Divisas 2 MVVM - #NoCodeBehind

1. Cree un proyecto Xamarin Forms, llamado Divisas y asegúrese que el “**Welcome to Xamarin forms**” le esté funcionando en las 3 plataformas:

|  |  |  |
| --- | --- | --- |
| Android  C:\Users\Usuario\Desktop\Nexus 5 (Lollipop) Screenshot 1.png | iOS  C:\Users\Usuario\Downloads\Simulator Screen Shot 2.02.2017, 1.40.05 p.m..png | UWP |

1. Cree la carpeta **ViewModels** y dentro de esta carpeta cree la clase **MainViewModel**:

public class MainViewModel  
{  
    public MainViewModel()  
    {  
    }  
}

1. Cree la carpeta **Infrastructure** y dentro de esta la clase **InstanceLocator**:

public class InstanceLocator  
{  
    public MainViewModel Main { get; set; }  
  
    public InstanceLocator()  
    {  
        Main = new MainViewModel();  
    }  
}

1. Agregue estas líneas al diccionario de recursos (no olvide cambiar Borrame por el nombre de su proyecto):

<?xml version="1.0" encoding="utf-8"?>  
<Application   
    xmlns="http://xamarin.com/schemas/2014/forms"   
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"   
    xmlns:infra="clr-namespace:Borrame.Infrastructure;assembly=Borrame"   
    x:Class="Borrame.App">  
    <Application.Resources>  
        <ResourceDictionary>  
            <!-- Locator -->  
            <infra:InstanceLocator x:Key="Locator"/>  
        </ResourceDictionary>  
    </Application.Resources>  
</Application>

1. Vamos a crear la carpeta **Views** y dentro de esta vamos a crear el **MainPage**:

<?xml version="1.0" encoding="UTF-8"?>  
<ContentPage   
    xmlns="http://xamarin.com/schemas/2014/forms"   
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"   
    x:Class="Divisas2Prep.Views.MainPage"   
    Title="Divisas"   
    BindingContext="{Binding Main, Source={StaticResource Locator}}">  
    <ContentPage.Content>  
        <ScrollView>  
            <StackLayout  
                Padding="8">  
                <Label  
                    TextColor="Black"  
                    FontSize="Large"  
                    Text="Main Page">  
                </Label>  
            </StackLayout>  
        </ScrollView>  
    </ContentPage.Content>  
</ContentPage>

1. Cambiamos el **App.xaml.cs** por (ya puedes borrar la Page que te crea por defecto con el “Welcome To Xamarin Forms!”):

public App()

{

InitializeComponent();

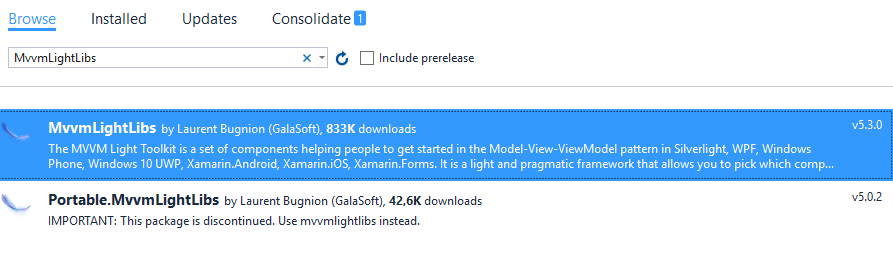
MainPage = new NavigationPage(new MainPage());

}

Y probamos que nuestro proyecto aun funcione:

|  |  |  |
| --- | --- | --- |
| Android  D:\Users\Zulu\Desktop\Nexus 4 (Marshmallow). Screenshot 1.png | iOS  C:\Users\Usuario\Downloads\Simulator Screen Shot 2.02.2017, 2.02.31 p.m..png | UWP |

1. Agregamos los Nugets de: **Microsoft.Net.Http**, **Newtonsoft.Json** y **MvvmLightLibs** al proyecto compartido.



1. Como vamos a usar el servicio: <https://openexchangerates.org/api/latest.json?app_id=f490efbcd52d48ee98fd62cf33c47b9e> para obtener las tasas de cambio. Creemos un objeto donde mapearemos la respuesta:

**Nota**: los códigos de moneda internacionales los podemos consultar en: <https://es.iban.com/currency-codes.html>

1. Creamos la carpeta **Models** y dentro de esta carpeta la clase **ExchangeRates** con el siguiente código (nos apoyamos en la página: <http://json2csharp.com/>):

using Newtonsoft.Json;

namespace Divisas2MVVM.Classes

{

public class ExchangeRates

{

[JsonProperty(PropertyName = "disclaimer")]

public string Disclaimer { get; set; }

[JsonProperty(PropertyName = "license")]

public string License { get; set; }

[JsonProperty(PropertyName = "timestamp")]

public int TimeStamp { get; set; }

[JsonProperty(PropertyName = "base")]

public string Base { get; set; }

[JsonProperty(PropertyName = "rates")]

public Rates Rates { get; set; }

}

public class Rates

{

public double AED { get; set; }

public double AFN { get; set; }

public double ALL { get; set; }

public double AMD { get; set; }

public double ANG { get; set; }

public double AOA { get; set; }

public double ARS { get; set; }

public double AUD { get; set; }

public double AWG { get; set; }

public double AZN { get; set; }

public double BAM { get; set; }

public double BBD { get; set; }

public double BDT { get; set; }

public double BGN { get; set; }

public double BHD { get; set; }

public double BIF { get; set; }

public double BMD { get; set; }

public double BND { get; set; }

public double BOB { get; set; }

public double BRL { get; set; }

public double BSD { get; set; }

public double BTC { get; set; }

public double BTN { get; set; }

public double BWP { get; set; }

public double BYN { get; set; }

public double BYR { get; set; }

public double BZD { get; set; }

public double CAD { get; set; }

public double CDF { get; set; }

public double CHF { get; set; }

public double CLF { get; set; }

public double CLP { get; set; }

public double CNY { get; set; }

public double COP { get; set; }

public double CRC { get; set; }

public double CUC { get; set; }

public double CUP { get; set; }

public double CVE { get; set; }

public double CZK { get; set; }

public double DJF { get; set; }

public double DKK { get; set; }

public double DOP { get; set; }

public double DZD { get; set; }

public double EEK { get; set; }

public double EGP { get; set; }

public double ERN { get; set; }

public double ETB { get; set; }

public double EUR { get; set; }

public double FJD { get; set; }

public double FKP { get; set; }

public double GBP { get; set; }

public double GEL { get; set; }

public double GGP { get; set; }

public double GHS { get; set; }

public double GIP { get; set; }

public double GMD { get; set; }

public double GNF { get; set; }

public double GTQ { get; set; }

public double GYD { get; set; }

public double HKD { get; set; }

public double HNL { get; set; }

public double HRK { get; set; }

public double HTG { get; set; }

public double HUF { get; set; }

public double IDR { get; set; }

public double ILS { get; set; }

public double IMP { get; set; }

public double INR { get; set; }

public double IQD { get; set; }

public double IRR { get; set; }

public double ISK { get; set; }

public double JEP { get; set; }

public double JMD { get; set; }

public double JOD { get; set; }

public double JPY { get; set; }

public double KES { get; set; }

public double KGS { get; set; }

public double KHR { get; set; }

public double KMF { get; set; }

public double KPW { get; set; }

public double KRW { get; set; }

public double KWD { get; set; }

public double KYD { get; set; }

public double KZT { get; set; }

public double LAK { get; set; }

public double LBP { get; set; }

public double LKR { get; set; }

public double LRD { get; set; }

public double LSL { get; set; }

public double LTL { get; set; }

public double LVL { get; set; }

public double LYD { get; set; }

public double MAD { get; set; }

public double MDL { get; set; }

public double MGA { get; set; }

public double MKD { get; set; }

public double MMK { get; set; }

public double MNT { get; set; }

public double MOP { get; set; }

public double MRO { get; set; }

public double MTL { get; set; }

public double MUR { get; set; }

public double MVR { get; set; }

public double MWK { get; set; }

public double MXN { get; set; }

public double MYR { get; set; }

public double MZN { get; set; }

public double NAD { get; set; }

public double NGN { get; set; }

public double NIO { get; set; }

public double NOK { get; set; }

public double NPR { get; set; }

public double NZD { get; set; }

public double OMR { get; set; }

public double PAB { get; set; }

public double PEN { get; set; }

public double PGK { get; set; }

public double PHP { get; set; }

public double PKR { get; set; }

public double PLN { get; set; }

public double PYG { get; set; }

public double QAR { get; set; }

public double RON { get; set; }

public double RSD { get; set; }

public double RUB { get; set; }

public double RWF { get; set; }

public double SAR { get; set; }

public double SBD { get; set; }

public double SCR { get; set; }

public double SDG { get; set; }

public double SEK { get; set; }

public double SGD { get; set; }

public double SHP { get; set; }

public double SLL { get; set; }

public double SOS { get; set; }

public double SRD { get; set; }

public double STD { get; set; }

public double SVC { get; set; }

public double SYP { get; set; }

public double SZL { get; set; }

public double THB { get; set; }

public double TJS { get; set; }

public double TMT { get; set; }

public double TND { get; set; }

public double TOP { get; set; }

public double TRY { get; set; }

public double TTD { get; set; }

public double TWD { get; set; }

public double TZS { get; set; }

public double UAH { get; set; }

public double UGX { get; set; }

public double USD { get; set; }

public double UYU { get; set; }

public double UZS { get; set; }

public double VEF { get; set; }

public double VND { get; set; }

public double VUV { get; set; }

public double WST { get; set; }

public double XAF { get; set; }

public double XAG { get; set; }

public double XAU { get; set; }

public double XCD { get; set; }

public double XDR { get; set; }

public double XOF { get; set; }

public double XPD { get; set; }

public double XPF { get; set; }

public double XPT { get; set; }

public double YER { get; set; }

public double ZAR { get; set; }

public double ZMK { get; set; }

public double ZMW { get; set; }

public double ZWL { get; set; }

}

public class Rate

{

public double TaxRate { get; set; }

public string Code { get; set; }

}

}

1. Modificamos la **MainViewModel** para que quede de la siguiente manera:

using Divisas2MVVM.Classes;

using Newtonsoft.Json;

using System;

using System.Collections.ObjectModel;

using System.ComponentModel;

using System.Net.Http;

using System.Reflection;

namespace Divisas2MVVM.ViewModels

{

public class MainViewModel : INotifyPropertyChanged

{

#region Attributes

private ExchangeRates exchangeRates;

private decimal amount;

private double sourceRate;

private double targetRate;

private bool isRunning;

private bool isEnabled;

#endregion

#region Events

public event PropertyChangedEventHandler PropertyChanged;

#endregion

#region Properties

public ObservableCollection<Rate> Rates { get; set; }

public decimal Amount

{

set

{

if (amount != value)

{

amount = value;

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("Amount"));

}

}

get

{

return amount;

}

}

public double SourceRate

{

set

{

if (sourceRate != value)

{

sourceRate = value;

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("SourceRate"));

}

}

get

{

return sourceRate;

}

}

public double TargetRate

{

set

{

if (targetRate != value)

{

targetRate = value;

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("TargetRate"));

}

}

get

{

return targetRate;

}

}

public bool IsRunning

{

set

{

if (isRunning != value)

{

isRunning = value;

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("IsRunning"));

}

}

get

{

return isRunning;

}

}

public bool IsEnabled

{

set

{

if (isEnabled != value)

{

isEnabled = value;

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("IsEnabled"));

}

}

get

{

return isEnabled;

}

}

#endregion

#region Constructors

public MainViewModel()

{

Rates = new ObservableCollection<Rate>();

IsEnabled = false;

GetRates();

}

#endregion

#region Methods

private void LoadRates()

{

Rates.Clear();

var type = typeof(Rates);

var properties = type.GetRuntimeFields();

foreach (var property in properties)

{

var code = property.Name.Substring(1, 3);

Rates.Add(new Rate

{

Code = code,

TaxRate = (double)property.GetValue(exchangeRates.Rates),

});

}

}

private async void GetRates()

{

try

{

var client = new HttpClient();

client.BaseAddress = new Uri("https://openexchangerates.org");

var url = "/api/latest.json?app\_id=f490efbcd52d48ee98fd62cf33c47b9e";

var response = await client.GetAsync(url);

if (!response.IsSuccessStatusCode)

{

await App.Current.MainPage.DisplayAlert("Error", response.StatusCode.ToString(), "Aceptar");

IsRunning = false;

IsEnabled = false;

return;

}

var result = await response.Content.ReadAsStringAsync();

exchangeRates = JsonConvert.DeserializeObject<ExchangeRates>(result);

}

catch (Exception ex)

{

await App.Current.MainPage.DisplayAlert("Error", ex.Message, "Aceptar");

IsRunning = false;

IsEnabled = false;

return;

}

LoadRates();

IsRunning = false;

IsEnabled = true;

}

#endregion

}

}

1. Creamos la carpeta **Controls** y dentro de esta creamos la clase **BindablePicker** con el siguiente código (no olvide cambiar el nombre del namespace):

namespace Divisas2Prep.Controls  
{  
    using System;  
    using System.Collections;  
    using System.Collections.Specialized;  
    using System.Reflection;  
    using Xamarin.Forms;  
  
    public class BindablePicker : Picker  
    {  
        bool \_disableNestedCalls;  
  
        public static readonly BindableProperty ItemsSourceProperty =  
            BindableProperty.Create("ItemsSource", typeof(IEnumerable), typeof(BindablePicker),  
                null, propertyChanged: OnItemsSourceChanged);  
  
        public static readonly BindableProperty SelectedItemProperty =  
            BindableProperty.Create("SelectedItem", typeof(object), typeof(BindablePicker),  
                null, BindingMode.TwoWay, propertyChanged: OnSelectedItemChanged);  
  
        public static readonly BindableProperty SelectedValueProperty =  
            BindableProperty.Create("SelectedValue", typeof(object), typeof(BindablePicker),  
                null, BindingMode.TwoWay, propertyChanged: OnSelectedValueChanged);  
  
        public string DisplayMemberPath { get; set; }  
  
        public IEnumerable ItemsSource  
        {  
            get { return (IEnumerable)GetValue(ItemsSourceProperty); }  
            set { SetValue(ItemsSourceProperty, value); }  
        }  
  
        public object SelectedItem  
        {  
            get { return GetValue(SelectedItemProperty); }  
            set  
            {  
                if (this.SelectedItem != value)  
                {  
                    SetValue(SelectedItemProperty, value);  
                    InternalSelectedItemChanged();  
                }  
            }  
        }  
  
        public object SelectedValue  
        {  
            get { return GetValue(SelectedValueProperty); }  
            set  
            {  
                SetValue(SelectedValueProperty, value);  
                InternalSelectedValueChanged();  
            }  
        }  
  
        public string SelectedValuePath { get; set; }  
  
        public BindablePicker()  
        {  
            this.SelectedIndexChanged += OnSelectedIndexChanged;  
        }  
  
        public event EventHandler<SelectedItemChangedEventArgs> ItemSelected;  
  
        void InstanceOnItemsSourceChanged(object oldValue, object newValue)  
        {  
            \_disableNestedCalls = true;  
            this.Items.Clear();  
  
            var oldCollectionINotifyCollectionChanged = oldValue as INotifyCollectionChanged;  
            if (oldCollectionINotifyCollectionChanged != null)  
            {  
                oldCollectionINotifyCollectionChanged.CollectionChanged -= ItemsSource\_CollectionChanged;  
            }  
  
            var newCollectionINotifyCollectionChanged = newValue as INotifyCollectionChanged;  
            if (newCollectionINotifyCollectionChanged != null)  
            {  
                newCollectionINotifyCollectionChanged.CollectionChanged += ItemsSource\_CollectionChanged;  
            }  
  
            if (!Equals(newValue, null))  
            {  
                var hasDisplayMemberPath = !string.IsNullOrWhiteSpace(this.DisplayMemberPath);  
  
                foreach (var item in (IEnumerable)newValue)  
                {  
                    if (hasDisplayMemberPath)  
                    {  
                        var type = item.GetType();  
                        var prop = type.GetRuntimeProperty(this.DisplayMemberPath);  
                        this.Items.Add(prop.GetValue(item).ToString());  
                    }  
                    else  
                    {  
                        this.Items.Add(item.ToString());  
                    }  
                }  
  
                this.SelectedIndex = -1;  
                this.\_disableNestedCalls = false;  
  
                if (this.SelectedItem != null)  
                {  
                    this.InternalSelectedItemChanged();  
                }  
                else if (hasDisplayMemberPath && this.SelectedValue != null)  
                {  
                    this.InternalSelectedValueChanged();  
                }  
            }  
            else  
            {  
                \_disableNestedCalls = true;  
                this.SelectedIndex = -1;  
                this.SelectedItem = null;  
                this.SelectedValue = null;  
                \_disableNestedCalls = false;  
            }  
        }  
  
        void InternalSelectedItemChanged()  
        {  
            if (\_disableNestedCalls)  
            {  
                return;  
            }  
  
            var selectedIndex = -1;  
            object selectedValue = null;  
            if (this.ItemsSource != null)  
            {  
                var index = 0;  
                var hasSelectedValuePath = !string.IsNullOrWhiteSpace(this.SelectedValuePath);  
                foreach (var item in this.ItemsSource)  
                {  
                    if (item != null && item.Equals(this.SelectedItem))  
                    {  
                        selectedIndex = index;  
                        if (hasSelectedValuePath)  
                        {  
                            var type = item.GetType();  
                            var prop = type.GetRuntimeProperty(this.SelectedValuePath);  
                            selectedValue = prop.GetValue(item);  
                        }  
                        break;  
                    }  
                    index++;  
                }  
            }  
            \_disableNestedCalls = true;  
            this.SelectedValue = selectedValue;  
            this.SelectedIndex = selectedIndex;  
            \_disableNestedCalls = false;  
        }  
  
        void InternalSelectedValueChanged()  
        {  
            if (\_disableNestedCalls)  
            {  
                return;  
            }  
  
            if (string.IsNullOrWhiteSpace(this.SelectedValuePath))  
            {  
                return;  
            }  
            var selectedIndex = -1;  
            object selectedItem = null;  
            var hasSelectedValuePath = !string.IsNullOrWhiteSpace(this.SelectedValuePath);  
            if (this.ItemsSource != null && hasSelectedValuePath)  
            {  
                var index = 0;  
                foreach (var item in this.ItemsSource)  
                {  
                    if (item != null)  
                    {  
                        var type = item.GetType();  
                        var prop = type.GetRuntimeProperty(this.SelectedValuePath);  
                        if (object.Equals(prop.GetValue(item), this.SelectedValue))  
                        {  
                            selectedIndex = index;  
                            selectedItem = item;  
                            break;  
                        }  
                    }  
  
                    index++;  
                }  
            }  
            \_disableNestedCalls = true;  
            this.SelectedItem = selectedItem;  
            this.SelectedIndex = selectedIndex;  
            \_disableNestedCalls = false;  
        }  
  
        void ItemsSource\_CollectionChanged(object sender, NotifyCollectionChangedEventArgs e)  
        {  
            var hasDisplayMemberPath = !string.IsNullOrWhiteSpace(this.DisplayMemberPath);  
            if (e.Action == NotifyCollectionChangedAction.Add)  
            {  
                foreach (var item in e.NewItems)  
                {  
                    if (hasDisplayMemberPath)  
                    {  
                        var type = item.GetType();  
                        var prop = type.GetRuntimeProperty(this.DisplayMemberPath);  
                        this.Items.Add(prop.GetValue(item).ToString());  
                    }  
                    else  
                    {  
                        this.Items.Add(item.ToString());  
                    }  
                }  
            }  
            else if (e.Action == NotifyCollectionChangedAction.Remove)  
            {  
                foreach (var item in e.NewItems)  
                {  
                    if (hasDisplayMemberPath)  
                    {  
                        var type = item.GetType();  
                        var prop = type.GetRuntimeProperty(this.DisplayMemberPath);  
                        this.Items.Remove(prop.GetValue(item).ToString());  
                    }  
                    else  
                    {  
                        this.Items.Remove(item.ToString());  
                    }  
                }  
            }  
            else if (e.Action == NotifyCollectionChangedAction.Replace)  
            {  
                foreach (var item in e.NewItems)  
                {  
                    if (hasDisplayMemberPath)  
                    {  
                        var type = item.GetType();  
                        var prop = type.GetRuntimeProperty(this.DisplayMemberPath);  
                        this.Items.Remove(prop.GetValue(item).ToString());  
                    }  
                    else  
                    {  
                        var index = this.Items.IndexOf(item.ToString());  
                        if (index > -1)  
                        {  
                            this.Items[index] = item.ToString();  
                        }  
                    }  
                }  
            }  
            else if (e.Action == NotifyCollectionChangedAction.Reset)  
            {  
                this.Items.Clear();  
                if (e.NewItems != null)  
                {  
                    foreach (var item in e.NewItems)  
                    {  
                        if (hasDisplayMemberPath)  
                        {  
                            var type = item.GetType();  
                            var prop = type.GetRuntimeProperty(this.DisplayMemberPath);  
                            this.Items.Remove(prop.GetValue(item).ToString());  
                        }  
                        else  
                        {  
                            var index = this.Items.IndexOf(item.ToString());  
                            if (index > -1)  
                            {  
                                this.Items[index] = item.ToString();  
                            }  
                        }  
                    }  
                }  
                else  
                {  
                    \_disableNestedCalls = true;  
                    this.SelectedItem = null;  
                    this.SelectedIndex = -1;  
                    this.SelectedValue = null;  
                    \_disableNestedCalls = false;  
                }  
            }  
        }  
  
        static void OnItemsSourceChanged(BindableObject bindable, object oldValue, object newValue)  
        {  
            if (Equals(newValue, null) && Equals(oldValue, null))  
            {  
                return;  
            }  
  
            var picker = (BindablePicker)bindable;  
            picker.InstanceOnItemsSourceChanged(oldValue, newValue);  
        }  
  
        void OnSelectedIndexChanged(object sender, EventArgs e)  
        {  
            if (\_disableNestedCalls)  
            {  
                return;  
            }  
  
            if (this.SelectedIndex < 0 || this.ItemsSource == null || !this.ItemsSource.GetEnumerator().MoveNext())  
            {  
                \_disableNestedCalls = true;  
                if (this.SelectedIndex != -1)  
                {  
                    this.SelectedIndex = -1;  
                }  
                this.SelectedItem = null;  
                this.SelectedValue = null;  
                \_disableNestedCalls = false;  
                return;  
            }  
  
            \_disableNestedCalls = true;  
  
            var index = 0;  
            var hasSelectedValuePath = !string.IsNullOrWhiteSpace(this.SelectedValuePath);  
            foreach (var item in this.ItemsSource)  
            {  
                if (index == this.SelectedIndex)  
                {  
                    this.SelectedItem = item;  
                    if (hasSelectedValuePath)  
                    {  
                        var type = item.GetType();  
                        var prop = type.GetRuntimeProperty(this.SelectedValuePath);  
                        this.SelectedValue = prop.GetValue(item);  
                    }  
  
                    break;  
                }  
                index++;  
            }  
  
            \_disableNestedCalls = false;  
        }  
  
        static void OnSelectedItemChanged(BindableObject bindable, object oldValue, object newValue)  
        {  
            var boundPicker = (BindablePicker)bindable;  
            boundPicker.ItemSelected?.Invoke(boundPicker, new SelectedItemChangedEventArgs(newValue));  
            boundPicker.InternalSelectedItemChanged();  
        }  
  
        static void OnSelectedValueChanged(BindableObject bindable, object oldValue, object newValue)  
        {  
            var boundPicker = (BindablePicker)bindable;  
            boundPicker.InternalSelectedValueChanged();  
        }  
  
    }  
}

1. Modificamos la **MainPage** para que quede así:

<?xml version="1.0" encoding="UTF-8"?>  
<ContentPage   
    xmlns="http://xamarin.com/schemas/2014/forms"   
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"   
    xmlns:control="clr-namespace:Divisas2Prep.Controls"   
    x:Class="Divisas2Prep.Views.MainPage"   
    Title="Divisas"   
    BindingContext="{Binding Main, Source={StaticResource Locator}}">  
    <ContentPage.Content>  
        <ScrollView>  
            <StackLayout   
                Padding="8">  
                <Label   
                    Text="Valor a convertir:">  
                </Label>  
                <Entry   
                    Keyboard="Numeric"   
                    Text="{Binding Amount}">  
                </Entry>  
                <Label   
                    Text="Moneda origen:">  
                </Label>  
                <control:BindablePicker   
                    Title="Seleccione moneda origen..."   
                    DisplayMemberPath="Code" SelectedValuePath="TaxRate"   
                    ItemsSource="{Binding Rates}"   
                    SelectedValue="{Binding Path=SourceRate, Mode=TwoWay}"   
                    HorizontalOptions="FillAndExpand"   
                    VerticalOptions="Center">  
                </control:BindablePicker>  
                <Label   
                    Text="Modena destino:">  
                </Label>  
                <control:BindablePicker   
                    Title="Seleccione moneda destino..."   
                    DisplayMemberPath="Code"   
                    SelectedValuePath="TaxRate"   
                    ItemsSource="{Binding Rates}"   
                    SelectedValue="{Binding Path=TargetRate, Mode=TwoWay}"   
                    HorizontalOptions="FillAndExpand"   
                    VerticalOptions="Center">  
                </control:BindablePicker>  
                <Button   
                    Command="{Binding ConvertCommand}"   
                    IsEnabled="{Binding IsEnabled}"   
                    Text="Convertir"   
                    BackgroundColor="Navy"   
                    TextColor="White"   
                    HeightRequest="40"   
                    BorderRadius="20">  
                </Button>  
                <ActivityIndicator   
                    IsRunning="{Binding IsRunning}">  
                </ActivityIndicator>  
                <Label   
                    Text="{Binding Message}"   
                    BackgroundColor="Silver"   
                    TextColor="Purple"   
                    FontAttributes="Bold"   
                    HorizontalTextAlignment="Center"   
                    VerticalTextAlignment="Center"   
                    HorizontalOptions="FillAndExpand"   
                    VerticalOptions="FillAndExpand">  
                </Label>  
            </StackLayout>  
        </ScrollView>  
    </ContentPage.Content>  
</ContentPage>

1. Probemos como queda nuestra interfaz de usuario:

|  |  |  |
| --- | --- | --- |
| Android  D:\Users\Zulu\Desktop\Nexus 4 (Marshmallow). Screenshot 2.png  D:\Users\Zulu\Desktop\Nexus 4 (Marshmallow). Screenshot 3.png | iOS  D:\Users\Zulu\Downloads\Simulator Screen Shot 6.02.2017, 10.40.17 p.m..png  D:\Users\Zulu\Downloads\Simulator Screen Shot 6.02.2017, 10.40.31 p.m..png | UWP |

1. Agregamos el comando a nuestro **MainViewModel** (también la propiedad y atributo para message con su respectiva inicialización en el constructor):

Propiedad:

private string message;

Atributo:

public string Message

{

set

{

if (message != value)

{

message = value;

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("Message"));

}

}

get

{

return message;

}

}

Inicialización en el constructor:

Message = "Ingrese la cantidad a convertir, la moneda orgien, la monda destino y presione el botón de 'Convertir'";

Comando:

        #region Commands  
        public ICommand ConvertCommand { get { return new RelayCommand(ConvertMoney); } }  
  
        private async void ConvertMoney()  
        {  
            if (Amount <= 0)  
            {  
                await App.Current.MainPage.DisplayAlert("Error", "Debes ingresar un valor a convertir", "Aceptar");  
                return;  
            }  
  
            if (SourceRate == 0)  
            {  
                await App.Current.MainPage.DisplayAlert("Error", "Debes seleccionar la moneda origen", "Aceptar");  
                return;  
            }  
  
            if (TargetRate == -1)  
            {  
                await App.Current.MainPage.DisplayAlert("Error", "Debes seleccionar la moneda destino", "Aceptar");  
                return;  
            }  
  
            decimal amountConverted = amount / (decimal)sourceRate \* (decimal)targetRate;  
  
            Message = string.Format("{0:N2} = {1:N2}", amount, amountConverted);  
        }  
        #endregion

1. Probemos como quedo:

|  |  |  |
| --- | --- | --- |
| Android  D:\Users\Zulu\Desktop\Nexus 4 (Marshmallow). Screenshot 4.png | iOS  D:\Users\Zulu\Downloads\Simulator Screen Shot 6.02.2017, 11.04.48 p.m..png | UWP |