Développement d’application Xamarin

XAMARIN AVANCÉ

NuGet & Xamarin Store

NuGet : ne réinventez pas la roue

- http://nuget.org/

- +30’000 librairies .NET à votre disposition.
- Beaucoup sont compatibles avec Xamarin !

- S’accompagne d’un gestionnaire de package dans Xamarin Studio et Visual Studio.

- Gère les dépendances automatiquement !

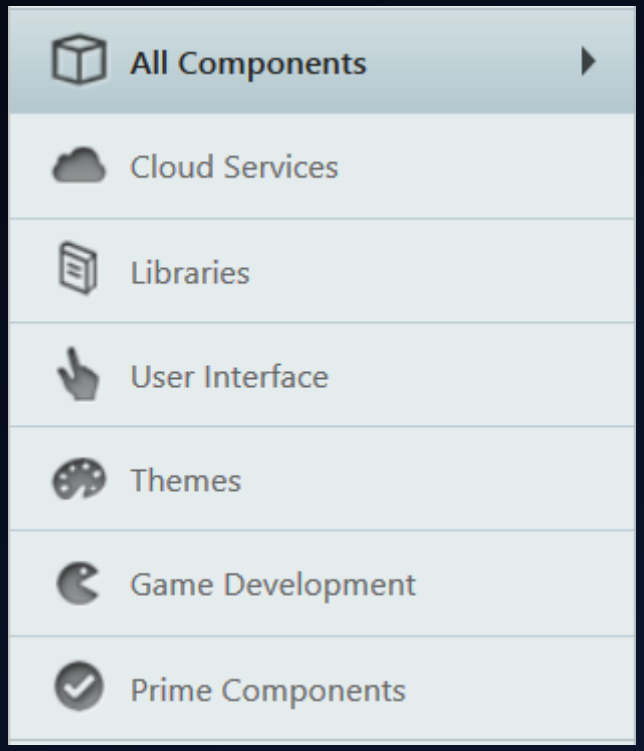
NuGet : les packages indispensable

- Json.Net : sérialization / déserizalisation Json très rapide
- PCL Storage : accèder au système de fichier en crossplatform
- Xamarin.Forms : vous l'avez déjà ☺
- Microsoft HTTP Client Libraries : requête http crossplatform
- XLabs - * : des composants très utiles avec Xamarin (préversion)
- Log4net : vous permet de logger vos erreurs

- Storm.Mvvm.Forms : vous simplifie la vie pour le MVVM (vous n'êtes pas obligés de l'utiliser)

Xamarin Components Store

- https://components.xamarin.com/
- Des composants spécialement prévus pour Xamarin
- Disponible sur iOS, Android et/ou Windows Phone
- Beaucoup sont gratuits, n'hésitez pas à les utiliser !



Github

- Beaucoup de plugins, librairies & exemples sont disponibles sur GitHub

- Exemple Xamarin.Forms
- https://github.com/xamarin/xamarin-forms-samples

- Plein d'exemples Xamarin & Xamarin.Forms
- https://github.com/jamesmontemagno

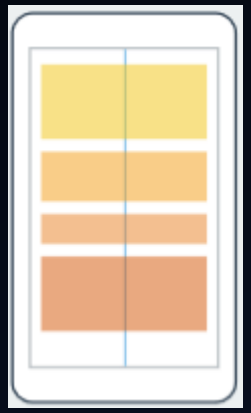
- http://developer.xamarin.com/guides/cross-platform/xamarin-forms/creating-mobile-apps-xamarin-forms/

Xamarin.Forms : Layouts

StackLayout

- Le layout le plus simple à utiliser

- Vous permet d'ajouter des éléments en horizontal ou vertical
  - Orientation="Vertical" / Orientation="Horizontal"



```
<StackLayout Orientation="Vertical">
    <Label Text="Label1"/>
    <Label Text="Label2"/>
</StackLayout>

<StackLayout Orientation="Horizontal">
    <Label Text="Label1"/>
    <Label Text="Label2"/>
</StackLayout>
```

Grid

- La Grid vous permet de positionner vos éléments en les alignant sur une grille dont vous fixez le nombre de colonnes et de lignes.

- Auto : prend la place nécessaire
- * : toute la place disponible
- 42 : 42 pixels de large (déconseillé)

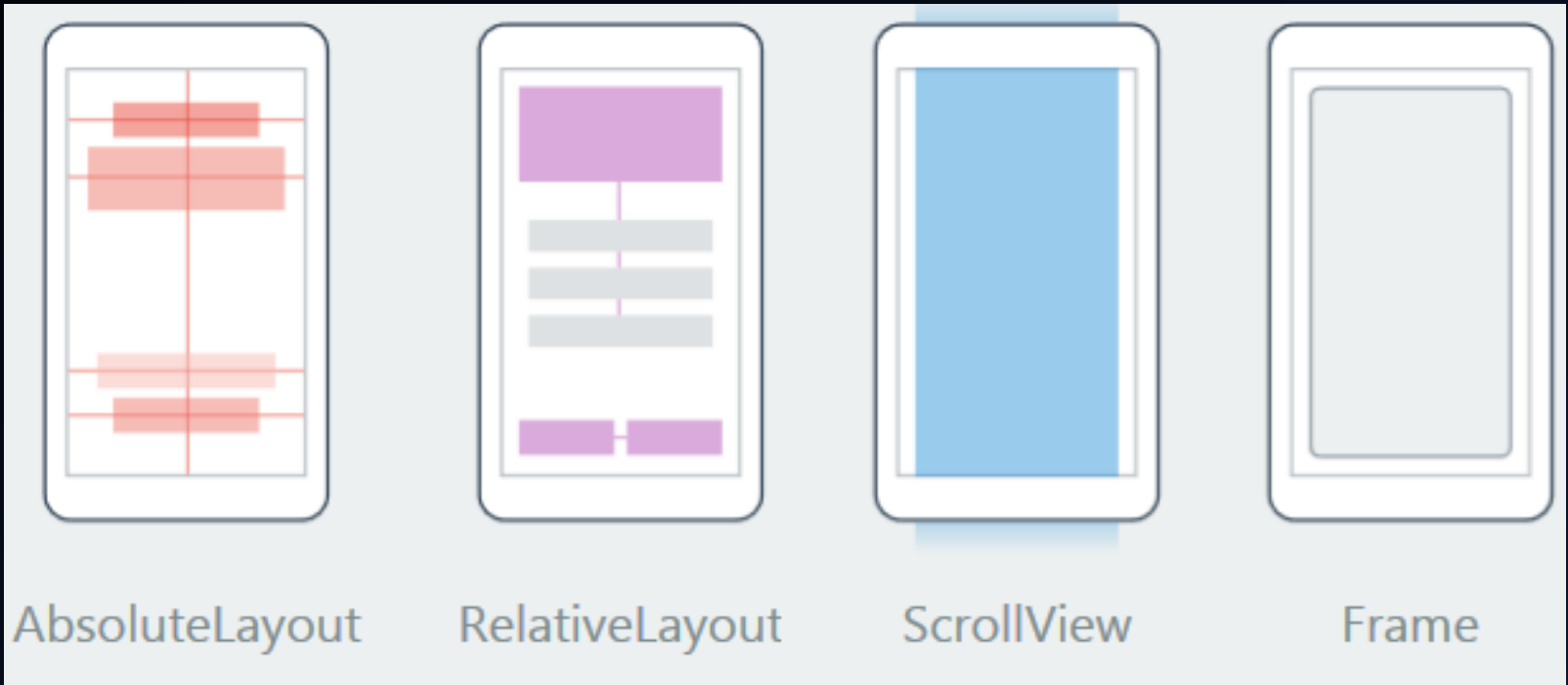
```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="42" />
        <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>

    <Label Text="Title"
           Grid.Row="0"
           Grid.Column="0"
           Grid.ColumnSpan="2"
           />
    <StackLayout Grid.Row="1"
                 Grid.Column="0"/>
    <StackLayout Grid.Row="1"
                 Grid.Column="1"/>
</Grid>
```

Grid

- Et comment je peux faire si je veux deux colonnes avec la première qui prend 1/3 de l'écran et la seconde, les 2/3 restants ?
- La solution consiste à préfixer le symbole étoile par un nombre.
- Dans notre cas, on mettrait la première colonne à 1* et la deuxième colonne à 2*.
- L'étoile devient en fait une variable x ce qui nous donne une équation simple de la forme 1x + 2x = largeur_écran
- *Avantage* : vous pouvez faire des interfaces qui s'adaptent à la taille de l'écran de l'utilisateur.

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="1*" />
        <ColumnDefinition Width="2*" />
    </Grid.ColumnDefinitions>
```

Et les autres



Xamarin.Forms : Ressources

Ressources

- Vous pouvez définir des ressources pour votre page (ou votre application complète).
- Par exemple : couleur, converters, template, …
- Très utiles pour changer facilement le « thème » de son application.
- Évitent également la duplication de code pour l'affichage avec l'utilisation de Template.

Ressources : pour une page

- Tous les éléments que vous pouvez mettre dans votre Xaml possèdent un attribut Resources (ici on prendra exemple sur la ContentPage)
- Cet attribut Resources doit prendre comme valeur un ResourceDictionnary et c'est dans ce ResourceDictionary que vous mettez toutes vos ressources.

```
<ContentPage.Resources>
    <ResourceDictionary>
        <converters:BooleanNegationConverter x:Key="BooleanNegationConverter" />
    </ResourceDictionary>
</ContentPage.Resources>
```

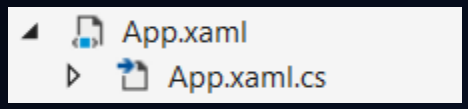
Ressources : pour l'application

- Contraintes :
  - Vous devez associer un fichier Xaml à votre App.cs de base
  - Vous devez hériter de MvvmApplication et non MvvmApplication<T>
  - Vous devez créer « manuellement » la page de lancement de votre application avec InitializeMainPage.

- Comment faire :
  - Supprimer votre App.cs automatiquement généré à la création du projet.
  - Créer une Forms Xaml page et changer l'héritage dans le fichier C# par MvvmApplication.
  - Après le InitializeComponent, appelez la fonction InitializeMainPage en lui donnant en paramètre votre page de démarrage.
  - Changer votre tag racine dans votre fichier Xaml par MvvmApplication (ajouter le namespace qui convient).

Ressources : pour l'application

- Deux fichier : App.xaml & App.xaml.cs

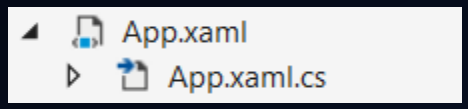


App.xaml

```
<mvvm:MvvmApplication xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:mvvm="clr-namespace:Storm.Mvvm;assembly=Storm.Mvvm.Forms"
             xmlns:converters="clr-namespace:TodoListFull.Converters;assembly=TodoListFull"
             x:Class="TodoListFull.App">
    <mvvm:MvvmApplication.Resources>
        <ResourceDictionary>
            <converters:BooleanNegationConverter x:Key="BooleanNegationConverter"/>
        </ResourceDictionary>
    </mvvm:MvvmApplication.Resources>
</mvvm:MvvmApplication>
```

Ressources : pour l'application

- Deux fichiers : App.xaml & App.xaml.cs



App.xaml.cs

```
public partial class App : MvvmApplication
{
    public App()
    {
        InitializeComponent();
        InitializeMainPage<HomePage>();
    }
}
```

Ressources : comment y accéder ?

- Toute vos ressources ont un attribut x:Key qui sert de clé pour les identifier.
- Deux ressources ne peuvent pas avoir la même clé (sauf si elles sont dans deux pages différentes)

- Pour s'en servir, {StaticResource Ma_Clé_De_Ressource}

Xamarin.Forms : ListView

ListView

- Composant vous permettant d'afficher des éléments sous forme de liste de taille (presque) infinie
- Très populaire dans les applications mobiles
- Peut être relié dans votre ViewModel à :
  - Une List<T> => si la liste n'est pas amenée à changer (pas d'ajout/suppression)
  - Une ObservableCollection<T> => si vous souhaitez ajouter/supprimer des éléments dans la collection

ListView

- Propriétés importantes
  - ItemSource => à relier à la collection de votre ViewModel
  - SelectedItem => vous permet de récupérer l'élément sélectionné dans votre ViewModel
  - ItemTemplate => définit la manière d'afficher chaque élément de la liste

```
<ListView ItemsSource="{Binding Items}"
          SelectedItem="{Binding SelectedItem, Mode=TwoWay}"
          ItemTemplate="{StaticResource ListTemplate}"
          />
```

```
<ContentPage.Resources>
    <ResourceDictionary>
        <DataTemplate x:Key="ListTemplate">
            <ViewCell>
                <Label Text="{Binding Title}"
                       FontSize="20"
                       />
            </ViewCell>
        </DataTemplate>
    </ResourceDictionary>
</ContentPage.Resources>
```

ListView : Evenement

- Comment réagir quand l'utilisateur sélectionne un élément ?
- Solution 1 : La ListView dispose d'un event ItemSelected.
  - Problème : on ne peut pas lier un événement avec MVVM !
- Solution 2 : Utiliser un Binding TwoWay sur le SelectedItem

```
SelectedItem="{Binding SelectedItem, Mode=TwoWay}"
```

```
public TodoItem SelectedItem
{
    get { return _selectedItem; }
    set
    {
        if (SetProperty<TodoItem>(ref _selectedItem, value))
        {
            if (value != null)
            {
                OnItemSelected(value);
            }
        }
    }
}
```

Platform customization

Platform customization

- *Cas pratique* : sur iOS 7+, la status bar (info réseaux) passe sur votre application. La solution : décaler votre application de 20 pixel vers le bas.
- *Solution* : mettre un padding « 0,20,0,0 » sur le conteneur principal de vos pages
- *Problème* : comment faire pour que ce soit que sur iOS et pas Windows Phone & Android ?
- *Solution* : tag xml OnPlatform

Platform customization : Xaml

- x:TypeArguments : le type (ici Thickness pour le Padding) de la propriété
- Android : valeur pour Android
- WinPhone : valeur pour WindowsPhone
- iOS : valeur pour iOS

```
<OnPlatform x:TypeArguments="Thickness"
            x:Key="RootPadding"
            Android="0,0,0,0"
            WinPhone="0,0,0,0"
            iOS="0,20,0,0"
            />
```

```
<StackLayout
    Padding="{StaticResource RootPadding}">
```

```
<StackLayout>
    <StackLayout.Padding>
        <OnPlatform x:TypeArguments="Thickness"
                    Android="0,0,0,0"
                    WinPhone="0,0,0,0"
                    iOS="0,20,0,0"
                    />
    </StackLayout.Padding>
```

Platform customization : C#

- Propriété OS de la classe Device vous donne l'OS en cours

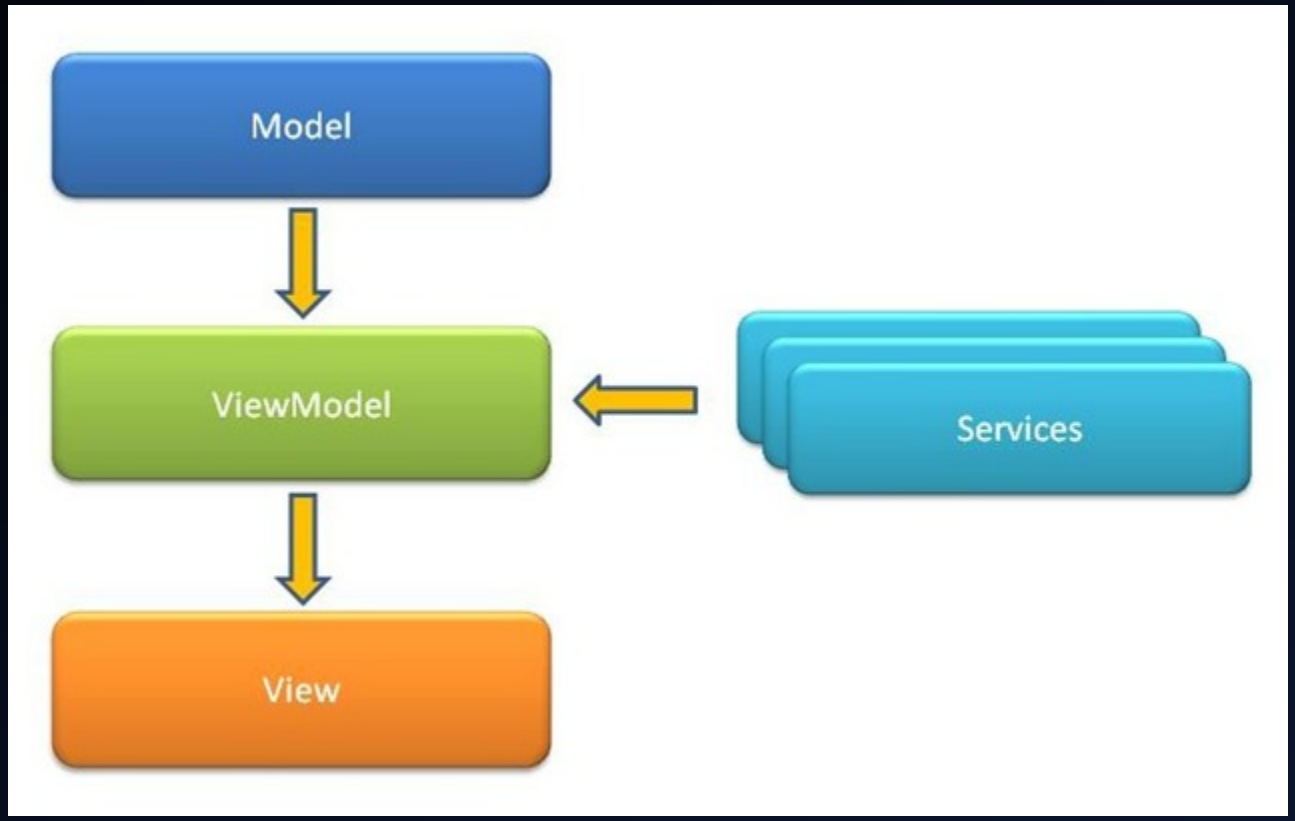
```
if (Device.OS == TargetPlatform.iOS)
{
    //iOS code
}
else if (Device.OS == TargetPlatform.Android)
{
    //Android code
}
else if (Device.OS == TargetPlatform.WinPhone)
{
    //WindowsPhone code
}
```

IoC : Inversion of Control

MVVM & Services

- Les services sont les composants gérant toute la partie Business de votre application.
- Les services ont pour but d'être utilisés dans plusieurs ViewModel de votre application.
- Un service = une tâche (communication avec un webservice, gestion de la persistance, etc…)
- En découpant ainsi votre application, vous créez des briques indépendantes et facilement testables (Unit test)

MVVM & Services



MVVM & Services

- Pour définir un service, définissez l'interface de votre service ainsi que son implémentation.
- Une même interface de service peut être implémentée plusieurs fois.
- Exemple : gestion des settings de votre application (Load, Save, Get, Set) avec deux implémentations :
  - La première sauvegarde dans un fichier json
  - La seconde sauvegarde sur un serveur distant via l'utilisation d'un WS Rest
- Avantage : l'appelant n'a pas besoin de savoir quelle implémentation est utilisée.

MVVM & Services

- Afin de transmettre les services aux ViewModels, nous utilisons un système d'IoC.
- Au démarrage de votre application, vous créez vos différents services et les enregistrez dans le conteneur d'IoC.
- Quand vous en avez besoin, vous pouvez les récupérer dans ce conteneur d'IoC à partir de vos ViewModels.

IoC : Inversion of Control

- Xamarin.Forms vous donne accès à un conteneur d'IoC, mais qu'est-ce qu'un conteneur d'IoC ?

- L'IoC vous permet d'enregistrer pour une interface, l'implémentation à utiliser quand on lui demandera.

- Grâce à cela, vous pouvez très simplement changer l'implémentation à utiliser pour un service et ce, sans aucun impact sur le reste de votre code.

IoC : Inversion of Control + Xamarin.Forms

- Le conteneur d'IoC fournit par Xamarin.Forms est accessible via la classe statique DependencyService.

- Enregistrement d'un service (son interface et l'implémentation)

```
DependencyService.Register<ITodoService, TodoService>();
```

- Récupération de l'instance du service (via son interface)

```
DependencyService.Get<ITodoService>();
```

- Pour fonctionner avec l'IoC de Xamarin.Forms, votre service doit avoir un constructeur sans paramètre.

IoC : Inversion of Control + MVVM

- Pour enregistrer vos services avec Mvvm

```
public class App : MvvmApplication<HomePage>
{
    public static void RegisterServices()
    {
        DependencyService.Register<ITodoService, TodoService>();
    }

    public App() : base(RegisterServices)
    {

    }
```

- Ou en utilisant une lambda

```
public App() : base(() =>
    {
        DependencyService.Register<ITodoService, TodoService>();
    })
{

}
```

IoC : Inversion of Control

- Grâce à l'IoC, vous pouvez tout à fait définir un service qui devra être implémenté sur chaque plateforme (exemple : envoi de SMS, appel, …)

- Vous pourrez ensuite utiliser ce service de manière transparente depuis vos ViewModels puisqu'il n'ont pas à connaître l'implémentation sous-jacente.

- Grâce à l'IoC, vous gardez un code crossplatform tout en pouvant personnaliser l'implémentation pour chaque plateforme.

Questions ?