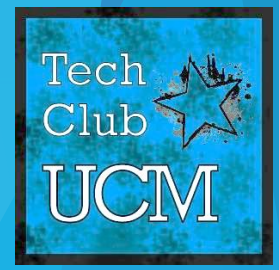
Workshop Xamarin is comming

Crea tu propia aplicación móvil *cross-platform*



¿Quién soy?

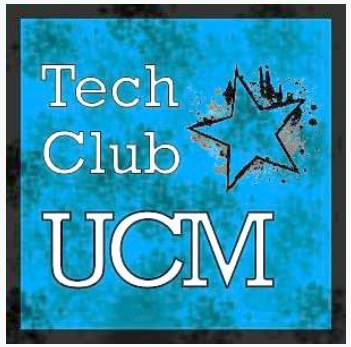
José Miguel Tajuelo Garrigós

Webpage: @JMTG888

Estudiante de Ingenería Informática en UCM

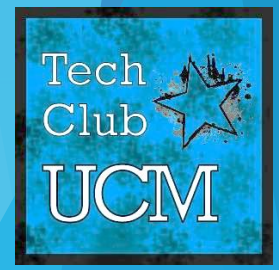
Microsoft Student Partner

Miembro de TechClubUCM



Prerrequisitos

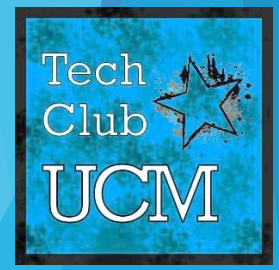
- Visual Studio: https://www.visualstudio.com/es/
- Subscripción Azure con Dreamspark:
  - https://www.dreamspark.com/
  - https://azure.microsoft.com/



Material

https://github.com/xamarin/dev-days-labs

- /SlideDecks
  - Introddución
- /HandsOnLab
  - Material de laboratorio, incluyendo versión inicial y final
- /Demos
  - Ejemplos

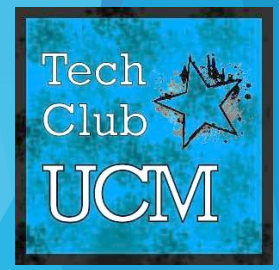


Importando la solución

La solución inicial se encuentra en la carpeta Start (dentro de HandsOnLab)

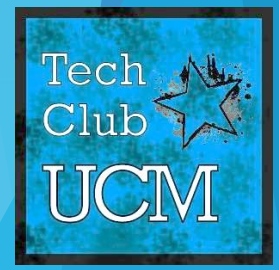
- Abrir/Importar la solución: Start/DevDaysSpeakers.sln

Veremos una solución 4 proyectos, cada uno perteneciente a una plataforma



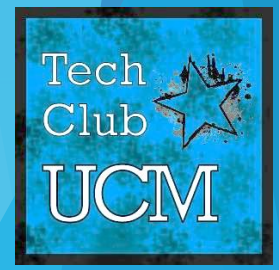
Importando la solución

- DevDaysSpeakers
- DevDaysSpeakers.Droid
- DevDaysSpeakers.iOS
- DevDaysSpeakers.UWP

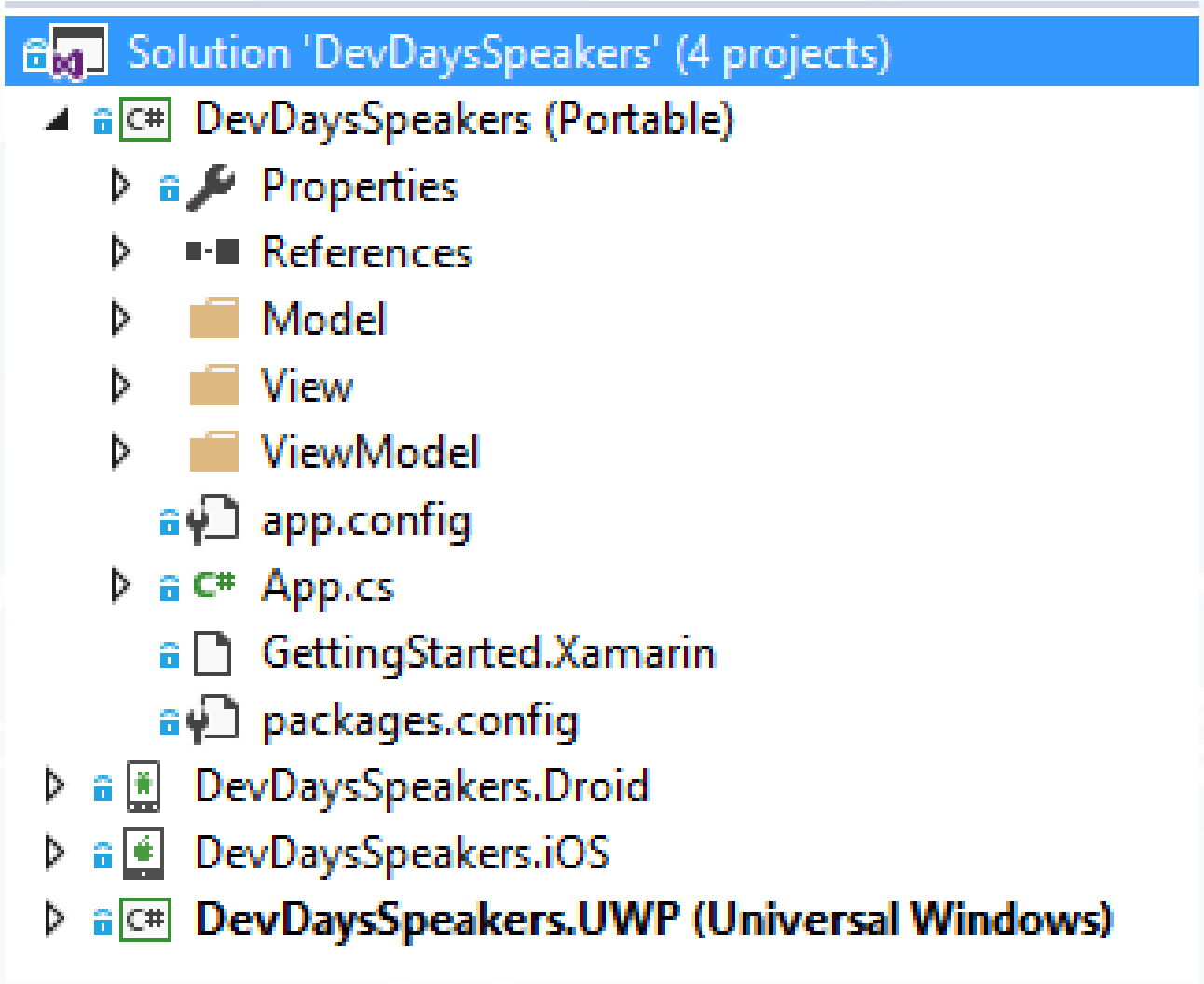


Importando la solución

- DevDaysSpeakers
  - Contiene todo el código compartido
- DevDaysSpeakers.Droid
  - Aplicación de Xamarin.Android
- DevDaysSpeakers.iOS
  - Aplicación de Xamarin.Android
- DevDaysSpeakers.UWP
  - Aplicación de Windows 10 UWP (Universal Application Platform)



Importando la solución

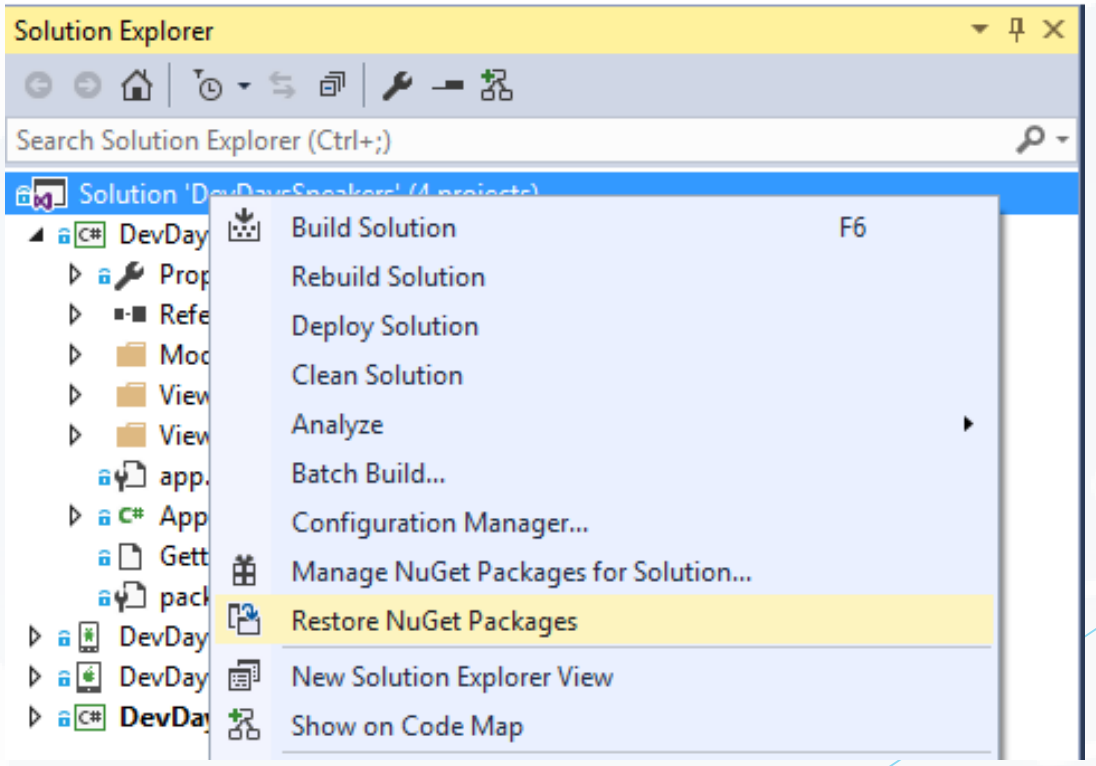


Tech Club UCM

Importando la solución: NuGet

Sistema de gestión de paquetes FOSS (Free and Open-Source Software)

- Extensión de VS



Tech Club UCM

Modelo

- Abrir DevDaysSpeakers/Moder/Speaker.cs
- Añadir las siguientes propiedades:

```
public string Id { get; set; }
public string Name { get; set; }
public string Description { get; set; }
public string Website { get; set; }
public string Title { get; set; }
public string Avatar { get; set; }
```

Tech Club UCM

Modelo

Speaker.cs

Archivos varios — DevDaysSpeakers.Model.Speaker — AzureVe

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DevDaysSpeakers.Model
{
    -referencias
    public class Speaker
    {
        //Add speaker attributes here


        //Azure information for version
        [Microsoft.WindowsAzure.MobileServices.Version]
        -referencias
        public string AzureVersion { get; set; }
    }
}
```

Tech Club UCM

Modelo

Speaker.cs*

Archivos varios | DevDaysSpeakers.Model.Speaker | Website

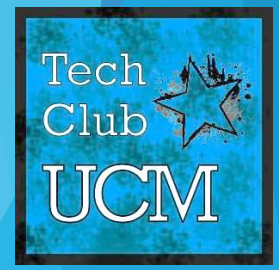
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DevDaysSpeakers.Model
{
    -referencias
    public class Speaker
    {
        //Add speaker attributes here
        -referencias
        public string Id { get; set; }
        -referencias
        public string Name { get; set; }
        -referencias
        public string Description { get; set; }
        -referencias
        public string Website { get; set; }
        -referencias
        public string Title { get; set; }
        -referencias
        public string Avatar { get; set; }

        //Azure information for version
        [Microsoft.WindowsAzure.MobileServices.Version]
        -referencias
        public string AzureVersion { get; set; }
    }
}
```

Tech Club UCM

View Model

- SpeakersViewModel.cs proporciona toda la funcionalidad (de como mostrar la información) a nuestra vista principal en Xamarin.Forms

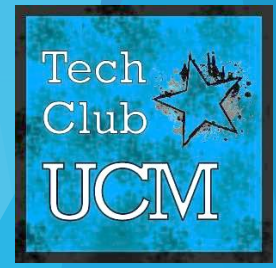
- Además, contiene un flag que indica si estamos recibiendo información en una tarea en segundo plano



View Model: Notify de cambio

- La implementación de la interfaz INotifyPropertyChanged notifica a la vista de cambios en el modelo

- Además, contiene un flag que indica si estamos recibiendo información en una tarea en segundo plano



View Model: Notify de cambio

- Añadimos dicha interfaz a la clase SpeakersViewModel

```
namespace DevDaysSpeakers.ViewModel
{
    -referencias
    public class SpeakersViewModel
    {
    }
}
```

Tech Club UCM

View Model: Notify de cambio

- Añadimos dicha interfaz a la clase SpeakersViewModel

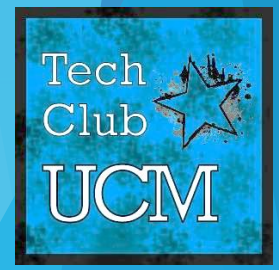
```
namespace DevDaysSpeakers.ViewModel
{
    -referencias
    public class SpeakersViewModel : INotifyPropertyChanged
    {
    }
}
```

View Model: Notify de cambio

- Implementamos automáticamente la interfaz

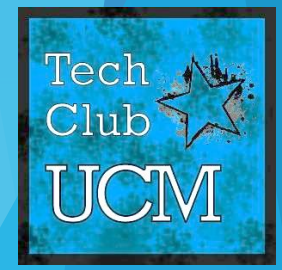
- Para hacerlo manualmente podemos añadir la siguente línea de código:

```
public event PropertyChangedEventHandler PropertyChanged;
```



View Model: Notify de cambio

- Añadimos el método ayudante OnPropertyChanged que lanzara el evento PropertyChanged
  - En Visual Studio 2015 podemos hacerlo de una forma mas concisa usando una función anónima



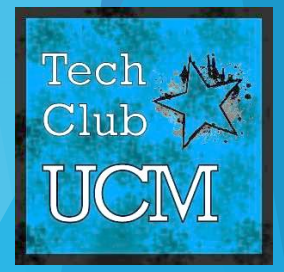
View Model: Notify de cambio

C# 6 (Visual Studio 2015 or Xamarin Studio on Mac)

```
private void OnPropertyChanged([CallerMemberName] string name = null) =>
PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
```

C# 5 (Visual Studio 2012 or 2013)

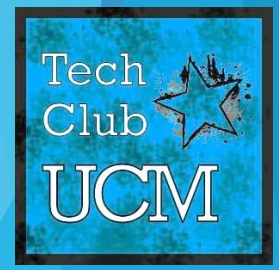
```
private void OnPropertyChanged([CallerMemberName] string name = null)
{
    var changed = PropertyChanged;
        if (changed == null)
            return;

        changed.Invoke(this, new PropertyChangedEventArgs(name));
}
```



View Model: busy

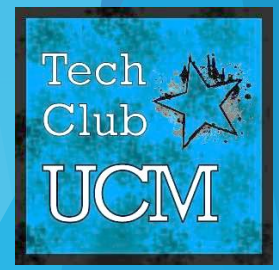
- Añadimos un campo busy (junto con sus getter y setter) para evitar realizar operaciones por duplicado mientras la vista esta ocupada

```
private bool busy;

public bool IsBusy
{
    get { return busy; }
    set
    {
        busy = value;
        OnPropertyChanged();
    }
}
```



View Model: Collection de Speakers

- Usaremos una ObservableCollection que será vaciada y después llenada de speakers.
- La razón por la que usaremos ObservableCollection es porque ya tiene integrado el evento CollectionChanged.
- Procedemos a declarar la auto-propiedad:

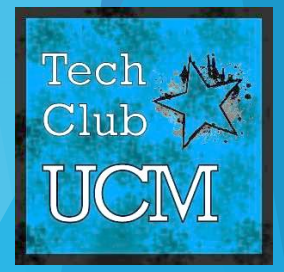


View Model: Collection de Speakers

```
public ObservableCollection<Speaker> Speakers { get; set; }
```

- Y creamos una instancia dentro del constructor

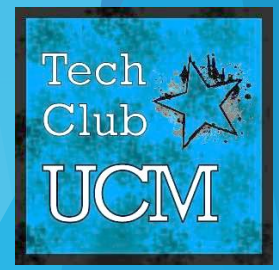
```
public SpeakersViewModel()
{
    Speakers = new ObservableCollection<Speaker>();
}
```



View Model: método GetSpeakers

- Llegados a este punto crearemos el método GetSpeakers, que obtendrá la información de los speaker a través de una HTTP request

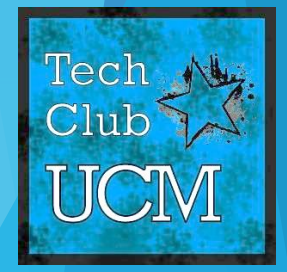
- No obstante, mas tarde podremos mejorarlo para que obtenga y sincronice la información con Azure



View Model: método GetSpeakers

- Primero creamos el método GetSpeakers como async Task
  - Nota: empleamos Task porque usara métodos Async

```
private async Task GetSpeakers()
{


}
```

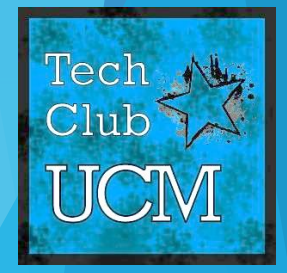


View Model: método GetSpeakers

- A continuación vamos a comprobar que no estamos obteniendo ya información

- Haremos uso de busy, visto anteriormente

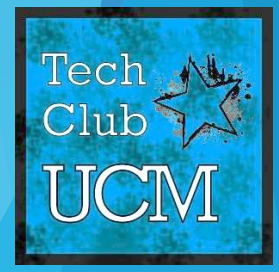
```
private async Task GetSpeakers()
{
    if(IsBusy)
        return;
}
```



View Model: método GetSpeakers

- Despues añadiremos un control de excepciones muy básico

```
private async Task GetSpeakers()
{
    if (IsBusy)
        return;

    Exception error = null;

    try { IsBusy = true; }

    catch (Exception ex) { error = ex; }

    finally { IsBusy = false; }
}
```

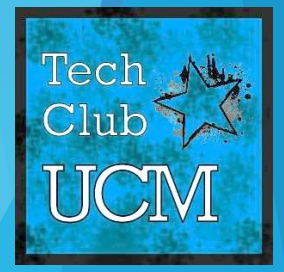


View Model: método GetSpeakers

- Nota: ponemos IsBusy a cierto antes de llamar al servidor y a falso cuando acabamos

- Ahora usamos HttpClient para obtener el json del servidor
  - Lo añadimos dentro del bloque try

```
using(var client = new HttpClient())
{
    //grab json from server
    var json = await client.GetStringAsync(
        http://demo4404797.mockable.io/speakers
    );
}
```



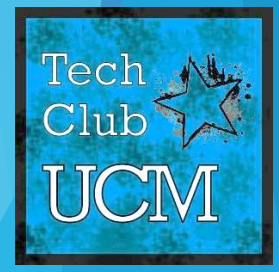
View Model: método GetSpeakers

- Añadimos dentro del método using 2 fragmentos de código
  - El primero para deserializar el json:

```
var items = JsonConvert.DeserializeObject<List<Speaker>>(json);
```

  - El segundo para vaciar y recargar los speakers:

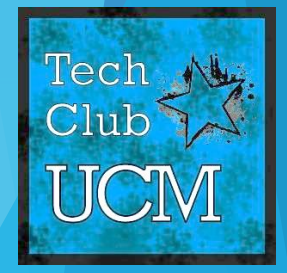
```
Speakers.Clear();
foreach (var item in items)
    Speakers.Add(item);
```



View Model: método GetSpeakers

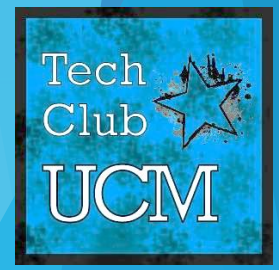
- Finalmente añadimos un alert en caso de que el catch falle:

```
if (error != null)
    await Application.Current.MainPage.DisplayAlert(
        "Error!", error.Message, "OK“
    );
```



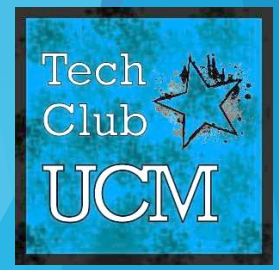
View Model: método GetSpeakers

```
private async Task GetSpeakers()
{
    if (IsBusy)
        return;

    Exception error = null;
    try
    {
        IsBusy = true;

        using(var client = new HttpClient())
        {
            //grab json from server
            var json = await client.GetStringAsync("http://demo4404797.mockable.io/speakers");

            //Deserialize json
            var items = JsonConvert.DeserializeObject<List<Speaker>>(json);

            //Load speakers into list
            Speakers.Clear();
            foreach (var item in items)
                Speakers.Add(item);
        }
    }
```



View Model: método GetSpeakers

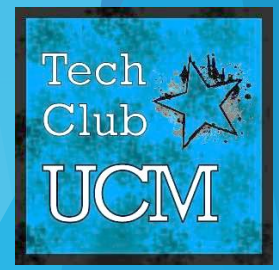
```
    catch (Exception ex)
    {
        Debug.WriteLine("Error: " + ex);
        error = ex;
    }
    finally
    {
        IsBusy = false;
    }

    if (error != null)
        await Application.Current.MainPage.DisplayAlert("Error!", error.Message, "OK");
}
```



View Model: Command GetSpeakers

- En vez de invocar el método GetSpeakers directamente, usaremos Command
  - Command contiene una interfaz (que conoce que método invocar) y una forma opcional de comprobar si está habilitado
  - Creamos el Command GetSpeakersCommand:

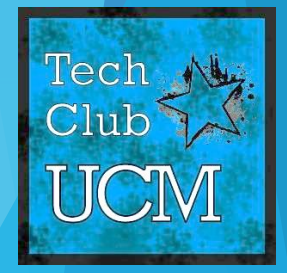
```
public Command GetSpeakersCommand { get; set; }
```



View Model: Command GetSpeakers

- Dentro del constructor de SpeakersViewModel creamos el Command pasándole dos métodos:
  - El que debe invocar cuando se ejecute
  - El que determina si esta habilitado
- Ambos pueden ser implementados usando Lamda expresiones:

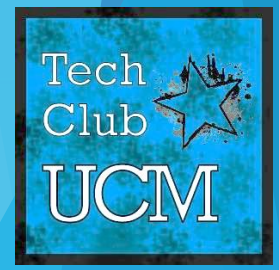
```
GetSpeakersCommand = new Command(
                    async () => await GetSpeakers(),
                    () => !IsBusy);
```



View Model: Command GetSpeakers

- Finalmente, tendremos que modificar IsBusy para que esta función pueda ser rehabilitada.
- Modificaremos pues el set de IsBusy asi:

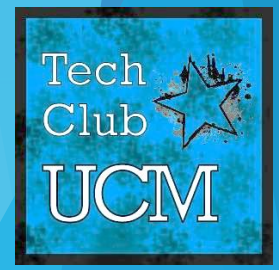
```
set
{
    busy = value;
    OnPropertyChanged(); //Update the can execute

}
```



View Model: Command GetSpeakers

- Finalmente, tendremos que modificar IsBusy para que esta función pueda ser rehabilitada.
- Modificaremos pues el set de IsBusy asi:

```
set
{
    busy = value;
    OnPropertyChanged(); //Update the can execute
    GetSpeakersCommand.ChangeCanExecute();
}
```

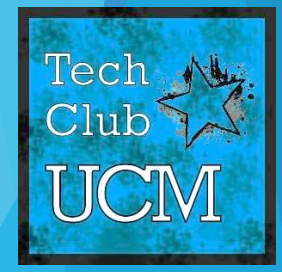


Interfaz de Usuario: SpeakersPage

- Finalmente, construiremos la Interfaz de usuario Xamarin.Forms en View/SpeakersPage.xaml

- Para la primera pagina usaremos unos cuantos controles apilados verticalmente.
  - Añadiremos lo siguiente entre los tags de ContentPage:

```
<StackLayout Spacing="0">

</StackLayout>
```



Interfaz de Usuario: SpeakersPage

SpeakersPage.xaml

```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="DevDaysSpeakers.View.SpeakersPage"
             Title="Speakers">

  <StackLayout Spacing="0">

  </StackLayout>

</ContentPage>
```

Tech Club UCM

Interfaz de Usuario: SpeakersPage

- Este StackLayout será el contenedor donde todos los controles hijos estarán.
  - Nota: dichos hijos no deberían tener espacio entre ellos

SpeakersPage.xaml

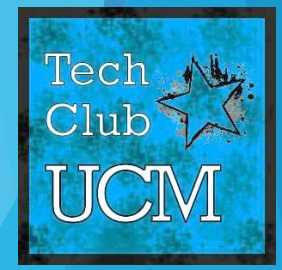
```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="DevDaysSpeakers.View.SpeakersPage"
             Title="Speakers">

  <StackLayout Spacing="0">

  </StackLayout>

</ContentPage>
```

Tech Club UCM

Interfaz de Usuario: SpeakersPage

- Ahora, añadiremos un Button asociado al método GetSpeakersCommand
  - El Command reemplazara al Handler de click

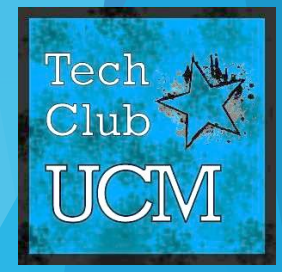
```
<Button Text="Sync Speakers" Command="{Binding GetSpeakersCommand}"/>
```



Interfaz de Usuario: SpeakersPage

- Debajo podemos mostrar una barra de carga mientras obtenemos la información del servidor.
  - Nota: usaremos IsBusy

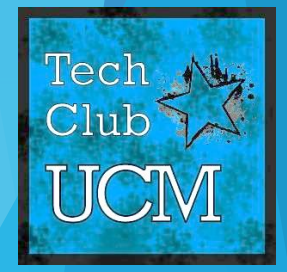
```
<ActivityIndicator IsRunning="{Binding IsBusy}" IsVisible="{Binding IsBusy}"/>
```



Interfaz de Usuario: SpeakersPage

- Después podemos añadir una ListView asociada a la colección de Speakers:
  - Usamos la propiedad x:Name=“” para poner nombre a cualquier control

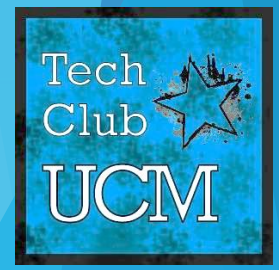
```
<ListView x:Name="ListViewSpeakers"
                ItemsSource="{Binding Speakers}">

    <!--Add ItemTemplate Here-->

</ListView>
```



Interfaz de Usuario: SpeakersPage

- Aun necesitamos especificar el aspecto que tendrá cada ítem, lo cual haremos usando un ItemTemplate que contenga un DataTemplate con una vista específica

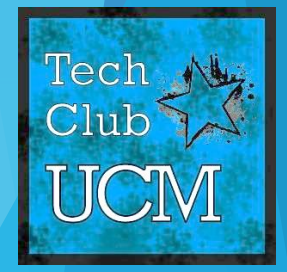
- Usaremos una de las Cells que nos ofrece Xamarin.Forms, ImageCell, que contiene una imagen y dos filas de texto



Interfaz de Usuario: SpeakersPage

- Reemplazar en el comentario anterior con el codigo:

```
<ListView.ItemTemplate>
    <DataTemplate>
        <ImageCell Text="{Binding Name}"
                   Detail="{Binding Title}"
                   ImageSource="{Binding Avatar}"/>
    </DataTemplate>
</ListView.ItemTemplate>
```



Interfaz de Usuario: SpeakersPage

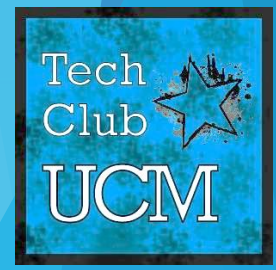
SpeakersPage.xaml

```
<?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="DevDaysSpeakers.View.SpeakersPage"
             Title="Speakers">

  <StackLayout Spacing="0">
      <Button Text="Sync Speakers" Command="{Binding GetSpeakersCommand}"/>
      <ActivityIndicator IsRunning="{Binding IsBusy}" IsVisible="{Binding IsBusy}"/>
      <ListView x:Name="ListViewSpeakers" ItemsSource="{Binding Speakers}">
        <ListView.ItemTemplate>
            <DataTemplate>
                <ImageCell Text="{Binding Name}"
                           Detail="{Binding Title}"
                           ImageSource="{Binding Avatar}"/>
            </DataTemplate>
        </ListView.ItemTemplate>
      </ListView>
  </StackLayout>

</ContentPage>
```

Tech Club UCM

Validación de App.cs (entry point)

- En App.cs se encuentra el punto de entrada (entry point) de la aplicación, el constructor de App()
- Simplemente crea el SpeakersPage y lo envuelve en una página de navegación (NavigationPage)



Validación de App.cs (entry point)

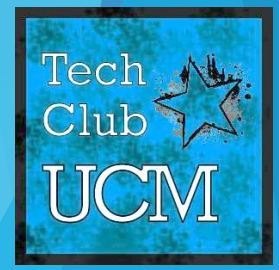
```
namespace DevDaysSpeakers
{
    -referencias
    public class App : Application
    {
        -referencias
        public App()
        {
            // The root page of your application
            var content = new SpeakersPage();

            MainPage = new NavigationPage(content);
        }

        -referencias
        protected override void OnStart()
        {
            // Handle when your app starts
        }

        -referencias
        protected override void OnSleep()
        {
            // Handle when your app sleeps
        }

        -referencias
        protected override void OnResume()
        {
            // Handle when your app resumes
        }
    }
}
```

Tech Club UCM

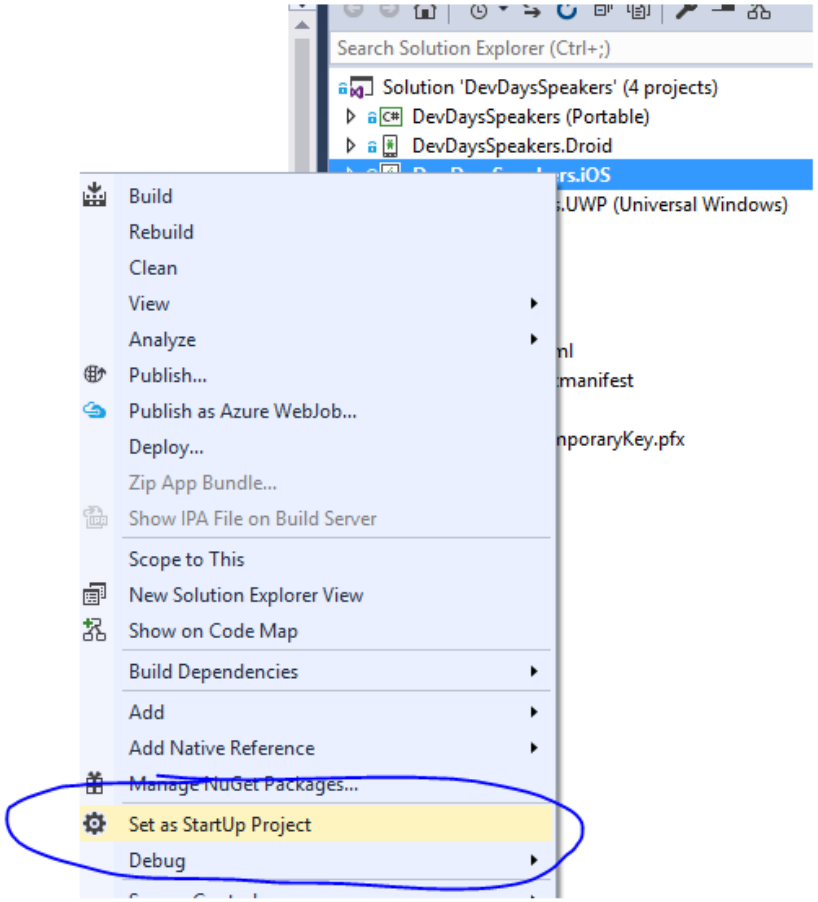
Ejecución de la aplicación

El procedimiento es el mismo para iOS, Android y UWP:

- Seleccionar el proyecto que se desea ejecutar
- Seleccionar la opción Establecer como proyecto de inicio(Set as StartUp Project) del menú contextual



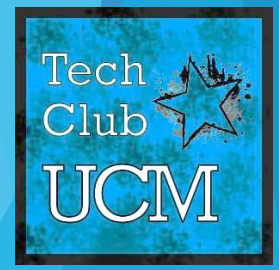
Ejecución de la aplicación



Tech Club UCM

Ejecución de la aplicación

- Seleccionar la plataforma de ejecución adecuadas para la plataforma que se desea ejecutar

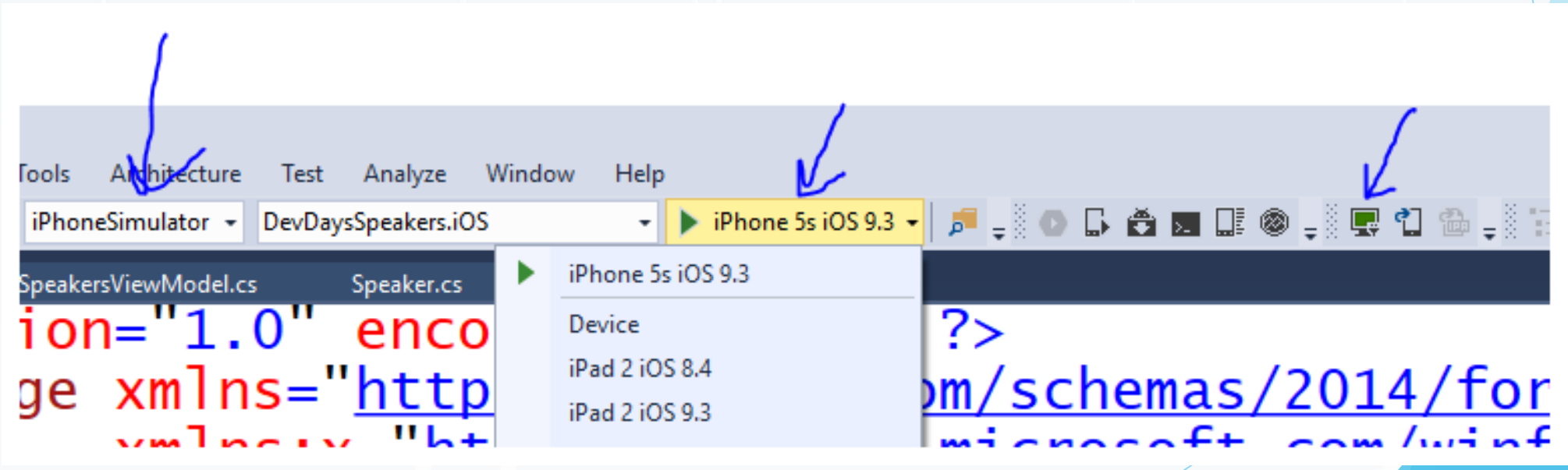


Ejecución de la aplicación

- Seleccionar la plataforma de ejecución adecuadas para la plataforma que se desea ejecutar

Ejecución de la aplicación

- Seleccionar la plataforma de ejecución adecuadas para la plataforma que se desea ejecutar
- Es decir, si eligió Android, no escoger un iPhone



Tech Club UCM

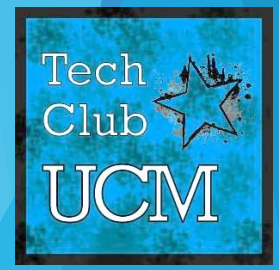
Ejecución de la aplicación

Tech Club UCM

Particularidades de iOS

Para ejecutar la aplicación en iOS sera necesario cumplir alguna de estas dos alternativas:

- Estar trabajando en un dispositivo macOS
  - Xamarin Studio
- Estar trabajando en un PC y conectado a un dispositivo macOS



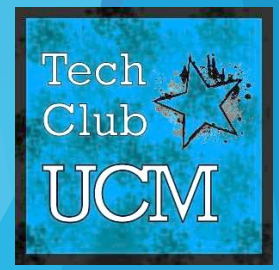
Particularidades de Android

Unos de los errores mas comunes son:

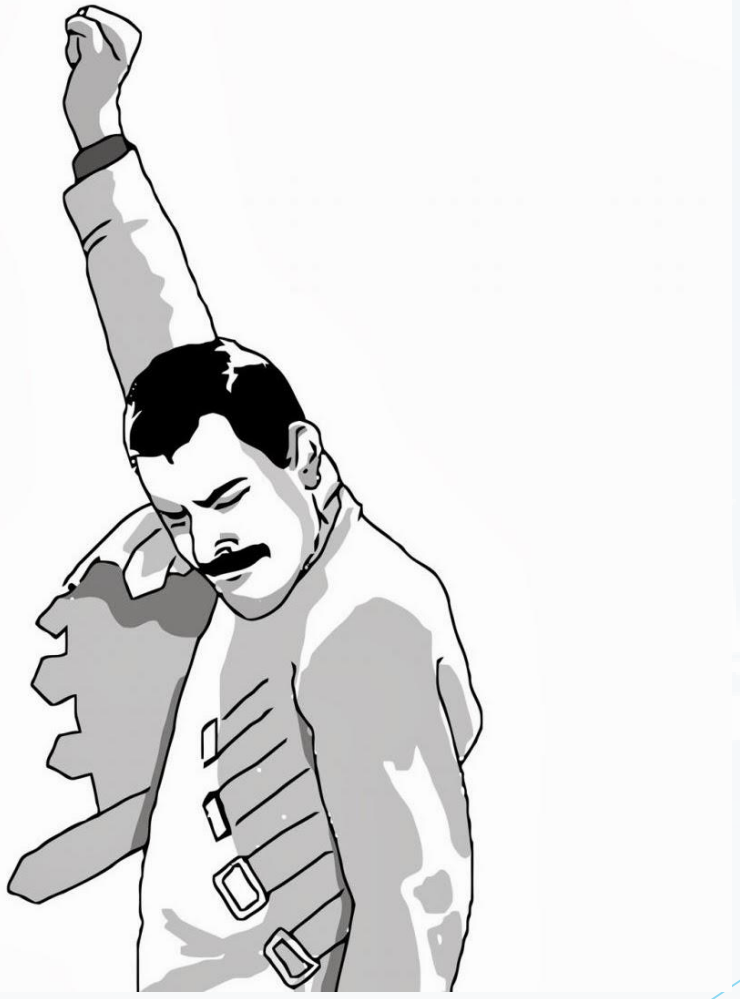
- aapt.exe exited with code
- Unsupported major.minor version 52

Habitualmente se deben a alguna de estas dos razones:

- Una mala configuración del Java JDK
- Tener herramientas de construcción más recientes que las soportadas



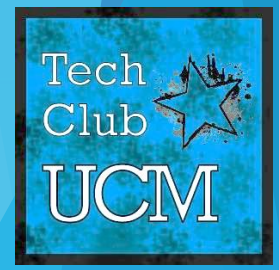
Y listo!



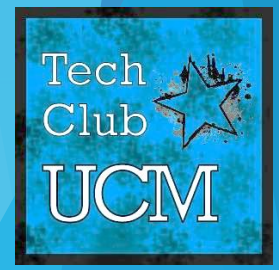
Contenido adicional:

https://github.com/xamarin/dev-days-labs/blob/master/HandsOnLab/README.md

1. Details
2. Connect to Azure Mobile Apps
3. Bonus Take Home Challenges



Muchas gracias por vuestra atención



Seguidnos en Twitter:
@techclubucm
@JMTG888

Visitad nuestro blog:
https://techclubucm.wordpress.com/

Página de Facebook:
https://www.facebook.com/TechClub-UCM-704081556347321