Create Your First .NET MAUI Application Labs

Perform these labs on your own computer using Visual Studio 2022 or later to ensure you understand the lessons presented in the corresponding videos and lectures.

Lab 1: Create a .NET MAUI Application

Open **Visual Studio 2022** and select **Create a new project** as shown in Figure 1.

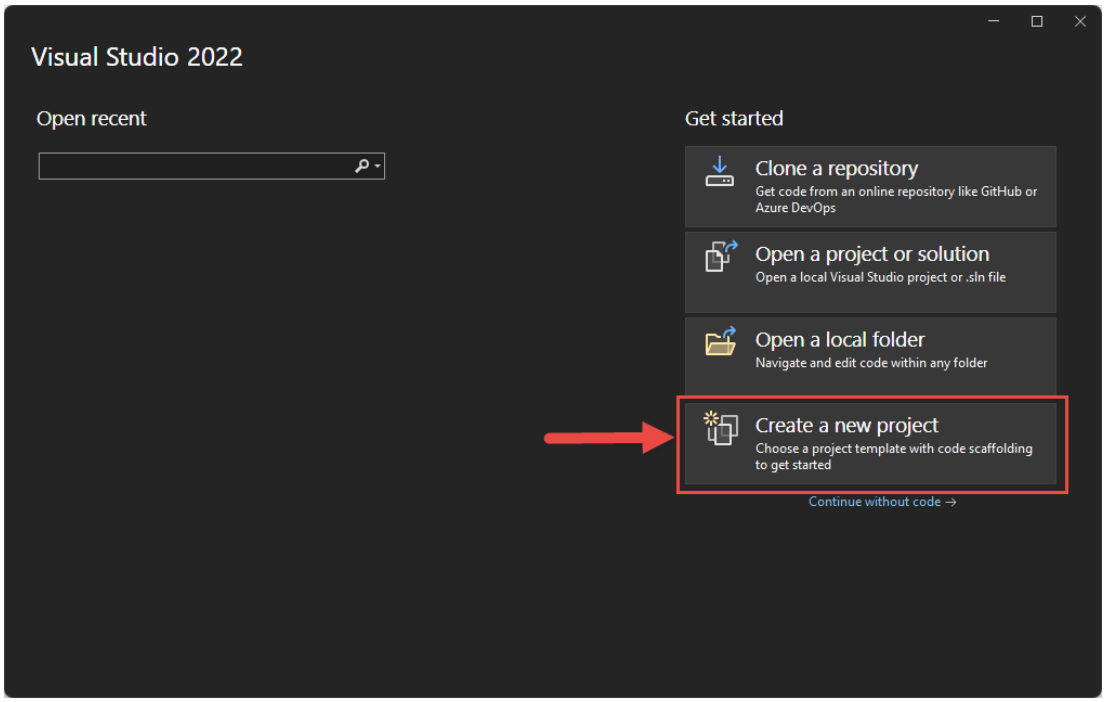

Figure 1: Select Create a new project from the Visual Studio screen.

On the Create a new project page, select a **.NET MAUI App** from the list of templates (Figure 2). Be sure to choose the **C#** language. Click the **Next** button.

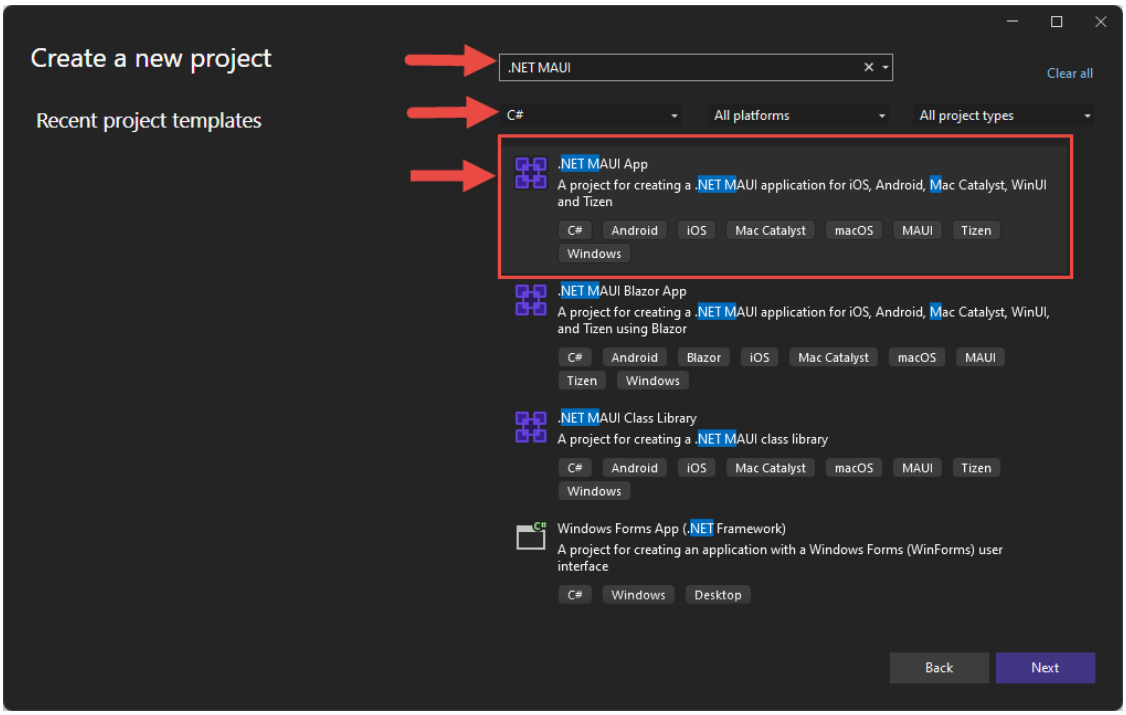

Figure 2: Select a .NET MAUI App from the list of templates.

On the Configure your new project screen, set the **Project name** of the new project to **AdventureWorks.MAUI** (Figure 3).

Set the **Location** to a folder on your hard drive.

Uncheck the **Place solution and project in the same directory**.

Click the **Next** button.

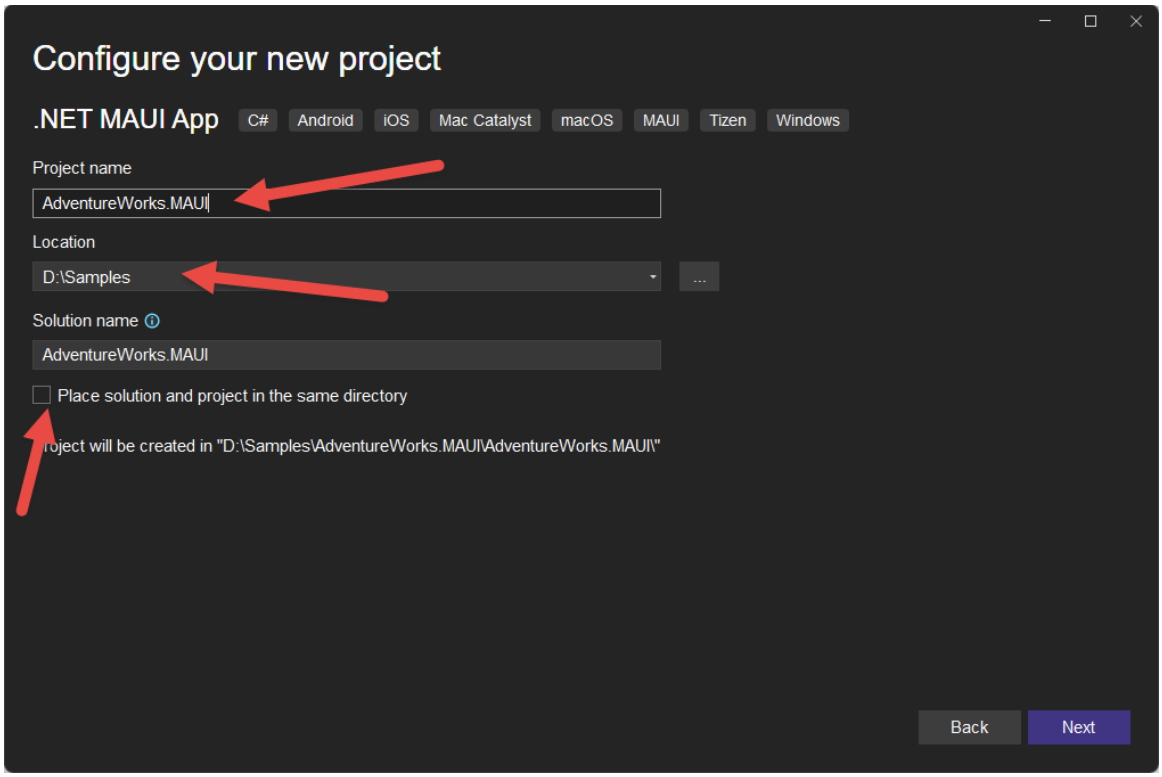

Figure 3: Set the project name and location.

On the Additional information screen, choose **.NET 8** or later, then click on the **Create** button.

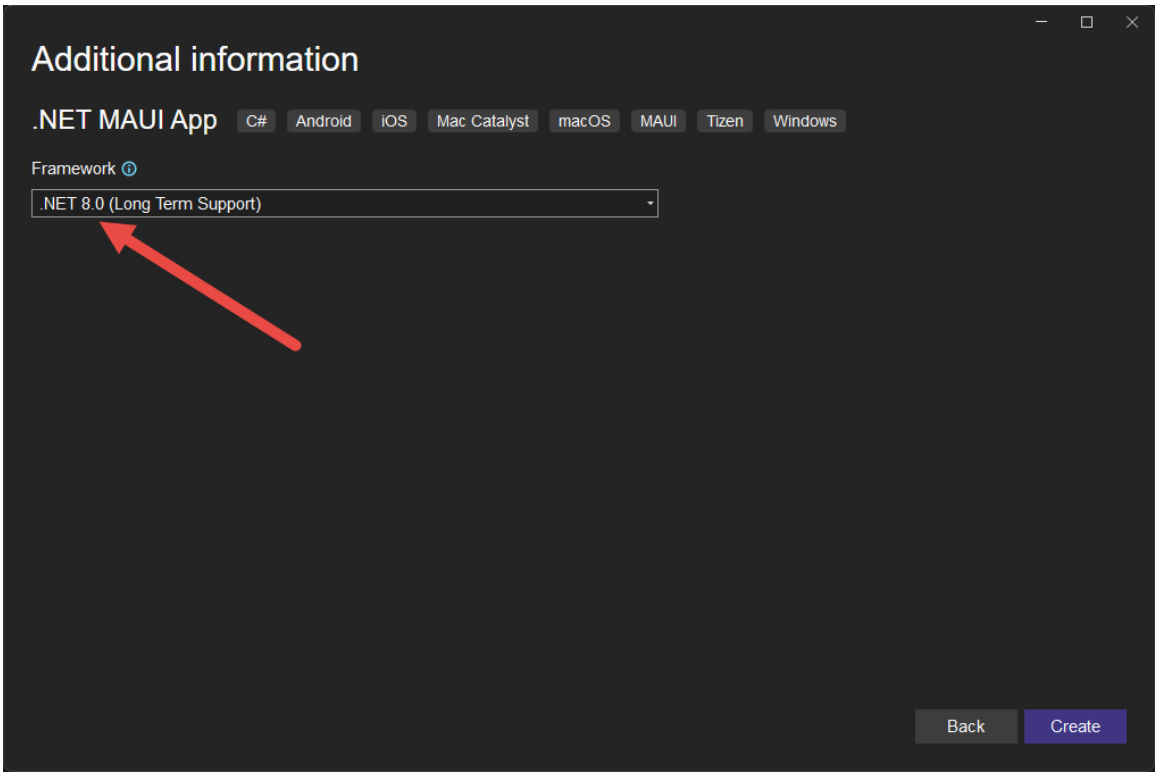

Figure 4: Choose .NET 8 or later when creating a .NET MAUI application.

Try It Out

Once the application has been created, click on the **Windows Machine** button (Figure 5) in the VS toolbar to run this application on Windows.

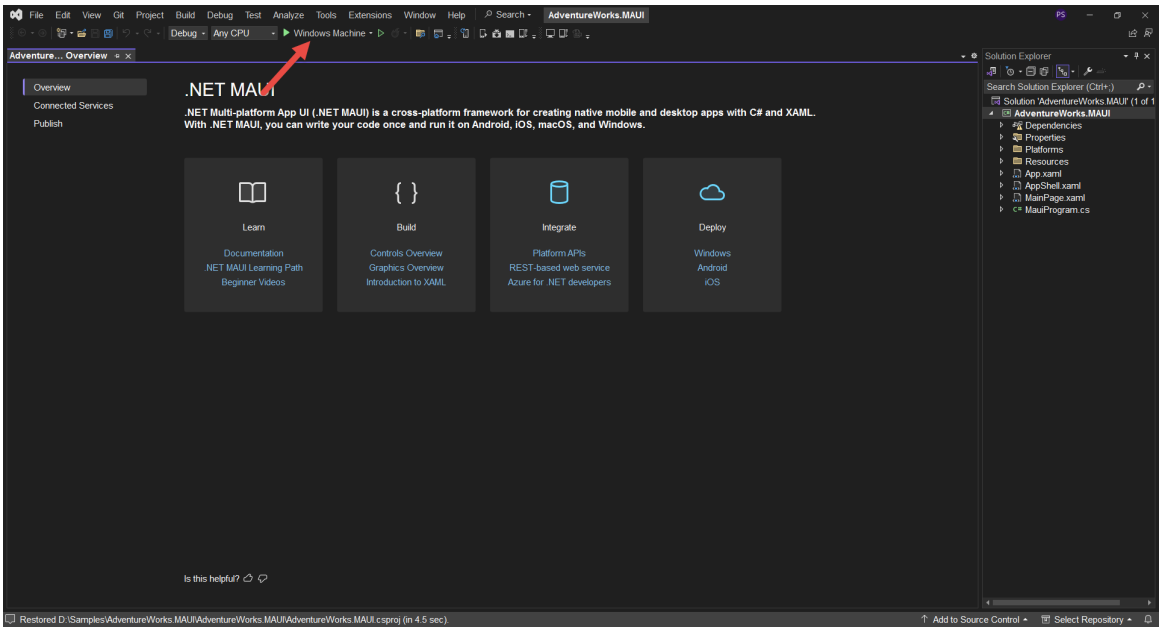

Figure 5: Run the .NET MAUI application on Windows.

The application should look like Figure 6.

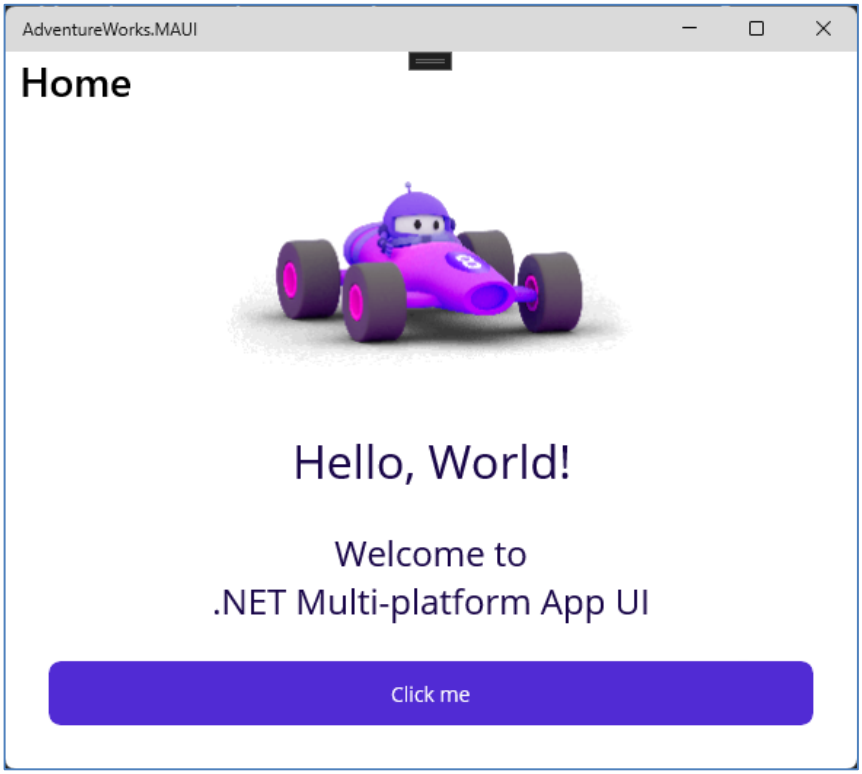

Figure 6: The .NET MAUI template application running on Windows.

Stop the application and now change to use the **Android Emulators | Pixel 5 – API 34** as shown in Figure 7

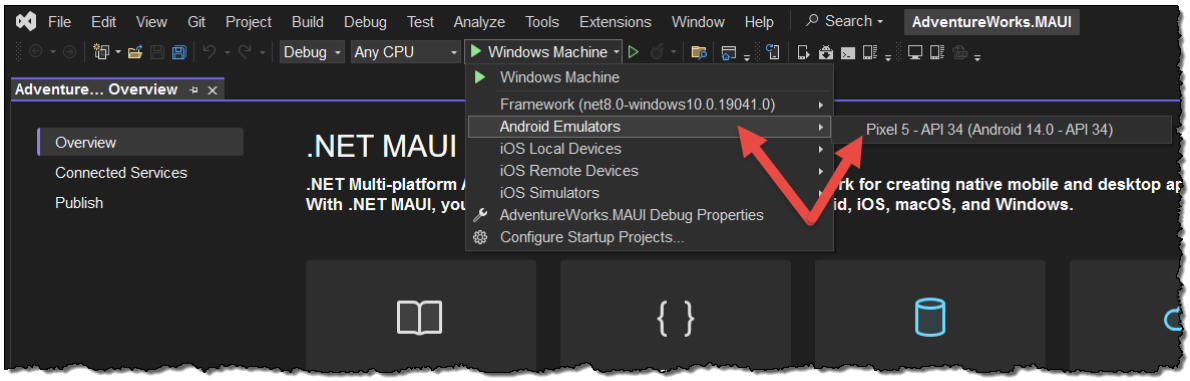

Figure 7: Choose the Android Emulators | Pixel 5 – API 34.

After selecting the Android emulator, it appears as the configuration item (Figure 8). Click on this button to run the Android emulator. It may take a few minutes to run this emulator, so be patient.

Even if the emulator comes up, it may still take a few minutes for VS to compile the application and deploy it to the emulator. Keep an eye on the status bar at the

bottom left of the VS window as it should show you the progress and provide you with messages as to what is happening.

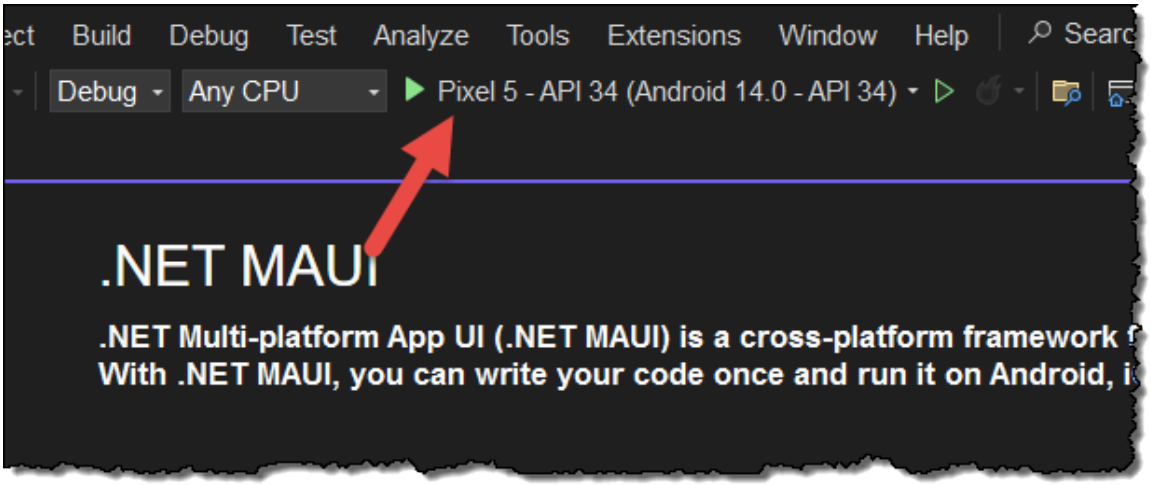

Figure 8: Run the application in the Android emulator.

Once it is completely deployed, you should see a screen that looks like Figure 9.

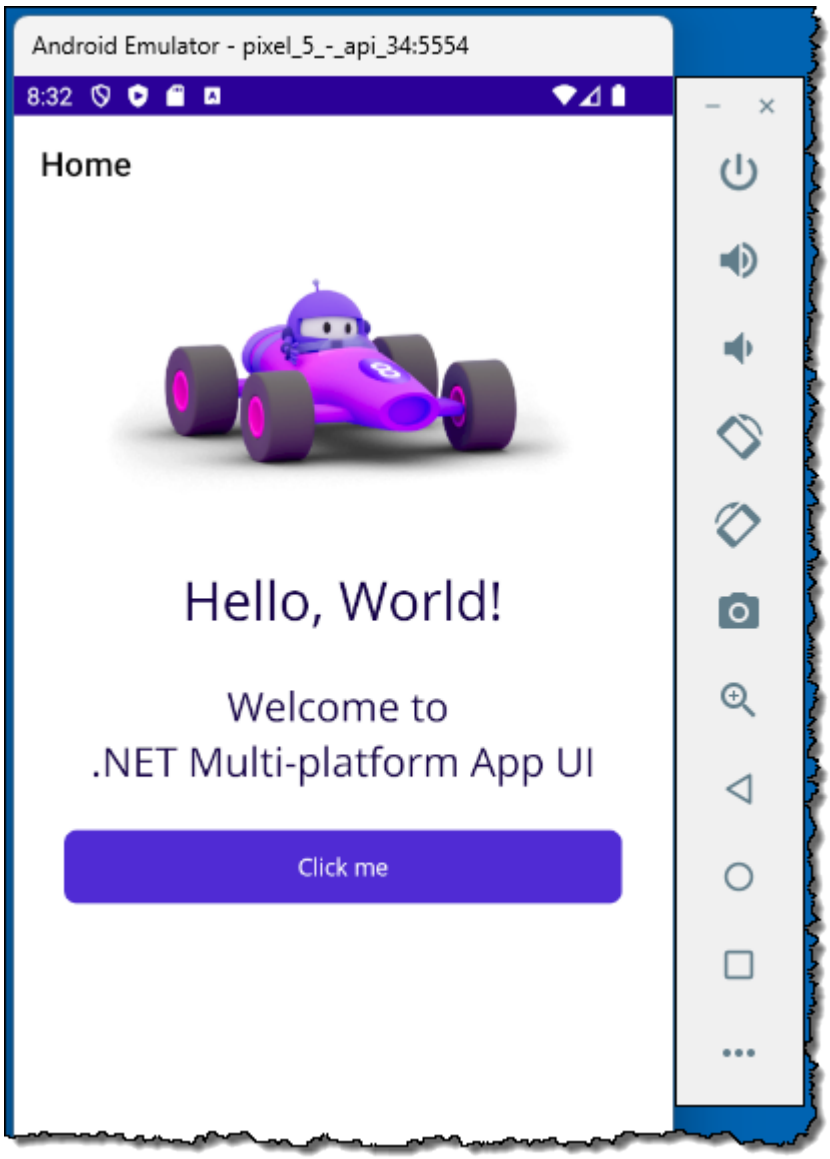

Figure 9: The .NET MAUI template application running on the Android emulator.

Lab 2: Content Page

Open the **MainPage.xaml** file and remove lines 7 through 34. This is the <ScrollView> tag and everything down to (and including) the closing </ScrollView> tag.

Add the following <Label> where the <ScrollView> element was located.

```
<Label Text="Adventure Works"
       FontSize="Large"
       VerticalOptions="Center"
       HorizontalOptions="Center" />
```

Open the **MainPage.xaml.cs** file and remove the code behind, except for the constructor as shown in Figure 10.

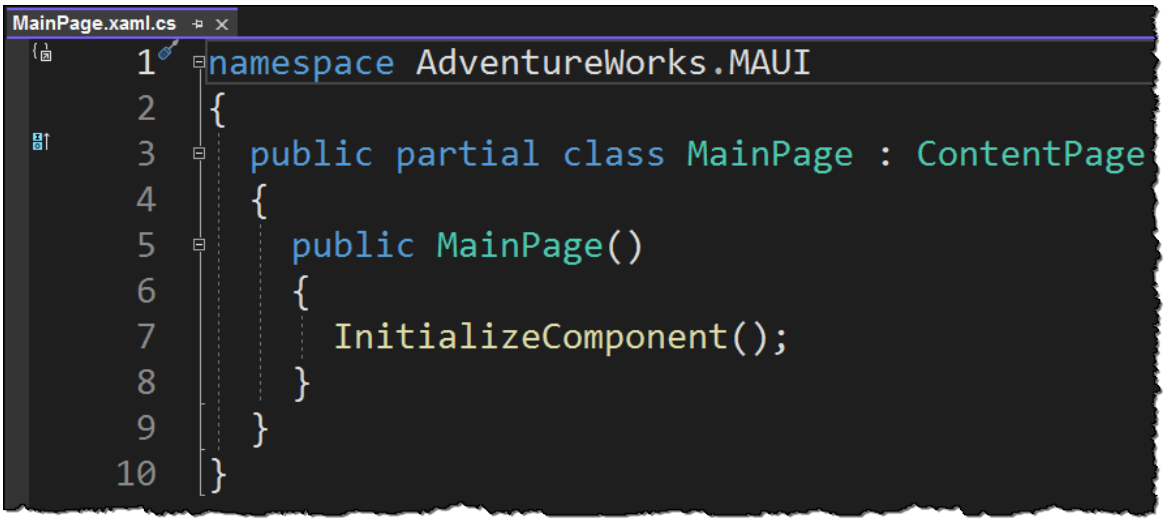

Figure 10: Remove everything but the constructor from the code-behind.

Set the Title Property

Open the **MainPage.xaml** file and click on line 2 so you are focused on the <ContentPage> element. Bring up the **Properties** window (Figure 11) by selecting **View | Properties Window** from the VS menu. The properties you can set for the <ContentPage> element are listed within this window.

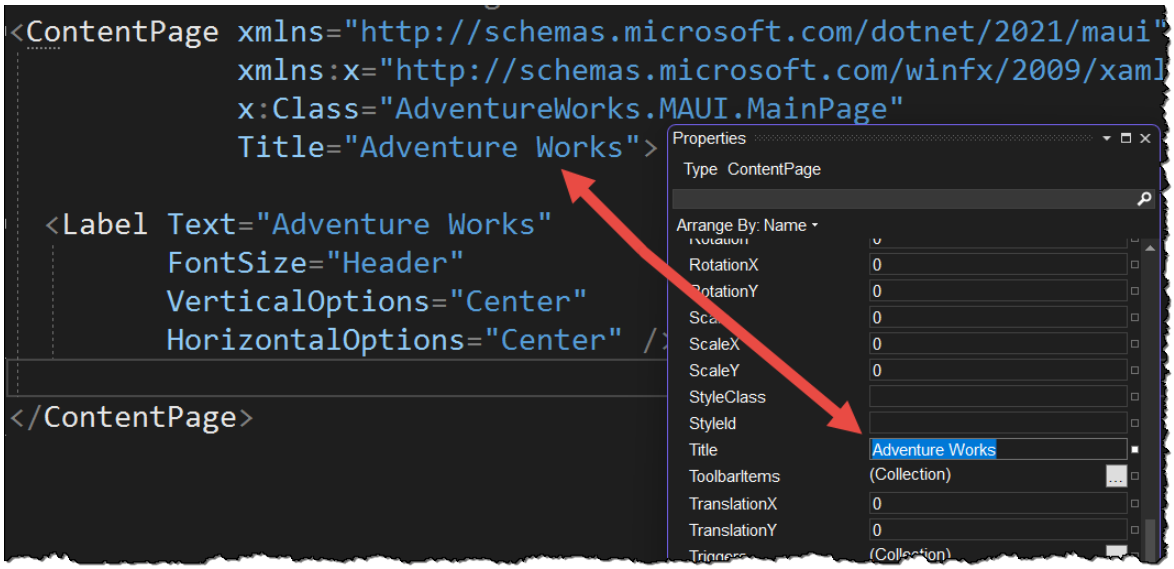

Figure 11: The Properties window allows you to set properties on whatever control has focus.

Set the **Title** property of the ContentPage to **Adventure Works** and press the **Enter** key. You should see the XAML has been written for you in the editor.

Close the **Properties** window and go to the editor and click just after the **Title="Adventure Works"** attribute and hit the spacebar. You should see an IntelliSense window open in your editor that has the same list of properties you saw in the Properties window.

Set the **BackgroundColor** attribute to **Gray**.

Try It Out

Run the application either on Windows or the Android emulator to see the new **Title** attribute on the screen. Notice the Gray background (Figure 12).

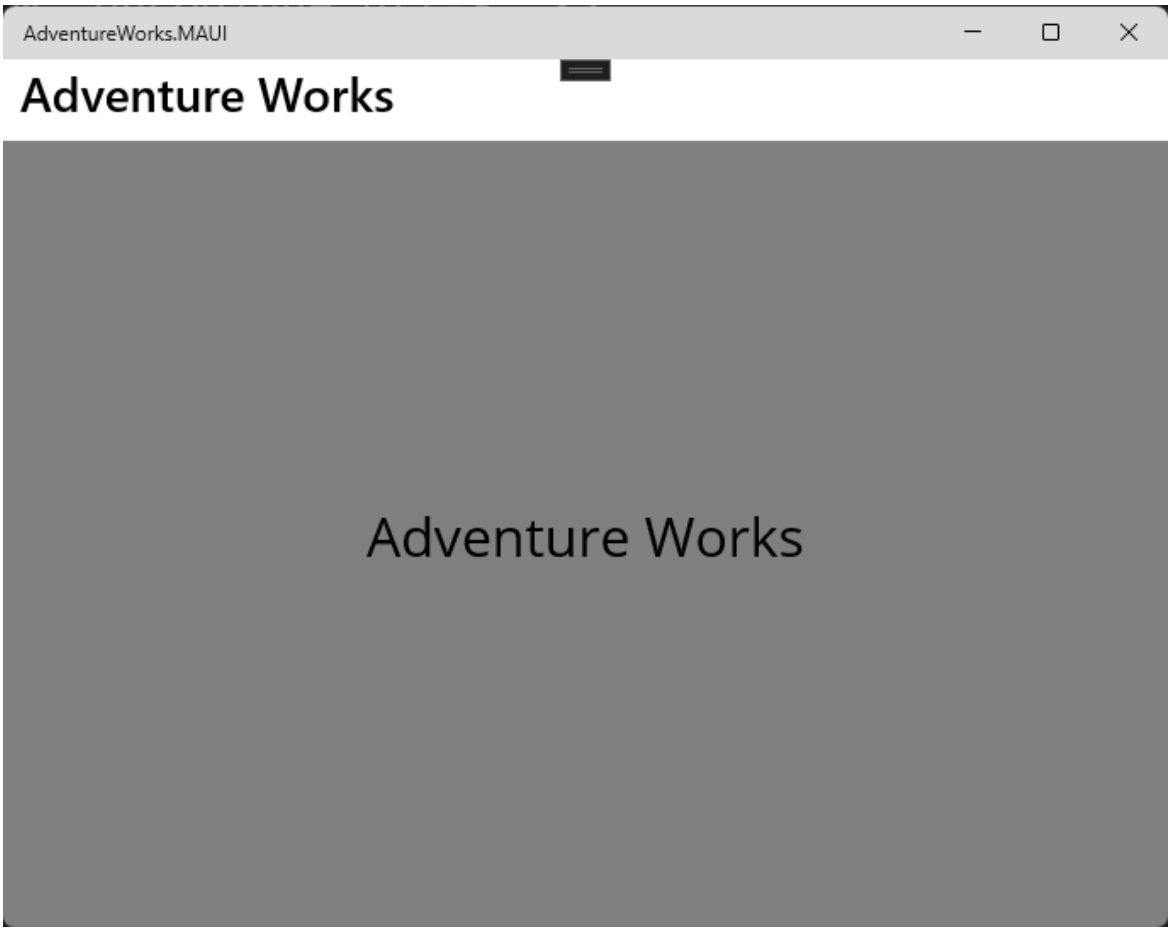

Figure 12: Setting properties on the <ContentPage> affects the appearance of the view.

DO NOT STOP the application.

Remove the **Background** property from the <ContentPage> element.

Show the instant results of the change.

Lab 3: Shell Properties

Notice the Window title still reads "AdventureWorks.MAUI". This is the name of the project, but it is probably not what you want on the Window title. Open the **AppShell.xaml** file and set the attributes on the <Shell> element shown in bold below.

```
<Shell x:Class="XamlBasicsMAUI.AppShell"
       xmlns="http://schemas.microsoft..."
       xmlns:x="http://schemas.microsoft..."
       xmlns:local="clr-namespace:XamlBasicsMAUI"
       Shell.TitleColor="Black"
       Shell.BackgroundColor="Gray"
       Title="Adventure Works"
       Shell.FlyoutBehavior="Disabled">
```

Try It Out

Run the application on the **Windows Machine** and view the background of the title area is gray and the title color is black (Figure 13). Also notice there is now a title on the Windows title bar area. The title bar only shows up on Windows and not on Android or iOS as that UI does not have a title bar area.

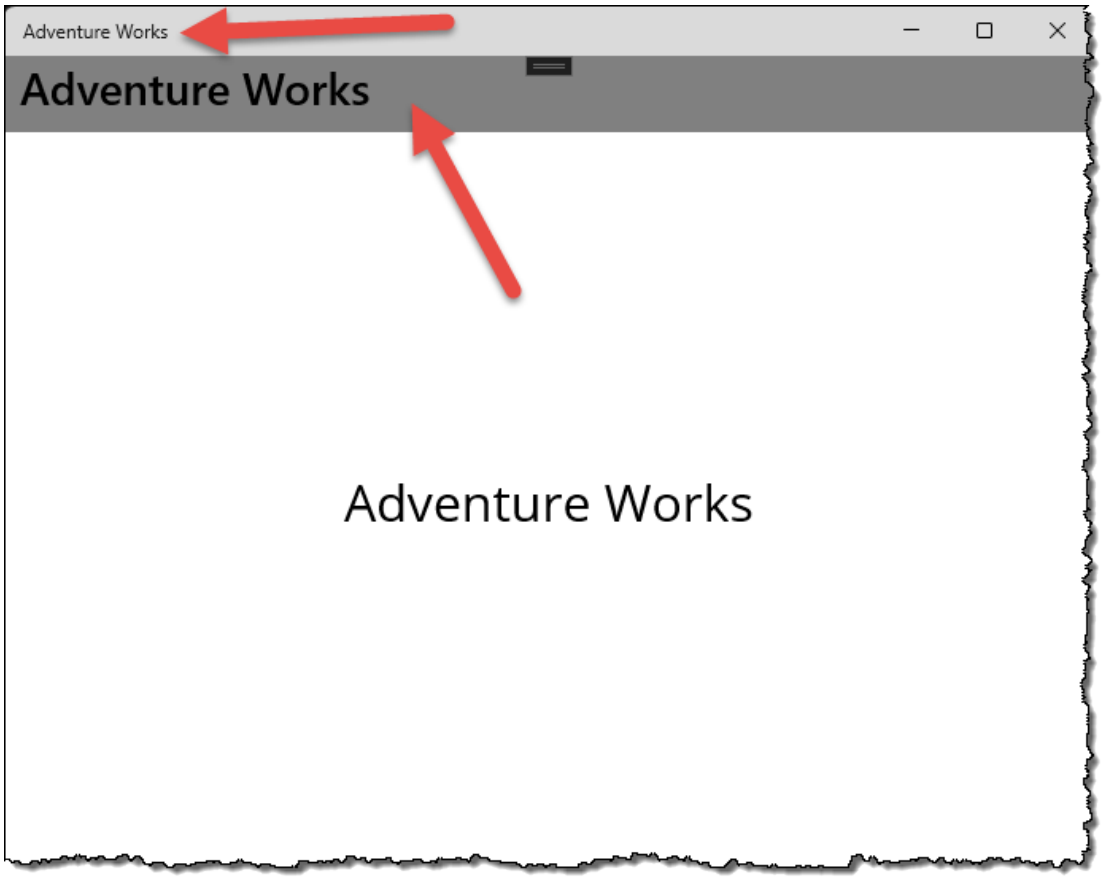

Figure 13: Changes to the application shell are controlled by the <Shell> element.

Remove the following attributes from the <Shell> element within the **AppShell.cs** file.

```
Shell.TitleColor="Black"
Shell.BackgroundColor="Gray"
```

Lab 4: Set Maximized Window

Open the **MauiProgram.cs** file and add a using statement at the top of the file.

```
#if WINDOWS
using Microsoft.Maui.LifecycleEvents;
#endif
```

NOTE: The code may look disabled, this is OK.

Add a new method within the **MauiProgram** class named SetWindowOptions() as shown in the code below.

```
#if WINDOWS
public static void SetWindowOptions(MauiAppBuilder
builder)
{
  builder.ConfigureLifecycleEvents(events =>
  {
    events.AddWindows(wndLifeCycleBuilder =>
    {
      wndLifeCycleBuilder.OnWindowCreated(window =>
      {
        IntPtr nativeWindowHandle =
WinRT.Interop.WindowNative.GetWindowHandle(window);
        Microsoft.UI.WindowId win32WindowsId =
Microsoft.UI.Win32Interop.GetWindowIdFromWindow(nativeWi
ndowHandle);
        Microsoft.UI.Windowing.AppWindow winuiAppWindow
=
Microsoft.UI.Windowing.AppWindow.GetFromWindowId(win32Wi
ndowsId);
        if (winuiAppWindow.Presenter is
Microsoft.UI.Windowing.OverlappedPresenter p) {
          p.Maximize();
          //p.IsResizable = false;
          //p.IsMaximizable = false;
          //p.IsMinimizable = false;
        }
      });
    });
  });
}
#endif
```

NOTE: The code may look disabled, this is OK.

Just before the #if DEBUG, call this new method.

```
#if WINDOWS
      SetWindowOptions(builder);
#endif
```

Try It Out

Run the application on the **Windows Machine** and see the window is now full screen.