**LANDS**

**Video 2**

App.xaml: es por donde arranca la Solución

<?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:d="http://xamarin.com/schemas/2014/forms/design"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

mc:Ignorable="d"

x:Class="lands.App">

Aquí va el Diccionario de Recursos

<Application.Resources>

</Application.Resources>

</Application>

# App.xaml.cs:

using System;

using Xamarin.Forms;

using Xamarin.Forms.Xaml;

namespace lands

{

public partial class App : Application

{

public App()

{

Aquí va la página por donde empieza la App

InitializeComponent();

MainPage = new MainPage();

}

protected override void OnStart()

{

// Handle when your app starts

}

protected override void OnSleep()

{

// Handle when your app sleeps

}

protected override void OnResume()

{

// Handle when your app resumes

}

}

}

# MainPage.xaml

<?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:d="http://xamarin.com/schemas/2014/forms/design"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

mc:Ignorable="d"

x:Class="lands.MainPage">

<StackLayout>

Página por donde empieza la App.

Muestra “Welcome to Xamarin Forms!”

<!-- Place new controls here -->

<Label Text="Welcome to Xamarin.Forms!"

HorizontalOptions="Center"

VerticalOptions="CenterAndExpand" />

</StackLayout>

</ContentPage>

Vamos a trabajar con el modelo MVVM.

Entonces se crean las carpetas:

* **M**odels
* **V**iews
* **V**iew**M**odels

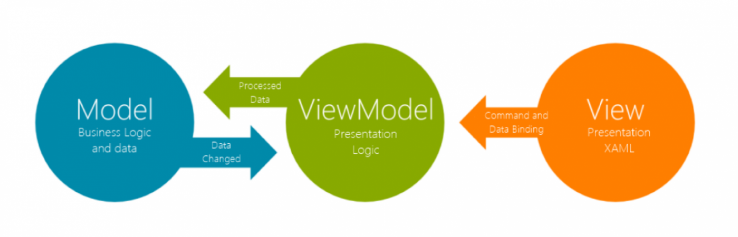
Esto se conoce como arquitectura del Proyecto **MVVM**

**Video 3**

Models: representación de los objetos que la aplicación va a manipular

View: acá va todo lo que verá el usuario

ViewModels: acá va lo lógica



Dentro de Views agregamos un Xamarin.Forms ContentPage llamado LoginPage que será nuestra pantalla de inicio.

# LoginPage.xaml

<?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:d="http://xamarin.com/schemas/2014/forms/design"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

mc:Ignorable="d"

x:Class="lands.Views.LoginPage">

<ContentPage.Content>

<StackLayout>

<Label Text="Welcome to Xamarin.Forms!"

VerticalOptions="CenterAndExpand"

HorizontalOptions="CenterAndExpand" />

</StackLayout>

</ContentPage.Content>

</ContentPage>

# LoginPage.xaml.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Xamarin.Forms;

using Xamarin.Forms.Xaml;

namespace lands.Views

{

[XamlCompilation(XamlCompilationOptions.Compile)]

public partial class LoginPage : ContentPage

{

public LoginPage()

{

InitializeComponent();

}

}

}

En LoginPage.xaml.cs se trata de no poner nada de nuevo código. Se escribe todo en clases aparte. No hay que poner la lógica de negocios en el mismo lugar de la interfaz de usuario.

**Para cambiar la pantalla de inicio:**

El Proyecto SIEMPRE arranca por el App.xaml. Entonces allí hay que decirle que vaya a la página LoginPage

En App.xaml.cs y cambiar:

MainPage = new MainPage();

por

this.MainPage = new LoginPage();

Hay que poner mas arriba

using Views;

namespace lands

{

Hay que agregar este using Views para que reconozca LoginPage

using Views;

using Xamarin.Forms;

public partial class App : Application

{

#region Constructors

public App()

{

InitializeComponent();

Nueva página de Inicio LoginPage

this.MainPage = new LoginPage();

}

Para mantener el código ordenado conviene ir poniendo “regiones” al código

#endregion

#region Methods

protected override void OnStart()

{

// Handle when your app starts

}

protected override void OnSleep()

{

// Handle when your app sleeps

Métodos para cuando empieza la App, se “duerme”, y se reestablece

}

protected override void OnResume()

{

// Handle when your app resumes

Para poner Región

**Control K Control S**

}

#endregion

}

}

<?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:d="http://xamarin.com/schemas/2014/forms/design"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

mc:Ignorable="d"

x:Class="lands.Views.LoginPage">

Cambiamos el label por “Login”

<ContentPage.Content>

<StackLayout>

<Label Text="Login"

VerticalOptions="CenterAndExpand"

HorizontalOptions="CenterAndExpand" />

</StackLayout>

Para corregir identación

**Control A**

(selecciona todo), y luego

**Control K Control D**

</ContentPage.Content>

</ContentPage>

**Propiedades que ponemos en el StackLayout:**

Padding

Padding="10" 10 para los 4 costados

Padding="10,20" 10,20 para izquierda-arriba y para derecha abajo

Padding="10,20,10,20" izquierda, arriba, derecha, abajo

No pueden ser 3 medidas

**Propiedades de Label, Entry y Bottom**

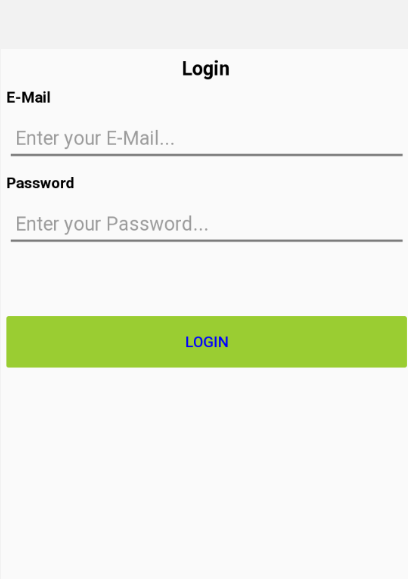
<ContentPage.Content>

<StackLayout Padding="5">

<Label FontAttributes="Bold"

FontSize="Large"

HorizontalOptions="Center"

 Text="Login"

TextColor="Black"

VerticalOptions="Start" />

<Label FontAttributes="Bold"

FontSize="Medium"

HorizontalOptions="Start"

Text="E-Mail"

TextColor="Black"

VerticalOptions="Start" />

<Entry Keyboard="Email"

Placeholder=" Enter your E-Mail..."></Entry>

<Label FontAttributes="Bold"

FontSize="Medium"

HorizontalOptions="Start"

Text="Password"

TextColor="Black"

VerticalOptions="Start" />

<Entry Keyboard="Email"

IsPassword="True"

Placeholder=" Enter your Password..."></Entry>

<ActivityIndicator></ActivityIndicator>

<Button BackgroundColor="YellowGreen"

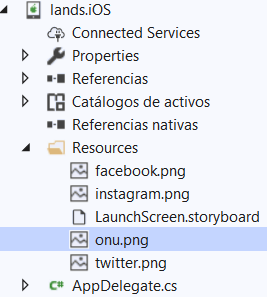
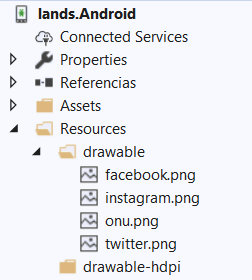
Text="Login"

Activity Indicator no se ve.

Es el relojito que se ve cuando está procesando

TextColor="Blue"></Button>

</StackLayout>



**Cargar una imagen en la Page**

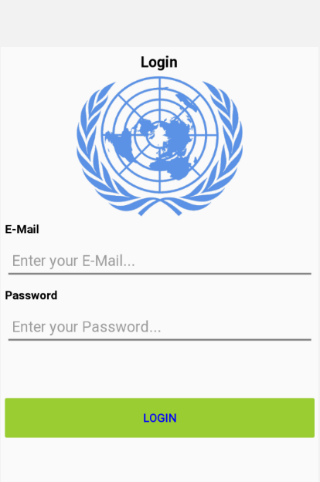
Hay que buscar imágenes PNG.

Para eso en Google Imágenes poner Herramientas-Color-Transparente

Las imágenes hay que cargarlas en cada Proyecto (Android e IOS).

En Android, en la carpeta Resources/drawable

En IOS, en la carpeta Resources



Entonces se agrega la imagen después del Label de Login:

<ContentPage.Content>

<StackLayout Padding="5">

<Label FontAttributes="Bold"

FontSize="Large"

HorizontalOptions="Center"

Text="Login"

TextColor="Black"

VerticalOptions="Start" />

<Image

Source="onu"

WidthRequest="200">

</Image>

<Label FontAttributes="Bold"

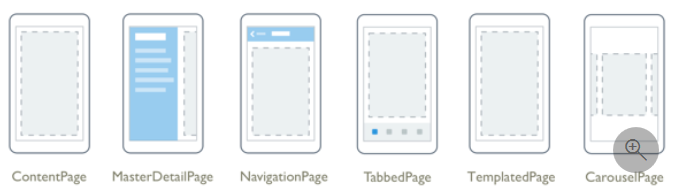
FontSize="Medium"

**Página de Microsoft que explica y muestra los distintos tipos de Contenedores:**

<https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/controls/>

* Pages
* Layouts
* Views
* Cells

**Pages:**



* **ContentPage:** es el contenedor más sencillo

### MasterDetailPage: menú hamburguesa (3 rayitas)

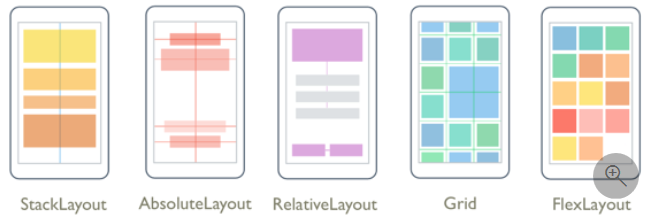
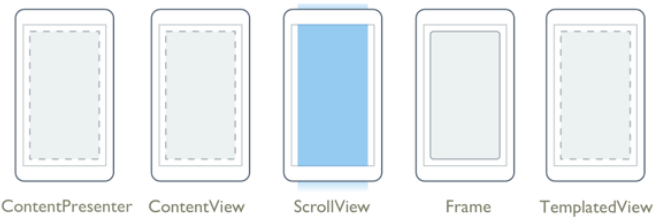
### NavigationPage: apilo paginas (hay flechita para volver para atrás)

### TabbedPage: se usa en whatsapp

### CarouselPage: se usa en las fotos

### TemplatedPage

**Layouts**



**Views**

**Para presentación**

Label

Image

BoxView

WebView

OpenGLView

Map

**Para inicializar comandos**

Button

ImageButton

SearchBar

**Para setear valores**

CheckBox

Slider

Stepper

Switch

DatePicker

TimePicker

**Para editar texto**

Entry

Editor

**Para indicar actividad**

ActivityIndicator

ProgressBar

**Para mostrar colecciones**

CollectionView

ListView

Picker

TableView

**Cells**

TextCell ImageCell SwitchCell EntryCell

Controles adicionales 🡪 buscar en Google “xamarin forms controls”

**Video 4**

**Poner el Entry al lado del Label**

Un stacklayout puede tener por dentro otro stacklayout que a su vez tiene dentro un grid.

Uso de Grid:

<Image Source="onu"

WidthRequest="200"></Image>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="3\*" />

</Grid.ColumnDefinitions>

<Label FontAttributes="Bold"

FontSize="Medium"

Grid.Row="0"

Grid.Column="0"

HorizontalOptions="Start"

VerticalOptions="Center"

Text="E-Mail"

TextColor="Black" />

<Entry Keyboard="Email"

Placeholder=" Enter your E-Mail..."

Grid.Row="0"

Grid.Column="1"></Entry>

</Grid>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="3\*" />

</Grid.ColumnDefinitions>

<Label FontAttributes="Bold"

FontSize="Medium"

Grid.Row="0"

Grid.Column="0"

HorizontalOptions="Start"

Text="Password"

TextColor="Black"

VerticalOptions="Center" />

<Entry Keyboard="Email"

IsPassword="True"

Grid.Row="0"

Grid.Column="1"

Placeholder=" Enter your Password..."></Entry>

</Grid>

Cambiamos en App.xaml.cs lo siguiente:

public partial class App : Application

{

public App()

{

InitializeComponent();

MainPage = new MainPage();

Cambiamos por:

public partial class App : Application

{

#region Constructors

public App()

{

InitializeComponent();

this.MainPage = new NavigationPage(new LoginPage());

Eso permite que en LoginPage.xaml pueda poner un Título a la Page:

<?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:d="http://xamarin.com/schemas/2014/forms/design"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

mc:Ignorable="d"

x:Class="lands.Views.LoginPage"

Title="Login">

<ContentPage.Content>

<StackLayout Padding="5">

<Label FontAttributes="Bold"

FontSize="Large"

Seguimos con el armado de la interfaz de la LoginPage:

<?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:d="http://xamarin.com/schemas/2014/forms/design"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

mc:Ignorable="d"

x:Class="lands.Views.LoginPage"

Title="Login">

<ContentPage.Content>

<StackLayout Padding="5">

<Label FontAttributes="Bold"

FontSize="Large"

HorizontalOptions="Center"

Text="Login"

TextColor="Black"

VerticalOptions="Start" />

<Image Source="onu"

WidthRequest="200"></Image>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="3\*" />

</Grid.ColumnDefinitions>

<Label FontAttributes="Bold"

FontSize="Medium"

Grid.Row="0"

Grid.Column="0"

HorizontalOptions="Start"

VerticalOptions="Center"

Text="E-Mail"

TextColor="Black" />

<Entry Keyboard="Email"

Placeholder=" Enter your E-Mail..."

Grid.Row="0"

Grid.Column="1"></Entry>

</Grid>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="3\*" />

</Grid.ColumnDefinitions>

<Label FontAttributes="Bold"

FontSize="Medium"

Grid.Row="0"

Grid.Column="0"

HorizontalOptions="Start"

Text="Password"

TextColor="Black"

VerticalOptions="Center" />

<Entry Keyboard="Email"

IsPassword="True"

Grid.Row="0"

Grid.Column="1"

Placeholder=" Enter your Password..."></Entry>

</Grid>

<StackLayout Orientation="Horizontal">

<Label Text="Rememberme in this Device"

TextColor="Black"

TextDecorations="Underline"

HorizontalOptions="StartAndExpand"

VerticalOptions="Center"></Label>

<Switch HorizontalOptions="EndAndExpand"></Switch>

</StackLayout>

<ActivityIndicator></ActivityIndicator>

<StackLayout Margin=" 40,0">

<Button BackgroundColor="YellowGreen"

Text="Login"

TextColor="Blue"></Button>

<Button BackgroundColor="Purple"

Text="Register"

TextColor="White"></Button>

</StackLayout>

<StackLayout Orientation="Horizontal"

HorizontalOptions="Center">

<Image Source="facebook"

WidthRequest="90"></Image>

<Image Source="twitter"

WidthRequest="90"></Image>

<Image Source="instagram"

WidthRequest="120"></Image>

</StackLayout>

</StackLayout>

</ContentPage.Content>

</ContentPage>

Agregamos las imágenes (botones) de Facebook, Twitter e Instagram

Agregamos “Recordar en este dispositivo” y un switch

**Video 5**

Creamos en la carpeta ViewModels, la Clase **MainViewModels. Será la clase MAS IMPORTANTE.**

**prop**

y luego

**Tab Tab**

Crea una Propiedad

namespace lands.ViewModels

{

public class MainViewModel

{

}

}

Creamos otra Carpeta, **Infrastructure.**

Dentro creamos una Clase **InstanceLocator.** Esta clase sirve para instanciar la MainViewModel

namespace lands.Infrastructure

**ctor**

y luego

**Tab Tab**

Crea un Constructor

{

using ViewModels;

public class InstanceLocator

{

#region Properties

public MainViewModel Main { get; set; }

#endregion

#region Constructors

public InstanceLocator()

{

this.Main = new MainViewModel();

}

#endregion

}

}

Sirve para unir la LoginPage con la MainViewModel

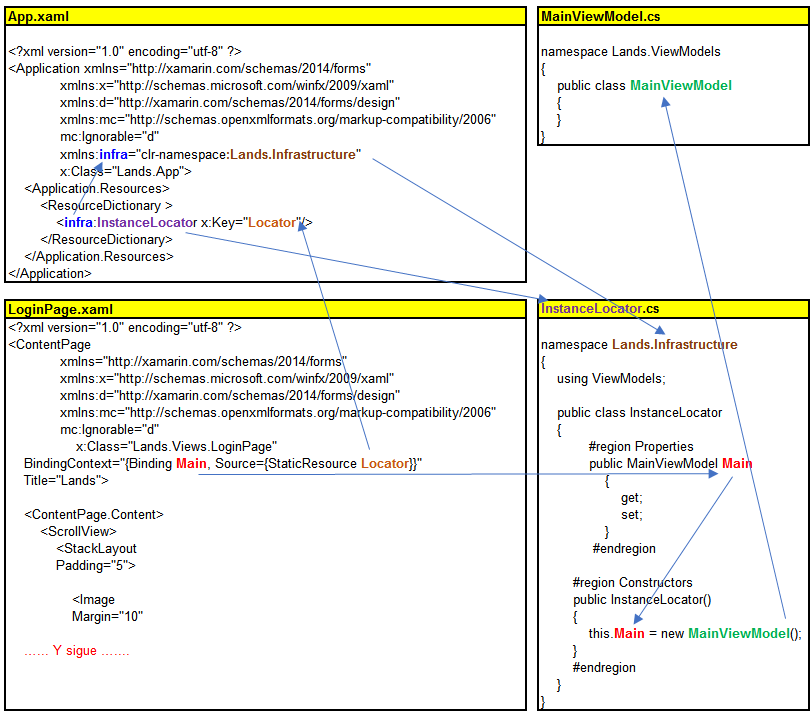
Creamos una clase InstanceLocator que referencia un objeto Main de tipo MainViewModel, que lo instanciamos en el constructor de InstanceLocator.

Cuando arranca la aplicación, pasa por el diccionario de Recursos en App.xaml

Ahí dice que InstanceLocator se encuentra en infra

Infra se encuentra en Lands.Insfrastructure

En Infrastructure encuentra la clase InstanceLocator que se llamará locator



En la LoginPage ponemos el BindingContext

BindingContext="{Binding Main,Source={StaticResource Locator}}"

Luego ponemos todo el contenido de la LoginPage dentro de un ScrollView

**Video 6**

Agregamos el ´Forgot your password´:

<Label

Text="Forgot your password?"

HorizontalOptions="Center"

TextColor="Blue"

TextDecorations="Underline">

</Label>

Ponemos los dos botones Login y Register uno al lado del otro poniendo

Orientation="Horizontal" al StackLayout que los contiene

y

HorizontalOptions="FillAndExpand" a cada botón.

Acomodo dinámico de espacios para que no queden lugares en blanco.

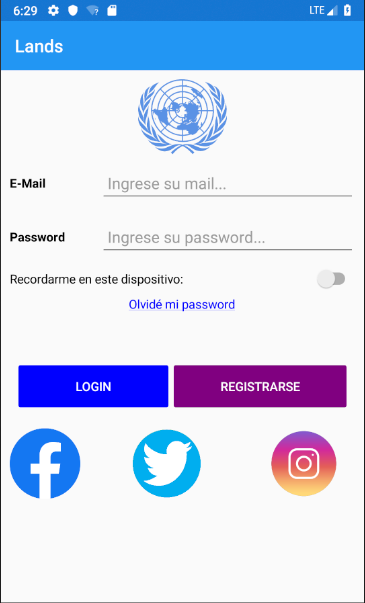
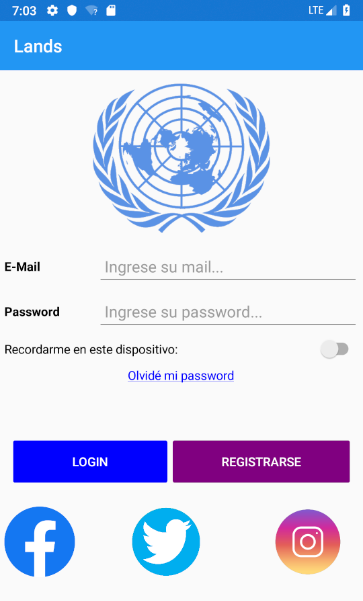
Usamos

VerticalOptions="CenterAndExpand"

o

VerticalOptions="EndAndExpand"

según el caso.

**Video 7**

Así se ponen comentarios en XAML

<!--Application resource dictionary -->

La App tendrá muchas Pages.

**Páginas que tendrá el Proyecto:**

* LoginPage
* LandsPages
* OneLandPage
* RegisterPage
* ForgottedPasswordPage
* LoginFacebookPage
* LoginTwitterPage
* LoginInstagramPage

Todas podrían referirse a la MainViewModel… pero se hace de otra forma.

Lo que se hace es que cada Page tenga su Model:

LoginPage 🡪 LoginViewModel

RegisterPage 🡪 RegisterViewModel

etc….

La MainViewModel es la que controla el resto de ViewModel

En la carpeta ViewModels creamos la Clase LoginViewModel

Para conectar la LoginViewModel con la MainVieModel hacemos lo siguiente en la MainViewModel:

namespace lands.ViewModels

{

public class MainViewModel

{

#region ViewModels

public LoginViewModel Login { get; set; }

Creamos una región ViewModels

Ahí referenciamos la LoginViewModel

* Creamos una propiedad Login de tipo LoginViewModel
* En el constructor la instanciamos

#endregion

#region constructors

public MainViewModel()

{

this.Login = new LoginViewModel();

}

#endregion

}

}

Hay que unir la propiedad Login con la LoginPage. Entonces en la LoginPage se ponen los Bindings

Se pone un Binding al Layout principal de la Page (el ScrollView)

BindingContext="{Binding Login}">

Luego se ponen Bindings a los controles en que hay que capturar acciones del usuario:

Text="{Binding Email,Mode=TwoWay}"

Las propiedades son distintas de acuerdo al control:

* Text para los Entries
* IsRunnig para el Activity Indicator
* IsToogled para los Switches
* Command para los Botones

Text="{Binding Password,Mode=TwoWay}"

IsRunning="{Binding IsRunning,Mode=TwoWay}"

IsToggled="{Binding IsRemembered,Mode=TwoWay}"

Command="{Binding LoginCommand}"

Mode=Twoway

es que se puede leer o se pueden

poner valores en el control

Todo esto deben ser propiedades en la LoginViewModel

namespace lands.ViewModels

{

using GalaSoft.MvvmLight.Command;

using System.Windows.Input;

public class LoginViewModel

{

#region Properties

En una región Properties ponemos las propiedades

public string Email { get; set; }

public string Password { get; set; }

public bool IsRunning { get; set; }

public bool IsRemembered { get; set; }

#endregion

#region Commands

public ICommand LoginCommand

{

get

Las Propiedades de botones las ponemos aparte.

Son de tipo ICommand.

Hay que importar la System.Wndows.Input

{

return new RelayCommand(Login);

}

}

**Los valores booleanos por defecto están en FALSE**

private void Login()

{

}

#endregion

#region Constructors

En la Región de los Constructores ponemos los valores iniciales

public LoginViewModel()

{

this.IsRemembered = true;

}

#endregion

}

}

**Nunca actualizar las actualizaciones de Nuggets**

Para que el botón funcione, hay que agregar el paquete nugget MvvmLigthsLibs

Se agrega sólo al Proyecto Lands (no agregar a Android e IOS).

En la LoginViewModels, dentro de los Comandos, sacamos el ‘Set’ del ICommand (solo puede usarse el get).

Y al get lo hacemos:

public ICommand LoginCommand

{

get

{

return new RelayCommand(Login); 🡨 para usar RelayCommand hay que agregar el using

} GalaSoft.MvvmLight.Command

} Lleva como parámetro el nombre del método (Login)

Y se crea el método:

private void Login()

{

}

Vamos a validar que el usuario haya digitado Email y Password

**Video 8**

private async void Login()

{

Hay que poner using Xamarin.Forms

//Verifica si se escribió un email y un password

if (string.IsNullOrEmpty(this.Email))

{

await Application.Current.MainPage.DisplayAlert(

"Error",

"You must enter an Email",

"Accept");

return;

Para un método asíncrono hay que poner await

}

if (string.IsNullOrEmpty(this.Email))

{

await Application.Current.MainPage.DisplayAlert(

"Error",

"You must enter a pasword",

"Accept");

return;

}

//Verifica si email y password son los correctos

if (this.Email !="ln" || this.Password!="123") // Alt 124 para |

{

await Application.Current.MainPage.DisplayAlert(

"Error",

"Email or password incorrect",

"Accept");

return;

}

//Acá llega sin email y password son los correctos

await Application.Current.MainPage.DisplayAlert(

"Ok",

Borra casillero password en memoria pero no lo refresca en la View

"Fuck yeahhh!!!",

"Accept");

this.Password = string.Empty;

return;

}

#endregion

#region Constructors

public LoginViewModel()

{

this.IsRemembered = true;

}

#endregion

}

}

Cuando uno aprieta el botón de Login hay que deshabilitar el botón.

En LoginPage.xaml ponemos

IsEnabled="{Binding iIsEnabled,Mode=TwoWay}" 🡨 para el botón Login y para el botón Register

Entonces en la LoginViewModel hay que poner esta propiedad:

public bool IsEnabled { get; set; }

Y en la región de los constructores hay que agregar que el valor de inicio es true (los booleanos por defecto están en false, sino el botón aparecería deshabilitado):

#region Constructors

public LoginViewModel()

{

this.IsRemembered = true;

this.IsEnabled = true; 🡨 poner en true

}

#endregion

Para que refresque la View cuando borramos el password, hay que disparar un evento.

Agregamos esto, y para que funcione hay que hacer

using System.ComponentModel

Dentro de la LoginViewModel se implementa una interfaz.

#region Events

public event PropertyChangedEventHandler PropertyChanged;

#endregion

Hay que crear una propiedad privada por cada atributo que se quiera refrescar. Los nombres son los mismos pero con minúsculas.

#region Attributes

private string password;

private bool isrunning;

private bool isenabled;

#endregion

Y luego cambian las propiedades (las que empiezan con mayúsculas):

public string Password

{

get

{

return password;

}

set

{

if (this.password != value)

{

this.password = value;

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(nameof(this.Password)));

}

}

}

public bool IsRunning

{

get

{

return isRunning;

}

set

{

if (this.isRunning != value)

{

this.isRunning = value;

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

}

}

}

public bool IsRemembered {get;set;}

public bool IsEnabled

{

get

{

return isEnabled ;

}

set

{

if (this.isEnabled != value)

{

this.isEnabled = value;

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

}

}

}

**Todo esto es para refrescar las propiedades en tiempo de ejecución.**

En LoginVIewModel, en el procedimiento Login, hay que poner como van quedando el valor de las propiedades

private async void Login()

{

//Verifica si se escribió un email y un password

if (string.IsNullOrEmpty(this.Email))

{

await Application.Current.MainPage.DisplayAlert(

"Error",

"You must enter an Email",

"Accept");

return;

}

if (string.IsNullOrEmpty(this.Email))

{

await Application.Current.MainPage.DisplayAlert(

"Error",

"You must enter a pasword",

"Accept");

return;

}

this.IsRunning = true;

this.IsEnabled = false;

//Verifica si email y password son los correctos

if (this.Email !="ln" || this.Password!="123") // Alt 124 para |

{

this.IsRunning = false;

this.IsEnabled = true;

await Application.Current.MainPage.DisplayAlert(

"Error",

"Email or password incorrect",

"Accept");

return;

}

//Acá llega sin email y password son los correctos

this.IsRunning = false;

this.IsEnabled = true;

await Application.Current.MainPage.DisplayAlert(

"Ok",

"Fuck yeahhh!!!",

"Accept");

this.Password = string.Empty;

return;

}

#endregion

Como hay que refrescar en muchos lugares dentro de una aplicación, vamos a hacer una Clase que haga eso.

Creamos una Clase BaseViewModel

namespace Lands.ViewModels

{

using System.Collections.Generic;

using System.ComponentModel;

using System.Runtime.CompilerServices;

public class BaseViewModel : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

protected void OnPropertyChanged([CallerMemberName] string propertyName = null)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

protected void SetValue<T>(ref T backingField, T value, [CallerMemberName] string propertyName = null)

{

if (EqualityComparer<T>.Default.Equals(backingField, value))

{

return;

}

backingField = value;

OnPropertyChanged(propertyName);

}

}

}

Entonces

Hacemos que la LoginViewModel herede de la BaseVieModel

public class LoginViewModel :BaseViewModel

Nos ahorramos los eventos . Borramos estas líneas:

#region Events

public event PropertyChangedEventHandler PropertyChanged;

#endregion

Luego, en cada una de las propiedades cambiamos los get-set:

public string Password

{

get

{

return password;

}

set

{

if (this.password != value)

{

this.password = value;

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(nameof(this.Password)));

}

}

}

lo cambiamos por:

{

get { return password; }

set {SetValue(ref password, value);}

}

Y hacemos lo mismo para las restantes:

public bool IsRunning

{

get { return this.isRunning ; }

set { SetValue(ref this.isRunning, value); }

}

public bool IsEnabled

{

get { return this.isEnabled ; }

set { SetValue(ref this.isEnabled, value); }

**Video 9**

Navegar a una segunda Page, la LandsPage.

Agregamos en la carpeta Views, la LandsPage.

En LandsPage.xaml agregamos:

BindingContext="{Binding Main,Source={StaticResource Locator}}"

Title="Lands">>

Y también dentro del Content Page

<StackLayout BindingContext="{Binding Lands}"

Agregamos la Clase LandsViewModel

En la MainViewModel agregamos la propiedad

public LandsViewModel Lands { get; set; }

No se pone en el constructor, ya que se instanciará cuando se necesite.

Para que navegue desde la LoginPage a la LandsPage, cambiampos en la LoginViewModels, en la región commands, lo siguiente:

// SI EMAIL Y PASSWORD SON LOS CORRECTOS

this.IsRunning = false;

this.IsEnabled = true;

await Application.Current.MainPage.DisplayAlert(

"Ok",

"Fuck yeahhh!!!",

"Accept");

this.Password = string.Empty;

return;

lo cambiamos por:

this.Email = string.Empty;

this.Password = string.Empty;

await Application.Current.MainPage.Navigation.PushAsync(new LandsPage());

y ponemos

using Views;

Antes de instanciar la LandsPage hay que instanciar la LandsViewModel, para lo cual hay que referenciar la MainViewModel, y para eso se implementa un patrón Singleton. Así se evita que se creen múltiples instancias de esa Clase.

Dentro de la MainViewModel creamos una región Singleton

#region Singleton

private static MainViewModel instance;

public static MainViewModel GetInstance()

{

if (instance == null)

{

return new MainViewModel();

}

return instance;

}

#endregion

Y en el constructor ponemos:

region constructors

public MainViewModel()

{

instance = this;

this.Login = new LoginViewModel();

}

Permite llamar a la MainViewModel desde cualquier Clase.

En la LoginViewModel ponemos

// SI EMAIL Y PASSWORD SON LOS CORRECTOS

this.IsRunning = false;

this.IsEnabled = true;

this.Email = string.Empty;

this.Password = string.Empty;

MainViewModel.GetInstance().Lands = new LandsViewModel ();

await Application.Current.MainPage.Navigation.PushAsync(new LandsPage());

**TRAER LISTA DE PAISES DESDE UNA API**

<http://restcountries.eu/rest/v2/all>

{

"name": "Argentina",

"topLevelDomain": [

".ar"

],

"alpha2Code": "AR",

"alpha3Code": "ARG",

"callingCodes": [

"54"

],

"capital": "Buenos Aires",

"altSpellings": [

"AR",

"Argentine Republic",

"República Argentina"

],

"region": "Americas",

"subregion": "South America",

"population": 43590400,

"latlng": [

-34,

-64

],

"demonym": "Argentinean",

"area": 2780400,

"gini": 44.5,

"timezones": [

"UTC-03:00"

],

"borders": [

"BOL",

"BRA",

"CHL",

"PRY",

"URY"

],

"nativeName": "Argentina",

"numericCode": "032",

"currencies": [

{

"code": "ARS",

"name": "Argentine peso",

"symbol": "$"

}

],

"languages": [

{

"iso639\_1": "es",

"iso639\_2": "spa",

"name": "Spanish",

"nativeName": "Español"

},

{

"iso639\_1": "gn",

"iso639\_2": "grn",

"name": "Guaraní",

"nativeName": "Avañe'ẽ"

}

],

"translations": {

"de": "Argentinien",

"es": "Argentina",

"fr": "Argentine",

"ja": "アルゼンチン",

"it": "Argentina",

"br": "Argentina",

"pt": "Argentina",

"nl": "Argentinië",

"hr": "Argentina",

"fa": "آرژانتین"

},

"flag": "https://restcountries.eu/data/arg.svg",

"regionalBlocs": [

{

"acronym": "USAN",

"name": "Union of South American Nations",

"otherAcronyms": [

"UNASUR",

"UNASUL",

"UZAN"

],

"otherNames": [

"Unión de Naciones Suramericanas",

"União de Nações Sul-Americanas",

"Unie van Zuid-Amerikaanse Naties",

"South American Union"

]

}

],

"cioc": "ARG"

}

Esto es formato JSon

Ponemos en Google json2csharp

Copiamos un país desde la AIP a través de Postman, y obtenemos:

public class Currency

{

public string code { get; set; }

public string name { get; set; }

public string symbol { get; set; }

}

public class Language

{

public string iso639\_1 { get; set; }

public string iso639\_2 { get; set; }

public string name { get; set; }

public string nativeName { get; set; }

}

public class Translations

{

public string de { get; set; }

public string es { get; set; }

public string fr { get; set; }

public string ja { get; set; }

public string it { get; set; }

public string br { get; set; }

public string pt { get; set; }

public string nl { get; set; }

public string hr { get; set; }

public string fa { get; set; }

}

public class RegionalBloc

{

public string acronym { get; set; }

public string name { get; set; }

public List<string> otherAcronyms { get; set; }

public List<string> otherNames { get; set; }

}

public class RootObject

{

public string name { get; set; }

public List<string> topLevelDomain { get; set; }

public string alpha2Code { get; set; }

public string alpha3Code { get; set; }

public List<string> callingCodes { get; set; }

public string capital { get; set; }

public List<string> altSpellings { get; set; }

public string region { get; set; }

public string subregion { get; set; }

public int population { get; set; }

public List<int> latlng { get; set; }

public string demonym { get; set; }

public int area { get; set; }

public double gini { get; set; }

public List<string> timezones { get; set; }

public List<string> borders { get; set; }

public string nativeName { get; set; }

public string numericCode { get; set; }

public List<Currency> currencies { get; set; }

public List<Language> languages { get; set; }

public Translations translations { get; set; }

public string flag { get; set; }

public List<RegionalBloc> regionalBlocs { get; set; }

public string cioc { get; set; }

}

El JSon tiene varias Clases:

* Currency
* Language
* Translations
* REgionalBloc
* RootObject

Crearemos estos modelos

En la carpeta Models creamos la Clase Currency

Copiamos desde Json2CSharp

namespace lands.Models

{

public class Currency

{

public string code { get; set; }

public string name { get; set; }

public string symbol { get; set; }

}

}

En el standard de C# las propiedades deben empezar con MAYUSCULAS.

Hay que agregarles una propiedad

Agregamos el Nuget Newtonsoft Json al Proyecto Lands

Y en la Clase Currency ponemos:

namespace lands.Models

{

using Newtonsoft.Json;

public class Currency

{

“code es como se recibe desde la API, y “Code es como lo llamaremos internamente

[JsonProperty(PropertyName = "code")]

public string Code { get; set; }

[JsonProperty(PropertyName = "name")]

public string Name { get; set; }

[JsonProperty(PropertyName = "symbol")]

public string Symbol { get; set; }

}

}

Hay que poner el using Newtonsoft.Json;

De manera similar, luego hacemos la Clase Languages

namespace lands.Models

{

using Newtonsoft.Json;

public class Languages

{

[JsonProperty(PropertyName = "iso639\_1")]

public string Iso6391 { get; set; }

[JsonProperty(PropertyName = "iso639\_2")]

public string Iso6392 { get; set; }

[JsonProperty(PropertyName = "name")]

public string Name { get; set; }

[JsonProperty(PropertyName = "nativeName")]

public string NativeName { get; set; }

}

}

De manera similar, luego hacemos la Clase Translations

namespace lands.Models

{

using Newtonsoft.Json;

public class Translations

{

[JsonProperty(PropertyName = "de")]

public string Germany { get; set; }

[JsonProperty(PropertyName = "es")]

public string Spanish { get; set; }

[JsonProperty(PropertyName = "fr")]

public string French { get; set; }

[JsonProperty(PropertyName = "ja")]

public string Japanese { get; set; }

[JsonProperty(PropertyName = "it")]

public string Italian { get; set; }

[JsonProperty(PropertyName = "br")]

public string Brazilian { get; set; }

[JsonProperty(PropertyName = "pt")]

public string Portuges { get; set; }

[JsonProperty(PropertyName = "nl")]

public string Dutch { get; set; }

[JsonProperty(PropertyName = "hr")]

public string Croatian { get; set; }

[JsonProperty(PropertyName = "fa")]

public string Persian { get; set; }

}

}

De manera similar, luego hacemos la Clase RegionalBloc

namespace lands.Models

{

using Newtonsoft.Json;

public class RegionalBloc

{

[JsonProperty(PropertyName = "acronym")]

public string Acronym { get; set; }

[JsonProperty(PropertyName = "name")]

public string Name { get; set; }

}

}

Por último hacemos la Clase Land que es la que tiene los datos principales de cada país

namespace lands.Models

{

using Newtonsoft.Json;

using System.Collections.Generic;

public class Land

{

[JsonProperty(PropertyName = "name")]

public string Name { get; set; }

[JsonProperty(PropertyName = "topLevelDomain")]

public List<string> TopLevelDomain { get; set; }

[JsonProperty(PropertyName = "alpha2Code")]

public string Alpha2Code { get; set; }

[JsonProperty(PropertyName = "alpha3Code")]

public string Alpha3Code { get; set; }

[JsonProperty(PropertyName = "callingCodes")]

public List<string> CallingCodes { get; set; }

[JsonProperty(PropertyName = "capital")]

public string Capital { get; set; }

[JsonProperty(PropertyName = "altSpellings")]

public List<string> AltSpellings { get; set; }

[JsonProperty(PropertyName = "region")]

public string Region { get; set; }

[JsonProperty(PropertyName = "subregion")]

public string Subregion { get; set; }

[JsonProperty(PropertyName = "population")]

public int Population { get; set; }

[JsonProperty(PropertyName = "latlng")]

public List<int> Latlng { get; set; }

[JsonProperty(PropertyName = "demonym")]

public string Demonym { get; set; }

[JsonProperty(PropertyName = "area")]

public int Area { get; set; }

[JsonProperty(PropertyName = "gini")]

public double Gini { get; set; }

[JsonProperty(PropertyName = "timezones")]

public List<string> Timezones { get; set; }

[JsonProperty(PropertyName = "borders")]

public List<string> Borders { get; set; }

[JsonProperty(PropertyName = "nativeName")]

public string NativeName { get; set; }

[JsonProperty(PropertyName = "numericCode")]

public string NumericCode { get; set; }

[JsonProperty(PropertyName = "currencies")]

public List<Currency> Currencies { get; set; }

[JsonProperty(PropertyName = "languages")]

public List<Languages> Languages { get; set; }

[JsonProperty(PropertyName = "translations")]

public Translations Translations { get; set; }

[JsonProperty(PropertyName = "flag")]

public string Flag { get; set; }

[JsonProperty(PropertyName = "regionalBlocs")]

public List<RegionalBloc> RegionalBlocs { get; set; }

[JsonProperty(PropertyName = "cioc")]

public string Cioc { get; set; }

}

}

**Video 10**

En este video se termina de hacer lo anterior

Ahora hay que CONSUMIR esta información.

Cada vez que vayamos a consumir un servicio, hay que verificar Internet.

Para eso agregamos el Nugget Xam.Plugin.Connectivity

**Video 11**

En la carpetas Models ponemos una Clase llamada Response

namespace lands.Models

{

public class Response

{

public bool IsSuccess { get; set; }

public string Message { get; set; }

public object Result { get; set; }

}

}

Creamos una Carpeta llamada Services

Dentro de esa Carpeta creamos una Clase llamada ApiService (copiada desde el profesor) <https://github.com/Zulu55/Lands/tree/master/Lands/Lands/Models>

También creamos una Clase llamada TokenResponse (copiada desde el profesor)

namespace Lands.Models

{

using System;

using Newtonsoft.Json;

public class TokenResponse

{

#region Properties

[JsonProperty(PropertyName = "access\_token")]

public string AccessToken { get; set; }

[JsonProperty(PropertyName = "token\_type")]

public string TokenType { get; set; }

[JsonProperty(PropertyName = "expires\_in")]

public int ExpiresIn { get; set; }

[JsonProperty(PropertyName = "userName")]

public string UserName { get; set; }

[JsonProperty(PropertyName = ".issued")]

public DateTime Issued { get; set; }

[JsonProperty(PropertyName = ".expires")]

public DateTime Expires { get; set; }

[JsonProperty(PropertyName = "error\_description")]

public string ErrorDescription { get; set; }

#endregion

}

}

En ApiService hay un método llamado CheckConnection que nos garantiza si hay o no hay conexión a Internet

Hay otros métodos:

* GetToken
* Get<T>
* GetList<T> 🡨 hay2, con distintos parámetros
* Post Para crear
* Put Para modificar
* Delete Para borrar

Creamos la Clase TokenResponse

namespace Lands.Models

{

using System;

using Newtonsoft.Json;

public class TokenResponse

{

#region Properties

[JsonProperty(PropertyName = "access\_token")]

public string AccessToken { get; set; }

[JsonProperty(PropertyName = "token\_type")]

public string TokenType { get; set; }

[JsonProperty(PropertyName = "expires\_in")]

public int ExpiresIn { get; set; }

[JsonProperty(PropertyName = "userName")]

public string UserName { get; set; }

[JsonProperty(PropertyName = ".issued")]

public DateTime Issued { get; set; }

[JsonProperty(PropertyName = ".expires")]

public DateTime Expires { get; set; }

[JsonProperty(PropertyName = "error\_description")]

public string ErrorDescription { get; set; }

#endregion

}

}

En LandsViewModels creamos la lista de países como una ObservableCollection

namespace lands.ViewModels

{

using Lands.Services;

using Lands.ViewModels;

using Models;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using Xamarin.Forms;

public class LandsViewModel:BaseViewModel

Creamos el Servicio apiService

{

#region Services

private ApiService apiService;

#endregion

#region Attributes

Aquí se guardarán los países

private ObservableCollection<Land> lands;

#endregion

#region Properties

public ObservableCollection<Land> Lands

Aquí la propiedad correspondiente

{

get { return this.lands; }

set { SetValue(ref this.lands, value); }

}

#endregion

#region Constructors

Instanciamos apiService

public LandsViewModel()

{

this.apiService = new ApiService();

Método para cargar los países

this.LoadLands();

}

#endregion

#region Methods

private async void LoadLands()

{

var response = await this.apiService.GetList<Land>( 🡨Creamos esta variabla asíncrona

"https://restcountries.eu", que trae una lista del modelo Land

"/rest", Tiene 3 parámetros que forman la URL

"/v2/all");

if (!response.IsSuccess )

{

await Application.Current.MainPage.DisplayAlert(

"Error",

response.Message,

"Accept");

return;

}

var list = (List<Land>)response.Result; 🡨Creamos esta variable asíncrona y la

this.Lands = new ObservableCollection<Land>(list); transformamos en una ObservableCollection

}

#endregion

}

ObservableCollection es el tipo que se usa para poner datos en un ListView

}

**Video 12**

Agregamos estas líneas en LandsViewModel:

private async void LoadLands()

Lo que está en gris, valida que haya conexión a Internet

{

//Chequeamos si hay conección a Internet

var connection = await this.apiService.CheckConnection();

if (!connection.IsSuccess)

{

await Application.Current.MainPage.DisplayAlert(

"Error",

Hace que vuelva a la LoginPage

connection.Message,

"Accept");

await Application.Current.MainPage.Navigation.PopAsync();

return;

}

//Conectamos a la API de países

var response = await this.apiService.GetList<Land>(

"https://restcountries.eu",

"/rest",

"/v2/all");

if (!response.IsSuccess)

{

await Application.Current.MainPage.DisplayAlert(

"Error",

response.Message,

"Accept");

await Application.Current.MainPage.Navigation.PopAsync();

return;

}

//Pasamos la lista de pa{ises a una ObservableConnection

var list = (List<Land>)response.Result;

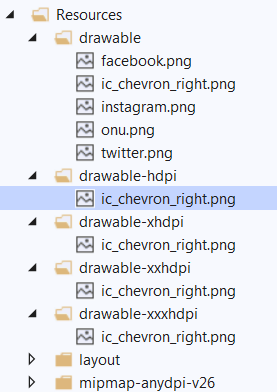
this.Lands = new ObservableCollection<Land>(list);

**Ahora vamos a hacer la ListView en LandsPage.xaml**

Sacamos el label y ponemos un ListView

Y lo Bindamos a la ObservableCollection Lands

De esa página bajamos el archivo del chevron y se agregan en las distintas carpetas de Resource:



**Android Asset Studio:** página web para bajar íconos

**Video 13**

**LandsPage.xaml**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

Permite o no filas de diferentes tamaños

xmlns:d="http://xamarin.com/schemas/2014/forms/design"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

mc:Ignorable="d"

x:Class="Lands.Views.LandsPage"

Fuente de datos del ListView

BindingContext="{Binding Main,Source={StaticResource Locator}}"

Title="Lands">

<ContentPage.Content>

Refresca al hacer Scroll hacia abajo

<StackLayout BindingContext="{Binding Lands}">

<ListView BackgroundColor="Silver"

HasUnevenRows="true"

IsPullToRefreshEnabled="true"

Los ListView tienen ActivityIndicator integrado. La propiedad es IsRefreshing

IsRefreshing="{Binding IsRefreshing, Mode=TwoWay}"

ItemsSource="{Binding Lands}"

SeparatorVisibility="None"

RefreshCommand="{Binding RefreshCommand}">

<ListView.ItemTemplate>

Este es el comando que se ejecuta cuando se hace el scrollview hacia abajo. Este comando hay que hacerlo en la LoginViewModel

<DataTemplate>

<ViewCell>

<Frame CornerRadius="20"

HasShadow="true"

Margin="5">

<Frame.GestureRecognizers>

<TapGestureRecognizer Command="{Binding SelectLandCommand}" />

Para personalizar una ListView se usa DataTemplate.

Dentro de eso se usa ViewCell

</Frame.GestureRecognizers>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="Auto" />

</Grid.ColumnDefinitions>

<StackLayout Grid.Column="0"

VerticalOptions="Center">

<Label FontAttributes="Bold"

País

FontSize="Medium"

Text="{Binding Name}"

TextColor="Black"></Label>

<Label Text="{Binding Capital}"

Capital

TextColor="Black"></Label>

</StackLayout>

<Image Grid.Column="1"

HorizontalOptions="End"

Source="ic\_chevron\_right"></Image>

</Grid>

</Frame>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage.Content>

</ContentPage>

Agregamos en LandsViewModels

#region Attributes

private ObservableCollection<Land> lands;

private bool isRefreshing;

#endregion

#region Properties

public ObservableCollection<Land> Lands

{

get { return this.lands; }

set { SetValue(ref this.lands, value); }

}

public bool IsRefreshing

{

get { return this.isRefreshing; }

set { SetValue(ref this.isRefreshing, value); }

}

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

#region Methods

private async void LoadLands()

{

//Chequeamos si hay conección a Internet

this.IsRefreshing = true;

var connection = await this.apiService.CheckConnection();

if (!connection.IsSuccess)

{

this.IsRefreshing = false;

await Application.Current.MainPage.DisplayAlert(

"Error",

connection.Message,

"Accept");

await Application.Current.MainPage.Navigation.PopAsync();

return;

}

//Conectamos a la API de países

var response = await this.apiService.GetList<Land>(

"https://restcountries.eu",

"/rest",

"/v2/all");

if (!response.IsSuccess)

{

this.IsRefreshing = false;

await Application.Current.MainPage.DisplayAlert(

"Error",

response.Message,

"Accept");

await Application.Current.MainPage.Navigation.PopAsync();

return;

}

//Pasamos la lista de pa{ises a una ObservableConnection

var list = (List<Land>)response.Result;

this.Lands = new ObservableCollection<Land>(list);

this.IsRefreshing = false;

}

Agregamos también:

#region Commands

public ICommand RefreshCommand

{

get

{

return new RelayCommand(LoadLands);

}

}

#endregion

Filter hay que implementarlo

**Colocar un buscador a la ListView:**

SearchCommand hay que implementarlo

<SearchBar

BackgroundColor="Silver"

HeightRequest="46"

Text="{Binding Filter, Mode=TwoWay}"

Placeholder="Search..."

SearchCommand="{Binding SearchCommand}"></SearchBar>

En LandsVieModels agregamos:

#region Attributes

private ObservableCollection<Land> lands;

private bool isRefreshing;

private string filter;

#endregion

#region Properties

public ObservableCollection<Land> Lands

{

get { return this.lands; }

set { SetValue(ref this.lands, value); }

}

public bool IsRefreshing

{

get { return this.isRefreshing; }

set { SetValue(ref this.isRefreshing, value); }

}

public string Filter

{

get { return this.filter; }

set { SetValue(ref this.filter, value); }

Siempre en LandViewModels, dentro de la región Commands implementamos lo siguiente:

public ICommand SearchCommand

{

get

{

Si Filter esta vacío, llenamos la ObservableCollection con la lista completa de países

return new RelayCommand(Search);

}

}

private void Search()

{

if (string.IsNullOrEmpty(this.Filter))

{

Si Filter no esta vacío, llenamos la ObservableCollection con la lista con el Filtro

this.Lands = new ObservableCollection<Land>(this.landList);

}

else

{

this.Lands = new ObservableCollection<Land>(this.landList.Where(

l =>l.Name.ToLower().Contains(this.Filter.ToLower())

|| l.Capital.ToLower().Contains(this.Filter.ToLower())));

}

Filtro de País o Capital. Y usamos ToLower para poner minúsculas a ambos lados de la comparación

}

Hay que poner using Linq.

Lo que viene después delWhere se llama expresión lambda.

Para no perder la lista de países que está en memoria, hacemos el siguiente cambio.

Ponemos la Lista como un atributo de la clase:

#region Attributes

private ObservableCollection<Land> lands;

private bool isRefreshing;

private string filter;

private List<Land> landList;

Y cambiamos en la region de los Methods, en LoadLands:

var list = (List<Land>)response.Result; (la list está como variable local)

por

this.landList = (List<Land>)response.Result;

Para filtrar a medida que se digite en Search, se hace lo siguiente.

En el set del Filter se agrega:

public string Filter

{

get { return this.filter; }

set

{

SetValue(ref this.filter, value);

this.Search();

}

}

Actualiza el valor del Filtro y luego ejecuta la búsqueda.

**Video 14**

Ponemos en la ListView un contenedor Frame, luego de ViewCell:

<?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:d="http://xamarin.com/schemas/2014/forms/design"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

mc:Ignorable="d"

x:Class="Lands.Views.LandsPage"

BindingContext="{Binding Main,Source={StaticResource Locator}}"

Title="Lands">

<ContentPage.Content>

<StackLayout BindingContext="{Binding Lands}">

Color de fondo de la barra de búsqueda

<SearchBar BackgroundColor="Silver"

HeightRequest="46"

Text="{Binding Filter, Mode=TwoWay}"

Placeholder="Search..."

SearchCommand="{Binding SearchCommand}"></SearchBar>

<ListView BackgroundColor="Silver"

Color de fondo de la ListView

HasUnevenRows="true"

IsPullToRefreshEnabled="true"

IsRefreshing="{Binding IsRefreshing, Mode=TwoWay}"

ItemsSource="{Binding Lands}"

SeparatorVisibility="None"

RefreshCommand="{Binding RefreshCommand}">

<ListView.ItemTemplate>

Frame tiene unas propiedades que no hay en otros controles.

Margin: margen

CornerRadius:redondea las esquinas

HasShadow:pone una sombra

<DataTemplate>

<ViewCell>

<Frame CornerRadius="20"

HasShadow="true"

Margin="5">

<Frame.GestureRecognizers>

<TapGestureRecognizer Command="{Binding SelectLandCommand}" />

</Frame.GestureRecognizers>

Como usamos lo anterior, ponemos SeparatorVisibility=”none” (rayitas que separan los ítems de un ListView)

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="Auto" />

</Grid.ColumnDefinitions>

<StackLayout Grid.Column="0"

VerticalOptions="Center">

<Label FontAttributes="Bold"

FontSize="Medium"

Text="{Binding Name}"

TextColor="Black"></Label>

<Label Text="{Binding Capital}"

TextColor="Black"></Label>

</StackLayout>

<Image Grid.Column="1"

HorizontalOptions="End"

Source="ic\_chevron\_right"></Image>

</Grid>

</Frame>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage.Content>

</ContentPage>

Lo próximo es que al tocar un país, pase a otra Page llamada LandPage, que muestra la info de ese país.

Creamos en la carpeta Views, la LandPage. Y creamos una Clase LandViewModel

namespace Lands.ViewModels

{

public class LandViewModel : BaseViewModel

{

}

}

En la MainViewModel agregamos la propiedad

namespace Lands.ViewModels

{

public class MainViewModel

{

#region Singleton

private static MainViewModel instance;

public static MainViewModel GetInstance()

{

if (instance == null)

{

return new MainViewModel();

}

return instance;

}

#endregion

#region ViewModels

public LoginViewModel Login { get; set; }

public LandsViewModel Lands { get; set; }

public LandViewModel Land { get; set; }

#endregion

#region constructors

public MainViewModel()

{

instance = this;

this.Login = new LoginViewModel();

}

#endregion

}

}

En LandPage.xaml agregamos:

<?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:d="http://xamarin.com/schemas/2014/forms/design"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

mc:Ignorable="d"

x:Class="Lands.Views.LandPage"

BindingContext="{Binding Main,Source={StaticResource Locator}}"

Title="Land">

<ContentPage.Content>

<StackLayout>

<Label Text="Land"

VerticalOptions="CenterAndExpand"

HorizontalOptions="CenterAndExpand" />

</StackLayout>

</ContentPage.Content>

</ContentPage>

Para navegar desde LandsPage (en la ListView de países) a la LandPage, quiero que pase al tocar en cualquier lado de la fila.

Se usa lo que se llama GestureRecognizer. Se lo ponemos al Frame de la ListView:

<Frame CornerRadius="20"

HasShadow="true"

Margin="5">

<Frame.GestureRecognizers>

<TapGestureRecognizer Command="{Binding SelectLandCommand}" />

</Frame.GestureRecognizers>

<Grid>

Y con esto se pasa a la LandPage.

Este comando SelectLandCommand hay que hacerlo

SelectLandCommand NO TIENE que ir en la LandsVieModel.

Debería ir en la Clase Land que es la que tiene las propiedades del país seleccionado.

Pero Land es Modelo y por eso es mejor no tocarlo.

Si le ponemos un Comando a un modelo, deja de ser Modelo para pasar a ser una ViewModel.

Creamos en la Carpeta ViewModels una Clase llamada LandItemViewModel

namespace Lands.ViewModels

{

using Models;

public class LandItemViewModel: Land

{

}

}

Hay que ponerle using Models

Hasta aquí, la Clase LandItemViewModel es un ESPEJO de la Clase Land. Son iguales.

Pero ahora, la ObservableCollection en vez de armarla sobre Land, la armamos sobre LanItemViewModel

En LandsViewModel cambiamos en

#region Attributes

private ObservableCollection<Land> lands;

private bool isRefreshing;

private string filter;

private List<Land> landList;

por

#region Attributes

private ObservableCollection<LandItemViewModel> lands;

private bool isRefreshing;

private string filter;

private List<Land> landList;

Al hacer eso también hay que corregir la propiedad:

Cambiamos en

public ObservableCollection<Land> Lands

{

get { return this.lands; }

set { SetValue(ref this.lands, value); }

}

por

#region Properties

public ObservableCollection<LandItemViewModel> Lands

{

get { return this.lands; }

set { SetValue(ref this.lands, value); }

}

Y en LoadLands

//Pasamos la lista de países a una ObservableConnection

this.landList = (List<Land>)response.Result;

this.Lands = new ObservableCollection<LandItemViewModel>(this.landList);

this.IsRefreshing = false;

y en el Filtro

this.Lands = new ObservableCollection<LandItemViewModel>(this.landList.Where(

l =>l.Name.ToLower().Contains(this.Filter.ToLower())

|| l.Capital.ToLower().Contains(this.Filter.ToLower())));

también se cambia, pero quedan unos errores (lo pintado en rojo).

Esto pasa porque las listas quedan desde diferentes orígenes, una desde la Clase Land y otra desde la Clase LandItemViewModel

No son compatibles.

Hay que hacer una transformación de listas

Hay que agarrar la lista de tipo Land y transformarla en una lista de tipo LandItemViewModel

Como eso se hará muchas veces, se hace un método para ello.

Lo llamaremos ToLandItemViewModel

Entonces lo anterior queda así:

En LoadLands:

//Pasamos la lista de países a una ObservableConnection

this.landList = (List<Land>)response.Result;

this.Lands = new ObservableCollection<LandItemViewModel>(this.ToLandItemViewModel());

this.IsRefreshing = false;

y en el Filtro:

this.Lands = new ObservableCollection<LandItemViewModel>(this.ToLandItemViewModel().Where(

l =>l.Name.ToLower().Contains(this.Filter.ToLower())

|| l.Capital.ToLower().Contains(this.Filter.ToLower())));

Ahora falta armar ese método. Se copia desde el profesor:

#region Methods

private IEnumerable<LandItemViewModel> ToLandItemViewModel()

{

return this.landList.Select(l => new LandItemViewModel

{

Alpha2Code = l.Alpha2Code,

Alpha3Code = l.Alpha3Code,

AltSpellings = l.AltSpellings,

Area = l.Area,

Borders = l.Borders,

CallingCodes = l.CallingCodes,

Capital = l.Capital,

Cioc = l.Cioc,

Currencies = l.Currencies,

Demonym = l.Demonym,

Flag = l.Flag,

Gini = l.Gini,

Languages = l.Languages,

Latlng = l.Latlng,

Name = l.Name,

NativeName = l.NativeName,

NumericCode = l.NumericCode,

Population = l.Population,

Region = l.Region,

RegionalBlocs = l.RegionalBlocs,

Subregion = l.Subregion,

Timezones = l.Timezones,

TopLevelDomain = l.TopLevelDomain,

Translations = l.Translations,

});

}

#endregion

**Video 15**

Está pendiente el comando SelectLandCommand que está en LandsPage.xaml

Ese comando va en la LandItemViewModel

namespace Lands.ViewModels

{

using GalaSoft.MvvmLight.Command;

using Views;

using Models;

using System.Windows.Input;

using Xamarin.Forms;

public class LandItemViewModel: Land

Creamos el Comando SelectLandCommand

{

#region Commands

public ICommand SelectLandCommand

{

get

{

Select Command nos lleva a la LandPage.

Pero antes de pintar la LandPage, hay que instanciar la VIewModel relacionada a LandPage

return new RelayCommand(SelectLand);

}

}

private async void SelectLand()

{

MainViewModel.GetInstance().Land = new LandViewModel(this);

await Application.Current.MainPage.Navigation.PushAsync(new LandPage());

}

#endregion

}

}

Así se pasa el país seleccionado.

Poniendo this se pasa la Clase actual completa

En LandViewModel se crea el Constructor

namespace Lands.ViewModels

{

using Models;

public class LandViewModel

{

#region Properties

public Land Land

{

get;

set;

}

#endregion

#region Constructors

public LandViewModel(Land land)

{

this.Land = land;

}

#endregion

}

}

**LanPage.xaml**

<?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:d="http://xamarin.com/schemas/2014/forms/design"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

mc:Ignorable="d"

x:Class="Lands.Views.LandPage"

BindingContext="{Binding Main,Source={StaticResource Locator}}"

Title="Land">

ScrollVIew

<ContentPage.Content>

<ScrollView

BindingContext="{Binding Land}">

<StackLayout

Binding a Land

Padding="5">

<Label

FontSize=" 15"

FontAttributes="Bold"

HorizontalOptions="Center"

Text="{Binding Land.Name}"

País arriba centrado

TextColor="Black">

</Label>

<Grid>

<Grid.ColumnDefinitions>

En la Grilla ponemos el resto de datos en dos columnas

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="2\*" />

</Grid.ColumnDefinitions>

<Label Grid.Column="0"

Grid.Row="0"

Text="Capital"

TextColor="Black"></Label>

Capital

<Label Grid.Column="1"

Grid.Row="0"

FontAttributes="Bold"

Text="{Binding Land.Capital}"

TextColor="Black"></Label>

<Label Grid.Column="0"

Grid.Row="1"

Text="Population"

TextColor="Black"></Label>

Población

<Label Grid.Column="1"

Grid.Row="1"

FontAttributes="Bold"

Text="{Binding Land.Population, StringFormat='{0:N0}'}"

TextColor="Black"></Label>

<Label Grid.Column="0"

Grid.Row="2"

Text="Area"

Para formatear un número con 0 decimales

TextColor="Black"></Label>

<Label Grid.Column="1"

Grid.Row="2"

FontAttributes="Bold"

Text="{Binding Land.Area, StringFormat='{0:N0}'}"

TextColor="Black"></Label>

<Label Grid.Column="0"

Grid.Row="3"

Text="Alpha Code 2"

TextColor="Black"></Label>

<Label Grid.Column="1"

Grid.Row="3"

FontAttributes="Bold"

Text="{Binding Land.Alpha2Code}"

TextColor="Black"></Label>

<Label Grid.Column="0"

Grid.Row="4"

Text="Alpha Code 3"

TextColor="Black"></Label>

<Label Grid.Column="1"

Grid.Row="4"

FontAttributes="Bold"

Text="{Binding Land.Alpha3Code}"

TextColor="Black"></Label>

<Label Grid.Column="0"

Grid.Row="5"

Text="Region"

TextColor="Black"></Label>

<Label Grid.Column="1"

Grid.Row="5"

FontAttributes="Bold"

Text="{Binding Land.Region}"

TextColor="Black"></Label>

<Label Grid.Column="0"

Grid.Row="6"

Text="Subregion"

TextColor="Black"></Label>

<Label Grid.Column="1"

Grid.Row="6"

FontAttributes="Bold"

Text="{Binding Land.Subregion}"

TextColor="Black"></Label>

<Label Grid.Column="0"

Grid.Row="7"

Text="Demonym"

TextColor="Black"></Label>

<Label Grid.Column="1"

Grid.Row="7"

FontAttributes="Bold"

Text="{Binding Land.Demonym}"

TextColor="Black"></Label>

<Label Grid.Column="0"

Grid.Row="8"

Text="GINI"

TextColor="Black"></Label>

<Label Grid.Column="1"

Grid.Row="8"

FontAttributes="Bold"

Text="{Binding Land.Gini, StringFormat='{0:N2}'}"

TextColor="Black"></Label>

<Label Grid.Column="0"

Grid.Row="9"

Text="Native Name"

TextColor="Black"></Label>

<Label Grid.Column="1"

Grid.Row="9"

FontAttributes="Bold"

Text="{Binding Land.NativeName}"

TextColor="Black"></Label>

<Label Grid.Column="0"

Grid.Row="10"

Text="Numeric Code}"

TextColor="Black"></Label>

<Label Grid.Column="1"

Grid.Row="10"

FontAttributes="Bold"

Text="{Binding Land.NumericCode}"

TextColor="Black"></Label>

<Label Grid.Column="0"

Grid.Row="11"

Text="CIOC"

TextColor="Black"></Label>

<Label Grid.Column="1"

Grid.Row="11"

FontAttributes="Bold"

Text="{Binding Land.Cioc}"

TextColor="Black"></Label>

</Grid>

</StackLayout>

</ScrollView>

</ContentPage.Content>

</ContentPage>

**Video 16**

Vamos a ver una TabbedPage

En la carpeta VIews ponemos estas Pages:

* LandTabbedPage

**LandTabbedPage.xaml**

<?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:d="http://xamarin.com/schemas/2014/forms/design"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

mc:Ignorable="d"

x:Class="Lands.Views.LandTabbedPage">

<ContentPage.Content>

<StackLayout>

Cambiamos ContetPage por TabbedPage

<Label Text="Welcome to Xamarin.Forms!"

VerticalOptions="CenterAndExpand"

HorizontalOptions="CenterAndExpand" />

</StackLayout>

</ContentPage.Content>

Este ContentPage lo borramos

</ContentPage>

**LandTabbedPage.xaml.cs**

namespace Lands.Views

{

using Xamarin.Forms;

using Xamarin.Forms.Xaml;

[XamlCompilation(XamlCompilationOptions.Compile)]

public partial class LandTabbedPage : ContentPage

{

public LandTabbedPage()

{

InitializeComponent();

}

}

}

Agregamos a **LandTabbedPage.xaml**

<?xml version="1.0" encoding="utf-8" ?>

<TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:d="http://xamarin.com/schemas/2014/forms/design"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

mc:Ignorable="d"

xmlns:tabs="clr-namespace:Lands.Views"

x:Class="Lands.Views.LandTabbedPage"

Title="Land">

<TabbedPage.Children>

<tabs:LandPage/>

<tabs:BordersPage/>

<tabs:CurrenciesPage/>

<tabs:TranslationsPage/>

<tabs:LanguagesPage/>

</TabbedPage.Children>

</TabbedPage>

Tenemos que crear estas páginas, salvo LandPage que ya la tenemos,

A LandPage le cambiamos el título:

Title="Land"> por Title="Information">

Creamos en la carpeta Views:

BorderPage

CurrenciesPage

TranslatiosnsPage

LanguagesPage

Las Pages de una TabbedPage conviene que compartan una única ViewModel

Copiamos desde LandPage

BindingContext="{Binding Main,Source={StaticResource Locator}}"

Title="Information">

y lo pegamos en cada una de las otras Page

Para que ahora la App navegue hacia la TabbedPage, en LandItemViewModel cambiamos:

private async void SelectLand()

{

MainViewModel.GetInstance().Land = new LandViewModel(this);

await Application.Current.MainPage.Navigation.PushAsync(new LandPage());

}

por

private async void SelectLand()

{

MainViewModel.GetInstance().Land = new LandViewModel(this);

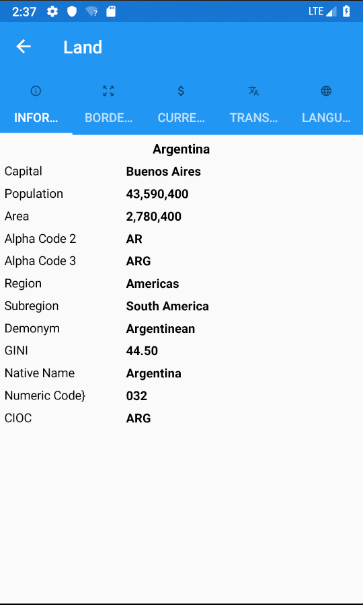
await Application.Current.MainPage.Navigation.PushAsync(new LandTabbedPage());

}

Agregamos los archivos de íconos para las pestañas en las carpetas Resources de los proyectos Android e Ios.

En las Page de cada pestaña se agrega

Icon="ic\_info\_outline"



Veamos la info que hay de Argentina en la API:

{

"name": "Argentina",

"topLevelDomain": [

".ar"

],

"alpha2Code": "AR",

"alpha3Code": "ARG",

"callingCodes": [

"54"

],

"capital": "Buenos Aires",

"altSpellings": [

"AR",

"Argentine Republic",

"República Argentina"

],

"region": "Americas",

"subregion": "South America",

"population": 43590400,

"latlng": [

-34,

-64

],

"demonym": "Argentinean",

"area": 2780400,

"gini": 44.5,

"timezones": [

"UTC-03:00"

],

"borders": [

"BOL",

"BRA",

"CHL",

"PRY",

"URY"

],

"nativeName": "Argentina",

"numericCode": "032",

"currencies": [

{

"code": "ARS",

"name": "Argentine peso",

"symbol": "$"

}

],

"languages": [

{

"iso639\_1": "es",

"iso639\_2": "spa",

"name": "Spanish",

"nativeName": "Español"

},

{

"iso639\_1": "gn",

"iso639\_2": "grn",

"name": "Guaraní",

"nativeName": "Avañe'ẽ"

}

],

"translations": {

"de": "Argentinien",

"es": "Argentina",

"fr": "Argentine",

"ja": "アルゼンチン",

"it": "Argentina",

"br": "Argentina",

"pt": "Argentina",

"nl": "Argentinië",

"hr": "Argentina",

"fa": "آرژانتین"

},

"flag": "https://restcountries.eu/data/arg.svg",

"regionalBlocs": [

{

"acronym": "USAN",

"name": "Union of South American Nations",

"otherAcronyms": [

"UNASUR",

"UNASUL",

"UZAN"

],

"otherNames": [

"Unión de Naciones Suramericanas",

"União de Nações Sul-Americanas",

"Unie van Zuid-Amerikaanse Naties",

"South American Union"

]

}

],

"cioc": "ARG"

}

Los bordes llegan con la iniciales internacionales de cada país

Lo pintado en amarillo es lo que pusimos en la LandPage

En BordersPage queremos mostrar no solo las siglas sino también el nombre de cada país.

Serán 2 columnas, sigla y nombre, en un ListView.

Vamos a mandar a la MainViewModel la lista de países, para disponer de ella en cualquier parte.

**Video 17**

Creamos en la Carpeta Models un modelo Border

namespace Lands.Models

{

public class Border

{

public string Code { get; set; }

public string Name { get; set; }

}

}

En MainViewModel creamos una propiedad

#region Properties

public List<Land> LandsList { get; set; }

#endregion

En LandsViewModel quitamos la propiedad

private List<Land> landList;

y más abajo cambiamos

this.landList = (List<Land>)response.Result;

por

MainViewModel .GetInstance().LandsList = (List<Land>)response.Result;

Y también cambiamos

return this.landList.Select(l => new LandItemViewModel

por

return MainViewModel.GetInstance().LandsList.Select(l => new LandItemViewModel

Acá lo único que hemos hecho es pasar la lista que estaba en la LandsViewModel a la MainViewModel como una propiedad.

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

Cuando entremos a la LandViewModel, hay que convertir el objeto Border a una ObservableCollection para poder ponerlo en una ListView.

En LandViewModel creamos:

#region Attributes

private ObservableCollection<Border> borders;

#endregion

También agregamos la propiedad:

#region Properties

public Land Land

{

get;

set;

}

public ObservableCollection<Border> Borders

{

get { return this.borders; };

set { this.SetValue(ref this.borders, value); };

}

Hacemos que la Clase herede de BaseViewModel

public class LandViewModel :BaseViewModel

En la región de los Constructores agregamos:

#region Constructors

public LandViewModel(Land land)

{

this.Land = land;

this.LoadBorders();

}

#endregion

Y en la región de los Methods creamos ese procedimiento

#region Methods

private void LoadBorders()

{

this.Borders = new ObservableCollection<Border>();

foreach (var border in this.Land.Borders)

{

var land = MainViewModel.GetInstance().LandsList

.Where(l => l.Alpha3Code == border).

FirstOrDefault();

if (land != null)

{

this.Borders.Add(new Border

{

Code = land.Alpha3Code,

Name = land.Name

});

}

}

}

#endregion

**Video 18**

CurrenciesPage

"currencies": [

{

"code": "ARS",

"name": "Argentine peso",

"symbol": "$"

}

Hay que hacer 3 columnas en una ListView. Pero no es una ObservableCollection….hay que crearla en la LandViewModel.

#region Attributes

private ObservableCollection<Border> borders;

private ObservableCollection<Currency> currencies;

#endregion

#region Properties

public Land Land

{

get;

set;

}

public ObservableCollection<Border> Borders

{

get { return this.borders; }

set { this.SetValue(ref this.borders, value); }

}

public ObservableCollection<Currency> Currencies

{

get { return this.currencies; }

set { this.SetValue(ref this.currencies, value); }

#region Constructors

public LandViewModel(Land land)

{

this.Land = land;

this.LoadBorders();

this.Currencies = new ObservableCollection<Currency>(this.Land.Currencies);

}

#endregion

Para CurrenciesPage.xaml, copiamos todo el StackLayout de Border, lo pegamos y lo corregimos.

<?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="Lands.Views.CurrenciesPage"

BindingContext="{Binding Main,Source={StaticResource Locator}}"

Icon="ic\_attach\_money"

Title="Currencies">

<ContentPage.Content>

<StackLayout BindingContext="{Binding Land}"

Padding="5">

<Label FontSize="Large"

FontAttributes="Bold"

HorizontalOptions="Center"

Text="{Binding Land.Name}"

TextColor="Red"></Label>

<ListView HasUnevenRows="True"

ItemsSource="{Binding Currencies}">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="3\*" />

<ColumnDefinition Width="\*" />

</Grid.ColumnDefinitions>

<Label FontAttributes="Bold"

FontSize="Medium"

Grid.Column="0"

Text="{Binding Code}"

TextColor="Blue"></Label>

<Label Text="{Binding Name}"

VerticalOptions="Center"

Grid.Column="1"

TextColor="Black"></Label>

<Label Text="{Binding Symbol}"

VerticalOptions="Center"

Grid.Column="2"

TextColor="Black"></Label>

</Grid>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage.Content>

</ContentPage>

TranslationsPage

"translations": {

"de": "Argentinien",

"es": "Argentina",

"fr": "Argentine",

"ja": "アルゼンチン",

"it": "Argentina",

"br": "Argentina",

"pt": "Argentina",

"nl": "Argentinië",

"hr": "Argentina",

"fa": "آرژانتین"

No hay que hacer una ObservableCollection ya que cada idioma viene como un campo separado.

Solo hay que hacer la TranslationsPage.xaml

<?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:d="http://xamarin.com/schemas/2014/forms/design"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

mc:Ignorable="d"

x:Class="Lands.Views.TranslationsPage"

BindingContext="{Binding Main, Source={StaticResource Locator}}"

Icon="ic\_translate"

Title="Translations">

<ContentPage.Content>

<StackLayout BindingContext="{Binding Land}"

Padding="5">

<Label FontAttributes="Bold"

FontSize="Large"

HorizontalOptions="Center"

Text="{Binding Land.Name}"

TextColor="Red"></Label>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="2\*" />

</Grid.ColumnDefinitions>

<Label Grid.Column="0"

Grid.Row="0"

Text="Germany"

TextColor="Blue"></Label>

<Label Grid.Column="1"

Grid.Row="0"

FontAttributes="Bold"

Text="{Binding Land.Translations.Germany}"

TextColor="Black"></Label>

<Label Grid.Column="0"

Grid.Row="1"

Text="Spanish"

TextColor="Blue"></Label>

<Label Grid.Column="1"

Grid.Row="1"

FontAttributes="Bold"

Text="{Binding Land.Translations.Spanish}"

TextColor="Black"></Label>

<Label Grid.Column="0"

Grid.Row="2"

Text="French"

TextColor="Blue"></Label>

<Label Grid.Column="1"

Grid.Row="2"

FontAttributes="Bold"

Text="{Binding Land.Translations.French}"

TextColor="Black"></Label>

<Label Grid.Column="0"

Grid.Row="3"

Text="Japanese"

TextColor="Blue"></Label>

<Label Grid.Column="1"

Grid.Row="3"

FontAttributes="Bold"

Text="{Binding Land.Translations.Japanese}"

TextColor="Black"></Label>

<Label Grid.Column="0"

Grid.Row="4"

Text="Italian"

TextColor="Blue"></Label>

<Label Grid.Column="1"

Grid.Row="4"

FontAttributes="Bold"

Text="{Binding Land.Translations.Italian}"

TextColor="Black"></Label>

<Label Grid.Column="0"

Grid.Row="5"

Text="Brazilian"

TextColor="Blue"></Label>

<Label Grid.Column="1"

Grid.Row="5"

FontAttributes="Bold"

Text="{Binding Land.Translations.Brazilian}"

TextColor="Black"></Label>

<Label Grid.Column="0"

Grid.Row="6"

Text="Portuguese"

TextColor="Blue"></Label>

<Label Grid.Column="1"

Grid.Row="6"

FontAttributes="Bold"

Text="{Binding Land.Translations.Portuges}"

TextColor="Black"></Label>

<Label Grid.Column="0"

Grid.Row="7"

Text="Dutch"

TextColor="Blue"></Label>

<Label Grid.Column="1"

Grid.Row="7"

FontAttributes="Bold"

Text="{Binding Land.Translations.Dutch}"

TextColor="Black"></Label>

<Label Grid.Column="0"

Grid.Row="8"

Text="Croatian"

TextColor="Blue"></Label>

<Label Grid.Column="1"

Grid.Row="8"

FontAttributes="Bold"

Text="{Binding Land.Translations.Croatian}"

TextColor="Black"></Label>

<Label Grid.Column="0"

Grid.Row="9"

Text="Persian"

TextColor="Blue"></Label>

<Label Grid.Column="1"

Grid.Row="9"

FontAttributes="Bold"

Text="{Binding Land.Translations.Persian}"

TextColor="Black"></Label>

</Grid>

</StackLayout>

</ContentPage.Content>

</ContentPage>

LanguagesPage

"languages": [

{

"iso639\_1": "es",

"iso639\_2": "spa",

"name": "Spanish",

"nativeName": "Español"

},

{

"iso639\_1": "gn",

"iso639\_2": "grn",

"name": "Guaraní",

"nativeName": "Avañe'ẽ"

}

Es parecido al de Currencies. Hacemos lo mismo.

Hay que hacer 4 columnas en una ListView. Pero no es una ObservableCollection….hay que crearla en la LandViewModel.

#region Attributes

private ObservableCollection<Border> borders;

private ObservableCollection<Currency> currencies;

private ObservableCollection<Language> languages;

#endregion

#region Properties

public Land Land

{

get;

set;

}

public ObservableCollection<Border> Borders

{

get { return this.borders; }

set { this.SetValue(ref this.borders, value); }

}

public ObservableCollection<Currency> Currencies

{

get { return this.currencies; }

set { this.SetValue(ref this.currencies, value); }

}

public ObservableCollection<Language> Languages

{

get { return this.languages; }

set { this.SetValue(ref this.languages, value); }

}

#region Constructors

public LandViewModel(Land land)

{

this.Land = land;

this.LoadBorders();

this.Currencies = new ObservableCollection<Currency>(this.Land.Currencies);

this.Languages = new ObservableCollection<Language>(this.Land.Languages);

}

#endregion

Para la LanguagesPage.xaml, como es parecido a CurrenciesPage.xaml, copiamos el StackLayout y luego lo modificamos.

<?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="Lands.Views.LanguagesPage"

BindingContext="{Binding Main,Source={StaticResource Locator}}"

Icon="ic\_language"

Title="Languages">

<ContentPage.Content>

<StackLayout BindingContext="{Binding Land}"

Padding="5">

<Label FontSize="Large"

FontAttributes="Bold"

HorizontalOptions="Center"

Text="{Binding Land.Name}"

TextColor="Red"></Label>

<ListView HasUnevenRows="True"

ItemsSource="{Binding Languages}">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="2\*" />

<ColumnDefinition Width="2\*" />

</Grid.ColumnDefinitions>

<Label FontAttributes="Bold"

FontSize="Medium"

Grid.Column="0"

Text="{Binding Iso6392}"

TextColor="Blue"></Label>

<Label Text="{Binding Name}"

VerticalOptions="Center"

Grid.Column="1"

TextColor="Black"></Label>

<Label Text="{Binding NativeName}"

VerticalOptions="Center"

Grid.Column="2"

TextColor="Black"></Label>

</Grid>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage.Content>

</ContentPage>

**Video 19 y 20**

Creamos 3 nuevos Proyectos

* Lands.Domain: de tipo Biblioteca de Clases .NET Framework
* Lands.Backend: de tipo web ASP.NET .NET Framework – MVC – Autenticación Cuentas de Usuario Individuales
* Lands.API: de tipo web ASP.NET .NET Framework – APIWeb – Autenticación Cuentas de Usuario Individuales

Hay que crear una cuenta en Azure

Esta es la Web de Azure 🡪 portal.azure.com

Resource group Xamarin

Database name Lands

Nombre del servidor luisnu East US

Inicio de sesión del administrador del servidor luisalbiazul

Contraseña Espora2052\*

El Backend queda publicado en:

Landsbackendluisnu.azurewebsites.net 🡪 <https://landsbackendluisnu.azurewebsites.net/>

El API queda publicado en:

LandsAPIluisnu.azurewebsites.net 🡪 <https://landsapiluisnu.azurewebsites.net/>

**Video 21**

Sacamos (o comentamos) de la LoginViewModel la parte donde habíamos puesto internamente un mail y un password

//CON ESTA PARTE SE LOGUEABA DIRECTAMENTE

//if (this.Email != "ln" || this.Password != "123") // Alt 124 para |

//{

// this.IsRunning = false;

// this.IsEnabled = true;

// await Application.Current.MainPage.DisplayAlert(

// "Error",

// "Email or password incorrect",

// "Accept");

// return;

//}

Para usar el Servicio hay que instanciarlo. Siempre dentro de LoginViewModel, ponemos una región:

#region Services

private ApiService apiService

#endregion

Y en el Constructor:

#region Constructors

public LoginViewModel()

{

this.apiService = new ApiService;

this.IsRemembered = true;

this.IsEnabled = true;

//Email = "ln";

//Password = "123";

}

#endregion

Dentro de LoginCommand:

1) hay que validar si hay conexión.

var connection = await this.apiService.CheckConnection();

if (!connection.IsSuccess)

{

this.IsRunning = true;

this.IsEnabled = false;

await Application.Current.MainPage.DisplayAlert(

"Error",

connection.Message,

"Accept");

return;

}

2) hay que validar que se genere el Token

var token = await this.apiService.GetToken(

"https://landsapiluisnu.azurewebsites.net",

this.Email,

this.Password);

if (token==null)

{

this.IsRunning = true;

this.IsEnabled = false;

await Application.Current.MainPage.DisplayAlert(

"Error",

"Something was wrong, please try later",

"Accept");

return;

}

Finalmente:

//Acá llega sin email y password son los correctos

//Acá guardo el Token para usarlo posteriormente por si se quiere cambiar la contraseña

var mainViewModel = MainViewModel.GetInstance();

mainViewModel.Token = token;

this.IsRunning = false;

this.IsEnabled = true;

this.Email = string.Empty;

this.Password = string.Empty;

mainViewModel.Lands = new LandsViewModel();

await Application.Current.MainPage.Navigation.PushAsync(new LandsPage());

this.Password = string.Empty;

return;

**Video 22**

Seguir instructivo:

Cargar Imagenes con tipo de extension SVG.docx

**Video 23**

**Carpetas, Clases y otros elementos que se van incorporando**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Video** | **Tema** | **Objeto** | **Lugar** | **Nombre** |
| 2 | Arquitectura MVVM | Carpeta | Proyecto Lands | Models |
| 2 | Arquitectura MVVM | Carpeta | Proyecto Lands | Views |
| 2 | Arquitectura MVVM | Carpeta | Proyecto Lands | ViewModels |
| 3 | Login | Forms.ContentPage | Carpeta Views | LoginPage |
| 5 | Arquitectura MVVM | Clase | Carpeta ViewsModels | MainViewModel |
| 5 | Arquitectura MVVM | Carpeta | Proyecto Lands | Infrastructure |
| 5 | Arquitectura MVVM | Clase | Infrastructure | InstanceLocator |
| 7 | Arquitectura MVVM | Clase | Carpeta ViewsModels | LoginViewModel |
| 8 | Arquitectura MVVM | Clase | Carpeta ViewsModels | BaseViewModel |
| 9 | Arquitectura MVVM | Page | Carpeta Views | LandsPage |
| 9 | Arquitectura MVVM | Clase | Carpeta ViewsModels | LandsViewModel |
| 9 | API Modelos | Clase | Carpeta Models | Currrency |
| 9 | API Modelos | Clase | Carpeta Models | Languages |
| 9 | API Modelos | Clase | Carpeta Models | Translations |
| 9 | API Modelos | Clase | Carpeta Models | RegionalBloc |
| 9 | API Modelos | Clase | Carpeta Models | Land |
| 10 | Conectividad | Clase | Carpeta Models | Response |
| 10 | Conectividad | Carpeta | Proyecto Lands | Services |
| 10 | Conectividad | Clase | Carpeta Services | ApiService |
| 10 | Conectividad | Clase | Models | TokenResponse |
| 12 | Iconos | Archivos | Carpetas Resources de Android e Ios | Crevron distintos tamaños |
| 14 | Info de un país | Page | Carpeta Views | LandPage |
| 14 | Info de un país | Clase | Carpeta ViewsModels | LandViewModel |
| 14 | Info de un país | Clase | Carpeta ViewsModels | LandItemViewModel |
| 15 | TabbedPage | Page | Carpeta Views | BordersPage |
| 15 | TabbedPage | Page | Carpeta Views | CurrenciesPage |
| 15 | TabbedPage | Page | Carpeta Views | TranslationsPage |
| 15 | TabbedPage | Page | Carpeta Views | LanguagesPage |
| 17 | BorderPage | Modelo | Carpeta Models | Border |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Nuggets que se incorporan a la Aplicación**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Video** | **Tema** | **Nugget** | **Lugar** | **Sirve para** |
| 7 | Botones | MvvmLigthsLibs | Proyecto Lands únicamente | Botones |
| 9 | Datos Json | Newtonsoft Json | Proyecto Lands | Datos Json |
| 10 | Conectividad a Internet | Xam.Plugin.Connectivity | Proyecto Lands | Conectividad a Internet |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |