MAUI

m2iformation.fr

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

MAUI

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Présentation de .NET MAUI

Introduction à .NET MAUI

- **MAUI** (Multi-platform App UI) : **framework** open-source de **développement d'applications multiplateformes** pour **Android**, **iOS**, **macOS**, et **Windows**
- Pourquoi choisir .NET MAUI pour le développement multiplateforme ?
  - Environnement .NET (BCL, ...)
  - UI adaptable aux différentes plateformes
  - Bibliothèques natives augmentant la performance
  - Prise en charge native des fonctionnalités par plateforme

Évolution de Xamarin.Forms vers .NET MAUI

MAUI est l'évolution de **Xamarin Forms**, technologie à la base centrée sur le développement mobile.

Aspect	Xamarin.Forms	.NET MAUI
Plateformes prises en charge	Multiplateforme (Android, iOS, macOS, Windows, ...)	Multiplateforme (Android, iOS, macOS, Windows, ...)
Interface utilisateur unifiée	Non	Oui
Réutilisation de code	Moins réutilisable	Hautement réutilisable
Support .NET 6	Non (jusque .NET 5)	Oui
Architecture	Multi Project	Single Project

MAUI Classique ou MAUI avec Blazor Hybrid ?

Caractéristique	Projet Maui classique	Projet Maui Blazor Hybrid
Type d'application	Native	Web
Langage de programmation	C#	C#
Visuel/Rendu	XAML	Razor components
Cadre de développement	Xamarin.Forms	Blazor
Exécution	Code natif	JavaScript
Performances	Optimisées pour chaque plateforme	Variable, en fonction de la plateforme et du navigateur
Fonctionnalités	Spécifiques à chaque plateforme	Générales, communes à tous les navigateurs web

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Projet .NET MAUI

Structure d'un projet MAUI

- **Platforms** : projets de lancement des différents environnements, utile pour définir des comportement adaptés par plateforme
- **MauiProgram.cs** : point d'entrée global (multi-plateforme)
- **App** : classe principale, responsable du démarrage de l'application et de la gestion de son cycle de vie
- **AppShell** : classe qui fournit une interface utilisateur de base ainsi que le nécessaire pour la navigation
- **MainPage** : page principale de l'application, chargée par AppShell
- **Ressources** : Images, Polices, Styles, Splash (image de chargement), Raw (données brutes style video/audio/binaire) ...

Fichier Projet (.csproj)

Le **fichier projet** d'une application MAUI est différent des autres.
On y retrouve des **éléments supplémentaires** tel que :

- **Nom** et **identifiants** globaux de l'applications
- Configurations des **versions par plateformes**
- Configurations des **Ressources** (Splash, images, icône, ...)
- Instructions de compilation des pages/views XAML

Comprendre MainPage.xaml

Analyse de la structure XAML de la page principale :

- **ContentPage** : une page de contenu dans une application
- **ScrollView** : une vue qui peut être défilée (scroll)
- **VerticalStackLayout** : disposition pour empiler verticalement des éléments
- **Image** : balise pour une image
- **Label** : balise pour du texte
- **Button** : bouton cliquable

Propriétés SemanticProperties : servent à l'accessibilité (lecteur d'écran, ...)

nous ne les utiliseront pas par la suite dans ce cours

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Interfaces Utilisateur (UI)

Page et Ficher Code-Behind

Lors de l'utilisation d'MAUI, les pages seront décomposées en **2 fichiers distincts** :

- **.xaml** : pour définir la partie visuelle d'une page
- **.cs** : pour définir le fonctionnement (**Code-Behind**)

Le constructeur comprendra toujours la méthode `InitializeComponent()` pour préparer la page.

Les fichiers XAML peuvent aussi permettre de définir des **dictionnaires de ressources**, exemple: Styles.xaml et Colors.xaml

XAML

XAML est un **langage de balisage** déclaratif utilisé pour définir l'**interface utilisateur d'une application**. Il est utilisé par de nombreux frameworks de développement d'applications, notamment **.NET MAUI**, **Xamarin/UWP** et **WPF**.

XAML est un langage déclaratif, ce qui signifie qu'**il décrit ce que l'interface utilisateur doit faire**, plutôt que comment elle doit le faire. Cela permet aux développeurs de **se concentrer sur la conception de l'interface utilisateur**, plutôt que sur les détails de sa mise en œuvre.

Contrôles/Composants MAUI

On retrouve dans le XAML de MAUI beaucoup de **composants** présents aussi dans d'autre frameworks permettant la **création d'interface graphique**.

Listes des contrôles/composants

Propriétés courantes des composants

Text : texte à afficher
TextColor : couleur du texte
FontSize : taille de la police
FontFamily : police
FontWeight : poids de la police (gras)
FontStyle : style de la police
TextAlignment : alignement du texte (gauche, droite, centré, ...)

Margin : marges du composant
Padding : rembourrage du composant
HorizontalOptions : alignement horizontal du composant
VerticalOptions : alignement vertical du composant
IsEnabled : activé/utilisable
IsVisible : visible

Méthodes de définitions de propriétés

Attributs xml

```
<Label Text="Hello, XAML!"
       VerticalOptions="Center"
       FontAttributes="Bold"
       FontSize="18"
       TextColor="Aqua" />
```

Balises

```
<Label Text="Hello, XAML!"
       VerticalOptions="Center">
    <Label.FontAttributes>
        Bold
    </Label.FontAttributes>
    <Label.FontSize>
        Large
    </Label.FontSize>
    <Label.TextColor>
        Aqua
    </Label.TextColor>
</Label>
```

Label

Un label est utilisé pour afficher du texte. Le texte est défini sur la propriété Text du label.

```
<Label Text="Ce label affiche du texte." />
```

Button

Un bouton est utilisé pour déclencher une action.

Propriété Clicked : Définit la méthode à appeler au click.

```
<Button Clicked="OnButtonClicked" Text="Ce bouton déclenche une action." />
```

IsToggle *et* ***IsChecked*** *pour un bouton à bascule.*

Entry

Un entry est utilisé pour saisir du texte.

Propriété Text : Définit la valeur par défaut du texte saisi.

```
<Entry Text="Entrez du texte ici." />
```

BoxView

Une BoxView est utilisée pour afficher un rectangle servant surtout au visuel, par exemple pour faire des bares de séparation.

```
<BoxView Color="Blue" CornerRadius="10" WidthRequest="160" HeightRequest="50" />
```

Picker

Un picker est utilisé pour choisir une valeur **textuelle** dans une liste.

Propriété SelectedIndex : Permet de récupérer la valeur.

```
<Picker x:Name="picker"
        Title="Select a monkey">
  <Picker.ItemsSource>
    <x:Array Type="{x:Type x:String}">
      <x:String>Baboon</x:String>
      <x:String>Capuchin Monkey</x:String>
      <x:String>Blue Monkey</x:String>
      <x:String>Squirrel Monkey</x:String>
      <x:String>Golden Lion Tamarin</x:String>
      <x:String>Howler Monkey</x:String>
      <x:String>Japanese Macaque</x:String>
    </x:Array>
  </Picker.ItemsSource>
</Picker>
```

ListView et CollectionView

Une ListView est utilisée pour choisir une valeur **objet** parmi une liste.

```
<ListView x:Name="TodoListView"
          Margin="20"
          ItemSelected="OnTodoListItemSelected">
    <ListView.ItemTemplate>
        <DataTemplate>
            <ViewCell>
                <StackLayout Margin="20,0,0,0" Orientation="Horizontal" >
                    <Label Text="{Binding Name}" />
                    <Label Text="Terminé" IsVisible="{Binding Done}" />
                </StackLayout>
            </ViewCell>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

```
public partial class TodoListPage : ContentPage
{
    public TodoListPage()
    {
        InitializeComponent();
        TodoListView.ItemsSource = new List<TodoItem>
        {
            new TodoItem() {Name = "item1", Done = false},
            new TodoItem() {Name = "item2", Done = true},
            new TodoItem() {Name = "item3", Done = false},
            new TodoItem() {Name = "item4", Done = false},
            new TodoItem() {Name = "item5", Done = true},
        };
    }
}
```

Ajout de comportement lié aux contrôles

Certaines **propriétés XAML** des contrôles permettent de **lier un comportement spécifique à une méthode** selon certains évènements du contrôle.

Lorsque l'on travaille avec nos contrôles, il faut leur **donner un nom** avec la **propriété XAML** `x:Name` pour pouvoir les utiliser en temps que **propriété de la classe CS**.

L'exemple le plus courant est le click d'un bouton :

```
<Button
    x:Name="CounterBtn"
    Text="Click me"
    Clicked="OnCounterClicked"
    HorizontalOptions="Center" />
```

```
private async void OnCounterClicked(object sender, EventArgs e)
{
        count++; // attribut privé int count = 0;

        if (count == 1)
                CounterBtn.Text = $"Clicked {count} time";
        else
                CounterBtn.Text = $"Clicked {count} times";
}
```

Gestion des événements

La méthode appelée aura toujours 2 paramètres d'entrée :

- `object sender` : correspond au contrôle qui a déclenché l'évènement
- `EventArgs e` : correspond à l'évènement en lui même

On pourra définir ainsi un comportement spécifique relatif au contrôle si besoin.

Mise en pages différentes selon les plateformes

```
<Label
    Text="Welcome to .NET MAUI!"
    VerticalOptions="Center"
    HorizontalOptions="Center" >
    <Label.TextColor>
        <OnPlatform x:TypeArguments="Color">
            <On Platform="Windows" Value="Red" />
            <On Platform="Android" Value="Blue" />
        </OnPlatform>
    </Label.TextColor>
</Label>
```

Documentation

Bases sur la liaison de données

Documentation Formater des Strings

```
<Label Text="ROTATION"
       BindingContext="{x:Reference slider}"
       Rotation="{Binding Value}"
       FontAttributes="Bold"
       FontSize="20"
       HorizontalOptions="Center"
       VerticalOptions="Center" />
<Slider x:Name="slider"
        Maximum="360"
        VerticalOptions="Center"
        Margin="20,0,20,0"
        Value="172"/>
<Slider x:Name="slider2"
        BindingContext="{x:Reference slider}"
        Value="{Binding Value, Mode=TwoWay}"
        Maximum="360"
        VerticalOptions="Center"
        Margin="20,0,20,0"
        />
<Label BindingContext="{x:Reference slider}"
       Text="{Binding Value, StringFormat='The angle is {0:F0} degrees'}"
       HorizontalOptions="Center"
       VerticalOptions="Center" />
```

Liaison de données avancée (exemple)

```
<VerticalStackLayout>
    <ListView x:Name="TodoListView"
              Margin="20"
              ItemSelected="OnTodoListItemSelected"
              ItemsSource="{Binding}">
              <!-- Relié au BindingContext donc TodoList -->
        <ListView.ItemTemplate>
            <DataTemplate>
                <ViewCell>
                    <VerticalStackLayout>
                        <Label  Text="{Binding Name}"/>
                        <Label  Text="{Binding Done}"/>
                    </VerticalStackLayout>
                </ViewCell>
            </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>
</VerticalStackLayout>
```

```
// ObservableCollection est utilisé lorsque
// l'on veut une mise à jour en temps réel
public ObservableCollection<TodoItem> TodoList { get; set; }
public TodoListPage()
{
    // définit le Binding de base pour toute la view
    BindingContext = TodoList = new ObservableCollection<TodoItem>
    {
        new TodoItem() {Name = "item1", Done = false},
        new TodoItem() {Name = "item2", Done = true},
        new TodoItem() {Name = "item3", Done = false},
        new TodoItem() {Name = "item4", Done = false},
        new TodoItem() {Name = "item5", Done = true},
    };
    InitializeComponent();
}
```

Ce cours est avant tout une initiation, pour aller plus loin, il est recommandé d'implémenter le **pattern MVVM (Model View ViewModel)** pour le **Data Binding**. => Tutoriel de microsoft

Notions supplémentaires utiles

- Couleurs et Dégradés
- Images
- Pop-ups
- Boutons de la barre d'outils
- Tooltips
- Ombres
- Polices
- Animations
- Définir des Thèmes
- Styles avec XAML
- Styles avec CSS

Exercice

Avec un design libre, réaliser un jeu du nombre mystère avec au minimum :

- 1 Label expliquant le jeu,
- 1 Entry pour la saisie,
- 1 Button pour tester le nombre,
- 1 Label ne s'affichant qu'a la validation avec du texte qui varie en contenu et en couleur en fonction de la saisie de l'utilisateur

Prendre en compte les cas ou l'utilisateur ne saisi pas un nombre entier et afficher un message d'erreur de saisie dans le label résultat.

Pour aller plus loin, utiliser un Layout au choix pour avoir un affichage propre

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Layouts

Types de layouts

- **Absolute** : Positionnement **absolut** avec coordonnées
- **StackLayouts** : En "**pile**" avec les éléments qui se suivent selon une direction. Il existe un type de base et 2 version améliorées selon la direction.
- **Grid** : En **grille** avec utilisation de colonnes et de lignes
- **FlexLayout** : **Flexible**, selon les mêmes principes qu'en CSS.

Chaque Layout sauf l'absolut a la possibilité d'être de type **BindableLayout**, permettant un affichage **dynamique** selon une **collection**, similairement à un ListView.

Absolute

cf documentation

StackLayouts

cf documentation

StackLayout

Versions optimisées :

VerticalStackLayout

HorizontalStackLayout

Grid

```
<Grid RowSpacing="6" ColumnSpacing="6">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
            <RowDefinition Height="100" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="100" />
        </Grid.ColumnDefinitions>

        <Label Text="Autosized cell"
               TextColor="White"
               BackgroundColor="Blue" />
        <BoxView Color="Silver"
                 Grid.Column="1" />
        <BoxView Color="Teal"
                 Grid.Row="1" />
        <Label Text="Leftover space"
               Grid.Row="1" Grid.Column="1"
               TextColor="Purple"
               BackgroundColor="Aqua"
               HorizontalTextAlignment="Center"
               VerticalTextAlignment="Center" />
        <Label Text="Span two rows (or more if you want)"
               Grid.Column="2" Grid.RowSpan="2"
               TextColor="Yellow"
               BackgroundColor="Blue"
               HorizontalTextAlignment="Center"
               VerticalTextAlignment="Center" />
        <Label Text="Span two columns"
               Grid.Row="2" Grid.ColumnSpan="2"
               TextColor="Blue"
               BackgroundColor="Yellow"
               HorizontalTextAlignment="Center"
               VerticalTextAlignment="Center" />
        <Label Text="Fixed 100x100"
               Grid.Row="2" Grid.Column="2"
               TextColor="Aqua"
               BackgroundColor="Red"
               HorizontalTextAlignment="Center"
               VerticalTextAlignment="Center" />
    </Grid>
```

FlexLayout

cf documentation

BindableLayout (avancé)

cf documentation

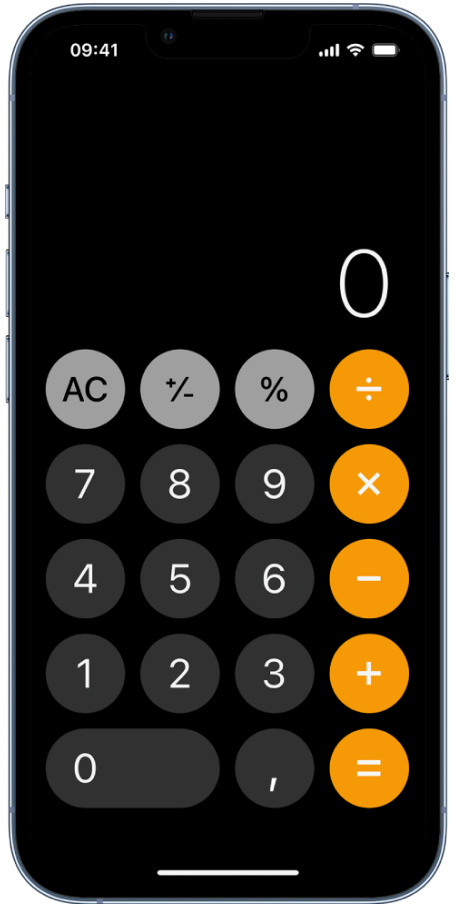
ScrollView

Un ScrollView est utilisé pour **pouvoir scroller** dans du **contenu** qui est **trop grand** pour tenir dans **son unique élément enfant** (pas possible d'en avoir plusieurs). Par défaut, la **direction** du scroll est **verticale** mais il est possible de la changer avec la propriété **Orientation**. Il s'applique aussi aux Layouts non-absolut.

```
<ScrollView HeightRequest="100" >
    <StackLayout>
        <Label Text="FOR the most wild, yet most homely narrative which I am about to pen, I neither expect nor solicit belief.
Mad indeed would I be to expect it, in a case where my very senses reject their own evidence.
Yet, mad am I not -- and very surely do I not dream. But to-morrow I die, and to-day I would unburthen my soul.
My immediate purpose is to place before the world, plainly, succinctly, and without comment, a series of mere household events.
In their consequences, these events have terrified -- have tortured -- have destroyed me.
Yet I will not attempt to expound them. To me, they have presented little but Horror --
to many they will seem less terrible than barroques.
Hereafter, perhaps, some intellect may be found which will reduce my phantasm to the common-place -- some intellect more calm, more logical,
and far less excitable than my own, which will perceive,
in the circumstances I detail with awe, nothing more than an ordinary succession of very natural causes and effects." />
        <!-- More Label objects go here -->
    </StackLayout>
</ScrollView>
```

Exercice

Avec un **Grid** dans un premier temps, puis avec des **StackLayout**s dans un second temps, reproduire l'**interface graphique** de la **calculatrice** de votre téléphone en essayant d'être **le plus fidèle possible**.



Pour aller plus loin, rendez la fonctionnelle avec vos compétences actuelles

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Pages, Shell et Navigation

ContentPage

Lorsque l'on utilise MAUI, nos applications consistent quasiment toujours en **plusieurs pages**. Jusqu'ici nous n'avons utilisé que la ContentPage, une page basique servent à **contenir** des **éléments à afficher**.

Il existe 2 façon de rédiger les interfaces des ces pages:

- avec un fichier **xaml** et un fichier code-behind **cs**
- avec un **unique fichier cs** dans lequel on instanciera les contrôles avec leur propriétés avant d'assigner la propriété `Content` à un unique contrôle parent des autres. Exemple :

```
Content = new StackLayout { Children = { new Label() { Text = "texte1" }, new Label() { Text = "texte2" } } };
```

Les autres Pages

D'après la documentation, il existe 3 autres types de Pages :
NavigationPage, FlyoutPage, TabbedPage

/!\ Cependant, ces dernières ne servent et ne fonctionnent que dans le cas où nous n'utilisons pas de Shell, mais cela change totalement le fonctionnement de notre application.

Push et Pop

Basé sur le même fonctionnement que la NavigationPage mais utilise quand même le Shell lorsqu'il est présent dans l'application.
Ici, on utilisera toujours des ContentPages.

De plus, lors d'un push, à l'instanciation d'une page, il est possible de lui **passer des données** avec son **constructeur** ou ses **propriétés**.

Exercice

A l'aide de plusieurs pages de type `ContentPages` et des notions de **push et pop**, réaliser **un quiz de 10 questions** avec un thème au choix ayant chacune **4 options** ou une **saisie à remplir**.

- A chaque réponse **valide**, on passe à la **question suivante**.
- Si une réponse **invalide**, on retourne à la question précédente. Dans un second temps, revenez au début du quiz en cas d'erreur en utilisant la méthode `PopToRootAsync`.
- Après la 10e question, affichez une image au choix à votre utilisateur en le félicitant

Shell

Le **Shell**, nommé dans le template par défaut `AppShell` fournit une "coquille" pour notre application avec une logique intégrée.
On retrouvera plusieurs **contrôles** en son sein pour faciliter la navigation entre les pages selon différents modes :

- **FlyoutItem** ou **TabBar**, permettant de définir des méthodes de navigations différentes, avec un **menu "volant"** à gauche ou avec une **barre d'onglets** fixée en bas de l'écran
- **Tab**, permettant de regrouper des sous ensembles dans les FlyoutItem ou TabBar
- **ShellContent**, les pages en elles-mêmes

Shell vs Pages

	Pages	Shell
Navigation	Chaque page est une unité de navigation autonome	La navigation est gérée par le shell
Structure	Les pages sont organisées de manière hiérarchique	Le shell fournit une structure arborescente de pages avec un système de routes
Performances	Les pages peuvent être lourdes pour les applications complexes.	Le shell est plus efficace pour les applications complexes.
Contrôles Menu	FlyoutPage	FlyoutItem MenuItem
Contrôles Onglets	TabbedPage	TabBar

Namespaces

Dans le **Shell**, lorsque l'on utilise des **views** de type **pages** qui sont dans des **namespaces particuliers** (exemple: dossier View ou Page), il faudra penser à ajouter une propriété au shell comme celles-ci :

```
<Shell
    ...
    xmlns:local="clr-namespace:DemosMAUI"
    xmlns:pages="clr-namespace:DemosMAUI.Pages"
/>
```

Ainsi, à l'appel de ces pages dans un ShellContent, on aura :

```
<ShellContent ContentTemplate="{DataTemplate local:MainPage}"/>
<ShellContent ContentTemplate="{DataTemplate pages:UneSuperPage}"/>
```

Flyout

- FlyoutItem
- Tab
- MenuItem et comportement spécifique
- Header/Footer

penser à retirer `Shell.FlyoutBehavior="Disabled"`

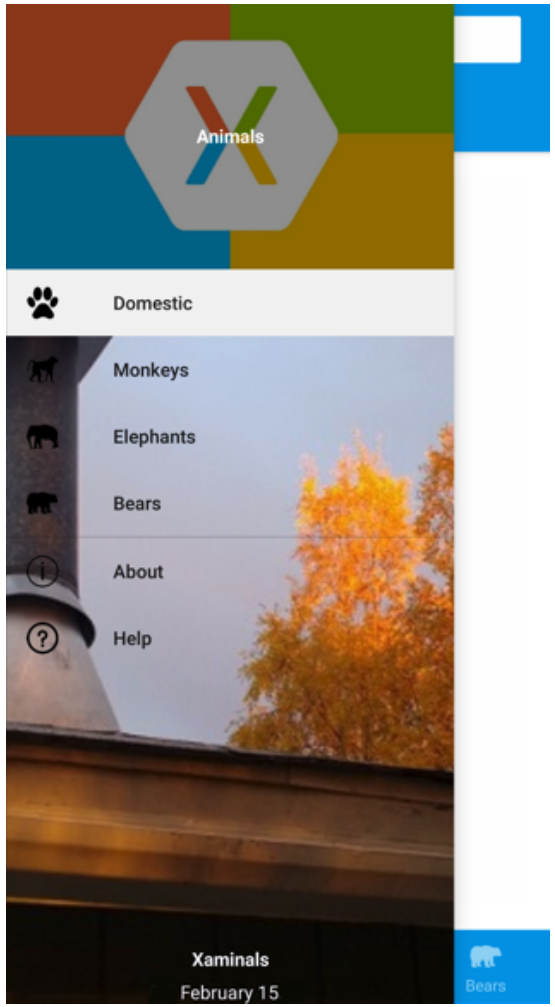
Tabs

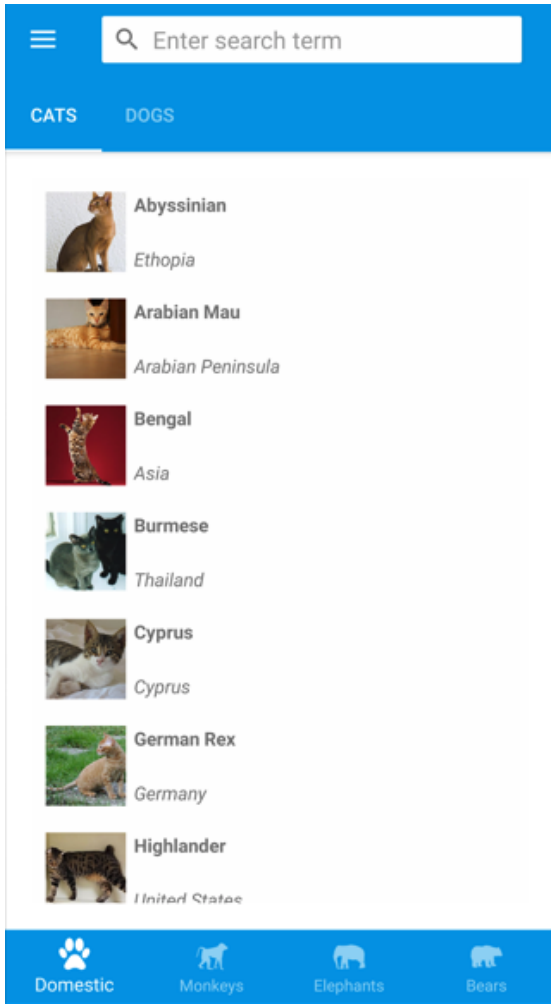
- TabBar et Tab
- Grouper avec Tab
- Affichage des onglets différent selon la plateforme

Exercice

Reprenez et adaptez à votre manière l'exemple de la documentation ci-contre en utilisant :

- **ListView** et **DataTemplate** pour afficher des animaux ou autres entités avec une mise en forme **utilisant un layout**
- le principe de **FlyOut**
- le principe de **Tabs**





Navigation

- Enregistrer des routes
- Goto et Routes

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Injection de dépendances et requêtes Http

Injection de dépendances

Pour réaliser l'**injection de dépendances**, il faut passer par le fichier **MauiProgram.cs** pour l'**enregistrement**.

/!\ Cependant pour qu'elle puisse **fonctionner** au niveau de nos **pages**, il faudra aussi **enregistrer ces pages** elles-mêmes.
Pour une utilisation plus avancée avec le pattern MVVM, il conviendra d'enregistrer aussi les **ViewModels**.

Exercice

En utilisant des services de types **FakeDb**, implémenter le **CRUD complet** pour l'**une des entités** affichée dans l'une des pages que vous avez fait au TP précédent.

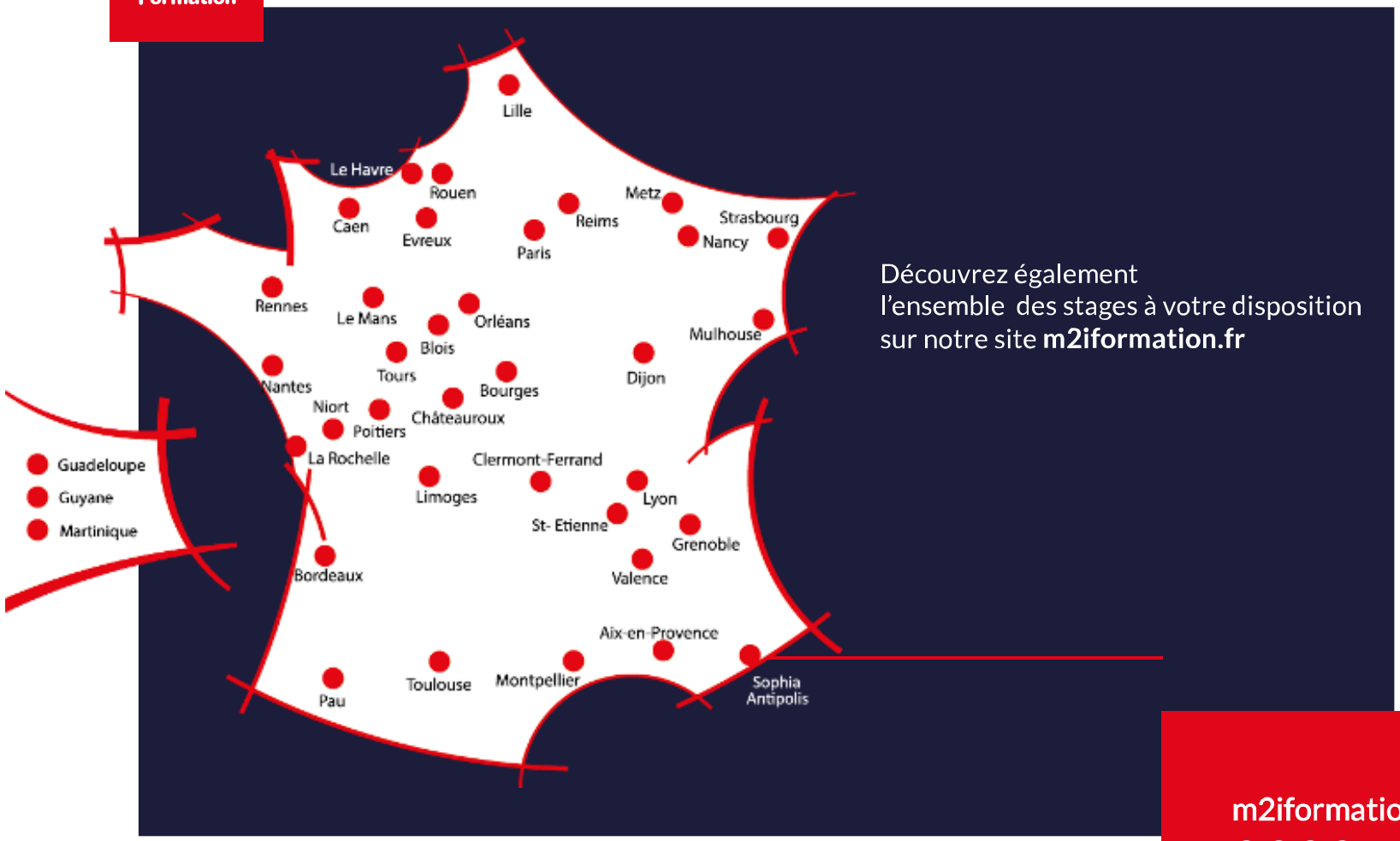
On aura donc la **page liste**, la **page détails** et la **page formulaire d'ajout/de modification**.

Utilisation de HttpClient pour les appels API

- Enregistrement de service
- Utilisation des extensions Json

Ressource complémentaires

- Tutoriels MAUI Divers
- Documentation MVVM
- Tutoriel MVVM
- Cycle de vie MAUI



Découvrez également
l'ensemble des stages à votre disposition
sur notre site **m2iformation.fr**

m2iformation.fr