

# ICB0-M08U02IA06

NAVEGACIÓ I INTERFÍCIE AVANÇADA

POL BENITO

# Índice

1. Introducción .....	2
2. Crear un nuevo proyecto .....	4
3. Models .....	5
4. Services .....	6
4.1. INavigationService.cs .....	6
4.2. NavigationService.cs .....	6
5. ViewModels .....	7
5.1. Base .....	7
5.1.1. DelegateCommand.cs .....	7
5.1.2. DelegateCommandAsync.cs .....	9
5.1.3. ViewModelBase.cs .....	12
5.1.4. ViewModelLocator.cs .....	12
5.2. CustomCellViewModel.cs .....	13
5.3. MainViewModel.cs .....	15
6. Views .....	16
6.1. CustomCellView.xaml .....	16
6.1.1. CustomCellView.xaml.cs .....	17
6.2. MainView.xaml .....	18
6.2.1. MainView.xaml.cs .....	19
6.3. SplashPage.xaml .....	19
6.3.1. SplashPage.xaml.cs .....	20
7. App.xaml.cs .....	22
8. Tips .....	23
8.1. Imágenes .....	23
8.1.1. URL .....	23
8.1.2. Guardarlas en caché .....	23
8.2. Colores .....	23

## 1. Introducción

Esta aplicación trata de aprender a cómo hacer un listado, a cómo pulsar en un objeto y ver, por ejemplo, información sobre ello en una nueva página.

En mi caso, la aplicación trata de unos personajes de una serie llamada '**Haikyuu**'. Esta serie trata de vóley, por lo tanto, lo que he hecho es un listado de los personajes que salen en el equipo protagonista. Entonces, cuando pulsas encima de un personaje, te salen diferentes datos: su nombre, la posición en la que juega, algunos detalles sobre éste y una imagen suya.

Además, también he añadido una ventana '**splash**' en la que sale una imagen de la serie.

En esta práctica, vamos a seguir utilizando el **patrón MVM** tal y como aprendimos en la práctica anterior.

A continuación, dejo diferentes capturas de cómo se ve mi aplicación:



*Ventana Splash*



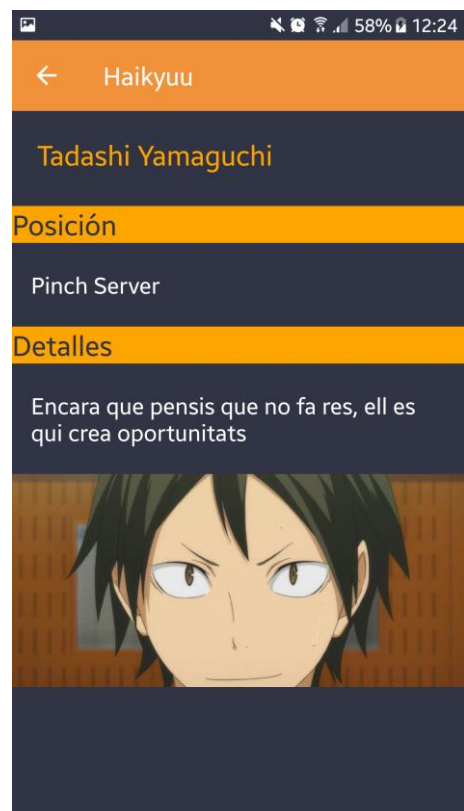
Listado 1



Listado 2



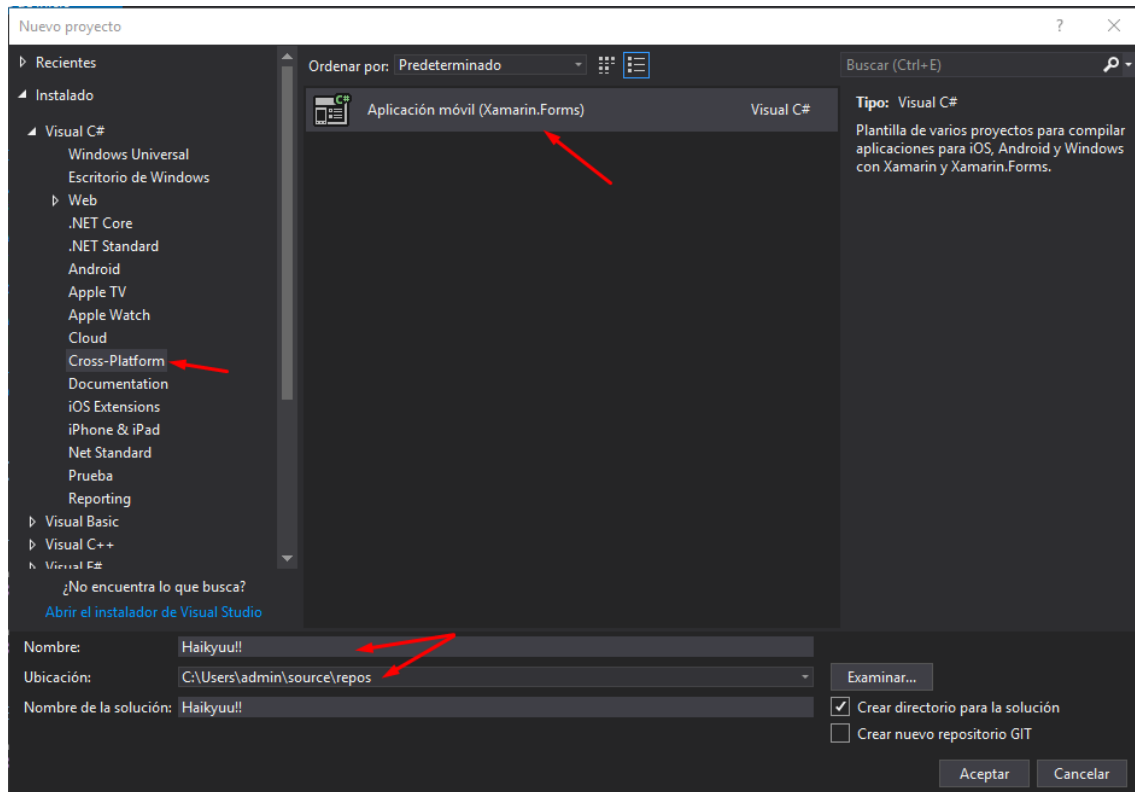
Personaje 1



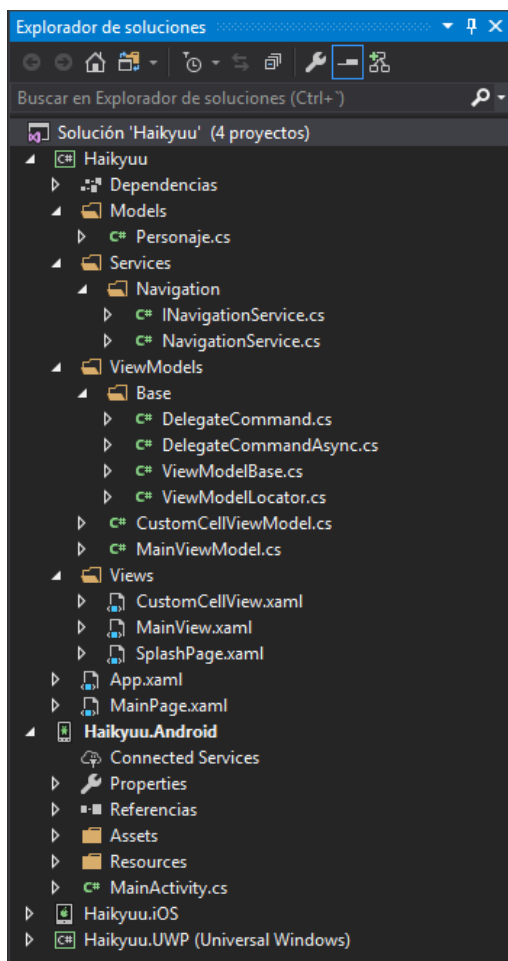
Personaje 2

## 2. Crear un nuevo proyecto

Vamos a empezar creando un proyecto normal de **Xamarin.Forms** con el nombre que queremos. Si queréis hacer la misma aplicación que la mía, llamad **Haikyuuu** (que es como se llama la serie) al programa.



Vamos a empezar a copiar el diseño de mi solución:



### 3. Models

En la carpeta **Models** vamos a crear la clase **Personaje** en el cuál añadimos lo que queremos que tenga cada personaje: un nombre, la posición, los detalles y la imagen.

```
1. namespace Haikyuu.Models
2. {
3.     public class Personaje
4.     {
5.         public string Name { get; set; }
6.         public string Location { get; set; }
7.         public string Details { get; set; }
8.         public string Image { get; set; }
9.     }
10. }
```

## 4. Services

Creamos la carpeta **Services** y dentro la carpeta **Navigation**, en la que añadiremos las clases de navegación que usamos en la aplicación. Por ejemplo, la flecha que tenemos para tirar hacia atrás, este hecho gracias a estas clases.

### 4.1. INavigationService.cs

```
1. using System;
2.
3. namespace Haikyuu.Services.Navigation
4. {
5.     public interface INavigationService
6.     {
7.         void NavigateTo<TDestinationViewModel>(object navigationContext = null
8.         );
9.         void NavigateTo(Type destinationType, object navigationContext = null)
10.        ;
11.        void NavigateBack();
12.    }
13. }
```

### 4.2. NavigationService.cs

```
1. namespace Haikyuu.Services.Navigation
2. {
3.     using System;
4.     using System.Collections.Generic;
5.     using ViewModels;
6.     using Views;
7.     using Xamarin.Forms;
8.
9.     public class NavigationService : INavigationService
10.    {
11.        private IDictionary<Type, Type> viewModelRouting = new Dictionary<Type
12.        , Type>()
13.        {
14.            { typeof(MainViewModel), typeof(MainView) },
15.            { typeof(CustomCellViewModel), typeof(CustomCellView) }
16.        };
17.        public void NavigateTo<TDestinationViewModel>(object navigationContext
18.        = null)
19.        {
20.            Type pageType = viewModelRouting[typeof(TDestinationViewModel)];
21.            var page = Activator.CreateInstance(pageType, navigationContext) a
22.            s Page;
23.            if (page != null)
24.                Application.Current.MainPage.Navigation.PushAsync(page);
25.        }
26.        public void NavigateTo(Type destinationType, object navigationContext
27.        = null)
28.        {
29.            Type pageType = viewModelRouting[destinationType];
30.            var page = Activator.CreateInstance(pageType, navigationContext) a
31.            s Page;
32.        }
33.    }
34. }
```

```

30.
31.         if (page != null)
32.             Application.Current.MainPage.Navigation.PushAsync(page);
33.     }
34.
35.     public void NavigateBack()
36.     {
37.         Application.Current.MainPage.Navigation.PopAsync();
38.     }
39. }
40. }

```

## 5. ViewModels

Dentro de **ViewModels** vamos a tener la carpeta **Base** y afuera los archivos que van a tener los datos de cada personaje.

### 5.1. Base

En **Base** están todos los archivos **necesarios** para crear la lista.

#### 5.1.1. DelegateCommand.cs

```

1. using System;
2. using System.Windows.Input;
3.
4. namespace Haikyuu.ViewModels.Base
5. {
6.     /// <summary>
7.     /// This class allows us to delegate command execution to viewmodels.
8.     /// </summary>
9.     public class DelegateCommand : ICommand
10.    {
11.        private readonly Action _execute;
12.        private readonly Func<bool> _canExecute;
13.
14.        /// <summary>
15.        /// Constructor not using canExecute.
16.        /// </summary>
17.        /// <param name="execute"></param>
18.        public DelegateCommand(Action execute) : this(execute, null) { }
19.
20.        /// <summary>
21.        /// Constructor using both execute and canExecute.
22.        /// </summary>
23.        /// <param name="execute"></param>
24.        /// <param name="canExecute"></param>
25.        public DelegateCommand(Action execute, Func<bool> canExecute)
26.        {
27.            _execute = execute;
28.            _canExecute = canExecute;
29.        }
30.
31.        /// <summary>
32.        /// This method is called from XAML to evaluate if the command can be
33.        /// executed.
34.        /// </summary>
35.        /// <param name="parameter"></param>
36.        /// <returns></returns>
37.        public bool CanExecute(object parameter)
38.        {
39.            if (_canExecute != null)

```



```

39.         return _canExecute();
40.
41.         return true;
42.     }
43.
44.     /// <summary>
45.     /// This method is called from XAML to execute the command.
46.     /// </summary>
47.     /// <param name="parameter"></param>
48.     public void Execute(object parameter)
49.     {
50.         _execute();
51.     }
52.
53.     /// <summary>
54.     /// This method allow us to force the execution of CanExecute method t
55.     o reevaluate execution.
56.     /// </summary>
57.     public void RaiseCanExecuteChanged()
58.     {
59.         var tmpHandle = CanExecuteChanged;
60.         if (tmpHandle != null)
61.             tmpHandle(this, new EventArgs());
62.     }
63.
64.     /// <summary>
65.     /// This event notify XAML controls using the command to reevaluate th
66.     e CanExecute of it.
67.     /// </summary>
68.     public event EventHandler CanExecuteChanged;
69.
70.     /// <summary>
71.     /// This class allows us to delegate command execution to viewmodels using
72.     a T type as parameter.
73.     /// </summary>
74.     public class DelegateCommand<T> : ICommand
75.     {
76.         private readonly Action<T> _execute;
77.         private readonly Func<T, bool> _canExecute;
78.
79.         /// <summary>
80.         /// Constructor not using canExecute.
81.         /// </summary>
82.         /// <param name="execute"></param>
83.         public DelegateCommand(Action<T> execute) : this(execute, null) { }
84.
85.         /// <summary>
86.         /// Constructor using both execute and canExecute.
87.         /// </summary>
88.         /// <param name="execute"></param>
89.         /// <param name="canExecute"></param>
90.         public DelegateCommand(Action<T> execute, Func<T, bool> canExecute)
91.         {
92.             _execute = execute;
93.             _canExecute = canExecute;
94.         }
95.
96.         /// <summary>
97.         /// This method is called from XAML to evaluate if the command can be
98.         executed.
99.         /// </summary>
100.        /// <param name="parameter"></param>
101.        /// <returns></returns>
102.        public bool CanExecute(object parameter)
103.        {

```

```

101.         if (_canExecute != null)
102.             return _canExecute((T)parameter);
103.
104.         return true;
105.     }
106.
107.     /// <summary>
108.     /// This method is called from XAML to execute the command.
109.     /// </summary>
110.     /// <param name="parameter"></param>
111.     public void Execute(object parameter)
112.     {
113.         _execute((T)parameter);
114.     }
115.
116.     /// <summary>
117.     /// This method allow us to force the execution of CanExecute m
118.     ethod to reevaluate execution.
119.     /// </summary>
120.     public void RaiseCanExecuteChanged()
121.     {
122.         var tmpHandle = CanExecuteChanged;
123.         if (tmpHandle != null)
124.             tmpHandle(this, new EventArgs());
125.     }
126.
127.     /// <summary>
128.     /// This event notify XAML controls using the command to reeval
129.     uate the CanExecute of it.
130.     /// </summary>
131.     public event EventHandler CanExecuteChanged;

```

### 5.1.2. DelegateCommandAsync.cs

```

1. using System;
2. using System.Threading.Tasks;
3. using System.Windows.Input;
4.
5. namespace Haikyuu.ViewModels.Base
6. {
7.     /// <summary>
8.     /// This class allows us to delegate command execution to viewmodels.
9.     /// This version of the command allow to use async/await.
10.    /// </summary>
11.    public class DelegateCommandAsync : ICommand
12.    {
13.        private readonly Func<Task<bool>> _canExecute;
14.        private readonly Func<Task> _execute;
15.
16.        /// <summary>
17.        /// Constructor not using canExecute.
18.        /// </summary>
19.        /// <param name="execute"></param>
20.        public DelegateCommandAsync(Func<Task> execute) : this(execute, null)
21.        {
22.        }
23.
24.        /// <summary>
25.        /// Constructor using both execute and canExecute.
26.        /// </summary>
27.        /// <param name="execute"></param>
28.        /// <param name="canExecute"></param>

```

```

29.         public DelegateCommandAsync(Func<Task> execute, Func<Task<bool>> canExecut
e)
30.         {
31.             _execute = execute;
32.             _canExecute = canExecute;
33.         }
34.
35.         /// <summary>
36.         ///     This method is called from XAML to evaluate if the command can be
executed.
37.         /// </summary>
38.         /// <param name="parameter"></param>
39.         /// <returns></returns>
40.         public bool CanExecute(object parameter)
41.         {
42.             if (_canExecute != null)
43.                 return _canExecute().Result;
44.
45.             return true;
46.         }
47.
48.         /// <summary>
49.         ///     This method is called from XAML to execute the command.
50.         /// </summary>
51.         /// <param name="parameter"></param>
52.         public void Execute(object parameter)
53.         {
54.             _execute();
55.         }
56.
57.         /// <summary>
58.         ///     This event notify XAML controls using the command to reevaluate th
e CanExecute of it.
59.         /// </summary>
60.         public event EventHandler CanExecuteChanged;
61.
62.         /// <summary>
63.         ///     This method allow us to force the execution of CanExecute method t
o reevaluate execution.
64.         /// </summary>
65.         public void RaiseCanExecuteChanged()
66.         {
67.             EventHandler tmpHandle = CanExecuteChanged;
68.             if (tmpHandle != null)
69.                 tmpHandle(this, new EventArgs());
70.         }
71.     }
72.
73.     /// <summary>
74.     ///     This class allows us to delegate command execution to viewmodels using
a T type as parameter.
75.     /// </summary>
76.     public class DelegateCommandAsync<T> : ICommand
77.     {
78.         private readonly Func<T, Task<bool>> _canExecute;
79.         private readonly Func<T, Task> _execute;
80.
81.         /// <summary>
82.         ///     Constructor not using canExecute.
83.         /// </summary>
84.         /// <param name="execute"></param>
85.         public DelegateCommandAsync(Func<T, Task> execute) : this(execute, null)
86.         {
87.         }
88.
89.         /// <summary>

```

```

90.         ///     Constructor using both execute and canExecute.
91.         /// </summary>
92.         /// <param name="execute"></param>
93.         /// <param name="canExecute"></param>
94.         public DelegateCommandAsync(Func<T, Task> execute, Func<T, Task<bool>> can
Execute)
95.         {
96.             _execute = execute;
97.             _canExecute = canExecute;
98.         }
99.
100.        /// <summary>
101.        ///     This method is called from XAML to evaluate if the command can
be executed.
102.        /// </summary>
103.        /// <param name="parameter"></param>
104.        /// <returns></returns>
105.        public bool CanExecute(object parameter)
106.        {
107.            if (_canExecute != null)
108.                return _canExecute((T) parameter).Result;
109.
110.            return true;
111.        }
112.
113.        /// <summary>
114.        ///     This method is called from XAML to execute the command.
115.        /// </summary>
116.        /// <param name="parameter"></param>
117.        public void Execute(object parameter)
118.        {
119.            _execute((T) parameter);
120.        }
121.
122.        /// <summary>
123.        ///     This event notify XAML controls using the command to reevaluat
e the CanExecute of it.
124.        /// </summary>
125.        public event EventHandler CanExecuteChanged;
126.
127.        /// <summary>
128.        ///     This method allow us to force the execution of CanExecute meth
od to reevaluate execution.
129.        /// </summary>
130.        public void RaiseCanExecuteChanged()
131.        {
132.            EventHandler tmpHandle = CanExecuteChanged;
133.            if (tmpHandle != null)
134.                tmpHandle(this, new EventArgs());
135.        }
136.    }
137. }

```

### 5.1.3. ViewModelBase.cs

```
1. using System.ComponentModel;
2. using System.Runtime.CompilerServices;
3.
4. namespace Haikyuu.ViewModels.Base
5. {
6.     public abstract class ViewModelBase : INotifyPropertyChanged
7.     {
8.         public virtual void OnAppearing(object navigationContext)
9.         {
10.         }
11.
12.         public virtual void OnDisappearing()
13.         {
14.         }
15.
16.         public event PropertyChangedEventHandler PropertyChanged;
17.
18.         public void RaisePropertyChanged([CallerMemberName]string propertyName
19. = "")
20.         {
21.             var handler = PropertyChanged;
22.             if (handler != null)
23.                 handler(this, new PropertyChangedEventArgs(propertyName));
24.         }
25. }
```

### 5.1.4. ViewModelLocator.cs

```
1. using Unity;
2. using Haikyuu.Services.Navigation;
3.
4. namespace Haikyuu.ViewModels.Base
5. {
6.     public class ViewModelLocator
7.     {
8.         readonly IUnityContainer _container;
9.
10.         public ViewModelLocator()
11.         {
12.             _container = new UnityContainer();
13.
14.             // ViewModels
15.             _container.RegisterType<CustomCellViewModel>();
16.             _container.RegisterType<MainViewModel>();
17.
18.             // Services
19.             _container.RegisterType<INavigationService, NavigationService>();
20.         }
21.
22.         public CustomCellViewModel CustomCellViewModel
23.         {
24.             get { return _container.Resolve<CustomCellViewModel>(); }
25.         }
26.
27.         public MainViewModel MainViewModel
28.         {
29.             get { return _container.Resolve<MainViewModel>(); }
30.         }
31.     }
32. }
```

## 5.2. CustomCellViewModel.cs

En esta clase vamos a copiar el formato de **new Personaje** para añadir tantos personajes como queramos. Dentro añadimos los **datos** de ese personaje. En el caso de la imagen, tienes dos opciones. Guardarlo en la carpeta **drawable** tal y como hemos hecho en prácticas anteriores o poner la **url** de la imagen directamente. Esta vez he decidido poner la **url** para saber cuál de las dos opciones era mejor.

Para mi gusto, prefiero **descargarme** las imágenes y ponerlas en **drawable**, ya que prefiero que la aplicación ocupe un poco más de espacio a que cuando estés con la aplicación las imágenes no se carguen, que es lo que pasa si pones la **url**.

```
1. using Haikyuu.Models;
2. using Haikyuu.ViewModels.Base;
3. using System.Collections.ObjectModel;
4.
5. namespace Haikyuu.ViewModels
6. {
7.     public class CustomCellViewModel : ViewModelBase
8.     {
9.         public ObservableCollection<Personaje> Personajes { get; set; }
10.
11.         public CustomCellViewModel()
12.         {
13.             Personajes = new ObservableCollection<Personaje>
14.             {
15.                 new Personaje
16.                 {
17.                     Name = "Shōyō Hinata",
18.                     Location = "Bloquejador central, Carnada",
19.                     Details =
20.                         "Vol arribar a ser com el Petit Gégant",
21.                     Image =
22.                         "https://vignette.wikia.nocookie.net/haikyuu/images/c/c2/Rostro_de_Hinata.png/revision/latest?cb=20160210050706&path-prefix=es"
23.                 },
24.                 new Personaje
25.                 {
26.                     Name = "Tobio Kageyama",
27.                     Location = "Col·locador",
28.                     Details =
29.                         "Juntament amb Hinata, un parella imparable",
30.                     Image =
31.                         "https://vignette.wikia.nocookie.net/haikyuu-pedia/images/4/47/Tobio_Kageyama.jpg/revision/latest?cb=20140721131354"
32.                 },
33.                 new Personaje
34.                 {
35.                     Name = "Daichi Sawamura",
36.                     Location = "Wing Spikers, Capità",
37.                     Details =
38.                         "Especialista en la defensa i una persona necessària per a
39.                         l equip",
40.                     Image =
41.                         "https://em.wattpad.com/8bd5055f8ed26c2660a747809f9d943e64c8cfe4/687474703a2f2f36382e6d656469612e74756d626c722e636f6d2f6637636266383238323336"
```

```

1303463353334373366561376530336262393933302f74756d626c725f696e6c696e655f6e6833736
46f464d714a3172696e3631382e6a7067?s=fit&h=758&w=1583&q=80"
41.         },
42.         new Personaje
43.         {
44.             Name = "Kōshi Sugawara",
45.             Location = "Segon col·locador, Segon capità",
46.             Details =
47.                 "No serà titular, però es un dels millors en la seva feina",
48.             Image =
49.                 "https://vignette.wikia.nocookie.net/haikyu/images/8/8d/Sugawara.png/revision/latest?cb=20150821230947&path-prefix=es"
50.         },
51.         new Personaje
52.         {
53.             Name = "Asahi Azumane",
54.             Location = "Estrella",
55.             Details =
56.                 "A qui pots recórrer quan estàs en problemes",
57.             Image =
58.                 "https://vignette.wikia.nocookie.net/haikyu-pedia/images/c/c4/Asahi_Azumane.jpg/revision/latest?cb=20140721131410"
59.         },
60.         new Personaje
61.         {
62.             Name = "Yū Nishinoya",
63.             Location = "Liberero",
64.             Details =
65.                 "El millor en salvar les pilotes impossibles",
66.             Image =
67.                 "https://vignette.wikia.nocookie.net/haikyu/images/8/80/Y%C5%AB_Nishinoya.png/revision/latest?cb=20170809205325&path-prefix=es"
68.         },
69.         new Personaje
70.         {
71.             Name = "Ryūnosuke Tanaka",
72.             Location = "Wing Spikers",
73.             Details =
74.                 "Vol ser la estrella i té le poder per ser-ho",
75.             Image =
76.                 "https://vignette.wikia.nocookie.net/haikyu/images/e/e7/RyuTanaka.png/revision/latest?cb=20150107144228&path-prefix=es"
77.         },
78.         new Personaje
79.         {
80.             Name = "Kei Tsukishima",
81.             Location = "Bloquejador central",
82.             Details =
83.                 "No creguis que només bloqueja, també pensa",
84.             Image =
85.                 "https://vignette.wikia.nocookie.net/haikyu/images/f/f9/Kei_Tsukishima.png/revision/latest?cb=20170822215947&path-prefix=es"
86.         },
87.         new Personaje
88.         {
89.             Name = "Tadashi Yamaguchi",
90.             Location = "Pinch Server",
91.             Details =
92.                 "Encara que pensis que no fa res, ell es qui crea oportunitats",
93.             Image =
94.                 "https://pbs.twimg.com/media/C2VZ_kGXcAE3oS2.jpg"
95.         },
96.         new Personaje
97.         {

```

```

98.             Name = "Keishin Ukai",
99.             Location = "Entrenador",
100.            Details =
101.                "Antic jugador del Karasuno i el seu avi va entrenar a
1    l Petit Gegant",
102.            Image =
103.                "https://vignette.wikia.nocookie.net/haikyu/images/4/
44/KeishinUkai.png/revision/latest?cb=20140616134254"
104.        },
105.        new Personaje
106.        {
107.            Name = "Ittetsu Takeda",
108.            Location = "Professor",
109.            Details =
110.                "Gràcies a ell Karasuno obté la seva força",
111.            Image =
112.                "https://vignette.wikia.nocookie.net/haikyu/images/d/
da/Takeda-sensei.png/revision/latest?cb=20140616135813"
113.        }
114.    };
115.    }
116. }
117. }

```

### 5.3. MainViewModel.cs

Aquí definimos que cuando pulses en un personaje, haga un llamamiento a la **clase** del CustomCellViewModel.

```

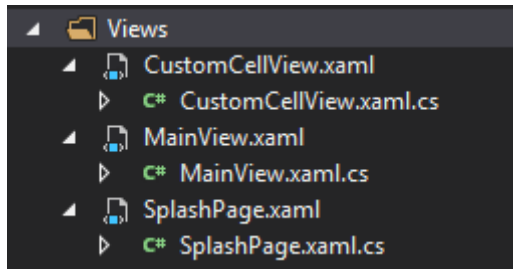
1. using Haikyu.Services.Navigation;
2. using Haikyu.ViewModels.Base;
3. using System.Windows.Input;
4.
5. namespace Haikyu.ViewModels
6. {
7.     public class MainViewModel : ViewModelBase
8.     {
9.         private ICommand _customCellCommand;
10.
11.         private INavigationService _navigationService;
12.
13.         public MainViewModel(INavigationService navigationService)
14.         {
15.             _navigationService = navigationService;
16.         }
17.
18.         public ICommand CustomCellCommand
19.         {
20.             get { return _customCellCommand = _customCellCommand ?? new DelegateCo
mmmand(CustomCellCommandExecute); }
21.         }
22.         private void CustomCellCommandExecute()
23.         {
24.             _navigationService.NavigateTo<CustomCellViewModel>();
25.         }
26.     }
27. }

```



## 6. Views

Dentro de **Views** tenemos las **páginas de contenido**, en las que tenemos el código en formato **.xaml** y después una **clase** dentro de ella.



### 6.1. CustomCellView.xaml

Aquí creamos el diseño a como se va a ver el listado. Para ello vamos a añadirle un título, hacerle la referencia al **model** con el **Binding Personajes**, darle un **nombre** a la **lista** para utilizarlo más adelante con el **x:Name** y **añadir** las **imágenes** con el **Binding Image**. Además de definirle que haga un **Fill** para que así **ocupe** todo el **espacio**.

```
1. <?xml version="1.0" encoding="utf-8" ?>
2. <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
3.             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
4.             x:Class="Haikyuu.Views.CustomCellView"
5.             Title="Haikyuu">
6.     <ListView
7.         ItemsSource="{Binding Personajes}"
8.         RowHeight="200"
9.         x:Name="ListaPersonajes">
10.        <ListView.ItemTemplate>
11.            <DataTemplate>
12.                <ViewCell>
13.                    <StackLayout
14.                        Orientation="Horizontal">
15.                        <Image
16.                            Source="{Binding Image}"
17.                            Aspect="AspectFill"
18.                            WidthRequest="380"/>
19.                    </StackLayout>
20.                </ViewCell>
21.            </DataTemplate>
22.        </ListView.ItemTemplate>
23.    </ListView>
24. </ContentPage>
```

### 6.1.1. CustomCellView.xaml.cs

En su **clase** vamos a hacer un llamamiento a la lista (con el nombre que le hemos definido anteriormente) y con el **evento** 'ItemSelected' vamos a crear una función en la que con ayuda del **Navigation.PushAsync** vamos a decir que cuando pulsen encima vayan al **MainView** (función que crearemos más adelante en el **MainView.xaml.cs**) y le pasaremos el **parámetro** de **Personaje**.

```
1. using System;
2. using Haikyu.Models;
3. using Haikyu.ViewModels;
4. using Xamarin.Forms;
5.
6. namespace Haikyu.Views
7. {
8.     public partial class CustomCellView : ContentPage
9.     {
10.         private object Parameter { get; set; }
11.
12.         public CustomCellView(object parameter)
13.         {
14.             InitializeComponent();
15.
16.             ListaPersonajes.ItemSelected += ListaPersonajes_ItemSelected;
17.             BindingContext = App.Locator.CustomCellViewModel;
18.
19.             Parameter = parameter;
20.         }
21.
22.         private void ListaPersonajes_ItemSelected(object sender, SelectedItemC
23.             hangedEventArgs e)
24.         {
25.             Navigation.PushAsync(new MainView((Personaje)e.SelectedItem));
26.         }
27.
28.         protected override void OnAppearing()
29.         {
30.             var viewModel = BindingContext as CustomCellViewModel;
31.             if (viewModel != null) viewModel.OnAppearing(Parameter);
32.         }
33.
34.         protected override void OnDisappearing()
35.         {
36.             var viewModel = BindingContext as CustomCellViewModel;
37.             if (viewModel != null) viewModel.OnDisappearing();
38.         }
39. }
```

## 6.2. MainView.xaml

Creamos el diseño que queremos cuando el usuario pulse encima de algún personaje.

En mi caso le pongo un título, le añado el nombre del personaje cambiando los colores, la posición, los detalles que hemos definido anteriormente y por último la imagen utilizada anteriormente.

Aquí le podemos añadir otra imagen o lo que queramos, a gusto del creador de la aplicación.

```
1. <?xml version="1.0" encoding="utf-8" ?>
2. <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
3.             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
4.             x:Class="Haikyuuu.Views.MainView"
5.             Title="Haikyuuu">
6.     <ContentPage.Content>
7.         <ScrollView>
8.             <StackLayout
9.                 BackgroundColor="#313445">
10.                <Label x:Name="Name"
11.                    FontSize="Large"
12.                    Text="Name"
13.                    TextColor="Orange"
14.                    Margin="20"
15.                />
16.
17.                <Label Text="Posición"
18.                    FontSize="Large"
19.                    BackgroundColor="Orange"
20.                    TextColor="#313445"
21.                />
22.
23.                <Label x:Name="Location"
24.                    FontSize="Medium"
25.                    Text="Location"
26.                    TextColor="White"
27.                    Margin="15"
28.                />
29.
30.                <Label Text="Detalles"
31.                    FontSize="Large"
32.                    BackgroundColor="Orange"
33.                    TextColor="#313445"
34.                />
35.
36.                <Label x:Name="Details"
37.                    Text="Details"
38.                    FontSize="Medium"
39.                    TextColor="White"
40.                    Margin="15"
41.                />
42.
43.                <Image x:Name="Foto"
44.                    Aspect="AspectFill"
45.                    WidthRequest="380"
46.                    MinimumHeightRequest="500"
47.                />
48.
```

```

49.         </StackLayout>
50.     </ScrollView>
51. </ContentPage.Content>
52. </ContentPage>

```

### 6.2.1. MainView.xaml.cs

En su **clase** vamos a definir las referencias que hemos hecho en el fichero **.xaml**. Es decir, creamos la función **MainView** pasándole el parámetro **Personaje** (que anteriormente hacíamos la llamada en el fichero **CustomCellView.xaml.cs**) donde decimos que **Name** (sacado del modelo **Personaje**) es lo mismo que el **Name** puesto anteriormente en el **MainView.xaml**.

```

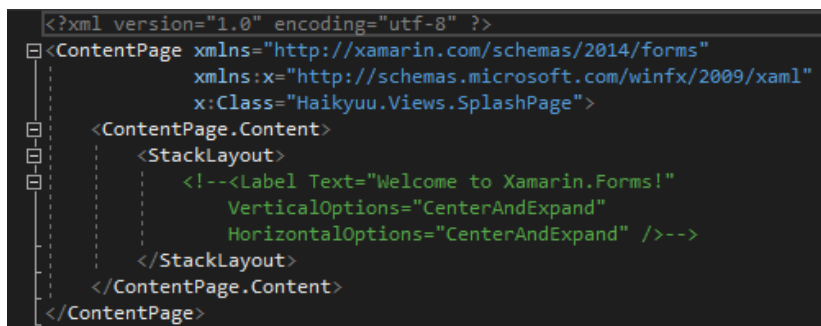
1. using Xamarin.Forms;
2. using Xamarin.Forms.Xaml;
3. using Haikyuu.ViewModels;
4. using Haikyuu.Models;
5.
6. namespace Haikyuu.Views
7. {
8.     [XamlCompilation(XamlCompilationOptions.Compile)]
9.     public partial class MainView : ContentPage
10.    {
11.        public MainView(Personaje parameter)
12.        {
13.            InitializeComponent();
14.
15.
16.            Name.Text = parameter.Name;
17.            Location.Text = parameter.Location;
18.            Details.Text = parameter.Details;
19.            Foto.Source = parameter.Image;
20.
21.            BindingContext = App.Locator.MainViewModel;
22.        }
23.    }
24. }

```

### 6.3. SplashPage.xaml

Ahora vamos a crear la pantalla **Splash**.

En este fichero simplemente comentamos la parte de código donde se crea el **Label** automáticamente:



```

<?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="Haikyuu.Views.SplashPage">
    <ContentPage.Content>
        <StackLayout>
            <!--<Label Text="Welcome to Xamarin.Forms!"
                    VerticalOptions="CenterAndExpand"
                    HorizontalOptions="CenterAndExpand" />-->
        </StackLayout>
    </ContentPage.Content>
</ContentPage>

```

### 6.3.1. SplashPage.xaml.cs

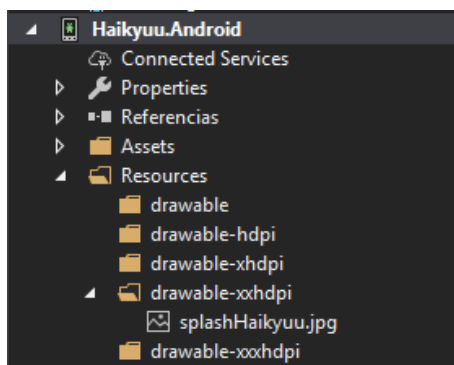
Aquí añadimos la **imagen** que queremos que salga, el **tiempo** que durará la imagen y la **ruta de navegación** que queremos que haga cuando termine.

En esta función he comentado dos líneas las cuáles hacen que la imagen haga un zoom hacia adelante. A mí no me gusta, pero dejo el código por si alguien lo quiere usar.

```
protected override async void OnAppearing()
{
    base.OnAppearing();

    await splashImage.ScaleTo(1, 2000);
    //await splashImage.ScaleTo(0.9, 1500, Easing.Linear);
    //await splashImage.ScaleTo(1.5, 1200, Easing.Linear);
    Application.Current.MainPage = new NavigationPage(new CustomCellView(null));
}
```

Para añadir la imagen, vamos a tener que ir dentro de **Haikyuu.Android** y guardarla dentro de la carpeta **drawable**. En un principio debería funcionar, pero dependiendo de la resolución de la imagen igual la tenéis que guardar dentro de **otra carpeta**. En mi caso la he tenido que guardarla dentro de **drawable-xxhdpi**.



Una vez la hayamos guardado, dentro de **splashImage**, en **Source**, ponemos el nombre de la imagen.

```

1.  using System;
2.  using System.Collections.Generic;
3.  using System.Linq;
4.  using System.Text;
5.  using System.Threading.Tasks;
6.
7.  using Xamarin.Forms;
8.  using Xamarin.Forms.Xaml;
9.
10. namespace Haikyuu.Views
11. {
12.     [XamlCompilation(XamlCompilationOptions.Compile)]
13.     public partial class SplashPage : ContentPage
14.     {
15.         Image splashImage;
16.         public SplashPage ()
17.         {
18.             InitializeComponent();
19.
20.             NavigationPage.SetHasNavigationBar(this, false);
21.
22.             var sub = new AbsoluteLayout();
23.             splashImage = new Image
24.             {
25.                 Source = "splashHaikyuu.jpg",
26.                 Aspect = Aspect.Fill,
27.                 WidthRequest = 380
28.                 //WidthRequest = 100,
29.                 //HeightRequest = 100
30.             };
31.
32.             AbsoluteLayout.SetLayoutFlags(splashImage,
33.                 AbsoluteLayoutFlags.PositionProportional);
34.             AbsoluteLayout.SetLayoutBounds(splashImage,
35.                 new Rectangle(0.5, 0.5, AbsoluteLayout.AutoSize, AbsoluteLayout.
AutoSize));
36.
37.             sub.Children.Add(splashImage);
38.
39.             this.BackgroundColor = Color.FromHex("#F1913A");
40.             this.Content = sub;
41.         }
42.
43.         protected override async void OnAppearing()
44.         {
45.             base.OnAppearing();
46.
47.             await splashImage.ScaleTo(1, 2000);
48.             //await splashImage.ScaleTo(0.9, 1500, Easing.Linear);
49.             //await splashImage.ScaleTo(150, 1200, Easing.Linear);
50.             Application.Current.MainPage = new NavigationPage(new CustomCellView
(null));
51.         }
52.
53.     }
54. }

```

## 7. App.xaml.cs

Dentro de **App** vamos a hacer referencia hacía el **ViewModelLocator** y dentro de la función **App()** decimos que queremos que la aplicación se inicie con la pantalla **Splash** diciendo que el **MainPage** sea una **navegación** hacia el **SplashPage**.

```
1. using Haikyuu.Views;
2. using Haikyuu.ViewModels.Base;
3. using Xamarin.Forms;
4. using Xamarin.Forms.Xaml;
5.
6. [assembly: XamlCompilation(XamlCompilationOptions.Compile)]
7. namespace Haikyuu
8. {
9.     public partial class App : Application
10.    {
11.        private static ViewModelLocator _locator;
12.
13.        public static ViewModelLocator Locator
14.        {
15.            get { return _locator = _locator ?? new ViewModelLocator(); }
16.        }
17.        public App()
18.        {
19.            InitializeComponent();
20.
21.            MainPage = new NavigationPage(new SplashPage());
22.        }
23.
24.        protected override void OnStart()
25.        {
26.            // Handle when your app starts
27.        }
28.
29.        protected override void OnSleep()
30.        {
31.            // Handle when your app sleeps
32.        }
33.
34.        protected override void OnResume()
35.        {
36.            // Handle when your app resumes
37.        }
38.    }
39. }
```

## 8. Tips

### 8.1. Imágenes

Tal y como he comentado tanto en otros tutoriales como a lo largo de éste, las imágenes que pueden poner de dos formas: por **url** o **guardándolas** dentro del programa en una **carpeta**.

#### 8.1.1. URL

Si ponemos la URL directamente de la imagen, tendremos diferentes problemas, aunque sea mucho más fácil y rápido de programar. Lo primero es que cuando abras la aplicación las imágenes no saldrán correctamente, ya que las estas cargando a través de internet y puede que vaya o **lento** o **no cargue** correctamente. Esto puede hacer que el diseño no se vea pulido y puede generar críticas negativas por parte del usuario.

#### 8.1.2. Guardarlas en caché

En el caso de que nos descarguemos las imágenes, no debería haber ningún problema a la hora de cargar las imágenes. Pero la parte negativa de guardarlas directamente en la carpeta, es que, si hay muchas imágenes, la aplicación pesará mucho más y es un proceso más lento. Aun así, es preferiblemente guardarlas en una carpeta, o como veremos más adelante, en una base de datos.

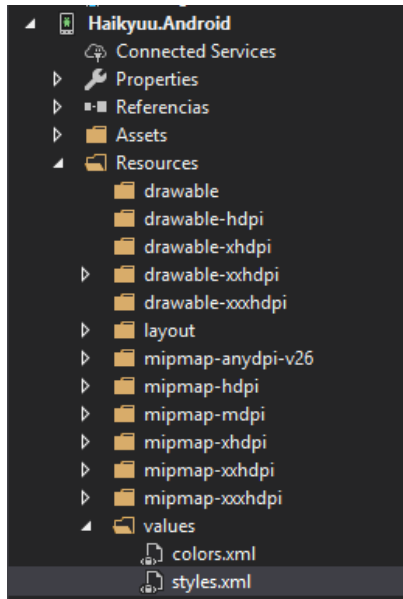
### 8.2. Colores

Cambiar los colores de tu aplicación es un parte fundamental del diseño. Es verdad que desde cada página puedes modificar el color de fondo, el color de la letra e incluso el color de fondo de las propias letras, pero, aun así, no puedes cambiar el color general de la aplicación a través de estas páginas.



Para poderlo hacer, necesitamos ir al archivo **styles.xml** que tenemos en **Haikyu.Android**

→ Resources → values.



Dentro de este archivo vamos a buscar el ítem llamado **colorPrimary** y **colorPrimaryDark**.

```
<!-- colorPrimary is used for the default action bar background -->  
<item name="colorPrimary">#F1913A</item><!--Orange-->  
<!-- colorPrimaryDark is used for the status bar -->  
<item name="colorPrimaryDark">#313445</item><!--Black~Gray-->
```

Tal y como nos sale en el comentario de arriba, el `colorPrimary` es la **parte naranja** que señalo con la **flecha naranja** y el `colorPrimaryDark` la **barra de arriba gris** que señalo con la **flecha gris**.



Simplemente tendremos que coger el `color` en **formato HTML** y añadirlo dentro del **ítem**.