Exercici_UD04_part2.Testing

TESTEANDO
MAUI

Sumario

Introducción:.....4
Objetivos:.....4
Herramientas utilizadas:.....4
Actividades:.....4
..... 11
Problemas encontrados:.....13
Conclusión:.....13
Bibliografía:.....13

Introducción:

Esta actividad consiste en desarrollar una aplicación que sea una calculadora y un testing de la propia calculadora .

Objetivos:

El objetivo de esta práctica terminar de dominar el uso de los Icommand y mientras realizaos la aplicacion/testing a la vez que aprendemos como funciona el testing de maui.

Herramientas utilizadas:

La documentacion dada.
Internet.

Wifi.

EL ordenador del centro.

Libre office.

Actividades:

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.Models"
             x:Class="Calculadora.MainPage">

    <ContentPage.BindingContext>
        <vm:Operaciones/>
    </ContentPage.BindingContext>
    <ScrollView>
        <VerticalStackLayout
            Padding="30,0"
            Spacing="25">

            <HorizontalStackLayout HorizontalOptions="Center" >
                <Entry Text="{Binding Numero1}" />
                <Label Text="{Binding Operacion}" Margin="10"/>
```

Como utiliaremos Icommands necesitaremos vincular el main page a un view model que en nuestro caso sera “Operacions.cs”.

```
            <Entry Text="{Binding Numero2}"/>
            <Label Text="{Binding Resultado} " Margin="10" />
        </HorizontalStackLayout>

        <HorizontalStackLayout HorizontalOptions="Center" >
            <Button Text = "+" Margin="5" x:Name="sumar" Command="{Binding SumarCommand}"/>
            <Button Text = "-"  Margin="5" x:Name="restar" Command="{Binding RestarCommand}"/>
            <Button Text = "*"  Margin="5" x:Name="multiplicar" Command="{Binding MultiplicarCommand}"/>
            <Button Text = "/"  Margin="5" x:Name="dividir" Command="{Binding DividirCommand}"/>
        </HorizontalStackLayout>

    </VerticalStackLayout>
</ScrollView>

</ContentPage>
```

Casteamos los botones y los entrys con sus binding que pertenecen a su Icommand.

Operaciones.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;

namespace Calculadora.Models
{
    5 referencias
    public class Operaciones:INotifyPropertyChanged
    {
        2 referencias | 1/1 pasando
        public ICommand SumarCommand { get; set; }
        2 referencias | 1/1 pasando
        public ICommand RestarCommand { get; set; }
        2 referencias | 1/1 pasando
        public ICommand MultiplicarCommand { get; set; }
        3 referencias | 2/2 pasando
        public ICommand DividirCommand { get; set; }

        private string _numero1;
        6 referencias | 5/5 pasando
        public string Numero1
        {
            get => _numero1;
            set
            {
                _numero1 = value;
                int.TryParse(Numero1, out numerito1);
                OnPropertyChanged();
            }
        }
        private string _numero2;
        6 referencias | 5/5 pasando
        public string Numero2
        {
            get => _numero2;
            set
            {
                _numero2 = value;
```

Se puede ver uso del constructor de los commands anteriormente meniconados en el main .

```
        get => _numero2;
        set
        {
            _numero2 = value;
            int.TryParse(Numero2, out numerito2);
            OnPropertyChanged();
        }
    }
    private string _operacion;
    4 referencias
    public string Operacion
    {
        get => _operacion;
        set
        {
            _operacion = value;
            OnPropertyChanged();
        }
    }

    public int numerito1;
    public int numerito2;

    private string _resultado;
    8 referencias | 4/4 pasando
    public string Resultado
    {
        get => _resultado;
        set
        {
            _resultado = value;
            OnPropertyChanged();
        }
    }

    public event PropertyChangedEventHandler? PropertyChanged;
```

Los constructores que faltan junto a su propertyChanged.

```
2 referencias
public Operaciones()
{
    SumarCommand = new Command(Sumar);
    RestarCommand = new Command(Restar);
    MultiplicarCommand = new Command(Multiplicar);
    DividirCommand = new Command(Dividir);
}
```

En el constructor del cs utilizamos los command.

```
1 referencia
private void Sumar() {
    Operacion = "+";
    Resultado = " = " + (numerito1 + numerito2) + "";
    OnPropertyChanged();
}
1 referencia
private void Restar()
{
    Operacion = "-";
    Resultado = " = " + (numerito1 - numerito2) + "";
    OnPropertyChanged();
}
1 referencia
private void Multiplicar()
{
    Operacion = "x";
    Resultado = " = " + numerito1 * numerito2 + "";
    OnPropertyChanged();
}
1 referencia
private  void Dividir()
{
    Operacion = "/";
    if(numerito2 != 0)
    {
        Resultado = " = " + numerito1 / numerito2;
        OnPropertyChanged();
    }
    else
    {
        MessagingCenter.Send(this, "DividirPorCero", "No se puede dividir entre 0");
    }

}
```

Lo que hacen los Icommand.

```
8 referencias
protected virtual void OnPropertyChanged(string propertyName = null)
{
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}
```

El evento que mas utilizamos “OnPropertyChanged”.

MainPage.xaml.cs

```
namespace Calculadora
{
    5 referencias
    public partial class MainPage : ContentPage
    {
        0 referencias
        public MainPage()
        {
            InitializeComponent();

            BindingContext = new Operaciones();

            // Escuchar el mensaje "DividirPorCero" desde cualquier parte de la app
            MessagingCenter.Subscribe<Operaciones, string>(this, "DividirPorCero", async (sender, message) =>
            {
                // Mostrar la alerta cuando se recibe el mensaje
                await DisplayAlert("Error", message, "OK");
            });


        }

    }
}
```

El cs lo utilizamos unicamente porque es el unico lugar desde el que podemos hacer un MessagingCenter.

Calculadora.proj

```
<Project Sdk="Microsoft.NET.Sdk">

    <PropertyGroup>
        <TargetFrameworks>net8.0;net8.0-android;net8.0-ios;net8.0-maccatalyst</TargetFrameworks>
        <TargetFrameworks Condition="$([MSBuild]::IsOSPlatform('windows'))">$(TargetFrameworks);net8.0-windows10.0.19041.0</T
        <!-- Uncomment to also build the tizen app. You will need to install tizen by following this: https://github.com/Sams
        <!-- <TargetFrameworks>$(TargetFrameworks);net8.0-tizen</TargetFrameworks> -->

        <!-- Note for MacCatalyst:
        The default runtime is maccatalyst-x64, except in Release config, in which case the default is maccatalyst-x64;maccat
        When specifying both architectures, use the plural <RuntimeIdentifiers> instead of the singular <RuntimeIdentifier>.
        The Mac App Store will NOT accept apps with ONLY maccatalyst-arm64 indicated;
        either BOTH runtimes must be indicated or ONLY macatalyst-x64. -->
        <!-- For example: <RuntimeIdentifiers>maccatalyst-x64;maccatalyst-arm64</RuntimeIdentifiers> -->

        <OutputType Condition="$(TargetFramework) != 'net8.0'">Exe</OutputType>
        <RootNamespace>Calculadora</RootNamespace>
        <UseMaui>true</UseMaui>
        <SingleProject>true</SingleProject>
        <ImplicitUsings>enable</ImplicitUsings>
        <Nullable>enable</Nullable>

        <!-- Display name -->
        <ApplicationTitle>Calculadora</ApplicationTitle>

        <!-- App Identifier -->
        <ApplicationId>com.companyname.calculadora</ApplicationId>

        <!-- Versions -->
        <ApplicationDisplayVersion>1.0</ApplicationDisplayVersion>
        <ApplicationVersion>1</ApplicationVersion>

        <SupportedOSPlatformVersion Condition="$([MSBuild]::GetTargetPlatformIdentifier('$(TargetFramework)')) == 'ios'">11.0
```

El proj que lo hemos tenido que modificar para poder hacer el unitTest.

xUnitTest:

```
using Calculadora.Models;
using Xunit;

namespace TestProject4
{
    1 referencia
    public class UnitTest1
    {
        private readonly Operaciones _operacion;

        0 referencias
        public UnitTest1()
        {
            _operacion = new Operaciones();
        }

        [Fact]
        | 0 referencias
        public void SumarTest()
        {
            _operacion.Numero1 = "5";
            _operacion.Numero2 = "3";

            _operacion.SumarCommand.Execute(null);

            Assert.Equal(" = 8", _operacion.Resultado);
        }

        [Fact]
```

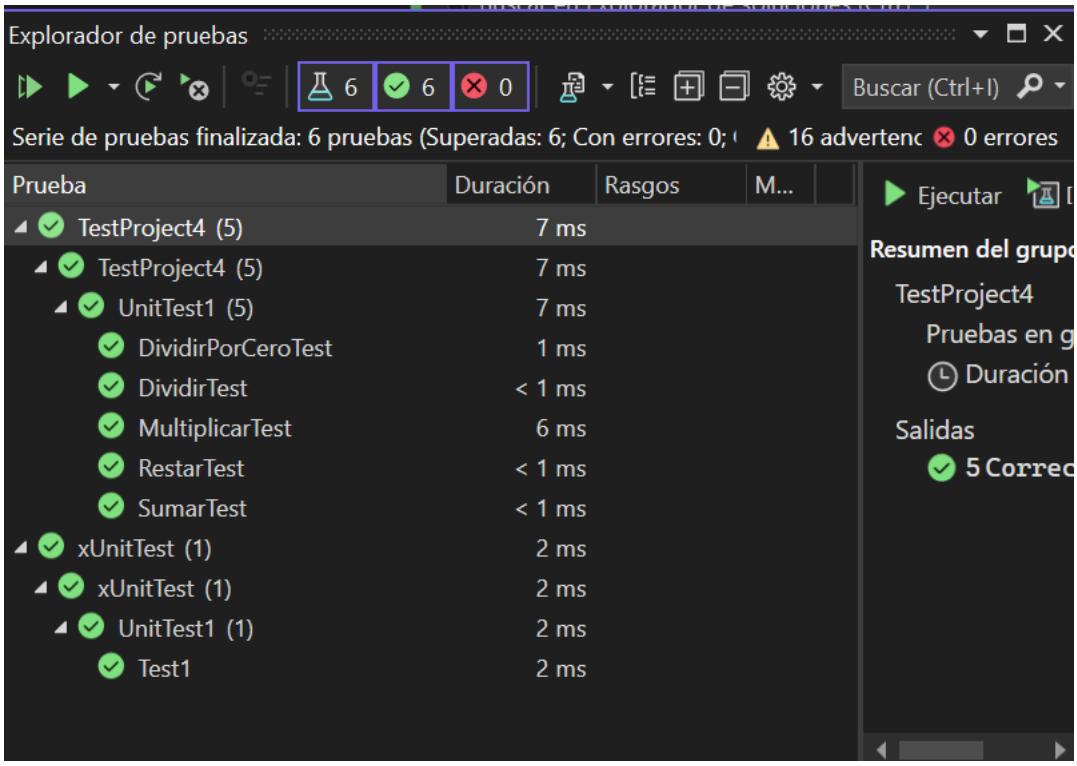
Las pruebas que hacemos del unit test para ver si funcionan bien los ICommand que pertenecen a “Operaciones.cs”.

```
[Fact]
0 referencias
public void RestarTest()
{
    _operacion.Numero1 = "10";
    _operacion.Numero2 = "4";

    _operacion.RestarCommand.Execute(null);

    Assert.Equal(" = 6", _operacion.Resultado);
}

[Fact]
0 referencias
public void MultiplicarTest()
{
    _operacion.Numero1 = "4";
    _operacion.Numero2 = "2";

    _operacion.MultiplicarCommand.Execute(null);

    Assert.Equal(" = 8", _operacion.Resultado);
}
```

Como comprobamos el RestarTest y el MultiplicarTest.

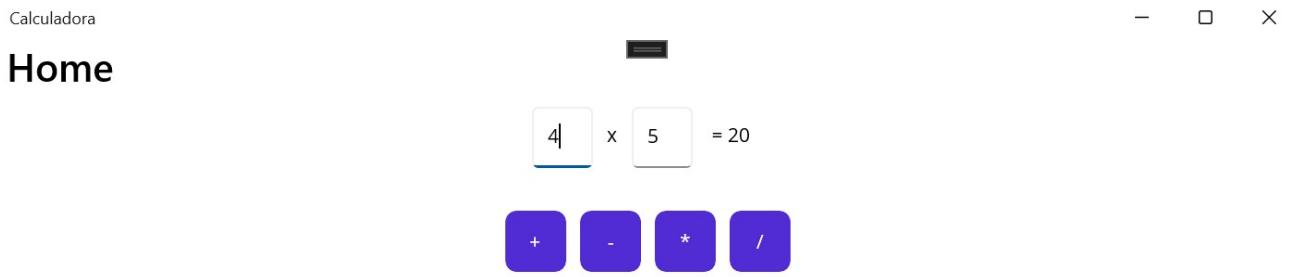
```
0 referencias
public void DividirTest()
{
    _operacion.Numero1 = "8";
    _operacion.Numero2 = "4";

    _operacion.DividirCommand.Execute(null);

    Assert.Equal(" = 2", _operacion.Resultado);
}

[Fact]
0 referencias
public void DividirPorCeroTest()
{
    _operacion.Numero1 = "8";
    _operacion.Numero2 = "0";

    // Simulando el comportamiento de dividir por cero, ya que estamos usando MessagingCenter
    _operacion.DividirCommand.Execute(null);

    // Si se realiza algún tipo de evento, puedes verificar si se llama.
    // En este caso no es sencillo de probar sin un mock o algo que simule el MessagingCenter.
    // Aquí solo estamos verificando que no se lance una excepción.
}
```

Como comprobamos el dividir y también comprobamos cuando se manda el display.message.

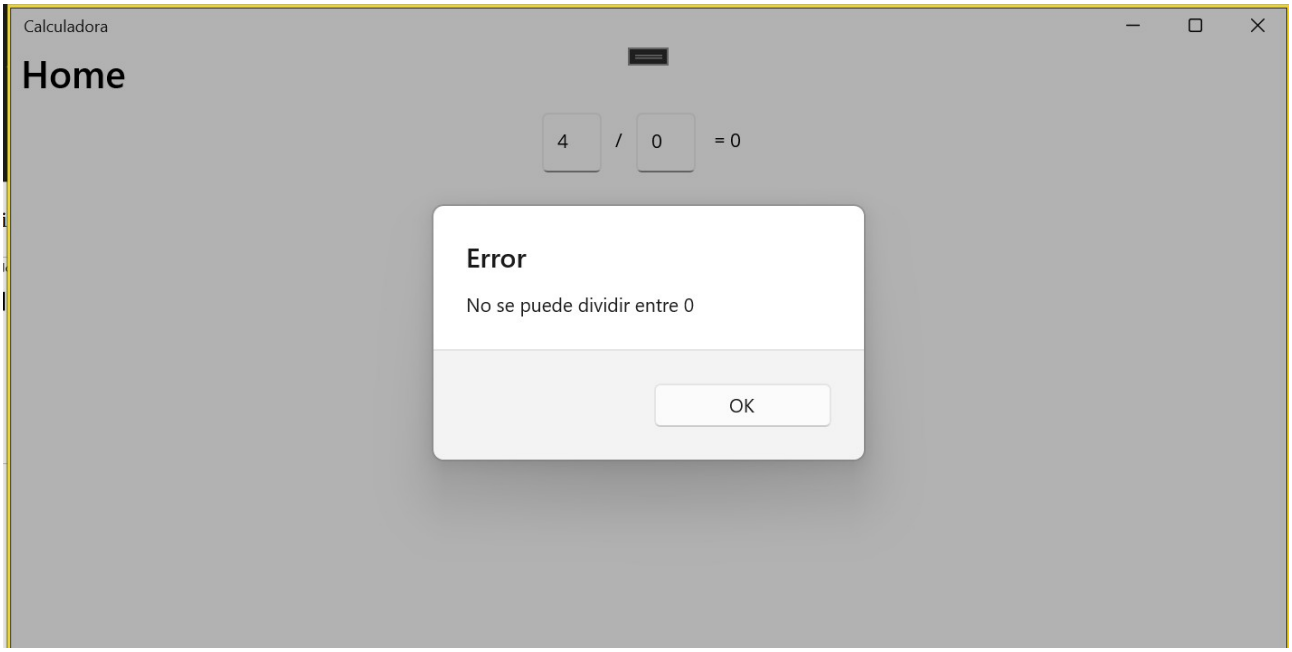


Como funciona el testing en la consola.

Vista de la aplicacion



Como funciona la aplicación desde la vista del cliente.



Como se ve el display.message.

Problemas encontrados:

He tenido problemas con el uso de la ventana emergente porque no encontraba la manera de usarlo desde “Operaciones.cs”.

Conclusión:

He podido solucionar el uso del display.message a partir de mandar una notificación desde “Operaciones” a el cs del mainpage y que se active la ventana.

Bibliografía:

visual studio 2022 maui - Búsqueda Imágenes

DI_ud02_ex01.pdf

Desenvolupament d'interfícies-2CFS.DAM 1726133660: Exercici_UD01_partI | AULES

Github:

OdooSge/Exercici_UD04_part2_Testing_IzanLopez at main · izancluac/OdooSge