Taller: Consumo de servicios Azure desde Xamarin Forms

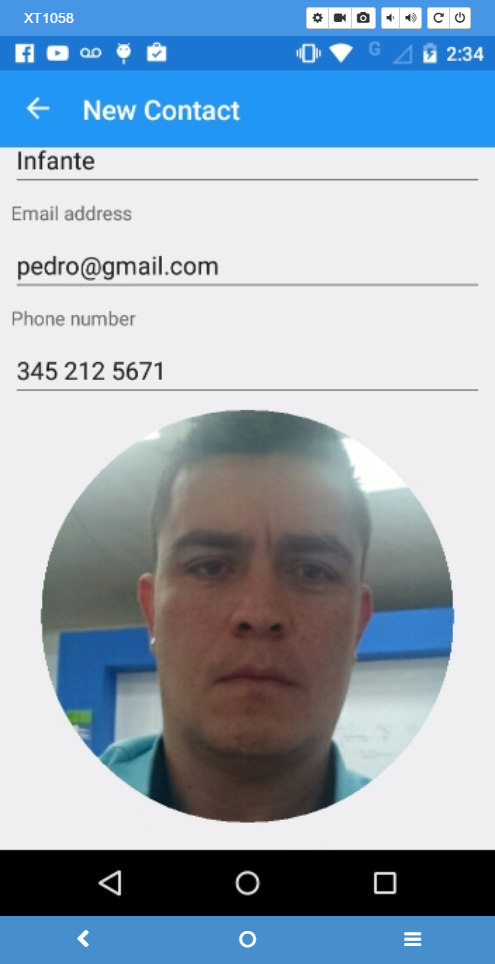
By: Zulu

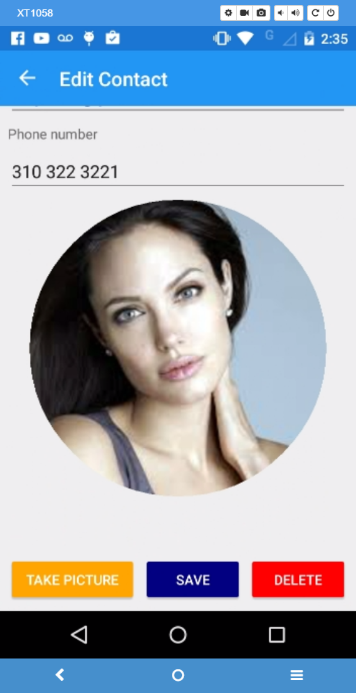
Email: [jzuluaga55@gmail.com](mailto:jzuluaga55@gmail.com)

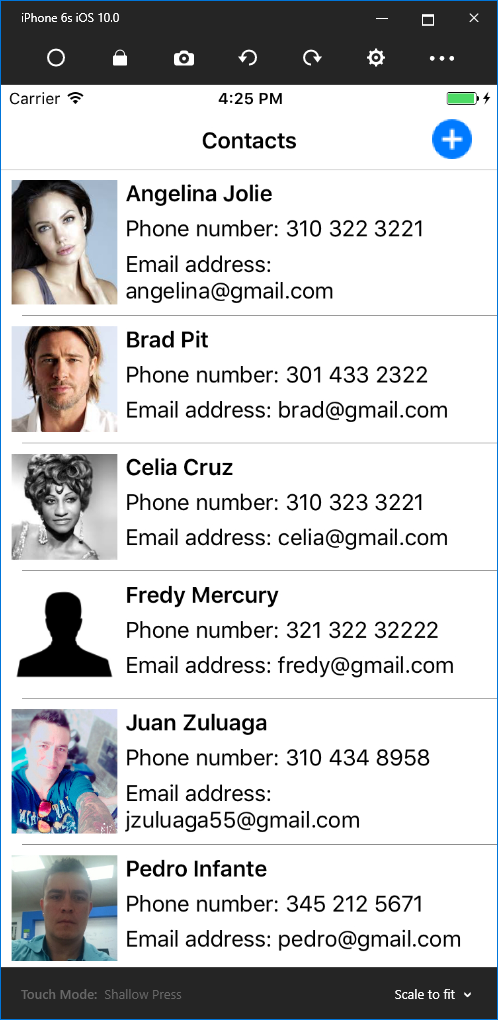
Twitter: @jzuluaga55

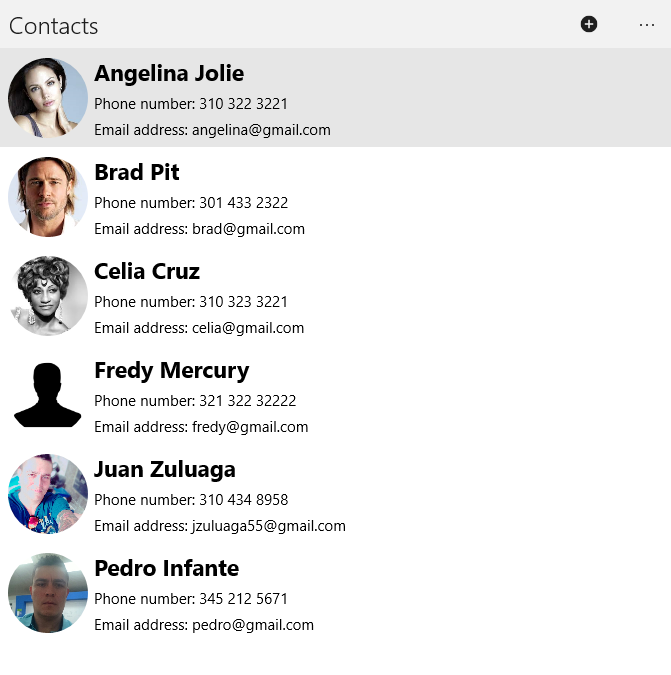


Vamos a desarrollar esta App

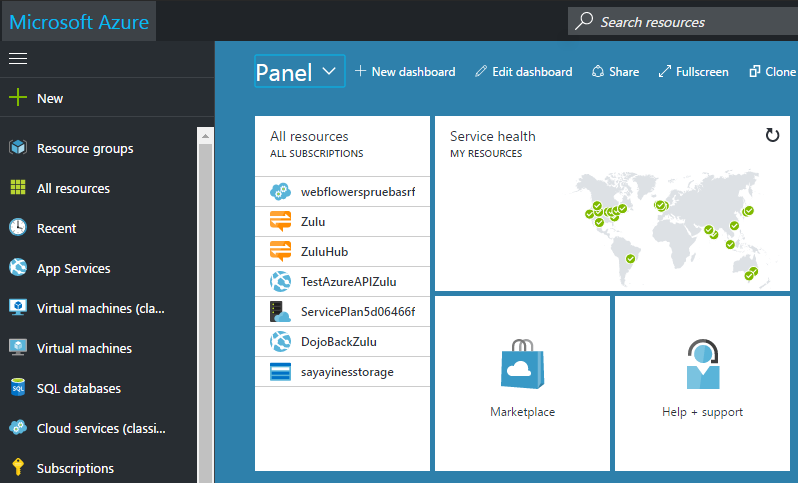




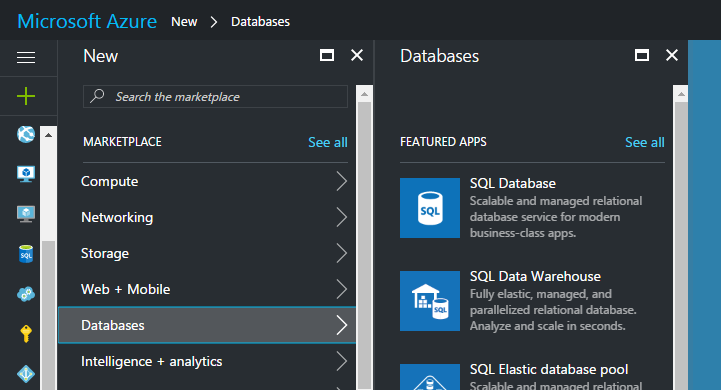




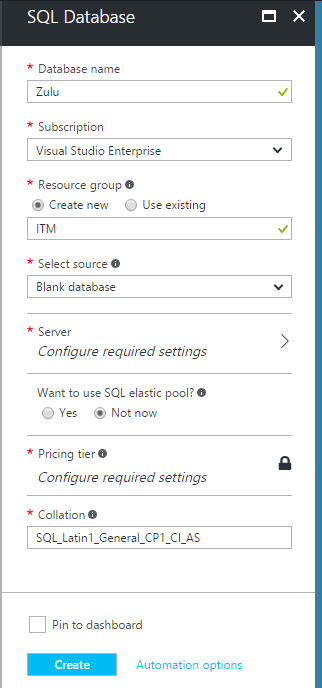
1. Cree en Azure, la base de datos y el servidor de base de datos. Entra al <https://portal.azure.com> y crea una base de datos y el servidor o utiliza una de la que ya tengas creada. Debes tener las credenciales de conexión a la mano.



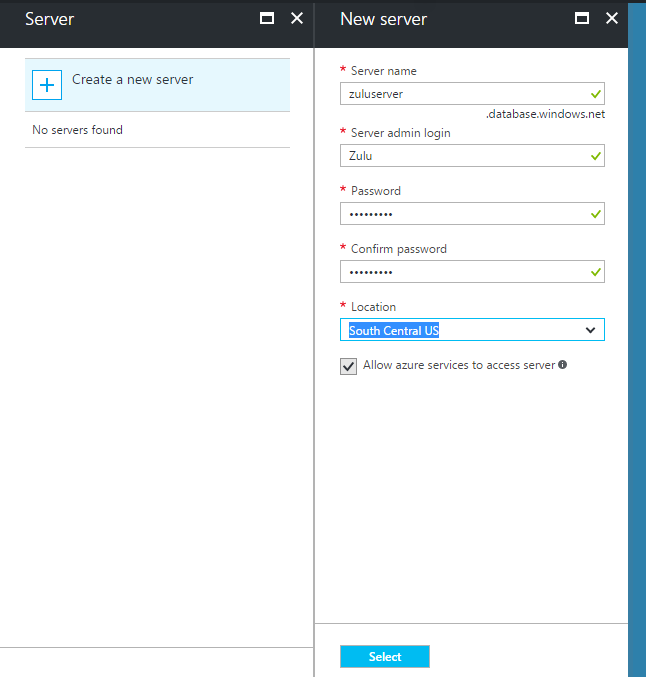
Presionamos New -> Databases -> SQL Database:



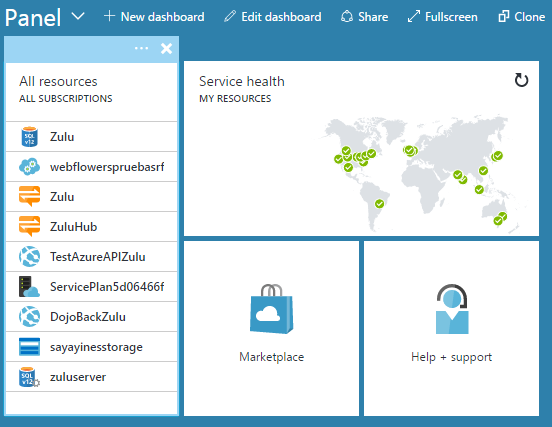
Llenamos los campos:



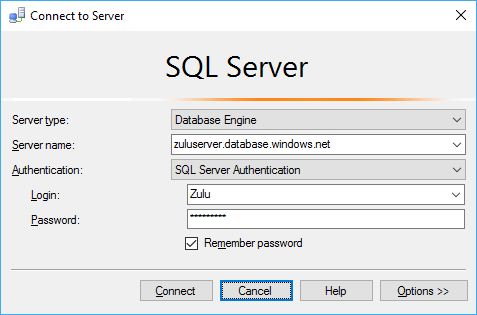
Creamos el servidor:

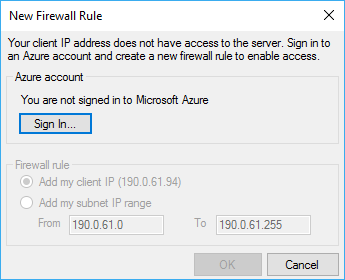


Y confirmamos la creación de la BD y esperamos que el servidor y la BD se creen correctamente.



Es importante mantener la base de datos en S0 o incluso en B, si no la vamos a usar, y subirla a S2 cuando vamos a trabajar. También dependiendo del lugar donde te conectes es posible que toque agregar la dirección IP en el Firewall del servidor. Luego de esto ya te puede conectar con el SQL Management Studio 2016 (disponible en: <https://msdn.microsoft.com/es-co/library/mt238290.aspx>)





1. Crea el proyecto Web y utiliza tus credenciales de Azure
2. Agregue el nuget de Entity Framework
3. Agregue esta sección al **Web.config**:

</configSections>

<connectionStrings>

<!-- Azure -->

<add name="DefaultConnection"

connectionString="Data Source=zuluserver.database.windows.net;Initial Catalog=ZuluDB;Persist Security Info=True;User ID=Zulu;Password=Roger1974"

providerName="System.Data.SqlClient" />

</connectionStrings>

<appSettings>

1. Crea el modelo **Contact**:

public class Contact

{

[Key]

public int ContactId { get; set; }

[Required(ErrorMessage = "You must enter a {0}")]

[StringLength(30, ErrorMessage = "The field {0} can contain maximun {1} and minimum {2} characters", MinimumLength = 1)]

[Display(Name = "First name")]

public string FirstName { get; set; }

[Required(ErrorMessage = "You must enter a {0}")]

[StringLength(30, ErrorMessage = "The field {0} can contain maximun {1} and minimum {2} characters", MinimumLength = 1)]

[Display(Name = "Last name")]

public string LastName { get; set; }

public string Image { get; set; }

[DataType(DataType.EmailAddress)]

[Display(Name = "Email address")]

[StringLength(100, ErrorMessage = "The field {0} can contain maximun {1} and minimum {2} characters", MinimumLength = 7)]

public string EmailAddress { get; set; }

[DataType(DataType.PhoneNumber)]

[Display(Name = "Phone number")]

[StringLength(20, ErrorMessage = "The field {0} can contain maximun {1} and minimum {2} characters", MinimumLength = 7)]

public string PhoneNumber { get; set; }

}

1. Crea la clase **DataContext**:

public class DataContext : DbContext

{

public DataContext() : base("DefaultConnection")

{

}

}

1. Compila la solución y crea los controladores básicos.
2. Modifica el menú para llamar al nuevo controlador. Ejecutemos el proyecto y probemos lo que llevamos.
3. Para poder colocar la imagen con un **FilePicker**, vamos a crear la clase **ContactView** en los modelos:

[NotMapped]

public class ContactView : Contact

{

[Display(Name = "Image")]

public HttpPostedFileBase ImageFile { get; set; }

}

1. Modificamos la vista de **Create** por (solo coloco las líneas que cambian):

@model ContactsBackPrep.Models.ContactView

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

<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>

1. Creamos la clase **FilesHelper** con el método **UploadPhoto**:

public static string UploadPhoto(HttpPostedFileBase file, string folder)

{

var path = string.Empty;

var pic = string.Empty;

if (file != null)

{

pic = Path.GetFileName(file.FileName);

path = Path.Combine(HttpContext.Current.Server.MapPath(folder), pic);

file.SaveAs(path);

}

return pic;

}

1. Modificamos el **POST** del **Create** en el controlador (no olvidar crear la carpeta **Images** dentro de la carpeta **Content**):

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create(ContactView view)

{

if (ModelState.IsValid)

{

var pic = string.Empty;

var folder = "~/Content/Images";

if (view.ImageFile != null)

{

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

pic = string.Format("{0}/{1}", folder, pic);

}

var contact = ToContact(view);

contact.Image = pic;

db.Contacts.Add(contact);

db.SaveChanges();

return RedirectToAction("Index");

}

return View(view);

}

private Contact ToContact(ContactView view)

{

return new Contact

{

ContactId = view.ContactId,

EmailAddress = view.EmailAddress,

FirstName = view.FirstName,

Image = view.Image,

LastName = view.LastName,

PhoneNumber = view.PhoneNumber,

};

}

1. Modifiquemos la vista **Index** para que nos muestre la foto y probamos:

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

{

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

}

1. Ahora hagamos algo muy similar en la vista **Edit** por (solo coloco las líneas que cambian):

@model ContactsBackPrep.Models.ContactView

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

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

<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>

1. Cambiemos el **GET** y el **POST** del **Edit** en el controlador:

public ActionResult Edit(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

var contact = db.Contacts.Find(id);

if (contact == null)

{

return HttpNotFound();

}

var view = ToView(contact);

return View(view);

}

private ContactView ToView(Contact contact)

{

return new ContactView

{

ContactId = contact.ContactId,

EmailAddress = contact.EmailAddress,

FirstName = contact.FirstName,

Image = contact.Image,

LastName = contact.LastName,

PhoneNumber = contact.PhoneNumber,

};

}

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit(ContactView view)

{

if (ModelState.IsValid)

{

var pic = view.Image;

var folder = "~/Content/Images";

if (view.ImageFile != null)

{

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

pic = string.Format("{0}/{1}", folder, pic);

}

var contact = ToContact(view);

contact.Image = pic;

db.Entry(contact).State = EntityState.Modified;

db.SaveChanges();

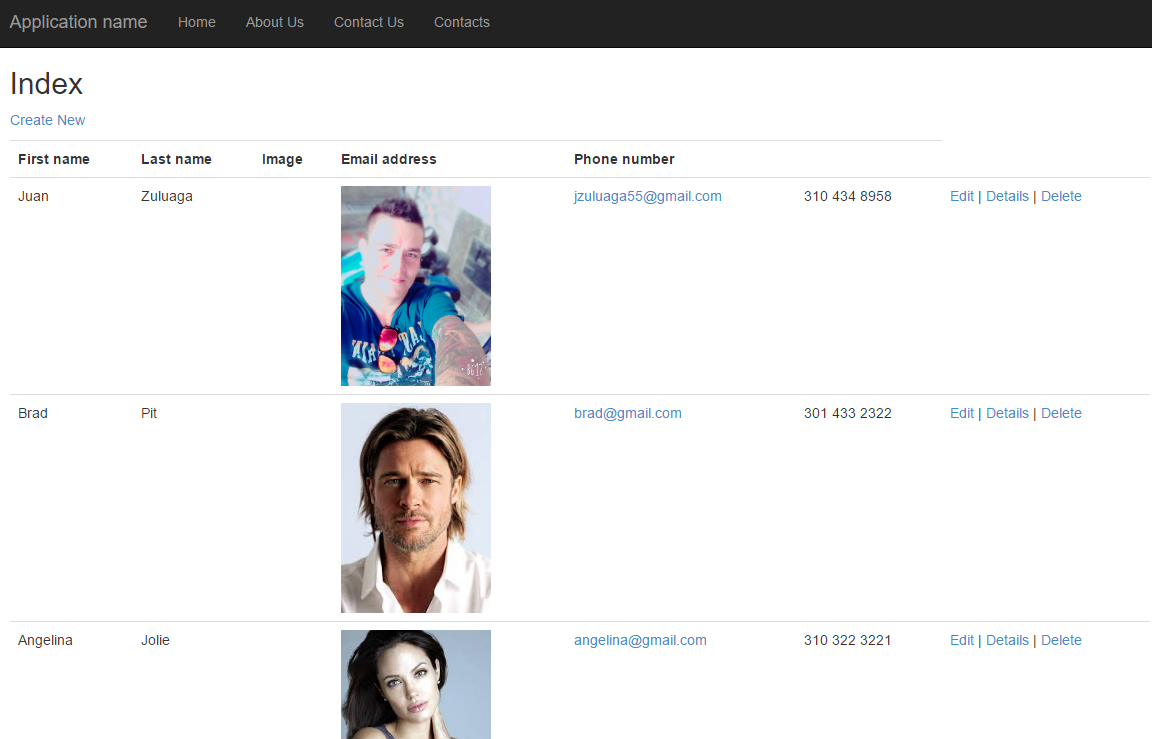
return RedirectToAction("Index");

}

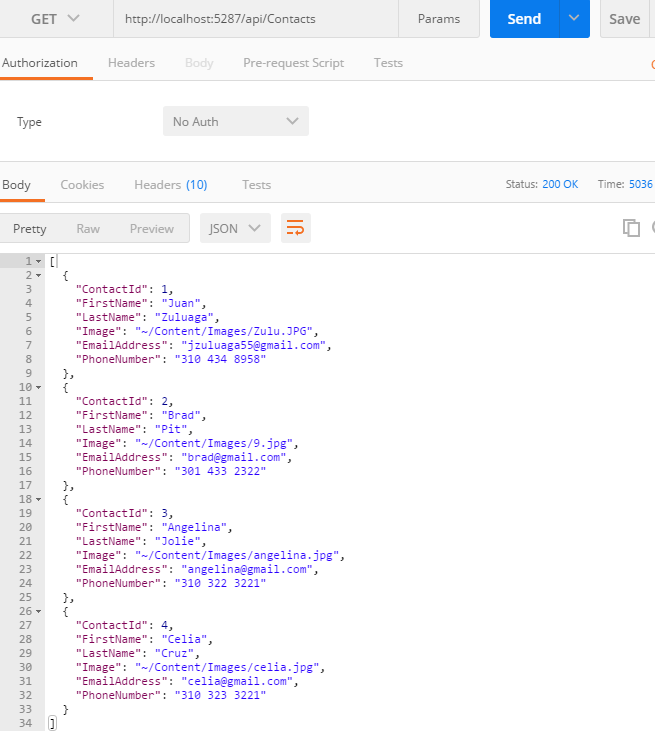
return View(view);

}

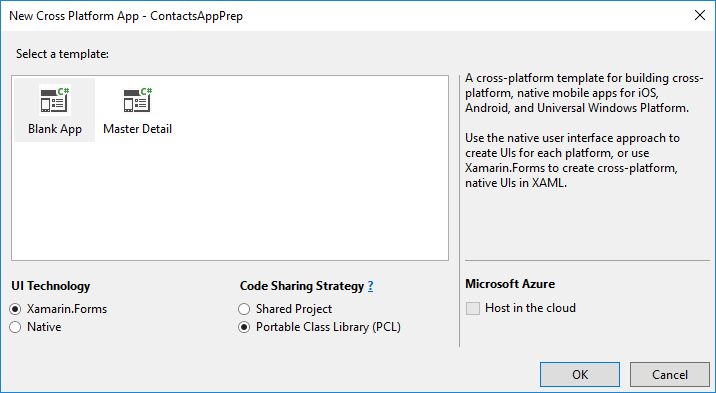
1. Probemos:



1. Ahora creemos el controlador API y probamos en POSTMAN:



1. Publiquemos nuestro sitio en Azure (incluye al menos una imagen de la carpeta **Images**, para que esta sea creada en la publicación).
2. Ahora creemos nuestra aplicación móvil:



1. Probamos que nos funcione el “Welcome to Xamarin Forms” en todas las plataformas.
2. Creamos la carpeta **ViewModels** y dentro de esta creamos la clase **MainViewModel**:

public class MainViewModel

{

}

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

public class InstanceLocator

{

public MainViewModel Main { get; set; }

public InstanceLocator()

{

Main = new MainViewModel();

}

}

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

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

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

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

xmlns:infra="clr-namespace:ContactsAppPrep.Infrastructure;assembly=ContactsAppPrep"

x:Class="ContactsAppPrep.App">

<Application.Resources>

<ResourceDictionary>

<!-- Locator -->

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

</ResourceDictionary>

</Application.Resources>

</Application>

1. Creamos la carpeta **Pages** y dentro de esta vamos a crear nuestra **ContactsPage**:

<?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="ContactsAppPrep.Pages.ContactsPage"

Title="Contacts"

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

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

</ContentPage>

1. Cambiemos el inicio de nuestra aplicación y ya podemos borrar la **MainPage** que no estamos usando:

public App ()

{

InitializeComponent();

MainPage = new NavigationPage(new ContactsPage());

}

1. Ahora vamos a consumir el API para obtener la lista de Flores, primero agregamos a la **SOLUCIÓN** los siguientes Nugets: **Microsoft.BCL.Build**, **Microsoft.BCL**, **Microsoft.Net.Http** y **Newtonsoft.Json**. De paso agreguemos el Nuget **MvvmLightLibs** para poder incluir comandos que sigan el patrón MVVM, que necesitaremos más adelante.
2. Creamos la carpeta **Classes** y dentro de esta la clase **Response**:

public class Response

{

public bool IsSuccess { get; set; }

public string Message { get; set; }

public object Result { get; set; }

}

1. Creamos la carpeta **Services** y dentro de esta la clase **ApiService**, e implementamos un método genérico que nos sirva para obtener cualquier lista de objetos de un servicio Rest full:

public class ApiService

{

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

{

try

{

var client = new HttpClient();

client.BaseAddress = new Uri(urlBase);

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

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, 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, 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}/{2}", servicePrefix, controller, model.GetHashCode());

var response = await client.PutAsync(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 updated OK",

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, T model)

{

try

{

var client = new HttpClient();

client.BaseAddress = new Uri(urlBase);

var url = string.Format("{0}{1}/{2}", servicePrefix, controller, model.GetHashCode());

var response = await client.DeleteAsync(url);

if (!response.IsSuccessStatusCode)

{

return new Response

{

IsSuccess = false,

Message = response.StatusCode.ToString(),

};

}

return new Response

{

IsSuccess = true,

Message = "Record deleted OK",

};

}

catch (Exception ex)

{

return new Response

{

IsSuccess = false,

Message = ex.Message,

};

}

}

}

1. Creamos la carpeta **Models** y dentro de esta la clase **Contact**, la misma que usamos en el backend pero sin los decorados y agregamos las propiedades de **ImageFullPath** y **FullName**, también hacemos un **override** al método **GetHashCode** que lo vamos a necesitar más adelante (no olvides agregar una imagen comodín para los usuarios que no tengan foto).

public class Contact

{

public int ContactId { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Image { get; set; }

public string EmailAddress { get; set; }

public string PhoneNumber { get; set; }

public string FullName { get { return string.Format("{0} {1}", FirstName, LastName); } }

public string ImageFullPath

{

get

{

if (string.IsNullOrEmpty(Image))

{

return "contact.png";

}

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

}

}

public override int GetHashCode()

{

return ContactId;

}

}

1. En la carpeta **ViewModels** creamos la clase **ContactItemViewModel**, esto lo hacemos para mantener el modelo “puro” y seguir los lineamientos del patrón MVVM, más adelante agregaremos una líneas de código a esta clase:

public class ContactItemViewModel : Contact

{

}

1. Creemos el **DialogService**:

public class DialogService

{

public async Task ShowMessage(string title, string message)

{

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

}

public async Task<bool> ShowConfirm(string title, string message)

{

return await App.Current.MainPage.DisplayAlert(title, message, "Yes", "No");

}

}

1. Modificamos nuestro **MainViewModel** para que consuma la lista de contactos del servicio:

public class MainViewModel

{

#region Attributes

private ApiService apiService;

private DialogService dialogService;

#endregion

#region Properties

public ObservableCollection<ContactItemViewModel> Contacts { get; set; }

#endregion

#region Constructors

public MainViewModel()

{

apiService = new ApiService();

dialogService = new DialogService();

Contacts = new ObservableCollection<ContactItemViewModel>();

LoadContacts();

}

#endregion

#region Methods

private async void LoadContacts()

{

var response = await apiService.Get<Contact>("http://contactsbackprep.azurewebsites.net/", "/api", "/Contacts");

if (!response.IsSuccess)

{

await dialogService.ShowMessage("Error", response.Message);

return;

}

ReloadContacts((List<Contact>)response.Result);

}

private void ReloadContacts(List<Contact> contacts)

{

Contacts.Clear();

foreach (var contact in contacts.OrderBy(c => c.FirstName).ThenBy(c => c.LastName))

{

Contacts.Add(new ContactItemViewModel

{

ContactId = contact.ContactId,

EmailAddress = contact.EmailAddress,

FirstName = contact.FirstName,

Image = contact.Image,

LastName = contact.LastName,

PhoneNumber = contact.PhoneNumber,

});

}

}

#endregion

}

1. Modificamos nuestra **ContactsPage** para que pinte la lista de contactos:

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

x:Class="ContactsAppPrep.Pages.ContactsPage"

Title="Contacts"

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

<ListView

ItemsSource="{Binding Contacts}"

HasUnevenRows="True">

<ListView.ItemTemplate>

<DataTemplate>

<ViewCell>

<Grid Padding="8">

<Grid.ColumnDefinitions>

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

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

</Grid.ColumnDefinitions>

<controls:CircleImage

Source="{Binding ImageFullPath}"

Aspect="AspectFill"

WidthRequest="80"

HeightRequest="80">

</controls:CircleImage>

<StackLayout Grid.Column="1">

<Label

FontSize="Medium"

FontAttributes="Bold"

VerticalOptions="Center"

Text="{Binding FullName}">

</Label>

<Label

VerticalOptions="Center"

Text="{Binding PhoneNumber, StringFormat='Phone number: {0}'}">

</Label>

<Label

VerticalOptions="Center"

Text="{Binding EmailAddress, StringFormat='Email address: {0}'}">

</Label>

</StackLayout>

</Grid>

</ViewCell>

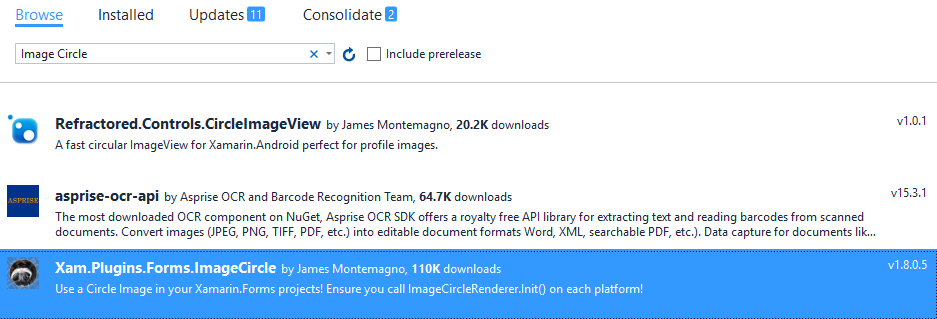
</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</ContentPage>

1. Probemos.
2. Es una tendencia ver las imágenes en un círculo, para tal fin agregamos el Nuget: (Image Circle):



1. Cambiamos nuestra ContactsPage:

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

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

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

x:Class="ContactsAppPrep.Pages.ContactsPage"

Title="Contacts"

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

<controls:CircleImage

Source="{Binding ImageFullPath}"

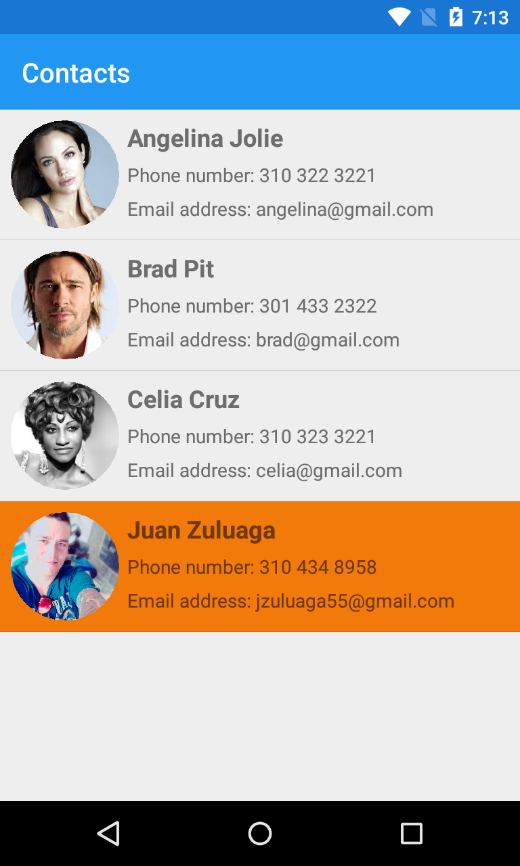
Aspect="AspectFill"

WidthRequest="80"

HeightRequest="80">

</controls:CircleImage>

1. Probemos



1. Agreguemos un ícono para adicionar contactos (+), recomiendo la página: <https://romannurik.github.io/AndroidAssetStudio/>
2. Creamos la opción de agregar en nuestra barra de herramientas, del **ContactsPage**:

</ListView>

<ContentPage.ToolbarItems>

<ToolbarItem Icon="ic\_action\_add\_circle.png" Command="{Binding AddContactCommand}"/>

</ContentPage.ToolbarItems>

</ContentPage>

1. Creamos nuestra página **NewContactPage**, como inicialmente solo lo vamos a poner a navegar ahí, dejémosla temporalmente con este código:

<?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="ContactsAppPrep.Pages.NewContactPage"

Title="New Contact"

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

<Label

Text="New Contact Page">

</Label>

</ContentPage>

1. Creamos el **NavigationService** en la carpeta de **Services**:

public class NavigationService

{

public async Task Navigate(string pageName)

{

switch (pageName)

{

case "NewContactPage":

await App.Current.MainPage.Navigation.PushAsync(new NewContactPage());

break;

default:

break;

}

}

public async Task Back()

{

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

}

}

1. Ahora implementemos el comando en la **MainViewModel**, primero creamos el objeto del servicio de navegación

private NavigationService navigationService;

…

navigationService = new NavigationService();

…

#region Commands

public ICommand AddContactCommand { get { return new RelayCommand(AddContact); } }

private async void AddContact()

{

await navigationService.Navigate("NewContactPage");

}

#endregion

1. Para evitar que nuestro **MainViewModel** crezca mucho, vamos a crear otro view model llamado **NewContactViewModel**:

public class NewContactViewModel

{

}

1. En nuestro **MainViewModel** vamos a crear un objeto del nuevo View Model:

public NewContactViewModel NewContact { get; set; }

1. Y modificamos el comando para que antes de navegar a la página cree el objeto view model que vamos a bindar en la página:

private async void AddContact()

{

NewContact = new NewContactViewModel();

await navigationService.Navigate("NewContactPage");

}

1. Modifiquemos nuestra **NewContactPage**:

<?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="ContactsAppPrep.Pages.NewContactPage"

Title="New Contact"

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

<ScrollView

BindingContext="{Binding NewContact}">

<StackLayout

Padding="8">

<Label

Text="First name">

</Label>

<Entry

Placeholder="Enter first name..."

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

</Entry>

<Label

Text="Last name">

</Label>

<Entry

Placeholder="Enter last name..."

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

</Entry>

<Label

Text="Email address">

</Label>

<Entry

Placeholder="Enter email address..."

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

</Entry>

<Label

Text="Phone number">

</Label>

<Entry

Placeholder="Enter phone number..."

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

</Entry>

<ActivityIndicator

IsRunning="{Binding IsRunning}">

</ActivityIndicator>

<StackLayout

Orientation="Horizontal">

<Button

HorizontalOptions="FillAndExpand"

Command="{Binding TakePictureCommand}"

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

BackgroundColor="Orange"

TextColor="White"

Text="Take Picture">

</Button>

<Button

HorizontalOptions="FillAndExpand"

Command="{Binding NewContactCommand}"

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

BackgroundColor="Navy"

TextColor="White"

Text="New Contact">

</Button>

</StackLayout>

</StackLayout>

</ScrollView>

</ContentPage>

1. Modifiquemos nuestro **NewContactViewModel**:

public class NewContactViewModel : Contact, INotifyPropertyChanged

{

#region Attributes

private DialogService dialogService;

private ApiService apiService;

private NavigationService navigationService;

private bool isRunning;

private bool isEnabled;

#endregion

#region Properties

public bool IsRunning

{

set

{

if (isRunning != value)

{

isRunning = value;

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

}

}

get

{

return isRunning;

}

}

public bool IsEnabled

{

set

{

if (isEnabled != value)

{

isEnabled = value;

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

}

}

get

{

return isEnabled;

}

}

#endregion

#region Events

public event PropertyChangedEventHandler PropertyChanged;

#endregion

#region Constructors

public NewContactViewModel()

{

dialogService = new DialogService();

apiService = new ApiService();

navigationService = new NavigationService();

IsEnabled = true;

}

#endregion

#region Commands

public ICommand NewContactCommand { get { return new RelayCommand(NewContact); } }

private async void NewContact()

{

if (string.IsNullOrEmpty(FirstName))

{

await dialogService.ShowMessage("Error", "You must enter a first name");

return;

}

if (string.IsNullOrEmpty(LastName))

{

await dialogService.ShowMessage("Error", "You must enter a last name");

return;

}

IsRunning = true;

IsEnabled = false;

var response = await apiService.Post("http://contactsbackprep.azurewebsites.net", "/api", "/Contacts", this);

IsRunning = false;

IsEnabled = true;

if (!response.IsSuccess)

{

await dialogService.ShowMessage("Error", response.Message);

return;

}

await navigationService.Back();

}

#endregion

}

1. Ya podemos crear nuevos contactos, pero para que refresque automáticamente la lista vamos a modificar lo siguiente, primero pasemos la definición de la lista de flores al diccionario de recursos. Así queda el diccionario de recursos:

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

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

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

xmlns:infra="clr-namespace:ContactsAppPrep.Infrastructure;assembly=ContactsAppPrep"

x:Class="ContactsAppPrep.App">

<Application.Resources>

<ResourceDictionary>

<!-- Locator -->

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

<!-- Data Templates -->

<DataTemplate x:Key="ContactsDataTemplate">

<ViewCell>

<Grid Padding="8">

<Grid.ColumnDefinitions>

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

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

</Grid.ColumnDefinitions>

<controls:CircleImage

Source="{Binding ImageFullPath}"

Aspect="AspectFill"

WidthRequest="80"

HeightRequest="80">

</controls:CircleImage>

<StackLayout Grid.Column="1">

<Label

FontSize="Medium"

FontAttributes="Bold"

VerticalOptions="Center"

Text="{Binding FullName}">

</Label>

<Label

VerticalOptions="Center"

Text="{Binding PhoneNumber, StringFormat='Phone number: {0}'}">

</Label>

<Label

VerticalOptions="Center"

Text="{Binding EmailAddress, StringFormat='Email address: {0}'}">

</Label>

</StackLayout>

</Grid>

</ViewCell>

</DataTemplate>

</ResourceDictionary>

</Application.Resources>

</Application>

Y así queda nuestra página:

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

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

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

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

x:Class="ContactsAppPrep.Pages.ContactsPage"

Title="Contacts"

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

<ListView

ItemTemplate="{StaticResource ContactsDataTemplate}"

ItemsSource="{Binding Contacts}"

HasUnevenRows="True"

IsPullToRefreshEnabled="True"

RefreshCommand="{Binding RefreshCommand}"

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

</ListView>

<ContentPage.ToolbarItems>

<ToolbarItem Icon="ic\_action\_add\_circle.png" Command="{Binding AddContactCommand}"/>

</ContentPage.ToolbarItems>

</ContentPage>

1. Implementemos un singleton en nuestro **MainViewModel**:

#region Singleton

private static MainViewModel instance;

public static MainViewModel GetInstance()

{

if (instance == null)

{

instance = new MainViewModel();

}

return instance;

}

#endregion

Y se instancia en el constructor:

instance = this;

1. Ahora implementamos la propiedad y el nuevo comando de refresco. Cambie el **MainViewModel** para que implemente la interfaz **INotifyPropertyChanged**

public class MainViewModel : INotifyPropertyChanged

Y cree el evento de la interfaz:

#region Events

public event PropertyChangedEventHandler PropertyChanged;

#endregion

Creamos en el **MainViewModel** el atributo y la propiedad del refresh:

private bool isRefreshing;

Y la propiedad:

public bool IsRefreshing

{

set

{

if (isRefreshing != value)

{

isRefreshing = value;

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

}

}

get

{

return isRefreshing;

}

}

Y obvio, crear el comando del Refresh:

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

private void Refresh()

{

IsRefreshing = true;

LoadContacts();

IsRefreshing = false;

}

Con esto debe refrescar al bajar el **ListView**, probemos. Pero para que refresque automáticamente debemos agregar estas líneas al code behind del **ContactsPage**:

public ContactsPage()

{

InitializeComponent();

var mainViewModel = MainViewModel.GetInstance();

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

{

mainViewModel.RefreshCommand.Execute(this);

};

}

Y debemos de eliminar el llamado del LoadContacts() en el constructor para evitar que se llame doblemente. Probemos si ya refresca.

1. Ahora cambiemos el backend para que pueda recibir las fotos tomadas desde el celular como un array de bytes. Primero creamos la clase **ContactRequest**:

[NotMapped]

public class ContactRequest : Contact

{

public byte[] ImageArray { get; set; }

}

1. Sobre cargar el método **UploadPhoto** en el **FilesHelper** para que reciba un **MemoryStream** como parámetro:

public static bool UploadPhoto(MemoryStream stream, string folder, string name)

{

try

{

stream.Position = 0;

var path = Path.Combine(HttpContext.Current.Server.MapPath(folder), name);

File.WriteAllBytes(path, stream.ToArray());

}

catch

{

return false;

}

return true;

}

1. Modifiquemos los métodos **POST** y **PUT** del API **ContactsController** para que puedan almacenar la imagen:

[ResponseType(typeof(Contact))]

public IHttpActionResult PostContact(ContactRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

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

{

var stream = new MemoryStream(request.ImageArray);

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

var file = string.Format("{0}.jpg", guid);

var folder = "~/Content/Images";

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

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

if (response)

{

request.Image = fullPath;

}

}

var contact = ToContact(request);

db.Contacts.Add(contact);

db.SaveChanges();

return CreatedAtRoute("DefaultApi", new { id = contact.ContactId }, contact);

}

Ahora el PUT:

[ResponseType(typeof(void))]

public IHttpActionResult PutContact(int id, ContactRequest request)

{

if (!ModelState.IsValid)

{

return BadRequest(ModelState);

}

if (id != request.ContactId)

{

return BadRequest();

}

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

{

var stream = new MemoryStream(request.ImageArray);

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

var file = string.Format("{0}.jpg", guid);

var folder = "~/Content/Images";

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

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

if (response)

{

request.Image = fullPath;

}

}

var contact = ToContact(request);

db.Entry(contact).State = EntityState.Modified;

try

{

db.SaveChanges();

}

catch (DbUpdateConcurrencyException)

{

if (!ContactExists(id))

{

return NotFound();

}

else

{

throw;

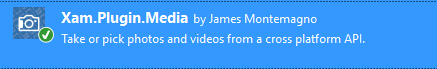
}

}

return StatusCode(HttpStatusCode.NoContent);

}

1. Publicamos los cambios del nuevo backend.
2. Instalar el Nuget: **Xam.Plugin.Media** en toda la solución:



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

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

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

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

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

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

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

<application android:label="ContactsAppPrep.Android"></application>

</manifest>

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

</array>

<key>UILaunchStoryboardName</key>

<string>LaunchScreen</string>

<key>NSCameraUsageDescription</key>

<key>NSPhotoLibraryUsageDescription</key>

</dict>

</plist>

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

}

Plugin.Media.MediaImplementation.OnFilesPicked(e);

base.OnActivated(e);

// Ensure the current window is active

Window.Current.Activate();

}

1. Modifiquemos nuestra **NewContactPage**:

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

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

x:Class="ContactsAppPrep.Pages.NewContactPage"

…

</Entry>

<controls:CircleImage

Source="{Binding ImageSource}"

Aspect="AspectFill">

</controls:CircleImage>

<ActivityIndicator

1. Dentro de la carpeta **Classes** crear la clase **FilesHelper**:

public class FilesHelper

{

public static byte[] ReadFully(Stream input)

{

using (MemoryStream ms = new MemoryStream())

{

input.CopyTo(ms);

return ms.ToArray();

}

}

}

1. Agregar al **NewContactViewModel** estos atributos:

private ImageSource imageSource;

private MediaFile file;

Y la propiedad:

public ImageSource ImageSource

{

set

{

if (imageSource != value)

{

imageSource = value;

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

}

}

get

{

return imageSource;

}

}

1. Agregar el comando en **NewContactViewModel**:

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

private async void TakePicture()

{

await CrossMedia.Current.Initialize();

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

{

await dialogService.ShowMessage("No Camera", ":( No camera available.");

}

file = await CrossMedia.Current.TakePhotoAsync(new StoreCameraMediaOptions

{

Directory = "Sample",

Name = "test.jpg",

PhotoSize = PhotoSize.Small,

});

IsRunning = true;

if (file != null)

{

ImageSource = ImageSource.FromStream(() =>

{

var stream = file.GetStream();

return stream;

});

}

IsRunning = false;

}

1. Agregar la propiedad al modelo:

public byte[] ImageArray { get; set; }

1. Agregar esta líneas en el **NewContactViewModel**:

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

file.Dispose();

var contact = new Contact

{

EmailAddress = EmailAddress,

FirstName = FirstName,

ImageArray = imageArray,

LastName = LastName,

PhoneNumber = PhoneNumber,

};

IsRunning = true;

IsEnabled = false;

var response = await apiService.Post("http://contactsbackprep.azurewebsites.net", "/api", "/Contacts", contact);

IsRunning = false;

IsEnabled = true;

1. Probar.
2. Ahora procedemos a crear la página para modificar y borrar contactos. Creemos nuestra **EditContactPage**:

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

x:Class="ContactsAppPrep.Pages.EditContactPage"

Title="Edit Contact"

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

<ScrollView

BindingContext="{Binding EditContact}">

<StackLayout

Padding="8">

<Label

Text="First name">

</Label>

<Entry

Placeholder="Enter first name..."

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

</Entry>

<Label

Text="Last name">

</Label>

<Entry

Placeholder="Enter last name..."

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

</Entry>

<Label

Text="Email address">

</Label>

<Entry

Placeholder="Enter email address..."

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

</Entry>

<Label

Text="Phone number">

</Label>

<Entry

Placeholder="Enter phone number..."

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

</Entry>

<controls:CircleImage

Source="{Binding ImageFullPath}"

Aspect="AspectFill"

WidthRequest="300"

HeightRequest="300">

</controls:CircleImage>

<ActivityIndicator

IsRunning="{Binding IsRunning}">

</ActivityIndicator>

<StackLayout

Orientation="Horizontal">

<Button

HorizontalOptions="FillAndExpand"

Command="{Binding TakePictureCommand}"

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

BackgroundColor="Orange"

TextColor="White"

Text="Take Picture">

</Button>

<Button

HorizontalOptions="FillAndExpand"

Command="{Binding SaveContactCommand}"

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

BackgroundColor="Navy"

TextColor="White"

Text="Save">

</Button>

<Button

HorizontalOptions="FillAndExpand"

Command="{Binding DeleteContactCommand}"

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

BackgroundColor="Red"

TextColor="White"

Text="Delete">

</Button>

</StackLayout>

</StackLayout>

</ScrollView>

</ContentPage>

1. Y creemos su **EditContactViewModel**:

public class EditContactViewModel : Contact, INotifyPropertyChanged

{

#region Attributes

private DialogService dialogService;

private ApiService apiService;

private NavigationService navigationService;

private bool isRunning;

private bool isEnabled;

private ImageSource imageSource;

private MediaFile file;

#endregion

#region Properties

public bool IsRunning

{

set

{

if (isRunning != value)

{

isRunning = value;

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

}

}

get

{

return isRunning;

}

}

public bool IsEnabled

{

set

{

if (isEnabled != value)

{

isEnabled = value;

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

}

}

get

{

return isEnabled;

}

}

public ImageSource ImageSource

{

set

{

if (imageSource != value)

{

imageSource = value;

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

}

}

get

{

return imageSource;

}

}

#endregion

#region Events

public event PropertyChangedEventHandler PropertyChanged;

#endregion

#region Constructors

public EditContactViewModel(Contact contact)

{

dialogService = new DialogService();

apiService = new ApiService();

navigationService = new NavigationService();

ContactId = contact.ContactId;

FirstName = contact.FirstName;

LastName = contact.FirstName;

Image = contact.FirstName;

EmailAddress = contact.FirstName;

PhoneNumber = contact.FirstName;

IsEnabled = true;

}

#endregion

#region Commands

#endregion

}

1. Y referenciemos un objeto de esta nueva clase en el **MainViewModel**:

public EditContactViewModel EditContact { get; set; }

1. Modifiquemos nuestro servicio de navegación para que pueda ir a la nueva página:

switch (pageName)

{

case "EditContactPage":

await App.Current.MainPage.Navigation.PushAsync(new EditContactPage());

break;

case "NewContactPage":

await App.Current.MainPage.Navigation.PushAsync(new NewContactPage());

break;

default:

break;

}

1. Ahora vamos a colocar un **Gesture Recognizer** a nuestro grid para que identifique cuando hagan un toque en algún elemento de la cuadricula, para eso editamos el **App.xaml** que es donde tenemos la definición de nuestra cuadricula:

<Grid Padding="8">

<Grid.GestureRecognizers>

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

</Grid.GestureRecognizers>

<Grid.ColumnDefinitions>

1. En el contexto de la cuadricula, es decir, el **ContactItemViewModel**, agregamos el comando para que navegue a la nueva página:

public class ContactItemViewModel : Contact

{

#region Attributes

private NavigationService navigationService;

#endregion

#region Constructors

public ContactItemViewModel()

{

navigationService = new NavigationService();

}

#endregion

#region Commands

public ICommand EditContactCommand { get { return new RelayCommand(EditContact); } }

private async void EditContact()

{

var mainViewModel = MainViewModel.GetInstance();

mainViewModel.EditContact = new EditContactViewModel(this);

await navigationService.Navigate("EditContactPage");

}

#endregion

}

1. Probemos que navegue bien a la pantalla de edición.
2. Luego implementemos los comandos de los botones:

public ICommand DeleteContactCommand { get { return new RelayCommand(DeleteContact); } }

private async void DeleteContact()

{

var answer = await dialogService.ShowConfirm("Confirm", "Are you sure to delete this record?");

if (!answer)

{

return;

}

var contact = new Contact

{

EmailAddress = EmailAddress,

FirstName = FirstName,

LastName = LastName,

PhoneNumber = PhoneNumber,

Image = Image,

ContactId = ContactId,

};

IsRunning = true;

IsEnabled = false;

var response = await apiService.Delete("http://contactsbackprep.azurewebsites.net", "/api", "/Contacts", contact);

IsRunning = false;

IsEnabled = true;

if (!response.IsSuccess)

{

await dialogService.ShowMessage("Error", response.Message);

return;

}

await navigationService.Back();

}

public ICommand SaveContactCommand { get { return new RelayCommand(SaveContact); } }

private async void SaveContact()

{

if (string.IsNullOrEmpty(FirstName))

{

await dialogService.ShowMessage("Error", "You must enter a first name");

return;

}

if (string.IsNullOrEmpty(LastName))

{

await dialogService.ShowMessage("Error", "You must enter a last name");

return;

}

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

file.Dispose();

var contact = new Contact

{

EmailAddress = EmailAddress,

FirstName = FirstName,

ImageArray = imageArray,

LastName = LastName,

PhoneNumber = PhoneNumber,

Image = Image,

ContactId = ContactId,

};

IsRunning = true;

IsEnabled = false;

var response = await apiService.Put("http://contactsbackprep.azurewebsites.net", "/api", "/Contacts", contact);

IsRunning = false;

IsEnabled = true;

if (!response.IsSuccess)

{

await dialogService.ShowMessage("Error", response.Message);

return;

}

await navigationService.Back();

}

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

private async void TakePicture()

{

await CrossMedia.Current.Initialize();

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

{

await dialogService.ShowMessage("No Camera", ":( No camera available.");

}

file = await CrossMedia.Current.TakePhotoAsync(new StoreCameraMediaOptions

{

Directory = "Sample",

Name = "test.jpg",

PhotoSize = PhotoSize.Small,

});

IsRunning = true;

if (file != null)

{

ImageSource = ImageSource.FromStream(() =>

{

var stream = file.GetStream();

return stream;

});

}

IsRunning = false;

}

1. ¡Hemos acabado!