1.- Crear las carpetas Models, MainViewModel y Views

1.1.- Crear una ViewModel principal, MainViewModel, desde donde arrancará el proyecto

1.2.- Crear la BaseViewModel para implementar el INotifyPropertyChanged:

public abstract class ViewModelBase: INotifyPropertyChanged

{

#region INotifyPropertyChanged implementation

public event PropertyChangedEventHandler PropertyChanged;

#endregion

protected void Notify(string propertyName)

{

if (this.PropertyChanged != null)

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

}

}

2.- Crear la infraestructura con el InstanceLocator

3.- Crear la clase para el servicio de mensajería. Para ello, se crea una carpeta Services en la raíz del proyecto, y dentro de la carpeta se crea la clase MessageService.cs la cual va a contener el siguiente código:

public class MessageService

{

public async Task SendMessage(string title, string message)

{

await MyStock.App.Current.MainPage.DisplayAlert(title, message, "Accept");

}

}

4.- Crear las View

4.1.- Se crea la primera View, LoginView, esta será la pantalla de inicio de la app. Esto hace que, en el constructor de la MainViewModel se cree una instancia de la LoginView, ya que siendo la MainViewModel, la principal, esta tiene que instanciar la vista de arranque del proyecto.

4.2.- Dentro del LoginView.xaml se debe agregar el bindingcontext principal hacia la MainViewModel, y luego, dentro del cuerpo del stacklayout se incorpora el bindingcontext hacia la LoginViewModel, cuyo identificador será Login.

5.- Indicar en el code behind de App.axml el MainPage apuntando a una NavigationPage que arranca por LoginView

6.- Desarrollar el código de la MainViewModel para implementar además el patrón Singlenton

public class MainViewModel

{

public TokenResponse tokenResponse { get; set; }\*

public LoginViewModel Login { get; set; }

public CategoriesViewModel Categories { get; set; }

public MainViewModel()

{

Login = new LoginViewModel();

}

//Singlenton

static MainViewModel instance;

public static MainViewModel GetIntance()

{

if (instance == null)

return new MainViewModel();

else

return instance;

}

}

7.- Desarrollar la estructura de la LoginView.xam

Register

Login with Facebook

Login

Forgot your password?

Remember me

Enter your email….

Password

Enter your password….

Email

PRODUCTS

Login

El código para hacer la anterior interfaz en el LoginView.xaml es:

<?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="MyStock.Views.LoginView"

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

Title="Login">

<ContentPage.Content>

<StackLayout

BindingContext="{Binding Login}"

Padding="8"

Spacing="16">

<Label Text="Products"

FontSize="Large"

FontAttributes="Bold"

HorizontalOptions="Center" />

<Grid

Margin="50,10">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="2\*"/>

</Grid.ColumnDefinitions>

<Label Text="Email"

Grid.Row="0"

Grid.Column="0"/>

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

Placeholder="Enter your email..."

Grid.Row="0"

Grid.Column="1"

Keyboard="Email"/>

<Label Text="Password"

Grid.Row="1"

Grid.Column="0"/>

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

Placeholder="Enter your password..."

Grid.Row="1"

Grid.Column="1"

IsPassword="True"/>

</Grid>

<StackLayout Orientation="Horizontal">

<Label Text="Remember me"

VerticalOptions="Center"/>

<Switch

IsToggled="{Binding IsToggled, Mode=TwoWay}"/>

</StackLayout>

<Label Text="Forgot your password?"

TextColor="Blue"

HorizontalOptions="Center">

<Label.GestureRecognizers>

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

</Label.GestureRecognizers>

</Label>

<ActivityIndicator

IsRunningLabel="{Binding IsRunning, Mode=TwoWay}"/>

<StackLayout Margin="50,0">

<Button Text="Login"

BackgroundColor="Navy"

Style="{StaticResource GeneralButton}"

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

Command="{Binding LoginCommand}"/>

<Button Text="Register"

BackgroundColor="Blue"

Style="{StaticResource GeneralButton}"

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

Command="{Binding LoginWithFacebookCommand}"/>

<Button Text="Register"

BackgroundColor="Blue"

Style="{StaticResource GeneralButton}"

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

Command="{Binding RegisterNewUserCommand}"/>

</StackLayout>

</StackLayout>

</ContentPage.Content>

</ContentPage>

8.- Instalar a través del Manager NuGet Package el MvvmLightLibs, la cual va a permitir asociar un binding sobre un command.

9.- Desarrollar el código de la LoginViewModel para procesar las acciones que se den desde la LoginView y verificar la existencia del usuario desde una Web API. Se debe agregar el paquete Microsoft.Net.Http V.2.2.29 (incluyendo BCL y Build) y Newtonsoft Json V10. Adicionalmente vamos a requerir el plugin Xam.Plugin.Connectivity V 3.1.1 para verificar la conectividad a Internet. Este último plugin debe agregarse también dentro del proyecto .Android. El código quedaría así:

using MyStock.Services;

using System;

using System.Collections.Generic;

using System.Text;

using System.Windows.Input;

namespace MyStock.ViewModels

{

public class LoginViewModel : ViewModelBase

{

string email;

public string Email

{

get

{

return email;

}

set

{

email = value;

this.Notify("Email");

}

}

string password;

public string Password

{

get

{

return password;

}

set

{

password = value;

this.Notify("Password");

}

}

bool istoggled;

public bool IsToggled

{

get

{

return istoggled;

}

set

{

istoggled = value;

this.Notify("IsToggled");

}

}

bool isrunning;

public bool IsRunning

{

get

{

return isrunning;

}

set

{

isrunning = value;

this.Notify("IsRunning");

}

}

bool isenabled;

public bool IsEnabled

{

get

{

return isenabled;

}

set

{

isenabled = value;

this.Notify("IsEnabled");

}

}

MessageService messageService;

ApiService apiService; \*\*\*

public LoginViewModel()

{

messageService = new MessageService();

IsEnabled = true;

IsToggled = true;

}

public ICommand LoginCommand

{

get;

set;

}

public ICommand LoginWithFacebookCommand

{

get;

set;

}

public ICommand RegisterNewUserCommand

{

get;

set;

}

async void Login()

{

if (string.IsNullOrEmpty(Email))

{

await messageService.SendMessage("Error", "You must enter an email");

return;

}

if (string.IsNullOrEmpty(Password))

{

await messageService.SendMessage("Error", "You must enter an password");

return;

}

IsRunning = true; \*\*\*

IsEnabled = false; \*\*\*

\*\*\* var connection = await apiService.CheckConnection();

if (connection.IsSuccess)

{

//\*\*\*nuevas líneas de código de acuerdo con el pto 10

await messageService.SendMessage("Error", connection.Message);

return;

}

var response = await apiService.GetToken("http://productszuluapi.azurewebsites.net", Email, Password);

if (response == null)

{

IsRunning = false;

IsEnabled = true;

await messageService.SendMessage("Error", "The service is not available, please try later.");

Password = null;

return;

}

if (string.IsNullOrEmpty(response.AccessToken))

{

IsRunning = false;

IsEnabled = true;

await messageService.SendMessage("Error", response.ErrorDescription);

Password = null;

return;

}

IsRunning = false;

IsEnabled = true;

var mainViewModel = MainViewModel.GetIntance();

mainViewModel.Categories = new CategoriesViewModel();

mainViewModel.tokenResponse = response;

await navigationService.NavigateTo("CategoriesView");

}

}

}

10.- Crear el servicio API. Para ello creamos una clase ApiService dentro de la carpeta Services. Luego creamos una clase Response dentro de la carpeta Models la cual va a contener el siguiente código:

public class Response

{

public bool IsSuccess { get; set; } //Propiedad para verificar si hay conexión

public string Message { get; set; } //Enviar mensaje dependiendo de si conecto o no

public Object Result { get; set; } //Obtener la respuesta http

}

10.1. Asimismo, se va a crear dentro de la carpeta Models una clase llamada TokenResponse con la finalidad de contener los parámetros del token que se va a generar para validar el acceso al usuario. El código de esta clase queda así:

public class TokenResponse

{

[JsonProperty(PropertyName ="access\_token")] //El uso de esta propiedad de Json permite tomar el nombre de la propiedad tal como es generada y mapearla con un nombre mas adecuado que se le de a dicha propiedad dentro de la clase.

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; }

}

10.2 Se desarrollar el código de la clase ApiService quedando como sigue:

public class ApiService

{

public async Task<Response> CheckConnection()

{

if (!CrossConnectivity.Current.IsConnected)

{

return new Response

{

IsSuccess = false,

Message = "Please check your Internet settings.",

};

}

var isReachable = await CrossConnectivity.Current.IsRemoteReachable("google.com");

if (!isReachable)

{

return new Response

{

IsSuccess = false,

Message = "Please check your Internet connection.",

};

}

return new Response

{

IsSuccess = true,

Message = "OK",

};

}

public async Task<TokenResponse> GetToken(string urlBase, string username, string password) //Este es el método que obtiene el token

{

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;

}

}

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

{

try

{

var client = new HttpClient();

client.DefaultRequestHeaders.Authorization =

new AuthenticationHeaderValue(tokenType, accessToken);

client.BaseAddress = new Uri(urlBase);

var url = string.Format(

"{0}{1}/{2}",

servicePrefix,

controller,

id);

var response = await client.GetAsync(url);

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = response.StatusCode.ToString(),

};

}

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

var model = JsonConvert.DeserializeObject<T>(result);

return new Response

{

IsSuccess = true,

Message = "Ok",

Result = model,

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

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

{

try

{

var client = new HttpClient();

client.DefaultRequestHeaders.Authorization =

new AuthenticationHeaderValue(tokenType, accessToken);

client.BaseAddress = new Uri(urlBase);

var url = string.Format("{0}{1}", servicePrefix, controller);

var response = await client.GetAsync(url);

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

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = result,

};

}

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

return new Response

{

IsSuccess = true,

Message = "Ok",

Result = list,

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

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

{

try

{

var client = new HttpClient();

client.DefaultRequestHeaders.Authorization =

new AuthenticationHeaderValue(tokenType, accessToken);

client.BaseAddress = new Uri(urlBase);

var url = string.Format(

"{0}{1}/{2}",

servicePrefix,

controller,

id);

var response = await client.GetAsync(url);

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = response.StatusCode.ToString(),

};

}

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

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

return new Response

{

IsSuccess = true,

Message = "Ok",

Result = list,

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

public async Task<Response> Post<T>(string urlBase, string servicePrefix, string controller, string tokenType, string accessToken, T model)

{

try

{

var request = JsonConvert.SerializeObject(model);

var content = new StringContent(

request, Encoding.UTF8,

"application/json");

var client = new HttpClient();

client.DefaultRequestHeaders.Authorization =

new AuthenticationHeaderValue(tokenType, accessToken);

client.BaseAddress = new Uri(urlBase);

var url = string.Format("{0}{1}", servicePrefix, controller);

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

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

if (!response.IsSuccessStatusCode)

{

var error = JsonConvert.DeserializeObject<Response>(result);

error.IsSuccess = false;

return error;

}

var newRecord = JsonConvert.DeserializeObject<T>(result);

return new Response

{

IsSuccess = true,

Message = "Record added OK",

Result = newRecord,

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

public async Task<Response> Post<T>(string urlBase, string servicePrefix, string controller, T model)

{

try

{

var request = JsonConvert.SerializeObject(model);

var content = new StringContent(

request,

Encoding.UTF8,

"application/json");

var client = new HttpClient();

client.BaseAddress = new Uri(urlBase);

var url = string.Format("{0}{1}", servicePrefix, controller);

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

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = response.StatusCode.ToString(),

};

}

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

var newRecord = JsonConvert.DeserializeObject<T>(result);

return new Response

{

IsSuccess = true,

Message = "Record added OK",

Result = newRecord,

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

public async Task<Response> Put<T>(string urlBase, string servicePrefix, string controller, string tokenType, string accessToken, T model)

{

try

{

var request = JsonConvert.SerializeObject(model);

var content = new StringContent(

request,

Encoding.UTF8, "application/json");

var client = new HttpClient();

client.DefaultRequestHeaders.Authorization =

new AuthenticationHeaderValue(tokenType, accessToken);

client.BaseAddress = new Uri(urlBase);

var url = string.Format(

"{0}{1}/{2}",

servicePrefix,

controller,

model.GetHashCode());

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

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

if (!response.IsSuccessStatusCode)

{

var error = JsonConvert.DeserializeObject<Response>(result);

error.IsSuccess = false;

return error;

}

var newRecord = JsonConvert.DeserializeObject<T>(result);

return new Response

{

IsSuccess = true,

Result = newRecord,

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

public async Task<Response> Delete<T>(string urlBase, string servicePrefix, string controller, string tokenType, string accessToken, T model)

{

try

{

var client = new HttpClient();

client.BaseAddress = new Uri(urlBase);

client.DefaultRequestHeaders.Authorization =

new AuthenticationHeaderValue(tokenType, accessToken);

var url = string.Format(

"{0}{1}/{2}",

servicePrefix,

controller,

model.GetHashCode());

var response = await client.DeleteAsync(url);

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

if (!response.IsSuccessStatusCode)

{

var error = JsonConvert.DeserializeObject<Response>(result);

error.IsSuccess = false;

return error;

}

return new Response

{

IsSuccess = true,

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

}

11.- Nos regresamos a la LoginViewModel, para desarrollar el código una vez que se llama a la Web API\*\*\*

12.- Creamos la siguiente View. En este caso la CategoriesView, la cual nos va a listar las categorias, mediante el consumo del Web API desarrollado: <http://xxx/api/Categories>.

12.1.- Como para poder listar las categorias se requiere del token del usuario previamente autenticado, entonces se debe manejar el token que se generó al momento del login. En este sentido, este token debe conservarse durante toda la sesión del usuario, para ello, se va a crear una propiedad del tipo TokenResponse (modelo construido) dentro de la MainViewModel\*

12.2.- Se van a listar todas las categorias que devuelve el Web API, pero como viene en formato JSON, hay que crear un modelo que en este caso va a contener solo dos atributos CategoryId y Description, para poder mapear los datos dentro de la ListView. Se creará una clase dentro de la carpeta Models del proyecto General llamada Category, cuya estructura queda así:

public class Category

{

public int CategoryId { get; set; }

public string Description { get; set; }

}

12.3.- Se crea la CategoriesViewModel con el siguiente código:

El objetivo es traer la lista de categorias, (esto lo hace el controlador actual), pero adicionalmente, se requiere que al hacer clic sobre una categoría, devuelva la lista de productos que se encuentran en dicha categoría. Para ello, se debe modificar el controlador de la API. \*\*

public class CategoriesViewModel

{

public ObservableCollection<Category> Categories { get; set; }

public CategoriesViewModel()

{

LoadCategories();

}

void LoadCategories()

{

}

}

\*\*Controlador modificado:

Primero se debe crear dentro de la carpeta Models del proyecto API, una clase llamada CategoryResponse, y vamos a la clase Category que se encuentra dentro del proyecto Domain, y copiamos todas las propiedades y la pegamos en la nueva clase creada, quedando la siguiente estructura:

public class CategoryResponse

{

public int CategoryId { get; set; }

public string Description { get; set; }

public List<ProductResponse> Productos { get; set; }

}

Lo mismo se hace con los productos, se crea una clase llamada ProductResponse dentro de la carpeta Models del proyecto API, y se despliega la clase Product del proyecto Domain para copiar las propiedades y solo dejar las que serán mostradas en la vista, quedando así:

public class ProductResponse

{

public int ProductId { get; set; }

public string Description { get; set; }

public decimal Price { get; set; }

public bool IsActive { get; set; }

public string Remarks { get; set; }

public DateTime LastPurchase { get; set; }

public string Image { get; set; }

public double Stock { get; set; }

}

Estas dos nuevas clases se realizaron para ejecutar un única acción desde el controlador API de las categorias, para que de una vez, liste los productos que están asociados a una misma categoría. Se recuerda que las clases del proyecto DOMAIN, solo se construyeron y están allí para el EntityFramework y poder manejar las relaciones entre las tablas.

Ahora con estas nuevas clases, nos regresamos al controlador del proyecto API que devuelve las categorías para hacer las modificaciones correspondientes, quedando así:

public async Task<IHttpActionResult> GetCategories()

{

var categories = await db.Categories.ToListAsync();

var categoriesResponse = new List<CategoryResponse>();

foreach (var item in categories)

{

var productsResponse = new List<ProductResponse>();

foreach (var itemProd in item.Productos)

{

productsResponse.Add(new ProductResponse

{

ProductId = itemProd.ProductId,

Description = itemProd.Description,

Image = itemProd.Image,

IsActive = itemProd.IsActive,

LastPurchase = itemProd.LastPurchase,

Price = itemProd.Price,

Stock = itemProd.Stock,

Remarks = itemProd.Remarks,

});

}

categoriesResponse.Add(new CategoryResponse

{

CategoryId = item.CategoryId,

Description = item.Description,

Productos = productsResponse,

});

}

return Ok(categoriesResponse);

}

Realizados estos cambios, se corre el proyecto de Web API y se prueba en POSTMAN el consumo del servicio para obtener la lista de productos por categorías. En este punto, se recuerda generar primero dentro del Postman, el Token para un usuario registrado y luego obtener el GET del servicio, usando el Token generado.

Una vez verificado el correcto funcionamiento se publica nuevamente el proyecto del WEB API.

12.4.- En la Web API se están listando todos los productos pertenecientes a una misma categoria, pero como viene en formato JSON, hay que crear un modelo que va a contener los atributos CategoryId y Description, para poder mapear los datos dentro de la ListView. Para ello, se creará una clase dentro de la carpeta Models del proyecto General llamada Product (así como se hizo en el apartado 12.2 que se creó la clase Category), cuya estructura queda así:

public class Product

{

public int ProductId { get; set; }

public string Description { get; set; }

public decimal Price { get; set; }

public bool IsActive { get; set; }

public string Remarks { get; set; }

public DateTime LastPurchase { get; set; }

public string Image { get; set; }

public double Stock { get; set; }

}

Y luego, para que el modelo quede estructurado con la relación de muchos productos dentro de una misma categoría, se debe agregar la siguiente línea de código a la clase Category (creada en el apartado 12.2):

public List<Product> Productos { get; set; }

public override string ToString()

{

return Description;

}

De hecho, esto puede observarse cuando se copia todo el JSON generado en Postman y es llevado a la página Json2Csharp para que lo convierta a código C#. Se observa que en la clase Category que se genera, existe esta línea de código.

NOTA: Seguimos con el apartado 12.3 para implementar el código que va a consumir el Web API y poder obtener la lista de categorías con sus respectivos productos:

List<Category> listCategories;

public ObservableCollection<Category> \_categories;

public ObservableCollection<Category> Categories

{

get

{

return \_categories;

}

set

{

\_categories = value;

this.Notify("Categories");

}

}

ApiService apiService;

MessageService messageService;

public CategoriesViewModel()

{

apiService = new ApiService();

messageService = new MessageService();

LoadCategories();

}

async void LoadCategories()

{

var connection = await apiService.CheckConnection();

if(!connection.IsSuccess)

{

await messageService.SendMessage("Error", connection.Message);

return;

}

var mainViewModel = MainViewModel.GetIntance();

var response = await apiService.GetList<Category>("http://productszuluapi.azurewebsites.net", "/api", "/Categories",

mainViewModel.tokenResponse.TokenType, mainViewModel.tokenResponse.AccessToken);

if (!response.IsSuccess)

{

await messageService.SendMessage("Error", response.Message);

return;

}

listCategories = (List<Category>)response.Result;

Categories = new ObservableCollection<Category>(listCategories.OrderBy(x => x.Description));

}

12.5.- El diseño de la CategoriesView quedaría así:

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

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

x:Class="MyStock.Views.CategoriesView"

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

Title="Categories">>

<ContentPage.Content>

<StackLayout

BindingContext="{Binding Categories}"

Padding="8"

Spacing="16">

<ListView

HasUnevenRows="True"

ItemsSource="{Binding Categories}">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<Label

Grid.Column="0"

HorizontalOptions="FillAndExpand"

VerticalOptions="Center"

Text="{Binding Description}"

FontSize="Large"/>

<Image

Source="ic\_chevron\_right"

HorizontalOptions="End">

<Image.GestureRecognizers>

<TapGestureRecognizers Command="{Binding SelectedCategoryCommand}"/>

</Image.GestureRecognizers>

</Image>

</Grid>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage.Content>

</ContentPage>

12.6 Creamos la ProductView para listar los productos pertenecientes a una categoría:

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

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

x:Class="MyStock.Views.ProductsView"

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

Title="Products">

<ContentPage.Content>

<StackLayout

BindingContext="{Binding Products}"

Padding="8"

Spacing="16">

<ListView

HasUnevenRows="True"

ItemsSource="{Binding Products}">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<Image

Grid.Column="0"

Source="{Binding ImageFullPath}"

HorizontalOptions="Start"

WidthRequest="100">

</Image>

<Label

Grid.Column="1"

HorizontalOptions="FillAndExpand"

VerticalOptions="Center"

Text="{Binding Description}"

FontSize="Medium">

</Label>

</Grid>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage.Content>

</ContentPage>

Como la imagen fue bindada a una nueva propiedad que debe ir dentro de la clase Product, esta se debe incorporar con el siguiente código:

public string ImageFullPath

{

get

{

return string.Format("http://productszulu.azurewebsites.net/{0}", Image.Substring(1));

}

}

Nos regresamos a la clase Category creada en el apartado 12.2 y que se encuentra en la carpeta Models del proyecto raíz para desarrollar el código de la acción SelectedCategoryCommand y que va a permitir desplegar la lista de productos:

Agregamos el siguiente código:

NavigationService navigationService;

public Category()

{

navigationService = new NavigationService();

}

public ICommand SelectedCategoryCommand

{

get;

set;

}

async void SelectedCategory()

{

var mainViewModel = MainViewModel.GetIntance();

mainViewModel.Products = new ProductsViewModel(Productos);

await navigationService.NavigateTo("ProductsView");

}

12.7 Desarrollamos el código para la ProductsViewModel:

public class ProductsViewModel : ViewModelBase

List<Product> listProducts;

public ObservableCollection<Product> \_products;

public ObservableCollection<Product> Products

{

get

{

return \_products;

}

set

{

\_products = value;

this.Notify("Products");

}

}

public ProductsViewModel(List<Product> lproducts)

{

this.listProducts = lproducts;

Products = new ObservableCollection<Product>(listProducts.OrderBy(x => x.Description));

}

}

12.8.- Hacer que desde la CategoryView se puedan agregar nuevas categorías:

12.8.1.- Escribir el código en CategoryView.axml para insertar un ícono que va a desencadenar la acción para agregar una nueva categoría:

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

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

x:Class="MyStock.Views.CategoriesView"

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

Title="Categories">

<ContentPage.ToolbarItems>

<ToolbarItem Icon="ic\_add\_cat"

Command="{Binding NewCategoryCommand}"/>

</ContentPage.ToolbarItems>

<ContentPage.Content>

<StackLayout

12.8.2.- La lógica del NewCategoryCommand se va desarrollar dentro de la MainViewModel, porque esta gobierna a todos los comandos que correspondan a la ToolbarItems. Incorporamos este código:

public TokenResponse tokenResponse { get; set; }

public LoginViewModel Login { get; set; }

public CategoriesViewModel Categories { get; set; }

public ProductsViewModel Products { get; set; }

public NewCategoryViewModel NewCategory { get; set; }

public ICommand NewCategoryCommand

{

get;

set;

}

public async void GoNewCategory()

{

NewCategory = new NewCategoryViewModel();

await navigationService.NavigateTo("NewCategoryView");

}

12.9.- Crear la NewCategoyView:

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

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

x:Class="MyStock.Views.NewCategoryView"

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

Title="New Category">

<ContentPage.Content>

<StackLayout

BindingContext="{Binding NewCategory}"

Padding="8"

Spacing="16">

<Label Text="Cetegory"

VerticalOptions="Center"/>

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

Placeholder="Enter category description..."/>

<ActivityIndicator

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

<Button Text="Save"

BackgroundColor="Navy"

Style="{StaticResource GeneralButton}"

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

Command="{Binding SaveCommand}"/>

</StackLayout>

</ContentPage.Content>

</ContentPage>

12.10.- Crear la NewCategoryViewModel:

public class NewCategoryViewModel : ViewModelBase

{

string description;

public string Description

{

get

{

return description;

}

set

{

description = value;

this.Notify("Description");

}

}

bool isrunning;

public bool IsRunning

{

get

{

return isrunning;

}

set

{

isrunning = value;

this.Notify("IsRunning");

}

}

bool isenabled;

public bool IsEnabled

{

get

{

return isenabled;

}

set

{

isenabled = value;

this.Notify("IsEnabled");

}

}

MessageService messageService;

NavigationService navigationService;

ApiService apiService;

public NewCategoryViewModel()

{

messageService = new MessageService();

navigationService = new NavigationService();

apiService = new ApiService();

this.SaveCommand = new Command(this.Save);

IsEnabled = true;

}

public ICommand SaveCommand

{

get;

set;

}

async void Save()

{

if (string.IsNullOrEmpty(Description))

{

await messageService.SendMessage("Error", "You must enter a description category.");

}

this.IsEnabled = false;

this.IsRunning = true;

var connection = await apiService.CheckConnection();

if (!connection.IsSuccess)

{

this.IsRunning = false;

this.IsEnabled = true;

await messageService.SendMessage("Error", connection.Message);

return;

}

var category = new Category

{

Description = this.Description,

};

var mainViewModel = MainViewModel.GetIntance();

var response = await apiService.Post("http://productszuluapi.azurewebsites.net", "/api","/Categories",

mainViewModel.tokenResponse.TokenType, mainViewModel.tokenResponse.AccessToken,category);

if (!response.IsSuccess)

{

this.IsRunning = false;

this.IsEnabled = true;

await messageService.SendMessage("Error", response.Message);

return;

}

category = (Category)response.Result;

var categoriesViewModel = CategoriesViewModel.GetIntance();

categoriesViewModel.AddCategory(category);

await navigationService.NavigateToBack();

IsEnabled = true;

IsRunning = false;

}

12.11.- Actualizar la nueva categoría creada en la lista de categorías (CategoryView). Para ello, tenemos que referenciar desde la NewCategoryViewModel a la ViewModel de Categories (CategoriesViewModel) y para hacer eso, debemos crear un singleton dentro de la ViewModel que será referenciada, es decir, en este caso, dentro de la CategoriesViewModel, por lo que se debe adicionar el siguiente código:

//Singlenton

static CategoriesViewModel instance;

public static CategoriesViewModel GetIntance()

{

if (instance == null)

return new CategoriesViewModel();

else

return instance;

}

Y luego dentro del constructor de la CategoriesViewModel, debemos agregar la siguiente línea:

instance = this;

Adicionalmente, dentro de la CategoryViewModel debemos desarrollar el método que permite agregar la nueva categoría, quedando su código de la siguiente manera:

public void AddCategory(Category newCategory)

{

listCategories.Add(newCategory);

Categories = new ObservableCollection<Category>(listCategories.OrderBy(x => x.Description));

}