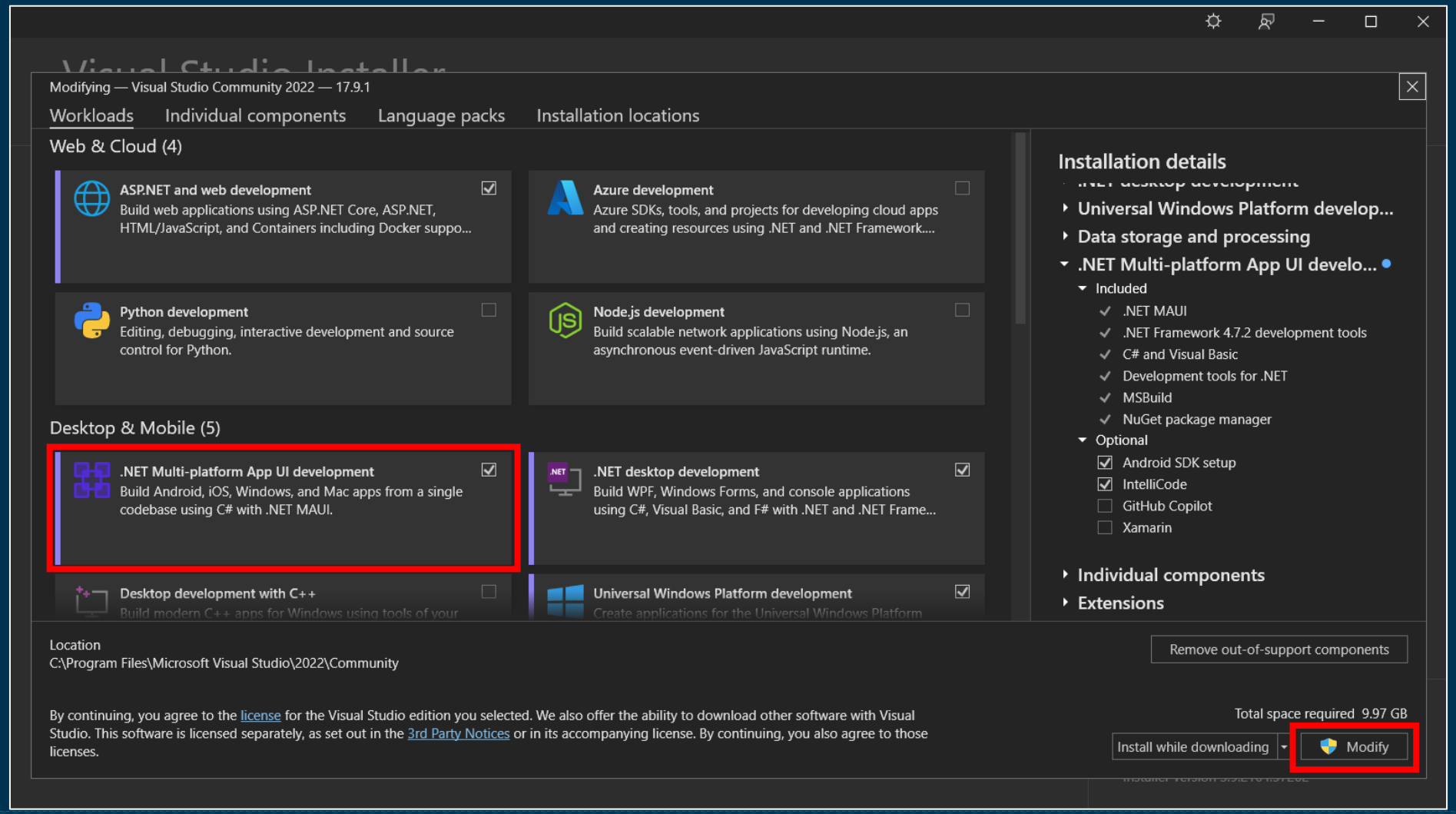
Introduction to .NET MAUI

Install the Workload

- In Visual Studio, the .NET Multi-platform App UI development workload should be installed.



Introduction

- .NET MAUI is an acronym for **M**ulti-platform **A**pplication **U**ser **I**nterface.
- .NET MAUI is a multi-platform framework for creating native desktop and mobile apps with C# and XAML.
- Using .NET MAUI, you can design mobile apps that can run on Windows, Android, iOS, iPadOS, and macOS.
- .NET MAUI was released on May 23, 2022.

- Learn more:
  - https://learn.microsoft.com/en-us/dotnet/maui/

Introduction

- A common pattern used for cross-platform app development is to distinguish the business logic from the user interface.
- And then develop separate user interfaces and UI logic for each platform.
- While the business logic remains unchanged for each type of device, the code that drives the app and presents the data can vary.
- This variance is because of the differing capabilities, APIs, and features the devices provide.
- Building a multiplatform app in this way involves dealing with not only separate SDKs, but altogether different languages and toolsets.

- The purpose of .NET MAUI is to simplify multiplatform app development.
- Using .NET MAUI, you create multiplatform apps using a single project, but you can add platform-specific source code and resources if necessary.
- The key aim of .NET MAUI is to enable you to implement as much of your application logic and UI layout as possible in a single codebase.

.NET MAUI Architecture

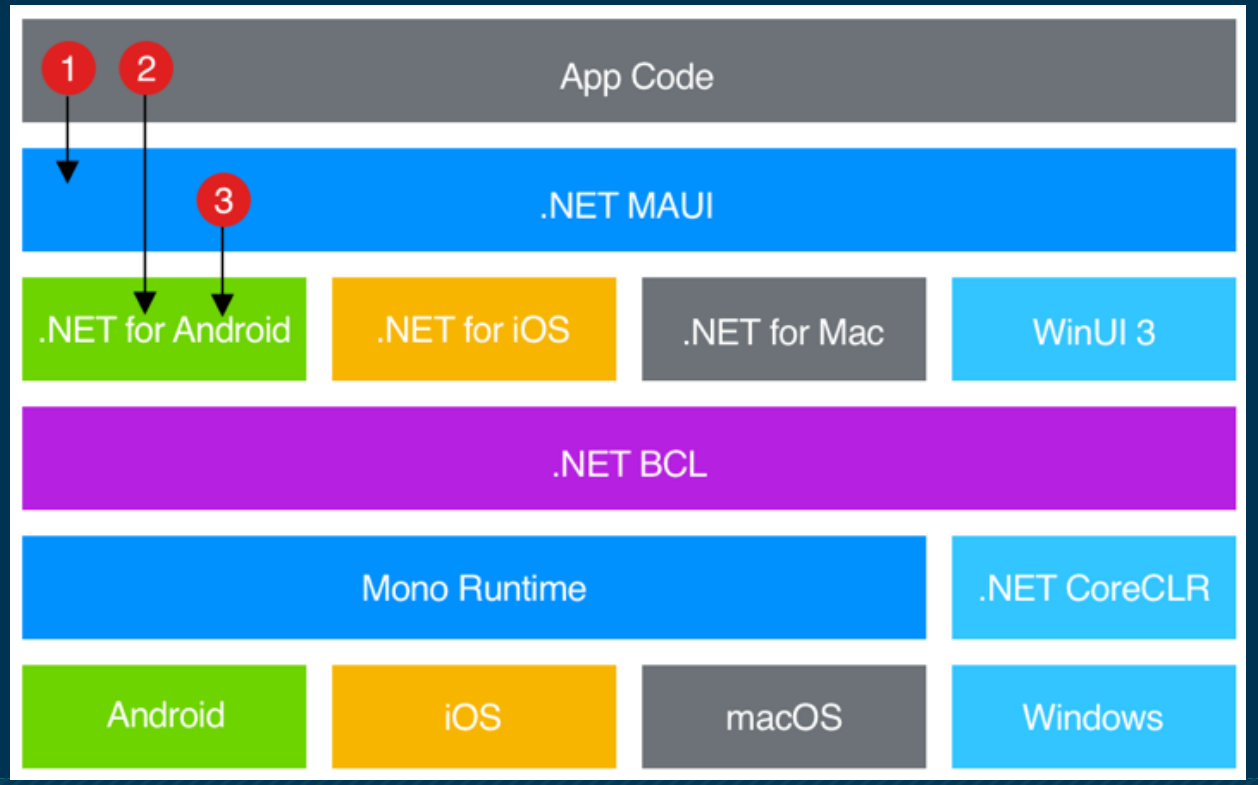
- .NET provides a series of platform-specific frameworks for creating apps:
  - .NET for Android
  - .NET for iOS (and iPadOS)
  - .NET for Mac
  - WinUI 3 for Windows
  - UWP for Windows (*now deprecated*)
- These frameworks all have access to the same .NET Base Class Library (BCL).
- This library provides the functionality for creating and managing resources, and for generally abstracting the details of the underlying device away from your code.
- The BCL depends on the .NET runtime to provide the execution environment for your code.
- For Android, iOS (and iPadOS), and macOS, the environment is implemented by Mono, an open-source implementation of the .NET runtime.
- On Windows, Win32 performs the same role, except it's optimized for the Windows platform.

.NET MAUI Architecture

- While the BCL enables applications running on different types of devices to share common business logic, the various platforms have different ways of defining an application's user interface.
- The platforms provide varying models for specifying how the user interface elements communicate and interoperate.
- You can craft the UI for each platform separately using the appropriate platform-specific framework (.NET for Android, .NET for iOS, .NET for Mac, or WinUI 3), but this approach then requires you to maintain a codebase for each individual family of devices.
- .NET MAUI provides a single framework for building the UIs for mobile and desktop applications.

.NET MAUI Architecture

- You create the UI using this framework (indicated by Arrow 1 in the following diagram), and .NET MAUI takes care of converting it to the appropriate platform (Arrow 2).
- There might be times when you need to implement a platform-specific feature.
- In these situations, you can invoke methods in the platform-specific framework, as highlighted by Arrow 3 in the following diagram.



How Does .NET MAUI Work?

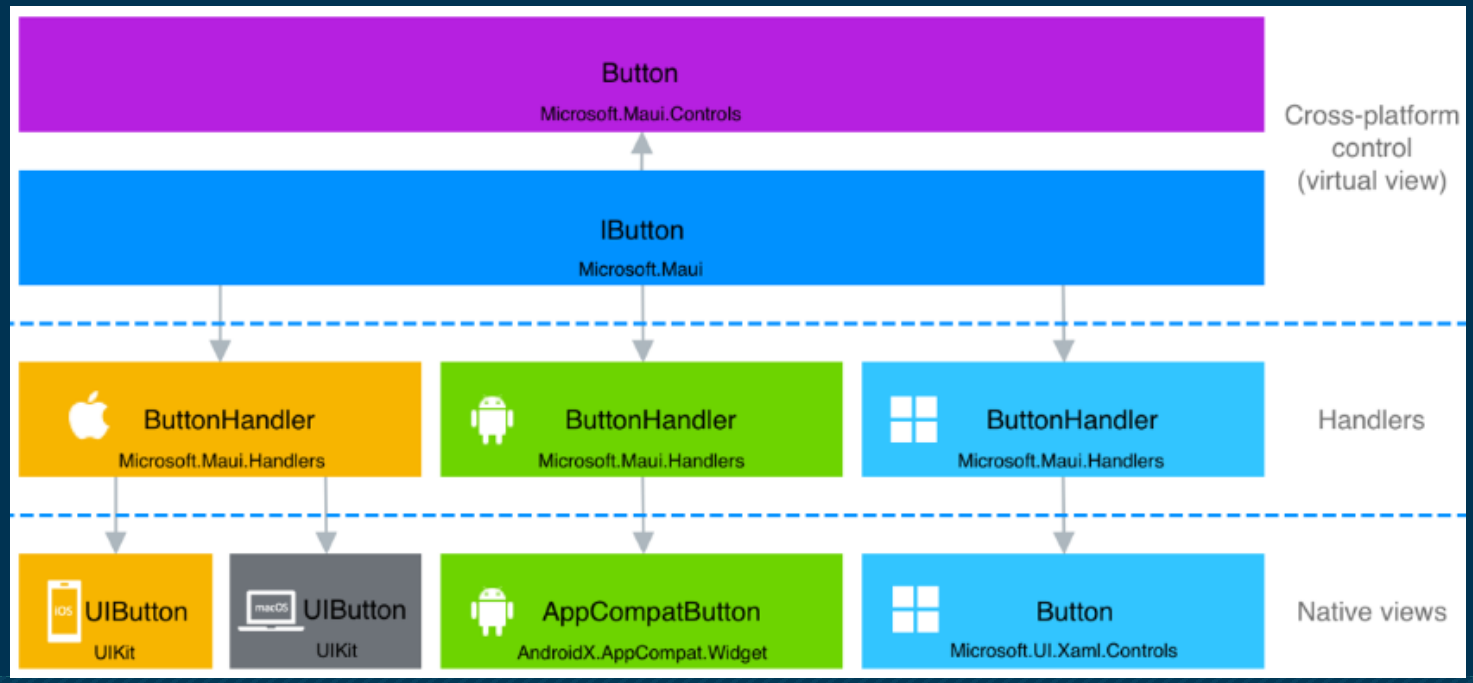
- .NET MAUI abstracts a UI element's implementation from its logical description.
- You can describe the UI using XAML.
- For example, the following XAML fragment shows the description of a button control:

```
<Button Text="Click me"
        SemanticProperties.Hint="Counts the number of times you click"
        Clicked="OnCounterClicked"
        HorizontalOptions="Center" />
```

- This example defines the text for the button ("Click me").
- Specifies that a method named `OnCounterClicked` should be run when the user selects the button.
- Other properties can modify the layout, such as, the text is centered on the button.
- Semantic properties provide support for accessibility for users with visual impairment.

How Does .NET MAUI Work?

- .NET MAUI always generates native code for the target device, so you get optimal performance.
- .NET MAUI uses handlers specific to each platform and UI element to carry out an operation.
- For example, if you target iOS, a .NET MAUI handler will map this code to an iOS `UIButton`.
- If you run on Android, you'll get an Android `AppCompatButton`.
- These handlers are accessed indirectly through a control-specific interface provided by .NET MAUI, such as `IButton` for a button.



.NET MAUI vs. Flutter vs. React Native

- There are many cross-platform frameworks but Flutter and React Native are the most popular ones used by developers across the globe.
- If we compare these based on community size and third-party library compatibility, .NET MAUI is a less mature option, released in May 2022.
- Therefore, it can be challenging to get help from the community if you get stuck somewhere.

- On the other hand, .NET MAUI uses C# and XAML code.
- So, if you're already familiar with the .NET ecosystem, then MAUI can easily become your go-to framework.
- Flutter uses Dart, which is a programming language introduced by Google that has a significantly higher learning curve.
- React Native, developed by Facebook and built on top of JavaScript, is a cross-platform application development framework.

.NET MAUI vs. Flutter vs. React Native

- Although all these frameworks can deploy to Android, iOS, macOS, and Windows, some require extra tweaking to do.
- Like React Native, which uses react-native-windows and react-native-macos for Windows and macOS support.
- On the other hand, .NET MAUI comes with support for these out of the box.

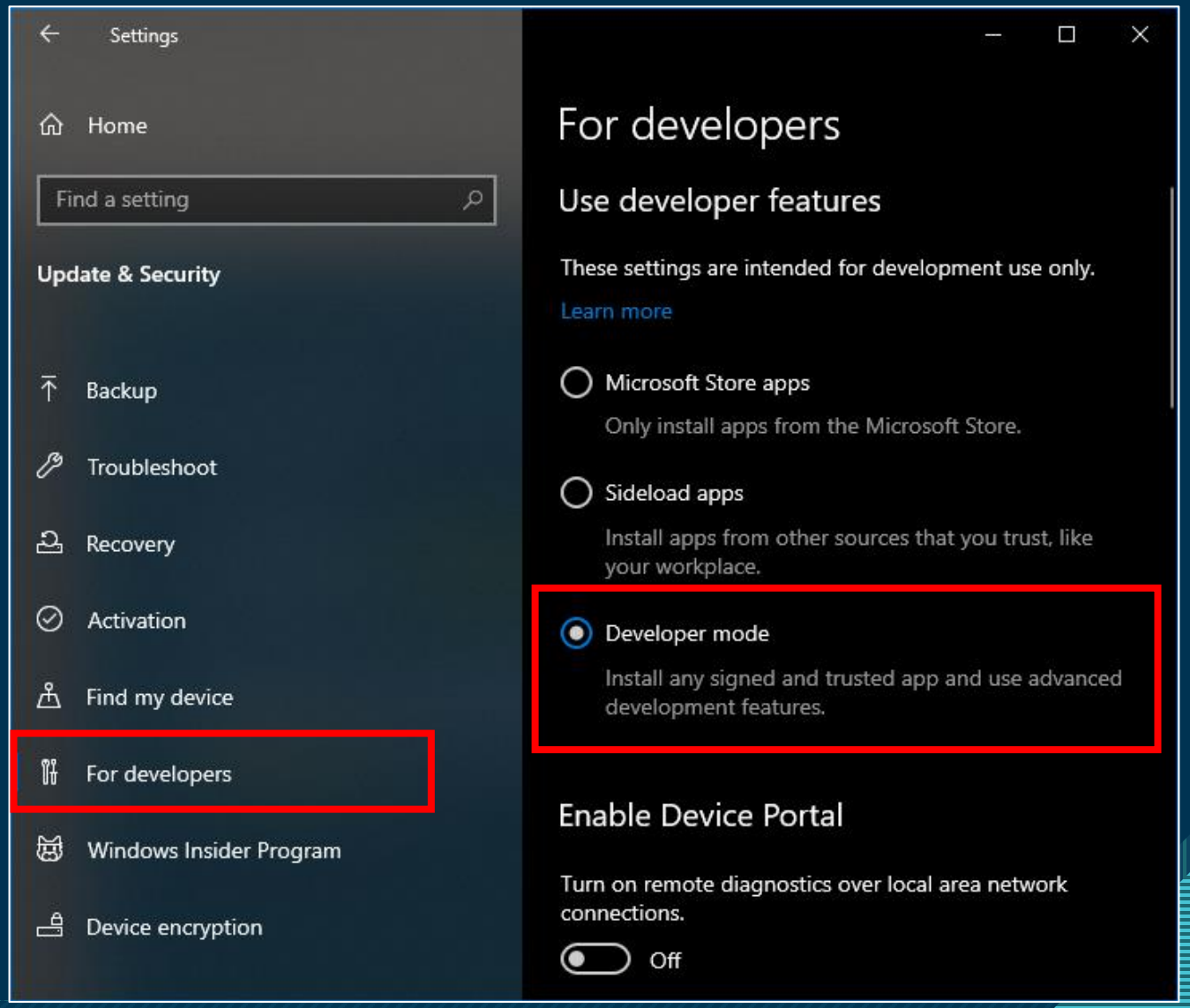
- However, unlike Flutter and React Native, you can’t deploy .NET MAUI apps directly on the web.
- As an added bonus for Flutter, Flutter apps can also be distributed and run on Linux based operating systems.
- Ultimately, it’s up to you to decide which framework is the best fit.
- But if you need a .NET ecosystem, .NET MAUI can surely be your framework of choice.

Enabling Developer Mode - Windows 10

- To be able to create apps for .NET MAUI, you must enable developer mode in Windows.

1. For Windows 10, navigate to Start | Settings | Update & Security | For developers, and then click on Developer mode.

2. Accept the warning about how it "could expose your device and personal data to security risk or harm your device," and then close the Settings app.

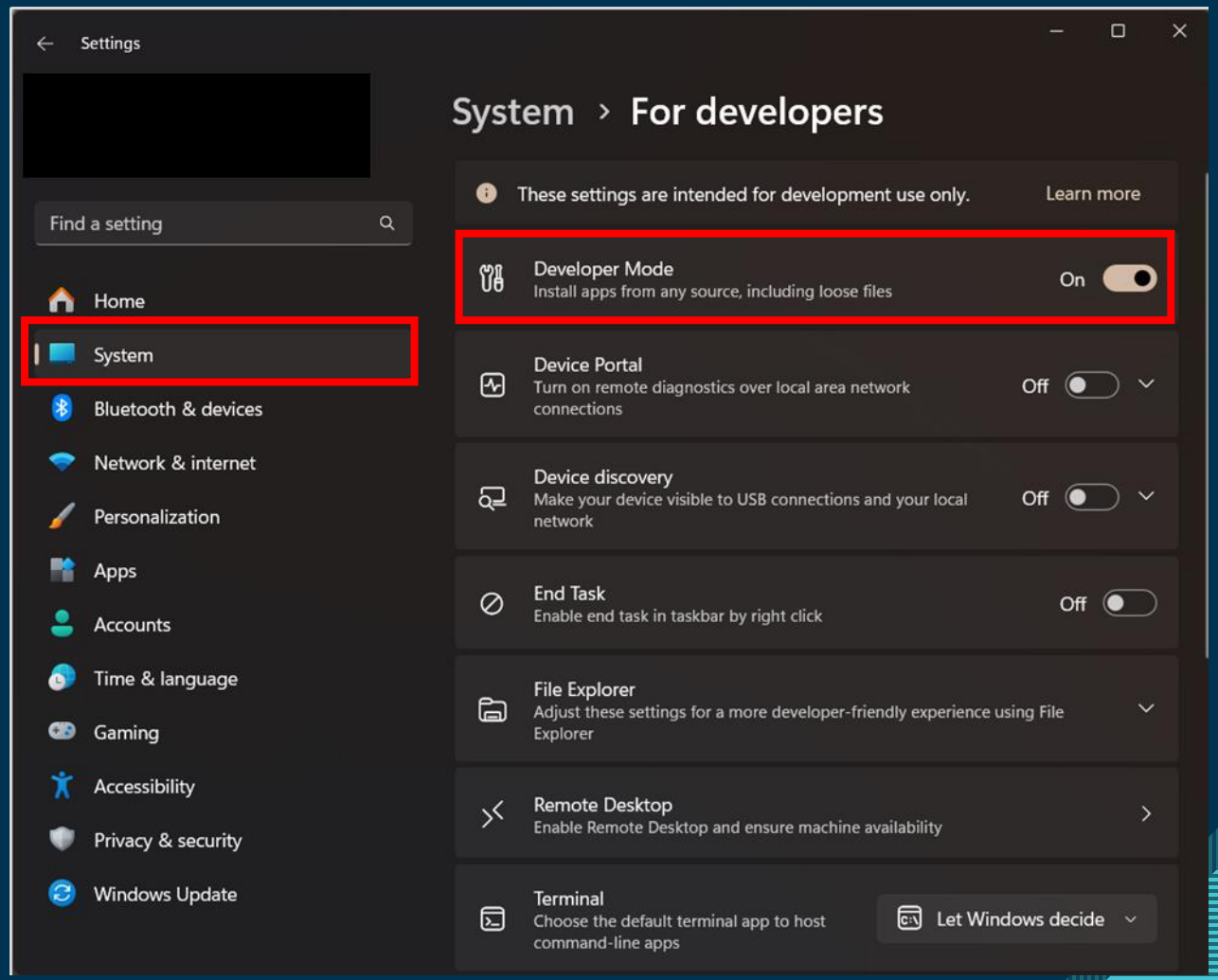


Enabling Developer Mode - Windows 11

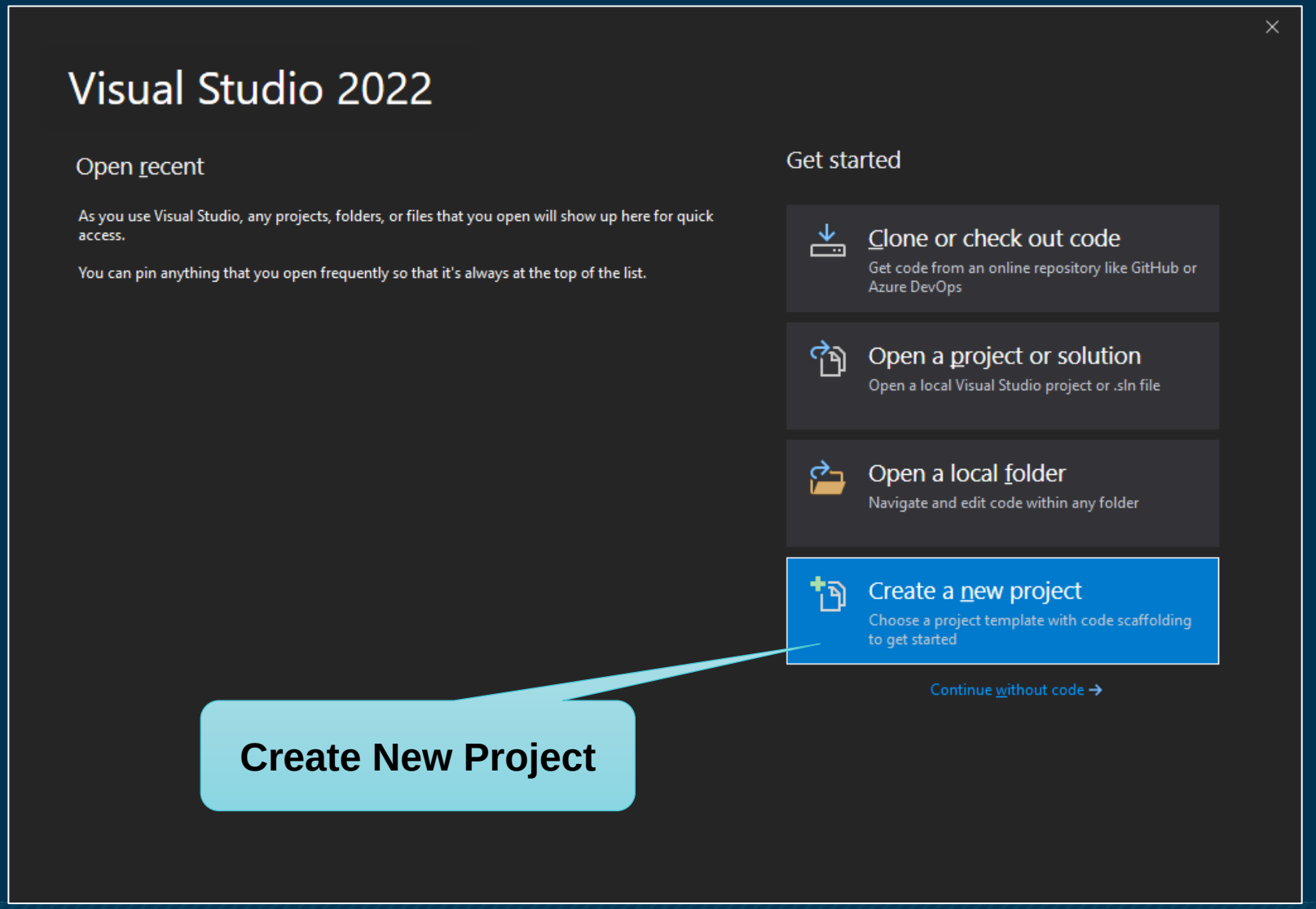
- For Windows 11:

1. Navigate to Start | Settings | System | For developers, and then enable the Developer mode.

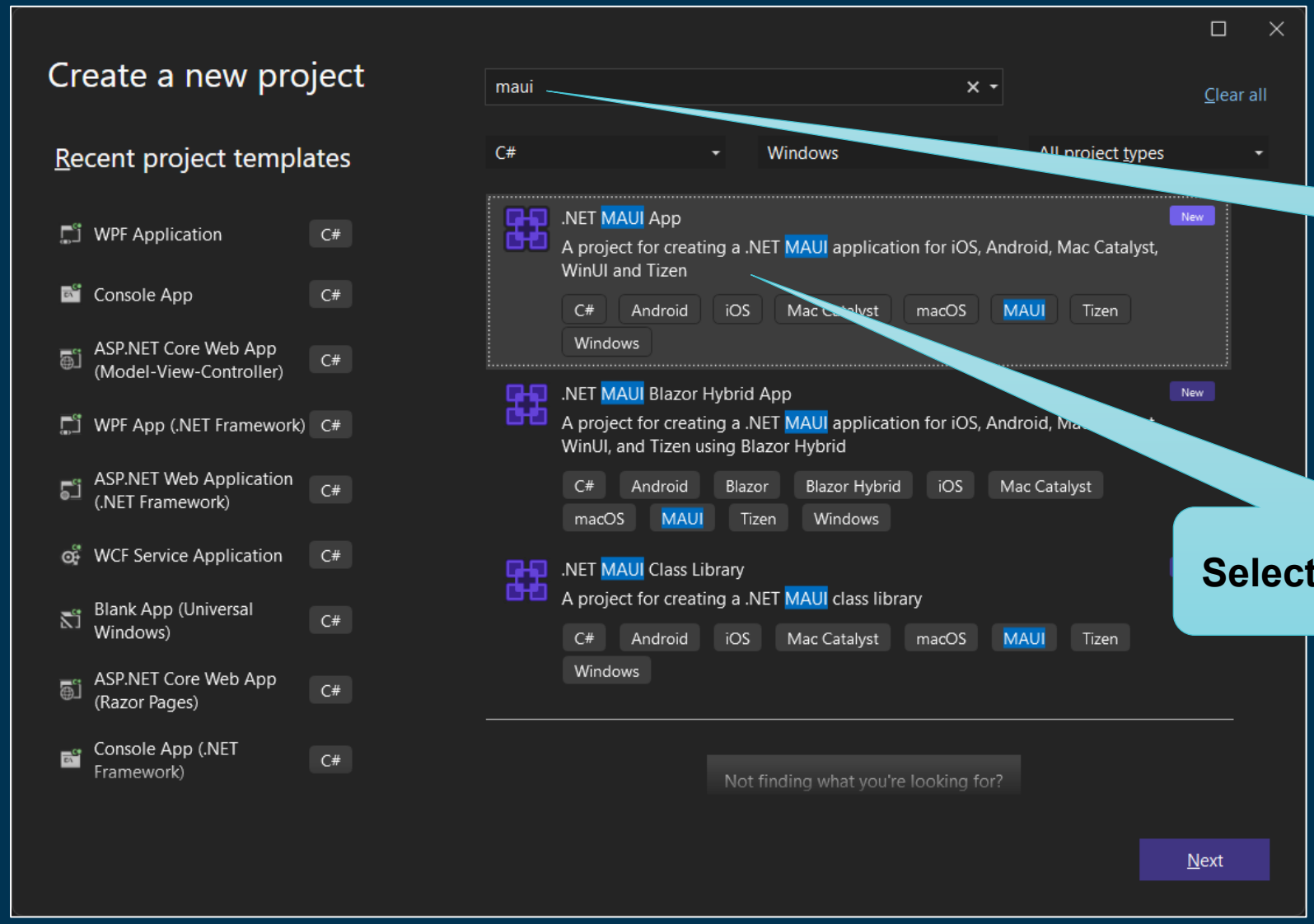
2. Accept the warning about how it "could expose your device and personal data to security risk or harm your device," and then close the Settings app.



Creating a .NET MAUI Project



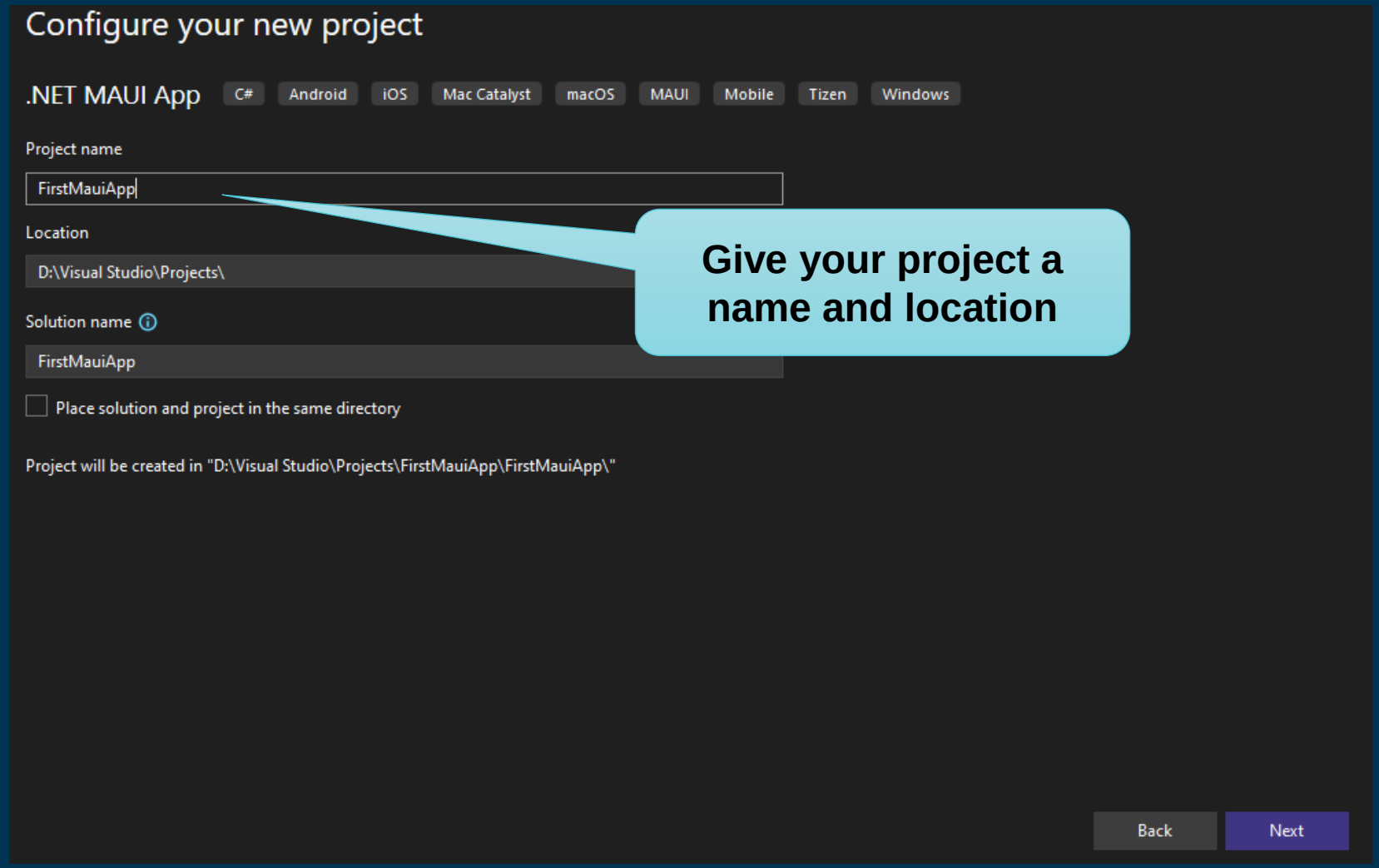
Creating a .NET MAUI Project



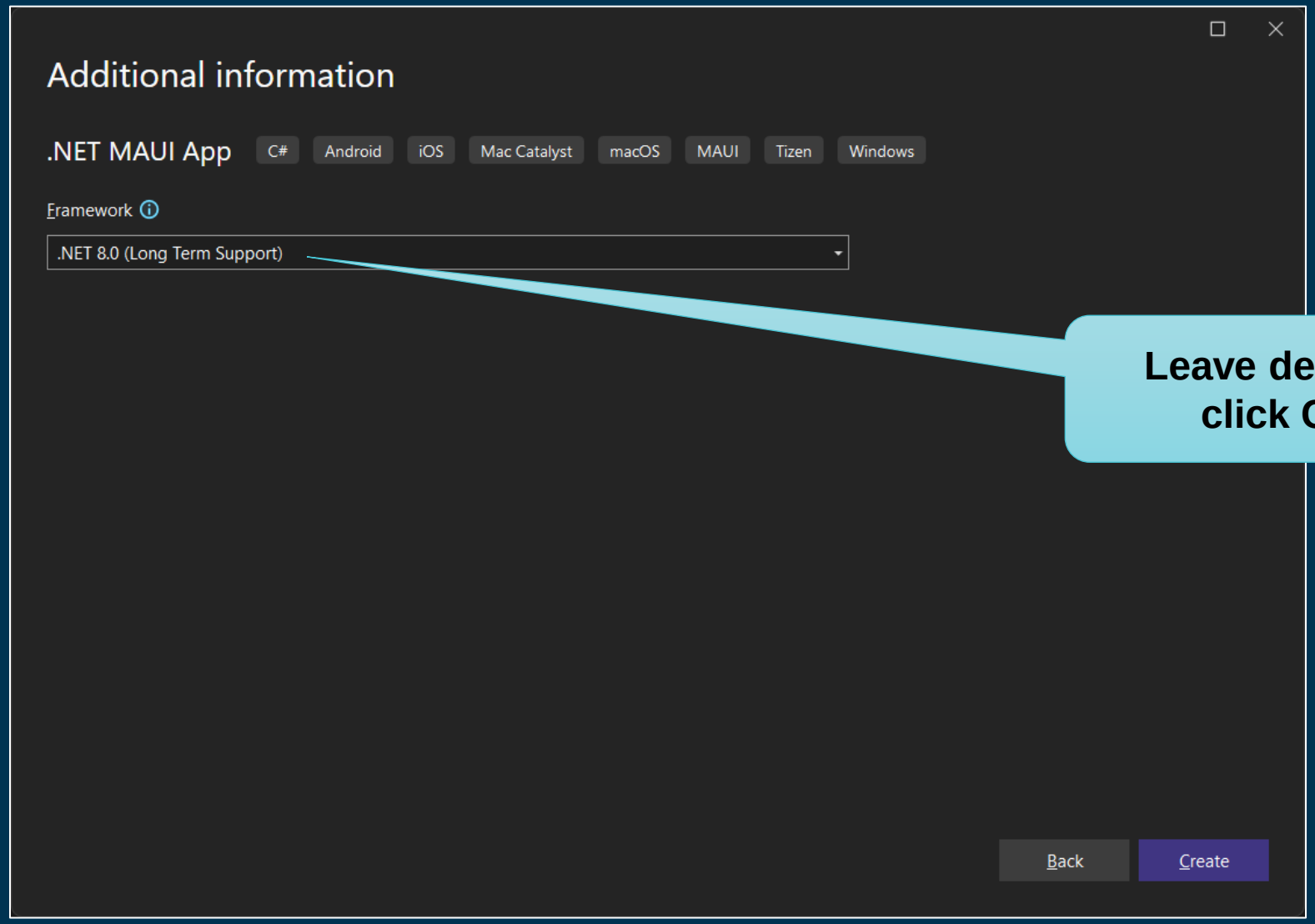
Search for "maui"

Select ".NET MAUI App"

Creating a .NET MAUI Project

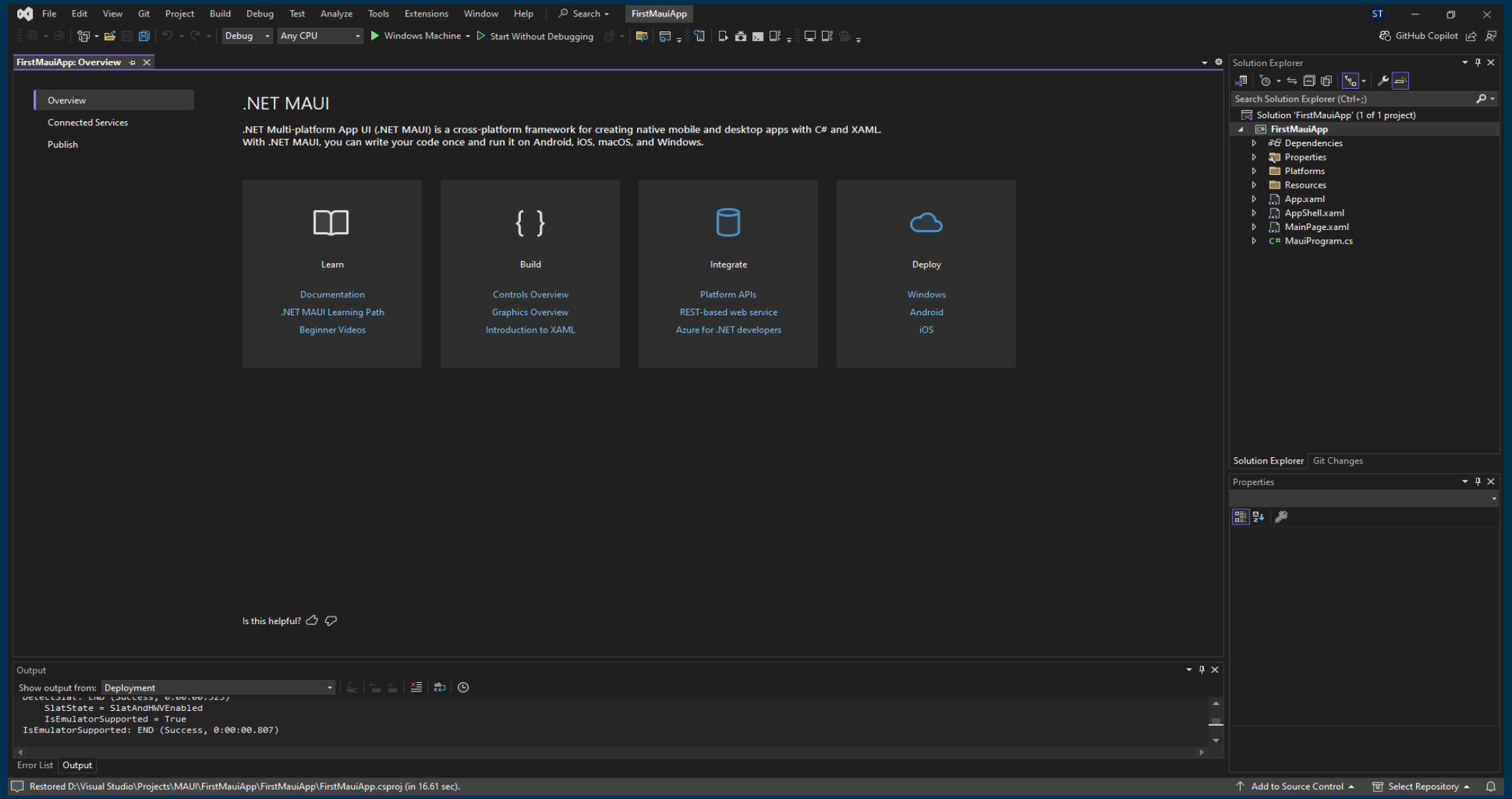


Creating a .NET MAUI Project



Leave default and click Create

Creating a .NET MAUI Project

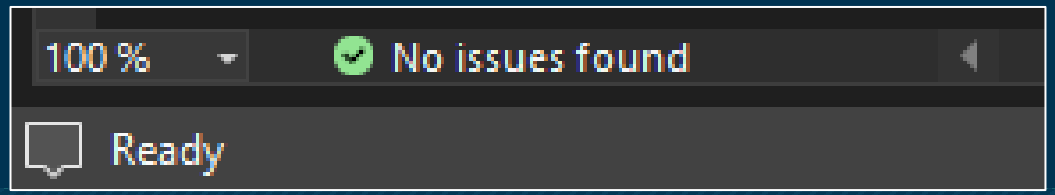


Creating a .NET MAUI Project

- Visual Studio creates the new project.
- If you get a Windows Security Alert warning you about the firewall blocking some features, select the Allow access button.

Wait for NuGet packages restore process:

- NuGet is a package manager that will bring in the dependencies of your new app.
- The package restore process will start automatically.
- Wait until the Restored or Ready message appears in the status bar at the bottom left of the screen.



.NET MAUI Project Structure and Application Startup

App.xaml:

- This file defines the application resources that the app will use in the XAML layout.
- The default resources are in the Resources folder and define app-wide colors and default styles for every .NET MAUI built-in control.
- Here, you'll see the two resource dictionaries being merged.

App.xaml

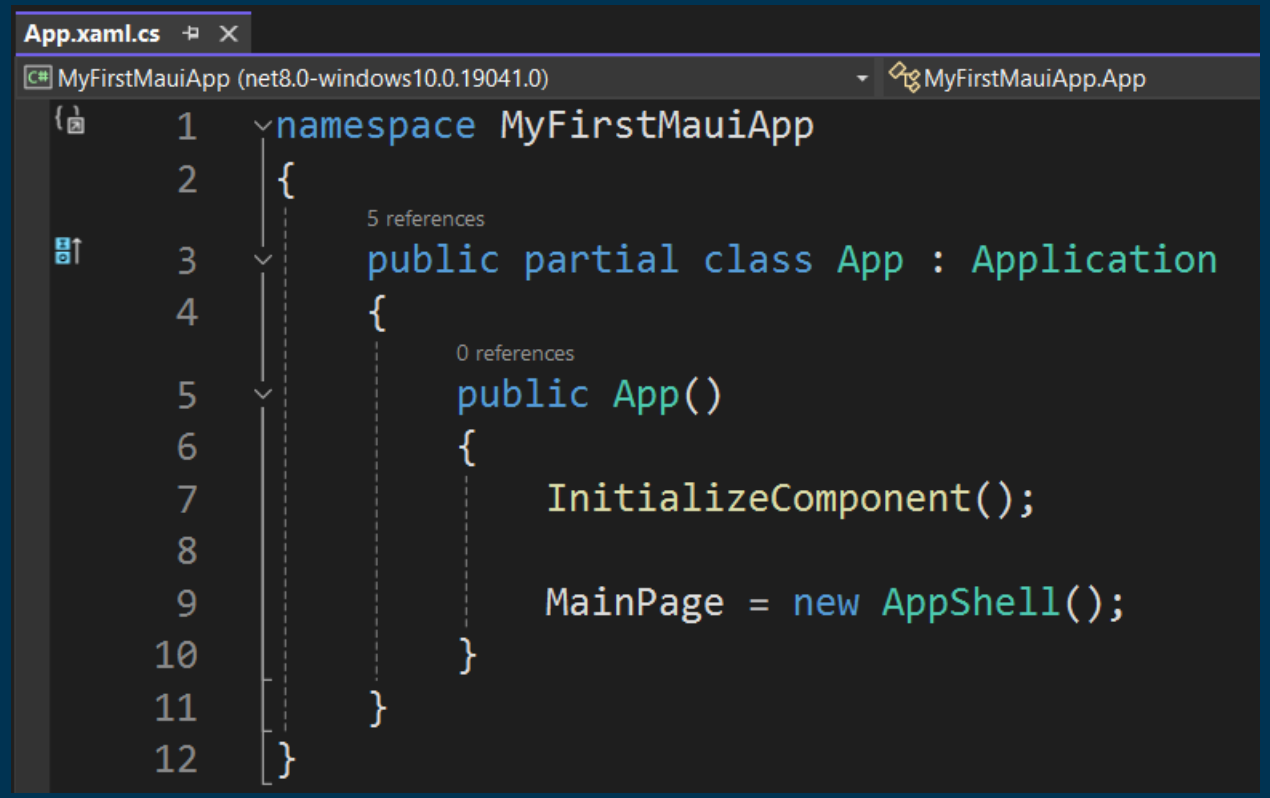
MyFirstMauiApp (net8.0-windows10.0.19041.0) | Application | Application

```
1   <?xml version = "1.0" encoding = "UTF-8" ?>
2   <Application xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
3                xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
4                xmlns:local="clr-namespace:MyFirstMauiApp"
5                x:Class="MyFirstMauiApp.App">
6       <Application.Resources>
7           <ResourceDictionary>
8               <ResourceDictionary.MergedDictionaries>
9                   <ResourceDictionary Source="Resources/Styles/Colors.xaml" />
10                  <ResourceDictionary Source="Resources/Styles/Styles.xaml" />
11              </ResourceDictionary.MergedDictionaries>
12          </ResourceDictionary>
13      </Application.Resources>
14  </Application>
```

.NET MAUI Project Structure and Application Startup

App.xaml.cs:

- This is the code-behind for the App.xaml file.
- This file defines the `App` class.
- This class represents your application at runtime.
- The constructor in this class creates an initial window and assigns it to the `MainPage` property.
- This property determines which page is displayed when the application starts running.



.NET MAUI Project Structure and Application Startup

AppShell.xaml:

- This file is a .NET MAUI application's main structure.
- The .NET MAUI Shell provides many features that are beneficial for multiple-platform apps including app styling, URI based navigation etc.
- The default template provides a single page (or `ShellContent`) that is displayed when the app starts.

AppShell.xaml

MyFirstMauiApp (net8.0-windows10.0.19041.0) | Shell

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <Shell
3      x:Class="MyFirstMauiApp.AppShell"
4      xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
5      xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
6      xmlns:local="clr-namespace:MyFirstMauiApp"
7      Shell.FlyoutBehavior="Disabled"
8      Title="MyFirstMauiApp">
9
10     <ShellContent
11         Title="Home"
12         ContentTemplate="{DataTemplate local:MainPage}"
13         Route="MainPage" />
14
15 </Shell>
```

.NET MAUI Project Structure and Application Startup

MainPage.xaml:

- This file contains the user-interface definition.
- The sample app that the MAUI App template generates contains two labels, a button, and an image.
- The controls are arranged using a `VerticalStackLayout` enclosed in a `ScrollView`.
- The `VerticalStackLayout` control arranges the controls vertically (in a stack), and the `ScrollView` provides a scrollbar if the view is too large to be displayed on the device.
- You're intended to replace the contents of this file with your own UI layout.
- You can also define more XAML pages if you have a multi-page app.

MainPage.xaml

MyFirstMauiApp (net8.0-windows10.0.19041.0) | ContentPage

```
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               x:Class="MyFirstMauiApp.MainPage">
5
6      <ScrollView>
7          <VerticalStackLayout
8              Padding="30,0"
9              Spacing="25">
10             <Image
11                 Source="dotnet_bot.png"
12                 HeightRequest="185"
13                 Aspect="AspectFit"
14                 SemanticProperties.Description="dot net bot in a race car number eight" />
15
16             <Label
17                 Text="Hello, World!"
18                 Style="{StaticResource Headline}"
19                 SemanticProperties.HeadingLevel="Level1" />
20
21             <Label
22                 Text="Welcome to &#10;.NET Multi-platform App UI"
23                 Style="{StaticResource SubHeadline}"
24                 SemanticProperties.HeadingLevel="Level2"
25                 SemanticProperties.Description="Welcome to dot net Multi platform App U I" />
26
27             <Button
28                 x:Name="CounterBtn"
29                 Text="Click me"
30                 SemanticProperties.Hint="Counts the number of times you click"
31                 Clicked="OnCounterClicked"
32                 HorizontalOptions="Fill" />
33         </VerticalStackLayout>
34     </ScrollView>
35
36 </ContentPage>
```

.NET MAUI Project Structure and Application Startup

MainPage.xaml.cs:

- This is the code-behind for the page.
- In this file, you define the logic for the various event handlers and other actions that are triggered by the controls on the page.
- The example code implements a handler for the `Clicked` event for the button on the page.
- The code simply increments a counter variable and displays the result in a label on the page.
- The Semantic service provided as part of the MAUI Essentials library supports accessibility.
- The static `Announce` method of the `SemanticScreenReader` class specifies the text announced by a screen reader when the user selects the button.

MainPage.xaml.cs

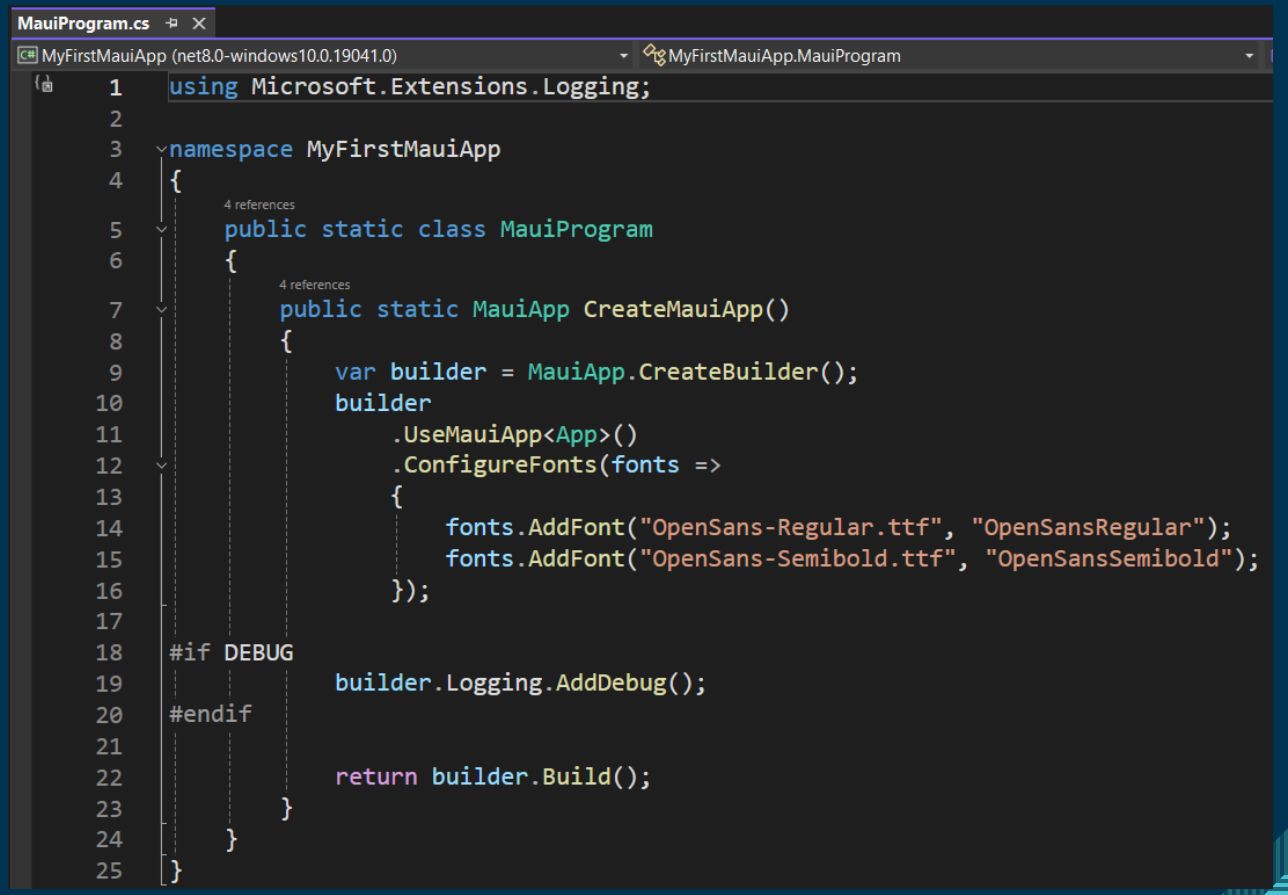
MyFirstMauiApp (net8.0-windows10.0.19041.0) — MyFirstMauiApp.MainPage

```
1  namespace MyFirstMauiApp
2  {
       4 references
3      public partial class MainPage : ContentPage
4      {
5          int count = 0;
6
           0 references
7          public MainPage()
8          {
9              InitializeComponent();
10         }
11
           0 references
12         private void OnCounterClicked(object sender, EventArgs e)
13         {
14             count++;
15
16             if (count == 1)
17                 CounterBtn.Text = $"Clicked {count} time";
18             else
19                 CounterBtn.Text = $"Clicked {count} times";
20
21             SemanticScreenReader.Announce(CounterBtn.Text);
22         }
23     }
24
25 }
```

.NET MAUI Project Structure and Application Startup

MauiProgram.cs:

- Each native platform has a different starting point that creates and initializes the application.
- You can find this code in the Platforms folder in the project.
- This code is platform-specific, but at the end it calls the `CreateMauiApp` method of the static `MauiProgram` class.
- You use the `CreateMauiApp` method to configure the application by creating an app builder object.
- At a minimum, you need to specify which class describes your application.
- You do this with the `UseMauiApp` generic method of the app builder object; the type parameter specifies the application class.
- The app builder also provides methods for tasks such as registering fonts, configuring services for dependency injection, registering custom handlers for controls, and more.

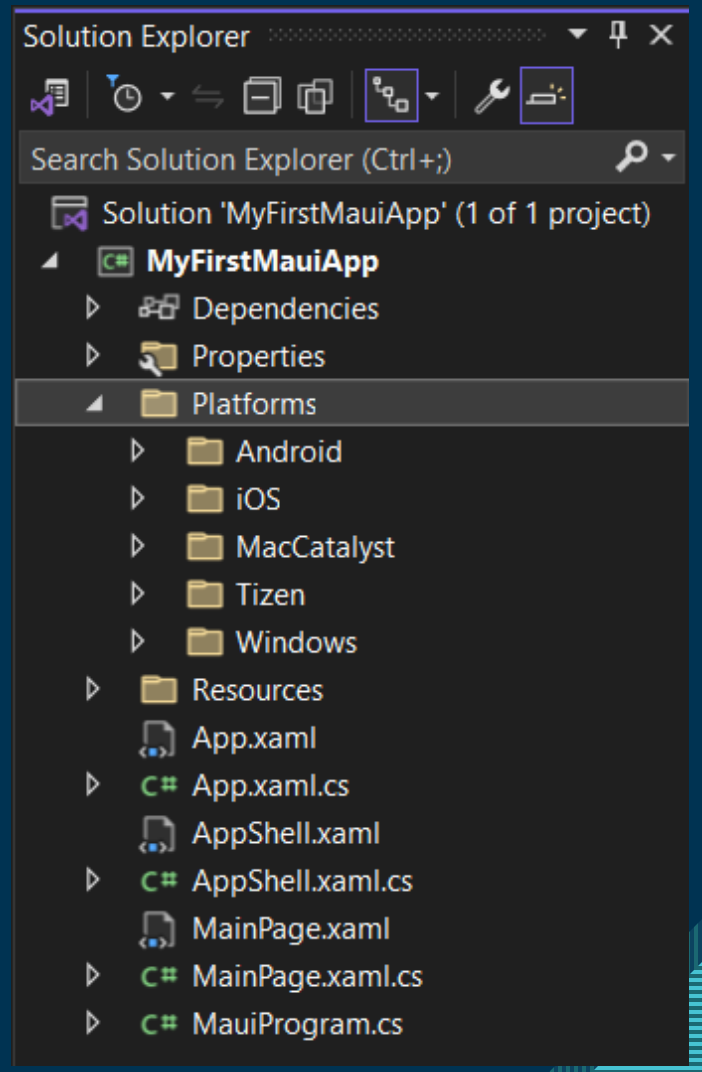


.NET MAUI Project Structure and Application Startup

Platforms:

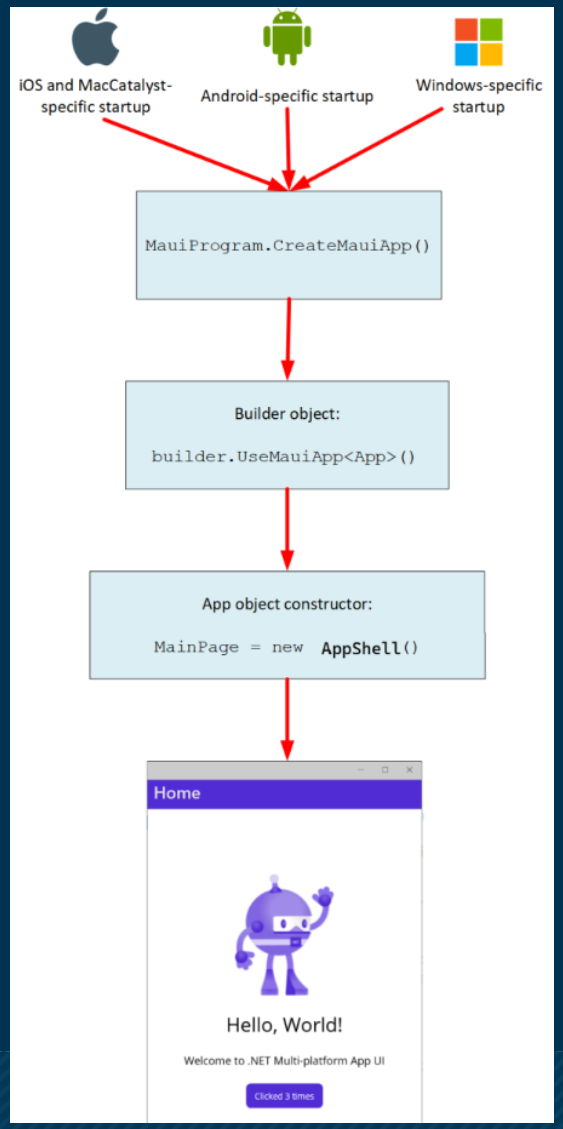
- This folder contains platform-specific initialization code files and resources.
- There are folders for Android, iOS, MacCatalyst, Tizen, and Windows.
- At runtime, the app starts up in a platform-specific way.
- Much of the start-up process is abstracted out by the MAUI libraries' internals, but the code files in these folders provide a mechanism for hooking up your own custom initialization.
- The important point is that when initialization is complete, the platform-specific code calls the `MauiProgram.CreateMauiApp` method, which then creates and runs the `App` object as described earlier.
- For example, the MainApplication.cs file in the Android folder, the AppDelegate.cs file in the iOS and MacCatalyst folder, and the App.xaml.cs file in the Windows folder all contain the overrides:

```
protected override MauiApp CreateMauiApp() => MauiProgram.CreateMauiApp();
```



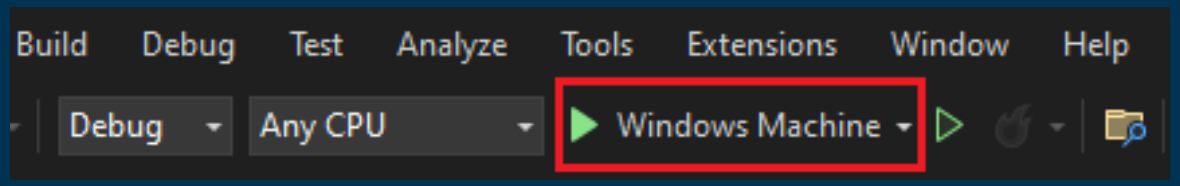
.NET MAUI Project Structure and Application Startup

- The following image illustrates the flow of control when a .NET MAUI app starts up:

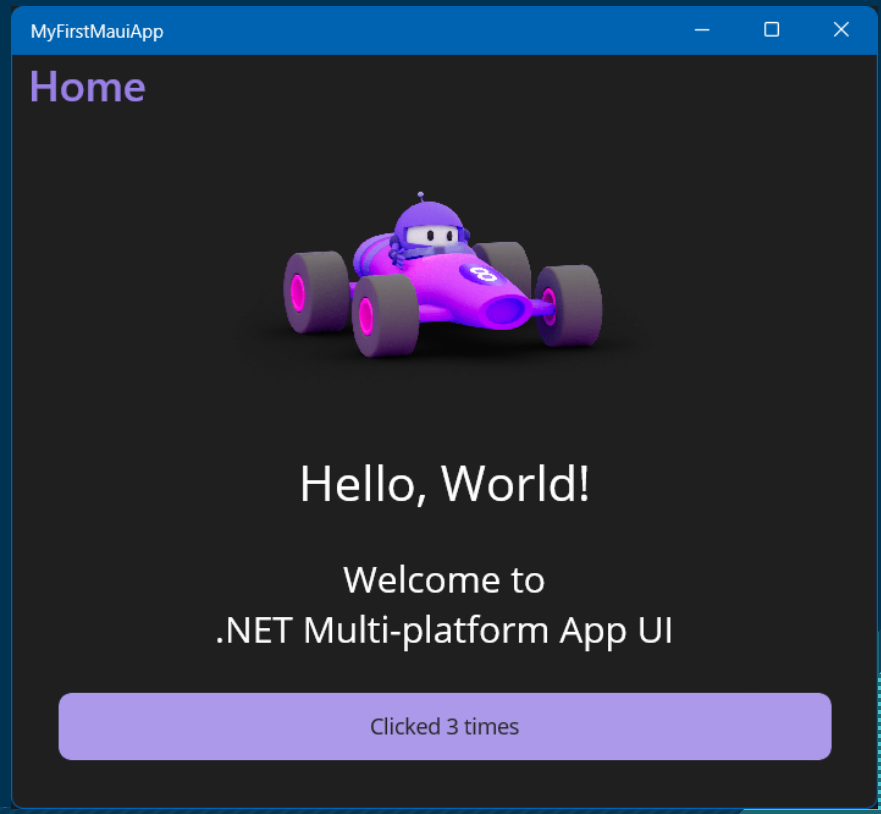


Run Your App

- In the Visual Studio toolbar, you should see Windows Machine as the debug target by default.
- Press the Windows Machine button to build and run the app.



- In the running app, select the Click me button a few times and observe that the count of the number of button clicks is incremented.

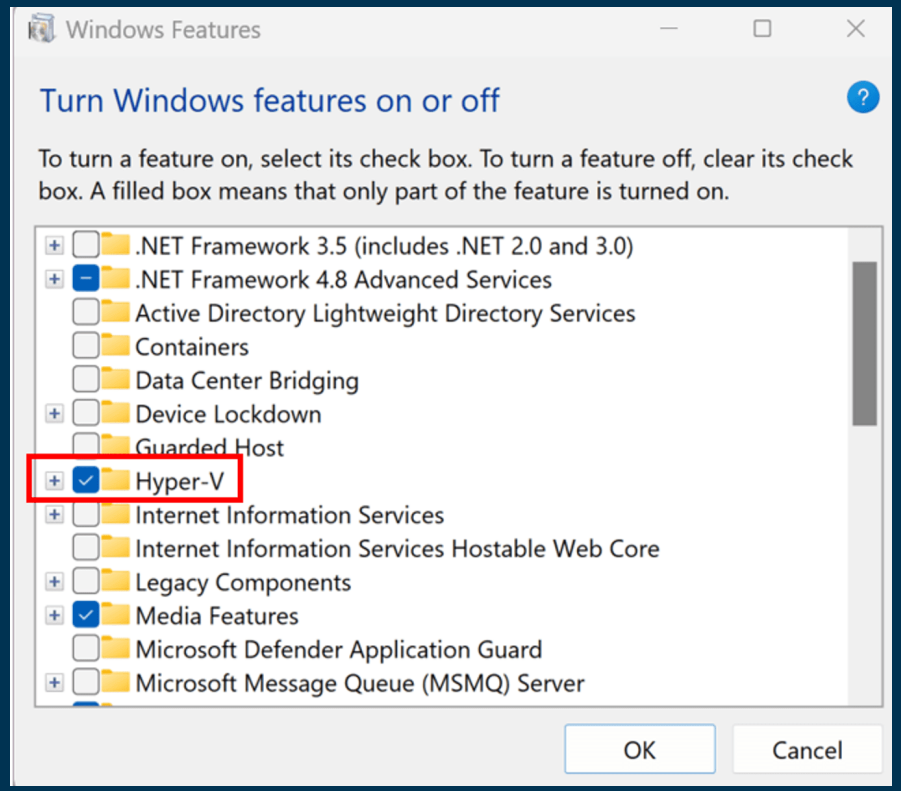


Run on Mobile Device

- You can decide to deploy to one or more of the platforms based on your development environment.
- You just ran and deployed your app to Windows.
- Now, let's set up an Android device or emulator.

Activate Hyper-V

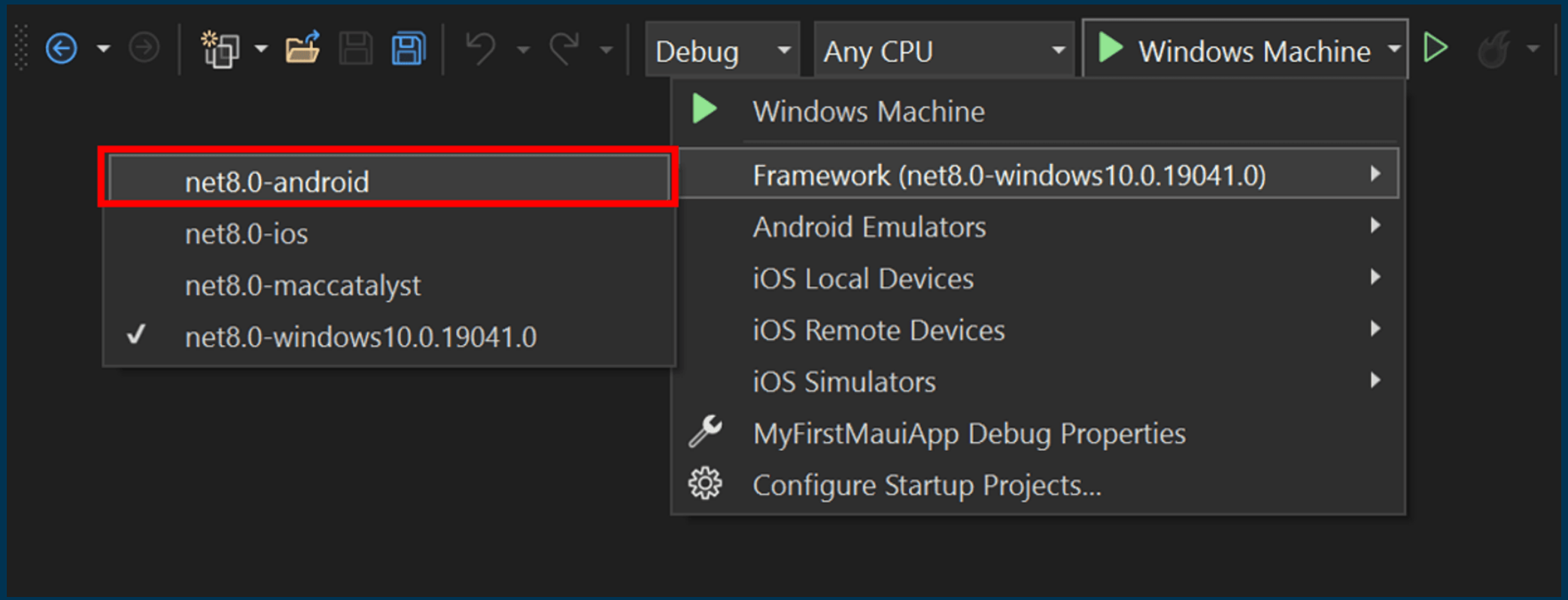
- In your Windows Search Bar, type Windows features and select Turn Windows features on or off.
- Make sure the Hyper-V box is checked and select Ok.



- **Note:**
  - You'll need to reboot your machine at this time for the change to take effect.

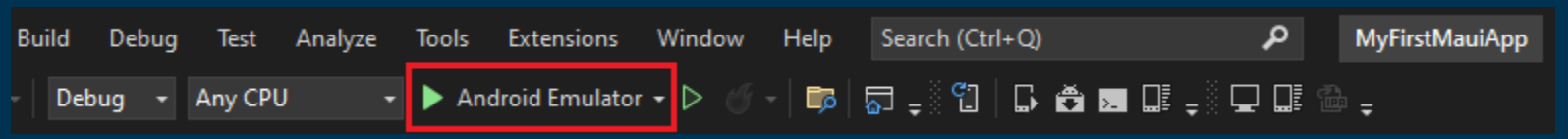
Install Android SDK

- Open Visual Studio and your MyFirstMauiApp solution.
- From the debug target drop-down menu, select net8.0-android under Framework.

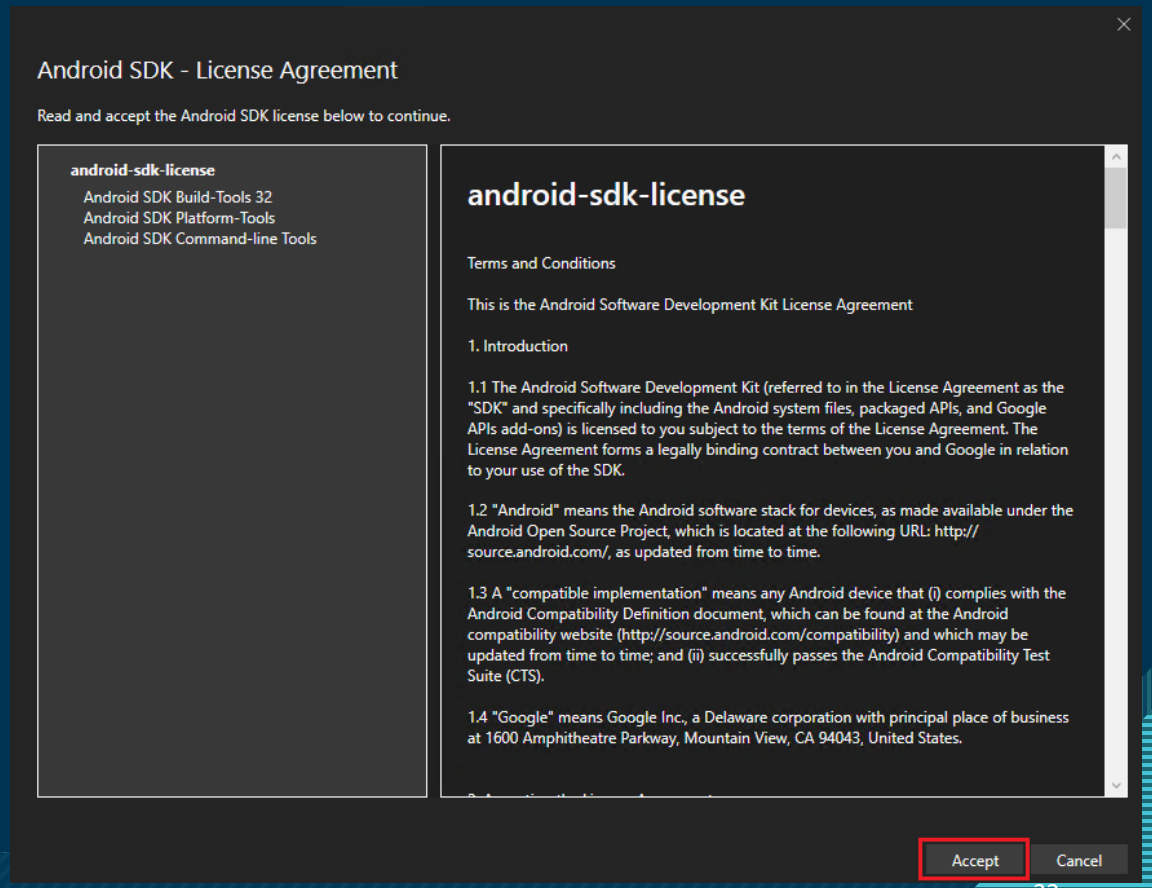


Set Up Android Emulator

- You'll see Android Emulator selected in the debug target drop-down menu.

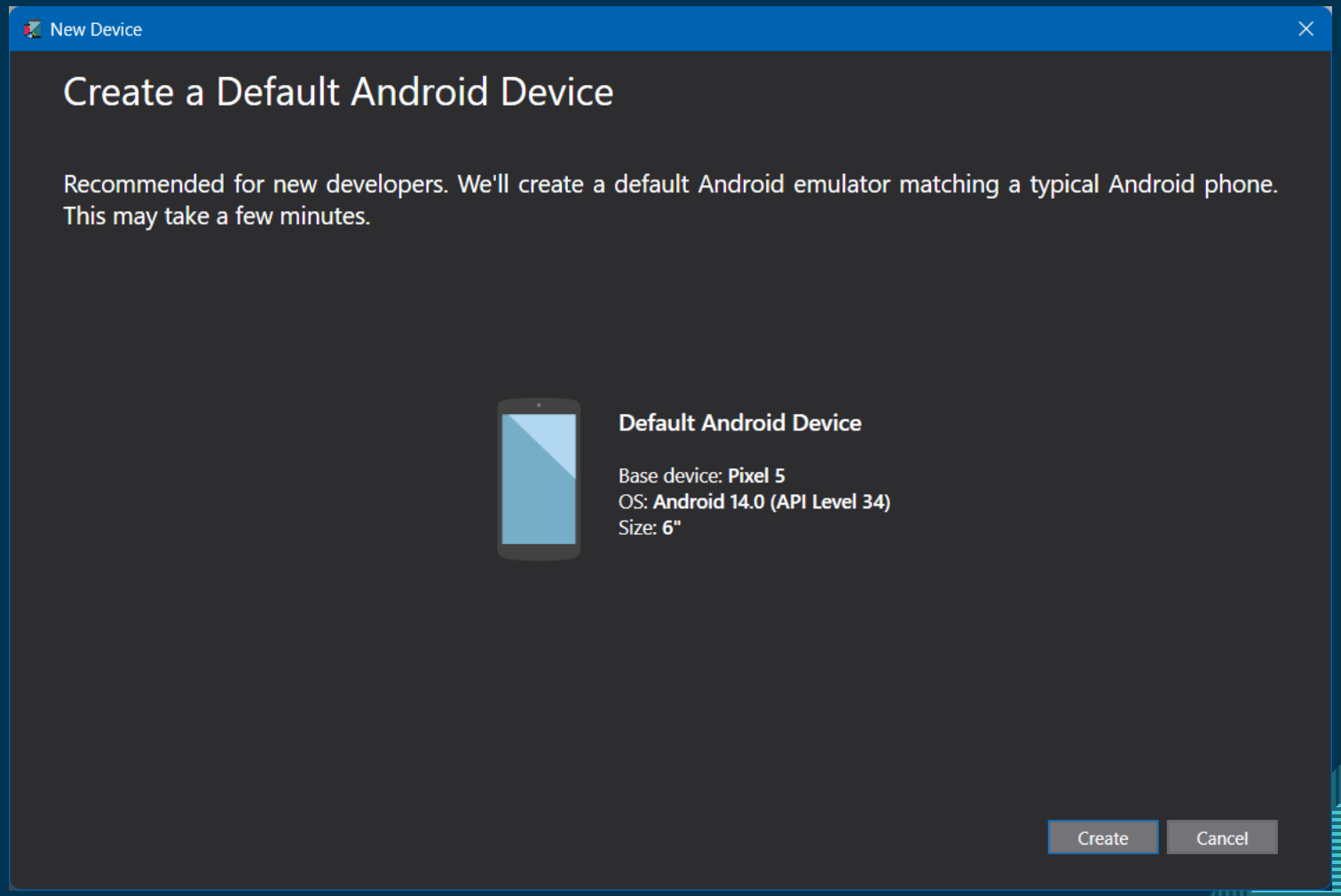


- Click it to start the creation process.
- If a license acceptance dialog shows up, select Accept.



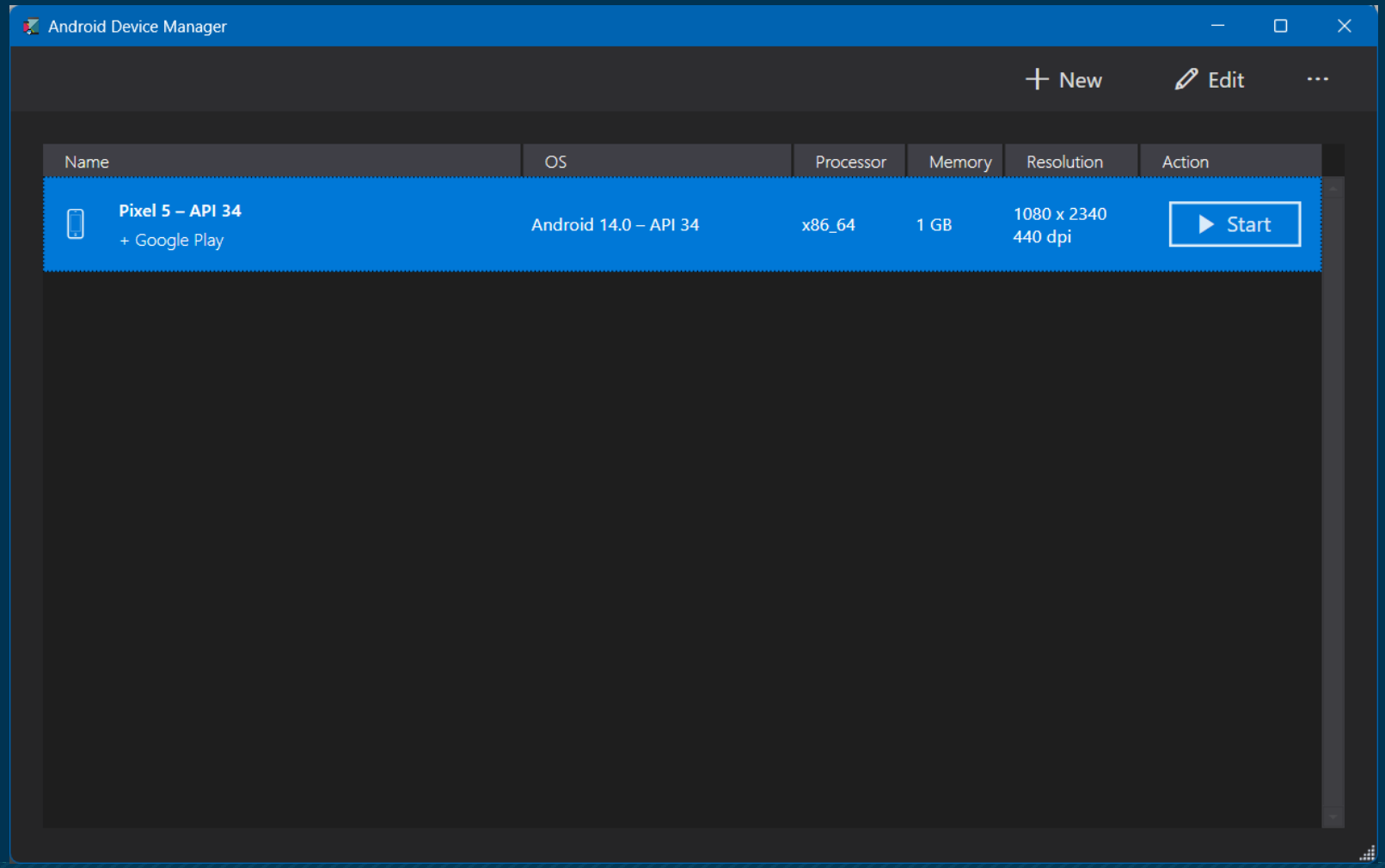
Set Up Android Emulator

- This brings up the New Device window.
- Select the Create button to create an emulator with default settings.
- This will download the emulator images and finalize the creation of the emulator for use in Visual Studio.
- This step might take a while.



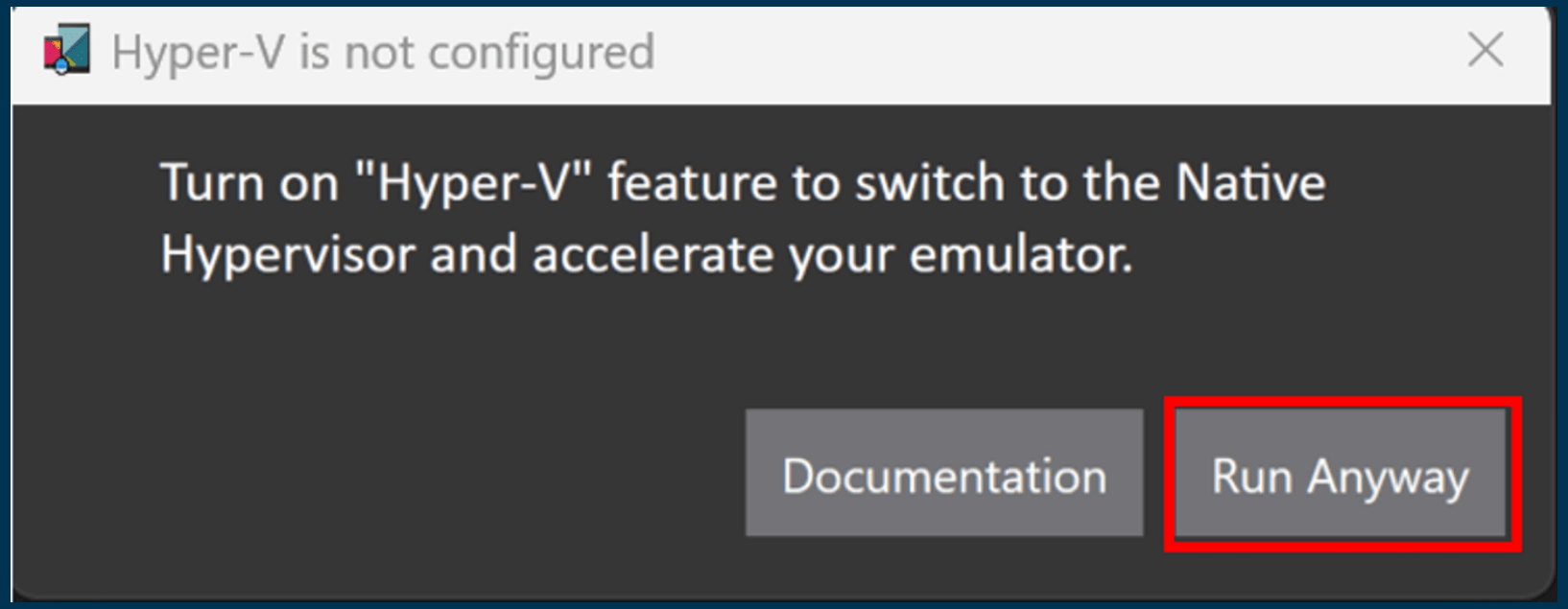
Set Up Android Emulator

- Once the emulator has been created, you'll see a button that says Start.
- Click it.



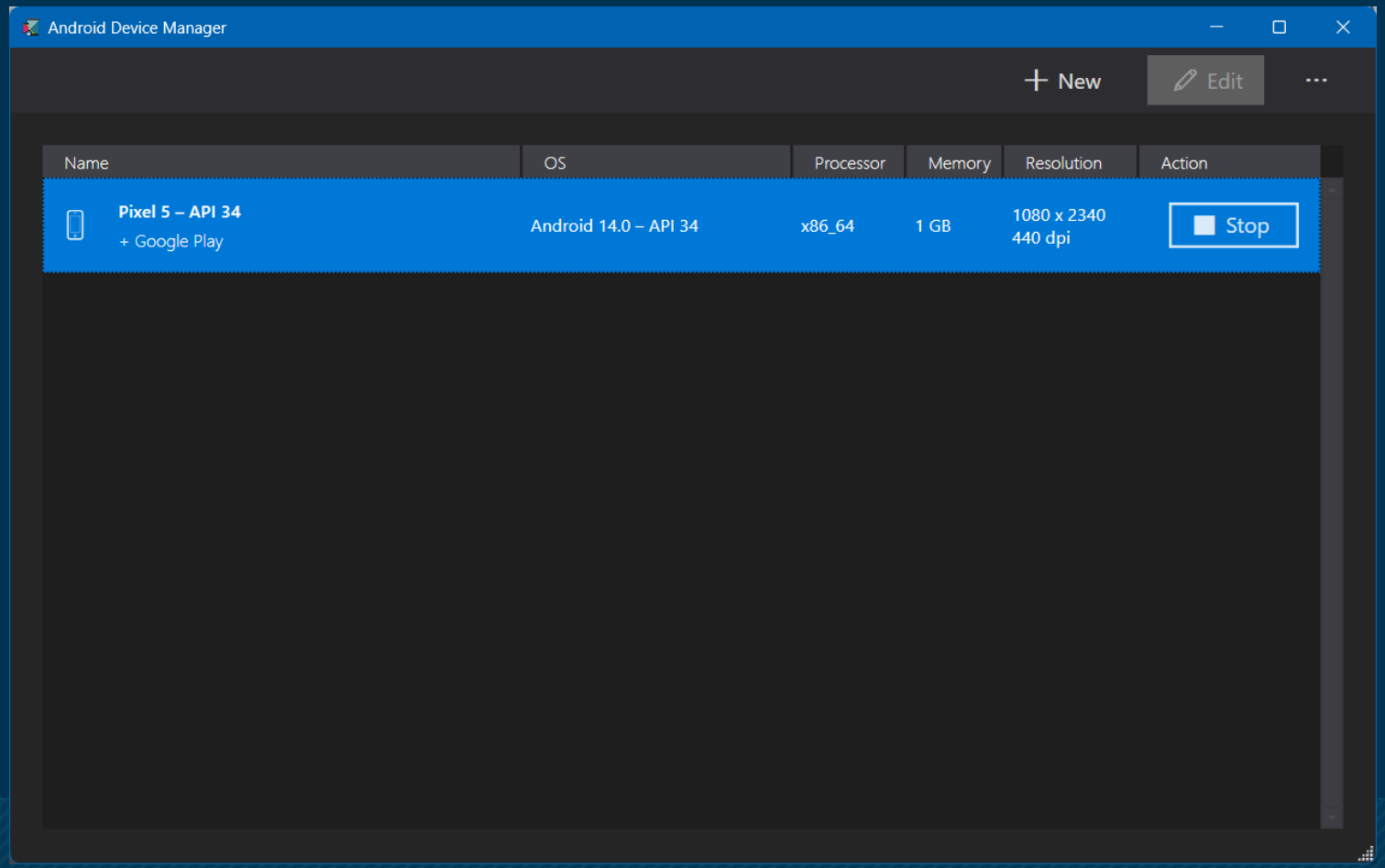
Set Up Android Emulator

- You may receive prompt to enable Windows Hypervisor Platform.
- Follow these instructions to enable acceleration to improve performance (required to get the emulator up and running).
- Select Run Anyway.



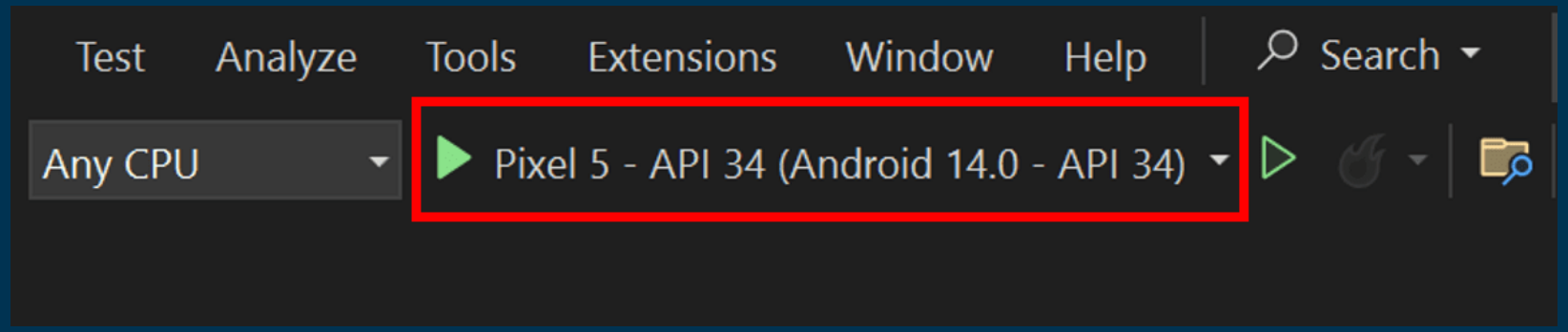
Set Up Android Emulator

- The Android emulator will launch and will be fully finished when you see the ability to Stop in the Android Device Manager.
- This may take some time.



Set Up Android Emulator

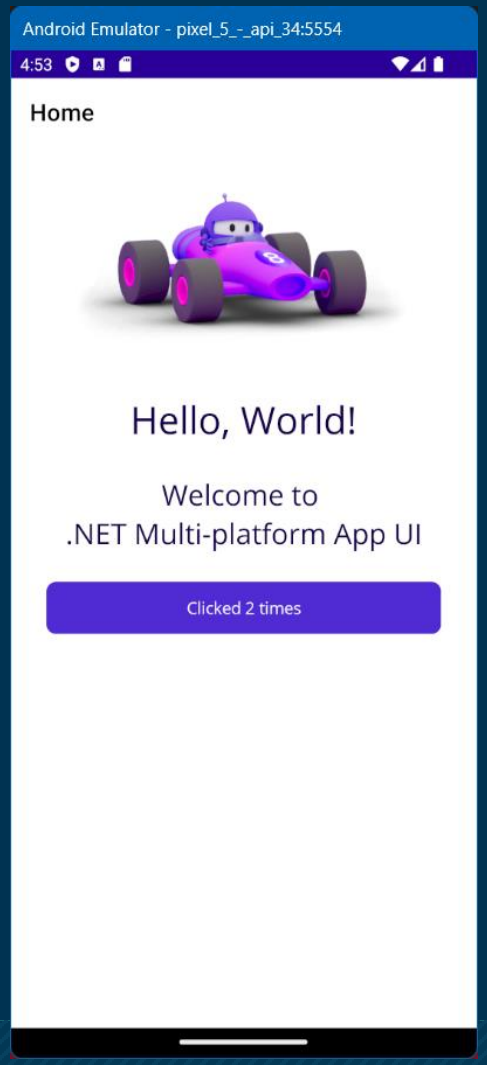
- Once it's fully finished, you'll see it displayed in the Visual Studio debug menu.



- Your Android emulator has now been created and is ready to use.
- Next time you run Visual Studio, the emulator will appear directly in the debug target window and will start when you select it.
- If you ran into any issues or have performance issues with the emulator, read through the full setup documentation.

Set Up Android Emulator

- Your application will build, deploy to the Android device/emulator selected, and run.



Pages

- Let’s learn more about the building blocks of a .NET MAUI application and navigation structures.

- Pages are the root of the UI hierarchy in .NET MAUI.
- The solution you've seen so far included a class called `MainPage`.
- This class derives from `ContentPage`, which is the simplest and most common page type.
- A content page simply displays its contents.
- .NET MAUI has several other built-in page types, too, including the following:
  - `TabbedPage`:
    - This is the root page used for tab navigation.
    - A tabbed page contains child page objects; one for each tab.

  - `FlyoutPage`:
    - A flyout page contains a list of items.
    - When you select an item, a view displaying the details for that item appears.

- Other page types are available, and are mostly used for enabling different navigation patterns in multi-screen apps.

- Learn more:
  - https://learn.microsoft.com/en-us/dotnet/maui/user-interface/pages/contentpage?view=net-maui-8.0

Views

- A content page typically displays a view.
- A view enables you to retrieve and present data in a specific manner.
- The default view for a content page is a `ContentView`, which displays items as-is.
  - If you shrink the view, items might disappear from the display until you resize the view.
- A `ScrollView` enables you to display items in a scrolling window.
  - If you shrink the window, you can scroll up and down to display items.
- A `CarouselView` is a scrollable view that enables the user to swipe through a items.
- A `CollectionView` can retrieve data from a named data source and present each item using a template.
- There are many other types of views available, as well.
- Learn more:
  - https://learn.microsoft.com/en-us/dotnet/maui/user-interface/controls/scrollview?view=net-maui-8.0

Controls and Layouts

- A view can contain a single control such as a button, label, or text boxes.
- However, a user interface restricted to a single control wouldn't be very useful, so controls are positioned in a layout.
- A layout defines the rules by which the controls are displayed relative to each other.
- A layout is also a control, so you can add it to a view.
- If you look at the default MainPage.xaml file, you'll see this page/view/layout/control hierarchy in action.
- In this XAML code, the `VerticalStackLayout` element is just another control that allows you to fine-tune the layout of other controls.

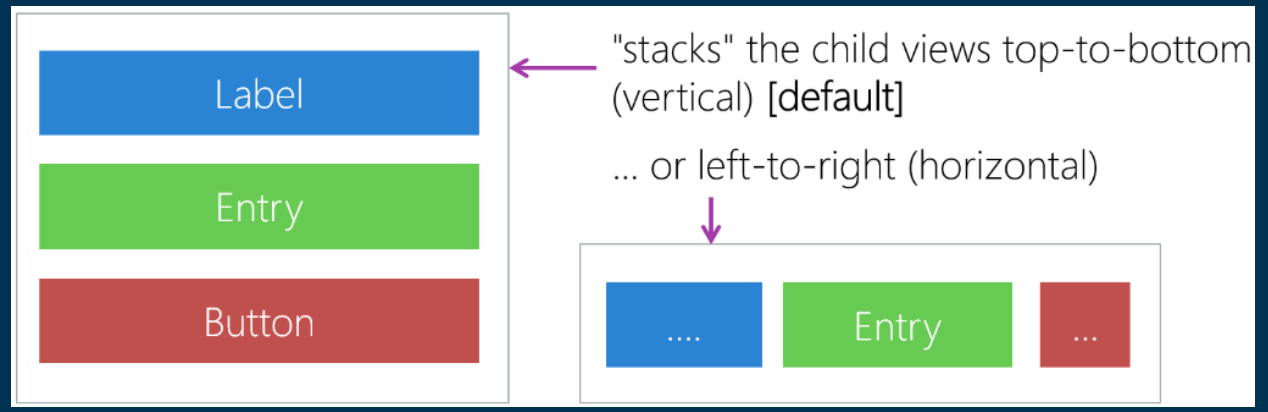
```
<ContentPage ...>
    <ScrollView ...>
        <VerticalStackLayout>
            <Image ... />
            <Label ... />
            <Label ... />
            <Button ... />
        </VerticalStackLayout>
    </ScrollView>
</ContentPage>
```

- Learn more:
  - https://learn.microsoft.com/en-us/dotnet/maui/user-interface/layouts/?view=net-maui-8.0

Controls and Layouts

- Some of the common controls used to define layouts are:

- `VerticalStackLayout` and `HorizontalStackLayout`, which are optimized stack layouts that lay out controls in a top-to-bottom or left-to-right stack.



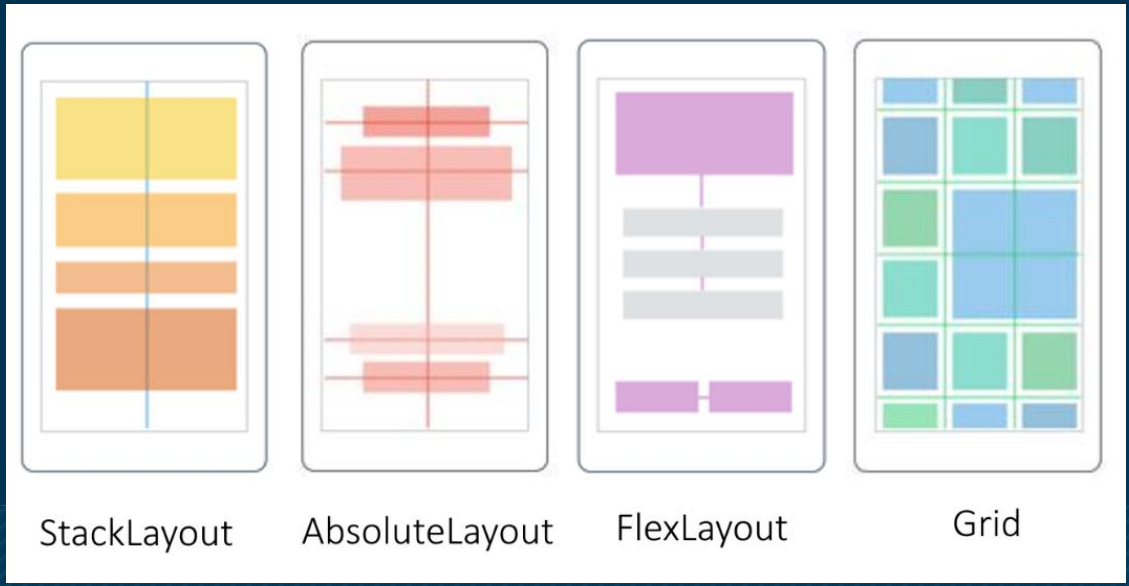
- A `StackLayout` is also available, which has a property named `StackOrientation` that you can set to `Horizontal` or `Vertical`.
- On a tablet or phone, modifying this property in your application code enables you to adjust the display if the user rotates the device.

Controls and Layouts

- `AbsoluteLayout`, which lets you set exact coordinates for controls.

- `FlexLayout`, which is similar to `StackLayout` except that it enables you to wrap the child controls it contains if they don't fit in a single row or column.

- `Grid`, which lays out its controls according to a column and row location we set.
- You can define the column and row sizes as well as spans, so grid layouts don't necessarily have a "checkerboard look" to them.



Tuning a Layout

- It's useful to add a little *breathing space* around a control.
- Each control has a `Margin` property that the layouts respect.

- All the layouts also have a `Padding` property that keeps any of their children from getting close to the border of the layout.
- One way to think of this concept is that all the controls are in a box, and that box has padded walls.

- Another useful whitespace setting is the `Spacing` property of `VerticalStackLayout` or `HorizontalStackLayout`.
- This is the space between all the children of the layout.
- This is additive with the control's own margin, so the actual whitespace will be margin plus spacing.

Thank You

Copyright

The materials provided in class and in SLATE are protected by copyright. They are intended for the personal, educational uses of students in this course and should not be shared externally or on websites such as Chegg, Course Hero or OneClass. Unauthorized distribution may result in copyright infringement and violation of Sheridan policies.

References

Material has been taken as is from:

- Microsoft Official Documentation:
  - https://learn.microsoft.com/en-us/training/modules/build-mobile-and-desktop-apps/1-introduction
  - https://dotnet.microsoft.com/en-us/learn/maui/first-app-tutorial/intro
- .NET MAUI guide: Complete overview + demo:
  - https://blog.logrocket.com/dotnet-maui-guide-complete-overview-demo/