MVVM

CONTENIDO

[1 HERRAMIENTAS 2](#_Toc466455353)

[2 GENERAL MVVM 2](#_Toc466455354)

[3 SPLASH 28](#_Toc466455355)

[4 REFRESCO DE LAS LISTAS 28](#_Toc466455356)

[5 PICKER BINDABLE 29](#_Toc466455357)

[6 STEEPER CON MVVM 35](#_Toc466455358)

[7 TERMINAR APLICACIÓN 37](#_Toc466455359)

[8 COLOCAR UNA IMAGEN EN UN CÍRCULO 38](#_Toc466455360)

[9 VERIFICAR SI HAY CONEXIÓN A INTERNET 38](#_Toc466455361)

[10 EJEMPLO DE POST 38](#_Toc466455362)

[11 SQLite 39](#_Toc466455363)

[12 SINGLETON 39](#_Toc466455364)

[13 MAPAS Y LOCALIZACIÓN 40](#_Toc466455365)

[14 FOTOS 45](#_Toc466455366)

[15 SEARCH BAR 48](#_Toc466455367)

[16 CORREO Y PASSWORD – EXPRESIONES REGULARES 49](#_Toc466455368)

[17 ENVIAR IMAGEN AL SERVICIO 49](#_Toc466455369)

[18 UTILIZAR LA CÁMARA COMO LECTOR DE CÓDIGO DE BARRAS 51](#_Toc466455370)

[19 PUBLICAR LA APLICACIÓN EN GOOGLE PLAY 54](#_Toc466455371)

# HERRAMIENTAS

* Los íconos para Android vamos a usar: <https://romannurik.github.io/AndroidAssetStudio/>
* Las combinaciones de colores las vamos a consultar en: <https://color.adobe.com/es/explore/most-popular/?time=all>

# GENERAL MVVM

1. Crea el proyecto y copia todo el código del App.cs, elimina el archivo y créalo nuevamente utilizando la plantilla de XAML Page. Debe de quedar de esta manera:

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

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

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

x:Class="MVVM.App">

</Application>

Ahora vamos al code behind y vamos reemplazar por el código que tenemos en la porta papeles, colocando el partial class y el InitializeComponent(), debe quedar así:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Xamarin.Forms;

namespace MVVM

{

public partial class App : Application

{

public App()

{

InitializeComponent();

// The root page of your application

MainPage = new ContentPage

{

Content = new StackLayout

{

VerticalOptions = LayoutOptions.Center,

Children = {

new Label {

HorizontalTextAlignment = TextAlignment.Center,

Text = "Welcome to Xamarin Forms!"

}

}

}

};

}

protected override void OnStart()

{

// Handle when your app starts

}

protected override void OnSleep()

{

// Handle when your app sleeps

}

protected override void OnResume()

{

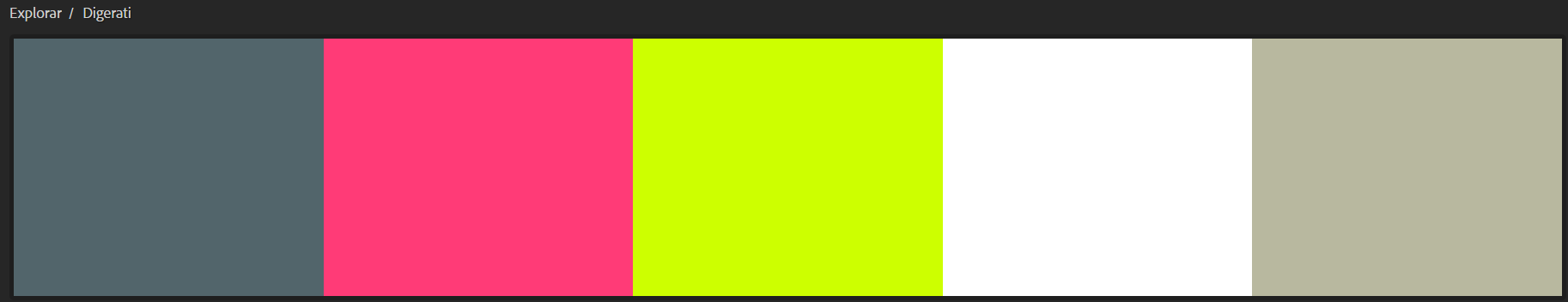
// Handle when your app resumes

}

}

}

1. Esta es la paleta de colores que vamos a usar en nuestra aplicación:



1. Vamos a modificar nuestra App para crear el diccionario de recursos, inicialmente con los colores que va a manejar nuestra aplicación, debe quedar así:

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

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

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

x:Class="MVVM.App">

<Application.Resources>

<ResourceDictionary>

<Color x:Key="MainColor">#ff3b77</Color>

<Color x:Key="FontColor">#52656b</Color>

<Color x:Key="AccentColor1">#cdff00</Color>

<Color x:Key="AccentColor2">#b8b89f</Color>

<Color x:Key="BackgroundColor">#ffffff</Color>

</ResourceDictionary>

</Application.Resources>

</Application>

1. Creamos la carpeta Pages y dentro de esta carpeta vamos a crear nuestra Master Detail que la llamaremos MasterPage. Vamos a cambiarlo para que quede así:

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

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

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

x:Class="MVVM.Pages.MasterPage">

</MasterDetailPage>

Y el code behind también debe quedar así:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Xamarin.Forms;

namespace MVVM.Pages

{

public partial class MasterPage : MasterDetailPage

{

public MasterPage()

{

InitializeComponent();

}

}

}

1. Ahora creemos nuestro MenuPage.
2. Luego nuestra página principal, que le llamaremos MainPage.
3. Modifique el MasterPage para que quede así:

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

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

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

xmlns:pages="clr-namespace:MVVM.Pages;assembly=MVVM"

x:Class="MVVM.Pages.MasterPage">

<MasterDetailPage.Master>

<pages:MenuPage></pages:MenuPage>

</MasterDetailPage.Master>

<MasterDetailPage.Detail>

<NavigationPage x:Name="Navigator">

<x:Arguments>

<pages:MainPage>

</pages:MainPage>

</x:Arguments>

</NavigationPage>

</MasterDetailPage.Detail>

</MasterDetailPage>

Y su code behind para que quede:

namespace MVVM.Pages

{

public partial class MasterPage : MasterDetailPage

{

public MasterPage()

{

InitializeComponent();

}

protected override void OnAppearing()

{

base.OnAppearing();

App.Navigator = Navigator;

}

}

}

1. Creemos la propiedad en App.Navigator

public static NavigationPage Navigator { get; internal set; }

1. Coloquemos un título a MenuPage:

<?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="MVVM.Pages.MenuPage"

Title="Menú">

<Label Text="{Binding MainText}" VerticalOptions="Center" HorizontalOptions="Center" />

</ContentPage>

1. Adicionemos estos valores al diccionario de recursos:

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

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

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

x:Class="MVVM.App">

<Application.Resources>

<ResourceDictionary>

<!-- General colors -->

<Color x:Key="MainColor">#ff3b77</Color>

<Color x:Key="FontColor">#52656b</Color>

<Color x:Key="AccentColor1">#cdff00</Color>

<Color x:Key="AccentColor2">#b8b89f</Color>

<Color x:Key="BackgroundColor">#ffffff</Color>

<!-- Menu colors -->

<Color x:Key="MenuColor">#52656b</Color>

<Color x:Key="MenuFontColor">#ffffff</Color>

</ResourceDictionary>

</Application.Resources>

</Application>

1. Asignemos colores a nuestra MainPage que quede:

<?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="MVVM.Pages.MainPage"

BackgroundColor="{StaticResource BackgroundColor}">

<Label Text="Main Page"

TextColor="{StaticResource FontColor}"

VerticalOptions="Center"

HorizontalOptions="Center" />

</ContentPage>

1. Lo mismo a nuestra MenuPage:

<?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="MVVM.Pages.MenuPage"

Title="Menú"

BackgroundColor="{StaticResource MenuColor}">

<Label Text="Menú"

TextColor="{StaticResource MenuFontColor}"

VerticalOptions="Center"

HorizontalOptions="Center" />

</ContentPage>

1. Cambie la página de inicio en el App.cs:

MainPage = new MasterPage();

1. Podemos probar lo que llevamos

1. Para cambiar el color de la barra del menú:

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

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

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

xmlns:pages="clr-namespace:MVVM.Pages;assembly=MVVM"

x:Class="MVVM.Pages.MasterPage">

<MasterDetailPage.Master>

<pages:MenuPage></pages:MenuPage>

</MasterDetailPage.Master>

<MasterDetailPage.Detail>

<NavigationPage x:Name="Navigator"

BarBackgroundColor="{StaticResource MainColor}">

<x:Arguments>

<pages:MainPage>

</pages:MainPage>

</x:Arguments>

</NavigationPage>

</MasterDetailPage.Detail>

</MasterDetailPage>

1. Para cambiar los íconos de la aplicación vamos a usar Android Asset Sudio (<https://romannurik.github.io/AndroidAssetStudio/>). Primero entramos por Launcher Icons, lo configuramos como deseamos y lo descargamos.
2. Luego descomprimimos el archivo y lo descargamos uno a uno en la carpeta nativa por aplicación. Según las reglas de cada plataforma.
3. Luego incluimos los nuevos íconos en nuestra solución. Y lo incluimos en el MainActivity para Android:

[Activity(Label = "MVVM", Icon = "@drawable/ic\_launcher", Theme = "@style/MainTheme", MainLauncher = true, ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation)]

Para iOS se configura en el archivo Properties.

1. Generemos un ícono de + para la barra, para esto entramos por: “Action bar and tab icons” (color blanco sin opacidad, fondo transparente). Estos se pueden copiar directamente en la carpeta de recursos de Android. Y los incluimos en Visual Studio.
2. Vamos a incluir el ícono de +, lo agregamos en 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="MVVM.Pages.MainPage"

BackgroundColor="{StaticResource BackgroundColor}">

<Label Text="Main Page"

TextColor="{StaticResource FontColor}"

VerticalOptions="Center"

HorizontalOptions="Center" />

<ContentPage.ToolbarItems>

<ToolbarItem Icon="ic\_action\_content\_add.png"/>

</ContentPage.ToolbarItems>

</ContentPage>

1. Podemos probar lo que llevamos
2. Adicionamos íconos para las opciones: Alarmas, Clientes, Ordenes y Configuración
3. Vamos a crear las instancias de las vistas modelo. Para tal fin vamos a crear una carpeta llamada ViewModels, y dentro de esta crearemos la clase MenuItemViewModel, con este código:

public class MenuItemViewModel

{

public string Icon { get; set; }

public string Title { get; set; }

public string PageName { get; set; }

}

1. Dentro de la carpeta llamada ViewModels, crearemos la clase MainViewModel

public class MainViewModel

{

public ObservableCollection<MenuItemViewModel> Menu { get; set; }

public MainViewModel()

{

LoadMenu();

}

private void LoadMenu()

{

Menu = new ObservableCollection<MenuItemViewModel>();

Menu.Add(new MenuItemViewModel

{

Icon = "ic\_action\_orders.png",

PageName = "NewOrderPage",

Title = "MainPage",

});

Menu.Add(new MenuItemViewModel

{

Icon = "ic\_action\_customers.png",

PageName = "ClientsPage",

Title = "Clientes",

});

Menu.Add(new MenuItemViewModel

{

Icon = "ic\_action\_alarm.png",

PageName = "AlarmsPage",

Title = "Alarmas",

});

Menu.Add(new MenuItemViewModel

{

Icon = "ic\_action\_settings.png",

PageName = "SettingsPage",

Title = "Ajustes",

});

}

}

1. Creemos las páginas que llamará muestro menú.
2. Y ahora mostremos el menú en nuestro MenuPage.

<?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="MVVM.Pages.MenuPage"

Title="Menú"

Icon="ic\_action\_action\_menu.png"

BackgroundColor="{StaticResource MenuColor}">

<ListView

ItemsSource="{Binding Menu}"

HasUnevenRows="True">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"></ColumnDefinition>

<ColumnDefinition Width="\*"></ColumnDefinition>

</Grid.ColumnDefinitions>

<Image

Source="{Binding Icon}"

WidthRequest="50"

HeightRequest="50"/>

<Label

Grid.Column="1"

VerticalOptions="Center"

TextColor="{StaticResource MenuFontColor}"

Text="{Binding Title}"/>

</Grid>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</ContentPage>

1. Crearemos esta clase para poder implementar el patrón **Locator**. Para tal fin vamos a crear la carpeta **Infrastructure** y dentro de esta la clase **InstanceLocator**

public class InstanceLocator

{

public MainViewModel Main { get; set; }

public InstanceLocator()

{

Main = new MainViewModel();

}

}

1. Vamos hacer que nuestro InstanceLocator sea un recurso general para toda la aplicación. Vamos al App.xaml y le hacemos las siguientes modificaciones:

<?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:Z\_Mobile.Infrastructure;assembly=Z\_Mobile"

x:Class="MVVM.App">

<Application.Resources>

<ResourceDictionary>

<!-- General colors -->

<Color x:Key="MainColor">#ff3b77</Color>

<Color x:Key="FontColor">#52656b</Color>

<Color x:Key="AccentColor1">#cdff00</Color>

<Color x:Key="AccentColor2">#b8b89f</Color>

<Color x:Key="BackgroundColor">#f9f9f9</Color>

<!-- Menu colors -->

<Color x:Key="MenuColor">#52656b</Color>

<Color x:Key="MenuFontColor">#ffffff</Color>

<!-- Locator -->

<infra:InstanceLocator x:Key="Locator"></infra:InstanceLocator>

</ResourceDictionary>

</Application.Resources>

</Application>

1. Ahora hagamos que nuestro Locator lo use nuestro MenuPage:

<?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="MVVM.Pages.MainPage"

BackgroundColor="{StaticResource BackgroundColor}"

Title="Mis Pedidos"

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

<ListView

ItemsSource="{Binding Orders}"

HasUnevenRows="True">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid Padding="8">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"></ColumnDefinition>

<ColumnDefinition Width="\*"></ColumnDefinition>

</Grid.ColumnDefinitions>

<Image

Source="icon.png"

VerticalOptions="Start"

WidthRequest="50"

HeightRequest="50"/>

<StackLayout Grid.Column="1">

<Label

FontAttributes="Bold"

VerticalOptions="Center"

TextColor="{StaticResource FontColor}"

Text="{Binding Title}"/>

<Label

VerticalOptions="Center"

TextColor="{StaticResource FontColor}"

Text="{Binding Description}"/>

<Label

VerticalOptions="Center"

TextColor="{StaticResource MainColor}"

Text="{Binding DeliveryDate, StringFormat='{0:yyyy/MM/dd}'}"/>

</StackLayout>

</Grid>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

<ContentPage.ToolbarItems>

<ToolbarItem Icon="ic\_action\_content\_add.png"/>

</ContentPage.ToolbarItems>

</ContentPage>

1. Podemos probar lo que llevamos
2. Ahora vamos a mostrar la página de pedidos. Primero modificamos nuestra página de pedidos (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="MVVM.Pages.MainPage"

BackgroundColor="{StaticResource BackgroundColor}"

Title="Mis Pedidos"

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

<ListView

ItemsSource="{Binding Orders}"

HasUnevenRows="True">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid Padding="8">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"></ColumnDefinition>

<ColumnDefinition Width="\*"></ColumnDefinition>

</Grid.ColumnDefinitions>

<Image

Source="icon.png"

VerticalOptions="Start"

WidthRequest="50"

HeightRequest="50"/>

<StackLayout Grid.Column="1">

<Label

FontAttributes="Bold"

VerticalOptions="Center"

TextColor="{StaticResource FontColor}"

Text="{Binding Title}"/>

<Label

VerticalOptions="Center"

TextColor="{StaticResource FontColor}"

Text="{Binding Description}"/>

<Label

VerticalOptions="Center"

TextColor="{StaticResource MainColor}"

Text="{Binding DeliveryDate, StringFormat='{0:yyyy/MM/dd}'}"/>

</StackLayout>

</Grid>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

<ContentPage.ToolbarItems>

<ToolbarItem Icon="ic\_action\_content\_add.png"/>

</ContentPage.ToolbarItems>

</ContentPage>

1. En el folder ViewModel creamos la clase OrderViewModel con los siguientes atributos:

public class OrderViewModel

{

public string Id { get; set; }

public string Title { get; set; }

public string Description { get; set; }

public DateTime CreationDate { get; set; }

public DateTime DeliveryDate { get; set; }

public string DeliveryInformation { get; set; }

public string Client { get; set; }

}

1. Modifiquemos el MainViewModel:

public class MainViewModel

{

public ObservableCollection<MenuItemViewModel> Menu { get; set; }

public ObservableCollection<OrderViewModel> Orders { get; set; }

public MainViewModel()

{

LoadMenu();

LoadData();

}

private void LoadData()

{

Orders = new ObservableCollection<OrderViewModel>();

for (int i = 0; i < 10; i++)

{

Orders.Add(new OrderViewModel

{

Title = "Lorem Ipsum",

DeliveryDate = DateTime.Today,

Description = "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.",

});

}

}

1. Y cambiemos en el MainViewModel la página que llama los pedidos que está errada:

Menu.Add(new MenuItemViewModel

{

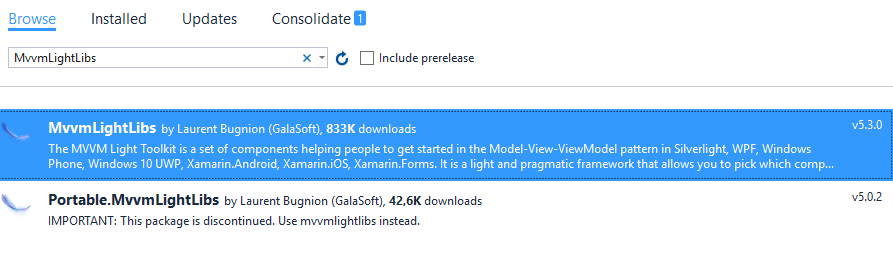
Icon = "ic\_action\_orders.png",

PageName = "MainPage",

Title = "Pedidos",

});

1. Podemos probar lo que llevamos
2. Hasta el momento hemos hecho el prototipo, para poder navegar, debemos de instalar un paquete de Nuget, MvvmLightLibs a todos los proyectos de la solución:



1. Luego en la MainPage asignamos el comando a la acción del toolbar:

<ContentPage.ToolbarItems>

<ToolbarItem Icon="ic\_action\_add.png" Command="{Binding GoToCommand}" CommandParameter="NewOrderPage"/>

</ContentPage.ToolbarItems>

1. Creemos nuestra NewOrderPage:

<?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="MVVM.Pages.NewOrderPage"

BackgroundColor="{StaticResource BackgroundColor}"

Title="Nuevo Pedido"

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

<ScrollView>

<StackLayout Padding="8">

<Label

TextColor="{StaticResource FontColor}"

Text="Título"/>

<Entry

TextColor="{StaticResource FontColor}"

BackgroundColor="White"/>

<Label

TextColor="{StaticResource FontColor}"

Text="Descripción"/>

<Editor

TextColor="{StaticResource FontColor}"

BackgroundColor="White"/>

<Label

TextColor="{StaticResource FontColor}"

Text="Fecha entrega"/>

<DatePicker

BackgroundColor="White" />

<Button

HeightRequest="50"

BorderRadius="25"

BackgroundColor="{StaticResource MainColor}"

TextColor="{StaticResource MenuFontColor}"

Text="Guardar" />

</StackLayout>

</ScrollView>

</ContentPage>

1. Ahora creemos el comando GoToCommand, vamos al MainViewModel y primero lo organizamos en regiones y luego implementamos la nueva región de comandos:

public class MainViewModel

{

#region Properties

public ObservableCollection<MenuItemViewModel> Menu { get; set; }

public ObservableCollection<OrderViewModel> Orders { get; set; }

#endregion

#region Constructors

public MainViewModel()

{

LoadMenu();

LoadData();

}

#endregion

#region Commands

public ICommand GoToCommand { get { return new RelayCommand<string>(GoTo) ; } }

private void GoTo(string pageName)

{

switch (pageName)

{

case "NewOrderPage":

App.Navigator.PushAsync(new NewOrderPage());

break;

default:

break;

}

}

#endregion

#region Methods

private void LoadData()

{

Orders = new ObservableCollection<OrderViewModel>();

for (int i = 0; i < 10; i++)

{

Orders.Add(new OrderViewModel

{

Title = "Lorem Ipsum",

DeliveryDate = DateTime.Today,

Description = "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.",

});

}

}

private void LoadMenu()

{

Menu = new ObservableCollection<MenuItemViewModel>();

Menu.Add(new MenuItemViewModel

{

Icon = "ic\_action\_orders.png",

PageName = "NewOrderPage",

Title = "MainPage",

});

Menu.Add(new MenuItemViewModel

{

Icon = "ic\_action\_customers.png",

PageName = "ClientsPage",

Title = "Clientes",

});

Menu.Add(new MenuItemViewModel

{

Icon = "ic\_action\_alarm.png",

PageName = "AlarmsPage",

Title = "Alarmas",

});

Menu.Add(new MenuItemViewModel

{

Icon = "ic\_action\_settings.png",

PageName = "SettingsPage",

Title = "Ajustes",

});

}

#endregion

}

1. Probemos como se ve la pantalla de nuevo pedido
2. Para que podamos probar, vamos a adicionar un Recongnizer a nuestra Grid del menú:

<?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="MVVM.Pages.MenuPage"

Title="Menú"

Icon="ic\_action\_action\_menu.png"

BackgroundColor="{StaticResource MenuColor}"

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

<ListView

ItemsSource="{Binding Menu}"

HasUnevenRows="True">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid>

<Grid.GestureRecognizers>

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

</Grid.GestureRecognizers>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"></ColumnDefinition>

<ColumnDefinition Width="\*"></ColumnDefinition>

</Grid.ColumnDefinitions>

<Image

Source="{Binding Icon}"

WidthRequest="50"

HeightRequest="50"/>

<Label

Grid.Column="1"

VerticalOptions="Center"

TextColor="{Binding MenuFontColor}"

Text="{Binding Title}"/>

</Grid>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</ContentPage>

1. Los comandos de las ListView deben estar ubicados en su contexto de datos, como el MainViewModel no es contexto de datos del ListView, debemos ubicar nuestro nuevo comando en el MenuItemViewModel, que si es el contexto de datos de los elementos del menú.

public class MenuItemViewModel

{

public string Icon { get; set; }

public string Title { get; set; }

public string PageName { get; set; }

public ICommand NavigateCommand { get { return new RelayCommand(Navigate); } }

private void Navigate()

{

switch (PageName)

{

case "AlarmsPage":

App.Navigator.PushAsync(new AlarmsPage());

break;

case "ClientsPage":

App.Navigator.PushAsync(new ClientsPage());

break;

case "SettingsPage":

App.Navigator.PushAsync(new SettingsPage());

break;

case "MainPage":

App.Navigator.PopToRootAsync();

break;

default:

break;

}

}

}

1. Coloquemos nombres temporales en cada una de las páginas para que verifiquemos que nuestra navegación funciona de forma correcta.

<?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="MVVM.Pages.AlarmsPage">

<Label Text="Alarms Page" VerticalOptions="Center" HorizontalOptions="Center" />

</ContentPage>

<?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="MVVM.Pages.CustomersPage">

<Label Text="Customers Page" VerticalOptions="Center" HorizontalOptions="Center" />

</ContentPage>

<?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="MVVM.Pages.SettingsPage">

<Label Text="Settings Page" VerticalOptions="Center" HorizontalOptions="Center" />

</ContentPage>

1. Probemos lo que llevamos
2. Para que se oculte automáticamente el menú de la Master Detail, en el load del MasterPage vamos a crear la propiedad Master:

protected override void OnAppearing()

{

base.OnAppearing();

App.Master = this;

App.Navigator = Navigator;

}

1. Y asignamos la propiedad IsPresented en false en el MenuItemViewModel, para cerrar el menú lateral:

private void Navigate()

{

switch (PageName)

{

case "AlarmsPage":

App.Navigator.PushAsync(new AlarmsPage());

break;

case "ClientsPage":

App.Navigator.PushAsync(new ClientsPage());

break;

case "SettingsPage":

App.Navigator.PushAsync(new SettingsPage());

break;

case "MainPage":

App.Navigator.PopToRootAsync();

break;

default:

break;

}

}

1. Probemos lo que llevamos hasta el momento.
2. Para mejorar algo en la arquitectura, nuestras ViewModel no deberían conocer la instancia de App. Para solucionar esto, vamos a crear un NavigationService que ejecute todas las acciones de navegación. Para tal fin vamos a crear una carpeta de Services, y dentro de esta vamos a crear la clase NavigationService. Vamos a trasladar el método Navigate que tenemos en el MenuItemViewModel para la nueva clase (no olvidar establecerlo como público, hacer los using correspondientes y colocar el PageName como parámetro). Queda de esta manera:

public class NavigationService

{

public void Navigate(string PageName)

{

App.Master.IsPresented = false;

switch (PageName)

{

case "AlarmsPage":

App.Navigator.PushAsync(new AlarmsPage());

break;

case "ClientsPage":

App.Navigator.PushAsync(new ClientsPage());

break;

case "SettingsPage":

App.Navigator.PushAsync(new SettingsPage());

break;

case "NewOrderPage":

App.Navigator.PushAsync(new NewOrderPage());

break;

case "MainPage":

App.Navigator.PopToRootAsync();

break;

default:

break;

}

}

}

1. Arreglamos el llamado en el MenuItemViewModel para la correcta navegación. Queda de esta manera:

public class MenuItemViewModel

{

private NavigationService navigationService;

public MenuItemViewModel()

{

navigationService = new NavigationService();

}

public string Icon { get; set; }

public string Title { get; set; }

public string PageName { get; set; }

public ICommand NavigateCommand

{

get

{

return new RelayCommand(() => navigationService.Navigate(PageName));

}

}

}

1. Ahora arreglamos la otra navegación, la del comando GoToCommand, en el MainViewModel, queda de esta manera:

public class MainViewModel

{

#region Attributes

private NavigationService navigationService;

#endregion

#region Properties

public ObservableCollection<MenuItemViewModel> Menu { get; set; }

public ObservableCollection<OrderViewModel> Orders { get; set; }

#endregion

#region Constructors

public MainViewModel()

{

navigationService = new NavigationService();

LoadMenu();

LoadData();

}

#endregion

#region Commands

public ICommand GoToCommand { get { return new RelayCommand<string>(GoTo) ; } }

private void GoTo(string pageName)

{

navigationService.Navigate(pageName);

}

#endregion

1. Probemos que este cambio no haya dañado nuestra navegación
2. El otro cambio que vamos a realizar es colocar el servicio de navegación, para que funcione de forma asíncrona, queda de esta manera:

public async void Navigate(string PageName)

{

App.Master.IsPresented = false;

switch (PageName)

{

case "AlarmsPage":

await App.Navigator.PushAsync(new AlarmsPage());

break;

case "ClientsPage":

await App.Navigator.PushAsync(new ClientsPage());

break;

case "SettingsPage":

await App.Navigator.PushAsync(new SettingsPage());

break;

case "NewOrderPage":

await App.Navigator.PushAsync(new NewOrderPage());

break;

case "MainPage":

await App.Navigator.PopToRootAsync();

break;

default:

break;

}

}

1. Probemos que este cambio no haya dañado nuestra navegación
2. Ahora vamos a colocar una página de inicio, la llamaremos WelcomePage y debe quedar dentro de la carpeta Pages, 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"

x:Class="MVVM.Pages.WelcomePage"

BackgroundColor="{StaticResource MainColor}"

Title="Mis Pedidos"

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

<StackLayout

Spacing="20"

VerticalOptions="Center"

HorizontalOptions="Center">

<Label

TextColor="{StaticResource MenuFontColor}"

FontSize="20"

Text="Bienveni@s"/>

<Button

Command="{Binding StartCommand}"

HeightRequest="40"

BorderRadius="20"

BackgroundColor="{StaticResource MenuFontColor}"

TextColor="{StaticResource MainColor}"

Text="Iniciar"/>

</StackLayout>

</ContentPage>

1. Luego creamos el comando en el MainViewModel en la región de comandos:

public ICommand StartCommand { get { return new RelayCommand(Start); } }

private void Start()

{

navigationService.SetMainPage(new MasterPage());

}

1. Creamos el método SetMainPage así:

internal void SetMainPage(Page page)

{

App.Current.MainPage = page;

}

1. Y cambiamos la página de inicio de la aplicación:

public App()

{

InitializeComponent();

MainPage = new WelcomePage();

}

1. Probemos lo que llevamos
2. Ahora vamos a definir unos atributos con el OnPlaform pero en XAML, adicionemos esto al diccionario de recursos:

<!-- Fonts -->

<OnPlatform x:Key="GeneralButtonFont" x:TypeArguments="Font" iOS="12" Android="14" WinPhone="16"/>

Y usemos este parámetro en el NewOrderPage:

<Button

Font="{StaticResource GeneralButtonFont}"

HeightRequest="50"

BorderRadius="25"

BackgroundColor="{StaticResource MainColor}"

TextColor="{StaticResource MenuFontColor}"

Text="Guardar" />

1. Podemos tener muchos botones y definir a cada uno de estos todas las propiedades, pero es mejor crear un estilo con todas esas propiedades y asignar este estilo a los botones. A manera de ejemplo, coloquemos este estilo en el diccionario de recursos:

<!-- Styles -->

<Style x:Key="GeneralButton" TargetType="Button">

<Setter Property="Font" Value="{StaticResource GeneralButtonFont}"/>

<Setter Property="BorderRadius" Value="25"/>

<Setter Property="HeightRequest" Value="50"/>

<Setter Property="BackgroundColor" Value="{StaticResource AccentColor}"/>

<Setter Property="TextColor" Value="{StaticResource MenuFontColor}"/>

<Setter Property="HorizontalOptions" Value="FillAndExpand"/>

<Setter Property="VerticalOptions" Value="Center"/>

</Style>

Quitamos las propiedades personalizadas y colocamos la propiedad de style en el botón que lo deseemos usar, para el ejemplo, lo usaré en el NewOrderPage:

<Button

Style="{StaticResource GeneralButton}"

Text="Guardar" />

1. Probemos a ver cómo va quedando.
2. Si queremos que todos los botones sean exactamente iguales podemos quitar el Key al Style y podemos quitar la propiedad Style al botón:

<!-- Styles -->

<Style TargetType="Button">

<Setter Property="Font" Value="{StaticResource GeneralButtonFont}"/>

<Setter Property="BorderRadius" Value="20"/>

<Setter Property="HeightRequest" Value="40"/>

<Setter Property="BackgroundColor" Value="{StaticResource MainColor}"/>

<Setter Property="TextColor" Value="{StaticResource BackgroundColor}"/>

</Style>

Y en el botón:

<Button

Text="Guardar" />

Probemos y podemos ver que la aplicación sigue lo mismo.

1. Si tuviéramos varias listas de pedidos, para no tener que replicar el Template en cada página lo podemos enviar al diccionario de recursos. Queda así el diccionario:

<!-- Data Templates -->

<DataTemplate x:Key="OrderItemTemplate">

<ViewCell>

<Grid Padding="8">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"></ColumnDefinition>

<ColumnDefinition Width="\*"></ColumnDefinition>

</Grid.ColumnDefinitions>

<Image

Source="icon.png"

VerticalOptions="Start"

WidthRequest="50"

HeightRequest="50"/>

<StackLayout Grid.Column="1">

<Label

FontAttributes="Bold"

VerticalOptions="Center"

TextColor="{StaticResource FontColor}"

Text="{Binding Title}"/>

<Label

VerticalOptions="Center"

TextColor="{StaticResource FontColor}"

Text="{Binding Description}"/>

<Label

VerticalOptions="Center"

TextColor="{StaticResource MainColor}"

Text="{Binding DeliveryDate, StringFormat='{0:yyyy/MM/dd}'}"/>

</StackLayout>

</Grid>

</ViewCell>

</DataTemplate>

Y queda así 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="MVVM.Pages.MainPage"

BackgroundColor="{StaticResource BackgroundColor}"

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

<ListView

ItemTemplate="{StaticResource OrderItemTemplate}"

ItemsSource="{Binding Orders}"

HasUnevenRows="True">

</ListView>

<ContentPage.ToolbarItems>

<ToolbarItem Icon="ic\_action\_add.png" Command="{Binding GoToCommand}" CommandParameter="NewOrderPage"/>

</ContentPage.ToolbarItems>

</ContentPage>

Probemos y podemos ver que la aplicación sigue lo mismo.

1. Creemos el API de Orders para poderlo consumir en la aplicación móvil con el siguiente modelo:

public class Order

{

public string Id { get; set; }

public string Title { get; set; }

public string Description { get; set; }

public DateTime CreationDate { get; set; }

public DateTime DeliveryDate { get; set; }

public string DeliveryInformation { get; set; }

public string Client { get; set; }

public string Phone { get; set; }

public bool IsDelivered { get; set; }

}

1. Agregar los Nuget necesarios para consumir servicios Rest: Microsoft.BCL, Microsoft.Net.Http, Newtonsoft.Json
2. Como ya estamos listos para cargar los datos dinámicos, vamos a comentar el método que carga los métodos estáticos:

public MainViewModel()

{

navigationService = new NavigationService();

LoadMenu();

//LoadData();

}

1. Creamos el folder Models y dentro de este vamos a crear la clase Order. Algunas personas se inclinan por usar el mismo ViewModel para serializar los objetos, pero los modelos no deben tener métodos, como los ViewModel si llevan métodos, por eso prefiero independizar las clases que serializan o deserializan objetos:

public class Order

{

public string Id { get; set; }

public string Title { get; set; }

public string Description { get; set; }

public DateTime CreationDate { get; set; }

public DateTime DeliveryDate { get; set; }

public string DeliveryInformation { get; set; }

public string Client { get; set; }

public string Phone { get; set; }

public bool IsDelivered { get; set; }

}

Recuerde que puede usar [JsonProperty] si el nombre del campo en el servicio es diferente al nombre del modelo. También actualizamos la ViewModel con los nuevos campos:

public class OrderViewModel

{

public string Id { get; set; }

public string Title { get; set; }

public string Description { get; set; }

public DateTime CreationDate { get; set; }

public DateTime DeliveryDate { get; set; }

public string DeliveryInformation { get; set; }

public string Client { get; set; }

public string Phone { get; set; }

public bool IsDelivered { get; set; }

}

1. Para seguir con la buena práctica del NavigationService, vamos a crear la clase ApiService en la carpeta Servives, queda de esta manera:

public class ApiService

{

public async Task<List<Order>> GetAllOrders()

{

try

{

var client = new HttpClient();

client.BaseAddress = new Uri("http://zulu-software.com");

var url = "/ECommerce/api/Order2";

var response = await client.GetAsync(url);

if (!response.IsSuccessStatusCode)

{

return new List<Order>();

}

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

return JsonConvert.DeserializeObject<List<Order>>(result);

}

catch

{

return new List<Order>();

}

}

}

1. Cargamos los datos dinámicos en el método Start, antes de navegar a la página. Para esto en nuestra MainViewModel vamos a definir el servicio como atributo privado y lo vamos a instanciar en el constructor:

#region Attributes

private NavigationService navigationService;

private ApiService apiService;

#endregion

#region Constructors

public MainViewModel()

{

navigationService = new NavigationService();

apiService = new ApiService();

LoadMenu();

//LoadData();

}

#endregion

Y modificamos el método Start para que quede de la siguiente manera:

private async void Start()

{

var orders = await apiService.GetAllOrders();

Orders.Clear();

foreach (var order in orders)

{

Orders.Add(new OrderViewModel

{

Title = order.Title,

Description = order.Description,

DeliveryDate = order.DeliveryDate,

});

}

navigationService.SetMainPage(new MasterPage());

}

Recordando que la inicialización de nuestro Orders, se hacia en el LoadData(), vamos a mover esa inicialización en el constructor del MainViewModel, queda así:

public MainViewModel()

{

navigationService = new NavigationService();

apiService = new ApiService();

Orders = new ObservableCollection<OrderViewModel>();

LoadMenu();

//LoadData();

}

1. Probemos a ver cómo va quedando.
2. Ahora vamos a crear una nueva orden, agregamos el método CreateOrder a nuestro ApiService:

public async Task<Order> CreateOrder(Order newOrder)

{

try

{

var content = JsonConvert.SerializeObject(newOrder);

var body = new StringContent(content, Encoding.UTF8, "application/json");

var client = new HttpClient();

client.BaseAddress = new Uri("http://zulu-software.com");

var url = "/ECommerce/api/Order2";

var response = await client.PostAsync(url, body);

if (!response.IsSuccessStatusCode)

{

return null;

}

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

return JsonConvert.DeserializeObject<Order>(result);

}

catch

{

return null;

}

}

1. En nuestro MainViewModel vamos a crear una nueva orden a nivel del MainViewModel. Antes de navegar vamos a identificar de que página viene para tomar acciones previas. El método GoTo queda de la siguiente manera:

private void GoTo(string pageName)

{

switch (pageName)

{

case "NewOrderPage":

NewOrder = new OrderViewModel();

break;

default:

break;

}

navigationService.Navigate(pageName);

}

Creamos la propiedad NewOrder y la ubicamos en la región de Properties:

#region Properties

public ObservableCollection<MenuItemViewModel> Menu { get; set; }

public ObservableCollection<OrderViewModel> Orders { get; set; }

public OrderViewModel NewOrder { get; private set; }

#endregion

1. Ahora vamos a ligar el objeto NewOrder con nuestra NewOrderPage, aprovecho y agrego algunos campos y el comando al botón:

<?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="MVVM.Pages.NewOrderPage"

BackgroundColor="{StaticResource BackgroundColor}"

Title="Nuevo Pedido"

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

<ScrollView BindingContext="{Binding NewOrder}">

<StackLayout Padding="8">

<Label

TextColor="{StaticResource FontColor}"

Text="Título"/>

<Entry

TextColor="{StaticResource FontColor}"

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

BackgroundColor="{StaticResource BackgroundColor}"/>

<Label

TextColor="{StaticResource FontColor}"

Text="Cliente"/>

<Entry

TextColor="{StaticResource FontColor}"

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

BackgroundColor="{StaticResource BackgroundColor}"/>

<Label

TextColor="{StaticResource FontColor}"

Text="Descripción"/>

<Editor

TextColor="{StaticResource FontColor}"

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

BackgroundColor="{StaticResource BackgroundColor}"/>

<Label

TextColor="{StaticResource FontColor}"

Text="Datos de envío"/>

<Editor

TextColor="{StaticResource FontColor}"

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

BackgroundColor="{StaticResource BackgroundColor}"/>

<Label

TextColor="{StaticResource FontColor}"

Text="Fecha entrega"/>

<DatePicker

Date="{Binding DeliveryDate, Mode=TwoWay}"

BackgroundColor="{StaticResource BackgroundColor}" />

<Button

Command="{Binding SaveCommand}"

Text="Guardar" />

</StackLayout>

</ScrollView>

</ContentPage>

1. Vamos a crear el comando en nuestro OrderViewModel, queda así:

public class OrderViewModel

{

#region Atributes

private ApiService apiService;

#endregion

#region Constructors

public OrderViewModel()

{

apiService = new ApiService();

}

#endregion

#region Properties

public string Id { get; set; }

public string Title { get; set; }

public string Description { get; set; }

public DateTime CreationDate { get; set; }

public DateTime DeliveryDate { get; set; }

public string DeliveryInformation { get; set; }

public string Client { get; set; }

public string Phone { get; set; }

public bool IsDelivered { get; set; }

#endregion

#region Commands

public ICommand SaveCommand

{

get

{

return new RelayCommand(Save);

}

}

private async void Save()

{

var order = new Order

{

Client = this.Client,

CreationDate = DateTime.Now,

DeliveryDate = this.DeliveryDate,

DeliveryInformation = this.DeliveryInformation,

Description = this.Description,

IsDelivered = false,

Title = this.Title,

};

await apiService.CreateOrder(order);

}

#endregion

}

1. Como no tenemos forma de comunicar nuestros mensajes de operación ok o con problemas, vamos a crear una nueva clase llamada DialogService, queda así:

public class DialogService

{

public async Task ShowMessage(string title, string message)

{

await App.Navigator.DisplayAlert(title, message, "OK");

}

}

1. Y lo involucramos en nuestro OrderViewModel, modificando el método Save:

private async void Save()

{

try

{

var order = new Order

{

Client = this.Client,

CreationDate = DateTime.Now,

DeliveryDate = this.DeliveryDate,

DeliveryInformation = this.DeliveryInformation,

Description = this.Description,

IsDelivered = false,

Title = this.Title,

};

await apiService.CreateOrder(order);

await dialogService.ShowMessage("Información", "El pedido ha sido creado");

}

catch

{

await dialogService.ShowMessage("Error", "Ha ocurrido un error inesperado");

}

}

En el constructor inicializamos la fecha:

public OrderViewModel()

{

apiService = new ApiService();

dialogService = new DialogService();

DeliveryDate = DateTime.Today;

}

En la página le damos formato a la fecha en el DatePicker:

<DatePicker

Date="{Binding DeliveryDate, Mode=TwoWay}"

BackgroundColor="{StaticResource BackgroundColor}">

<DatePicker.Format>yyyy-MM-dd</DatePicker.Format>

</DatePicker>

1. Hemos acabado, Probemos.

# SPLASH

Para colocar una pantalla de Splash podemos seguir el siguiente blog: <http://codeworks.it/blog/?p=294>

# REFRESCO DE LAS LISTAS

1. Cambie el ListView para que quede así:

<ListView

ItemTemplate="{StaticResource OrderItemTemplate}"

ItemsSource="{Binding Orders}"

HasUnevenRows="True"

IsPullToRefreshEnabled="True"

RefreshCommand="{Binding RefreshCommand}"

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

</ListView>

1. Cambie el Main View Model para que implemente la interfaz: INotifyPropertyChanged

public class MainViewModel : INotifyPropertyChanged

Y cree el evento de la interfaz:

#region Events

public event PropertyChangedEventHandler PropertyChanged;

#endregion

1. Creamos en el MainViewModel el atributo y la propiedad del refresh:

private bool isRefreshing = false;

Y la propiedad:

public bool IsRefreshing

{

set

{

if (isRefreshing != value)

{

isRefreshing = value;

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs("IsRefreshing"));

}

}

}

get

{

return isRefreshing;

}

}

1. Y obvio, crear el comando del Refresh:

public ICommand RefreshCommand { get { return new RelayCommand(Refresh); } }

public async void Refresh()

{

var orders = await apiService.GetAllOrders();

Orders.Clear();

foreach (var order in orders)

{

Orders.Add(new OrderViewModel

{

Title = order.Title,

Description = order.Description,

DeliveryDate = order.DeliveryDate,

});

}

IsRefreshing = false;

}

1. Con esto debe refrescar al bajar el ListView. Probemos.
2. En la Page donde queremos llamar nuestros eventos, debemos de agregar las siguientes líneas al constructor:

public ServicesPage()

{

InitializeComponent();

var main = (MainViewModel)this.BindingContext;

base.Appearing += (object sender, EventArgs e) =>

{

main.ServicesRefreshCommand.Execute(this);

};

}

1. Ya puedes probar.

# PICKER BINDABLE

1. Creamos la carpeta Controls y dentro de esta creamos la clase BindablePicker con el siguiente código:

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. La página donde vamos a incluir el picker queda de la siguiente manera:

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

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

xmlns:control="clr-namespace:Servicios.Controls"

x:Class="Servicios.Pages.NewServicePage"

Title="Nuevo Servicio"

BackgroundColor="{StaticResource BackgroundColor}"

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

<ScrollView BindingContext="{Binding NewService}">

<StackLayout Spacing="10" Padding="15">

<StackLayout Orientation="Horizontal">

<control:BindablePicker

Title="Seleccione un producto"

DisplayMemberPath="Description"

SelectedValuePath="ProductId"

ItemsSource="{Binding Products}"

SelectedValue ="{Binding Path=ProductId, Mode=TwoWay}"

HorizontalOptions="FillAndExpand"

VerticalOptions="Center"/>

</StackLayout>

1. Agregamos la colección observable de productos al ViewModel de página:

public ObservableCollection<ProductViewModel> Products { get; set; }

1. Llamamos la carga de productos en el constructor de la página:

Products = new ObservableCollection<ProductViewModel>();

LoadProducts();

E implementamos la carga de productos:

private void LoadProducts()

{

var products = dataService.GetAllProducts(false);

Products.Clear();

foreach (var product in products)

{

Products.Add(new ProductViewModel

{

Description = product.Description,

Price = product.Price,

ProductId = product.ProductId,

});

}

}

1. Ya podemos probar.

# STEEPER CON MVVM

1. El modelo debe implementar la interfaz INotifyPropertyChanged, ejemplo:

using SQLite.Net.Attributes;

using SQLiteNetExtensions.Attributes;

using System;

using System.ComponentModel;

namespace Servicios.Models

{

public class Service : INotifyPropertyChanged

{

#region Attributes

private double quantity;

#endregion

#region Properties

[PrimaryKey, AutoIncrement]

public int ServiceId { get; set; }

public DateTime DateService { get; set; }

public DateTime DateRegistered { get; set; }

public int ProductId { get; set; }

[ManyToOne]

public Product Product { get; set; }

public decimal Price { get; set; }

public double Quantity

{

set

{

if (quantity != value)

{

quantity = value;

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs("Quantity"));

}

}

}

get

{

return quantity;

}

}

public decimal Value { get { return Price \* (decimal)Quantity; } }

#endregion

#region Events

public event PropertyChangedEventHandler PropertyChanged;

#endregion

#region Methods

public override int GetHashCode()

{

return ServiceId;

}

public override string ToString()

{

return string.Format("{0} {1:d} {2:C2}", ServiceId, DateService, Value);

}

#endregion

}

}

1. En el ServiceViewModel se crea un método publico así:

public void UpdateQuantity(object sender, int value)

{

Quantity = value;

}

1. En la pagina ServiceNewPage el stepper y el entry quedan asi:

<Entry

IsEnabled="False"

Text="{Binding Quantity}"

FontAttributes="Bold"

FontSize="20"

HorizontalOptions="FillAndExpand"

VerticalOptions="Center"/>

<Stepper

Value="{Binding Quantity}"

Increment="1"

Minimum="1"

Maximum="999999"

HorizontalOptions="End"

VerticalOptions="Center"

ValueChanged="OnQuantityChanged"/>

1. Y el code behind de la pagina ServiceNewPage queda asi:

public partial class NewServicePage : ContentPage

{

private ServiceViewModel service;

public NewServicePage()

{

InitializeComponent();

service = new ServiceViewModel();

}

private void OnQuantityChanged(object sender, ValueChangedEventArgs e)

{

service.UpdateQuantity(sender, (int)e.NewValue);

}

}

1. Ya podemos probar.

# TERMINAR APLICACIÓN

Para terminar la aplicación:

private void Logout()

{

App.User.IsRemembered = false;

dataService.UpdateUser(App.User);

if (Device.OS == TargetPlatform.Android)

{

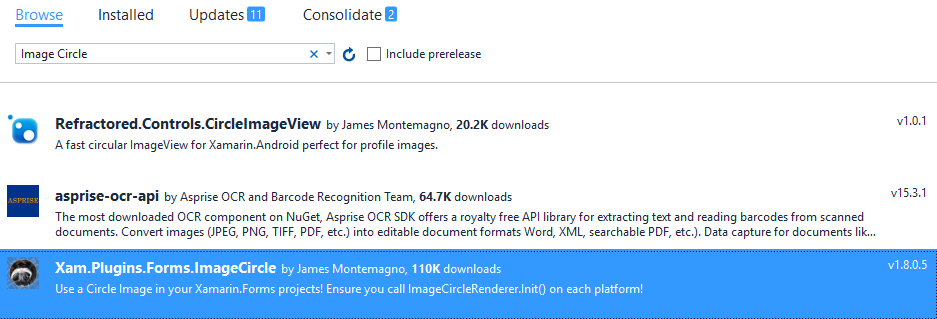
Android.OS.Process.KillProcess(Android.OS.Process.MyPid());

}

}

# COLOCAR UNA IMAGEN EN UN CÍRCULO

1. Agregamos el Nuget: (Image Circle):



1. Luego en la pagina que se va a utilizar se referencia así:

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

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

xmlns:controls="clr-namespace:ImageCircle.Forms.Plugin.Abstractions;assembly=ImageCircle.Forms.Plugin.Abstractions"

x:Class="Z\_Mobile.Pages.UserPage"

Title="Z-Mobile"

BackgroundColor="{StaticResource BackgroundColor}"

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

1. El control queda así:

<controls:CircleImage

Source="{Binding Photo}"

Aspect="AspectFill"

WidthRequest="300"

HeightRequest="300">

</controls:CircleImage>

1. Puedes probar.

# VERIFICAR SI HAY CONEXIÓN A INTERNET

Revisa este link: <http://www.codeproject.com/Tips/870548/Xamarin-forms-Check-network-connectivity-in-iOS-an>

# EJEMPLO DE POST

Este es un ejemplo:

public async Task<Response> Login(string email, string password)

{

try

{

var loginRequest = new LoginRequest

{

Email = email,

Password = password,

};

var request = JsonConvert.SerializeObject(loginRequest);

var body = new StringContent(request, Encoding.UTF8, "application/json");

var client = new HttpClient();

client.BaseAddress = new Uri("http://zulu-software.com");

var url = "/ECommerce/api/Users/Login";

var response = await client.PostAsync(url, body);

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSucces = false,

Message = "Usuario o contraseña incorrectos",

};

}

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

var user = JsonConvert.DeserializeObject<User>(result);

return new Response

{

IsSucces = true,

Message = "Login Ok",

Result = user,

};

}

catch (Exception ex)

{

return new Response

{

IsSucces = false,

Message = ex.Message,

};

}

}

# SQLite

Esta documentación, contiene información valiosa para SQLite en todas las plataformas: <https://developer.xamarin.com/guides/xamarin-forms/working-with/databases/>

# SINGLETON

1. Creamos el singleton en el ViewModel que desdeamos instanciar:

#region Singleton

static MainViewModel instance;

public static MainViewModel GetInstance()

{

if (instance == null)

{

instance = new MainViewModel();

}

return instance;

}

#endregion

1. Se asigna en el constructor:

public MainViewModel()

{

instance = this;

dataService = new DataService();

Menu = new ObservableCollection<MenuItemViewModel>();

NewLogin = new LoginViewModel();

LogedUser = new UserViewModel();

LoadMenu();

}

1. Ya podemos obtener una instancia del ViewModel y por ende, acceder a sus propiedades y métodos desde otras clases:

var mainViewModel = MainViewModel.GetInstance();

mainViewModel.LoadUser(user);

# MAPAS Y LOCALIZACIÓN

1. Desistalar el Nuget de Xamarin.Forms (en toda la solución)
2. Instalar el Nuget Xamarin.Forms.Maps (en toda la solución)
3. Instalar el Nuget de Geolocator:



1. Obtener un key de aplicación para los servicios de Google de: <https://developers.google.com/maps/?hl=es-419>
2. Modificar el AndroidManifest.xml para que quede así (ojo colocar el valor del key obtenido de google):

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

<manifest xmlns:android="http://schemas.android.com/apk/res/android">

<uses-sdk android:minSdkVersion="15" />

<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE" />

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />

<uses-permission android:name="android.permission.ACCESS\_MOCK\_LOCATION" />

<uses-permission android:name="android.permission.ACCESS\_LOCATION\_EXTRA\_COMMANDS" />

<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />

<application android:label="MapsConcept.Droid">

<meta-data android:name="com.google.android.maps.v2.API\_KEY" android:value="AIzaSyAtxvXVhbzV9OTwZh8UxVsW2A58WYf-Btc" />

</application>

</manifest>

1. En el MainActivity.cs del proyecto Droid, se debe adicionar esta línea:

protected override void OnCreate(Bundle bundle)

{

TabLayoutResource = Resource.Layout.Tabbar;

ToolbarResource = Resource.Layout.Toolbar;

base.OnCreate(bundle);

global::Xamarin.Forms.Forms.Init(this, bundle);

Xamarin.FormsMaps.Init(this, bundle);

LoadApplication(new App());

}

1. En el  AppDelegate.cs del proyecto .iOS:

public override bool FinishedLaunching(UIApplication app, NSDictionary options)

{

global::Xamarin.Forms.Forms.Init();

Xamarin.FormsMaps.Init();

LoadApplication(new App());

return base.FinishedLaunching(app, options);

}

1. En los App.xaml.cs del los proyectos Windows en el evento protected override void OnLaunched(LaunchActivatedEventArgs e) se adiciona la línea: Xamarin.FormsMaps.Init("key"); Reemplazar el "Key" por el que brinda la api de Bing cada uno deberá activar este servicio el evento debe quedar parecido a este cambian algunas cosas dependiente del proyecto Windows queda así:

// TODO: change this value to a cache size that is appropriate for your application

rootFrame.CacheSize = 1;

Xamarin.Forms.Forms.Init(e);

Xamarin.FormsMaps.Init("AIzaSyAtxvXVhbzV9OTwZh8UxVsW2A58WYf-Btc");

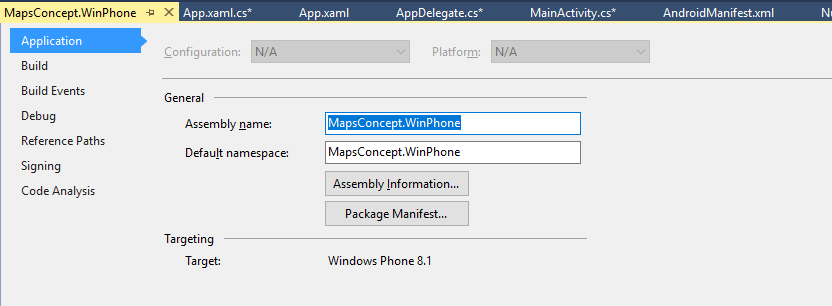
if (e.PreviousExecutionState == ApplicationExecutionState.Terminated)

{

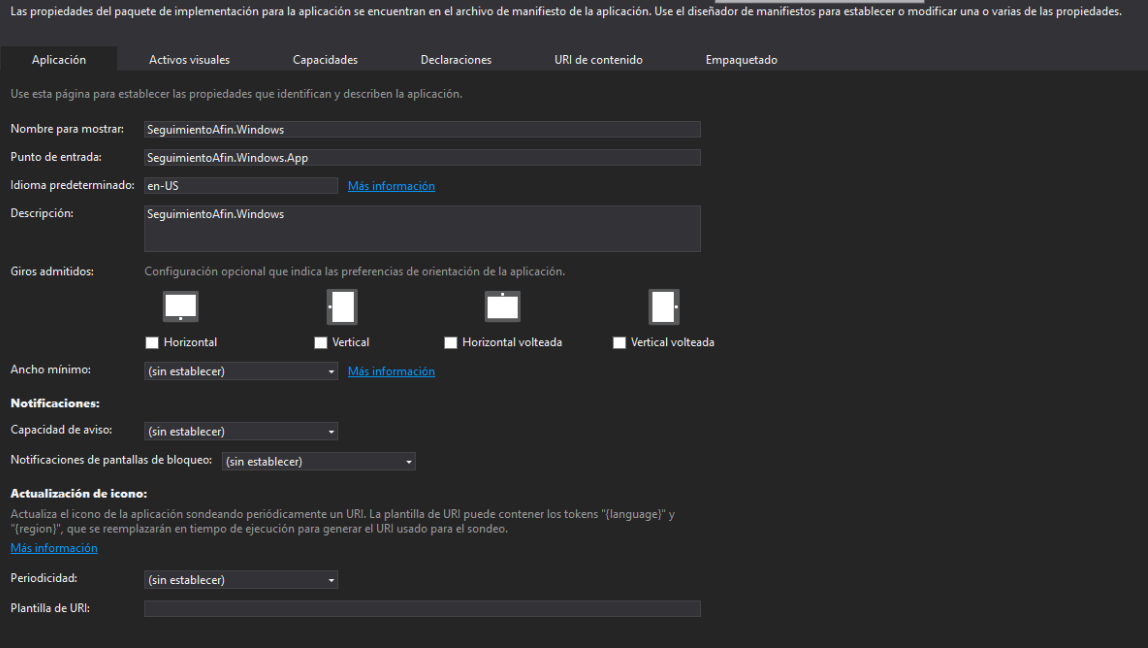
// TODO: Load state from previously suspended application

}

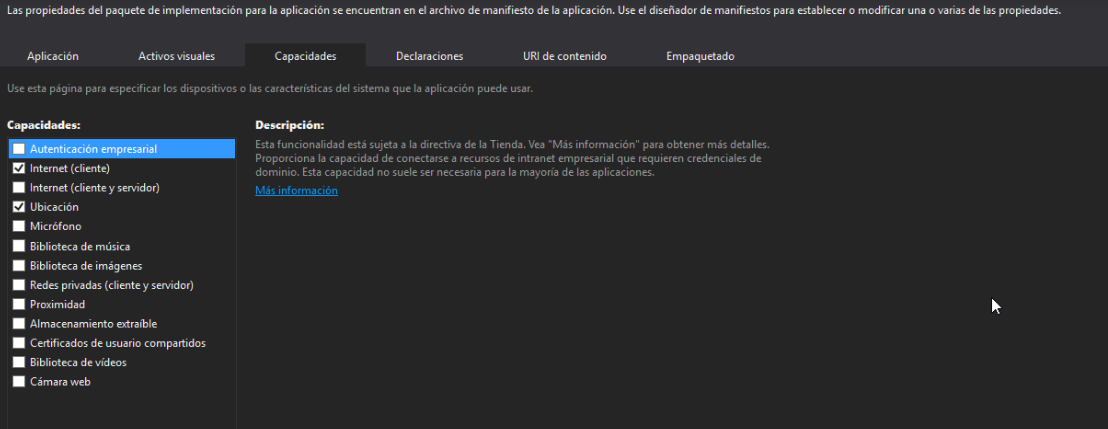
1. En los proyectos Windows estando parado sobre el proyecto le damos Click Derecho y elegimos propiedades, debe salir algo como esto:



Estando en esta pantalla vamos a botón Manifiesto del paquete y debe salir una pantalla como esta:



En esta pantalla vamos al tab de Capacidades y la pantalla debe ser algo como esto:



Acá activamos la capacidad que se llama Ubicación, esto se debe hacer por cada uno de los proyectos Windows.

1. En el XAML de la pagina que contendrá el mapa se mapea la librería de mapas 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:maps="clr-namespace:Xamarin.Forms.Maps;assembly=Xamarin.Forms.Maps"

x:Class="MapsConcept.MapPage">

<maps:Map

WidthRequest="320"

HeightRequest="200"

x:Name="MyMap"

IsShowingUser="true"

MapType="Hybrid">

</maps:Map>

</ContentPage>

1. El codeBehind de la pagina queda algo así:

public partial class MapPage : ContentPage

{

public MapPage()

{

InitializeComponent();

var mainViewModel = new MainViewModel();

mainViewModel.GetGeolotation();

foreach (Pin item in mainViewModel.Pins)

{

MyMap.Pins.Add(item);

}

Locator();

}

private async void Locator()

{

var locator = CrossGeolocator.Current;

locator.DesiredAccuracy = 50;

var location = await locator.GetPositionAsync(timeoutMilliseconds: 10000);

var position = new Position(location.Latitude, location.Longitude);

MyMap.MoveToRegion(MapSpan.FromCenterAndRadius(position, Distance.FromMiles(.3)));

}

}

1. En el viewmodel del pagina se crea atributo así:

public class MainViewModel

{

public ObservableCollection<Pin> Pins { get; set; }

public MainViewModel()

{

Pins = new ObservableCollection<Pin>();

}

public void GetGeolotation()

{

var position1 = new Position(6.2652880, -75.5098530);

var pin1 = new Pin

{

Type = PinType.Place,

Position = position1,

Label = "Pin1",

Address = "prueba pin1"

};

Pins.Add(pin1);

var position2 = new Position(6.2652880, -75.4598530);

var pin2 = new Pin

{

Type = PinType.Place,

Position = position2,

Label = "Pin2",

Address = "prueba pin2"

};

Pins.Add(pin2);

var position3 = new Position(6.2652880, -75.4898530);

var pin3 = new Pin

{

Type = PinType.Place,

Position = position3,

Label = "Pin3",

Address = "prueba pin3"

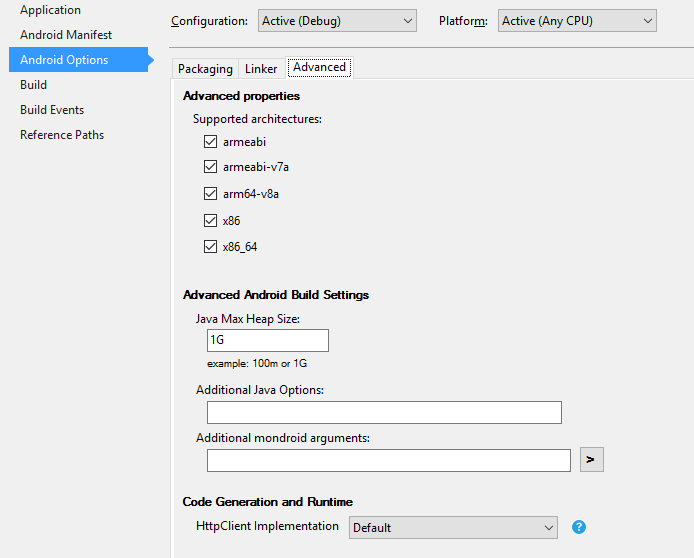
};

Pins.Add(pin3);

}

}

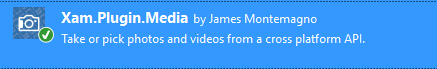
1. Se debe cambiar las opciones de compilación del proyecto Droid:



1. Para poderlo probar en emulador necesitas instalarle al emulador los servicios de Google, en este link encontraras como hacerlo para el Xamarin Android Player: <https://university.xamarin.com/resources/how-to-install-google-play-on-android-emulator>
2. ¡Probemos! Tambien puedes ensayar estos otros tipos de mapas: **Hybrid, Satellite,** **Street**

# FOTOS

1. Instalar el Nuget Xam.Plugin.Media:



1. En el proyecto Droid en el archivo AndroidManifest.xml se agregar estas líneas, debe quedar:

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

<manifest xmlns:android="http://schemas.android.com/apk/res/android">

<uses-sdk android:minSdkVersion="15" />

<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE" />

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />

<uses-permission android:name="android.permission.ACCESS\_MOCK\_LOCATION" />

<uses-permission android:name="android.permission.ACCESS\_LOCATION\_EXTRA\_COMMANDS" />

<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />

<uses-permission android:name="android.permission.CAMERA" />

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />

<uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE" />

<application android:label="MapsConcept.Droid">

<meta-data android:name="com.google.android.maps.v2.API\_KEY" android:value="AIzaSyAtxvXVhbzV9OTwZh8UxVsW2A58WYf-Btc" />

</application>

</manifest>

1. En el proyecto iOS en el archivo Info.plist, se agregar estas líneas:

</array>

<key>UILaunchStoryboardName</key>

<string>LaunchScreen</string>

<key>NSCameraUsageDescription</key>

<key>NSPhotoLibraryUsageDescription</key>

</dict>

</plist>

1. En los proyectos Windows en el archivo App.xaml.cs en el evento OnLaunched agregamos estas lineas:

}

Plugin.Media.MediaImplementation.OnFilesPicked(e);

base.OnActivated(e);

// Ensure the current window is active

Window.Current.Activate();

}

1. En la página donde vamos a tomar y mostar la foto:

<?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:maps="clr-namespace:Xamarin.Forms.Maps;assembly=Xamarin.Forms.Maps"

x:Class="MapsConcept.MapPage"

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

<StackLayout>

<Button

Command="{Binding TakePictureCommand}"

Text="Tomar Foto" />

<Image

WidthRequest="300"

HeightRequest="300"

Source="{Binding ImageSource}"

VerticalOptions="CenterAndExpand" />

<maps:Map

WidthRequest="320"

HeightRequest="200"

x:Name="MyMap"

IsShowingUser="true"

MapType="Hybrid">

</maps:Map>

</StackLayout>

</ContentPage>

1. En el ViewModel, implementamos la interfaz:

public class MainViewModel : INotifyPropertyChanged

Creamos el evento:

public event PropertyChangedEventHandler PropertyChanged;

El atributo:

private ImageSource imageSource;

La propiedad:

public ImageSource ImageSource

{

set

{

if (imageSource != value)

{

imageSource = value;

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

}

}

get

{

return imageSource;

}

}

El comando:

public ICommand TakePictureCommand { get { return new RelayCommand(TakePicture); } }

El método:

private async void TakePicture()

{

await CrossMedia.Current.Initialize();

if (!CrossMedia.Current.IsCameraAvailable || !CrossMedia.Current.IsTakePhotoSupported)

{

await App.Current.MainPage.DisplayAlert("No Camera", ":( No camera available.", "Aceptar");

}

var file = await CrossMedia.Current.TakePhotoAsync(new Plugin.Media.Abstractions.StoreCameraMediaOptions

{

Directory = "Sample",

Name = "test.jpg",

PhotoSize = PhotoSize.Small,

});

if (file != null)

{

ImageSource = ImageSource.FromStream(() =>

{

var stream = file.GetStream();

file.Dispose();

return stream;

});

}

}

1. ¡Probemos!

# SEARCH BAR

1. En la página:

<SearchBar

Placeholder="Buscar..."

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

SearchCommand="{Binding SearchProductCommand}">

</SearchBar>

1. En el ViewModel, la propiedad bindada al filtro:

public string Filter { get; set; }

El comando:

public ICommand SearchProductCommand { get { return new RelayCommand(SearchProduct); } }

El método:

private void SearchProduct()

{

var response = dataService.GetProducts(Filter);

if (!response.IsSucces)

{

return;

}

var products = (List<Product>)response.Result;

Products.Clear();

foreach (var product in products)

{

Products.Add(new ProductViewModel

{

BarCode = product.BarCode,

Category = product.Category,

CategoryId = product.CategoryId,

Company = product.Company,

CompanyId = product.CompanyId,

Description = product.Description,

Image = product.Image,

Inventories = product.Inventories,

Price = product.Price,

ProductId = product.ProductId,

Remarks = product.Remarks,

Stock = product.Stock,

Tax = product.Tax,

TaxId = product.TaxId,

});

}

}

1. En el servicio:

public Response GetProducts(string filter)

{

try

{

using (var da = new DataAccess())

{

var products = da.GetList<Product>(true)

.Where(p => p.Description.ToUpper().Contains(filter.ToUpper()))

.ToList();

return new Response

{

IsSucces = true,

Message = "Consulta de productos OK",

Result = products,

};

}

}

catch (Exception ex)

{

return new Response

{

IsSucces = false,

Message = ex.Message,

};

}

}

1. ¡Probemos!

# CORREO Y PASSWORD – EXPRESIONES REGULARES

Este es un ejemplo para validar un correo válido y un password seguro:

public class Utilities

{

public static bool IsValidEmail(string email)

{

return Regex.Match(email, @"^([\w\.\-]+)@([\w\-]+)((\.(\w){2,3})+)$").Success;

}

public static bool IsValidPassword(string password)

{

return Regex.Match(password, @"^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[$@$!%\*?&])[A-Za-z\d$@$!%\*?&]{8,}").Success;

}

}

# ENVIAR IMAGEN AL SERVICIO

Validamos si el servicio de inserción del registro funcionó correctamente:

var response = await apiService.InsertCustomer(customer);

if (response.IsSucces)

{

if (ImageSource != null)

{

var newCustomer = (Customer)response.Result;

response = await apiService.SetPhoto(newCustomer.CustomerId, file.GetStream());

var filenName = string.Format("{0}.jpg", newCustomer.CustomerId);

var folder = "~/Content/Customers";

var fullPath = string.Format("{0}/{1}", folder, filenName);

customer.Photo = fullPath;

}

}

else

{

customer.IsUpdated = false;

}

Asi debe ser la llamada del método en el API:

public static byte[] ReadFully(Stream input)

{

byte[] buffer = new byte[16 \* 1024];

using (MemoryStream ms = new MemoryStream())

{

int read;

while ((read = input.Read(buffer, 0, buffer.Length)) > 0)

{

ms.Write(buffer, 0, read);

}

return ms.ToArray();

}

}

public async Task<Response> SetPhoto(int customerId, Stream stream)

{

try

{

var array = ReadFully(stream);

var photoRequest = new PhotoRequest

{

Id = customerId,

Array = array,

};

var request = JsonConvert.SerializeObject(photoRequest);

var body = new StringContent(request, Encoding.UTF8, "application/json");

var client = new HttpClient();

client.BaseAddress = new Uri("http://zulu-software.com");

var url = "/ECommerce/api/Customers/SetPhoto";

var response = await client.PostAsync(url, body);

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSucces = false,

Message = response.StatusCode.ToString(),

};

}

return new Response

{

IsSucces = true,

Message = "Foto asignada Ok",

};

}

catch (Exception ex)

{

return new Response

{

IsSucces = false,

Message = ex.Message,

};

}

}

Ver para publicar app:

<https://www.youtube.com/watch?v=Y0gm8jGCduU>

private async void Scaner()

{

var CodigoBarras = DependencyService.Get<IQrCodeScanningService>();

var result= await CodigoBarras.ScanAsync();

EntryCodigoBarras.Text = result.ToString();

ActivityProcees.IsEnabled = true;

Producto Producto = await CargarPreciosServer(EntryCodigoBarras.Text);

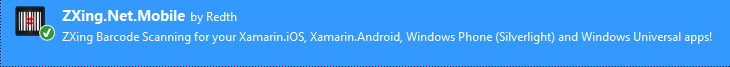
EntryDescripcion.Text = Producto.Description;

ActivityProcees.IsEnabled = false;

}

# UTILIZAR LA CÁMARA COMO LECTOR DE CÓDIGO DE BARRAS

1. Agregar el nuget: ZXing.Net.Mobile a toda la solución:



1. Crear la interfaz: IQrCodeScanningService con el siguiete código:

public interface IQrCodeScanningService

{

Task<string> ScanAsync();

}

1. Implementar la interfaz en cada una de las plataformas, en Droid:

[assembly: Dependency(typeof(ECommerceApp3.Droid.QrCodeScanningService))]

namespace ECommerceApp3.Droid

{

public class QrCodeScanningService : IQrCodeScanningService

{

public async Task<string> ScanAsync()

{

var options = new MobileBarcodeScanningOptions();

var scanner = new MobileBarcodeScanner();

var scanResults = await scanner.Scan(options);

return scanResults.Text;

}

}

}

En iOS:

[assembly: Dependency(typeof(ECommerceApp3.iOS.QrCodeScanningService))]

namespace ECommerceApp3.iOS

{

public class QrCodeScanningService : IQrCodeScanningService

{

public async Task<string> ScanAsync()

{

var scanner = new MobileBarcodeScanner();

var scanResults = await scanner.Scan();

return scanResults.Text;

}

}

}

En Windows:

[assembly: Dependency(typeof(ECommerceApp3.WinPhone.QrCodeScanningService))]

namespace ECommerceApp3.WinPhone

{

public class QrCodeScanningService : IQrCodeScanningService

{

public async Task<string> ScanAsync()

{

var scanner = new MobileBarcodeScanner();

var scanResults = await scanner.Scan();

return scanResults.Text;

}

}

}

1. El siguiente es un ejemplo sencillo de una página que tiene un Entry para la captura del código de barras y un botón que invoca la acción de escanear. Ejemplo de la página:

<?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="ECommerceApp3.Pages.NewOrderPage"

Title="Nuevo Pedido"

BackgroundColor="{StaticResource BackgroundColor}"

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

<ScrollView

BindingContext="{Binding OrderNew}">

<StackLayout

VerticalOptions="CenterAndExpand">

<Entry

Placeholder="código de barras"

Text="{Binding BarCode, Mode=TwoWay}">

</Entry>

<Button

Command="{Binding ScanCommand}"

Text="Scan">

</Button>

</StackLayout>

</ScrollView>

</ContentPage>

Ejemplo del ViewModel:

namespace ECommerceApp3.ViewModels

{

public class NewOrderViewModel : INotifyPropertyChanged

{

private string barCode;

public string BarCode

{

set

{

if (barCode != value)

{

barCode = value;

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

}

}

get

{

return barCode;

}

}

public ICommand ScanCommand { get { return new RelayCommand(Scan); } }

private async void Scan()

{

BarCode = await ScannerSKU();

}

public event PropertyChangedEventHandler PropertyChanged;

public async Task<string> ScannerSKU()

{

try

{

var scanner = DependencyService.Get<IQrCodeScanningService>();

var result = await scanner.ScanAsync();

return result.ToString();

}

catch(Exception ex)

{

ex.ToString();

return string.Empty;

}

}

}

}

1. En el main activity de Android debemos de inicializar el scanner:

protected override void OnCreate(Bundle bundle)

{

TabLayoutResource = Resource.Layout.Tabbar;

ToolbarResource = Resource.Layout.Toolbar;

base.OnCreate(bundle);

global::Xamarin.Forms.Forms.Init(this, bundle);

Xamarin.FormsMaps.Init(this, bundle);

MobileBarcodeScanner.Initialize(this.Application);

LoadApplication(new App());

}

# PUBLICAR LA APLICACIÓN EN GOOGLE PLAY

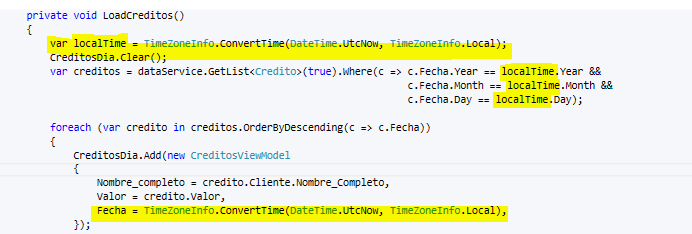
1. Colocar le proyecto Droid en modo release y ejecuta para verificar que no haya ningún problema.
2. Seguir las instrucciones de: <https://developer.xamarin.com/guides/android/deployment,_testing,_and_metrics/publishing_an_application/part_1_-_preparing_an_application_for_release/>
3. Publicar el apk en: <https://developers.google.com/>

# FECHAS EN SQLITE

Para guardar las fechas:

  Fecha = TimeZoneInfo.ConvertTime(DateTime.UtcNow,TimeZoneInfo.Local)

Para recuperar las fechas:



            var creditos = dataService.GetList<Credito>(true).Where(c => TimeZoneInfo.ConvertTime(c.Fecha, TimeZoneInfo.Local).Year == localTime.Year &&

                                                             TimeZoneInfo.ConvertTime(c.Fecha, TimeZoneInfo.Local).Month == localTime.Month &&

                                                            TimeZoneInfo.ConvertTime(c.Fecha, TimeZoneInfo.Local).Day == localTime.Day);