Usuario GitHub: **serraguti**

Hot Reload. Lo que permite es visualizar las Vistas en ejecución y ver los cambios sin volver a cargar, simplemente guardando.

Es una característica que está activada ya por defecto y es igual a lo que sucede con React o Angular o Vue.

Utilizar una aplicación de Xamarin. Es una aplicación Multiplataforma que está orientada a dispositivos móviles.

Xamarin ya no existe (según Microsoft), ahora está Maui. Todavía no han sacado la versión, pero las diferencias son mínimas, simplemente que permite también incluir Apps para Linux o MacOS.

Vamos a trabajar con un Api publicado que consumirá una base de datos de Azure.

Utilizaremos un CRUD y os iré poniendo en los proyectos, este documento y también el Script de bases de datos.

Comenzamos creando un nuevo proyecto Net Core Web Api llamado **ApiProductosLabs**

Agregamos los Nuget de acceso a datos

Texto

Descripción generada automáticamente

Creamos una carpeta llamada **Models** y una clase llamada **Producto**

**PRODUCTO**

{

[Table("PRODUCTOSLABS")]

public class Producto

{

[Key]

[Column("IDPRODUCTO")]

public int IdProducto { get; set; }

[Column("NOMBRE")]

public string Nombre { get; set; }

[Column("DESCRIPCION")]

public string Descripcion { get; set; }

[Column("PRECIO")]

public int Precio { get; set; }

[Column("IMAGEN")]

public string Imagen { get; set; }

}

Creamos una carpeta llamada **Data** y una clase llamada **ProductosContext**

**PRODUCTOSCONTEXT**

public class ProductosContext: DbContext

{

public ProductosContext(DbContextOptions<ProductosContext> options) : base(options) { }

public DbSet<Producto> Productos { get; set; }

}

Creamos una carpeta llamada **Repositories** y una clase llamada **RepositoryProductos**

**REPOSITORYPRODUCTOS**

public class RepositoryProductos

{

private ProductosContext context;

public RepositoryProductos(ProductosContext context)

{

this.context = context;

}

public List<Producto> GetProductos()

{

return this.context.Productos.ToList();

}

public Producto FindProducto(int id)

{

return this.context.Productos.FirstOrDefault(x => x.IdProducto == id);

}

public int GetMaxIdProducto()

{

if (this.context.Productos.Count() == 0)

{

return 1;

}

else

{

return this.context.Productos.Max(x => x.IdProducto) + 1;

}

}

public async Task AddProducto(string nombre, string descripcion, int precio, string imagen)

{

Producto producto = new Producto()

{

Nombre = nombre,

IdProducto = this.GetMaxIdProducto(),

Descripcion = descripcion,

Precio = precio,

Imagen = imagen

};

await this.context.Productos.AddAsync(producto);

await this.context.SaveChangesAsync();

}

public async Task UpdateProducto(int id, string nombre, string descripcion

, int precio, string imagen)

{

Producto producto = this.FindProducto(id);

producto.Nombre = nombre;

producto.Descripcion = descripcion;

producto.Precio = precio;

producto.Imagen = imagen;

await this.context.SaveChangesAsync();

}

public async Task DeleteProducto(int id)

{

Producto producto = this.FindProducto(id);

this.context.Productos.Remove(producto);

await this.context.SaveChangesAsync();

}

}

Sobre la carpeta **Controllers**, agregamos un nuevo controlador llamado **ProductosController**

**PRODUCTOSCONTROLLER**

[Route("api/[controller]")]

[ApiController]

public class ProductosController : ControllerBase

{

private RepositoryProductos repo;

public ProductosController(RepositoryProductos repo)

{

this.repo = repo;

}

[HttpGet]

public ActionResult<List<Producto>> GetProductos()

{

return this.repo.GetProductos();

}

[HttpGet("{id}")]

public ActionResult<Producto> FindProducto(int id)

{

return this.repo.FindProducto(id);

}

[HttpPost]

public async Task<ActionResult> AddProducto(Producto producto)

{

await this.repo.AddProducto(producto.Nombre, producto.Descripcion

, producto.Precio, producto.Imagen);

return Ok();

}

[HttpDelete("{id}")]

public async Task<ActionResult> DeleteProducto(int id)

{

await this.repo.DeleteProducto(id);

return Ok();

}

[HttpPut]

public async Task<ActionResult> UpdateProducto(Producto producto)

{

await this.repo.UpdateProducto(producto.IdProducto, producto.Nombre

, producto.Descripcion, producto.Precio, producto.Imagen);

return Ok();

}

}

Incluimos nuestra cadena de conexión dentro de **appsettings.json**

**APPSETTINGS.JSON**

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"AzureSql": "Data Source=azurecastraininglabs.database.windows.net;Initial Catalog=AZUREDATABASE;Persist Security Info=True;User ID=adminsql;Password=Admin123"

}

}

Resolvemos las dependencias dentro de **Program.cs**

**PROGRAM.CS**

using ApiProductosLabs.Data;

using ApiProductosLabs.Repositories;

using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddTransient<RepositoryProductos>();

string connectionString = builder.Configuration.GetConnectionString("AzureSql");

builder.Services.AddDbContext<ProductosContext>

(options => options.UseSqlServer(connectionString));

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

}

app.UseSwagger();

app.UseSwaggerUI(

options =>

{

options.SwaggerEndpoint("/swagger/v1/swagger.json", "Api Microsoft Labs v1");

options.RoutePrefix = "";

});

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

Creamos una nueva aplicación llamada **AppMobileProductos** de tipo Xamarin Forms.

Agregamos el siguiente Nuget

Interfaz de usuario gráfica, Texto, Aplicación, Word

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Word

Descripción generada automáticamente

Creamos una carpeta **Models** y una clase llamada **Producto**

**PRODUCTO**

public class Producto

{

public int IdProducto { get; set; }

public string Nombre { get; set; }

public string Descripcion { get; set; }

public int Precio { get; set; }

public string Imagen { get; set; }

}

Creamos una carpeta llamada **Services** y una clase llamada **ServiceApiProductos**

**SERVICEAPIPRODUCTOS**

public class ServiceApiProductos

{

private string ApiUrl;

private MediaTypeWithQualityHeaderValue Header;

public ServiceApiProductos()

{

this.ApiUrl = "https://apiproductoslabs.azurewebsites.net/";

this.Header =

new MediaTypeWithQualityHeaderValue("application/json");

}

private async Task<T> CallApiAsync<T>(string request)

{

using (HttpClient client = new HttpClient())

{

client.DefaultRequestHeaders.Accept.Clear();

client.DefaultRequestHeaders.Accept.Add(this.Header);

Uri uri = new Uri(this.ApiUrl + request);

HttpResponseMessage response =

await client.GetAsync(uri);

if (response.IsSuccessStatusCode)

{

T data =

await response.Content.ReadAsAsync<T>();

return data;

}else

{

return default(T);

}

}

}

public async Task<List<Producto>> GetProductosAsync()

{

string request = "/api/productos";

List<Producto> products = await CallApiAsync<List<Producto>>(request);

return products;

}

public async Task<Producto> FindProductoAsync(int id)

{

string request = "/api/productos/" + id;

Producto producto = await CallApiAsync<Producto>(request);

return producto;

}

public async Task AddProductoAsync(Producto producto)

{

using (HttpClient client = new HttpClient())

{

string request = "/api/productos";

Uri uri = new Uri(this.ApiUrl + request);

client.DefaultRequestHeaders.Accept.Clear();

client.DefaultRequestHeaders.Accept.Add(this.Header);

string json = JsonConvert.SerializeObject(producto);

StringContent content =

new StringContent(json, Encoding.UTF8, "application/json");

HttpResponseMessage response =

await client.PostAsync(uri, content);

}

}

public async Task UpdateProductoAsync(Producto producto)

{

using (HttpClient client = new HttpClient())

{

string request = "/api/productos";

Uri uri = new Uri(this.ApiUrl + request);

client.DefaultRequestHeaders.Accept.Clear();

client.DefaultRequestHeaders.Accept.Add(this.Header);

string json = JsonConvert.SerializeObject(producto);

StringContent content =

new StringContent(json, Encoding.UTF8, "application/json");

HttpResponseMessage response =

await client.PutAsync(uri, content);

}

}

public async Task DeleteProductoAsync(int id)

{

using (HttpClient client = new HttpClient())

{

string request = "/api/productos/" + id;

Uri uri = new Uri(this.ApiUrl + request);

client.DefaultRequestHeaders.Accept.Clear();

client.DefaultRequestHeaders.Accept.Add(this.Header);

HttpResponseMessage response =

await client.DeleteAsync(uri);

}

}

}

Creamos una carpeta llamada **Base** y una clase llamada **ViewModelBase**

**VIEWMODELBASE**

public class ViewModelBase : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

public void OnPropertyChanged(string propertyName)

{

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

}

}

Creamos una carpeta llamada **ViewModels** y una clase llamada **ProductosListViewModel**

**PRODUCTOSLISTVIEWMODEL**

public class ProductosListViewModel: ViewModelBase

{

private ServiceApiProductos service;

public ProductosListViewModel()

{

this.service = new ServiceApiProductos();

}

public Command CargarProductos

{

get

{

return new Command(async() =>

{

await this.LoadProductosAsync();

});

}

}

private async Task LoadProductosAsync()

{

List<Producto> productos =

await this.service.GetProductosAsync();

this.Productos =

new ObservableCollection<Producto>(productos);

}

private ObservableCollection<Producto> \_Productos;

public ObservableCollection<Producto> Productos

{

get { return this.\_Productos; }

set {

this.\_Productos = value;

OnPropertyChanged("Productos");

}

}

}

Sobre **Views**, creamos un nuevo **ContentPage** llamado **ProductosListView.xaml**

**PRODUCTOSLISTVIEW.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"

x:Class="AppMobileProductos.Views.ProductosListView"

xmlns:viewmodels="clr-namespace:AppMobileProductos.ViewModels">

<ContentPage.Content>

<StackLayout>

<StackLayout.BindingContext>

<viewmodels:ProductosListViewModel/>

</StackLayout.BindingContext>

<Label Text="Productos Api"

VerticalOptions="Start" FontSize="30"

HorizontalOptions="CenterAndExpand" />

<Button Text="Cargar Productos"

Command="{Binding CargarProductos}"/>

<ListView ItemsSource="{Binding Productos}"

RowHeight="150">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<StackLayout Orientation="Horizontal">

<Image Source="{Binding Imagen}"

HeightRequest="145"/>

<StackLayout Orientation="Vertical">

<Label Text="{Binding Nombre}" TextColor="Blue"/>

<Label Text="{Binding Precio}" TextColor="Red"/>

</StackLayout>

</StackLayout>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage.Content>

</ContentPage>

El siguiente paso será utilizar la inyección de dependencias, algo básico en este tipo de Apps.

Agregamos el siguiente Nuget al proyecto.

Interfaz de usuario gráfica, Texto, Aplicación, Word

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Word

Descripción generada automáticamente

Sobre el proyecto, creamos un nuevo fichero llamado **appsettings.json** y lo ponemos, en sus propiedades, como **Recurso incrustado**

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

{

"ApiUrls": {

"ApiProductos": "https://apiproductoslabs.azurewebsites.net/"

}

}

Creamos una clase, dentro de **Services** llamada **ServiceIoC**

**SERVICEIOC**

public class ServiceIoC

{

private IContainer container;

public ServiceIoC()

{

this.RegisterDependencies();

}

private void RegisterDependencies()

{

ContainerBuilder builder =

new ContainerBuilder();

builder.RegisterType<ServiceApiProductos>();

builder.RegisterType<ProductosListViewModel>();

string resourceName = "AppMobileProductos.appsettings.json";

Stream stream = GetType().GetTypeInfo().Assembly.GetManifestResourceStream(resourceName);

IConfiguration configuration =

new ConfigurationBuilder().AddJsonStream(stream)

.Build();

builder.Register<IConfiguration>(x => configuration);

this.container = builder.Build();

}

//PROPIEDADES CON LOS VIEWMODELS PARA PODER RECUPERARLOS DENTRO

//DE LOS CODIGOS XAML

public ProductosListViewModel ProductosListViewModel

{

get

{

return this.container.Resolve<ProductosListViewModel>();

}

}

}

Para que XAML sea capaz de recuperar los ViewModels de ServiceIoC, vamos a declarar una propiedad Static para buscar los ViewModels dentro de la clase **App.xaml.cs**

**APP.XAML.CS**

private static ServiceIoC \_ServiceLocator;

public static ServiceIoC ServiceLocator

{

get

{

return \_ServiceLocator = \_ServiceLocator ?? new ServiceIoC();

}

}

Debemos cambiar el Binding de la vista, de forma que BUSCARA en lugar de INSTANCIAR el ViewModel

**PRODUCTOSLISTVIEW.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"

x:Class="AppMobileProductos.Views.ProductosListView"

xmlns:viewmodels="clr-namespace:AppMobileProductos.ViewModels"

xmlns:local="clr-namespace:AppMobileProductos">

<ContentPage.Content>

<StackLayout

BindingContext="{Binding Source={x:Static local:App.ServiceLocator}

, Path=ProductosListViewModel}">

<Label Text="Productos Api"

VerticalOptions="Start" FontSize="30"

HorizontalOptions="CenterAndExpand" />

<Button Text="Cargar Productos"

Command="{Binding CargarProductos}"/>

<ListView ItemsSource="{Binding Productos}"

RowHeight="150">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<StackLayout Orientation="Horizontal">

<Image Source="{Binding Imagen}"

HeightRequest="145"/>

<StackLayout Orientation="Vertical">

<Label Text="{Binding Nombre}" TextColor="Blue"/>

<Label Text="{Binding Precio}" TextColor="Red"/>

</StackLayout>

</StackLayout>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage.Content>

</ContentPage>

A continuación, vamos a incluir, en la vista del conjunto de productos un botón para visualizar los Detalles de un producto.

Comenzamos creando un nuevo ViewModel llamado **ProductoDetailsViewModel**, dentro de la carpeta **ViewModels**

**PRODUCTOSDETAILSVIEWMODEL**

public class ProductoDetailsViewModel: ViewModelBase

{

private Producto \_Producto;

public Producto Producto

{

get { return this.\_Producto; }

set {

this.\_Producto = value;

OnPropertyChanged("Producto");

}

}

}

Sobre **Views**, creamos una vista llamada **ProductoDetailsView.xaml**

**PRODUCTODETAILSVIEW.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"

x:Class="AppMobileProductos.Views.ProductoDetailsView">

<ContentPage.Content>

<ScrollView Orientation="Vertical">

<StackLayout>

<Label Text="Producto Details"

TextColor="Blue" FontSize="40"

VerticalOptions="Start"

HorizontalOptions="CenterAndExpand" />

<Label Text="{Binding Producto.Nombre}"

FontSize="25"/>

<Image Source="{Binding Producto.Imagen}"

WidthRequest="300"/>

<Label Text="{Binding Producto.Descripcion}" TextColor="Black"/>

<Label Text="{Binding Producto.Precio, StringFormat='Precio {0} €'}"

FontSize="25" TextColor="Red"/>

</StackLayout>

</ScrollView>

</ContentPage.Content>

</ContentPage>

Implementamos un Command (Acción) dentro de la lista de productos ViewModel

**PRODUCTOSLISTVIEWMODEL**

public class ProductosListViewModel: ViewModelBase

{

private ServiceApiProductos service;

public ProductosListViewModel(ServiceApiProductos service)

{

this.service = service;

}

public Command ShowProductDetails

{

get

{

return new Command(async(idProducto) =>

{

int id = (int)idProducto;

//BUSCAMOS EL PRODUCTO DENTRO DEL API

Producto producto = await this.service.FindProductoAsync(id);

//CREAMOS EL VIEWMODEL

ProductoDetailsViewModel viewmodel =

new ProductoDetailsViewModel();

//INDICAMOS AL VIEWMODEL QUE ELEMENTO DIBUJARA

viewmodel.Producto = producto;

//CREAMOS LA VISTA DE DETALLES

ProductoDetailsView view = new ProductoDetailsView();

//ENLAZAMOS LA VISTA CON SU VIEWMODEL

view.BindingContext = viewmodel;

//MOSTRAMOS LA VISTA PARA VISUALIZAR SUS DATOS

await Application.Current.MainPage.Navigation

.PushModalAsync(view);

});

}

}

public Command CargarProductos

{

get

{

return new Command(async() =>

{

await this.LoadProductosAsync();

});

}

}

private async Task LoadProductosAsync()

{

List<Producto> productos =

await this.service.GetProductosAsync();

this.Productos =

new ObservableCollection<Producto>(productos);

}

private ObservableCollection<Producto> \_Productos;

public ObservableCollection<Producto> Productos

{

get { return this.\_Productos; }

set {

this.\_Productos = value;

OnPropertyChanged("Productos");

}

}

}

Necesitamos incluir un Button dentro del control **ListView** en la vista del conjunto de Productos.

**PRODUCTOSLISTVIEW.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"

x:Class="AppMobileProductos.Views.ProductosListView"

xmlns:viewmodels="clr-namespace:AppMobileProductos.ViewModels"

xmlns:local="clr-namespace:AppMobileProductos">

<ContentPage.Content>

<StackLayout x:Name="stackProductos"

BindingContext="{Binding Source={x:Static local:App.ServiceLocator}

, Path=ProductosListViewModel}">

<Label Text="Productos Api"

VerticalOptions="Start" FontSize="30"

HorizontalOptions="CenterAndExpand" />

<Button Text="Cargar Productos"

Command="{Binding CargarProductos}"/>

<ListView ItemsSource="{Binding Productos}"

RowHeight="150">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<StackLayout Orientation="Horizontal">

<Image Source="{Binding Imagen}"

HeightRequest="145"/>

<StackLayout Orientation="Vertical">

<Label Text="{Binding Nombre}" TextColor="Blue"/>

<Label Text="{Binding Precio}" TextColor="Red"/>

<Button Text="Details"

Command="{Binding BindingContext.ShowProductDetails

, Source={x:Reference stackProductos}}"

CommandParameter="{Binding IdProducto}"/>

</StackLayout>

</StackLayout>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage.Content>

</ContentPage>

Vamos a realizar la funcionalidad para Eliminar productos.

Lo que haremos será eliminar desde dos sitios diferentes:

1. Podremos eliminar desde la LISTA

Incluimos una propiedad Command para eliminar en la lista dentro del ViewModel de **ProductosListViewModel**

**PRODUCTOSLISTVIEWMODEL**

public Command DeleteProduct

{

get

{

return new Command(async(idProducto) =>

{

int id = (int)idProducto;

await this.service.DeleteProductoAsync(id);

await Application.Current.MainPage.DisplayAlert("Alert"

, "Producto eliminado", "Ok");

//RECARGAMOS LOS PRODUCTOS

await this.LoadProductosAsync();

});

}

}

Incluimos un Button dentro de la vista de colección de productos.

**PRODUCTOSLISTVIEW.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"

x:Class="AppMobileProductos.Views.ProductosListView"

xmlns:viewmodels="clr-namespace:AppMobileProductos.ViewModels"

xmlns:local="clr-namespace:AppMobileProductos">

<ContentPage.Content>

<StackLayout x:Name="stackProductos"

BindingContext="{Binding Source={x:Static local:App.ServiceLocator}

, Path=ProductosListViewModel}">

<Label Text="Productos Api"

VerticalOptions="Start" FontSize="30"

HorizontalOptions="CenterAndExpand" />

<Button Text="Cargar Productos"

Command="{Binding CargarProductos}"/>

<ListView ItemsSource="{Binding Productos}"

RowHeight="150">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<StackLayout Orientation="Horizontal">

<Image Source="{Binding Imagen}"

HeightRequest="145"/>

<StackLayout Orientation="Vertical">

<Label Text="{Binding Nombre}" TextColor="Blue"/>

<Label Text="{Binding Precio}" TextColor="Red"/>

<Button Text="Details"

Command="{Binding BindingContext.ShowProductDetails

, Source={x:Reference stackProductos}}"

CommandParameter="{Binding IdProducto}"/>

<Button Text="Delete" TextColor="Red"

Command="{Binding BindingContext.DeleteProduct

, Source={x:Reference stackProductos}}"

CommandParameter="{Binding IdProducto}"/>

</StackLayout>

</StackLayout>

</ViewCell>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</StackLayout>

</ContentPage.Content>

</ContentPage>

1. Podremos eliminar desde DETALLES

A continuación, vamos a eliminar, pero desde la ventana Detalles.

Al pulsar sobre el botón que dibujaremos, mostraremos un mensaje y cerraremos la ventana.

A ver que pasa…

Lo primero será, dentro de **ProductoDetailsViewModel**, recibiremos el ServicioApi para poder llamar a la acción de eliminar

**PRODUCTODETAILSVIEWMODEL**

public class ProductoDetailsViewModel: ViewModelBase

{

private ServiceApiProductos service;

public ProductoDetailsViewModel(ServiceApiProductos service)

{

this.service = service;

}

public Command DeleteProduct

{

get

{

return new Command(async() =>

{

int id = this.Producto.IdProducto;

await this.service.DeleteProductoAsync(id);

await

Application.Current.MainPage

.DisplayAlert("Alert", "Producto eliminado", "Ok");

await

Application.Current.MainPage.Navigation

.PopModalAsync();

});

}

}

private Producto \_Producto;

public Producto Producto

{

get { return this.\_Producto; }

set {

this.\_Producto = value;

OnPropertyChanged("Producto");

}

}

}

A continuación, como hemos realizado inyección de dependencias con el Service, necesitamos poder recuperar el ViewModel desde nuestro ServiceIoC.

**SERVICEIOC**

public class ServiceIoC

{

private IContainer container;

public ServiceIoC()

{

this.RegisterDependencies();

}

private void RegisterDependencies()

{

ContainerBuilder builder =

new ContainerBuilder();

builder.RegisterType<ServiceApiProductos>();

builder.RegisterType<ProductosListViewModel>();

builder.RegisterType<ProductoDetailsViewModel>();

string resourceName = "AppMobileProductos.appsettings.json";

Stream stream = GetType().GetTypeInfo().Assembly.GetManifestResourceStream(resourceName);

IConfiguration configuration =

new ConfigurationBuilder().AddJsonStream(stream)

.Build();

builder.Register<IConfiguration>(x => configuration);

this.container = builder.Build();

}

public ProductoDetailsViewModel ProductoDetailsViewModel

{

get

{

return this.container.Resolve<ProductoDetailsViewModel>();

}

}

//PROPIEDADES CON LOS VIEWMODELS PARA PODER RECUPERARLOS DENTRO

//DE LOS CODIGOS XAML

public ProductosListViewModel ProductosListViewModel

{

get

{

return this.container.Resolve<ProductosListViewModel>();

}

}

}

Modificamos el código de **ShowDetails** para recuperar el ViewModel desde C#**,** en lugar de instanciar la clase, que ya no podemos por IoC

**PRODUCTOSLISTVIEWMODEL**

public Command ShowProductDetails

{

get

{

return new Command(async(idProducto) =>

{

int id = (int)idProducto;

//BUSCAMOS EL PRODUCTO DENTRO DEL API

Producto producto = await this.service.FindProductoAsync(id);

//RECUPERAMOS EL VIEWMODEL

ProductoDetailsViewModel viewmodel =

App.ServiceLocator.ProductoDetailsViewModel;

//INDICAMOS AL VIEWMODEL QUE ELEMENTO DIBUJARA

viewmodel.Producto = producto;

//CREAMOS LA VISTA DE DETALLES

ProductoDetailsView view = new ProductoDetailsView();

//ENLAZAMOS LA VISTA CON SU VIEWMODEL

view.BindingContext = viewmodel;

//MOSTRAMOS LA VISTA PARA VISUALIZAR SUS DATOS

await Application.Current.MainPage.Navigation

.PushModalAsync(view);

});

}

}

Por último, incluimos un Button dentro de Details para poder eliminar un Producto.

**PRODUCTODETAILSVIEW.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"

x:Class="AppMobileProductos.Views.ProductoDetailsView">

<ContentPage.Content>

<ScrollView Orientation="Vertical">

<StackLayout>

<Label Text="Producto Details"

TextColor="Blue" FontSize="40"

VerticalOptions="Start"

HorizontalOptions="CenterAndExpand" />

<Label Text="{Binding Producto.Nombre}"

FontSize="25"/>

<Image Source="{Binding Producto.Imagen}"

WidthRequest="300"/>

<Label Text="{Binding Producto.Descripcion}" TextColor="Black"/>

<Label Text="{Binding Producto.Precio, StringFormat='Precio {0} €'}"

FontSize="25" TextColor="Red"/>

<Button Text="Delete" BackgroundColor="Red"

Command="{Binding DeleteProduct}"/>

</StackLayout>

</ScrollView>

</ContentPage.Content>

</ContentPage>

Como podremos comprobar, el dato se elimina perfectamente utilizando el Api, pero el DIBUJO del conjunto de Productos **no cambia**.

Tampoco podemos hacer que cambie, debemos volver a recargar los productos, pero en el ViewModel de la colección. Nosotros estamos en Detalles y no podemos acceder a la vista de la colección de productos.

La solución es utilizar MessagingCenter

Diagrama

Descripción generada automáticamente

Este modelo de programación, nos permite subscribirnos a un Evento y poder lanzar dicho Evento cuando lo deseemos desde cualquier clase.

Esto es básico implementado junto al concepto de IoC.

Para poder enviar mensajes a eventos entre ViewModels, necesitamos que las clases estén inyectadas en un Container.

Comenzamos creando un método dentro de **ProductosListViewModel** que ejecutará la acción de **LoadProductosAsync** cuando reciba un mensaje de Evento en **MessagingCenter**

**PRODUCTOSLISTVIEWMODEL**

public class ProductosListViewModel: ViewModelBase

{

private ServiceApiProductos service;

public ProductosListViewModel(ServiceApiProductos service)

{

this.service = service;

//CUANDO SE INSTANCIA EL OBJETO, CREAMOS UNA

//SUBSCRIPCION PARA REALIZAR ALGUN CODIGO AUNQUE

//NO SEA VISIBLE LA VISTA

MessagingCenter.Subscribe<ProductosListViewModel>

(this, "RELOAD", async (sender) =>

{

//CUANDO RECIBAMOS EL MENSAJE RELOAD

//REALIZARA ESTE CODIGO

await this.LoadProductosAsync();

});

}

El siguiente paso será enviar un mensaje para ejecutar el código desde la vista de Detalles.

**PRODUCTODETAILSVIEWMODEL**

public Command DeleteProduct

{

get

{

return new Command(async() =>

{

int id = this.Producto.IdProducto;

await this.service.DeleteProductoAsync(id);

await

Application.Current.MainPage

.DisplayAlert("Alert", "Producto eliminado", "Ok");

MessagingCenter.Send<ProductosListViewModel>

(App.ServiceLocator.ProductosListViewModel, "RELOAD");

await

Application.Current.MainPage.Navigation

.PopModalAsync();

});

}

}

Y ya tendremos la aplicación funcional

Interfaz de usuario gráfica

Descripción generada automáticamente Texto

Descripción generada automáticamente