# Graphical User Interface

With inspiration from the [Avalonia Documentation]

# Agenda

1. GUI & Event Driven Programming
2. Introduction to Avalonia
3. Basic Avalonia Concepts
   - *AXAML*
   - *Controls*
   - *Layouts*
   - *File Dialogs*

# DISCLAIMER ☺

- This is not a design course.
- Every aspect of Avalonia will not be covered.
- This lecture is designed to provide a foundational understanding of Avalonia.
- The practical focus will be on demonstrating core concepts and building a simple Avalonia application.

# GUI

A graphical user interface (GUI) allows a user to interact with a program through elements such as a keyboard, mouse, or touchscreen.

- This is in contrast to a command-line interface (CLI) where interaction is solely through text commands.
- Interaction happens through components (often called controls).
- The user interacts with the program by performing actions (such as clicking a mouse button) on these components.

# GUI vs. Command-Line Interface (CLI):

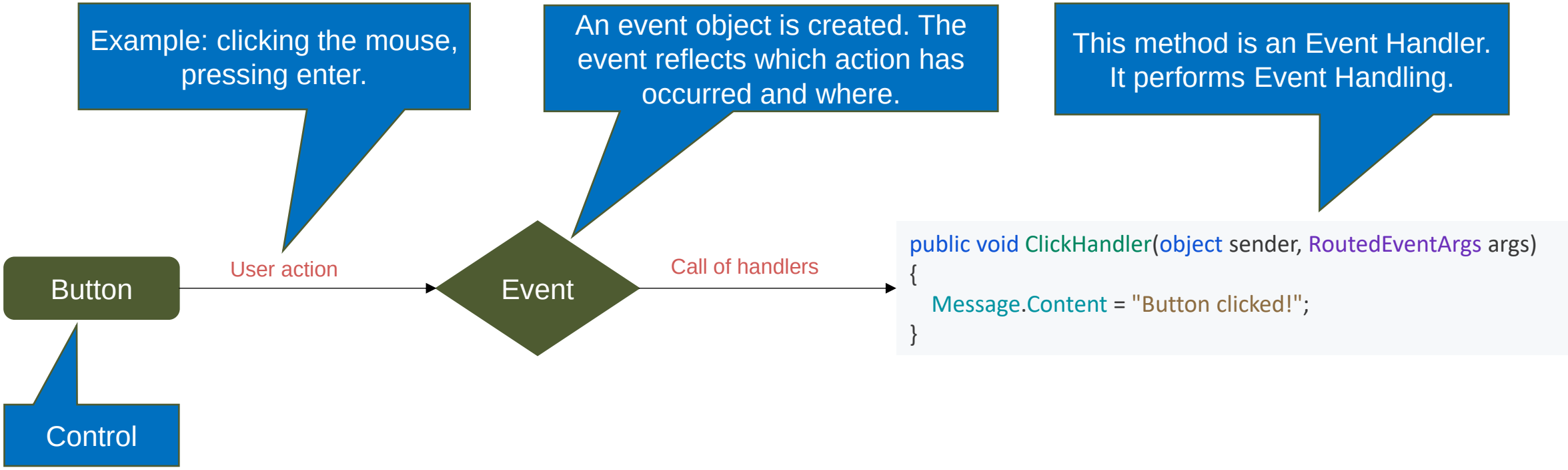
A GUI differs significantly from a Command-Line Interface (CLI):
In a CLI:

- The program executes sequentially through the main(...) method.
- The flow is controlled by the programmer and can be influenced by the user.

In a GUI, event-driven programming is used:

- Code execution occurs when an event arises.
- An event can be triggered by a user action (foreground event) or by a system action (background event).
- The program handles this event to determine what should happen next.

# Event-Driven Programming



# GUI Flow

**A user interacts with a component:** The user performs an action on an element in the interface, such as clicking a button or typing in a text box.

**An Event object is created:** When the user interacts, an object called an event object is created. This object contains information about what happened, such as the type of action performed (e.g., a click) and where on the screen.

**Event Handler(s) are called:** If there is one or more methods (called event handlers) associated with the specific type of event, these methods are called. The event object is passed as an argument to the method.

**The Event Handler executes code:** The called method contains the code that should be executed in response to the event. This code can update the interface, perform calculations, or communicate with other parts of the program.

# Introduction to Avalonia

With inspiration from the [Avalonia Documentation]

# What is Avalonia?

- **Cross-platform compatibility:** Write once, run anywhere.
- **Modern and flexible:** Create beautiful and responsive UIs.
- **Open-source and community-driven:** Benefit from a vibrant community and contribute to the framework's development.
- **Avalonia uses XAML** (Extensible Application Markup Language) to define the user interface and **C#** for the application logic.







https://docs.avaloniaui.net/docs/basics/user-interface/introduction-to-xaml

# Avalonia Templates Installation

- Get Avalonia templates in your Rider environment. Follow the instructions given in the link below:

  https://docs.avaloniaui.net/docs/get-started/install

- After Avalonia templates Installation, and Rider restart, you should be able to see the Avalonia templates (shown at the next slide)

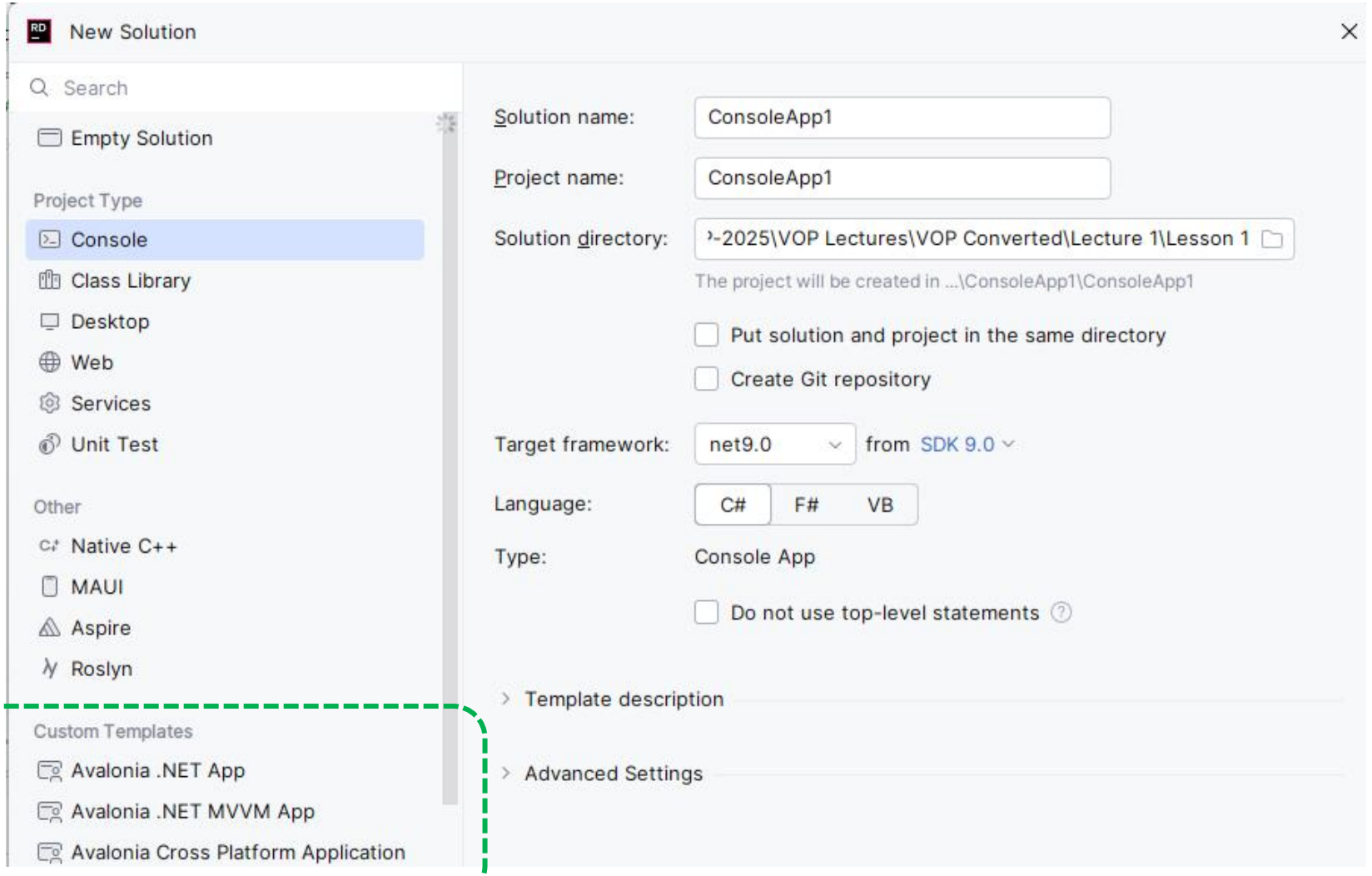






https://docs.avaloniaui.net/docs/basics/user-interface/file-dialogs

# Avalonia Templates Installation








https://docs.avaloniaui.net/docs/basics/user-interface/file-dialogs

# Avalonia Plugin Installation

Install Avalonia plugin to provide a design previewer for your axaml file, follow the instructions below:

- Open Rider's plugin settings: Go to "Settings/Preferences" -> "Plugins".
- Search for the plugin: In the Marketplace tab, search for "AvaloniaRider".
- Install the plugin: Click "Install" on the AvaloniaRider plugin.
- Restart Rider: Rider will prompt you to restart the IDE. Click "Restart IDE" to apply the changes.

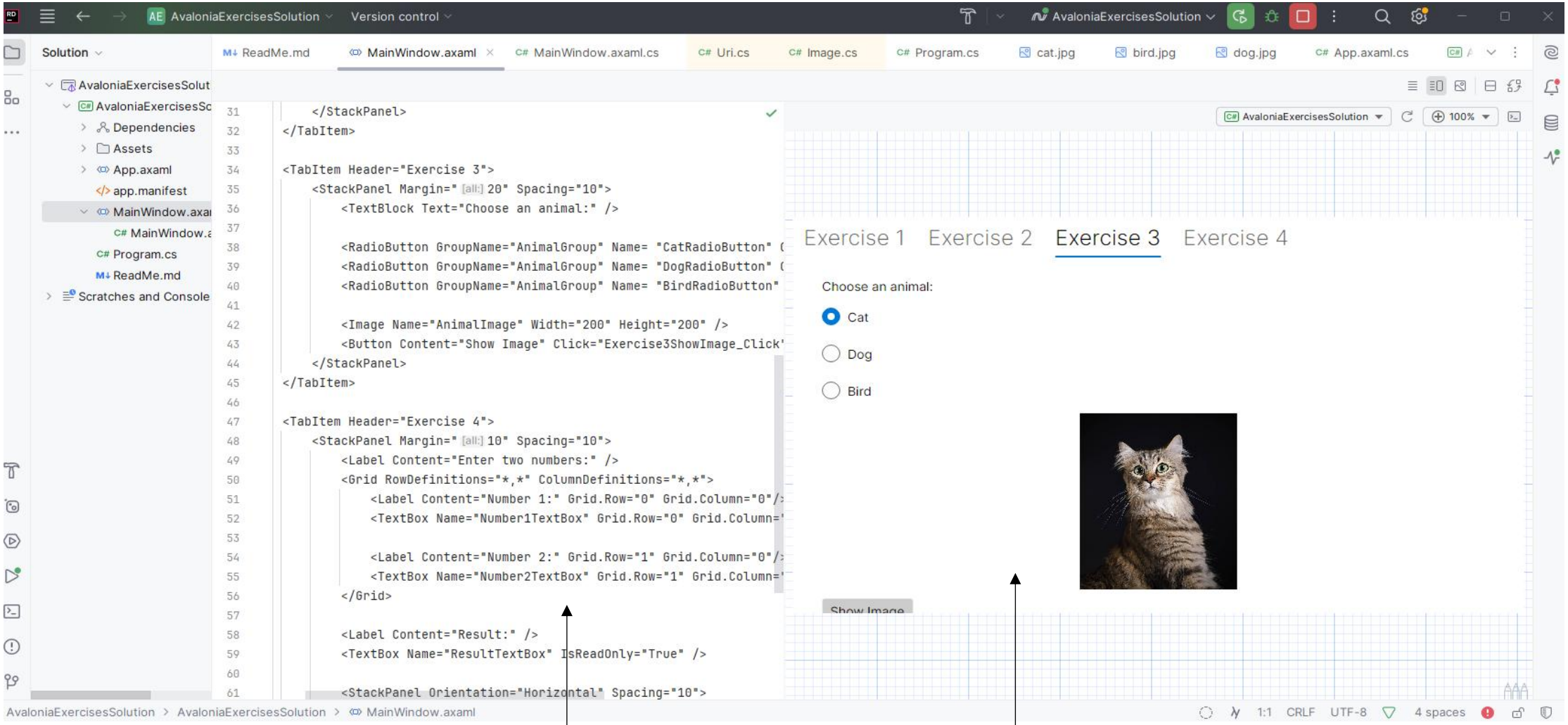






https://docs.avaloniaui.net/docs/basics/user-interface/file-dialogs

# Avalonia Plugin Installation




axaml file

design previewer for your axaml file





https://docs.avaloniaui.net/docs/basics/user-interface/file-dialogs

# Basic Avalonia Concepts

- **AXAML**
- **Controls**
- **Layout**
- **File Dialogs**

# What is AXAML?

Why so many different XML's?

- XML (Extensible Markup Language) – A markup language
- X**A**ML (Extensible **Application** Markup Language) – Declarative language used to build user interfaces
- **A**XAML (**Avalonia** XAML) – It is essentially the same as XAML, but due to technical issues it needed another name.
- They all utilize tags to define their content



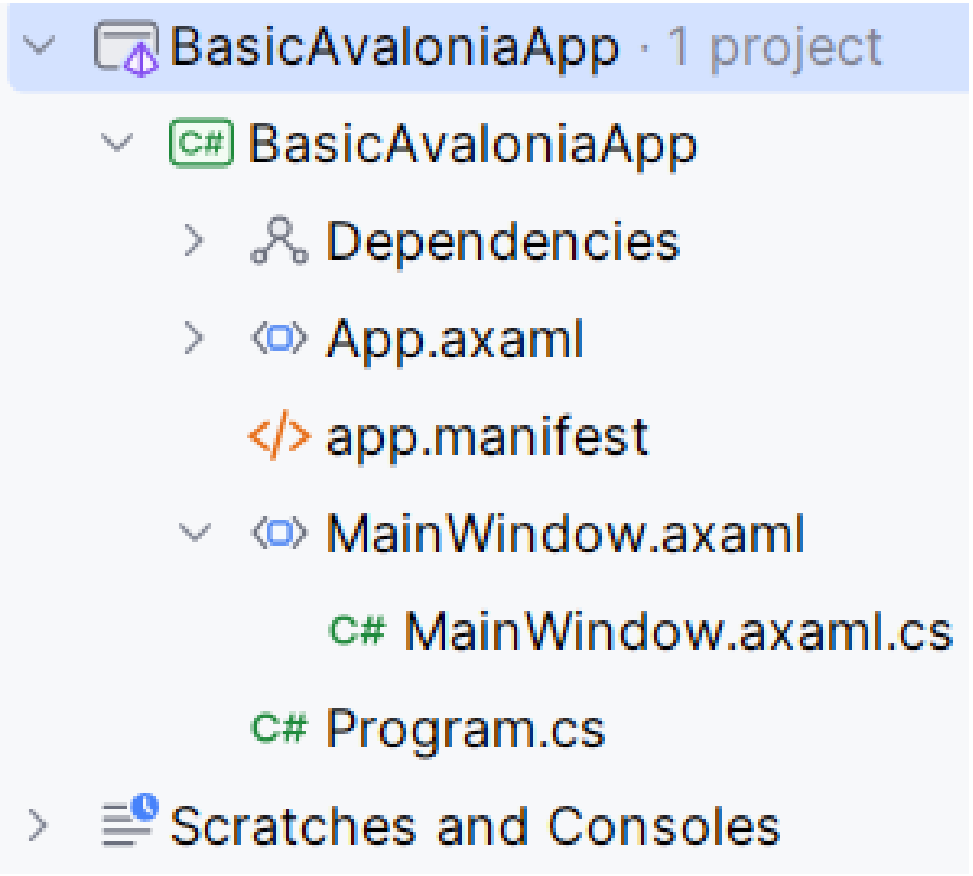




https://docs.avaloniaui.net/docs/basics/user-interface/introduction-to-xaml

# Avalonia Application Templates

- Avalonia .NET App
- Avalonia Cross-Platform Application
- Avalonia .NET MVVM Application

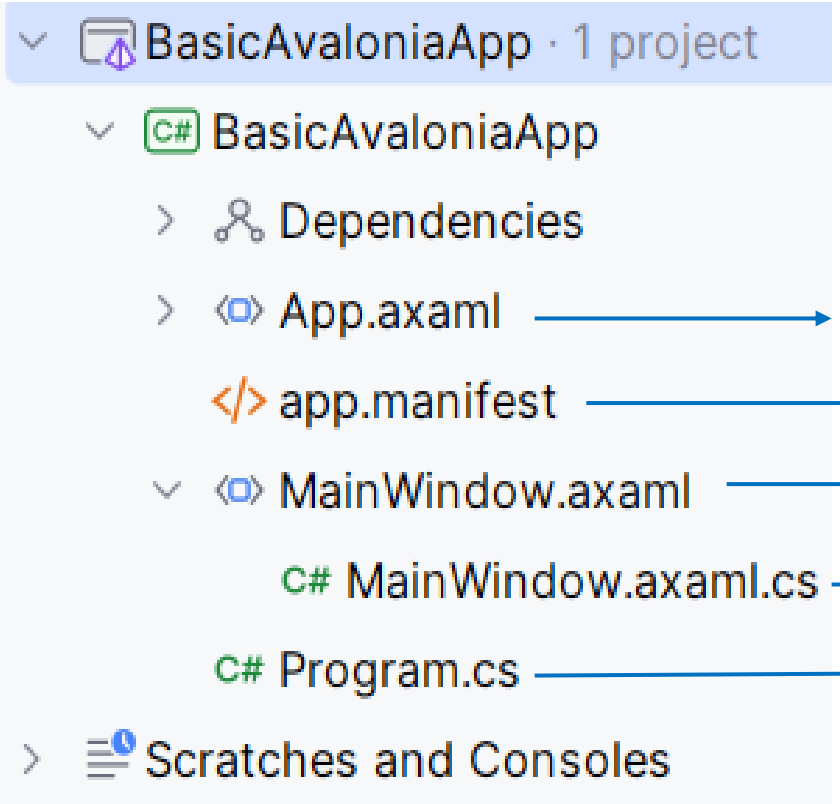







https://docs.avaloniaui.net/docs/basics/user-interface/introduction-to-xaml

# Avalonia .NET App Template



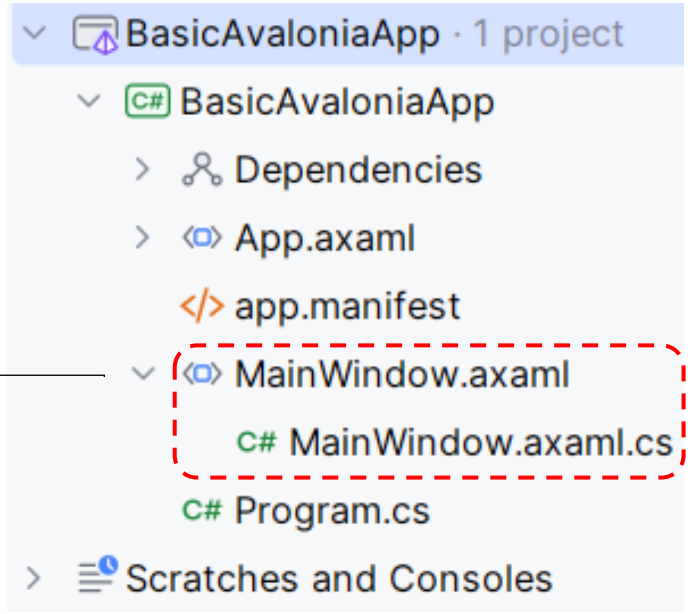

- App.axaml → This file defines the application-level resources, styles, and settings for your Avalonia application.
- app.manifest → This file contains metadata about your application
- MainWindow.axaml → This file defines the user interface (UI) of your main application window
- MainWindow.axaml.cs → This file contains the logic and event handlers for the main window.
- Program.cs → This file contains the code to initialize and run the application.





https://docs.avaloniaui.net/docs/basics/user-interface/introduction-to-xaml

# What is “Code-behind”?




MainWindow.axaml

```
<Window xmlns="https://github.com/avaloniaui"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        x:Class="BasicAvaloniaApp.MainWindow">

    <Button Click="ButtonClickHandler">Hello World</Button>
</Window>
```

MainWindow.axaml.cs

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
    }

    public void ButtonClickHandler(object sender, RoutedEventArgs e)
    {
        // code here.
    }
}
```

https://docs.avaloniaui.net/docs/basics/user-interface/code-behind

# What is “Code-behind”?

- It is the file where we write and control the logic of our UI
  - By convention it is suffixed with `.axaml.cs` and is displayed nested below the `.axaml` file in the IDE.
- This is where we can handle events and write event handlers

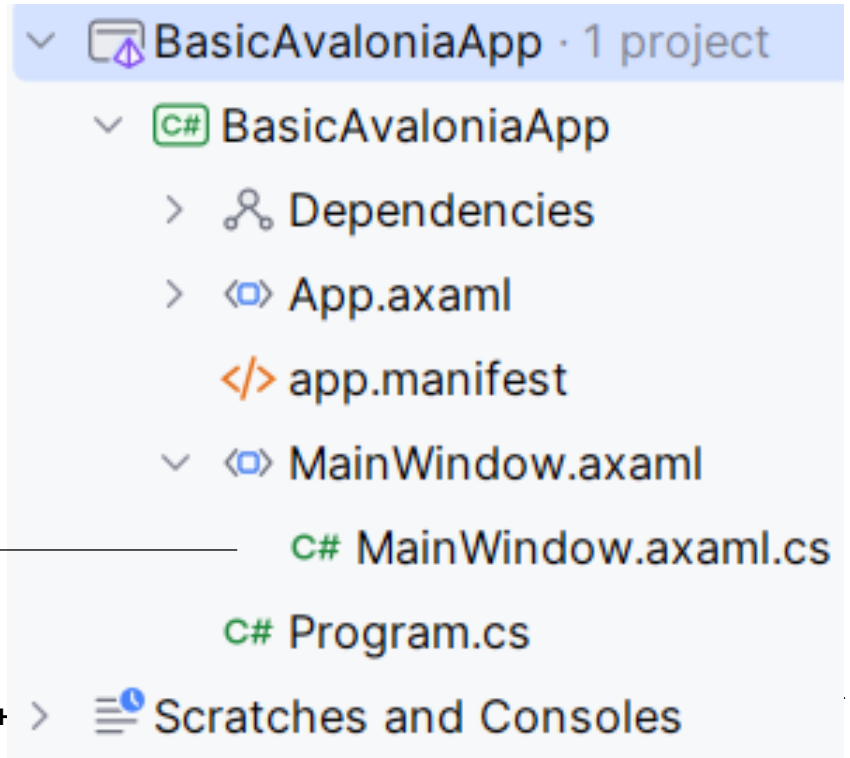
`MainWindow.axaml.cs`

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
    }

    public void ButtonClickHandler(object sender, RoutedEventArgs e)
    {
            // code here.
    }
}
```

object that triggered the event

more information about the event






https://docs.avaloniaui.net/docs/basics/user-interface/code-behind

# Event Handling

Event handling allows you to respond to user interactions, such as button clicks, mouse movements, or keyboard input.

- **Attaching event handlers:** You can attach event handlers to controls in XAML or code-behind.
- **Event handler signature:** Event handlers are methods with a specific signature,

  typically taking two parameters: object **sender** and RoutedEventArgs **e**.

- **Common events:** Click, MouseDown, MouseUp, MouseMove, KeyDown, KeyUp, TextChanged, etc.





https://docs.avaloniaui.net/docs/basics/user-interface/controls/builtin-controls

# Layouts

# Avalonia Layouts

- Avalonia has a lot of different layout options, that can be used to arrange **controls** within the window, i.e., stack, tab, grid, dock, wrap, canvas…..
- We will be looking at a subset of these layout controls, but feel free to experiment with all the available ones.

We will be using:

- ✓ StackPanel
- ✓ TabControl
- ✓ Grid







https://docs.avaloniaui.net/docs/basics/user-interface/controls/builtin-controls

# Element Positioning

- An Avalonia control exposes several properties that can be used to control the position of an element.
- The four most important are:
  - HorizontalAlignment
  - VerticalAlignment
  - Margin
  - Padding







https://docs.avaloniaui.net/docs/basics/user-interface/building-layouts/alignment-margins-and-padding

# Element Positioning

## HorizontalAlignment

- Specifies how an element should be aligned horizontally within its parent container.
- Possible Values: Left, Center, Right, Stretch.

## VerticalAlignment

- Specifies how an element should be aligned vertically within its parent container.
- Possible Values: Top, Bottom, Center, Stretch.

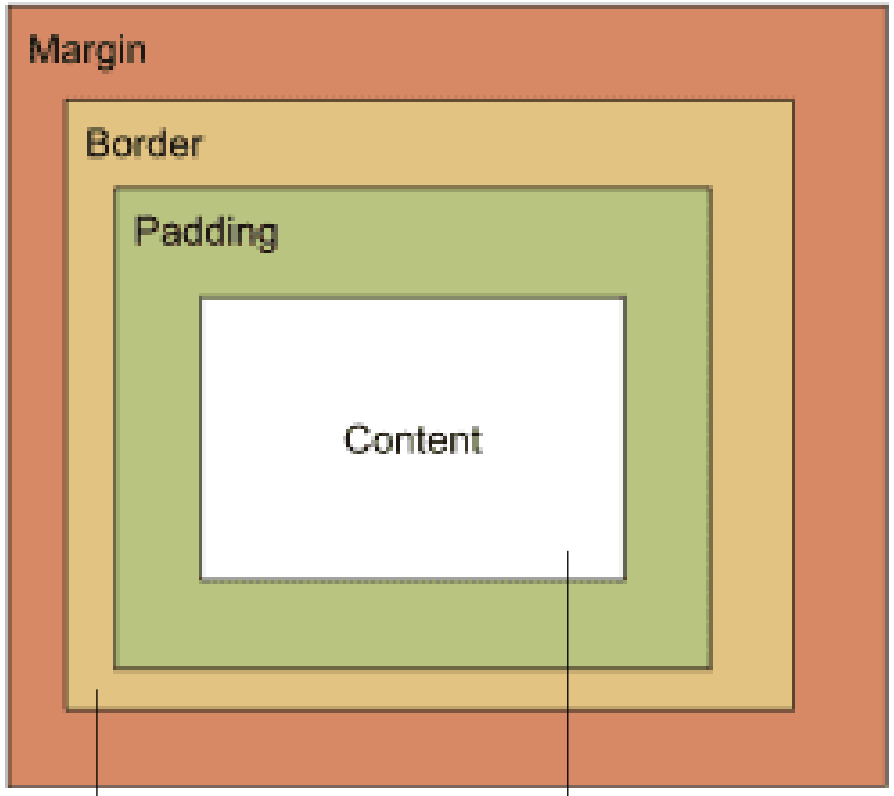
## Margin

- Purpose: Adds empty space around the outside of an element.
- Values: A single value for all sides (e.g., Margin="10"), or individual values for left, top, right, and bottom (e.g., Margin="5,10,15,20").

## Padding

- Purpose: Adds empty space around the inside of an element (between the element's content and its border).
- Values: A single value or individual values for each side.

# Example

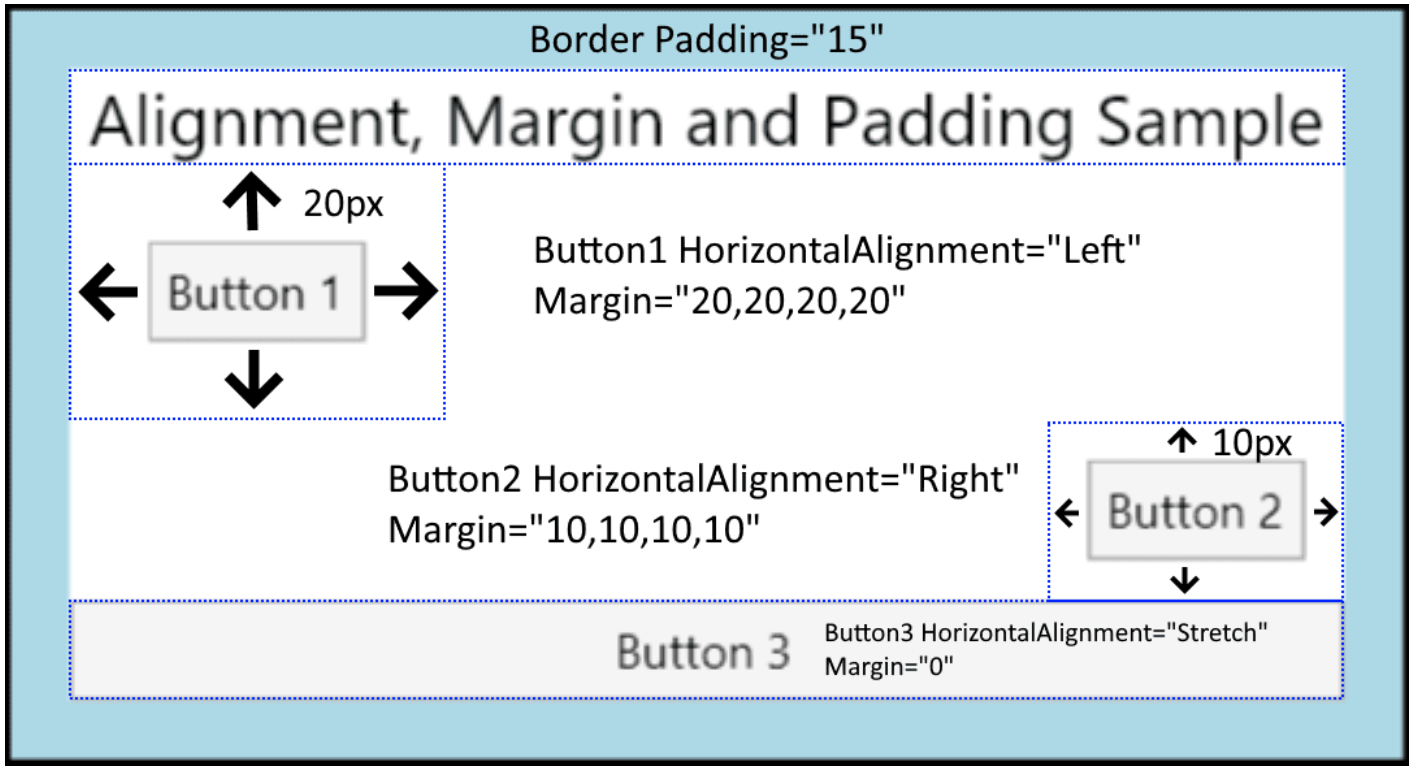



yellow part

white part

button border

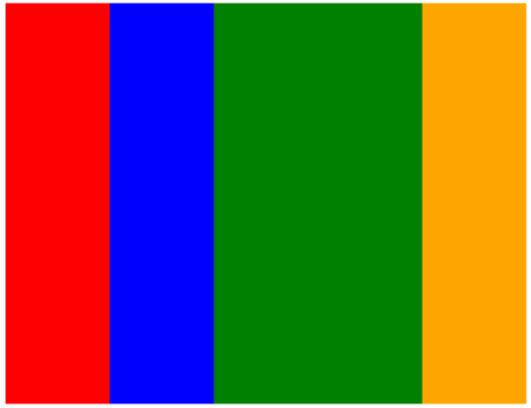
button content



# StackPanel

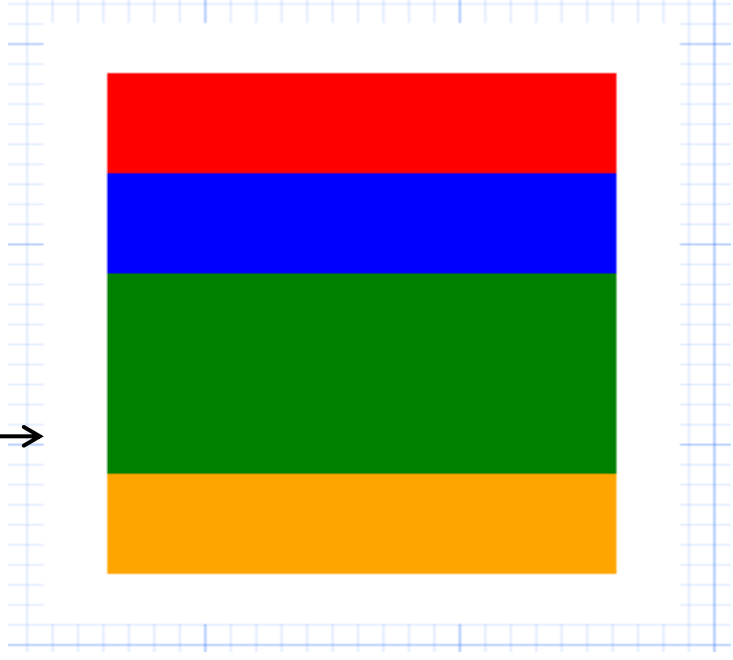
- A StackPanel allows you to stack controls and other elements (even other StackPanels!) horizontally or vertically.
- Used as a container to arrange a subsection of the UI.
- Useful Properties

| Property | Description |
|---|---|
| **Orientation** | Sets the direction of the stack. Choose from horizontal or vertical (default). |



```
<StackPanel Height="200" Orientation="Horizontal">
  <Rectangle Fill="Red" Width="50" />
  <Rectangle Fill="Blue" Width="50" />
  <Rectangle Fill="Green" Width="100" />
  <Rectangle Fill="Orange" Width="50"/>
</StackPanel>

<StackPanel  Width="200">
  <Rectangle Fill="Red" Height="50" />
  <Rectangle Fill="Blue" Height="50" />
  <Rectangle Fill="Green" Height="100" />
  <Rectangle Fill="Orange" Height="50"/>
</StackPanel>
```



https://docs.avaloniaui.net/docs/reference/controls/stackpanel

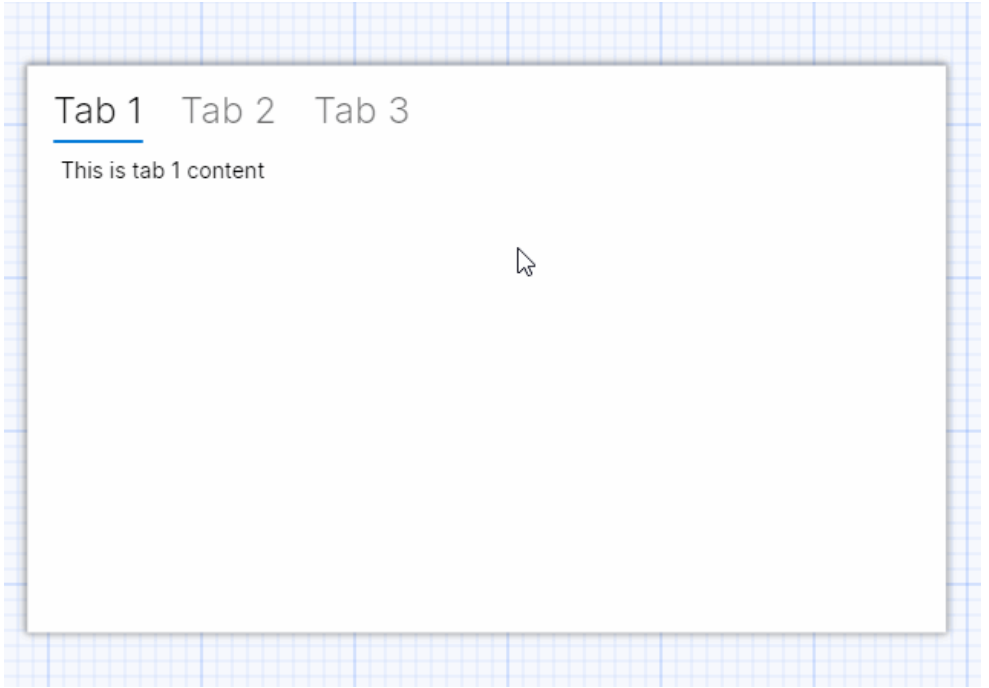
# TabControl

- The TabControl allows you to sub-divide a view into tab items.
- Each tab item has a header and a content zone.
  - When the user clicks a tab header, its content will become available.
    - Within this content you could for example put a StackPanel to define the rest of your UI!

**Useful Properties**

| Property | Description |
|---|---|
| `<TabItem> </TabItem>` | Creates a new tab |
| `Header` | The text for the tab |

```
<TabControl Margin="5">
            <TabItem Header="Tab 1">
                        <StackPanel>
                              <TextBlock Margin="5">This is tab 1 content</TextBlock>
                        </StackPanel>
            </TabItem>
            <TabItem Header="Tab 2">
                        <TextBlock Margin="5">This is tab 2 content</TextBlock>
            </TabItem>
            <TabItem Header="Tab 3">
                        <TextBlock Margin="5">This is tab 3 content</TextBlock>
            </TabItem>
</TabControl>
```



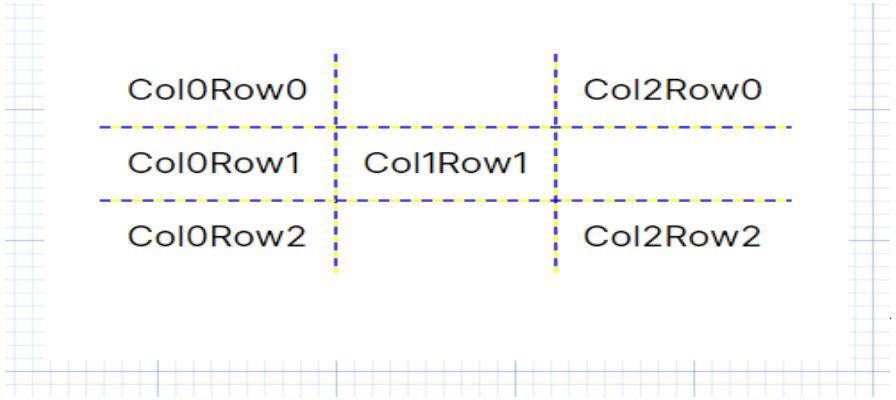

https://docs.avaloniaui.net/docs/reference/controls/tabcontrol

# Grid

- The Grid allows you to arrange elements within columns and rows.
- Each element can be positioned in the Grid using column and row coordinates.
- The coordinates are zero-based and have zero as a default.
- If you do not give any column and row coordinates, they will be drawn in the top left corner (column=0, row=0)
- Useful Properties

| Property | Description |
|---|---|
| **`ColumnDefinitions`** | Size definitions describing the widths of columns in the **`Grid`** |
| **`RowDefinitions`** | Size definitions describing the heights of rows in the **`Grid`** |
| **`Grid.Column`** | Puts the control element within the specified **`Column`** |
| **`Grid.Row`** | Puts the control element within the specified **`Row`** |
| **`ShowGridLines`** | Shows the gridlines between cells (Useful for debugging) |

```
<Grid ColumnDefinitions="Auto,Auto,Auto" RowDefinitions="Auto,Auto,Auto" ShowGridLines="True">
<TextBlock Text="Col0Row0" Grid.Row="0" Grid.Column="0"/>
 <TextBlock Text="Col0Row1" Grid.Row="1" Grid.Column="0"/>
 <TextBlock Text="Col0Row2" Grid.Row="2" Grid.Column="0"/>
 <TextBlock Text="Col2Row0" Grid.Row="0" Grid.Column="2"/>
<TextBlock Text="Col1Row1" Grid.Row="1" Grid.Column="1"/>
 <TextBlock Text="Col2Row2" Grid.Row="2" Grid.Column="2"/>
</Grid>
```






https://docs.avaloniaui.net/docs/reference/controls/grid/

# Controls

# Avalonia Controls

- Avalonia has a lot of different controls, that can be used to create a responsive UI.
- Some common controls include:
  - Button
  - Label
  - TextBox
  - Radio Button
  - ComboBox
  - Image

**Useful Properties**

| Property | Description |
|---|---|
| **Name** | Uniquely identifies the element and allows it to be controlled from the code |





https://docs.avaloniaui.net/docs/basics/user-interface/controls/builtin-controls

# Button

- A Button is used for triggering actions when clicked or tapped.
- Useful for tasks like submitting forms, opening dialogs, or executing commands.
- Useful Properties

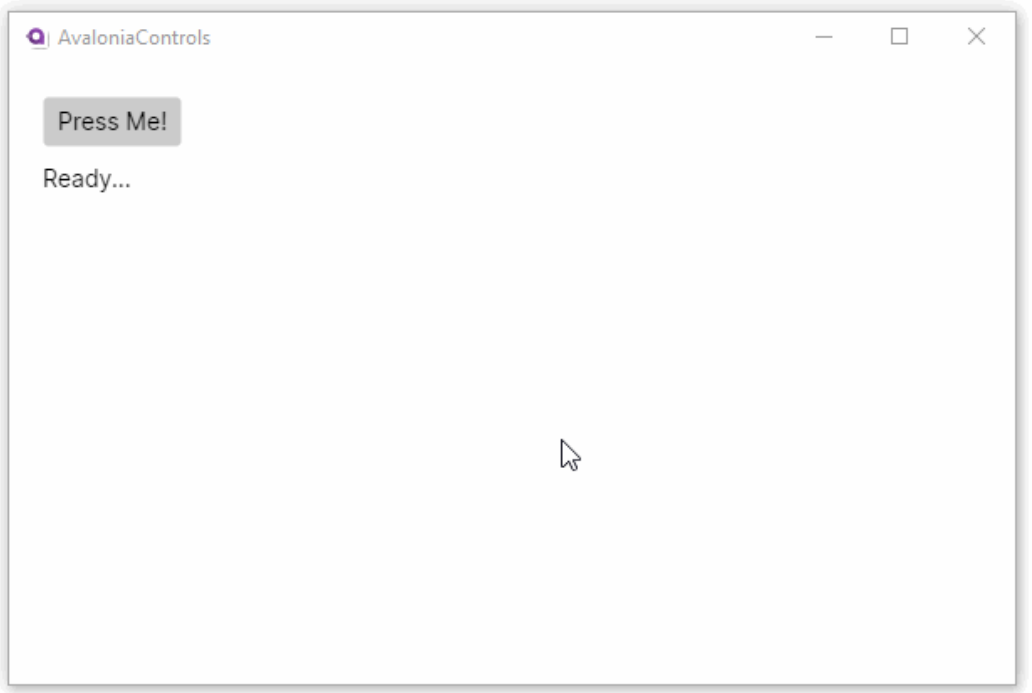
| Property | Description |
|---|---|
| `Click` | Called when the user clicks the button. Calls the handler specified. |

MainWindow.axaml

```
<StackPanel Margin="20">
   <Button Click="ClickHandler">Press Me!</Button>
   <Label Name="Message"> Ready...</Label>
</StackPanel>
```

MainWindow.axaml.cs

```
public void ClickHandler(object sender, RoutedEventArgs args)
{
   Message.Content = "Button clicked!";
}
```

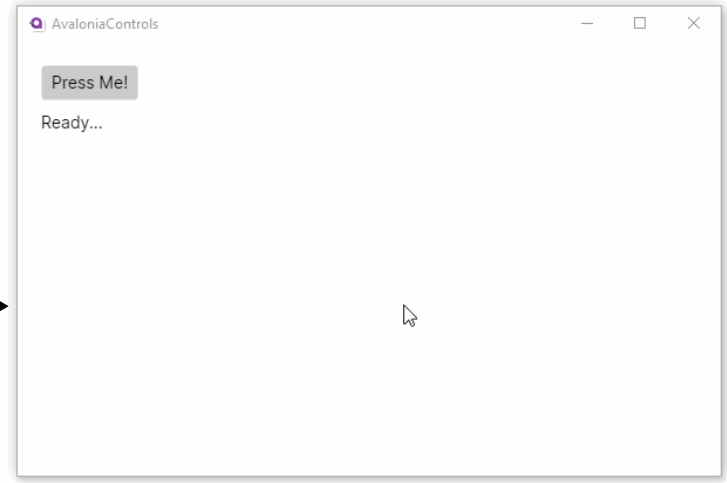





https://docs.avaloniaui.net/docs/reference/controls/buttons/button

MainWindow.axaml

```
<StackPanel Margin="20">
    <Button Click="ClickHandler">Press Me!</Button>
    <Label Name="Message"> Ready...</Label>
</StackPanel>
```




MainWindow.axaml.cs

```
public void ClickHandler(object sender, RoutedEventArgs args)
{
    var message =this.FindControl<Label>(“Message");
    message.Content = "Button clicked!";
}
```

MainWindow.axaml.cs

```
public void ClickHandler(object sender, RoutedEventArgs args)
{
    Message.Content = "Button clicked!";
}
```

- ✓ If you are **dynamically** creating controls at runtime, then use findControl method, otherwise
- ✓ **Direct access** is generally preferred for its simplicity and efficiency.



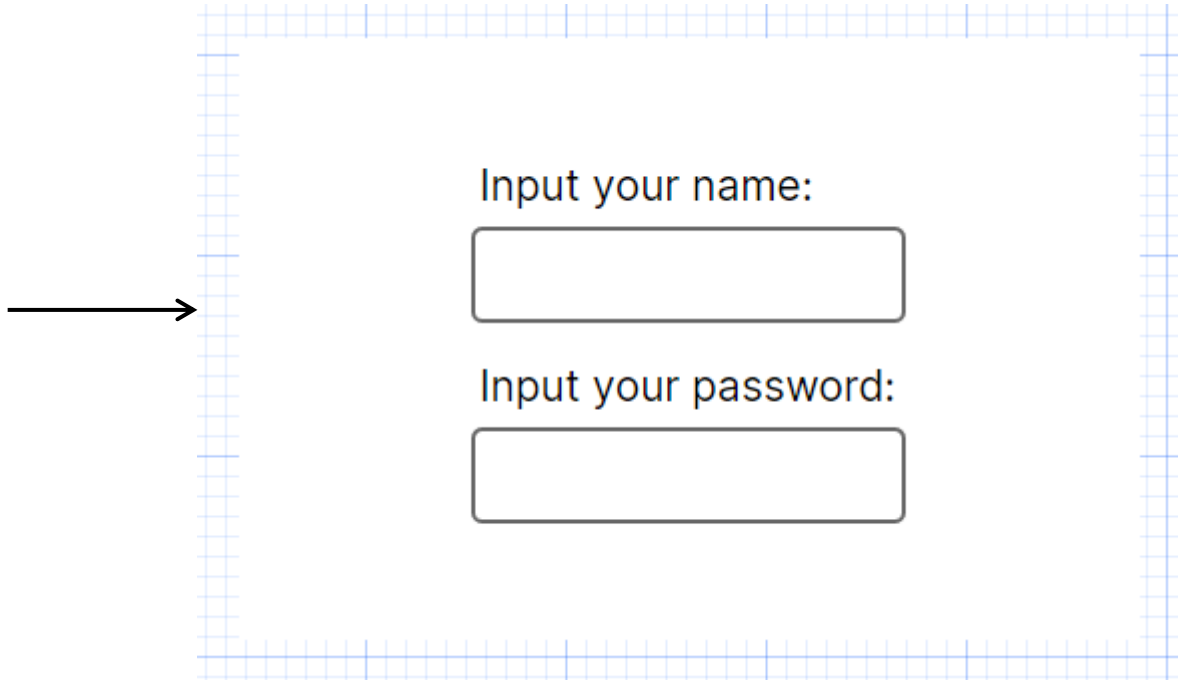
https://docs.avaloniaui.net/docs/reference/controls/buttons/button

# Label

- A Label is used to display static text or captions for other controls.
- They provide additional information or context to the user and are generally non-interactive.
- Useful Properties

| Property | Description |
|---|---|
| **`Content`** | Allows you to specify the content (Text) within the label. |

```
<StackPanel Margin="20">
   <Label Content="Input your name:"/>
   <TextBox/>
   <Label Content="Input your password:"/>
   <TextBox/>
</StackPanel>
```






https://docs.avaloniaui.net/docs/reference/controls/label

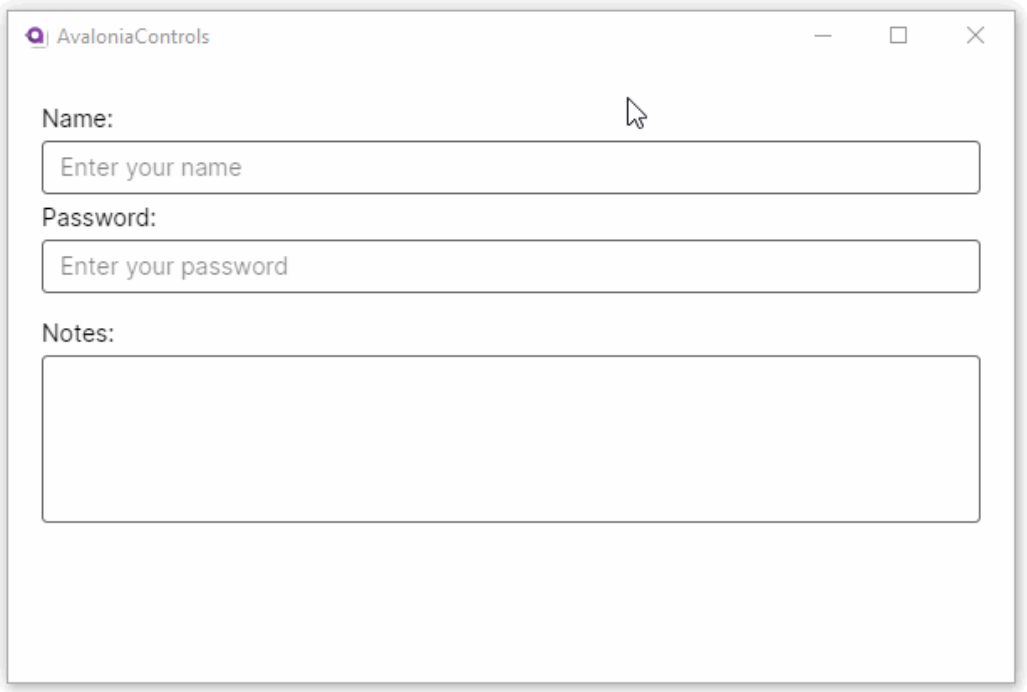
# TextBox

- A Text box allows users to enter and edit text.
- They are used for capturing user input.
- Useful properties →

| Property | Description |
|---|---|
| `Content` | Allows you to grab the text from a user input within a TextBox. |
| `xml:space="preserve"` | Allows you to keep whitespace, such as "\n" |
| `AcceptsReturn` | Accept line returns |
| `TextWrapping` | Wraps text in case of overflow |

```
<StackPanel Margin="20">
   <TextBlock Margin="0 5" >Name:</TextBlock>
   <TextBox  Watermark="Enter your name"/>
   <TextBlock Margin="0 5" >Password:</TextBlock>
   <TextBox PasswordChar="*" Watermark="Enter your password"/>
   <TextBlock Margin="0 15 0 5">Notes:</TextBlock>
   <TextBox Height="100" AcceptsReturn="True" TextWrapping="Wrap"/>
</StackPanel>
```

→



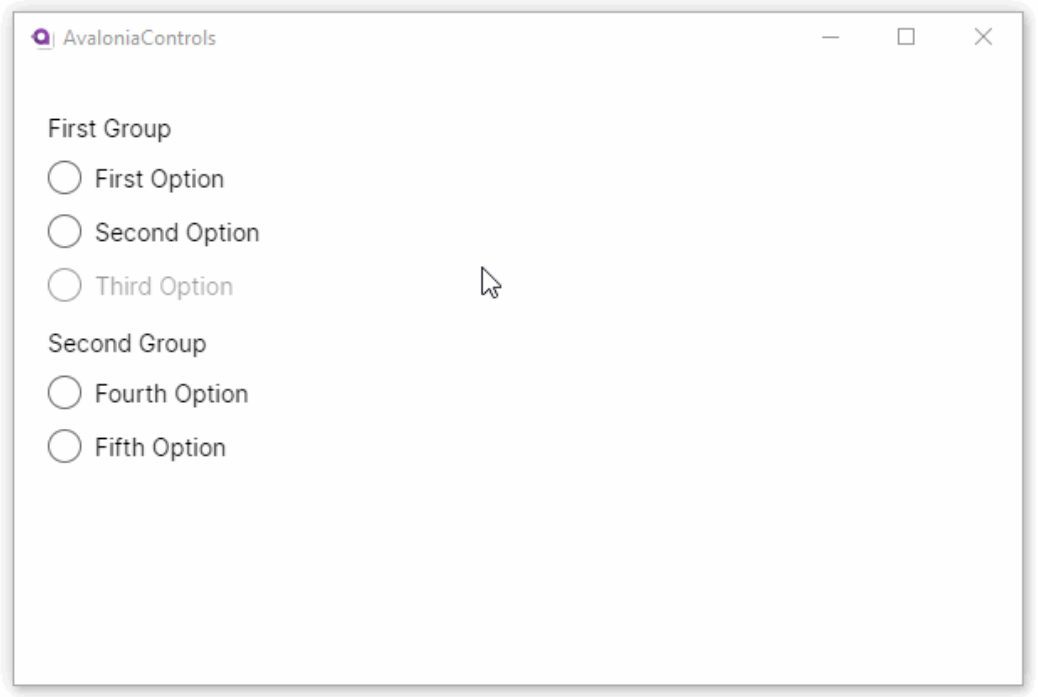



https://docs.avaloniaui.net/docs/reference/controls/textbox

# Radio Button

- Radio Buttons are used for selection and multiple-choice options.
- Radio buttons enable users to choose a single option from a group.
- Useful Properties

| Property | Description |
|---|---|
| `Content` | Allows you to grab the content within a Radio Button. |
| `IsChecked` | Whether a radio button option is selected (true) or unselected (false). |
| `IsEnabled` | Whether a radio button option is enabled. Disabled options are presented faded. |
| `GroupName` | Defines the name of the group of options, that will interact as radio buttons. |




```
<StackPanel Margin="20">
  <TextBlock Margin="0 10 0 5">First Group</TextBlock>
  <RadioButton GroupName="First Group" Content="First Option"/>
  <RadioButton GroupName="First Group" Content="Second Option"/>
  <RadioButton IsEnabled="False" GroupName="First Group" Content="Third Option"/>

  <TextBlock Margin="0 10 0 5">Second Group</TextBlock>
  <RadioButton GroupName="Second Group" Content="Fourth Option"/>
  <RadioButton GroupName="Second Group" Content="Fifth Option"/>
</StackPanel>
```

https://docs.avaloniaui.net/docs/reference/controls/buttons/radiobutton

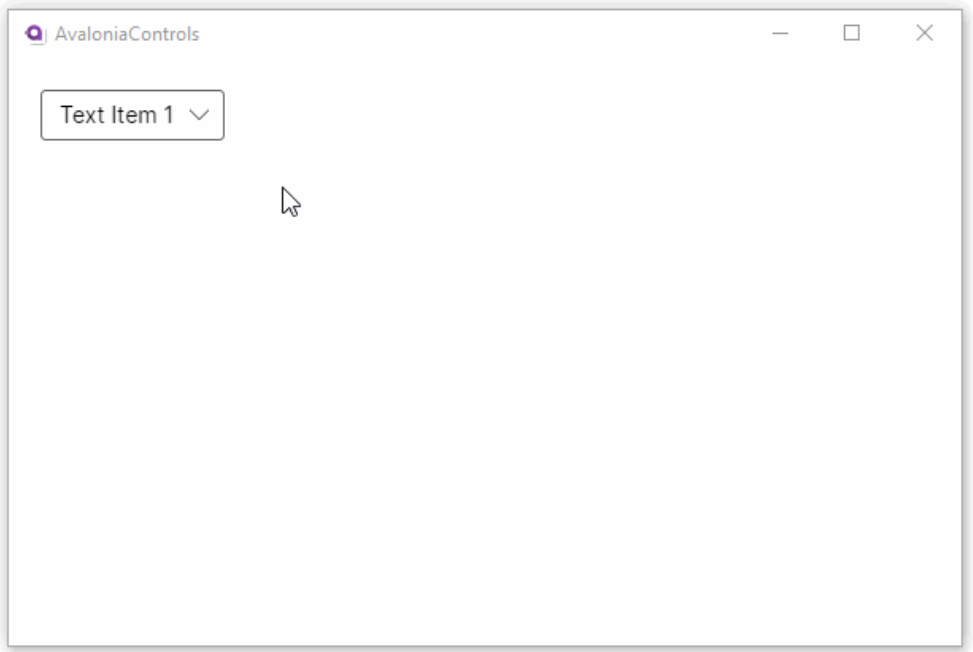
# ComboBox

- Combo Boxes allow users to select an item from a list or a dropdown menu.
- ComboBoxes show a single item initially and expand to display a list when clicked.
- Useful properties →

| Property | Description |
|---|---|
| `SelectedItem` | Gets the selected item itself. |
| `Items` | All items in the ComboBox as a collection. |

```
<StackPanel Margin="20">
  <ComboBox SelectedIndex="0" MaxDropDownHeight="100">
    <ComboBoxItem>Text Item 1</ComboBoxItem>
    <ComboBoxItem>Text Item 2</ComboBoxItem>
    <ComboBoxItem>Text Item 3</ComboBoxItem>
    <ComboBoxItem>Text Item 4</ComboBoxItem>
    <ComboBoxItem>Text Item 5</ComboBoxItem>
    <ComboBoxItem>Text Item 6</ComboBoxItem>
    <ComboBoxItem>Text Item 7</ComboBoxItem>
    <ComboBoxItem>Text Item 8</ComboBoxItem>
    <ComboBoxItem>Text Item 9</ComboBoxItem>
  </ComboBox>
</StackPanel>
```

→




https://docs.avaloniaui.net/docs/reference/controls/combobox

# Image

- Displays images from a specified image source. The source can be:
  - A string constant pointing to an asset resource
  - Loaded as a bitmap through an asset
- Images can be resized and scaled using **`Height`** and **`Width`** properties.


| Property | Description |
|---|---|
| **`Source`** | The image itself. Updating an image's source, will update the image displayed |

```
<Panel>
  <Rectangle Height="300" Width="300" Fill="LightGray"/>
  <Image Margin="20" Height="200" Width="200"
      Source="avares://AvaloniaControls/Assets/pipes.jpg"/>
</Panel>
```

URI Scheme — Name of application — Path to Image



https://docs.avaloniaui.net/docs/reference/controls/image

# Image

- Include the <AvaloniaResource> element in your project file, especially if you have multiple assets in the "Assets" folder.

```
<ItemGroup>
  <AvaloniaResource Include="Assets\**" />
</ItemGroup>
```

**OR**

- Ensure that the "Build Action" for your image files in the "Assets" folder is set to “**AvaloniaResource**" in the Solution Explorer. This is crucial for embedding the images into your application.
- Make sure the "Assets" folder is at the root level of your project, as shown in the screenshot.



https://docs.avaloniaui.net/docs/reference/controls/image

# File Dialogs

File Dialogs are used to prompt the user to open a file or to save a file.

```
private async void OpenFileButton_Clicked(object sender, RoutedEventArgs args)
{
    // Get top level from the current control.
    var topLevel = GetTopLevel(this);

    // Start async operation to open the dialog.
    var files = await topLevel.StorageProvider.OpenFilePickerAsync(new FilePickerOpenOptions
    {
        Title = "Open Text File",
        AllowMultiple = false
    });

    if (files.Count >= 1)
    {
        // Open reading stream from the first file.
        await using var stream = await files[0].OpenReadAsync();
        using var streamReader = new StreamReader(stream);
        // Reads all the content of file as a text.
        var fileContent = await streamReader.ReadToEndAsync();
        Console.WriteLine(fileContent);
    }
}
```

https://docs.avaloniaui.net/docs/basics/user-interface/file-dialogs

Opening a file picker dialog.
Uses the **`OpenFilePickerAsync`** method


Will be covered in upcoming lectures

# File Dialogs

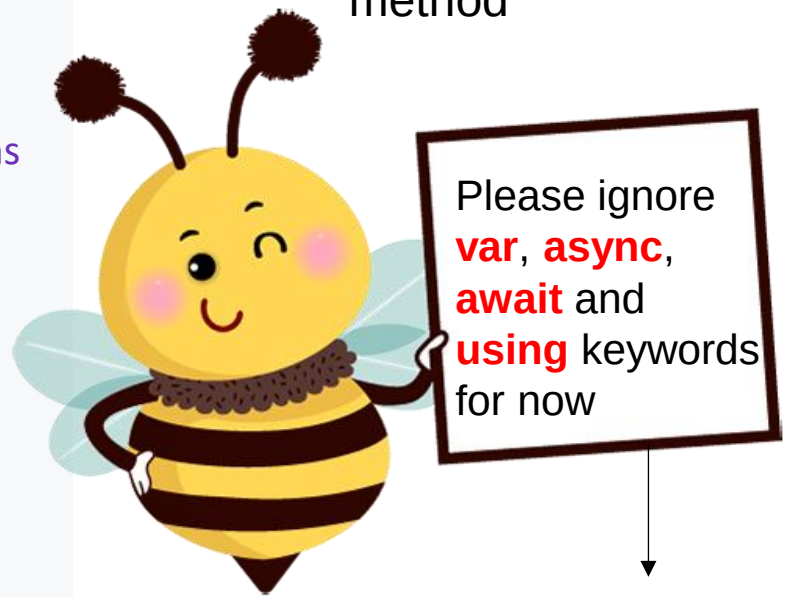
File Dialogs are used to prompt the user to open a file or to save a file.

```
private async void SaveFileButton_Clicked(object sender, RoutedEventArgs args)
{
    // Get top level from the current control.
    var topLevel = GetTopLevel(this);

    // Start async operation to open the dialog.
    var file = await topLevel.StorageProvider.SaveFilePickerAsync(new FilePickerSaveOptions
    {
        Title = "Save Text File"
    });

    if (file is not null)
    {
        // Open writing stream from the file.
        await using var stream = await file.OpenWriteAsync();
        using var streamWriter = new StreamWriter(stream);
        // Write some content to the file.
        await streamWriter.WriteLineAsync("Hello World!");
    }
}
```

Opening a file save dialog.
Uses the **`SaveFilePickerAsync`** method

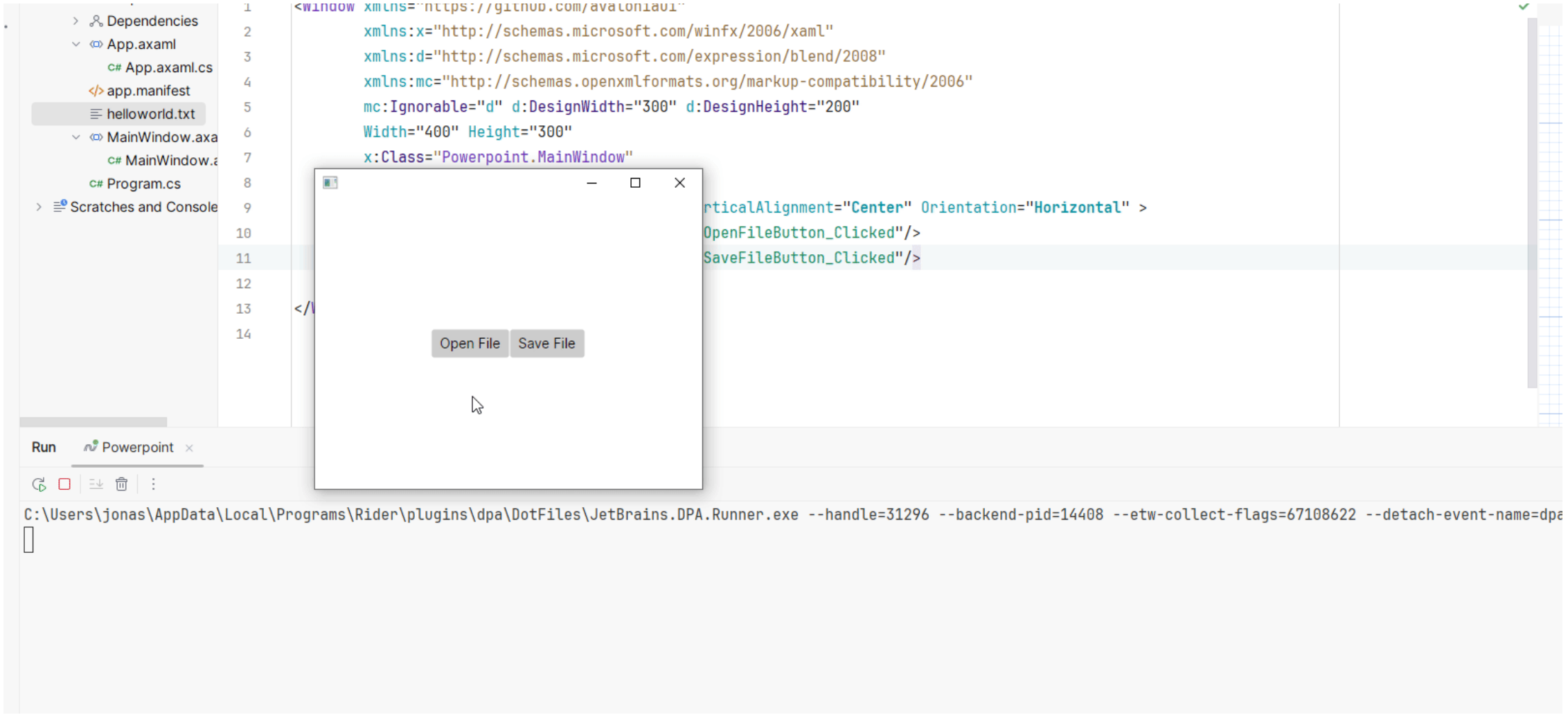



Will be covered in upcoming lectures



https://docs.avaloniaui.net/docs/basics/user-interface/file-dialogs

# File Dialogs Example




https://docs.avaloniaui.net/docs/basics/user-interface/file-dialogs

# Video Tutorial

https://www.youtube.com/watch?v=cbkS2RdHldA

**By Jonas Solhaug Kaad (Student Instructor)**

SDU

# MCQs Quiz

Go to Plans -> VOP-2 -> VOP-2 (Lecture) -> Avalonia Test

Good Luck ☺

# Avalonia Application Templates

Back

## Avalonia .NET App

- Creates a basic Avalonia application with a minimal structure.
- Includes the core Avalonia libraries.
- Suitable for simple applications.

## Avalonia Cross-Platform Application

- Creates applications that can run seamlessly across different platforms (Windows, macOS, Linux).
- Includes platform-specific project configurations to handle platform differences.

## Avalonia .NET MVVM Application

- Creates Avalonia application using the MVVM (Model-View-ViewModel) architectural pattern.
- Provides organized project structure with pre defined folders and classes for Models, Views, and ViewModels.
- Suitable for larger applications where separating UI logic from business logic is beneficial.



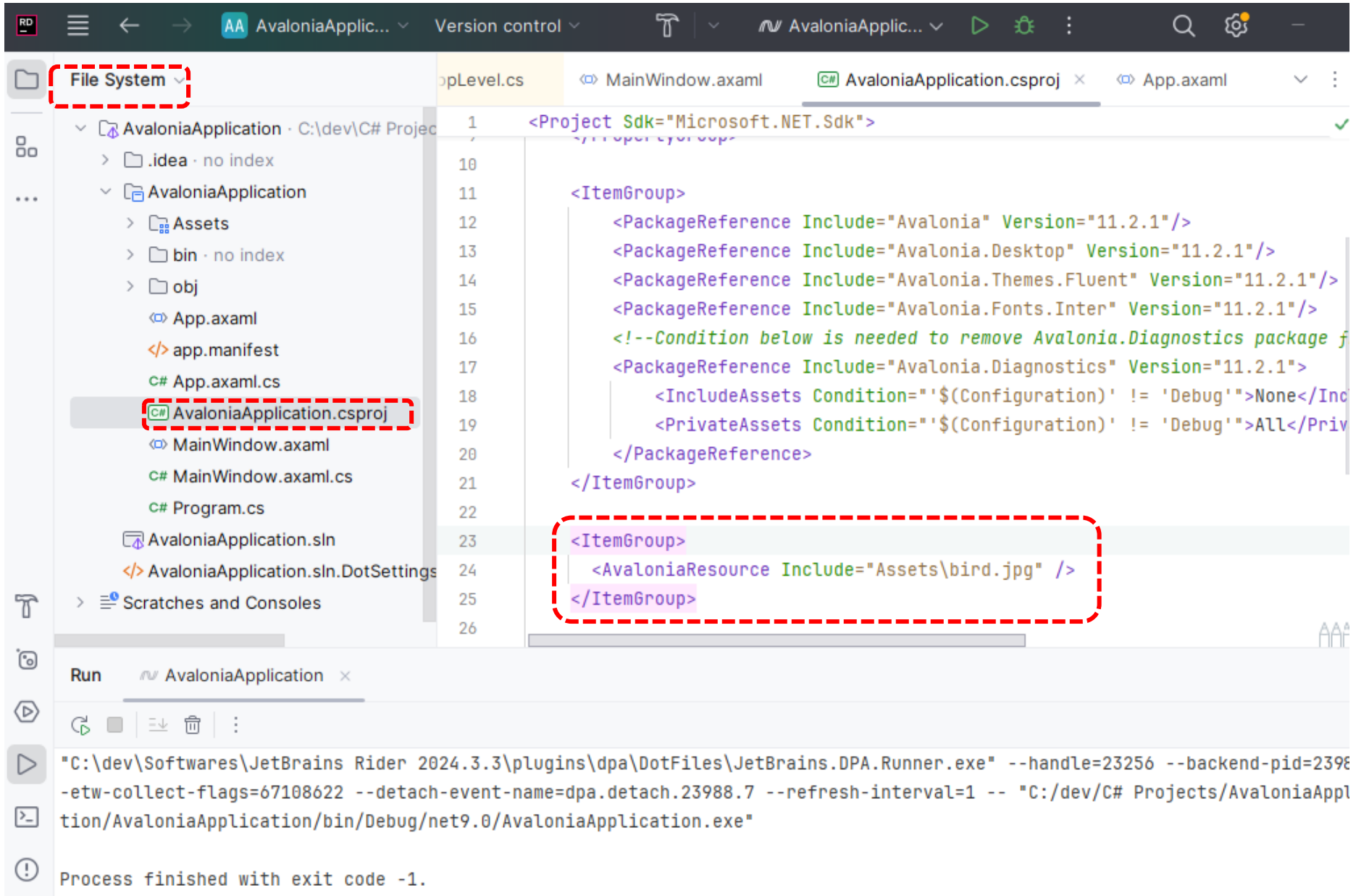




https://docs.avaloniaui.net/docs/basics/user-interface/introduction-to-xaml

# Locating Project File

Back

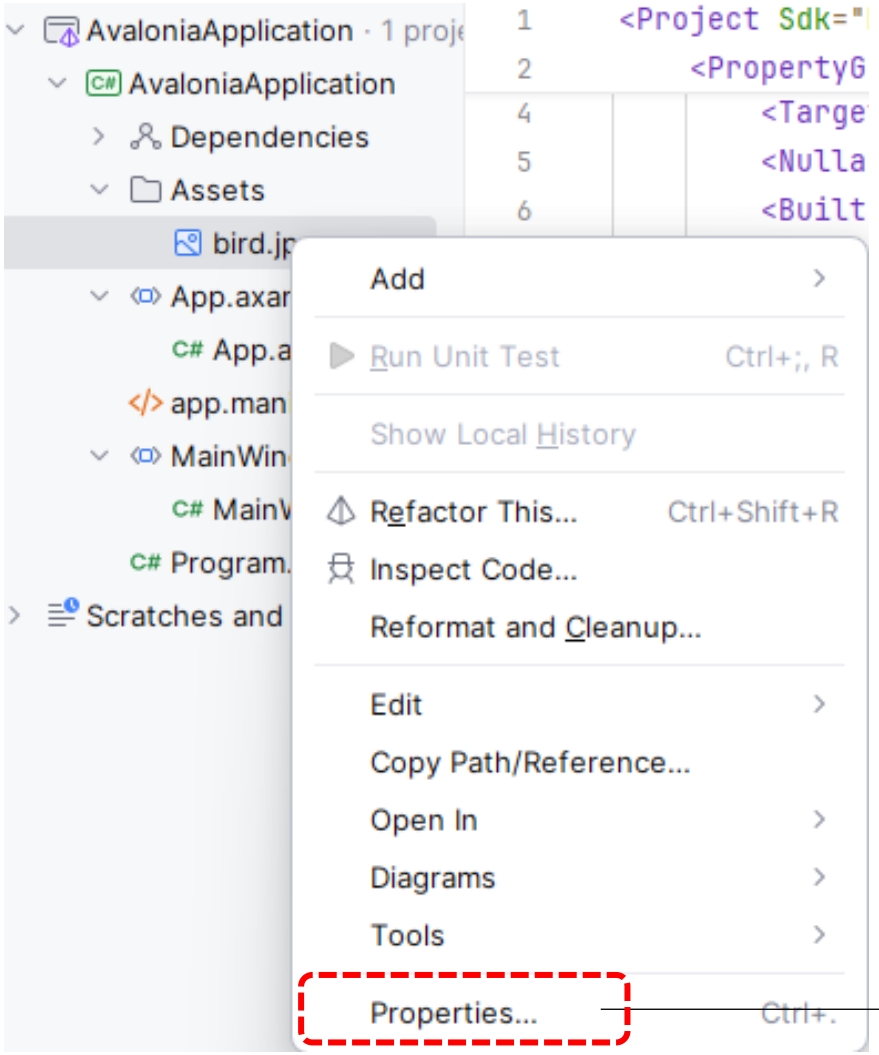


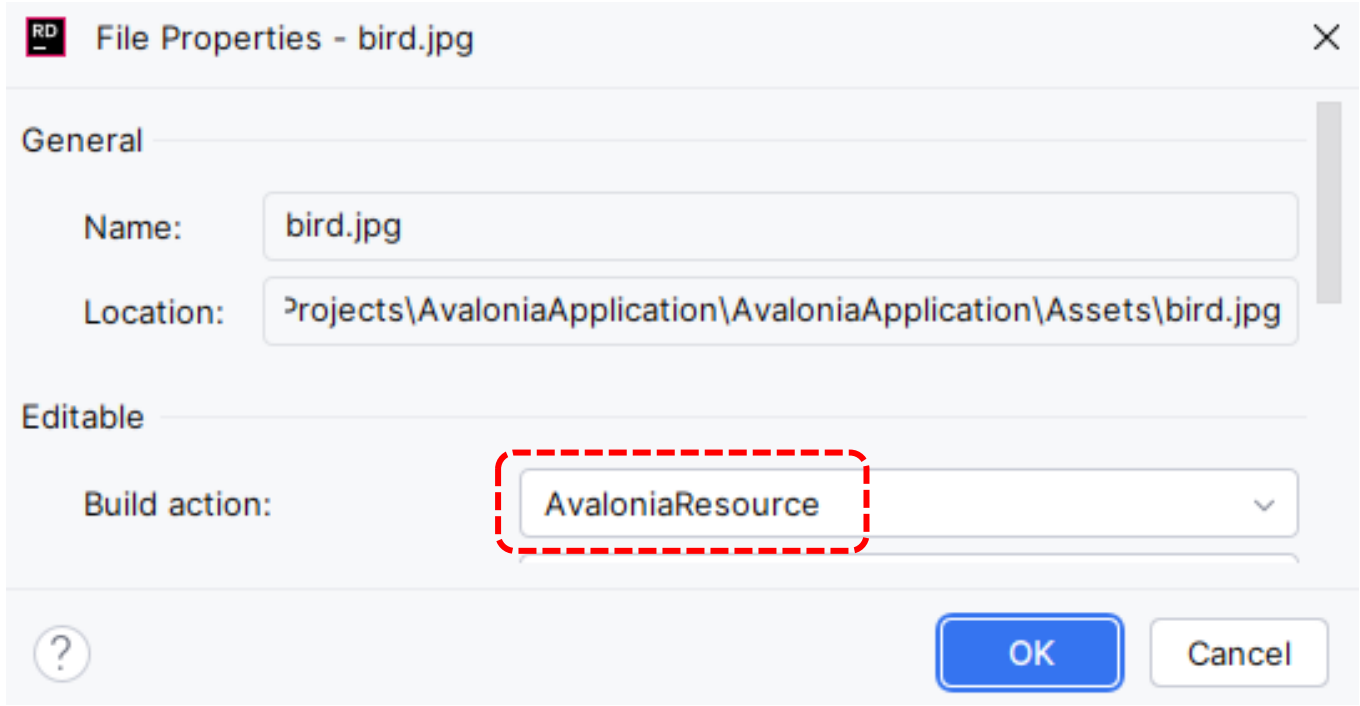



https://docs.avaloniaui.net/docs/reference/controls/image

# Build Action: Avalonia Resource

Back






https://docs.avaloniaui.net/docs/reference/controls/image