CS6004NI
Application Development

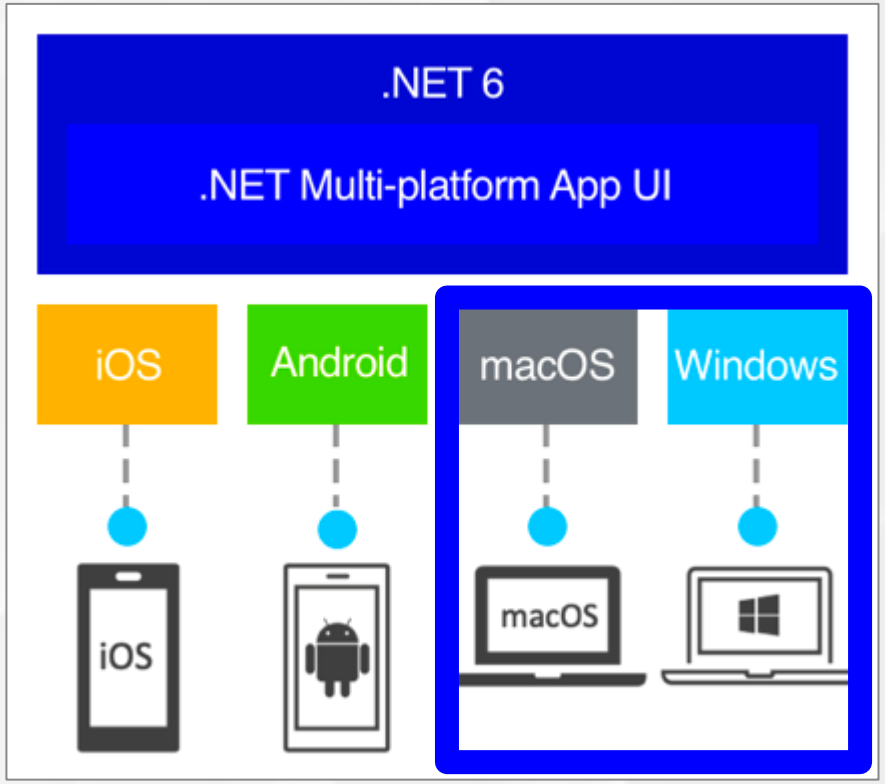
samyush

Samyush.maharjan@islingtoncollege.edu.np

.NET MAUI

.NET MAUI is a cross-platform framework for creating **native mobile and desktop apps**. MAUI can target Android, iOS, Windows, and MacOS.

- Build cross-platform apps in C# and XAML
- A single shared code-base
- Share UI layout and design across platforms
- Share code, tests, and business logic across platforms



.NET MAUI Blazor

.NET MAUI Blazor allows us to build hybrid cross-platform apps using **Web technologies** (HTML, CSS, and optionally JS).

Using .NET CLI:

1. Install **.NET MAUI workload**

```
dotnet workload install maui
```

1. Create a new project using **.NET MAUI Blazor App** template

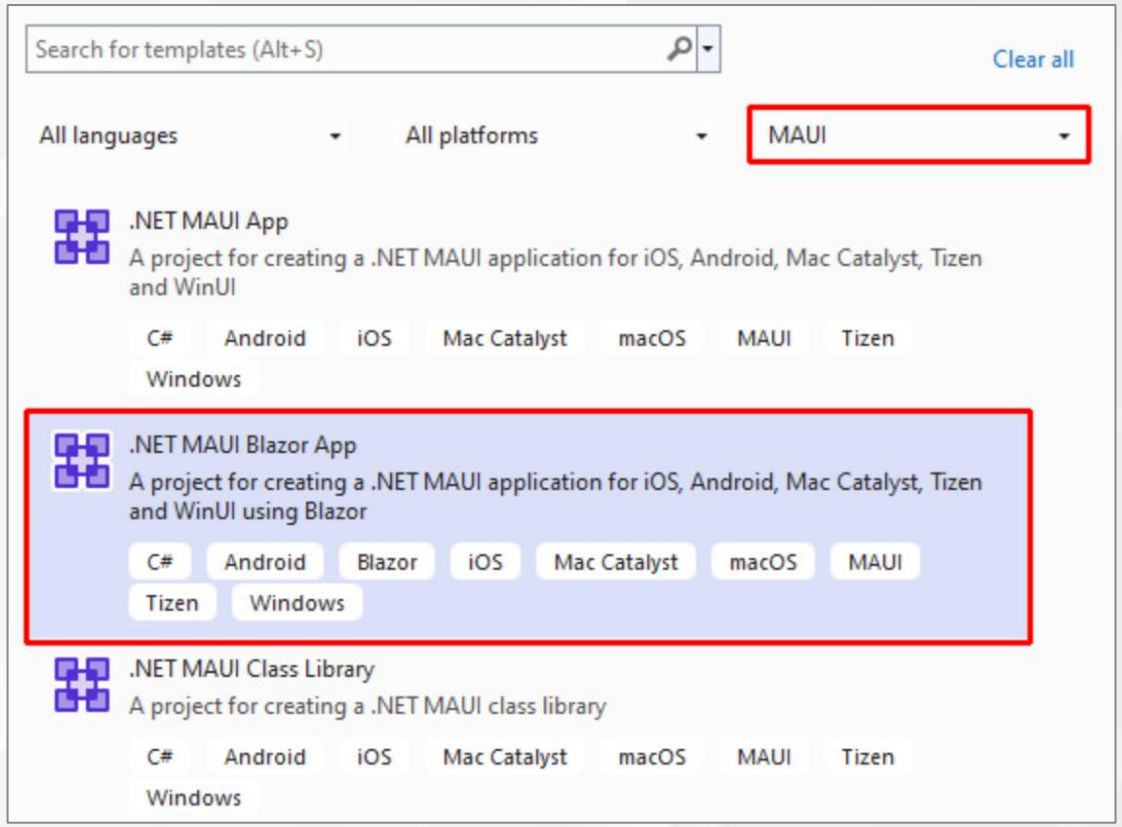
```
dotnet new maui-blazor --name ProjectName
```

1. Run the project

```
dotnet run -f net6.0-maccatalyst
```

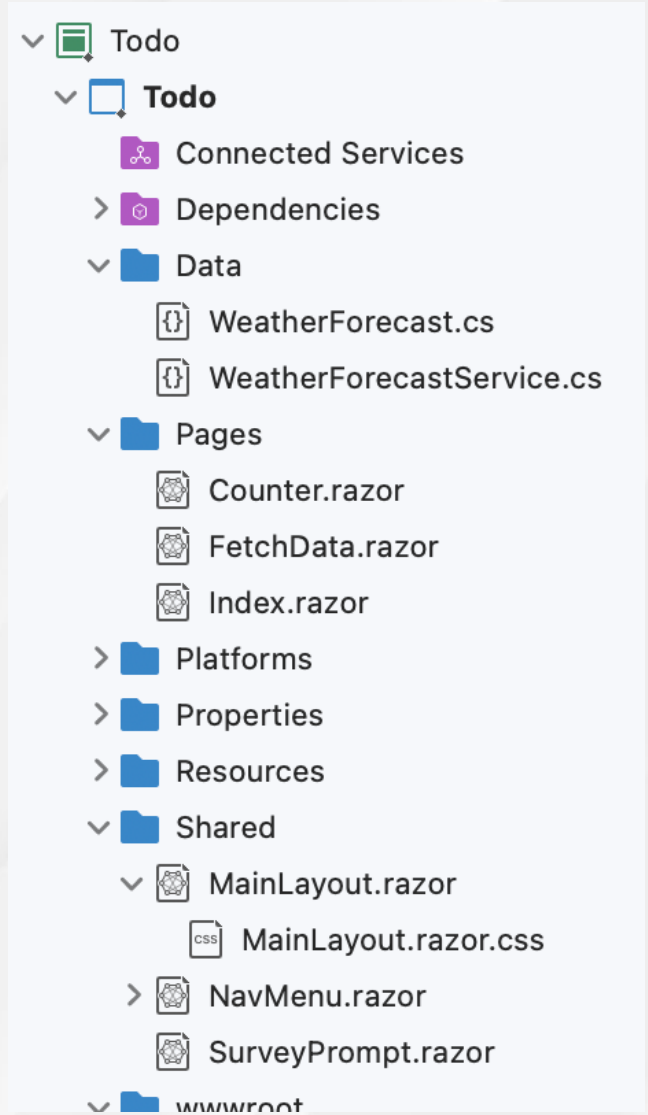
Using Visual Studio:

1. Install Visual Studio 2022, or modify your existing installation to install the **.NET MAUI workload**
2. Create a new project using **.NET MAUI Blazor App** template



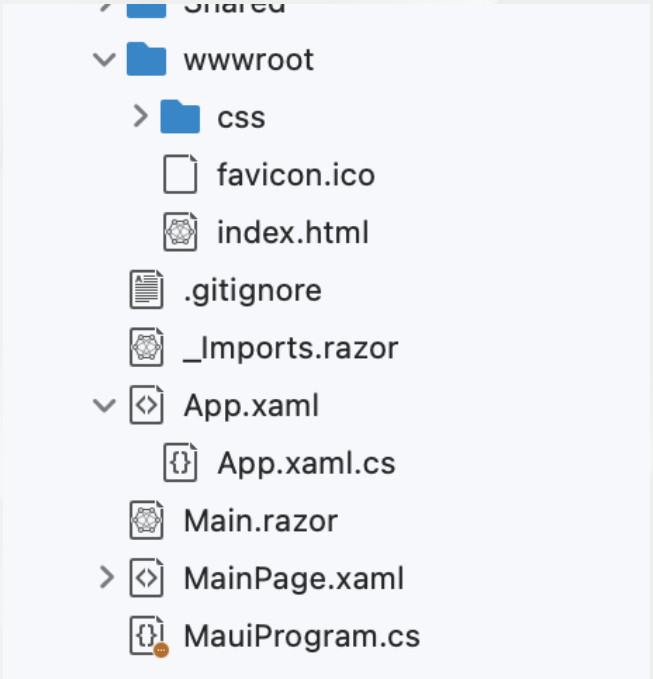
.NET MAUI Blazor Project

- **Data**: It contains **services** that manages app data.
- **Pages**: It contains the routable components/pages (.razor file). The route for each page is specified using the **@page directive** and contains the HTML for UI also may contain **@code** directive for C# code.
- **Platforms**: It contains a folder for each **platform supported by the MAUI** app which contains **platform specific config** (manifest, plist) and entrypoint.
- **Resources**: It contains **App Icons, Fonts, Images, Splash, Style** XAMLs**, Raw** asset files like txt or json for initial app data.
- **Shared**: It contains **shared Razor components and stylesheets**.



.NET MAUI Blazor Project

- **wwwroot**: The Web Root folder for the app and contains public **static assets** like index.html, css, images, fonts.
- **_Imports.razor**: Global **using statements** (**@using** directive) for commonly used namespaces for Razor components (.razor).
- **App.xaml**: The resources like Colors, and Styles declared here for **native (XAML) app**.
- **Main.razor**: The root component of the app that sets up app **routes for pages** and **not found page**.
- **MainPage.xaml**: Renders Blazor web app using **BlazorWebView** control.
- **MauiProgram.cs**: Project entry point where MAUI App services are initialized and configured, similar to Program.cs.



.NET MAUI Blazor Project

1. - ● **MauiProgram** (MauiProgram.cs) => **App** (App.xamal.cs)
2. ------- ○ **MainPage** *(MainPage.xaml)*
3. ---------- ■ **BlazorWebView** => **index.html** *(wwwroot/index.html)*, **Main** *(Main.razor)*
4. ---------- ● **Router**
5. ---------- ○ **RouteView / LayoutView** => **MainLayout** *(Shared/MainLayout.razor)*
6. ---------- ■ **NavMenu** *(Shared/NavMenu.razor)*
7. ---------- ● **NavLink** *(Built-in razor component)*
8. ---------- ○ **/** *(Pages/Index.razor)*
9. ---------- ■ **SurveyPrompt** (shared/SurveryPrompt.razor)
10. ---------- ○ **/counter** *(Pages/Counter.razor)*
11. ---------- ○ **/fetchdata** *(Pages/FetchData.razor)*
12. ---------- ■ **WeatherForecastService**.GetForecastAsync *(Data/WeatherForecastService.cs)*
13. ---------- ● **WeatherForecast** *(Data/WeatherForecast.cs)*

MAUI MainPage

Mainpage.xamal

```
1. <?xml version="1.0" encoding="utf-8" ?>
2. <ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
3.              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
4.              xmlns:local="clr-namespace:Todo"
5.              x:Class="Todo.MainPage"
6.              BackgroundColor="{DynamicResource PageBackgroundColor}">
7.     <BlazorWebView HostPage="wwwroot/index.html">
8.         <BlazorWebView.RootComponents>
9.             <RootComponent Selector="#app" ComponentType="{x:Type local:Main}" />
10.        </BlazorWebView.RootComponents>
11.    </BlazorWebView>
12.</ContentPage>
```

The root page of the Blazor web app.

The CSS selector to specify where the component should be placed.

The type of the root component.

BlazorWebView HostPage

index.html

```
1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4.   ...
5.   <link rel="stylesheet" href="css/bootstrap/bootstrap.min.css" />
6.   <link href="css/app.css" rel="stylesheet" />
7. </head>
8. <body>
9.   <div class="status-bar-safe-area"></div>
10.  <div id="app">Loading...</div>
11.  <div id="blazor-error-ui">
12.    ...
13.  </div>
14.  <script src="_framework/blazor.webview.js" autostart="false"></script>
15.</body>
16.</html>
```

Blazor Main Component

Main.razor

```
1. <Router AppAssembly="@typeof(Main).Assembly">
2.   <Found Context="routeData">
3.     <RouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)" />
4.     <FocusOnNavigate RouteData="@routeData" Selector="h1" />
5.   </Found>
6.   <NotFound>
7.     <LayoutView Layout="@typeof(MainLayout)">
8.       <p role="alert">Sorry, there's nothing at this address.</p>
9.     </LayoutView>
10.  </NotFound>
11.</Router>
```

Render the page using route data with default layout.

Render when route or content not found.

Blazor Layout Component

MainLayout.razor

```
1. @inherits LayoutComponentBase


2. <div class="page">
3.   <div class="sidebar">
4.     <NavMenu />
5.   </div>
6.   <main>
7.     <div class="top-row px-4">
8.       <a href="https://docs.microsoft.com/aspnet/" target="_blank">About</a>
9.     </div>
10.    <article class="content px-4">
11.      @Body
12.    </article>
13.  </main>
14.</div>
```

This is where current page component is rendered.

Blazor Page Component

Counter.razor

```
1. @page "/counter"

2. <h1>Counter</h1>
3. <p role="status">Current count: @currentCount</p>
4. <button class="btn btn-primary" @onclick="IncrementCount">Click me</button>

5. @code {
6.     private int currentCount = 0;

7.     private void IncrementCount()
8.     {
9.         currentCount++;
10.    }
11.}
```

/counter route is defined the for this **razor component** using **@page directive**.

One-way data binding.

Button Event listener.

C# code block

Blazor Component

```
<div class="alert alert-secondary mt-4">
    <span class="oi oi-pencil me-2" aria-hidden="true"></span>
    <strong>@Title</strong>

    <span class="text-nowrap">
        ...
    </span>
    and tell us what you think.
</div>

@code {
    // Demonstrates how a parent component can supply parameters
    [Parameter]
    public string Title { get; set; }
}
```

Example of using the component:

```
<SurveyPrompt Title="Blazor App" />
```

Blazor Data Binding

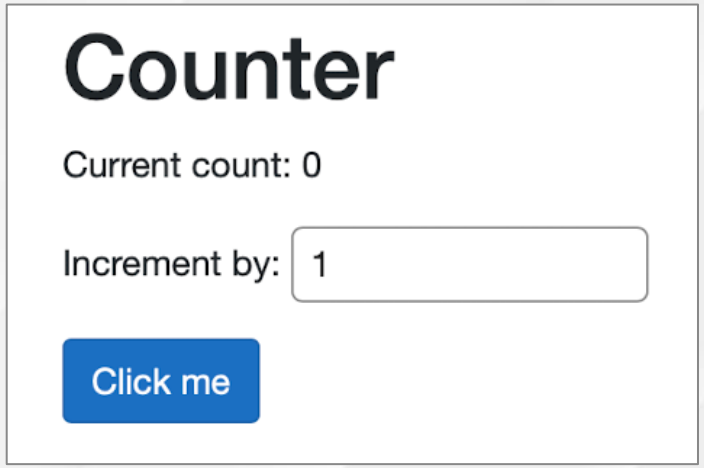
Data Binding is the connection bridge between View and the business logic (View Model) of the application.

- **One-Way Data Binding**: The data flows from the component to the DOM/UI or vice versa, but only in one direction.
- **Two-Way Data Binding**: The data flows in both direction between component and DOM/UI.
- Event Handling **and Data Binding:** This is also one-way data binding, but upon DOM events (like click event).

Example:

```
1.  @page "/counter"

2.  <h1>Counter</h1>
3.  <p>Current count: @currentCount</p>
4.  <p>Increment by: <input type="number" @bind="@increment" /></p>
5.  <button class="btn btn-primary" @onclick="IncrementCount">Click me</button>
6.  @code {
7.      private int currentCount = 0;
8.      private int increment = 1;
9.      private void IncrementCount()
10.     {
11.         currentCount += increment;
12.     }
13. }
```



Blazor Directives

Directictives are built-in macros that alter the transpiled C# code that is generated from Razor mark-up. A directiv is represented by implicit expressions with **reserved keywords** following the **@** symbol.

Below are some commonly used other directives:

@if:

Example:

```
1. @if (currentCount % 2 == 0)
2. {
3.     <p>Current count is an even number.</p>
4. }
5. else
6. {
7.     <p>Current count is an odd number.</p>
8. }
```

Blazor Directives

@for / @foreach:

Example:

```
@foreach (var todo in TodoList)
{
    <div>
        <label>
            <input type="checkbox" checked="@todo.IsDone" /> @todo.TaskName |
            <small>@todo.DueDate.ToShortDateString()</small>
        </label>
    </div>
}
```

Blazor Directives

@ block:

Example:

```
<ul>
    @{
        var i = 0;
        while (i < TodoList.Count)
        {
            var todo = TodoList[i];
            <li>
                <label>
                    <input type="checkbox" checked="@todo.IsDone" /> @todo.TaskName |
                    <small>@todo.DueDate.ToShortDateString()</small>
                </label>
            </li>
            i++;
        }
    }
</ul>
```

Split Blazor Component

Partial classes allows us to split the definition of **a class over multiple cs files**. So using this feature we can split the @code block to a separate cs file.

Example file: **Counter.razor**

```
@page "/counter"

<h1>Counter</h1>
<p role="status">Current count: @currentCount</p>
<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>

@code {
    private int currentCount = 0;

    private void IncrementCount()
    {
        currentCount++;
    }
}
```

Split Blazor Component

1. Create a partial class file named **Counter.razor.cs** than move the content of **@code** block to **partial class Counter**.

```
1. using System;
2. namespace Todo.Pages;

3. public partial class Counter
4. {
5.     private int currentCount = 0;
6.     private void IncrementCount()
7.     {
8.         currentCount++;
9.     }
10.}
```

2. Remove the @code block from **Counter.razor** file.

```
1. @page "/counter"

2. <h1>Counter</h1>
3. <p role="status">Current count: @currentCount</p>
4. <button class="btn btn-primary"
   @onclick="IncrementCount">Click me</button>
```

Questions?