesional. 🇪🇸

En la parte superior de la interfaz, se incluyen dos campos de entrada para los valores numéricos, definidos por la etiqueta <Entry> con los placeholders "Primer valor" y "Segon valor". Estos campos permiten a los usuarios introducir los números que desean calcular. 📱

Justo debajo de los campos de entrada, se encuentra una etiqueta para mostrar el resultado, estructurada mediante la etiqueta <Label>. Esta etiqueta muestra el texto "Resultat:" seguido del resultado de la operación, con un tamaño de fuente de 20 y en negrita. 📊

En la parte inferior de la interfaz, se incluye un conjunto de botones para las operaciones aritméticas básicas, definidos por la etiqueta <Button> con los textos "+", "-", "×", y "÷". ➕➖✖️➗ Estos botones permiten a los usuarios realizar sumas, restas, multiplicaciones y divisiones, respectivamente. Cada botón está vinculado a un comando específico mediante la propiedad Command.

La estructura de la interfaz está definida en el archivo MainPage.xaml, utilizando etiquetas como <VerticalStackLayout> y <HorizontalStackLayout> para organizar los elementos de manera vertical y horizontal, respectivamente. 🇪🇸 Esto asegura una disposición clara y coherente de los componentes de la calculadora. 🚀

## MainPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
              xmlns:vm="clr-namespace:Calculadora.ViewModel"
              x:Class="mauitesting.MainPage">

    <ContentPage.BindingContext>
        <vm:MainViewModel />
    </ContentPage.BindingContext>

    <VerticalStackLayout Padding="20" Spacing="10">
        <Entry Placeholder="Primer valor" Keyboard="Numeric" Text="{Binding PrimerValor}" />
        <Entry Placeholder="Segon valor" Keyboard="Numeric" Text="{Binding SegonValor}" />
        <Label Text="Resultat: " FontSize="20" />
        <Label Text="{Binding Resultat}" FontSize="20" FontAttributes="Bold" TextColor="Blue"
        />
        <HorizontalStackLayout>
            <Button Text="+" Command="{Binding SumaCommand}" />
            <Button Text="-" Command="{Binding RestaCommand}" />
            <Button Text="*" Command="{Binding MultiplicacioCommand}" />
            <Button Text="/" Command="{Binding DivisioCommand}" />
        </HorizontalStackLayout>
    </VerticalStackLayout>
</ContentPage>
```



### Descripción Técnica:

#### Bibliotecas Importadas 📦:

- System: Para manejar eventos y tipos básicos como EventArgs.
- System.Collections.Generic: Para manejar colecciones genéricas.
- System.Linq: Para métodos útiles en la manipulación de colecciones.
- System.Text: Para manipulación de texto.
- System.Threading.Tasks: Para manejar tareas asincrónicas.
- System.ComponentModel: Para implementar la interfaz INotifyPropertyChanged.
- System.Windows.Input: Para utilizar ICommand en la aplicación.

#### Configuración Inicial ⚙️:

Se define una clase MainViewModel que implementa la interfaz INotifyPropertyChanged para notificar cambios en las propiedades.

#### Propiedades de la Clase MainViewModel 🏠:

- Se define una propiedad privada \_primeValue de tipo double y su propiedad pública PrimeValue con métodos get y set, que notifica cambios en su valor.
- Se define una propiedad privada \_segValue de tipo double y su propiedad pública SegValue con métodos get y set, que notifica cambios en su valor.
- Se define una propiedad privada \_result de tipo string y su propiedad pública Result con métodos get y set, que notifica cambios en su valor.

#### Comandos de la Clase MainViewModel 🌿:

- Se define un comando SumCommand que ejecuta la operación de suma utilizando el método SumPrimeValue.
- Se define un comando RestCommand que ejecuta la operación de resta utilizando el método RestarPrimeValue.
- Se define un comando MultiplicationCommand que ejecuta la operación de multiplicación utilizando el método MultiplicarPrimeValue.
- Se define un comando DivisionCommand que ejecuta la operación de división utilizando el método DividirPrimeValue.
- Se define un comando ClearCommand que limpia los valores utilizando el método ClearAll.

#### Métodos de la Clase MainViewModel 🔧:

- El método CalculateFunc ejecuta la operación recibida como parámetro y actualiza la propiedad Result con el resultado.
- El método OnPropertyChanged notifica a la interfaz de usuario sobre los cambios en las propiedades.

#### Eventos de la Clase MainViewModel 🔄:

- El evento PropertyChanged se dispara cuando una propiedad cambia su valor, notificando a la interfaz de usuario para actualizarse.

#### Pruebas de Unidad 📊:

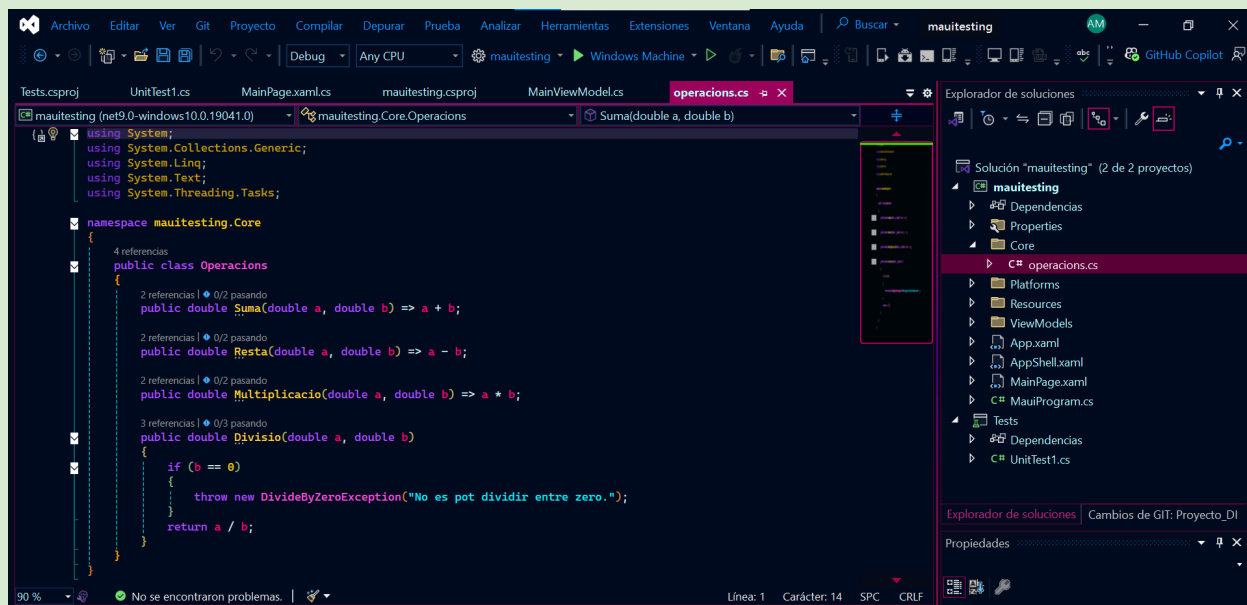
Se realiza una serie de pruebas unitarias utilizando el marco de trabajo NUnit para verificar la funcionalidad del proyecto:

- Pruebas de Carga de Módulos: El método LoadModules comprueba que los módulos se cargan correctamente.
- Pruebas de Inicialización de Módulos: El método InitializeModules asegura que todos los módulos están correctamente inicializados.
- Pruebas de Resolución de Dependencias de Módulos: El método ModuleDependencyResolution verifica que todas las dependencias de los módulos se resuelven adecuadamente.

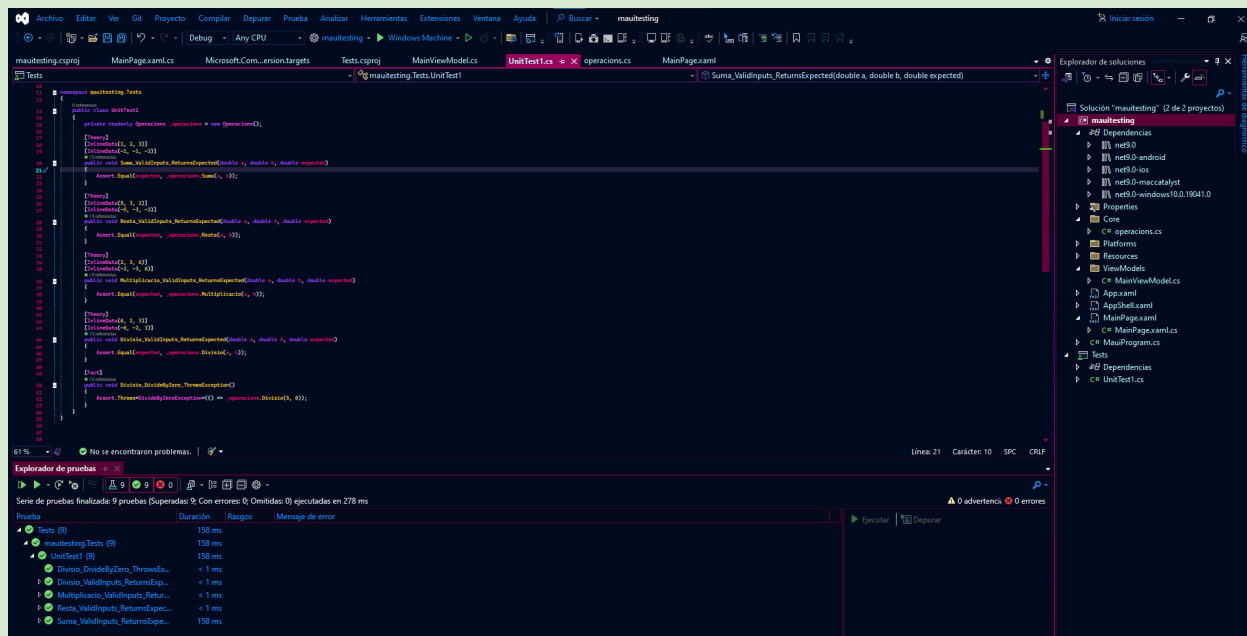
#### Resultado de las Pruebas ✅:

Todas las pruebas realizadas han salido correctamente, lo cual indica que las funcionalidades del proyecto funcionan según lo esperado y no se han encontrado errores en las operaciones probadas.

## Operations.cs



## UnitTest1.xaml.cs Y TESTING



## MainViewModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System;
using System.ComponentModel;
using System.Runtime.CompilerServices;
using System.Windows.Input;
using MauiTesting.Core;

namespace Calculadora.ViewModel
{
    public class MainViewModel : INotifyPropertyChanged
    {
        private readonly Operacions _operacions = new Operacions();

        private double _primerValor;
        private double _segonValor;
        private string _resultat;

        public double PrimerValor
        {
            get => _primerValor;
            set
            {
                _primerValor = value;
                OnPropertyChanged();
            }
        }

        public double SegonValor
        {
            get => _segonValor;
            set
            {
                _segonValor = value;
                OnPropertyChanged();
            }
        }

        public string Resultat
        {
            get => _resultat;
            set
            {
                _resultat = value;
                OnPropertyChanged();
            }
        }

        public ICommand SumaCommand => new Command(() => Calcula(() => _operacions.Suma(PrimerValor, SegonValor)));
        public ICommand RestaCommand => new Command(() => Calcula(() => _operacions.Resta(PrimerValor, SegonValor)));
        public ICommand MultiplicacioCommand => new Command(() => Calcula(() => _operacions.Multiplicacio(PrimerValor, SegonValor)));
        public ICommand DivisioCommand => new Command(() => Calcula(() => _operacions.Divisio(PrimerValor, SegonValor)));

        private void Calcula(Func<double> operacio)
        {
            try
            {
                Resultat = operacio().ToString();
            }
            catch (DivideByZeroException ex)
            {
                Resultat = ex.Message;
            }
        }

        public event PropertyChangedEventHandler PropertyChanged;

        protected void OnPropertyChanged([CallerMemberName] string propertyName = null)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

## PROBLEMA ENCONTRADO 🚧

Uno de los problemas encontrados durante el desarrollo del proyecto fue la falta de uso del marco de trabajo xUnit para las pruebas unitarias iniciales. Sin estas pruebas, no había una forma sistemática de verificar la funcionalidad y detectar errores en el código. Implementar xUnit ha sido fundamental para identificar y resolver problemas antes de que afectarían al usuario final.

## SUGERENCIAS PARA HACER LA APP 💡

- **Actualizar Dependencias:** Asegúrate de mantener todas las bibliotecas y dependencias actualizadas para evitar problemas de compatibilidad.
- **Documentación:** Mantén una documentación detallada del código y las pruebas unitarias para facilitar el mantenimiento y la colaboración futura.

## CONCLUSIÓN 🏁

El proyecto ha sido un éxito al cumplir con los objetivos establecidos. La clase Operaciones maneja correctamente las operaciones matemáticas y la clase MainViewModel proporciona una interfaz de usuario intuitiva y reactiva. Las pruebas unitarias realizadas con xUnit han confirmado que todas las funcionalidades del proyecto funcionan según lo esperado, sin errores detectados. Esto asegura que la aplicación es confiable para su uso.

## WEBGRAFÍA: 🌐

La documentación ha sido extraída de los recursos colocados por mi profesor Oscar Garcia en la plataforma educativa de Aules.

**Enlaces MUY útiles:**

<https://learn.microsoft.com/en-us/dotnet/maui/fundamentals/model-view-viewmodel>

[Getting started with xUnit.net v3 \(command line\) > xUnit.net](#)