Chapitre VI

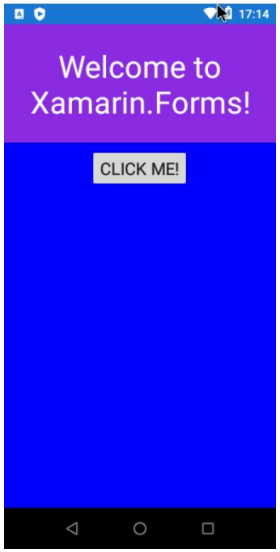
Programmation Xamarin.Forms

Plan

1. Organisation des GUI en C#
2. Programmation d'événements en C#
3. Le « binding » : liaison de données
4. Les « pop-ups » (fenêtres contextuelles)
5. Les pages modales

1. Organisation des GUI en C# (1/6)

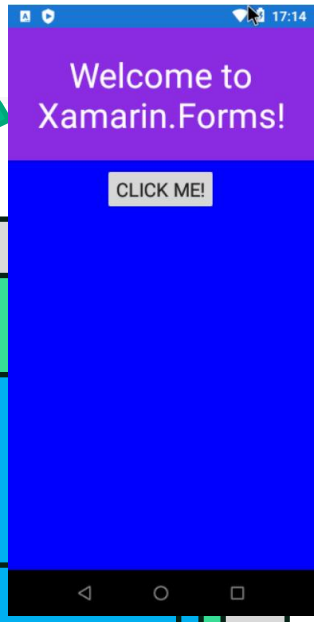
- Les composants (Pages, Layouts et Vues) sont représentés dans l'application
  - sous formes d'objets C# **imbriqués**
  - Les **objets contenus** se trouvent dans l'attribut Content ou comme Children (enfants) des **objets contenants**
  - Les **attributs XAML** d'un objet sont aussi ses **attributs en C#**
- **Exemple**



1. Organisation des GUI en C# (2/6)

- **Exemple**

Structure des objets C#



(MainPage : classe dérivée de ContentPage)

Content =

page1 (StackLayout)

BackgroundColor = Blue

Children =

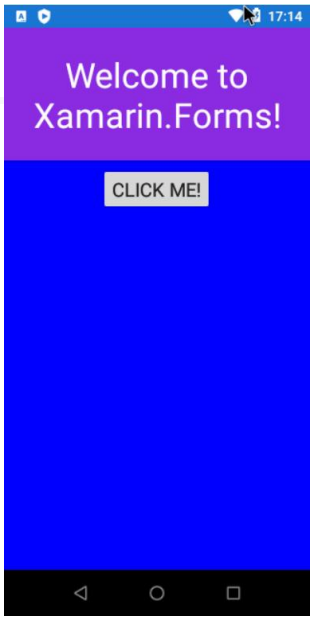
frame1 (Frame)
BackgroundColor = BlueViolet
Padding = 24
CornerRadius = 0
Content =

label1 (Label)
Text = "Welcome to Xamarin.Forms!"
HorizontalTextAlignment = Center
TextColor = White
FontSize = 36

button1 (Button)
Text = "Click me!"
FontSize = 20
HorizontalOptions = Center

1. Organisation des GUI en C# (3/6)

- **XAML de l'interface**




```
<?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="ExempleCours.MainPage">

    <StackLayout x:Name="page1" BackgroundColor="Blue">

        <Frame x:Name="frame1" BackgroundColor="BlueViolet" Padding="24" CornerRadius="0">

            <Label x:Name="label1" Text="Welcome to Xamarin.Forms!"
                   HorizontalTextAlignment="Center" TextColor="White" FontSize="36" />
        </Frame>

        <Button x:Name="button1" Text="Click me!" FontSize="20" HorizontalOptions="Center"/>

    </StackLayout>

</ContentPage>
```

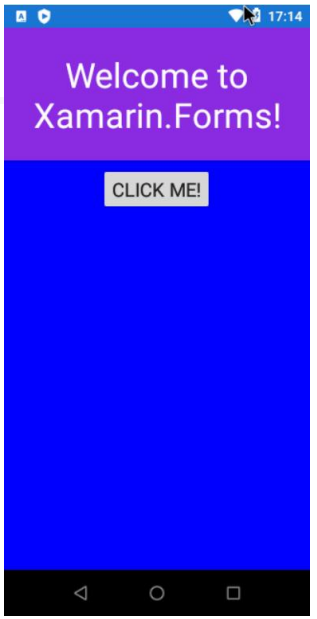
1. Organisation des GUI en C# (4/6)

- **Version 1 : Code C# (équivalent à XAML) de l'interface (dans la classe MainPage.xaml.cs)**

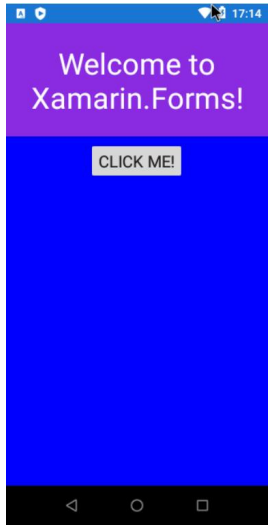
```
public partial class MainPage : ContentPage
    {
        StackLayout page1 = new StackLayout {
            BackgroundColor = Color.Blue
        };
        Frame frame1 = new Frame {
            BackgroundColor = Color.BlueViolet,
            Padding = new Thickness(24),
            CornerRadius = 0
        };
        Label label1 = new Label
        {
            Text = "Welcome to Xamarin.Forms!",
            HorizontalTextAlignment = TextAlignment.Center,
            TextColor = Color.White,
            FontSize = 36
        };
        Button button1 = new Button {
            Text = "Click me!",
            FontSize = 20,
            HorizontalOptions = LayoutOptions.Center
        };
```

Suite (imbrication des objets dans le contructeur MainPage()

```
public MainPage()
        {
            frame1.Content = label1;
            page1.Children.Add(frame1);
            page1.Children.Add(button1);
            Content = page1;
            InitializeComponent();

        }
    }
}
```

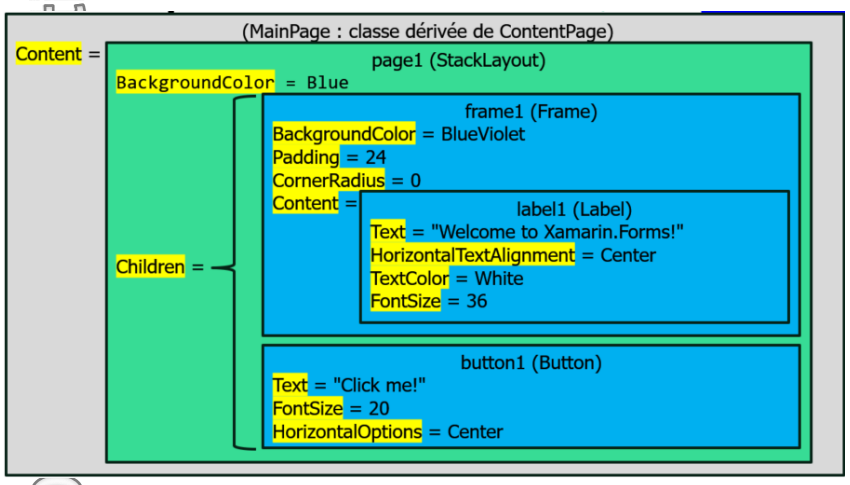


1. Organisation des GUI en C# (5/6)

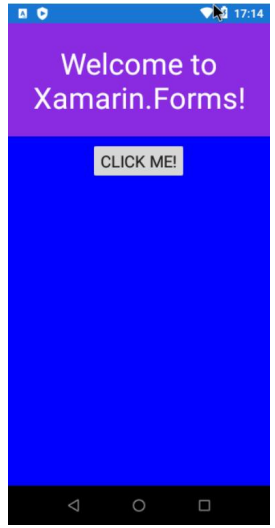


- **Version 2 : Code C# (équivalent à XAML) de l'interface (dans la classe MainPage.xaml.cs)**

```
public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            Content = new StackLayout
            {
                BackgroundColor = Color.Blue,
                Children =
                {
                    new Frame
                    {
                        BackgroundColor = Color.BlueViolet,
                        Padding = new Thickness(24),
                        CornerRadius = 0,
                        Content = new Label
                        {
                            Text = "Welcome to Xamarin.Forms!",
                            HorizontalTextAlignment = TextAlignment.Center,
                            TextColor = Color.White,
                            FontSize = 36,
                        },
                    },
                    new Button
                    {
                        Text = "Click me!",
                        FontSize = 20,
                        HorizontalOptions = LayoutOptions.Center,
                    },
                },
            };
            InitializeComponent();
            ...
```



1. Organisation des GUI en C# (6/6)



- **Version 3 : Code C# (équivalent à XAML) de l'interface (dans la classe MainPage.xaml.cs)**

=> On met le code de l'interface dans une méthode Build()

=> Build() est appelée à partir du constructeur MainPage()

```
public partial class MainPage : ContentPage
    {

        void Build() => Content = new StackLayout
            {
                BackgroundColor = Color.Blue,
                Children =
                {
                    new Frame
                    {
                        BackgroundColor = Color.BlueViolet,
                        Padding = new Thickness(24),
                        CornerRadius = 0,
                        Content = new Label
                        {
                            Text = "Welcome to Xamarin.Forms!",
                            HorizontalTextAlignment = TextAlignment.Center,
                            TextColor = Color.White,
                            FontSize = 36,
                        },
                    },
                    new Button
                    {
                        Text = "Click me!",
                        FontSize = 20,
                        HorizontalOptions = LayoutOptions.Center,
                    },
                },
            };
        public MainPage()
        {
            Build();
            InitializeComponent();
```

…

2. Programmation d'événements en C# (1/7)

- Les composants de la GUI lancent des évènements dans des conditions particulières
- Ces évènements peuvent être associée à des méthodes C#
  - Appelée Call-backs
  - Utilisées pour traiter ces évènements
- C'est de la programmation événementielle

2. Programmation d'événements en C# (2/7)

- Exemple d'événements
  - Entry : TextChanged, Completed
  - Button : Clicked, Pressed, Released
  - SwitchCell : Changed, Tapped
  - Slider : ValueChanged, DragStarted, DragCompleted

- Autre événements communs :
  - Focused
  - Unfocused
  - SizeChanged
  - PropertyChanged
  - BindingContextChanged…

2. Programmation d'événements en C# (3/7)

- Exemple 1 : Affichage d'une boite de dialogue

XAML:

```
<StackLayout>
        <Button Text="Message secret" Clicked="Button_Clicked"/>
</StackLayout>
```

C#:

```
private async void Button_Clicked(object sender, EventArgs e)
{
  await DisplayAlert("Alert", "C'est une surprise!", "OK");
}
```

2. Programmation d'événements en C# (4/7)

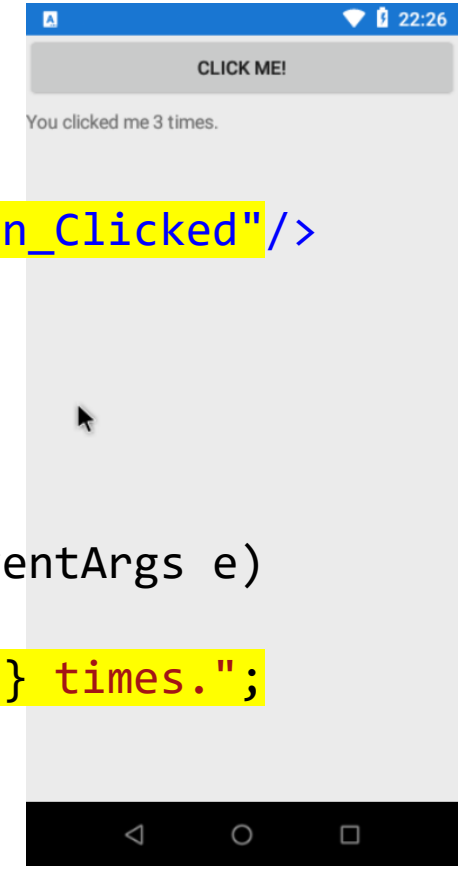
- Exemple 2 : Compteur de nombres de clicks d'un bouton

XAML:

```
 <StackLayout>
        <Button Text="Click me!" Clicked="Button_Clicked"/>
        <Label x:Name="label1" Text="Ready"/>
</StackLayout>
```

C#:

```
int nbreClicks = 1;
  private void Button_Clicked(object sender, EventArgs e)
  {
      label1.Text = $"You clicked me {nbreClicks} times.";
      nbreClicks++;
  }
```



2. Programmation d'événements en C# (5/7)

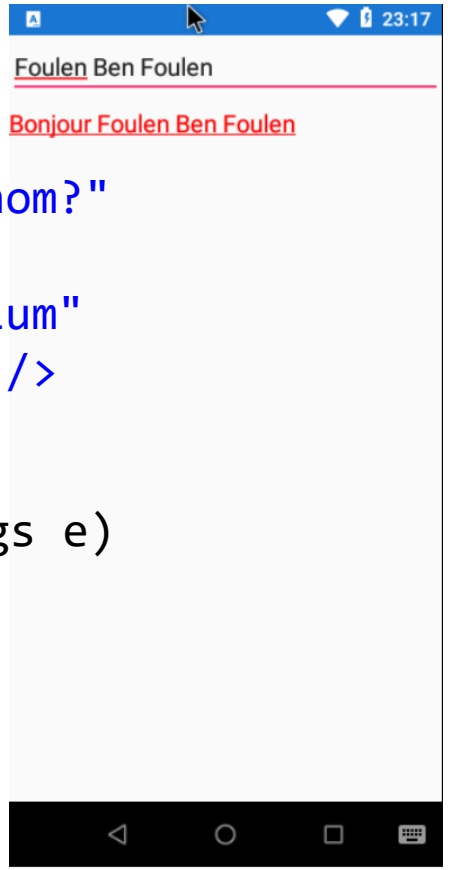
- Exemple 3 : Salutation cordiale avec zone de texte

XAML:

```
<Entry x:Name="entry1" Placeholder="Donner votre nom?"
       Text="" TextChanged="Entry_TextChanged"/>
<Label x:Name="label1" Text="Ready" FontSize="Medium"
       TextDecorations="Underline" TextColor="Red"/>
```

C#:

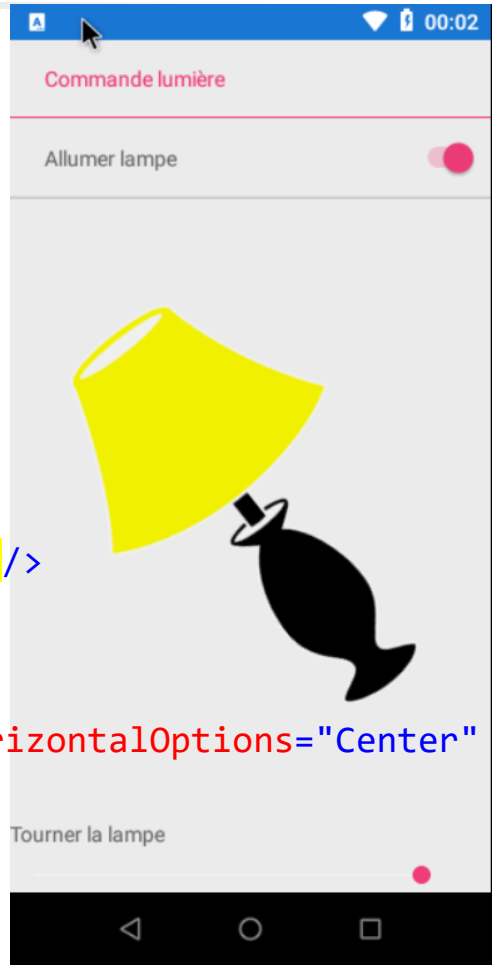
```
private void Entry_TextChanged(object sender,
                               TextChangedEventArgs e)
{
  label1.Text = $"Bonjour {entry1.Text}";
}
```



2. Programmation d'événements en C# (6/7)

- Exemple 4 : Lampe rotative (1/2)

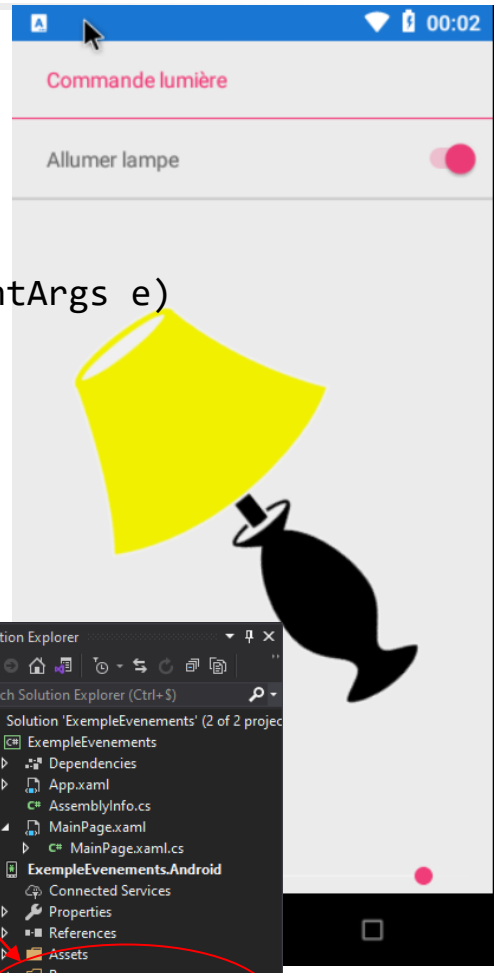
```
XAML:
<TableView>
  <TableView.Root>
     <TableSection Title="Commande lumière">
        <SwitchCell x:Name="switchcell1"
                            Text="Allumer lampe"
                            On="False"
                            OnChanged="SwitchCell_OnChanged"/>
     </TableSection>
  </TableView.Root>
</TableView>
<Image x:Name="image1" Source="lamp_off" Margin="10, 50" HorizontalOptions="Center"
         HeightRequest="300" />
<Label Text="Tourner la lampe" />
<Slider x:Name="slider1" Minimum="0" Maximum="359" Value="0"
          ValueChanged="slider1_ValueChanged"/>
```



2. Programmation d'événements en C# (7/7)

- Exemple 4 : Lampe rotative (2/2)

```
C#:
private void SwitchCell_OnChanged(object sender, ToggledEventArgs e)
{
      if (switchcell1.On == true)
            image1.Source = "lamp_on";
      else
            image1.Source = "lamp_off";
      }

private void slider1_ValueChanged(object sender,
                                  ValueChangedEventArgs e)
{
            image1.RotateTo(rotation: slider1.Value);
}
```



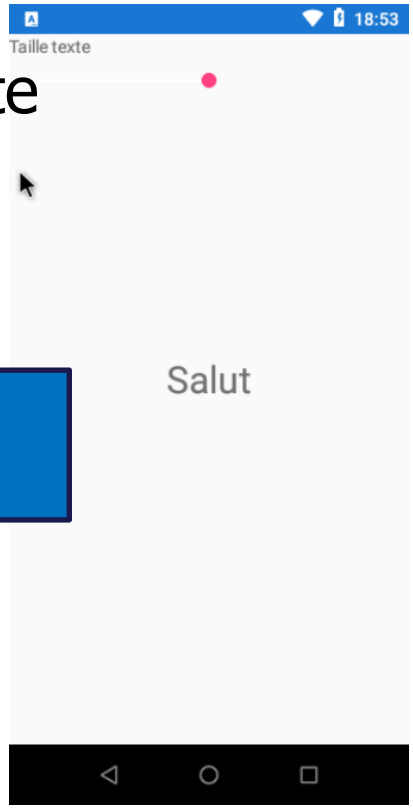
3. Le « binding » : liaison de données (1/8)

- Il est possible de lier
  - des valeurs de propriétés d'objets XAML à d'autres valeurs de propriétés d'objets XAML (View-to-View)
  - des valeurs de variables C# à des valeurs de propriétés d'objets XAML
- Lorsque une valeurs change, sa/ses valeur(s) liée(s) change(nt) aussi!
  - Cette association peut avoir plusieurs directions possibles
    - par défaut (Default)
    - Valeur source => Valeur cible (OneWay)
    - Valeur source <= Valeur cible (OneWayToSource)
    - Valeur source <=> Valeur cible (TwoWay)
    - Valeur source => Valeur cible (1 fois seulement au changement de la valeur source) (OneTime)
- => Utilisation de l'attribut BindingContext

3. Le « binding » : liaison de données (2/8)

- Binding **propriété <=> propriété (1/2)**
  - **Liaison entre propriétés (attributs) d'objets différents**
  - =>BindingContext utilisé depuis XAML
- Exemple : Contrôle de la taille de texte
  - Deux propriétés liées
    - Source
      - Value entre 12 et 48 de slider1 (Slider)
    - Cible
      - FontSize de (Label)

  - Remarque : l'objet de la source doit toujours posséder un nom (x:Name=«…»)



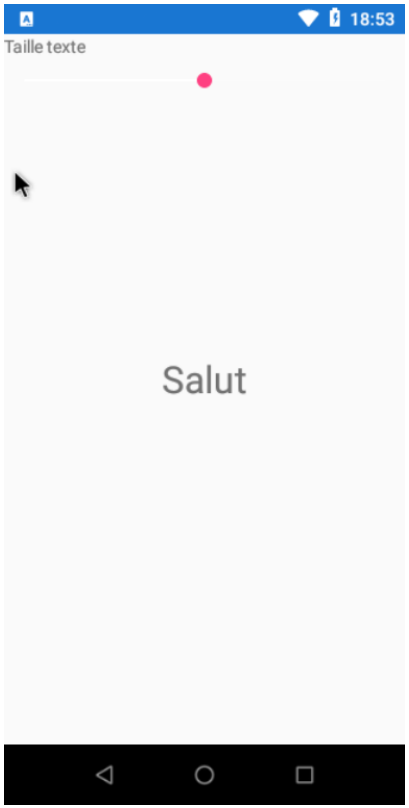
Dév. d'applications mobiles

3. Le « binding » : liaison de données (3/8)

- Binding **propriété <=> propriété (2/2)**
- Exemple (suite)

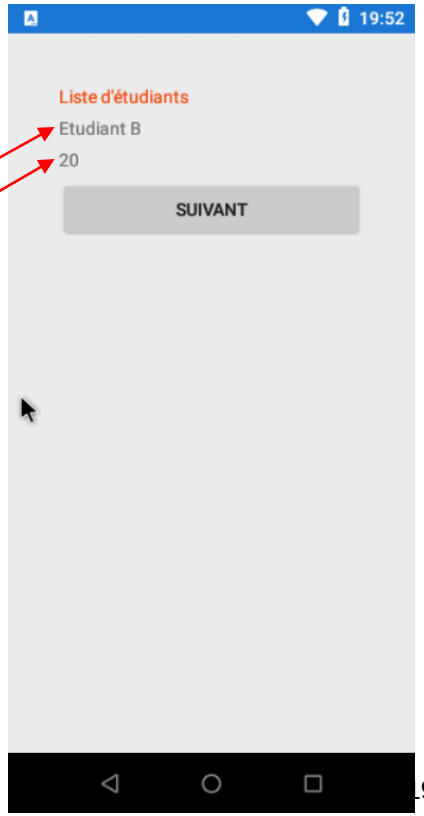
XAML:

```
…
<Slider x:Name="slider1" Maximum="48" Minimum="12" Value="30"/>
<Label Text="Salut" HorizontalOptions="Center" Margin="10,200"
       BindingContext="{x:Reference slider1}"
       FontSize="{Binding Value, Mode=OneWay}"/>
```



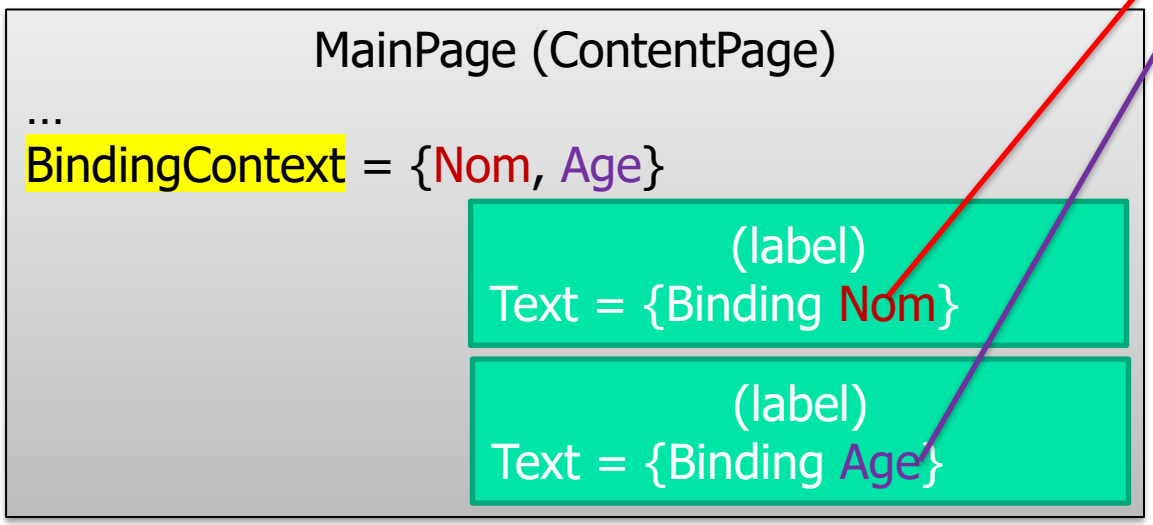
3. Le « binding » : liaison de données (4/8)

- Binding **variable <=> propriété (1/5)**
  - **Liaison entre variable C# et propriétés (attributs) d'objet**
  - =>BindingContext utilisé depuis C#
- Exemple : Navigation dans une liste d'étudiants
  - Liaison
    - Variables
      - Nom
      - Age
    - Cibles
      - Text de (Label)
      - Text de (Label)



3. Le « binding » : liaison de données (5/8)

- Binding **variable <=> propriété (2/5)**
  - **Liaison entre variable C# et propriétés (attributs) d'objet**
  - =>BindingContext utilisé depuis C#
- Exemple : Navigation dans une liste d'étudiants



3. Le « binding » : liaison de données (6/8)

- Binding **variable <=> propriété (3/5)**
  - **Liaison entre variable C# et propriétés (attributs) d'objet**
  - =>BindingContext utilisé depuis C#
- Exemple : Navigation dans une liste d'étudiants

XAML :

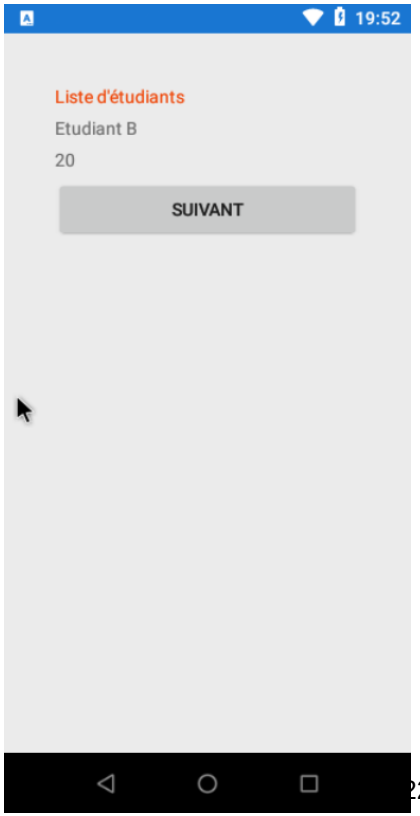
```
…
<Label Text="Liste d'étudiants" TextColor="OrangeRed" />
<!-- Pour le nom de l'étudiant -->
<Label Text="{Binding Nom}" />
<!-- Pour l'age de l'étudiant -->
<Label Text="{Binding Age}" />
<Button Text="Suivant" Clicked="Button_Clicked_suiv"/>
```

3. Le « binding » : liaison de données (7/8)

- Binding **variable <=> propriété (4/5)**

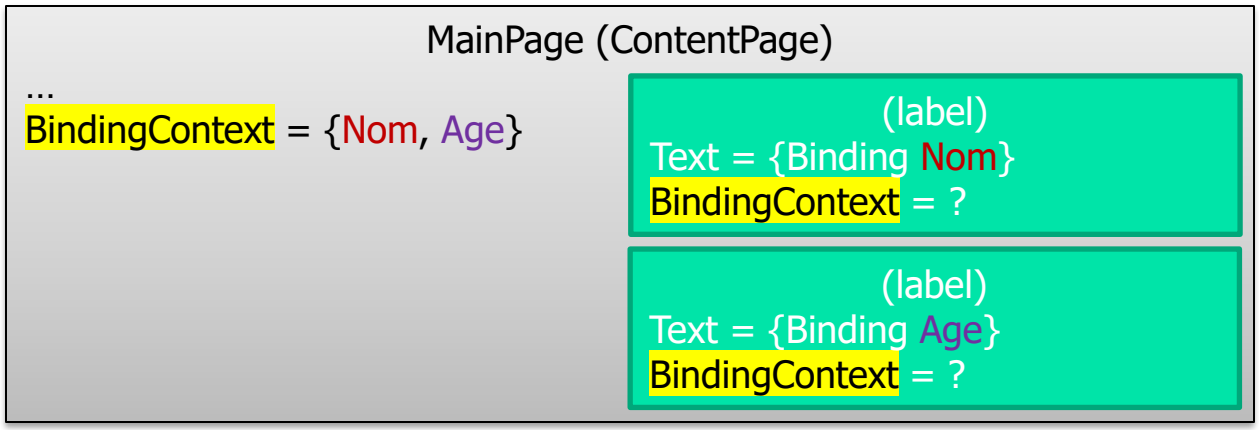
C#:

```
public partial class MainPage : ContentPage    {
        static List<object> lstEtud = new List<object>
        {
            new {Nom = "Etudiant A", Age = 19 },
            new {Nom = "Etudiant B", Age = 20 },
            new {Nom = "Etudiant C", Age = 21 },
        };

        int compteur = 0;

        object etudiantAffiche = lstEtud.ElementAt(0);

        public MainPage()
        {
            InitializeComponent();
            BindingContext = etudiantAffiche;
        }
        private void Button_Clicked_suiv(object sender, EventArgs e)
        {
            compteur = ++compteur % lstEtud.Count;
            etudiantAffiche = lstEtud.ElementAt(compteur);
            BindingContext = etudiantAffiche;
        }
    }
...
```



3. Le « binding » : liaison de données (8/8)

- Binding **variable <=> propriété (5/5)**
  - Remarque
    - Dans l'exemple précédent, les composants enfants (Label) ont accédé au BindingContext de leur parent (MainPage)
    - Mais chaque composant possède aussi sont propre BindingContext (qui peut être utilisé)



- Autre exemple
  - https://docs.microsoft.com/fr-fr/xamarin/xamarin-forms/xaml/xaml-basics/data-binding-basics

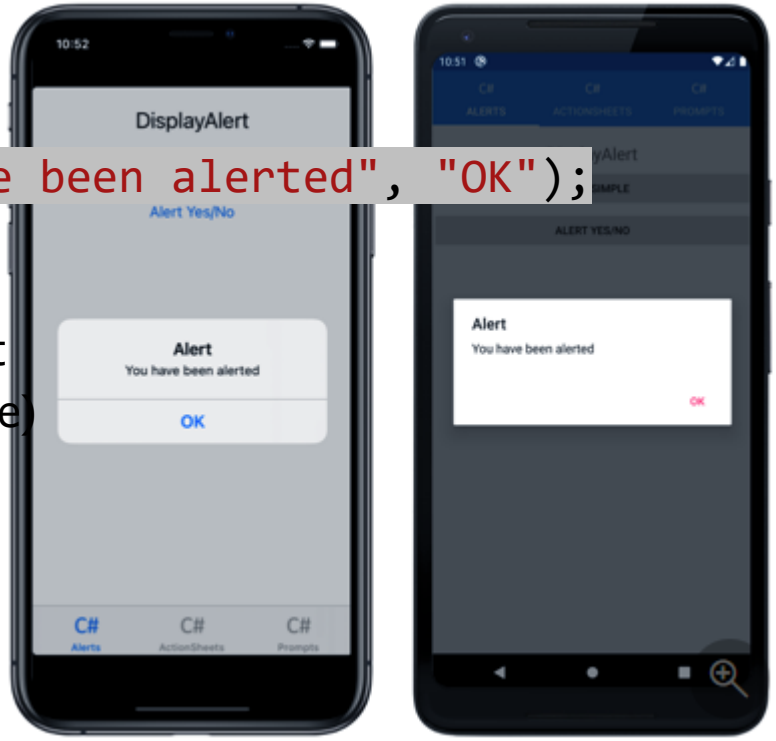
4. Les « pop-ups » (fenêtres contextuelles) (1/7)

- Source officielle
  - https://docs.microsoft.com/fr-fr/xamarin/xamarin-forms/user-interface/pop-ups
- Afficher une alerte

C#:

```
await DisplayAlert("Alert", "You have been alerted", "OK");
```

- Remarque
  - Toute méthode contenant `await` doit être marquée comme `async` (asynchrone)

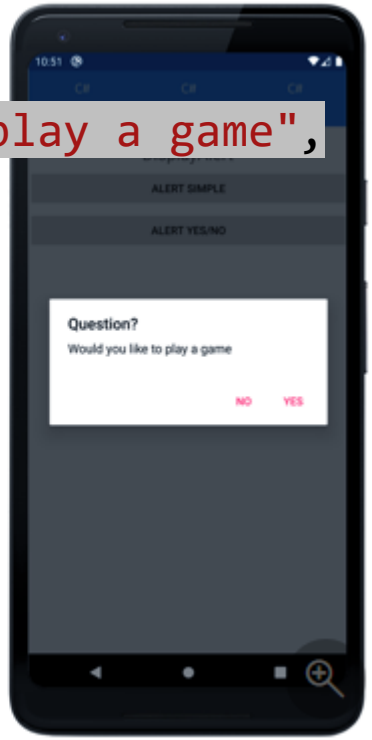


4. Les « pop-ups » (fenêtres contextuelles) (2/7)

- Afficher une alerte (Yes/No)

C#:

```
async void OnAlertYesNoClicked(object sender, EventArgs e)
{
bool answer = await DisplayAlert("Question?",
                                 "Would you like to play a game",
                                 "Yes", "No");
Debug.WriteLine("Answer: " + answer);
}
```

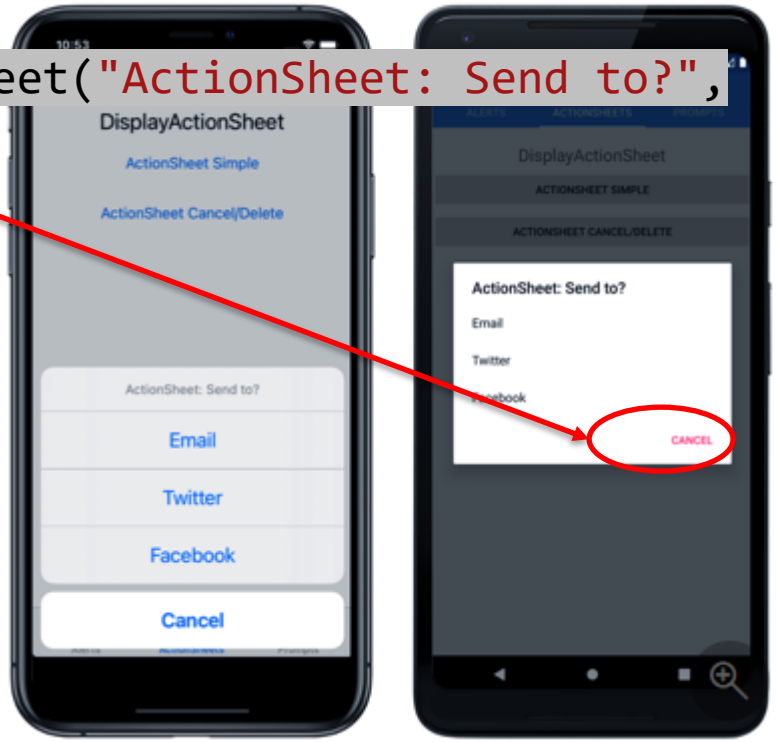


4. Les « pop-ups » (fenêtres contextuelles) (3/7)

- Guider les utilisateurs à travers les tâches (1/2)

C#:

```
async void OnActionSheetSimpleClicked(object sender, EventArgs e)
{
string action = await DisplayActionSheet("ActionSheet: Send to?",
        "Cancel",
        null, //bouton enlevé!
        "Email",
        "Twitter",
        "Facebook");
Debug.WriteLine("Action: " + action);
}
```

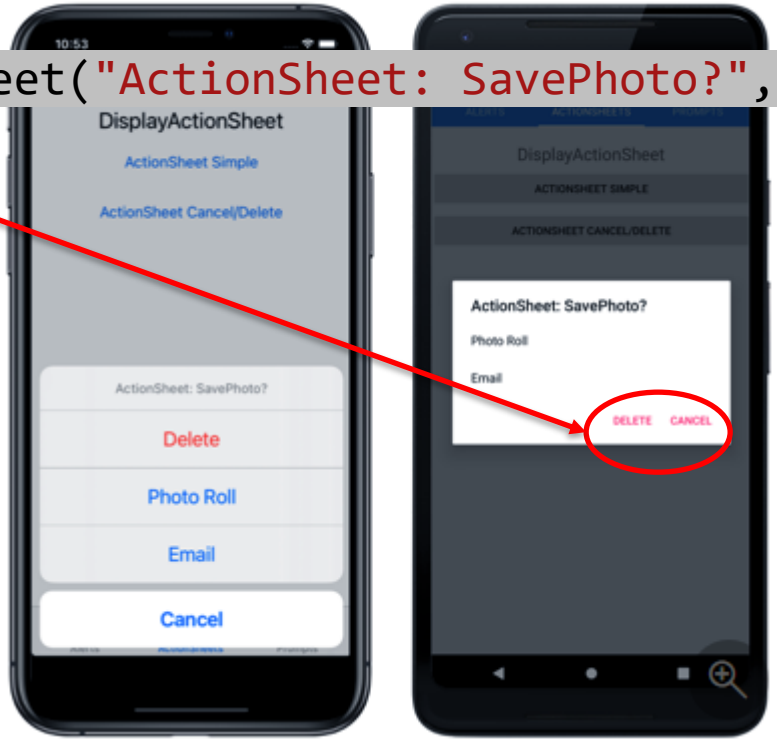


4. Les « pop-ups » (fenêtres contextuelles) (4/7)

- Guider les utilisateurs à travers les tâches (2/2)

C#:

```
async void OnActionSheetCancelDeleteClicked(object sender, EventArgs
{
string action = await DisplayActionSheet("ActionSheet: SavePhoto?",
        "Cancel",
        "Delete",//deux boutons
        "Photo Roll",
        "Email");
Debug.WriteLine("Action: " + action);
}
```

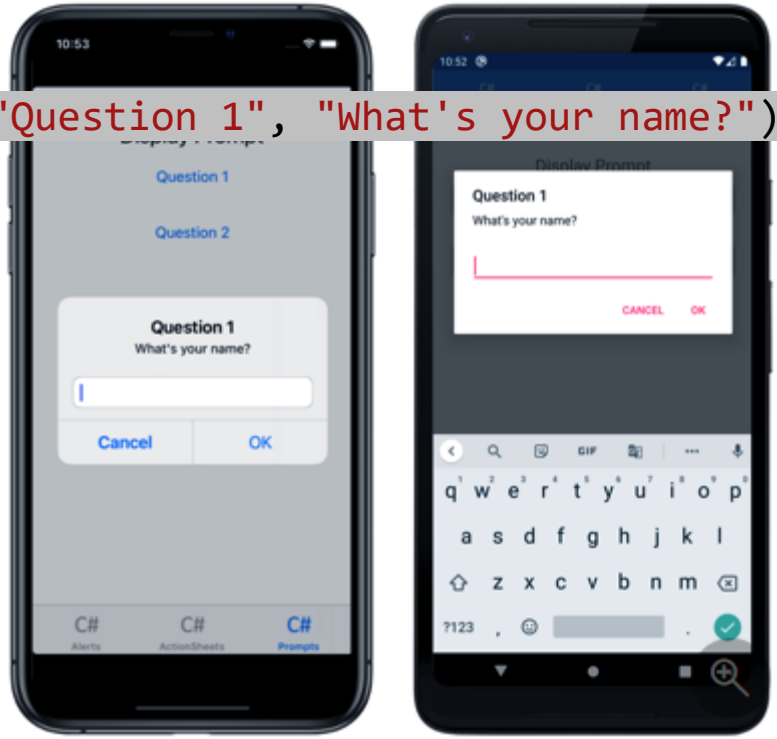


4. Les « pop-ups » (fenêtres contextuelles) (5/7)

- Afficher une invite (1/3)
  - DisplayPromptAsync en passant un titre et un message (string arguments)

C#:

```
string result = await DisplayPromptAsync("Question 1", "What's your name?");
```



  - Si OK cliquée
    - chaine (string) retournée
  - Si Annuler cliquée
    - null retournée

4. Les « pop-ups » (fenêtres contextuelles) (6/7)

- Afficher une invite (2/3)

Arguments de DisplayPromptAsync

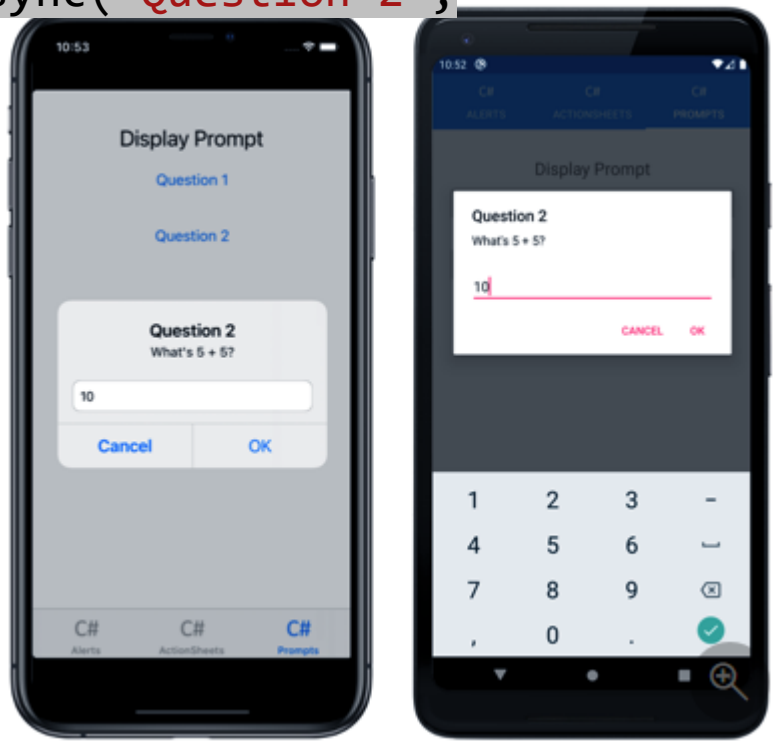
- titre à afficher : title de type string
- message à afficher : message de type string
- texte du bouton accepter (facultatif, défaut = OK) : accept de type string
- texte du bouton annuler (facultatif, défaut = Cancel) : cancel de type string
- texte d'espace réservé à afficher (facultatif, défaut = null) : placeholder de type string
- longueur maximale de la réponse (facultatif, défaut = -1) : maxlength de type int
- type de clavier à utiliser (facultatif, défaut = Keyboard.Default) : keyboard de type Keyboard
- réponse prédéfinie qui s'affiche et qui peut être modifiée (facultatif, défaut = « ») : initialValue de type string

4. Les « pop-ups » (fenêtres contextuelles) (7/7)

- Afficher une invite (3/3)

C#:

```
string result = await DisplayPromptAsync("Question 2",
        "What's 5 + 5?",
        initialValue: "10",
        maxLength: 2,
        keyboard: Keyboard.Numeric);
```



5. Les pages modales

- Cette section est à consulter à partir de la documentation Microsoft officielle sur ce lien
  - https://docs.microsoft.com/fr-fr/xamarin/xamarin-forms/app-fundamentals/navigation/modal