Lo que sigue

1. Search bar.
2. Pantalla de login.
3. Obtener token.
4. Guardar datos en persistencia.
5. Consumir servicios de forma segura.
6. Crear una Master Detail.
7. Base de datos SQLite.
8. Registrar nuevos usuarios por la App.
9. Saludar el usuario en el menú y colocar la imagen en un círculo.
10. Cambiar el ícono de la App.
11. Colocar un Splash.
12. Integración con redes sociales (Facebook)
13. Taller de trabajo colaborativo con GIT
14. Integración con redes sociales (Twitter)
15. Integración con redes sociales (Instagram)
16. Relaciones en los modelos (Categorías).
17. Mapas y Geolocalización
18. Notificaciones Push
19. Publicación en tiendas

# Contenido

[**Contenido**](#_ixz623u13nrk) **2**

[**Nuevos literales**](#_ixh11be0veql) **4**

[Inglés](#_51hrc3uy7tqb) 4

[Español](#_xeqruinyi9ze) 5

[Italiano](#_yz207obmvx4b) 6

[Portugués](#_gb3wlb8ymgxc) 7

[Languages](#_11bmuref7q9i) 8

[**Pantalla login**](#_x4zwirtoq6yx) **11**

[**Obtener un token**](#_7mqsq0olas3x) **14**

[**Master Detail**](#_wct6spzbng1y) **16**

[XAML](#_flq1etelkk34) 16

[C#](#_amv0vggja5ez) 16

[MenuPage](#_muxmeuquuynt) 17

[MenuItemViewModel](#_on0eogd8bifs) 18

[Load menu in MainViewModel](#_z8iu4n7ghxli) 18

[Crear propiedad:](#_sv57hw50zn9k) 18

[Crear método:](#_d2epucxp49j4) 18

[**Consumir servicio de forma segura**](#_oowyst5zffei) **19**

[**Nuevos Literales**](#_h1l85yg1t7dj) **19**

[En Inglés](#_cphz1nbrzzie) 19

[En Español](#_g2xvpqf67jiw) 20

[En Italiano](#_jmiob1wfeesm) 22

[En Portugués](#_bjz996wvlms7) 23

[En Languages](#_sz4rgg20xqnn) 24

[**SQLite**](#_jaizduaqtikd) **26**

[**Registrar Nuevos Usuarios**](#_g9fyxiy4wpit) **30**

[**Colocar la imagen en un círculo**](#_tscvit9of862) **53**

[**Integración con redes sociales**](#_vbliinb7qw78) **54**

[Preliminares](#_imm59tits9f3) 54

[Generales & Facebook](#_6rxi9oewj9ga) 56

[Instagram](#_gayfl1wlwcdh) 86

[Twitter](#_e5vlv7eveb95) 93

[**Relaciones en los modelos**](#_1xy4zapoed9a) **101**

[**GitFlow**](#_hqc3aqf4423) **126**

[**Mapas y geolocalización**](#_r2dwzny340kv) **129**

[**Notificaciones Push**](#_h0jbvupikf4h) **129**

[**Publicación en tiendas**](#_c9y8q8ld5ogp) **129**

# 

# Nuevos literales

## Inglés

<data name="Search" xml:space="preserve">

<value>Search...</value>

</data>

<data name="Login" xml:space="preserve">

<value>Login</value>

</data>

<data name="EMail" xml:space="preserve">

<value>EMail</value>

</data>

<data name="EmailPlaceHolder" xml:space="preserve">

<value>Enter your email...</value>

</data>

<data name="Password" xml:space="preserve">

<value>Password</value>

</data>

<data name="PasswordPlaceHolder" xml:space="preserve">

<value>Enter your password...</value>

</data>

<data name="Rememberme" xml:space="preserve">

<value>Remember me</value>

</data>

<data name="Forgot" xml:space="preserve">

<value>Do you forgot your password?</value>

</data>

<data name="Register" xml:space="preserve">

<value>New user</value>

</data>

<data name="EmailValidation" xml:space="preserve">

<value>You must enter your email.</value>

</data>

<data name="PasswordValidation" xml:space="preserve">

<value>You must enter your password.</value>

</data>

<data name="SomethingWrong" xml:space="preserve">

<value>The email or password are not valid.</value>

</data>

<data name="Menu" xml:space="preserve">

<value>Menu</value>

</data>

<data name="About" xml:space="preserve">

<value>About</value>

</data>

<data name="Setup" xml:space="preserve">

<value>Setup</value>

</data>

<data name="Exit" xml:space="preserve">

<value>Close session</value>

</data>

## Español

<data name="Search" xml:space="preserve">

<value>Buscar…</value>

</data>

<data name="Login" xml:space="preserve">

<value>Inicio de sesión</value>

</data>

<data name="EMail" xml:space="preserve">

<value>Correo electrónico</value>

</data>

<data name="EmailPlaceHolder" xml:space="preserve">

<value>Ingresa tu correo electrónico...</value>

</data>

<data name="Password" xml:space="preserve">

<value>Contraseña</value>

</data>

<data name="PasswordPlaceHolder" xml:space="preserve">

<value>Ingresa tu contraseña...</value>

</data>

<data name="Forgot" xml:space="preserve">

<value>¿Has olvidado la contraseña?</value>

</data>

<data name="Rememberme" xml:space="preserve">

<value>Recordar cuenta</value>

</data>

<data name="Register" xml:space="preserve">

<value>Nuevo usuario</value>

</data>

<data name="EmailValidation" xml:space="preserve">

<value>Debes introducir tu dirección de correo electrónico</value>

</data>

<data name="PasswordValidation" xml:space="preserve">

<value>Debes ingresar tu contraseña</value>

</data>

<data name="SomethingWrong" xml:space="preserve">

<value>El correo electrónico o la contraseña no son válidas.</value>

</data>

<data name="Menu" xml:space="preserve">

<value>Menú</value>

</data>

<data name="About" xml:space="preserve">

<value>Acerca de</value>

</data>

<data name="Setup" xml:space="preserve">

<value>Configuración</value>

</data>

<data name="Exit" xml:space="preserve">

<value>Cerrar sesión</value>

</data>

## Italiano

<data name="Search" xml:space="preserve">

<value>Cerca…</value>

</data>

<data name="Login" xml:space="preserve">

<value>Accesso</value>

</data>

<data name="EMail" xml:space="preserve">

<value>Posta elettronica</value>

</data>

<data name="EmailPlaceHolder" xml:space="preserve">

<value>Inserisci la tua email</value>

</data>

<data name="Password" xml:space="preserve">

<value>Password:</value>

</data>

<data name="PasswordPlaceHolder" xml:space="preserve">

<value>Immettere la password</value>

</data>

<data name="Forgot" xml:space="preserve">

<value>Hai dimenticato la password?</value>

</data>

<data name="Rememberme" xml:space="preserve">

<value>Memorizza account</value>

</data>

<data name="Register" xml:space="preserve">

<value>Nuovo utente</value>

</data>

<data name="EmailValidation" xml:space="preserve">

<value>Devi inserire la tua email</value>

</data>

<data name="PasswordValidation" xml:space="preserve">

<value>Devi inserire la tua password</value>

</data>

<data name="SomethingWrong" xml:space="preserve">

<value>L'e-mail o la password non sono validi.</value>

</data>

<data name="Menu" xml:space="preserve">

<value>Menu</value>

</data>

<data name="About" xml:space="preserve">

<value>Informazioni</value>

</data>

<data name="Setup" xml:space="preserve">

<value>Impostazione</value>

</data>

<data name="Exit" xml:space="preserve">

<value>Chiudi sessione</value>

</data>

## Portugués

<data name="Search" xml:space="preserve">

<value>Pesquisar...</value>

</data>

<data name="Login" xml:space="preserve">

<value>Logon:</value>

</data>

<data name="EMail" xml:space="preserve">

<value>Email</value>

</data>

<data name="EmailPlaceHolder" xml:space="preserve">

<value>Insira o seu e-mail...</value>

</data>

<data name="Password" xml:space="preserve">

<value>Senha</value>

</data>

<data name="PasswordPlaceHolder" xml:space="preserve">

<value>Digite sua senha</value>

</data>

<data name="Forgot" xml:space="preserve">

<value>Esqueceu-se da palavra-passe?</value>

</data>

<data name="Rememberme" xml:space="preserve">

<value>Lembrar-me</value>

</data>

<data name="Register" xml:space="preserve">

<value>Novo utilizador</value>

</data>

<data name="EmailValidation" xml:space="preserve">

<value>Informe seu email</value>

</data>

<data name="PasswordValidation" xml:space="preserve">

<value>Precisa de inserir a sua palavra-passe.</value>

</data>

<data name="SomethingWrong" xml:space="preserve">

<value>O e-mail ou senha não são válidos.</value>

</data>

<data name="Menu" xml:space="preserve">

<value>Menu</value>

</data>

<data name="About" xml:space="preserve">

<value>Sobre</value>

</data>

<data name="Setup" xml:space="preserve">

<value>Configuração</value>

</data>

<data name="Exit" xml:space="preserve">

<value>Fechar sessão</value>

</data>

## Languages

public static string Search

{

get { return Resource.Search; }

}

public static string Login

{

get { return Resource.Login; }

}

public static string EMail

{

get { return Resource.EMail; }

}

public static string EmailPlaceHolder

{

get { return Resource.EmailPlaceHolder; }

}

public static string Password

{

get { return Resource.Password; }

}

public static string PasswordPlaceHolder

{

get { return Resource.PasswordPlaceHolder; }

}

public static string Rememberme

{

get { return Resource.Rememberme; }

}

public static string Forgot

{

get { return Resource.Forgot; }

}

public static string Register

{

get { return Resource.Register; }

}

public static string EmailValidation

{

get { return Resource.EmailValidation; }

}

public static string PasswordValidation

{

get { return Resource.PasswordValidation; }

}

public static string SomethingWrong

{

get { return Resource.SomethingWrong; }

}

public static string Menu

{

get { return Resource.Menu; }

}

public static string Setup

{

get { return Resource.Setup; }

}

public static string About

{

get { return Resource.About; }

}

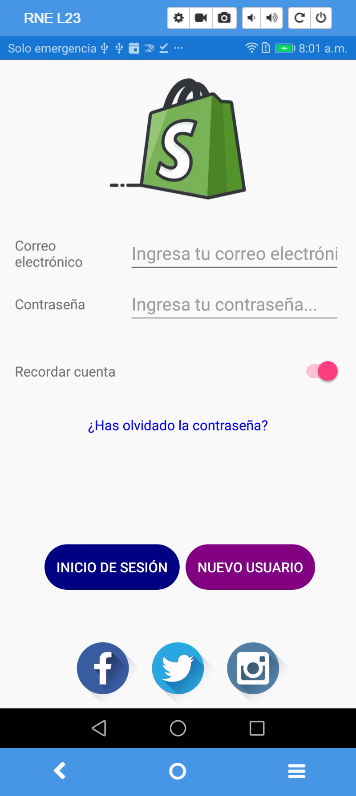
public static string Exit

{

get { return Resource.Exit; }

}

# Pantalla login



La cual tiene este código 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:i18n="clr-namespace:Sales.Helpers"

x:Class="Sales.Views.LoginPage"

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

Title="{i18n:Translate Login}">

<ContentPage.Content>

<ScrollView

BindingContext="{Binding Login}">

<StackLayout

Padding="5">

<Image

HeightRequest="150"

Source="sales">

</Image>

<Grid

Margin="10,0"

VerticalOptions="CenterAndExpand">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="2\*"/>

</Grid.ColumnDefinitions>

<Label

Grid.Row="0"

Grid.Column="0"

Text="{i18n:Translate EMail}"

VerticalOptions="Center">

</Label>

<Entry

Grid.Row="0"

Grid.Column="1"

Keyboard="Email"

Text="{Binding Email}"

Placeholder="{i18n:Translate EmailPlaceHolder}">

</Entry>

<Label

Grid.Row="1"

Grid.Column="0"

Text="{i18n:Translate Password}"

VerticalOptions="Center">

</Label>

<Entry

Grid.Row="1"

Grid.Column="1"

IsPassword="true"

Text="{Binding Password}"

Placeholder="{i18n:Translate PasswordPlaceHolder}">

</Entry>

</Grid>

<StackLayout

Margin="10,0"

Orientation="Horizontal"

VerticalOptions="CenterAndExpand">

<Label

HorizontalOptions="StartAndExpand"

Text="{i18n:Translate Rememberme}"

VerticalOptions="Center">

</Label>

<Switch

IsToggled="{Binding IsRemembered}"

HorizontalOptions="End">

</Switch>

</StackLayout>

<Label

HorizontalOptions="Center"

Text="{i18n:Translate Forgot}"

TextColor="Blue"

VerticalOptions="CenterAndExpand">

<Label.GestureRecognizers>

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

</Label.GestureRecognizers>

</Label>

<ActivityIndicator

IsRunning="{Binding IsRunning}"

VerticalOptions="CenterAndExpand">

</ActivityIndicator>

<StackLayout

Margin="40,0"

Orientation="Horizontal"

VerticalOptions="CenterAndExpand">

<Button

BackgroundColor="Navy"

BorderRadius="23"

Command="{Binding LoginCommand}"

HeightRequest="46"

HorizontalOptions="FillAndExpand"

IsEnabled="{Binding IsEnabled}"

Text="{i18n:Translate Login}"

TextColor="White">

</Button>

<Button

BackgroundColor="Purple"

BorderRadius="23"

Command="{Binding RegisterCommand}"

HeightRequest="46"

HorizontalOptions="FillAndExpand"

IsEnabled="{Binding IsEnabled}"

Text="{i18n:Translate Register}"

TextColor="White">

</Button>

</StackLayout>

<StackLayout

HorizontalOptions="Center"

Orientation="Horizontal"

VerticalOptions="EndAndExpand">

<Image

Aspect="AspectFit"

HeightRequest="70"

Source="facebook\_logo"

WidthRequest="70">

<Image.GestureRecognizers>

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

</Image.GestureRecognizers>

</Image>

<Image

Aspect="AspectFit"

HeightRequest="70"

Source="twitter\_logo"

WidthRequest="70">

<Image.GestureRecognizers>

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

</Image.GestureRecognizers>

</Image>

<Image

Aspect="AspectFit"

HeightRequest="70"

Source="instagram\_logo"

WidthRequest="70">

<Image.GestureRecognizers>

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

</Image.GestureRecognizers>

</Image>

</StackLayout>

</StackLayout>

</ScrollView>

</ContentPage.Content>

</ContentPage>

# Obtener un token

public async Task<TokenResponse> GetToken(

string urlBase,

string username,

string password)

{

try

{

var client = new HttpClient();

client.BaseAddress = new Uri(urlBase);

var response = await client.PostAsync("Token",

new StringContent(string.Format(

"grant\_type=password&username={0}&password={1}",

username, password),

Encoding.UTF8, "application/x-www-form-urlencoded"));

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

var result = JsonConvert.DeserializeObject<TokenResponse>(

resultJSON);

return result;

}

catch

{

return null;

}

}

namespace Sales.Common.Models

{

using System;

using Newtonsoft.Json;

public class TokenResponse

{

[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; }

}

}

# Master Detail

## XAML

<?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:Sales.Views"

x:Class="Sales.Views.MasterPage">

<MasterDetailPage.Master>

<pages:MenuPage/>

</MasterDetailPage.Master>

<MasterDetailPage.Detail>

<NavigationPage x:Name="Navigator">

<x:Arguments>

<pages:ProductsPage/>

</x:Arguments>

</NavigationPage>

</MasterDetailPage.Detail>

</MasterDetailPage>

## C#

namespace Sales.Views

{

using Xamarin.Forms;

using Xamarin.Forms.Xaml;

[XamlCompilation(XamlCompilationOptions.Compile)]

public partial class MasterPage : MasterDetailPage

{

public MasterPage ()

{

InitializeComponent ();

}

protected override void OnAppearing()

{

base.OnAppearing();

App.Navigator = Navigator;

}

}

}

## 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"

xmlns:i18n="clr-namespace:Sales.Helpers"

x:Class="Sales.Views.MenuPage"

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

Title="{i18n:Translate Menu}"

BackgroundColor="Gray">

<ContentPage.Content>

<StackLayout

Padding="5">

<Image

Source="sales">

</Image>

<ListView

ItemsSource="{Binding Menu}"

HasUnevenRows="True">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid>

<Grid.GestureRecognizers>

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

</Grid.GestureRecognizers>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Image

Grid.Column="0"

Source="{Binding Icon}"

WidthRequest="50"

HeightRequest="50">

</Image>

<Label

Grid.Column="1"

VerticalOptions="Center"

TextColor="White"

Text="{Binding Title}">

</Label>

</Grid>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage.Content>

</ContentPage>

## MenuItemViewModel

public string Icon { get; set; }

public string Title { get; set; }

public string PageName { get; set; }

## Load menu in MainViewModel

### Crear propiedad:

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

### Crear método:

private void LoadMenu()

{

this.Menu = new ObservableCollection<MenuItemViewModel>();

this.Menu.Add(new MenuItemViewModel

{

Icon = "ic\_info",

PageName = "AboutPage",

Title = Languages.About,

});

this.Menu.Add(new MenuItemViewModel

{

Icon = "ic\_phonelink\_setup",

PageName = "SetupPage",

Title = Languages.Setup,

});

this.Menu.Add(new MenuItemViewModel

{

Icon = "ic\_exit\_to\_app",

PageName = "LoginPage",

Title = Languages.Exit,

});

}

# Consumir servicio de forma segura

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(tokenType, accessToken);

# Nuevos Literales

## En Inglés

<data name="NoProductsMessage" xml:space="preserve">

<value>No products information available.</value>

</data>

<data name="FirstName" xml:space="preserve">

<value>First name</value>

</data>

<data name="LastName" xml:space="preserve">

<value>Last name</value>

</data>

<data name="Phone" xml:space="preserve">

<value>Phone</value>

</data>

<data name="Address" xml:space="preserve">

<value>Address</value>

</data>

<data name="FirstNamePlaceholder" xml:space="preserve">

<value>Enter your first name...</value>

</data>

<data name="LastNamePlaceholder" xml:space="preserve">

<value>Enter your last name...</value>

</data>

<data name="PhonePlaceHolder" xml:space="preserve">

<value>Enter your phone number...</value>

</data>

<data name="AddressPlaceHolder" xml:space="preserve">

<value>Enter your address...</value>

</data>

<data name="PasswordConfirm" xml:space="preserve">

<value>Confirm</value>

</data>

<data name="PasswordConfirmPlaceHolder" xml:space="preserve">

<value>Enter your password confirm...</value>

</data>

<data name="FirstNameError" xml:space="preserve">

<value>You must enter a fist name.</value>

</data>

<data name="LastNameError" xml:space="preserve">

<value>You must enter a last name.</value>

</data>

<data name="EMailError" xml:space="preserve">

<value>You must enter a valid email.</value>

</data>

<data name="PhoneError" xml:space="preserve">

<value>You must enter a phone number.</value>

</data>

<data name="PasswordError" xml:space="preserve">

<value>You must enter a password. At least 6 characters.</value>

</data>

<data name="PasswordConfirmError" xml:space="preserve">

<value>You must enter a password confirm.</value>

</data>

<data name="PasswordsNoMatch" xml:space="preserve">

<value>The password and confirm must be equals.</value>

</data>

<data name="RegisterConfirmation" xml:space="preserve">

<value>The user was created successfully. Now you can enter with the email and the assigned password.</value>

</data>

## En Español

<data name="NoProductsMessage" xml:space="preserve">

<value>No hay información de productos disponible.</value>

</data>

<data name="FirstName" xml:space="preserve">

<value>Nombre</value>

</data>

<data name="LastName" xml:space="preserve">

<value>Apellidos</value>

</data>

<data name="Phone" xml:space="preserve">

<value>Teléfono</value>

</data>

<data name="Address" xml:space="preserve">

<value>Dirección</value>

</data>

<data name="FirstNamePlaceholder" xml:space="preserve">

<value>Escribe tu nombre...</value>

</data>

<data name="LastNamePlaceholder" xml:space="preserve">

<value>Escribe tu apellido...</value>

</data>

<data name="PhonePlaceHolder" xml:space="preserve">

<value>Escribir el número de teléfono...</value>

</data>

<data name="AddressPlaceHolder" xml:space="preserve">

<value>Introduzca su dirección</value>

</data>

<data name="PasswordConfirm" xml:space="preserve">

<value>Confirmación</value>

</data>

<data name="PasswordConfirmPlaceHolder" xml:space="preserve">

<value>Introduzca su confirmación de contraseña...</value>

</data>

<data name="FirstNameError" xml:space="preserve">

<value>Debe introducir un nombre.</value>

</data>

<data name="LastNameError" xml:space="preserve">

<value>Debe escribir apellidos</value>

</data>

<data name="EMailError" xml:space="preserve">

<value>Hay que introducir una dirección de correo válida.</value>

</data>

<data name="PhoneError" xml:space="preserve">

<value>Debe introducir un número de teléfono.</value>

</data>

<data name="PasswordConfirmError" xml:space="preserve">

<value>Debes ingresar una contraseña.</value>

</data>

<data name="PasswordsNoMatch" xml:space="preserve">

<value>La contraseña y la confirmación deben ser iguales.</value>

</data>

<data name="PasswordError" xml:space="preserve">

<value>Debe introducir una contraseña. Por lo menos 6 caracteres.</value>

</data>

<data name="RegisterConfirmation" xml:space="preserve">

<value>El usuario se ha creado correctamente. Ahora puede ingresar con el correo electrónico y la contraseña asignada.</value>

</data>

## En Italiano

<data name="NoProductsMessage" xml:space="preserve">

<value>Nessuna informazione sui prodotti disponibile.</value>

</data>

<data name="FirstName" xml:space="preserve">

<value>Nome</value>

</data>

<data name="LastName" xml:space="preserve">

<value>Cognome</value>

</data>

<data name="Phone" xml:space="preserve">

<value>Telefono</value>

</data>

<data name="Address" xml:space="preserve">

<value>Indirizzo</value>

</data>

<data name="FirstNamePlaceholder" xml:space="preserve">

<value>Immetti il tuo nome...</value>

</data>

<data name="LastNamePlaceholder" xml:space="preserve">

<value>Immetti il tuo cognome....</value>

</data>

<data name="PhonePlaceHolder" xml:space="preserve">

<value>Immetti il numero di telefono...</value>

</data>

<data name="AddressPlaceHolder" xml:space="preserve">

<value>Inserisci il tuo indirizzo...</value>

</data>

<data name="PasswordConfirm" xml:space="preserve">

<value>Conferma</value>

</data>

<data name="PasswordConfirmPlaceHolder" xml:space="preserve">

<value>Inserisci la password</value>

</data>

<data name="FirstNameError" xml:space="preserve">

<value>È necessario immettere un nome di pugno.</value>

</data>

<data name="LastNameError" xml:space="preserve">

<value>Immettere un cognome</value>

</data>

<data name="EMailError" xml:space="preserve">

<value>Si deve inserire un indirizzo email valido.</value>

</data>

<data name="PhoneError" xml:space="preserve">

<value>È necessario immettere un numero di telefono.</value>

</data>

<data name="PasswordConfirmError" xml:space="preserve">

<value>Devi inserire una password per confermare.</value>

</data>

<data name="PasswordsNoMatch" xml:space="preserve">

<value>La password e conferma devono essere uguali.</value>

</data>

<data name="PasswordError" xml:space="preserve">

<value>È necessario immettere una password. Almeno 6 caratteri.</value>

</data>

<data name="RegisterConfirmation" xml:space="preserve">

<value>L'utente è stato creato correttamente. Ora è possibile immettere con l'email e la password assegnata.</value>

</data>

## En Portugués

<data name="NoProductsMessage" xml:space="preserve">

<value>Nenhuma informação de produtos disponível.</value>

</data>

<data name="FirstName" xml:space="preserve">

<value>Nome</value>

</data>

<data name="LastName" xml:space="preserve">

<value>Sobrenome</value>

</data>

<data name="Phone" xml:space="preserve">

<value>Telefone</value>

</data>

<data name="Address" xml:space="preserve">

<value>Endereço</value>

</data>

<data name="FirstNamePlaceholder" xml:space="preserve">

<value>Insira seu nome...</value>

</data>

<data name="LastNamePlaceholder" xml:space="preserve">

<value>Insira seu sobrenome...</value>

</data>

<data name="PhonePlaceHolder" xml:space="preserve">

<value>Insira o seu número de telefone...</value>

</data>

<data name="AddressPlaceHolder" xml:space="preserve">

<value>Insira o seu endereço...</value>

</data>

<data name="PasswordConfirm" xml:space="preserve">

<value>Confirmar</value>

</data>

<data name="PasswordConfirmPlaceHolder" xml:space="preserve">

<value>Digite sua senha para confirmar...</value>

</data>

<data name="FirstNameError" xml:space="preserve">

<value>Você deve inserir um nome.</value>

</data>

<data name="LastNameError" xml:space="preserve">

<value>Digite um sobrenome.</value>

</data>

<data name="EMailError" xml:space="preserve">

<value>Você deve inserir um endereço de email válido.</value>

</data>

<data name="PhoneError" xml:space="preserve">

<value>Você deve digitar um número de telefone.</value>

</data>

<data name="PasswordConfirmError" xml:space="preserve">

<value>Tem de introduzir uma palavra-passe</value>

</data>

<data name="PasswordsNoMatch" xml:space="preserve">

<value>A senha e confirme a deve ser igual a.</value>

</data>

<data name="PasswordError" xml:space="preserve">

<value>Informe uma nova senha com no mínimo 6 caracteres.</value>

</data>

<data name="RegisterConfirmation" xml:space="preserve">

<value>O usuário foi criado com êxito. Agora você pode entrar com o e-mail e a senha atribuída.</value>

</data>

## En Languages

public static string NoProductsMessage

{

get { return Resource.NoProductsMessage; }

}

public static string FirstName

{

get { return Resource.FirstName; }

}

public static string FirstNamePlaceholder

{

get { return Resource.FirstNamePlaceholder; }

}

public static string LastName

{

get { return Resource.LastName; }

}

public static string LastNamePlaceholder

{

get { return Resource.LastNamePlaceholder; }

}

public static string Phone

{

get { return Resource.Phone; }

}

public static string PhonePlaceHolder

{

get { return Resource.PhonePlaceHolder; }

}

public static string PasswordConfirm

{

get { return Resource.PasswordConfirm; }

}

public static string PasswordConfirmPlaceHolder

{

get { return Resource.PasswordConfirmPlaceHolder; }

}

public static string Address

{

get { return Resource.Address; }

}

public static string AddressPlaceHolder

{

get { return Resource.AddressPlaceHolder; }

}

public static string FirstNameError

{

get { return Resource.FirstNameError; }

}

public static string LastNameError

{

get { return Resource.LastNameError; }

}

public static string EMailError

{

get { return Resource.EMailError; }

}

public static string PhoneError

{

get { return Resource.PhoneError; }

}

public static string PasswordError

{

get { return Resource.PasswordError; }

}

public static string PasswordConfirmError

{

get { return Resource.PasswordConfirmError; }

}

public static string PasswordsNoMatch

{

get { return Resource.PasswordsNoMatch; }

}

public static string RegisterConfirmation

{

get { return Resource.RegisterConfirmation; }

}

# SQLite

1. Agrega el Nuget: **sqlite-net-pcl** a todos los proyectos del front.
2. Crear la interfaz **IPathService**:

public interface IPathService

{

string GetDatabasePath();

}

1. Implementarla en Android y iOS:

[assembly: Xamarin.Forms.Dependency(typeof(Sales.Droid.Implementations.PathService))]

namespace Sales.Droid.Implementations

{

using Interfaces;

using System;

using System.IO;

public class PathService : IPathService

{

public string GetDatabasePath()

{

string path = Environment.GetFolderPath(Environment.SpecialFolder.Personal);

return Path.Combine(path, "Sales.db3");

}

}

}

[assembly: Xamarin.Forms.Dependency(typeof(Sales.iOS.Implementations.PathService))]

namespace Sales.iOS.Implementations

{

using Interfaces;

using System;

using System.IO;

public class PathService : IPathService

{

public string GetDatabasePath()

{

string docFolder = Environment.GetFolderPath(Environment.SpecialFolder.Personal);

string libFolder = Path.Combine(docFolder, "..", "Library", "Databases");

if (!Directory.Exists(libFolder))

{

Directory.CreateDirectory(libFolder);

}

return Path.Combine(libFolder, "Sales.db3");

}

}

}

1. Ahora creamos el **DataService** para nuestras operaciones con datos:

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Common.Models;

using Interfaces;

using SQLite;

using Xamarin.Forms;

public class DataService

{

private SQLiteAsyncConnection connection;

public DataService()

{

this.OpenOrCreateDB();

}

private async Task OpenOrCreateDB()

{

var databasePath = DependencyService.Get<IPathService>().GetDatabasePath();

this.connection = new SQLiteAsyncConnection(databasePath);

await connection.CreateTableAsync<Product>().ConfigureAwait(false);

}

public async Task Insert<T>(T model)

{

await this.connection.InsertAsync(model);

}

public async Task Insert<T>(List<T> models)

{

await this.connection.InsertAllAsync(models);

}

public async Task Update<T>(T model)

{

await this.connection.UpdateAsync(model);

}

public async Task Update<T>(List<T> models)

{

await this.connection.UpdateAllAsync(models);

}

public async Task Delete<T>(T model)

{

await this.connection.DeleteAsync(model);

}

public async Task<List<Product>> GetAllProducts()

{

var query = await this.connection.QueryAsync<Product>("select \* from [Product]");

var array = query.ToArray();

var list = array.Select(p => new Product

{

Description = p.Description,

ImagePath = p.ImagePath,

IsAvailable = p.IsAvailable,

Price = p.Price,

ProductId = p.ProductId,

PublishOn = p.PublishOn,

Remarks = p.Remarks,

}).ToList();

return list;

}

public async Task DeleteAllProducts()

{

var query = await this.connection.QueryAsync<Product>("delete from [Product]");

}

}

1. Para mayor información ver: <https://www.codeproject.com/Articles/1097179/SQLite-with-Xamarin-Forms-Step-by-Step-guide>

# 

# Registrar Nuevos Usuarios

1. Crear la pantalla de registro:



Con su respectivo 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:i18n="clr-namespace:Sales.Helpers"

x:Class="Sales.Views.RegisterPage"

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

Title="{i18n:Translate Register}">

<ContentPage.Content>

<ScrollView

BindingContext="{Binding Register}">

<StackLayout

Padding="5">

<Image

Source="{Binding ImageSource}">

<Image.GestureRecognizers>

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

</Image.GestureRecognizers>

</Image>

<Label

FontSize="Small"

HorizontalOptions="Center"

Text="{i18n:Translate ChangeImage}">

</Label>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

Grid.Row="0"

Text="{i18n:Translate FirstName}"

VerticalOptions="Center">

</Label>

<Entry

Grid.Column="1"

Grid.Row="0"

Text="{Binding FirstName}"

Placeholder="{i18n:Translate FirstNamePlaceholder}">

</Entry>

<Label

Grid.Column="0"

Grid.Row="1"

Text="{i18n:Translate LastName}"

VerticalOptions="Center">

</Label>

<Entry

Grid.Column="1"

Grid.Row="1"

Text="{Binding LastName}"

Placeholder="{i18n:Translate LastNamePlaceholder}">

</Entry>

<Label

Grid.Column="0"

Grid.Row="2"

Text="{i18n:Translate EMail}"

VerticalOptions="Center">

</Label>

<Entry

Grid.Column="1"

Grid.Row="2"

Keyboard="Email"

Text="{Binding EMail}"

Placeholder="{i18n:Translate EmailPlaceHolder}">

</Entry>

<Label

Grid.Column="0"

Grid.Row="3"

Text="{i18n:Translate Phone}"

VerticalOptions="Center">

</Label>

<Entry

Grid.Column="1"

Grid.Row="3"

Keyboard="Telephone"

Text="{Binding Phone}"

Placeholder="{i18n:Translate PhonePlaceHolder}">

</Entry>

<Label

Grid.Column="0"

Grid.Row="4"

Text="{i18n:Translate Address}"

VerticalOptions="Center">

</Label>

<Entry

Grid.Column="1"

Grid.Row="4"

Text="{Binding Address}"

Placeholder="{i18n:Translate AddressPlaceHolder}">

</Entry>

<Label

Grid.Column="0"

Grid.Row="5"

Text="{i18n:Translate Password}"

VerticalOptions="Center">

</Label>

<Entry

Grid.Column="1"

Grid.Row="5"

IsPassword="True"

Text="{Binding Password}"

Placeholder="{i18n:Translate PasswordPlaceHolder}">

</Entry>

<Label

Grid.Column="0"

Grid.Row="6"

Text="{i18n:Translate PasswordConfirm}"

VerticalOptions="Center">

</Label>

<Entry

Grid.Column="1"

Grid.Row="6"

IsPassword="True"

Text="{Binding PasswordConfirm}"

Placeholder="{i18n:Translate PasswordConfirmPlaceHolder}">

</Entry>

</Grid>

<ActivityIndicator

IsRunning="{Binding IsRunning}"

VerticalOptions="CenterAndExpand">

</ActivityIndicator>

<Button

BackgroundColor="Navy"

BorderRadius="23"

Command="{Binding SaveCommand}"

HeightRequest="46"

IsEnabled="{Binding IsEnabled}"

Text="{i18n:Translate Save}"

TextColor="White">

</Button>

</StackLayout>

</ScrollView>

</ContentPage.Content>

</ContentPage>

1. Adicionamos la imagen de “no user”.
2. Creamos el nuevo controlador en el diccionario de recursos:

<x:String x:Key="UrlUsersController">/Users</x:String>

1. Creamos el modelo **UserRequest**:

public class UserRequest

{

public string FirstName { get; set; }

public string LastName { get; set; }

public string EMail { get; set; }

public string Phone { get; set; }

public string Address { get; set; }

public string Password { get; set; }

public string ImagePath { get; set; }

public byte[] ImageArray { get; set; }

}

1. Creamos el helper **RegexHelper**:

using System;

using System.Net.Mail;

public static class RegexHelper

{

public static bool IsValidEmailAddress(string emailaddress)

{

try

{

var email = new MailAddress(emailaddress);

return true;

}

catch (FormatException)

{

return false;

}

}

}

1. Creamos la ViewModel correspondiente:

namespace Sales.ViewModels

{

using System.Windows.Input;

using GalaSoft.MvvmLight.Command;

using Helpers;

using Plugin.Media;

using Plugin.Media.Abstractions;

using Sales.Common.Models;

using Services;

using Xamarin.Forms;

public class RegisterViewModel : BaseViewModel

{

#region Attributes

private MediaFile file;

private ImageSource imageSource;

private ApiService apiService;

private bool isRunning;

private bool isEnabled;

#endregion

#region Properties

public string FirstName { get; set; }

public string LastName { get; set; }

public string EMail { get; set; }

public string Phone { get; set; }

public string Address { get; set; }

public string Password { get; set; }

public string PasswordConfirm { get; set; }

public bool IsRunning

{

get { return this.isRunning; }

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

}

public bool IsEnabled

{

get { return this.isEnabled; }

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

}

public ImageSource ImageSource

{

get { return this.imageSource; }

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

}

#endregion

#region Constructors

public RegisterViewModel()

{

this.apiService = new ApiService();

this.IsEnabled = true;

this.ImageSource = "nouser";

}

#endregion

#region Commands

public ICommand ChangeImageCommand

{

get

{

return new RelayCommand(ChangeImage);

}

}

private async void ChangeImage()

{

await CrossMedia.Current.Initialize();

var source = await Application.Current.MainPage.DisplayActionSheet(

Languages.ImageSource,

Languages.Cancel,

null,

Languages.FromGallery,

Languages.NewPicture);

if (source == Languages.Cancel)

{

this.file = null;

return;

}

if (source == Languages.NewPicture)

{

this.file = await CrossMedia.Current.TakePhotoAsync(

new StoreCameraMediaOptions

{

Directory = "Sample",

Name = "test.jpg",

PhotoSize = PhotoSize.Small,

}

);

}

else

{

this.file = await CrossMedia.Current.PickPhotoAsync();

}

if (this.file != null)

{

this.ImageSource = ImageSource.FromStream(() =>

{

var stream = this.file.GetStream();

return stream;

});

}

}

public ICommand SaveCommand

{

get

{

return new RelayCommand(Save);

}

}

private async void Save()

{

if (string.IsNullOrEmpty(this.FirstName))

{

await Application.Current.MainPage.DisplayAlert(

Languages.Error,

Languages.FirstNameError,

Languages.Accept);

return;

}

if (string.IsNullOrEmpty(this.LastName))

{

await Application.Current.MainPage.DisplayAlert(

Languages.Error,

Languages.LastNameError,

Languages.Accept);

return;

}

if (string.IsNullOrEmpty(this.EMail))

{

await Application.Current.MainPage.DisplayAlert(

Languages.Error,

Languages.EMailError,

Languages.Accept);

return;

}

if (!RegexHelper.IsValidEmailAddress(this.EMail))

{

await Application.Current.MainPage.DisplayAlert(

Languages.Error,

Languages.EMailError,

Languages.Accept);

return;

}

if (string.IsNullOrEmpty(this.Phone))

{

await Application.Current.MainPage.DisplayAlert(

Languages.Error,

Languages.PhoneError,

Languages.Accept);

return;

}

if (string.IsNullOrEmpty(this.Password))

{

await Application.Current.MainPage.DisplayAlert(

Languages.Error,

Languages.PasswordError,

Languages.Accept);

return;

}

if (this.Password.Length < 6)

{

await Application.Current.MainPage.DisplayAlert(

Languages.Error,

Languages.PasswordError,

Languages.Accept);

return;

}

if (string.IsNullOrEmpty(this.PasswordConfirm))

{

await Application.Current.MainPage.DisplayAlert(

Languages.Error,

Languages.PasswordConfirmError,

Languages.Accept);

return;

}

if (!this.Password.Equals(this.PasswordConfirm))

{

await Application.Current.MainPage.DisplayAlert(

Languages.Error,

Languages.PasswordsNoMatch,

Languages.Accept);

return;

}

this.IsRunning = true;

this.IsEnabled = false;

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

if (!connection.IsSuccess)

{

this.IsRunning = false;

this.IsEnabled = true;

await Application.Current.MainPage.DisplayAlert(

Languages.Error,

connection.Message,

Languages.Accept);

return;

}

byte[] imageArray = null;

if (this.file != null)

{

imageArray = FilesHelper.ReadFully(this.file.GetStream());

}

var userRequest = new UserRequest

{

Address = this.Address,

EMail = this.EMail,

FirstName = this.FirstName,

ImageArray = imageArray,

LastName = this.LastName,

Password = this.Password,

};

var url = Application.Current.Resources["UrlAPI"].ToString();

var prefix = Application.Current.Resources["UrlPrefix"].ToString();

var controller = Application.Current.Resources["UrlUsersController"].ToString();

var response = await this.apiService.Post(url, prefix, controller, userRequest);

if (!response.IsSuccess)

{

this.IsRunning = false;

this.IsEnabled = true;

await Application.Current.MainPage.DisplayAlert(

Languages.Error,

response.Message,

Languages.Accept);

return;

}

this.IsRunning = false;

this.IsEnabled = true;

await Application.Current.MainPage.DisplayAlert(

Languages.Confirm,

Languages.RegisterConfirmation,

Languages.Accept);

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

}

#endregion

}

}

1. En el API creamos los siguientes valores en parámetros:

<appSettings>

<add key="AdminUser" value="jzuluaga55@hotmail.com" />

<add key="AdminPassWord" value="\*\*\*\*" />

<add key="SMTPName" value="smtp.live.com" />

<add key="SMTPPort" value="25" />

</appSettings>

1. Creamos helper **MailHelper** en el **API**:

namespace Sales.API.Helpers

{

using System.Collections.Generic;

using System.Net;

using System.Net.Mail;

using System.Threading.Tasks;

using System.Web.Configuration;

public class MailHelper

{

public static async Task SendMail(string to, string subject, string body)

{

var message = new MailMessage();

message.To.Add(new MailAddress(to));

message.From = new MailAddress(WebConfigurationManager.AppSettings["AdminUser"]);

message.Subject = subject;

message.Body = body;

message.IsBodyHtml = true;

using (var smtp = new SmtpClient())

{

var credential = new NetworkCredential

{

UserName = WebConfigurationManager.AppSettings["AdminUser"],

Password = WebConfigurationManager.AppSettings["AdminPassWord"]

};

smtp.Credentials = credential;

smtp.Host = WebConfigurationManager.AppSettings["SMTPName"];

smtp.Port = int.Parse(WebConfigurationManager.AppSettings["SMTPPort"]);

smtp.EnableSsl = true;

await smtp.SendMailAsync(message);

}

}

public static async Task SendMail(List<string> mails, string subject, string body)

{

var message = new MailMessage();

foreach (var to in mails)

{

message.To.Add(new MailAddress(to));

}

message.From = new MailAddress(WebConfigurationManager.AppSettings["AdminUser"]);

message.Subject = subject;

message.Body = body;

message.IsBodyHtml = true;

using (var smtp = new SmtpClient())

{

var credential = new NetworkCredential

{

UserName = WebConfigurationManager.AppSettings["AdminUser"],

Password = WebConfigurationManager.AppSettings["AdminPassWord"]

};

smtp.Credentials = credential;

smtp.Host = WebConfigurationManager.AppSettings["SMTPName"];

smtp.Port = int.Parse(WebConfigurationManager.AppSettings["SMTPPort"]);

smtp.EnableSsl = true;

await smtp.SendMailAsync(message);

}

}

}

}

1. Creamos helper **UsersHelper** en el **API**:

namespace Sales.API.Helpers

{

using System;

using System.Security.Claims;

using System.Threading.Tasks;

using System.Web.Configuration;

using Common.Models;

using Domain.Models;

using Microsoft.AspNet.Identity;

using Microsoft.AspNet.Identity.EntityFramework;

using Models;

public class UsersHelper : IDisposable

{

private static ApplicationDbContext userContext = new ApplicationDbContext();

private static DataContext db = new DataContext();

public static bool DeleteUser(string userName, string roleName)

{

var userManager = new UserManager<ApplicationUser>(new UserStore<ApplicationUser>(userContext));

var userASP = userManager.FindByEmail(userName);

if (userASP == null)

{

return false;

}

var response = userManager.RemoveFromRole(userASP.Id, roleName);

return response.Succeeded;

}

public static ApplicationUser GetUserASP(string email)

{

var userManager = new UserManager<ApplicationUser>(new UserStore<ApplicationUser>(userContext));

var userASP = userManager.FindByEmail(email);

return userASP;

}

public static Response CreateUserASP(UserRequest userRequest)

{

try

{

var userManager = new UserManager<ApplicationUser>(new UserStore<ApplicationUser>(userContext));

var oldUserASP = userManager.FindByEmail(userRequest.EMail);

if (oldUserASP != null)

{

return new Response

{

IsSuccess = false,

Message = "001. User already exists.",

};

}

var userASP = new ApplicationUser

{

Email = userRequest.EMail,

UserName = userRequest.EMail,

PhoneNumber = userRequest.Phone,

};

var result = userManager.Create(userASP, userRequest.Password);

if (result.Succeeded)

{

var newUserASP = userManager.FindByEmail(userRequest.EMail);

userManager.AddClaim(newUserASP.Id, new Claim(ClaimTypes.GivenName, userRequest.FirstName));

userManager.AddClaim(newUserASP.Id, new Claim(ClaimTypes.Name, userRequest.LastName));

if (!string.IsNullOrEmpty(userRequest.Address))

{

userManager.AddClaim(newUserASP.Id, new Claim(ClaimTypes.StreetAddress, userRequest.Address));

}

if (!string.IsNullOrEmpty(userRequest.ImagePath))

{

userManager.AddClaim(newUserASP.Id, new Claim(ClaimTypes.Uri, userRequest.ImagePath));

}

return new Response

{

IsSuccess = true,

};

}

var errors = string.Empty;

foreach (var error in result.Errors)

{

errors += $"{error}, ";

}

return new Response

{

IsSuccess = false,

Message = errors,

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

public static bool UpdateUserName(string currentUserName, string newUserName)

{

var userManager = new UserManager<ApplicationUser>(new UserStore<ApplicationUser>(userContext));

var userASP = userManager.FindByEmail(currentUserName);

if (userASP == null)

{

return false;

}

userASP.UserName = newUserName;

userASP.Email = newUserName;

var response = userManager.Update(userASP);

return response.Succeeded;

}

public static void CheckRole(string roleName)

{

var roleManager = new RoleManager<IdentityRole>(new RoleStore<IdentityRole>(userContext));

// Check to see if Role Exists, if not create it

if (!roleManager.RoleExists(roleName))

{

roleManager.Create(new IdentityRole(roleName));

}

}

public static void CheckSuperUser()

{

var userManager = new UserManager<ApplicationUser>(new UserStore<ApplicationUser>(userContext));

var email = WebConfigurationManager.AppSettings["AdminUser"];

var password = WebConfigurationManager.AppSettings["AdminPassWord"];

var userASP = userManager.FindByName(email);

if (userASP == null)

{

CreateUserASP(email, "Admin", password);

return;

}

}

public static void CreateUserASP(string email, string roleName)

{

var userManager = new UserManager<ApplicationUser>(new UserStore<ApplicationUser>(userContext));

var userASP = userManager.FindByEmail(email);

if (userASP == null)

{

userASP = new ApplicationUser

{

Email = email,

UserName = email,

};

userManager.Create(userASP, email);

}

userManager.AddToRole(userASP.Id, roleName);

}

public static void CreateUserASP(string email, string roleName, string password)

{

var userManager = new UserManager<ApplicationUser>(new UserStore<ApplicationUser>(userContext));

var userASP = new ApplicationUser

{

Email = email,

UserName = email,

};

var result = userManager.Create(userASP, password);

if (result.Succeeded)

{

userManager.AddToRole(userASP.Id, roleName);

}

}

public static async Task PasswordRecovery(string email)

{

var userManager = new UserManager<ApplicationUser>(new UserStore<ApplicationUser>(userContext));

var userASP = userManager.FindByEmail(email);

if (userASP == null)

{

return;

}

var random = new Random();

var newPassword = string.Format("{0}", random.Next(100000, 999999));

var response = await userManager.AddPasswordAsync(userASP.Id, newPassword);

if (response.Succeeded)

{

var subject = "Sales App - Recuperación de contraseña";

var body = string.Format(@"

<h1>Sales App - Recuperación de contraseña</h1>

<p>Su nueva contraseña es: <strong>{0}</strong></p>

<p>Por favor no olvide cambiarla por una de fácil recordación",

newPassword);

await MailHelper.SendMail(email, subject, body);

}

}

public void Dispose()

{

userContext.Dispose();

db.Dispose();

}

}

}

1. Creamos controlador **UsersController** en el **API**:

namespace Sales.API.Controllers

{

using System;

using System.IO;

using System.Web.Http;

using Common.Models;

using Helpers;

using Newtonsoft.Json.Linq;

[RoutePrefix("api/Users")]

public class UsersController : ApiController

{

public IHttpActionResult PostUser(UserRequest userRequest)

{

if (userRequest.ImageArray != null && userRequest.ImageArray.Length > 0)

{

var stream = new MemoryStream(userRequest.ImageArray);

var guid = Guid.NewGuid().ToString();

var file = $"{guid}.jpg";

var folder = "~/Content/Users";

var fullPath = $"{folder}/{file}";

var response = FilesHelper.UploadPhoto(stream, folder, file);

if (response)

{

userRequest.ImagePath = fullPath;

}

}

var answer = UsersHelper.CreateUserASP(userRequest);

return Ok(answer);

}

[HttpPost]

[Authorize]

[Route("GetUser")]

public IHttpActionResult GetUser(JObject form)

{

try

{

var email = string.Empty;

dynamic jsonObject = form;

try

{

email = jsonObject.Email.Value;

}

catch

{

return BadRequest("Incorrect call.");

}

var user = UsersHelper.GetUserASP(email);

return Ok(user);

}

catch (Exception ex)

{

return BadRequest(ex.Message);

}

}

}

}

1. Publicamos el controlador y probamos todo.
2. Ahora obtengamos el usuario luego de ingresar a la App. Crea estos modelos en **Models**:

public class Claim

{

public int Id { get; set; }

public string UserId { get; set; }

public string ClaimType { get; set; }

public string ClaimValue { get; set; }

}

Y:

using System.Collections.Generic;

public class MyUserASP

{

public string Email { get; set; }

public bool EmailConfirmed { get; set; }

public string PasswordHash { get; set; }

public string SecurityStamp { get; set; }

public string PhoneNumber { get; set; }

public bool PhoneNumberConfirmed { get; set; }

public bool TwoFactorEnabled { get; set; }

public object LockoutEndDateUtc { get; set; }

public bool LockoutEnabled { get; set; }

public int AccessFailedCount { get; set; }

public List<Claim> Claims { get; set; }

public string Id { get; set; }

public string UserName { get; set; }

}

Y:

public class GetUserRequest

{

public string Email { get; set; }

}

1. Y el método **GetUser** en el API:

public async Task<Response> GetUser(string urlBase, string prefix, string controller, string email, string tokenType, string accessToken)

{

try

{

var getUserRequest = new GetUserRequest

{

Email = email,

};

var request = JsonConvert.SerializeObject(getUserRequest);

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

var client = new HttpClient();

client.BaseAddress = new Uri(urlBase);

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(tokenType, accessToken);

var url = $"{prefix}{controller}";

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

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

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = answer,

};

}

var user = JsonConvert.DeserializeObject<UserASP>(answer);

return new Response

{

IsSuccess = true,

Result = user,

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

1. Creamos la propiedad **UserASP** en la **MainViewModel** y en **Settings**.
2. En el **LoginViewModel**, adicionar las siguientes líneas:

Settings.IsRemembered = this.IsRemembered;

Settings.AccessToken = token.AccessToken;

Settings.TokenType = token.TokenType;

var prefix = Application.Current.Resources["UrlPrefix"].ToString();

var controller = Application.Current.Resources["UrlUsersController"].ToString();

var response = await this.apiService.GetUser(url, prefix, $"{controller}/GetUser", this.Email, token.TokenType, token.AccessToken);

if (response.IsSuccess)

{

var userASP = (UserASP)response.Result;

MainViewModel.GetInstance().UserASP = userASP;

Settings.UserASP = JsonConvert.SerializeObject(userASP);

}

this.IsRunning = false;

this.IsEnabled = true;

this.Email = string.Empty;

1. Lo mismo en la clase **App**:

public App()

{

InitializeComponent();

var mainViewModel = MainViewModel.GetInstance();

if (Settings.IsRemembered)

{

if (!string.IsNullOrEmpty(Settings.UserASP))

{

mainViewModel.UserASP = JsonConvert.DeserializeObject<UserASP>(Settings.UserASP);

}

mainViewModel.Products = new ProductsViewModel();

this.MainPage = new MasterPage();

}

else

{

mainViewModel.Login = new LoginViewModel();

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

}

}

1. Crear la propiedad en la **MainViewModel**:

public string UserFullName

{

get

{

if (this.UserASP != null && this.UserASP.Claims != null && this.UserASP.Claims.Count > 1)

{

return $"{this.UserASP.Claims[0].ClaimValue} {this.UserASP.Claims[1].ClaimValue}";

}

return null;

}

}

1. Adicionar label en el **MenuPage**:

<Image

Source="sales">

</Image>

<Label

FontAttributes="Bold"

FontSize="Large"

HorizontalOptions="Center"

Text="{Binding UserFullName}"

TextColor="White">

</Label>

<ListView

ItemsSource="{Binding Menu}"

HasUnevenRows="True">

<ListView.ItemTemplate>

# Colocar la imagen en un círculo

1. Agregar el nuget: **Xam.Plugins.Forms.ImageCircle** a los proyectos del front.
2. Inicializar el render en Android:

base.OnCreate(savedInstanceState);

CrossCurrentActivity.Current.Init(this, savedInstanceState);

ImageCircleRenderer.Init();

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

1. Inicializar el render en iOS:

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

ImageCircle.Forms.Plugin.iOS.ImageCircleRenderer.Init();

LoadApplication(new App());

1. Agregar al XAML donde va la imagen:

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

1. Reemplazar la imagen por:

<controls:CircleImage

Source="{Binding UserImageFullPath}"

Aspect="AspectFill"

WidthRequest="200"

HeightRequest="200">

</controls:CircleImage>

# Integración con redes sociales

## Preliminares

1. Agregar la vista **LoginFacebookPage** sin contenido, solo el “coco”:

<?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:i18n="clr-namespace:Sales.Helpers"

x:Class="Sales.Views.LoginFacebookPage"

Title="{i18n:Translate Login}">

</ContentPage>

1. Agregar la vista **LoginInstagramPage** sin contenido, solo el “coco”:

<?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:i18n="clr-namespace:Sales.Helpers"

x:Class="Sales.Views.LoginInstagramPage"

Title="{i18n:Translate Login}">

</ContentPage>

1. Agregar la vista **LoginTwitterPage** sin contenido, solo el “coco”:

<?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:i18n="clr-namespace:Sales.Helpers"

x:Class="Sales.Views.LoginTwitterPage"

Title="{i18n:Translate Login}">

</ContentPage>

1. Agregar los comandos de llamado a ingreso con redes sociales.

public ICommand LoginFacebookComand

{

get

{

return new RelayCommand(LoginFacebook);

}

}

private async void LoginFacebook()

{

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

if (!connection.IsSuccess)

{

this.IsRunning = false;

this.IsEnabled = true;

await Application.Current.MainPage.DisplayAlert(

Languages.Error,

connection.Message,

Languages.Accept);

return;

}

await Application.Current.MainPage.Navigation.PushAsync(

new LoginFacebookPage());

}

public ICommand LoginInstagramComand

{

get

{

return new RelayCommand(LoginInstagram);

}

}

private async void LoginInstagram()

{

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

if (!connection.IsSuccess)

{

this.IsRunning = false;

this.IsEnabled = true;

await Application.Current.MainPage.DisplayAlert(

Languages.Error,

connection.Message,

Languages.Accept);

return;

}

await Application.Current.MainPage.Navigation.PushAsync(

new LoginInstagramPage());

}

public ICommand LoginTwitterComand

{

get

{

return new RelayCommand(LoginTwitter);

}

}

private async void LoginTwitter()

{

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

if (!connection.IsSuccess)

{

this.IsRunning = false;

this.IsEnabled = true;

await Application.Current.MainPage.DisplayAlert(

Languages.Error,

connection.Message,

Languages.Accept);

return;

}

await Application.Current.MainPage.Navigation.PushAsync(

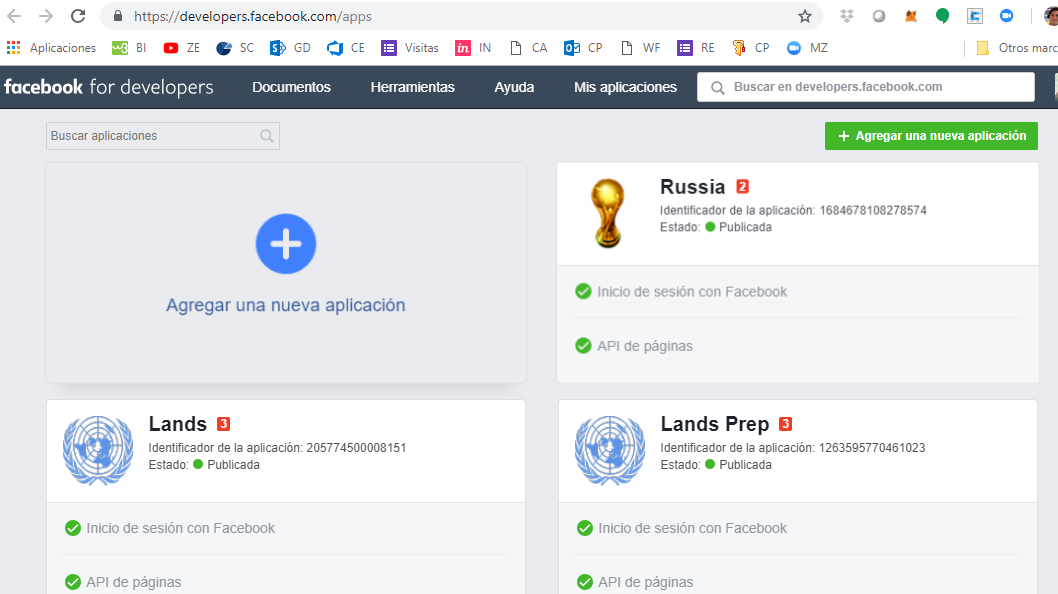
new LoginTwitterPage());

}

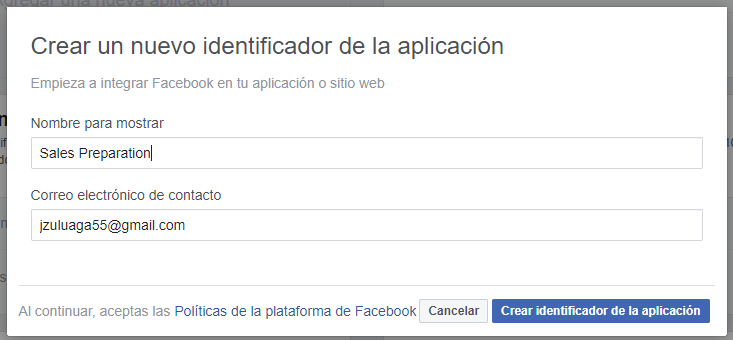
1. Probemos la navegación, ¿funciona cierto? Ahora implementando unos **page rendereres** vamos a poner a funcionar la integración con las redes sociales.
2. Agregue el nuget: **Xamarin.Android.Support.Compat** al proyecto Android.
3. Agregue el nuget: **Xamarin.Auth** a los proyectos iOS y Android.

## Generales & Facebook

1. Primero vamos a configurar todo en Facebook y vamos hacer algo general para las otras redes sociales. Pero para concentranos en Facebool primero vamos a matricular la App en esta red social, para esto ingrese a: <https://developers.facebook.com/apps>



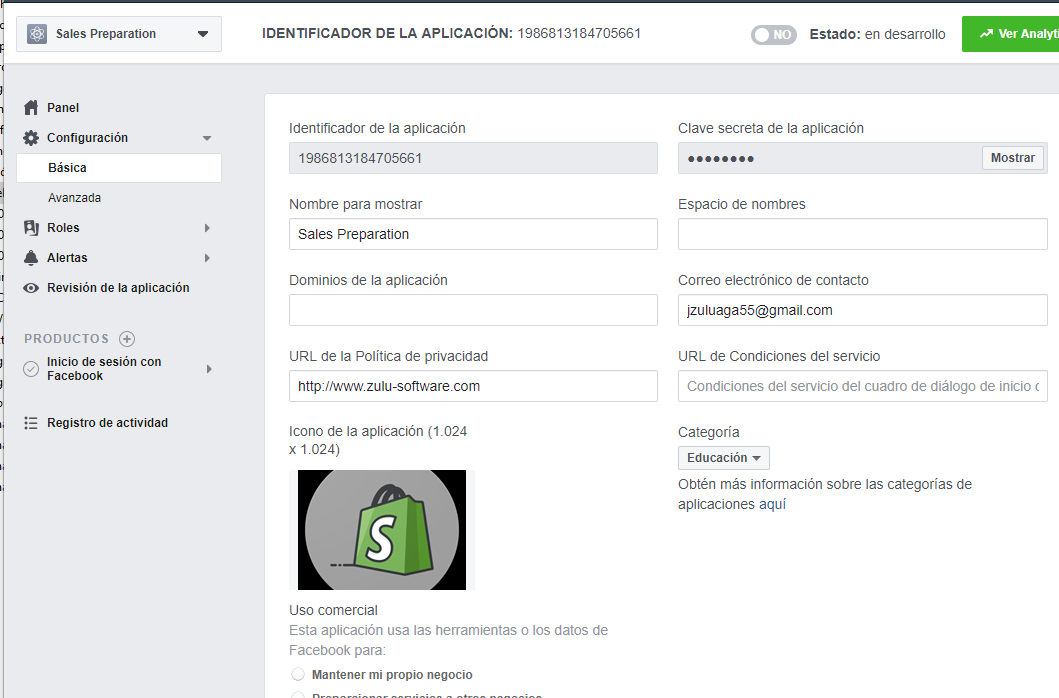
Y crea una nueva aplicación:



Vamos a configurar un inicio de sesión:



Luego a configuración básica y llena la URL de política de privacidad, puede ser con: <http://www.zulu-software.com> y opcional coloca un ícono de tamaño 1024 x 1024, y selcciona una categoría en nuestro caso, colocamos: Educación:



Luego ve a inicio de sesión con Facebook, configuración y coloca la URL de retorno en: <https://www.facebook.com/connect/login_success.html> y los siguientes suiches:



Por último ve a revisión de la aplicación y colócala pública:



Ten el identificador de la App a la mano que lo vamos a necesitar más adelante.

1. Agregue los valores del **OAuth** en el diccionario de recursos:

<!-- Facebook -->

<x:String x:Key="FacebookAppID">1684678108278574</x:String>

<x:String x:Key="FacebookAuthURL">https://www.facebook.com/dialog/oauth/</x:String>

<x:String x:Key="FacebookRedirectURL">https://www.facebook.com/connect/login\_success.html</x:String>

<x:String x:Key="FacebookScope">email</x:String>

<!-- Instagram -->

<x:String x:Key="InstagramAppID">95e62f4c54f544018e5c982beaeffb62</x:String>

<x:String x:Key="InstagramAuthURL">https://api.instagram.com/oauth/authorize/</x:String>

<x:String x:Key="InstagramRedirectURL">https://www.instagram.com/</x:String>

<x:String x:Key="InstagramScope">basic</x:String>

<x:String x:Key="InstagramProfileInfoURL">https://api.instagram.com/v1/users/self/?access\_token</x:String>

<!-- Twitter -->

<x:String x:Key="TwitterConsumerKey">YQTr4UuxgOs9IFkry9LorHN8k</x:String>

<x:String x:Key="TwitterConsumerSecret">Zz0V2rikyOB0ZcHWegpwU7PbndiOpPmNS2NeyUryrmlEsL2eEt</x:String>

<x:String x:Key="TwitterRequestTokenURL">https://api.twitter.com/oauth/request\_token</x:String>

<x:String x:Key="TwitterAuthorizeURL">https://api.twitter.com/oauth/authorize</x:String>

<x:String x:Key="TwitterAccessTokenURL">https://api.twitter.com/oauth/access\_token</x:String>

<x:String x:Key="Url">http://www.zulu-software.com</x:String>

<x:String x:Key="TwitterProfileInfoURL">https://api.twitter.com/1.1/account/verify\_credentials.json</x:String>

1. Cree en **Common** los modelos de respuesta de Facebook, es decir el **FacebookResponse**:

namespace Sales.Common.Models

{

using System.Collections.Generic;

using Newtonsoft.Json;

public class DataFacebook

{

[JsonProperty(PropertyName = "is\_silhouette")]

public bool IsSilhouette { get; set; }

[JsonProperty(PropertyName = "url")]

public string Url { get; set; }

}

public class Picture

{

[JsonProperty(PropertyName = "data")]

public DataFacebook Data { get; set; }

}

public class Cover

{

[JsonProperty(PropertyName = "id")]

public string Id { get; set; }

[JsonProperty(PropertyName = "offset\_y")]

public int OffsetY { get; set; }

[JsonProperty(PropertyName = "source")]

public string Source { get; set; }

}

public class AgeRange

{

[JsonProperty(PropertyName = "min")]

public int Min { get; set; }

}

public class Device

{

[JsonProperty(PropertyName = "hardware")]

public string Hardware { get; set; }

[JsonProperty(PropertyName = "os")]

public string Os { get; set; }

}

public class FacebookResponse

{

[JsonProperty(PropertyName = "name")]

public string Name { get; set; }

[JsonProperty(PropertyName = "picture")]

public Picture Picture { get; set; }

[JsonProperty(PropertyName = "cover")]

public Cover Cover { get; set; }

[JsonProperty(PropertyName = "age\_range")]

public AgeRange AgeRange { get; set; }

[JsonProperty(PropertyName = "devices")]

public List<Device> Devices { get; set; }

[JsonProperty(PropertyName = "gender")]

public string Gender { get; set; }

[JsonProperty(PropertyName = "is\_verified")]

public bool IsVerified { get; set; }

[JsonProperty(PropertyName = "locale")]

public string Locale { get; set; }

[JsonProperty(PropertyName = "link")]

public string Link { get; set; }

[JsonProperty(PropertyName = "first\_name")]

public string FirstName { get; set; }

[JsonProperty(PropertyName = "last\_name")]

public string LastName { get; set; }

[JsonProperty(PropertyName = "id")]

public string Id { get; set; }

}

}

1. Cree en **Common** los modelos de respuesta de Instagram, es decir el **InstagramResponse**:

namespace Sales.Common.Models

{

using Newtonsoft.Json;

public class Counts

{

[JsonProperty("media")]

public int Media { get; set; }

[JsonProperty("follows")]

public int Follows { get; set; }

[JsonProperty("followed\_by")]

public int FollowedBy { get; set; }

}

public class UserData

{

[JsonProperty("id")]

public string Id { get; set; }

[JsonProperty("username")]

public string Username { get; set; }

[JsonProperty("profile\_picture")]

public string ProfilePicture { get; set; }

[JsonProperty("full\_name")]

public string FullName { get; set; }

[JsonProperty("bio")]

public string Bio { get; set; }

[JsonProperty("website")]

public string Website { get; set; }

[JsonProperty("is\_business")]

public bool IsBusiness { get; set; }

[JsonProperty("counts")]

public Counts Counts { get; set; }

}

public class Meta

{

[JsonProperty("code")]

public int Code { get; set; }

}

public class InstagramResponse

{

[JsonProperty("data")]

public UserData UserData { get; set; }

[JsonProperty("meta")]

public Meta Meta { get; set; }

}

}

1. Cree en **Common** los modelos de respuesta de Twitter, es decir el **TwitterResponse**:

namespace Sales.Common.Models

{

using Newtonsoft.Json;

using System.Collections.Generic;

public class Url2

{

[JsonProperty(PropertyName = "url")]

public string Url { get; set; }

[JsonProperty(PropertyName = "expanded\_url")]

public string ExpandedUrl { get; set; }

[JsonProperty(PropertyName = "display\_url")]

public string DisplayUrl { get; set; }

[JsonProperty(PropertyName = "indices")]

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

}

public class Url

{

[JsonProperty(PropertyName = "urls")]

public List<Url2> Urls { get; set; }

}

public class Description

{

[JsonProperty(PropertyName = "urls")]

public List<object> Urls { get; set; }

}

public class Entities

{

[JsonProperty(PropertyName = "url")]

public Url Url { get; set; }

[JsonProperty(PropertyName = "description")]

public Description Description { get; set; }

}

public class UserMention

{

[JsonProperty(PropertyName = "screen\_name")]

public string ScreenName { get; set; }

[JsonProperty(PropertyName = "name")]

public string Name { get; set; }

[JsonProperty(PropertyName = "id")]

public int Id { get; set; }

[JsonProperty(PropertyName = "id\_str")]

public string IdStr { get; set; }

[JsonProperty(PropertyName = "indices")]

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

}

public class UserMention2

{

[JsonProperty(PropertyName = "screen\_name")]

public string ScreenName { get; set; }

[JsonProperty(PropertyName = "name")]

public string Name { get; set; }

[JsonProperty(PropertyName = "id")]

public int Id { get; set; }

[JsonProperty(PropertyName = "id\_str")]

public string IdStr { get; set; }

[JsonProperty(PropertyName = "indices")]

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

}

public class Entities2

{

[JsonProperty(PropertyName = "hashtags")]

public List<object> HashTags { get; set; }

[JsonProperty(PropertyName = "symbols")]

public List<object> Symbols { get; set; }

[JsonProperty(PropertyName = "user\_mentions")]

public List<UserMention2> UserMentions { get; set; }

[JsonProperty(PropertyName = "urls")]

public List<object> Urls { get; set; }

}

public class Entities3

{

[JsonProperty(PropertyName = "hashtags")]

public List<object> HashTags { get; set; }

[JsonProperty(PropertyName = "symbols")]

public List<object> Symbols { get; set; }

[JsonProperty(PropertyName = "user\_mentions")]

public List<UserMention2> UserMentions { get; set; }

[JsonProperty(PropertyName = "urls")]

public List<object> Urls { get; set; }

}

public class RetweetedStatus

{

[JsonProperty(PropertyName = "created\_at")]

public string CreatedAt { get; set; }

[JsonProperty(PropertyName = "id")]

public long Id { get; set; }

[JsonProperty(PropertyName = "id\_str")]

public string IdStr { get; set; }

[JsonProperty(PropertyName = "text")]

public string Text { get; set; }

[JsonProperty(PropertyName = "truncated")]

public bool Truncated { get; set; }

[JsonProperty(PropertyName = "entities")]

public Entities3 Entities { get; set; }

[JsonProperty(PropertyName = "source")]

public string Source { get; set; }

[JsonProperty(PropertyName = "in\_reply\_to\_status\_id")]

public object InReplyToStatusId { get; set; }

[JsonProperty(PropertyName = "in\_reply\_to\_status\_id\_str")]

public object InReplyToStatusIdStr { get; set; }

[JsonProperty(PropertyName = "in\_reply\_to\_user\_id")]

public object InReplyToUserId { get; set; }

[JsonProperty(PropertyName = "in\_reply\_to\_user\_id\_str")]

public object InReplyToUserIdStr { get; set; }

[JsonProperty(PropertyName = "in\_reply\_to\_screen\_name")]

public object InReplyToScreenName { get; set; }

[JsonProperty(PropertyName = "geo")]

public object Geo { get; set; }

[JsonProperty(PropertyName = "coordinates")]

public object Coordinates { get; set; }

[JsonProperty(PropertyName = "place")]

public object Place { get; set; }

[JsonProperty(PropertyName = "contributors")]

public object Contributors { get; set; }

[JsonProperty(PropertyName = "is\_quote\_status")]

public bool IsQuoteStatus { get; set; }

[JsonProperty(PropertyName = "retweet\_count")]

public int RetweetCount { get; set; }

[JsonProperty(PropertyName = "favorite\_count")]

public int FavoriteCount { get; set; }

[JsonProperty(PropertyName = "favorited")]

public bool Favorited { get; set; }

[JsonProperty(PropertyName = "retweeted")]

public bool Retweeted { get; set; }

[JsonProperty(PropertyName = "lang")]

public string Lang { get; set; }

}

public class Status

{

[JsonProperty(PropertyName = "created\_at")]

public string CreatedAt { get; set; }

[JsonProperty(PropertyName = "id")]

public long Id { get; set; }

[JsonProperty(PropertyName = "id\_str")]

public string IdStr { get; set; }

[JsonProperty(PropertyName = "text")]

public string Text { get; set; }

[JsonProperty(PropertyName = "truncated")]

public bool Truncated { get; set; }

[JsonProperty(PropertyName = "entities")]

public Entities2 Entities { get; set; }

[JsonProperty(PropertyName = "source")]

public string Source { get; set; }

[JsonProperty(PropertyName = "in\_reply\_to\_status\_id")]

public object InReplyToStatusId { get; set; }

[JsonProperty(PropertyName = "in\_reply\_to\_status\_id\_str")]

public object InReplyToStatusIdStr { get; set; }

[JsonProperty(PropertyName = "in\_reply\_to\_user\_id")]

public object InReplyToUserId { get; set; }

[JsonProperty(PropertyName = "in\_reply\_to\_user\_id\_str")]

public object InReplyToUserIdStr { get; set; }

[JsonProperty(PropertyName = "in\_reply\_to\_screen\_name")]

public object InReplyToScreenName { get; set; }

[JsonProperty(PropertyName = "geo")]

public object Geo { get; set; }

[JsonProperty(PropertyName = "coordinates")]

public object Coordinates { get; set; }

[JsonProperty(PropertyName = "place")]

public object Place { get; set; }

[JsonProperty(PropertyName = "contributors")]

public object Contributors { get; set; }

[JsonProperty(PropertyName = "retweeted\_status")]

public RetweetedStatus RetweetedStatus { get; set; }

[JsonProperty(PropertyName = "is\_quote\_status")]

public bool IsQuoteStatus { get; set; }

[JsonProperty(PropertyName = "retweet\_count")]

public int RetweetCount { get; set; }

[JsonProperty(PropertyName = "favorite\_count")]

public int FavoriteCount { get; set; }

[JsonProperty(PropertyName = "favorited")]

public bool Favorited { get; set; }

}

public class TwitterResponse

{

[JsonProperty(PropertyName = "id")]

public int Id { get; set; }

[JsonProperty(PropertyName = "id\_str")]

public string IdStr { get; set; }

[JsonProperty(PropertyName = "name")]

public string Name { get; set; }

[JsonProperty(PropertyName = "screen\_name")]

public string ScreenName { get; set; }

[JsonProperty(PropertyName = "location")]

public string Location { get; set; }

[JsonProperty(PropertyName = "description")]

public string Description { get; set; }

[JsonProperty(PropertyName = "url")]

public string Url { get; set; }

[JsonProperty(PropertyName = "entities")]

public Entities Entities { get; set; }

[JsonProperty(PropertyName = "followers\_count")]

public int FollowersCount { get; set; }

[JsonProperty(PropertyName = "friends\_count")]

public int FriendsCount { get; set; }

[JsonProperty(PropertyName = "listed\_count")]

public int ListedCount { get; set; }

[JsonProperty(PropertyName = "created\_at")]

public string CreatedAt { get; set; }

[JsonProperty(PropertyName = "favourites\_count")]

public int FavouritesCount { get; set; }

[JsonProperty(PropertyName = "utc\_offset")]

public int? UtcOffset { get; set; }

[JsonProperty(PropertyName = "time\_zone")]

public string TimeZone { get; set; }

[JsonProperty(PropertyName = "geo\_enabled")]

public bool GeoEnabled { get; set; }

[JsonProperty(PropertyName = "verified")]

public bool IsVerified { get; set; }

[JsonProperty(PropertyName = "statuses\_count")]

public int StatusesCount { get; set; }

[JsonProperty(PropertyName = "lang")]

public string Lang { get; set; }

[JsonProperty(PropertyName = "status")]

public Status Status { get; set; }

[JsonProperty(PropertyName = "contributors\_enabled")]

public bool ContributorsEnabled { get; set; }

[JsonProperty(PropertyName = "is\_translator")]

public bool IsTranslator { get; set; }

[JsonProperty(PropertyName = "is\_translation\_enabled")]

public bool IsTranslationEnabled { get; set; }

[JsonProperty(PropertyName = "profile\_background\_color")]

public string ProfileBackgroundColor { get; set; }

[JsonProperty(PropertyName = "profile\_background\_image\_url")]

public string ProfileBackgroundImageUrl { get; set; }

[JsonProperty(PropertyName = "profile\_background\_image\_url\_https")]

public string ProfileBackgroundImageUrlHttps { get; set; }

[JsonProperty(PropertyName = "profile\_background\_tile")]

public bool ProfileBackgroundTile { get; set; }

[JsonProperty(PropertyName = "profile\_image\_url")]

public string ProfileImageUrl { get; set; }

[JsonProperty(PropertyName = "profile\_image\_url\_https")]

public string ProfileImageUrlHttps { get; set; }

[JsonProperty(PropertyName = "profile\_banner\_url")]

public string ProfileBannerUrl { get; set; }

[JsonProperty(PropertyName = "profile\_link\_color")]

public string ProfileLinkColor { get; set; }

[JsonProperty(PropertyName = "profile\_sidebar\_border\_color")]

public string ProfileSidebarBorderColor { get; set; }

[JsonProperty(PropertyName = "profile\_sidebar\_fill\_color")]

public string ProfileSidebarFillColor { get; set; }

[JsonProperty(PropertyName = "profile\_text\_color")]

public string ProfileTextColor { get; set; }

[JsonProperty(PropertyName = "profile\_use\_background\_image")]

public bool ProfileUseBackgroundImage { get; set; }

[JsonProperty(PropertyName = "has\_extended\_profile")]

public bool HasExtendedProfile { get; set; }

[JsonProperty(PropertyName = "default\_profile")]

public bool DefaultProfile { get; set; }

[JsonProperty(PropertyName = "default\_profile\_image")]

public bool DefaultProfileImage { get; set; }

[JsonProperty(PropertyName = "following")]

public bool Following { get; set; }

[JsonProperty(PropertyName = "follow\_request\_sent")]

public bool FollowRequestSent { get; set; }

[JsonProperty(PropertyName = "notifications")]

public bool Notifications { get; set; }

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

public string TranslatorType { get; set; }

}

}

1. Crea en el API service los siguientes métodos:

public async Task<FacebookResponse> GetFacebook(string accessToken)

{

var requestUrl = "https://graph.facebook.com/v2.8/me/?fields=name," +

"picture.width(999),cover,age\_range,devices,email,gender," +

"is\_verified,birthday,languages,work,website,religion," +

"location,locale,link,first\_name,last\_name," +

"hometown&access\_token=" + accessToken;

var httpClient = new HttpClient();

var userJson = await httpClient.GetStringAsync(requestUrl);

var facebookResponse =

JsonConvert.DeserializeObject<FacebookResponse>(userJson);

return facebookResponse;

}

public async Task<InstagramResponse> GetInstagram(string accessToken)

{

var client = new HttpClient();

var userJson = await client.GetStringAsync(accessToken);

var InstagramJson = JsonConvert.DeserializeObject<InstagramResponse>(userJson);

return InstagramJson;

}

1. Crea estos métodos en el **ApiService**:

public async Task<TokenResponse> LoginTwitter(string urlBase, string servicePrefix, string controller, TwitterResponse profile)

{

try

{

var request = JsonConvert.SerializeObject(profile);

var content = new StringContent(

request,

Encoding.UTF8,

"application/json");

var client = new HttpClient();

client.BaseAddress = new Uri(urlBase);

var url = $"{servicePrefix}{controller}";

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

if (!response.IsSuccessStatusCode)

{

return null;

}

var tokenResponse = await GetToken(

urlBase,

profile.IdStr,

profile.IdStr);

return tokenResponse;

}

catch

{

return null;

}

}

public async Task<TokenResponse> LoginInstagram(string urlBase, string servicePrefix, string controller, InstagramResponse profile)

{

try

{

var request = JsonConvert.SerializeObject(profile);

var content = new StringContent(

request,

Encoding.UTF8,

"application/json");

var client = new HttpClient();

client.BaseAddress = new Uri(urlBase);

var url = $"{servicePrefix}{controller}";

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

if (!response.IsSuccessStatusCode)

{

return null;

}

var tokenResponse = await GetToken(

urlBase,

profile.UserData.Id,

profile.UserData.Id);

return tokenResponse;

}

catch

{

return null;

}

}

public async Task<TokenResponse> LoginFacebook(string urlBase, string servicePrefix, string controller, FacebookResponse profile)

{

try

{

var request = JsonConvert.SerializeObject(profile);

var content = new StringContent(

request,

Encoding.UTF8,

"application/json");

var client = new HttpClient();

client.BaseAddress = new Uri(urlBase);

var url = $"{servicePrefix}{controller}";

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

if (!response.IsSuccessStatusCode)

{

return null;

}

var tokenResponse = await GetToken(

urlBase,

profile.Id,

profile.Id);

return tokenResponse;

}

catch

{

return null;

}

}

1. Agregue estos métodos a la clase **App**:

public static Action HideLoginView

{

get

{

return new Action(() => Current.MainPage = new NavigationPage(new LoginPage()));

}

}

public static async Task NavigateToProfile(TokenResponse token)

{

if (token == null)

{

Application.Current.MainPage = new NavigationPage(new LoginPage());

return;

}

Settings.IsRemembered = true;

Settings.AccessToken = token.AccessToken;

Settings.TokenType = token.TokenType;

var apiService = new ApiService();

var url = Application.Current.Resources["UrlAPI"].ToString();

var prefix = Application.Current.Resources["UrlPrefix"].ToString();

var controller = Application.Current.Resources["UrlUsersController"].ToString();

var response = await apiService.GetUser(url, prefix, $"{controller}/GetUser", token.UserName, token.TokenType, token.AccessToken);

if (response.IsSuccess)

{

var userASP = (UserASP)response.Result;

MainViewModel.GetInstance().UserASP = userASP;

Settings.UserASP = JsonConvert.SerializeObject(userASP);

}

MainViewModel.GetInstance().Products = new ProductsViewModel();

Application.Current.MainPage = new MasterPage();

}

1. Agregue la clase: **LoginFacebookPageRenderer** en el proyecto Android:

[assembly: Xamarin.Forms.ExportRenderer(

typeof(Sales.Views.LoginFacebookPage),

typeof(Sales.Droid.Implementations.LoginFacebookPageRenderer))]

namespace Sales.Droid.Implementations

{

using System;

using System.Threading.Tasks;

using Android.App;

using Common.Models;

using Services;

using Xamarin.Auth;

using Xamarin.Forms.Platform.Android;

public class LoginFacebookPageRenderer : PageRenderer

{

public LoginFacebookPageRenderer()

{

var activity = this.Context as Activity;

var facebookAppID = Xamarin.Forms.Application.Current.Resources["FacebookAppID"].ToString();

var facebookAuthURL = Xamarin.Forms.Application.Current.Resources["FacebookAuthURL"].ToString();

var facebookRedirectURL = Xamarin.Forms.Application.Current.Resources["FacebookRedirectURL"].ToString();

var facebookScope = Xamarin.Forms.Application.Current.Resources["FacebookScope"].ToString();

var auth = new OAuth2Authenticator(

clientId: facebookAppID,

scope: facebookScope,

authorizeUrl: new Uri(facebookAuthURL),

redirectUrl: new Uri(facebookRedirectURL));

auth.Completed += async (sender, eventArgs) =>

{

if (eventArgs.IsAuthenticated)

{

var accessToken = eventArgs.Account.Properties["access\_token"].ToString();

var token = await GetFacebookProfileAsync(accessToken);

await App.NavigateToProfile(token);

}

else

{

App.HideLoginView();

}

};

activity.StartActivity(auth.GetUI(activity));

}

private async Task<TokenResponse> GetFacebookProfileAsync(string accessToken)

{

var url = Xamarin.Forms.Application.Current.Resources["UrlAPI"].ToString();

var prefix = Xamarin.Forms.Application.Current.Resources["UrlPrefix"].ToString();

var controller = Xamarin.Forms.Application.Current.Resources["UrlUsersController"].ToString();

var apiService = new ApiService();

var facebookResponse = await apiService.GetFacebook(accessToken);

var token = await apiService.LoginFacebook(

url,

prefix,

$"{controller}/LoginFacebook",

facebookResponse);

return token;

}

}

}

1. Agregue la clase: **LoginFacebookPageRenderer** en el proyecto iOS:

[assembly: Xamarin.Forms.ExportRenderer(

typeof(Sales.Views.LoginFacebookPage),

typeof(Sales.iOS.Implementations.LoginFacebookPageRenderer))]

namespace Sales.iOS.Implementations

{

using System;

using System.Threading.Tasks;

using Common.Models;

using Services;

using Xamarin.Auth;

using Xamarin.Forms.Platform.iOS;

public class LoginFacebookPageRenderer : PageRenderer

{

bool done = false;

public override void ViewDidAppear(bool animated)

{

base.ViewDidAppear(animated);

if (done)

{

return;

}

var facebookAppID = Xamarin.Forms.Application.Current.Resources["FacebookAppID"].ToString();

var facebookAuthURL = Xamarin.Forms.Application.Current.Resources["FacebookAuthURL"].ToString();

var facebookRedirectURL = Xamarin.Forms.Application.Current.Resources["FacebookRedirectURL"].ToString();

var facebookScope = Xamarin.Forms.Application.Current.Resources["FacebookScope"].ToString();

var auth = new OAuth2Authenticator(

clientId: facebookAppID,

scope: facebookScope,

authorizeUrl: new Uri(facebookAuthURL),

redirectUrl: new Uri(facebookRedirectURL));

auth.Completed += async (sender, eventArgs) =>

{

DismissViewController(true, null);

App.HideLoginView();

if (eventArgs.IsAuthenticated)

{

var accessToken = eventArgs.Account.Properties["access\_token"].ToString();

var token = await GetFacebookProfileAsync(accessToken);

await App.NavigateToProfile(token);

}

else

{

App.HideLoginView();

}

};

done = true;

PresentViewController(auth.GetUI(), true, null);

}

private async Task<TokenResponse> GetFacebookProfileAsync(string accessToken)

{

var url = Xamarin.Forms.Application.Current.Resources["UrlAPI"].ToString();

var prefix = Xamarin.Forms.Application.Current.Resources["UrlPrefix"].ToString();

var controller = Xamarin.Forms.Application.Current.Resources["UrlUsersController"].ToString();

var apiService = new ApiService();

var facebookResponse = await apiService.GetFacebook(accessToken);

var token = await apiService.LoginFacebook(

url,

prefix,

$"{controller}/LoginFacebook",

facebookResponse);

return token;

}

}

}

1. Agregue la clase: **LoginInstagramPageRenderer** en el proyecto Android:

[assembly: Xamarin.Forms.ExportRenderer(

typeof(Sales.Views.LoginInstagramPage),

typeof(Sales.Droid.Implementations.LoginInstagramPageRenderer))]

namespace Sales.Droid.Implementations

{

using System;

using System.Threading.Tasks;

using Android.App;

using Sales;

using Sales.Common.Models;

using Sales.Services;

using Xamarin.Auth;

using Xamarin.Forms;

using Xamarin.Forms.Platform.Android;

public class LoginInstagramPageRenderer : PageRenderer

{

protected override void OnElementChanged(ElementChangedEventArgs<Page> e)

{

base.OnElementChanged(e);

var activity = this.Context as Activity;

var InstagramAppID = Xamarin.Forms.Application.Current.Resources["InstagramAppID"].ToString();

var InstagramAuthURL = Xamarin.Forms.Application.Current.Resources["InstagramAuthURL"].ToString();

var InstagramRedirectURL = Xamarin.Forms.Application.Current.Resources["InstagramRedirectURL"].ToString();

var InstagramScope = Xamarin.Forms.Application.Current.Resources["InstagramScope"].ToString();

var auth = new OAuth2Authenticator(

clientId: InstagramAppID,

scope: InstagramScope,

authorizeUrl: new Uri(InstagramAuthURL),

redirectUrl: new Uri(InstagramRedirectURL));

auth.Completed += async (sender, eventArgs) =>

{

if (eventArgs.IsAuthenticated)

{

var accessToken = eventArgs.Account.Properties["access\_token"].ToString();

var profile = await GetInstagramProfileAsync(accessToken);

App.NavigateToProfile(profile, "Instagram");

}

else

{

App.HideLoginView();

}

};

activity.StartActivity(auth.GetUI(activity));

}

public async Task<InstagramResponse> GetInstagramProfileAsync(string accessToken)

{

var InstagramProfileInfoURL = Xamarin.Forms.Application.Current.Resources["InstagramProfileInfoURL"].ToString();

var requestUrl = string.Format("{0}={1}",

InstagramProfileInfoURL,

accessToken);

var apiService = new ApiService();

return await apiService.GetInstagram(requestUrl);

}

}

}

1. Agregue la clase: **LoginInstagramPageRenderer** en el proyecto iOS:

[assembly: Xamarin.Forms.ExportRenderer(

typeof(Sales.Views.LoginInstagramPage),

typeof(Sales.iOS.Implementations.LoginInstagramPageRenderer))]

namespace Sales.iOS.Implementations

{

using System;

using System.Threading.Tasks;

using Common.Models;

using Services;

using Xamarin.Auth;

using Xamarin.Forms.Platform.iOS;

public class LoginInstagramPageRenderer : PageRenderer

{

bool done = false;

public override void ViewDidAppear(bool animated)

{

base.ViewDidAppear(animated);

if (done)

{

return;

}

var instagramAppID = Xamarin.Forms.Application.Current.Resources["InstagramAppID"].ToString();

var instagramAuthURL = Xamarin.Forms.Application.Current.Resources["InstagramAuthURL"].ToString();

var instagramRedirectURL = Xamarin.Forms.Application.Current.Resources["InstagramRedirectURL"].ToString();

var instagramScope = Xamarin.Forms.Application.Current.Resources["InstagramScope"].ToString();

var auth = new OAuth2Authenticator(

clientId: instagramAppID,

scope: instagramScope,

authorizeUrl: new Uri(instagramAuthURL),

redirectUrl: new Uri(instagramRedirectURL));

auth.Completed += async (sender, eventArgs) =>

{

DismissViewController(true, null);

App.HideLoginView();

if (eventArgs.IsAuthenticated)

{

var accessToken = eventArgs.Account.Properties["access\_token"].ToString();

var profile = await GetInstagramProfileAsync(accessToken);

App.NavigateToProfile(profile, "Instagram");

}

else

{

App.HideLoginView();

}

};

done = true;

PresentViewController(auth.GetUI(), true, null);

}

public async Task<InstagramResponse> GetInstagramProfileAsync(string accessToken)

{

var InstagramProfileInfoURL = Xamarin.Forms.Application.Current.Resources["InstagramProfileInfoURL"].ToString();

var requestUrl = string.Format("{0}={1}",

InstagramProfileInfoURL,

accessToken);

var apiService = new ApiService();

return await apiService.GetInstagram(requestUrl);

}

}

}

1. Agregue la clase: **LoginTwitterPageRenderer** en el proyecto Android:

[assembly: Xamarin.Forms.ExportRenderer(

typeof(Sales.Views.LoginTwitterPage),

typeof(Sales.Droid.Implementations.LoginTwitterPageRenderer))]

namespace Sales.Droid.Implementations

{

using System;

using System.Threading.Tasks;

using Android.App;

using Common.Models;

using Newtonsoft.Json;

using Xamarin.Auth;

using Xamarin.Forms;

using Xamarin.Forms.Platform.Android;

public class LoginTwitterPageRenderer : PageRenderer

{

protected override void OnElementChanged(ElementChangedEventArgs<Page> e)

{

base.OnElementChanged(e);

var activity = this.Context as Activity;

var twitterConsumerKey = Xamarin.Forms.Application.Current.Resources["TwitterConsumerKey"].ToString();

var twitterConsumerSecret = Xamarin.Forms.Application.Current.Resources["TwitterConsumerSecret"].ToString();

var twitterRequestTokenURL = Xamarin.Forms.Application.Current.Resources["TwitterRequestTokenURL"].ToString();

var twitterAuthorizeURL = Xamarin.Forms.Application.Current.Resources["TwitterAuthorizeURL"].ToString();

var url = Xamarin.Forms.Application.Current.Resources["Url"].ToString();

var twitterAccessTokenURL = Xamarin.Forms.Application.Current.Resources["TwitterAccessTokenURL"].ToString();

var auth = new OAuth1Authenticator(

consumerKey: twitterConsumerKey,

consumerSecret: twitterConsumerSecret,

requestTokenUrl: new Uri(twitterRequestTokenURL),

authorizeUrl: new Uri(twitterAuthorizeURL), callbackUrl: new Uri(url),

accessTokenUrl: new Uri(twitterAccessTokenURL));

auth.Completed += async (sender, eventArgs) =>

{

if (eventArgs.IsAuthenticated)

{

TwitterResponse profile = await GetTwitterProfileAsync(eventArgs.Account);

App.NavigateToProfile(profile, "Twitter");

}

else

{

App.HideLoginView();

}

};

activity.StartActivity(auth.GetUI(activity));

}

public async Task<TwitterResponse> GetTwitterProfileAsync(Account account)

{

var TwitterProfileInfoURL = Xamarin.Forms.Application.Current.Resources["TwitterProfileInfoURL"].ToString(); var requestUrl = new OAuth1Request(

"GET",

new Uri(TwitterProfileInfoURL), null,

account);

var response = await requestUrl.GetResponseAsync();

return JsonConvert.DeserializeObject<TwitterResponse>(response.GetResponseText());

}

}

}

1. Agregue la clase: **LoginTwitterPageRenderer** en el proyecto iOS:

[assembly: Xamarin.Forms.ExportRenderer(

typeof(Sales.Views.LoginTwitterPage),

typeof(Sales.iOS.Implementations.LoginTwitterPageRenderer))]

namespace Sales.iOS.Implementations

{

using System;

using System.Threading.Tasks;

using Common.Models;

using Newtonsoft.Json;

using Xamarin.Auth;

using Xamarin.Forms.Platform.iOS;

public class LoginTwitterPageRenderer : PageRenderer

{

bool done = false;

public override void ViewDidAppear(bool animated)

{

base.ViewDidAppear(animated);

if (done)

{

return;

}

var twitterConsumerKey = Xamarin.Forms.Application.Current.Resources["TwitterConsumerKey"].ToString();

var twitterConsumerSecret = Xamarin.Forms.Application.Current.Resources["TwitterConsumerSecret"].ToString();

var twitterRequestTokenURL = Xamarin.Forms.Application.Current.Resources["TwitterRequestTokenURL"].ToString();

var twitterAuthorizeURL = Xamarin.Forms.Application.Current.Resources["TwitterAuthorizeURL"].ToString();

var url = Xamarin.Forms.Application.Current.Resources["Url"].ToString();

var twitterAccessTokenURL = Xamarin.Forms.Application.Current.Resources["TwitterAccessTokenURL"].ToString();

var auth = new OAuth1Authenticator(

consumerKey: twitterConsumerKey,

consumerSecret: twitterConsumerSecret,

requestTokenUrl: new Uri(twitterRequestTokenURL),

authorizeUrl: new Uri(twitterAuthorizeURL),

callbackUrl: new Uri(url),

accessTokenUrl: new Uri(twitterAccessTokenURL));

auth.Completed += async (sender, eventArgs) =>

{

DismissViewController(true, null); App.HideLoginView();

if (eventArgs.IsAuthenticated)

{

var profile = await GetTwitterProfileAsync(eventArgs.Account);

App.NavigateToProfile(profile, "Twitter");

}

else

{

App.HideLoginView();

}

};

done = true;

PresentViewController(auth.GetUI(), true, null);

}

public async Task<TwitterResponse> GetTwitterProfileAsync(Account account)

{

var TwitterProfileInfoURL = Xamarin.Forms.Application.Current.Resources["TwitterProfileInfoURL"].ToString(); var requestUrl = new OAuth1Request(

"GET",

new Uri(TwitterProfileInfoURL), null,

account);

var response = await requestUrl.GetResponseAsync();

return JsonConvert.DeserializeObject<TwitterResponse>(response.GetResponseText());

}

}

}

1. Antes de probar, debemos de crear los métodos en el API que quedamos pendientes:

[HttpPost]

[Route("LoginFacebook")]

public IHttpActionResult LoginFacebook(FacebookResponse profile)

{

var user = UsersHelper.GetUserASP(profile.Id);

if (user != null)

{

return Ok(true);

}

var userRequest = new UserRequest

{

EMail = profile.Id,

FirstName = profile.FirstName,

ImagePath = profile.Picture.Data.Url,

LastName = profile.LastName,

Password = profile.Id,

};

var answer = UsersHelper.CreateUserASP(userRequest);

return Ok(answer);

}

1. Cambiemos la propiedad de **UserImageFullPath** para que nos muestre en el menú la foto correcta de las redes sociales:

public string UserImageFullPath

{

get

{

foreach (var claim in this.UserASP.Claims)

{

if (claim.ClaimType == "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/uri")

{

if (claim.ClaimValue.StartsWith("~"))

{

return $"https://salesapiservices.azurewebsites.net{claim.ClaimValue.Substring(1)}";

}

return claim.ClaimValue;

}

}

return null;

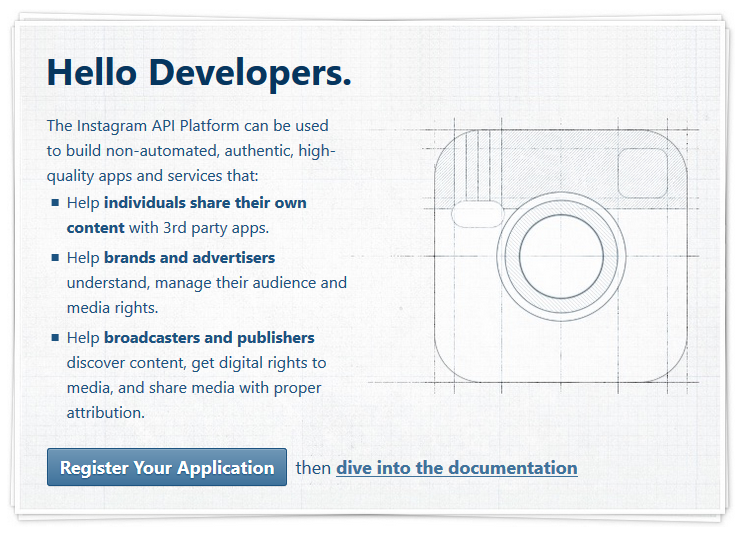
}

}

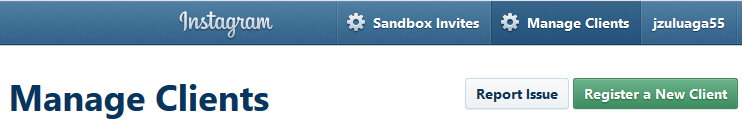
1. Publiquemos el nuevo API y probemos que funcione bien el Login por cada una de las redes sociales, iniciamos con Facebook.

## Instagram

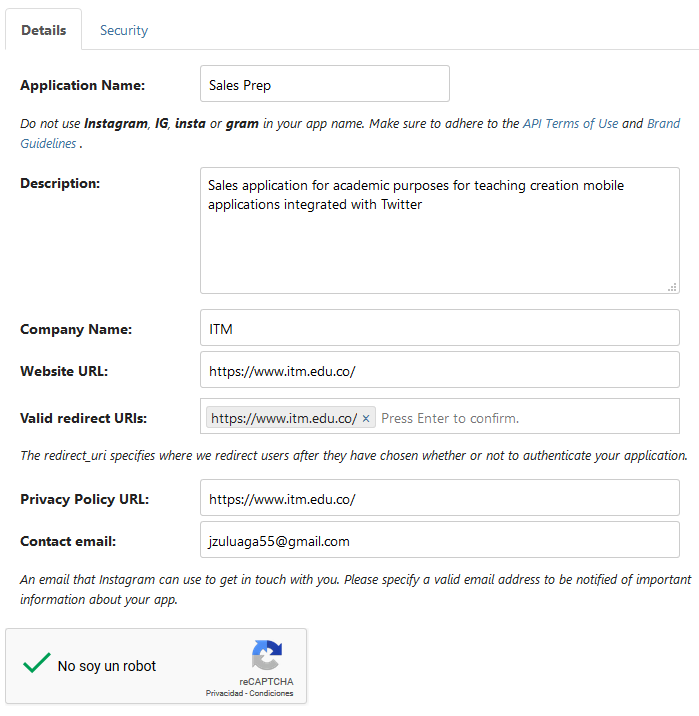
1. Matricular la App en Instagram, Ingresar a <https://www.instagram.com/developer/>



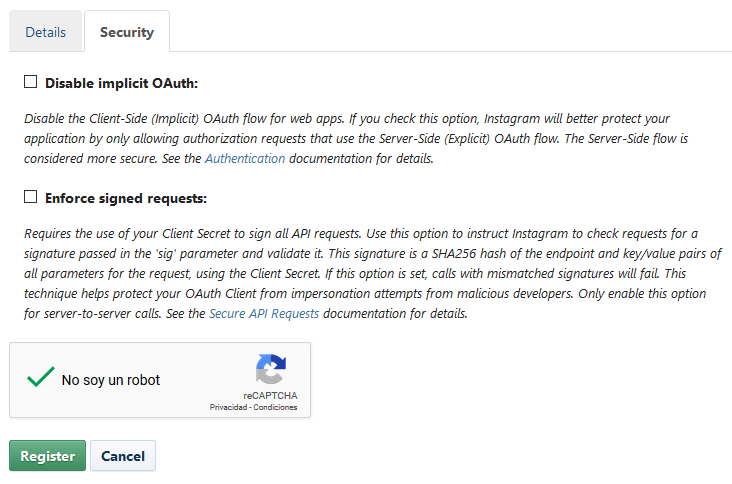
Y registra una nueva aplicación:



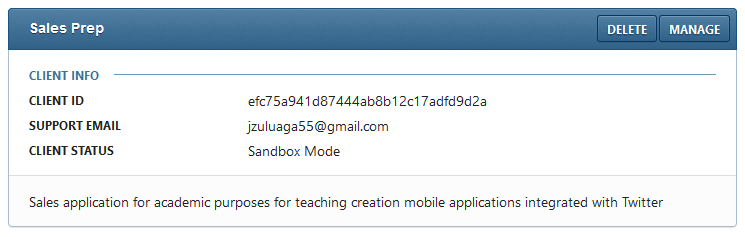
Completa los datos del formulario y coloca como dirección de retorno: <https://www.instagram.com>



Y en la opción de security desactiva la casilla “Disable implicit OAuth:”



Luego de esto registra la App, guarda el Client ID que lo vamos a necesitar más adelante:



1. Actualiza el Client ID en el diccionario de recursos:

<!-- Instagram -->

<x:String x:Key="InstagramAppID">efc75a941d87444ab8b12c17adfd9d2a</x:String>

<x:String x:Key="InstagramAuthURL">https://api.instagram.com/oauth/authorize</x:String>

<x:String x:Key="InstagramRedirectURL">https://www.instagram.com</x:String>

<x:String x:Key="InstagramScope">basic</x:String>

<x:String x:Key="InstagramProfileInfoURL">https://api.instagram.com/v1/users/self/?access\_token</x:String>

1. Crear el método en el API **LoginInstagram** y publicarlo:

[HttpPost]

[Route("LoginInstagram")]

public IHttpActionResult LoginInstagram(InstagramResponse profile)

{

var user = UsersHelper.GetUserASP(profile.UserData.Id);

if (user != null)

{

return Ok(true); // TODO: Pending update the user with new instagram data

}

var firstName = string.Empty;

var lastName = string.Empty;

var fullName = profile.UserData.FullName;

var posSpace = fullName.IndexOf(' ');

if (posSpace == -1)

{

firstName = fullName;

lastName = fullName;

}

else

{

firstName = fullName.Substring(0, posSpace);

lastName = fullName.Substring(posSpace + 1);

}

var userRequest = new UserRequest

{

EMail = profile.UserData.Id,

FirstName = firstName,

ImagePath = profile.UserData.ProfilePicture,

LastName = lastName,

Password = profile.UserData.Id,

};

var answer = UsersHelper.CreateUserASP(userRequest);

return Ok(answer);

}

1. Agregamos la clase **LoginInstagramPageRenderer** en el proyecto **Android**.

[assembly: Xamarin.Forms.ExportRenderer(

typeof(Sales.Views.LoginInstagramPage),

typeof(Sales.Droid.Implementations.LoginInstagramPageRenderer))]

namespace Sales.Droid.Implementations

{

using System;

using System.Threading.Tasks;

using Android.App;

using Common.Models;

using Services;

using Xamarin.Auth;

using Xamarin.Forms;

using Xamarin.Forms.Platform.Android;

public class LoginInstagramPageRenderer : PageRenderer

{

protected override void OnElementChanged(ElementChangedEventArgs<Page> e)

{

base.OnElementChanged(e);

var activity = this.Context as Activity;

var InstagramAppID = Xamarin.Forms.Application.Current.Resources["InstagramAppID"].ToString();

var InstagramAuthURL = Xamarin.Forms.Application.Current.Resources["InstagramAuthURL"].ToString();

var InstagramRedirectURL = Xamarin.Forms.Application.Current.Resources["InstagramRedirectURL"].ToString();

var InstagramScope = Xamarin.Forms.Application.Current.Resources["InstagramScope"].ToString();

var auth = new OAuth2Authenticator(

clientId: InstagramAppID,

scope: InstagramScope,

authorizeUrl: new Uri(InstagramAuthURL),

redirectUrl: new Uri(InstagramRedirectURL));

auth.Completed += async (sender, eventArgs) =>

{

if (eventArgs.IsAuthenticated)

{

var accessToken = eventArgs.Account.Properties["access\_token"].ToString();

var token = await GetInstagramProfileAsync(accessToken);

App.NavigateToProfile(token);

}

else

{

App.HideLoginView();

}

};

activity.StartActivity(auth.GetUI(activity));

}

public async Task<TokenResponse> GetInstagramProfileAsync(string accessToken)

{

var InstagramProfileInfoURL = Xamarin.Forms.Application.Current.Resources["InstagramProfileInfoURL"].ToString();

var requestUrl = string.Format("{0}={1}",

InstagramProfileInfoURL,

accessToken);

var apiService = new ApiService();

var responseInstagram = await apiService.GetInstagram(requestUrl);

var url = Xamarin.Forms.Application.Current.Resources["UrlAPI"].ToString();

var prefix = Xamarin.Forms.Application.Current.Resources["UrlPrefix"].ToString();

var controller = Xamarin.Forms.Application.Current.Resources["UrlUsersController"].ToString();

var token = await apiService.LoginInstagram(

url,

prefix,

$"{controller}/LoginInstagram",

responseInstagram);

return token;

}

}

}

1. Agregamos la clase **LoginInstagramPageRenderer** en el proyecto **iOS**.

[assembly: Xamarin.Forms.ExportRenderer(

typeof(Sales.Views.LoginInstagramPage),

typeof(Sales.iOS.Implementations.LoginInstagramPageRenderer))]

namespace Sales.iOS.Implementations

{

using System;

using System.Threading.Tasks;

using Common.Models;

using Services;

using Xamarin.Auth;

using Xamarin.Forms.Platform.iOS;

public class LoginInstagramPageRenderer : PageRenderer

{

bool done = false;

public override void ViewDidAppear(bool animated)

{

base.ViewDidAppear(animated);

if (done)

{

return;

}

var instagramAppID = Xamarin.Forms.Application.Current.Resources["InstagramAppID"].ToString();

var instagramAuthURL = Xamarin.Forms.Application.Current.Resources["InstagramAuthURL"].ToString();

var instagramRedirectURL = Xamarin.Forms.Application.Current.Resources["InstagramRedirectURL"].ToString();

var instagramScope = Xamarin.Forms.Application.Current.Resources["InstagramScope"].ToString();

var auth = new OAuth2Authenticator(

clientId: instagramAppID,

scope: instagramScope,

authorizeUrl: new Uri(instagramAuthURL),

redirectUrl: new Uri(instagramRedirectURL));

auth.Completed += async (sender, eventArgs) =>

{

DismissViewController(true, null);

App.HideLoginView();

if (eventArgs.IsAuthenticated)

{

var accessToken = eventArgs.Account.Properties["access\_token"].ToString();

var token = await GetInstagramProfileAsync(accessToken);

App.NavigateToProfile(token);

}

else

{

App.HideLoginView();

}

};

done = true;

PresentViewController(auth.GetUI(), true, null);

}

public async Task<TokenResponse> GetInstagramProfileAsync(string accessToken)

{

var InstagramProfileInfoURL = Xamarin.Forms.Application.Current.Resources["InstagramProfileInfoURL"].ToString();

var requestUrl = string.Format("{0}={1}",

InstagramProfileInfoURL,

accessToken);

var apiService = new ApiService();

var responseInstagram = await apiService.GetInstagram(requestUrl);

var url = Xamarin.Forms.Application.Current.Resources["UrlAPI"].ToString();

var prefix = Xamarin.Forms.Application.Current.Resources["UrlPrefix"].ToString();

var controller = Xamarin.Forms.Application.Current.Resources["UrlUsersController"].ToString();

var token = await apiService.LoginInstagram(

url,

prefix,

$"{controller}/LoginInstagram",

responseInstagram);

return token;

}

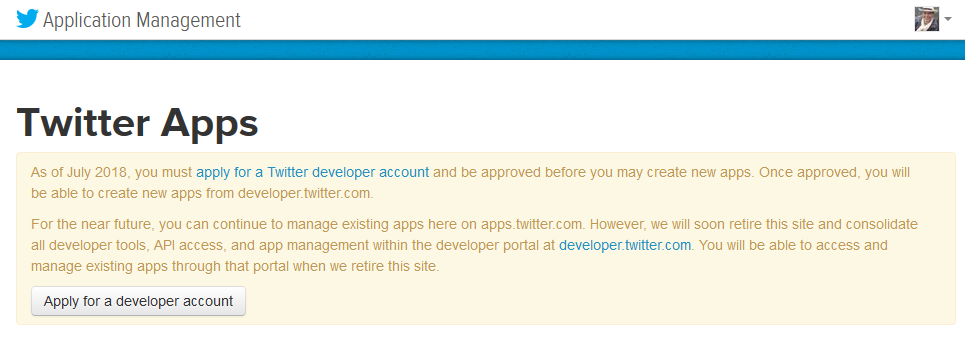
}

}

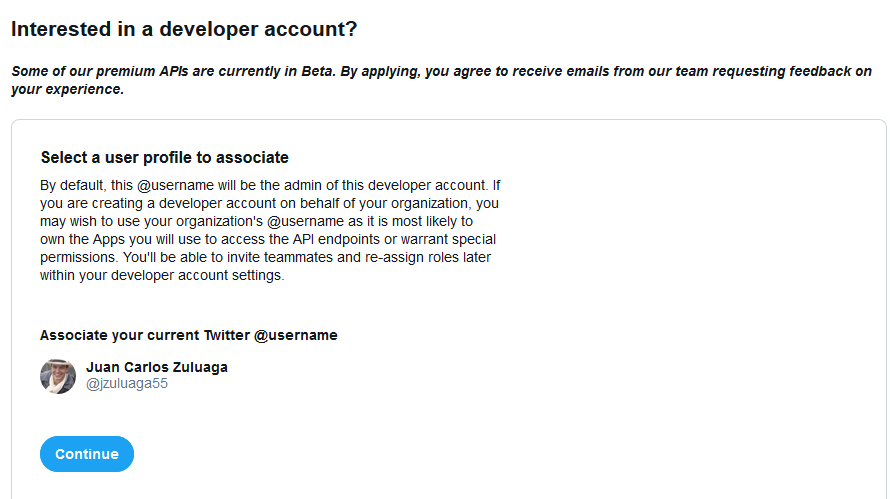
1. Ya podemos probar la integración con **Instagram**.

## Twitter

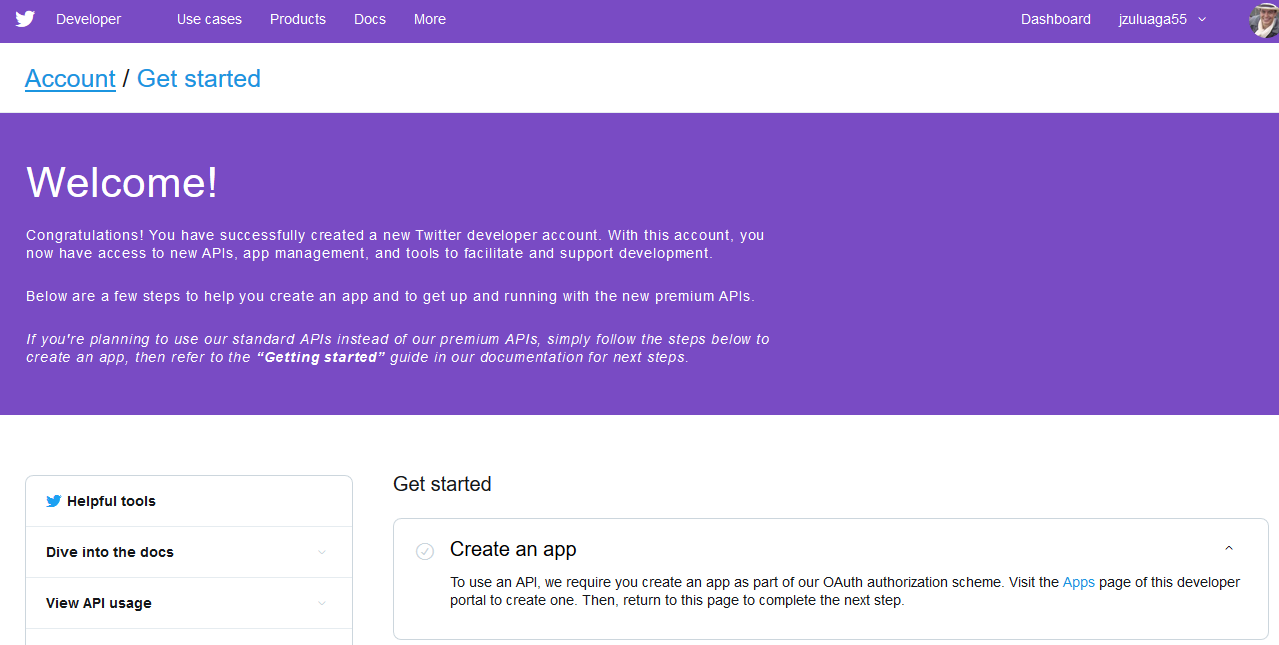
1. Matricular la App en Twitter, nos dirigimos al link de Twitter (<https://apps.twitter.com>) para crear nuestra App e iniciamos sesión y nos aparecerá esta pantalla.



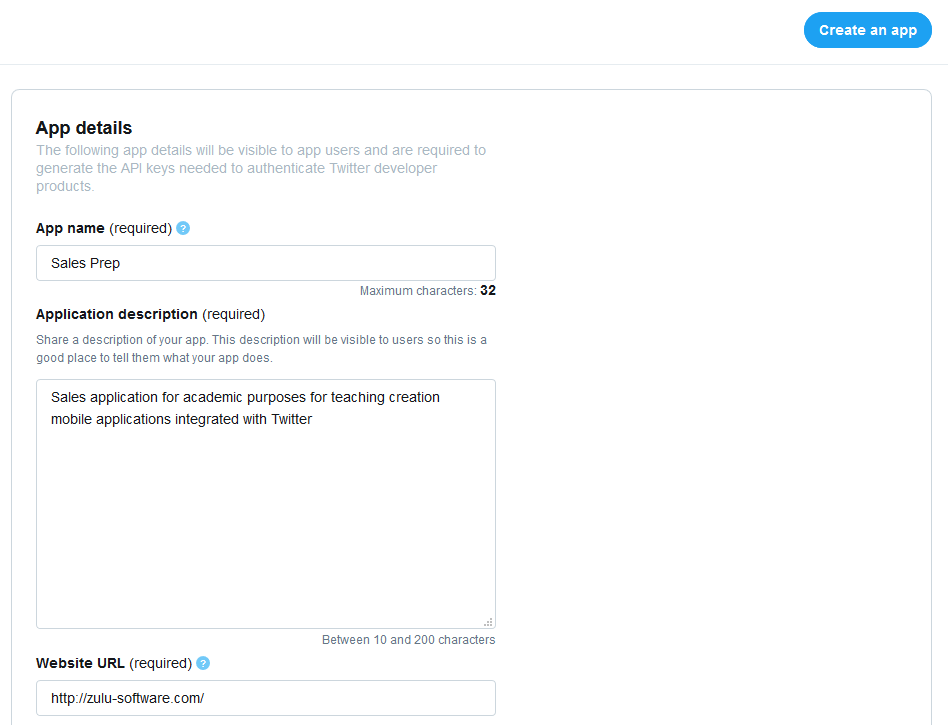
Y debemos de aplicar para crear una cuenta de desarrollador.



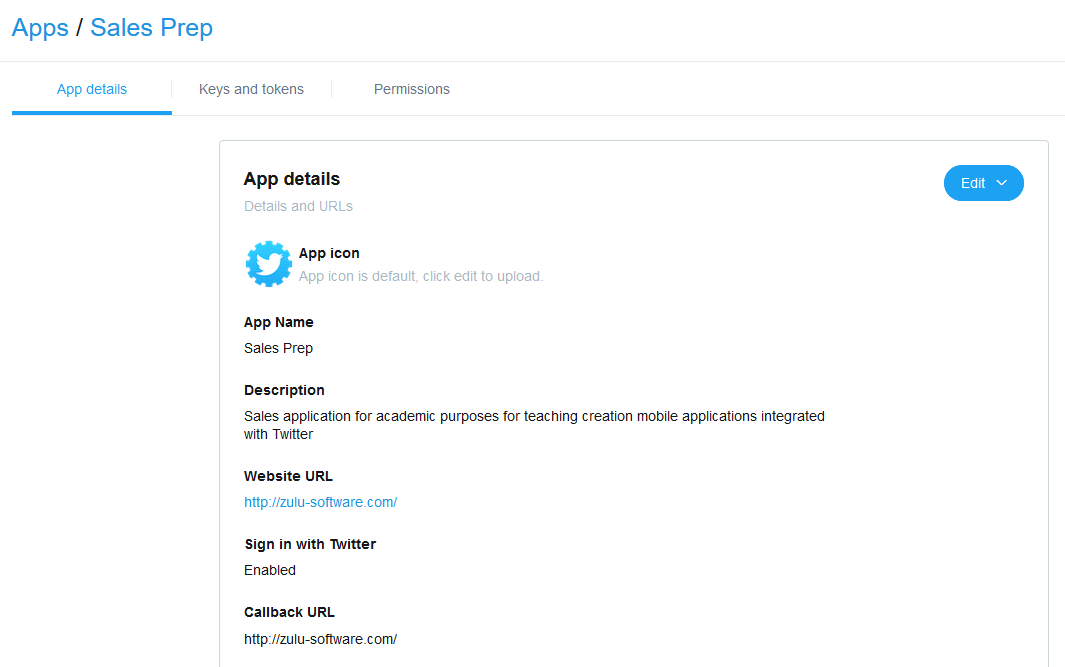
Siga los pasos para hacer la solicitud de cuenta de desarrollador de Twitter hasta que le llegue el correo de confirmación y llegue a esta pantalla:



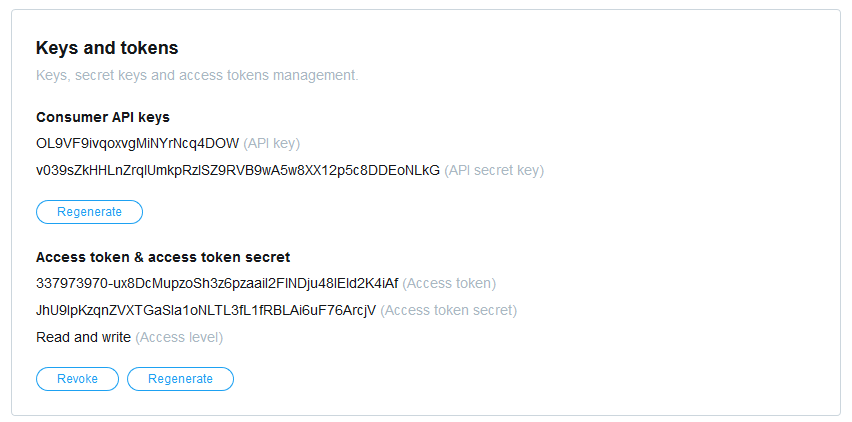
Entonces haga clic en la opción de crear una nueva App:



Coloca un nombre, puedes colocar esto en la descripción: “Sales application for academic purposes for teaching creation mobile applications integrated with Twitter” y puedes colocar esto en las URL: <https://www.itm.edu.co> Luego de esto te debe salir una pantalla similar a esta, donde podrás colocar un ícono a la App y obtener los Keys necesarios para integrarlo con nuestra aplicación móvil:



Damos clic en generar access token y obtenemos todas las llaves necesarias:



Luego vamos a usar estos valores en el diccionario de recursos de nuestra app móvil.

1. Vamos a reemplazar los valores obtenidos por el portal de Twitter en nuestro diccionario de recursos:

<!-- Twitter -->

<x:String x:Key="TwitterKey">8BGQv1IC3oS25Nd9OKjEjkZPt</x:String>

<x:String x:Key="TwitterSecret">XEQMBCozx44OYwsqlCd95sKnfNnUxGQXZwKCmCT3dhnbFErAoC</x:String>

<x:String x:Key="TwitterRequestURL">https://api.twitter.com/oauth/request\_token</x:String>

<x:String x:Key="TwitterAuthURL">https://twitter.com/oauth/authenticate</x:String>

<x:String x:Key="TwitterURLAccess">https://api.twitter.com/oauth/access\_token</x:String>

<x:String x:Key="TwitterCallbackURL">https://mobile.twitter.com/</x:String>

<x:String x:Key="TwitterProfileInfoURL">https://api.twitter.com/1.1/account/verify\_credentials.json</x:String>

1. Crear el método **LoginTwitter** en el proyecto **API** y publicarlo:

[HttpPost]

[Route("LoginTwitter")]

public IHttpActionResult LoginTwitter(TwitterResponse profile)

{

var user = UsersHelper.GetUserASP(profile.IdStr);

if (user != null)

{

return Ok(true); // TODO: Pending update the user with new twitter data

}

var firstName = string.Empty;

var lastName = string.Empty;

var fullName = profile.Name;

var posSpace = fullName.IndexOf(' ');

if (posSpace == -1)

{

firstName = fullName;

lastName = fullName;

}

else

{

firstName = fullName.Substring(0, posSpace);

lastName = fullName.Substring(posSpace + 1);

}

var userRequest = new UserRequest

{

EMail = profile.IdStr,

FirstName = firstName,

ImagePath = profile.ProfileImageUrl,

LastName = lastName,

Password = profile.IdStr,

};

var answer = UsersHelper.CreateUserASP(userRequest);

return Ok(answer);

}

1. Agregamos la clase **LoginTwitterPageRenderer** en el proyecto **Android**.

[assembly: Xamarin.Forms.ExportRenderer(

typeof(Sales.Views.LoginTwitterPage),

typeof(Sales.Droid.Implementations.LoginTwitterPageRenderer))]

namespace Sales.Droid.Implementations

{

using Android.App;

using Common.Models;

using Newtonsoft.Json;

using Sales.Services;

using System;

using System.Threading.Tasks;

using Xamarin.Auth;

using Xamarin.Forms;

using Xamarin.Forms.Platform.Android;

public class LoginTwitterPageRenderer : PageRenderer

{

protected override void OnElementChanged(ElementChangedEventArgs<Page> e)

{

base.OnElementChanged(e);

var activity = this.Context as Activity;

var TwitterKey = Xamarin.Forms.Application.Current.Resources["TwitterKey"].ToString();

var TwitterSecret = Xamarin.Forms.Application.Current.Resources["TwitterSecret"].ToString();

var TwitterRequestURL = Xamarin.Forms.Application.Current.Resources["TwitterRequestURL"].ToString();

var TwitterAuthURL = Xamarin.Forms.Application.Current.Resources["TwitterAuthURL"].ToString();

var TwitterCallbackURL = Xamarin.Forms.Application.Current.Resources["TwitterCallbackURL"].ToString();

var TwitterURLAccess = Xamarin.Forms.Application.Current.Resources["TwitterURLAccess"].ToString();

var auth = new OAuth1Authenticator(

consumerKey: TwitterKey,

consumerSecret: TwitterSecret,

requestTokenUrl: new Uri(TwitterRequestURL),

authorizeUrl: new Uri(TwitterAuthURL),

callbackUrl: new Uri(TwitterCallbackURL),

accessTokenUrl: new Uri(TwitterURLAccess));

auth.Completed += async (sender, eventArgs) =>

{

if (eventArgs.IsAuthenticated)

{

var token = await GetTwitterProfileAsync(eventArgs.Account);

App.NavigateToProfile(token);

}

else

{

App.HideLoginView();

}

};

activity.StartActivity(auth.GetUI(activity));

}

public async Task<TokenResponse> GetTwitterProfileAsync(Account account)

{

var TwitterProfileInfoURL = Xamarin.Forms.Application.Current.Resources["TwitterProfileInfoURL"].ToString(); var requestUrl = new OAuth1Request(

"GET",

new Uri(TwitterProfileInfoURL), null,

account);

var response = await requestUrl.GetResponseAsync();

var responseTwitter = JsonConvert.DeserializeObject<TwitterResponse>(response.GetResponseText());

var url = Xamarin.Forms.Application.Current.Resources["UrlAPI"].ToString();

var prefix = Xamarin.Forms.Application.Current.Resources["UrlPrefix"].ToString();

var controller = Xamarin.Forms.Application.Current.Resources["UrlUsersController"].ToString();

var apiService = new ApiService();

var token = await apiService.LoginTwitter(

url,

prefix,

$"{controller}/LoginTwitter",

responseTwitter);

return token;

}

}

}

1. Agregamos la clase **LoginTwitterPageRenderer** en el proyecto **iOS**.

[assembly: Xamarin.Forms.ExportRenderer(

typeof(Sales.Views.LoginTwitterPage),

typeof(Sales.iOS.Implementations.LoginTwitterPageRenderer))]

namespace Sales.iOS.Implementations

{

using System;

using System.Threading.Tasks;

using Common.Models;

using Newtonsoft.Json;

using Services;

using Xamarin.Auth;

using Xamarin.Forms.Platform.iOS;

public class LoginTwitterPageRenderer : PageRenderer

{

bool done = false;

public override void ViewDidAppear(bool animated)

{

base.ViewDidAppear(animated);

if (done)

{

return;

}

var TwitterKey = Xamarin.Forms.Application.Current.Resources["TwitterKey"].ToString();

var TwitterSecret = Xamarin.Forms.Application.Current.Resources["TwitterSecret"].ToString();

var TwitterRequestURL = Xamarin.Forms.Application.Current.Resources["TwitterRequestURL"].ToString();

var TwitterAuthURL = Xamarin.Forms.Application.Current.Resources["TwitterAuthURL"].ToString();

var TwitterCallbackURL = Xamarin.Forms.Application.Current.Resources["TwitterCallbackURL"].ToString();

var TwitterURLAccess = Xamarin.Forms.Application.Current.Resources["TwitterURLAccess"].ToString();

var auth = new OAuth1Authenticator(

consumerKey: TwitterKey,

consumerSecret: TwitterSecret,

requestTokenUrl: new Uri(TwitterRequestURL),

authorizeUrl: new Uri(TwitterAuthURL),

callbackUrl: new Uri(TwitterCallbackURL),

accessTokenUrl: new Uri(TwitterURLAccess));

auth.Completed += async (sender, eventArgs) =>

{

DismissViewController(true, null); App.HideLoginView();

if (eventArgs.IsAuthenticated)

{

var token = await GetTwitterProfileAsync(eventArgs.Account);

App.NavigateToProfile(token);

}

else

{

App.HideLoginView();

}

};

done = true;

PresentViewController(auth.GetUI(), true, null);

}

public async Task<TokenResponse> GetTwitterProfileAsync(Account account)

{

var TwitterProfileInfoURL = Xamarin.Forms.Application.Current.Resources["TwitterProfileInfoURL"].ToString(); var requestUrl = new OAuth1Request(

"GET",

new Uri(TwitterProfileInfoURL), null,

account);

var response = await requestUrl.GetResponseAsync();

var responseTwitter = JsonConvert.DeserializeObject<TwitterResponse>(response.GetResponseText());

var url = Xamarin.Forms.Application.Current.Resources["UrlAPI"].ToString();

var prefix = Xamarin.Forms.Application.Current.Resources["UrlPrefix"].ToString();

var controller = Xamarin.Forms.Application.Current.Resources["UrlUsersController"].ToString();

var apiService = new ApiService();

var token = await apiService.LoginTwitter(

url,

prefix,

$"{controller}/LoginTwitter",

responseTwitter);

return token;

}

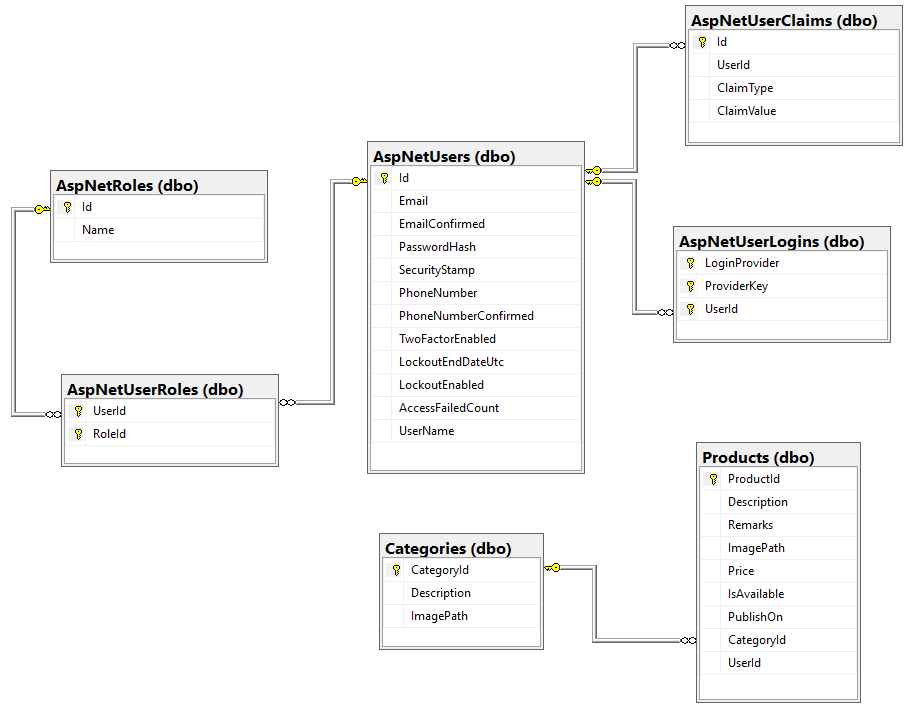
}

}

1. Ya podemos probar la integración con **Twitter**.

# Relaciones en los modelos

Vamos a crear este modelo entidad relación:



Para tal fin:

1. Primero creamos los nuevos literales.

Para Inglés:

<data name="Categories" xml:space="preserve">

<value>Categories</value>

</data>

<data name="Category" xml:space="preserve">

<value>Category</value>

</data>

<data name="CategoryPlaceholder" xml:space="preserve">

<value>Select the category...</value>

</data>

<data name="CategoryError" xml:space="preserve">

<value>You must select a category.</value>

</data>

Para Español:

<data name="Categories" xml:space="preserve">

<value>Categorías</value>

</data>

<data name="Category" xml:space="preserve">

<value>Categoría</value>

</data>

<data name="CategoryPlaceholder" xml:space="preserve">

<value>Seleccione la categoría...</value>

</data>

<data name="CategoryError" xml:space="preserve">

<value>Debes seleccionar una categoría.</value>

</data>

Para Italiano:

<data name="Categories" xml:space="preserve">

<value>Categorie</value>

</data>

<data name="Category" xml:space="preserve">

<value>Categorie</value>

</data>

<data name="CategoryPlaceholder" xml:space="preserve">

<value>Seleziona la categoria...</value>

</data>

<data name="CategoryError" xml:space="preserve">

<value>Devi selezionare una categoria.</value>

</data>

Para Portugués:

<data name="Categories" xml:space="preserve">

<value>Categorias</value>

</data>

<data name="Category" xml:space="preserve">

<value>Categoria</value>

</data>

<data name="CategoryPlaceholder" xml:space="preserve">

<value>Selecionar categoria..</value>

</data>

<data name="CategoryError" xml:space="preserve">

<value>Você deve selecionar uma categoria</value>

</data>

Y en Languages:

public static string Categories

{

get { return Resource.Categories; }

}

public static string Category

{

get { return Resource.Category; }

}

public static string CategoryPlaceholder

{

get { return Resource.CategoryPlaceholder; }

}

public static string CategoryError

{

get { return Resource.CategoryError; }

}

1. Creamos el nuevo modelo **Category**:

using Newtonsoft.Json;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

public class Category

{

[Key]

public int CategoryId { get; set; }

[Required]

[StringLength(50)]

public string Description { get; set; }

[Display(Name = "Image")]

public string ImagePath { get; set; }

[JsonIgnore]

public virtual ICollection<Product> Products { get; set; }

public string ImageFullPath

{

get

{

if (string.IsNullOrEmpty(this.ImagePath))

{

return "noproduct";

}

return $"https://salesbackend.azurewebsites.net{this.ImagePath.Substring(1)}";

}

}

}

1. Modificamos el modelo de productos.

[Required]

[StringLength(128)]

public string UserId { get; set; }

1. Creamos el CategoryView

using System.ComponentModel.DataAnnotations;

using System.Web;

using Common.Models;

public class CategoryView : Category

{

[Display(Name = "Image")]

public HttpPostedFileBase ImageFile { get; set; }

}

1. Creamos el controlador en el backend, publicamos y creamos algunas categorías (no olvidar crear la carpeta **Categories** en **Content**):

using System.Data.Entity;

using System.Linq;

using System.Net;

using System.Threading.Tasks;

using System.Web.Mvc;

using Common.Models;

using Helpers;

using Models;

[Authorize]

public class CategoriesController : Controller

{

private LocalDataContext db = new LocalDataContext();

public async Task<ActionResult> Index()

{

return View(await this.db.Categories.OrderBy(c => c.Description).ToListAsync());

}

public async Task<ActionResult> Details(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

var category = await this.db.Categories.FindAsync(id);

if (category == null)

{

return HttpNotFound();

}

return View(category);

}

public ActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Create(CategoryView view)

{

if (ModelState.IsValid)

{

var pic = string.Empty;

var folder = "~/Content/Categories";

if (view.ImageFile != null)

{

pic = FilesHelper.UploadPhoto(view.ImageFile, folder);

pic = $"{folder}/{pic}";

}

var category = this.ToCategory(view, pic);

this.db.Categories.Add(category);

await this.db.SaveChangesAsync();

return RedirectToAction("Index");

}

return View(view);

}

private Category ToCategory(CategoryView view, string pic)

{

return new Category

{

CategoryId = view.CategoryId,

Description = view.Description,

ImagePath = pic,

};

}

public async Task<ActionResult> Edit(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

var category = await this.db.Categories.FindAsync(id);

if (category == null)

{

return HttpNotFound();

}

var view = this.ToView(category);

return View(view);

}

private CategoryView ToView(Category category)

{

return new CategoryView

{

CategoryId = category.CategoryId,

Description = category.Description,

ImagePath = category.ImagePath,

};

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Edit(CategoryView view)

{

if (ModelState.IsValid)

{

var pic = view.ImagePath;

var folder = "~/Content/Categories";

if (view.ImageFile != null)

{

pic = FilesHelper.UploadPhoto(view.ImageFile, folder);

pic = $"{folder}/{pic}";

}

var category = this.ToCategory(view, pic);

this.db.Entry(category).State = EntityState.Modified;

await this.db.SaveChangesAsync();

return RedirectToAction("Index");

}

return View(view);

}

public async Task<ActionResult> Delete(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

var category = await this.db.Categories.FindAsync(id);

if (category == null)

{

return HttpNotFound();

}

return View(category);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<ActionResult> DeleteConfirmed(int id)

{

var category = await this.db.Categories.FindAsync(id);

this.db.Categories.Remove(category);

await this.db.SaveChangesAsync();

return RedirectToAction("Index");

}

protected override void Dispose(bool disposing)

{

if (disposing)

{

this.db.Dispose();

}

base.Dispose(disposing);

}

}

1. Modificamos las vistas de creación, edición e Index:

@model IEnumerable<Sales.Common.Models.Category>

@{

ViewBag.Title = "Index";

}

<h2>Categories</h2>

<p>

@Html.ActionLink("Create New", "Create", new { }, new { @class = "btn btn-primary" })

</p>

<table class="table">

<tr>

<th>

@Html.DisplayNameFor(model => model.Description)

</th>

<th>

@Html.DisplayNameFor(model => model.ImagePath)

</th>

<th></th>

</tr>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.Description)

</td>

<td>

@if (!string.IsNullOrEmpty(item.ImagePath))

{

<img src="@Url.Content(item.ImagePath)" alt="Image" style="**width**:**100px**;**height**:**150px**;**max-width**: **100%**; **height**: **auto**;" />

}

</td>

<td>

@Html.ActionLink("Edit", "Edit", new { id=item.CategoryId }, new { @class = "btn btn-warning" })

@Html.ActionLink("Details", "Details", new { id=item.CategoryId }, new { @class = "btn btn-info" })

@Html.ActionLink("Delete", "Delete", new { id=item.CategoryId }, new { @class = "btn btn-danger" })

</td>

</tr>

}

</table>

-----------------

@model Sales.Backend.Models.CategoryView

@{

ViewBag.Title = "Create";

}

<h2>Create</h2>

@using (Html.BeginForm("Create", "Categories", FormMethod.Post, new { enctype = "multipart/form-data" }))

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

<h4>Category</h4>

<hr />

@Html.ValidationSummary(true, "", new { @class = "text-danger" })

<div class="form-group">

@Html.LabelFor(model => model.Description, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Description, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Description, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.ImageFile, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

<span class="btn btn-default btn-file">

@Html.TextBoxFor(modelo => modelo.ImageFile, new { type = "file" })

</span>

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Create" class="btn btn-primary" />

@Html.ActionLink("Back to List", "Index", new { }, new { @class = "btn btn-success" })

</div>

</div>

</div>

}

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

-----------------

@model Sales.Backend.Models.CategoryView

@{

ViewBag.Title = "Edit";

}

<h2>Edit</h2>

@using (Html.BeginForm("Edit", "Categories", FormMethod.Post, new { enctype = "multipart/form-data" }))

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

<h4>Category</h4>

<hr />

@Html.ValidationSummary(true, "", new { @class = "text-danger" })

@Html.HiddenFor(model => model.CategoryId)

@Html.HiddenFor(model => model.ImagePath)

<div class="form-group">

@Html.LabelFor(model => model.Description, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Description, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Description, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.ImageFile, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

<span class="btn btn-default btn-file">

@Html.TextBoxFor(modelo => modelo.ImageFile, new { type = "file" })

</span>

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Save" class="btn btn-primary" />

@Html.ActionLink("Back to List", "Index", new { }, new { @class = "btn btn-success" })

</div>

</div>

</div>

}

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}

1. Creamos el controlador en el API y publicamos:

using System.Linq;

using System.Web.Http;

using Common.Models;

using Domain.Models;

[Authorize]

public class CategoriesController : ApiController

{

private DataContext db = new DataContext();

public IQueryable<Category> GetCategories()

{

return db.Categories.OrderBy(c => c.Description);

}

}

1. Creamos el nuevo controlador en el diccionario de recursos:

<x:String x:Key="UrlCategoriesController">/Categories</x:String>

1. Creamos la **CategoryItemViewModel**:

using System.Windows.Input;

using Common.Models;

using GalaSoft.MvvmLight.Command;

using Sales.Views;

public class CategoryItemViewModel : Category

{

#region Commands

public ICommand GotoCategoryCommand

{

get

{

return new RelayCommand(GotoCategory);

}

}

private async void GotoCategory()

{

MainViewModel.GetInstance().Products = new ProductsViewModel(this);

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

}

#endregion

}

1. Creamos la **CategoriesPage**:

<?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:i18n="clr-namespace:Sales.Helpers"

x:Class="Sales.Views.CategoriesPage"

BackgroundColor="Silver"

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

Title="{i18n:Translate Categories}">

<ContentPage.Content>

<StackLayout

BindingContext="{Binding Categories}"

Padding="5">

<SearchBar

BackgroundColor="Transparent"

HeightRequest="40"

Placeholder="{i18n:Translate Search}"

SearchCommand="{Binding SearchCommand}"

Text="{Binding Filter}">

</SearchBar>

<ListView

BackgroundColor="Transparent"

HasUnevenRows="True"

IsRefreshing="{Binding IsRefreshing}"

IsPullToRefreshEnabled="True"

ItemsSource="{Binding Categories}"

RefreshCommand="{Binding RefreshCommand}"

SeparatorVisibility="None">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Frame

CornerRadius="20"

HasShadow="true"

Margin="5">

<Grid>

<Grid.GestureRecognizers>

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

</Grid.GestureRecognizers>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<Image

Grid.Column="0"

Source="{Binding ImageFullPath}"

WidthRequest="100">

</Image>

<Label

Grid.Column="1"

FontAttributes="Bold"

FontSize="Large"

Text="{Binding Description}"

VerticalOptions="Center">

</Label>

<Image

Grid.Column="2"

Source="ic\_chevron\_right"

VerticalOptions="Center">

</Image>

</Grid>

</Frame>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage.Content>

</ContentPage>

1. Creamos la traducción para el literal **Categories**:
2. Creamos la **CategoriesViewModel**:

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

using System.Windows.Input;

using Common.Models;

using GalaSoft.MvvmLight.Command;

using Helpers;

using Services;

using Xamarin.Forms;

public class CategoriesViewModel : BaseViewModel

{

#region Attributes

private string filter;

private ApiService apiService;

private bool isRefreshing;

private ObservableCollection<CategoryItemViewModel> categories;

#endregion

#region Properties

public string Filter

{

get { return this.filter; }

set

{

this.filter = value;

this.RefreshList();

}

}

public List<Category> MyCategories { get; set; }

public ObservableCollection<CategoryItemViewModel> Categories

{

get { return this.categories; }

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

}

public bool IsRefreshing

{

get { return this.isRefreshing; }

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

}

#endregion

#region Constructors

public CategoriesViewModel()

{

this.apiService = new ApiService();

this.LoadCategories();

}

#endregion

#region Methods

private async void LoadCategories()

{

this.IsRefreshing = true;

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

if (!connection.IsSuccess)

{

this.IsRefreshing = false;

await Application.Current.MainPage.DisplayAlert(

Languages.Error,

connection.Message,

Languages.Accept);

return;

}

var url = Application.Current.Resources["UrlAPI"].ToString();

var prefix = Application.Current.Resources["UrlPrefix"].ToString();

var controller = Application.Current.Resources["UrlCategoriesController"].ToString();

var response = await this.apiService.GetList<Category>(url, prefix, controller, Settings.TokenType, Settings.AccessToken);

if (!response.IsSuccess)

{

this.IsRefreshing = false;

await Application.Current.MainPage.DisplayAlert(

Languages.Error,

response.Message,

Languages.Accept);

return;

}

this.MyCategories = (List<Category>)response.Result;

this.RefreshList();

this.IsRefreshing = false;

}

private void RefreshList()

{

if (string.IsNullOrEmpty(this.Filter))

{

var myListCategoriesItemViewModel = this.MyCategories.Select(c => new CategoryItemViewModel

{

CategoryId = c.CategoryId,

Description = c.Description,

ImagePath = c.ImagePath,

});

this.Categories = new ObservableCollection<CategoryItemViewModel>(

myListCategoriesItemViewModel.OrderBy(c => c.Description));

}

else

{

var myListCategoriesItemViewModel = this.MyCategories.Select(c => new CategoryItemViewModel

{

CategoryId = c.CategoryId,

Description = c.Description,

ImagePath = c.ImagePath,

}).Where(c => c.Description.ToLower().Contains(this.Filter.ToLower())).ToList();

this.Categories = new ObservableCollection<CategoryItemViewModel>(

myListCategoriesItemViewModel.OrderBy(c => c.Description));

}

}

#endregion

#region Commands

public ICommand SearchCommand

{

get

{

return new RelayCommand(RefreshList);

}

}

public ICommand RefreshCommand

{

get

{

return new RelayCommand(LoadCategories);

}

}

#endregion

}

1. Cambiamos la **MasterPage** para que el nuevo Home sea **Categories** y cambiamos el inicio para que empiece por **Categories**:

mainViewModel.Categories = new CategoriesViewModel();

1. Probamos el nuevo Home de nuestra App.
2. Modificamos el método de **GetProducts** con parámetro para que nos traiga los productos de una categoría:

[ResponseType(typeof(Product))]

public async Task<IHttpActionResult> GetProduct(int id)

{

var products = await this.db.Products.Where(p => p.CategoryId == id).ToListAsync();

return Ok(products);

}

1. Sobre carguemos el método en el API que obtiene la lista de productos para que nos de la lista de solo la categoría solicitada, y publiquemos el nuevo API:

public async Task<Response> GetList<T>(string urlBase, string prefix, string controller, int id, string tokenType, string accessToken)

{

try

{

var client = new HttpClient();

client.BaseAddress = new Uri(urlBase);

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(tokenType, accessToken);

var url = $"{prefix}{controller}/{id}";

var response = await client.GetAsync(url);

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

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = answer,

};

}

var list = JsonConvert.DeserializeObject<List<T>>(answer);

return new Response

{

IsSuccess = true,

Result = list,

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

1. Cambiemos el código que obtiene la lista de los productos por el siguiente (por el momento no vamos a utilizar base de datos local):

private async void LoadProducts()

{

this.IsRefreshing = true;

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

if (!connection.IsSuccess)

{

this.IsRefreshing = false;

await Application.Current.MainPage.DisplayAlert(Languages.Error, connection.Message, Languages.Accept);

return;

}

var answer = await this.LoadProductsFromAPI();

if (answer)

{

this.RefreshList();

}

this.IsRefreshing = false;

}

private async Task<bool> LoadProductsFromAPI()

{

var url = Application.Current.Resources["UrlAPI"].ToString();

var prefix = Application.Current.Resources["UrlPrefix"].ToString();

var controller = Application.Current.Resources["UrlProductsController"].ToString();

var response = await this.apiService.GetList<Product>(url, prefix, controller, this.Category.CategoryId, Settings.TokenType, Settings.AccessToken);

if (!response.IsSuccess)

{

return false;

}

this.MyProducts = (List<Product>)response.Result;

return true;

}

1. Probemos lo que llevamos.
2. Cambiamos la vista de adicionar producto para adicionar la categoría:

<Label

Grid.Column="0"

Grid.Row="2"

Text="{i18n:Translate Category}"

VerticalOptions="Center">

</Label>

<Picker

Grid.Column="1"

Grid.Row="2"

ItemDisplayBinding="{Binding Description}"

ItemsSource="{Binding Categories}"

SelectedItem="{Binding Category}"

Title="{i18n:Translate CategoryPlaceholder}">

</Picker>

<Label

Grid.Column="0"

Grid.Row="3"

Text="{i18n:Translate Remarks}"

VerticalOptions="Center">

</Label>

<Editor

Grid.Column="1"

Grid.Row="3"

Text="{Binding Remarks}"

VerticalOptions="FillAndExpand">

</Editor>

1. Modificamos la **AddProductViewModel** para que cargue las categorías:

En atributos y propiedades:

private ObservableCollection<Category> categories;

public List<Category> MyCategories { get; set; }

public Category Category { get; set; }

public ObservableCollection<Category> Categories

{

get { return this.categories; }

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

}

En el constructor:

this.LoadCategories();

En los métodos:

#region Methods

private async void LoadCategories()

{

this.IsRunning = true;

this.IsEnabled = false;

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

if (!connection.IsSuccess)

{

this.IsRunning = false;

this.IsEnabled = true;

await Application.Current.MainPage.DisplayAlert(Languages.Error, connection.Message, Languages.Accept);

return;

}

var answer = await this.LoadCategoriesFromAPI();

if (answer)

{

this.RefreshList();

}

this.IsRunning = false;

this.IsEnabled = true;

}

private void RefreshList()

{

this.Categories = new ObservableCollection<Category>(this.MyCategories.OrderBy(c => c.Description));

}

private async Task<bool> LoadCategoriesFromAPI()

{

var url = Application.Current.Resources["UrlAPI"].ToString();

var prefix = Application.Current.Resources["UrlPrefix"].ToString();

var controller = Application.Current.Resources["UrlCategoriesController"].ToString();

var response = await this.apiService.GetList<Category>(url, prefix, controller, Settings.TokenType, Settings.AccessToken);

if (!response.IsSuccess)

{

return false;

}

this.MyCategories = (List<Category>)response.Result;

return true;

}

#endregion

1. Probamos y verifiquemos que ya cargan nuestras categorías en el Picker.
2. Modificamos el método **Save** para que mande la categoría y el **UserId**:

if (this.Category == null)

{

await Application.Current.MainPage.DisplayAlert(

Languages.Error,

Languages.CategoryError,

Languages.Accept);

return;

}

…

var product = new Product

{

Description = this.Description,

Price = price,

Remarks = this.Remarks,

ImageArray = imageArray,

CategoryId = this.Category.CategoryId,

UserId = MainViewModel.GetInstance().UserASP.Id,

};

1. Probamos que podamos agregar productos ahora con categoría e identificador de usuario.
2. Ahora vamos hacer lo mismo pero para editar el producto. Primero organicemos la lista para que envíe los 2 nuevos atributos en el producto en la **ProductsViewModel** en el método **RefreshList**:

CategoryId = p.CategoryId,

UserId = p.UserId,

1. Modificamos la **EditProductPage** para que cargue las categorías:

<Label

Grid.Column="0"

Grid.Row="2"

Text="{i18n:Translate Category}"

VerticalOptions="Center">

</Label>

<Picker

Grid.Column="1"

Grid.Row="2"

ItemDisplayBinding="{Binding Description}"

ItemsSource="{Binding Categories}"

SelectedItem="{Binding Category}"

Title="{i18n:Translate CategoryPlaceholder}">

</Picker>

<Label

Grid.Column="0"

Grid.Row="3"

Text="{i18n:Translate Remarks}"

VerticalOptions="Center">

</Label>

<Editor

Grid.Column="1"

Grid.Row="3"

Text="{Binding Product.Remarks}"

VerticalOptions="FillAndExpand">

</Editor>

<Label

Grid.Column="0"

Grid.Row="4"

Text="{i18n:Translate IsAvailable}"

VerticalOptions="Center">

</Label>

<Switch

Grid.Column="1"

Grid.Row="4"

HorizontalOptions="End"

IsToggled="{Binding Product.IsAvailable}">

</Switch>

1. Modificamos la **ProductsViewModel** para que llene la lista de productos con los nuevos atributos:

public void RefreshList()

{

if (string.IsNullOrEmpty(this.Filter))

{

var myListProductItemViewModel = this.MyProducts.Select(p => new ProductItemViewModel

{

Description = p.Description,

ImageArray = p.ImageArray,

ImagePath = p.ImagePath,

IsAvailable = p.IsAvailable,

Price = p.Price,

ProductId = p.ProductId,

PublishOn = p.PublishOn,

Remarks = p.Remarks,

CategoryId = p.CategoryId,

UserId = p.UserId,

});

this.Products = new ObservableCollection<ProductItemViewModel>(

myListProductItemViewModel.OrderBy(p => p.Description));

}

else

{

var myListProductItemViewModel = this.MyProducts.Select(p => new ProductItemViewModel

{

Description = p.Description,

ImageArray = p.ImageArray,

ImagePath = p.ImagePath,

IsAvailable = p.IsAvailable,

Price = p.Price,

ProductId = p.ProductId,

PublishOn = p.PublishOn,

Remarks = p.Remarks,

CategoryId = p.CategoryId,

UserId = p.UserId,

}).Where(p => p.Description.ToLower().Contains(this.Filter.ToLower())).ToList();

this.Products = new ObservableCollection<ProductItemViewModel>(

myListProductItemViewModel.OrderBy(p => p.Description));

}

}

1. Modificamos la **EditProductViewModel** para que cargue las categorías:

En atributos y propiedades:

private ObservableCollection<Category> categories;

private Category category;

public List<Category> MyCategories { get; set; }

public Category Category

{

get { return this.category; }

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

}

public ObservableCollection<Category> Categories

{

get { return this.categories; }

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

}

En el constructor:

this.LoadCategories();

En los métodos:

#region Methods

private async void LoadCategories()

{

this.IsRunning = true;

this.IsEnabled = false;

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

if (!connection.IsSuccess)

{

this.IsRunning = false;

this.IsEnabled = true;

await Application.Current.MainPage.DisplayAlert(Languages.Error, connection.Message, Languages.Accept);

return;

}

var answer = await this.LoadCategoriesFromAPI();

if (answer)

{

this.RefreshList();

}

this.Category = this.MyCategories.FirstOrDefault(c => c.CategoryId == this.Product.CategoryId);

this.IsRunning = false;

this.IsEnabled = true;

}

private void RefreshList()

{

this.Categories = new ObservableCollection<Category>(this.MyCategories.OrderBy(c => c.Description));

}

private async Task<bool> LoadCategoriesFromAPI()

{

var url = Application.Current.Resources["UrlAPI"].ToString();

var prefix = Application.Current.Resources["UrlPrefix"].ToString();

var controller = Application.Current.Resources["UrlCategoriesController"].ToString();

var response = await this.apiService.GetList<Category>(url, prefix, controller, Settings.TokenType, Settings.AccessToken);

if (!response.IsSuccess)

{

return false;

}

this.MyCategories = (List<Category>)response.Result;

return true;

}

#endregion

1. Probemos que se nos cargue la categoría.
2. Modificamos el método **Save** para que tome la modificación de categoría:

if (this.Category == null)

{

await Application.Current.MainPage.DisplayAlert(

Languages.Error,

Languages.CategoryError,

Languages.Accept);

return;

}

…

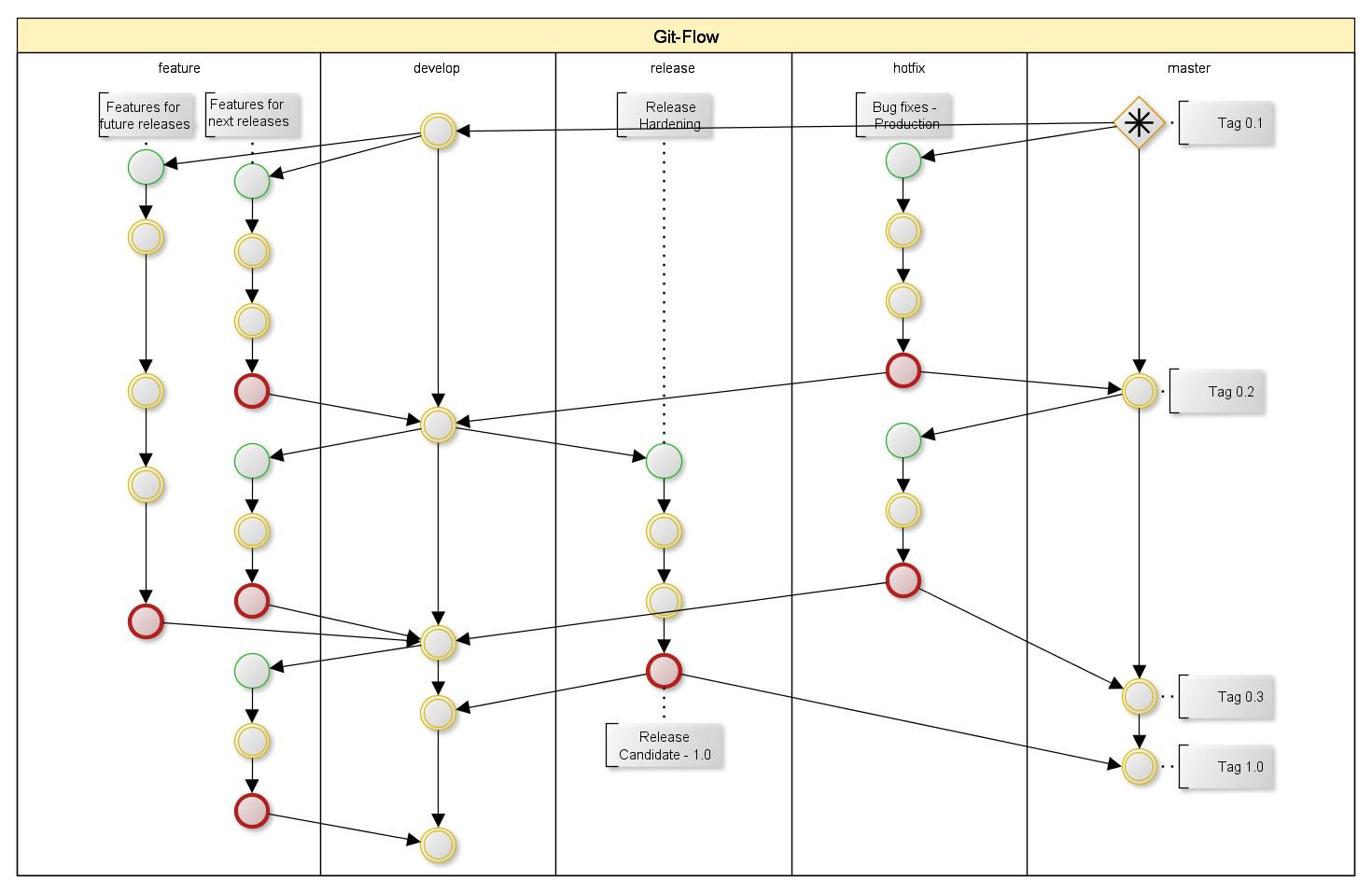
this.Product.CategoryId = this.Category.CategoryId;

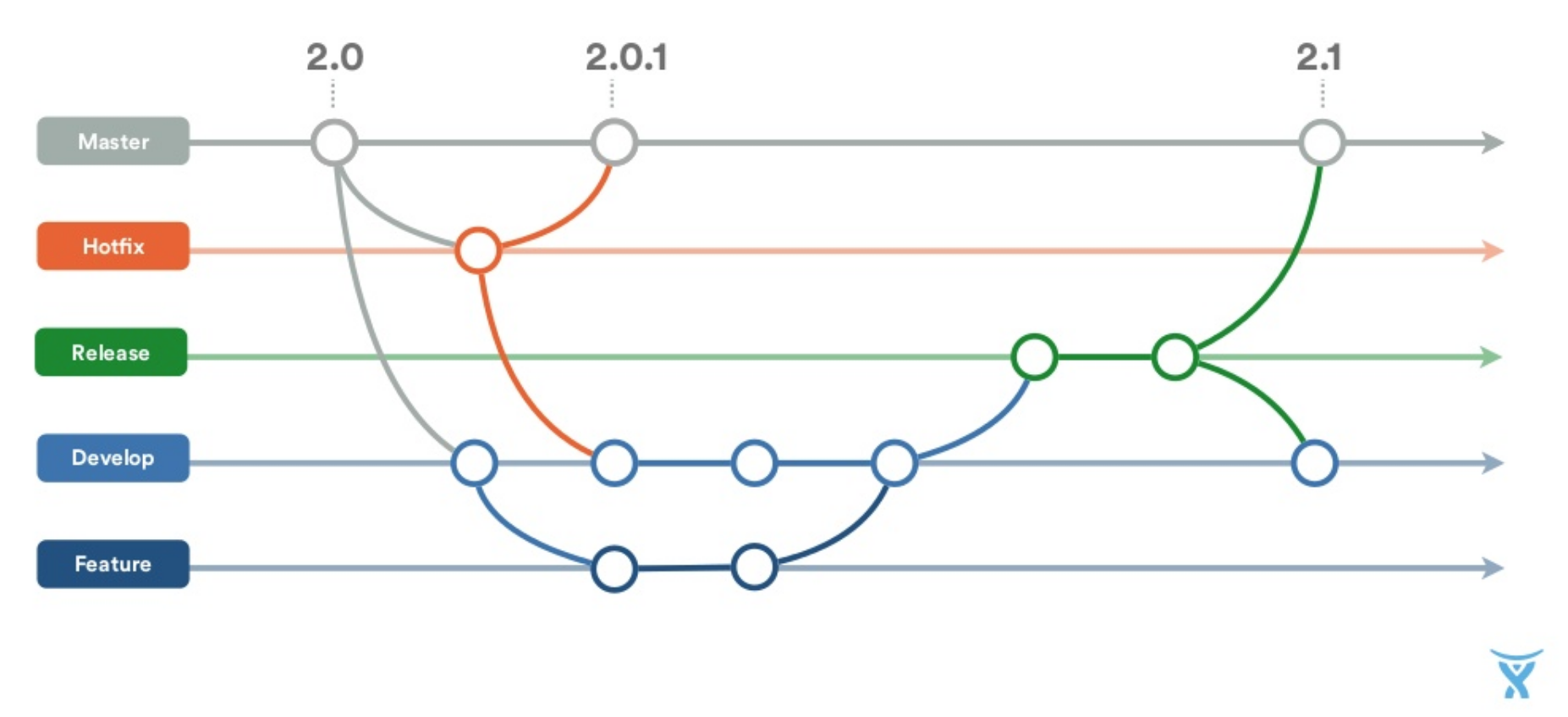
1. Probar la modificación categoría.

# GitFlow



Recomiendo leer este artículo: <http://aprendegit.com/que-es-git-flow/>









Representación gráfica de lo que es un merge: <https://www.youtube.com/watch?v=AqocDsE_32c>

# Mapas y geolocalización

Para poder pintar un mapa con puntos en nuestra App que representan la ubicación física de los artículos,

1. Primero vamos a agregar 2 atributos al modelo de productos.

public double Latitude { get; set; }

public double Longitude { get; set; }

Publicamos un nuevo API con este cambio.

1. Creamos la **MapPage** vacía y hacemos el llamado desde el 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="Sales.Views.MapPage"

Title="Mapas">

<ContentPage.Content>

<StackLayout>

<Label

FontAttributes="Bold"

FontSize="Large"

Text="Aca va el mapa"

TextColor="Black">

</Label>

</StackLayout>

</ContentPage.Content>

</ContentPage>

Para llamarla desde el menú agregamos estas líneas en la MenuItemViewModel:

if (this.PageName == "LoginPage")

{

Settings.AccessToken = string.Empty;

Settings.TokenType = string.Empty;

Settings.IsRemembered = false;

MainViewModel.GetInstance().Login = new LoginViewModel();

Application.Current.MainPage = new NavigationPage(new LoginPage());

}

else if (this.PageName == "AboutPage")

{

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

}

Probemos y nos daremos cuenta que no se oculta el menú, para que este se oculte debemos adicionar esta línea en el **MasterPage**:

protected override void OnAppearing()

{

base.OnAppearing();

App.Navigator = Navigator;

App.Master = this;

}

Y crear la propiedad en el **App**:

#region Properties

public static NavigationPage Navigator { get; internal set; }

public static MasterPage Master { get; internal set; }

#endregion

Y modificamos el llamado a la **MapPage** por:

if (this.PageName == "LoginPage")

{

Settings.AccessToken = string.Empty;

Settings.TokenType = string.Empty;

Settings.IsRemembered = false;

MainViewModel.GetInstance().Login = new LoginViewModel();

Application.Current.MainPage = new NavigationPage(new LoginPage());

}

else if (this.PageName == "AboutPage")

{

App.Master.IsPresented = false;

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

}

Ahora si probamos ya si muestra la pantalla del mapa de forma correcta.

1. Ahora actualizamos el nuget de **Xamarin.Forms** y adicionamos el nuget de **Xamarin.Forms.Maps** en toda la solución.
2. En este punto debemos hacer un **Clean** y luego correr la solución para evitar un error.
3. Debemos de agregar este configuración adicional en Android, en el

**MainActivity** colocamos esta inicialización:

protected override void OnCreate(Bundle savedInstanceState)

{

TabLayoutResource = Resource.Layout.Tabbar;

ToolbarResource = Resource.Layout.Toolbar;

base.OnCreate(savedInstanceState);

CrossCurrentActivity.Current.Init(this, savedInstanceState);

ImageCircleRenderer.Init();

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

global::Xamarin.FormsMaps.Init(this, savedInstanceState);

LoadApplication(new App());

}

Y adicionamos estas línea en el **AndroidManifest**:

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

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

android:versionCode="1"

android:versionName="1.0"

package="com.companyname.Sales"

android:installLocation="auto">

<uses-sdk android:minSdkVersion="21" android:targetSdkVersion="27" />

<application android:label="Sales" android:icon="@drawable/ic\_launcher">

<provider android:name="android.support.v4.content.FileProvider"

android:authorities="${applicationId}.fileprovider"

android:exported="false"

android:grantUriPermissions="true">

<meta-data android:name="android.support.FILE\_PROVIDER\_PATHS"

android:resource="@xml/file\_paths"/>

</provider>

<meta-data android:name="com.google.android.maps.v2.API\_KEY"

android:value="AIzaSyAtxvXVhbzV9OTwZh8UxVsW2A58WYf-Btc" />

</application>

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

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

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

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

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

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

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

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

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

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

</manifest>

Puedes obtener tu propio key de mapas de: <https://developers.google.com/maps/?hl=es-419>.

1. De forma similar a Android, debemos de agregar este configuración adicional en iOS, en el

**AppDelegate** colocamos esta inicialización:

public override bool FinishedLaunching(UIApplication app, NSDictionary options)

{

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

ImageCircleRenderer.Init();

Xamarin.FormsMaps.Init();

LoadApplication(new App());

return base.FinishedLaunching(app, options);

}

Y agregamos esta configuración al **info.plist**:

<key>NSLocationAlwaysUsageDescription</key>

<string>Can we use your location at all times?</string>

<key>NSLocationWhenInUseUsageDescription</key>

<string>Can we use your location when your app is being used?</string>

<key>NSLocationAlwaysAndWhenInUseUsageDescription</key>

<string>Can we use your location at all times?</string>

1. Cambiamos la **MapPage** por lo siguiente:

<?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="Sales.Views.MapPage"

Title="Mapas">

<ContentPage.Content>

<StackLayout>

<maps:Map

x:Name="MyMap"

HorizontalOptions="FillAndExpand"

IsShowingUser="true"

MapType="Street"

VerticalOptions="FillAndExpand">

</maps:Map>

<Slider

x:Name="MySlider"

Value="0.8">

</Slider>

</StackLayout>

</ContentPage.Content>

</ContentPage>

Y probamos inicialmente en iOS.

1. Ahora movamos el mapa para nuestra ubicación actual, para ese fin primero agregamos el nugget de **Xam.Plugin.Geolocator** en toda la solución.
2. Luego le agregamos el método cuando cambie el Slider. Y cambiamos el code behind de la página del Map por lo siguiente:

namespace Sales.Views

{

using System;

using Plugin.Geolocator;

using Xamarin.Forms;

using Xamarin.Forms.Maps;

public partial class MapPage : ContentPage

{

public MapPage()

{

InitializeComponent();

}

protected override void OnAppearing()

{

base.OnAppearing();

this.Locator();

}

private async void Locator()

{

var locator = CrossGeolocator.Current;

locator.DesiredAccuracy = 50;

var location = await locator.GetPositionAsync();

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

this.MyMap.MoveToRegion(MapSpan.FromCenterAndRadius(position, Distance.FromKilometers(1)));

}

private void Handle\_ValueChanged(object sender, Xamarin.Forms.ValueChangedEventArgs e)

{

var zoomLevel = double.Parse(e.NewValue.ToString()) \* 18.0;

var latlongdegrees = 360 / (Math.Pow(2, zoomLevel));

this.MyMap.MoveToRegion(new MapSpan(this.MyMap.VisibleRegion.Center, latlongdegrees, latlongdegrees));

}

}

}

Y probamos inicialmente en iOS. Y luego tratas de probar en Android. Como en Android saca error, vamos a quitar el parámetro: IsShowingUser="true" de la vista y vamos a modificar el code behind del **MapPage** por este:

private async void Locator()

{

var locator = CrossGeolocator.Current;

locator.DesiredAccuracy = 50;

var location = await locator.GetPositionAsync();

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

this.MyMap.MoveToRegion(MapSpan.FromCenterAndRadius(position, Distance.FromKilometers(1)));

try

{

this.MyMap.IsShowingUser = true;

}

catch (Exception ex)

{

ex.ToString();

}

}

Luego probamos de nuevo.

1. Ahora vamos a grabar la localización del producto al momento de crearlo, para luego mostrarlo en el mapa. Para tal fin, modifiquemos el **AddProductViewModel** por lo siguiente:

byte[] imageArray = null;

if (this.file != null)

{

imageArray = FilesHelper.ReadFully(this.file.GetStream());

}

var location = await this.GetLocation();

var product = new Product

{

Description = this.Description,

Price = price,

Remarks = this.Remarks,

ImageArray = imageArray,

CategoryId = this.Category.CategoryId,

UserId = MainViewModel.GetInstance().UserASP.Id,

Latitude = location == null ? 0 : location.Latitude,

Longitude = location == null ? 0 : location.Longitude,

};

Y obviamente creamos el nuevo método:

private async Task<Position> GetLocation()

{

var locator = CrossGeolocator.Current;

locator.DesiredAccuracy = 50;

var location = await locator.GetPositionAsync();

return location;

}

Y probamos que estemos grabando la ubicación de los productos.

1. Modificamos la **MapPage** para que obtenga y muestre los puntos en el mapa:

private async void Locator()

{

var locator = CrossGeolocator.Current;

locator.DesiredAccuracy = 50;

var location = await locator.GetPositionAsync();

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

this.MyMap.MoveToRegion(MapSpan.FromCenterAndRadius(position, Distance.FromKilometers(1)));

try

{

this.MyMap.IsShowingUser = true;

}

catch (Exception ex)

{

ex.ToString();

}

var pins = await this.GetPins();

this.ShowPins(pins);

}

private void ShowPins(List<Pin> pins)

{

foreach (var pin in pins)

{

this.MyMap.Pins.Add(pin);

}

}

private async Task<List<Pin>> GetPins()

{

var pins = new List<Pin>();

var apiService = new ApiService();

var url = Application.Current.Resources["UrlAPI"].ToString();

var prefix = Application.Current.Resources["UrlPrefix"].ToString();

var controller = Application.Current.Resources["UrlProductsController"].ToString();

var response = await apiService.GetList<Product>(url, prefix, controller, Settings.TokenType, Settings.AccessToken);

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

foreach (var product in products.Where(p => p.Latitude != 0 && p.Longitude != 0).ToList())

{

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

pins.Add(new Pin

{

Address = product.Remarks,

Label = product.Description,

Position = position,

Type = PinType.Place,

});

}

return pins;

}

Y probamos, acá es cuando decimos Taran!!!!!.

# Notificaciones Push

Vamos a implementar una opción que diga, quiero contactar al vendedor y cuando este ocurra, se le enviará una notificación Push al vendedor del producto para para que se ponga en contacto con el potencial comprador. Primero para entender el concepto vamos a enviar una notificación genérica a todos los usuarios de la App.

Ver documento: <https://docs.google.com/document/d/1ws0yue7sk9IKCwgOBwCBie7vhU4NiTXTNx_kTz6BCfc/edit>

# Publicación en tiendas

Por último vamos hacer una publicación en tiendas de nuestra App.