.NET Multi-platform App UI (.NET MAUI) (1)

Programowanie Wizualne

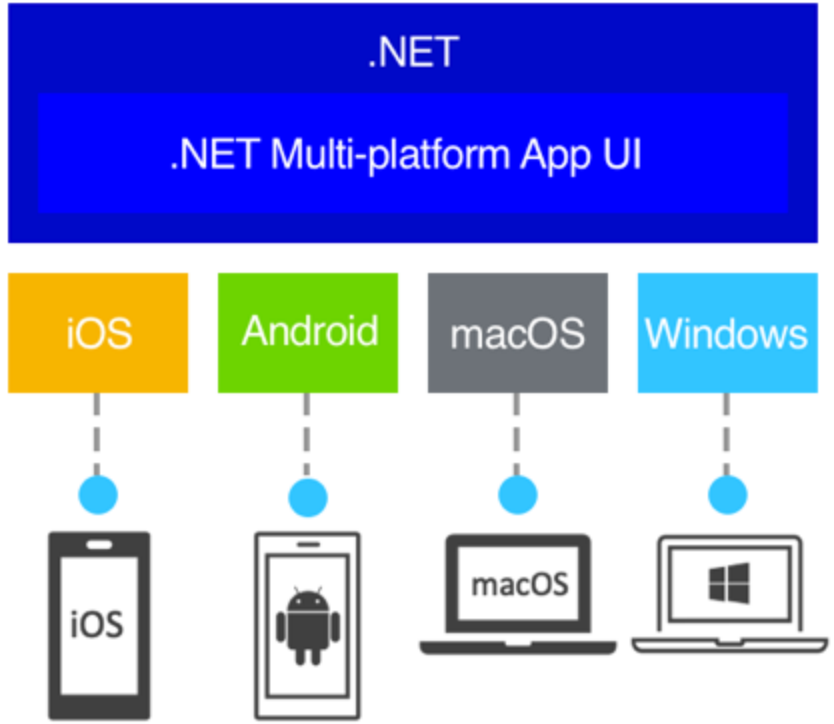
Paweł Wojciechowski

Instytut Informatyki, Politechniki Poznańskiej

2023

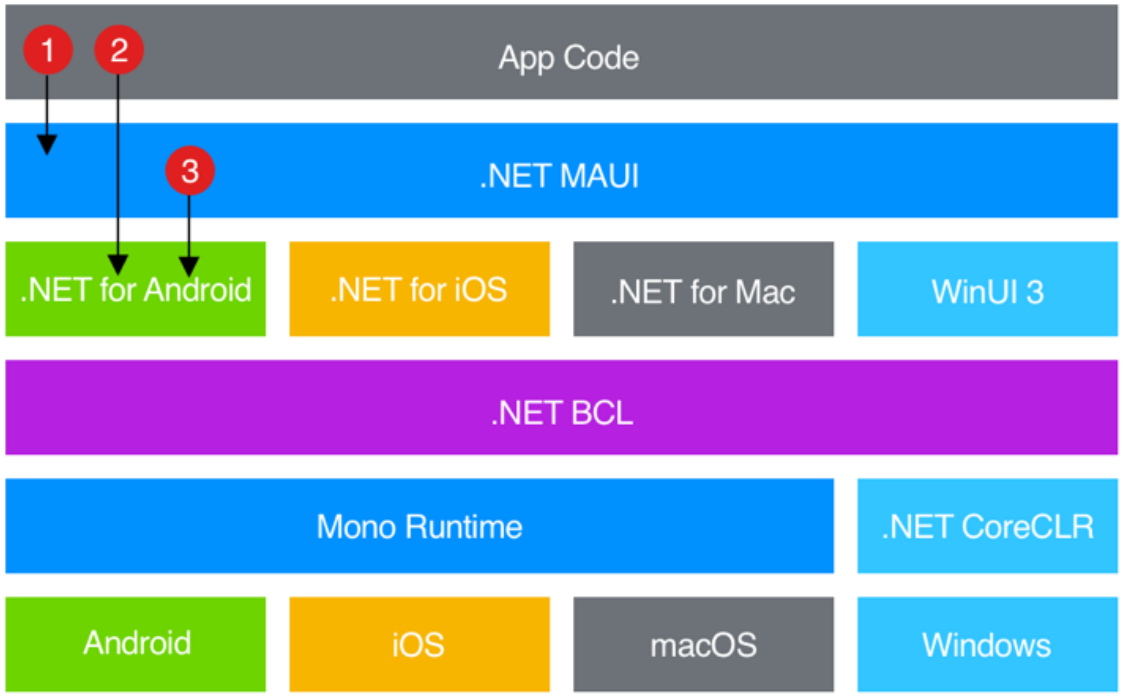
.NET MAUI główne cechy

- .NET MAUI jest dystrybuowany na zasadzie open source i naturalnie wyewoluował z Xamarin.Forms
- deklarowane interfejsy użytkownika
  interfejs użytkownika deklarowany jest w specjalnym języku XAML



Działanie .NET MAUI

- 1. pisany kod korzysta z API, który jest współdzielony przez platformy
- 2. kod aplikacji może korzystać bezpośrednio z natywnych interfejsów API platformy
- 3. .NET MAUI na bazie kodu napisanego w p. 1. generuje kod właściwy dla platformy
- Platformy:
  - Android – kompilacja do IL, a następnie do JIT
  - iOS – w pełni (ahead-of-time – AOT) kompilowane są do natywnego kodu assemblera dla ARM
  - macOS – wykorzystanie Mac Catalyst do budowania aplikacji desktopowych z biblioteką UIKit i AppKit
  - Windows wykorzystanie biblioteki WinUI 3 – biblioteki do tworzenia aplikacji dla Windowsa



https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-8.0

.NET MAUI – co programista z tego ma?

- Zapewnia zbiór elementów sterujących (kontrolki)
- Zbiór pojemników do tworzenia stron zawartości
- Obsługa aplikacji wielostronicowych wraz z nawigacją
- Wsparcie wiązania danych
- Wieloplatformowe API wspierające natywne cechy urządzeń takich jak np. GPS, akcelerometr, stan baterii i sieci.
- Wieloplatformowe wsparcie grafiki obejmujące rysowanie i kolorowanie kształtów i obrazków, operacji na nich i wsparcie transformacji
- .NET hot reload – umożliwia wprowadzanie zmian do kodu XAML w czasie działania aplikacji bez konieczności ponownej kompilacji kodu.

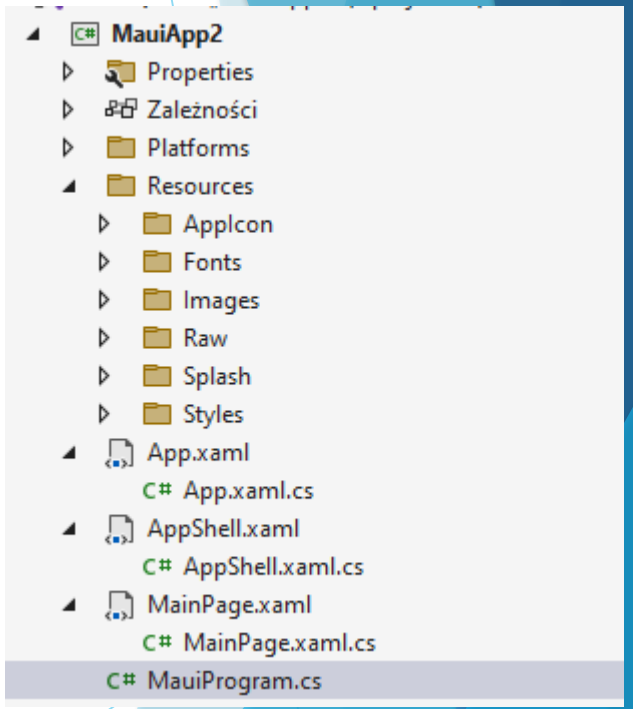
XAML

- XAML e*Xtensible Application Markup Language*
  - WPF XAML
  - XPS XAML (*XML Paper Specification*)
  - WF XAML (*Windows Workflow*)
- *case sensitive* – na poziomie nazw znaczników
- główny cel – oddzielenie logiki aplikacji od jej wyglądu
- XAML nie jest niezbędny do działania WPFa
- Podstawowe właściwości:
  - Każdy element zdefiniowany w XAMLu odpowiada klasie w .NET.
  - Jak w każdym dokumencie XML jeden element można osadzać w innym
  - Właściwości klas mogą zostać ustawione poprzez atrybuty.
  - XAML nie zawiera kodu
  - Najważniejszymi unikalnymi cechami są: właściwości, właściwości wiązane (attached properties) i tzw. markup extensions

Pierwsza aplikacja

Składowe projektu:

- MauiProgram.cs – plik z kodem, który konfiguruje i ładuję aplikację. Kod startowy szablonu wskazuje na klasę App.
- Pliki App.xaml i App.xaml.cs
  Zawiera zasoby xaml dla całej aplikacji (style, szablony, kolory). W kodzie tworzy instancję aplikacji powłoki AppShell.
- Pliki AppShell.xaml i AppShell.xaml.cs
  Służy do definiowania hierarchii wizualnej aplikacji
- Pliki MainPage.xaml i MainPage.xaml.cs
  strona startowa aplikacji.



Pierwsza aplikacja (2)

```xml
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="MAUIDemo.MainPage">

    <ScrollView>
        <VerticalStackLayout
            Padding="30,0"
            Spacing="25">
            <Image
                Source="dotnet_bot.png"
                HeightRequest="185"
                Aspect="AspectFit"
                SemanticProperties.Description="dot net bot in a race car number eight" />

            <Label
                Text="Hello, World!"
                Style="{StaticResource Headline}"
                SemanticProperties.HeadingLevel="Level1" />

            <Label
                Text="Welcome to &#10;.NET Multi-platform App UI"
                Style="{StaticResource SubHeadline}"
                SemanticProperties.HeadingLevel="Level2"
                SemanticProperties.Description="Welcome to dot net Multi platform App U I"
/>

            <Button
                x:Name="CounterBtn"
                Text="Click me"
                SemanticProperties.Hint="Counts the number of times you click"
                Clicked="OnCounterClicked"
                HorizontalOptions="Fill" />
        </VerticalStackLayout>
    </ScrollView>

</ContentPage>
```

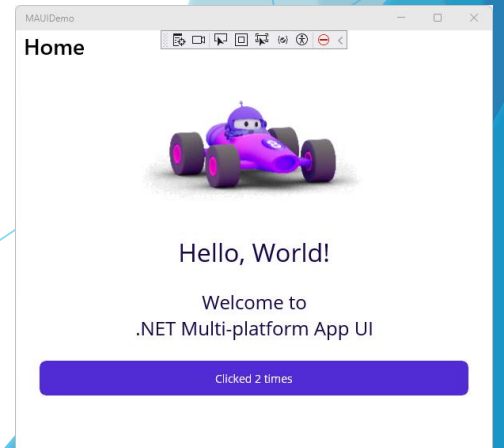
```csharp
namespace MAUIDemo
{
    public partial class MainPage : ContentPage
    {
        int count = 0;

        public MainPage()
        {
            InitializeComponent();
        }

        private void OnCounterClicked(object sender, EventArgs
e)
        {
            count++;

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

            SemanticScreenReader.Announce(CounterBtn.Text);
        }
    }
}
```



Przestrzenie nazw

„Namiary” na lokację klas .NET – atrybut **xmlns**

- xmlns="http://schemas.microsoft.com/dotnet/2021/maui" domyślny pakiet w aplikacji zawierający wszystkie klasy .NET MAUI.
- xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  prefiks dla xmlns określa inną niż domyślną przestrzeń nazw. W tym przypadku, przestrzeń x określa elementy i atrybuty, tóre są wewnętrzne dla XAML
- Odwołanie do własnych klas

  xmlns:*prefix*="clr-namespace:*Namespace*;assembly=*AssemblyName*"

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="MauiApp2.MainPage">
             xmlns:custom="clr-namespace:Sample;assembly=SampleLibrary">
      ...
         <custom:ExampleClass/>
      ...
</ContentPage>
```

Nazywanie elementów interfejsu

- Nadawanie nazw elementom nie jest obowiązkowe.

```
<Grid>
        ...
</Grid>
```

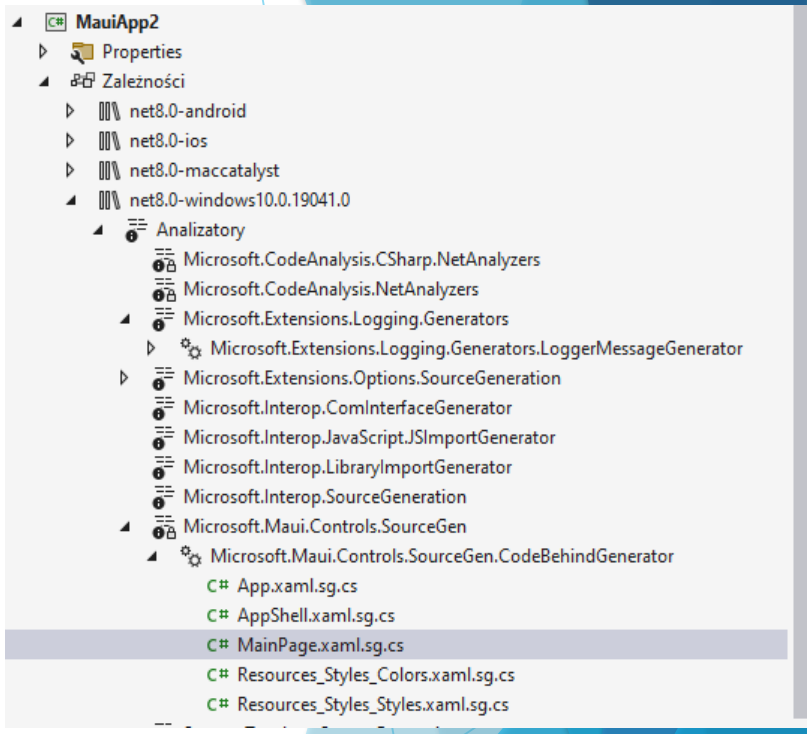
- Wymagane jest jednak jeśli wymagane są zmiany parametrów elementu kontrolnego w kodzie programu

```
<Grid x:Name="MojGrid">

</Grid>
```

- Dla tak zdefiniowanej nazwy zostanie wygenerowana automatycznie następująca część kodu klasy `MainPage`

```
[global::System.CodeDom.Compiler.GeneratedCode("Microsoft.Maui.Controls.SourceGen", "1.0.0.0")]
private global::Microsoft.Maui.Controls.Grid MojGrid;
```



Właściwości komponentów
proste typy i konwertery

```
<Image
    Source="dotnet_bot.png"
    HeightRequest="185"
    Aspect="Fill"
    Background="Aqua"
    SemanticProperties.Description="dot net bot in a race car number eight"
    HorizontalOptions="Center"
    />
```

Semantic properties are used to define information about which controls should receive accessibility focus and which text should be read aloud to the user.

https://learn.microsoft.com/en-us/dotnet/communitytoolkit/maui/markup/extensions/semantic-properties

Dla powyższego komponentu (`Image`) muszą być zdefiniowane następujące właściwości:
`HeightRequest, Aspect, Background, HorizontalOptions`

Typy proste – prosta konwersja.
Typy złożone np. właściwość `Background` wymaga specjalnego konwertera.

Zaleca się nadawania wartości parametrów w XAMLu nie w kodzie aplikacji, ze względu na to, iż XAML jest parsowany i sprawdzany w czasie kompilacji, dzięki czemu wcześniej można wykryć potencjalne błędy.

Właściwości komponentów
właściwości złożone

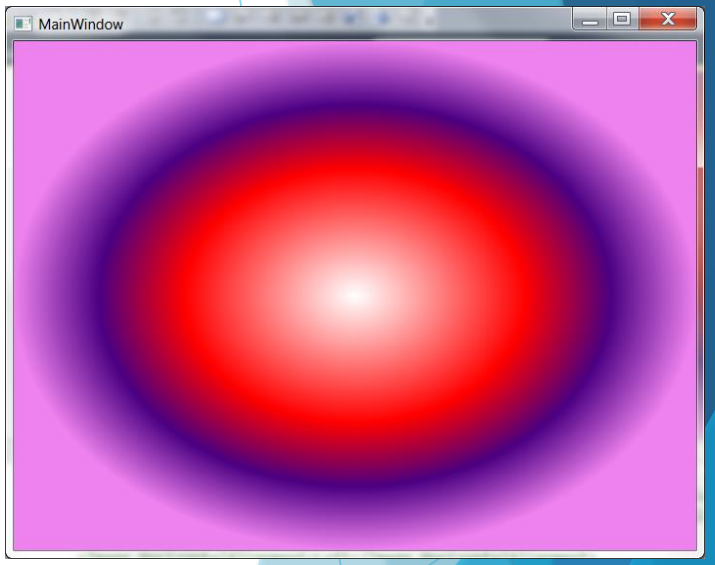
- niektóre właściwości mogą być obiektami – problem z konwersją
- w XAML-u można definiować elementy zagnieżdżone w formie `Rodzic.NazwaWłaściwości`

```
<Image
    Source="dotnet_bot.png"
    HeightRequest="185"
    Aspect="Fill"
    Background="Aqua"
    SemanticProperties.Description="dot net bot in a race car number eight"
    HorizontalOptions="Center"
    />


<Image
    Source="dotnet_bot.png"
    SemanticProperties.Description="dot net bot in a race car number eight"
    >
    <Image.HeightRequest>185</Image.HeightRequest>
    <Image.Aspect>Fill</Image.Aspect>
    <Image.Background>Aqua</Image.Background>
    <Image.HorizontalOptions>Center</Image.HorizontalOptions>
</Image>
```

Właściwości komponentów
właściwości złożone (2)

```
<Grid Name="Grid_1">
    <Grid.Background>
        <RadialGradientBrush>
            <GradientStop Offset="0.00" Color="White" />
            <GradientStop Offset="0.5" Color="Red"/>
            <GradientStop Offset="0.75" Color="Indigo"/>
            <GradientStop Offset="1" Color="Violet"/>
        </RadialGradientBrush>
    </Grid.Background>
</Grid>
```



```
RadialGradientBrush rgb = new RadialGradientBrush();

GradientStop gradientStop1 = new GradientStop();
gradientStop1.Color = Colors.White;
gradientStop1.Offset = 0;
rgb.GradientStops.Add(gradientStop1);

GradientStop gradientStop2 = new GradientStop();
gradientStop2.Color = Colors.Red;
gradientStop2.Offset = 0.5;
rgb.GradientStops.Add(gradientStop2);

GradientStop gradientStop3 = new GradientStop();
gradientStop3.Color = Colors.Indigo;
gradientStop3.Offset = 0.75;
rgb.GradientStops.Add(gradientStop3);

GradientStop gradientStop4 = new GradientStop();
gradientStop4.Color = Colors.Violet;
gradientStop4.Offset = 1;
rgb.GradientStops.Add(gradientStop4);

Grid_1.Background = rgb;
```

Właściwości komponentów
Dynamiczne wartości właściwości

- w poprzednich przykładach wartości właściwości były statyczne
- Specjalna składnia umożliwia nadawanie wartości dynamicznych przez wiązanie jej z wartościami właściwości klas

```
<Label Name="label1" Content="Hello World" Background="{x:Static SystemColors.HighlightColor}"/>
```

- wszystkie rozszerzone znaczniki (*markup extensions)* są zaimplementowane w klasach implementujących Microsoft.Maui.Controls.Xaml.IMarkupExtension

```
label1.Background = new SolidColorBrush(SystemColors.HighlightColor);
```

```
<Label Name="label1" Content="Hello World">
    <Label.Background>
        <SolidColorBrush>
            <SolidColorBrush.Color>
                <x:Static Member="SystemColors.HighlightColor"></x:Static>
            </SolidColorBrush.Color>
        </SolidColorBrush>
    </Label.Background>
</Label>
```

Właściwości komponentów
właściwości powiązane

- *Attached Properties*
- właściwości zdefiniowane w jednym z komponentów, ale zdefiniowane w innej klasie
- używanie poprzez: `TypDefniniujący.NazwaWłaściwości`

```
<Label x:Name="label1" Content="Hello World" Grid.Row="0" Grid.Column="0"/>
```

- są one przekształcane w wywołania metod `TypDefiniujący.SetNazwaWłaściwości`

```
Grid.SetColumn(label1, 1);
```

Zdarzenia

- w XAMLu można również definiować zdarzenia – NazwaZdarzenia="NazwaMetodyObsługiZdarzenia"

```
<Button
    x:Name="CounterBtn"
    Text="Click me"
    SemanticProperties.Hint="Counts the number of times you click"
    Clicked="OnCounterClicked"
    HorizontalOptions="Fill" />
```

```
private void OnCounterClicked(object sender, EventArgs e)
{
    DisplayAlert("Message", "ButtonClicked", "Ok");
}
```

Ładowanie i kompilacja XAML

- Kod .NET MAUI w XAML jest domyślnie kompilowany do języka IL przez kompilator XAMLC. Zapewnia to:
  - Walidację kodu XAML w trakcie kompilacji
  - Zmniejszenie pliku wyjściowego – sam kod w XAML nie jest osadzany w pakiecie
  - Można wyłączyć domyślną kompilację XAML – plik jest osadzany w pakiecie i kompilowany dopiero przy uruchomieniu – można tym sterować atrybutem XamlCompilationAttribute
  - Dla WPF-a można było utworzyć aplikację za pomocą samego XAML-a bez kodu. Dla .NET MAUI – nie próbowałem, ale wydaje się to możliwe.

Rozmieszczenie komponentów

Layouts

Filozofia rozmieszczania komponentów

- odejście od sztywnych ustawień pozycji i rozmiaru
- Część komponentów zwiera jeden element (content), część wiele (pojemniki/Layouts albo obsługa kolekcji np. ListView, CollectionView), a niektóre wcale (np. Button)
- zasady budowania interfejsów:
  - elementy kontrolne nie powinny mieć ustawionego rozmiaru – powinny one wypełniać pojemnik w którym się znajdują
  - można ograniczać minimalny i maksymalny rozmiar komponentów
  - położenie elementów nie jest opisane współrzędnymi ekranu, ale rozmiarem, rozkładem itp. pojemnika(ów) w których się znajdują. Przerwy pomiędzy komponentami narzuca się poprzez wartości parametru `Margin`.
  - pojemniki dzielą swoją przestrzeń pomiędzy swoje dzieci
  - pojemniki mogą być zagnieżdżone

Podstawowe rodzaje pojemników

- StackLayout
- HorizontalStackLayout
- VerticalStackLayout
- Grid
- FlexLayout
- AbsoluteLayout

Layout - właściwości

Rozmieszczenie elementów w pojemniku zależy od jego rodzaju, ale komponenty mają właściwości, które mogą w pewien sposób to rozmieszczenie modyfikować:

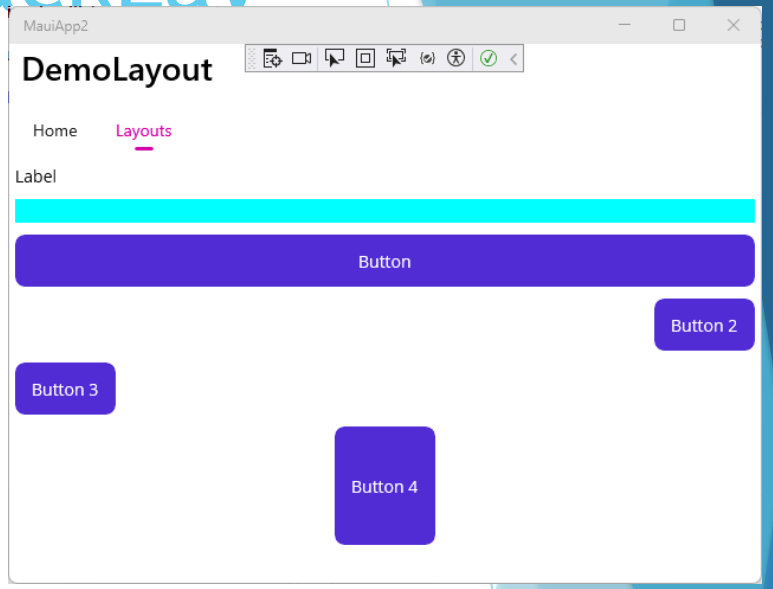
- HorizontalOptions: Start, Center, End, Fill
- VerticalOptions: Start, Center, End, Fill
- Margin
- MinimumWidthRequest/MinimumHeightRequest
- MaximumWidthRequest/MaximumHeightRequest
- WidthRequest/HeightRequest

Wyznaczanie wielkości komponentów

- Rozmieszczenie elementów pojemnika składa się z dwóch etapów:
  - pomiaru (*measure stage*) – pojemnik „pyta” swoje dzieci o ich preferowany rozmiar
  - rozmieszczenia (*arrange stage*) – dzieci są rozmieszczane w pojemniku na właściwych pozycjach
- Pojemniki nie wspierają przewijania (*scrolls*). Jest za to komponent – `ScrollViewer`.
- Informacje wykorzystywane przy określaniu rozmiaru komponentu to:
  - **Minimalny rozmiar** – rozmiar komponentu będzie co najmniej taki, jak jego minimalny rozmiar
  - **Maksymalny rozmiar** – rozmiar komponentu będzie mniejszy od jego maksymalnego rozmiaru (chyba, że minSize > maxSize)
  - **Zawartość -** jeśli zawartość komponentu będzie wymagała większego rozmiaru, to zostanie on powiększony
  - **Wielkość pojemnika -** jeżeli wielkość pojemnika jest mniejsza niż wielkość komponentu, to część komponentu zostanie obcięta
  - **`(Horizontal|Vertical)Options` –** pojemnik zwiększy rozmiar komponentu jeśli dla tej własności zostanie ustawiona wartość `Fill`

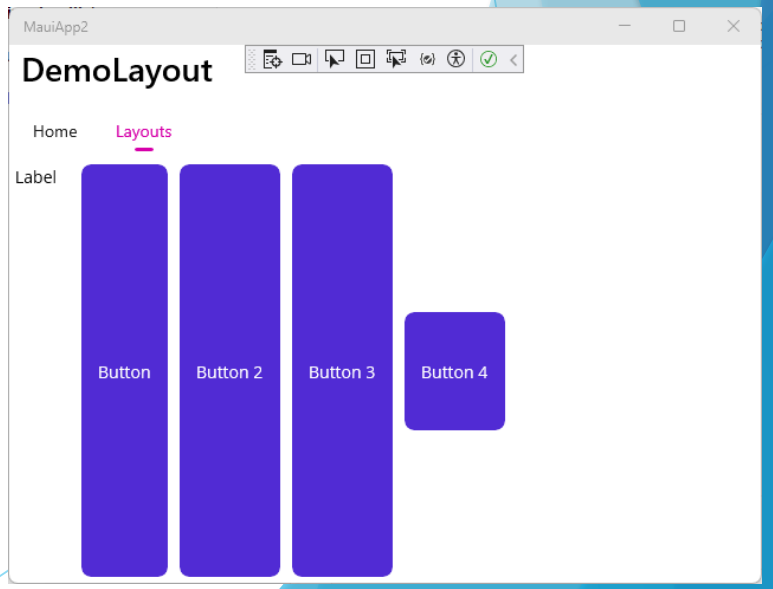
VerticalStackLayout/HorizontalStackLayout

```
    <VerticalStackLayout Margin="5" Spacing="10">
        <Label Text="Label"/>
        <BoxView Color="Aqua"  HeightRequest="20" />
        <Button Text="Button"></Button>
        <Button Text="Button 2" HorizontalOptions="End"/>
        <Button Text="Button 3" HorizontalOptions="Start"/>
        <Button Text="Button 4" HorizontalOptions="Center"
HeightRequest="100"/>
    </VerticalStackLayout>
```



Brakuje parametru width – komponent niewidoczny

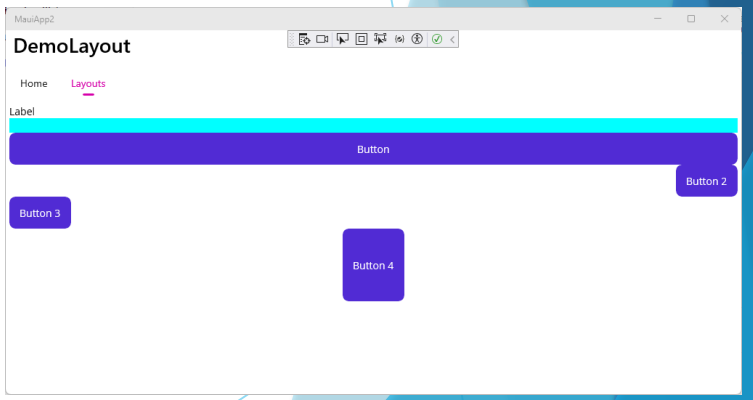
```
      <HorizontalStackLayout Margin="5" Spacing="10">
          <Label Text="Label"/>
          <BoxView Color="Aqua"  HeightRequest="20"  />
          <Button Text="Button"></Button>
          <Button Text="Button 2" HorizontalOptions="End"/>
          <Button Text="Button 3" HorizontalOptions="Start"/>
          <Button Text="Button 4" HorizontalOptions="Center"
  HeightRequest="100"/>
      </HorizontalStackLayout>
```

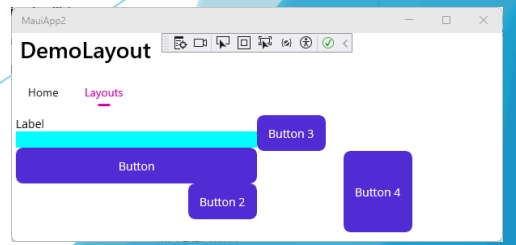


FlexLayout

- Umożliwia rozmieszczenie komponentów w pionie albo w poziomie, a także zawijanie elementów jeśli nie mieszczą się w jednej kolumnie albo wierszu.
- Komponent jest oparty o CSS Flexible Box Layout
- Właściwości komponentu:
  - Direction: Column, Row, ColumnReverse, RowReverse
  - Wrap: NoWrap, Wrap, Reverse
  - FlowDirection: LeftToRight, RightToLeft

```
    <FlexLayout Margin="5" Wrap="Wrap" FlowDirection="LeftToRight" Direction="Column"
>
        <Label Text="Label"/>
        <BoxView Color="Aqua"  HeightRequest="20"  />
        <Button Text="Button"></Button>
        <Button Text="Button 2" HorizontalOptions="End"/>
        <Button Text="Button 3" HorizontalOptions="Start"/>
        <Button Text="Button 4" HorizontalOptions="Center" HeightRequest="100"/>
    </FlexLayout>
```



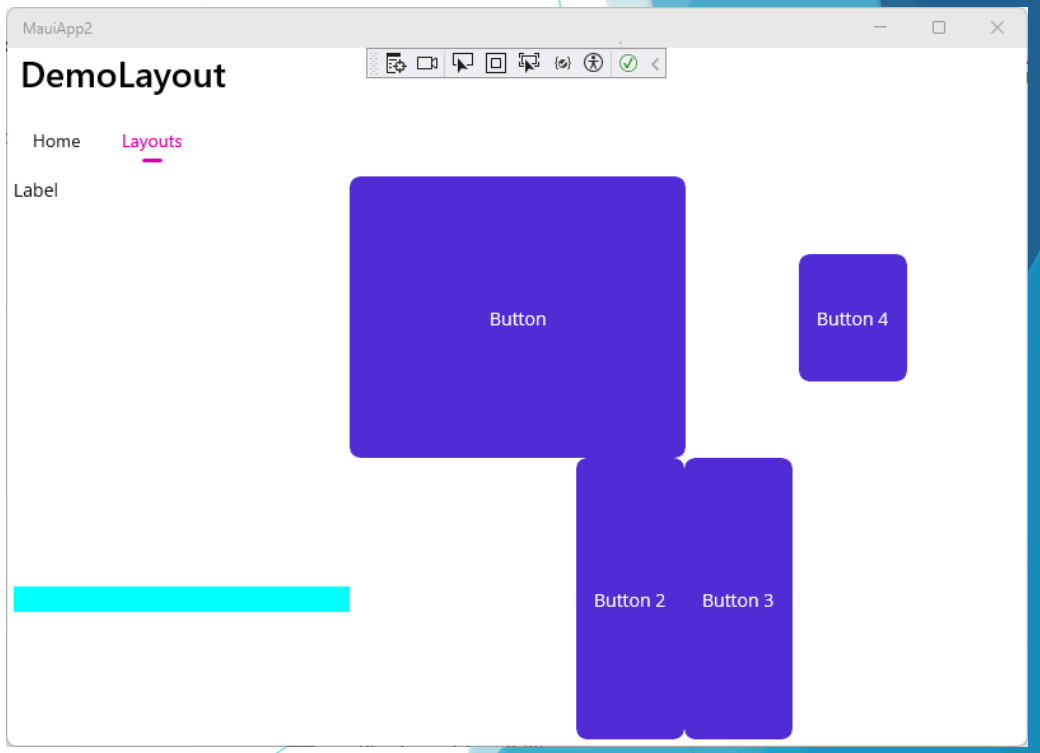


Grid

- najbardziej zaawansowany pojemnik
- tworzenie rozkładu za pomocą tego pojemnika przebiega dwuetapowo
  - Definicja wierszy (`Grid.RowDefinitions`) i kolumn (`Grid.ColumnDefinitions`)
  - Rozmieszczenie komponentów – właściwości `Grid.Row` i `Grid.Column`
- domyślnie 1 komórka – 1 komponent
- włożenie dwóch elementów do komórki powoduje ich nałożenie

Grid (2)

```
<Grid Margin="5" >
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Label Grid.Row="0" Grid.Column="0" Text="Label"/>
    <BoxView Grid.Row="1" Grid.Column="0"  Color="Aqua"  HeightRequest="20"  />
    <Button Grid.Row="0" Grid.Column="1" Text="Button" ></Button>
    <Button Grid.Row="1" Grid.Column="1" Text="Button 2" HorizontalOptions="End"/>
    <Button Grid.Row="1" Grid.Column="2" Text="Button 3" HorizontalOptions="Start"/>
    <Button Grid.Row="0" Grid.Column="2" Text="Button 4" HorizontalOptions="Center"
ightRequest="100"/>
</Grid>
```



Grid – rozmiary wierszy i kolumn

- rozmiary wierszy i kolumn podlegają jednej z następujących reguł, od której zależy sposób obsługi zmiany rozmiaru komponentu w przypadku zwiększenia rozmiaru pojemnika:
  - stały rozmiar – nieelastyczne

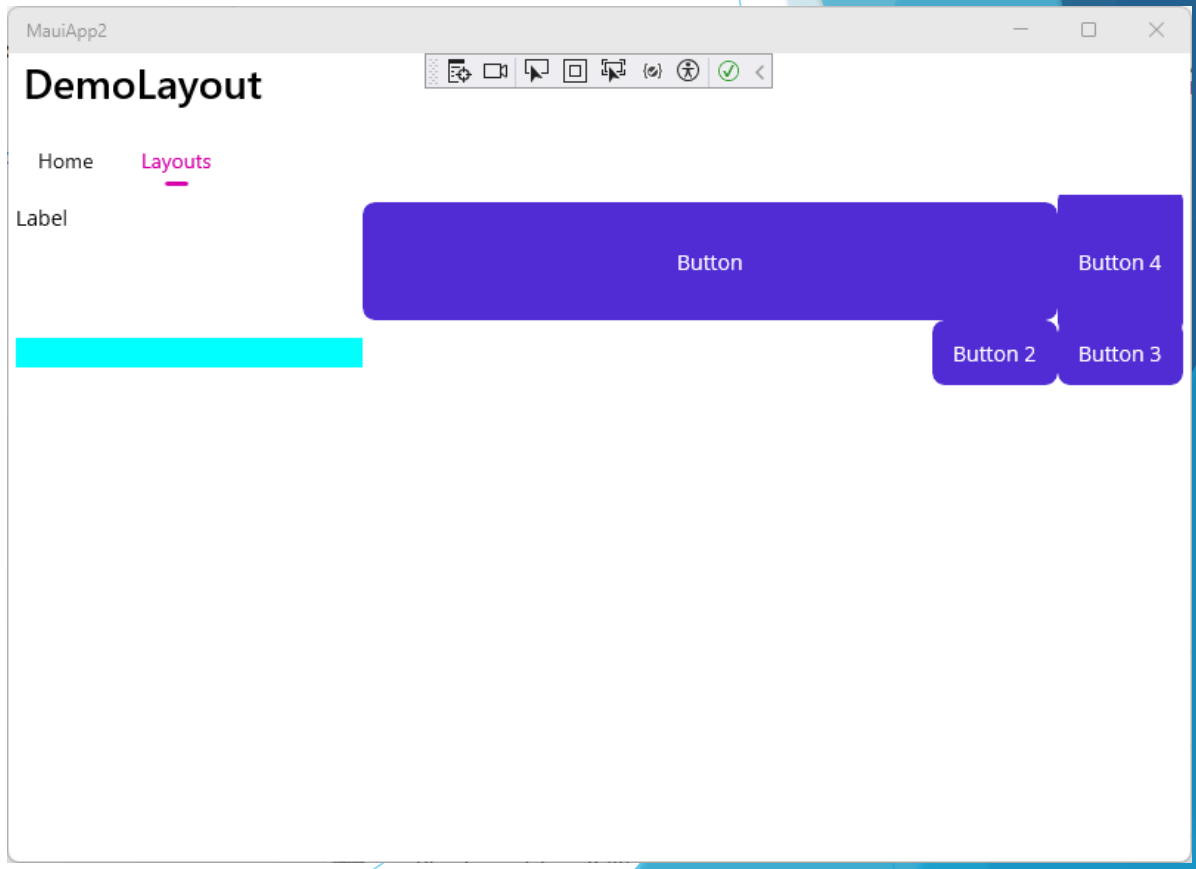
    ```
    <RowDefinition Height="80"/>
    ```
  - automatyczny – wiersz/kolumna dostają dokładnie tyle miejsca ile potrzebują

    ```
    <RowDefinition Height="Auto"/>
    ```
  - proporcjonalny – cały dostępny rozmiar jest proporcjonalnie dzielony przez wiersze/kolumny – wartość domyślna

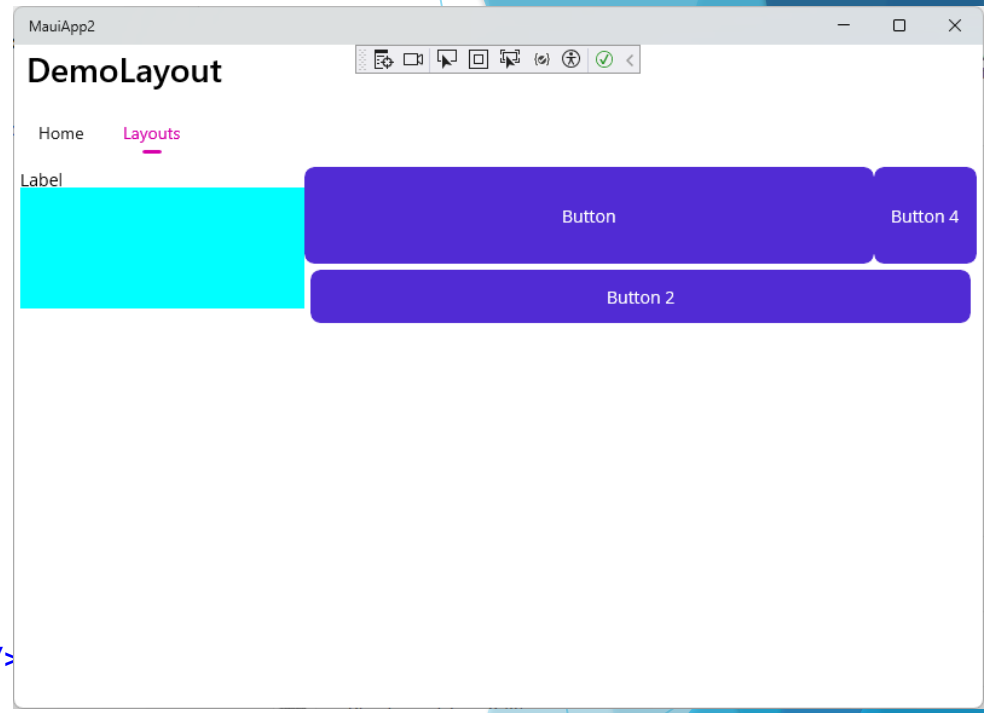
    ```
    <ColumnDefinition Width="*"/>
    <ColumnDefinition Width="2*"/>
    <ColumnDefinition Width="Auto"/>
    ```

Grid – rozmiary wierszy i kolumn

```
<Grid.RowDefinitions>
    <RowDefinition Height="80"/>
    <RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="*"/>
    <ColumnDefinition Width="2*"/>
    <ColumnDefinition Width="Auto"/>
</Grid.ColumnDefinitions>
```



Grid – „Rozpiętość” (Span) kolumn i wierszy




```
<Grid Margin="5" >
    <Grid.RowDefinitions>
        <RowDefinition Height="80"/>
        <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*"/>
        <ColumnDefinition Width="2*"/>
        <ColumnDefinition Width="Auto"/>
    </Grid.ColumnDefinitions>
    <Label Grid.Row="0" Grid.Column="0" Text="Label"/>
    <BoxView Grid.Row="0" Grid.Column="0"  Color="Aqua" Grid.RowSpan="2"  HeightRequest="100"  />
    <Button Grid.Row="0" Grid.Column="1" Text="Button" ></Button>
    <Button Grid.Row="1" Grid.Column="1" Text="Button 2" Grid.ColumnSpan="2" HorizontalOptions="Fill" Margin="5"/>
    <!--<Button Grid.Row="1" Grid.Column="2" Text="Button 3" HorizontalOptions="Start"/>-->
    <Button Grid.Row="0" Grid.Column="2" Text="Button 4" HorizontalOptions="Center" />
</Grid>
```

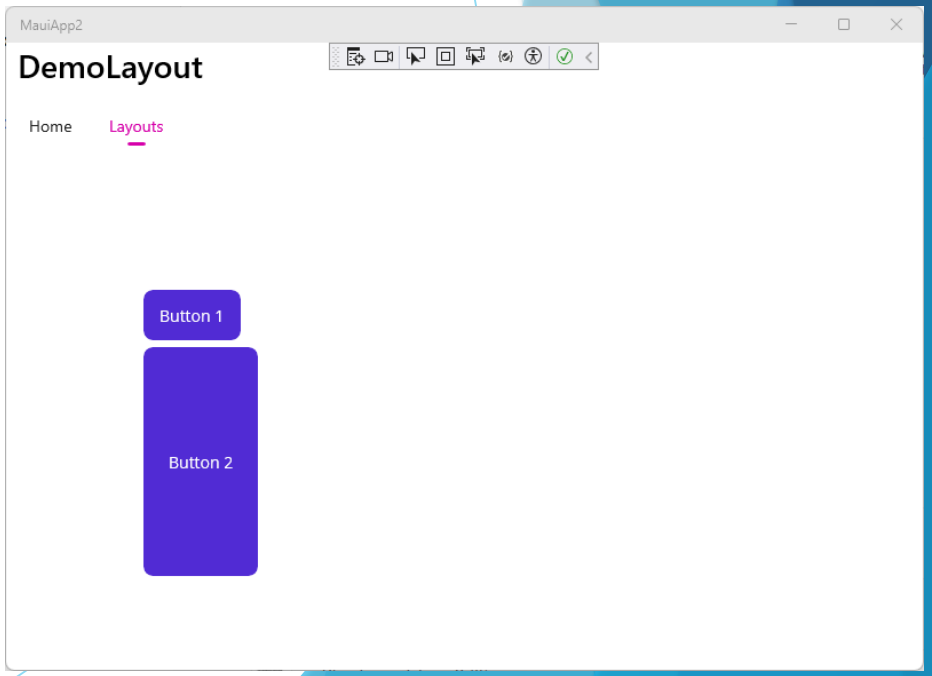
Skrócony opis wierszy i kolumn:

```
<Grid RowDefinitions="80, Auto"
      ColumnDefinitions="*, 2*, Auto"> ...
</Grid>
```

AbsoluteLayout

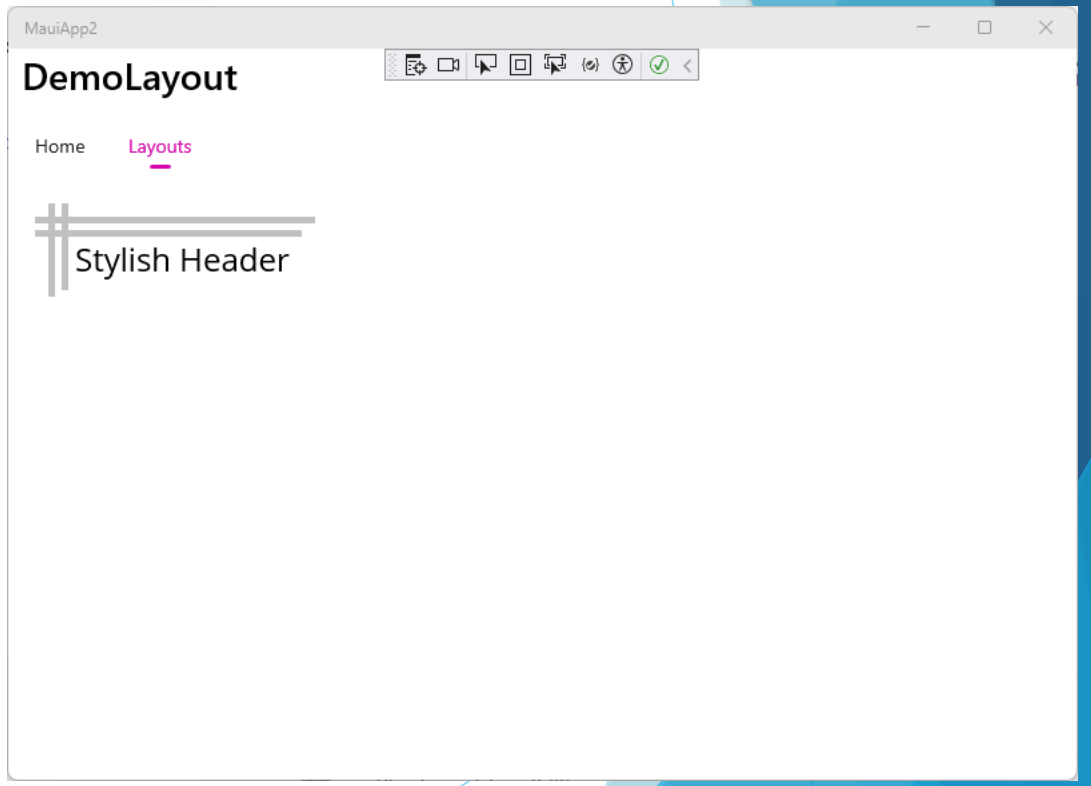
- umożliwia ręczne umiejscowienie elementów
- wartości podaje się jako proporcjonalne albo bezwzględne
- wykorzystuje się do tego właściwość AbsoluteLayout.LayoutBounds
- x, y – współrzędne
- x, y, width, height – współrzędne i rozmiar

```
<AbsoluteLayout Margin="20">
    <Button AbsoluteLayout.LayoutBounds="100, 100" Text="Button 1"/>
    <Button AbsoluteLayout.LayoutBounds="100, 150, 100, 200" Text="Button 2"/>
</AbsoluteLayout>
```



AbsoluteLayout pozycjonowanie bezwzględne

```
<AbsoluteLayout Margin="20">
    <BoxView Color="Silver"
             AbsoluteLayout.LayoutBounds="0, 10, 210, 5" />
    <BoxView Color="Silver"
             AbsoluteLayout.LayoutBounds="0, 20, 200, 5" />
    <BoxView Color="Silver"
             AbsoluteLayout.LayoutBounds="10, 0, 5, 70" />
    <BoxView Color="Silver"
             AbsoluteLayout.LayoutBounds="20, 0, 5, 65" />
    <Label Text="Stylish Header"
           FontSize="24"
           AbsoluteLayout.LayoutBounds="30, 25" />
</AbsoluteLayout>
```

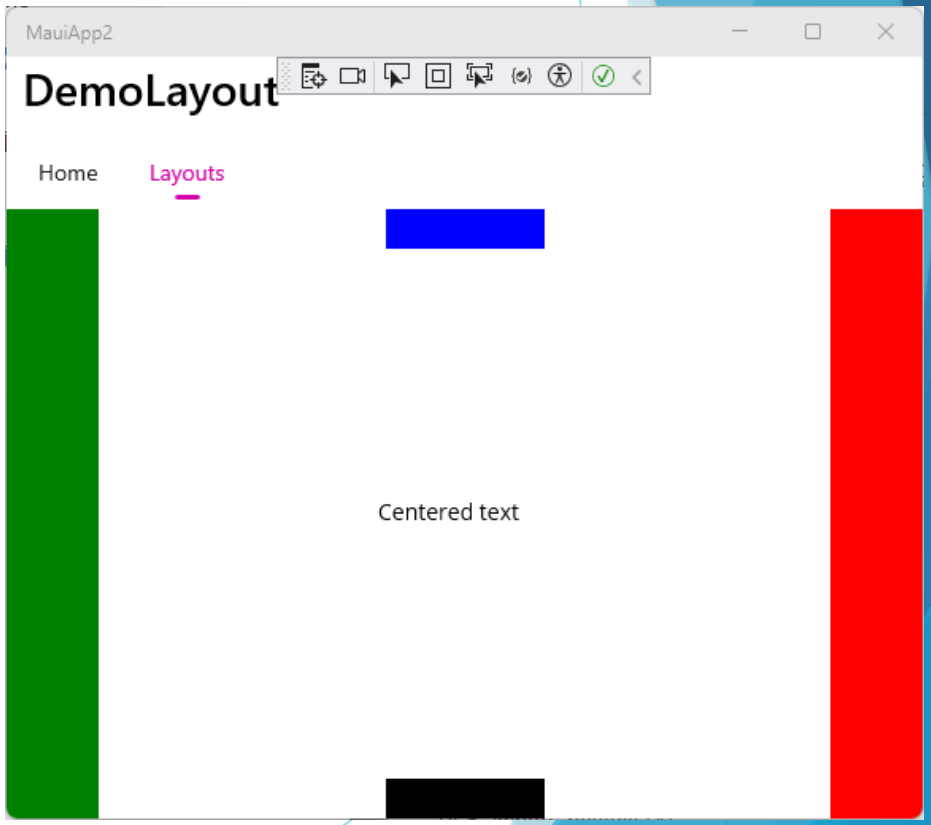


AbsoluteLayout – pozycjonowanie proporcjonalne

- Ustala się właściwość AbsoluteLayout.LayoutFlags, która może przyjmować następujące wartości:
  - None – wartości będą traktowane jako bezwzględne
  - XProportional/YProportional oznacza, że wartość X/Y będą proporcjonalne, a reszta bezwzględna
  - WidthProportional/HeightProportional – width/height będzie traktowana jako proporcjonalna a pozostałe jako bezwzględne
  - PositionProportional – pozycja proporcjonalna – reszta bezwzględna
  - SizeProportional – rozmiar proporcjonalny – reszta bezwzględna
  - All – wszystkie proporcjonalne

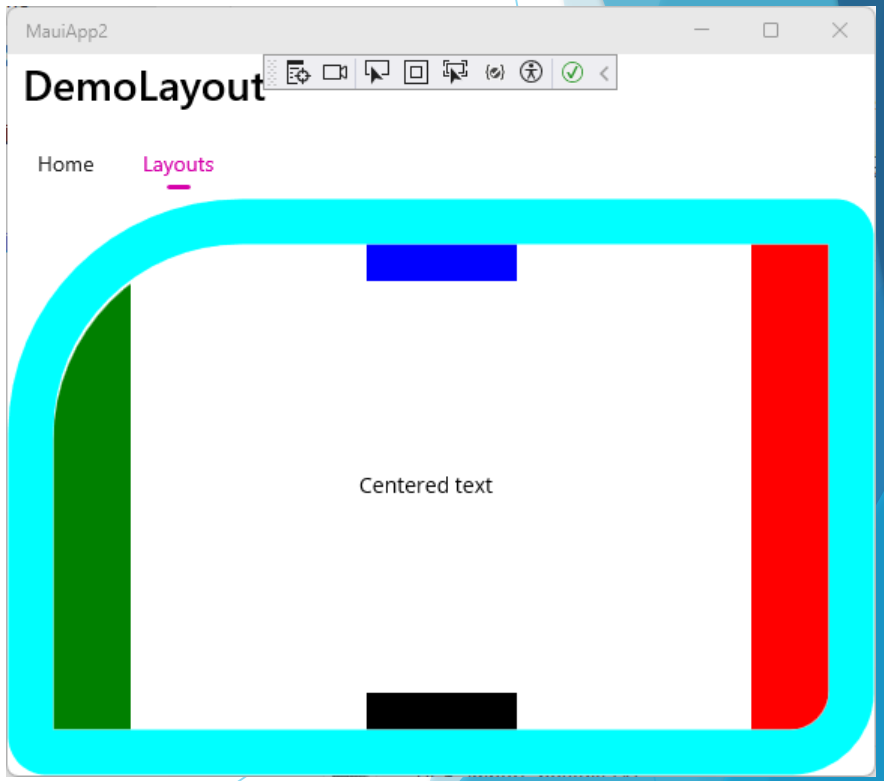
AbsoluteLayout pozycjonowanie bezwzględne

```
<AbsoluteLayout>
    <BoxView Color="Blue"
             AbsoluteLayout.LayoutBounds="0.5,0,100,25"
             AbsoluteLayout.LayoutFlags="PositionProportional" />
    <BoxView Color="Green"
             AbsoluteLayout.LayoutBounds="0,0,0.1,1.0"
             AbsoluteLayout.LayoutFlags="All" />
    <BoxView Color="Red"
             AbsoluteLayout.LayoutBounds="1,0,0.1,1.00"
             AbsoluteLayout.LayoutFlags="All" />
    <BoxView Color="Black"
             AbsoluteLayout.LayoutBounds="0.5,1,100,25"
             AbsoluteLayout.LayoutFlags="PositionProportional" />
    <Label Text="Centered text"
           AbsoluteLayout.LayoutBounds="0.5,0.5,110,25"
           AbsoluteLayout.LayoutFlags="PositionProportional" />
</AbsoluteLayout>
```



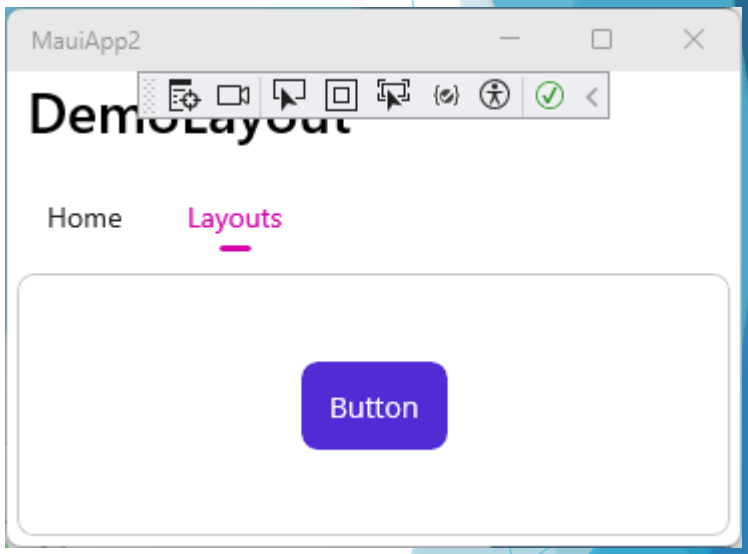
Ramka - *Border*

```
<Border Stroke="Aqua" StrokeThickness="30" StrokeDashArray="90,80"
        StrokeShape="RoundRectangle 140,10,10,40">
    <AbsoluteLayout>
        <BoxView Color="Blue"
                 AbsoluteLayout.LayoutBounds="0.5,0,100,25"
                 AbsoluteLayout.LayoutFlags="PositionProportional" />
        <BoxView Color="Green"
                 AbsoluteLayout.LayoutBounds="0,0,0.1,1.0"
                 AbsoluteLayout.LayoutFlags="All" />
        <BoxView Color="Red"
                 AbsoluteLayout.LayoutBounds="1,0,0.1,1.00"
                 AbsoluteLayout.LayoutFlags="All" />
        <BoxView Color="Black"
                 AbsoluteLayout.LayoutBounds="0.5,1,100,25"
                 AbsoluteLayout.LayoutFlags="PositionProportional" />
        <Label Text="Centered text"
               AbsoluteLayout.LayoutBounds="0.5,0.5,110,25"
               AbsoluteLayout.LayoutFlags="PositionProportional" />
    </AbsoluteLayout>
</Border>
```



Ramka :) – *Frame*

```
   <Frame Padding="7" Margin="5">
       <Button Text="Button" HorizontalOptions="Center"
VerticalOptions="Center"/>
   </Frame>
```



- Właściwości:
  - BorderColor
  - CornerRadius
  - HasShadow
  - x, y, width, height – współrzędne i rozmiar
- Zaleca się używanie kontrolki Border.

Wiązanie (binding) danych

Ręczna synchronizacja danych – bez wiązania

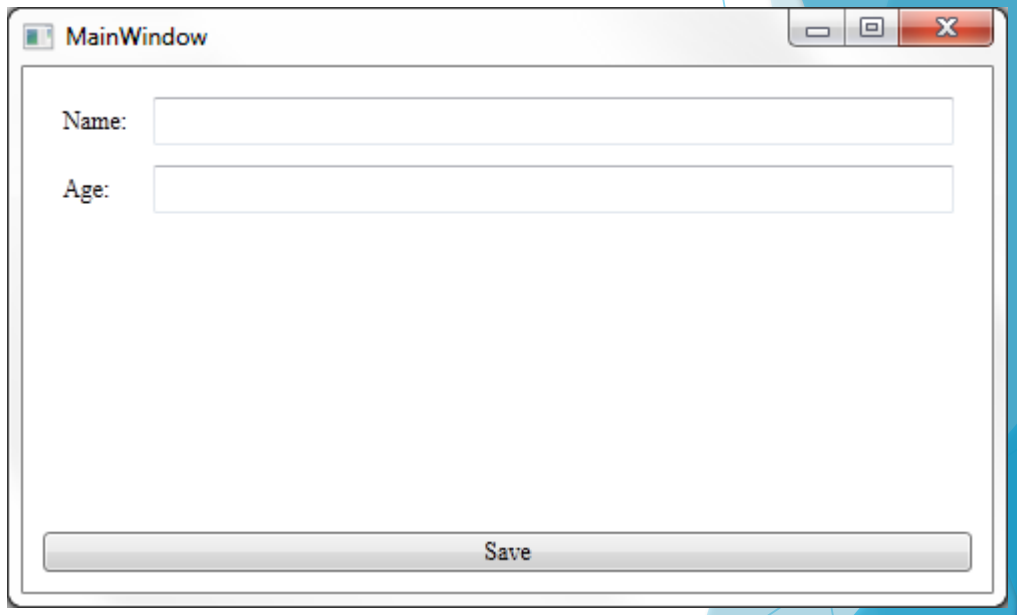
```
public class Person
{
    public string Name { get; set; }
    public int Age { get; set; }

}
```

```
<TextBox Name="txtName"
         TextChanged="txtName_TextChanged">
</TextBox>

<TextBox Name="txtAge"
         TextChanged="txtName_TextChanged">
</TextBox>

<Button Name="SaveButton"
        Click="SaveButton_Click">Save
</Button>
```



Wiązanie danych

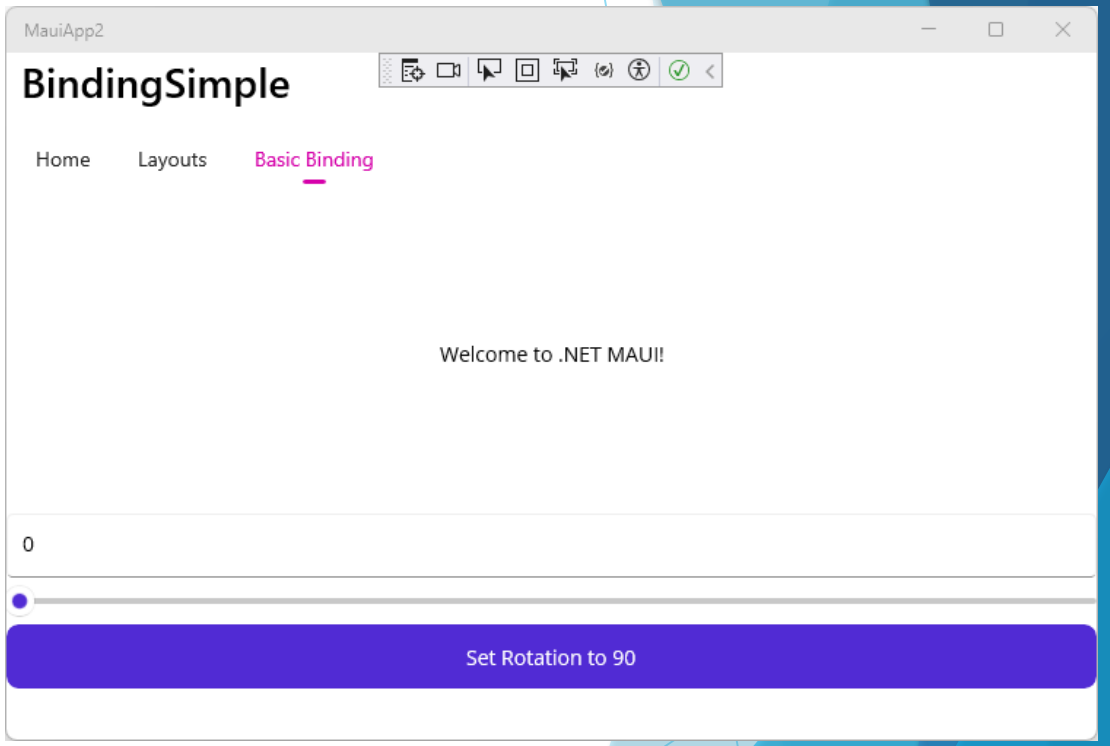
- jest to związek, który mówi skąd dane powinny zostać pobrane (obiekt źródłowy) aby ustawić właściwość w obiekcie docelowym
- właściwość docelowa jest zawsze *BindableProperty*
- najprostszym zastosowaniem jest wiązanie dwóch takich właściwości
- realizacja wiązania poprzez klasę `Binding` z pakietu `Microsoft.Maui.Controls`
- niepoprawne wiązanie nie generuje wyjątków

Proste wiązanie właściwości

```
<VerticalStackLayout>

    <Label
        x:Name="WelcomeLabel"
        Text="Welcome to .NET MAUI!"
        VerticalOptions="Center"
        HorizontalOptions="Center"
        Rotation="{Binding Source={x:Reference rotationSlider},
                           Path=Value }"
        Padding="100"
        />
    <Entry Text="{Binding Source={x:Reference rotationSlider},
                          Path=Value}"/>
    <Slider x:Name="rotationSlider"  Minimum="0" Maximum="360" />
    <Button Text="Set Rotation to 90" Clicked="Button_Clicked"/>
</VerticalStackLayout>
```

```
public BindingSimple()
{
    InitializeComponent();
    Binding binding = new Binding();
    binding.Source = rotationSlider;
    binding.Path = "Value";
    WelcomeLabel.SetBinding(Slider.RotationProperty, binding);
}
```



Tryby wiązania

„Ręczne” ustawienie wartości - zerwanie wiązania!

```
private void Button_Clicked(object sender, EventArgs e)
{
    //rotationSlider.Value = 90;
    WelcomeLabel.Rotation = 90;
}
```

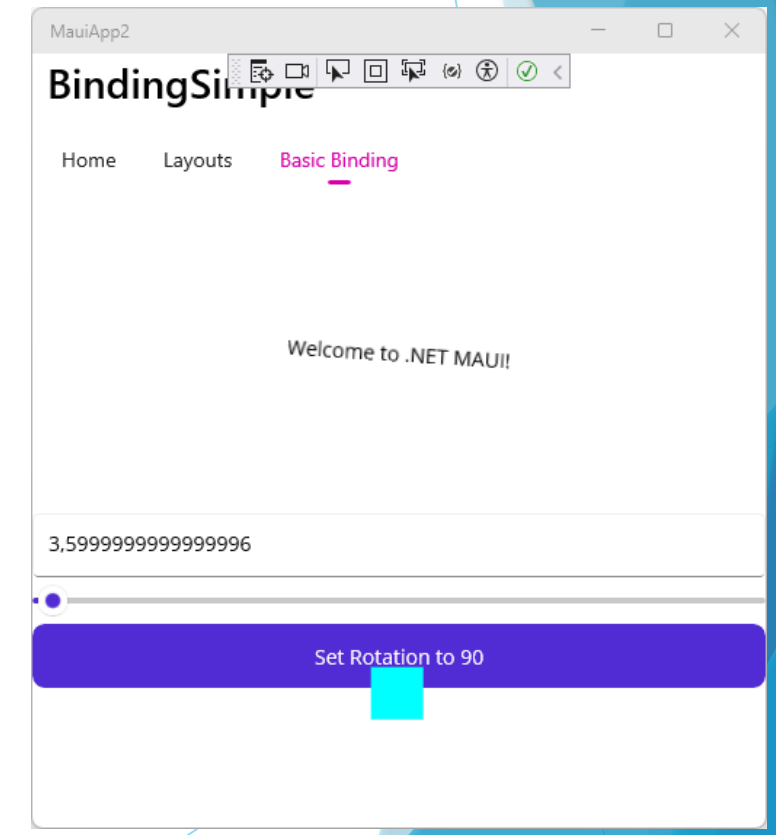
Ustawienie wiązania dwukierunkowego:

```
Rotation="{Binding Source={x:Reference rotationSlider},
                   Path=Value,
                   Mode=TwoWay}"
```

A co będzie, gdy dane będą sprzeczne?

```
<BoxView x:Name="AquaBox" Color="Aqua" HeightRequest="10" WidthRequest="10"
     ScaleX="{Binding Source={x:Reference rotationSlider},
                      Path=Value,Mode=TwoWay}"
     ScaleY="{Binding Source={x:Reference rotationSlider},
                      Path=Value,Mode=TwoWay}"></BoxView>

private void Button_Clicked(object sender, EventArgs e)
{
   WelcomeLabel.Rotation = 90;
   AquaBox.ScaleX = 2;
   AquaBox.ScaleY = 1;
}
```



Tryby wiązania

`BindingMode`

- `OneWay` - jednostronne od źródła do celu
- `TwoWay` - dwustronne
- `OneTime` – jednorazowe od źródła do celu ale tylko gdy zmienia się BindingContext
- `OneWayToSource` – jednostronne od celu do źródła
- `Default` - domyślne – zależne od właściwości

Wiązanie do obiektów nie będących elementami wizualnymi

- dane muszą być publicznymi właściwościami obiektu
- dopuszczalne są następujące rozwiązania:
  - Source – wiązanie do obiektu
  - RelativeSource – pozwala wiązać właściwości do właściwości obiektów w hierarchii komponentów
  - BindingContext – wiązanie do właściwości `BindingContext` komponentu – źródłem zostaje pierwszy element w hierarchii który ma wartość różną od `null`

Wiązanie typu *Source*

```
namespace MauiApp2;

public partial class BindingSimple : ContentPage
{
    public double DefaultRotation { get; set; } = 45.0;

    public BindingSimple()
    {
        InitializeComponent();

    }
}
```

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace=MauiApp2"
             x:Class="MauiApp2.BindingSimple"
             x:Name="MyPage"
             Title="BindingSimple">
    <VerticalStackLayout>

        <Label
            x:Name="WelcomeLabel"
            Text="Welcome to .NET MAUI!"
            VerticalOptions="Center"
            HorizontalOptions="Center"
            Rotation="{Binding Source={x:Reference MyPage}, Path=DefaultRotation}"
            Padding="100"
            />
```

Wiązania typu *Source*

```
<Window x:Class="WpfApplication1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:sys="clr-namespace:System;assembly=mscorlib"
        xmlns:local="clr-namespace:WpfApplication1"
        Title="MainWindow" >

    <Window.Resources>
        <sys:Double x:Key="DefaultNameX">1.0</sys:Double>
    </Window.Resources>

    ...
    <Slider Name="ScaleSlider" Value="{Binding Source={StaticResource DefaultNameX},Mode=OneWay}" />
```

Wiązania typu RelativeSource

Tryb wyszukiwania:

- Self – wiązanie do innej właściwości tego samego obiektu
- FindAncestor – wiązanie do rodzica. Podaje się AncestorType – czyli jakiego typu rodzica szukamy, a następnie opcjonalnie AncestorLevel w celu pominięcia zadanej liczby wystąpień tego typu
- TemplatedParent – używane w szablonach

Wiązania typu `RelativeSource` (2)

- Wiązanie do siebie samego albo elementu nadrzędnego na nieznanym poziomie w hierarchii
- Wiązanie tego typu wykorzystuje obiekt klasy `RelativeSource`

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:p="clr-namespace:MauiApp2"
             x:Class="MauiApp2.BindingSimple"
             x:Name="MyPage"
             Title="BindingSimple">

    <VerticalStackLayout>

        <Label Text="{Binding Source={x:RelativeSource Mode=FindAncestor,
                                                       AncestorType={x:Type ContentPage}},
                              Path=Title,
                              StringFormat='{0}'}">
        </Label>


    </VerticalStackLayout>
</ContentPage>
```

Wiązania typu BindingContext

```
<ContentPage.Resources>
    <p:Person x:Key="PersonKowalski" Id="1" Name="Jan Kowalski" Title="Pan" ></p:Person>
</ContentPage.Resources>
```

```
 <HorizontalStackLayout>
     <Label Text="{Binding Source={StaticResource Key=PersonKowalski},Path=Id}"></Label>
     <Label Text="{Binding Source={StaticResource Key=PersonKowalski},Path=Title}"></Label>
     <Label Text="{Binding Source={StaticResource Key=PersonKowalski},Path=Name}"></Label>
 </HorizontalStackLayout>
```

```
<HorizontalStackLayout BindingContext="{Binding Source={StaticResource Key=PersonKowalski}}">
    <Label Text="{Binding Id}"></Label>
    <Label Text="{Binding Title}"></Label>
    <Label Text="{Binding Name}"></Label>
</HorizontalStackLayout>
```

Dane są wyszukiwane w hierarchii komponentów do czasu napotkania właściwości BindingContext różnej od null.

Konwersja powiązanych danych

- Można wyróżnić dwa rodzaje konwersji:
  - **formatowanie tekstów** – gdy dane są w postaci tekstowej – zazwyczaj zawierają liczby lub daty
  - **konwertery wartości** (*value converters*) – pozwalające na konwersję danych dowolnego typu

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:p="clr-namespace:MauiApp2"
             xmlns:sys="clr-namespace:System;assembly=mscorlib"


 <Label Text="{Binding Source={x:Static sys:DateTime.Now},StringFormat='Data: {0:yyyy-MMMM-dd}'}"></Label>
```

Konwertery wartości

```
namespace MauiApp2
{
    public class BoolToColorConverter : IValueConverter
    {
        public object? Convert(object? value, Type targetType, object? parameter, CultureInfo culture)
        {
            bool val = (bool)value;

            if ( val == true)
                return new Color(0, 255, 0);

            else
                return new Color(255, 0, 0);
        }

        public object? ConvertBack(object? value, Type targetType, object? parameter, CultureInfo culture)
        {
            throw new NotImplementedException();
        }
    }
}
```

```
<ContentPage.Resources>
    <p:BoolToColorConverter x:Key="BTCConverter"/>
</ContentPage.Resources>


<CheckBox x:Name="ColorCheckBox"/>

 <Label  Text="To jest tekst" TextColor="{Binding Source={x:Reference ColorCheckBox},
                                                  Path=IsChecked,
                                                  Converter={StaticResource BTCConverter}}">
```

Konwertery wartości (2)

```
public class ValueToColorConverter:IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
    {

        double val = (double)value;
        if (val < (double) parameter)
        {
            return new SolidColorBrush(Colors.Red);
        }
        else
        {
            return new SolidColorBrush(Colors.Black);
        }
    }

    public object ConvertBack(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

Konwertery wartości (3)

```
<Button Name="SetScaleButton" Click="SetScaleButton_Click"  >
     <Button.Foreground>
          <Binding ElementName="ScaleSlider" Path="Value" Converter="{StaticResource ColorConverter}">

               <Binding.ConverterParameter>
                    <sys:Double>
                        0.5
                    </sys:Double>
               </Binding.ConverterParameter>

          </Binding>
      </Button.Foreground>
      Set Scale = 1
</Button>
```

Konwertery wartości (4)

```
public class ValueToColorConverter : IValueConverter
{
    public double MinWarningThreshold { get; set; }
    public double MaxWarningThreshold { get; set; }

    public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
    {

        double val = (double)value;
        if (val < MinWarningThreshold || val > MaxWarningThreshold)
        {
            return new Color(0, 0, 0);
        }
        else
        {
            return new Color(255, 0, 0);
        }
    }

    public object ConvertBack(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

Konwertery wartości (4)

```
<Label Text="Jakiś tam tekst">
    <Label.TextColor>
        <Binding Source="{x:Reference rotationSlider}" Path="Value">
            <Binding.Converter>
                <p:ValueToColorConverter MinWarningThreshold="100" MaxWarningThreshold="200"/>
            </Binding.Converter>
        </Binding>
    </Label.TextColor>
</Label>
```

Konwertery wielu wiązań (*multibinding*)

```
public class ValueToColorConverter:IMultiValueConverter
   {
       public double MinWarningThreshold { get; set; }
       public double MaxWarningThreshold { get; set; }

       public object Convert(object[] values, Type targetType, object parameter, System.Globalization.CultureInfo culture)
       {
           bool isChecked = (bool)values[0];
           double val = (double)values[1];
           if ( isChecked && ( val < MinWarningThreshold || val > MaxWarningThreshold))
           {
               return new SolidColorBrush(Colors.Red);
           }
           else
           {
               return new SolidColorBrush(Colors.Black);
           }

       }

       public object[] ConvertBack(object value, Type[] targetTypes, object parameter, System.Globalization.CultureInfo culture)
       {
           throw new NotImplementedException();
       }
   }
```

Konwertery wielu wiązań (*multibinding*)

```
<Button Name="SetScaleButton" Click="SetScaleButton_Click"  >
     <Button.Foreground>
         <MultiBinding >
             <MultiBinding.Converter>
                 <local:ValueToColorConverter MinWarningThreshold="0.5" MaxWarningThreshold="1.5"/>
             </MultiBinding.Converter>
             <Binding ElementName="IsScaleWarningEnabled" Path="IsChecked"/>
             <Binding ElementName="ScaleSlider" Path="Value"/>
         </MultiBinding>
     </Button.Foreground>

     Set Scale = 1</Button>

  <CheckBox Name="IsScaleWarningEnabled"  Content="scale warning"/>
```