Xamarin

Applications Mobiles en C#

Antoine Dieudonné

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

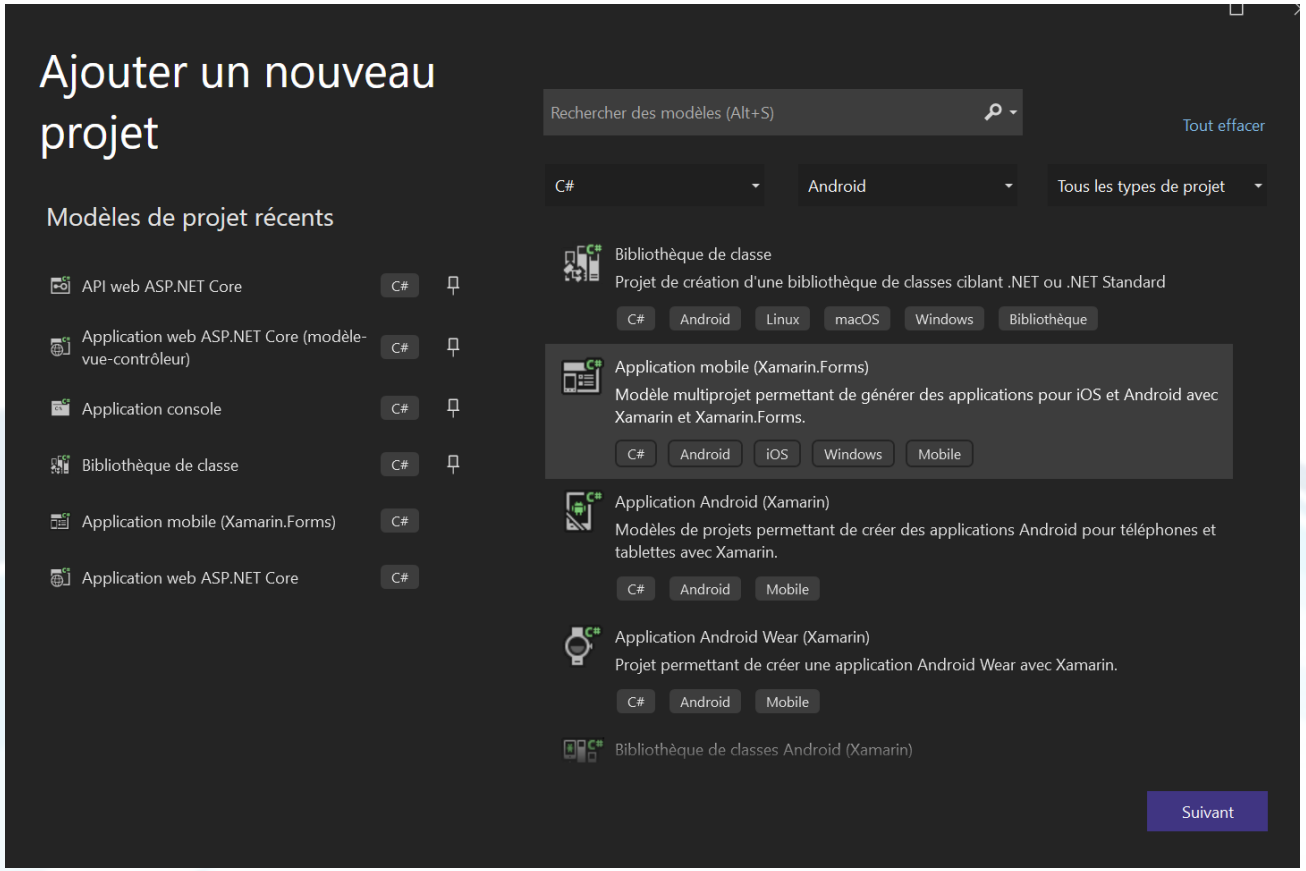
La Création d'un Projet

La création d'un projet Xamarin se fait, comme d'habitude, via le lancement de Visual Studio:

Dans notre cas, nous allons réaliser des applications de type Xamarin.Forms (Vide) qui seront compatibles à la fois avec Android et iOS. Cependant, au vu de l'utilisation d'un ordinateur Windows, nous nous concentrerons sur la manipulation de ces applications dans l'environnement Windows.

Pour plus de facilité, nous utiliserons également un émulateur Android, mais il est possible de relier son ordinateur à son Smartphone via l'utilisation d'un câble USB ou via le Wifi après avoir activé le mode développeur sur ledit smartphone.

Pour activer le mode développeur, il vous suffit d'aller sur les paramètres de votre téléphone et d'appuyer 7 fois consécutives sur le numéro de build de votre version d'Android.



La Structure d'un Projet

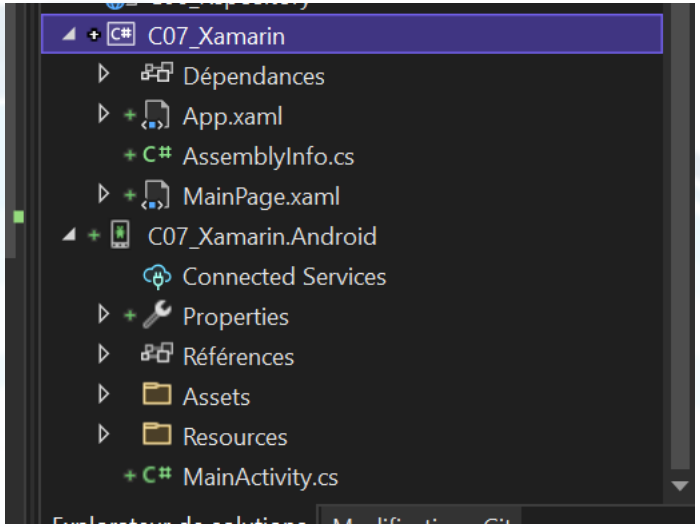
Une fois le projet créé, on observe que ce dernier est en réalité composé de deux projets (ou 3 ou 4 si l'on choisit de travailler également pour iOS ou UWP).

Le projet sur lequel on passera le plus de temps se trouve être le projet commun aux autres projets, et son nom correspond à celui que nous avons entré lors de la création du projet. Les autres, suffixés du nom de l'OS, servent à lancer et tester l'application.

Les différentes pages de notre application se trouvent être composées de deux parties, reliées par le même nom de classe qu'elles se partagent via l'utilisation du mot clé **partial.** Ce mot-clé sert au C# à relier plusieurs fichiers en une même classe, qui sera compilée de façon symbiote.

Chaque page possède ainsi une partie Design en **.xaml** (une variante du XML) et une partie Logique en **.xaml.cs** qui nous servira dans la construction des fonctionnalités de l'application.

```
<StackLayout>
    <Frame BackgroundColor="#2196F3" Padding="24" CornerRadius="0">
        <Label Text="Welcome to Xamarin.Forms!" HorizontalTextAlignment="Center" TextColor="White" FontSize="36"/>
    </Frame>
    <Label Text="Start developing now" FontSize="Title" Padding="30,10,30,10"/>
```



Les Displays de base

L'**empilement** est le premier type de layout utilisé lors d'une application Xamarin.Forms, et il permet tout simplement d'entasser des éléments à la suite les uns des autres. De base, son orientation est de type verticale, mais il est possible d'en changer pour s'en servir de façon horizontale.

```
<StackLayout Orientation="Horizontal"
             VerticalOptions="Center"
             HorizontalOptions="FillAndExpand"
             Spacing="5">
```

Il est possible de mettre des **StackLayouts** dans d'autres **StackLayouts**, mais lors de l'organisation de plusieurs élements les uns par rapport aux autres, il est en général préférable de se servir d'un layout de type **Grid,** que l'on peut également remplir de **StackLayouts** ou d'autres **Grids** si l'on en a envie…

Une **grille** se construit par un enchainement de définitions de lignes et de colonnes, que l'on peut fixer à une taille fixe (**xxx**), à une taille automatique en fonction du contenu (**Auto**), ou à une taille fractionnée du restant de l'espace disponible via l'utilisation de (*).
Il est également possible de faire dépasser les éléments des colonnes et lignes de départ via l'utilisation de **ColumnSpan** et de **RowSpan.**

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="50"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
        <RowDefinition Height="5*"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*"/>
        <ColumnDefinition Width="250"/>
    </Grid.ColumnDefinitions>

    <Label Grid.Row="0" Grid.Column="1"/>
    <Entry Grid.Row="2" Grid.Column="0" Grid.ColumnSpan="1"/>
</Grid>
```

```
<ScrollView Orientation="Both"
            Margin="10"
            VerticalScrollBarVisibility="Always">
    <StackLayout>
```

Les Elements de base

Les élements les plus communs lors de l'utilisation d'une application de type Xamarin.Forms sont sans doute les **Entries**, les **Labels**, les **Images** et les **Buttons**. A cela peuvent s'ajouter d'autres éléments comme les **ListViews**, les **Pickers**, les **CheckBoxes** et les **RadioButtons**.

L'**Entry** est un élément permettant de récupérer des valeurs saisies par l'utilisateur, et de s'en servir par la suite dans notre application.

Le **Label** est un élément permettant tout simplement de visionner les données contenues dans l'application.

Les **Buttons** sont généralement reliées à des évènements ou à des commandes qui sont déclenchés afin d'apporter des fonctionnalités à notre application.

La **ListView** est en élément servant à afficher une série de données sous la forme d'un tableau, que l'on relie souvent aux données :

```
<ListView x:Name="dogsListView" ItemsSource="{Binding Dogs}" ItemSelected="dogsListView_ItemSelected" Margin="10, 0, 10, 5">
    <ListView.Header>Liste des Chiens</ListView.Header>
    <ListView.ItemTemplate>
        <DataTemplate>
            <TextCell Text="{Binding DogName}" Detail="{Binding DogDescription}"/>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

```
    <Entry x:Name="phoneEntry"
           Placeholder="Numéro de téléphone"
           TextColor="White"
           TextTransform="Uppercase"
           IsPassword="False"
           Keyboard="Telephone"
           IsSpellCheckEnabled="False"
           PlaceholderColor="Red"/>

    <Label x:Name="nameLabel"
           FontSize="Large"
           FontAttributes="Bold,Italic"
           FontFamily="Verdana"
           LineBreakMode="WordWrap"
           HorizontalTextAlignment="Center"
           Text="Martin Dupont"/>

    <Button x:Name="saveButton"
            IsEnabled="True"
            Padding="5"
            BackgroundColor="Purple"
            Text="Save"
            TextColor="White"
            Clicked="saveButton_Clicked"/>
</Grid>
```

SQLite

SQLite est une technologie de gestion de base de données optimisée dans le but d'être incroyablement petite et facilement portative, ce qui en fait l'élément de choix d'une application de type **mobile** ou d'un **jeu vidéo**.

Pour utiliser SQLite, il est recommandé d'utiliser le package NuGet **sqlite-net-pcl**. Une fois cela fait, il faudra modifier le point d'entrée de notre application de sorte à récupérer dans une variable statique l'endroit où se situe la base de donnée :

```
    string dbName = "aston_demo_C07_XamarinDB.sqlite";
    string folderPath = System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal);
    string fullPath = Path.Combine(folderPath, dbName);

    LoadApplication(new App(fullPath));
}
```

```
public App(string databaseLocation)
{
    InitializeComponent();

    MainPage = new NavigationPage(new MainPage());
    DatabaseLocation = databaseLocation;
}
```

Une fois cela fait, il est alors très simple d'utiliser **SQLite**, car il nous suffit d'ouvrir une connexion et de modifier les tables de la sorte :

```
List<Dog> list = new List<Dog>();

using (SQLiteConnection conn = new SQLiteConnection(App.DatabaseLocation))
{
    conn.CreateTable<Dog>();
    list = conn.Table<Dog>().OrderBy(x => x.Name).ToList();
}
```

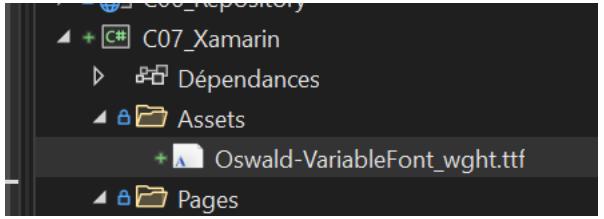
```
using (SQLiteConnection conn = new SQLiteConnection(App.DatabaseLocation))
{
    conn.CreateTable<Dog>();
    int nbRows = conn.Delete(dogToDelete);
}
```

```
0 références
private void addDogButton_Clicked(object sender, EventArgs e)
{
    Dog newDog = new Dog
    {
        Name = dogDescEntry.Text,
        Description = dogDescEntry.Text,
    };

    using (SQLiteConnection conn = new SQLiteConnection(App.DatabaseLocation))
    {
        conn.CreateTable<Dog>();
        int nbRows = conn.Insert(newDog);

        if (nbRows > 0)
        {
            DisplayAlert("Réussite", "Le chien a été ajouté avec succès dans la base de données", "Ok");
        }
    }

    Navigation.PopAsync();
}
```

Ajouter des Assets

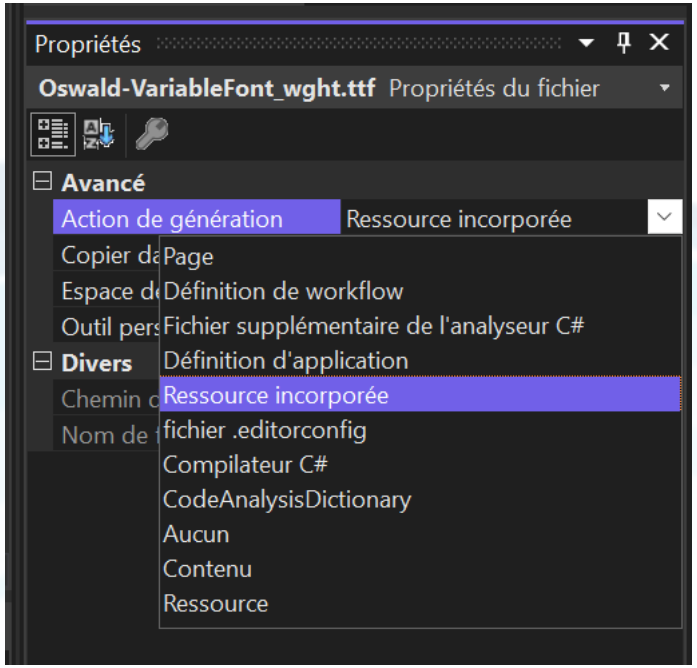
Pour ajouter une **police** dans Xamarin, il faut la placer dans un dossier prévu à cet effet, que l'on nommera en général Assets, de façon à ce qu'elle soit une ressource incorporée (**embedded ressource**), puis ajouter une référence à cette police dans l'assembly de l'application:

```
[assembly: ExportFont("Verdana.ttf", Alias = "Verdana")]
[assembly: ExportFont("Specify.ttf", Alias = "Specify")]
[assembly: ExportFont("Impact.ttf", Alias = "Impact")]

namespace C07_Xamarin
{
    4 références
    public partial class App : Application
    {
```



Une fois la police incorporée à l'application, elle sera accessible via l'utilisation de la propriété **FontFamily** d'un élement de type Label ou Entry par exemple :

```
<Label Text="Dog List Page"
       FontFamily="Oswald"
       FontSize="20"
       HorizontalTextAlignment="Center"/>
```

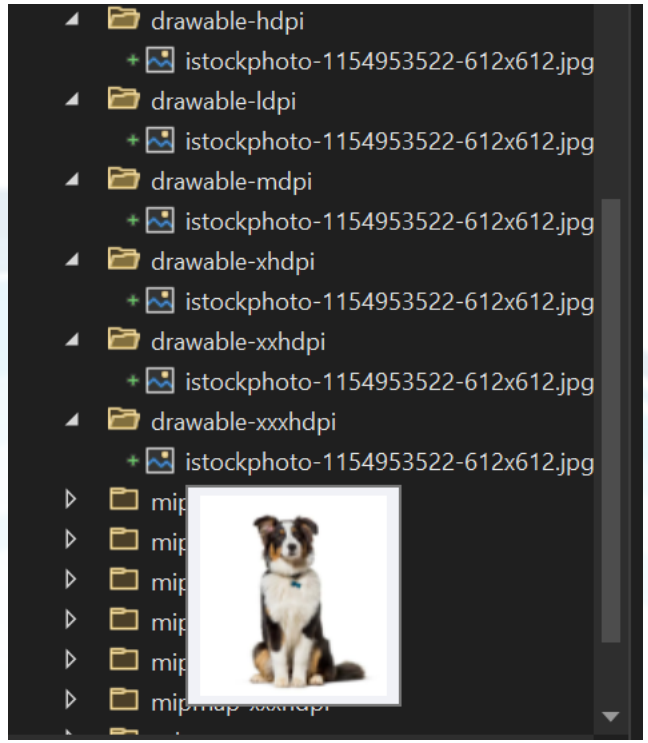


Ajouter des Images

Quand il s'agit d'images, la tâche s'avèrera plus difficile que pour les polices. En effet, que cela soit pour iOS ou pour Android, l'utilisation d'image se fait via l'ajout d'une série d'images et non d'une seule. En effet, le fait qu'une application mobile doit pouvoir fonctionner de façon optimale sur un smartphone avec une basse, moyenne ou haute résolution amène à se contraindre à cette dynamique lors de l'ajout des images.

Pour **Android**, nous devrons donc ajouter une image sous 5 formats différents dans le projet Android, puis les placer dans les dossier nommés **drawable-res**, chacun possédant un identifiant lui étant propre :

De plus, lors de l'ajout d'icones pour notre application, la même opération devra être observée, mais celle-ci concernera alors les dossiers **mipmaps-res** de résolutions diverses :



La NavigationPage

Les pages de Xamarin peuvent être vues comme des feuilles que l'on empile ou dépile en fonction de la navigation de notre application au sein d'une même **NavigationPage**, qui doit être instanciée en lieu d'une simple MainPage dans le point d'entrée de notre application, qui se trouve être **App.xaml.cs**

```
0 références
public App()
{
    InitializeComponent();

    MainPage = new NavigationPage(new MainPage());
}
```

L'utilisation d'une NavigationPage permet de se déplacer dans l'application de sorte à pouvoir passer d'une page à l'autre via l'utilisation des méthodes ***.PushAsync()*** et ***.PopAsync()*** faisant partie de la propriété des pages **Navigation**

```
    Navigation.PushAsync(new MainPage());
}
```

L'utilisation d'une NavigationPage provoque l'apparition d'une barre au dessus de notre application servant à la navigation vers les éléments précédents (une sorte de bouton retour).

Cette barre est personalisable via le XAML si l'on désire modifier par exemple le titre du bouton de retour :

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="C07_Xamarin.Pages.MainPage"
             NavigationPage.BackButtonTitle="Retour">
```

La TabbedPage

Une **TabbedPage** est une page de Xamarin.Forms servant à l'utilisation d'onglets que l'on peut naviguer en faisant glisser l'écran de façon horizontale ou en appuyant sur les onglets.

Chaque page ajoutée en contenu de cette fameuse page à onglet sera accessible via l'utilisation de l'onglet, que l'on peut agrémenter d'une image sous la forme d'une **icone**. L'accès aux **namespaces** d'**AndroidSpecific** permet de placer la barre des onglets en bas de l'écran, alors que l'ajout du namespace **local** sert à accéder aux pages que l'on cherche à relier aux onglets.

```
<TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            xmlns:android="clr-namespace:Xamarin.Forms.PlatformConfiguration.AndroidSpecific;assembly=Xamarin.Forms.Core"
            xmlns:local="clr-namespace:C07_Xamarin.Pages"
            x:Class="C07_Xamarin.Pages.HomePage"
            android:TabbedPage.ToolbarPlacement="Bottom">

    <local:DogListPage Title="Chiens" IconImageSource="Dog_Icon.png"/>
    <local:MasterListPage Title="Maîtres" IconImageSource="Master_Icon.png"/>
    <local:AdressesListPage Title="Adresses" IconImageSource="Adress_Icon.png"/>

</TabbedPage>
```

Le DataBinding

Lorsque l'on en vient à utiliser des **ListViews**, on a bien souvent recourd à ce que l'on appelle le **DataBinding**. Cette notion permet de relier des élements à une source qui est bien généralement une propriété. Pour profiter du DataBinding, il faut tout d'abord spécifié le contexte des données, que cela soit dans l'**ItemSource** d'une ListView ou dans le **BindingContext** d'une page XAML :

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

    dogsListView.ItemsSource = null;
    dogsListView.ItemsSource = App.Dogs;
}
```

Lors de l'utilisation d'un **Binding**, il faut faire attention à la casse et au nom donné à nos propriétés sous peine d'avoir des problèmes de liens.

Ici, la surcharge de la méthode ***OnAppearing()*** de la **ContentPage** sert à permettre le rafraichissement des valeurs contenues dans la **ListView**, sous peine de devoir quitter et revenir sur la page pour voir le changement s'opérer.

```
<ListView.ItemTemplate>
    <DataTemplate>
        <TextCell Text="{Binding Name}" Detail="{Binding Description}" TextColor="DodgerBlue"/>
    </DataTemplate>
</ListView.ItemTemplate>
</ListView>
```

```
public class Dog
{

    public int Id { get; set; }
    public static int Count;

    public string Name { get; set; }
    4 références
    public string Description { get; set; }

    3 références
    public Dog()
    {
        Id = ++Count;
    }
}
```