

Unit Testing

 Visual Studio

Natalia Coronado Romero 2ºDAM

ÍNDICE

Introducción.....	3
Objetivo de la memoria.....	3
Material utilizado.....	3
GitHub.....	3
Desarrollo.....	4
Proyecto Calculadora.....	4
Interfaz Visual.....	4
-MainPage.xaml y MainPage.cs.....	5
-MainPageViewModel.cs.....	6
-Operations.cs.....	7
Testing.....	8
-UnitTest1.cs.....	8
Pruebas del testing.....	9
Problemas encontrados y sugerencias.....	10
Conclusión.....	10
Bibliografía/webgrafía.....	10
Video del funcionamiento.....	10

Introducción

Proyecto de una calculadora básica en .NET MAUI utilizando MVVM, junto con un segundo proyecto que realiza de testing sobre las operaciones matemáticas.

Objetivo de la memoria

El objetivo de esta práctica es diseñar una aplicación de una calculadora simple con MVVM complementándolo con un proyecto que realiza pruebas unitarias sobre este.

Material utilizado

Marca y Modelo del Procesador: Intel Pentium CPU G4400 @ 3.30GHz

Tipo Memoria RAM (memoria y slots): 4GB RAM, 2400 MHz **Tipo**

de dispositivo de almacenamiento capacidad (GiB): HDD 2TB

Programas utilizados: Spectacle, Visual Studio 2022, OBS Studio

GitHub

Es es el enlace de github al repositorio de la asignatura:

<https://github.com/MalvaLego/Desarrollo-de-Interfaces.git>

Desarrollo

Proyecto Calculadora Interfaz Visual

Esta es la vista previa al ejecutar la aplicación. Simula ser una calculadora simple por lo que se escribirán dos números y el usuario y el elige la operación a realizar.

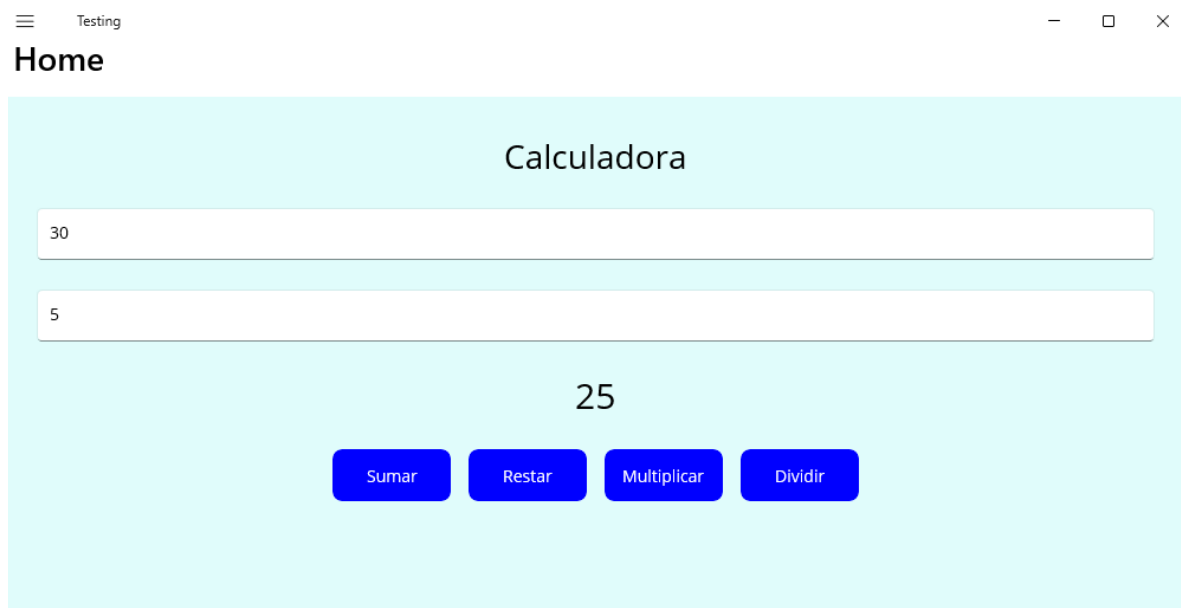
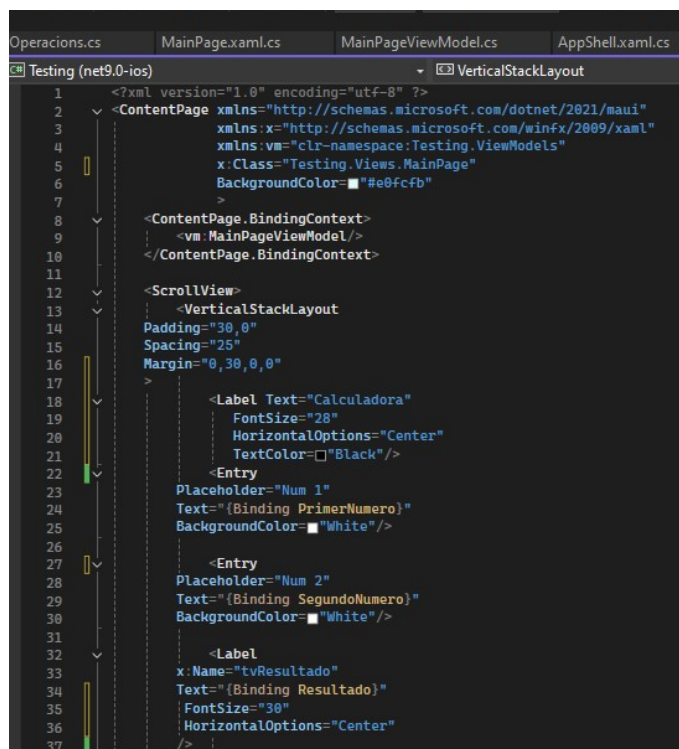


Figura 1: Vista previa del MainPage de la aplicación

-MainPage.xaml y MainPage.cs

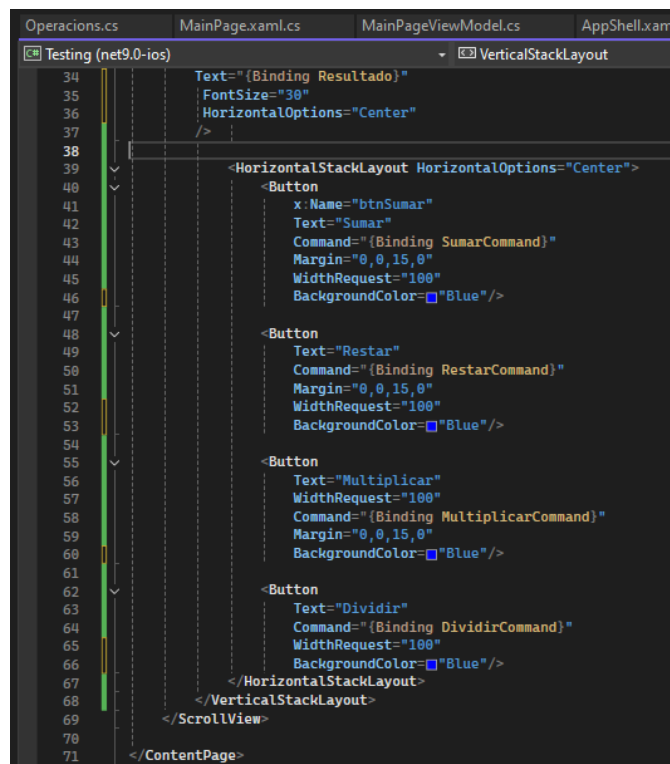
Este es el código que hará se muestre en pantalla lo que queramos. Contiene dos Entry donde se pondrán los números, un Label con el resultado y os cuatros botones para hacer las operaciones.

Se añade el nombre de la clase que tiene y la conexión de viewmodels con la carpeta ViewModel para el binding.



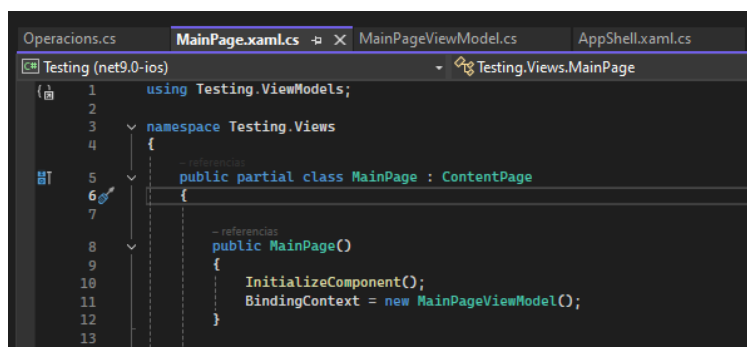
```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
3             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
4             xmlns:vm="clr-namespace:Testing.ViewModels"
5             x:Class="Testing.Views.MainPage"
6             BackgroundColor="#e0fcfb">
7
8     <ContentPage.BindingContext>
9         <vm:MainPageViewModel/>
10    </ContentPage.BindingContext>
11
12    <ScrollView>
13        <VerticalStackLayout
14            Padding="30,0"
15            Spacing="25"
16            Margin="0,30,0,0">
17
18            <Label Text="Calculadora"
19                FontSize="28"
20                HorizontalOptions="Center"
21                TextColor="Black"/>
22
23            <Entry
24                Placeholder="Num 1"
25                Text="{Binding PrimerNumero}"
26                BackgroundColor="White"/>
27
28            <Entry
29                Placeholder="Num 2"
30                Text="{Binding SegundoNumero}"
31                BackgroundColor="White"/>
32
33            <Label
34                x:Name="tvResultado"
35                Text="{Binding Resultado}"
36                FontSize="30"
37                HorizontalOptions="Center"
38            />
39
40        </VerticalStackLayout>
41    </ScrollView>
42</ContentPage>
```

Figura 3: Primera parte del código visual del MainPage: dos labels y dos entry



```
34 Text="{Binding Resultado}"
35 FontSize="30"
36 HorizontalOptions="Center"
37 />
38
39 <HorizontalStackLayout HorizontalOptions="Center">
40     <Button
41         x:Name="btnSumar"
42         Text="Sumar"
43         Command="{Binding SumarCommand}"
44         Margin="0,0,15,0"
45         WidthRequest="100"
46         BackgroundColor="Blue"/>
47
48     <Button
49         Text="Restar"
50         Command="{Binding RestarCommand}"
51         Margin="0,0,15,0"
52         WidthRequest="100"
53         BackgroundColor="Blue"/>
54
55     <Button
56         Text="Multiplicar"
57         WidthRequest="100"
58         Command="{Binding MultiplicarCommand}"
59         Margin="0,0,15,0"
60         BackgroundColor="Blue"/>
61
62     <Button
63         Text="Dividir"
64         Command="{Binding DividirCommand}"
65         WidthRequest="100"
66         BackgroundColor="Blue"/>
67 </HorizontalStackLayout>
68 </VerticalStackLayout>
69 </ScrollView>
70 </ContentPage>
```

Figura 2: Segunda parte del código visual del MainPage: cuatros botones de operaciones



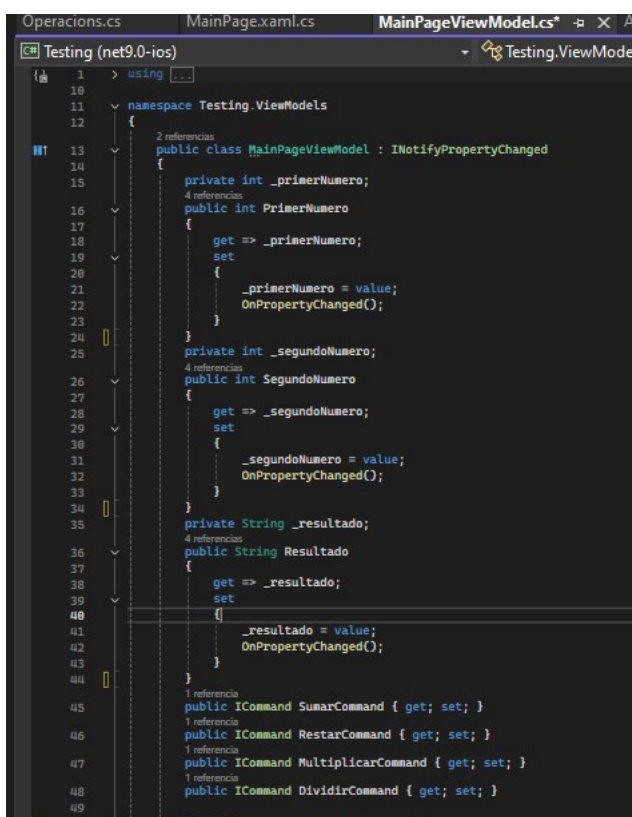
```
1 using Testing.ViewModels;
2
3 namespace Testing.Views
4 {
5     public partial class MainPage : ContentPage
6     {
7
8         ~referencias
9         public MainPage()
10         {
11             InitializeComponent();
12             BindingContext = new MainPageViewModel();
13         }
14     }
15 }
```

Figura 4: Código del MainPage: solo funcional para el binding

-MainPageViewModel.cs

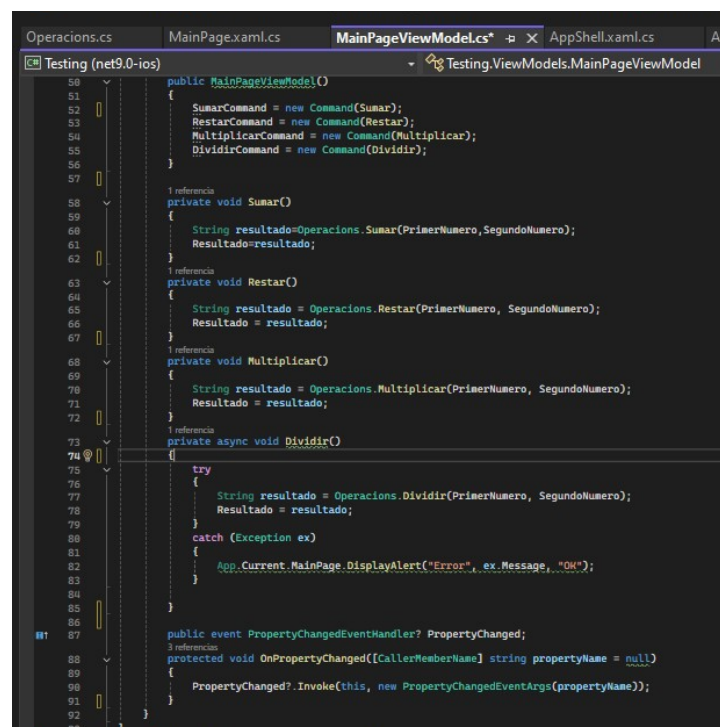
Esta clase sigue el patrón MVVM y utiliza INotifyPropertyChanged para actualizar la interfaz automáticamente cuando cambian sus propiedades, almacenando los valores ingresados y el resultado de la operación.

Se utilizan ICommand para definir los botones de suma, resta, multiplicación y división. Al pulsarlos, llaman respectivas funciones de la clase Operacions y devuelve el resultado. En Dividir(), se usa un try-catch para capturar la excepción de división por 0 y mostrar un DisplayAlert en la interfaz. Esto evita que la app se detenga por errores.



```
1  > using ...
10
11 namespace Testing.ViewModels
12 {
13     2 referencias
14     public class MainPageViewModel : INotifyPropertyChanged
15     {
16         private int _primerNumero;
17         4 referencias
18         public int PrimerNumero
19         {
20             get => _primerNumero;
21             set
22             {
23                 _primerNumero = value;
24                 OnPropertyChanged();
25             }
26         }
27         private int _segundoNumero;
28         4 referencias
29         public int SegundoNumero
30         {
31             get => _segundoNumero;
32             set
33             {
34                 _segundoNumero = value;
35                 OnPropertyChanged();
36             }
37         }
38         private String _resultado;
39         4 referencias
40         public String Resultado
41         {
42             get => _resultado;
43             set
44             {
45                 _resultado = value;
46                 OnPropertyChanged();
47             }
48         }
49         1 referencia
50         public ICommand SumarCommand { get; set; }
51         1 referencia
52         public ICommand RestarCommand { get; set; }
53         1 referencia
54         public ICommand MultiplicarCommand { get; set; }
55         1 referencia
56         public ICommand DividirCommand { get; set; }
```

Figura 5: Primera parte del código de MainPageViewModel: Iniciación de propiedades, ICommand



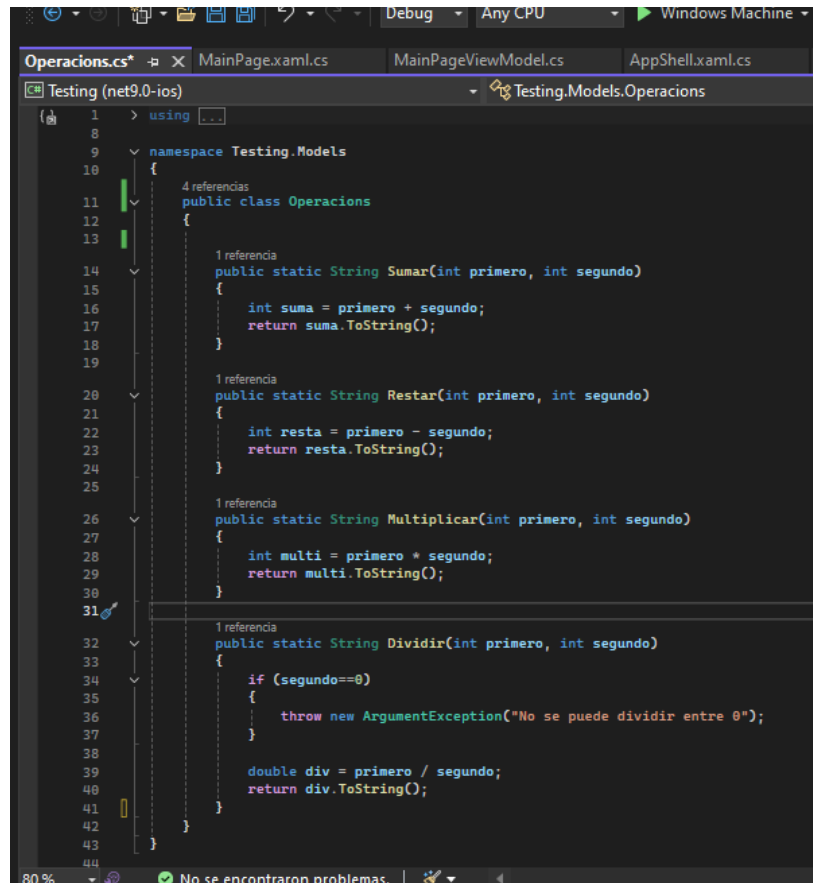
```
50
51
52     public MainPageViewModel()
53     {
54         SumarCommand = new Command(Sumar);
55         RestarCommand = new Command(Restar);
56         MultiplicarCommand = new Command(Multiplicar);
57         DividirCommand = new Command(Dividir);
58     }
59
60     1 referencia
61     private void Sumar()
62     {
63         String resultado = Operacions.Sumar(PrimerNumero, SegundoNumero);
64         Resultado = resultado;
65     }
66
67     1 referencia
68     private void Restar()
69     {
70         String resultado = Operacions.Restar(PrimerNumero, SegundoNumero);
71         Resultado = resultado;
72     }
73
74     1 referencia
75     private void Multiplicar()
76     {
77         String resultado = Operacions.Multiplicar(PrimerNumero, SegundoNumero);
78         Resultado = resultado;
79     }
80
81     1 referencia
82     private async void Dividir()
83     {
84         try
85         {
86             String resultado = Operacions.Dividir(PrimerNumero, SegundoNumero);
87             Resultado = resultado;
88         }
89         catch (Exception ex)
90         {
91             App.Current.MainPage.DisplayAlert("Error", ex.Message, "OK");
92         }
93     }
94
95     public event PropertyChangedEventHandler? PropertyChanged;
96     3 referencias
97     protected void OnPropertyChanged([CallerMemberName] string propertyName = null)
98     {
99         PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
100     }
```

Figura 6: Segunda parte del código de MainPageViewModel: Constructor, funciones de las operaciones, PropertyChanged

-Operacions.cs

La clase Operacions define métodos para sumar, restar, multiplicar y dividir dos números enteros, devolviendo el resultado como string.

Los métodos son estáticos, lo que permite llamarlos sin tener que instanciar la clase. En Dividir(), se captura la excepción de división por 0, lanzando un mensaje de error.



```
1  > using ...
2
3
4
5
6
7
8
9  namespace Testing.Models
10 {
11     4 referencias
12     public class Operacions
13     {
14         1 referencia
15         public static String Sumar(int primero, int segundo)
16         {
17             int suma = primero + segundo;
18             return suma.ToString();
19         }
20
21         1 referencia
22         public static String Restar(int primero, int segundo)
23         {
24             int resta = primero - segundo;
25             return resta.ToString();
26         }
27
28         1 referencia
29         public static String Multiplicar(int primero, int segundo)
30         {
31             int multi = primero * segundo;
32             return multi.ToString();
33         }
34
35         1 referencia
36         public static String Dividir(int primero, int segundo)
37         {
38             if (segundo==0)
39             {
40                 throw new ArgumentException("No se puede dividir entre 0");
41             }
42
43             double div = primero / segundo;
44             return div.ToString();
45         }
46     }
47 }
```

Figura 7: Código de Operacions:cuatro funciones para cada operación

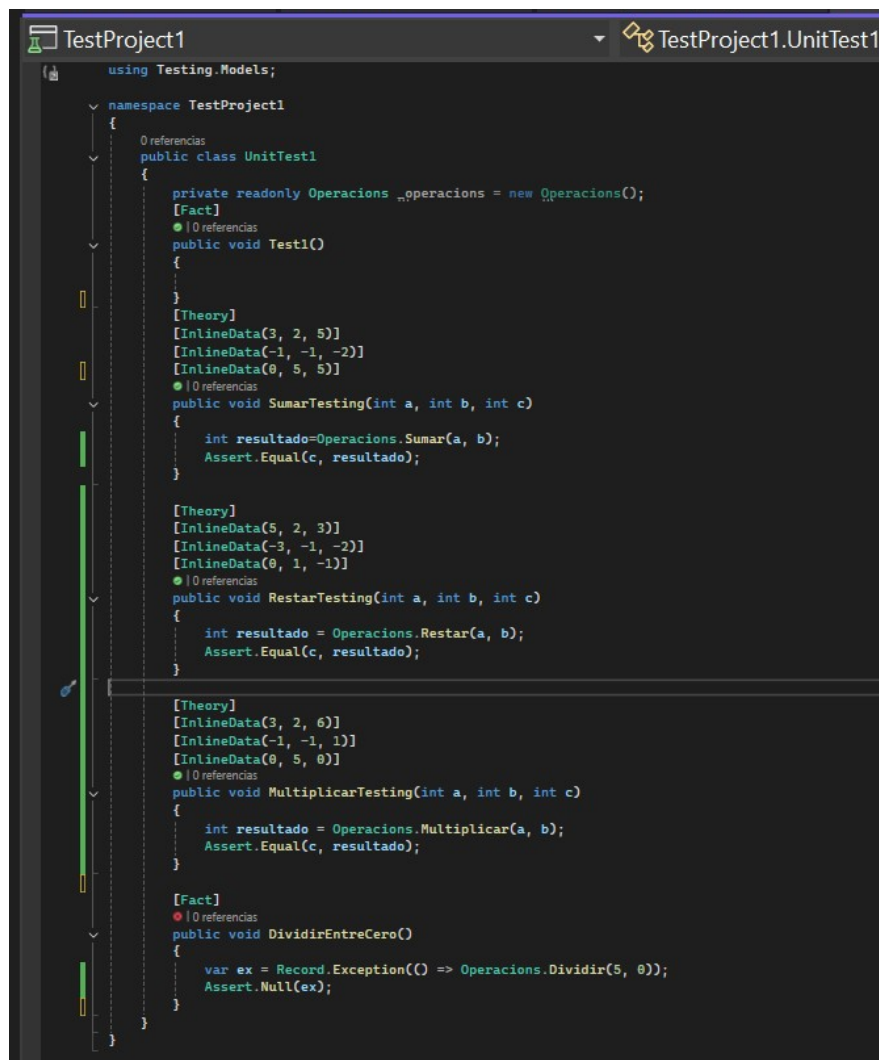
Testing

-UnitTest1.cs

Para hacer el testing del proyecto que acabamos de realizar es necesario crear uno nuevo en la misma carpeta pero esta vez utilizando xUnit, el cual lo implementaremos con la calculadora.

Al crear este nuevo proyecto se generará un archivo .cs donde se pondrán las pruebas unitarias para la clase **Operacions**, verificando que sus métodos de suma, resta, multiplicación y división funcionen correctamente.

La suma, resta y multiplicación devuelven los resultados esperados según los datos puestos con **InlineData**, en cambio la división comprueba que al dividir entre 0 lance una excepción.



```
using Testing.Models;

namespace TestProject1
{
    0 referencias
    public class UnitTest1
    {
        private readonly Operacions _operacions = new Operacions();

        [Fact]
        0 referencias
        public void Test1()
        {
        }

        [Theory]
        [InlineData(3, 2, 5)]
        [InlineData(-1, -1, -2)]
        [InlineData(0, 5, 5)]
        0 referencias
        public void SumarTesting(int a, int b, int c)
        {
            int resultado = Operacions.Sumar(a, b);
            Assert.Equal(c, resultado);
        }

        [Theory]
        [InlineData(5, 2, 3)]
        [InlineData(-3, -1, -2)]
        [InlineData(0, 1, -1)]
        0 referencias
        public void RestarTesting(int a, int b, int c)
        {
            int resultado = Operacions.Restar(a, b);
            Assert.Equal(c, resultado);
        }

        [Theory]
        [InlineData(3, 2, 6)]
        [InlineData(-1, -1, 1)]
        [InlineData(0, 5, 0)]
        0 referencias
        public void MultiplicarTesting(int a, int b, int c)
        {
            int resultado = Operacions.Multiplicar(a, b);
            Assert.Equal(c, resultado);
        }

        [Fact]
        0 referencias
        public void DividirEntreCero()
        {
            var ex = Record.Exception(() => Operacions.Dividir(5, 0));
            Assert.Null(ex);
        }
    }
}
```

Figura 8: Código de UnitTest1: método de prueba por cada operación de Operacions.cs

Pruebas del testing

Este es el resultado al ejecutar las pruebas, todos pasan correctamente menos la división entre cero que lanza una excepción y la captura.

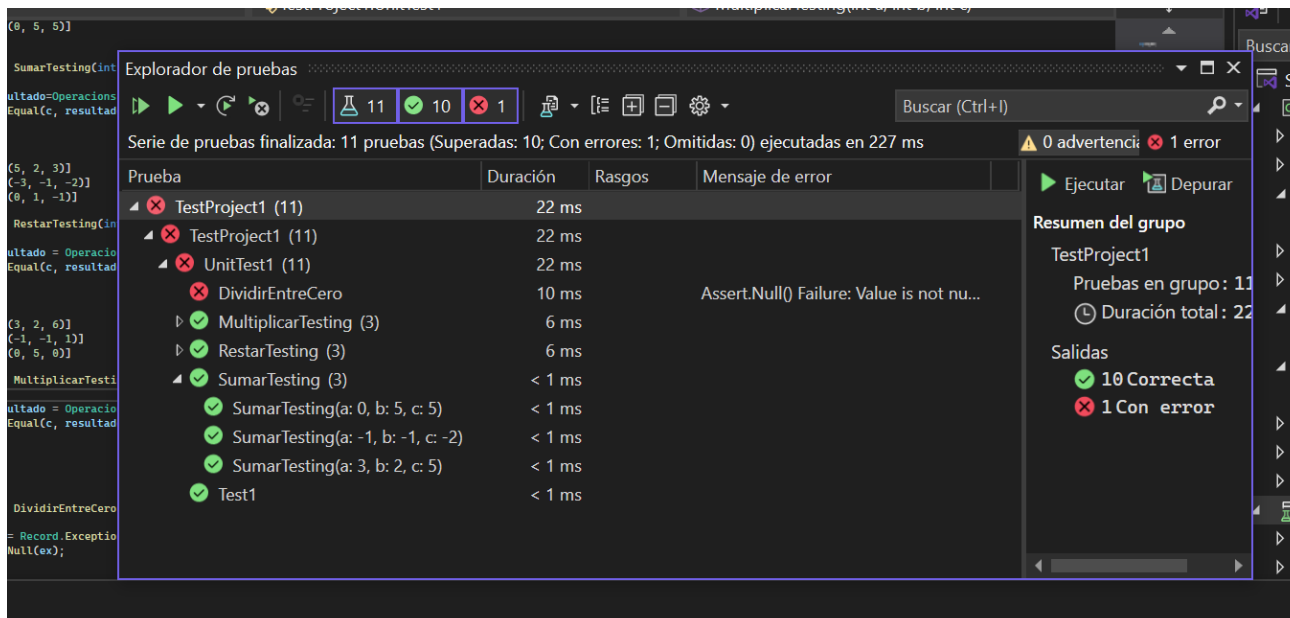


Figura 9: Resultado del testing

Problemas encontrados y sugerencias

Ha habido distintos errores al igual que siempre, después de ir probando, hacer otras veces el proyecto y ayudas externas para conseguir información se ha conseguido.

Conclusión

Buena actividad para repasar el patrón MVVM y aprender a realizar pruebas unitarias para garantizar la fiabilidad del código.

Bibliografía/webgrafía

_.Pdfs del profesor de mi profesor.

Video del funcionamiento



Dibujo 1: <https://youtu.be/YMAxY-pnaz0>